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 D1.6

VICINITY Architectural Design

Work Package:	**WP1 – VICINITY concept Requirements, Barriers, Specification and Architecture**
Task(s):	**T1.4 – Functional & Technical Specification, Architectural design**
Lead Beneficiary:	BVR
Due Date:	31 March 2017
Submission Date:	31 March 2017
Deliverable Status:	Final
Deliverable Type:	R
Dissemination Level:	PU
File Name:	**VICINITY_D1_6_Architectural_Design_1.0.pdf**

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

Authors List

Leading Author (Editor)			
Surname	**First Name**	**Beneficiary**	**Contact email**
Oravec	Viktor	BVR	viktor.oravec@bavenir.eu

Co-authors (in alphabetic order)				
No	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1.	Heinz	Christopher	UNIKL	heinz@cs.uni-kl.de
2.	Hovstø	Asbjørn	HITS	hostvo@online.no
3.	Kaggelides	Kostis	GNOMON	k.kaggelides@gnomon.com.gr
4.	Karkaletsis	Kostas	GNOMON	k.karkaletsis@gnomon.com.gr

5.	Karageorgopoulou	Anastasia	CERTH	nkarageor@iti.gr
6.	Kostelnik	Peter	IS	peter.kostelnik@intersoft.sk
7.	Margariti	Katerina	CERTH	kmargariti@iti.gr
8.	Moll Nilsen	Rolv	TINYM	rmn@serinustechnology.com
9.	Mynzhasova	Aida	UNIKL	infrared.ng@gmail.com
10.	Poveda	Maria	UPM	mpoveda@fi.upm.es
11.	Serena	Fernando	UPM	fserena@fi.upm.es
12.	Sveen	Flemming	HITS	flsveen@online.no
13.	Tryferidis	Athanasios	CERTH	thanasic@iti.gr
14.	Zandes	Dimitrios	GNOMON	d.zandes@gnomon.com.gr

Reviewers List

List of Reviewers (in alphabetic order)				
No	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1.	Dickerson	Keith	CAL	keith.dickerson@mac.com
2.	Koelsch	Johannes	UNIKL	koelsch@cs.uni-kl.de
3.	Raúl	García-Castro	UPM	rgarcia@fi.upm.es
4.	Tryferidis	Athanasios	CERTH	thanasic@iti.gr
5.	Vinkovic	Saso	GRN	saso.vinkovic@gorenje.com

Revision Control

Version	Date	Status	Modifications made by
0.1	17. June 2016	Initial Draft	Oravec (BVR)
0.1.1	1. July 2016	Update based on GA 01	Oravec (BVR)
0.1.2	12. October 2016	Update based on Pilot site visits	Oravec (BVR)
0.1.3	11. November 2016	Update based on preliminary VICINITY Review	Oravec (BVR)
0.1.4	22. December	Update based on D1.5 QaR review	Oravec (BVR)
0.1.5	17. February 2017	Contributions from IS	Kostelnik (BVR)
0.1.6	23. February 2017	Contributions from UPM	Serena (BVR)
0.1.7	24. February 2017	Contributions from HITS	Hovstø (BVR)
0.2	5.March 2017	Deliverable version uploaded for Quality Check	Oravec (BVR)
0.3	30. March 2017	Quality checked final version	Oravec (BVR)
0.4	31. March 2017	Final Draft reviewed	Oravec (BVR)
1.0	31. March 2017	Submission to the EC	Oravec (BVR)

Table of Contents

List of Tables

List of Figures

List of Definitions & Abbreviations

Abbreviation	Definition
5Vs	Volume, Velocity, Veracity, Variability, and Variety
6LoWPAN	IPv6 over Low Power Wireless Personal Area Network
API	Application Programming Interface
CM	Conceptual Model
CoAP	Constrained Application Protocol
CPU	Central processing unit
DEVOPS	software **DEV**elopment and information technology **OP**eration**S**
EC	European Commission
EU	European Union
GDPR	General Data Protection Regulation
HDD	Hard disk drive
ICU	International Components for Unicode
IG	Interest Group
IoT	Internet of Things
IoT RA	Internet of Things Reference Architecture
JMX	Java Management Extensions
MQTT	Message Queue Telemetry Transport
OWL	W3C Web Ontology Language
PII	Personally Identifiable Information
P2P	Peer to Peer
RA	Reference Architecture
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RDF	Resource Description Framework
REST	Representational state transfer
RM	Reference Model
SNMP	Simple Network Management Protocol
SPARQL	SPARQL Protocol and RDF Query Language
TD	Thing Description
TED	Thing Ecosystem Description
UML	Universal Modelling Language

Abbreviation	Definition
VTED	Virtual TED
XMPP	Extensible Messaging and Presence Protocol
WoT	Web of Things

Executive Summary

This document, referred as "D1.6 VICINITY Architectural design", is a deliverable of the VICINITY project, funded by the European Commission (EC) Directorate-General for Research and Innovation (DG RTD), under its Horizon 2020 Research and Innovation Programme (H2020). **VICINITY** is building a device and standard independent platform for IoT infrastructures that offers „Interoperability as a Service". They aim of the VICINITY is to solve the interoperability problem with a virtual neighbourhood concept which is a decentralized, user-centric approach that allows transparency and full control over exchanged data.
This deliverable directly addresses the Objective 2.4 "VICINITY Technical requirements and solution architecture specified", in terms of describing architectural decisions which shape the VICINITY interoperability platform. The selected architectural options and decisions are based on functional and quality requirements extracted from D1.5 VICINITY Technical requirement specification document and can be summarized in the following Figure 1.

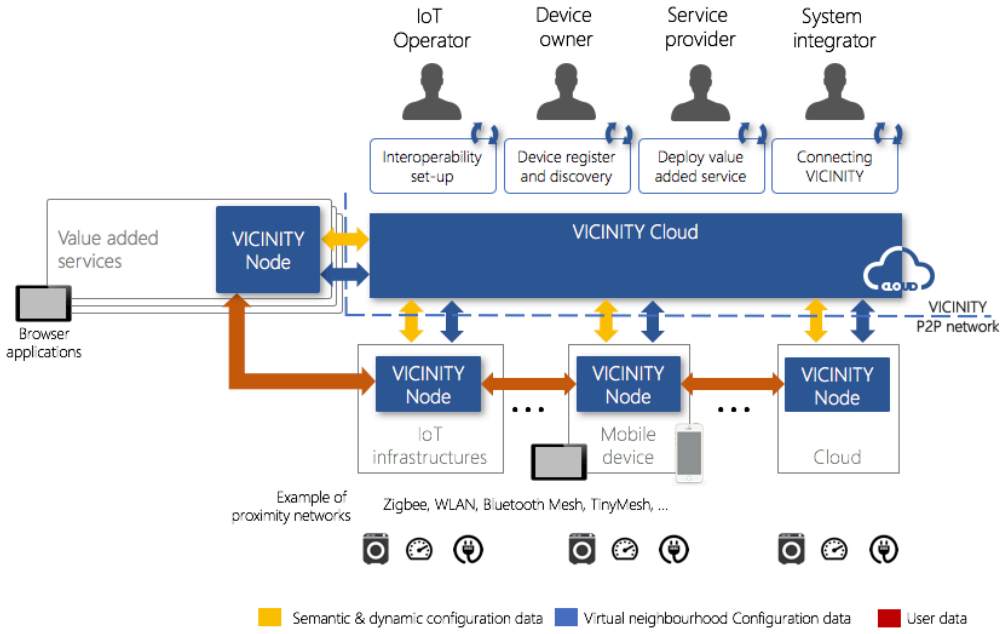

Figure 1 High-level logical VICINITY architecture

The VICINITY architecture is divided into the following principal components:

- VICINITY Cloud providing set of services to setup peer-to-peer interoperability between IoT environments (further referred as virtual neighbourhoods), that are including:
  - access control to the VICINITY-connected IoT objects,
  - services for device discovery and registration,
  - deployment of value added services over VICINITY-connected IoT objects and environments,
  - and setting up connection to VICINITY (e.g. getting integrated to VICINITY);
- VICINITY Node, integrating IoT infrastructures and value-added services to the VICINITY interoperability network. The VICINITY Node includes:

- VICINITY Communication Node which handles the secure and peer-to-peer exchange of IoT data (IoT events, actions as well as IoT properties) and configuration data with other integrated infrastructures, value added services and the VICINITY Cloud;
- VICINITY Gateway API providing semantic interoperability interface for VICINITY Adapters/ Agents to register IoT objects, to access shared IoT objects or to discover and query IoT objects;
- VICINITY Adapters and Agents adapting VICINITY into local infrastructures.

The VICINITY Cloud and set of VICINITY Nodes are creating secure peer-to-peer communication network. This network of loosely coupled peers copying geographically distribution of integrated infrastructures and value-added service thus user data exchange load is distributed accordingly. Moreover, any performance, availability and system failure issues originated in integrated infrastructures and value-added services or even in VICINITY Cloud can be localised in the peer (i.e. spread of issues can be kept under control in the certain part of the peer-to-peer network).

The VICINITY Cloud components such as VICINITY Neighbourhood manager (providing user interface to VICINITY Users), Semantic discovery and dynamic configuration agent platform (providing semantic platform) and VICINITY Communication Server (providing control of communication between integrated infrastructures) shall be deployed as high available software components being to scale in/out VICINITY cloud to current needs to the integrated infrastructures and value added services.

The distributed infrastructure of VICINITY is enhanced by interoperability approach to provide a standard way to both *Discover* and *Access* heterogeneous IoT objects distributed among sparse IoT infrastructures based on the work being done by the W3C Web of Things (WoT) WG[1]. Therefore, VICINITY shall rely on *Thing description* (TD) introduced by WoT to describe every IoT object (which can represent either physical or abstract Things) that belong to any integrated IoT infrastructure, which in turn shall be described as an ecosystem of IoT objects (Things ecosystem descriptions – TEDs). Accordingly, each interested part in taking advantage of the semantic interoperability solution provided by VICINITY must *only* be able to understand such description frames as well as the Web of Things ontology through the VICINITY Gateway API interface.

The VICINITY trust, privacy and security features are mainly covered:

- Verification of VICINITY users through VICINITY Neighbourhood Manager and verification and validation of IoT objects (value-added services and devices) through the VICINITY communication server security services;
- End-to-end security and authenticity of exchanged data through peer-to-peer network using encryption and data integrity services of VICINITY communication Server and Node components;
- Controlled registration and access to value-added services and IoT objects properties, action and events based on sharing access rules with private data processing consents[2] through VICINITY Neighbourhood manager set by device owner or service provided;

[1] https://www.w3.org/WoT/IG/

[2] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)

- Exchanging of private user data only through VICINITY Nodes and Peer-to-peer network without storing data in VICINITY Cloud[3];

The VICINITY Architectural design together with VICINITY Technical requirements specification defines the base line for the following implementation of VICINITY core components, Semantic Discovery and Dynamic Configuration services in WP3 and VICINITY Gateway Adapters and VICINITY Agent and Auto-Discovery platform in WP4.

[3] VICINITY Cloud stores only meta-data necessary to manage access to devices, value-added services and facilitate exchanged information. Meta-data stored in VICINITY Cloud can be accessed by VICINITY Neighborhood manager user interface by appropriate users.

1. Introduction

This document provides the architectural design as developed within "Task 1.4 – Functional & Technical Specification, Architectural design".

The relevant tasks (from which architecture design is derived) are as follows:

- Task 1.1 – Elicitation of user requirements and barriers related to IoT interoperability;
- Task 1.2 – Pilot Sites Surveys and extraction of Use Case requirements;
- Task 1.3 – VICINITY Platform User and Business Requirement Definition;
- Task 2.1 – Analysis of available platforms, IoT infrastructures, IoT ontologies and standards.

Thus, it is suggested that readers should be familiar with and have access to the following deliverables:

- D1.1 VICINITY requirement capture framework;
- D1.2 Report on business drivers and barriers of IoT interoperability and value added services;
- D1.3 Report on pilot sites and operational requirements;
- D1.4 Report on VICINITY business requirements;
- D1.6 Technical requirements specification;
- D2.1 Analysis of Standardisation Context and Recommendations for Standards Involvement.

This document is prepared as a starting point for technical audience to understand the overall architecture design and decision of the VICINITY solution. However, the document will not provide comprehensive and detailed documentation with all the technical information. It should help technical partners within consortium during the development of the VICINITY solution. Consequently, the most important source of information will be the actual source code and its documentation (e.g. user manuals, installation guides and source code comments).

The system will be developed through repeated cycles (iterative) and in smaller portions at a time (incremental), what means that each iteration will contain part of the analysis, implementation and testing. This technical documentation will be also continuously updated throughout the project.

1.1. Deliverable objectives

The objectives of this deliverable are as follows:

- to define VICINITY architecture design of core VICINITY components and concepts;
- to define implementation of usability, reliability, efficiency, maintenance.

1.2. Deliverable structure

The deliverable is divided into the following sections:

- Section 1 includes executive summary of this deliverable;
- Section 2 provides to deliverable (this section);
- Section 3 coins an overall VICINITY Architecture and principal use-cases of the architecture;
- Section 4 presents a logical view of the VICINITY interoperability platform;
- Section 5 defines the information flow and important VICINITY data storages;
- Section 6 encompasses cross components' processes in VICINITY;
- Section 7 summarizes the interfaces between VICINITY components;
- Section 8 presents the conceptual deployment scenarios of VICINITY Cloud and VICINITY Node;

- Section 9 proposes important concepts of each VICINITY Architectural component;
- Section 10 provides Security features of VICINITY;
- Section 11: Conclusions;
- Appendix A: Deployment of VICINITY Node on VICINITY Gateway;
- Appendix B: Internet of Things Reference Architecture (IoT RA);
- Appendix C: Register of important terms.

1.3. Relation to other Tasks and Deliverables

The following documents have been used to derive detail design:

- D1.1 VICINITY requirement capture framework – defines how the requirements are managed in VICINITY project;
- D1.2 Report on business drivers and barriers of IoT interoperability and value added services – defines constraints for functional specification based on stakeholders' drivers and barriers;
- D1.3 Report on pilot sites and operational requirements – defines constraints deployment and environment constraints for functional specification;
- D1.4 Report on VICINITY business requirements – provides inputs on desired VICINITY functionality;
- D1.5 VICINITY technical requirements specification – provides non-functional requirements to shape the architecture design.

The following tasks utilize the architecture design reported in this document:

- Task 3.1, Task 3.2 and Task 3.3 to detail design and implement of VICINITY core components, Semantic Discovery and Dynamic Configuration services;
- Task 4.1, Task 4.2 and Task 4.3 to detail design and to implement of VICINITY Client components and security services.

2. Overall architecture design of VICINITY

The VICINITY is built around the concept of connecting different IoT ecosystems through VICINITY (interoperability as a services) which enable to interact with IoT objects from other different ecosystems as if they were their own. The interoperability services create an environment where value-added services can be deployed and processes cross-domain exchanged information across different domains (Figure 2).

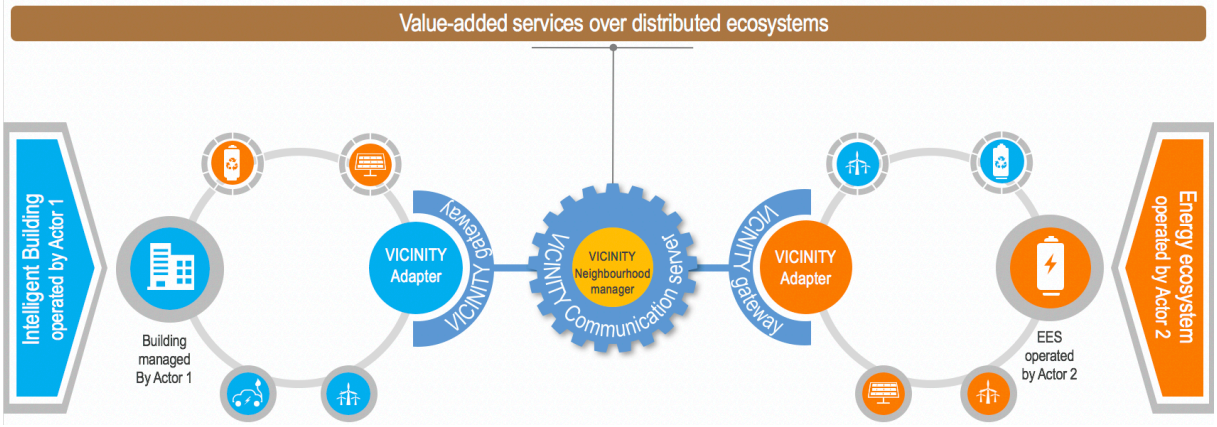

Figure 2 VICINITY Concept

In presented figure two separate ecosystems are presented intelligent building and energy ecosystem. Each of these ecosystems are integrated to VICINITY by its VICINITY Adapter through the VICINITY gateway. Based on the setup of virtual neighbourhood in VICINITY Neighbourhood manager, VICINITY Adapters may access remote IoT objects, for example a battery in an Energy ecosystem, and simulate it as a part of their ecosystem. Moreover, IoT objects shared by VICINITY Adapter within a virtual neighbourhood may be accessed by value-added services to provide cross-domain services using common a VICINITY ontology.

2.1. VICINITY architecture concept

The objective of VICINITY architecture is to build-up a decentralised interoperability between integrated IoT infrastructures and value-added services (called virtual neighbourhood) through a peer-to-peer (P2P) network of VICINITY Nodes. VICINITY Nodes integrates infrastructures in to VICINITY. The infrastructures' managers can setup the sharing access of their IoT objects without losing the control over them using VICINITY Cloud.

The VICINITY Nodes create a closed and secure VICINITY P2P Network for the exchange of user data. The VICINITY P2P Network is configured via the VICINITY Cloud, based on a virtual neighbourhood configuration data – sharing access rules of IoT objects within the VICINITY with security properties (such as encryption, data integrity, etc.) to ensure security and privacy of exchanged use data.

The VICINITY Nodes are configured based on a semantic & dynamic configuration data (semantic descriptions of IoT objects) from the VICINITY Cloud to ensure semantic interoperability between integrated infrastructures and value added services.

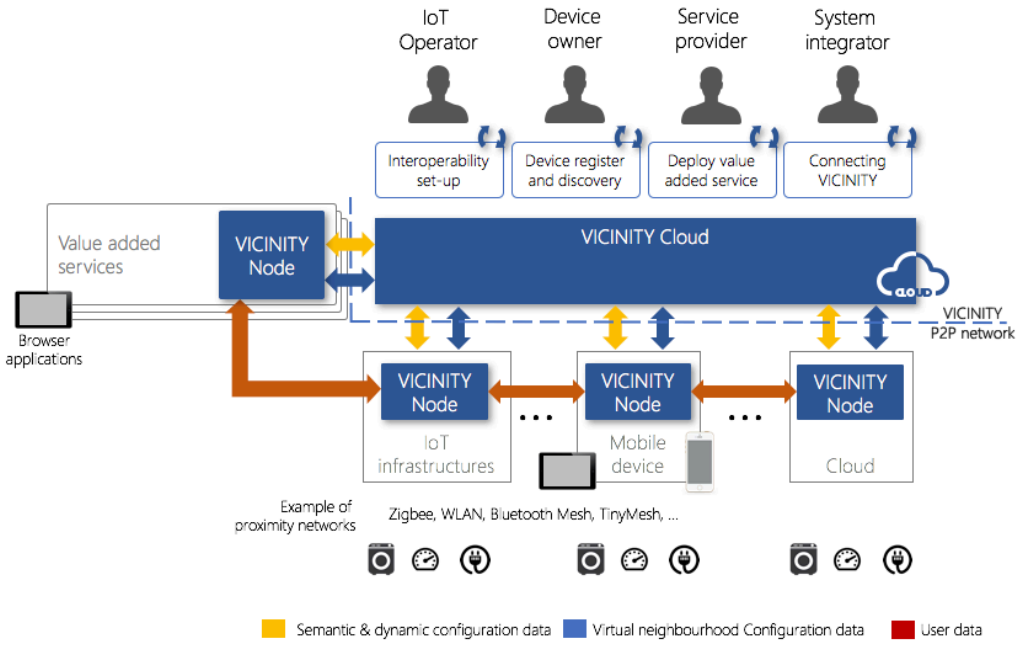

Figure 3 High-level logical VICINITY architecture

On the top of the VICINITY Cloud and VICINITY Nodes the following VICINITY functionality are executed (see D1.5 for detailed description):

- Interoperability setup;
- Device register and discovery;
- Deploy value added service;
- Connecting VICINITY.

2.1.1.VICINITY Cloud

The VICINITY Cloud provides the set of the following services to support VICINITY functionality:

- Configure a virtual neighbourhood of integrated infrastructures and value-added services including the setup of sharing access rules of any IoT object visible in VICINITY through the user-friendly interface (9.1);
- Semantic search of IoT objects in VICINITY virtual neighbourhood;
- Semantic mapping and normalization of new IoT objects to VICINITY IoT ontology;
- Configuration of VICINITY Nodes based on:
    - IoT object description (such as data-integration and privacy services),
    - sharing access rules (configuring of access to IoT objects in P2P network) and
    - configuration of the communication with integrated infrastructure or value-added services (such as encryption and data integrity features);
- Auditing and user notification services of changes and events in virtual neighbourhood (such as new integrated infrastructure request, change of sharing access rules, new device or service in virtual neighbourhood events).

2.1.2.VICINITY Node

A VICINITY Node is the set of software components providing the following set of services for the integration of the IoT infrastructure and/or value-added service into VICINITY:

- Remote IoT object semantic discovery service to look-up for the objects provided by other integrated infrastructures and/or value-added services integrated with VICINITY;
- User data annotation following a semantic format derived from the VICINITY IoT ontology.
- User data forwarding within the P2P network according the share access rules defined in the VICINITY Cloud;
- Encryption and data-integration services for forwarded user data to ensure secure transmission of the data within VICINITY P2P network (9.5);
- Configurable of auditing and logging of exchanged user data.

2.1.3. VICINITY P2P network

The VICINITY P2P network represents the distributed application architecture composed of all VICINITY Nodes that are registered in VICINITY. The connections between such nodes are established on-demand and facilitated by the VICINITY Cloud. That is, whenever a Node requires information from the Network, VICINITY Cloud informs about the candidate peers (Nodes) that may be relevant to establish connection with, in the context of specific information requests. The VICINITY P2P network provides scalable (9.3), closed and secure common communication network for VICINITY Nodes and VICINITY Cloud (i.e. node-to-node and cloud-to-node communication) to exchange:

- User data between VICINITY Nodes based on the share access rules defined in VICINITY Cloud services;
- Control and configuration (such as encryption and privacy features mechanism, control communication channels within P2P network, configure communication between VICINITY Nodes and integration infrastructures) messages between VICINITY Nodes and VICINITY Cloud.

2.2. VICINITY Architecture use case scenarios

This section describes the different architectural scenarios on how VICINITY supports interoperability between different IoT infrastructures and value-added services.

2.2.1. Local application accessing IoT objects connected to smart hub of different IoT infrastructure

This architecture use case scenario interconnects two IoT infrastructures through VICINITY using VICINITY Nodes. Both infrastructures are integrated to the VICINITY through respective VICINITY Nodes. Regardless of the location of integrated IoT infrastructures (i.e. same or different building, same communication network), they are communicating through the VICINITY P2P network. VICINITY Node communicates with Smart infrastructures with SmartHUB (such open-M2M platforms) or IoT devices (depending on implementation).

Goal of use case: Application running in Smart Infrastructure A can access IoT objects connected to Smart infrastructure B as part of its own infrastructure.

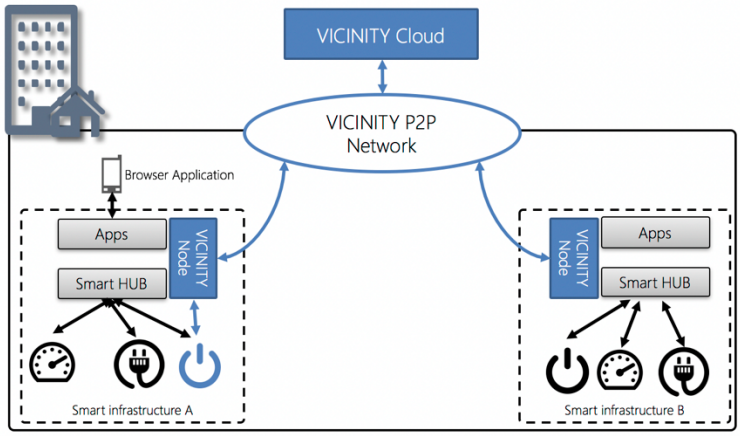

Figure 4 Local application accessing IoT objects connected to smart hub of different Vendor

Discovery: Browser application discovers IoT objects connected to Smart Infrastructure A locally. Browser application discovers IoT objects connected to Smart Infrastructure B locally (using VICINITY discovery semantic services provided by VICINITY Node).

Data access: Browser application accesses data streams from IoT objects connected to Smart Infrastructure A locally (without VICINITY Node). Browser application accesses data streams from IoT objects connected to Smart Infrastructure B locally (using VICINITY Node and VICINITY P2P network).

