| | |
|---|---|
| Project Acronym: | **VICINITY** |
| Project Full Title: | **Open virtual neighbourhood network to connect intelligent buildings and smart objects** |
| Grant Agreement: | **688467** |
| Project Duration: | **48 months (01/01/2016 - 31/12/2019)** |

# Deliverable D6.2

# Report on VICINITY test-bed deployment, including Validation, Parameterization and Testing

| | |
|---|---|
| Work Package: | WP6 – VICINITY Framework Integration & Lab Testing |
| Task(s): | T6.2 – Lab setup, Testing & Validation |
| Lead Beneficiary: | AAU |
| Due Date: | 31st December 2018 (M36) |
| Submission Date: | 18th December 2018 (M36) |
| Deliverable Status: | Final |
| Deliverable Type: | DEM |
| Dissemination Level: | PU |
| File Name: | VICINITY_Report on VICINITY test-bed deployment, including Validation, Parameterization and Testing_v1.0.pdf |

*This project has received funding from the European Union's Horizon 2020 Research and innovation programme under Grant Agreement n°688467*

## VICINITY Consortium

| No | Beneficiary | | Country |
|---|---|---|---|
| 1. | TU Kaiserslautern (Coordinator) | UNIKL | Germany |
| 2. | ATOS SPAIN SA | ATOS | Spain |
| 3. | Centre for Research and Technology Hellas | CERTH | Greece |
| 4. | Aalborg University | AAU | Denmark |
| 5. | GORENJE GOSPODINJSKI APARATI D.D. | GRN | Slovenia |
| 6. | Hellenic Telecommunications Organization S.A. | OTE | Greece |
| 7. | bAvenir s.r.o. | BVR | Slovakia |
| 8. | Climate Associates Ltd | CAL | United Kingdom |
| 9. | InterSoft A.S. | IS | Slovakia |
| 10. | Universidad Politécnica de Madrid | UPM | Spain |
| 11. | Gnomon Informatics S.A. | GNOMON | Greece |
| 12. | Tiny Mesh AS | TINYM | Norway |
| 13. | HAFENSTROM AS | HITS | Norway |
| 14. | Enercoutim – Associação Empresarial de Energia Solar de Alcoutim | ENERC | Portugal |
| 15. | Municipality of Pylaia-Hortiatis | MPH | Greece |



[1] ***Deliverable Type:***
R: Document, report (excluding the periodic and final reports)
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
OTHER: Software, technical diagram, etc.

[2] ***Dissemination level:***
PU: Public, fully open, e.g. web
CO: Confidential, restricted under conditions set out in Model Grant Agreement
CI: Classified, information as referred to in Commission Decision 2001/844/EC.

## Authors List


| Leading Author (Editor) | | | | |
|---|---|---|---|---|
| **Surname** | | **First Name** | **Beneficiary** | **Contact email** |
| Guan | | Yajuan | AAU | ygu@et.aau.dk |
| **Co-authors (in alphabetic order)** | | | | |
| **No** | **Surname** | **First Name** | **Beneficiary** | **Contact email** |
| 1. | Čolić | Nikolaj | GRN | Nikolaj.Colic@gorenje.com |
| 2. | Feng | Wei | AAU | wfe@et.aau.dk |
| 3. | Fernandez | David | ATOS | david.gomez@atos.net |
| 4. | Flsveen | Flemming | HITS | flsveen@online.no |
| 5. | Filosofov | Dmitry | TINYM | dmitry@tiny-mesh.com |
| 6. | Guerrero | Josep | AAU | joz@et.aau.dk |
| 7. | Heinz | Christopher | UNIKL | heinz@cs.uni-kl.de |
| 8. | Horniak | Martin | BVR | martin.horniak@bavenir.eu |
| 9. | Hovstø | Asbjørn | HITS | hovsto@online.no |
| 10. | Koutli | Mary | CERTH | mkoutli@iti.gr |
| 11. | Koelsch | Johannes | UNIKL | koelsch@cs.uni-kl.de |
| 12. | Oravec | Viktor | BVR | viktor.oravec@bavenir.eu |
| 13. | Palacios-Garcia | Emilio | AAU | epg@et.aau.dk |
| 14. | Poljakov | German | TINYM | german@tiny-mesh.com |
| 15. | Tryferidis | Thanasis | CERTH | thanasic@iti.gr |
| 16. | Theologou | Natalia | CERTH | nataliath@iti.gr |
| 17. | Vásquez | Juan | AAU | juq@et.aau.dk |
| 18. | Vanya | Stefan | BVR | stefan.vanya@bavenir.eu |

## Reviewers List

| List of Reviewers (in alphabetic order) | | | | |
|---|---|---|---|---|
| **No** | **Surname** | **First Name** | **Beneficiary** | **Contact email** |
| 1. | Almela Miralles | Jorge | BVR | jorge.almela@bavenir.eu |
| 2. | Paralič | Marek | IS | marek.paralic@intersoft.sk |
| 3. | Sundvor | Mariann | TINYM (Chair) | mariann@tiny-mesh.com |

## Revision Control

| Version | Date | Status | Modifications made by |
|---|---|---|---|
| 0.1 | 10 May 2018 | Initial draft | Guan (AAU), Guerrero (AAU), Vásquez (AAU) |
| 0.2 | 20 May 2018 | Updated draft | Guan (AAU), Guerrero (AAU), Vásquez (AAU) |
| 0.3 | 20 May 2018 | First Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH) |
| 0.4 | 3 July 2018 | Second Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH), Fernandez (ATOS) |
| 0.5 | 20 July 2018 | Third Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH), Fernandez (ATOS) |
| 0.6 | 25 Oct 2018 | Fourth Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH), Fernandez (ATOS) |
| 0.7 | 18 Nov 2018 | Fifth Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH), Fernandez (ATOS), Heinz (UNIKL), Koelsch (UNIKL), Feng (UNIKL), Emilio (AAU) |
| 0.8 | 26 Nov 2018 | Seventh Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH), Fernandez (ATOS), Heinz (UNIKL), Koelsch (UNIKL), Feng (UNIKL), Emilio (AAU) |
| 0.9 | 27 Nov 2018 | Seventh Draft formatted includes feedbacks from WP 6 contributors | Guan (AAU), Koutli (CERTH), Fernandez (ATOS), Heinz (UNIKL), Koelsch (UNIKL), Feng (UNIKL), Emilio (AAU), Guerrero (AAU), Vásquez (AAU) |
| 0.9.1 | 28 Nov 2018 | First Final Draft to be sent for reviewing | Guan (AAU), Koutli (CERTH), Fernandez (ATOS), Heinz (UNIKL), Koelsch (UNIKL), Feng (UNIKL), Emilio (AAU), Guerrero (AAU), Vásquez (AAU), Oravec (BVR), Horniak (BVR), Vanya (BVR) |
| 0.9.2 | 17 Dec 2018 | Final Draft reviewed | Guan (AAU), Koutli (CERTH), Fernandez (ATOS), Heinz (UNIKL), Koelsch (UNIKL), Feng (UNIKL) |
| 1.0 | 18 Dec 2018 | Submission to the EC | Zivkovic (UNIKL) |

# Executive Summary

This deliverable and demonstrator (Annex) present the Lab testing and validation for each of the components, such as Adapters, Agent, Gateway API, that constitute the VICINITY [1] framework, and for the user-cases defined in D5.2. It is an important part to reach Milestone 7 in the conduction of Lab testing and validation within Task T.6.2.

After the fundamental integration tests in D6.1, testing plans of D6.2 are developed in terms of Edge Case Testing Methodology which includes both edge case testing and internal point testing. Two important results have been achieved from the lab-testing:

A. VICINITY prototype performance when closing to the edges/limits has been tested and restricted by means of Edge Case Testing. Therefore, specify a stable operation zone for VICINITY platform.

B. The internal point testing scenarios are mostly designed to be consistent with use-cases defined in WP5. They mainly focus on prototype functionality and performance, including cross-domain testing cases, in order to validate and improve VICINITY prototype functionality.

This deliverable and demonstrator cover all individual modules that have been developed in WP3 - “VICINITY Server Implementation”, WP4 - “VICINITY Client Infrastructures Implementation”, and keep in line with WP5 – “Value-Added Services Implementation”. Core components functions, integration performance, features are tested and validated. Problems identified are timely reported and solved. The Lab testing results are forwarded to WP8 – “Pilot demonstration and Overall evaluation”.

In addition, this deliverable addresses some of the feedback comments given by our reviewers: VICINITY and especially the academic partners should research the use of new technologies and their application and value for VICINITY. The Lab Testing of partner UNIKL will hence also go beyond what was written in the Description of Work and will evaluate Network Simulators for virtual Prototyping of VICINITY use cases and also homomorphic encryption to further enhance Privacy of the project’s solution.

The related Adapters and VASs are publicly available in VICINITY H2020 GitHub with configuration and installation documentation including source code [2].

In conclusion, the deliverable and demonstrator focus on iterative Lab testing and validation from VICINITY node to node communication to the real-time experimental platform and complex cross-domain testing scenarios.

## Table of Contents

## List of Tables



## List of Figures

## List of Definitions & Abbreviations

| Abbreviation | Definition |
|---|---|
| AC | Alternating Current |
| API | Application Program Interface |
| DC | Direct Current |
| EC | European Commission |
| EMS | Energy Management System |
| EU | European Union |
| EV | Electric Vehicle |
| GUI | Graphical User Interface |
| HIL | Hardware in the Loop |
| MG | Microgrid |
| NM | Neighbourhood Manager |
| PV | Photovoltaic |
| RMEMS | Residential microgrid energy management system |
| SoC | State of Charge |
| TD | Thing description |
| UI | User Interface |
| VAS | Value-Added Services |
| WP | Work Package |

# 1. Introduction

The deliverable and demonstrator describe the Lab testing process and results for the core components that constitute the VICINITY prototype and the pilot user cases to validate their performance and functionality under both edge cases and internal point cases.

Focus of this deliverable and demonstrator is providing a detailed handbook of the conducted Lab tests to describe the testing steps, actual testing results and solved technical issues.

The tested components cover all individual modules that have been developed in WP3 - "VICINITY Server Implementation", WP4 - "VICINITY Client Infrastructures Implementation". The testing scenarios are designed by considering both unusual circumstances and normal operation points. The formal restrictions of the discovery process with unusual circumstances are defined and validated in D6.3. The normal operation points are designed consisting of the user cases defined in D5.2.

## 1.1. Context within VICINITY

**Fehler! Verweisquelle konnte nicht gefunden werden.** gives an overview of the context of D6.2 within VICINITY. As already mentioned, D6.2 is an important step to reach Milestone 7 (MS7) which marks the conduction of intensive integrated Lab testing for VICINITY prototype, with the use of the VICINITY server components/services, client infrastructures and value-added services that were made available by the previous milestones.

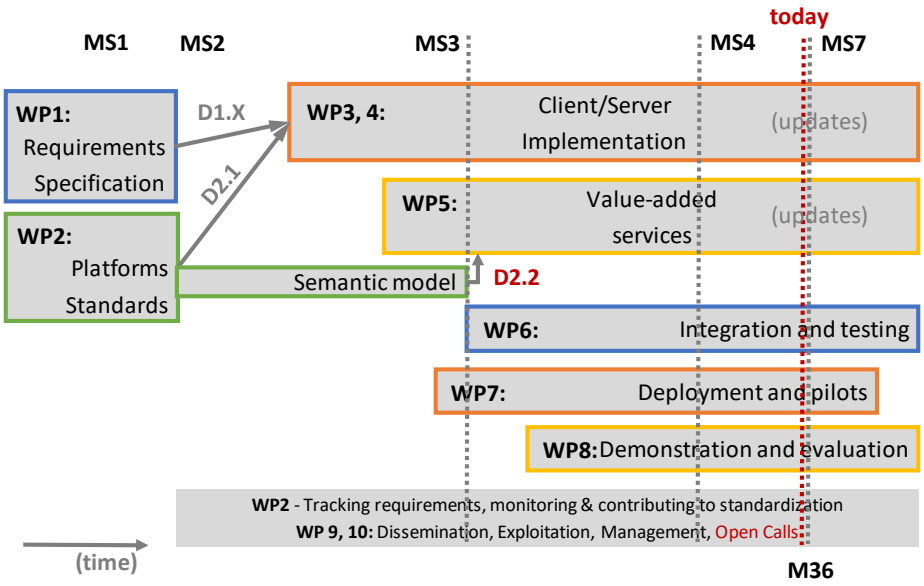



Figure 1 Work Package Architecture

Regarding the relation to other WPs, the current document and demonstrator builds on the results of previous WPs, specifically:

- WP1 – VICINITY Concept Requirements, Barriers, Specification, and Architecture
- WP3 – VICINITY Server Implementation
- WP4 – VICINITY Client Infrastructures Implementation
- WP5 – Value-Added Services implementation

The outcome of this deliverable and demonstrator will form the basis of work for the following WPs and Tasks:

- WP7 – On-site Deployment and Pilot Installations
- WP8 – Pilot Demonstration and Overall Evaluation

## 1.2. Objectives in Work Package 6 and Task 6.2

The purpose of WP6 "VICINITY Framework Integration and Lab Testing" is to ensure that the VICINITY platform operates correctly from a technical perspective prior to deployment at the pilot sites in WP7.

T6.1 "Integration of VICINITY Components", focusses on integrating the components that form server and client infrastructures, along with the related value-added services to form the first version of the VICINITY prototype. The layout and scope of the tests in T6.1 were decided, based on: pilot site definitions, functional requirements, operational requirements and the VICINITY architecture as defined by WP1 "Requirements Specification" and the value-added services as defined by WP5 "Value-Added Services Implementation". The issues that were uncovered during the process are documented in the VICINITY Issues Log which is available for all partners of the project, with the status and context of individual issues. Evidence of the progress in solving these issues with cross-pilot cooperation can also be found on the internal project website. Resolved issues resulted in new versions of the software components, which were deployed following regression testing.

T6.2 "Lab setup, Testing & Validation" deals with two kinds of lab-testing. The first is Edge case testing to validate the expected prototype performance when close to the edges/limits according to the requirements detailed in WP1. The second kind of lab testing focuses on functionality and performance, including cross-domain testing scenarios, in line with value-added services defined in WP5. The diagnosed problems during the lab-testing process are discussed and resolved by collaboration among partners to improve and enrich VICINITY prototype functionality.

T6.3 "Auto-discovery space deployment and validation", establishes the quality and performance of the auto-discovery platform which identifies IoT device types and enables interoperability at the semantic level. Any limitations of the discovery process are identified and resolved as reported in D6.3.

## 1.3. Structure of the Deliverable

**Chapter 1:** Introduction to the deliverable, and the context of the Tasks in Vicinity. This section outlines the role this document plays in the development process.

**Chapter 2:** Test Methodology and Test Scope.

**Chapter 3:** Edge Case Testing.

**Chapter 4-9:** Internal point testing.

**Annex I-XV:** Demonstrators.

# 2. Test Methodology and Test Scope

## 2.1. Test Methodology

After the fundamental integration tests in D6.1, the testing methodology employed in this deliverable is Edge Case Testing Methodology which can be defined as strategies and testing types used to certify that the Application Under Test meets client expectations. The testing plans are designed for examining both edge cases and internal point cases, therefore ensuring that the testing cases have good coverage over the range of values.

The edge testing cases consist of stress registration properties, limit of parallel registrations, and large size of the payload for GET request. They are designed to restrict some features of Gateway API by considering the requirements and installation specifications envisioned in WP1 in order to define a stability and proper operating range for VICINITY platform.

The internal testing points keep in line with the user-cases defined in WP5 to verify the adapter/VAS functional performance and ensure the expected operation. They cover mobility, building, energy, and eHealth domains and refer to privacy, GDPR VAS, LoRa, and FIWARE-compliant device, Omnet++ network simulator and homomorphic encryption.

If the testing results or the design behave unexpectedly, a bug and a trace that lead to it are reported through Open Project, emails, Skype and Slack. Iterative tests have been conducted to verify the solutions and evenly to solve the bugs.

The general structure for each testing case mainly includes test scenario and goal, VICINITY components/functions involved, equipment and testing environments, expected results, test procedure, testing platforms, real results, user interfaces, deviations encountered from expected result and solutions, and an annex for the demonstrator.

## 2.2. Tested VICINITY Platform configuration and coverage

The tested VICINITY Platform configuration and interfaces are referred to Sections 2.2 and 2.3 of D6.1.

# 3. Edge Case Testing (AAU - DK)

## 3.1. Testing objective and the Role of the Vicinity Prototype

The objective of the edge case testing is trying to push some features of Gateway API to its limits, in order to get knowledge of the behaviour of a VICINITY prototype under unusual circumstances. For instance, with heavy load.

Three different edge case tests have been carried out, which have directly interacted with Neighbourhood Manager, Gateway API (v0.6.3), Agent (v0.6.3) and a testing virtual device node.

The formal restrictions of the discovery process with unusual circumstances are defined and validated in D6.3.

## 3.2. Edge Case Testing 1 - Stress registration properties

### 3.2.1. Testing case design

| **Edge case testing 1** | **Stress registration properties** |
|---|---|
| *Test scenario and goal* | The current testing case aims to test whether the VICINITY prototype can deal with a registration with heavy payload. |
| *Iterations* | The test was conducted 3 times for the Agent 0.6.3. |
| *VICINITY components/functions involved* | • Adapter of a virtue testing device v0.0.1<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • None. |
| *Deployment* | • The adapter for the virtue testing device is established based on Python 3. It can be run on any PC by executing the .py file. |
| *Expected results* | • Response should contain the information of “Discovery for adapter successfully done”. The status code of the response should be 200. The registered device should be found in the Neighbourhood manager. |
| *Test procedure* | The test procedure consists of the following steps:<br>1. Create an access point in Neighbourhood manager,<br>  ○ Choose VICINITY agent<br>  ○ Give it a name and set a password<br>2. Use password and AID received after completing step 1 to setup /agent/gateway combo.<br>3. Start gateway<br>4. Start agent<br>5. Create a device thing description with 10000 properties through python3.6<br>6. Send HTTP request to agent to register thing through python3.6,<br>  ○ Request: POST http://<agent_URL>:<agent_port>/agent/objects<br>  ○ Required payload is generated testing thing description |

| | |
|---|---|
| | 7. End of test. |

### 3.2.2. Testing Platform

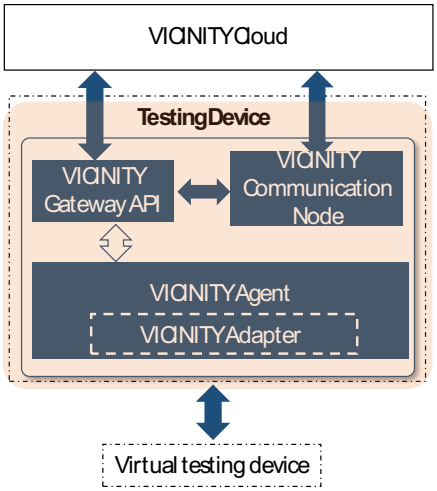



Figure 2 Testing platform and involved components for edge case testing 1.

### 3.2.3. Testing results

| Edge case testing 1 | Stress registration properties |
|---|---|
| *Real results and constraints identified* | 1. After sending the POST request, a response containing the information of successful registration is received, and the status code of the response is 200. The device with 10000 properties can be found in the Neighbourhood Manager. The VICINITY prototype can deal with a registration with at least 10000 properties. |
| *Developed User Interfaces* | None.<br>*Functionalities:* |
| *Real results (demo)* | Related snapshots of data flows are included in Annex I. |
| *Deviations* | None. |
| *Other technical issues* | None. |
| *Status* | Passed. |
| *Notes* | None. |

## 3.3. Edge Case Testing 2 - Limit of parallel registrations

### 3.3.1. Testing case design

| Edge case testing 2 | Limit of parallel registrations |
|---|---|
| *Test scenario and goal* | The current testing case aims to identify how many parallel registrations can be handled by VICINITY prototype. |
| *Iterations* | The test was conducted 5 times for the Agent 0.6.3. |
| *VICINITY components/functions involved* | • Adapter of a virtue testing device v0.0.1<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY neighborhood manager v0.6.3 |
| *Equipment and testing environments* | • None. |
| *Deployment* | • The adapter for the virtue testing device is established based on Python 3. It can be run on any PC by executing the .py file. |
| *Expected results* | • A response containing the information of "Discovery for adapter successfully done!" should be received after step 6 and step 13 respectively. The status code of the response should be 200. The registered devices should be found in the Neighbourhood manager. |
| *Test procedure* | The test procedure consists of the following steps:<br>1. Create an access point in Neighbourhood manager,<br>  ○ Choose VICINITY agent<br>  ○ Give it a name and set a password.<br>2. Use password and AID received after completing step 1 to setup /agent/gateway combo.<br>3. Start gateway<br>4. Start agent<br>5. Create 64 thing descriptions through python 3.6<br>6. Send HTTP request to agent to register thing through python 3.6,<br>  ○ Request: POST http://<agent_URL>:<agent_port>/agent/objects<br>  ○ Required payload is generated testing thing description<br>7. Stop agent<br>8. Stop gateway<br>9. Delete agent/config/db folder in agent to delete 64 things in Neighbourhood manager.<br>10. Start gateway<br>11. Start agent<br>12. Create 65 thing descriptions through python3.6<br>13. Send HTTP request to agent to register thing through python3.6,<br>  ○ Request: POST http://<agent_URL>:<agent_port>/agent/objects<br>  ○ Required payload is generated testing thing description<br>14. End of test. |

### 3.3.2. Testing Platform

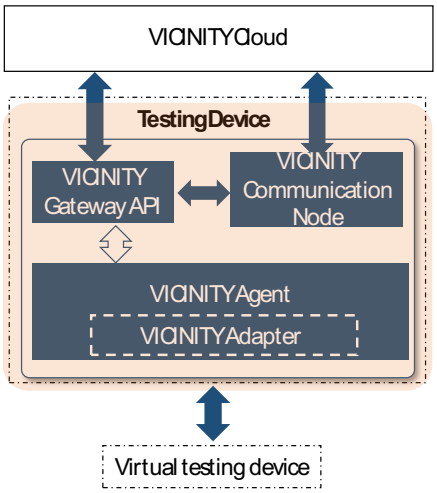



Figure 3 Testing platform and involved components for edge case testing 2.

### 3.3.3. Testing results

| Edge case testing 2 | Limit of parallel registrations |
|---|---|
| *Real results and constraints identified* | • After step 6, a response containing the information of successful registration is received, and the status code of the response is 200. The 64 devices can be found in the Neighbourhood manager.<br>• After step 13, a response containing the information of "Discovery for adapter failed" is received, and the status code of the response is 400. However, 65 devices can still be found in the Neighbourhood manager. The limitation of simultaneous registrations is found which is equal to 64 with light payload. |
| *Developed* | None. |
| *User Interfaces* | *Functionalities:* |
| *Real results (demo)* | Related snapshots of data flows are included in Annex II. |
| *Deviations* | • Although a response of "Discovery for adapter failed" is received and the status code of the response is 400, the 65 devices can still be found in Neighbourhood manager. |
| *Other technical issues* | None. |
| *Status* | Passed. |
| *Notes* | None. |

## 3.4. Edge Case Testing 3 - Large size of payload for GET request

### 3.4.1. Testing case design

| Edge case testing 3 | Large size of payload for GET request |
|---|---|
| *Test scenario and goal* | The current testing case aims to identify whether the VICINITY prototype can successfully respond to a GET request for heavy payload. |
| *Iterations* | The test was conducted 3 times for the Agent 0.6.3. |
| *VICINITY components/functions involved* | • Adapter of two virtue testing device v0.0.1<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • None. |
| *Deployment* | • The adapter for the virtue testing device is established based on Python 3. It can be run on any PC by executing the .py file. |
| *Expected results* | • Response to the request should be received. Postman should receive the response containing 200,000 data and 200 status code. |
| *Test procedure* | The test procedure consists of the following steps:<br>1. Create an access point in Neighbourhood manager<br>  ○ Choose VICINITY agent<br>  ○ Give it a name and set a password<br>2. Use password and AID received after completing step 1 to setup /agent/gateway combo.<br>3. Start gateway<br>4. Start agent<br>5. Create thing description for two devices through python 3.6, in which one device with a property emulates sender, the other device emulates receiver to read sender property.<br>6. Send HTTP request to agent to register two testing things through python 3.6,<br>  ○ Request: POST http://<agent_URL>:<agent_port>/agent/objects<br>  ○ Required payload is generated testing thing description<br>7. Run python 3.6 based adapter for the sender, it will send 200,000 string values as data.<br>8. Run Postman to emulate adapter for receiver and send HTTP request to agent to read sender's property.<br>  ○ Request: GET http://<agent_URL>:<agent_port>/agent/objects/<oid>/properties/maxpayloadtest<br>  ○ Required header is adapter ID and infrastructure ID of receiver.<br>9. End of test. |

### 3.4.2. Testing Platform

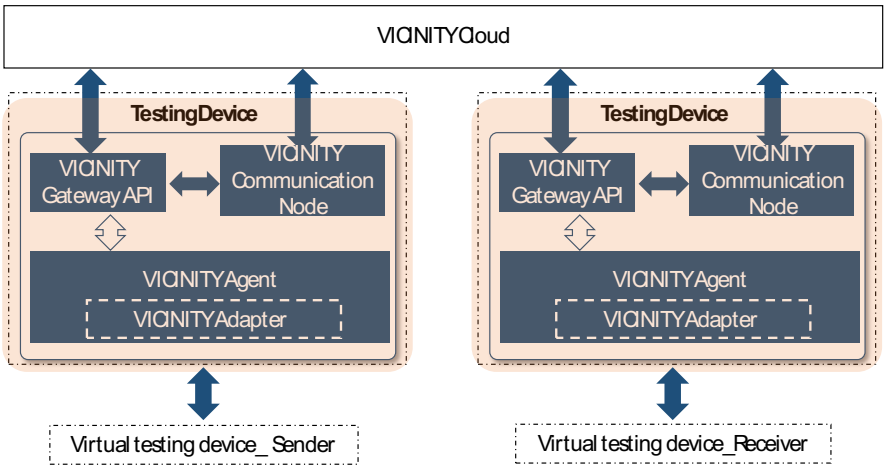



Figure 4 Testing platform and involved components for edge case testing 3.

### 3.4.3. Testing results

| . | **Large size of payload for GET request** |
|---|---|
| *Real results and constraints identified* | • After sending GET request by Postman, a response is received with 200,000 data and 200 status code. |
| *Developed User Interfaces* | None.<br>*Functionalities:* |
| *Real results (demo)* | Related snapshots of data flows are included in Annex III. |
| *Deviations* | None. |
| *Other technical issues* | None. |
| *Status* | Passed. |
| *Notes* | None. |

# 4. Internal point testing – Smart Parking & Residential Microgrid (AAU - DK)

## 4.1. Testing objective, Testing Environment and the Role of the Vicinity Prototype

In the development of traffic management systems, intelligent parking systems get a lot of attention in terms of sharing private and public parking space, reducing the cost of hiring people and for optimal use of resources for car-park owners. In line with Pilot Use Case 1b.1: Shared parking/priority parking defined in WP5, the Internal point testing 1 deals with providing users with data about the number of free parking slots and the real-time charging price for EVs. In line with Pilot Use Case 1b.2: eHealth Emergency parking listed in WP5, the Internal point testing 2 copes with providing a panic button function for the end-user by collecting the smart appliance properties.

Parking slot usage data is collected through VICINITY by using three parking sensors to achieve monitoring function. A residential microgrid, which consists of PV, a wind turbine and batteries, is emulated based on a real-time dSPACE experimental platform in AAU IoT-microgrid Lab. The residential microgrid is assumed to supply power to EV chargers in the three parking slots. GORENJE smart refrigerator is included in the residential microgrid. The real-time charging price is calculated by considering the simulated real-time utility electricity price, state-of-charge of batteries, and forecasts of the PV and wind turbine power generation. The parking slot usage and the real-time charging price will be sent automatically to users after subscribing Optimal Usage of Parking Slots by Considering Energy Costs VAS. The abnormal situation will be reported, and a parking slot will be reserved by Abnormal Situation Identification for Elderly Residents VAS. LabVIEW-based user interfaces are developed for monitoring and notification.

The VAS adapters, PlacePod parking sensor adapter, GORENJE appliance, Agent, Gateway API and all interaction patterns in VICINITY are tested during the VAS implementation process. Active and Passive Discovery of the Agent is used for the parking sensor adapter and the VAS respectively. The VAS can GET the properties of the parking sensor through VICINITY. The VASs subscribe the event published by the parking sensors and publish events to an end-user thus testing the publish/subscribe performance of VICINITY.

## 4.2. Internal point testing 1 - Optimal usage of parking slots by considering energy costs

### 4.2.1. Testing case design

| Internal point testing 1 | Optimal usage of parking slots by considering energy costs |
|---|---|
| *Test scenario and goal* | The current testing case deals with providing users with data about the number of free parking slots and the real-time charging price for EVs in order to optimize energy and parking slot usages and to reduce end-users' bills. |
| *Iterations* | The test was conducted 6 times for the Agent 0.6.2 and Agent 0.6.3 and consists of continuously running the residential microgrid for a period of 5 to 10 minutes. |
| *VICINITY components/functions involved* | • Adapter of PlacePod parking sensors v0.0.1<br>• Adapter of VAS - "Vacant parking slot and charging price notifications service" v1.0.0<br>• Agent v0.6.3 |

| | |
|---|---|
| | • Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • Three PlacePod parking sensors for Tromsø (NO) pilot site<br>• Multitech LoRaWAN gateway for Tromsø (NO) pilot site<br>• Microgrid emulation workstations in AAU lab<br>• Chroma Grid Simulator<br>• Local resistive loads<br>• LabVIEW-based residential microgrid energy management system (RMEMS) |
| *Deployment* | • The VAS is established based on Python 3, which is connected to LabVIEW-based Energy Management System through TCP/IP (http://localhost:10005). It can be run on any PC by executing the .py file. |
| *Expected results* | • A Residential microgrid setup that involves renewable energy resources (PV panels and a wind turbine), DC/AC power converters, energy storage systems, and three PlacePod parking sensors are properly running.<br>• Parking sensors, a testing device, and VAS registered can be viewed in the Neighbourhood Manager under "Devices" and "Services" menu items separately.<br>• VAS (service) node can make a request to its agent for data from the parking sensor (device) node.<br>• VAS node can subscribe to the event of the parking sensor node.<br>• The VAS monitors the parking slot usage and calculates the real-time charging price.<br>• Parking slots usage and the residential microgrid operation status are shown on a LabVIEW-based graphical user interface (GUI) 1 (as shown in Section 4.2.3).<br>• VAS publishes an event with parking slot usage and real-time charging price.<br>• The testing device node can subscribe to the event published by the VAS.<br>• Parking slots usage and the real-time charging price are shown on a simplified GUI 2 (as shown in Section 4.2.3). |
| *Test procedure* | The test procedure consists of the following steps:<br>• The tester first needs to build a residential microgrid with renewable energy resources, microgrid emulation setups, grid simulator, dSPACE real-time simulation platform, and PlacePod parking sensors.<br>• Register parking sensors, the Optimal Usage of Parking Slots by Considering Energy Costs VAS, and a testing device node in VICINITY Neighbourhood Manager with the same Organisation.<br>• The VAS node subscribes to the event of the parking sensor node.<br>• The VAS publishes an event with parking slot usage and real-time charging price.<br>• The testing device node becomes friends with the VAS.<br>• The testing device node subscribes to the event of the VAS.<br>• The tester monitors the operation status of the residential microgrid, the parking slot usage, and real-time charging price from GUIs 1 and 2.<br>• The tester occupies one parking sensor and turns on one load to emulate the EV charging process. The rest two parking slots are free. |

- The tester verifies that the energy balance calculation is conducted by considering the energy generation forecast and energy consumption for one occupied parking slot. Based on the calculation results, a parking and charging rate is announced for the rest two free parking slots.
- End of test.

### 4.2.2. Testing Platform

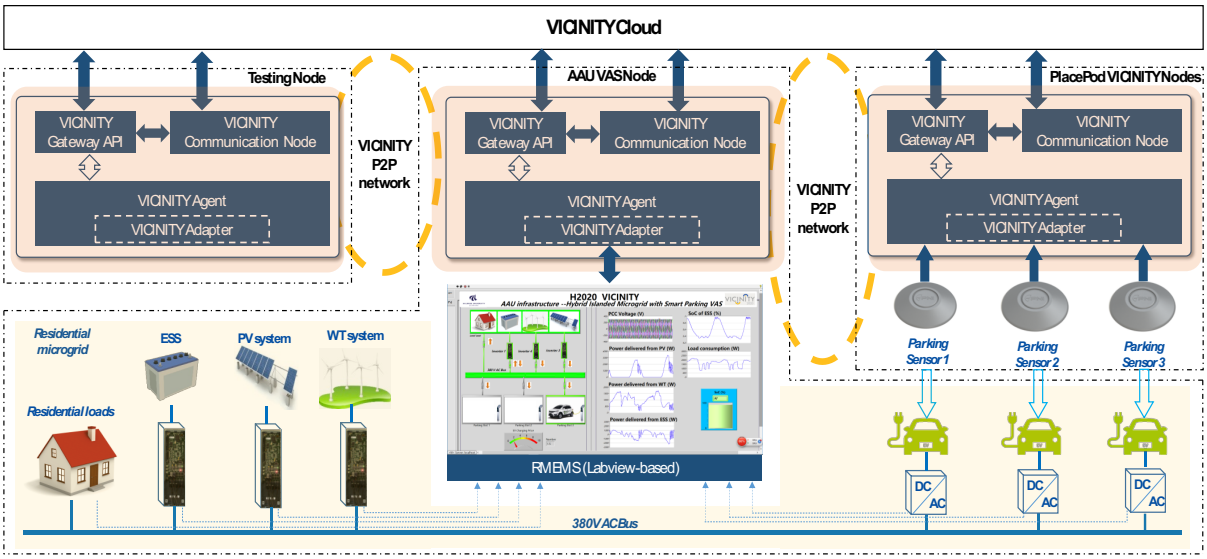
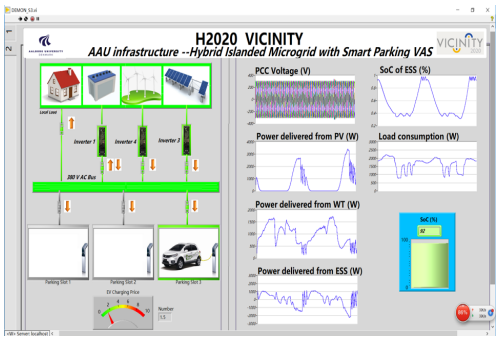



Figure 5 Testing platform and involved components for internal point testing 1.

