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 D6.1

VICINITY integrated prototype

Work Package:	WP6 – VICINITY Framework Integration & Lab Testing
Task(s):	T6.1 – Integration of VICINITY Components
Lead Beneficiary:	TINYM
Due Date:	31/12/2018
Submission Date:	31/12/2018
Deliverable Status:	Final
Deliverable Type:	DEM
Dissemination Level:	Public
File Name:	VICINITY_D6.1_VICINITY_Integrated_Prototype_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
	Sundvor	Mariann	TINYM	Mariann@tiny-mesh.com
Co-authors (in alphabetic order)				
No	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1.	Abreu	Vítor	ENERC	v.abreu@enercoutim.eu
2.	Cimmino	Andrea	UPM	cimmino@fi.upm.es
3.	Gomez Fernandez	David	ATOS	david.gomez@atos.net
4.	Guan	Yajuan	AAU	ygu@et.aau.dk
5.	Heinz	Christopher	UNIKL	heinz@cs.uni-kl.de
6.	Koutli	Maria	CERTH	mkoutli@iti.gr
7.	Larsen	Ruben	TINYM	Ruben@tiny-mesh.com
8.	Nikolaj	Čolić	GRN	Nikolaj.Colic@gorenje.com
9.	Oravec	Viktor	BVR	viktor.oravec@bavenir.eu
10.	Samovich	Natalie	ENERC	n.samovich@enercoutim.eu
11.	Sveen	Flemming	HITS	flsveen@online.no
12.	Theologou	Natalia	CERTH	nataliath@iti.gr

Reviewers List

List of Reviewers (in alphabetic order)				
No	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1.	Čolić	Nicolaj	GRN	Nikolaj.Colic@gorenje.com
2.	Hovstø	Asbjørn	HITS	hovsto@online.no
3.	Zandes	Dimitris	GNOMON	d.zandes@gnomon.com.gr

Revision Control

Version	Date	Status	Modifications made by
0.1	10 October 2018	Initial draft	Ruben Larsen, TINYM
0.2	11 November 2018	Draft	Ruben Larsen, TINYM Mariann Sundvor, TINYM
0.3	29 November 2018	Quality Check	Ruben Larsen, TINYM Mariann Sundvor, TINYM
0.4	30 November 2018	Comments sent reviewers	Ruben Larsen, TINYM Mariann Sundvor, TINYM
0.5	13 December 2018	Sent reviewers with changes according to QAR	Ruben Larsen, TINYM Mariann Sundvor, TINYM
0.6	16 December2018	Input from UPM	Ruben Larsen, TINYM Mariann Sundvor, TINYM
0.7	18 December 2018	Final sent reviewers	Ruben Larsen, TINYM Mariann Sundvor, TINYM
0.9	19 December 2018	Review	Asbjørn Hovstø, HITS, QAR-team chair
1.0	31 December 2018	Submission to the EC	Ruben Larsen, TINYM Mariann Sundvor, TINYM

Executive Summary

This deliverable describes the integration tests of individual adapters, specifically the connection between the adapters and VICINITY. This involves running test cases and resolving any bugs that are encountered. The significance of this deliverable is establishing the use cases described in T5.2. The test cases described were instrumental in completing the use cases in a timely manner; multiple bugs were solved, resulting in improved data flow from the adapters and improved reliability in registering new cases with VICINITY.

The main points of the deliverable include making value added services available to VICINITY, ensuring the data from adapters reaches their intended targets, and closing bugs that impede the completion of the tasks from T5.2. All sites achieved functional adapters, enabling multiple VASs. Consequently, VASs were deployed on test servers, allowing for a more complete testing of features. Furthermore, the tools needed for P2P connections were enabled.

This deliverable covers the merging of aspects of multiple work packets, especially WP3, WP4 – "Client/Server Implementation" and WP1 - "Requirements Specification". Much of the effort in the document encompasses validating the previous work on the VICINITY client/server infrastructure and the marriage of individual nodes with VICINITY proper. The work done satisfies use cases, on user experience for setting up adapters for example, outlined in documents such as D1.5.

In layman's terms (in words that someone who is not an expert can understand), each adapter from the different sites needs to be able to be available through VICINITY and needs to communicate using VICINITY infrastructure. This implies running test cases for the different sorts of communication the adapters may use, be it node-to-node, discovering available adapters, or event subscriptions. Once adapters behave as expected when engaging with VICINITY, they can be featured in use cases, or used for further development by the pilot sites or Open Project partners.

When the adapters are all integrated with VICINITY, they will encompass the overarching integrated VICINITY prototype. That is to say, the prototype adapters will be able to provide their services over VICINITY once they are integrated. Here, "integrated" means connected to VICINITY, communicating with nodes through VICINITY, and sending and receiving legible data.

In conclusion, the highlights of the deliverable encompass the finishing touches of working adapters and node structures. Test cases were run, problems were identified and documented before being solved. The overall results include completed adapters, reliably registering events and VASs, and node to node communication.

The tests and validation of integration of components will be continuously updated each time a new version of Core Components is released, and eventually bugs will be triggered and results logged in the systems and methodology developed in this task.

Table of Contents

List of Tables

List of Figures

List of Definitions & Abbreviations

Abbreviation	Definition
EC	European Commission
EU	European Union
P2P	Peer to Peer
API	Application Program Interface
VAS	Value-Added Services
EV	Electric Vehicle
EMS	Energy Management System
GUI	Graphical User Interface
RMEMS	Residential microgrid energy management system
WP	Work Package
HIL	Hardware in the Loop
UI	User Interface
MG	Microgrid
GDPR	General Data Protection Regulation
TD	Thing Description

1. Introduction

The primary objective of this task is to establish communication between the individual adapters and VICINITY. This is achieved when coherent data from the adapters can reliably be accessed through VICINITY, establishing evidence that the adapter is integrated with VICINITY. An adapter takes device data and provides it to VICINITY in a universal context. Use cases from D5.2 need to be tied into the overarching architecture of VICINITY when they are proven to be functional.

A VICINITY client node consists of an adapter, agent and gateway, see WP4 – "Client/Server Implementation", section 4.1.

A node may communicate with other nodes in a P2P (Peer-to-Peer) fashion or with the VICINITY infrastructure. The neighbourhood manager is the principal user interface, and thus allows for the establishing of new nodes, see WP3 – "Client/Server Implementation", section 3.2. This document provides evidence of adhering to the client/server structure outlined in earlier work packages by exploring P2P communication, accessing adapters through VICINITY, and creating event-based relationships between nodes.

The structure of the components tested have remained consistent with uses cases described in earlier work packages. This will facilitate the deployment of the related Value-Added Services described in T5.2. Some use cases include nodes which consume data from other nodes; this is tested by having nodes subscribe to events generated by other nodes. This conforms to the structure described in WP1 – "Requirements Specification", section 1.3 by elaborating on VICINITY based on the requirements that are uncovered during the component integration process. To elaborate, WP1 as a whole explains what roles the adapters are meant to fill, and how they intend to fill these roles within the VICINITY architecture based on the different types of requirements adapters have (expanded on in section 1.1, 1.3, 1.4 and 1.5 specifically).

Once adapters have been confirmed to be integrated into VICINITY, and their services are functional, they will serve as a platform for further development. This is the integrated VICINITY prototype, where adapters are not only functional but communicating with VICINITY as well. This document is not intended to serve as a guide for developers or Open Call partners; rather as an addition to the demonstrators in order to describe the work that is done on the adapters featured in D6.1.

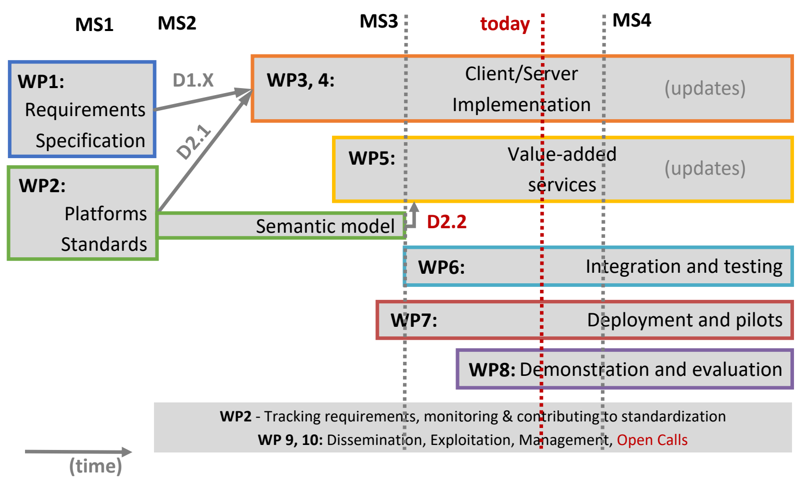

MS1
MS2
MS3
today
MS4
WP1: Requirements Specification
D1.X
D2.1
WP3, 4: Client/Server Implementation (updates)
WP5: Value-added services (updates)
WP2: Platforms Standards
Semantic model
D2.2
WP6: Integration and testing
WP7: Deployment and pilots
WP8: Demonstration and evaluation
WP2 - Tracking requirements, monitoring & contributing to standardization
WP 9, 10: Dissemination, Exploitation, Management, Open Calls
(time)

Figure 1 Work Package Architecture (updated figure from ATOS that cover WP6, see QAR review)

1.1. Context within VICINITY

This deliverable describes the final effort on establishing working versions of the Value-Added Services by solving the integration test cases. The essence of this work involves ensuring data flow from adapters and through VICINITY as well as setting up new VASs. The integration process serves to iron out the wrinkles and clearing out any bugs to enable the architecture described in WP3 – Client/Server Implementation, section 3.5 and WP4 – Client/Server Implementation, sections 4.2 and 4.4 The tests cover the communication between nodes and establishing client nodes described in these work packages.

The overall context of WP6 – "Integration and testing" is to validate previous efforts by ensuring the current VICINITY structure is compatible with the work done on individual use cases. This means VICINITY needs to be able to perform the tasks individual test cases require without bugs, to guarantee that VICINITY can handle the data managing needs of the pilot sites.