2.2.2. Local application accessing IoT objects connected to smart hub of different IoT infrastructure through Cloud services

This architecture use case scenario interconnects two IoT infrastructures through VICINITY using VICINITY Nodes. One of the infrastructures is integrated on level of Smart HUB, other is integrated on the level of cloud services[4]. Regardless of location of integrated IoT infrastructures (like same or different building, same communication network) they are communicating through VICINITY P2P network.

Goal of use case: Application running in Smart Infrastructure A can access IoT objects connected to Smart Infrastructure B through its cloud services as part of its own infrastructure.

[4] Cloud services can be deployed as public, private or hybrid cloud services.

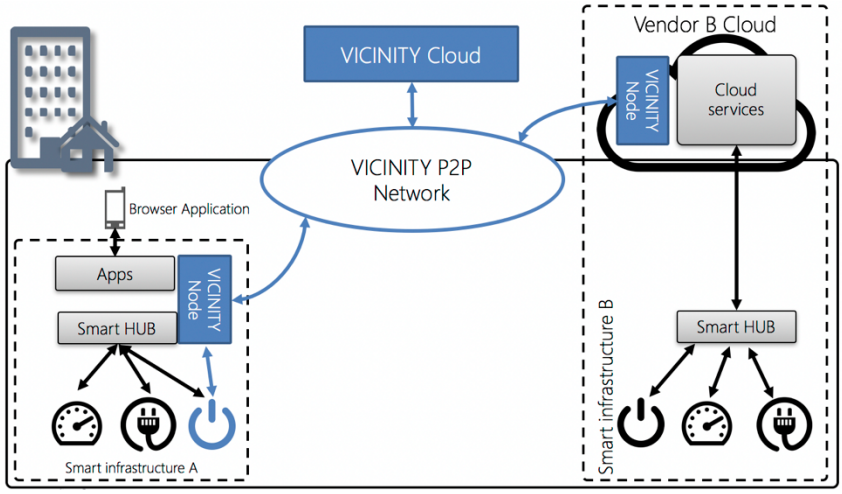

Figure 5 Local application accessing IoT objects connected to smart hub of different Vendor through Cloud services

Discovery: Browser application discovers IoT objects connected to Smart Infrastructure A locally (using infrastructure A discovery services). Browser application discovers IoT objects connected to Smart Infrastructure B locally (using VICINITY discovery semantic services provided by VICINITY Node, VICINITY Node discovers IoT objects using cloud services only).

Data access: Browser application accesses data streams from IoT objects connected to Smart Infrastructure A locally (without VICINITY Node), Browser application accesses data streams from IoT objects connected to Smart Infrastructure B remotely (using VICINITY Node and VICINITY P2P network, VICINITY Node of Smart Infrastructure B access data using cloud services only).

2.2.3. Remote application provided by value-added service accessing IoT objects connected to Smart HUB and/or cloud service

This architecture use case scenario interconnects two IoT infrastructures and Value-added service through VICINITY using VICINITY Nodes. IoT infrastructures are integrated on level of the Smart HUB and the Cloud service.

Goal of use case: Browser application running on value-added service needs access to IoT objects connected Smart Infrastructure A and Smart Infrastructure B.

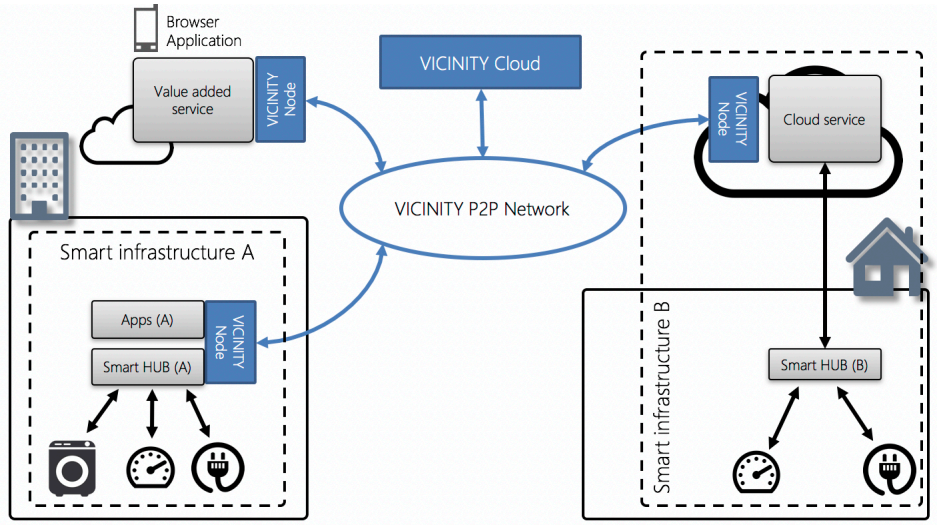

Figure 6 Remote application provided by value-added service accessing IoT objects connected to smart HUB and/or cloud service

Discovery: Browser application discovers IoT objects connected to Smart Infrastructure A and B remotely (using VICINITY Node). Applications running in Smart Infrastructure A or B discover only their IoT objects.

Data access: Browser application accesses data streams from IoT objects connected to Smart Infrastructure A and B remotely (using VICINITY Node and underlying VICINITY P2P network).

3. Logical VICINITY architecture

Based on VICINITY architecture concept and VICINITY Functionality (see Deliverable D1.5) logical VICINITY architecture is decomposed into following components:

- VICINITY Cloud:
  - VICINITY Neighbourhood Manager;
  - Semantic discovery and agent configuration platform;
  - VICINITY Communication Server;
  - VICINITY Gateway API Services;
- VICINITY Node:
  - VICINITY Communication Node;
  - VICINITY Gateway API;
  - VICINITY Agent/ Adapter.

3.1. VICINITY Cloud

3.1.1. VICINITY Neighbourhood Manager

The VICINITY Neighbourhood manager shall provide following service:

- Virtual neighbourhood search for IoT objects, users, organizations;
- IoT objects management;
- Sharing access rules management;
- VICINITY Node configuration management;
- Consent and terms of condition management;
- User and organization management;
- User important notification services;
- User activity auditing.

3.1.2. Semantic discovery and agent configuration

The Semantic discovery and agent configuration platform shall provide:

- Semantic search services;
- IoT objects semantic normalization to VICINITY IoT ontology;
- IoT objects registry services;
- External IoT object template repository crawling and downloading;
- Semantic mapping of IoT objects templates to VICINITY IoT ontology;
- Semantic data changes provision;
- VICINITY Agent configuration provision.

3.1.3. VICINITY Gateway API Services

The VICINITY Gateway API Services shall provide supporting services for VICINITY Gateway API:

- Semantic search of IoT objects in VICINITY neighbourhood services.

3.1.4. VICINITY Communication Server

The VICINITY Communication server shall provide following:

- communication channel setup;

- transaction services between VICINITY Cloud components;
- security services for IoT device authentication
- VICINITY Cloud services forwarding to/from VICINITY P2P network:
    - IoT Objects descriptions;
    - VICINITY Node (auto-) configuration messages.

3.2. VICINITY Node

3.2.1. VICINITY Communication Node

The VICINITY Communication Node shall provide the following functionality:

- data forwarding in VICINITY P2P network;
- data access authorization services;
- data integrity and encryption services;
- privacy filtering services (see 8.2.2);
- VICINITY Configuration service (forwarded from VICINITY Cloud);
- IoT objects registry service (forwarded from VICINITY Cloud).

3.2.2. VICINITY Gateway API

The VICINITY Gateway API shall provide the following functionality:

- Consume Things API service (i.e. to access properties, actions and events of IoT objects through VICINITY);
- Expose Things API service (to expose things from integrated infrastructure into VICINITY);
- Gateway API configuration service to configure consume and expose things APIs;
- IoT objects registry service (from VICINITY Adapter/ Agent).

3.2.3. VICINITY Agent/ Adapter

The VICINITY Agent/ Adapter shall provide the following functionality:

- Integration of IoT Infrastructure or value-added service into VICINITY through VICINITY Gateway API by exposing local IoT objects into VICINITY and simulate supported type of IoT objects from neighbourhood in integrated infrastructure;
- VICINITY Agent/ Adapter configuration service;
- Semantic matching between parametrized demands and available IoT node descriptors.

3.3. How interoperability works in VICINITY

The VICINITY architecture follows an interoperability approach whose main goal is to provide a standard way to both *Discover* and *Access* heterogeneous IoT objects distributed among sparse IoT infrastructures. The approach is based on the work being done by the Web of Things (WoT) WG[5], where a proposal for describing, exposing and consuming *web things* by leveraging Semantic Web technologies is in development. Such web things are things that can be accessed through the Web, either physical or abstract.

[5] https://www.w3.org/WoT/IG/

One of the pillars of the W3C WoT is the Thing Description (TD), which aims to be a standard frame to support describing *web things* semantically to make them interoperable. Thus, TDs are expected to cover the following aspects:

- Semantic meta-data, so to explicitly specify the semantics of a web thing;
- Thing's interaction resources: property, action and event;
- Security including concrete prerequisites to access things are stated;
- Communications, i.e., what kind of protocols and data exchange formats are supported, and which *endpoints* are exposed to give access to the existing interaction resources of a web thing.

VICINITY builds on this standard frame rather to define a brand new one: Thing Ecosystem Description (TED). The purpose of bringing the TED frame into the VICINITY's interoperability scenario is to support describing *ecosystems* of Things, i.e., sets of Things that *coexist* in the same environment. In VICINITY, an IoT infrastructure makes up an ecosystem of IoT objects whose *grounding* environment is the infrastructure itself. However, the VICINITY interoperability approach also considers as ecosystems those sets of IoT objects whose common environment is defined by the scope of a query context for discovery and/or accessing. Such ecosystems made up of query-relevant IoT objects shall be described in what we call Virtual TEDs (VTEDs).

Therefore, VICINITY shall rely on TDs to describe every IoT object (which can represent either physical or abstract Things) that belong to any integrated IoT infrastructure, which in turn shall be described in TEDs. Accordingly, each interested part in taking advantage of the semantic interoperability solution provided by VICINITY must *at least* be able to understand such description frames as well as the Web of Things ontology (see section 4.3.2).

3.3.1. IoT objects' discovery

One of the main challenges of implementing interoperability in the IoT context is to enable consumers to discover, in a distributed and dynamic scenario, those IoT objects that are relevant to their needs but without having any prior knowledge about them. The VICINITY interoperability approach makes the following assumptions so that discovery works:

1. There is a common and abstract information model to semantically describe *Things*;
2. A semantic repository of *Thing* descriptions is available for consumers to search for *Things*;
3. Only those *Things* whose description is included in the semantic repository can be discovered;
4. Consumers learn and leverage the common model and are aware of the semantic repository;
5. Consumers specify their discovery needs as a search criteria that makes use of both the semantic model and the TD frame.

Thus, the same assumptions in VICINITY's terms are as follows:

1. The VICINITY Ontology (see section 4.3.2) is the common and abstract information model to be used;
2. The Semantic discovery & dynamic configuration agent platform (3.1.2) is the semantic repository. The W3C Web of Things TD is the frame to be used for describing any IoT object integrated in VICINITY.
3. Each time an IoT object is to be properly integrated in VICINITY, either at registration time or after any change in its configuration, a TD must be inserted or updated in the semantic repository.

4. Gateway APIs (section 3.2.2) of VICINITY Nodes are the semantic mediators between the actual consumers, e.g., Adapters, and the repository of TDs. Therefore, they provide an interface for Discovery requests.
5. Gateway APIs must be able to specify discovery needs as *semantic-based* search criteria (SPARQL query).

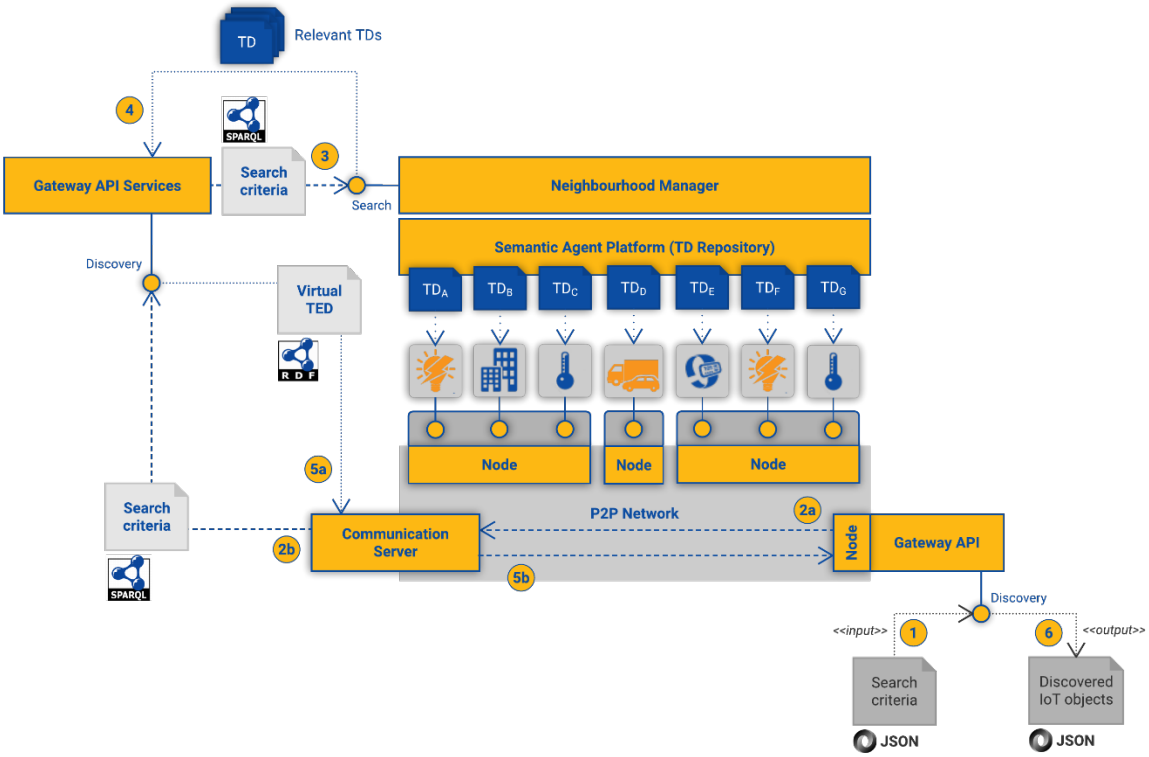

Figure 7 Semantic interoperability approach for Discovery

As described in section 2 and represented in Figure 7, the VICINITY is based on secure P2P communications between VICINITY Nodes for user data transfers. Further, the P2P Network is also the secure channel that supports the discovery information flow between Gateway APIs and the Gateway API Services, through the VICINITY Communication Server. Consequently, VICINITY protects IoT objects from being discovered by those Nodes that are out of their owner's Neighbourhood, implicitly including potential consumers that are not part of VICINITY.

In order to illustrate how discovery works in VICINITY, Figure 5 shows the following sequence of interactions:

1. With the aim of keeping actual consumers agnostic of semantics and ontology details, Gateway APIs shall provide an interface that works at syntactic level, utilizing the common VICINITY format to specify a schema-based search criteria. It is the Gateway API who must transform such request into its corresponding SPARQL query.
2. The newly created SPARQL query that represents the given search criteria is securely transmitted to the Gateway API Services through the P2P Network.
3. Once the Gateway API services component receives a search criteria in the form of a SPARQL query, it is forwarded to the Neighbourhood Manager, which is responsible of applying neighbourhood-filtering to all matching TDs obtained from the TD Repository; keeping only those TDs that represent IoT objects that belong to consumer's virtual neighbourhood.

4. As the Gateway API Services component receives the filtered TDs mentioned in step 3, it encapsulates them into a Virtual TED. Such VTED describes the virtual ecosystem that puts in conjunction the TDs that matches the underlying search criteria.
5. The Gateway API Services component sends the newly created VTED to the requester through the P2P Network.
6. After the VTED is intercepted by the Gateway API and properly processed, the set of discovered IoT objects along with all relevant identifiers and meta-data is extracted and returned, in the common VICINITY format, to the actual consumer.

3.3.2. IoT objects' access

As mentioned at very beginning of this section, the second goal of semantic interoperability in VICINITY is to enable accessing to heterogeneous IoT objects in such a way that any of their interaction resources can be effectively *consumed*. Here, effectiveness implies that the information provided and/or required by these interaction resources must not only be collected but also understood.

The present interoperability approach for consuming IoT objects assumes the following:

1. All *endpoint* declarations that are included in TDs for accessing interaction resources must use schema-based representations (syntactic interoperability) for the data they provide/require;
2. TDs should contain the required *access mappings* to explicitly specify the semantics of data to be exchanged with the endpoints. By applying such access mappings, schema-based data becomes semantic data (data *lifting*[6]);
3. Context enrichment of TDs may include concepts and predicates from vocabularies to be used to specify semantics in those access mappings.

Again, in VICINITY's terms:

1. The common VICINITY format shall be used as the schema to represent the data provided/ required by endpoints.
2. TDs of IoT objects may include predefined access mappings for each declared endpoint that gives access to the corresponding IoT object's data. If necessary, at the time of the discovery process, the Gateway API Services may dynamically attach query-relevant access mappings for each TD included in the resulting Virtual TED.
3. Since the VICINITY Ontology aims to cover different level of abstractions (e.g., core, domain-specific, etc.), any concept or predicate, at any abstraction level, may be used to enrich the context of the TDs.

[6] Lifting of schema-based data to reach semantically structured and interlinked data, e.g., from plain JSON to RDF.

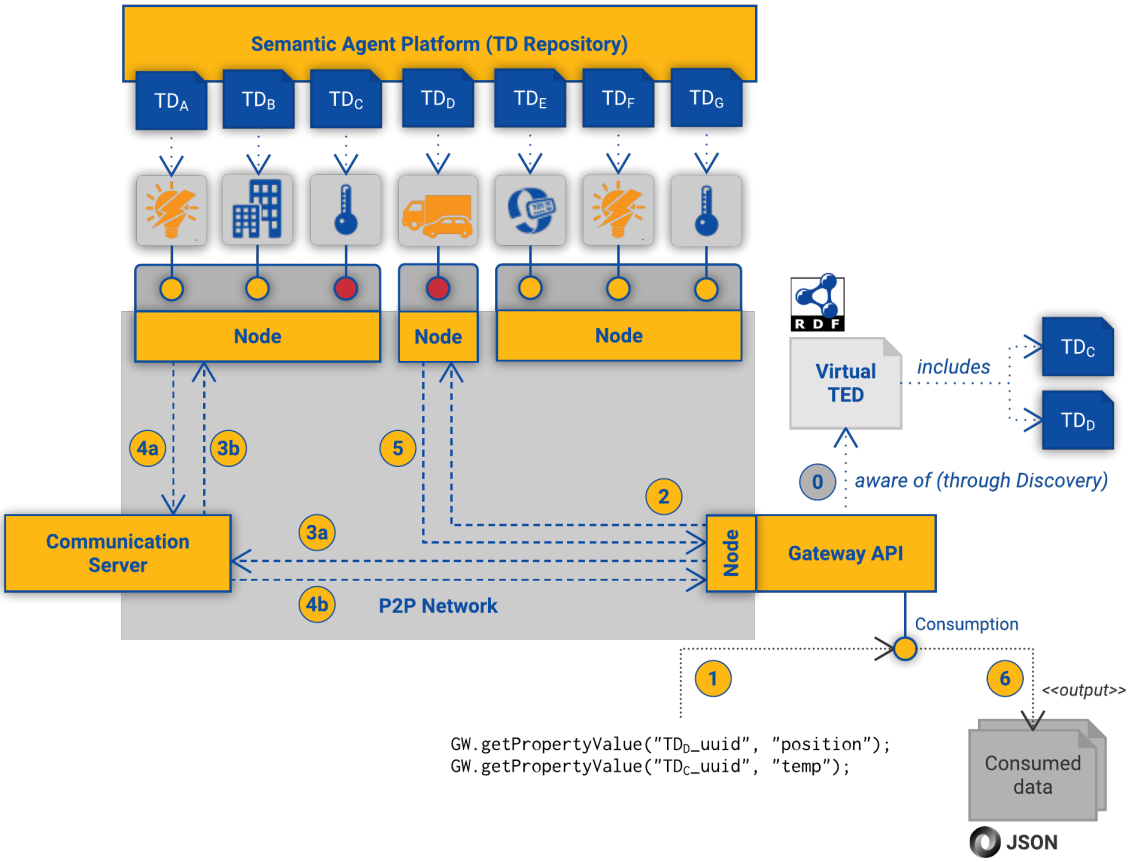

Figure 8 Semantic interoperability approach for Accessing

Figure 8 illustrates an example of how VICINITY shall support semantic interoperability for *consumption* of IoT objects, concretely, for getting property values. Similarly, to how discovery was illustrated in Figure 7, this diagram describes how the approach works by means of a sequence of interactions:

1. The Gateway API shall offer a dedicated interface for Consumption requests, which in the example, receives two *getPropertyValue* commands pointing at different IoT objects (T_C and T_D). It is assumed that the Gateway API was previously given a Virtual TED through a discovery request (step 0). Thanks to this, the consumer can use the corresponding identifiers of each IoT object and their properties at the time of invoking both commands. Again, the Consumption interface allows consumers to be agnostic of ontologies.
2. The Gateway API processes the first command and sends a message to the Node that hosts the referenced IoT object. In this case, the P2P communication can be directly established between the two involved Nodes.
3. As in the previous step, the Gateway API processes the received command, but in this case, the recipient Node is not directly reachable due to some network constraint in-between. The Communication Server takes care of the issue and makes sure that the message finally reaches its addressed Node.
4. Once the corresponding Gateway API of the recipient Node gets the message, it queries the TED that describes its own infrastructure to determine the specific endpoints that must be invoked. Having all raw data collected from its underlying infrastructure, the recipient

Gateway API composes a response message and sends it back to the requester (through the Communication Server).

5. Same process as described in step 4, except for the P2P communication does not require an intermediary.

The Gateway API processes both incoming response messages separately and applies the corresponding access mappings specified in the previously given Virtual TED. For each response and after the data *lifting* process is completed, the Gateway API extracts the property value by querying the just lifted semantic data. Finally, the Gateway API returns each result obtained and represented in the common VICINITY format.

4. Information view

This section describes the flow of information between VICINITY components as summarized on the following Figure 9. The information flows can be grouped in following groups:

- Semantic data flows including semantic discovery and query, IoT object descriptions, virtual IoT object descriptions and templates;
- Configuration data flows including IoT object meta-data and various VICINITY configurations;
- Syntactic data flows for transmission of user data.

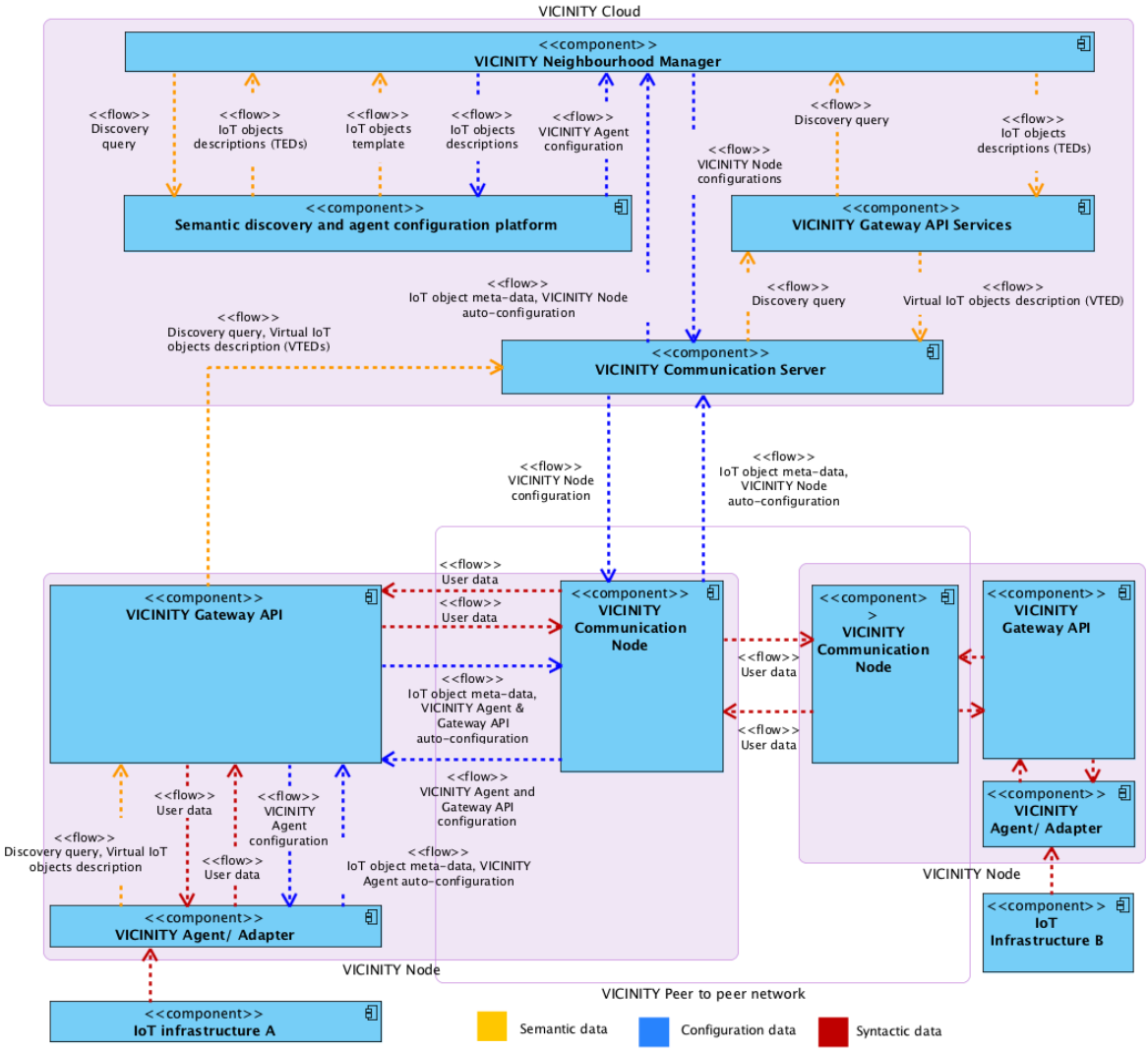

Figure 9 VICINITY Architecture information flow

4.1. VICINITY Cloud

4.1.1. VICINITY Neighbourhood manager

The VICINITY Neighbourhood manager shall facilitate the following information flows:

Table 1 Information flow from VICINITY Neighbourhood manager

Information flow	Destination	Reason
Discovery query	Semantic discovery and dynamic configuration platform	Virtual neighbourhood manager uses Discovery query to search for Things in neighbourhood.
IoT objects descriptions (TDs)	Semantic discovery and dynamic configuration platform	Semantic discovery and dynamic configuration platform needs IoT object description to perform semantic lifting on IoT object description.
IoT objects descriptions (TDs)	VICINITY Gateway API Services	VICINITY Gateway API Services needs set of IoT objects description (as result of Discovery query) constrained by virtual neighbourhood sharing access rules.
VICINITY Node configurations	VICINITY Communication Server	VICINITY Communication Server distributes configuration to respective VICINITY Nodes.