### 4.2.3. Testing Results

| Internal point testing 1 | Optimal usage of parking slots by considering energy costs |
|---|---|
| *Real results* | • A Residential microgrid setup that involves renewable energy resources (PV panels and a wind turbine), DC/AC power converters, local loads, and three PlacePod parking sensors are properly running.<br>• Agent starts up without failing and successfully registers devices from configuration file with TDs in the VICINITY. Parking sensors, a testing device, and VAS registered can be viewed in the Neighbourhood Manager under "Devices" and "Services" menu items separately.<br>• Once the event is sent to subscribers of the VAS/testing device node, the publisher (parking sensor node/VAS) of the event gets a response with the success message and information about the event was sent to how many subscribers.<br>• The VAS node is able to receive the parking sensor node events, the data of the event contains the number of free parking slot and time-stamp.<br>• The testing device node is able to receive the VAS events, the data of the event contains the number of free parking slots, EV charging price, and time-stamp.<br>• Parking slot usage, EV charging price, and the residential microgrid operation status such as the power generation and state-of-charge of batteries are shown on GUI 1. |

| | |
|---|---|
| | • Parking slots usage and real-time EV charging price are shown on GUI 2. |
| *Developed User Interfaces* |  GUI 1  GUI 2 |
| | *Functionalities:*<br>The residential microgrid manager can monitor the operation performance of devices, energy resources, and the usage status of the parking lot.<br>The user can choose the preferred parking time based on the vacant parking slot number and real-time charging price. |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex IV**. |
| *Deviations* | 21.09.2018<br>Bug #46: Change of status trigger exception.<br>Every time a change of status in a parking sensor occurs, the field ParkingSensor[”sentralTime”] is out of range triggering an exception.<br>23.09.2018<br>Changed the type from Integer to Long and the issue has been solved. |
| *Other technical issues* | None. |
| *Status* | Passed after corrections. |
| *Notes* | None. |

## 4.3. Internal point testing 2 - Abnormal situation identification for elderly residents

### 4.3.1. Testing case design

| **Internal point testing 2** | **Abnormal situation identification for elderly residents** |
|---|---|
| *Test scenario and goal* | Identify abnormal situations, for instance, a refrigerator’ door has been left open more than normal time and trigger notifications to a care provider (a testing node) and reserve a free parking slot for an ambulance. |
| *Iterations* | The test was conducted 5 times for the Agent 0.6.3. |

| | |
|---|---|
| *VICINITY components/functions involved* | • Adapter of PlacePod parking sensors v0.0.1<br>• Adapter of VAS - "Abnormal situation identification for elderly residents" v1.0.0<br>• Cloud-based Adapter of GORENJE smart refrigerator #7 v1.0.0<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • Three PlacePod parking sensors for Tromsø (NO) pilot site<br>• Multitech LoRaWAN gateway for Tromsø (NO) pilot site<br>• GORENJE smart refrigerator #7<br>• LabVIEW-based RMEMS |
| *Deployment* | • The VAS is established based on Python 3, which is connected to LabVIEW-based GUI through TCP/IP (http://localhost:10005). It can be run on any PC by executing the .py file. |
| *Expected results* | • Parking sensors, GORENJE smart refrigerator #7, a testing device and the VAS registered can be viewed in the Neighbourhood Manager under "Devices" and "Services" menu items separately.<br>• VAS node can subscribe to the events of the parking sensor node and GORENJE refrigerator #7.<br>• The VAS monitors the refrigerator door status and the parking lot usage.<br>• Refrigerator door status and parking lot usage are shown on a LabVIEW-based GUI.<br>• Once the refrigerator door has been left open than 15 minutes, an alarm light turns red in the GUI and the VAS publishes an event with abnormal situation notification.<br>• The testing device node can subscribe to the event published by the VAS. |
| *Test procedure* | The test procedure consists of the following steps:<br>• Register parking sensors, GORENJE smart refrigerator #7, the Abnormal situation identification for elderly residents VAS, and a testing device node in VICINITY Neighbourhood Manager with different Organisations.<br>• GORENJE smart refrigerator #7 establish the friendship with Abnormal situation identification for elderly residents VAS.<br>• The VAS node subscribes to the event of the parking sensor node and GORENJE smart refrigerator #7.<br>• The VAS publishes an event with parking slot usage and abnormal situation alarm.<br>• The testing device node becomes friends with the VAS.<br>• The testing device node subscribes to the event of the VAS.<br>• The tester monitors the parking slot usage and refrigerator status from a GUI.<br>• The tester leaves the refrigerator up door open for 15 minutes.<br>• End of test. |

### 4.3.2. Testing Platform

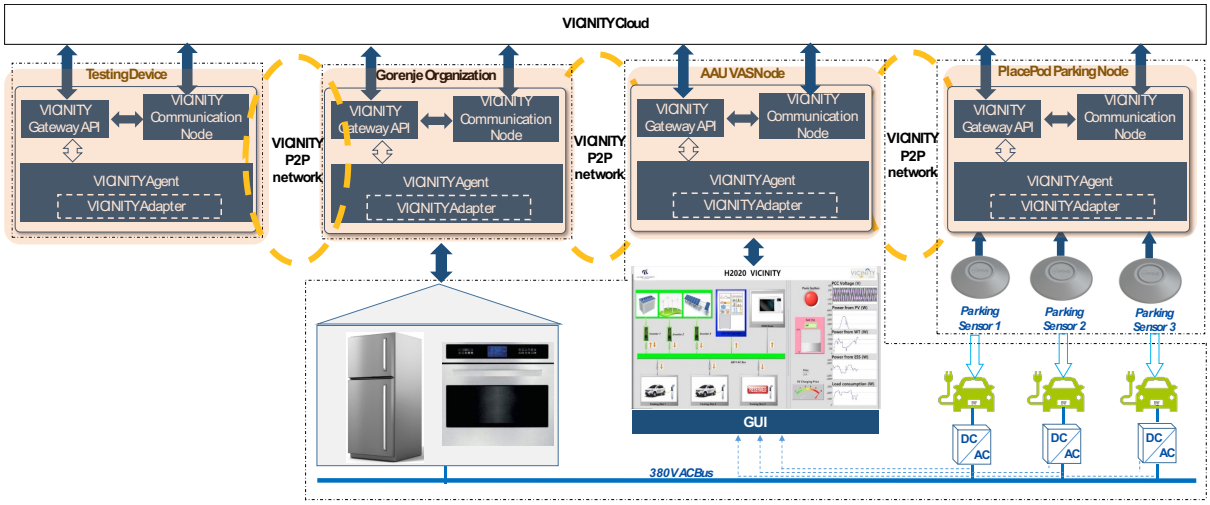
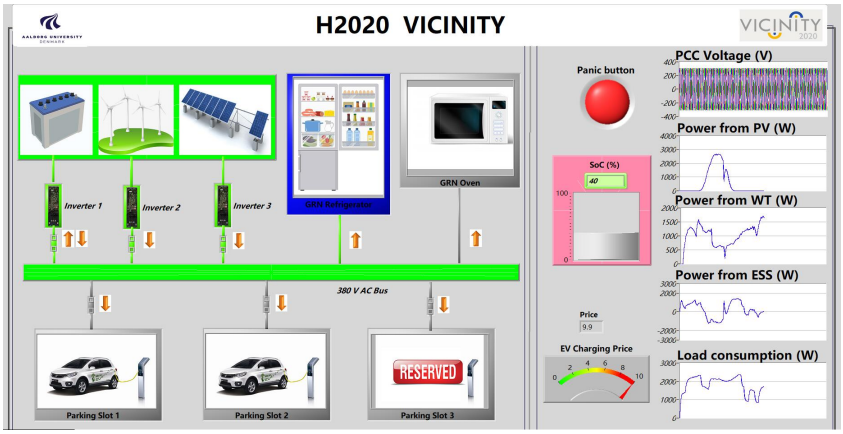



Figure 6 Testing platform and involved components for internal point testing 2.

### 4.3.3. Testing Results

| Internal point testing 2 | Abnormal situation identification for elderly residents |
|---|---|
| *Real results* | • Agent starts up without failing and successfully registers devices from configuration file with TDs in the VICINITY. Parking sensors, GORENJE smart refrigerator #7, a testing device, and the VAS registered can be viewed in the Neighbourhood Manager under "Devices" and "Services" menu items separately.<br>• The VAS node is able to receive the event of GORENJE refrigerator #7 which contains the door status.<br>• The VAS node is able to publish an event to the testing device for an abnormal situation.<br>• Parking slots usage and refrigerator status such as the refrigerator' door open/close status are shown on the GUI.<br>• Once the refrigerator' door is opened more than 15 minutes, a red alarm light (Panic button) is turned on in the GUI and the VAS triggers an event to the testing node. Meanwhile, a free parking slot is reserved for the potential ambulance. |
| *Developed User Interfaces* |  |
| *Functionalities:* | |

| | |
|---|---|
| | The Care Center can monitor the refrigerator door status, panic button, and the usage of the parking lot. |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex V**. |
| *Deviations* | None. |
| *Other technical issues* | None. |
| *Status* | Passed. |
| *Notes* | None. |

# 5. Internal point testing – Smart Building & Residential Microgrid (AAU - DK)

## 5.1. Testing objective, Testing Environment and the Role of the Vicinity Prototype

By means of smart sensors and devices, activities can be automatically detected and identified. By comparing these sensing data with the recorded behaviour patterns, many services and controls can be achieved, such as energy cost notification and accident detection. In line with Pilot Use Case 1a.1 – Predictive operations defined in WP5, the Internal point testing 3 deals with providing users with data about the room usage and cleaning notification. In line with Pilot Use Case 1a.2 – Resource management, the Internal point testing 4 copes with energy consumption abnormal alarm for the end-user in a residential microgrid.

Room usage data is collected through VICINITY by using one Tinymesh door sensor. If the room usage data is over a pre-set threshold, a cleaning notification will be triggered by the VAS. The energy consumption data is collected through an emulated residential microgrid which includes PV, a wind turbine, batteries, a GORENJE smart oven, and a GORENJE refrigerator. An energy cost alarm will be triggered by Energy consumption abnormal VAS if the energy consumption reaches the nominal level. LabVIEW-based GUIs are used for monitoring and notification.

The VAS adapters, Tinymesh door sensor adapter, GORENJE appliances, Agent, Gateway API and all interaction patterns in VICINITY are tested during the VAS implementation process. The VAS can GET the properties of the door sensor through VICINITY. The VAS subscribes to the events published by the door sensors and GORENJE appliances. The VAS also publishes events to an end-user thus testing the publish/subscribe performance of VICINITY.

## 5.2. Internal point testing 3 - Predictive operations_Cleaning and Waste Removal Notification

### 5.2.1. Testing case design

| **Internal point testing 3** | **Predictive operations_Cleaning and Waste Removal Notification** |
|---|---|
| *Test scenario and goal* | The door sensor registers if a person passes the door (in an anonymized way) and can thus keep tracking of the approximate number of room been visited. When the number reaches the threshold, a cleaning notification will be reported. |
| *Iterations* | The test was conducted 2 times for the Agent 0.6.3. |
| *VICINITY components/functions involved* | • Adapter of Tinymesh door sensor v0.0.1<br>• Adapter of VAS - “Cleaning Notification” v1.0.0<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • One Tinymesh door sensor for Oslo (NO) pilot site<br>• Raspberry Pi-based Tinymesh gateway for Oslo (NO) pilot site<br>• LabVIEW-based GUI |

| | |
|---|---|
| *Deployment* | The VAS is established based on Python 3, which is connected to LabVIEW-based GUI through TCP/IP (http://localhost:10005). It can be run on any PC by executing the .py file. |
| *Expected results* | • The Tinymesh door sensor, a testing device, and the Cleaning Notification VAS registered in the Neighbourhood Manager.<br>• VAS node can subscribe to the event of door sensor node.<br>• The VAS monitors the door sensor on/off count.<br>• Room usage is shown on the GUI.<br>• If the room usage data collected from the door sensor is over 10 times, the VAS publishes an event with the room usage count and a cleaning notification.<br>• The testing device node can subscribe to the event published by the VAS. |
| *Test procedure* | The test procedure consists of the following steps:<br>• Register the door sensor, Cleaning Notification VAS, and a testing device node in the VICINITY Neighbourhood Manager.<br>• The VAS node subscribes to the event of the door sensor.<br>• The VAS publishes an event with room usage count and a cleaning notification.<br>• The testing device node becomes friends with the VAS and subscribes to the event of the VAS.<br>• The tester monitors the room usage from the GUI.<br>• End of test. |

### 5.2.2. Testing Platform

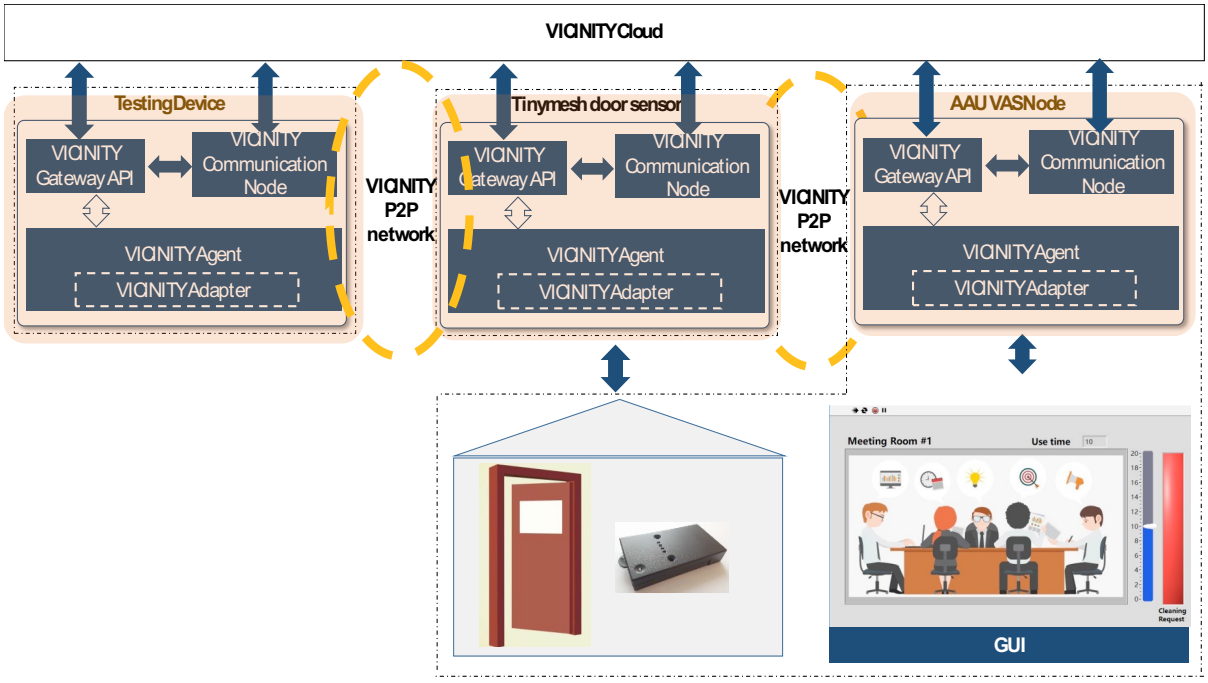
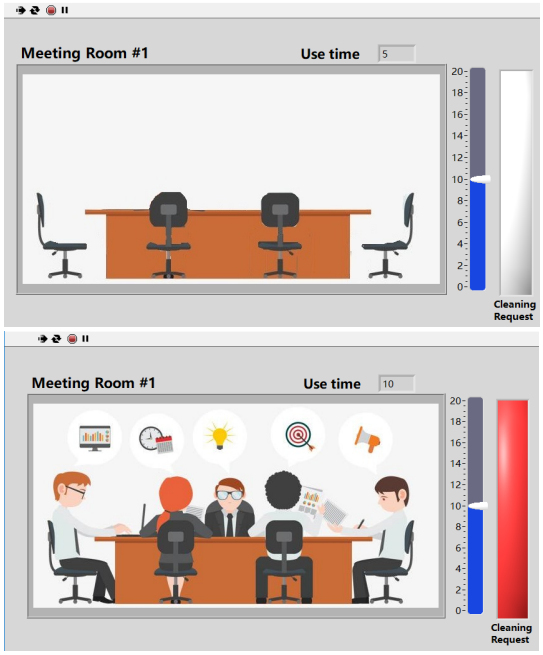



Figure 7 Testing platform and involved components for internal point testing 3.

### 5.2.3. Testing Results

| Internal point testing 3 | Predictive operations_Cleaning and Waste Removal Notification |
|---|---|
| *Real results* | • Agent starts up without failing and successfully registers a Tinymesh door sensor, a testing device, and the VAS from configuration file with TDs in the VICINITY.<br>• The VAS node is able to receive an event of Tinymesh door sensor which contains the door status.<br>• The VAS node is able to publish an event to the testing device for cleaning notification and room usage count.<br>• Room usage is shown on the GUI.<br>• Once the room usage count is over 10 times, a red cleaning notification light is turned on in the GUI and the VAS publishes an event to the testing node. |
| *Developed*<br><br>*User Interfaces* |  |
| | *Functionalities:*<br><br>The user can monitor room usage status and view a notification when the room usage amount reaches the pre-set threshold. |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex VI**. |
| *Deviations* | The door sensor has some latency for the status changes. |
| *Other technical issues* | The quality of the door sensor needs to be improved for the accuracy of the measurements. |
| *Status* | Passed. |
| *Notes* | Change event subscribe manner from dynamic to static. |

## 5.3. Internal point testing 4 - Energy consumption optimization and abnormal alarm

### 5.3.1. Testing case design

| Internal point testing 4 | Energy consumption optimization and abnormal alarm |
|---|---|
| *Test scenario and goal* | The energy management system of a residential microgrid optimizes microgrid operation to reduce energy cost. In case that the energy consumption exceeds desired thresholds, for instance, continuously baking, an energy consumption abnormal alarm will be triggered. |
| *Iterations* | The test was conducted 3 times for the Agent 0.6.3. |
| *VICINITY components/functions involved* | • Adapter of GORENJE smart oven #7 v1.0.0<br>• Adapter of VAS - "Energy consumption abnormal" v1.0.0<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • GORENJE smart oven #7<br>• Microgrid emulation workstations in AAU lab<br>• Chroma Grid Simulator<br>• Local resistive loads<br>• Energy management system |
| *Deployment* | • The VAS is established based on Python 3, which is connected to LabVIEW-based Energy Management System through TCP/IP (http://localhost:10005). It can be run on any PC by executing the .py file. |
| *Expected results* | • A Residential microgrid setup that involves renewable energy resources (PV panels and a wind turbine), DC/AC power converters, and a GORENJE smart oven is properly running.<br>• GORENJE smart oven #7, a testing device and VAS registered can be viewed in the Neighbourhood Manager.<br>• VAS node can make an action request to its agent to start oven baking function.<br>• VAS node can subscribe to the event of oven device status.<br>• The VAS monitors the microgrid operation and oven status and shows the data on a GUI.<br>• VAS publishes an event with energy consumption abnormal notification.<br>• The testing device node can subscribe to the event published by the VAS. |
| *Test procedure* | The test procedure consists of the following steps:<br>• The tester first needs to build a residential microgrid.<br>• Register GORENJE oven #7, Energy Consumption Notification VAS, and a testing device node in VICINITY Neighbourhood Manager with different organisations.<br>• The GORENJE organisation establishes the friendship with Energy Consumption Notification VAS.<br>• The VAS node subscribes to the event of oven #7 device status.<br>• The VAS publishes an event with energy consumption abnormal notification.<br>• The testing device node subscribes to the event of the VAS. |

- The tester monitors the operation of the residential microgrid and the oven status from the GUI.
- The tester let the oven baking for 10 minutes.
- End of test.

### 5.3.2. Testing Platform

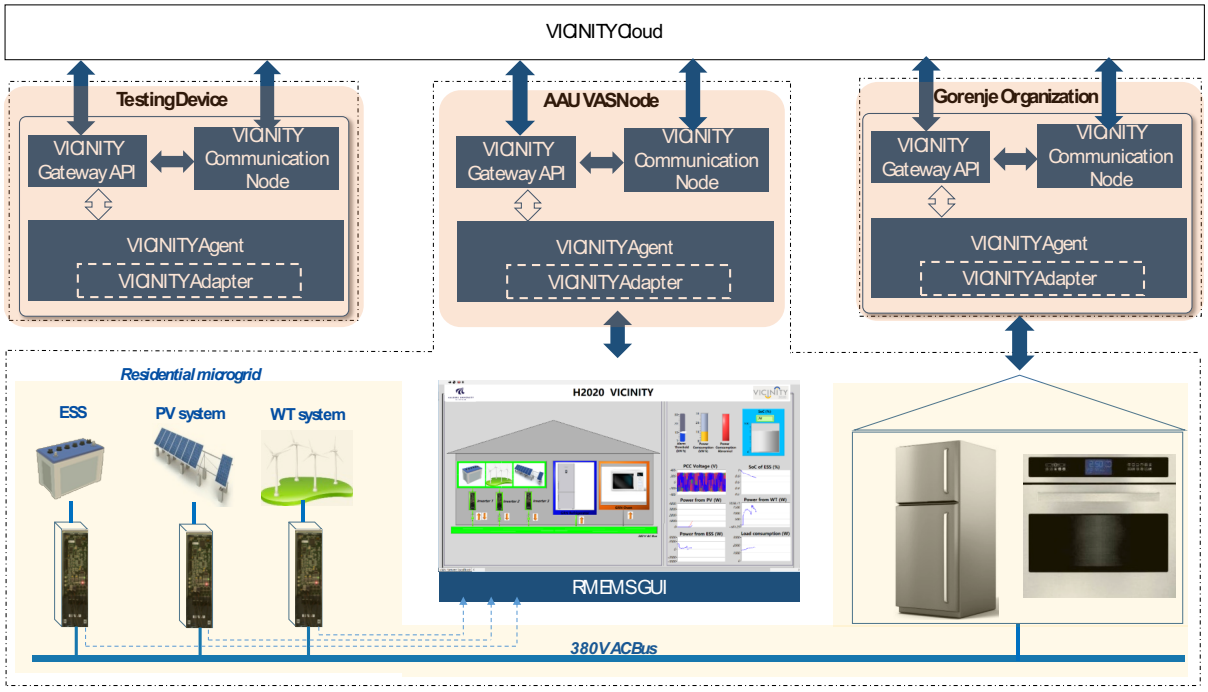
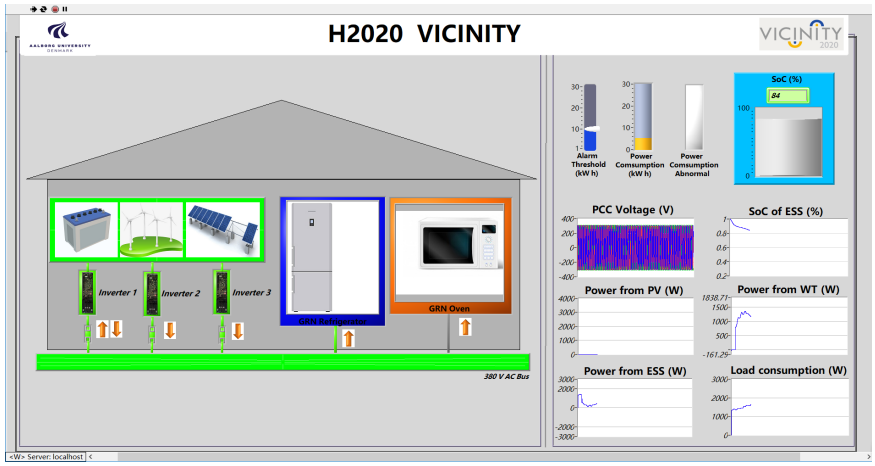
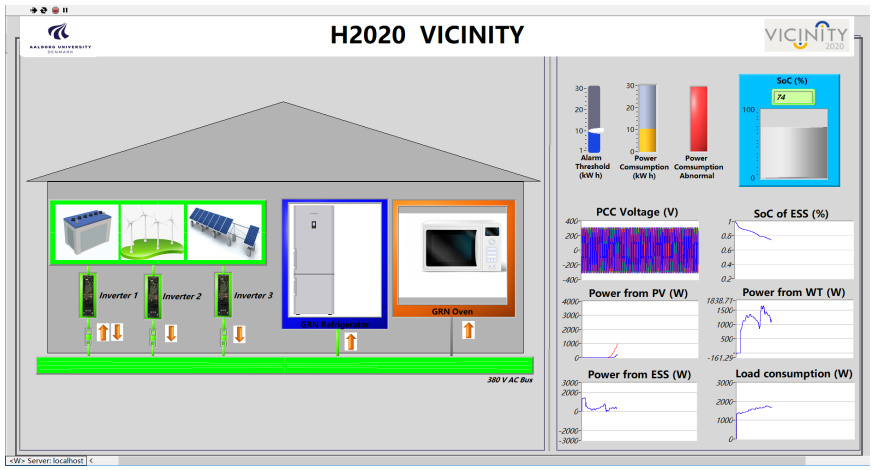



Figure 8 Testing platform and involved components for internal point testing 4.

### 5.3.3. Testing results

| Internal point testing 4 | Energy consumption optimization and abnormal alarm |
|---|---|
| *Real results* | • A Residential microgrid setup is properly running.<br>• GORENJE oven #7 is registered in the Neighbourhood Manager under "Devices" menu item with GORENJE organisation.<br>• Agent starts up without failing and successfully registers the testing device and Energy Consumption Notification VAS from configuration file with TDs in the VICINITY Neighbourhood Manager with AAU organisation.<br>• The contract is established between the two organisations.<br>• The VAS node is able to receive the oven status event with the data of name, status string (Running, Idle, and Pause) and time-stamp.<br>• The testing device node is able to receive the VAS events, the data of the event contains the energy consumption abnormal notification and time-stamp.<br>• Microgrid operation and oven status are shown on the GUI.<br>• If the oven is continuously baking over 10 minutes, the total energy consumption will exceed the pre-set threshold, therefore trigger the event. |

| | |
|---|---|
| *Developed User Interfaces* |  <br>  <br>■<br>*Functionalities:*<br>The user can monitor the power generation, energy consumption, and oven status. |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex VII**. |
| *Deviations e* | None. |
| *Other technical issues* | Remote baking only works when the VAS agent just started. It is fixed by GORENJE by setting the previous task status to finished/cancelled. |
| *Status* | Passed after corrections. |
| *Notes* | None. |

# 6. Internal point testing – Smart Residential Microgrid Energy Management (AAU - DK)

## 6.1. Testing objective, Testing Environment and the Role of the Vicinity Prototype

Microgrids are energy systems that aggregate distributed energy resources, loads, and power electronics devices in a stable, optimal and balanced way. Energy management is central to the concept of a microgrid in order to achieve substation monitoring, improve energy efficiency and demand profile, reduce utility and economic cost, and optimize coordinative operation. In line with Pilot User Case 2.1-2.5, 2.10, 2.11, and 1a.2, Internal point testing 5 - Optimal Scheduling and Operation Energy Management is conducted to validate a peak demand shifting and scheduling, thereby maintaining the reliable power supply and reducing the resident's bills. In line with Pilot User Case 2.9 – UV (Ultraviolet radiation) info services for Citizens and Tourists – Local to Local Services, Internal point testing 6 is designed for providing solar irradiance forecast.

The energy consumption data is collected through an emulated residential microgrid which includes PV, a wind turbine, batteries, a GORENJE smart oven, and a refrigerator. A LabVIEW-based energy management system is developed to achieve optimized control for energy resources and local loads and to perform a scheduling function. The Optimal Scheduling and Operation Energy Management VAS subscribes to the event published by GORENJE appliances and send actions (baking and delay) to the appliances. The Solar Irradiance Forecast VAS provides a short-term prediction of solar irradiance for subscribers.

The VAS adapters, GORENJE appliances, Agent, Gateway API and all interaction patterns in VICINITY are tested during the VAS implementation process. The VASs subscribe to the event published by GORENJE appliances and remotely control them. The VASs can post commands to the appliances, therefore testing the action performance. The VAS also publishes events to an end-user thus testing the publish/subscribe performance of VICINITY.

## 6.2. Internal point testing 5 - Optimal Load Scheduling and Microgrid Operation

### 6.2.1. Testing case design

| **Internal point testing 5** | **Optimal Load Scheduling and Microgrid Operation** |
|---|---|
| *Test scenario and goal* | Maintain power balance and reduce electricity cost by encouraging residential customers to shift loads according to the renewable energy generation. |
| *Iterations* | The test will be conducted 3 times. |
| *VICINITY components/functions involved* | • Adapter of GORENJE smart oven #7 v1.0.0<br>• Adapter of GORENJE smart refrigerator #7 v1.0.0<br>• Adapter of VAS - "Optimal Load Scheduling and Microgrid Operation" v1.0.0<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood Manager v0.6.3 |

| | |
|---|---|
| *Equipment and testing environments* | • GORENJE smart oven #7<br>• GORENJE smart refrigerator #7<br>• Microgrid emulation workstations in AAU lab<br>• Chroma Grid Simulator<br>• Local resistive loads<br>• Energy management system |
| *Deployment* | • The VAS is established based on Python 3, which is connected to LabVIEW-based Energy Management System through TCP/IP (http://localhost:10005). It can be run on any PC by executing the .py file. |
| *Expected results* | • A Residential microgrid setup that involves renewable energy resources (PV panels and a wind turbine), DC/AC power converters, energy storage systems, a GORENJE smart oven, and a GORENJE smart refrigerator is properly running.<br>• GORENJE oven #7, GORENJE refrigerator #7, and VAS registered can be viewed in the Neighbourhood Manager.<br>• VAS node can make action requests to its agent to start oven baking function and start refrigerator Fastfreeze function.<br>• VAS node can subscribe to the event of oven device status and get the properties of the refrigerator.<br>• The VAS monitors the microgrid operation and oven/refrigerator status and shows the data on a GUI. |
| *Test procedure* | The test procedure consists of the following steps:<br>• The tester first needs to build a residential microgrid.<br>• Register GORENJE oven #7, refrigerator #7, and Optimal Scheduling and Operation Energy Management VAS in VICINITY Neighbourhood Manager with different organisations.<br>• The GORENJE organisation establishes the friendship with the VAS.<br>• The VAS node sends POST commands to the oven for starting baking.<br>• The VAS node turns on the Fastfreeze function of the refrigerator.<br>• The tester monitors the operation status of the residential microgrid, oven and refrigerator from the GUI.<br>• End of test. |

### 6.2.2. Testing Platform

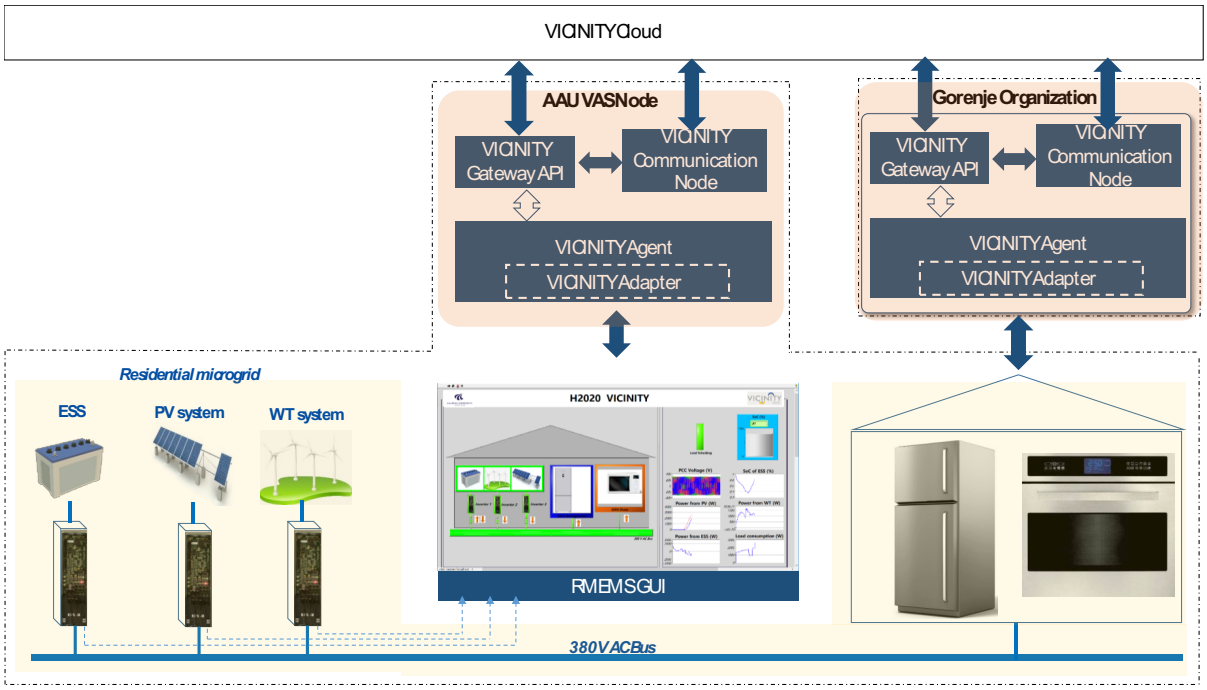
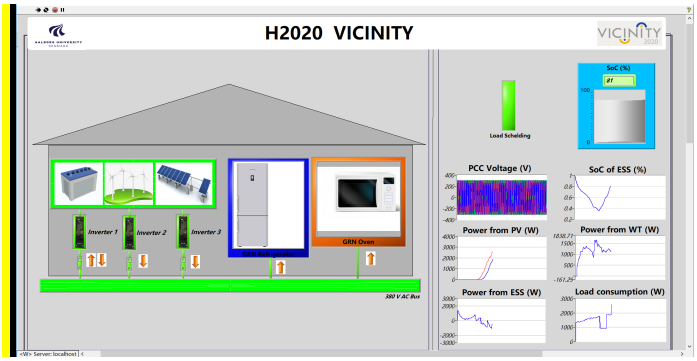



Figure 9 Testing platform and involved components for internal point testing 5.

### 6.2.3. Testing results

| Internal point testing 5 | Optimal Load Scheduling and Microgrid Operation |
|---|---|
| *Real results* | • A Residential microgrid setup is properly running.<br>• Agent starts up without failing and successfully registers the GORENJE oven #7, refrigerator #7, and the VAS in the Neighbourhood Manager with GORENJE and AAU organisations respectively.<br>• The VAS node is able to start oven baking action by posting the command and can receive the oven status event with the data of name, status string (Running, Idle, and Pause) and time-stamp.<br>• The VAS node is able to put the refrigerator's working status to Fastfreeze and to read the property.<br>• Once the solar and wind energy are abundant during the next 20 minutes according to the forecast, the VAS sends a baking start command to the oven and put the refrigerator's property to Fastfreeze to take full advantages of renewable energies. |
| *Developed User Interfaces* |  |

| | |
|---|---|
| | *Functionalities:*<br>The user can monitor the microgrid operation and smart appliances working status. |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex VIII**. |
| *Deviations* | None. |
| *Other technical issues* | Remote baking only works when the VAS agent just started. It is fixed by GORENJE by setting the previous task status to finished/cancelled. |
| *Status* | Passed after correction. |
| *Notes* | None. |

## 6.3. Internal point testing 6 – Solar irradiance forecast

### 6.3.1. Testing case design

| **Internal point testing 6** | **Solar irradiance forecast** |
|---|---|
| *Test scenario and goal* | Provide a short-term prediction of solar irradiance for residences and utility who have PV panels to enhance the energy management system capability. |
| *Iterations* | The test will be conducted 3 times. |
| *VICINITY components/functions involved* | • Adapter of VAS - "Solar irradiance forecast" v1.0.0<br>• Agent v0.6.3<br>• Gateway API v0.6.3<br>• VICINITY Neighbourhood manager v0.6.3 |
| *Equipment and testing environments* | • Microgrid emulation workstations in AAU lab<br>• Chroma Grid Simulator<br>• Local resistive loads<br>• Energy management system |
| *Deployment* | • The VAS is established based on Python 3, which is connected to LabVIEW-based Energy Management System through TCP/IP (http://localhost:10005). It can be run on any PC by executing the .py file. |
| *Expected results* | • A Residential microgrid setup that involves renewable energy resources (PV panels and a wind turbine) and DC/AC power converters is properly running.<br>• A testing device node and the VAS are registered in VICINITY.<br>• VAS node can publish an event with solar irradiance forecast in 15 minutes and the testing device node can subscribe to the event. |
| *Test procedure* | The test procedure consists of the following steps:<br>• The tester first needs to build a residential microgrid.<br>• Register Solar irradiance forecast VAS and a testing device in VICINITY Neighbourhood Manager.<br>• The testing device node subscribes to the event published by the VAS.<br>• End of test. |

### 6.3.2. Testing Platform

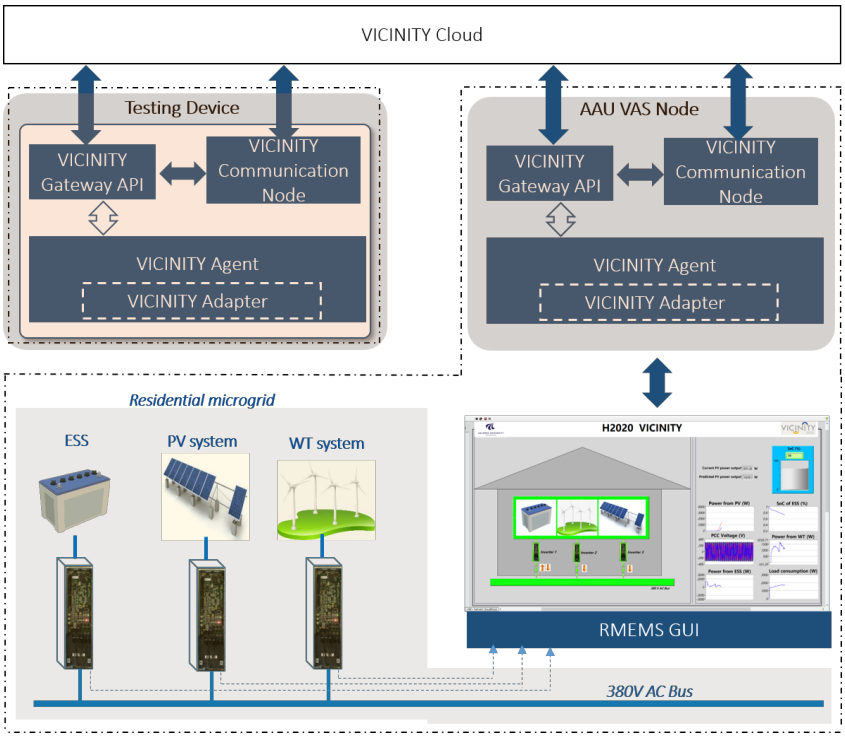
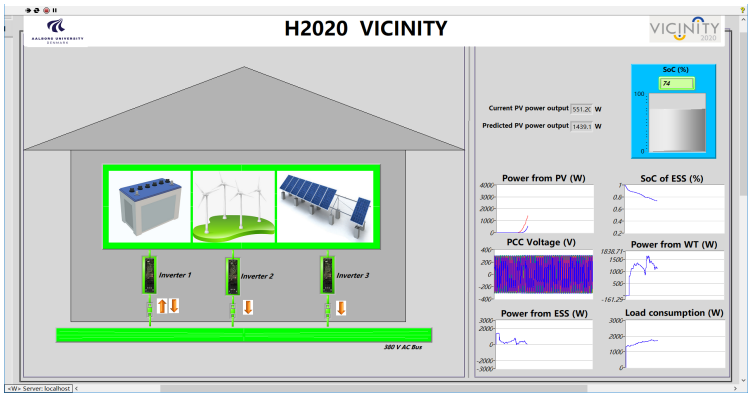



Figure 10 Testing platform and involved components for internal point testing 6.