1.2. Objectives in Work Package 6 and Task 6.1

Work Package 6 – "Integration and testing" attempts to address the inadequacies of VICINITY by uncovering the needs of test cases and where VICINITY currently falls flat in regard to the functions it needs to perform. For example, if VICINITY is not equipped to handle the size of a particular device's Thing Description, the issue needs to be addressed for future cases. Thus, WP6 – "Integration and testing" serves to prove that VICINITY fulfils its requirements, and to address the most pressing areas of different use cases. This document also serves to ensure current iterations of use cases conform to the requirements outlined in Work Package 1 – "Requirements Specification", section 1.3 by seeing that the different parts of the VICINITY node architecture still satisfy the objectives of VICINITY after changes have been made.

The objective of Task 6.1, specifically, is the integration of the different VICINITY components. In short, this means an initial working prototype of the deliverables of T5.2 using the server/client infrastructure from earlier deliverables. This ensures that the connection between individual VICINITY nodes can handle the use cases.

Once integration tests have been satisfied, the individual use cases will be positioned to expand and ready to focus on details. WP6 – "Integration and testing" is therefore instrumental in ensuring the smooth deployment of uses cases (WP7 – "Deployment and pilots") and that individual adapters have the tools necessary to provide value-added services (WP 5 – "Value-added services"). This document describes the integration tests performed but not the lab test cases described in D6.2. The integration tests serve to solve common adapter to VICINITY communication issues, improving the user experience of establishing new adapters and devices, satisfying use cases outlined in D1.3 and D1.5. For reference:

T6.1 "Integration and testing", focuses 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", sections 1.1, 1.2, 1.3 etc., and the Value-Added Services 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 and on open project. Once issues are resolved, the subsequent changes manifest as 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 Cases 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 known and unknown IoT device types. Any limitations of the discovery process are identified and resolved as reported in D6.3.

1.3. Structure of 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: Methodology and scope of integration tests. This section outlines how the tests were envisioned and what their purpose are.

Chapter 3-9: This chapter explains the link between the use Cases, VAS and VICINITY Components Data from individual pilot sites. These sections include methods on individual tests and sites. The test results can be found in the annex.

Chapter 10: Conclusion. This section compiles the results of the test process; what was gained from the tests, what was learned during the process, how effective this sort of activity was.

2. Structure and Methodology of Individual Integration Tests

The integration structure and methodology focus on performing integration in parallel of integration of VICINITY core components with integration of the pilot sites, testing labs and Value-Added Service into the platform to ensure that VICINITY Interface view is fully covered.

2.1. Planning, Scope, and Purpose of Integration Tests

The layout and scope of the integration tests were decided based on pilot site definitions, functional, operational requirements and VICINITY architecture defined by WP1 – "Requirements Specification" and specification of the Value-Added Services defined by WP5 – "Value-added services implementation". The issues that were uncovered during the process are documented in VICINITY issues log available for all partner of the project[1], with the status and context of individual issues. Evidence of the progress on solving these issues and the cross-site cooperation can also be found on the open project website. Resolved issues resulted in the new versions of the software components, which were deployed in conjunction with necessary regression tests.

While the pilot sites and testing labs have varying requirements from VICINITY, tests were grouped into test areas by pilot sites and testing labs. Each test area verifies necessary features (such as P2P communication and connections to VICINITY) required from the VICINITY core components, value-added services and other peers. Each test, in essence, consists of verifying that local infrastructure communicates with the VICINITY platform as expected. The tests also serve to ensure that the exchanged control and/or user data can be processed by VICINITY adapters, VICINITY core components or Value-Added Services.

The general structure of the integration tests cover (1) how the test was conducted, (2) which data the adapter was expected to provide, (3) how the actual results compared to the expected results, and thereby (4) a summary of the issues that have been uncovered.

2.2. VICINITY Integrated Prototype

The VICINITY Integrated Prototype configuration version Release Candidate 1 (RC 1) consists of the set of software and hardware components summarized in the following table (Table 1) which were integrated together and verified by execution of the integration tests (Section **Error! Reference source not found.** - **Error! Reference source not found.**).

Table 1 Integrated VICNITY prototype Release Candidate 1 (RC 1)

Components	**Version**
VICINITY Cloud	
VICINITY Neighbourhood Manager	0.6.3.1
VICINITY Communication Server	0.6.3
VICINITY Gateway API Distributed Query Client	0.6.5
VICINITY Gateway API Services	0.6.4
VICINITY Semantic repository	0.6.1

[1] https://cpsproject.cs.uni-kl.de

Components	**Version**
Oslo Pilot Site (NO) – Buildings (TINYM)	
VICINITY Gateway API	0.6.3.1
VICINITY Agent	0.6.3.1
TINYM Adapter	0.0.1
IWMAC Adapter	0.0.1
Door Adapter	0.0.1
Adapter of VAS – Room usage	0.0.1
Adapter of VAS – Resource management	0.0.1
ATOS IoE Lab	
VICINITY Gateway API	0.6.3
VICINITY Agent	0.6.3
Cayenne adapter	0.5
FIWARE-NGSIv2 adapter	0.6
AAU Research Lab	
VICINITY Gateway API	0.6.2, 0.6.3
VICINITY Agent	0.6.2, 0.6.3
Adapter of VAS-Vacant parking slot and charging price notifications	0.0.1
UNIKL Research Lab	
VICINITY Gateway API	0.6.3.2 (draft)
VICINITY Agent	0.6.3.1
OpenHab Adapter	0.6.2
Pilea-Hortiatis (GR) – eHealth & Assisted Living	
VICINITY Gateway API	0.6.3
VICINITY Agent	0.6.3
VICINITY Neighbourhood Manager API	0.6.3
VAS 3.1.1, 3.2.1 Storage and GDPR	0.0.1
VAS 3.1.2 Individual Statistics	0.0.1
VAS 3.1.3 Abnormal Detection	0.0.1
VAS 3.2.2 Urban Marathon	0.0.1

Components	**Version**
VAS 3.2.3 Aggregated Statistics	0.0.1
Adapter for building sensors based on IoTivity Platform	0.0.1
Adapter for medical devices based on NodeRed Platform	0.0.1
Adapter for medical devices integrated in eHealthPass VICINITY extended mobile App	0.0.1
Gorenje Testing Lab	
VICINITY Gateway API	0.6.3.1
VICINITY Agent	0.6.3.1
GRN VICINITY adapter	1.0.0
Tromsø (NO) – Neighbourhood Smart Parking Assisted Living ecosystem	
VICINITY Gateway API	0.6.3
VICINITY Agent	0.6.3
Adapter PNI PlacePod	0.0.1
Adapter IKEA smartlight	0.0.1
GRN VICINITY adapter	1.0.0
Martim Longo (PO) – Neighbourhood GRID ecosystem (ENERC)	
VICINITY Gateway API	0.6.3
VICINITY Agent	0.6.3
ENERCOUTIM Device Adapters Pack for Energomonitor, SERINUS and dataTaker	0.7.0
ENERCOUTIM Service Adapter Pack for Dynamic Building Audit, Smart School, UV for Citizens and PV Plant Operations Management	0.7.0
ENERCOUTIM Service Monitor Pack for Dynamic Building Audit, Smart School, UV for Citizens and PV Plant Operations Management	0.7.0

2.3. Integration testing coverage

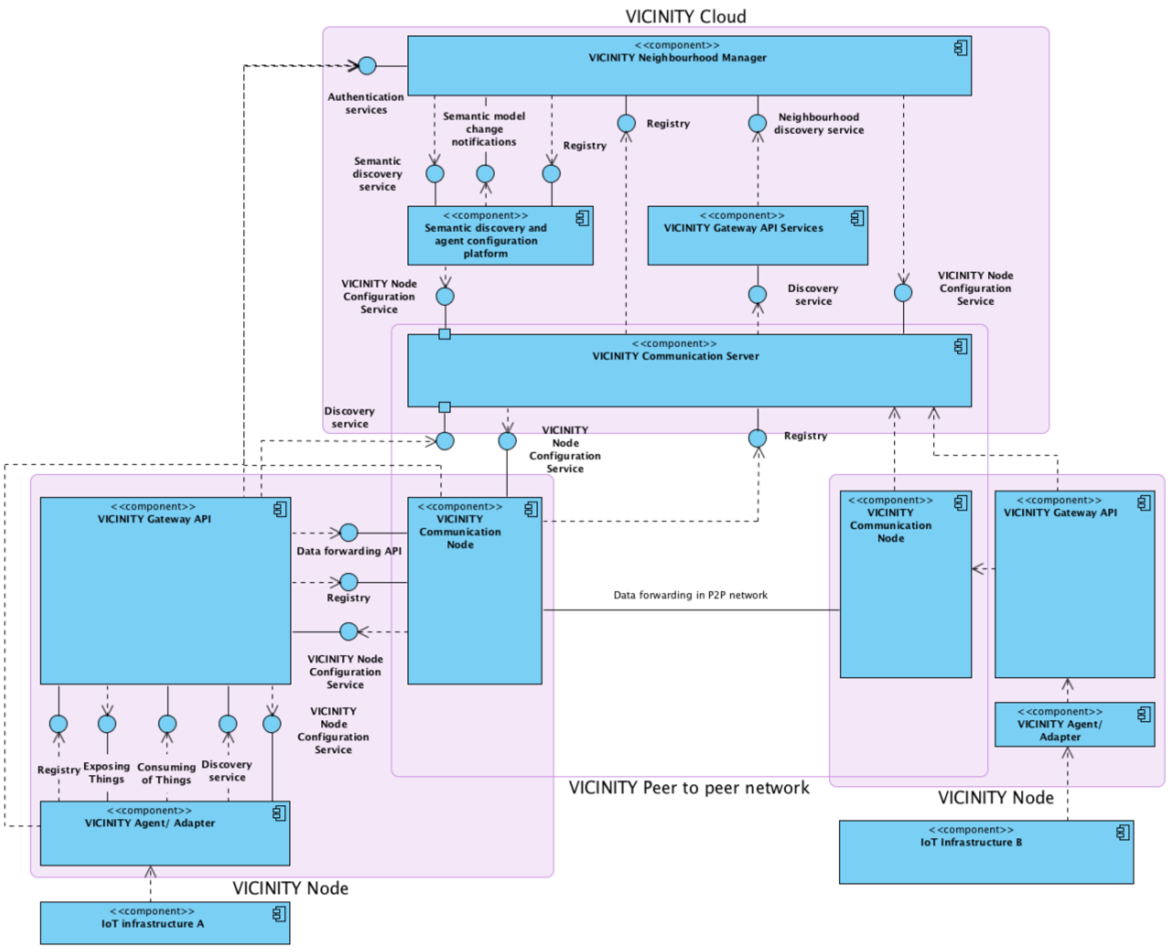

VICINITY Cloud
<<component>> VICINITY Neighbourhood Manager
Authentication services
Semantic model change notifications
Registry
Neighbourhood discovery service
Semantic discovery service
Registry
<<component>> Semantic discovery and agent configuration platform
<<component>> VICINITY Gateway API Services
VICINITY Node Configuration Service
Discovery service
VICINITY Node Configuration Service
<<component>> VICINITY Communication Server
Discovery service
VICINITY Node Configuration Service
Registry
<<component>> VICINITY Gateway API
Data forwarding API
Registry
VICINITY Node Configuration Service
<<component>> VICINITY Communication Node
VICINITY Node Configuration Service
Registry
Exposing Things
Consuming of Things
Discovery service
<<component>> VICINITY Agent/ Adapter
VICINITY Node
<<component>> IoT infrastructure A
Data forwarding in P2P network
VICINITY Peer to peer network
<<component>> VICINITY Communication Node
<<component>> VICINITY Gateway API
<<component>> VICINITY Agent/ Adapter
VICINITY Node
<<component>> IoT Infrastructure B