4.1.2. Semantic discovery and agent configuration

Semantic discovery and agent configuration shall facilitate the following information flows:

Table 2 Information flow from Semantic discovery and agent configuration

Information flow	Destination	Reason
IoT objects templates	VICINITY Neighbourhood Manager	VICINITY Neighbourhood Manager enables to configure IoT objects manually by user.
IoT objects descriptions (TDs)	VICINITY Neighbourhood Manager	VICINITY Neighbourhood Manager needs TDs as result of the discovery queries and semantic lifting.
VICINITY Agent configurations	VICINITY Neighbourhood Manager	VICINITY Neighbourhood Manager shall forward an Agent configuration to VICINITY Node.

4.1.3. VICINITY Gateway API Services

VICINITY Gateway API Services shall facilitate the following information flows:

Table 3 Information flow from VICINITY Gateway API Services

Information flow	Destination	Reason
Discovery query	VICINITY Neighbourhood Manager	VICINITY Neighbourhood Manager forwards the discovery query to semantic platform.
Virtual IoT objects description (VTEDs)	VICINITY Communication Server	VICINITY Communication Server forwards VTED to VICINITY Gateway API.

4.1.4. VICINITY Communication Server

The VICINITY Communication server shall facilitate the following information flows:

Table 4 Information flow from VICINITY Communication Server

Information flow	Destination	Reason
Discovery query	VICINITY Gateway API Services	VICINITY Communication Server forwards Discovery query to VICINITY Gateway API Services.
VICINITY Node configurations	VICINITY Communication Node	VICINITY Communication Server shall forward VICINITY Node configuration to relevant VICINITY Nodes.
VICINITY Node auto-configurations	VICINITY Neighbourhood Manager	VICINITY Communication Server shall forward auto-configuration of VICINITY Node for VICINITY User acknowledgement.
IoT object meta-data	VICINITY Neighbourhood Manager	VICINITY Communication Server shall forward to VICINITY Neighbourhood Manager new IoT object description received from VICINITY Nodes.

4.2. VICINITY Node

4.2.1. VICINITY Communication Node

The VICINITY Communication Node shall facilitate the following information flows:

Table 5 Information flow from VICINITY Communication Node

Information flow	Destination	Reason
User data	VICINITY Communication Node, VICINITY Gateway API	To exchange of user data through P2P Network
VICINITY Node auto-configurations	VICINITY Gateway API	To forwards Agent and Gateway API configuration updates

Information flow	Destination	Reason
VICINITY Node auto-configurations	VICINITY Communication Server	To forward auto-configuration to VICINITY Neighbourhood Manager for processing
IoT object meta-data	VICINITY Communication Server	To forward description for semantic mapping
IoT object descriptions (VTED)	VICINITY Gateway API	To forward VTEDS to configure APIs

4.2.2. VICINITY Gateway API

The VICINITY Gateway API shall facilitate following information flows:

Table 6 Information flow from VICINITY Gateway API

Information flow	Destination	Reason
Discovery query	VICINITY Communication Server	To forward discovery query to VICINITY Neighbourhood Manager.
User data	VICINITY Communication Node, VICINITY Agent/Adapter	To forward user data through P2P Network.
VICINITY Agent & Gateway API auto-configurations	VICINITY Communication Node	To forward Agent and Gateway API configuration to VICINITY Neighbourhood Manager to setup interoperability during initialization process.
VICINITY Agent configuration	VICINITY Agent	To forward Agent configuration updates to VICINITY Agent.
IoT object meta-data	VICINITY Communication Node	To forward new IoT object description for semantic mapping in semantic platform.

4.2.3. VICINITY Agent/ Adapter

The VICINITY Agent/ Adapter shall facilitate the following information flows:

Table 7 Information flow from VICINITY Agent/ Adapter

Information flow	Destination	Reason
Discovery query	VICINITY Gateway API	To discover the virtual neighbourhood.
User data	VICINITY Gateway API	To forward user data through P2P Network.

Information flow	Destination	Reason
VICINITY Agent auto-configurations	VICINITY Gateway API	To forward initial Agent configuration to VICINITY Neighbourhood Manager to initialize interoperability setup.
IoT object meta-data	VICINITY Gateway API	To forward new IoT object description for semantic mapping in semantic platform.

4.3. Data storages

This section describes the concept of information management in the VICINITY. The following type of storages has been identified:

- Global neighbourhood storage storing virtual neighbourhood data including social network of organizations, users, IoT objects including sharing rules and profile[7]s. This storage includes VICINITY Node components configuration as well;
- Semantic Model and Agent Configuration Storage encompasses semantic description of each IoT objects and their relationships.

These storages are maintained by two core VICINITY components:

- VICINITY Neighbourhood Manager masters global neighbourhood storage;
- Semantic discovery & dynamic configuration agent platform masters semantic model & agent configurations.

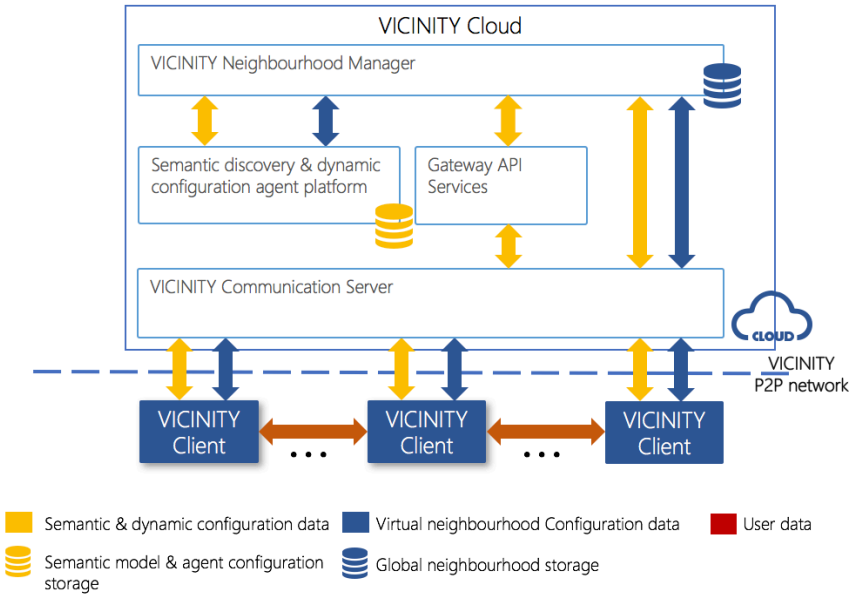

Figure 10 VICINITY Data storages

[7] The profile is set of entity (IoT object, user, organization) properties configurable in VICINITY Neighborhood manager. Sub set of IoT object profile is IoT object description.

Data from storages are distributed with the VICINITY architecture by executing VICINITY processes described in section 5 through defined information flows defined in 4.1 and 4.2.

4.3.1. Global neighbourhood storage

The Global neighbourhood storage managed by VICINITY Neighbourhood Manager includes following items:

- User and Organization;
- IoT objects references (devices and value-added services) and Group of IoT objects;
- Security Access Rules;
- VICINITY Node configurations;

4.3.1.1. *User and Organization*

VICINITY-ARCH-INF-010	**User and Organization entities**

Global neighbourhood storage shall include Organization with assigned Users registered in VICINITY.

Users (human and non-human) entity includes:

- Identifier,
- credentials,
- profile,
- set of roles within related organization.

Organization entity includes:

- organization profile,
- set of references to assigned users,
- set of references to owned VICINITY Nodes,
- set of references to owned groups of IoT Objects,
- set of references to assigned sharing access rules.

Considered requirements:

VICINITY-NFUNC-PRV010

4.3.1.2. *Sharing Access Rules*

VICINITY-ARCH-INF-030	**Sharing Access Rules**

Global neighbourhood storage shall include Sharing Access Rules which defines single sharing decision of IoT object virtual neighbourhood.

Sharing Access Entry shall reference:

- Reference to Organization to which sharing access is granted;
- Sharing access privilege;
- Reference to IoT object (including property, action and event) or Group of IoT objects.

Considered requirements:

VICINITY-NFUNC-PRV010, VICINITY-FUNC-UCR020, VICINITY-FUNC-UCR070,

4.3.1.3. *VICINITY Node Configurations*

VICINITY-ARCH-INF-040	VICINITY Nodes Configurations

Global neighbourhood storage shall include each VICINITY Node, which is referenced to organization and IoT Objects.

VICINITY Nodes shall include set of actual configurations of:

- VICINITY Communication Node;
- VICINITY Gateway API;
- VICINITY Agent / Adapter.

Configuration shall include:

- Identity of the configuration's source (e.g. VICINITY Neighbourhood Manager, Semantic discovery and agent configuration platform);
- Identity of the configuration's destination;
- Configuration identifier;
- Configuration data;
- Data integrity attributes (optionally).

Considered requirements:

VICINITY-NFUNC-PRV010

VICINITY-ARCH-INF-045	VICINITY Node configuration acknowledgement

The configuration update acknowledgement shall include:

- Identity of the configuration's source;
- Identity of the configuration's destination;
- Configuration identifier;
- Configuration ACK;
- Data integrity parameters (optionally).

Considered requirements:

VICINITY-NFUNC-SEC050

4.3.2. Semantic Model and Agent Configuration Storage

The storage includes the models of all objects able to interact with VICINITY agents/adapters. The purpose of this storage is to provide the semantic descriptions of stored items to enable the machine-readable semantic interpretation. Semantic enhancement of object descriptions also enable to perform several querying and lookups in semantic way.

The storage will be implemented as high-performance semantic triplestore providing standard semantic search services, such as SPARQL endpoint.

Generally, the storage includes two basic items: semantic meta-models and semantic representation of IoT objects (TEDs).

VICINITY-ARCH-INF-050	Semantic meta-models

Semantic meta-models: the upper ontologies, hierarchies of objects, types, domain specific models. Generally, the models providing basic semantic context of IoT objects. All semantic instances are mapped to these meta-models to enable common automatic interpretation of semantic information.

Considered requirements:

VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR120

Semantic description of objects, which reside in infrastructures, that are integrated into VICINITY platform. Usually, the IoT object provides the set of services for interaction. It can be physical device or sensor, but in terms of VICINITY, it can be also the value-added service. The purpose of semantic enrichment is to enable performance semantic queries/lookups, but also automatic interpretation of object properties and services.

VICINITY-ARCH-INF-060	IoT object description

The IoT object description of IoT object contains:

- the identifier of IoT object;
- description of interaction with IoT object: the set of services, actions or events provided by device;
- description of grounding information of device interaction possibilities (e.g. endpoints, protocols, input/output parameter description);
- additional properties enabling more precise semantic interpretation, such as capabilities (e.g. has display or measures temperature);
- the manufacturer and model information in case if IoT object is physical device.

Considered requirements:

VICINITY-FUNC-UCR070, VICINITY-FUNC-UCR110

There are two types of IoT object instances:

- IoT object templates: preconfigured models for each specific IoT object types, generally describing the objects (e.g. specific device model). These templates are used in discovery and configuration process to create mapping between physical IoT objects and corresponding models. For each specific IoT object there exists the generic IoT object template. For each IoT object, the discovery process finds the template, which is used to create specific IoT object instance.
- IoT object description: the role of discovery process is to find the proper object template for each physical IoT object. Once this matching is successful, the new unique instance for each physical IoT object is created from identified template and mapping is created. Thus, for each physical IoT object there exists its own corresponding instance in semantic repository.

Generally, there are two different descriptors of IoT objects:

- the IoT object meta-data: provided by VICINITY Agent, containing basic information about the object used to find the corresponding IoT object template

- IoT object template: representing generic model of specific IoT object type (e.g. model of concrete sensor).

4.3.2.1. *VICINITY semantic models*

In the information technologies context, ontologies can be defined as "formal, explicit specifications of a shared conceptualisation". The VICINITY ontologies will be formal in the sense of following Description Logics and being implemented in the W3C Web Ontology Language standard OWL[8]. The conceptualization to be shared among the VICINITY components and plugged systems will cover different domains of interest ranging from horizontal domains like time and space to specific definitions need within the VICINITY ecosystem. For this reason, as shown in Figure 11, the VICINITY approach is based on a modular ontology network in which existing standard ontologies will be reused whenever possible.

In summary, the ontology network will be composed by: 1) cross-domain ontologies (horizontal domains) addressing the modelling of general concepts like time, space, web things, that would be reused and probably extended by 2) the VICINITY platform oriented ontology that will represent the information needed to exchange IoT descriptor data between peers and that would be extended by 3) domain oriented ontologies that would cover vertical domains such as health, transport, buildings, etc.

[8] https://www.w3.org/TR/owl-ref/

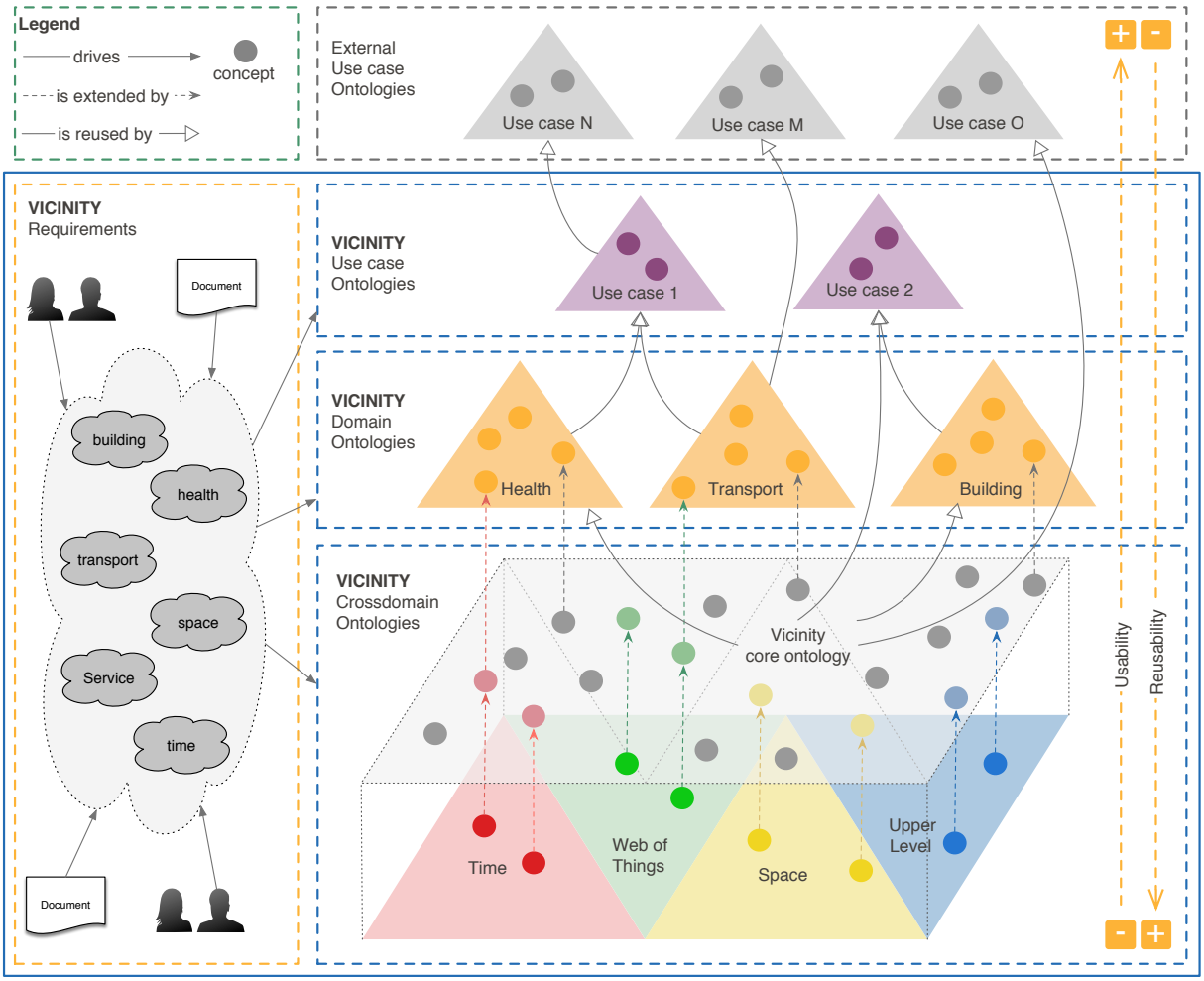

Figure 11 VICINITY ontology network design

As it is also shown in the Figure 11, VICINITY ontological modules might be reused by external use cases or applications. As depicted in the left side of the figure, the ontological requirements that will guide the ontology development will come from VICINITY partners needs and VICINITY documentation.

Apart from the domain-specific ontological requirements, the VICINITY ontologies development is based on the following non-functional requirements:

- Reuse: existing ontologies or standard models will be reused when possible increasing interoperability with external systems that might be already using such ontologies. This point is also applied at a meta-level by using standard technologies to implement the ontologies themselves.
- Modularity: the ontology should be designed as a network in which modules might be interconnected and refer to others.
- Extensibility: the ontologies should allow the development of third-party extensions.

Finally, the ontologies will be developed following methodologies and best practices commonly used in ontological engineering to address ontology development activities such as design, implementation, evaluation, publication, and documentation, among others.

5. Process view

This chapter defines the set of processes driving interaction between VICINITY components supporting the following functionality provided by VICINITY described in D1.5.

- Device register and discovery;
- Deploy value-added services;
- Interoperability setup;
- Connecting IoT infrastructure into VICINITY;
- VICINITY Security & privacy features;
- VICINITY User notification;

VICINITY-ARCH-PRC-010	VICINITY Cross component processes:

The VICINITY shall support functionalities by the following cross component processes:

- **Connecting VICINITY** (5.1.1) supports setup of communication channels between VICINITY Nodes' components and VICINITY Cloud;
- **VICINITY Node configuration distribution** (5.1.2) manages distribution of any configuration change (such as IoT object lifecycle change in VICINITY, sharing access rule change, security and privacy configuration change, agent configuration change) originated in the VICINITY Cloud to relevant VICINITY Nodes components;
- **Semantic discovery of IoT Objects** (5.1.3) provides IoT objects registration processes within VICINITY, including IoT objects semantic normalization and IoT objects search within VICINITY neighbourhood;
- **User data exchange facilitation** (5.1.4) provides processes to exchange data between VICINITY Nodes utilizing the semantic interoperability.

Considered requirements:

VICINITY-FUNC-UCR060, VICINITY-FUNC-UCR090, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR130, VICINITY-FUNC-UCR140, VICINITY-FUNC-UCR145

This chapter focuses only on processes spanning across more than one component. Significant single component processes (functions) are subject of detail design.

5.1.1. Connecting VICINITY

The Connecting VICINITY process goal is to setup communication between the VICINITY Node components and VICINITY Cloud component including configuration of virtual neighbourhood and agent platform model. The communication is setup manually with optional automatic pre-configuration.

VICINITY-ARCH-PRC-020	VICINITY Node communication setup

The VICINITY shall support manual setup communication interfaces of VICINITY Node components with VICINITY Cloud components:

- VICINITY Communication Node,
- VICINITY Gateway API,
- VICINITY Agent/Adapter.

by System integrator.

Considered requirements:
VICINITY-FUNC-UCR140

VICINITY-ARCH-PRC-030	**VICINITY Node auto-configuration**

The VICINITY shall support automatic auto-configuration of communication interfaces setup of VICINITY Node components:

- VICINITY Communication Node,
- VICINITY Gateway API,
- VICINITY Agent/Adapter

by System integrator.

Considered requirements:
VICINITY-FUNC-UCR140

VICINITY-ARCH-PRC-040	**VICINITY Node configuration store**

The VICINITY shall store VICINITY Node components identities and its configuration in virtual neighbourhood and agent platform.

Considered requirements:
VICINITY-FUNC-UCR140, VICINITY-NFUNC-PER050

VICINITY-ARCH-PRC-050	**VICINITY Node registration request**

The VICINITY Node registration request should include:

- Identity of the registration request;
- Identity of user under which registration is performed;
- Identity of VICINITY Communication Node;
- Identity of VICINITY Gateway API;
- Identity of VICINITY Agent/Adapter;
- Initial configuration of each component including at least:
  - Component identifier (if available);
  - Identifier of component's interfaces.

Considered requirements:
VICINITY-FUNC-UCR140

5.1.1.1. *Manual VICINITY Node registration*

The manual registration of VICINITY Node is performed by System integrator through VICINITY Neighbourhood Manager (Figure 12). VICINITY Neighbourhood Manager registers VICINITY Nodes in agent platform and performs VICINITY Node configuration change process to distribute configuration to VICINITY Nodes components.

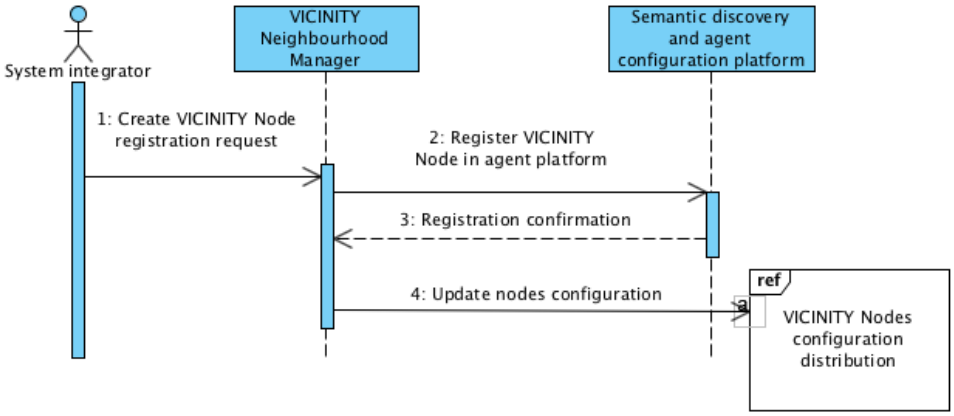

Figure 12 Manual VICINITY Node registration process

5.1.1.2. *Manual VICINITY Node removal*

The manual removal of VICINITY Node is performed by IoT Operator through VICINITY Neighbourhood Manager (Figure 13). VICINITY Neighbourhood Manager sends removal request to VICINITY Node. When the removal of VICINITY Node from P2P network is acknowledged by VICINITY Communication Server then VICINITY Node is removed from the Semantic discovery and agent configuration platform.

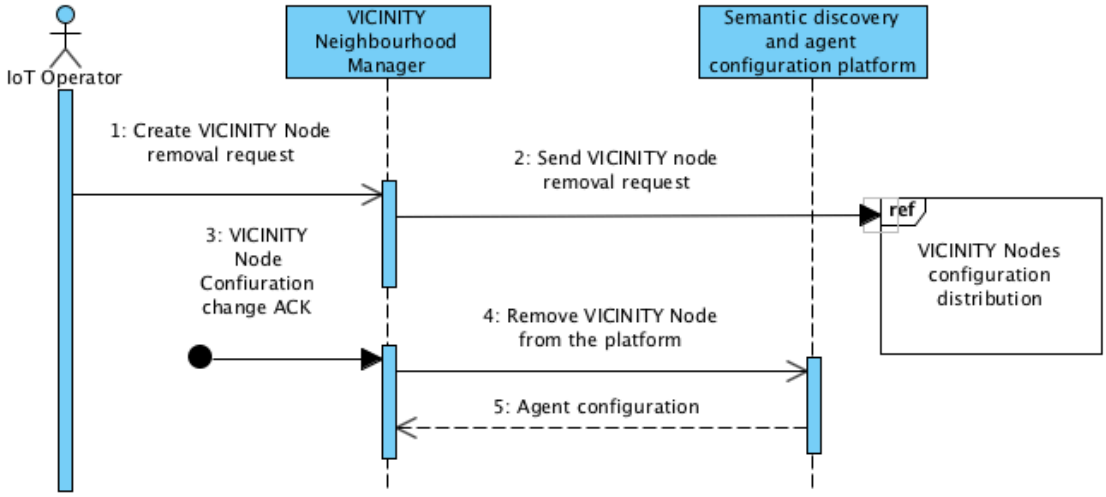

Figure 13 Manual VICINITY Node removal

5.1.1.3. *Auto-configuration of VICINITY Node*

The auto-configuration of VICINITY Node initiates registration process of VICINITY Nodes from VICINITY Agent, through VICINITY Gateway API to VICINITY Communication Node (Figure 14). After a request arrives in VICINITY Neighbourhood Manager the manual registration process shall be executed after acknowledgement of a System integrator, where System integrator can validate and/or update VICNITY Node configurations. The auto-configuration process is an extension of the manual registration process.