### 6.3.3. Testing Results

| Internal point testing 6 | Solar irradiance forecast |
|---|---|
| *Real results* | • A Residential microgrid setup is properly running.<br>• Agent starts up without failing and successfully registers the testing device and the VAS in the Neighbourhood Manager.<br>• The VAS node is able to publish an event with the data of short-term solar irradiance forecast which is calculated by energy management system.<br>• The testing device node is able to subscribe to the published VAS. |
| *Developed*<br><br>*User Interfaces* |  |
| | *Functionalities:*<br><br>Allow the residence and utility plan their power demand and load scheduling based on the solar irradiance prediction. |

| | |
|---|---|
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex IX**. |
| *Deviations* | None. |
| *Other technical issues* | None. |
| *Status (Passed/Passed after corrections/Failed)* | Passed. |
| *Notes* | None. |

# 7. Internal point testing – CERTH/ITI Smart House (smart living and eHealth at Home) (CERTH, GNOMON - GR)

## 7.1. Testing objective, Testing Environment and the Role of the Vicinity Prototype

CERTH/ITI Smart house is an excellent candidate for conducting tests for VICINITY IoT Platform, since it can offer an environment similar to the elder citizens' houses of MPH. The Smart House was used both for conducting T6.1 and T6.2 tests before actual deployment. The scope of T6.2 tests is to validate the developed VICINITY components in cases further than the simple integration that took place in T6.1.

VICINITY creates virtual Neighbourhoods of devices and services, which can interact with one another under a common language. The Use Case at MPH premises will be consisted of many virtual Neighbourhoods. Thus, it is very important to ensure the privacy of data, by ensuring that data can be seen and transmitted to certain services inside a virtual Neighbourhood, only if access is granted to these services. This will be the scope of Test 7.

Moreover, MPH is a large-scale use case which will integrate many different infrastructures (Organisations) to VICINITY. While in T6.1 we test the simple integration of an infrastructure to VICINITY, in T6.2 we test the integration of a big number of different IoT infrastructures, in terms of Organisation creation, device registration, friendships and contracts. An automated procedure has been developed in order to ease the integration of each elder home and middle-aged citizen, which is also tested. This will be the scope of Test 8.

## 7.2. Internal point testing 7 - Privacy testing using a Smart Home scenario

### 7.2.1. Testing case design

| **Internal point testing 7** | **Privacy testing using a Smart Home scenario** |
|---|---|
| *Test scenario and goal* | This test focuses on the testing of VICINITY Neighbourhood Manager, Agent and Gateway, in terms of privacy for the use case 3.1 regarding the elder citizens homes. The privacy is tested in three different ways. |
| *Iterations* | The test will be conducted five times. |
| *VICINITY components/functions involved* | • Neighbourhood Manager v0.6.3<br>• Agent v0.6.3.1<br>• Gateway API v0.6.3.1<br>• Adapter for building sensors based on IoTivity Platform v0.0.1<br>• Adapter for Gorenje fridge and oven v1.0.0<br>• Testing VAS v0.0.1 |
| *Equipment and testing environments* | • Motion sensors<br>• Door sensor<br>• Pressure sensor<br>• Panic Button<br>• Gorenje Fridge<br>• Gorenje Oven |
| *Deployment* | • The building sensors adapter is deployed on a Raspberry Pi, similarly to what will be deployed in the elder house. |

| | |
|---|---|
| | • The adapter for Gorenje devices is deployed on Gorenje Cloud Infrastructure.<br>• The testing VAS is deployed on local server. |
| *Expected results* | • Motion, door and bed sensors are expected to be able to send new values to the testing VAS when there is a contract between them but not when the contract is removed.<br>• Subscription to Gorenje devices events should not be possible if a contract does not exist between the devices and the subscriber (VAS). |
| *Test procedure* | The test procedure consists of the following steps:<br>*Privacy testing (a)*:<br>• Two VICINITY Organisations (e.g. A, B) are created through Neighbourhood Manager, each one having one VICINITY Agent. The Organisation A owns a group of devices in the smart house while the Organisation B is the service provider.<br>• The gateway API and the multi-tenant agent are running on a server.<br>• The Organisation of the VAS (B) will become friends with Organisation A.<br>• The VAS will make a contract with group's A devices.<br>• Motion, door, bed sensors and panic button of Organisation A will be triggered in order to send new values to the testing VAS.<br>• The contract between devices of group A and the VAS will be removed from the Neighbourhood manager.<br>• Motion, door, bed sensors and panic button of Organisation A will be triggered in order to send new values to the testing VAS. The operation should no longer be possible.<br>*Privacy testing (b)*:<br>• The Organisation of the VAS (B) will become friends with Organisation of Gorenje.<br>• The VAS declares the subscription to the Gorenje devices door events in agent configuration file.<br>• The VAS should be denied subscription, since there is no contract. The fridge/oven door is opened. The VAS should not be able to get any events yet.<br>• The VAS will make a contract with the two Gorenje devices.<br>• The fridge/oven door is opened. The VAS should be able to get events.<br>*Privacy testing (c)*:<br>• Update the TD of the VAS.<br>• Trigger sensors in order to send measurement to VAS. (this should not be possible)<br>• Re-accept contracts.<br>• End of test. |

### 7.2.2. Testing Platform

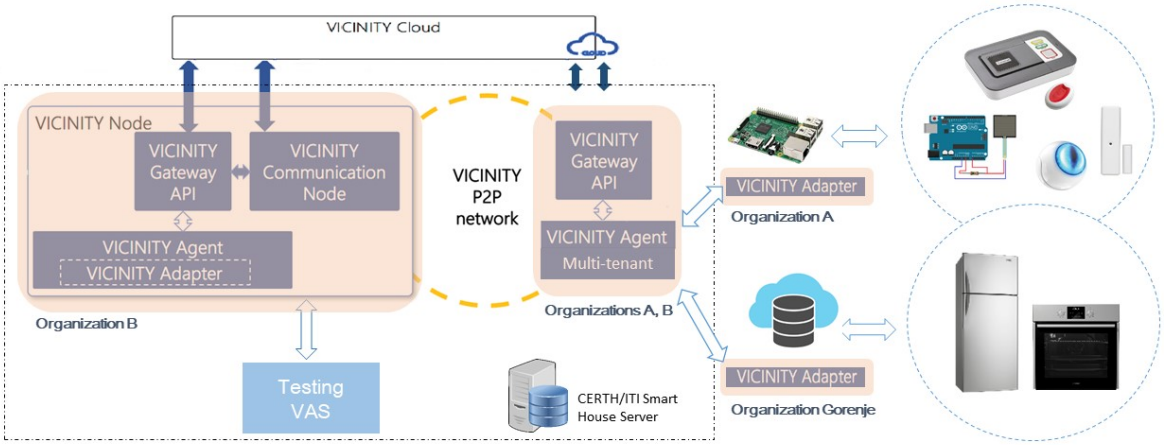



Figure 11 Testing platform and involved components for internal point testing 7.

### 7.2.3. Testing results

| Internal point testing 7 | Privacy testing using a Smart Home scenario |
|---|---|
| *Real results* | • All sensor values of Organisation A reach VAS of Organisation B when there is an active contract and cannot reach it when contract is removed.<br>• The VAS is not able to subscribe to devices if there is not an active contract between them. Events from Gorenje devices reach VAS after successful subscription.<br>• Transfer of measurement is not possible after update in the TD of the VAS. |
| *Developed User Interfaces* | *Functionalities:* |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex X**. |
| *Deviations* | No deviations |
| *Other technical issues* | During this procedure the feature of having the ability to update items in VICINITY was proposed by CERTH and GNOMON. In discussion with BVR and IS the contracts behaviour in such case was decided. |
| *Status* | Passed |
| *Notes* | Privacy testing guarantees that whenever a citizen wishes to no longer share his/her device data through VICINITY, he/she is able to do so by removing the contract between the devices and the VAS. (Should not be confused with the testing of the integration of the infrastructure and the communication with the VAS which has been tested in T6.1.) |

| | |
|---|---|
| | Privacy testing b is important since many VICINITY Organisations will be friends with Gorenje Organisation but only the VAS that has contract with a specific device can see its properties/actions/events. This means that other partners Organisations can see the devices that are deployed in the elder citizen house when looking at the Neighbourhood Manager interface, but they cannot access them through VICINITY P2P network. (Should not be confused with the testing of VICINITY event functionality which is done in T6.1.)<br><br>Privacy testing c guarantees that the citizen gives his/her data only for the service operations upon which he/she has agreed (by accepting contract). If a service exposes new operations to VICINITY, the citizen will need to give new permission to the service for his/her data. (Should not be confused with the testing of the integration of the infrastructure and the communication with the VAS which has been tested in T6.1.) |

## 7.3. Internal point testing 8 – Large scale integration of eHealth infrastructures

### 7.3.1. Testing case design

| **Internal point testing 8** | **Large scale integration of eHealth infrastructures** |
|---|---|
| *Test scenario and goal* | This test focuses on testing of automatic integration of a big number of infrastructures into VICINITY for both use case 3.1 and 3.2. |
| *Iterations* | More than 50 times |
| *VICINITY components/functions involved* | • Neighbourhood Manager API v0.6.3<br>• Agent v0.6.3.1<br>• Service for automatic registration to VICINITY<br>• Adapter for building sensors based on IoTivity Platform v0.0.1<br>• Adapter for medical devices based on NodeRed Platform v0.0.1<br>• Adapter for middle-aged citizen device integrated in EHealthPass App v0.0.1<br>• Storage and GDPR VAS v1.0.0 |
| *Equipment and testing environments* | • Blood-pressure monitor<br>• Weight scale<br>• Activity tracker<br>• Motion sensors<br>• Door sensor<br>• Pressure sensor<br>• Panic Button |
| *Deployment* | • The medical and building sensors adapters are deployed on Raspberry Pis (uc 3.1) and smartphones (uc 3.2).<br>• The GDPR VAS is deployed on local server. |
| *Expected results* | • After first measurement is taken, the organisation should be created in VICINITY, the device should be registered and, friendships and contracts should be made.<br>• Measurement reaches the GDPR VAS. |

| | |
|---|---|
| *Test procedure* | The test procedure consists of the following steps:<br>*For Raspberry Pi*:<br>• A new measurement from a device/sensor is taken for the first time and transferred to Raspberry Pi.<br>• Neighbourhood Manager API is used to automatically create the Organisation of this device, the registration of device to VICINITY (including Agent configuration), the friendship between the device Organisation and the GDPR VAS Organisation (Municipality), the contract between the device and VAS.<br>• Log in to Vicinity Neighbourhood Manager and search the newly created Organisation, device etc.<br>*For smartphone*:<br>• Launch EHealthPass Vicinity extended App and select "Register now".<br>• Fill in name, email etc. in the registration form and press "Register".<br>• Log in to EHealth Pass and wait for registration process to complete. At this step Neighbourhood Manager API is used to automatically create the Organisation for the middle-aged citizen, the registration of activity tracker, weight scale and beacon reader to VICINITY (including Agent configuration), the friendship between the citizen Organisation and the GDPR VAS Organisation (Municipality), the contract between the devices and VAS.<br>• Log in to Vicinity Neighbourhood Manager and search the newly created Organisation, device etc. |

### 7.3.2. Testing Platform

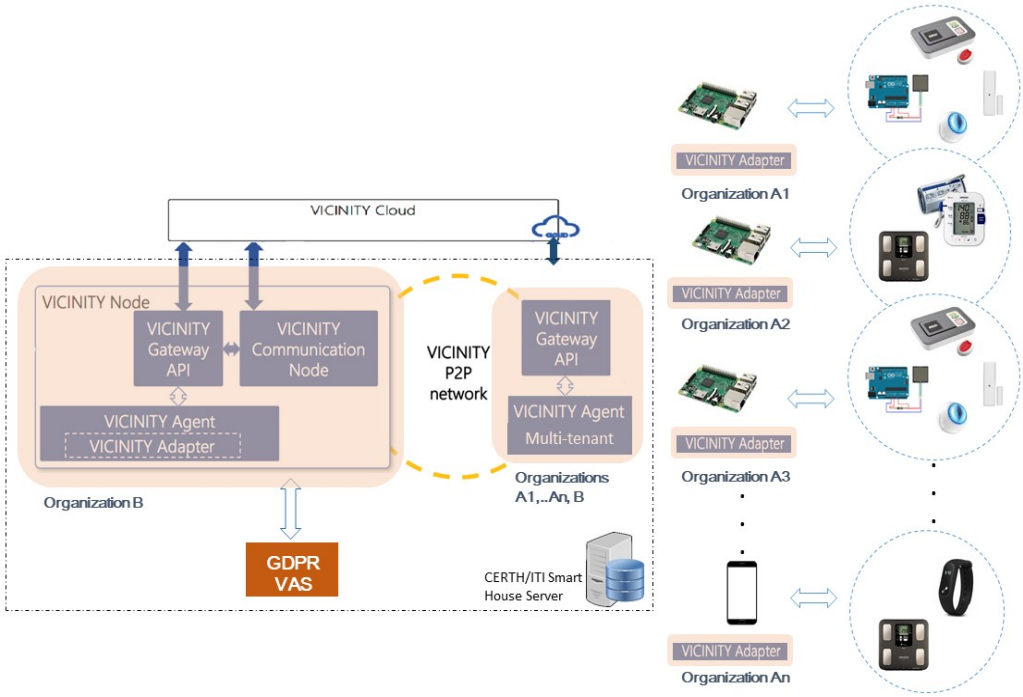



Figure 12 Testing platform and involved components for internal point testing 8.

### 7.3.3. Testing Results

| Internal point testing 8 | Large scale integration of eHealth infrastructures |
|---|---|
| *Real results* | • Organisation is created in VICINITY, device(s) is/are registered, friendships and contracts are made.<br>• Measurement is successfully transferred to GDPR VAS. |
| *Developed User Interfaces* | -<br>*Functionalities:-* |
| *Real results (demo)* | Related snapshots of the GUIs, waveforms, control boards, setups, data flows are included in **Annex XI**. |
| *Deviations* | No deviations |
| *Other technical issues* | Problems in the view of NM interface were identified due to the big number of contracts and contracted devices/services, which was tracked and solved.<br>Enhancements to the NM API services were proposed and implemented (e.g. avoid double creation of contracts). |
| *Status* | Passed after corrections |
| *Notes* | This test is important for checking what happens when increasing the usage limits of VICINITY Platform. (Should not be confused with the testing of the integration of the infrastructure and the communication with the VAS which has been tested in T6.1.) |

# 8. Internal point testing – Integration of Internet of Everything Lab (ATOS - ESP)

## 8.1. Testing objective, Testing Environment and the Role of the Vicinity Prototype

The goal behind the integration of the assets of the Internet of Everything (IoE) lab will be twofold: on the one hand, the devices will be made available through VICINITY, thus increasing the overall tally of "visible" and heterogeneous objects. On the other hand, this testing phase will lead to the implementation of a couple of adapters tailored to "translate" the legacy data sets into VICINITY's *The Thing Description* semantic model. In this sense, the adapters are something that might be leveraged by others, hence opening a huge door to the fostering of new platforms in a straightforward and almost "plug-and-play" way.

Namely, the integration of the IoE Lab has given rise to the following adapters:

- MQTT/Cayenne Low Payload Protocol (LoRaWAN) - https://github.com/vicinityh2020/vicinity-adapter-cayenne
- FIWARE-NGSIv2 - https://github.com/vicinityh2020/vicinity-adapter-ngsiv2

Last, but not least, it is worth highlighting that ATOS did not have any kind of participation of the development of Value-Added Services; hence, the scope of these test is bounded to the integration of Lab's devices into VICINITY.

## 8.2. Internal point testing 9 - Integration of LoRa devices

### 8.2.1. Testing case design

| Internal point testing 9 | Integration of LoRa devices |
|---|---|
| *Test scenario and goal* | Integration of the IoE Lab's assets through a LoRaWAN Network Server |
| *Iterations* | The test was repeated in a daily basis from the moment the whole stack was ready. Moreover, the process was repeated between different VICINITY nodes deployed on different networks and locations |
| *VICINITY components/functions involved* | • Cayenne Adapter v0.5<br>• Agent (Active and Passive Discovery) v0.6.3<br>• Gateway API v0.6.3<br>• Neighbourhood Manager v0.6.3 |
| *Equipment and testing environments* | • LoRaWAN (based on STM32 boards [1]) devices with board temperature sensors<br>• Raspberry Pi3 + LoRaWAN hat behaving as LoRa nodes, with temperature, relative humidity, barometric pressure and GPS sensors<br>• LoRaWAN gateway (Multitech Conduit2)<br>• Raspberry Pi3 + Multi-channel concentrator behaving as LoRaWAN gateway |

[1] https://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l0-series/stm32l0x3/stm32l073rz.html

[2] https://www.multitech.com/brands/multiconnect-conduit

| | |
|---|---|
| | • The Things Network open-source Application Server3<br>LoRaServer open-source LoRaWAN network server4 |
| *Deployment* | • Cayenne adapter was developed using Python 3, running of one of the Raspberry Pi 3 owned by the lab. Anyway, as it is based on Python, any kind of computer is able to run the module<br>• As for the LoRaWAN stack, based on TTN, which provides an MQTT Broker where we are going to create a subscription (**mqtt://eu.thethings.network**). The gateway chosen for this was the Multitech Conduit<br>• A second Raspberry Pi 3 was used to host the VICINITY Agent and Gateway API instances. |
| *Expected results* | • A number of MQTT/Cayenne-connected devices deployed within the Internet of Everything will be accordingly registered onto VICINITY<br>• Together with the data harvested from the sensors, the Cayenne protocol also transmits information about the exchange of information (e.g. Signal to Noise Ratio – SNR, Received Signal to Strength Indicator – RSSI, etc).<br>• Measures from the different devices' sensors (e.g. temperature, relative humidity, battery…) will be available and exposed through the VICINITY ecosystem<br>• Nodes should also generate events that another VAS could subscribe to |
| *Test procedure* | • First of all, apart from the VICINITY steps that are defined below, it is deemed necessary to have the whole LoRaWAN stack up-and-running in order to stick to the dataflows generated by the nodes. That is:<br>  ○ The Gateway must have installed either TTN or LoRaServer stacks. In the scope of this tests, we have opted for the former one (TTN). Moreover, we have chosen the Over-The-Air-Activation (OTAA) mode as the mode nodes are getting connected to the network.<br>  ○ After configuring the LoRaWAN's Network Server, nodes' setup must be tweaked so that they can be bound to the appropriate Application Server (i.e. via App EUI and App Key).<br>  ○ Some information displayed at the Network Server user interface must be used when it comes the set up the Cayenne adapter. Technically speaking, the location and credentials of the Network Server's MQTT (Message Queue Telemetry Transport) Broker are to be included in the adapter's configuration file.<br>• The testing phase must validate the correct implementation of the Cayenne adapter in two different phases: first, assuming that the payload arrives to the adapter encoded, that is, in a raw base-64 format. Second, off-the-shelf packet forwarders, like TTN's, do perform a parsing operation before forwarding to the next stage. On this, the adapter will receive the payload already processed and, thus, the first steps could be skipped<br>• IoE Lab's devices must be correctly integrated within the VICINITY ecosystem. This way, their output data will be accessible through VICINITY nodes (i.e. Agent) and VICINITY neighbourhood manager |

[3] https://www.thethingsnetwork.org/
[4] https://www.loraserver.io/loraserver/overview/

| | |
|---|---|
| | • The tester shall be able to subscribe (via VICINITY legacy operation and through the Network Server MQTT Broker) to the devices' data streams, so that he/she would be able to see, at the same time, the legacy information flow coming from the LoRaWAN stack and its VICINITY equivalent.<br>• The tester can verify the correct operation by assessing that the information seen through the two methods described in the above point is identical (the payload). |

### 8.2.2. Testing Platform

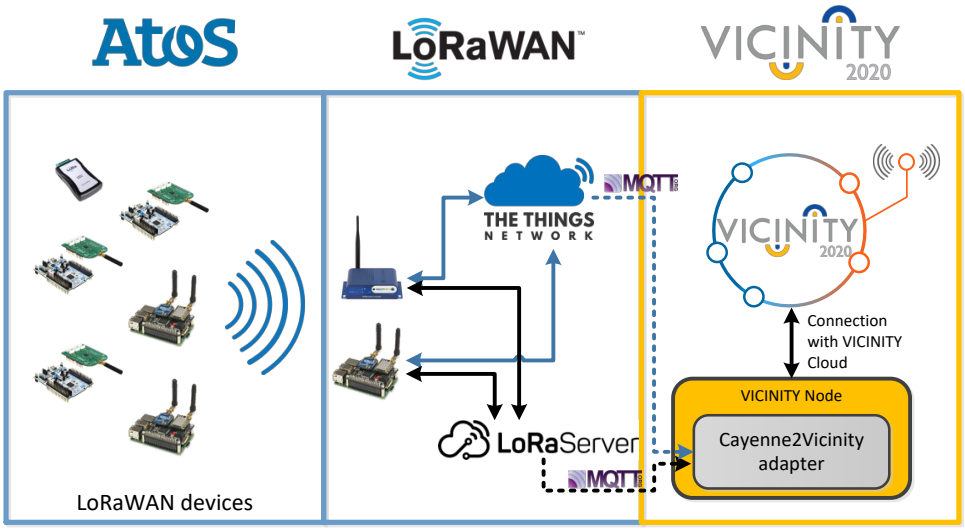



Figure 13 Testing platform and involved components for internal point testing 9.

### 8.2.3. Testing results

| Internal point testing 9 | Integration of LoRa devices |
|---|---|
| *Real results* | • For this integration phase, a single node was subject of the recurrent tests<br>• With all the LoRaWAN/TTN stack online, we ran the MQTT/Cayenne adapter (important to say that this must go before starting the Agent)<br>• Assuming that the VICINITY node's Gateway API instance is running, the Agent was assessed in its two different discovery operations: 1- Passive discovery, where the Agent explicitly requested the adapter the list of available devices/objects; 2- Active discovery, where is the adapter the responsible for spreading the apparition of a new object<br>• Done this, the node is successfully displayed on the Neighbourhood Manager user interface |
| *Developed User Interfaces* | (No user interface) |
| *Real results (demo)* | All tangible results have been shifted to **Annex XII** |
| *Deviations* | 09.09.2018 |

| | |
|---|---|
| | Bug #48 Mismatch between agent and adapter paths<br><br>In a first version, as the sample of thing description was caught from a raw documentation page, the developer did not realize that the read_link endpoints were different to the ones they should have been. As soon as the error was spotted, the solution was immediate |
| *Other technical issues* | None |
| *Status* | Passed after corrections |
| *Notes* | As hinted throughout the text |

## 8.3. Internal point testing 10 – Generic integration of FIWARE-compliant devices

### 8.3.1. Testing case design

| **Internal point testing 10** | **Generic integration of FIWARE-compliant devices** |
|---|---|
| *Test scenario and goal* | The main goal of this test is to integrate the same set of devices that was registered in the above case (Section 8.2). However, in this case, the path followed by the data until it gets the VICINITY Cloud is rather different. Instead of relying on a fully-fledged-open-source LoRaWAN stack to cater data to VICINITY, this test leads to a new element. |
| *Iterations* | Same operation as that of Section 8.2 |
| *VICINITY components/functions involved* | • FIWARE-NGSIv2 Adapter v0.6<br>• Agent (Active and Passive Discovery) v0.6.3<br>• Gateway API v0.6.3<br>• Neighbourhood Manager v0.6.3 |
| *Equipment and testing environments* | • LoRaWAN (based on STM32 boards) devices with board temperature sensors<br>• Raspberry Pi3 + LoRaWAN hat behaving as LoRa nodes, with temperature, relative humidity, barometric pressure and GPS sensors<br>• Raspberry Pi3 + Multi-channel concentrator behaving as LoRaWAN gateway<br>• The Things Network open-source Application Server<br>• Raspberry Pi3 hosting the required FIWARE components (Orion Context Broker + LoRaWAN IoT Agent) |
| *Deployment* | • In this case, the framework used for implementing the FIWARE-NGSIv2 is NodeJS. This is due to the fact that most of the Generic Enablers provided by FIWARE has been done with it. Therefore, the adapter can be run on any platform (in this case, a Raspberry Pi 3)<br>• Regarding the LoRaWAN gateway, for this integration we have used a Raspberry Pi 3 plus a Multi-channel concentrator<br>• A third Raspberry Pi 3 was used to host the VICINITY Agent and Gateway API instances. |
| *Expected results* | • A number of devices deployed within the Internet of Everything will be accordingly registered onto VICINITY; this time via FIWARE-NGSI APIs |

| | |
|---|---|
| | • All the expected results are identical to the ones described in Section 8.2 |
| *Test procedure* | • Regarding the configuration of the LoRaWAN stack (two boxes at the left in Fig.14), we have harnessed that the process had been already carried out in the previous phase (Section 8.2)<br>• In between LoRaWAN and VICINITY networks, some FIWARE components are to be deployed so as to connect all the data flow. Namely, a LoRaWAN IoT-Agent has to hook at the LoRaWAN Network Server MQTT Broker (alike the Cayenne case). Besides, the devices must be manually registered at the Orion Context Broker. These steps done, the adapter only has to point out to the Context Broker and stay tuned to the updates coming from the sensors |

### 8.3.2. Testing Platform

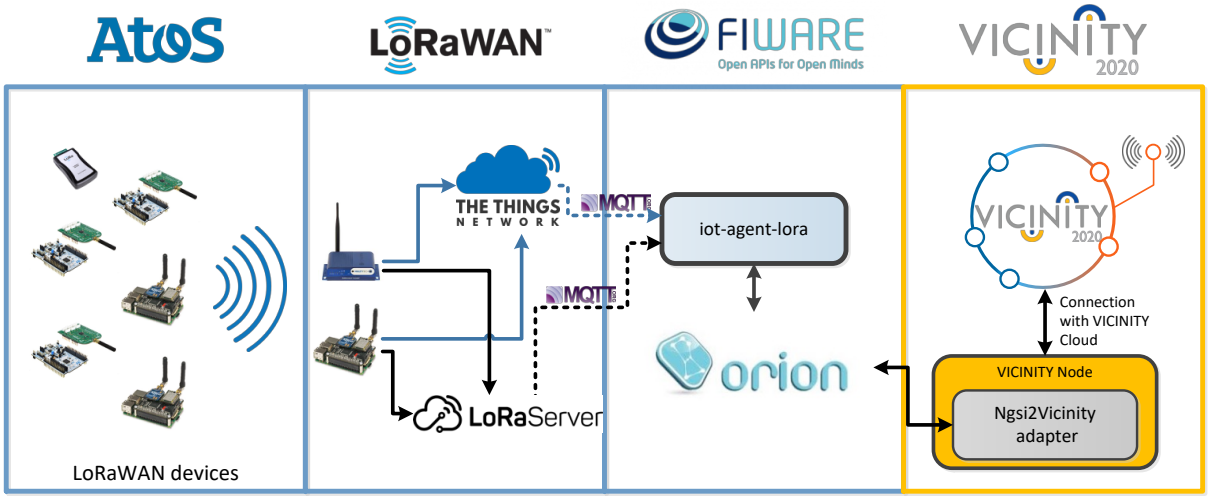



Figure 14 Testing platform and involved components for internal point testing 10.

### 8.3.3. Testing Results

| Internal point testing 10 | Generic integration of FIWARE-compliant devices |
|---|---|
| *Real results* | • Same as Section 8.2 |
| *Developed User Interfaces* | (No user interface) |
| *Real results (demo)* | All tangible results have been shifted to **Annex XIII** |
| *Deviations* | All the lessons learned from Section 8.2 were applied to this stage and the integration was almost seamless. |
| *Other technical issues* | None |
| *Status* | Passed after corrections |
| *Notes* | It is important to highlight that the introduction of the FIWARE-NGSIv2 adapter does not only offer access to these LoRaWAN devices, but also to |

| | |
|---|---|
| | a countless number of off-the-shelf IoT infrastructures based on this well-known framework. |

# 9. Internal point testing – Scalability and Privacy Evaluations of VICINITY Architecture (UNIKL - GER)

The following Chapter gives an Outlook on what VICINITY is currently evaluating, beyond the scope of the Description of Work. VICINITY aims to not only fulfil its goals, formulated at the very beginning, but also to adapt to latest trends and technologies, that arise in this fast-paced world of the Internet of Things. While keeping the intended structure of this deliverable, please note, that the following contains current, ongoing work, still in early stages of development. The previous chapters have demonstrated extensive Lab-testing of the VICINITY components, yet the following work can be seen as an outlook onto what will be integrated and what methodologies will be used for testing in VICINITYs upcoming, final year! At the time of writing, the following is not yet integrated into the VICINITY network. However, future integration is already being discussed with some of the pilot sites.

## 9.1. Testing objective, Testing Environment and the Role of the Vicinity Prototype

Two major challenges in the Internet of Things in general and hence for VICINITY in particular, is the scalability, so the ability to handle the rapidly growing number of connected devices on one hand, and ensuring users privacy on the other hand.

The first internal point testing (Internal point testing 11 - Integration of Omnet++ Network Simulator into VICINITY) will evaluate VICINITYs scalability. To this end, one of the Pilot Site Use Cases is simulated and evaluated before the actual deployment on site.

The second test case (Internal point testing 12 – Evaluation and Research on Homomorphic Encryption to be used for data aggregation for VAS) attempts to further improve the existing privacy, which is already built into VICINITYs design. Yet still the VICINITY team is constantly taking users privacy very seriously and is hence trying to further improve wherever possible.

## 9.2. Internal point testing 11 - Integration of Omnet++ Network Simulator into VICINITY

### 9.2.1. Testing case design

| **Internal point testing 11** | **Integration of Omnet++ Network Simulator into VICINITY** |
|---|---|
| *Test scenario and goal* | In order to test if the VICINITY approach can handle the ever-growing number of devices and thus test its scalability before a potential limit is reached, the team at UNIKL is working on a framework to simulate IoT devices and connect them to VICINITY. To this end, the Omnet++ network simulator is utilized and extended with new capabilities, specifically designed for the simulation of IoT scenarios and stress-testing IoT infrastructures like VICINITY.<br><br>As a first test, the Smart Parking Use Case, which will be deployed at the Tromsø Pilot Site is simulated. |
| *Iterations* | Early Prototype |

| | |
|---|---|
| *VICINITY components/functions involved* | • Not integrated yet |
| *Equipment and testing environments* | • Lab testing<br>• Omnet++ Network Simulator<br>• Tromso Smart Parking Use case is simulated<br>  ○ Simulated Cars enter/leave the facility |
| *Deployment* | • Omnet++ Simulator deployed in Lab Environment<br>• Hardware-in-the-Loop integration possible in future testing<br>• Interaction with real VICINITY virtual Neighbourhood planned for future version |
| *Expected results* | • Simulated Cars are able to book parking spaces<br>• Cars can enter booked parking spaces<br>• Cars leave parking spaces and get billed |
| *Test procedure* | The Smart Parking Use Case is implemented using the Omnet++ Network Simulator and modelled as follows (see Figure 1):<br>• Simulated Cars are generated and enter the Parking Facility<br>• The Simulated Cars utilize the Smart Parking App to book a parking spot<br>• After some random time, the cars leave the parking facility again |

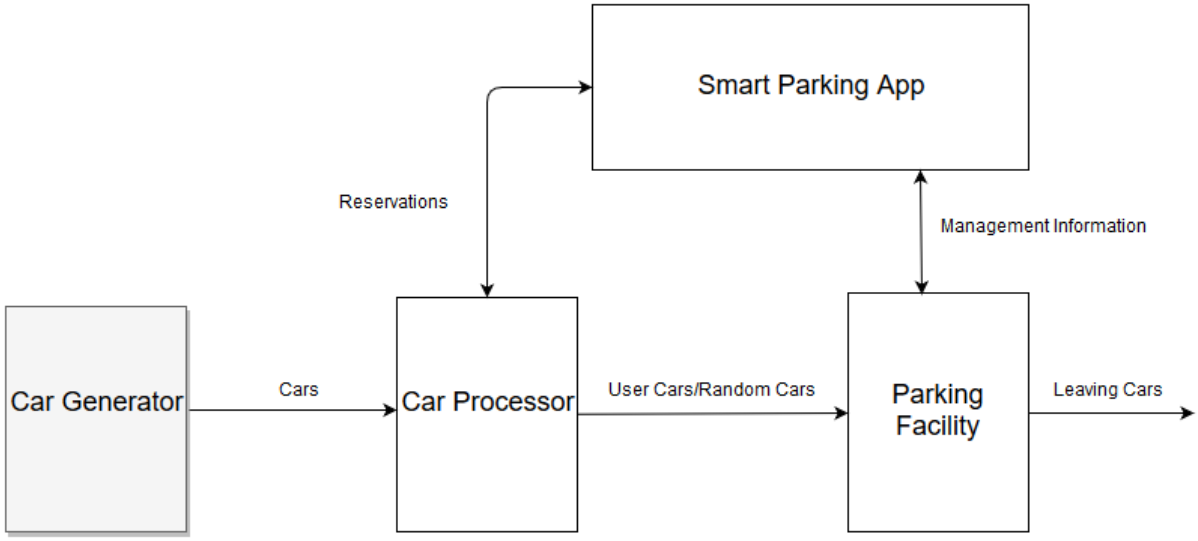



Figure 15 High-level model of Smart Parking Use Case.