Figure 2 VICINITY Architecture integration view

Based on the VICINITY Platform interface view[2] (**Error! Reference source not found.**) the following i nterfaces were integrated and directly and/or indirectly tested (Table 2) by test campaigns (Section **Error! Reference source not found.** - **Error! Reference source not found.**).

Table 2 VICINITY Interface integration test coverage

Name of Interface	**Used by**	**Covered by test**
VICINITY Neighbourhood Manager		
Authentication service	VICINITY Communication Node, VICINITY Gateway API, VICINITY Agent	TC010_IT020, TC020_IT010, TC030_IT010, TC040_IT010, TC040_IT020, TC050_IT010, TC060_IT010, TC070_IT010

[2] VICINITY D1.6 VICINITY Architectural Design

Name of Interface	Used by	Covered by test
Neighbourhood discovery service	VICINITY Gateway API Services, VICINITY Gateway API Distributed Query Client	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
Registry service	VICINITY Communication Server	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
Semantic model change notifications	Semantic Platform	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
Semantic discovery and dynamic configuration agent platform		
Semantic discovery service	VICINITY Neighbourhood Manager	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
Registry Service	VICINITY Neighbourhood Manager	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
VICINITY Gateway API Services		
VICINITY Communication server	Request/ Response	TC040_IT040, TC040_IT050
VICINITY Communication Server		
Discovery service	VICINITY Gateway API	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100

Name of Interface	Used by	Covered by test
VICINITY Node Configuration Service	VICINITY Neighbourhood Manager	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
Registry Service	VICINITY Communication Node	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
VICINITY Communication Node		
Data forwarding API	VICINITY Gateway API	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
VICINITY Node Configuration Service	VICINITY Communication Server	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
Registry Service	VICINITY Communication Server	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
Data forwarding P2P	VICINITY Communication Node	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
VICINITY Gateway API		

Name of Interface	**Used by**	**Covered by test**
Discovery and query service	VICINITY Agent/Adapter	TC040_IT040, TC040_IT050
Consuming service	**VICINITY Agent/Adapter**	
Consuming service	VICINITY Agent/Adapter	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
VICINITY Node Configuration Service	VICINITY Agent/Adapter	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
Registry Service	VICINITY Agent/Adapter	TC010_IT010, TC010_IT030, TC020_IT010, TC030_IT020, TC040_IT010, TC050_IT020, TC050_IT040, TC050_IT060, TC050_IT100
VICINITY Agent/ Adapter		
VICINITY Node Configuration Service	VICINITY Gateway API	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030
Exposing service	VICINITY Gateway API	TC010_IT020, TC010_IT030, TC030_IT020, TC030_IT030, TC040_IT030, TC050_IT010, TC050_IT070, TC050_IT080, TC050_IT090, TC050_IT120, TC060_IT020, TC060_IT030, TC070_IT020, TC070_IT030

3. TC010: Oslo Pilot Site (NO) – Buildings (TINYM)

3.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

Use Case 1a.1 – Predictive operations

In this use case, TINYM prepares sensors to judge room usage, which are connected to VICINITY. When VICINITY registers a given number of visits to a room, a notification is sent to the interested parties. VICINITY logs the room usage history allowing this data to be analysed by staff to improve their custodial experience (this is the VAS) or used further by VICINITY in aggregate. The sensors establish events which can be subscribed to, essentially allowing VICINITY users easy access to room usage data which may be pertinent to them. The VAS consists of room sensors which need to be established and communicate its events with the server or other nodes.

Use Case 1a.2 – Resource management

In this use case, TINYM receives utility consumption data from local partners (IWMAC) through an adapter on their network attached to VICINITY. Smart devices that can be accessed through VICINITY can then be manipulated to smooth out or lower the overall resource consumption curve for a location/site; in the case of energy tariffs, for example, this reduces costs by lowering resource consumption peaks by automatically responding to increases in utility usage. These same tools (provided by the VASs) can also be used to diagnose anomalies (like water leaks), and predict resource expenditure once baseline data has been gathered. The VAS includes setting up, communicating with, and regulating individual smart devices through VICINITY.

In the following tests Use Case and VAS is proved by establishing reliably new nodes, ensuring node to node communication, and establishing subscription relationships with endpoints.

3.2. TC010_IT010: Integration Test 1 – Establishing a new VICINITY Node

3.2.1. Integration Test Steps

On the website for neighborhood manager perform following steps.

1. In the “Gateways” section create new gateway,
    a. Choose VICINITY agent
    b. Give it a name and set a password.
2. Use password and agent id received after completing step 1 to setup adapter/agent/gateway combo.
3. Prepare agent
    a. Edit config to reflect your set up
        i. Correct URLs for: adapter and gateway
        ii. Correct password and id of the agent
4. Prepare gateway
    a. Edit config to set the path where the logs will be saved
    b. Correct URL for the agent
5. Make sure adapter has correct thing description.
6. Start gateway
7. Start adapter
8. Start agent

3.2.2. Expected Results

Agent starts up without failing and successfully registers devices from the Thing Description in VICINITY. Devices registered can be viewed in the Neighbourhood Manager under "Devices" menu item.

3.2.3. Actual Results

The device is visible in the Neighbourhood Manager, with the relevant TD, without errors.

3.2.4. Problems Identified

None

3.3. TC010_IT020: Integration Test 2 – Node to Node Communication

3.3.1. Integration Test Steps

Assumptions:

1. One VICINITY node representing a service that will consume data from devices exposed by another node
2. One VICINITY node that exposes devices to VICINITY
3. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.

1. Start both nodes
2. The Service node should now make a request to its agent for data from device belonging to another node. This can be emulated using curl or similar program to send a request.
   a. Request: GET
      http://<agent_URL>:<agent_port>/agent/remote/objects/<oid>/properties/<pid>
   b. Required headers:
      i. *infrastructure-id* which is the internal id of the object requesting data from device
      ii. *adapter-id* which is a unique adapter id specified in the agent configuration and TD.

3.3.2. Expected Results

Response should contain data from the device in the format specified in its thing description. The status code of the response should be 200.

3.3.3. Actual Results

Coherent data arrives reliably from the node.

3.3.4. Problems Identified

None

3.4. TC010_IT030: Integration Test 3 – Events Publishing and Subscribing to Events

3.4.1. Integration Test Steps

Assumptions:

4. One VICINITY node representing a service that will consume data from devices exposed by another node
5. One VICINITY node that exposes devices to VICINITY

6. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.
7. The Node receiving events implements an endpoint for receiving events – PUT /objects/<oid>/events/<eid>

1. Make sure the Node publishing the event has the following:
   a. Events are described in the thing description
   b. Channel is described in the agent configuration (https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/AGENT.md#configuration)
2. Make sure the Node subscribing to events has the following
   a. Subscription is defined in the agent configuration
   b. Endpoint for receiving events is implemented (see assumptions)
3. Start both Nodes
4. Trigger event so it will be sent to subscribers.

3.4.2. Expected Results

Once triggered, an event is sent to all subscribers, the publisher of the event gets a response with the success message and information about to how many subscribers the event was sent.

The subscribing node should be able to receive the events, the data of the event must conform to the definition in the TD of the publishing node.

3.4.3. Actual Results

The event is triggered for all subscribers, however there was an issue where events could not be subscribed to, despite a running adapter.

3.4.4. Problems Identified

Events not arriving in the log; events could not be subscribed to even though the adapter was running. An issue with the gateway was resolved and events behave as expected.

4. TC020: ATOS

4.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

Regarding the integration of the Internet of Everything Lab provided by ATOS, unlike the rest of pilots/labs presented throughout this document, there is no implementation of any VAS during the process. This is due to the fact ATOS had no effort on this in WP5. In turn, they focused on the development of two different adapters that, aside from being used for their own assets, might be also leveraged by others, as they are based on well-known standards/interfaces for IoT infrastructures: Cayenne-LPP[3] and FIWARE NGSIv2[4]. Namely, the reader might refer to the following links to reach the VICINITY adapter for Cayenne (https://github.com/VICINITYh2020/VICINITY-adapter-cayenne) and FIWARE NGSIv2 (https://github.com/VICINITYh2020/VICINITY-adapter-ngsiv2).

Focusing on the lab itself, a thorough description of the assets can be found in D6.2 <REF to D6.2>. In a nutshell, we have used:

- a handful of (different) LoRaWAN nodes (class A)
- a LoRaWAN gateway (two different options), which runs an open source-based LoRaWAN Network server based on The Things Network (https://www.thethingsnetwork.org/) or LoRaServer (https://www.loraserver.io/)
- a FIWARE Orion Context Broker plus a FIWARE LoRaWAN IoT Agent[5] (for the FIWARE NGSI v2 integration)

In terms of the components that have been assessed during this integration phase, they have directly interacted with three of them, defining aside the explicit version used during the tests: Neighbourhood Manager, Gateway API (v0.6.3) and Agent (v0.6.3).