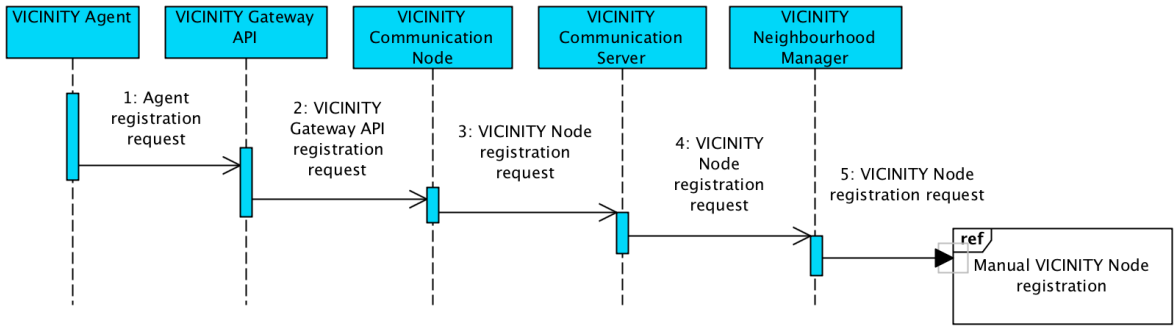

Figure 14 Auto-configuration of VICINITY Node

5.1.2. VICINITY Node configuration distribution

The VICINITY goal of VICINITY Node configuration distribution process is to distribute the VICINITY configuration changes to relevant VICINITY Node. Every configuration change needs to be acknowledged by the configuration destination component.

VICINITY-ARCH-PRC-060	**VICINITY Node configuration distribution**

The VICINITY shall support distribution of the VICINITY Node configuration (4.3.1.3) to:

- VICINITY Communication Node,
- VICINITY Gateway API,
- VICINITY Agents

through VICINITY Communication server.

Considered requirements:

VICINITY-FUNC-UCR060, VICINITY-FUNC-UCR090, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR130, VICINITY-FUNC-UCR140

VICINITY-ARCH-PRC-090	**VICINITY Node configuration processing**

The VICINITY Node's Component shall validate VICINITY Node configuration before its processing.

The VICINITY Node's Component forwards only configuration attributes relevant for destination component.

Each VICINITY Node's component shall send VICINITY Node configuration updated acknowledgement after successful processing of configuration update.

Considered requirements:

VICINITY-FUNC-UCR060, VICINITY-FUNC-UCR090, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR130, VICINITY-FUNC-UCR140

5.1.2.1. *Configuration update distribution*

This process transmits the IoT object configuration update from VICINITY Neighbourhood Manager to VICINITY Communication Node, and optionally to VICINITY Agent/Adapter and VICINITY Gateway API. The Configuration is distributed through VICINITY's P2P Network to VICINITY Node components. Afterwards, it is processed by VICINITY Node's components in cascade manner. Each VICINITY Node's

component acknowledges the process of its configuration separately in case of configuration was provided through the configuration distribution process.

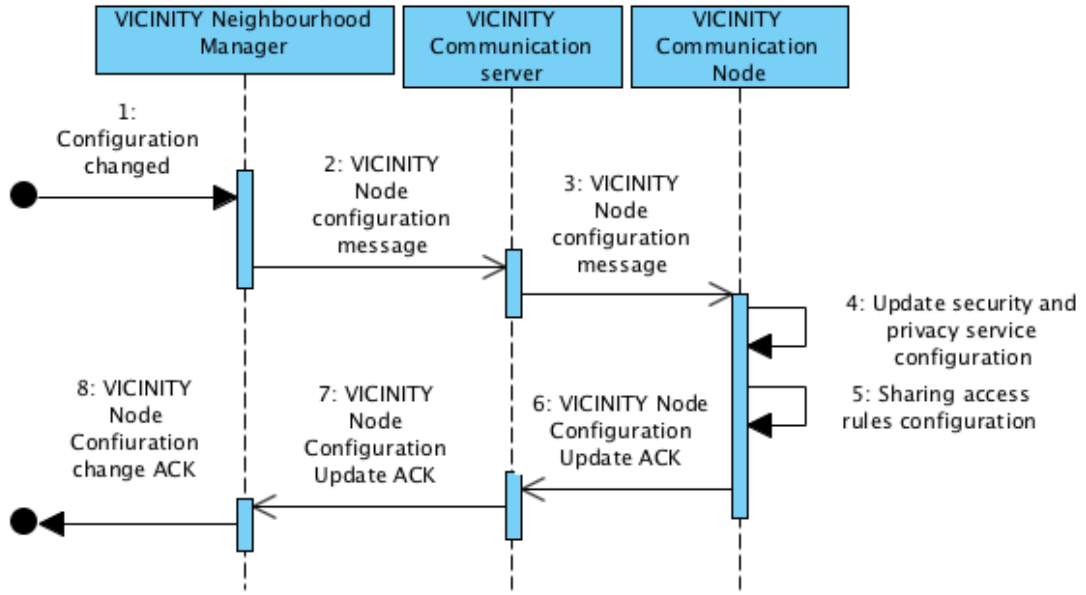

Figure 15 VICINITY Node configuration distribution process

5.1.3. Semantic discovery of IoT objects

The goal of the Semantic discovery of IoT objects is twofold:

- to search of IoT objects in the VICINITY neighbourhood manager with support of the semantic data stored in Semantic discovery and Agent configuration platform;
- to maintain semantic data of VICINITY neighbourhood manager including agent configuration.

VICINITY-ARCH-PRC-110	**Search of VICINITY Neighbourhood**

The VICINITY shall provide search in vicinity neighbourhood manager to discover IoT objects based on request from:

- VICINITY Neighbourhood manager requested by VICINITY Users;
- VICINITY Gateway API requested by VICINITY Adapter or VICINITY Agent.

The VICINITY shall constrain search by applying of sharing access rules based on identity resulted from discovery search.

Considered requirements:

VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR120, VICINITY-FUNC-UCR160

VICINITY-ARCH-PRC-120	**Manual IoT object registration**

The VICINITY shall support registration of new IoT objects based on request from:

- VICINITY Neighbourhood manager requested by VICINITY Users.

VICINITY-ARCH-PRC-120	Manual IoT object registration

The new IoT object requested from VICINITY Gateway API shall be acknowledged by VICINITY User.

VICINITY shall store IoT object description (TD) in semantic model and distribute IoT object configuration to VICINITY Node components.

Considered requirements:

VICINITY-FUNC-UCR060, VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR120

VICINITY-ARCH-PRC-130	IoT object registration from VICINITY Adapter/Agent

The VICINITY shall optionally support registration of new IoT objects based on request from:

- VICINITY Gateway API requested by VICINITY Adapter or VICINITY Agent.

The new IoT object requested from VICINITY Gateway API shall be acknowledged by VICINITY User.

Considered requirements:

VICINITY-FUNC-UCR060, VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR120

VICINITY-ARCH-PRC-140	IoT object descriptors

The VICINITY shall support periodically search of external IoT descriptors repository for new IoT object descriptors.

Each new IoT object descriptor shall be mapped to VICINITY Ontology and relevant VICINITY Agent should be configured to be able to discover IoT objects implementing the descriptor.

Considered requirements:

N/A

5.1.3.1. *Search for IoT objects in VICINITY*

These processes perform the search of IoT objects based on request from:

- VICINITY Neighbourhood Manager performed by VICINITY Users (Figure 15);
- VICINITY Gateway API performed by VICINITY Agents and/ or VICINITY Adapter (Figure 16).

Search for IoT objects can be performed by VICINITY User directly in VICINITY Neighbourhood Manager which utilizes semantic data search request and returns the set of IoT object descriptions (Figure 15).

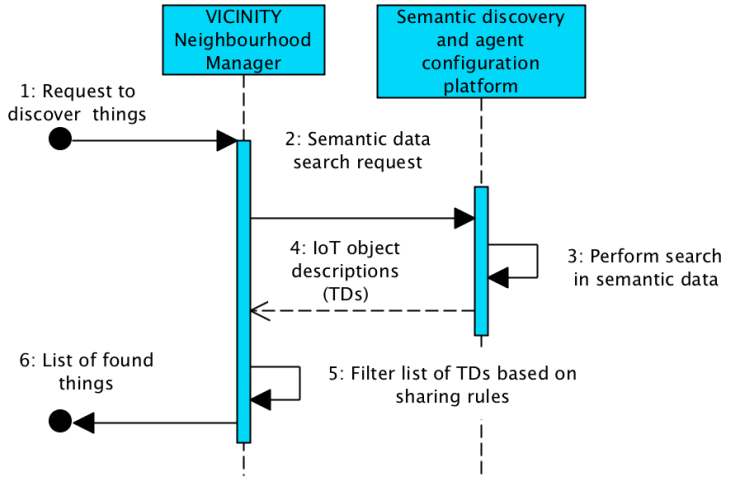

Figure 16 Semantic search of IoT objects from VICINITY Neighbourhood Manager

However, VICINITY Gateway API provides discovery service for VICINITY Agents / Adapters. The VICINITY Gateway API utilizes VICINITY Gateway API Services through VICINITY Communication Server. The VICINITY Communication Server acts as communication border between VICINITY Node components and VICINITY Cloud. Afterwards, request is forwarded through VICINITY Neighbourhood Manager, which applies sharing access rules to result of the semantic data search results (List of TDs) based on context of identity under which search is performed.

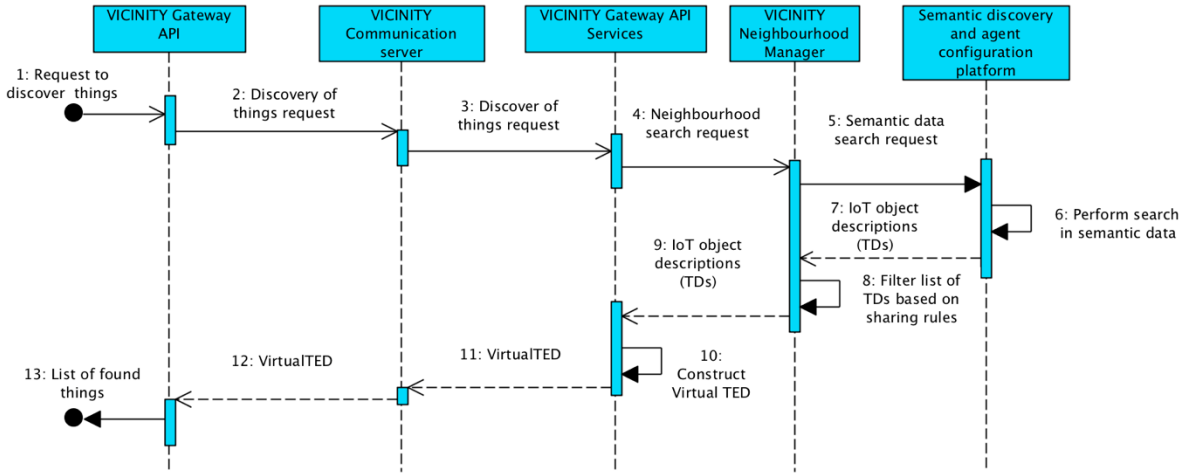

Figure 17 Semantic search of IoT objects from VICINITY Gateway API

5.1.3.2. *Manual IoT Object registration, change of configuration and removal*

Any manual change of IoT object (registration, configuration change, removal) is performed by VICINITY User through VICINITY Neighbourhood Manager. VICINITY Neighbourhood Manager sends an IoT Object change in Semantic discovery platform and update relevant VICINITY Nodes with sharing access rules utilizing VICINITY Node configuration change process.

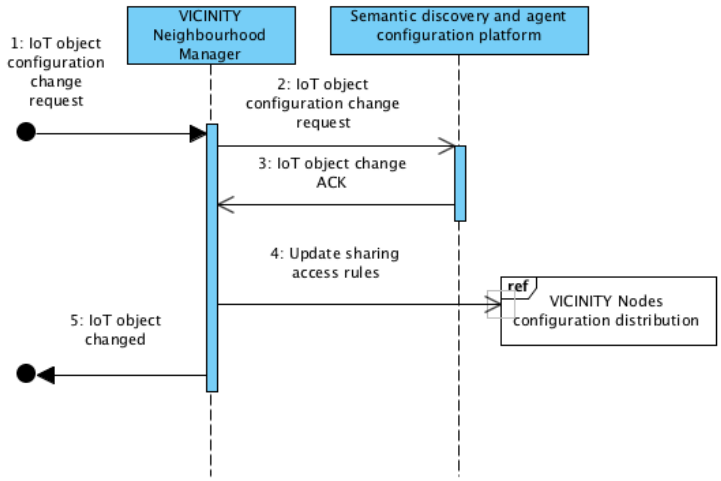

Figure **18 Manual change of IoT object**

5.1.3.3. *Automatic registration of new IoT Objects*

This process enables to register new IoT objects by VICINITY Agent / Adapter. Requests for new IoT objects are sent to VICINITY Neighbourhood Manager for configuration and approval by VICINITY User (Device owner or Service provider). IoT objects descriptions shall be stored in semantic model and IoT object configuration should be forwarded to VICINITY Node components.

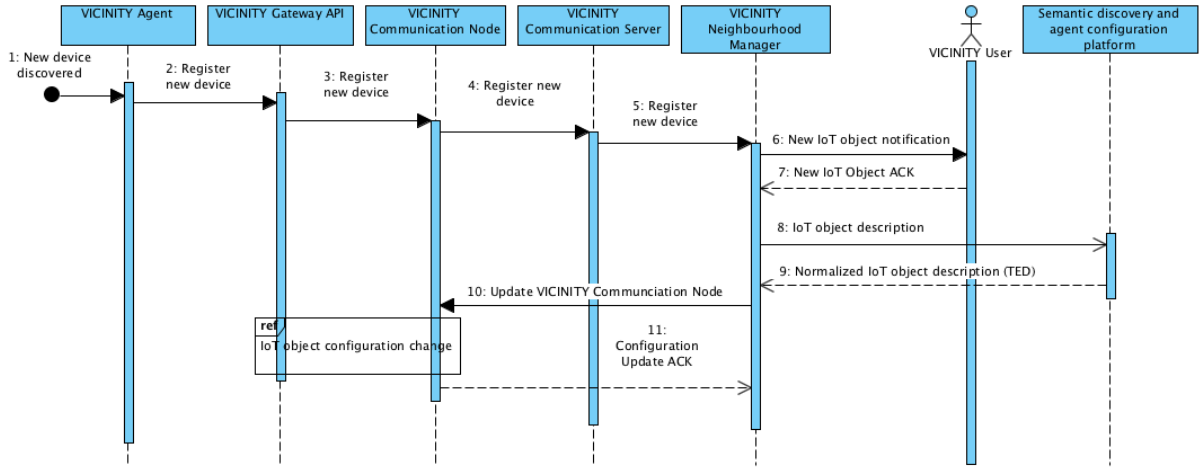

Figure 19 Automatic registration of IoT object in VICINITY

The same process is applied in case of any automatic device configuration change, however automatic device removal cannot be performed.

5.1.3.4. *Crawling of external repositories of IoT object templates*

This process performs crawling of external repositories of IoT objects templates (see section 4.3.2). Each new IoT object template is mapped to VICINITY Ontology. IoT normalized description is provided to VICINITY Neighbourhood manager for manual IoT object registration functionality. Moreover, the relevant VICINITY Agents are updated being able to discover new IoT objects.

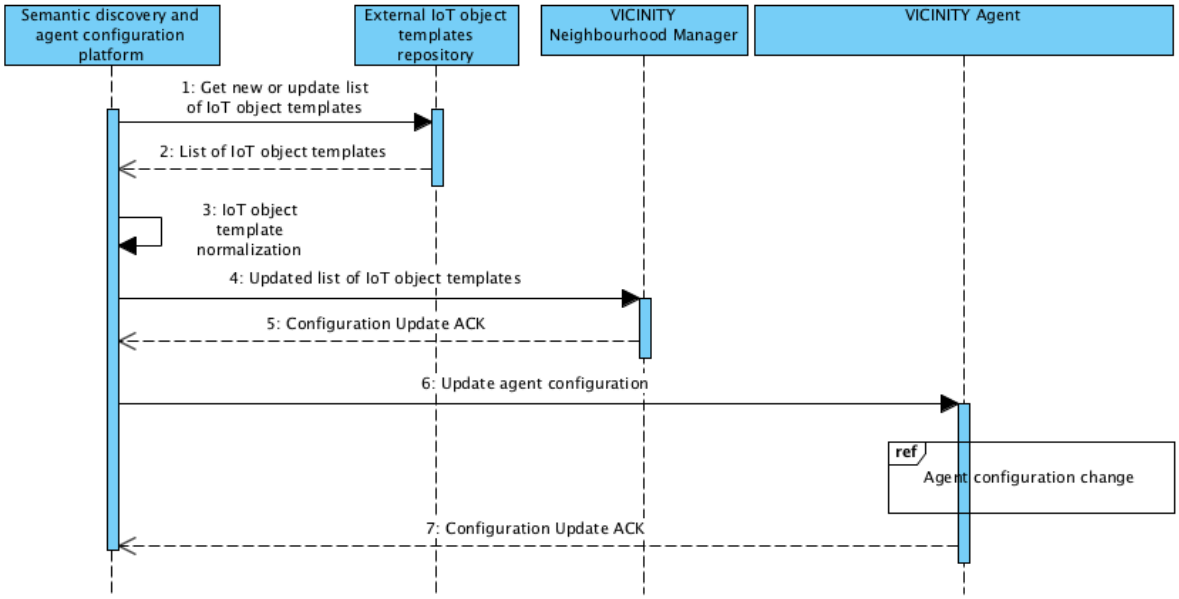

Figure 20 Crawling external repositories process

5.1.4. User data exchange facilitation

The User data exchange facilitation processes enable to consume and expose IoT objects through VICINITY, thus VICINITY Nodes can have the role of clients (consuming IoT objects) and/or servers (exposing IoT objects) of VICINITY P2P network.

VICINITY-ARCH-PRC-150	**IoT objects interaction patterns**
VICINITY shall support the following interaction patterns to exchange user data between VICINITY Node through VICINITY P2P network: • Getting IoT object property; • Setting IoT object property; • Calling action of IoT object; • Receiving of event from IoT object.	
Considered requirements:	
VICINITY-FUNC-UCR145	

5.1.4.1. *Consuming of Things through the VICINITY*

This section defines the user data exchange processes from the point of view of VICINITY Agent/ Adapter (A), which consumes IoT objects through the VICINITY from VICINITY Communication Node (B). The VICINITY Agent/ Adapter (A) consumes IoT objects by:

- "getting" IoT object's property;
- "setting" IoT object's property;
- calling action of IoT object;
- receiving of event from IoT object.

5.1.4.1.1. "Getting" property of consumed IoT object

This process enables VICINITY Agent / Adapter (A) to get property of consumed IoT object. The get property request and response are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Agent/ Adapter (A) identity.

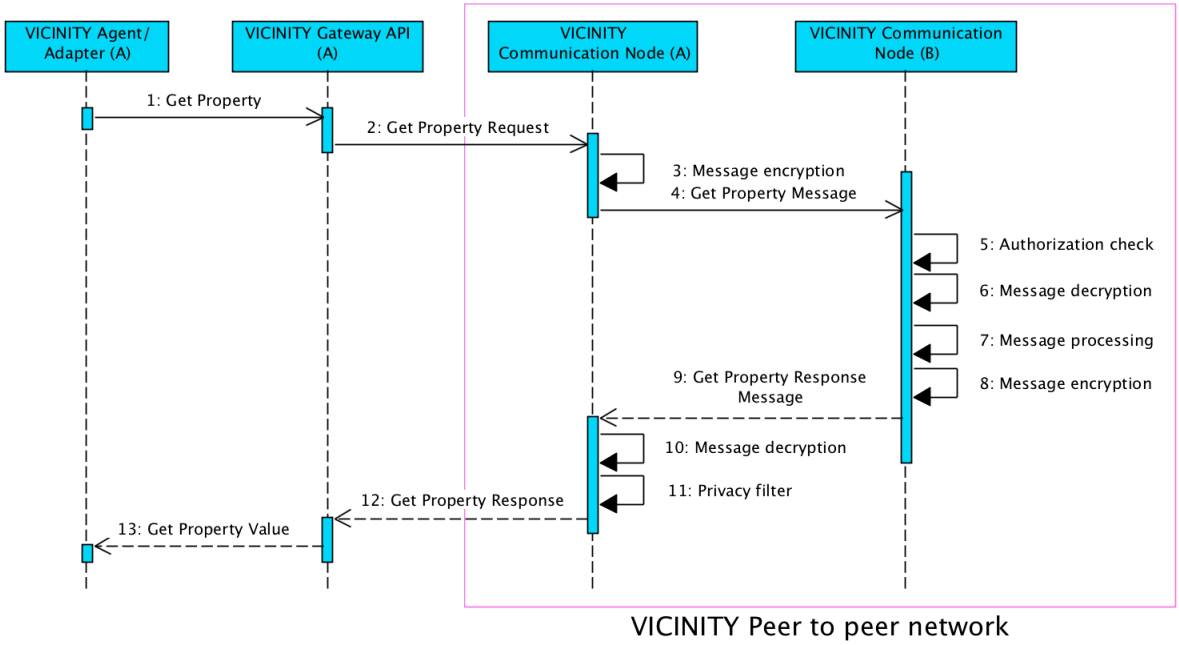

Figure 21 "Getting" property of consumed IoT object

5.1.4.1.2. "Setting" property of consumed IoT object

This process enables VICINITY Agent / Adapter (A) to set property of consumed IoT object. The set property request and response are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Agent/ Adapter (A) identity.

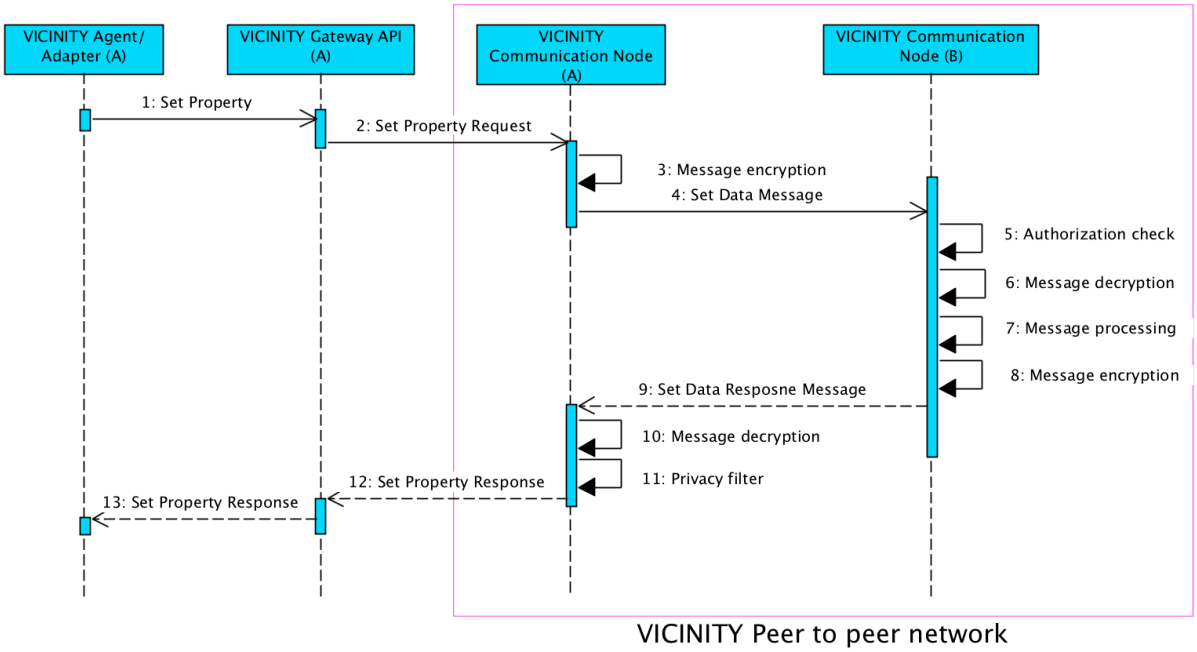

Figure 22 "Setting" property of consumed IoT object

5.1.4.1.3. Calling action of consumed IoT object

This process enables VICINITY Agent / Adapter (A) to call action of consumed IoT object. The action request and response are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Agent/ Adapter (A) identity.