### 9.2.2. Testing Platform

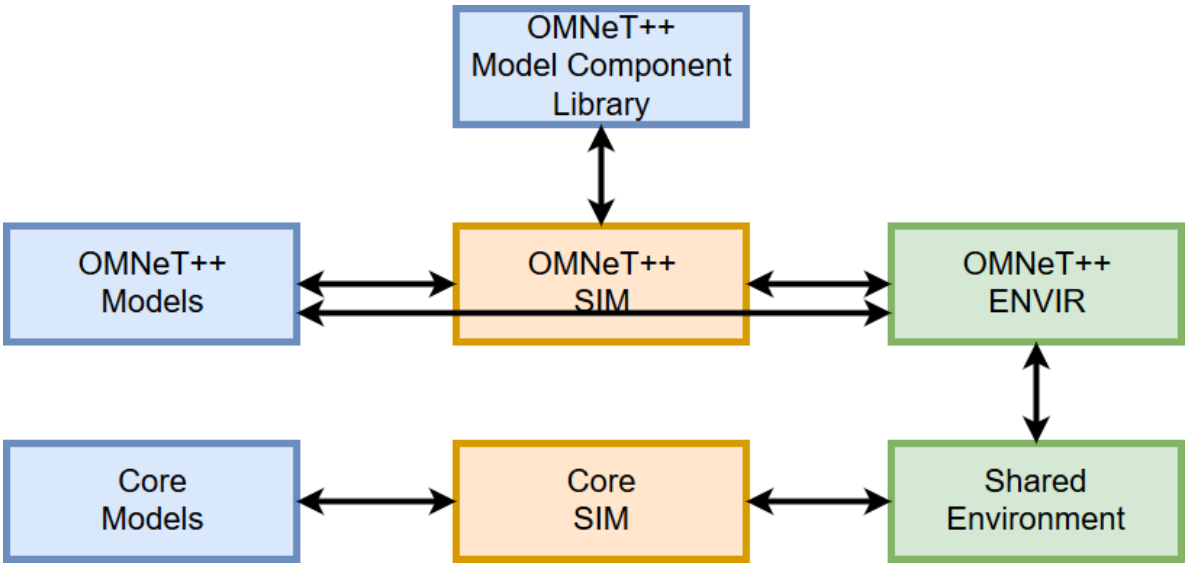



Figure 16 Testing Framework Architecture- internal point testing 11.

### 9.2.3. Testing Results

| Internal point testing 11 | Integration of Omnet++ Network Simulator into VICINITY |
|---|---|
| *Real results* | In order to examine the scaling capabilities of the proposed simulator, the lowest abstraction level has been divided into three parts (one for each parking deck at Level 2). In the following, they will be denoted as L2a, L2b, and L2c. Level 0 and Level 1 will be denoted as L0 and L1, respectively.<br>The ability to run simulations in almost real-time is of particular importance and for this reason, the experiments have also been centered around the wall clock time (WCT) as an indicator.<br>The simulation time for each simulation run was set to 120 seconds and the wall clock time for the single runs was recorded. The wall clock times reported in Table 1 represent the average of several independent simulation run times.<br>As expected, the simulation runs with only the first level L0 and the first two levels L0 and L1 active have achieved similar average wall clock times. Since L0 was used to produce information for the levels below, the runs at this level finished relatively fast. L1 served only as a space division for the L2, and therefore the overhead added by this level is negligible (only 0.256 seconds). This is also shown in Table 1.<br>As expected, the first real spike in the average WCT has occurred with the activation of L2a. In this case, the average WCT increases from 4.516s (with the activated L0 and L1) to 6 seconds. However, with the addition of the next two levels L2b and L2c respectively, the overhead did not increase dramatically and even for the last case, the WCT dropped back to 5.903 seconds. This was a direct consequence of the architecture of the underlying model; the three parking decks that are represented by the levels L2a - L2c, are traversed by cars in sequence. Therefore, we suppose that the simulation time of 120 seconds is not sufficient to create an adequate number of nodes at the later levels L2b and L2c respectively. For this reason, the simulation with all levels activated was repeated with |

| | |
|---|---|
| | an additional number of cars as mobile nodes created from the beginning of the simulation instead of dynamically relying on the information provided by L0. The WCT for this scenario can be seen in Table 2; it is again expressed as the average of times required for single simulation runs. |
| *Real results (demo)* | We developed an approach towards modeling Internet of Things infrastructures together with the implementation of a prototype of a multi-level simulator. The approach proposes inter-connection between models at different abstraction levels within the discrete event simulation framework and has been demonstrated on a smart parking use case of the VICINITY pilot site in Tromsø. The specific solution for this use case has used 3 levels of simulation. The first level has been used to generate abstract information on the general movement of simulated entities and communication between them. The second level has served as a space division for the lower level. It produced more detailed information about the movement that has been used as a basis to dynamically activate the different parts of the lowest level - Level 2. Level 2 has then used the powerful network simulator OMNeT++ with the INET framework to simulate the details of a smart parking service, the movement of users, the communication between them, and the environment. The experiments executed on the use case show that with all three levels active the execution time increases almost two times. With respect to interoperability, the simulator has proven to fulfill the requirements for an IoT simulator. This is achieved through the following capabilities:<br>(1) Dynamic switching between models at different levels of abstraction<br>(2) Spreading multiple simulation engines across the model tree shown in Figure 17<br>(3 )Modeling and simulation of mobile system entities and their communication through the OMNeT++ integration<br>Annex XIV |
| *Deviations* | - |
| *Other technical issues* | • Early development phase, the models still need improvement |
| *Status* | Not yet ready for production |
| *Notes* | Research currently being done |

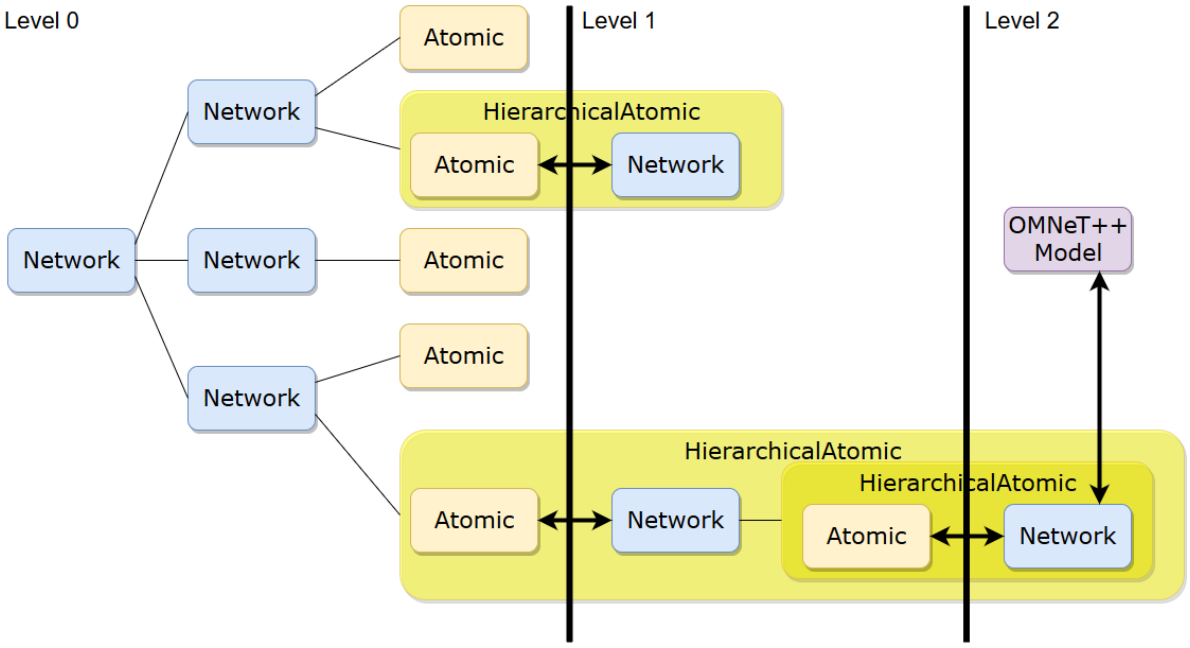



Figure 17 The model tree and organisation of hierarchical levels.

Table 1 Wall clock time with different levels activated and dynamically relying on the information provided by L0

| Level | WCT |
|---|---|
| L0 | 4.260315s |
| L0 + L1 | 4.516574s |
| L0 + L1 + L2a | 6.003036s |
| L0 + L1 + L2a + L2b | 6.627343s |
| L0 + L1 + L2a + L2c | 5.9030517s |

Table 2 Wall clock time with all levels activated

| Level | WCT |
|---|---|
| L0 + L1 + L2a + L2b + L2c | 9.442795s |

## 9.3. Internal point testing 12 – Evaluation and Research on Homomorphic Encryption to be used for data aggregation for VAS

### 9.3.1. Testing case design

| Internal point testing 12 | Evaluation and Research on Homomorphic Encryption to be used for data aggregation for VAS |
|---|---|

| | |
|---|---|
| *Test scenario and goal* | Personal data needs to be handled with special care. Even before GDPR came into action, VICINITY has given much thought on ensuring privacy. One step further on what VICINITY has already accomplished, would be to completely anonymize private data. This may find application in some of VICINITY planned pilot setups.<br>Homomorphic encryption can help with data anonymization, as it allows mathematical operations to be performed on encrypted ciphertexts, rather than on plain text. Data can be collected and aggregated, while being fully encrypted. After enough input data has been collected, there is no telling, which part of this aggregation belonged to which user. Data is anonymized and this aggregation can be decrypted and given to any VAS to work with.<br>VICINITY is currently researching if and how this technique can be applied. |
| *Iterations* | Early prototype |
| *VICINITY components/functions involved* | • VICINITY Agent v0.6.3.1 |
| *Equipment and testing environments* | • Lab Setup<br>• Fake Data is generated |
| Deployment | • Lab testing. Not yet deployed |
| *Expected results* | • Only aggregated, anonymized data is visible to the VAS |
| *Test procedure* | • Data is produced and encrypted with a homomorphic encryption scheme at the source<br>• Encrypted data is transmitted as any other payload through the VICINITY P2P network<br>• Encrypted data is aggregated<br>• Encrypted, aggregated data is decrypted and handed to Consumer (e.g. VAS) |

### 9.3.2. Testing Platform

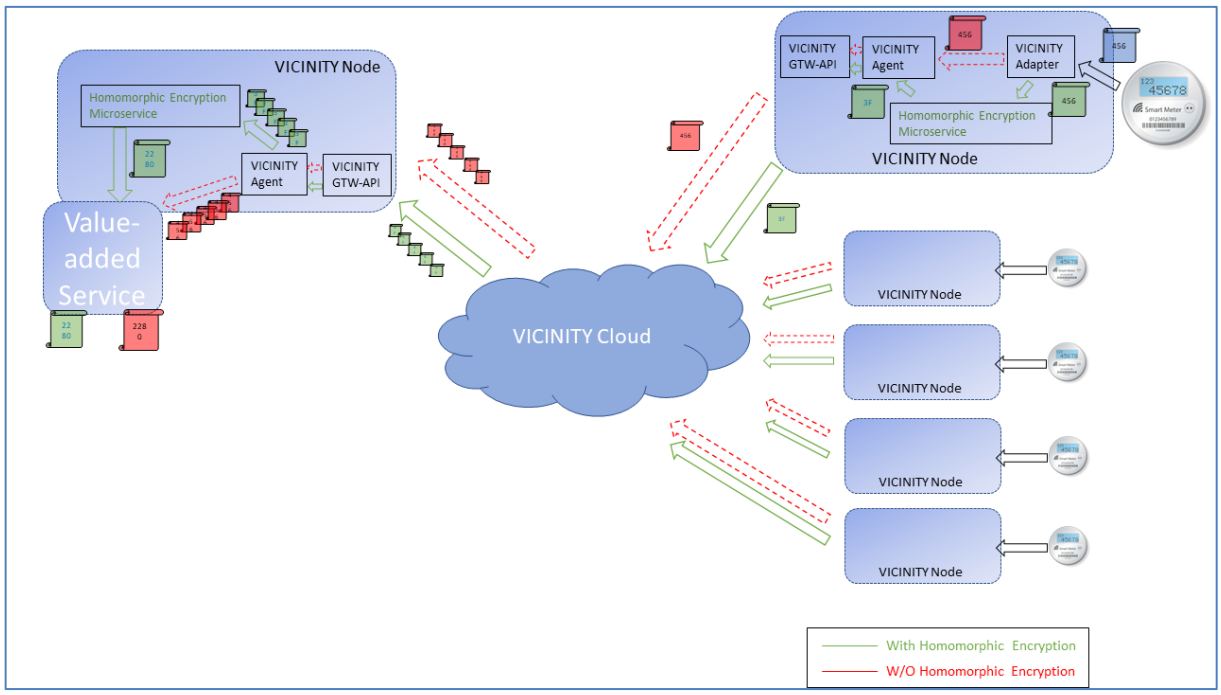



Figure 18 Testing platform and involved components for internal point testing 12.

### 9.3.3. Testing results

| **Internal point testing 12** | **Evaluation and Research on Homomorphic Encryption to be used for data aggregation for VAS** |
|---|---|
| *Real results* | • Data can be encrypted, transmitted and aggregated<br>• Encryption is computationally expensive<br>• Ciphertexts and Encryption Keys become rather larger |
| *Developed* | Component will be integrated between Agent and Adapter. No GUI |
| *User Interfaces* | *Functionalities:*<br>- |
| *Real results (demo)* | - Annex XV |
| *Deviations* | - |
| *Other technical issues* | • Key management needs improvement<br>• Key and Ciphertext size need improvement |
| *Status* | Not yet ready for production |
| *Notes* | Research currently being done |

# 10. Conclusions

Deliverable D6.2 has been one of the steps to reach milestone MS7, which aims to make a first integrated and full-featured VICINITY prototype available.

This deliverable provides an overview of the conducted lab-testing for VICINITY prototype under the scope of each test-bed infrastructure, further summarizing the testing environments/platforms, testing steps, actual testing results, user interfaces, solved and pending issues.

D6.2 is a continuation of the D6.1, the current document provides detailed information about the Lab tests for VICINITY server and client infrastructures implemented in WP3 and WP4, and value-added services defined in WP5 based on cross-domain testing scenarios.

In summarizing the activities of T6.2, intensive and iterative Lab tests have been conducted on the HIL/experimental platform to ensure that local infrastructure, to be deployed at pilot sites, operates with the VICINITY platform as expected. This includes:

- Correct processing of control data via the VICINITY adapters and VICINITY core components
- Peer-to-peer communication of user data in real time
- Correct operation of value-added services including cross-domain examples
- Assured data privacy and encryption
- Successful integration of a big number of different IoT infrastructures

Thus the expected technical functionality of the VICINITY prototype has been validated.

The lessons learned from the Lab trial will be forwarded to WP7 for helping a correct deployment of VICINITY platform at the pilot sites from a technical perspective and to WP8 for helping a technical evaluation approach design.

Finally, relevant screenshots of test platforms, data flow, GUI, and homomorphic encryption are included in Annexes as the demonstrator.

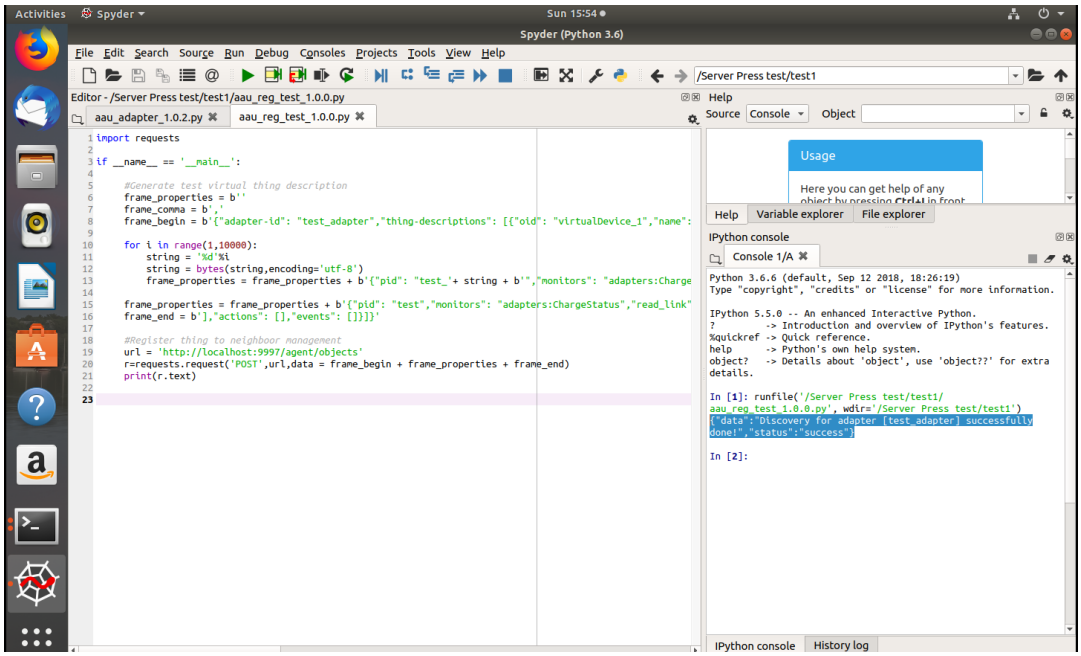
## 11. References

[1] http://www.vicinity-h2020.eu

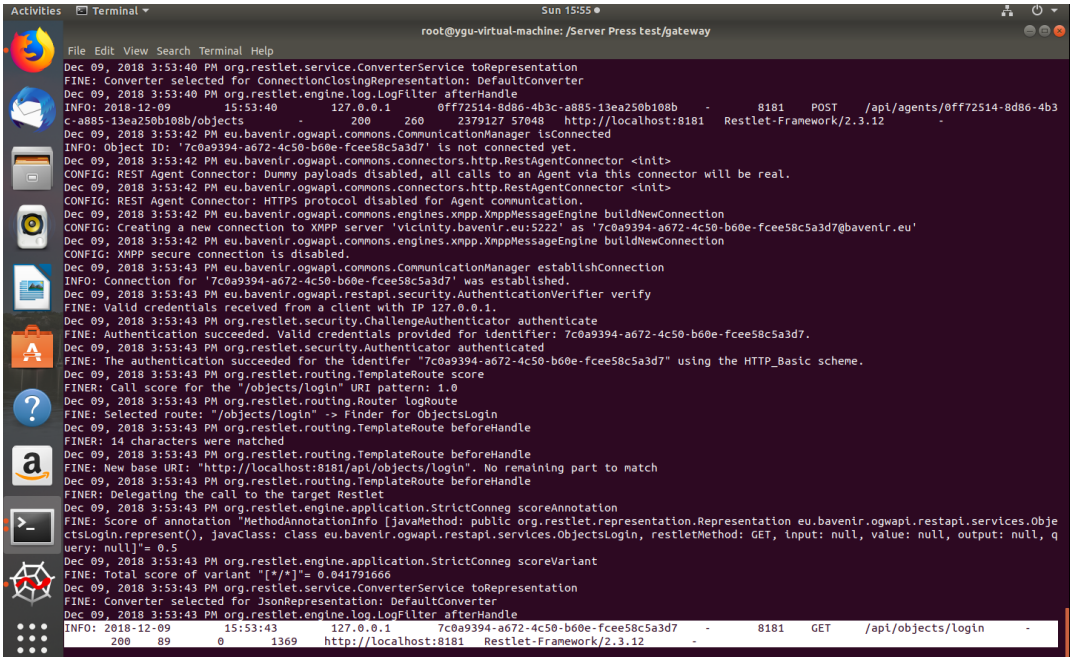
[2] https://vicinityh2020.github.io

# Annex I – Edge Case Testing 1 - Stress registration properties (AAU-DK)

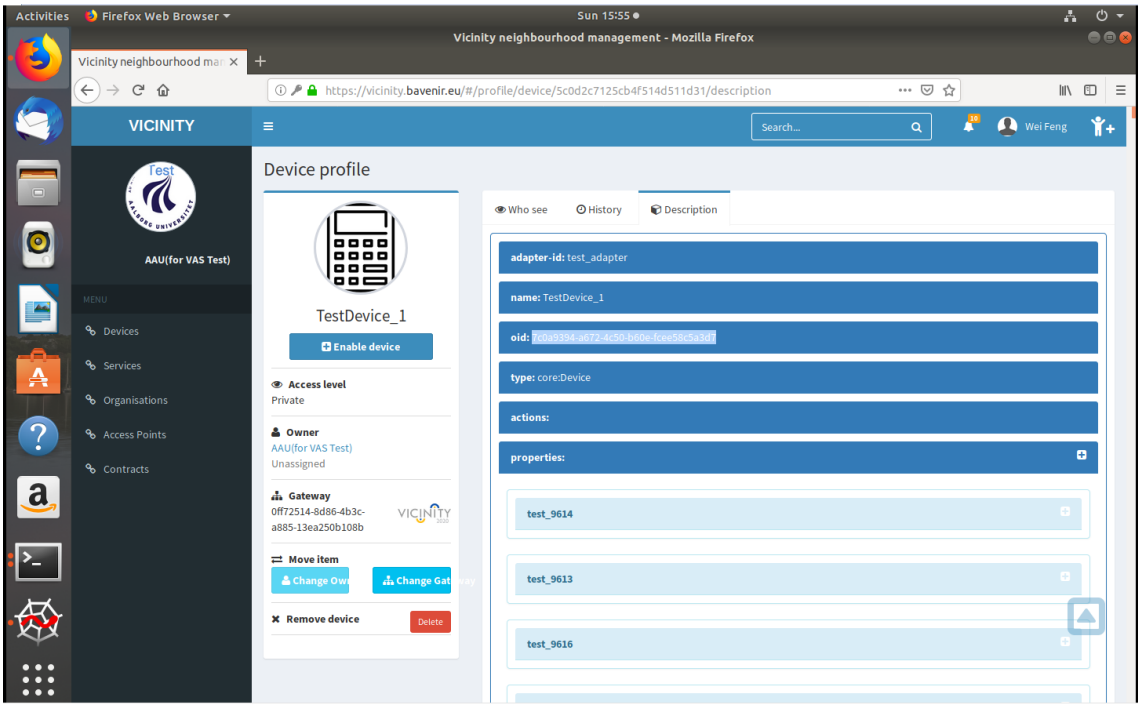
1. Successful registration response




2. Successful Gateway API response for the device registration

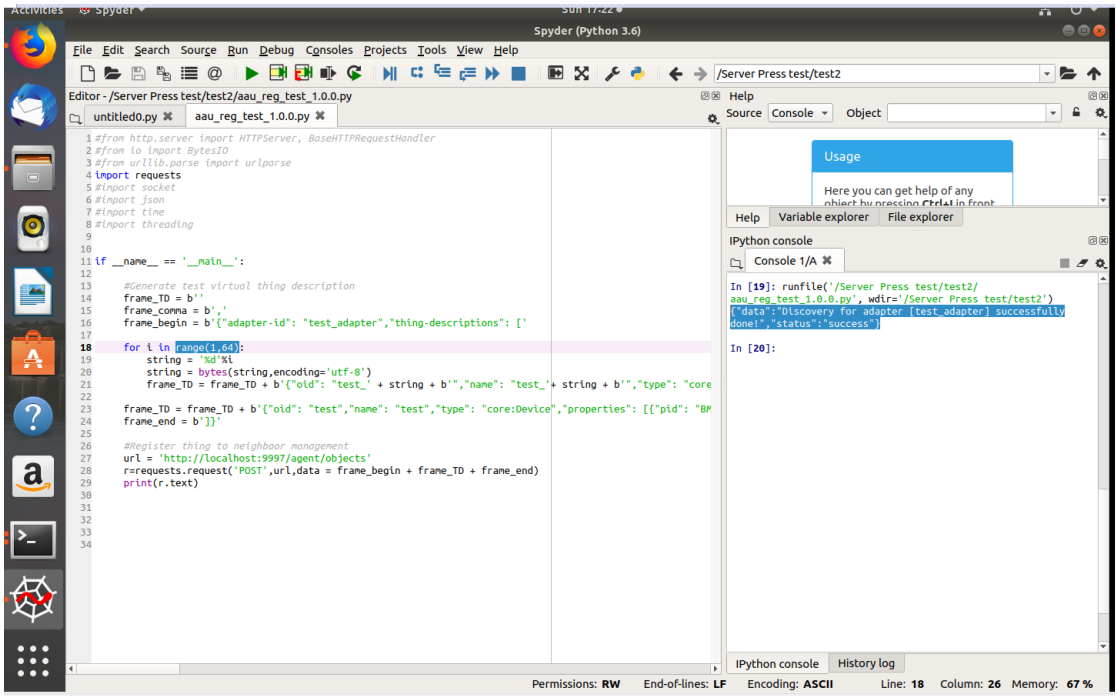


3. Device successfully registered in Neighbourhood manager

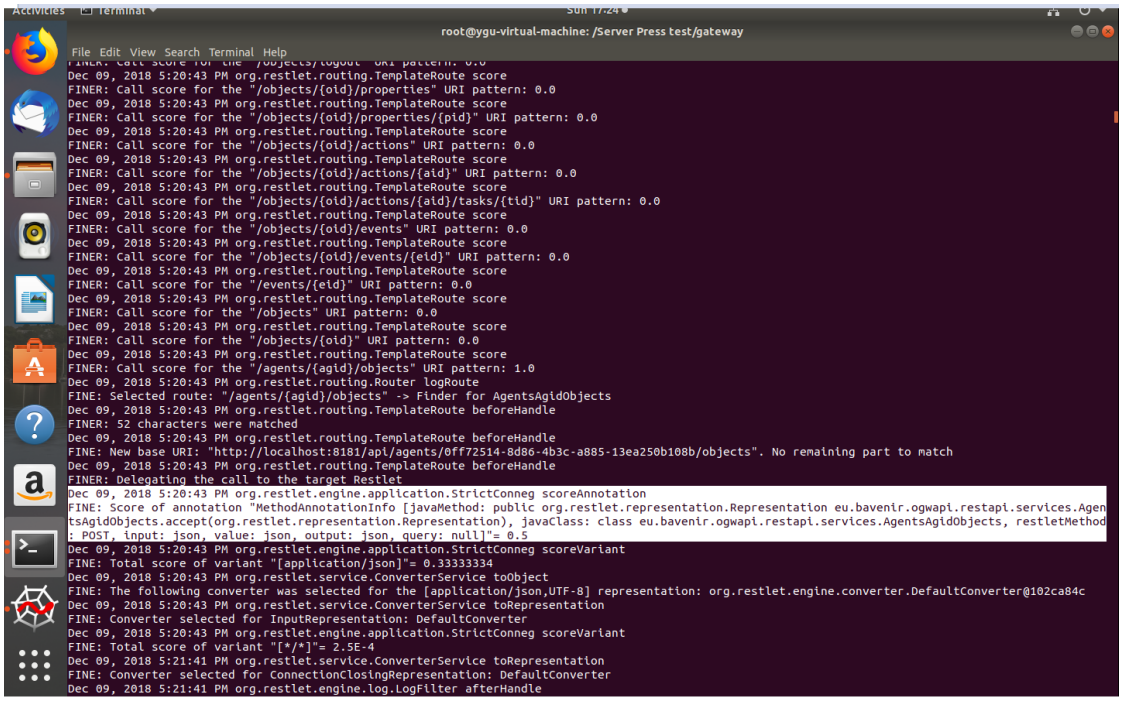


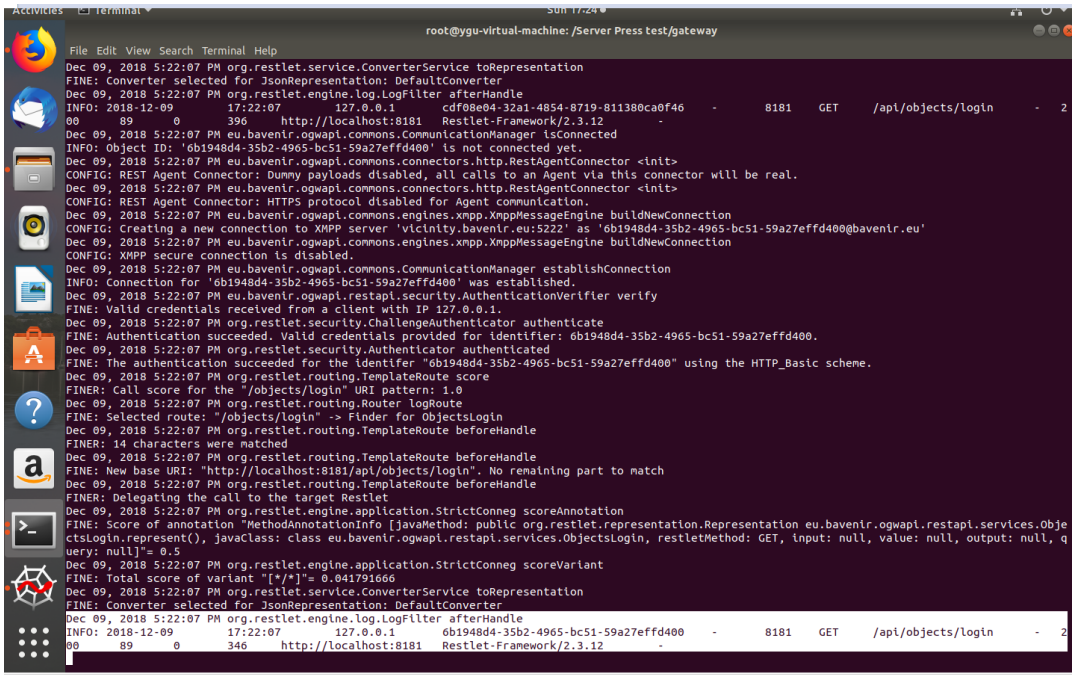
# Annex II – Edge Case Testing 2 - Limit of parallel registrations (AAU-DK)

1. Successful registration response with 64 devices

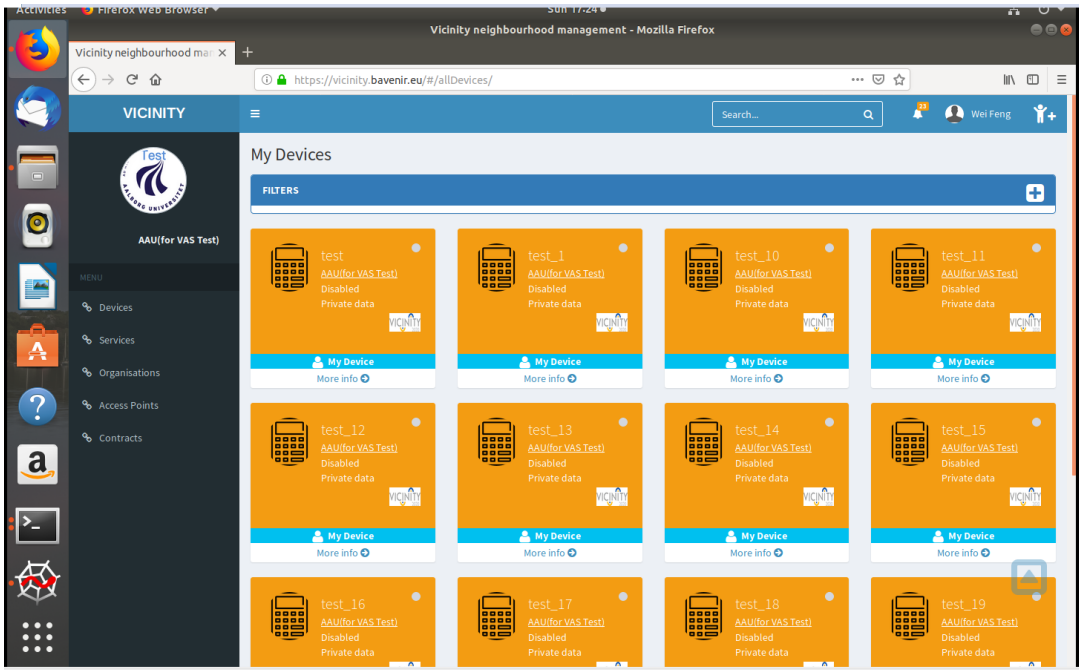



2. Gateway API successful response for the 64 devices registration

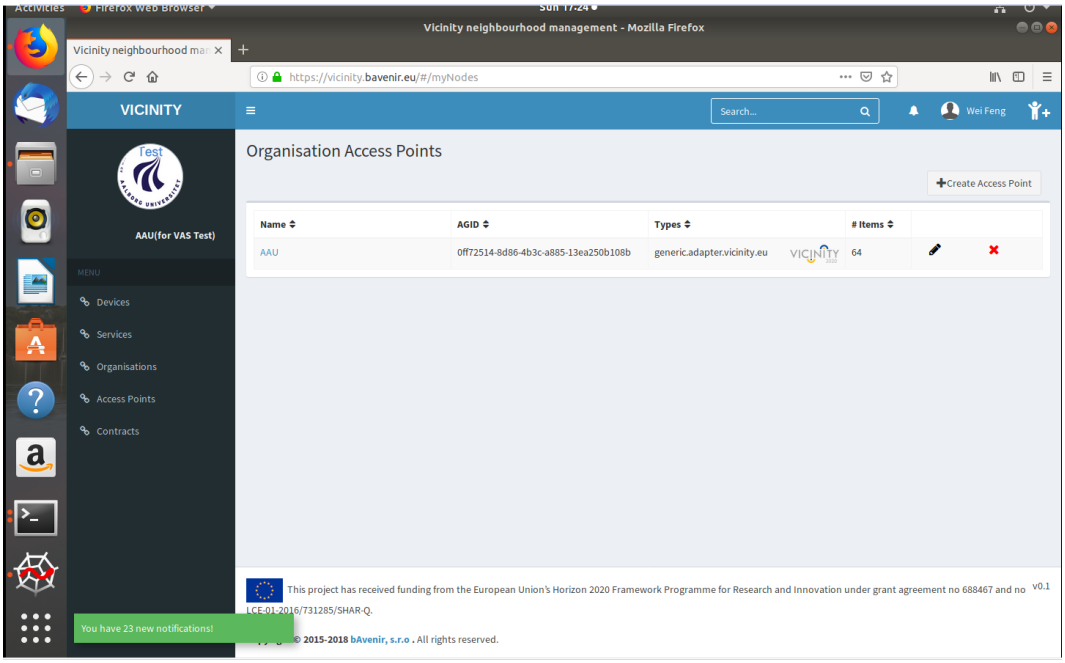




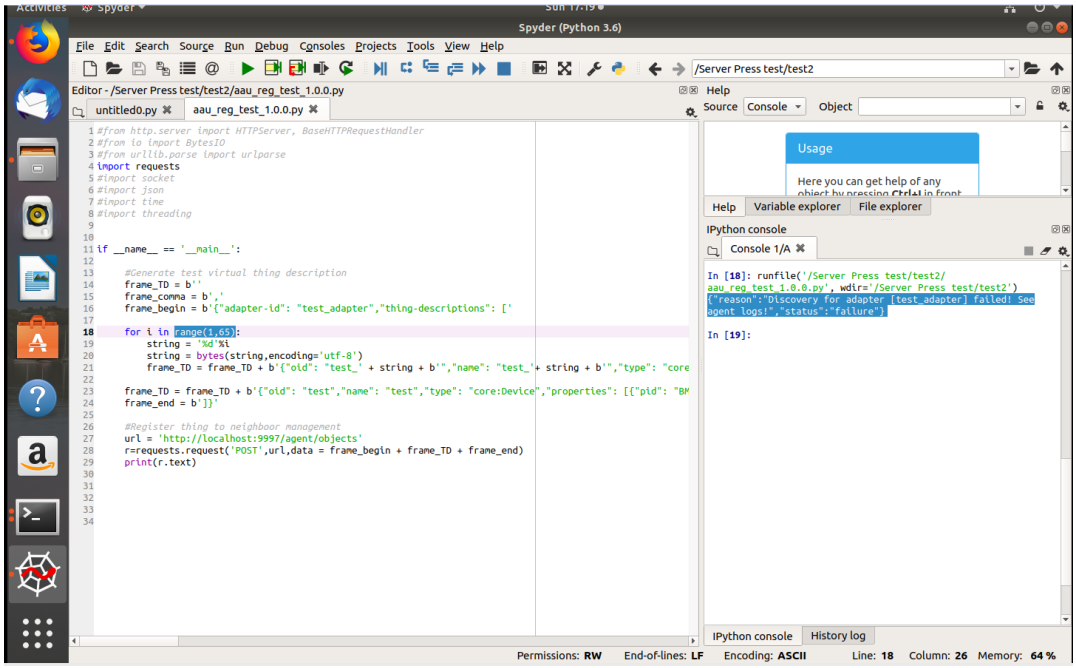

3. 64 registered devices in Neighbourhood Manager



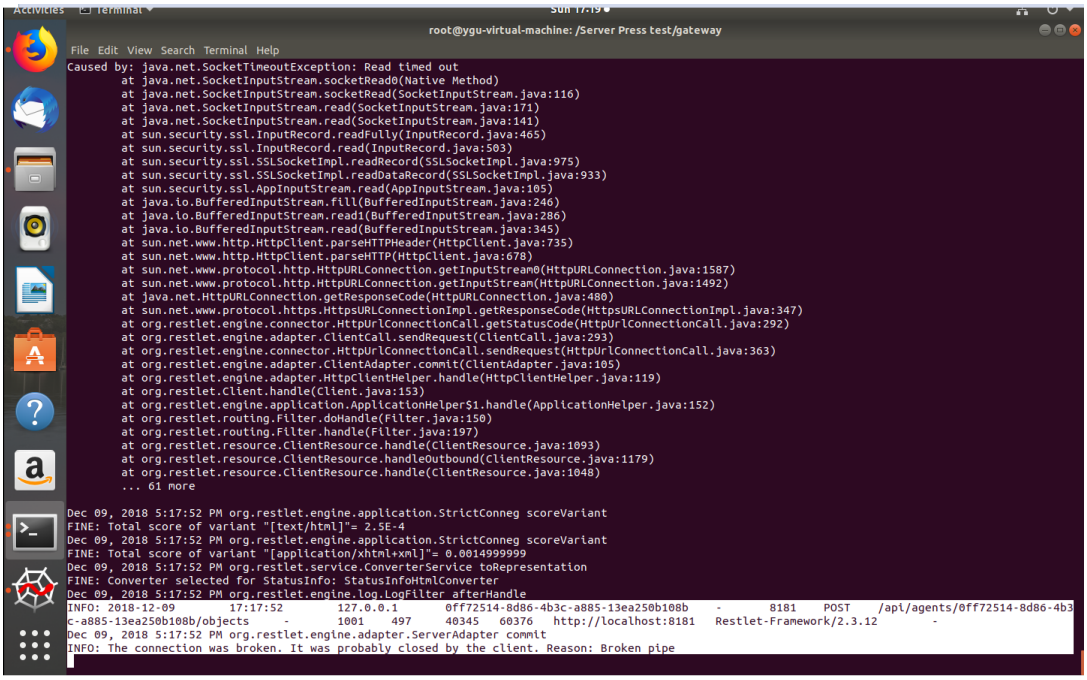
4. “Access Points” with 64 devices in Neighbourhood Manager