In summary, up to five different integration tests have been carried out, covering from the very first contact with the Neighbourhood Manager/Agent/Adapter to the publication/reception of events produced in the different nodes.

4.2. TC020_IT010: Integration Test 1 – Device Registration

4.2.1. Integration Test Steps

This first phase of the integration embraces all the preliminaries that are to be done to set the environment ready. In particular, these are the steps that any user must carry out if they want to register a device for his/her first time

On the website for neighbourhood manager perform following steps.

1. In the “Gateways” section create new gateway,
    a. Choose VICINITY agent
    b. Give it a name and set a password.
2. Use password and agent id received after completing step 1 to setup adapter/agent/gateway combo.
3. Prepare agent

[3] https://mydevices.com/cayenne/docs/lora/
[4] https://orioncontextbroker.docs.apiary.io/#
[5] https://github.com/FIWARE-GEs/iot-agent.LoRaWAN

   a. Edit config to reflect your set up
      i. Correct URLs for: adapter and gateway
      ii. Correct password and id of the agent
4. Prepare gateway
   a. Edit config to set the path where the logs will be saved
   b. Correct URL for the agent
5. Make sure adapter has correct thing description.
6. Start gateway
7. Start adapter
8. Start agent

4.2.2. Expected Results

Everything worked as expected. With the adapter configured in a "passive discovery" mode (within the Agent config file), it catches the devices exposed by the adapter and forwards the information to the Gateway API. With this, the devices are visible from the Neighbourhood Manager User Interface (UI).

4.2.3. Actual Results

Bugs encountered and addressed.

4.2.4. Problems Identified

When it came to register an organization (onto the development server, though), the automatic validation mail redirected to the production portal. Behind the scenes, the user/organization was registered in the wrong server, thus the access was not possible. The bug was reported and the responsible stuff reacted almost instantaneously, thus its real effect was negligible.

4.3. TC020_IT020: Integration Test 2 – Validate Adapter (via Thing Description validation)

4.3.1. Integration Test Steps

1. Before proceeding to register a device, VICINITY offers a feature of high utility: the validation of the Things Descriptions. Technically speaking, we can rely on this tool to assess that the outcomes from adapters are compliant with the last version of TDs. To do so, what we have to do is to send a POST message to the below endpoint, including the TD as a JSON object in the HTTP header's body.
   https://development.VICINITY.bavenir.eu:3000/api/repository/validate

2. Under the scenes, the service internally checks that the non-semantic representation of the TD matches the data models defined. As a result, it returns a JSON object with the result of the validation. As an illustrative example, we show below how this response looks like.

```
{
    "data": [
        {
            "oid": "cayenne1",
            "message": "Thing is valid"
        },
    ],
    "status": "success"
}
```

3. From the output of this service we can get an idea of how “accurate” is our adapter.

4.3.2. Expected Results

As shown above, a “success” value in the “status” field means that the TD is compliant with VICINITY’s.

4.3.3. Actual Results

Success

4.3.4. Problems Identified

None

4.4. TC020_IT040: Integration Test 4 – Node-to-Node communication (request/response)

4.4.1. Integration Test Steps

Assumptions:

8. One VICINITY node representing a service that will consume data from devices exposed by another node
9. One VICINITY node that exposes devices to the VICINITY
10. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.
11. Node receiving events implements endpoint for receiving events – PUT /objects/<oid>/events/<eid>

4.5. TC020_IT050: Integration Test 5 – Node-to-Node communication (pub/sub)

4.5.1. Integration Test Steps

Assumptions:

12. One VICINITY node representing a service that will consume data from devices exposed by another node
13. One VICINITY node that exposes devices to the VICINITY
14. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.
15. Node receiving events implements endpoint for receiving events – PUT /objects/<oid>/events/<eid>

5. TC030: AAU

5.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

The use case implemented at the AAU IoT-Microgrid Lab is designed as an integrated residential microgrid energy management system (EMS) and smart parking service. A PV/wind/battery hybrid residential microgrid is emulated with a real-time dSPACE-based experimental platform. Three PlacePod parking sensors are connected to VICINITY. The parking slot status is integrated with the residential microgrid EMS. A 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 implemented EMS-Parking VAS provides users with data about the numbers of vacant parking slot and the real-time charging price for EVs in order to optimize energy and parking slot usages and to reduce end-users' bills.

EMS-Parking VAS adapter, PlacePod parking sensor adapter, 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 VAS subscribes the event published by the parking sensors and publishes events to an end-user thus testing the publish/subscribe performance of VICINITY.

5.2. TC030_IT010: Integration Test 1 - Use NM UI to establish new VICINITY Nodes and establish friendships

5.2.1. Integration Test Steps

According to the requirements: https://documenter.getpostman.com/view/2413103/VICINITY-neighbourhood-manager-api/RVg29U8c#1c6bf2bf-d9fb-dd32-511a-1ae0b3ac9440, perform the following steps:

1. POST Fast Organisation Registration
2. POST Authenticate
3. PUT User update (visibility, roles)
4. POST Agent create
5. Repeat steps 1-4 for another user/organization
6. Make sure adapter has correct thing description.
7. Start gateway
8. Start adapter
9. Start agent
10. POST Friendship request from Organization AAU to Organization TE
11. Organization B to call GET Friendship feeds
12. Organization B to call PUT Friendship update to accept friendship

5.2.2. Expected Results

Agent starts up without failing and successfully registers devices from TD in the VICINITY. Devices and services registered can be viewed in the Neighbourhood Manager under “Devices” and “Services” menu items separately.

5.2.3. Actual Results

5.2.4. Problems Identified

Bug #46: Change of status trigger exception.

Every time a change of status in a parking sensor occurs, the field ParkingSensor[”sentralTime”] is out of range triggering an exception. Changed the type from Integer to Long and the issue has been solved.

5.3. TC030_IT020: Integration Test 2 - Service Node to device Node communication

5.3.1. Integration Test Steps

Assumptions:

16. One VICINITY node representing a service that will consume data from devices exposed by another node
17. One VICINITY node that exposes devices to the VICINITY
18. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.

1. Start both nodes
2. Service node should now make a request to it’s agent for data from device belonging to another node.
   a. Request: GET
      http://<agent_URL>:<agent_port>/agent/remote/objects/<oid>/properties/<pid>
   b. Required headers:
      i. *infrastructure-id* which is the internal id of the object requesting data from device
      ii. *adapter-id* which is a unique adapter id specified in the agent configuration and TD.

5.3.2. Expected Results

Response should contain data from the device in the format specified in its thing description. The status code of the response should be 200.

5.4. TC030_IT030: Integration Test 3 - Events: publishing and subscribing to events

5.4.1. Integration Test Steps

Assumptions:

19. AAU Client node representing a service that will consume data from devices exposed by another node
20. AAU Server node that exposes Energy Management System of Microgrid to the VICINITY
21. The contract between AAU Client node and AAU Server node belong to the different organisation is established.
22. AAU client node is described to the event of EMS exposed by AAU server node.
23. AAU server node is described to the event of PNI exposed by PNI adapter.
24. AAU client Node receiving events by receiving PUT request –/objects/<oid>/events/<eid>

Steps:

1. Start Labview Interface for EMS
2. Start simulink simulation for microgrid
3. Start gateway, agent and PNI client adapter

4. Start gateway, agent and AAU server adapter
5. Start gateway, agent and AAU client adapter
6. The AAU server node will receive event published by PNI adapter automatically and sends the status of parking sensor to microgrid EMS
7. EMS send the number of free parking slot and EV charging price to AAU server node adapter and AAU server node adapter publish data to subscribers automatically

5.4.2. Expected Results

Once event is sent to subscribers of AAU client node, the publisher of the event gets a response with the success message and information about to how many subscribers event was sent.

The AAU client node should be able to receive the events, the data of the event contains the number of free parking slot, EV charging prince and time-stamp.

5.4.3. Actual Results

Information is sent.

5.4.4. Problems Identified

None.

6. TC040: UNIKL

6.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

The Use Case implemented at the UNIKL Test-Lab is designed as smart lighting control service. The existing ceiling lights have been modified and made controllable via WiFi. Combined with events received from motion sensors, the lights will "follow" the user. In particular, this means only the necessary lights will turn on, when a user walks along the hallway.

In this Use Case a VAS is implemented, testing all interaction patterns provided by VICINITY. Active and Passive Discovery of the Agent is used for the Lights Adapter and the VAS itself respectively. The VAS subscribes to motion events emitted by the sensors and thus testing the publish/subscribe pattern in VICINITY. Finally, the VAS controls the lights according to the received events directly in a request/response pattern.
In the current setup, only two of the ceiling lights are equipped with sensors and are connected to VICINITY. Their oid is hard-coded into our VAS. Potentially, more sensors will follow and ideally, we do not want to touch our code every time again. Thankfully, VICINITY offers semantic search via its Gateway API Services and allows a distributed client search. This will further be tested and evaluated.

6.2. TC040_IT010: Integration Test 1 - Device registration using NM and available VICINITY Components

6.2.1. Integration Test Steps

Assumptions:

Necessary VICINITY Components, namely VICINITY Gateway-API and VICINITY Agent are available open source on Github. They are set up and configured according to their documentation on each Gateway device in the Lab. Running Gateways/Adapters/Frameworks are:

- Eclipse Kura
- OpenHAB
- Adapter for MQTT enabled devices (with hard-coded MQTT-topics)

On the website for neighbourhood manager perform following steps.

1. In the "Gateways" section create new gateway,
    a. Choose VICINITY agent
    b. Give it a name and set a password.
2. Use password and agent id received after completing step 1 to setup adapter/agent/gateway combo on each gateway.
3. Prepare agent
    a. Edit config to reflect your set up
        i. Correct URLs for: adapter and gateway
        ii. Correct password and id of the agent
4. Prepare gateway
    a. Edit config to set the path where the logs will be saved
    b. Correct URL for the agent
5. Make sure adapter has correct thing description.
6. Start gateway/adapter
7. Start agent

6.2.2. Expected Results

After successful registration, all devices and the value-added service will show up in the Neighbourhood Manager.