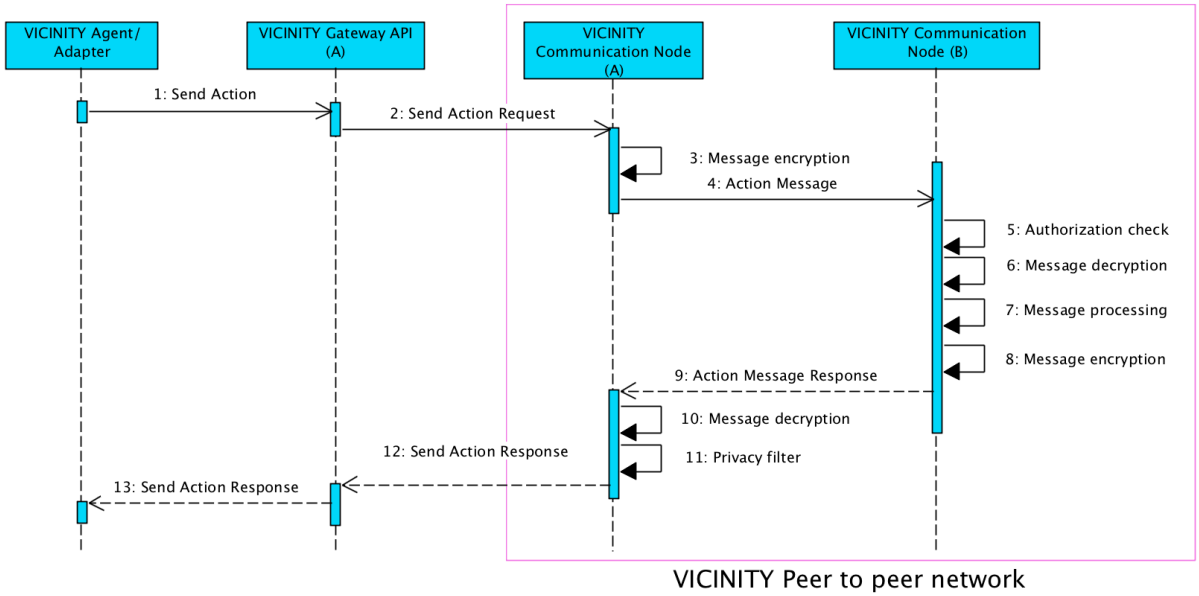

Figure 23 Calling action of consumed IoT object

5.1.4.1.4. Receiving event from consumed IoT object

This process enables VICINITY Agent / Adapter (A) to receive consumed IoT object. The event message and optionally event acknowledgement are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Agent/ Adapter (A) identity.

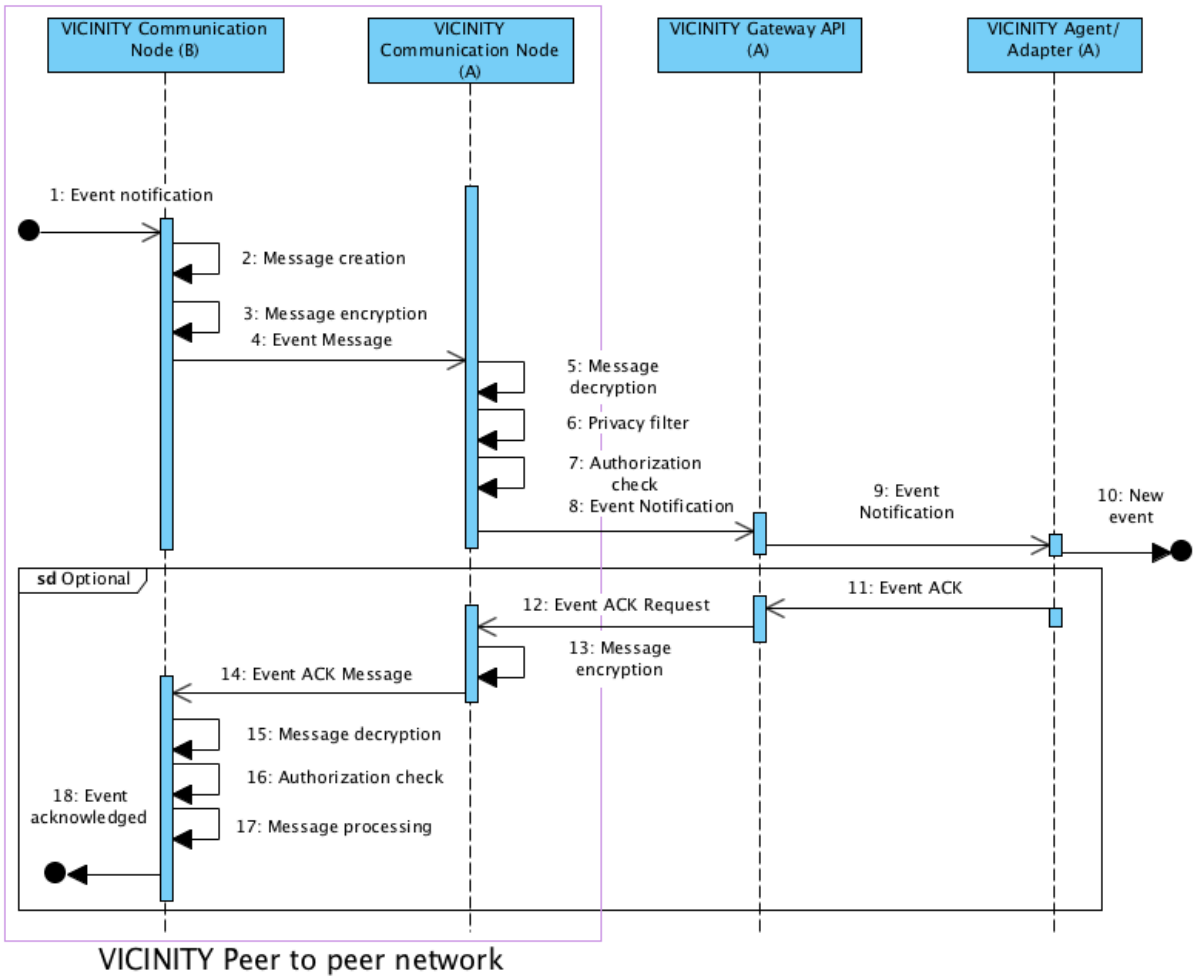

Figure 24 Receiving event from consumed IoT object

5.1.4.2. Exposing of things through the VICINITY

This section defines the user data exchange processes from the point of view of VICINITY Agent/ Adapter (B), which exposes IoT objects through the VICINITY for VICINITY Communication Node (A). The VICINITY Agent/ Adapter (B) exposes IoT objects by:

- "getting" IoT object's property;
- "setting" IoT object's property;
- receiving action of IoT object;
- emitting of event from IoT object.

5.1.4.2.1. "Getting" property of exposed IoT object

This process enables VICINITY Agent / Adapter (B) to receive request for get property of exposed IoT object including provision of property value for VICINITY Communication Node (A). The get property request and response are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Agent/ Adapter (A) identity.

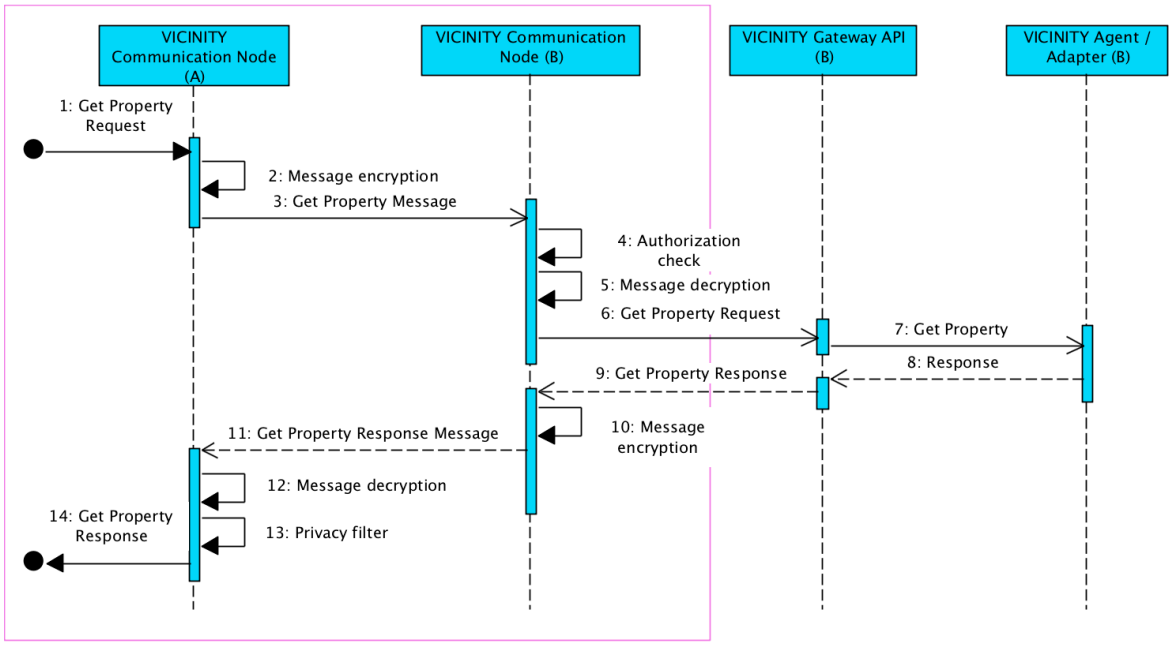

Figure 25 "Getting" property of exposed IoT object

5.1.4.2.2. "Setting" property of exposed IoT object

This process enables VICINITY Agent / Adapter (B) to receive request to set property of exposed IoT object including provision of property value for VICINITY Communication Node (A). The set property request and response are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Communication Node (A) identity.

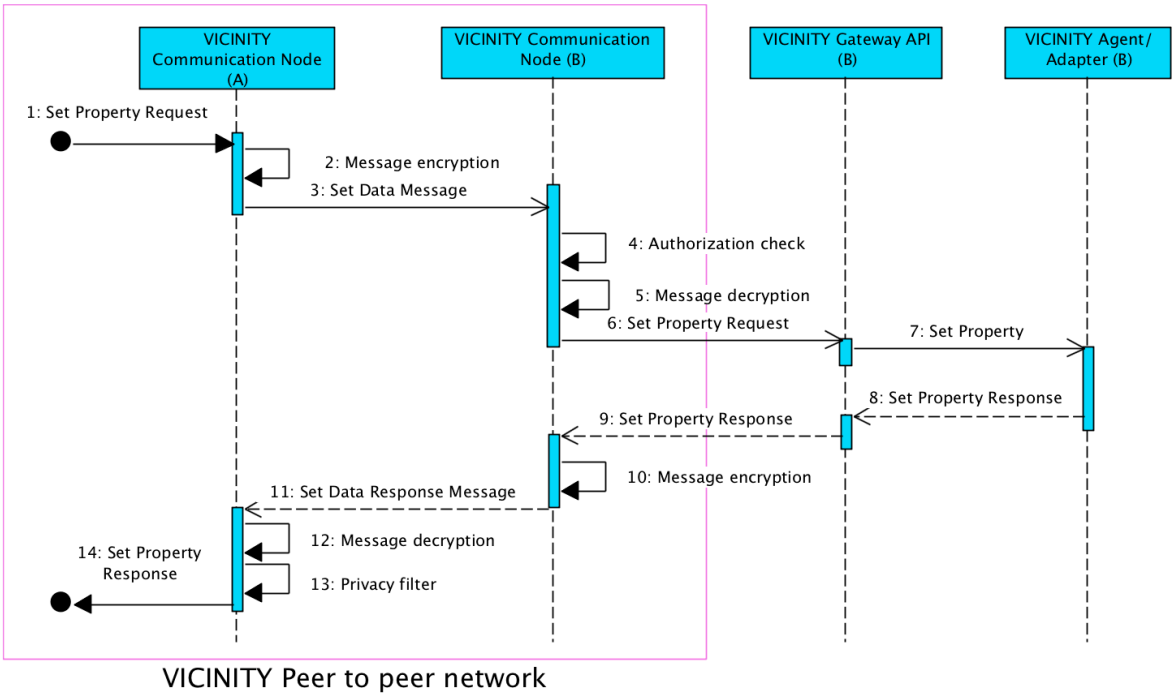

Figure 26 "Setting" property of exposed IoT object

5.1.4.2.3. Receiving action of exposed IoT object

This process enables VICINITY Agent / Adapter (B) to receive action of exposed IoT object. The action request and response are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINIT Agent/ Adapter (A) identity.

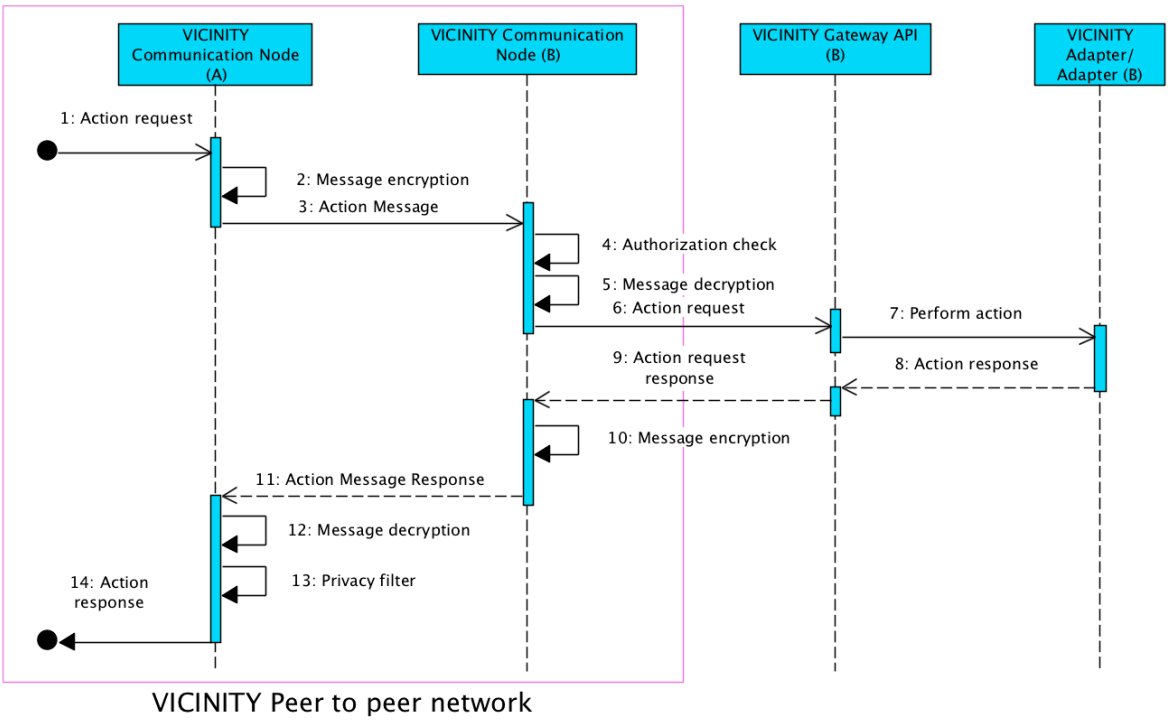

Figure 27 Receiving action of exposed IoT object

5.1.4.2.4. Emitting event of exposed IoT object

This process enables VICINITY Agent / Adapter (B) to emit event of exposed IoT object. The event message and optionally event acknowledgement are communicated through the VICINITY P2P network applying message en/decryption (based on consumed IoT object configuration) and authorization check based on sharing access rules considering context of VICINITY Communication Node (A) identity.

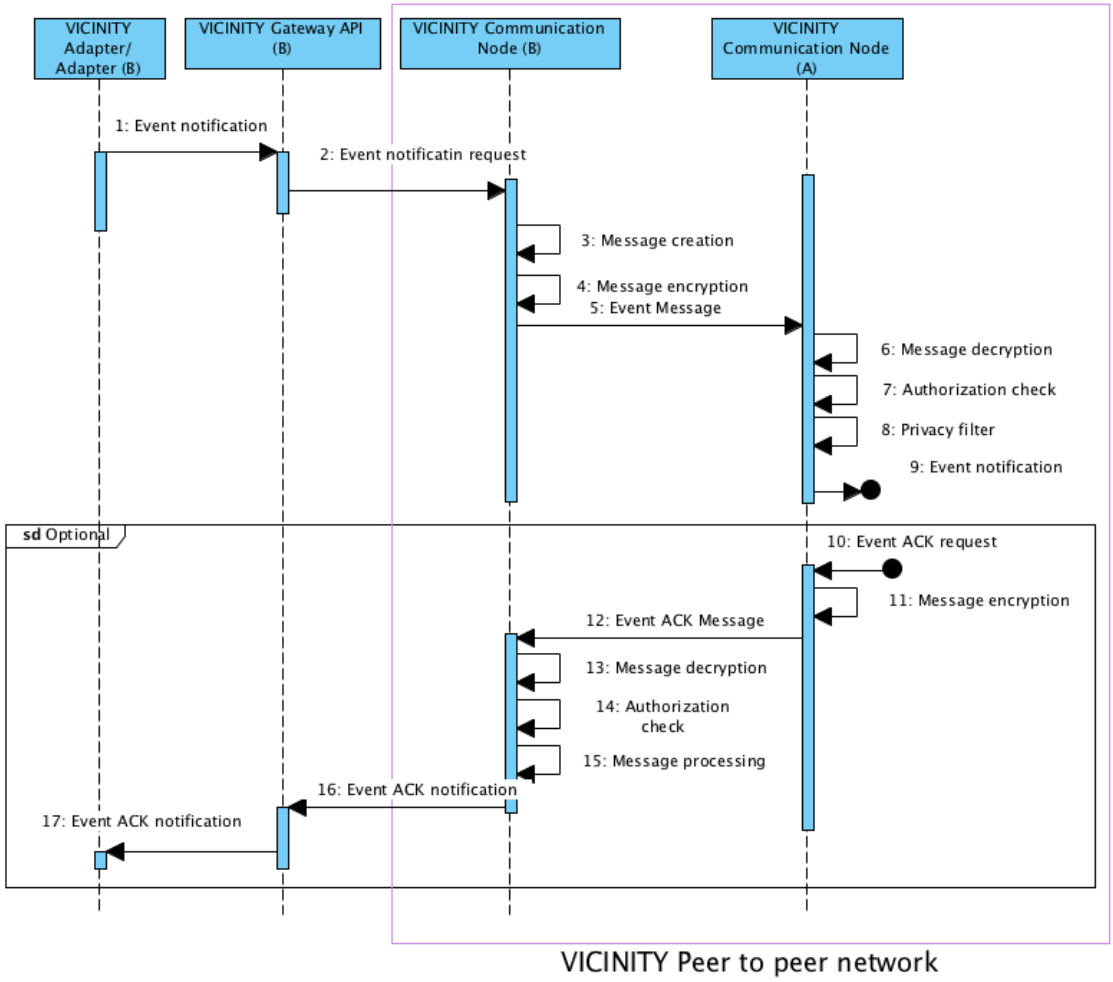

Figure 28 Emitting event of exposed IoT object

5.1.5. Mapping of VICINITY Functionalities to VICINITY Architecture processes

These processes can be mapped to the VICINITY functionalities defined in D1.5 as follows:

Table 8 Mapping of VICINITY Functionalities to VICINITY Architecture processes

Main functionality	Detailed functions	Process
Connecting IoT infrastructure into VICINITY	VICINITY Nodes registration and deregistration	Connecting VICINITY
	Data exchange facilitation	Consuming of things through the VICINITY;

Main functionality	Detailed functions	Process
		Exposing of things through the VICINITY;
Device register and discovery	Register new device	Registration of new IoT object; VICINITY Node configuration distribution; Crawling of external repositories
	Configure IoT device	VICINITY Node configuration distribution
	Remove IoT device	VICINITY Node configuration distribution
Deploy value-added services	Register new value-added service	Registration of new IoT object; VICINITY Node configuration distribution; Crawling of external repositories
	Configure Value-added service	VICINITY Node configuration distribution
	Remove value-added service	VICINITY Node configuration distribution
Interoperability setup	Change in organization partnership;	VICINITY Node configuration distribution
	Change in IoT Device sharing access rules;	VICINITY Node configuration distribution
	Change in Value-added service sharing access rules;	VICINITY Node configuration distribution
VICINITY Security & privacy features	Updating access rules to IoT objects	VICINITY Node configuration distribution; VICINITY User notification;
	Updating security & privacy configuration	VICINITY Node configuration distribution;
VICINITY User notifications		VICINITY User notification;

6. Interfaces View

This section defines set of interfaces between VICINITY Components summarized in Figure 29. The interfaces are specified by name, component providing interface, component using interfaces integration pattern and conceptual data exchanged through the interface.

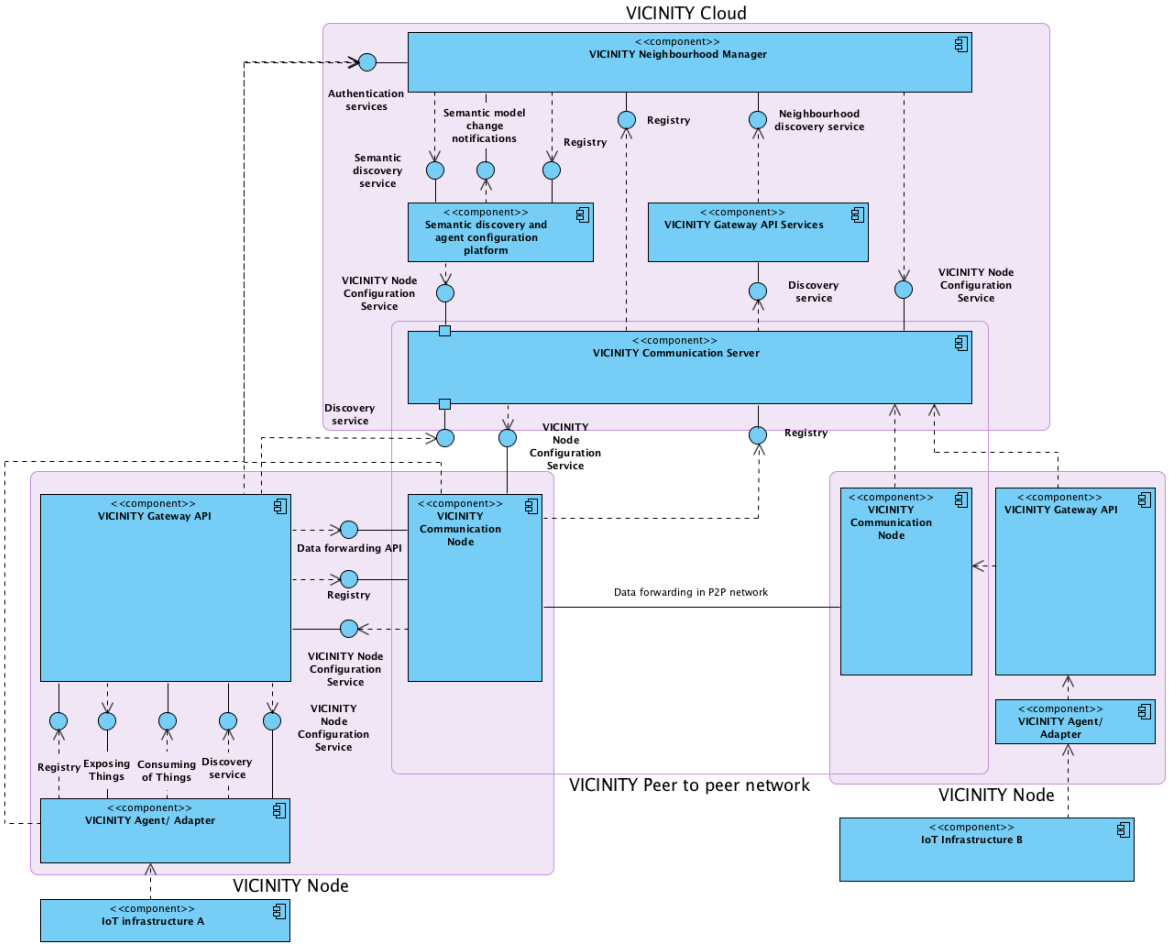

Figure 29 VICINITY Interface view

6.1. VICINITY Cloud

6.1.1. VICINITY Neighbourhood manager

The VICINITY Neighbourhood manager shall provide following interfaces:

Table 9 Interfaces provided by VICINITY Neighbourhood manager

Name of Interface	Used by	Integration pattern	Data Exchanged
Authentication service	VICINITY Communication Node, VICINITY Gateway API, VICINITY Agent	Token request and validation	User credentials, Security tokens

Name of Interface	Used by	Integration pattern	Data Exchanged
Neighbourhood discovery service	VICINITY Gateway API Services	Request/Response	Discovery query, IoT Object descriptions (TEDs)
Registry service	VICINITY Communication Server	Request/Response	IoT Objects meta-data
Semantic model change notifications	Semantic Platform	Request/Response	IoT Objects templates

6.1.1.1. *Semantic discovery and dynamic configuration agent platform*

Semantic discovery and agent configuration shall provide the following interfaces:

Table 10 Interfaces provided by Semantic discovery and dynamic configuration agent platform

Name of Interface	Used by	Integration pattern	Data Exchanged
Semantic discovery service	VICINITY Neighbourhood Manager	Request/ Response	Discovery query, IoT Objects descriptions (TEDs)
Registry Service	VICINITY Neighbourhood Manager	Request/ Response	IoT Object meta-data, IoT Object description (TD)

6.1.1.2. *VICINITY Gateway API Services*

VICINITY Gateway API Services shall provide following interfaces:

Table 11 Interfaces provided by VICINITY Gateway API Services

Name of Interface	Used by	Integration pattern	Data Exchanged
Discovery service	VICINITY Communication server	Request/ Response	Discovery query, Virtual IoT objects descriptions (VTEDs)

6.1.1.3. *VICINITY Communication Server*

The VICINITY Communication server shall provide following interfaces:

Table 12 Interface provided by VICINITY Communication Server

Name of Interface	Used by	Integration pattern	Data Exchanged
Discovery service	VICINITY Gateway API	Request/Response	Discovery query, Virtual IoT Objects descriptions (VTEDs)

Name of Interface	Used by	Integration pattern	Data Exchanged
VICINITY Node Configuration Service	VICINITY Neighbourhood Manager	Message Delivery	VICINITY Node Configuration
Registry Service	VICINITY Communication Node	Message Delivery	IoT Object meta-data, VICINITY Node auto-configurations

6.1.2. VICINITY Node

The VICINITY Node encompasses following components:

- VICINITY Communication Node;
- VICINITY Gateway API;
- VICINITY Agent/Adapter.