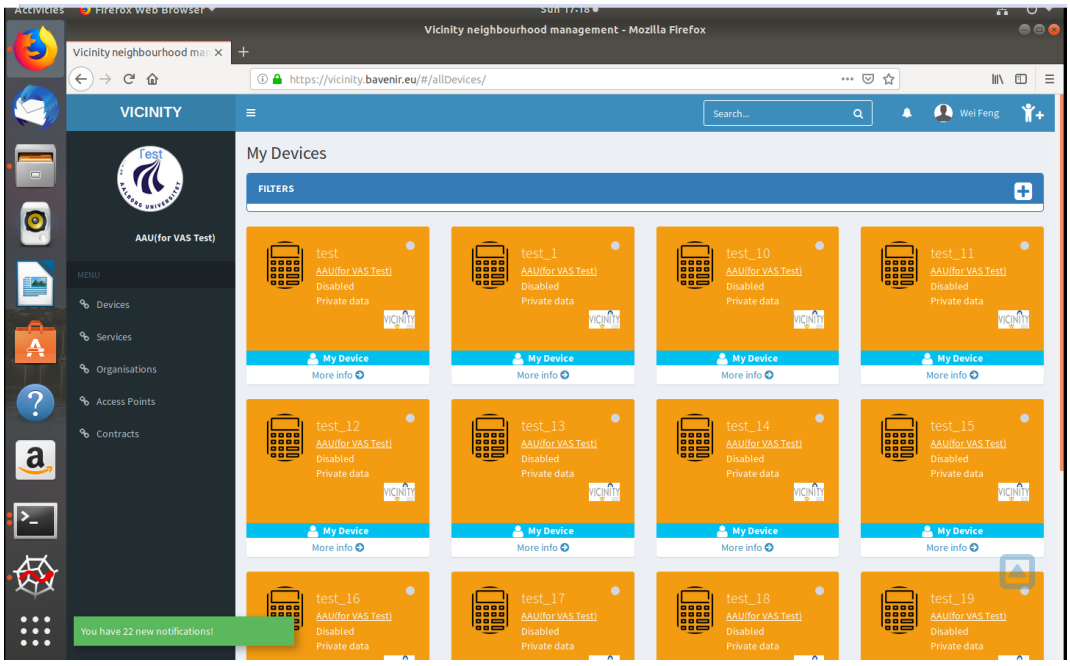

5. Failed registration response with 65 devices



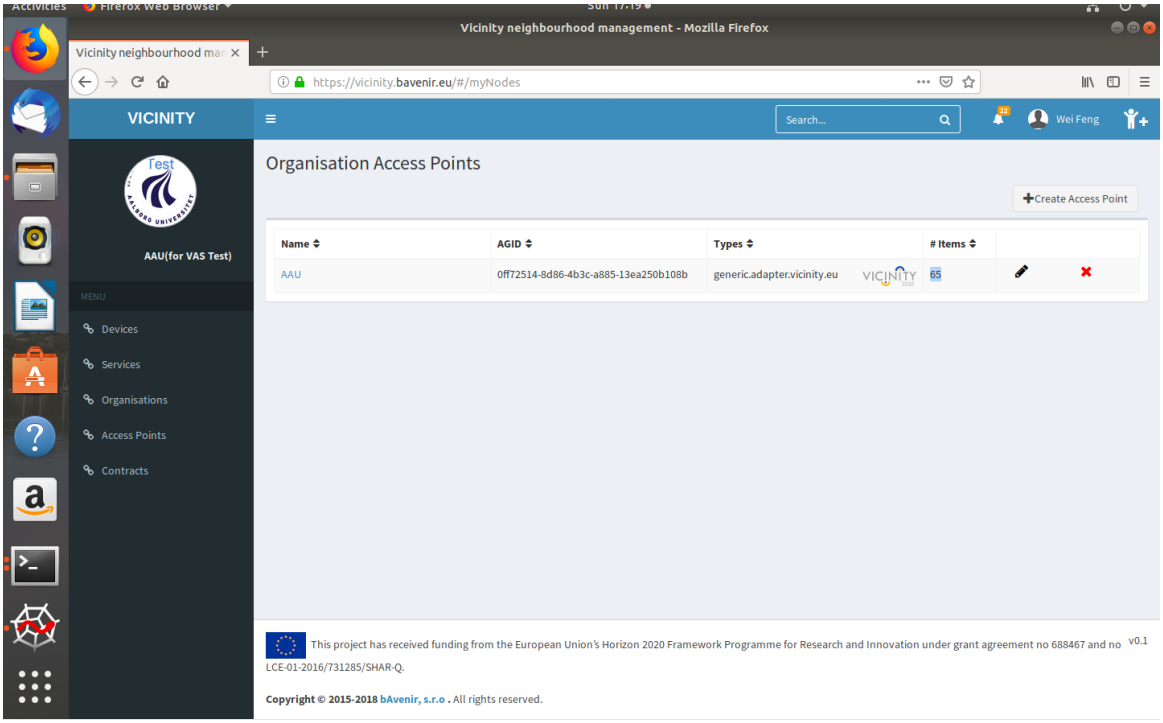
6. Failed gateway response for 65-device registrations




7. 65 registered devices in Neighbourhood Manager

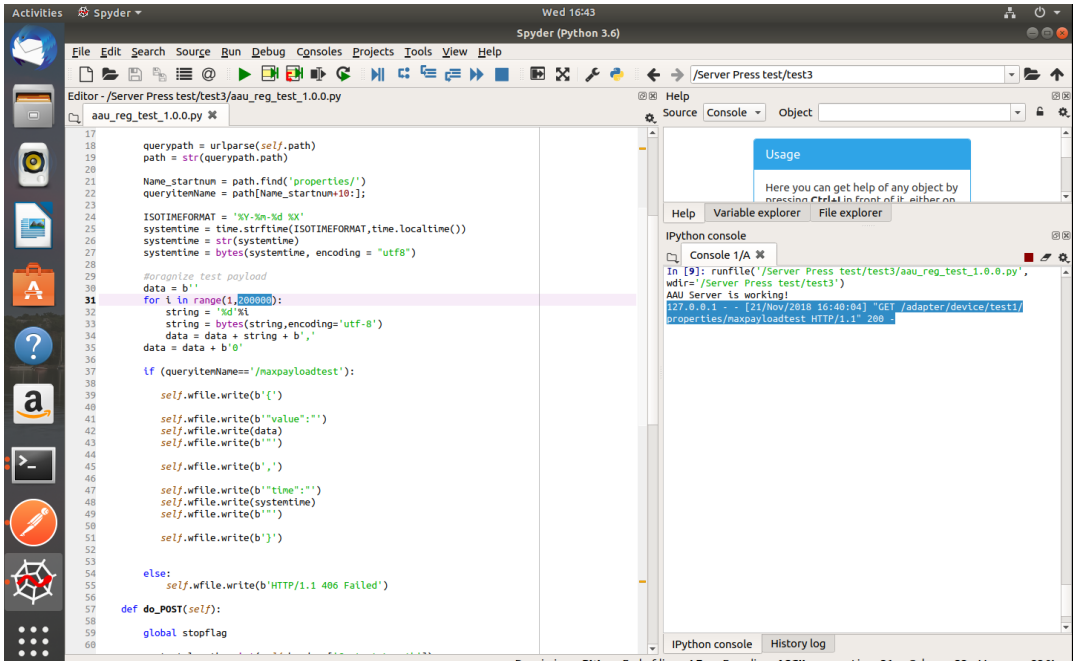


8. “Access Points” with 65 registered devices in Neighbourhood Manager

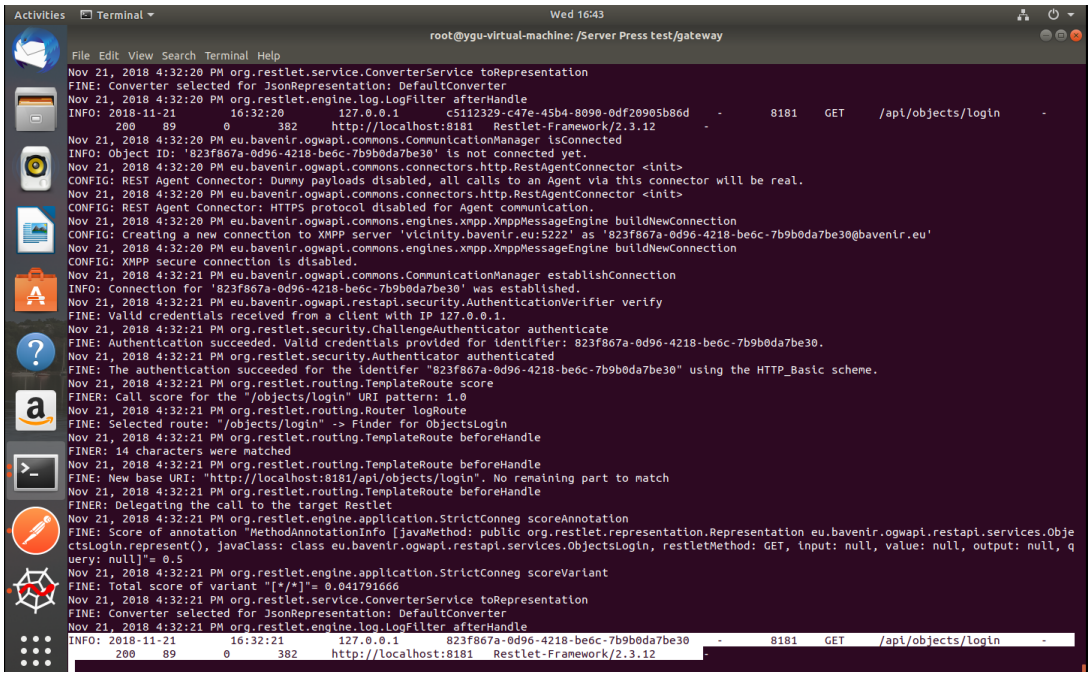


# Annex III – Edge Case Testing 3 - Large size of payload for GET request (AAU-DK)

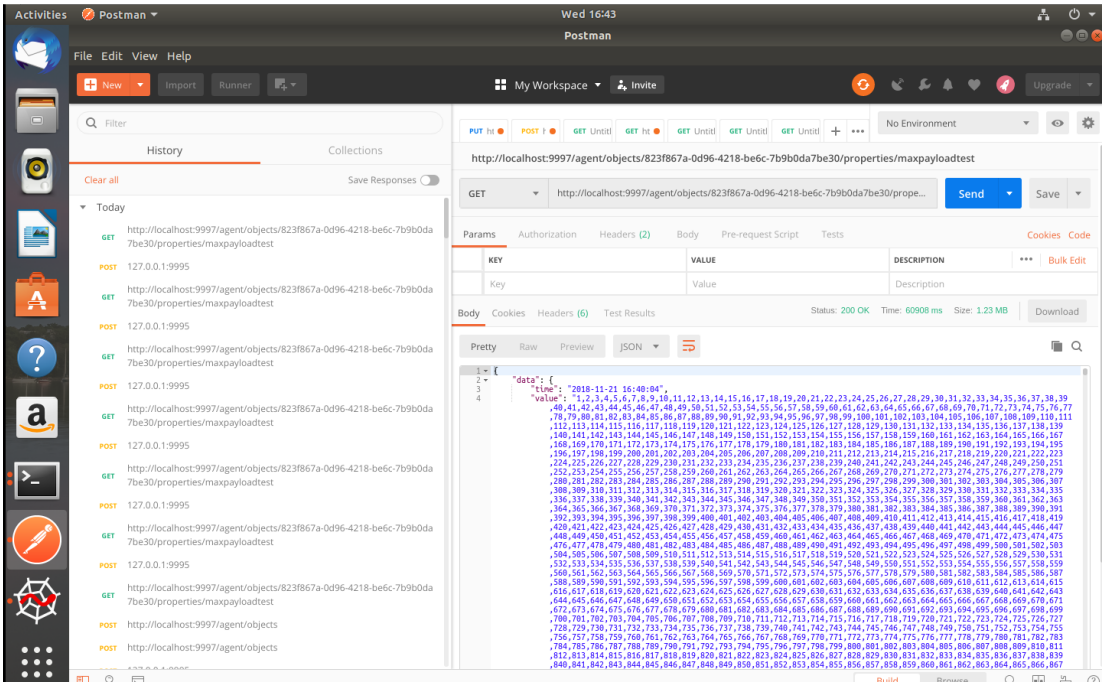
1. GET request is successfully received by the device adapter.




2. GET request is successfully received by the device Gateway.



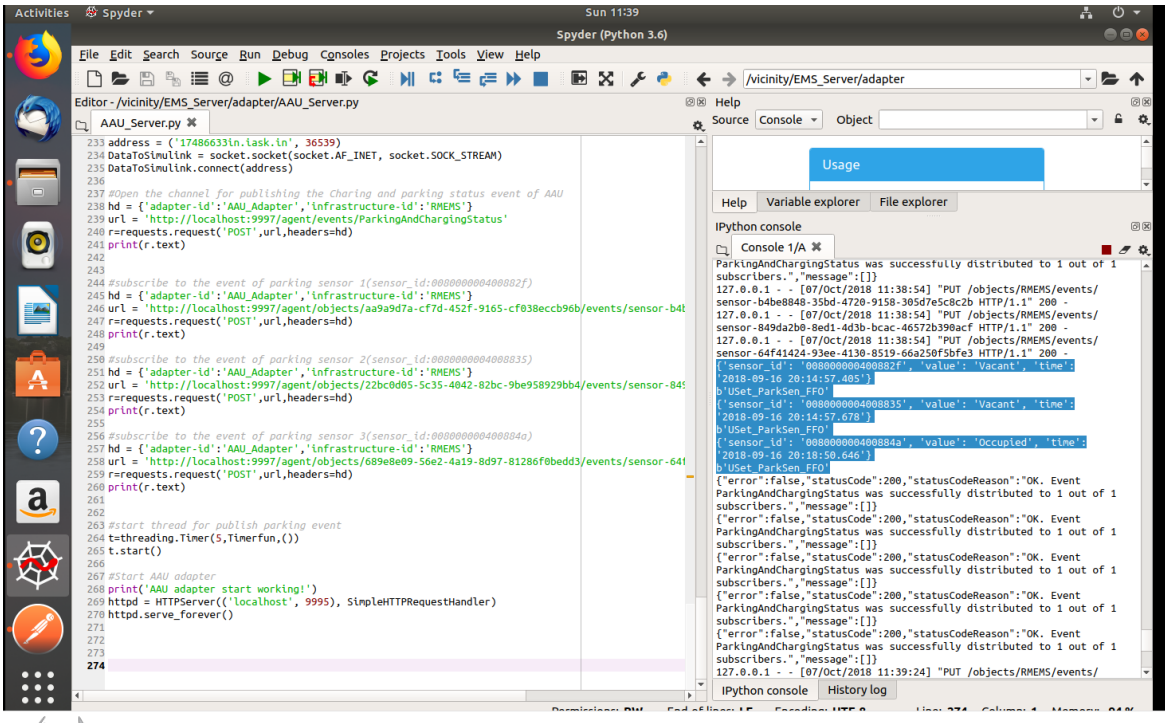
3. Postman successfully receives the requested response with 200000 data.



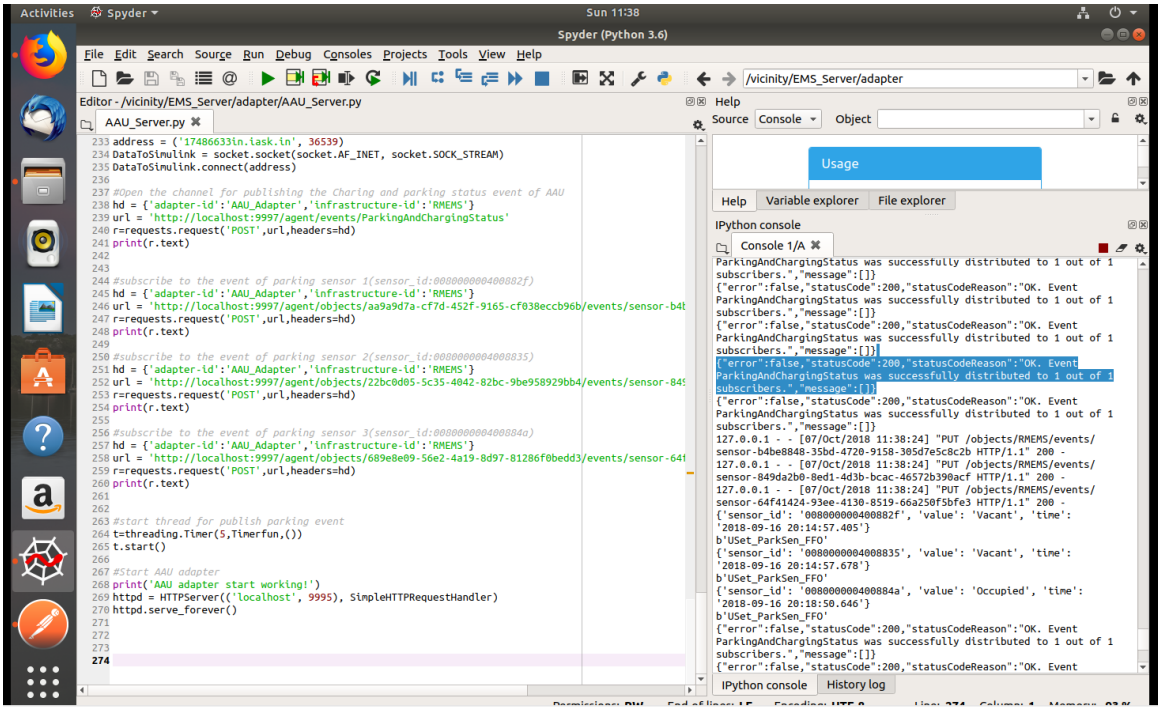
# Annex IV – Internal point testing 1 - Optimal usage of parking slots by considering energy costs (AAU - DK)

1. The real-time experimental platform in AAU IoT-microgrid Lab.

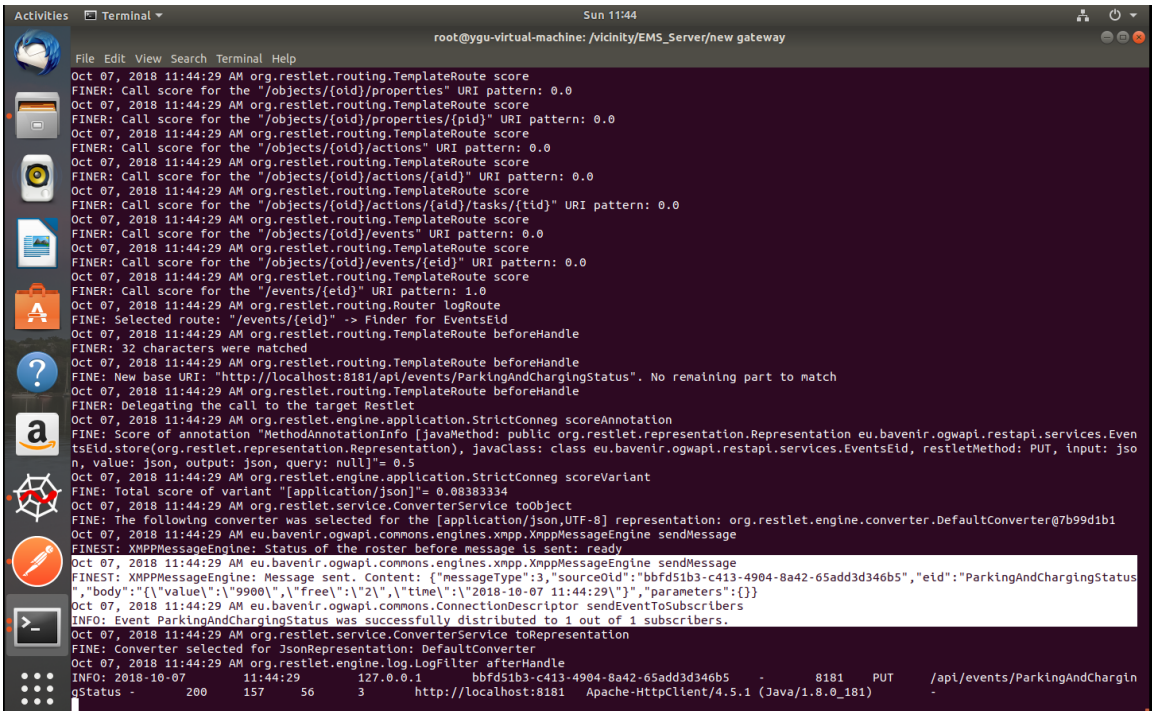

2. After subscribing the event of parking sensor node, the VAS node is able to receive the parking sensor node events which contains the number of free parking slot and time-stamp.



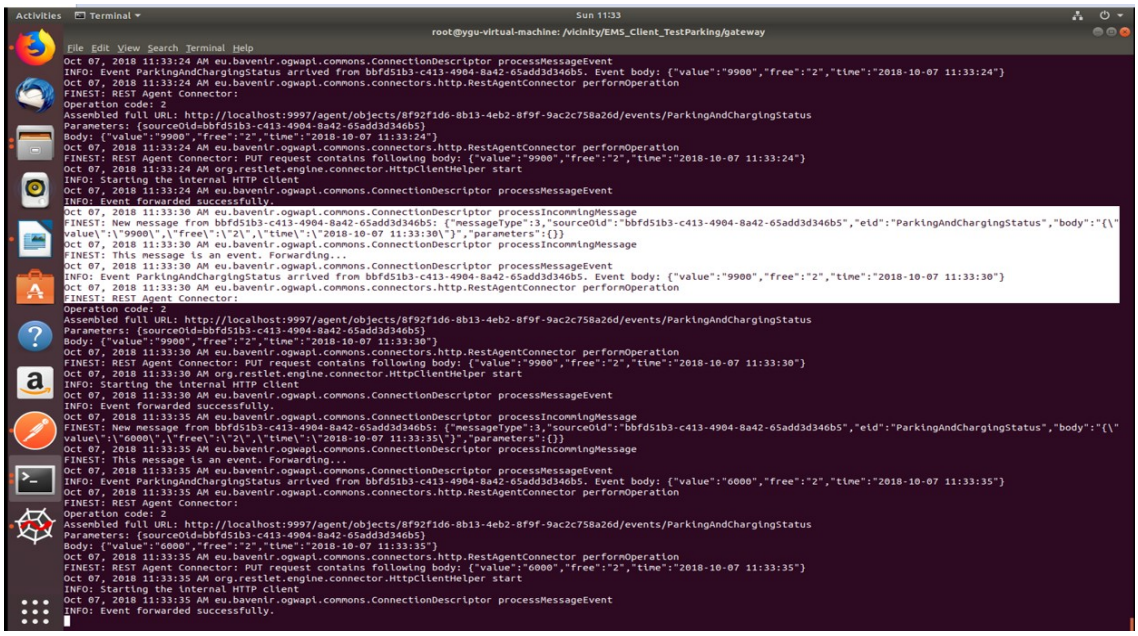
3. The VAS node gets a response stating that the event was successfully sent to a subscriber.




4. The VAS node is able to send an event to the subscriber (testing device node) which contains the number of free parking slot, real-time charging price and time-stamp.

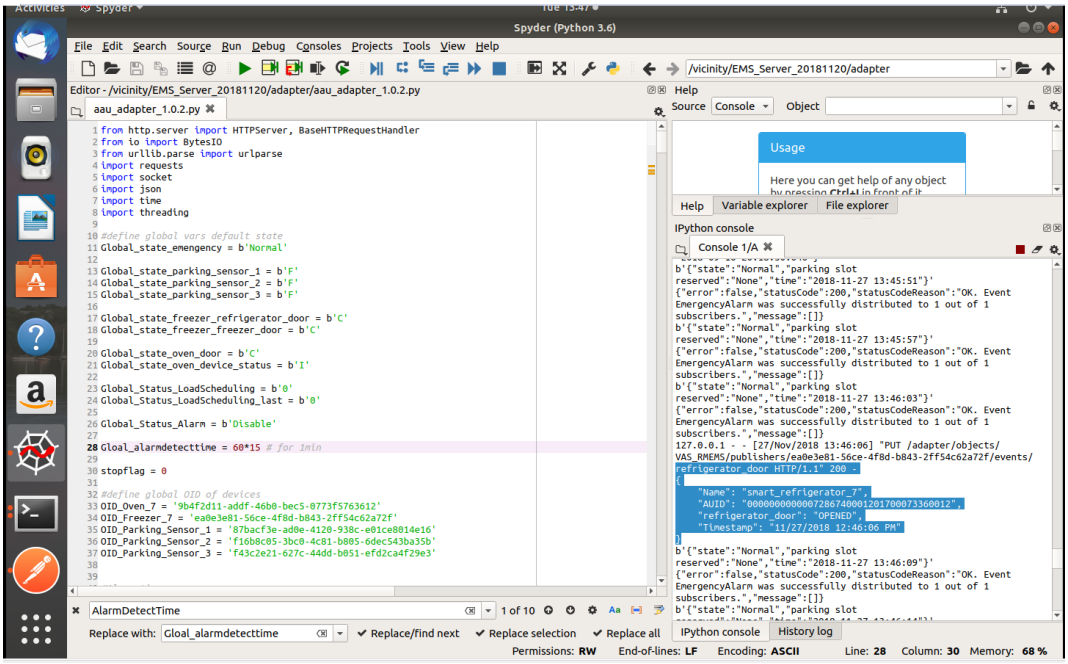


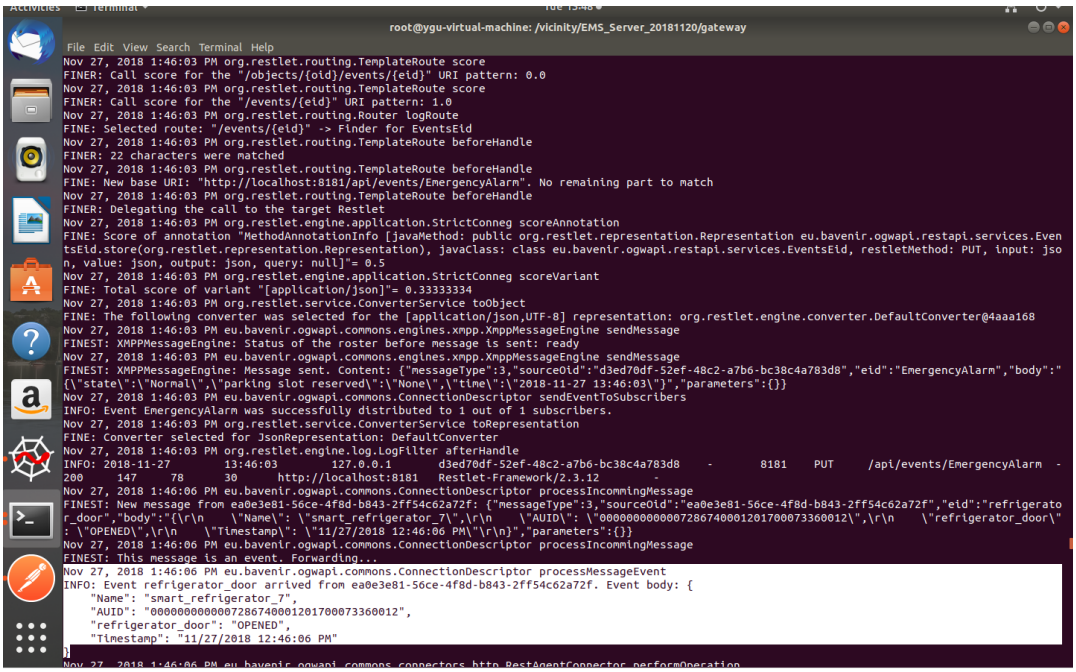
5. The testing device node is able to receive an event which contains the number of free parking slots, real-time charging price and time-stamp.



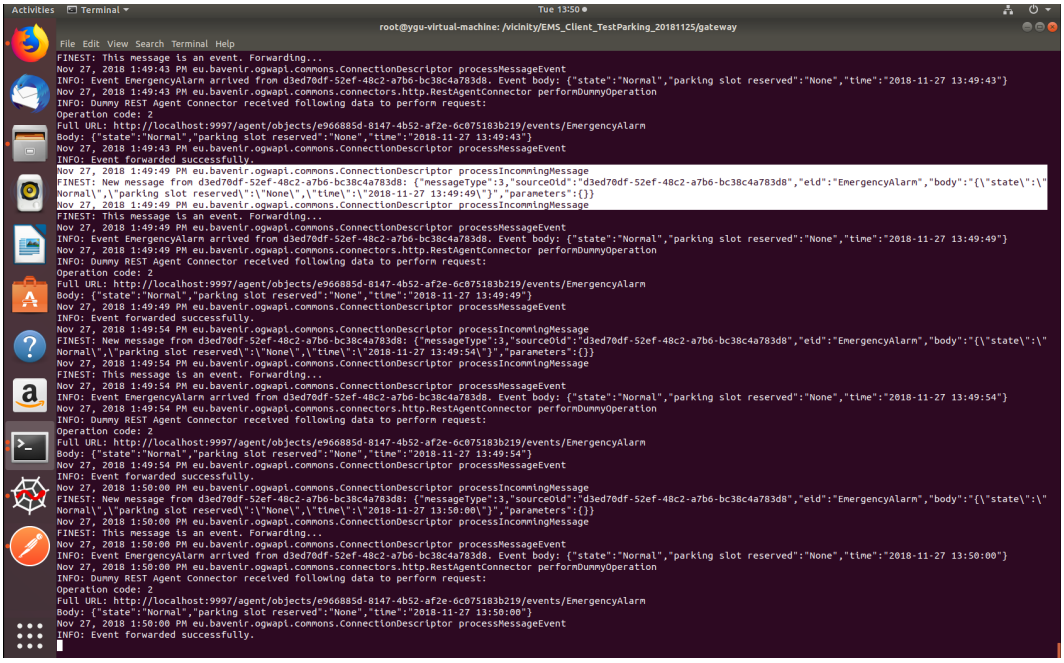
# Annex V – Internal point testing 2 - Abnormal situation identification for elderly residents (AAU - DK)

1. The refrigerator door status can be requested by the VAS adapter and gateway. The response now is ”Opened”.

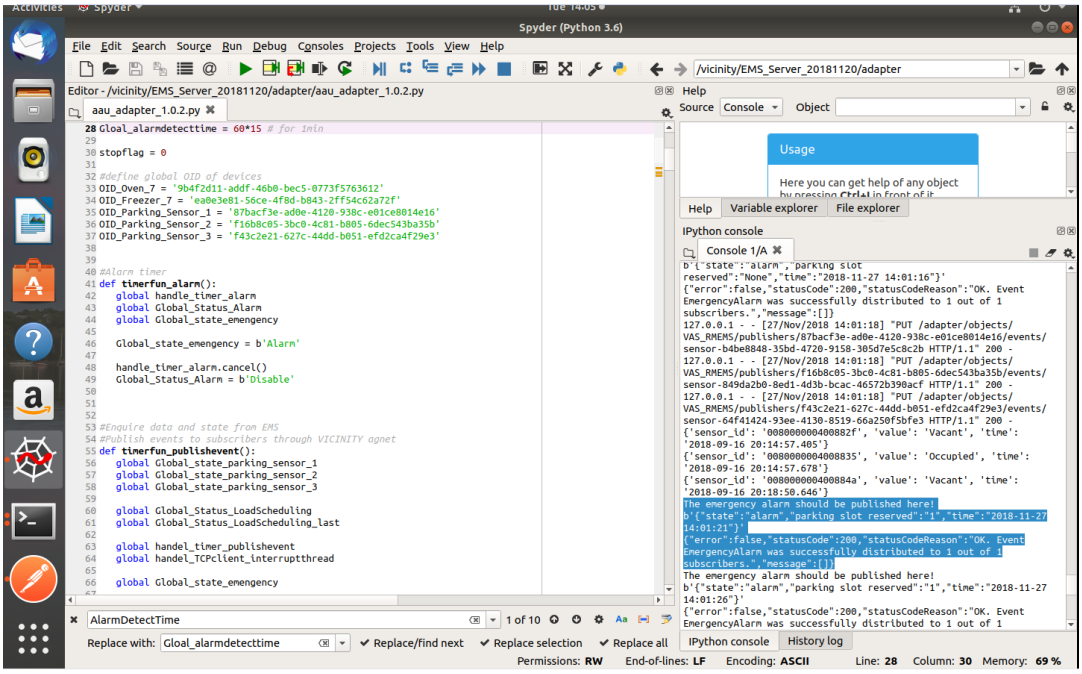


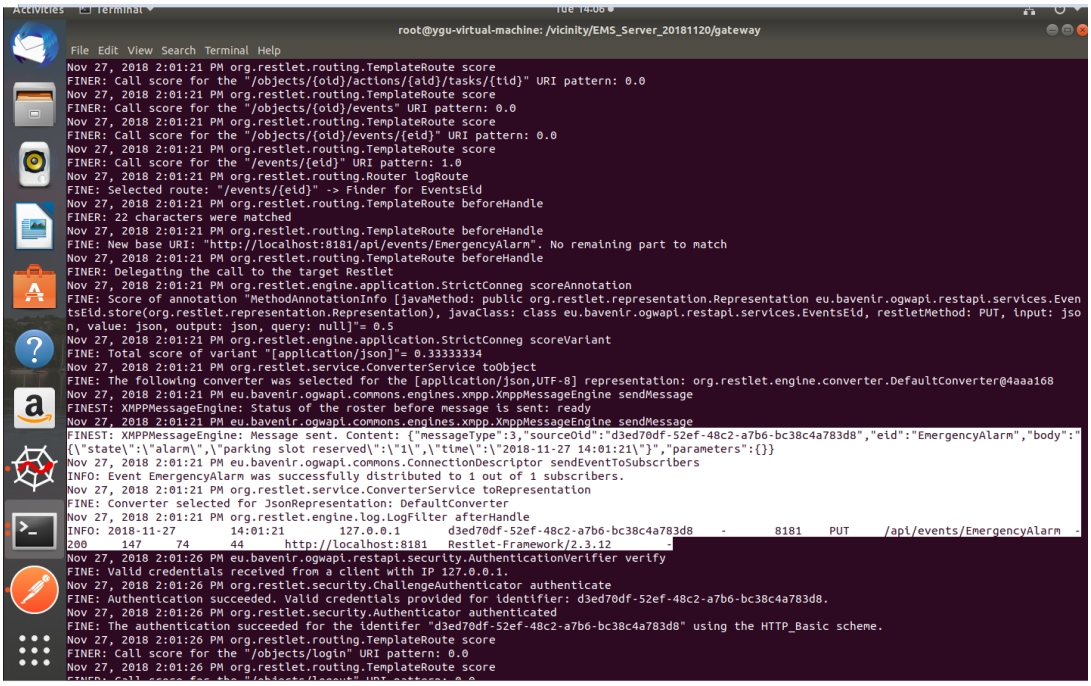



2. The care centre (a testing device node) subscribes to the event published by the VAS, The event data includes the alarm state which now is normal, parking slot reservation number, and the time-step.