6.2.3. Actual Results

Everything worked as expected. With the adapter configured in a “passive discovery” mode (within the Agent config file), it fetches the devices exposed by the adapter and forwards the information to the Gateway API. In “active discovery” mode, the VAS calls the agent, once startup sequence is complete. The VAS registers itself to the agent, who is forwarding this registration to the NM. Finally, the VAS subscribes to the event channels dynamically. With this, the devices are visible from the Neighbourhood Manager User Interface (UI).

6.2.4. Problems Identified

None

6.3. TC040_IT020: Integration Test 2 - Node-to-Node communication (request/response)

6.3.1. Integration Test Steps

Assumptions:

25. Multiple VICINITY nodes that expose devices to the VICINITY
26. VICINITY Gateway-API set up on Desktop Machine
27. Nodes belong to the same organisation

1. Start up each VICINITY node. Each Component is started in correct order (agent goes last)
2. Devices show up on NM
3. Get oid of device, which should be tested via NM
4. Get available properties of device, which should be tested via NM
5. Use local set up of Gateway-API to control devices/read sensor values, providing valid credentials of a dummy device (also registered and shown in NM)

6.3.2. Expected Results

Devices should react to requests given (e.g. light up, dim down, change color, give sensor values).

6.3.3. Actual Results

Correct response from registered devices

6.3.4. Problems Identified

None

6.4. TC040_IT030: Integration Test 3 - Node-to-Node communication (pub/sub)

6.4.1. Integration Test Steps

Assumptions:

28. One VICINITY node representing a service that will consume data from devices exposed by another node
29. One VICINITY node that exposes devices to the VICINITY
30. Nodes belong to the same organisation

31. Node receiving events implements endpoint for receiving events – PUT /objects/<oid>/events/<eid>

Steps:

1. Start up VICINITY node, that exposes devices to the VICINITY
2. Start up VICINITY node, that is running the VAS
3. VAS subscribes to two events, by different motion sensors
4. VAS receives motion events and, depending on which node triggered the event, will turn on the lights in that particular sector, where motion was detected

6.4.2. Expected Results

Lights will go on, when there is motion in a particular sector of the room. VAS is able to determine the source of the event received by the oid of the sender

6.4.3. Actual Results

VAS is not able to determine which sensor is reporting motion and cannot correctly control the respective lights.

Running 24/7, objects get logged out occasionally?

6.4.4. Problems Identified

No information available, as to WHO was emitting the event. Two motion sensors (motion_1 and motion_2) are publishing their event (“motion”).

The subscriber will implement its receiving endpoint (/objects/<oid of subscriber>/events/motion)

Events from both sensors end up at the same receiving endpoint (as both events are called “motion”).

VAS is not able to distinguish, where the event was emitted.

1. Documentation states, that the oid of the sender, should be part of the payload. This is not the case. Bug reported on OpenProject (ticket #33)

This issue has been resolved within the versions 0.6.3 of Agent and GTW-API!

Having the service online 24/7 and being subscribed to a couple of sensors, these connections is occasionally dropped sometimes. Objects are reported as not being online, which is only resolved by manually logging them in again e.g. by restarting the agent (ticket #47)

This issue seems to be resolved within versions 0.6.3.1 of Agent and GTW-API! This will further be tested

6.5. TC040_IT040: Integration Test 4 – Using Gateway API Services to search for devices and their current real-time data

6.5.1. Integration Test Steps

Assumptions:

1. (at least) one VICINITY node exposes devices, which have their output data annotated for semantic search
2. One VICINITY node used for distributed client search

3. Nodes belong to the same neighborhood

Steps:

1. Startup VICINITY node, that exposes devices to the VICINITY
2. Startup VICINITY Gateway API on receiving side
3. Log in to VICINITY on receiving side
4. Send search Query to Sparql endpoint of Gateway API

6.5.2. Expected Results

A list of registered devices in neighborhood is received. Where properly annotated, also real-time information of the devices is shown.

6.5.3. Actual Results

List of registered devices is received. Real-time data is not shown in version 0.6.3.1 of the Gateway API.

This issue is resolved within version 0.6.3.2 of the Gateway API!

6.5.4. Problems Identified

Real-time values of the found devices is queried and retrieved (according to logfiles of Gateway API), but is lost. In the final result (response to the query) no information is shown, when using version 0.6.3.1 of the Gateway API.

With the latest draft version 0.6.3.2, this issue was resolved!

6.6. TC040_IT050: Integration Test 5 – Using Gateway API Services to search for lightbulbs with attached motion sensors

6.6.1. Integration Test Steps

Assumptions:

1. (at least) one VICINITY node exposes Lightbulb devices, which have a motion detection property as well
2. One VICINITY node used for distributed client search
3. Nodes belong to the same neighborhood

Steps:

1. Startup VICINITY node, that exposes devices to the VICINITY
2. Startup VICINITY Gateway API on receiving side
3. Log in to VICINITY on receiving side
4. Send search Query to Sparql endpoint of Gateway API

6.6.2. Expected Results

List of Lightbulbs with motion sensors is received. No other devices (not of interest for the given query) are shown.

6.6.3. Actual Results

List of devices, only showing the intended lightbulbs is returned.

6.6.4. Problems Identified

None

7. TC050: Pilea-Hortiatis (GR) – eHealth & Assisted Living (CERTH – GNOMON – MPH)

7.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

MPH Pilot case consists of five VASs, as defined in D5.1, the equipment and infrastructure which is defined in D7.1 and their adapters, defined in D4.1. It is divided in two sub-use cases, one regarding assisted living of elder citizens and one regarding the promotion of healthy lifestyle to middle-aged citizens. This is a large-scale use case which involves the integration of a big number of different infrastructures (~100) to VICINITY Platform and thus has specific testing requirements.

Due to the big number of involved infrastructure owners (citizens), the procedure of the infrastructure registration to VICINITY needed to be automated. For this purpose, the Neighbourhood Manager (NM) API endpoints were used in order to automate this procedure through the mobile UI or Raspberry Pi of each citizen, and thus GNOMON with the help of CERTH were the main testers of NM API. Moreover, the automatic configuration of the multi-agent needed to be tested for this procedure. Two other important aspects of this pilot were the communication between the devices and the GDPR and Storage VAS and the communication between the aforementioned VAS and the other VASs, which were both tested extensively by CERTH and GNOMON partners. During this testing period a lot of the core components (gtw, agent) functionalities were also tested, as part of the integration.

Twelve Integration Tests were documented as the most important, although many other smaller tests were conducted. For each Integration Test the steps and the expected results were specified. During the test procedure a number of bugs and needed features was identified and documented. The most recent results of the tests can be seen in Table 5.

7.2. TC050_IT010: Integration Test 1 - Use NM API to create organization, agent, friendships

7.2.1. Integration Test Steps

Perform the following REST calls from any client. The inputs and endpoints required for the request can be found at: https://documenter.getpostman.com/view/2413103/VICINITY-neighbourhood-manager-api/RVg29U8c#1c6bf2bf-d9fb-dd32-511a-1ae0b3ac9440

1. POST Fast Organisation Registration
2. POST Authenticate
3. PUT User update (visibility, roles)
4. POST Agent create
5. Repeat steps 1-4 for another user/organization
6. POST Friendship request from Organization A to Organization B
7. Organization B to call GET Friendship feeds
8. Organization B to call PUT Friendship update to accept friendship

7.2.2. Expected Results

GET Organisations friends should return the friend organization

7.3. TC050_IT020: Integration Test 2 - Auto-registration of devices and services

7.3.1. Integration Test Steps

1. Follow steps 1-4 to create organization and agent in NM.

2. Prepare adapter as described in: https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/ADAPTER.md
3. Prepare Thing Description of devices and/or services as described in: https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/TD.md and https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/TDExamples.md
4. Prepare agent configuration according to https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/AGENT.md#configuration (auto-registration is set to true)
5. Start gtw api
6. Start agent
7. POST TD to agent endpoint " /objects".

7.3.2. Expected Results

Be able to view registered items in both agent configuration and Neighbourhood Manager.

7.4. TC050_IT030: Integration Test 3 - Use NM API to create contracts

7.4.1. Integration Test Steps

Pre-requisites: Perform integration Tests 1, 2. Organization A has 4 devices. Organization B has 2 services.

1. GET friend ids
2. POST Contract request from Org A to Org B
3. Organization B to call GET Contract feeds
4. Organization B to call PUT Contract update to accept contract

7.4.2. Expected Results

Org A device adapter should be able to call Org B service adapter through XMPP network. The contract should appear in NM.

7.5. TC050_IT040: Integration Test 4 - Addition of a new device in configuration

7.5.1. Integration Test Steps

Pre-requisites: Perform integration Tests 1,2,3. Auto-registration is true.

1. Stop adapter of Org A.
2. Add new device in the adapter's TD.
3. Start adapter, so that it pushes new TD to the agent.

7.5.2. Expected Results

Contract between Org A and Org B should be unaffected. New device should not be able to communicate with the service of Org B.

7.6. TC050_IT050: Integration Test 5 - Check that agent persists TD configuration after shutdown

7.6.1. Integration Test Steps

Pre-requisites: Auto-registration is true.

1. Start gtw api.
2. Start multi-agent.
3. Check agent configuration http://localhost:9997/agent/configuration (there should be no things if this is the first time)

4. Push TD1 to agent (POST http://localhost:9997/agent/objects)
5. Check agent configuration (TD1 should be present)
6. Stop multi-agent
7. Start multi-agent
8. Check agent configuration (TD1 should be present)
9. Repeat steps 1-8 for a 2nd TD.

7.6.2. Expected Results

TD1 should be present after agent re-start. No need for adapter to push again.

7.7. TC050_IT060: Integration Test 6 - Delete adapter and its things when adapter-id is missing from agent config file (Task 26)

7.7.1. Integration Test Steps

Pre-requisites: Auto-registration is true.

1. Add a new adapter in agent config file.
2. Start gtw api and then agent.
3. Check that new adapter exists in agent configuration (call API) -> exists!
4. Push TD to agent -> check NM that object is registered -> it is!
5. Remove adapter from agent config file and re-configure or re-start agent.
6. Check that adapter doesn't exist in agent configuration anymore (call API) -> doesn't exist
7. Check NM that object is deleted --> doesn't exist

7.7.2. Expected Results

Adapter and object are deleted from persisted storage of agent and Semantic Repo.