6.1.2.1. *VICINITY Communication Node*

The VICINITY Communication Node shall provide the following interfaces:

Table 13 Interface provided by VICINITY Communication Node

Name of Interface	Used by	Integration pattern	Data Exchanged
Data forwarding API	VICINITY Gateway API	Message Delivery	User data
VICINITY Node Configuration Service	VICINITY Communication Server	Message Delivery	VICINITY Node Configuration
Registry Service	VICINITY Communication Server	Message Delivery	IoT Object meta-data, VICINITY Node auto-configurations
Data forwarding P2P	VICINITY Communication Node	Message Delivery	User data

6.1.2.2. *VICINITY Gateway API*

The VICINITY Gateway API shall provide the following interfaces:

Table 14 Interface provided by VICINITY Gateway API

Name of Interface	Used by	Integration pattern	Data Exchanged
Discovery and query service	VICINITY Agent/Adapter	Request/Response	Discovery query, Virtual IoT Objects descriptions (VTEDs)
Consuming service	VICINITY Agent/Adapter	Request/Response	User data
VICINITY Node Configuration Service	VICINITY Agent/Adapter	Message Delivery	VICINITY Node Configuration

Name of Interface	Used by	Integration pattern	Data Exchanged
Registry Service	VICINITY Agent/Adapter	Message Delivery	IoT Object meta-data, VICINITY Node auto-configurations

6.1.2.3. *VICINITY Agent/ Adapter*

The VICINITY Adapter/ Agent shall provide the following interfaces:

Table 15 Interface provided by VICINITY Agent/ Adapter

Name of Interface	Used by	Integration pattern	Data Exchanged
VICINITY Node Configuration Service	VICINITY Gateway API	Message Delivery	VICINITY Configuration messages
Exposing service	VICINITY Gateway API	Request/Response	User data

7. Deployment view

As described in architecture concept section 2, the VICINITY is broken down into two main architecture components VICINITY Cloud and VICINITY Node. The VICINITY Cloud and VICINITY Node instances (One VICINITY Node instance per integrated infrastructure and Value-added service platform) are connected to build distributed VICINITY P2P network.

The VICINITY P2P network is in line with geographical distribution of integrated IoT infrastructures and value added service platform. Thus, VICINITY P2P network by its nature enables distribution of data exchange and computation load according to current needs of integrated infrastructures.

Moreover, loosely coupled VICINITY P2P network promotes localisation of the single failure of the VICINITY Node or VICINITY Cloud.

7.1. VICINITY deployment of VICINITY Cloud

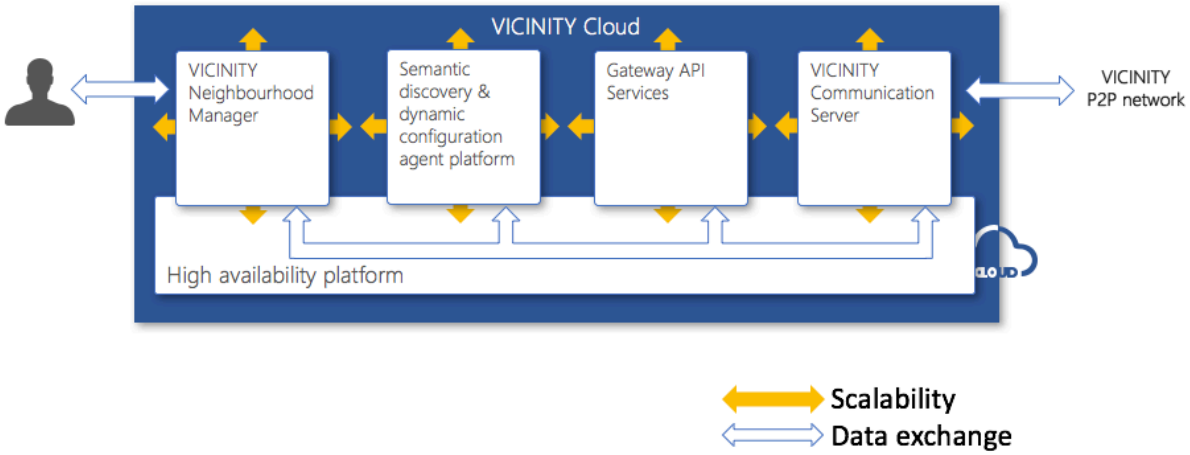

Figure 30 VICINITY Cloud deployment strategy

The VICINITY Cloud encompasses of four scalable components deployed on platform enabling high availability. The high availability platform shall provide features to manage life-cycle of the VICINITY Cloud components on the level of "virtualized operating service".

VICINITY-ARCH-DEP010	**VICINITY Cloud components scalability**
VICINITY Cloud components (including subcomponents and underlying technology) shall support horizontal (scale in/out) and vertical (scale up/down) scaling independently in each layer such as user, service (including integration services) and data.	
Considered requirements:	
VICINITY-NFUNC-AVL010, VICINITY-NFUNC-AVL020, VICINITY-NFUNC-PER010, VICINITY-NFUNC-PER020	

Scale in/out or horizontal scaling enables to execute several instances of the same component (or its subcomponents). Number of executed instances can be adjusted to serve necessary amount of execution requests. Components need to support load-balancing of their interfaces to other components or users transparently.

Scale up/down or vertical scaling enables optimize the usage of high availability platform resources (such as CPU, Memory, Storage space) assigned to instances of components.

Detailed deployment strategy of each VICINITY Cloud component is part of their detailed design.

VICINITY-ARCH-DEP020	VICINITY Cloud components' deployment independence
VICINITY Cloud components' detailed deployment designs shall not influence of designs of other VICINITY Cloud components. Any VICINITY Cloud component deployment shall not influence of deployment of other components.	
Considered requirements:	
VICINITY-NFUNC-AVL020	

7.2. VICINITY deployment of VICINITY Node

To enable communication between IoT devices via the VICINITY P2P Network, the VICINITY Node needs to act as a "gateway". Thus, it needs to "translate" native communication and semantics of its attached devices into VICINITY ontology and expose it via the VICINITY Gateway API to other VICINITY Nodes through the VICINITY P2P network.

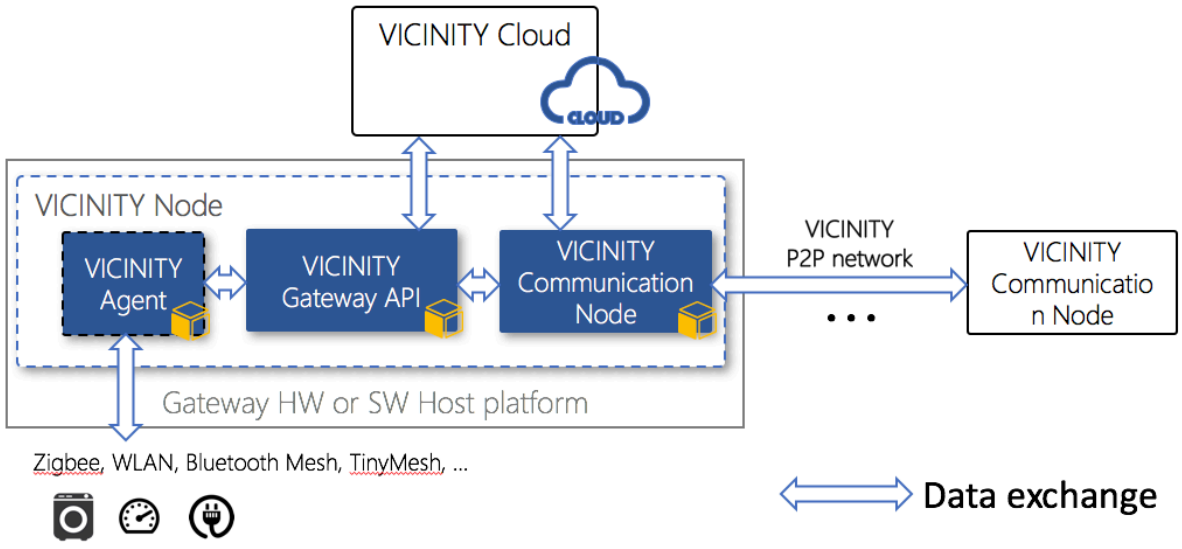

Figure 31 VICINITY Node deployment strategy

VICINITY Nodes is the "virtual" component hosted on the Gateway HW or SW Host platform. Each component of VICINITY Node such as VICINITY Agent, VICINITY Gateway API and VICINITY Communication Node acts as independent loosely coupled deployable unit.

VICINITY-ARCH-DEP030	VICINITY Node components flexible deployment
VICINITY Node components shall support loosely coupling enabling flexible deployment setup according to host platform abilities.	
Considered requirements:	

VICINITY-NFUNC-MNT007, VICINITY-NFUNC-MNT020

7.3. Components monitoring

VICINITY-ARCH-DEP040	**VICINITY components' deployment monitoring**
VICINITY components shall provide means to support standard protocols for monitoring of resources and endpoints and services (such as JMX, SNMP, etc.). Any VICINITY component deployment shall not influence of deployments of other components.	
Considered requirements:	
VICINITY-NFUNC-MNT010	

8. Detail architecture design of VICINITY components

8.1. VICINITY Neighbourhood Manager

8.1.1. Purpose

The goal of VICINITY Neighbourhood Manager is to provide user interface for VICINITY Users to manage virtual neighbourhood and interoperability as service.

8.1.2. Functions

The VICINITY Neighbourhood manager shall have the following components:

- User interface providing searching and virtual neighbourhood management including VICINITY Node configuration;
- Virtual Neighbourhood manager providing internal services such as:
  - Social network management including sharing access rules setup;
  - VICINITY Node configuration update services;
  - User notification services.
- Authentication and authorization module shall authenticate users and technical users including VICINITY components;
- Integration module shall support integration to other VICINITY components such as VICINITY Communication Server, Gateway API Services and Semantic discovery & dynamic configuration agent platform;
- Router component for management of communication within VICINITY Neighbourhood management;
- Storages for:
  - User authentication and authorization,
  - Global neighbourhood storage,
  - Consent and terms of condition storage,
  - Auditing storage.

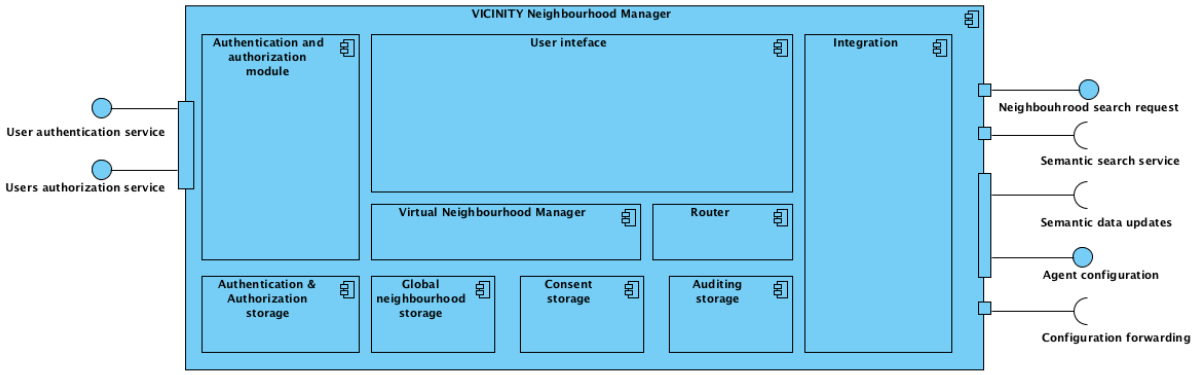

Figure 32 Component diagram of VICINITY Neighbourhood Manager

VICINITY-ARCH-DD-010	VICINITY Neighbourhood manager functions

The VICINITY Neighbourhood Manager shall support VICINITY User through user interface to:

- to search for IoT objects, users, organizations through;
- to manage social network of organizations and IoT objects;

VICINITY-ARCH-DD-010	VICINITY Neighbourhood manager functions

- to manage sharing access rules;
- to configure VICINITY Nodes;
- to manage consents and terms of conditions;
- to manage users and organizations;
- to process user notifications;
- to store user auditing;
- to authenticate and authorize users.

Considered requirements:

VICINITY-NFUNC-SEC060, VICINITY-NFUNC-SEC070, VICINITY-NFUNC-PRV020, VICINITY-NFUNC-PRV030, VICINITY-NFUNC-PRV040, VICINITY-FUNC-UCR010, VICINITY-FUNC-UCR020, VICINITY-FUNC-UCR030, VICINITY-FUNC-UCR040, VICINITY-FUNC-UCR050, VICINITY-FUNC-UCR060, VICINITY-FUNC-UCR070, VICINITY-FUNC-UCR090, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR110, VICINITY-FUNC-UCR130, VICINITY-FUNC-UCR140, VICINITY-FUNC-UCR150, VICINITY-FUNC-UCR160, VICINITY-FUNC-UCR165, VICINITY-FUNC-UCR170, VICINITY-FUNC-UCR190

8.1.3. Dependencies

The VICINITY Neighbourhood manager has the following dependencies to other VICINITY Core components:

- Semantic discovery & dynamic configuration agent platform;
- Gateway API Services;
- VICINITY Communication Server;

8.1.4. Interfaces

Interfaces of VICINITY Neighbourhood manager are identified and described in sections 6.1.1.

VICINITY-ARCH-DD-020	VICINITY Neighbourhood Manager User Interface performance

VICINITY Neighbourhood Manager shall provide user interface response time up to 5s in routine operations such as: authentication, simple search, list of IoT objects, exploring own neighbourhood.

Considered requirements:

VICINITY-FUNC-UCR180, VICINITY-NFUNC-PER010

8.1.5. Resources

The VICINITY Neighbourhood manager manages following resources:

- User authentication and authorization,
- Global neighbourhood storage,
- Consent and terms of condition storage,
- Auditing storage.

VICINITY-ARCH-DD-030	VICINITY Neighbourhood Manager Storage performance

VICINITY Neighbourhood Manager shall provide means to recover of the following storages in case

VICINITY-ARCH-DD-030	VICINITY Neighbourhood Manager Storage performance

of component failure:

- User authentication and authorization storage;
- Global neighbourhood storage;
- Consent and terms of condition storage;
- Auditing storage.

Considered requirements:

VICINITY-NFUNC-PRV020

8.1.6. Data

The data managed by VICINITY Neighbourhood manager are described in 4.3.1.

8.2. VICINITY Communication Server and Node

8.2.1. Purpose

The objective of the VICINITY Communication server and Node:

- Setup communication channels;
- Data forwarding between peers in P2P network;
- Providing VICINITY security services including end-to-end encryption and device verification.

8.2.2. Functions

The VICINITY Communication Server and VICINITY Node are surrounding components for VICINITY P2P network. These components can be broken in the following functional blocks.

VICINITY-ARCH-DD-040	VICINITY Communication Server functional blocks

The VICINITY Communication Server shall consist of the following functional blocks:

- Transaction Module – support communication to Discovery services, Registry and Configuration service to Gateway API, Gateway API Services and Virtual Neighbourhood Management it consists of three components:
  - Discovery processor as simple discovery of things service forwarder between Gateway API and Gateway API Service;
  - VICINITY Configuration processor as configuration splitter to VICINITY Node;
  - Registration processor as validator of IoT object registration services.
- P2P Network Manager – manages the P2P peer network including in P2P network authentication and global white/black list management;
- P2P Network Server Endpoint – message broker for the P2P Network;
- Configuration storage.

Considered requirements:

VICINITY-FUNC-UCR145, VICINITY-FUNC-UCR140, VICINITY-FUNC-UCR130, VICINITY-FUNC-UCR090, VICINITY-FUNC-UCR040

VICINITY-ARCH-DD-040	VICINITY Communication Node functional blocks

The VICINITY Communication Node shall consist of the following functional blocks:

- P2P Network Endpoint as message broker for the P2P Network transmitting P2P Network Messages including local white and blacklist of peers;
- Authorization Services filtering out unauthorized messages from P2P Network;
- Encryption Services performing encryption or decryption of the messages;
- Message Router performing inner VICINITY Node Routing of the messages;
- Gateway API Endpoint translates Gateway API request and responses to/from P2P Network messages and provides privacy filtering (see following note);
- Node Configuration stores all VICINTIY Communication Node related configuration.

Considered requirements:

VICINITY-FUNC-UCR145

Note that, privacy filter service filters out unnecessary user data from changed messages based on IoT object interface description, to ensure that only requested data are transmitted. Software integrators usually provide more generic user data services than necessary to simplify interfaces and implementation, which might result in privacy issues. Privacy filter simple erases any user data which should not be transmitted for selected user data formats (such as JSON or XML).

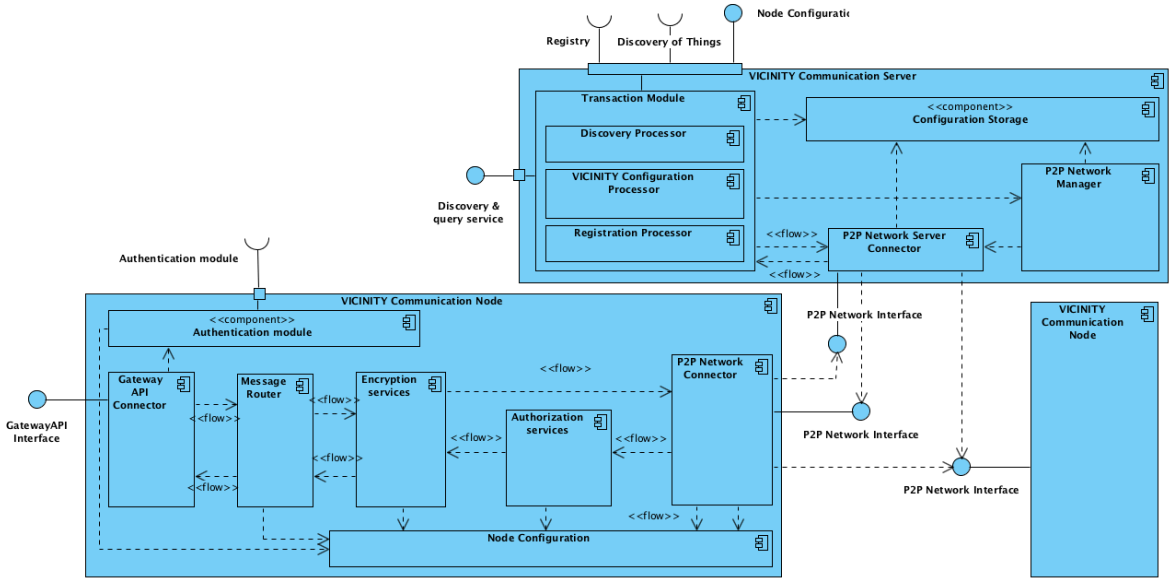

Figure 33 Functional blocks of VICINITY Communication Server and Node

VICINITY-ARCH-DD-040	VICINITY P2P network security

VICINITY shall support following security services on VICINITY Communication Node based on configuration and service availability on the IoT objects:

- P2P message payload encryptions and decryptions based on IoT object descriptions;
- Authorization checks of P2P messages based on sharing access rules.

Considered requirements:

VICINITY-NFUNC-SEC020

VICINITY-ARCH-DD-050	**VICINITY P2P network protection**

VICINITY shall provide means to support protection of VICINITY P2P against:

- Message flooding in case of VICINITY Communication Node reconnection into P2P network;
- VICINITY Communication Node message flooding in VICINITY P2P Network.

VICINITY Communication Node shall support to block or unblock other VICINITY Communication Nodes.

Considered requirements:
VICINITY-NFUNC-SEC020, VICINITY-NFUNC-SEC030

VICINITY-ARCH-DD-060	**VICINITY P2P network performance**

VICINITY P2P Network shall support near to real-time VICINITY P2P message delivery between VICINITY Communication Nodes.

VICINITY P2P Network shall provide means to support prioritization of the P2P messages.

Considered requirements:
VICINITY-NFUNC-MNT020, VICINITY-NFUNC-PER010, VICINITY-NFUNC-PER020, VICINITY-NFUNC-PER030

Note that VICINITY Communication Server and Node providing and consuming interfaces in-line with integration view (6.1.1.3, 6.1.2.1).

8.2.3. Dependencies

The VICINITY Communication Server depends on:

- VICINITY Neighbourhood manager;
- VICINITY Gateway API Services;
- Semantic discovery & dynamic configuration platform.

The VICINITY Communication Node:

- VICINITY Neighbourhood manager;
- VICINITY Communication Server.

8.2.4. Interfaces

Interfaces of VICINITY Communication Server and VICINITY Communication Node are identified and described in sections 6.16.1.1.3 and 6.1.2.1.

8.2.5. Resources

The VICINITY Communication Server and Node manages following resources:

- Storage of IoT object descriptions;
- Storage of IoT object sharing rules;
- Storage of VICINITY Communication Server and Node configuration;

VICINITY-ARCH-DD-070	VICINITY Communication Server and Node storage performance

VICINITY Communication Server and Node shall provide means to recover of the following storages in case of component failure:

- Storage of IoT object descriptions;
- Storage of IoT object sharing rules;
- Storage of VICINITY Communication Server and Node configuration.

Considered requirements:

VICINITY-NFUNC-PER040

8.2.6. Data

The VICINITY Communication Server maintains following information:

- VICINITY Configuration splitting rules;
- P2P Network Server Endpoint configuration
- P2P Network configuration;
- Registry and Discovery of Things service configuration;

The VICINITY Communication Node maintains following information:

- IoT object descriptions (TDs) including encryption services configuration (4.3.2);
- Set of sharing access entries for authorization services (4.3.1.2);
- Gateway API endpoint configuration;
- VICINITY Communication Node credentials;
- Message router configuration;
- P2P Network endpoint configuration;

VICINITY-ARCH-DD-080	VICINITY P2P network messages

VICINITY P2P network messages transmitted in P2P network shall include:

- Message identity context;
- Source of message;
- Destination of message;
- Message identifier;
- Message type;
- Message contents;
- Data integrity attributes.

Considered requirements:
VICINITY-FUNC-UCR145, VICINITY-NFUNC-SEC040

VICINITY-ARCH-DD-090	**VICINITY Communication Node data availability**
VICINITY Communication Node shall support serialization of exchanged messages (VICINITY P2P Network and Gateway API messages) including ability to recovery from failure behaviour.	
Considered requirements:	
VICINITY-NFUNC-PER040	

8.3. Semantic discovery & dynamic configuration agent platform (IS)

8.3.1. Purpose

General purpose of this component is to provide the semantic interoperability when handling the IoT objects. There are following main purposes of this component:

- to perform semantic discovery of IoT objects;
- to perform dynamic configuration of agents;
- to store semantic information and to provide search/lookup services for IoT objects;
- to manage a list of external IoT descriptor repositories.

8.3.2. Function

The core of automatic semantic discovery and dynamic agent configuration is the process of mapping of physical IoT object into its semantic representation. Each time, the new IoT object is introduced, discovery platform will match the semantic template corresponding to this object (e.g. specific device model) and create the new unique instance with persistent identifier shared across VICINITY platform. The new instance (the semantic description - TED) will be mapped to the physical IoT object and will be used as its semantic representation. Semantic search will be then performed on semantic representations of IoT objects and matching instances will be returned together with the mapping to corresponding physical IoT devices and their location (identifier of VICINITY node responsible for interaction with IoT object). The process of semantic discovery and configuration enables to automatically introduce the set of IoT objects for corresponding VICINITY node. This configuration can be further manually updated via Neighbourhood manager. Once, the configuration is changed, it is updated and shared with corresponding VICINITY agent, responsible for interaction with this IoT object.