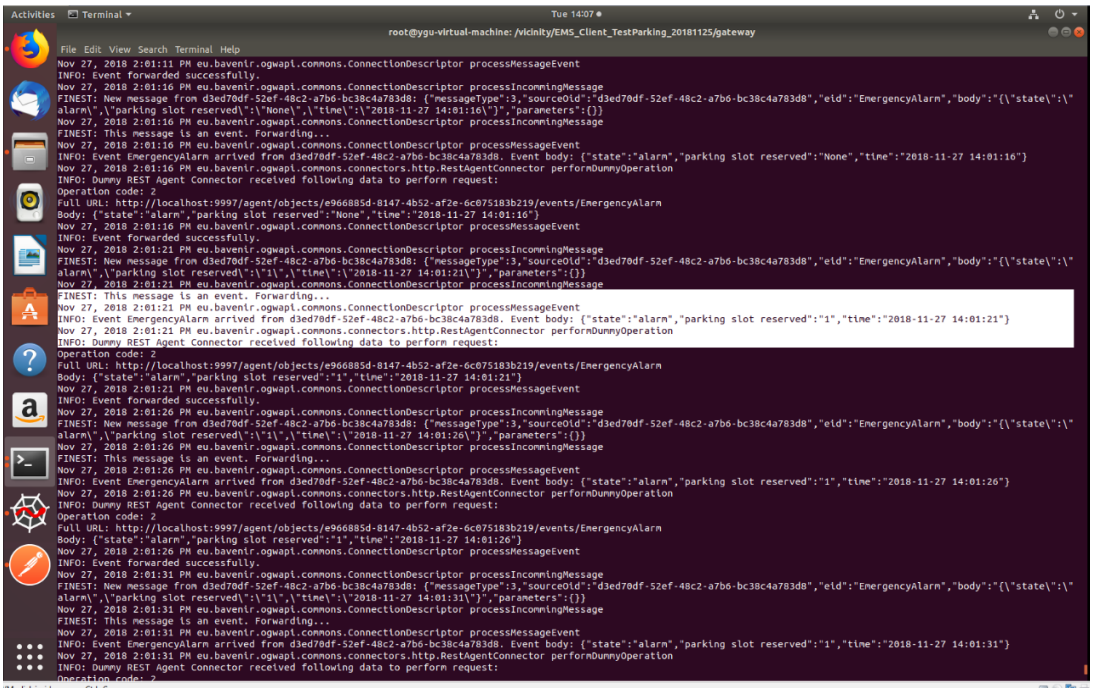



3. Once the refrigerator door is opened more than 15 minutes, an emergency alarm event is published from the VAS adapter and gateway to the subscriber. The data contains the alarm state, reserved parking slot number for the ambulance, and the time-step.



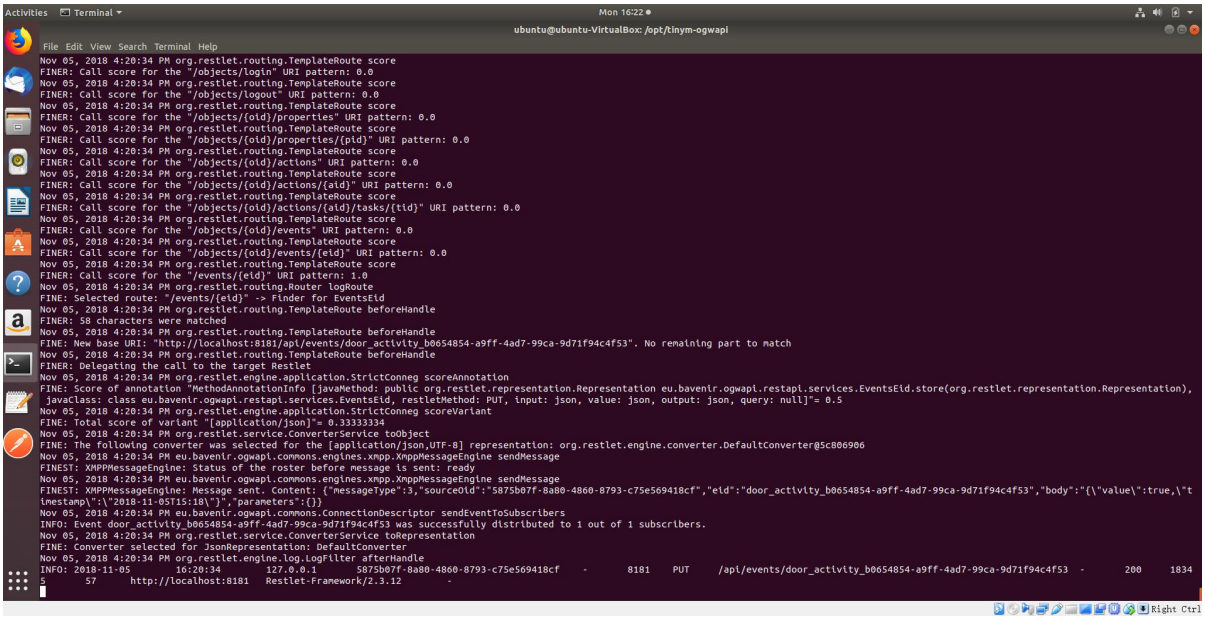



4. The care centre (a testing device node) receives the emergency alarm event which now contains the alarm state (alarm), reserved parking slot number (1) for the ambulance, and the time-step.

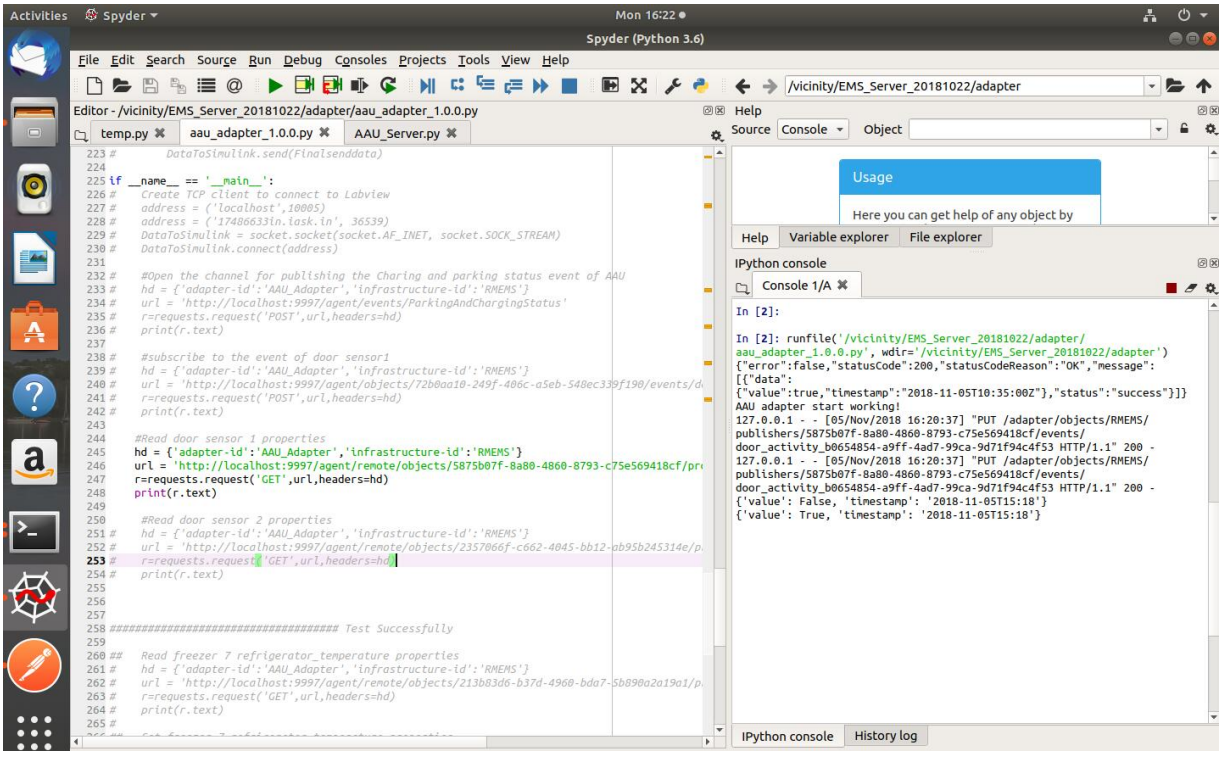


# Annex VI – Internal point testing 3 - Cleaning and Waste Removal Notification (AAU - DK)

1. The door sensor status is published through VICINITY gateway.



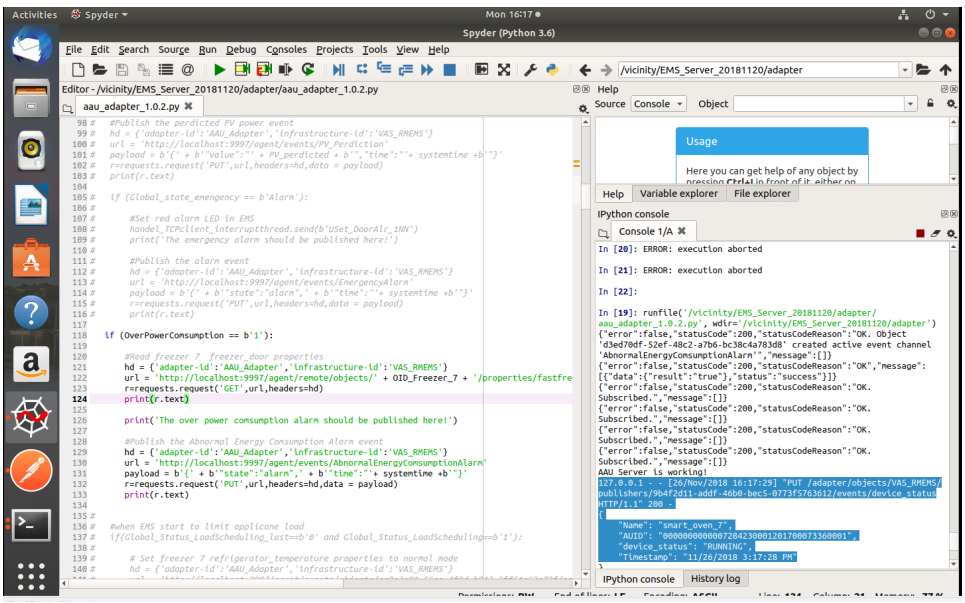

2. The Cleaning Notification VAS can get the door sensor status by subscribing its event.



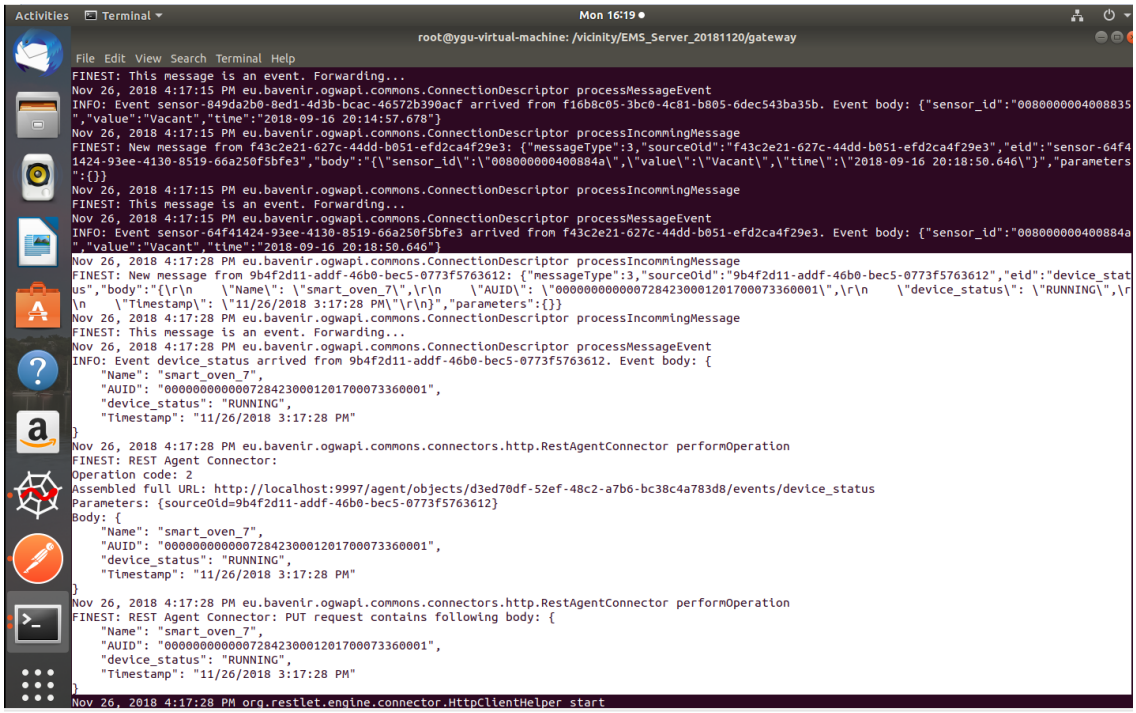
# Annex VII – Internal point testing 4 - Energy consumption optimization and abnormal alarm (AAU - DK)

1. The real-time experimental platform in AAU IoT-microgrid Lab.

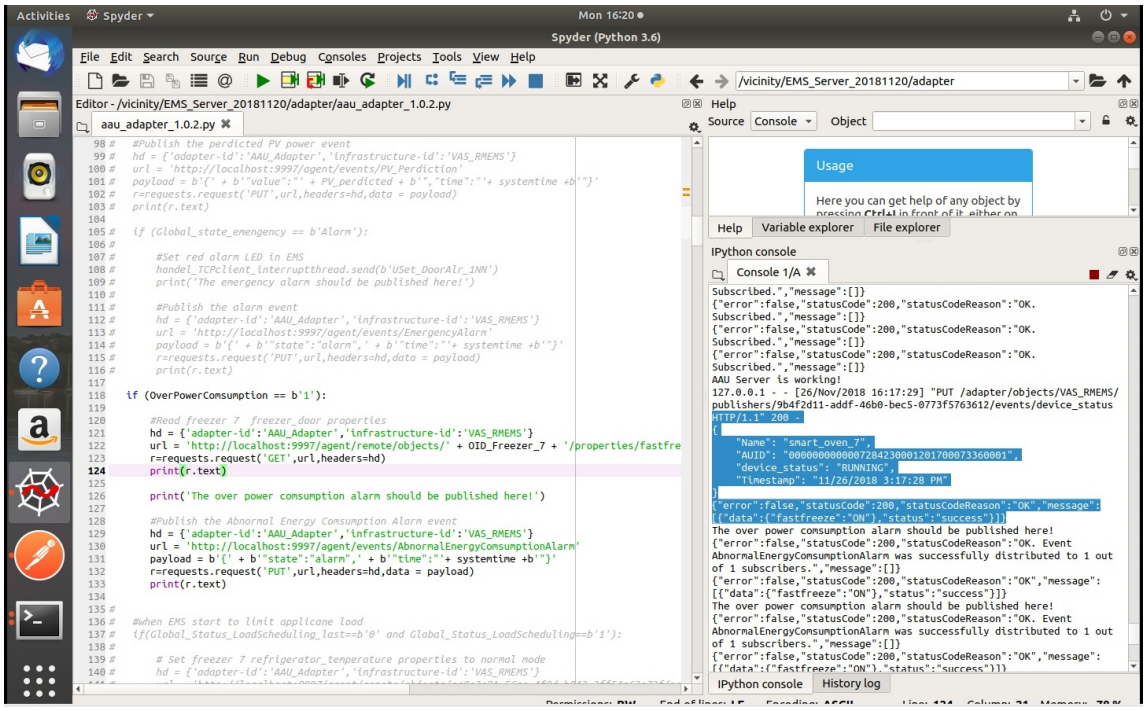

2. GORENJE oven is running and the event can be subscribed by the VAS.



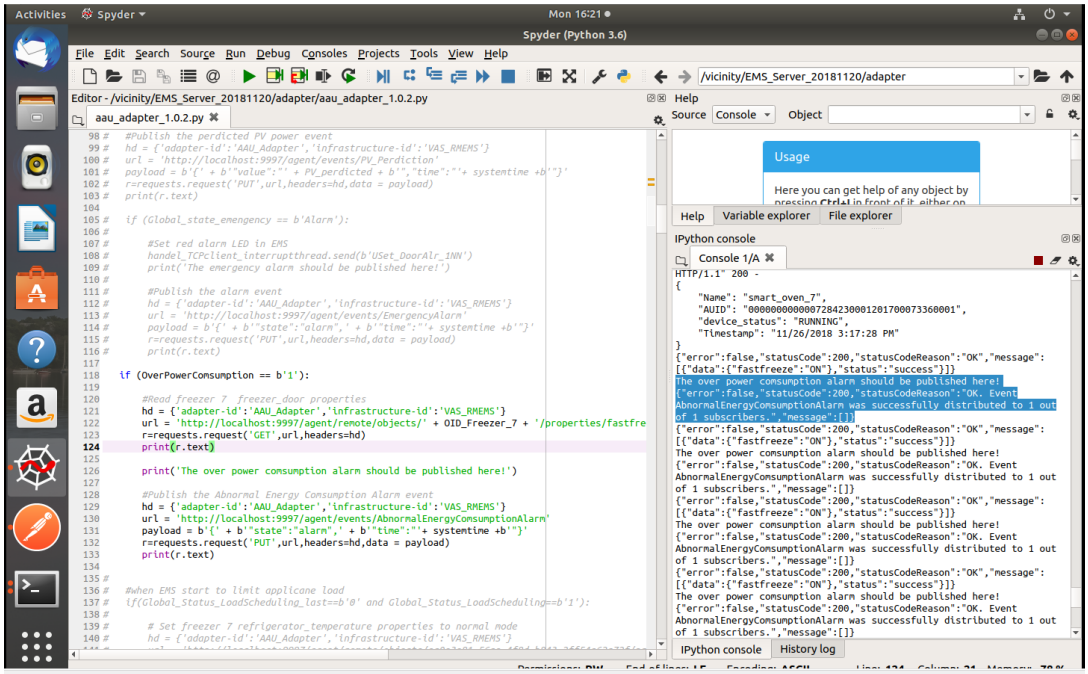
3. GORENJE oven is running and the event is sent to VAS gateway.



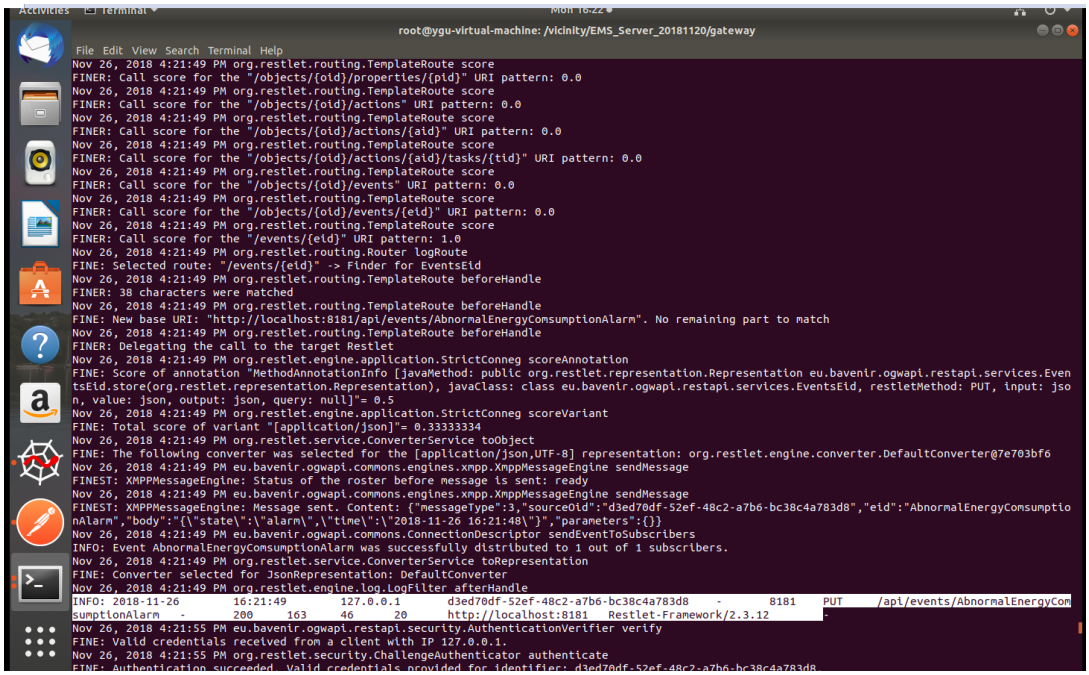

4. The working status of the refrigerator can be requested by VAS and now the response is ”fastfreeze”.



5. After a while, the total energy consumption is over the normal value. Thus, an energy consumption abnormal alarm is sent from the VAS to the subscriber.



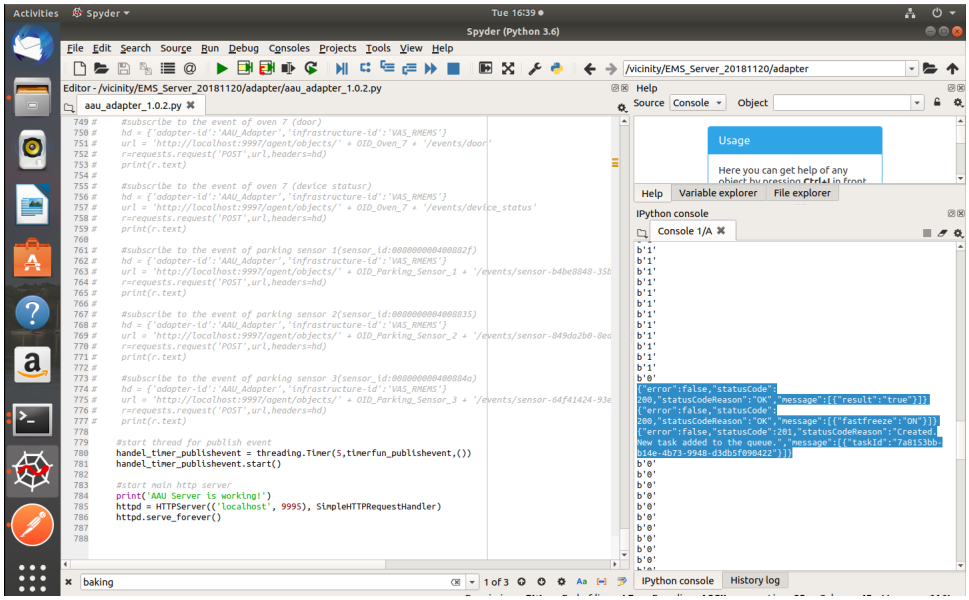

6. The energy consumption abnormal alarm is sent from the VAS gateway to the subscriber.



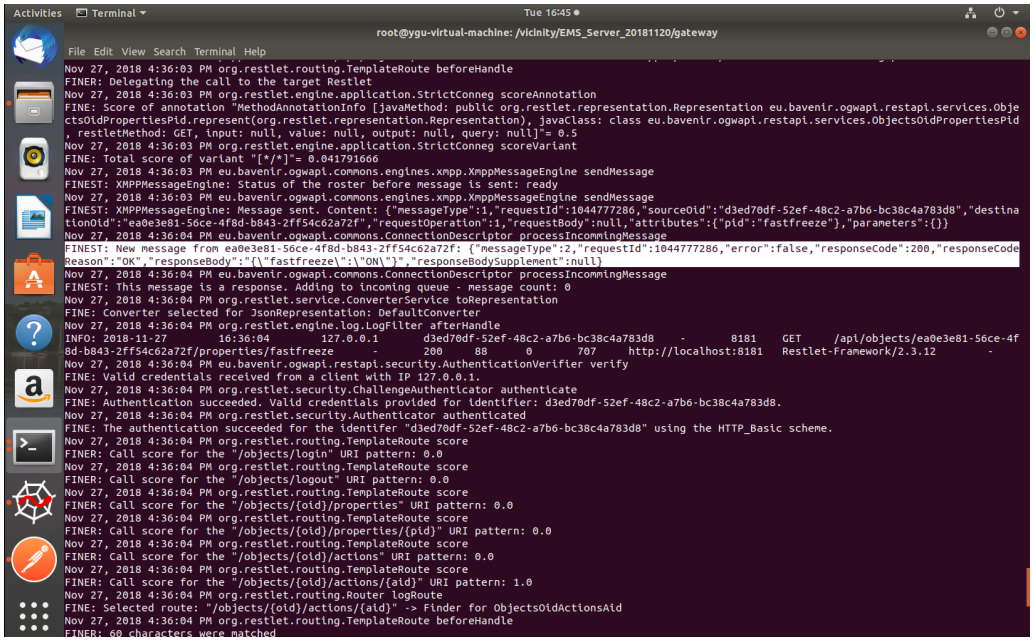
# Annex VIII – Internal point testing 5 - Optimal Scheduling and Microgrid Operation (AAU - DK)

1. The real-time experimental platform in AAU IoT-microgrid Lab.

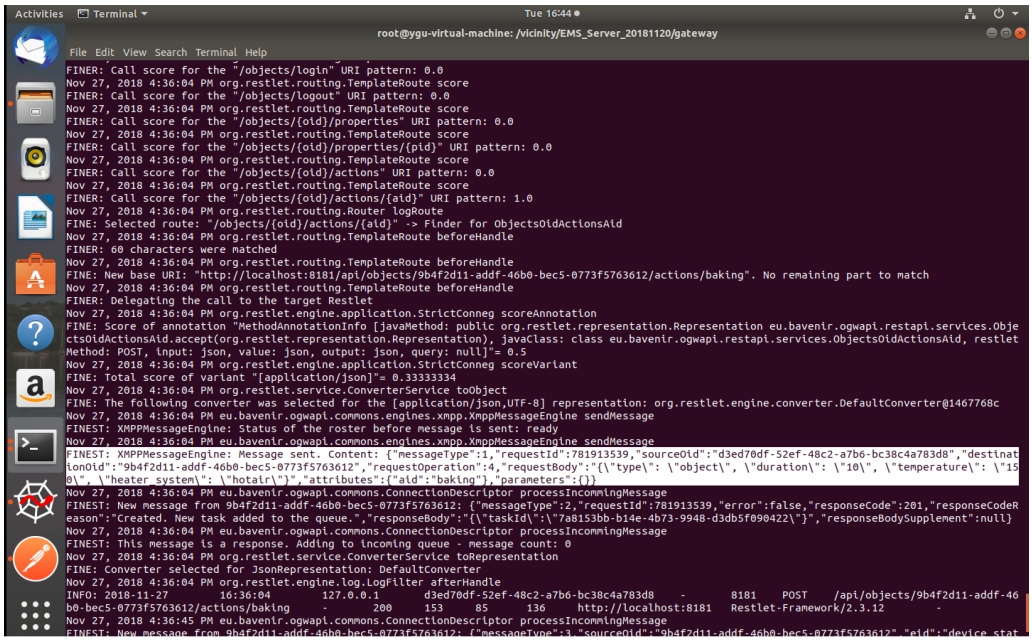

2. Once the SoC of battery reaches 80%, the VAS will set the baking parameter and send the baking command to GORENJE oven and put the GORENJE refrigerator working status to “Fastfreeze”. The commands are sent oud and VAS adapter receives the responses.



3. VAS puts the GORENJE refrigerator working status to “Fastfreeze”.



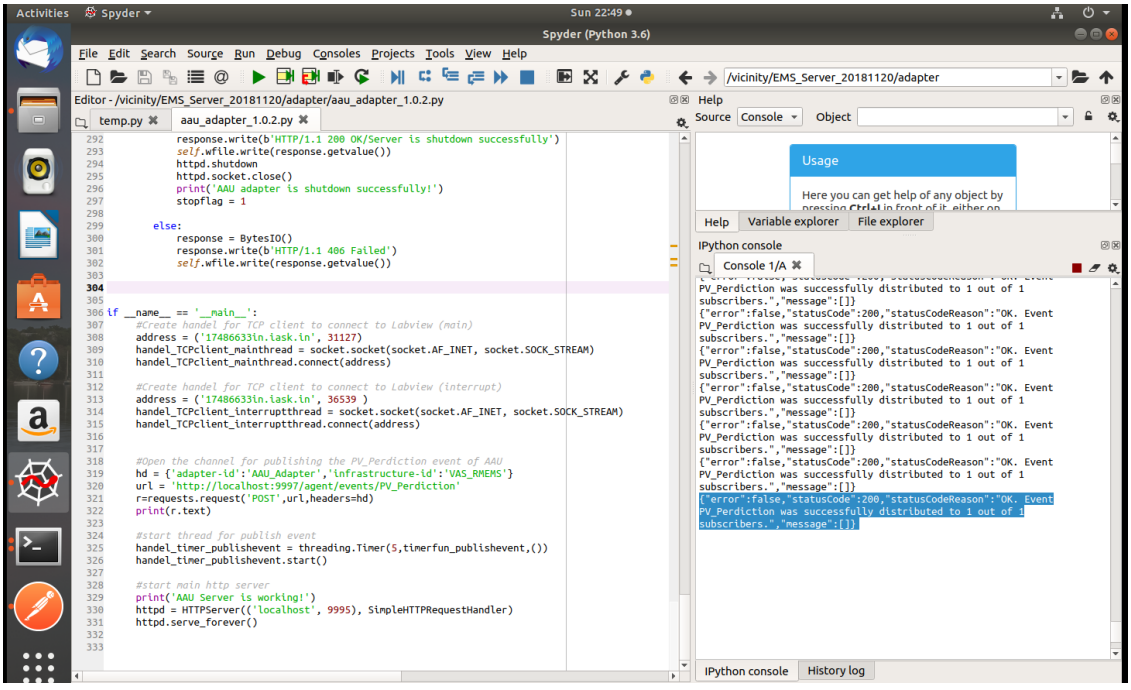

4. VAS sets the baking parameter and sends the baking command to GORENJE oven.



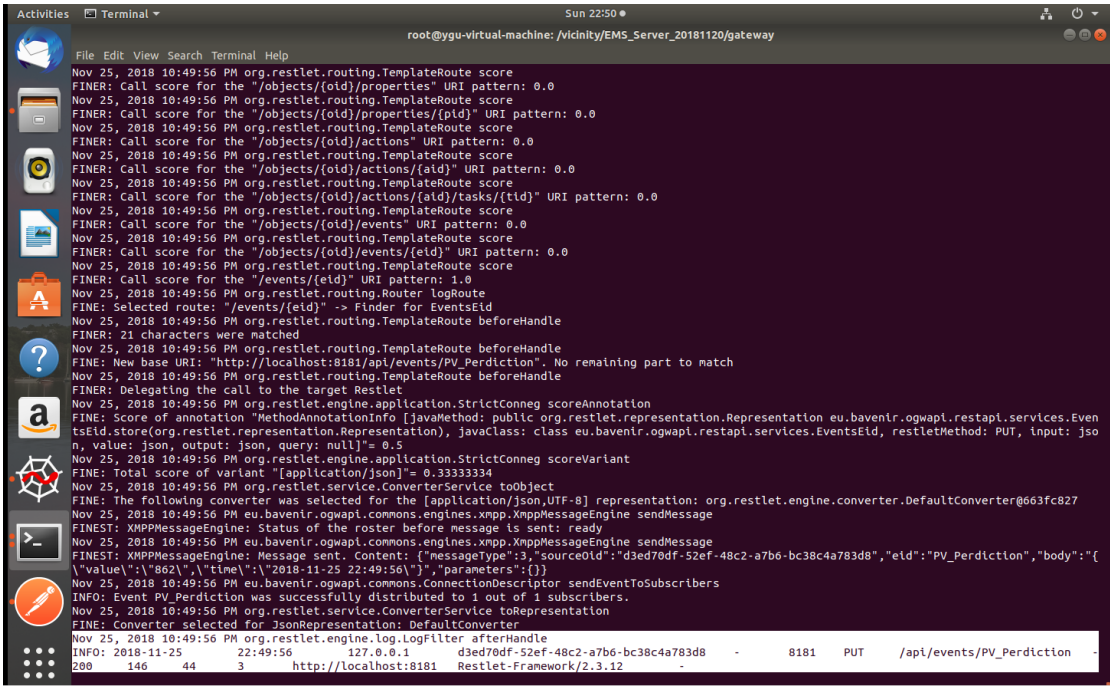
# Annex IX – Internal point testing 6 – Solar irradiance forecast (AAU - DK)

1. The real-time experimental platform in AAU IoT-microgrid Lab.

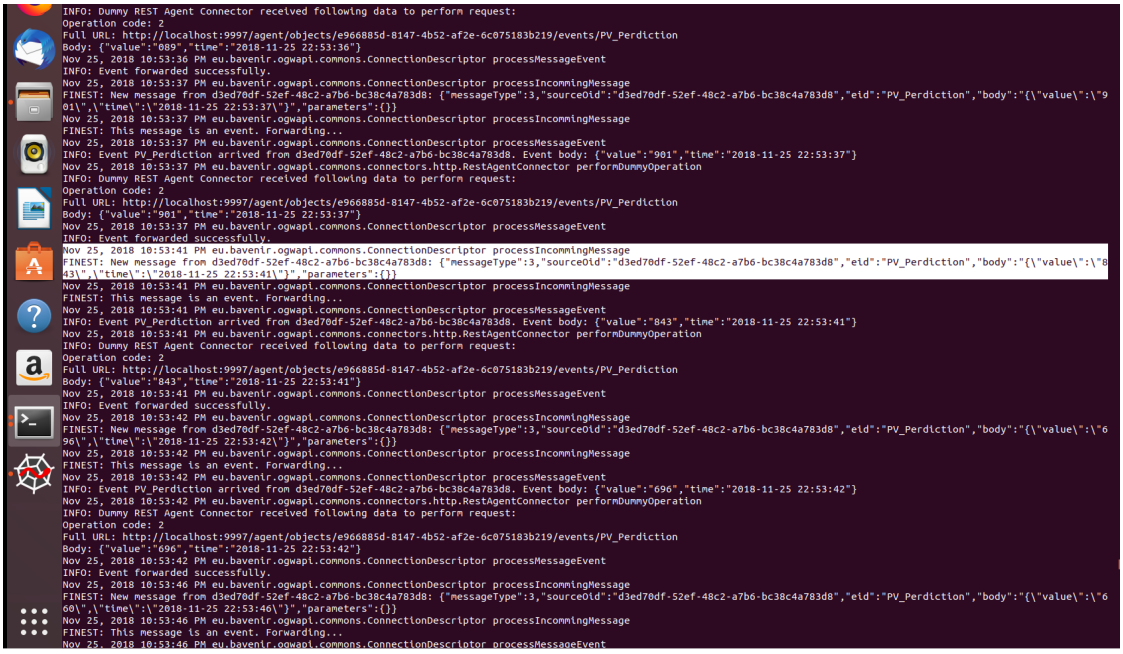

2. The event about solar irradiance prediction is successfully sent from VAS adapter to the subscriber.



3. Response received at the VAS gateway.




4. The solar forecast data is received at the subscriber's gateway.



# Annex X – Internal point testing 7 - Privacy testing using a Smart Home scenario (CERTH/ITI - GR)

Privacy testing (a):
The Organisation of the VAS receives a contract request for the building sensors.