7.8. TC050_IT070: Integration Test 7 - Device->GDPR VAS->Individual Statistics VAS for pushing measurement

7.8.1. Integration Test Steps

1. Measurement from blood-pressure monitor/weight scale/panic button is taken.
2. It is sent to GDPR VAS.
3. It is sent to Individual Statistics VAS.

7.8.2. Expected Results

Measurement reaches GDPR VAS and Individual Statistics VAS.

7.9. TC050_IT080: Integration Test 8 - Device->GDPR VAS->Urban Marathon VAS for pushing measurement

7.9.1. Integration Test Steps

1. Measurement from activity tracker/beacon is taken.
2. It is sent to GDPR VAS.
3. It is sent to Urban Marathon VAS.

7.9.2. Expected Results

Measurement reaches GDPR VAS and Urban Marathon VAS.

7.10. TC050_IT090: Integration Test 9 – GET properties of Gorenje devices

7.10.1. Integration Test Steps

1. Friendship between CERTHTestMunicipality - Gorenje.
2. Contract between Fridge, Oven and TestGDPR VAS.
3. Request one of devices properties.

7.10.2. Expected Results

Get the actual value of the property.

7.11. TC050_IT100: Integration Test 10 - Deletion of items and contracts

7.11.1. Integration Test Steps

1. Send an empty TD to agent for a specific adapter

2. Check NM that all devices were successfully deleted

3. Check contracts

7.11.2. Expected Results

If devices are deleted contracts should not exist.

7.12. TC050_IT110: Integration Test 11 - Ability to update things (devices, services)

7.12.1. Integration Test Steps

1. Follow Integration Test 2 procedure with an updated TD (e.g. add a property).
2. Re-approve contract between device and VAS.

7.12.2. Expected Results

The new property is callable through VICINITY.

7.13. TC050_IT120: Integration Test 12 - Events of Gorenje devices

7.13.1. Integration Test Steps

1. Follow steps 1,2 of Integration Test 9
2. Subscribe to door events of Gorenje devices (inside agent configuration file).

7.13.2. Expected Results

Open/close oven/fridge door and observe events in the subscribed VAS.

8. TC060: GRN

8.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

We have implemented adapter that is using VICINITY components and is being used by other partners. More about usage of our adapter you can read in following chapters: TINYM – chapter 3.1, CERTH – chapter 7.1 and HITS – chapter 9.1.

With that we proved the integration of the Core Components and made following tests.

8.2. TC060_IT010: Integration Test 1 - Use NM UI to create new agent to establish a new VICINITY Node

8.2.1. Integration Test Steps

On the website for neighbourhood manager perform following steps.

1. In the "Gateways" section create new gateway,
    a. Choose VICINITY agent
    b. Give it a name and set a password.
2. Use password and agent id received after completing step 1 to setup adapter/agent/gateway combo.
3. Prepare agent
    a. Edit config to reflect your set up
        i. Correct URLs for: adapter and gateway
        ii. Correct password and id of the agent
4. Prepare gateway
    a. Edit config to set the path where the logs will be saved
    b. Correct URL for the agent
5. Make sure adapter has correct thing description.
6. Start gateway
7. Start adapter
8. Start agent

8.2.2. Expected Results

Agent starts up without failing and successfully registers devices from TD in the VICINITY. Devices registered can be viewed in the Neighbourhood Manager under "Devices" menu item.

8.3. TC060_IT020: Integration Test 2 – Node to Node communication

8.3.1. Integration Test Steps

Assumptions:

4. One VICINITY node representing a service that will consume data from devices exposed by another node
5. One VICINITY node that exposes devices to the VICINITY
6. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.

Steps:

1. Start both nodes

2. Service node should now make a request to it's agent for data from device belonging to another node. This can be emulated using curl or similar program to send a request.
    a. Request: GET

       http://<agent_URL>:<agent_port>/agent/remote/objects/<oid>/properties/<pid>
    b. Required headers:
        i. *infrastructure-id* which is the internal id of the object requesting data from device
        ii. *adapter-id* which is a unique adapter id specified in the agent configuration and TD.

8.3.2. Expected Results

Response should contain data from the device in the format specified in its thing description. The status code of the response should be 200.

8.4. TC060_IT030: Integration Test 3 – Events: publishing and subscribing to events

8.4.1. Integration Test Steps

Assumptions:

7. One VICINITY node representing a service that will consume data from devices exposed by another node
8. One VICINITY node that exposes devices to the VICINITY
9. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.
10. Node receiving events implements endpoint for receiving events – PUT /objects/<oid>/events/<eid>

Steps:

1. Make sure Node publishing the event has following:
    a. Events are described in the thing description
    b. Channel is described in the agent configuration (https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/AGENT.md#configuration)
2. Make sure Node subscribing to events has following
    a. Subscription is defined in the agent configuration
    b. Endpoint for receiving events is implemented (see assumptions)
3. Start both Nodes
4. Trigger event so it will be sent to subscribers.

8.4.2. Expected Results

Once triggered event is sent to all subscribers, the publisher of the event gets a response with the success message and information about to how many subscribers event was sent.

The subscribing node should be able to receive the events, the data of the event must conform to the definition in the TD of the publishing node.

9. TC070: Tromsø (NO) – Neighbourhood Smart Parking Assisted Living ecosystem (HITS)

9.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

Use Case 1a.1 – Resource management

PNI Parking sensors are reporting vacancy status and temperature. These data are exposed through an adapter. An Android app is used for booking parking space. The visibility of a given parking space is based on reported state from the sensor. It is not possible to book an occupied parking space. IKEA Smart light is managed by business logic through a VAS, and use colour coding to represent the state of the parking sensor.

Use Case 1a.2 – event trigger

Smart appliances are made part of the Tromsø pilot ecosystem through the Gorenje adapter. The use case subscribes to events that are triggered by the opening and closing of doors of the smart appliances. These events are exposing data and delivering payloads through VICINITY. Refrigerator doors and freezer doors trigger differentiated warning levels based on how long they are kept open. Smart oven trigger warning levels based on how long the door is kept closed after the food has been prepared. The warning levels trigger support for emergency booking that is made available for caretakers using the Android app. This opens for priority parking on parking space where the parking sensors are reporting vacancy. This allows for decreasing the response time in case of emergencies, in addition to provide insight in other irregularities that may occur.

In the following tests Use Case and VAS is proved by establishing reliably new nodes, ensuring node to node communication, and establishing subscription relationships with endpoints.

9.2. TC070: Integration Test 1 - Use NM UI to create new agent to establish a new VICINITY Node

9.2.1. Integration Test Steps

On the website for neighbourhood manager perform following steps.

1. In the “Gateways” section create new gateway,
    a. Choose VICINITY agent
    b. Give it a name and set a password.
2. Use password and agent id received after completing step 1 to setup adapter/agent/gateway combo.
3. Prepare agent
    a. Edit config to reflect your set up
        i. Correct URLs for: adapter and gateway
        ii. Correct password and id of the agent
4. Prepare gateway
    a. Edit config to set the path where the logs will be saved
    b. Correct URL for the agent
5. Make sure adapter has correct thing description.
6. Start gateway
7. Start adapter
8. Start agent

9.2.2. Expected Results

Agent starts up without failing and successfully registers devices from TD in the VICINITY. Devices registered can be viewed in the Neighbourhood Manager under "Devices" menu item.

9.3. TC070_IT020: Integration Test 2 – Node to Node communication

9.3.1. Integration Test Steps

Assumptions:

11. One VICINITY node representing a service that will consume data from devices exposed by another node
12. One VICINITY node that exposes devices to the VICINITY
13. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.

Steps:

1. Start both nodes
2. Service node should now make a request to it's agent for data from device belonging to another node. This can be emulated using curl or similar program to send a request.
    a. Request: GET

       http://<agent_URL>:<agent_port>/agent/remote/objects/<oid>/properties/<pid>
    b. Required headers:
        i. *infrastructure-id* which is the internal id of the object requesting data from device
        ii. *adapter-id* which is a unique adapter id specified in the agent configuration and TD.

9.3.2. Expected Results

Response should contain data from the device in the format specified in its thing description. The status code of the response should be 200.

9.4. TC070_IT030: Integration Test 3 - Events: publishing and subscribing to events

9.4.1. Integration Test Steps

Assumptions:

14. One VICINITY node representing a service that will consume data from devices exposed by another node
15. One VICINITY node that exposes devices to the VICINITY
16. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.
17. Node receiving events implements endpoint for receiving events – PUT /objects/<oid>/events/<eid>

Steps:

1. Make sure Node publishing the event has following:
    a. Events are described in the thing description

   b. Channel is described in the agent configuration (https://github.com/VICINITYh2020/VICINITY-agent/blob/master/docs/AGENT.md#configuration)
2. Make sure Node subscribing to events has following
   a. Subscription is defined in the agent configuration
   b. Endpoint for receiving events is implemented (see assumptions)
3. Start both Nodes
4. Trigger event so it will be sent to subscribers.

9.4.2. Expected Results

Once triggered event is sent to all subscribers, the publisher of the event gets a response with the success message and information about to how many subscribers event was sent.

The subscribing node should be able to receive the events, the data of the event must conform to the definition in the TD of the publishing node.