VICINITY-ARCH-DD-100	**Discovery related functionality**
• Registration discovery of IoT objects: automatic and by request; • Update of IoT objects and agent configuration (enable/disable IoT object, enable/disable IoT object service/action, etc.); • Automatic creation of IoT object model templates from specified external repositories of IoT object descriptors.	
Considered requirements:	
VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR120, VICINITY-FUNC-UCR160	

VICINITY-ARCH-DD-110	Handling semantic models

- providing common semantic vocabulary to describe IoT objects;
- performing CRUD operations on IoT object models and agent configuration;
- executing semantic queries and look-ups;
- serialization of semantic information into common machine-readable format (e.g. JSON, XML).

Considered requirements:

VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR120, VICINITY-FUNC-UCR130, VICINITY-FUNC-UCR160

8.3.3. Dependencies

Based on architectural design, this component interacts only with the Neighbourhood manager. Generally, any VICINITY component (mainly the Node Gateway API), may request the semantic information via Gateway API Services and Neighbourhood manager.

8.3.4. Interfaces

VICINITY-ARCH-DD-120	Discovery API

- Discovery request: given list of IoT object meta-data from VICINITY agent, semantic discovery platform will create semantic models of corresponding IoT objects, map them to physical IoT objects and store in repository. The result of this process is the creation of TEDs for provided IoT object meta-data.
- Get TEDs – IoT object instances: given list of IoT object identifiers, semantic discovery platform will return the semantic instances of corresponding IoT objects.
- Get Agent Configuration: returns the actual configuration of requested agent;
- Set Agent Configuration: updates the actual configuration for requested agent;

Considered requirements:

VICINITY-FUNC-UCR140, VICINITY-FUNC-UCR160

VICINITY-ARCH-DD-130	Query API

- Execute Query: performs the semantic search by executing the query and returns the matches;

Considered requirements:

VICINITY-FUNC-UCR160

8.3.5. Resources

Semantic discovery & dynamic configuration agent platform shall manage the following resource:

- semantic repository which is populated in advance with semantic meta-data – the domain ontologies and IoT object templates, which are used as the basis for semantic discovery.

Semantic repository is a semantic storage and reasoner used for manipulation and querying the semantic data.

Semantic repository is implemented as high-performance semantic storages available to scale up to billions of statements. These storages are optimized to execute semantic queries and manipulate data in real-time responses comparable to performance of common SQL databases.

VICINITY-ARCH-DD-140	Semantic repository performance
Semantic discovery & dynamic configuration agent platform shall support high-performance search and CRUD operation on semantic data to support performance requirements of components accessing semantic repository.	
Considered requirements:	
VICINITY-NFUNC-PER010	

8.3.6. Data

IoT object models are described in more details in section 4.5.2 Semantic Model and Agent Configuration Storage.

8.4. VICINITY Gateway API

8.4.1. Purpose

The goal of the VICINITY Gateway API is to enable semantic interoperability within VICINITY Nodes so that they can easily expose their own IoT objects, as well as discover and interact with any IoT object known to VICINITY within their VICINITY Neighbourhood.

8.4.2. Functions

Most functions of the VICINITY Gateway API can be directly invoked through its available interfaces. However, to support incoming P2P communications, there are additional related functions that are indirectly triggered by a message-queue consumption mechanism provided by the VICINITY Communication Node interface, e.g. receiving a message that requests to get the property value of an exposed IoT object.

For all those functions listed below that involve interaction with an IoT object, it is assumed that they are in scope of VICINITY Node's neighbourhood.

VICINITY-ARCH-DD-150	VICINITY Gateway API functions
The VICINITY Gateway API shall support the following functions to: • Request authentication in VICINITY; • Register as VICINITY Gateway API; • Register/Expose new IoT objects to VICINITY Agent/Adapter; • Discover IoT objects that match a required search pattern;	

- Get the current value of a specific property of an IoT object;
- Set the value of a specific property of an IoT object;
- Perform a specific action on an IoT object;
- Get the status of on-going specific actions on an IoT object;
- Subscribe/Unsubscribe/Listen to events issued by an IoT object;
- Get the list of available interaction patterns (i.e., properties, actions and events) of an IoT object along with each value constraints and units;
- Get the value of a specific property of an exposed IoT object;
- Set the value of a specific property of an exposed IoT object;
- Perform a specific action on an exposed IoT object;
- Get the status of on-going specific actions on an exposed IoT object;
- Subscribe/Unsubscribe/Listen to events issued by exposed IoT object;
- Process configuration updates of exposed IoT objects from VICINITY Cloud;
- Query the VICINITY P2P Network by means of a combination of discovery and access functions, by using SPARQL[9] and the VICINITY Ontology.

Considered requirements:

VICINITY-NFUNC-PRV040, VICINITY-FUNC-UCR145

8.4.3. Interfaces

To support functions listed in section 8.4.2, the VICINITY Gateway API shall be divided into specific groups of sub-interfaces for Registry, Discovery, Consumption and Querying.

All interfaces will require working under an authentication context with VICINITY.

VICINITY-ARCH-DD-160 VICINITY Gateway API for Registry

The Registry interface shall support registration of IoT objects within a Node in two ways:

- Aggregated registration of a set of IoT objects;
- Individual registration of each IoT object.

Supported functions:

- Register as VICINITY Node;
- Register/Expose new IoT objects.

Considered requirements:

VICINITY-FUNC-UCR145

Individual registrations can happen at any time after the Node is successfully registered in VICINITY; it supports automatic discovery and dynamic integration of IoT objects in the network.

When it comes to invoke a registration, the Registry interface expects to receive IoT object descriptions represented in the common VICINITY format (see section 8.5.6). Internally, the Gateway

[9] https://www.w3.org/TR/sparql11-query/

API shall validate the received descriptions against the schema before sending it as a message to the VICINITY Cloud (though the P2P Network).

Once the described process is successfully completed, the Gateway API gets prepared to asynchronously receive feedback from the VICINITY Cloud (Semantic discovery & dynamic configuration agent platform, see section 8.3) in the form of enriched versions of such semantic descriptions (TED). Such TED will provide the Gateway API a better knowledge about its underling infrastructure of IoT objects.

VICINITY-ARCH-DD-170	**VICINITY Gateway API for Discovery**

The Discovery interface will allow to discover those IoT objects that match a fixed search pattern within VICINITY Neighbourhood. The search pattern (aligned to the common VICINITY format for IoT object descriptions) shall at least support the following criteria:

- Type of IoT object;
- Regarding IoT object properties,
  - Type;
  - Data format;
  - Units.
- Regarding IoT object actions,
  - Type;
- Regarding IoT object events,
  - Type;
  - Throughput.

In order to support more expressive search patterns and take advantage of the VICINITY Ontology, the Discovery interface shall optionally support constrained SPARQL queries for discovery requests.

Supported functions:

- Discover IoT objects that match a required search pattern.

Considered requirements:

VICINITY-FUNC-UCR160

Once the search criteria are validated, the discovery process shall continue as follows:

1. Generate the corresponding SPARQL query (using the VICINITY Ontology) to be sent to the VICINITY Gateway API Services within the VICINITY Cloud.
2. Wait until a response is received. The result shall be a VTED (Virtual Things Ecosystem Description) that contains the set of TDs of the specific IoT objects that match the provided search criteria.
3. Transform all TDs into the common VICINITY format (see section 8.5.6).

The responses provided by Discovery interface will consist of a set of tuples (built from output of step 3 of the discovery process) which identify the matching IoT objects by specifying identifiers and relevant meta-data so that they can be addressable afterwards, e.g., the Node, IoT object and property IDs.

VICINITY-ARCH-DD-180	VICINITY Gateway API for Consumption

The Consumption interface of the Gateway API aims at:

- Providing access methods to IoT objects that belong to integrated infrastructures within VICINITY Neighbourhood.

The access operations offered by this interface takes a tuple of identifiers as input, e.g. one of the results provided by a Discovery operation, which shall serve as pointer to a specific IoT object and even to one of its properties, actions or events.

Supported functions:

- Get the current value of a specific property of an IoT object;
- Set the value of a specific property of an IoT object;
- Perform a specific action on an IoT object;
- Get the status of on-going specific actions on an IoT object;
- Subscribe/Unsubscribe/Listen to events issued by an IoT object;
- Get the list of available interaction patterns (i.e., properties, actions and events) of an IoT object along with each value constraints and units.

Considered requirements:

VICINITY-FUNC-UCR145

The Gateway API it leverages the VICINITY P2P Network as a distributed transport mechanism for requesting and receiving user data from the relevant and addressable VICINITY Node.

Once a tuple of identifiers is provided, the following Consumption process takes place:

1. Request a VTED (Virtual Things Ecosystem Description) to the VICINITY Cloud (if it was not already cached in the Gateway API) that strictly, semantically and completely describes the referenced IoT object. This VTED should contain everything that VICINITY knows so far about such IoT object.
2. Query the VTED to:
    - Identify the VICINITY Node that provides secure access to the referenced IoT object;
    - Extract specific security and privacy constraints and any further meta-data that is required to establish communication properly.
3. Prepare, serialize and send a message to the VICINITY Communication Node. This message must unequivocally describe the intended operation plus the identifiers that address the interaction resource of the IoT object to be consumed. Due to P2P communications are asynchronous, the request will be given a unique identifier so that its expected response can be correctly identified.

As a response is received through the VICINITY Communication Node, it is de-serialized and processed to extract the corresponding result of the operation. When such result encapsulates a value that may require some transformation, e.g., unit conversion, the Gateway API shall optionally perform it just before returning the result.

From the recipient Gateway API point of view, at the time a consumption request is received the following process is triggered:

1. Query the TED that describes the underlying IoT infrastructure (see Registry interface) to determine which are the required endpoints that need to be invoked to satisfy the

consumption request. Such query shall consider the identifiers provided in the request that refers to:
    - Specific IoT object;
    - Specific interaction (property, action or event);
    - Required operation (get, set, subscribe, etc.).
2. Invoke the extracted endpoints (that should be offered by the corresponding Agent/Adapter).
3. Once all endpoints' responses are gathered, the Gateway API shall process, unify and serialize them in a unique message as response to the requester.

VICINITY-ARCH-DD-190	VICINITY Gateway API for Querying

The goal of the Querying interface is to:

- Provide full support to SPARQL queries whose intention is not only to discover *things* but also to access and consume their data in one-step interaction;
- Become a standard SPARQL endpoint for consuming VICINITY IoT objects. The interface shall implement the SPARQL 1.1 Protocol[10].

Supported functions:

- Query the VICINITY P2P Network by means of a combination of discovery and access functions, by using SPARQL and the VICINITY Ontology.

Considered requirements:

VICINITY-FUNC-UCR145, VICINITY-FUNC-UCR160

Therefore, given a SPARQL query, the clients of this interface will rely on the Gateway API to do the following:

1. Discover those IoT objects that are relevant for the given query. This process shall be supported by the VICINITY Gateway API Services (see section 6.6), concretely for providing the VTED (Virtual Things Ecosystem) that fully describe the involved *things*.
2. Create a query plan that describes what to collect and where from as well as how the query shall be evaluated (steps 3 and 4).
3. Collect all relevant information from the IoT objects by establishing communication with their corresponding VICINITY Nodes (as explained in the Consumption interface table).
4. Evaluate the query against the collected triples (RDF graph) and generate the existing solutions.

8.4.4. Resources

The Gateway API shall make use of a graph database (triple store) for storing and caching different semantic data.

[10] https://www.w3.org/TR/sparql11-protocol/

VICINITY-ARCH-DD-200	VICINITY Gateway API performance
VICINITY Gateway API shall provide means to support caching of the exchanged user data.	
Considered requirements:	
VICINITY-NFUNC-PER050	

8.4.5. Data

All the different interactions and processes that take place within VICINITY Gateway API will involve the usage of two different kinds of data: syntactic and semantic.

Syntactic data refers to the data which the Gateway API will exchange with its surrounding Node components; simple schema-based information that is not semantically described, thus allowing these data exchanges to be agnostic of semantics defined in the VICINITY Ontology.

- Syntactic data
  - IoT object descriptions in VICINITY common format (see section 6.5);
  - VICINITY P2P network messages (see section 6.2.2).

Semantic data represents all data that is described using the VICINITY Ontology.

- Semantic data
  - Thing Descriptions (TD) of IoT objects;
  - Things Ecosystem Description (TED) of the underlying IoT infrastructure;
  - Virtual TED for discovery processes and querying requests.

8.5. VICINITY Agent / Adapter

8.5.1. Purpose

The main goal of VICINITY Agent is to provide access to IoT objects of underlying infrastructure (e.g. specific middleware) or value-added services in uniform way.

8.5.2. Function

The main functionality of VICINITY Agent is to integrate existing solution, which provides access to IoT objects (specific middleware, application, value-added services), into VICINITY. The role of VICINITY Agent is to map all IoT objects available via underlying infrastructure into common VICINITY form, to enable uniform access to all integrated infrastructures.

The most important part of VICINITY Agent is the VICINITY Adapter subcomponent. Adapter serves as the proxy between common VICINITY services and underlying infrastructure. For each specific infrastructure to be integrated, there must exist specific VICINITY Adapter. The role of Adapter is to translate VICINITY services into infrastructure specific services. VICINITY Adapter provides the API, which must be implemented, when integrating new infrastructure.

VICINITY-ARCH-DD-210	Adapter functionality

Adapter must be able to:

- Read descriptions of included IoT objects and translate it into common VICINITY format. Acquired descriptors are used for automatic discovery, creating corresponding semantic models of IoT objects and mapping between semantic models and physical IoT objects in infrastructure.
- Interact with specific IoT objects in infrastructure: read/write data from specific IoT object service or perform actions.

Considered requirements:

VICINITY-FUNC-UCR145

VICINITY-ARCH-DD-220	Agent functionality

The VICINITY Agent is the logical component using assigned Adapter to perform VICINITY services for interaction with underlying IoT objects.
The main functionalities of VICINITY Agent are:

- To discover and register IoT objects in underlying architecture: All available descriptions of IoT objects are mapped into VICINITY semantic representation. IoT objects discovery can be performed by request, by it can be performed also automatically, when the VICINITY node is first time introduced into platform;
- To interact with IoT objects: Given identifier of IoT objects and its service, the read/write data operations can be performed by processing get/set property, execute action or read event services. Data services may be further parametrized, for example by adding time interval, ordering and number of results request;
- To simulate supported type of IoT objects from neighbourhood in underlying integrated infrastructure;
- To actualize Agent configuration updated via Neighbourhood manager.

Considered requirements:

VICINITY-FUNC-UCR145

8.5.3. Dependencies

Based on architectural design, this component interacts only with the Gateway API.

8.5.4. Interfaces

Execution of all interface services always considers the actual configuration of Agent.

VICINITY-ARCH-DD-230	Adapter API

- Expose IoT objects: returns the list of available IoT object meta-data (to be discovered) in common VICINITY format.
- Execute Request: given identifier of IoT object and its service (optionally with additional parameters), the corresponding service of underlying infrastructure is executed and results are returned. The VICINITY get/set property, perform action or read event services are translated info form of underlying infrastructure.

Considered requirements:
VICINITY-FUNC-UCR145, VICINITY-FUNC-UCR100, VICINITY-FUNC-UCR110, VICINITY-FUNC-UCR130

VICINITY-ARCH-DD-240	**Agent API**

- Discovery Request: all IoT objects in underlying infrastructure are passed into semantic discovery component and registration/discovery is performed
- Get/Set property, perform action, read event: IoT object interaction services
- Update configuration: the configuration is updated

Considered requirements:
VICINITY-FUNC-UCR145, VICINITY-FUNC-UCR160, VICINITY-FUNC-UCR140, VICINITY-FUNC-UCR145

8.5.5. Resources

VICINITY Agent and Adapter operate above the existing infrastructure of IoT objects.

8.5.6. Data

- IoT object descriptions in VICINITY common format provided by Adapter: used in discovery and registration process;
- The result of IoT object interactions: read data results or operation success, both provided in common VICINITY format;

8.6. VICINITY Gateway API Services

8.6.1. Purpose

The goal of the VICINITY Gateway API Services is to support discovery in VICINITY, being the VICINITY Cloud entry point for the P2P Network that treats all incoming discovery requests.

8.6.2. Functions

All functions described below shall only be invoked through the VICINITY P2P Network. Thus, they will only work for authenticated VICINITY Nodes in a secure communication context.

VICINITY-ARCH-DD-250	**VICINITY Gateway API Services' functions**

The VICINITY Gateway API Services shall support the following functions to:

- Generate semantic descriptions of things' ecosystems (Virtual Things Ecosystem Description - VTED) that are relevant for discovery requests;
- Generate discovery and consumption plans (query plans) for queries issued from VICINITY Nodes.

Considered requirements:
VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR120, VICINITY-FUNC-UCR145

8.6.3. Dependencies

In order to correctly perform its task, the Gateway API Services shall only require the VICINITY Neighbourhood Manager to be available. The VNM will be required to discover relevant and accessible (within Neighbourhood) IoT objects for both discovery and querying requests.

8.6.4. Interfaces

The VICINITY Gateway API Services shall define a dedicated interface per each function described in section 6.6.2, namely the Discovery and Querying interfaces.

VICINITY-ARCH-DD-260	**VICINITY Gateway API Services for Discovery**

The Discovery interface will have a very concrete purpose: to provide the VTED that describes so far, the set of *things* that matches a search criteria given in the form of a *constrained* SPARQL query. The Gateway APIs will be the actual clients of this interface, thus responsible for creating and issuing the corresponding query (see section 6.4.3).

The supported SPARQL queries are those whose pattern only refers to concepts and properties that are in scope of the semantics of search patterns defined in section 6.4.3, i.e., only IoT object meta-data will be involved.

Clients of the Discovery interface shall not be returned with the intuitive result of the query evaluation (set of solutions as tuples), but a graph (semantic data) that describes everything that VICINITY knows about all involved IoT objects (VTED).

Supported functions:

- Generate semantic descriptions of things' ecosystems (Virtual Things Ecosystem Description - VTED) that are relevant for discovery requests.

Considered requirements:

VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR120, VICINITY-FUNC-UCR145

VICINITY-ARCH-DD-270	**VICINITY Gateway API Services for Querying**

The Querying interface can be considered as an extension of the Discovery interface. Its purpose is to support SPARQL queries that uses the VICINITY Ontology not only for discovery purposes but also for consumption.

Its main purpose is to reduce the workload of Gateway APIs, specifically to their querying process (see Gateway API for Querying in section 6.4.3).

Supported functions:

- Generate discovery and consumption plans (query plans) for queries issued from VICINITY Nodes.

Considered requirements:

VICINITY-FUNC-UCR080, VICINITY-FUNC-UCR120, VICINITY-FUNC-UCR145

From the Gateway APIs point of view, they will receive a query plan that will prevent them from having to:

1. Prepare a constrained SPARQL query from the given one, just to discover the *things* that may be relevant to obtain results.
2. Process and analyse the result of a discovery request for such constrained SPARQL query (VTED) to identify all IoT objects and their access configuration (address, security aspects, endpoints, etc.)
3. Finally, create a query plan themselves.

Therefore, those *lightweight* Gateway APIs that rely on this interface shall have just to follow straightforwardly the provided query plan for a given SPARQL query.

8.6.5. Resources

The Gateway API Services shall make use of a graph database (triple store) for storing and caching different semantic data.

8.6.6. Data

Both discovery and querying processes work with semantic data (TD, VTED) and the VICINITY Ontology.

9. Quality considerations

The section deal with quality considerations in VICINITY architecture. It focuses on the following quality issues:

- Usability,
- Reliability,
- Scalability and Performance,
- Maintenance,
- Security & Privacy.

9.1. Usability

Localization: user interface provided by VICINITY components (such as VICINITY Neighbourhood Manager) shall provide tools to support localization of text, navigation, date, time and currency through ICU (International Components for Unicode) standards. Localized messages and formatted texts shall not be integral part of VICINITY components' source code ("hard coded") and shall be configurable separately.

Accessibility: user interface provided by VICINITY components shall support different type of devices such as desktop screens, laptops, tables and smart phone through user interface responsive design.

User interface shall provide layer of technical services (user interface service layer) which can be used by current or future devices (wearables, smart watches, augmented reality devices) and custom application to facilitate users' interaction with VICINITY.

Responsiveness of user interface: user interface shall react to users requests without causing any discomfort to user. Activities with longer processing time shall provide "progress bar or waiting icon" for users. User interface service layer shall provide status for long process time activities.

9.2. Reliability

9.2.1. Availability

Availability of VICINITY Cloud components provide common interoperability services for the integrated infrastructures and value-added services deployed on high-availability platform (see Section 7). High-availability platform shall support availability on hardware layer (for software layer see scalability section 9.3) including:

- Redundant internet connections to protect infrastructure from loosing of connectivity,
- Infrastructure layer protection against DDoS attacks to protect services from overloading,
- RAID support for data storages to support protection against storage hardware failure,
- Operating system virtualized environment to protect running components from failure of underlying physical infrastructure and optimize available resources (CPU, RAM and HDD for operating systems),
- Support of infrastructure load balancing to protect components from overloading of underlying physical infrastructure.

Availability of VICINITY Node components are deployed in integrated infrastructures and thus their availability is directly influenced by the availability of underlying hardware infrastructure.

VICINITY Cloud and Node components shall respect independence from physical infrastructure and lifecycle of operating system maintenance (such as restart, migration within virtualized environment

etc.). Components shall provide means to support health-check mechanism of its services' aliveness and functioning (9.4). Components shall provide means to support secure runtime upgrades of components, however VICINITY user or administrator shall be notified through user interface if physical intervention is necessary.

9.2.2. Fault tolerance

Fault tolerance against VICINITY Communication Server is critical while the server manages P2P network and facilitates all communications between VICINITY Cloud components and VICINITY Nodes. Unavailable VICINITY Communication Server will suspend any communication between VICINITY Nodes. VICINITY Nodes and shall be able to securely reconnect to VICINITY Communication Server once available. While VICINITY Communication Server is unavailable:

- VICINITY Neighbourhood manager shall buffer any VICINITY Configuration changes;
- VICINITY Nodes shall buffer any P2P messages or refuse any request on the gateway API interface with an appropriate status code.

Fault tolerance against VICINITY Node components – failure of VICINITY Node might have influence on VICINITY Cloud components and another VICINITY Nodes. VICINITY Node and Communication server are aware of availability[11] of any VICINITY Node in P2P network, thus they shall decide whether to send messages to the unavailable node. P2P messages already sent to unavailable VICINITY Nodes shall be buffered in the VICINITY Communication Server's P2P Network manager for later delivery.

Fault tolerance against VICINITY Neighbourhood manager – failure of VICINITY Neighbourhood Manager results in unavailability of VICINITY User interfaces, i.e. it will not be possible to perform any configuration change in vicinity neighbourhood and discovery and query service in open gateway API. User data exchange within P2P network is constrained by unavailability to perform any changes in existing VICINITY Configuration and sharing access rules.

Fault tolerance against integrated infrastructures services – assuming VICINITY Node components are alive and functioning, the node (VICINITY Agent or Adapter) shall resolve any request for interaction with (temporary or permanent) unavailable IoT object by an appropriate status codes.

Fault tolerance against Semantic platform – unavailability of semantic platform constraints discovery and query services of VICINITY Gateway API. VICINITY Neighbourhood Manager shall resend any request to Semantic platform once available to complete IoT objects registration and discovery search requests. While configuration of the semantic interoperability is distributed among VICINITY Nodes, user data exchange between nodes is constrained by missing latest updates in semantic model and data.

Fault tolerance against VICINITY Gateway API Services – unavailable VICINITY Gateway API Services temporarily constraints discovery and querying services of VICINITY Gateway API only, these services shall return an appropriate status codes.

[11] visibility of VICINITY Node availability is subject of security imposed by sharing access rules.

9.3. Scalability & Performance

9.3.1. Latency

Latency of exchanged user data: requirements for latency of exchanged user data varies from application to application. Latency cannot be lower that latency introduced by integrated infrastructure, thus VICINITY should focus on minimisation of additional latency in VICNITY Node components such as VICINITY Open Gateway API, VICINITY Communication Node and VICINITY Peer-to-Peer network.

VICINITY Open Gateway API provides interoperability services with different complexity (8.4.3) from Consuming services with lowest latency up to semantic discovery and querying services with high complexity and highest latency. VICINITY Adapter can choose a set of services to use based on latency requirements.

VICINITY Communication Node pre-processes exchanged user data with privacy filtering or end-to-end security, these privacy and security services add another latency in the communication. However, these features may be bypassed by configuration in IoT object description.

VICINITY Peer-to-peer network shall be built on implementation of XMPP protocol which enables near to real-time message exchange between nodes. In certain cases, it is possible to prioritize exchanged messages to suite needs of application.

9.3.2. Capacity

Throughout architecture design of VICINITY, the following capacity issues have been identified, these capacity issues will be addressed in following scalability section. Users and IoT objects growth rate influences the size and the complexity of the semantic model, semantic data storage (4.3.2) and global neighbourhood storage (4.3.1) which might constraint comfortability of VICINITY usage, performance of discovery queries, raise of user data exchange rate.

9.3.3. Scalability

Scalability of VICINITY Cloud components: as described in deployment view (Section 7.1) all VICINITY Cloud components are deployed on high-availability platform. High-availability platform will deal with vertical scalability of the components (such as assigning necessary CPU, RAM and HDD) and hardware/ infrastructure level failovers. However, VICINITY Cloud components shall support a horizontal scalability on different layers (data storage clusters, application server farm, web server farm, etc.), i.e. provided their services across multiple machines in parallel, being able to response even high computation volumes and storage loads and response to VICINITY users and IoT objects growth rate.