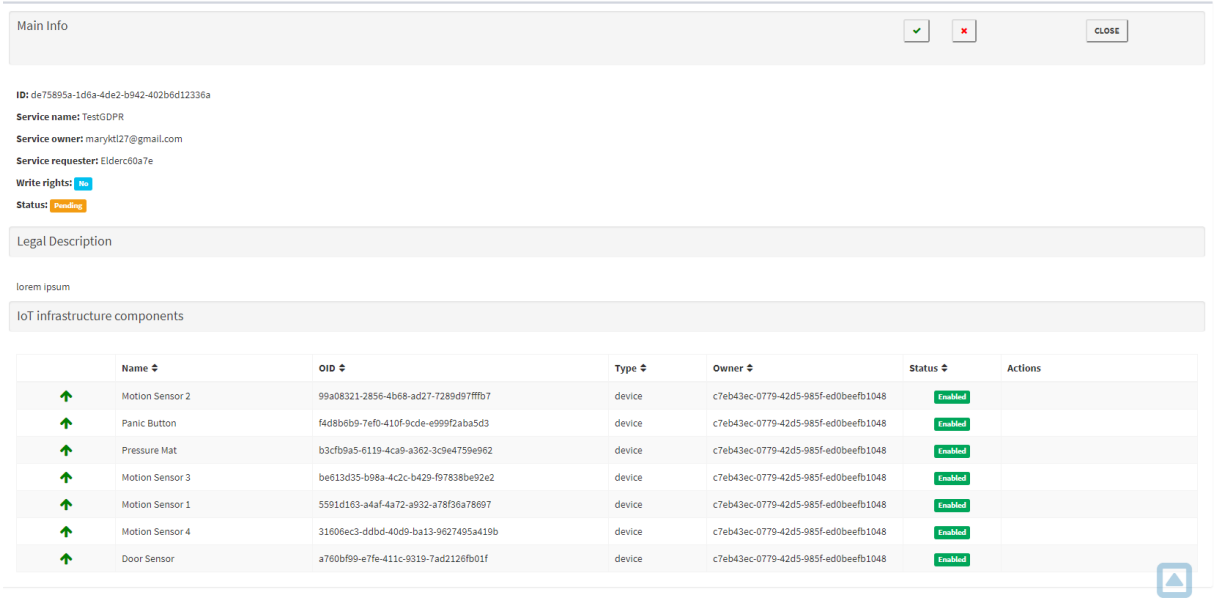



Figure X-1 Contract request needs approval

The owner of the VAS accepts the contract.

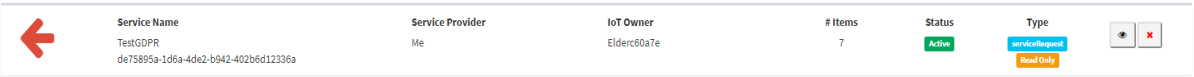



Figure X-2 Approved contract between VAS and building sensors

The door sensor is triggered and signal is sent to the VAS.

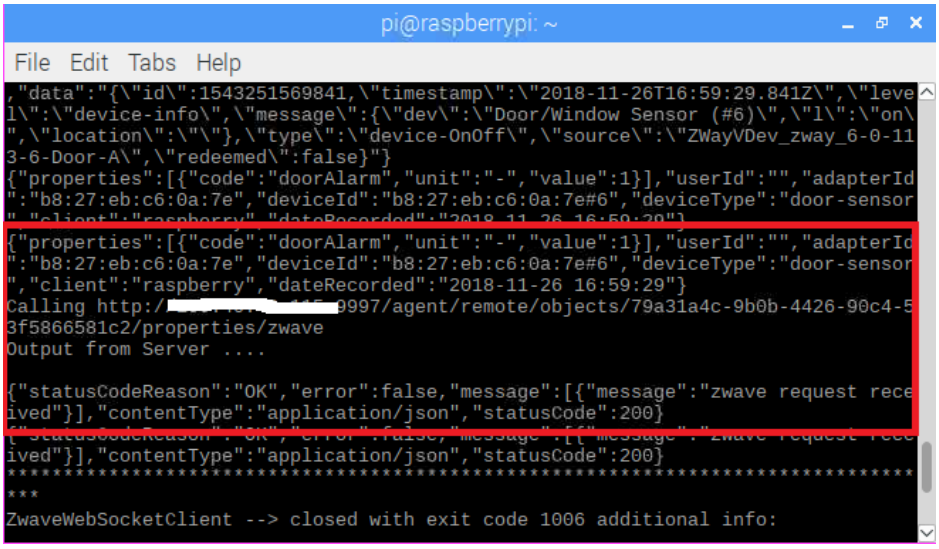



Figure X -3 Door sensor new measurement is sent to the VAS via agent (raspberry pi log)

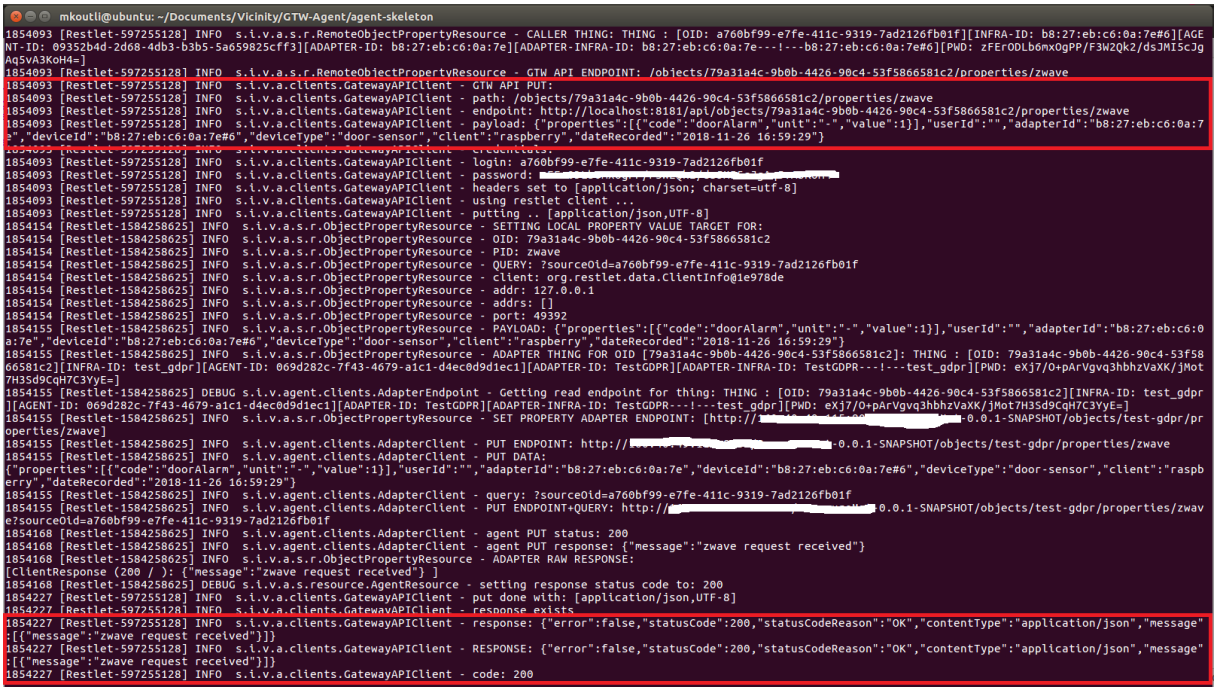



Figure X -4 Door sensor new measurement is sent to the VAS via agent (agent log)

Contract is deleted and door sensor is triggered again. This time the measurement cannot reach the VAS. The following message is produced by the gateway api : {"statusCodeReason":"Not found. Destination object 79a31a4c-9b0b-4426-90c4-53f5866581c2 is not in the list of available objects or it was not possible to send the message.","error":true,"message":[],"contentType":"application/json","statusCode":404}

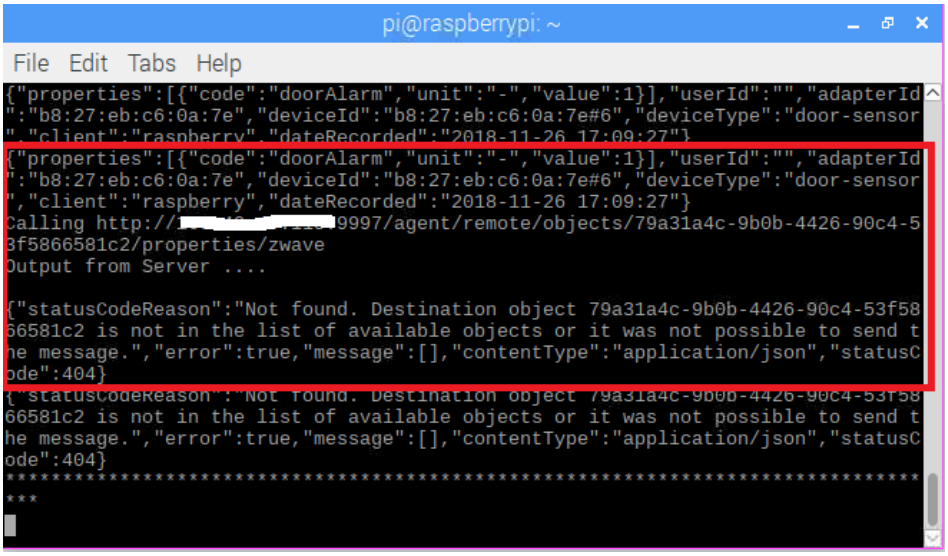



Figure X-5 Door sensor new measurement cannot be sent to the VAS via agent (raspberry pi log)

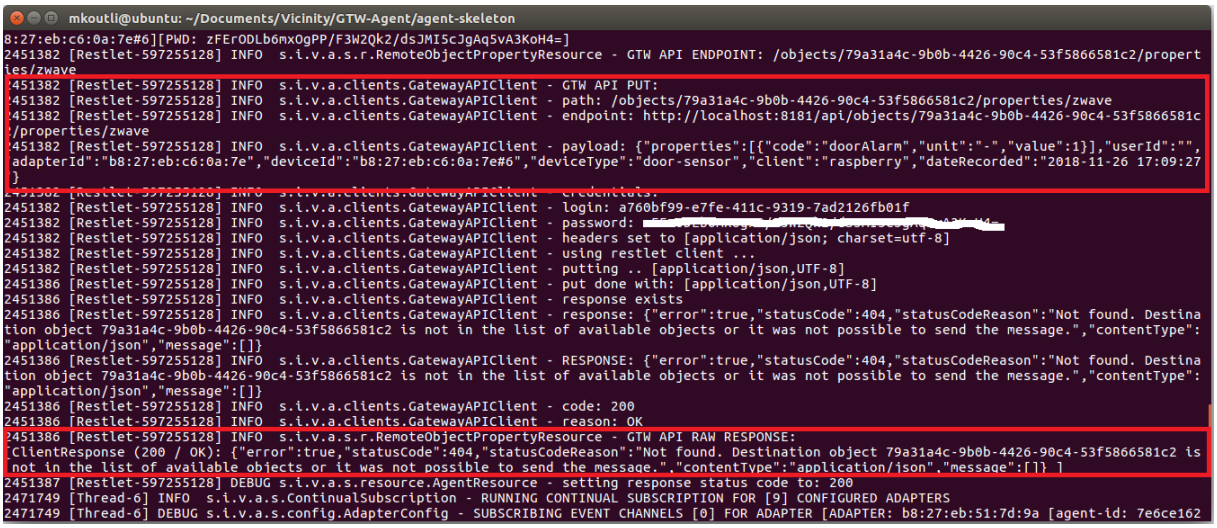



Figure X-6 Door sensor new measurement cannot be sent to the VAS via agent (agent log)

The same procedure was followed for all the sensors.

Privacy testing (b):

The VAS is not able to subscribe to Gorenje Fridge event with id “freezer_door”, since VAS and the device don’t have a contract yet. Although there is friendship between the two Organisations and the Organisation of the VAS can see the VICINITY id of the fridge and the events it exposes to VICINITY, the subscription is denied by gateway api. Below, we see the response from gateway api, to the agent request for subscription.

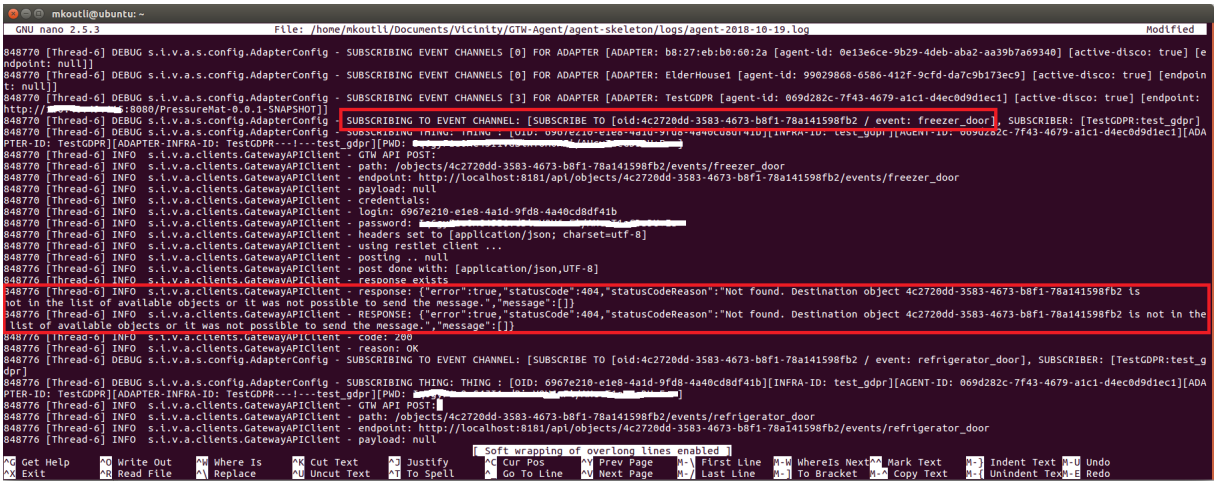



Figure X-7 Subscription to Gorenje fridge event is denied

When a contract is made between the VAS and the device, the subscription request from the agent to gateway api is accepted.

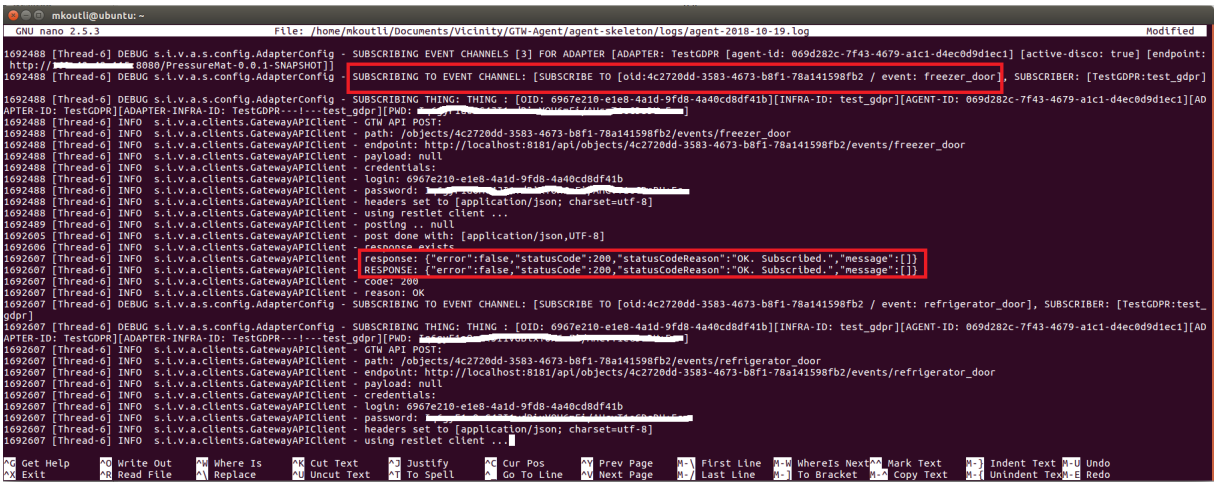



Figure X-8 Successfully subscribe to Gorenje fridge event

Opening the freezer door creates an event, which now reaches the VAS.

Figure X-9 Gorenje freezer door is opened

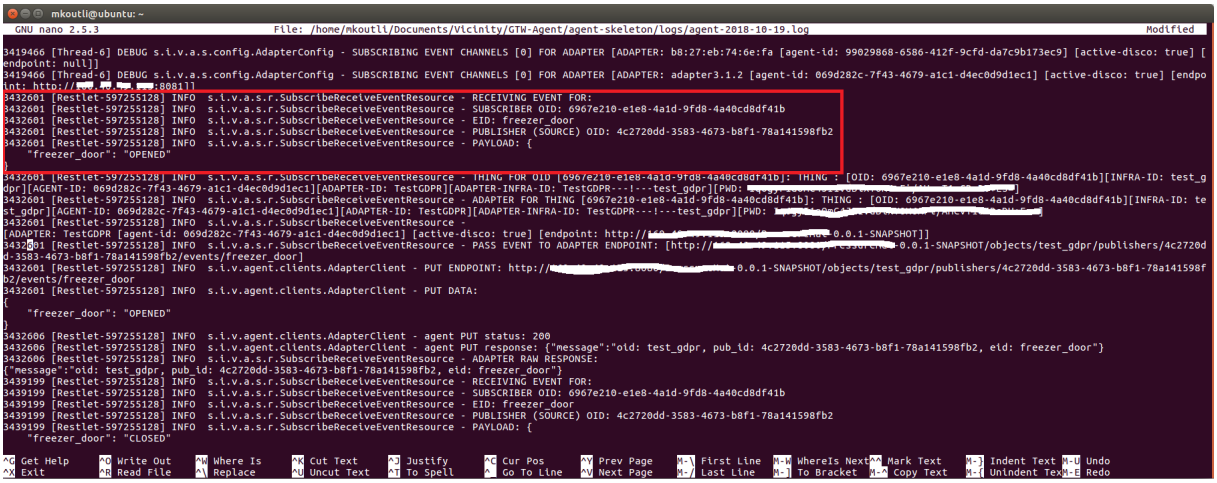



Figure X-10 Door event from Gorenje fridge reaches the VAS

Privacy testing (c):

A new property called 'test' is added in the TD of the TestGDPR VAS. The TD is pushed to the Agent, which recognizes the change and triggers update operation in NM.

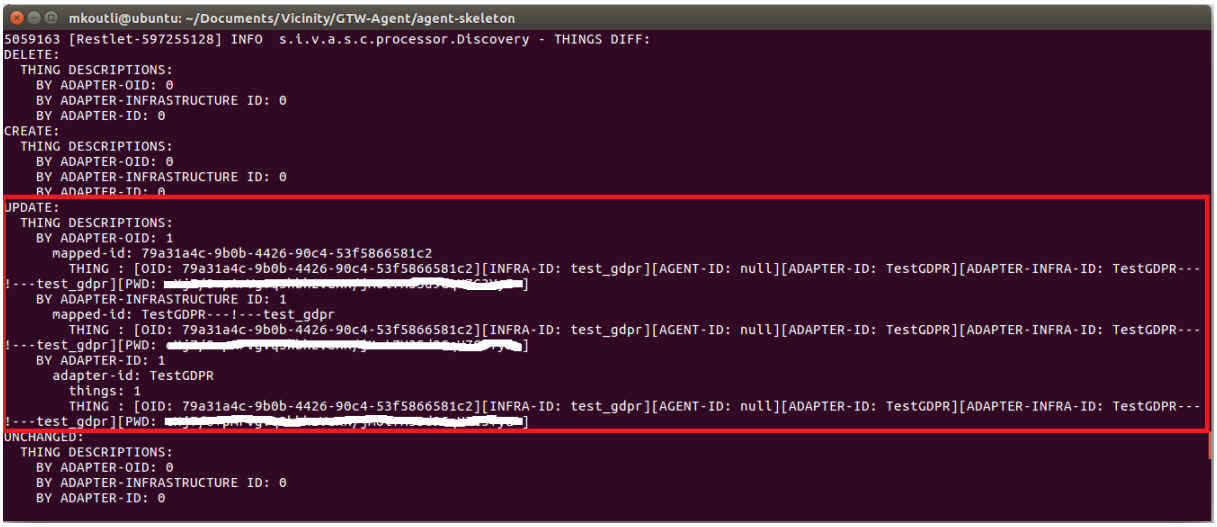



Figure X-11 Agent discovers the change in the service TD and triggers update in NM

The new property can be seen in NM interface.

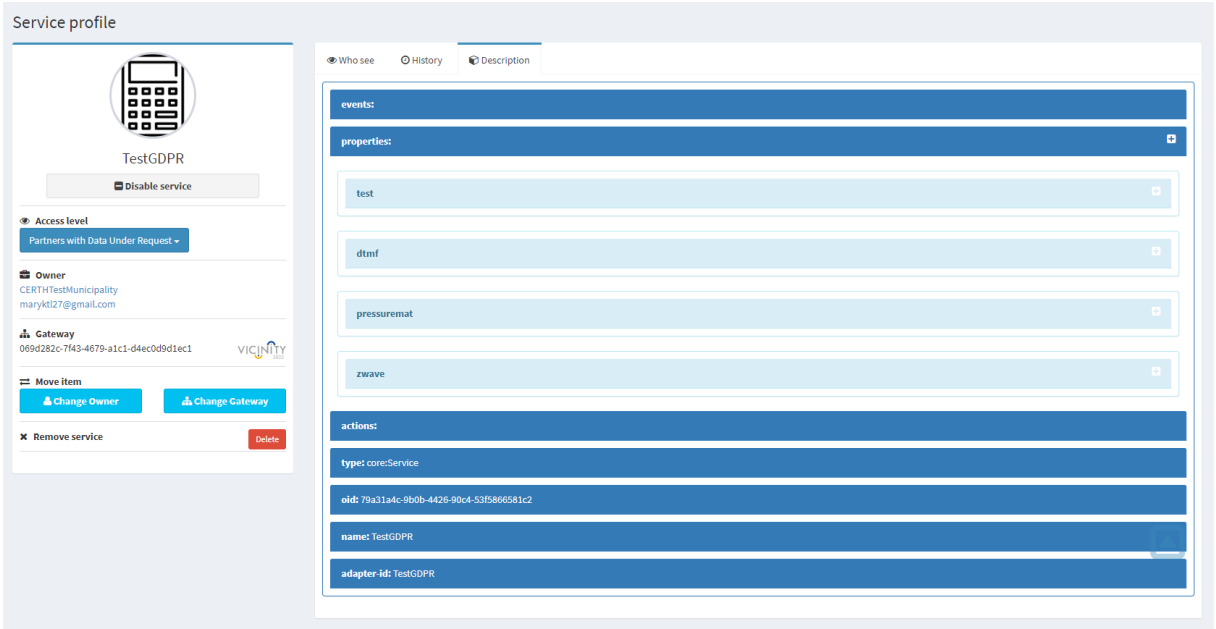



Figure X-12 Service is updated in NM

After the update, all the service's contracts become in active. The service owner will need to re-accept the contracts.

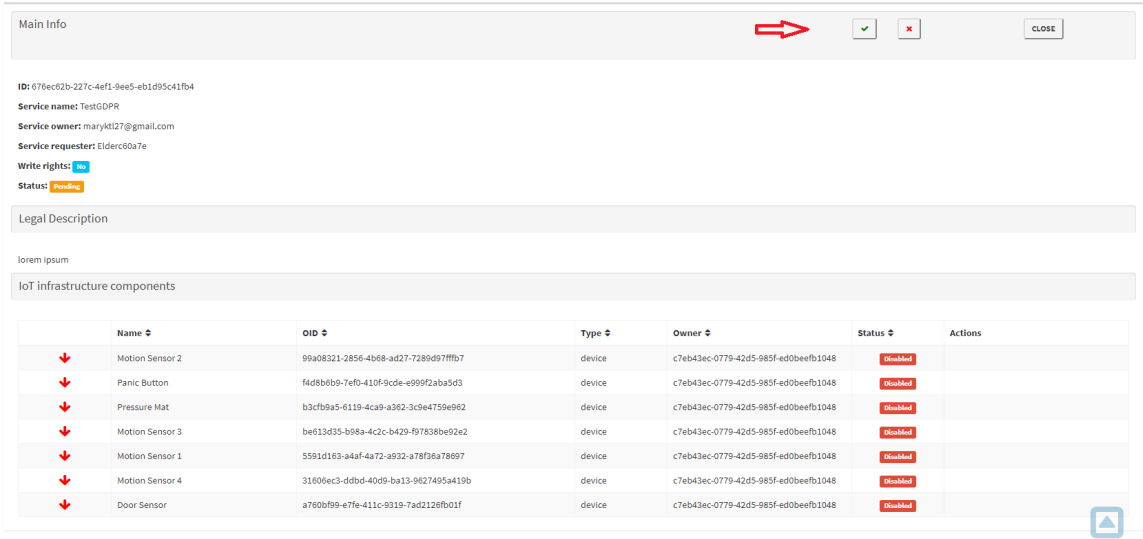



Figure X-13 Service owner needs to re-accept the contract

After re-approval of contracts on the service side, the service owner will have to wait for the infrastructure to also re-approve the contract.

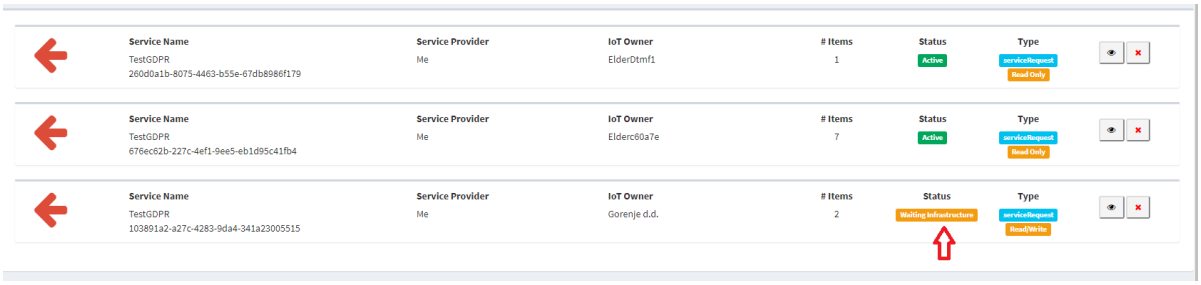



Figure X-14 Contracts need to be validated from both Organisations

The infrastructure owner can re-approve the contract with the service, for all or for only some of his devices.

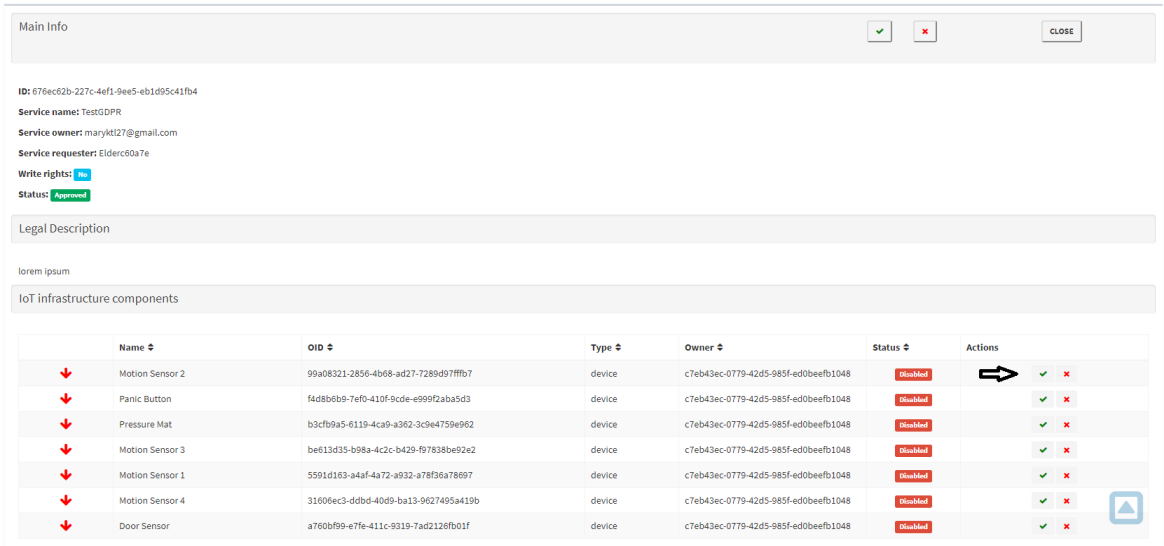



Figure X-15 Devices owner needs to re-accept the contract

# Annex XI – Internal point testing 8 – Large scale integration of eHealth infrastructures (CERTH, GNOMON - GR)

When a measurement of a sensor reaches the VICINITY adapter running on the raspberry pi or the smartphone, for the first time, the following procedure is executed:

- Create User, Organisation
- Update User roles
- Create Agent
- Create new config file in the multi-Agent (which contains both the agent id and the adapter id)
- Reconfigure Agent
- Push TD to Agent
- Enable newly registered items
- Create friendship between the created Organisation and the Municipality Organisation
- Create contracts between the registered items and the Storage and GDPR VAS

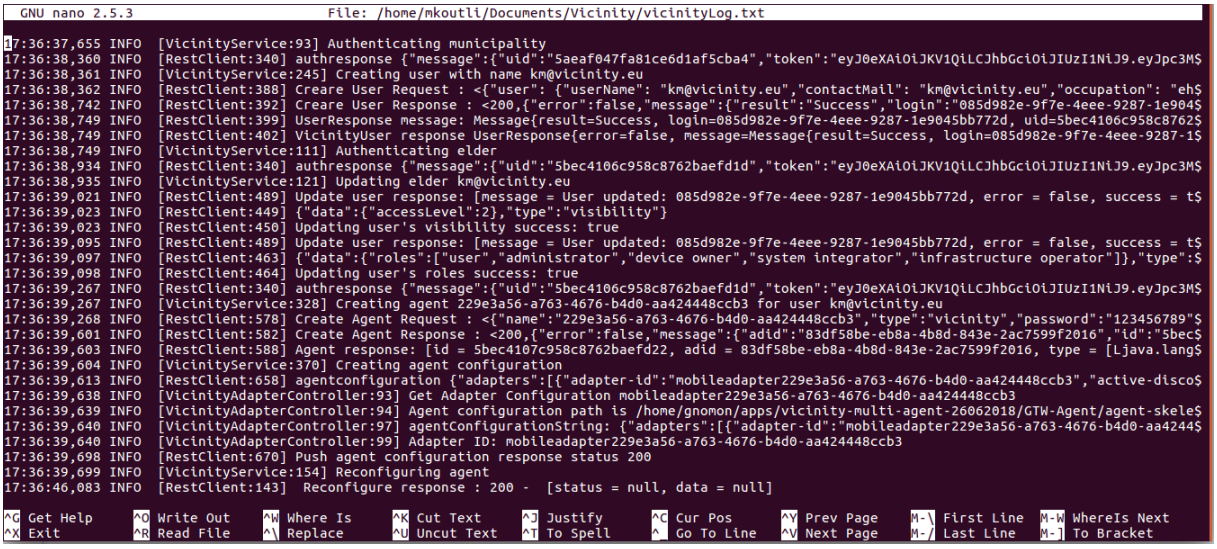



Figure XI-1 Log of automatic registration service

The created User and Organisation can be seen in the NM interface.