10. Martim Longo (PT) – Neighbourhood GRID ecosystem (ENERC)

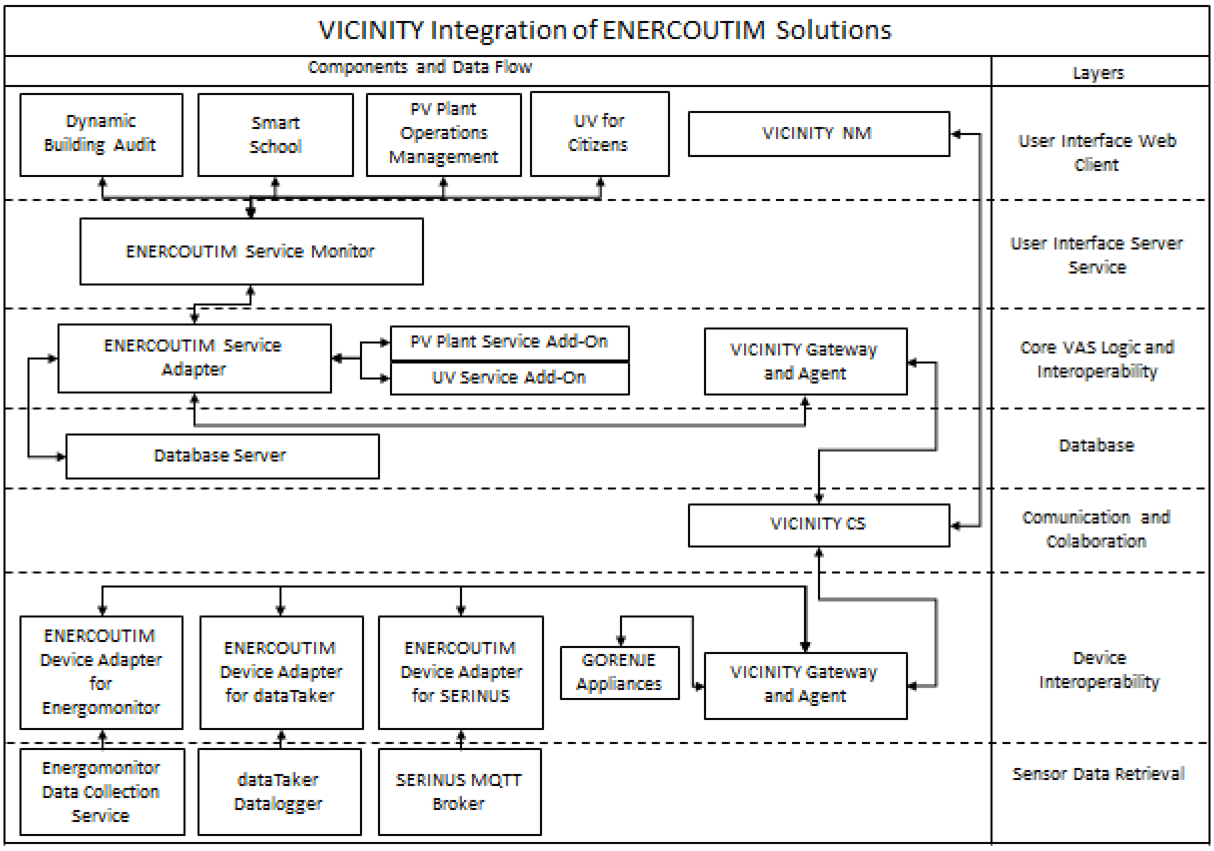
10.1. Explanation of the VAS and the Role of the VICINITY Prototype in the Use Case

ENERCOUTIM solutions comprises three sets of use cases: Dynamic Building Audit and Smart School; UV for Citizens; and PV Plant Operations Management.

The Dynamic Building Audit use cases comprise overall Indoor Environment Quality (IEQ) and Energy Consumption. The Smart School use case is focused on a subset of IEQ parameters (Temperature, CO2, Humidity and Light) relevant for the zones frequented by the students. PV Plant Operations Management VAS is responsible for combining data from different sensors and evaluate opportunity for current maintenance operations. UV for Citizens use case consists in an informational service about current UV radiation and recommended precautions.

The VAS user interface is implemented as a web application with responsive layout, usable in desktop and mobile browsers, displaying real-time sensor data, alert levels and historical sensor data for analytics. VAS components are configurable using a json file and extendable using java-based add-ons to meet requirements for the different use cases.

ENERCOUTIM solutions use VICINITY to manage and route data retrieval from diverse sensors installed in pilot site locations to the appropriate VAS mainly installed on the application server. Data from sensors is retrieved locally for the dataTaker datalogger and via data collection services operated by device manufacturers for Energomonitor (API based data service) and SERINUS (MQTT based data service). Gorenje Appliances data is retrieved using a VICINITY service, managed using a VICINITY contract.


VICINITY Integration of ENERCOUTIM Solutions
Components and Data Flow
Layers
Dynamic Building Audit
Smart School
PV Plant Operations Management
UV for Citizens
VICINITY NM
User Interface Web Client
ENERCOUTIM Service Monitor
User Interface Server Service
ENERCOUTIM Service Adapter
PV Plant Service Add-On
UV Service Add-On
VICINITY Gateway and Agent
Core VAS Logic and Interoperability
Database Server
Database
VICINITY CS
Comunication and Colaboration
ENERCOUTIM Device Adapter for Energomonitor
ENERCOUTIM Device Adapter for dataTaker
ENERCOUTIM Device Adapter for SERINUS
GORENJE Appliances
VICINITY Gateway and Agent
Device Interoperability
Energomonitor Data Collection Service
dataTaker Datalogger
SERINUS MQTT Broker
Sensor Data Retrieval

10.2. Integration Test 1 – Device registration using NM

VICINITY device registration using NM tests were completed in two Raspberry Pi 3 running Raspbian 9 with Oracle Java JDK 8 (SolarLab and Martim Longo School) and a Linode VPS running UBUNTU Server 18.04 LTS with Oracle Java JDK 8.

10.2.1. Integration Test Steps

On the website for neighbourhood manager perform following steps.

1. In the "Gateways" section create new gateway,
   a. Choose VICINITY agent
   b. Give it a name and set a password.
2. Use password and agent id received after completing step 1 to setup adapter/agent/gateway combo.
3. Prepare agent
   a. Edit config to reflect your set up
      i. Correct URLs for: adapter and gateway
      ii. Correct password and id of the agent
4. Prepare gateway
   a. Edit config to set the path where the logs will be saved
   b. Correct URL for the agent
5. Make sure adapter has correct thing description.
6. Start gateway
7. Start adapter
8. Start agent

10.2.2. Expected Results

Agent starts up without failing and successfully registers devices from the Thing Description in VICINITY. Devices registered can be viewed in the Neighbourhood Manager under "Devices" menu item.

10.2.3. Actual Results

The device is visible in the Neighbourhood Manager, with the relevant TD, without errors.

10.2.4. Problems Identified

No problems identified. A support incident related to not finding newly registered devices on user interface of VICINITY NM was reported and solved.

10.3. Integration Test 2 – Node to Node Communication

VICINITY node to node communication tests were completed in two Raspberry Pi 3 running Raspbian 9 with Oracle Java JDK 8 (SolarLab and Martim Longo School) and a Linode VPS running UBUNTU Server 18.04 LTS with Oracle Java JDK 8.

10.3.1. Integration Test Steps

Assumptions:

1. One VICINITY node representing a service that will consume data from devices exposed by another node
2. One VICINITY node that exposes devices to VICINITY
3. Nodes belong to the same organisation or there is a contract established between two different organisations for access between nodes.

4. Start both nodes
5. The Service node should now make a request to its agent for data from device belonging to another node. This can be emulated using curl or similar program to send a request.
    a. Request: GET

       http://<agent_URL>:<agent_port>/agent/remote/objects/<oid>/properties/<pid>
    b. Required headers:
        i. *infrastructure-id* which is the internal id of the object requesting data from device
        ii. *adapter-id* which is a unique adapter id specified in the agent configuration and TD.

10.3.2. Expected Results

Response should contain data from the device in the format specified in its thing description. The status code of the response should be 200.

10.3.3. Actual Results

Coherent data arrives reliably from the node.

10.3.4. Problems Identified

No problems identified. A communication incident occurred while testing the device adapter for Energomonitor on the Raspberry Pi, but found to be related with Raspbian 9 specific Java certificate repository configuration issues while accessing the Energomonitor API and solved by reconfiguring the certificate repository.

11. Conclusion

The integration tests and finalizing work on the adapters was vital to the timely completion of the prototypes. The VICINITY Core components and client infrastructure (testing labs and pilot sites), including integrated VICINITY Gateway APIs, Agents and adapters, were integrated into one VICINITY Integrated Prototype Release Candidate 1. The list of integrated VICINITY prototypes is covered in section 2.2.

Integration of the components and infrastructures was covered by 42 integration tests with 34 identified bugs while 33 were resolved. Note that one bug is fixed, but requires long running testing. The integration tests cover all interfaces in VICINITY between VICINITY Core components and client infrastructure.

The Integrated VICINITY prototype is subject of the deployment in the pilot site installations in WP7 and will be under VICINITY evaluation in WP8. During the deployment and evaluation, it is anticipated that other issues will be identified. These issues will be fixed and regression integration test will be performed. The results will be covered in WP3 and WP4 continuous updates and/ evaluation report.

Together with workshops, weekly meetings and a tight collaboration from all partners, task T6.1 Integration of VICINITY Components has established the needed confirmation and result to a system where all partners have integrated their adapters and been able to prove and develop a stable and well-integrated solution.

Annex I Test Results TINMY

Table 3 Test Result TINYM

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Establishing a new VICINITY Node	20.07.2018	OK!	—
Integration test 2: Node to Node communication	20.07.2018	OK!	—
Integration test 3: Events publishing and subscribing to events	23.07.2018	Failed!	Filed on Open Project, #22
Integration test 1: Establishing a new VICINITY Node	29.08.2018	OK!	
Integration test 2: Node to Node communication	29.08.2018	OK!	
Integration test 3: Events publishing and subscribing to events	29.08.2018	OK!	
Integration test 1: Establishing a new VICINITY Node	05.11.2018	OK!	
Integration test 2: Node to Node communication	05.11.2018	OK!	
Integration test 3: Events publishing and subscribing to events	05.11.2018	OK!	

Annex II Test results ATOS

Table 4 Test Results ATOS

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Device registration	24.08.2018	OK!	—
Integration test 2: Validate Adapter (via Thing Description validation)	28.08.2018	OK!	—
Integration test 3: Active vs passive discovery	08.09.2018	OK!	—
Integration test 4: Node-to-Node communication (request/response)	08.09.2018	OK!	Issue #41 (solved)
Integration test 5: Node-to-Node communication (pub/sub)	—	—	—

Annex III Test results AAU

Table 5 test results AAU

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Use NM UI to establish new VICINITY Nodes and establish friendships	21.09.2018	Problems identified	Bug #46: Change of status trigger exception.
Integration test 2: Communication between service node to device node	28.09.2018	OK!	
Integration test 3: Events publishing and subscribing to events	30.09.2018	OK!	

Annex IV Test results UNIKL

Table 6 Test Results UNIKL

Test case	Date	Outcome	Issue in OpenProject if any
Agent/GTW-API Version 0.6.2			
Integration test 1: Device registration using NM	9/10/2018	OK!	—
Integration test 2: Node-to-Node communication (request/response)	9/10/2018	OK!	---
Integration test 3: Node-to-Node communication (pub/sub)	9/10/2018	Failed!	Issue #33
Agent/GTW-API Version 0.6.3			
Integration test 1: Device registration using NM	11/2/2018	OK!	—
Integration test 2: Node-to-Node communication (request/response)	11/2/2018	OK!	---
Integration test 3: Node-to-Node communication (pub/sub)	11/2/2018	OK!	Issue was resolved in latest agent/gtw-api

Annex V Test results CERTH and GNOMON

Table 7 Test results CERTH and GNOMON

Test case	Date	Outcome	Issues identified in OpenProject	Conducted by
Integration test 1: Use NM API to create organization, agent, friendships	8/24/2018	OK!	—	GNOMON, CERTH
Integration test 2: Auto-registration of devices and services	9-20-2018	OK!	Bug 19 closed, Bug 20 closed, Bug 31 closed	GNOMON, CERTH
Integration test 3: Use NM API to create contracts	2/10/2018	OK!	Bug 29 (closed)	GNOMON
Integration Test 4-Addition of a new device in configuration	7/19/2018	OK!		CERTH
Integration Test 5-Check that agent persists TD configuration after shutdown	7/12/2018	OK!		CERTH
Integration Test 6- Delete adapter and its things when adapter-id is missing from agent config file	8/30/2018	OK!	Task 26 (closed)	CERTH
Integration Test 7- Device->GDPR VAS->Individual Statistics VAS for pushing measurement	9-21-2018	OK!	Bug 18 (in progress) seems to be solved, Bug 39 (in progress) seems to be solved, Bug 47 Reason: Destination object e5c654dd-788d-4e0c-aaf2-5836aa9f1139 is not online Bug 83: Message might got lost.	CERTH, GNOMON
Integration Test 8- Device->GDPR VAS->Urban Marathon VAS for pushing measurement	9-21-2018	OK!	Bug 18 (in progress) seems to be solved, Bug 39 (in progress) seems to be solved, Bug 47 Reason: Destination object e5c654dd-788d-4e0c-aaf2-5836aa9f1139 is not online Bug 83: Message might get lost.	CERTH, GNOMON
Integration Test 9 – GET properties of Gorenje devices	9-20-2018	OK!	Bug 47: Reason: Destination object 42760305-553c-407d-8640-b8f3bb949e34 is not online.	CERTH, Gorenje
Integration Test 10 – Deletion of items	9-21-2018	OK!	Bug 23 (closed)	CERTH
Integration Test 11 - Ability to update things (devices, services)	2/10/2018	OK!	Feature 25 (done)	CERTH, GNOMON
Integration Test 12 - Events of Gorenje devices	10-19-2018	OK!		CERTH, Gorenje

Annex VI Test results GRN

Table 8 Test results GRN

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Establishing a new VICINITY Node	10/21/2018	OK!	Closed bug on Open Project, #45
Integration test 2: Node to Node communication	9/20/2018	OK!	
Integration test 3: Events publishing and subscribing to events	10/21/2018	OK!	

Annex VII Test Results HITS

Table 9 Test Results HITS

Test case	Date	Outcome	Issue in OpenProject if any	Parter Name	Pilot cite
Integration test 1: Establishing a new VICINITY Node	7/20/2018	OK!	—	HITS	Tromsø
Integration test 2: Device registration using NM	7/20/2018	OK!	—	HITS	Tromsø
Integration test 3: Node to Node communication (request/response)	7/3/2018	OK!	—	HITS	Tromsø
Integration test 4: Node-to-Node communication (pub/sub)	9/10/2018	Failed	Filed on Open Project; #46	HITS, AAU	Tromsø
Integration test 4	9/17/2018	OK!	-	HITS, AAU	Tromsø

Annex VIII Test Results ENERC

Table 10 Test Results ENERC

Test case	Date	Outcome	Issue in OpenProject if any	Parter Name	Pilot site
Integration test 1: Device registration using NM	7 Nov 2018	OK!	—	ENERC	Martim Longo
Integration test 2: Node to Node communication	8 Nov 2018	OK!	—	ENERC	Martim Longo

Annex I Test Results TINMY

Table 11 Test Result TINYM

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Establishing a new VICINITY Node	20.07.2018	OK!	—
Integration test 2: Node to Node communication	20.07.2018	OK!	—
Integration test 3: Events publishing and subscribing to events	23.07.2018	Failed!	Filed on Open Project, #22
Integration test 1: Establishing a new VICINITY Node	29.08.2018	OK!	
Integration test 2: Node to Node communication	29.08.2018	OK!	
Integration test 3: Events publishing and subscribing to events	29.08.2018	OK!	
Integration test 1: Establishing a new VICINITY Node	05.11.2018	OK!	
Integration test 2: Node to Node communication	05.11.2018	OK!	
Integration test 3: Events publishing and subscribing to events	05.11.2018	OK!	

Annex II Test results ATOS

Table 12 Test Results ATOS

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Device registration	24.08.2018	OK!	—
Integration test 2: Validate Adapter (via Thing Description validation)	28.08.2018	OK!	—
Integration test 3: Active vs passive discovery	08.09.2018	OK!	—
Integration test 4: Node-to-Node communication (request/response)	08.09.2018	OK!	Issue #41 (solved)
Integration test 5: Node-to-Node communication (pub/sub)	—	—	—

Annex III Test results AAU

Table 13 test results AAU

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Use NM UI to establish new VICINITY Nodes and establish friendships	21.09.2018	Problems identified	Bug #46: Change of status trigger exception.
Integration test 2: Communication between service node to device node	28.09.2018	OK!	
Integration test 3: Events publishing and subscribing to events	30.09.2018	OK!	

Annex IV Test results UNIKL

Table 14 Test Results UNIKL

Test case	Date	Outcome	Issue in OpenProject if any
Agent/GTW-API Version 0.6.2			
Integration test 1: Device registration using NM	9/10/2018	OK!	—
Integration test 2: Node-to-Node communication (request/response)	9/10/2018	OK!	---
Integration test 3: Node-to-Node communication (pub/sub)	9/10/2018	Failed!	Issue #33
Agent/GTW-API Version 0.6.3			
Integration test 1: Device registration using NM	11/2/2018	OK!	—
Integration test 2: Node-to-Node communication (request/response)	11/2/2018	OK!	---
Integration test 3: Node-to-Node communication (pub/sub)	11/2/2018	OK!	Issue was resolved in latest agent/gtw-api

Annex V Test results CERTH and GNOMON

Table 15 Test results CERTH and GNOMON

Test case	Date	Outcome	Issues identified in OpenProject	Conducted by
Integration test 1: Use NM API to create organization, agent, friendships	8/24/2018	OK!	—	GNOMON, CERTH
Integration test 2: Auto-registration of devices and services	9-20-2018	OK!	Bug 19 closed, Bug 20 closed, Bug 31 closed	GNOMON, CERTH
Integration test 3: Use NM API to create contracts	2/10/2018	OK!	Bug 29 (closed)	GNOMON
Integration Test 4-Addition of a new device in configuration	7/19/2018	OK!		CERTH
Integration Test 5-Check that agent persists TD configuration after shutdown	7/12/2018	OK!		CERTH
Integration Test 6- Delete adapter and its things when adapter-id is missing from agent config file	8/30/2018	OK!	Task 26 (closed)	CERTH
Integration Test 7- Device->GDPR VAS->Individual Statistics VAS for pushing measurement	9-21-2018	OK!	Bug 18 (in progress) seems to be solved, Bug 39 (in progress) seems to be solved, Bug 47 Reason: Destination object e5c654dd-788d-4e0c-aaf2-5836aa9f1139 is not online Bug 83: Message might got lost.	CERTH, GNOMON
Integration Test 8- Device->GDPR VAS->Urban Marathon VAS for pushing measurement	9-21-2018	OK!	Bug 18 (in progress) seems to be solved, Bug 39 (in progress) seems to be solved, Bug 47 Reason: Destination object e5c654dd-788d-4e0c-aaf2-5836aa9f1139 is not online Bug 83: Message might get lost.	CERTH, GNOMON
Integration Test 9 – GET properties of Gorenje devices	9-20-2018	OK!	Bug 47: Reason: Destination object 42760305-553c-407d-8640-b8f3bb949e34 is not online.	CERTH, Gorenje
Integration Test 10 – Deletion of items	9-21-2018	OK!	Bug 23 (closed)	CERTH
Integration Test 11 - Ability to update things (devices, services)	2/10/2018	OK!	Feature 25 (done)	CERTH, GNOMON
Integration Test 12 - Events of Gorenje devices	10-19-2018	OK!		CERTH, Gorenje

Annex VI Test results GRN

Table 16 Test results GRN

Test case	Date	Outcome	Issue in OpenProject if any
Integration test 1: Establishing a new VICINITY Node	10/21/2018	OK!	Closed bug on Open Project, #45
Integration test 2: Node to Node communication	9/20/2018	OK!	
Integration test 3: Events publishing and subscribing to events	10/21/2018	OK!	

Annex VII Test Results HITS

Table 17 Test Results HITS

Test case	Date	Outcome	Issue in OpenProject if any	Parter Name	Pilot cite
Integration test 1: Establishing a new VICINITY Node	7/20/2018	OK!	—	HITS	Tromsø
Integration test 2: Device registration using NM	7/20/2018	OK!	—	HITS	Tromsø
Integration test 3: Node to Node communication (request/response)	7/3/2018	OK!	—	HITS	Tromsø
Integration test 4: Node-to-Node communication (pub/sub)	9/10/2018	Failed	Filed on Open Project; #46	HITS, AAU	Tromsø
Integration test 4	9/17/2018	OK!	-	HITS, AAU	Tromsø

Annex VIII Test Results ENERC

Table 18 Test Results ENERC

Test case	Date	Outcome	Issue in OpenProject if any	Parter Name	Pilot site
Integration test 1: Device registration using NM	7 Nov 2018	OK!	—	ENERC	Martim Longo
Integration test 2: Node to Node communication	8 Nov 2018	OK!	—	ENERC	Martim Longo