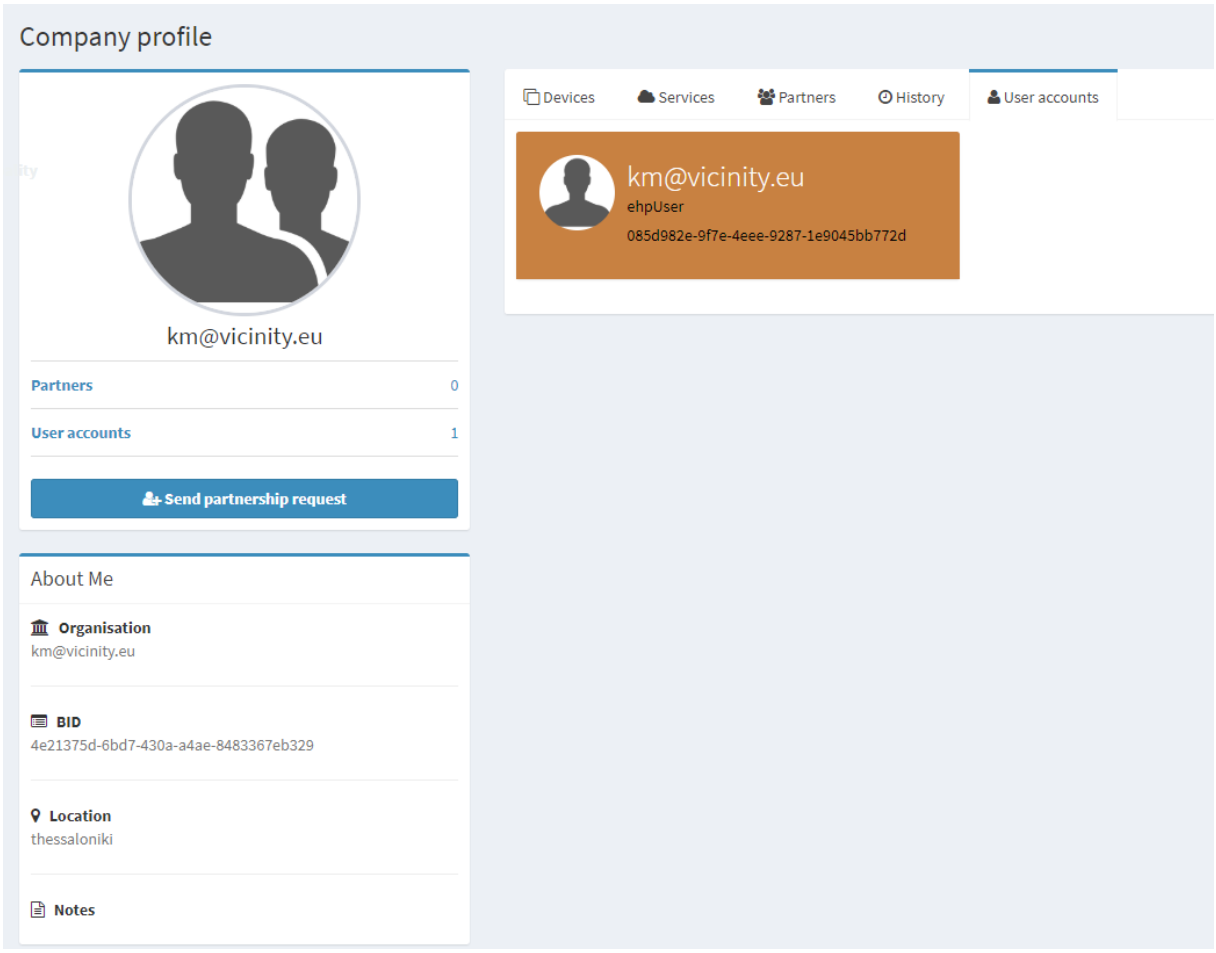
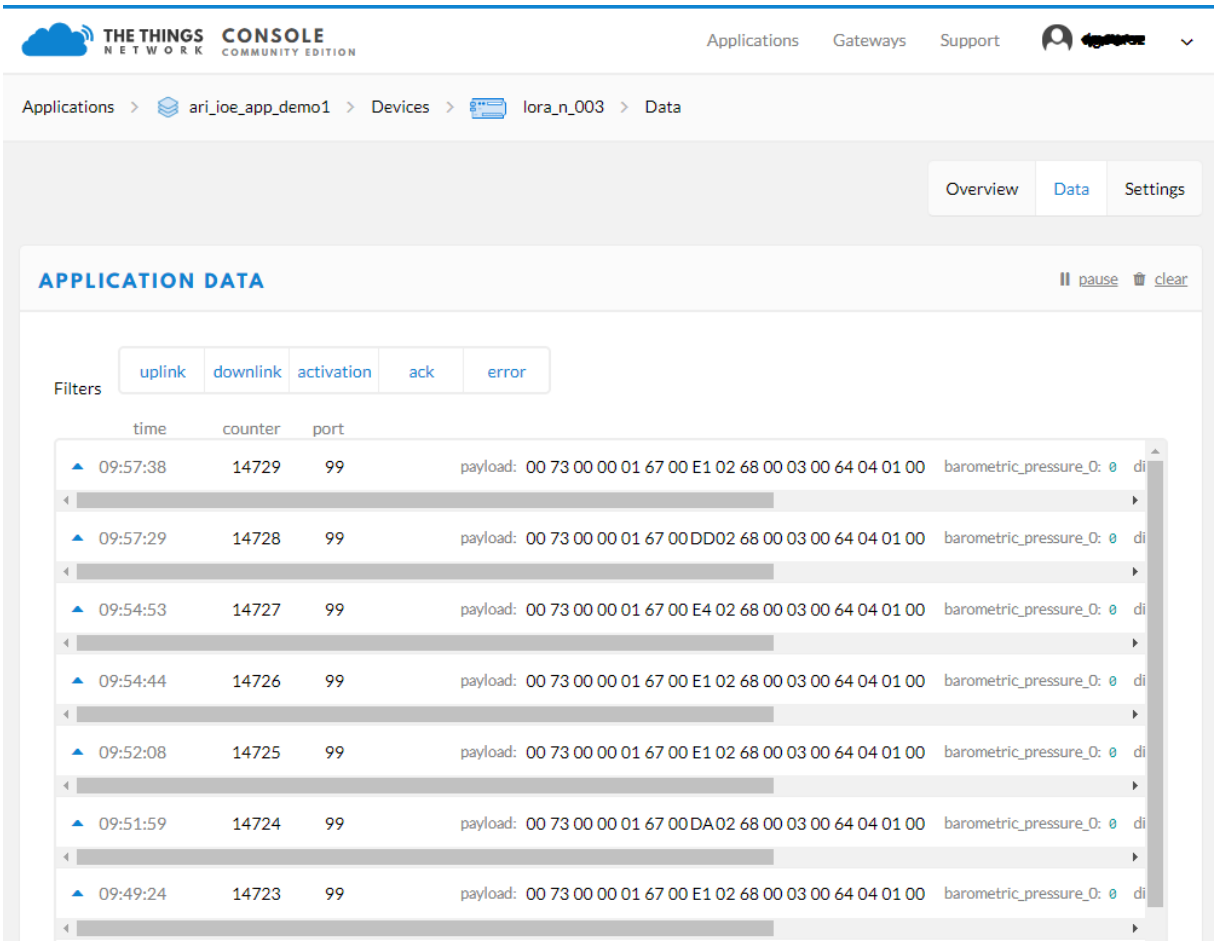



Figure XI-2 Organisation view in NM interface

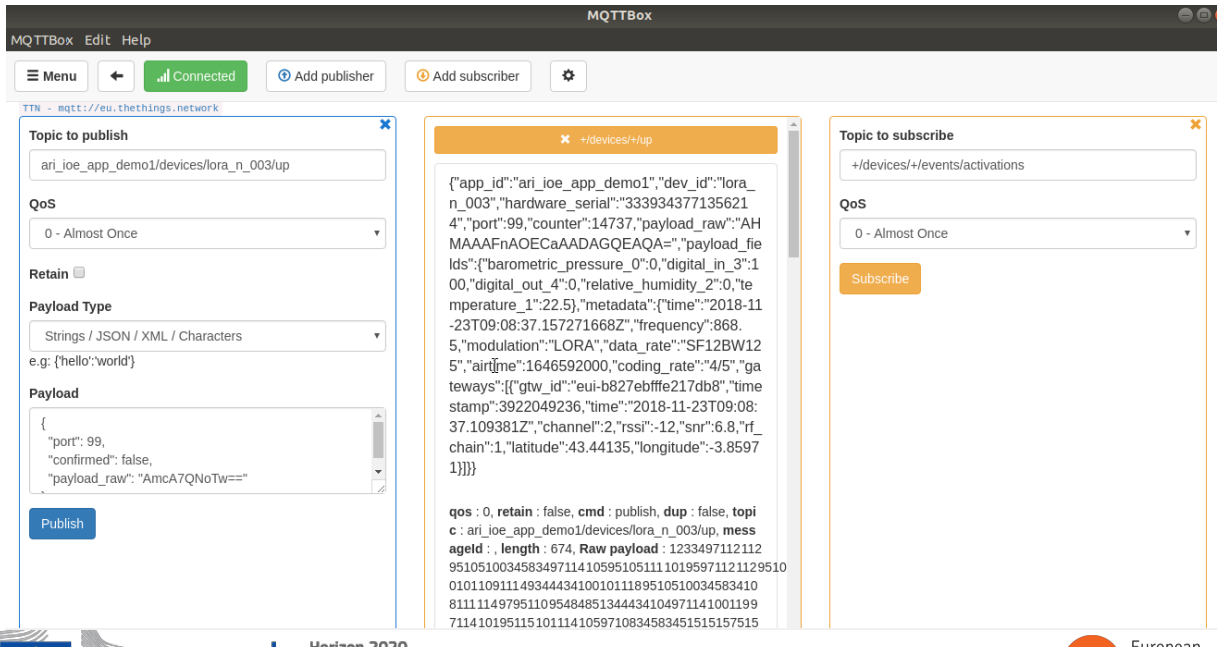
# Annex XII – Internal point testing 9 - Integration of LoRa devices (ATOS - ESP)

1. ATOS' IoE Lab devices used for the testbed deployment and validation. We can appreciate up to four different devices: the first two are Raspberry Pi 3 that will play the role of LoRaWAN nodes, likewise the last one, an embedded system based on an STM32 board, with very limited computational capacity. In the middle, the third object consists in another Raspberry Pi 3 that has attached a multi-channel LoRaWAN concentrator on top of it. In this case, this element will behave as the LoRaWAN Server.

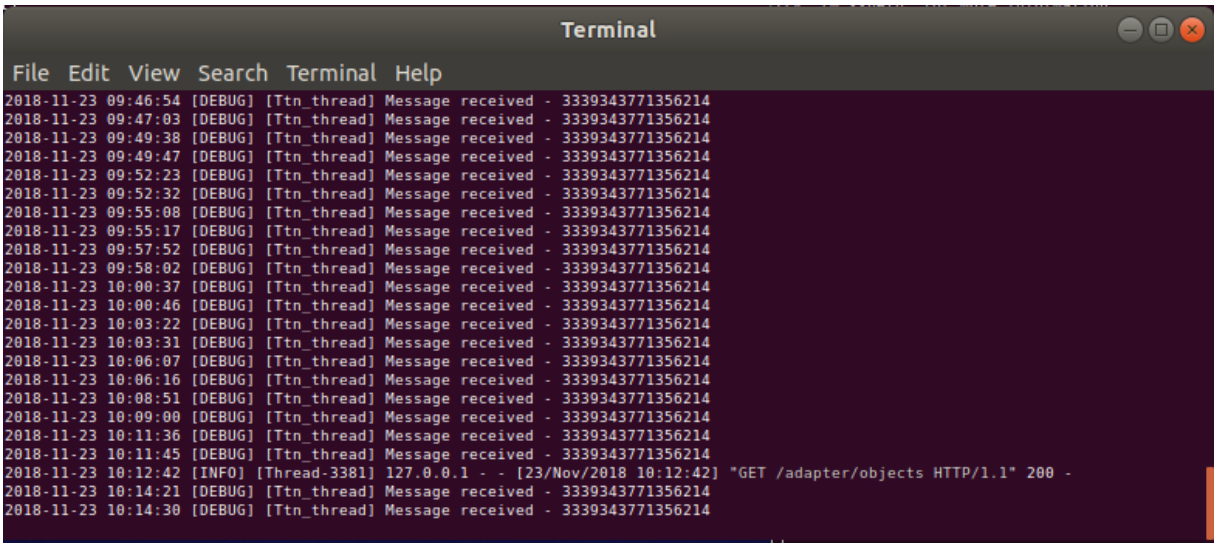
2. After the nodes and the LoRaWAN Gateway Network and Application Servers are up-and-running, we can see on TTN's Console that data is actually being sent



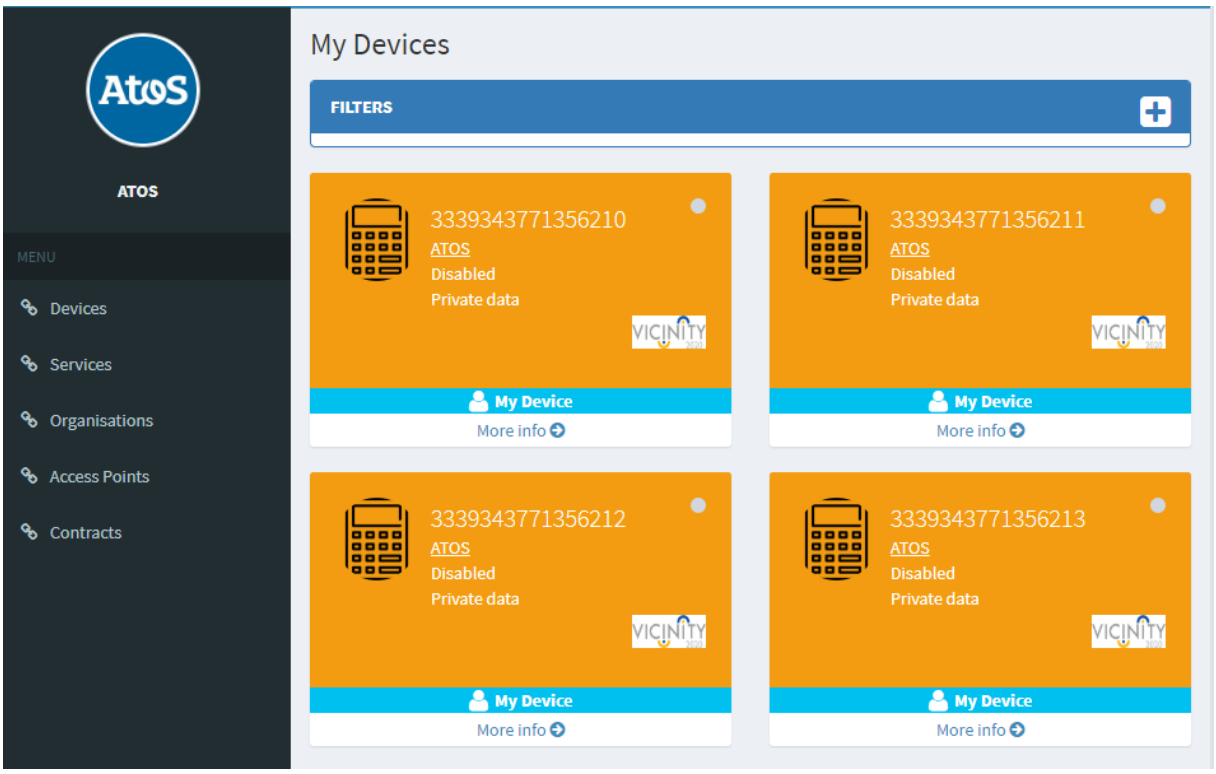

3. Thanks to the built-in, MQTT Broker provided by TTN, we can subscribe to every message generated by the nodes.



4. For the sake of illustration, once the VICINITY Agent is executed (passive discovery mode), the adapter's log displays a request, coming from the Agent, of all the available nodes.



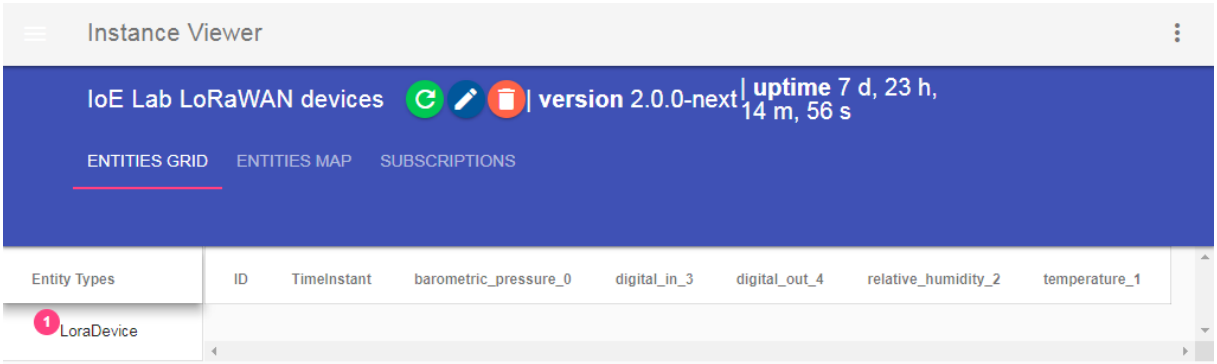

5. After all the process, the Neighbourhood Manager dashboard displays the registered devices.



# Annex XIII – Internal point testing 10 – Generic integration of FIWARE-compliant devices (ATOS - ESP)

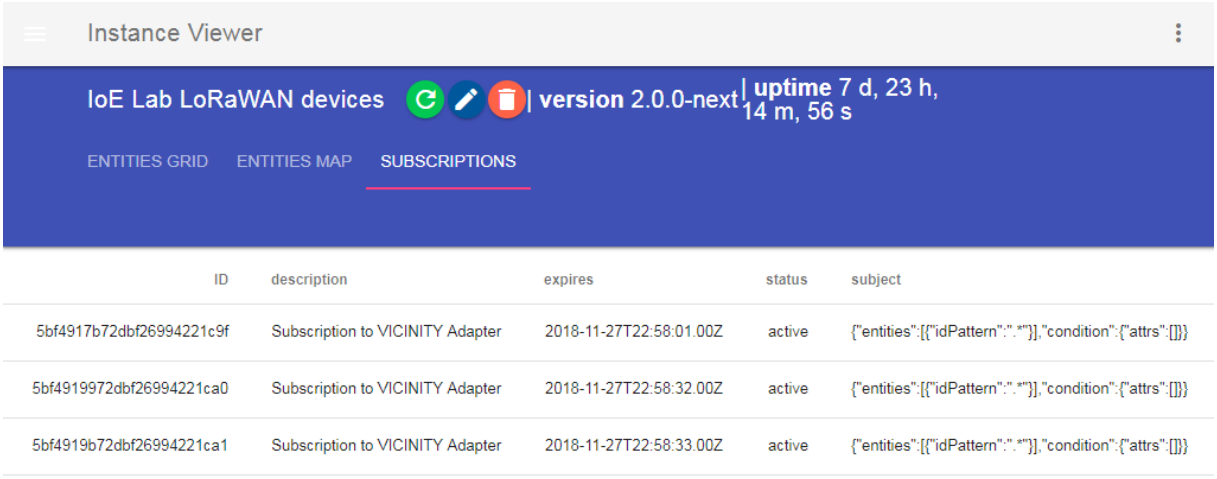
1. ATOS' IoE Lab devices used for the testbed deployment and validation. At the upper part of the picture we can see a Multitech Conduit LoRaWAN server; on the other hand, we can find below a LoRaWAN node (again, based on an STM32 board).

2. After registering the devices onto the Orion Context Broker, we can use an instance of a tool called Orion Explorer[5] to visually check that the device has been successfully registered.



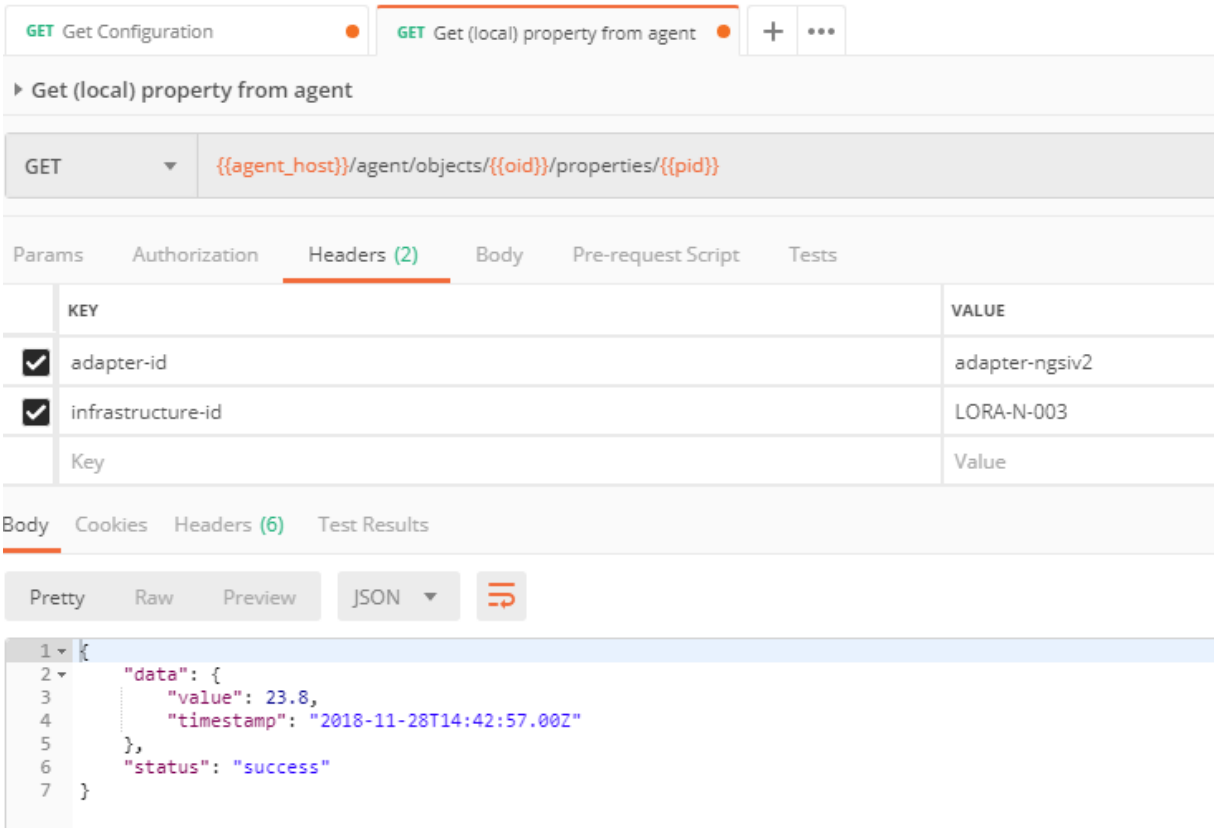

3. At the same time, we can see as the VICINITY adapter subscribe to ORION's events (all devices)

[5] https://github.com/VM9/fiware-orion-explorer

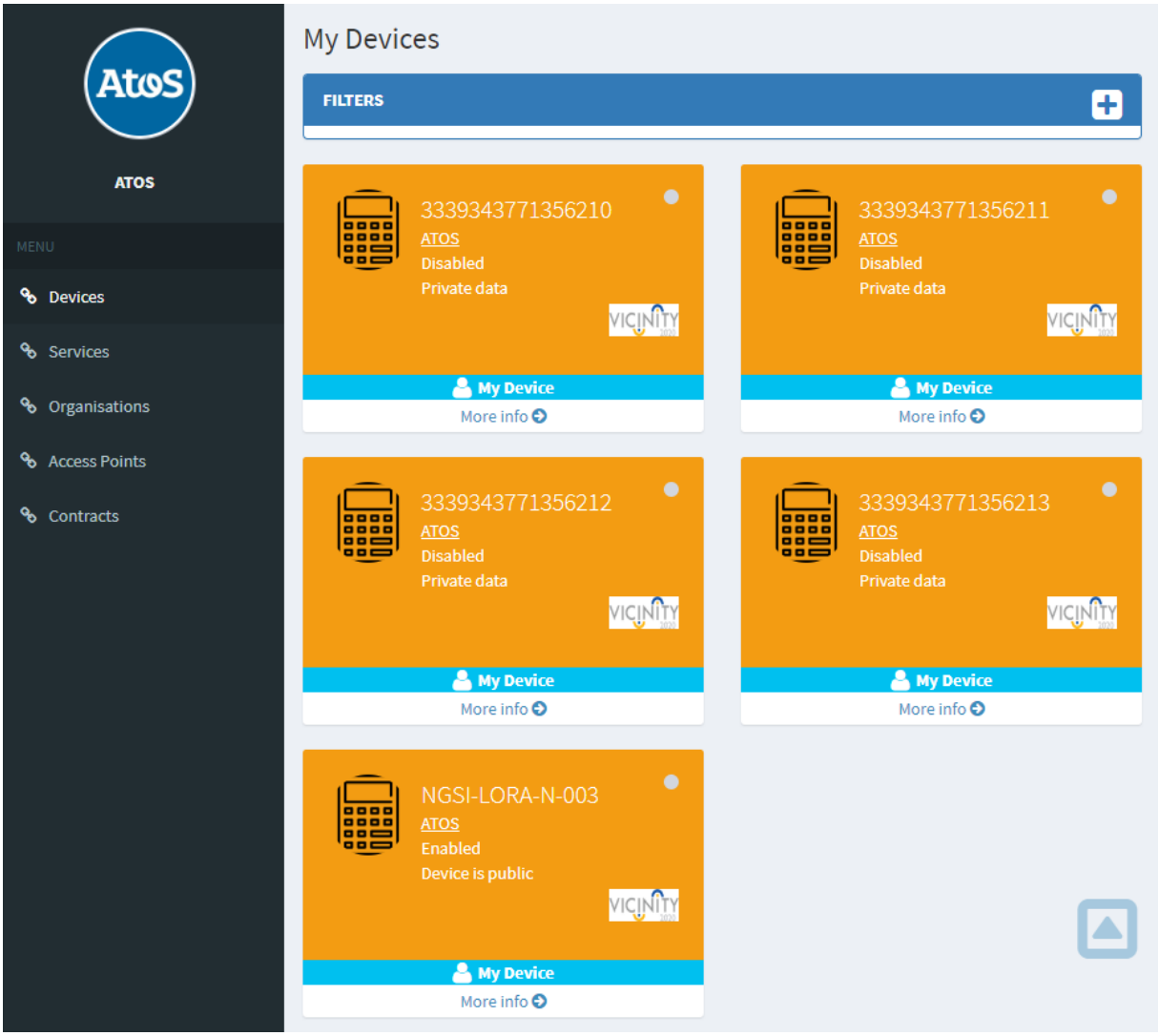



| ID | description | expires | status | subject |
|---|---|---|---|---|
| 5bf4917b72dbf26994221c9f | Subscription to VICINITY Adapter | 2018-11-27T22:58:01.00Z | active | {"entities":[{"idPattern":".*"}],"condition":{"attrs":[]}} |
| 5bf4919972dbf26994221ca0 | Subscription to VICINITY Adapter | 2018-11-27T22:58:32.00Z | active | {"entities":[{"idPattern":".*"}],"condition":{"attrs":[]}} |
| 5bf4919b72dbf26994221ca1 | Subscription to VICINITY Adapter | 2018-11-27T22:58:33.00Z | active | {"entities":[{"idPattern":".*"}],"condition":{"attrs":[]}} |

4. To illustrate the correct operation, below we show how the agent correctly extracts the information from the adapter




5. Last, but not least, the following screenshot proofs that the device has been successfully registered and shown on the Neighbourhood Manager user interface. NOTE: We have kept the registered devices from our other integration test.


ATOS
MENU
Devices
Services
Organisations
Access Points
Contracts
My Devices
FILTERS
3339343771356210
ATOS
Disabled
Private data
My Device
More info
3339343771356211
ATOS
Disabled
Private data
My Device
More info
3339343771356212
ATOS
Disabled
Private data
My Device
More info
3339343771356213
ATOS
Disabled
Private data
My Device
More info
NGSI-LORA-N-003
ATOS
Enabled
Device is public
My Device
More info

# Annex XIV – Internal point testing 11 - Integration of Omnet++ Network Simulator into VICINITY (UNIKL - GER)

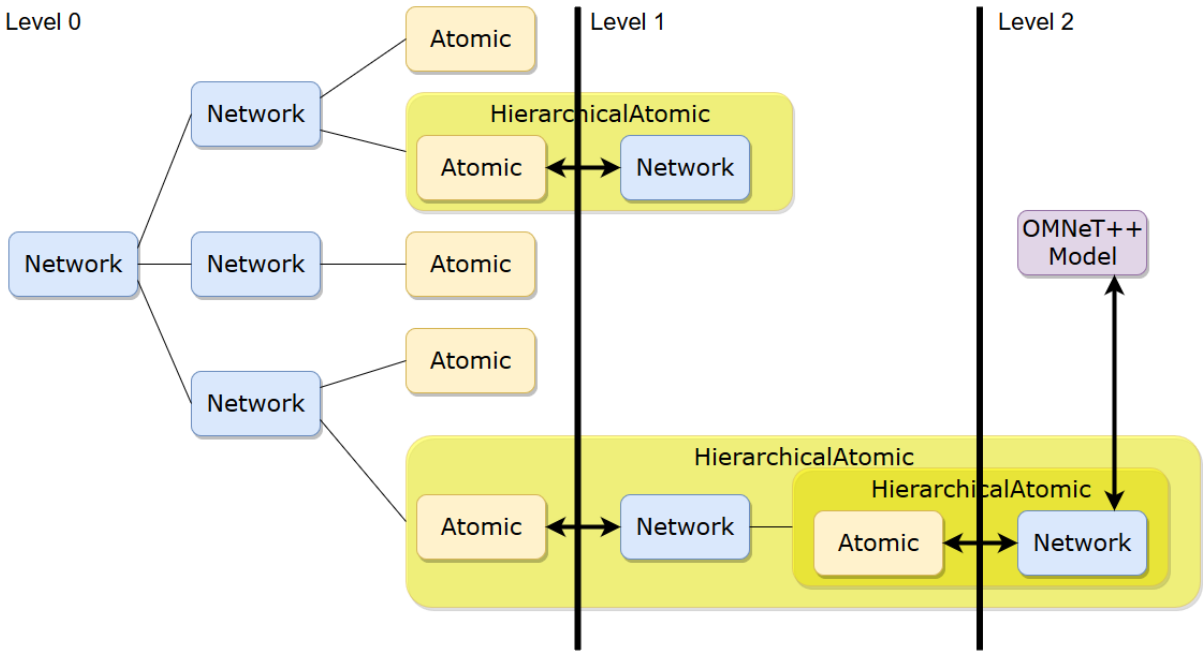



Figure XIV_1 Multi-level simulation framework based on Omnet++

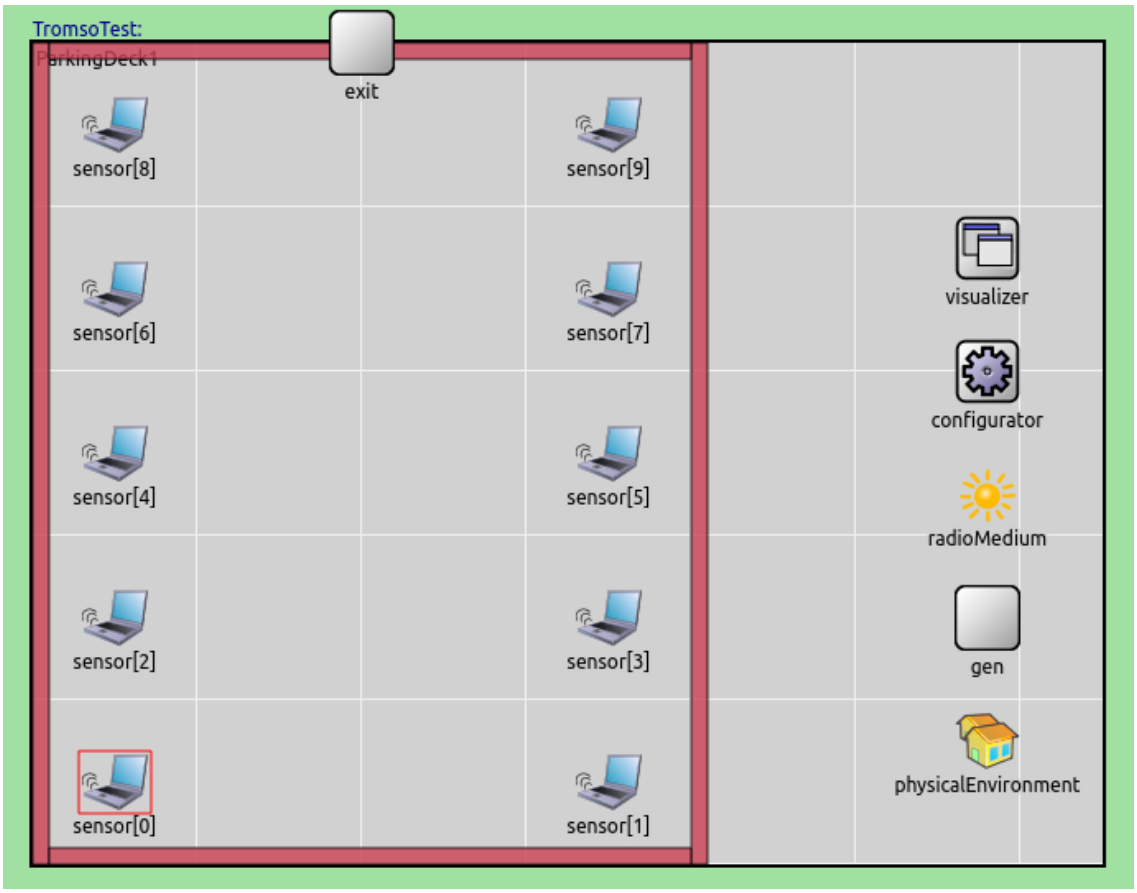



Figure XIV_2 Omnet++ GUI of Tromso Smart Parking Use Case Simulation

# Annex XV – Internal point testing 12 – Evaluation and Research on Homomorphic Encryption to be used for data aggregation for VAS (UNIKL - GER)

(Partially) Homomorphic encryption is a form of encryption that enables (limited) mathematical operations to be performed on the encrypted cyphertext, without the need to decrypt the information first. The generated results match the result of the operations, as if they had been performed on the plaintext.

Formally speaking, if E(x) denotes the encryption of the data x, this means (w.l.o.g.) e.g. for multiplications of data x,y,

E(x) * E(y) = E(x*y)

Ultimately, one does not even need access the plaintext to calculate e.g. the sum. This feature can be utilized to first encrypt personal data, like the energy consumption mentioned in the above example, using a homomorphic encryption scheme. Only the encrypted data is then transmitted to the receiving end, like the VICINITY node running the micro-service for anonymous data-aggregation. The micro-service can then sum up multiple encrypted data, even though it cannot read the individual plaintexts. After the calculation, the individual parties need to decrypt their part of the encrypted sum and now only the overall sum will be available as plaintext to the VAS.

UNIKL is currently looking into ways to further exploit the P2P nature of VICINITY to further optimize this process and enhance its performance. A first working prototype is already implemented and basis for further research and testing.

## Applications in VICINITY pilots and intended prove-of-concept

Discussions in the VICINITY consortium have shown that the micro-service described above is useful for some of the VICINITY pilots as of now. The team at ENERC has already given some scenarios where this very technique can be applied. The existing Prototype is a good starting point for further discussion. Next steps necessary to have it applied on the pilot site have been identified.

The use-case currently used for lab-testing is depicted in Figure 1: Multiple household appliances are equipped with smart energy meters. Their data is collected for internal evaluation (e.g. how many hours and energy, one has wasted watching TV). However, for external use, only the overall energy consumption is required, yet still making sure that each device is indeed reporting its consumption. To this end, data can be encrypted using a homomorphic encryption scheme and sent to the aggregation micro-service. While the receiving side will still be able to validate, that all appliances did send their data, none of the individual readings is exposed, as only the ciphertext is transmitted. The aggregation micro-service can calculate the sum, by adding up all ciphertexts. The resulting sum can then be decrypted, of course giving back the same result, as if it were calculated on the individual plaintexts.

### Integration into VICINITY components

Encryption/Decryption, in general, and homomorphic encryption schemes, in particular, are costly in terms of computational effort. However, the same applies to the encryption inside the P2P network already in place. This needs to be implemented on some gateway device on the edge, also running other VICINITY components like the VICINITY Gateway API or the VICINITY agent. The homomorphic encryption can be available as a kind of micro-service that is taking place on the VICINITY nodes and before data is sent out to the P2P network at all.

Figure 1 illustrates the potential Integration of such a microservice into the current VICINITY architecture: VICINITY Nodes interested in using Homomorphic encryption can add the respective microservice in addition to the other components such as the Agent and the Gateway-API. Instead of directly transmitting new e.g. sensor readings (marked in red), data is first encrypted with the homomorphic encryption scheme. The encrypted value is then sent, similar to regular payload, through the rest of the chain (agent, gtw-api) and through the VICINITY network to the receiving end. The same holds for all other measurements, which will encrypt their data the same way.

On the receiving end, instead of forwarding individual readings to the Value-added Service directly, the encrypted data is again sent to the homomorphic encryption microservice, which will aggregate the encrypted values (e.g. calculate the sum over all received data) and will only then decrypt and forward the aggregated data to the VAS. In both (encrypted and normal) cases, the VAS will end up with an aggregation of data, which it requires to operate. In contrast to the normal procedure, where the aggregation is calculated by the VAS itself, with homomorphic encryption the VAS will never know any individual, private data. This further enhances the “privacy by design” philosophy, which is fundamental to VICINITY from the very beginning.