Geographical distribution of Peer-to-Peer network: the nature of the P2P network is its geographical distribution according to geographical distribution of the peers – VICINITY Nodes. Thus, the computation volumes performed by peers is distributed and not concentrated in one location. Moreover, P2P network is loosely coupled thus overloading one node influence indirectly only nodes with active communication. Note that, high user data exchange rate in P2P network can result in higher load of on P2P network manager component (8.2.1), this issue shall be addressed by multiple instances of the manager geographically distributed according to P2P network load.

9.4. Maintenance

Configuration of VICINITY components: configuration of static and behavioural properties of each component is necessary to customize components to changes of their environment (such as

communication interfaces configuration, logging mechanism, etc.). Each component shall include configuration separated from compiled and linked source code. Configuration parameters can be online and offline. Online configuration parameters can be changed without restarting of component. Offline needs restart of the component. VICINITY Node components shall promote online configuration parameters over offline. Important configuration changes with potential impact on security and privacy should be approved by VICINITY User (see processes in 5.1.1, 5.1.2, 5.1.3).

Installability of VICINITY Node components: VICINITY Node components shall support deployment in different environments (7.2), thus VICINITY Node components should use technology which supports different type of software packaging (such as WAR file for java based application servers, Docker container for Microsoft Azure or Amazon AWS, OS virtual machines).

Monitoring of VICINITY components: to keep VICINITY components running for long time in the production it is important to monitor hardware resources allocation and VICINITY components performance behaviour. Selected technology used in VICINITY components shall support monitoring of these resources through standard protocols and frameworks (such as SNMP, JMX, etc.), thus industry standard DEVOPS monitoring tools like Nagios or Microsoft System Center can be used. While VICINITY Nodes components might be unreachable by DEVOPS monitoring tools, logging mechanism shall be implemented to collect logs.

9.5. Security & Privacy

Security architecture of the VICINITY platform will define the security services and mechanisms enabling those services to run and provide the necessary protection and safe operation of the platform in the security context of VICINITY (Figure 34). This document gives an overview of the enabling security services of the planned architectural model and the enabling security mechanisms.

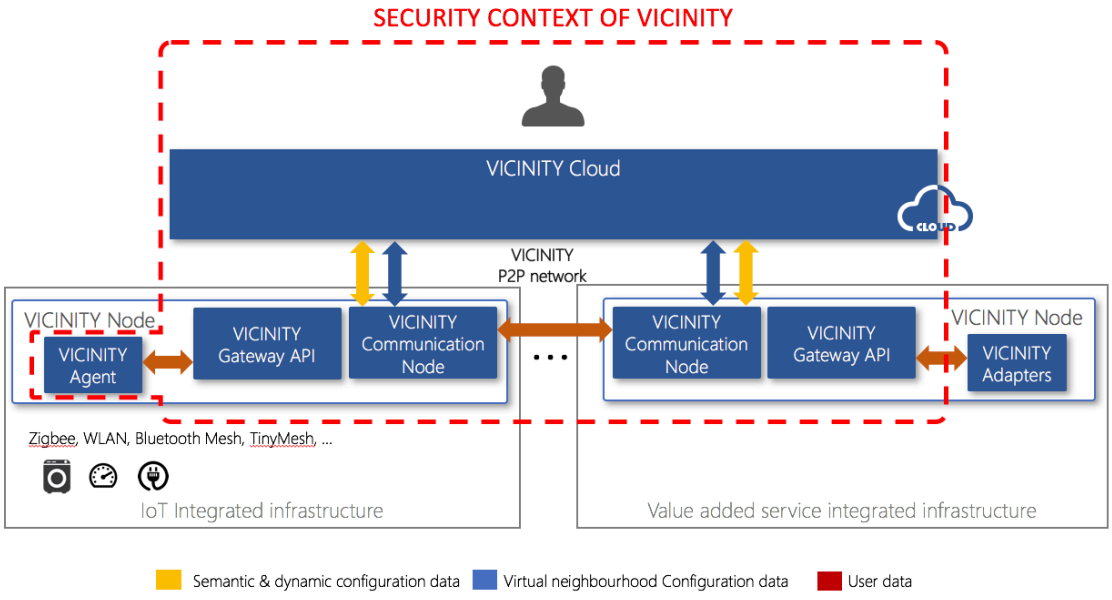

Figure 34 Security context of VICINITY

The security context of VICINITY Solutions focuses on VICINITY Cloud components, security of VICINITY P2P Network and VICINITY Node components including VICINITY Communication Node, VICINITY Gateway API and VICINITY Agent. The security service considers of IoT integrated infrastructures, proximity network, and devices indirectly through VICINITY Agent and VICINITY Gateway API in situation where they can pose relevant security risk.

The selected set of services proposed comes from best practices and is adapted to the requirements set within the VICINITY project. However, the final selection of the security component should be based on the requirements and the VICINITY solution implemented procedures for information exchange. The final selection and the security components will be provided in D4.3 VICINITY Security service of WP4.

The security architectural model with suggested security components is illustrated in Figure 35. The security services are implemented through the security mechanism. This mechanism implies selection of the logical security architectures and provision technology solutions and standards. Selection of the mechanism and provision of technology solutions and standards will be part of the security service implementation Task 4.3.

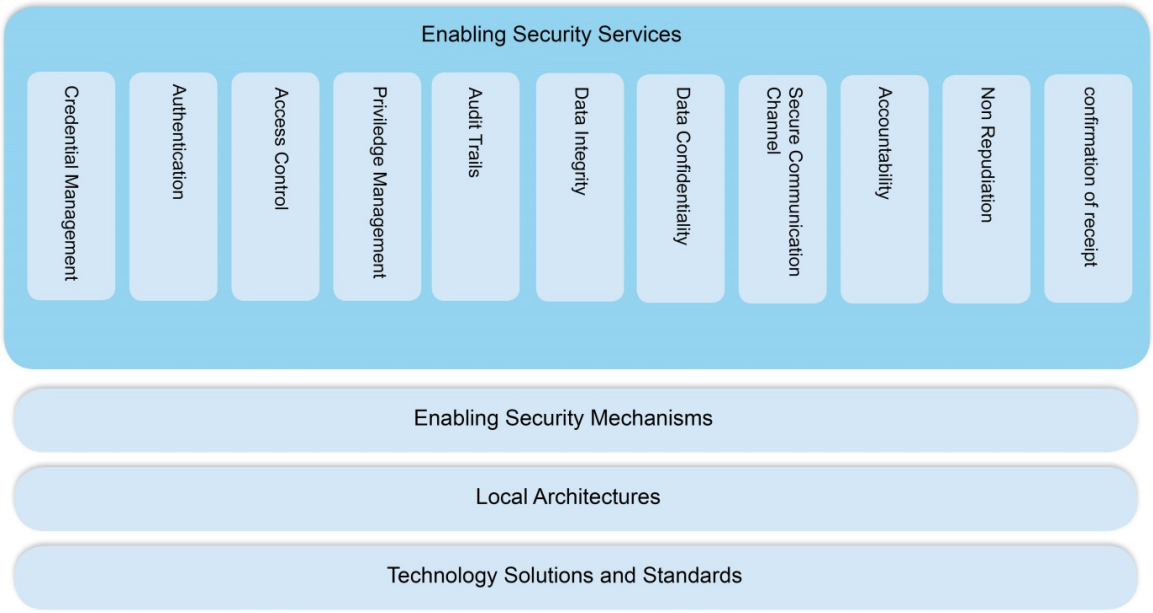

Figure 35 Security architecture

The security services can be mapped to AIOTI WG04[12] security policy recommendations as follows:

Table 16 VICINITY Security features mapped to AIOTI recommendations

AIOTI WG04 Security policy recommendation	VICINITY Position`
Embed 'safe and secure software' design and development methodologies across all levels of device/ application design and development and implement security into that life cycle at the same time.	VICINITY security has been analysed from technical specification (D1.5) and architecture design (9.5) point of view. Set of security requirements and security service has been identified and will be implemented in the WP4 – Task 4.3, security will be evaluated in WP6 – Task 6.4.
Design, deliver and operate	VICINITY shall provide following adaptive and dynamic end-to-

[12] www.aioti.org/wp-content/uploads/2016/10/AIOTIWG04Report2015.pdf

AIOTI WG04 Security policy recommendation	VICINITY Position`
adaptive and dynamic end–to-end security over heterogeneous infrastructures integrating IoT, networks and cloud infrastructures. We recommend underlying standardised OS and hardware security features where architecture permits. The deployment should not be specific or propose a modification of existing OS and hardware already integrated by IoT.	end security features: • controlling access to share objects (devices, value-added services) within VICINITY through sharing rules in virtual neighbourhood (D1.5 – UC0100, 3.1.1, 5.1.2); • notification and approval mechanism for any important change in virtual neighbourhood such as devices registration, request for data sharing (D1.5 – UC NTF010, 5.1.2); • adaptation to heterogeneous security features in infrastructures through agents and adapters.
Develop best practices confirming minimum requirements for provision of secure, encrypted and integrity-protected channel, mutual authentication processes between devices and measures securing that only authorised agents can change settings on communication and functionality.	VICINITY shall provides set of state of the art security services with ensures: • end-to-end encryption and data integrity services of exchanging of the user-data between integrated infrastructure (9.5.1.6, 9.5.1.8); • verification and credential management of the devices, application and VICINITY users identities (9.5.1.1, 9.5.1.2); • secure communication channel between VICINITY components and environment;
Develop a ‘New identity for Things’ – To date, Identity and Access Management (IAM) processes and infrastructure have been primarily focused on managing the identities of people. IAM processes and infrastructure must now be re-envisioned to encompass the amazing variety of the virtualized infrastructure components. For example, authentication and authorization functions will be expanded and enhanced to address people, software and devices as a single converged framework.	VICINITY will rely on existing identity management of devices of integrated infrastructures and their adaptation through adapters and agents supported by interoperability services of VICINITY Gateway API (3.3).
Develop a Common Authentication architecture – WG4 recommends investigation of a Secure Identity and Trusted Authentication mechanism, for example one which takes into account different	VICINITY shall rely on current authentication of value added services and services of integrated infrastructure through VICINITY Adapters on VICINITY Gateway API (9.5.1.1, 9.5.1.2, 9.5.1.3).

AIOTI WG04 Security policy recommendation	VICINITY Position`
authentication standards and will provide a single-sign-on solution for IoT applications moving between different systems.	
Certification – the certification framework and self-certification solutions for IoT applications have not been developed yet. The challenge will be to have generic and common framework, while developing business specific provisions. This framework should provide evaluation assurance levels similar to the Common Criteria for Information Technology Security Evaluation (IS0/IEC 15408), which should serve as the reference.	VICINITY shall rely on device, value-added services and organization profiles including information about device, value-added services and organization. These profiles might include security and privacy claims provided by device owner, service provider and organization. Profiles are accessible thorough VICINITY Neighbourhood manager and can be used during evaluation vicinity neighbourhood sharing rules.

The privacy concepts introduced in VICINITY can be mapped to AIOTI WG04 [13] privacy recommendations as follows:

Table 17 VICINITY Privacy features mapped to AIOTI recommendations

AIOTI WG04 Privacy recommendation	VICINITY Position`
Privacy Impact Assessments should be performed before any new IoT applications are launched.	VICINITY shall rely on Privacy Impact Assessments of organization and service provided. Result of the privacy impact shall be included in organization and service profiles.
Stakeholders must delete raw data as soon as they have extracted the data required for their data processing.	VICINITY shall not store any user data from integrated infrastructure, devices and service. VICINITY only facilitates exchange of the user data. Any private data stored in user, organization, device and service profiles can be updated by user and removed when not used any more if possible[14] (device removed, service removed, user sign out, organization sign out).
Every IoT stakeholder should apply the principles of Privacy by Design.	VICINITY privacy has been analysed from technical specification (D1.5) and architecture design (9.5) point of view. Set of

[13] www.aioti.org/wp-content/uploads/2016/10/AIOTIWG04Report2015.pdf

[14] legal and security measures can prevent to remove a private data if not actively used any more.

AIOTI WG04 Privacy recommendation	VICINITY Position`
	privacy requirements and privacy functions (mostly derived from GDPR[15]) has been identified and will be implemented in the WP3 – Task 3.1, privacy will be evaluated in WP6 – Task 6.4.
Data subjects and users must be able to exercise their rights and be "in control" of their data at any time.	VICINITY enables VICINITY user to revoke access to device and services at any time using VICINITY Neighbourhood manager (D1.5 – UC 0100).
The methods for giving information, offering a right to refuse consent should be made as user-friendly as possible.	VICINITY shall provide features to manage private data processing consent for device and service in VICINITY Neighbourhood Manager (D1.5 – UC PRV000).
Devices and applications should also be designed so as to inform users and non-user data-subjects.	VICINITY shall notify by VICINITY user (device owner, service provider, IoT operator) about any important changes or action required (accepting access to device or service) (D1.5 – UC NTF010).

9.5.1. Enabling security services and mechanism

This section list the following potential security services as part of the VICINITY architecture:

- Credential Management;
- Authentication;
- Access Control;
- Privilege Management;
- Audit Trails;
- Data Integrity;
- Data Confidentiality;
- Secure Communication Channel;
- Accountability;
- Non-Repudiation;
- Confirmation of Receipt.

The set of security services and their security mechanism is not final and will be adjusted throughout of the life cycle project and VICINITY solution based on project progress and needs.

Within each of the security service description short conceptual description of the security mechanism is provided.

[15] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)

9.5.1.1. *Authentication*

The security service authentication should ensure that entity (application, person) has the claimed for provided identity.

In authentication entities are considered:

- Users and technical users (software components, services, storages, applications);
- IoT objects shared within the VICINITY Neighbourhood.

VICINITY-ARCH-SEC-010	**Authentication**

The VICINITY shall support only standard based authentication mechanism. The VICINITY shall support he the following (one or more) mechanism:

- Username and password authentication;
- Digital certificate-based authentication (such as X.509 certificates);
- OAuth 2.0;
- JSON Web Token;
- SAML 2.0.

All entities should be subject of the authentication.

Considered requirements:
VICINITY-NFUNC-SEC010, VICINITY-NFUNC-SEC050

9.5.1.2. *Credential Management*

The credential management service shall manage the life cycle of entity credentials used in authentication and access control (authorization).

The credential management service shall include credential generation, registration, change, recovery, revocation and deletion.

VICINITY-ARCH-SEC-020	**Credential Management**

The VICINITY shall store and transmit passwords, tokens and keys in irreversible encrypted form. Password should have not been displayed on device display.

The password policy should be applied including at least following rules:

- Minimum length specification;
- Required character set;
- Password lifetime expiry;
- List of previous passwords;
- Passwords must be changed after initial login.

Considered requirements:
VICINITY-NFUNC-SEC010

The credential management service shall continuously revise requirements for security parameters of digital certificates and tokens based on the current best practice.

9.5.1.3. *Access (authorization) control*

The access control (authorization) service shall ensure that the entity has access to resources if and only if it is permitted.

VICINITY-ARCH-SEC030	Access (authorization) control

The access control mechanism shall support authorization based on:

- Identity – user access to VICINITY neighbourhood manage user interfaces and entities access to VICINITY internal services and resources (such as storage services, etc.);
- Role – entity authorization to functions of VICINITY Neighbourhood Manager based on users' roles;
- Context – such as entity authorization to IoT object properties, actions and events.

Considered requirements:

VICINITY-NFUNC-SEC020

The access control mechanism shall support distribute sharing access rules within P2P Network (5.1.2.1).

The access control mechanism shall secure fall back scenario when authentication mechanism is not temporary unavailable.

9.5.1.4. *Privilege Management*

The privilege management service shall support consistent and controlled mechanism to associate access rules to entities.

The privilege management shall define:

- Dedicated owner – each subject of the access control should have its owner;
- Reasonable defaults – default privileges should be selected sensible (e.g. organisation manager has privilege to add user to organisation);
- Explicit grant – implicit granting should be reduced to minimum;
- Privilege granting only by the owner;
- Privilege recovery – in case of the software component recovery the privileges should be recovered (i.e. recovery to previous state or most secure level);

9.5.1.5. *Audit trails*

VICINITY-ARCH-SEC040	Audit trails

The VICINIT shall provide means to support auditing in for important (which might be subject of relevant dispute) actions taken in VICINITY:

- VICINITY Cloud components;
- VICINITY P2P components;
- VICINITY Node components.

Considered requirements:

VICINITY-NFUNC-SEC060, VICINITY-NFUNC-SEC050, VICINITY-NFUNC-SEC070

9.5.1.6. *Data integrity*

VICINITY-ARCH-SEC050	Data integrity

The VICINIT shall provide means to secure data integrity of:

- Exchange of user data from data origin to data destination;
- Meta data exchange between components;
- And audit logs;

Considered requirements:

VICINITY-NFUNC-SEC040

9.5.1.7. *Accountability*

VICINITY-ARCH-SEC060	Accountability

Each action (such as service call, message exchange) performed should be perceived in context of the action originated entity.

Considered requirements:

VICINITY-NFUNC-SEC050

9.5.1.8. *Data confidentiality*

The data confidentiality service shall ensure that data is not disclosed to system entities unless they have been authorized to know the data.

VICINITY-ARCH-SEC070	Data confidentiality

The VICINITY shall apply strong encryption algorithms (publicly available and subjected of public review) to ensure data confidentially main on exchange of the user data.

The encryption algorithm parameters shall be configurable to support alignment with current best practice.

The encryption algorithm should be based on symmetric all asymmetric encryption schemas. Key management mechanism should ensure that data key is stored secure and reliable.

Keys should be cryptographically strong and generated by well understood algorithms with sufficient randomness only.

Considered requirements:

VICINITY-NFUNC-SEC030sa

9.5.1.9. *Secured communication channel*

The secured communication channel service is to ensure confidentiality, integrity, availability, authenticity and accountability of exchanged data between peers (architecture components, user and components).

VICINITY-ARCH-SEC080	Secured communication channel
The VICINITY shall support secured communication channels between architecture components, user and components including: • mutual authentication of peers; • adequate protection of exchanged data against eavesdropping and data tempering;	
Considered requirements:	
VICINITY-NFUNC-SEC030	

9.5.1.10. *Non-repudiations*

The non-repudiation service shall provide protection against false denial of involvement in an association.

The VICINIT shall provide means to support of non-repudiation for critical actions (such as: data processing consents, setting privilege, etc.) performed by (technical) user or user data exchange through the VICINITY Nodes (such as performing action, event reception, etc.).

10. Conclusions

The goal of the VICINITY Architecture includes description of VICINITY components and concepts from static and behavioural point of view. Architecture design encompasses specifications of components functions and interactions, implementation and deployment. VICINITY Architecture design should be interpreted together with D1.5 VICINITY Technical requirements specification.

The VICINITY Architecture has been broken down into several architecture components grouped in VICINITY Cloud components to include:

- VICINITY Neighbourhood manager;
- Semantic discovery & dynamic configuration agent platform;
- VICINITY Communication Server;
- VICINITY Gateway API Services;

and VICINITY Node components:

- VICINITY Communication Node;
- VICINITY Gateway API;
- VICINITY Agent and Adapter.

Each component was defined by set of principal functions and information flows needed to provide these functions. These all information flows together define processes executed between these components. The processes define set of internal and external interfaces of each component.

Detail architecture design of these components elaborate concepts of their features focusing on:

- User interface functionality in VICINITY Neighbourhood Manager;
- Managing of semantic model in Semantic discovery & dynamic configuration agent platform;
- Controlling of VICINITY P2P network by VICINITY Communication Server;
- Creating Virtual Thing Ecosystem Description in VICINITY Gateway API Services;
- Securing data forwarding between integrated infrastructures by VICINITY Communication Node;
- Integration of infrastructures into VICINITY by VICINITY Agent and Adapter.

The VICINITY architecture is supported by security, privacy, performance and availability concept such as:

- security transparently protects VICINITY itself and exchanged data, including role based access to data, data integrity and end-to-end security on VICINITY Communication Node supported by IoT objects description and sharing access rules;
- privacy by design to protect privacy of exchanged data between peers utilizing the VICINITY concept of end-to-end encryption between VICINITY Communication Nodes defined by IoT object descriptions and authorization based on sharing access rules provided by device owners and service providers;
- performance addressed by near to real-time message exchange in peer-to-peer network;
- availability of components deployed in high available scale out VICINITY cloud and loosely coupled P2P network of VICINITY Nodes supporting of localisation of component availability and performance issues.

VICINITY Technical requirements specification is an input for WP3 and WP4 to break down VICINITY solution into the smaller manageable software component for detail design and implementation.

Appendix A. Deployment of VICINITY Node on VICINITY Gateway

As shown in 7.2, the VICINITY Agent, running on each VICINITY Node, abstracts from device bundles (that is actual hardware components of IoT devices as well as their potentially closed-source drivers). This on one hand gives the opportunity to manufacturers to connect their devices to the VICINITY by simply providing adapters/drivers for their hardware to the VICINITY agent, without revealing details about their internal implementation (Figure 36). On the other hand, this offers software developers a uniform access to the attached hardware. Device discovery, offered by any software framework, may then take place amongst these device bundles. The communication between attached devices and the used software framework can use any arbitrary messaging protocols like e.g. CoAP, MQTT, REST, etc.

Finally, a VICINITY Node offers the VICINITY Gateway API for communication to/from the VICINITY Cloud. It also acts as a VICINITY communication Node to enable P2P communication amongst VICINITY Nodes. To this end, the VICINITY Agent ultimately needs to map the semantics of the used Software framework onto the VICINITY ontology.

Software and Hardware Frameworks are used to provide the necessary Gateway functionality. An overview on potential Hardware and Software platforms is given in VICINITY Deliverable D2.1 - Analysis of Standardisation Context and Recommendations for Standards Involvement - Chapter 2, which was submitted in September 2016.

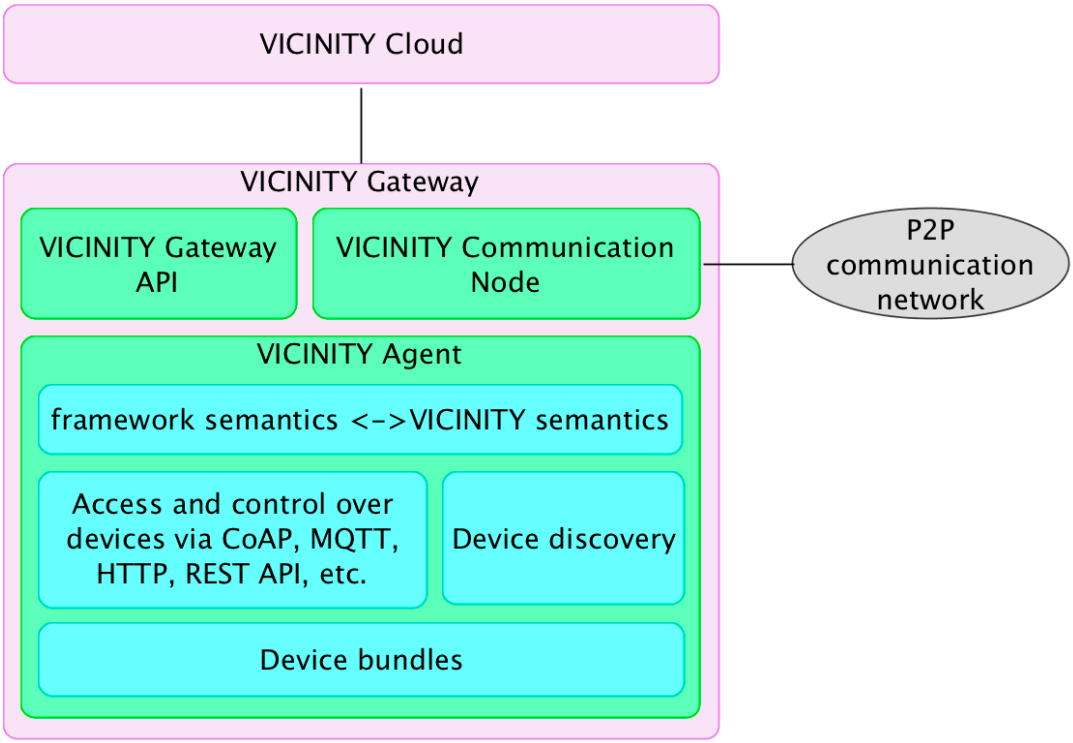

Figure 36 Deployment-Scenario of a VICINITY Node on VICINITY Gateway

Appendix B. Reference architecture

ISO/IEC is the standardisation body responsible for developing international standards. The sub-committee SC41 is responsible for Internet of Things, Sensor networks and Wearables. A standardised IoT reference architecture has been drafted in ISO/IEC 30141:2017 using a common vocabulary, reusable designs and industry best practices. It uses a top down approach, beginning with collecting the most important characteristics of IoT, abstracting those into a generic IoT conceptual model, deriving from the conceptual model to a high-level system based reference model and then breaking down from reference model to the five architecture views (functional view, system view, user view, information view and communication view) from different perspectives. The content of this appendix comes from the ISO/IEC standard (not from VICINITY) and the document is still under development (i.e., subject to changes).

Appendix B.1. IoT Reference architecture in relation to VICINITY Architecture

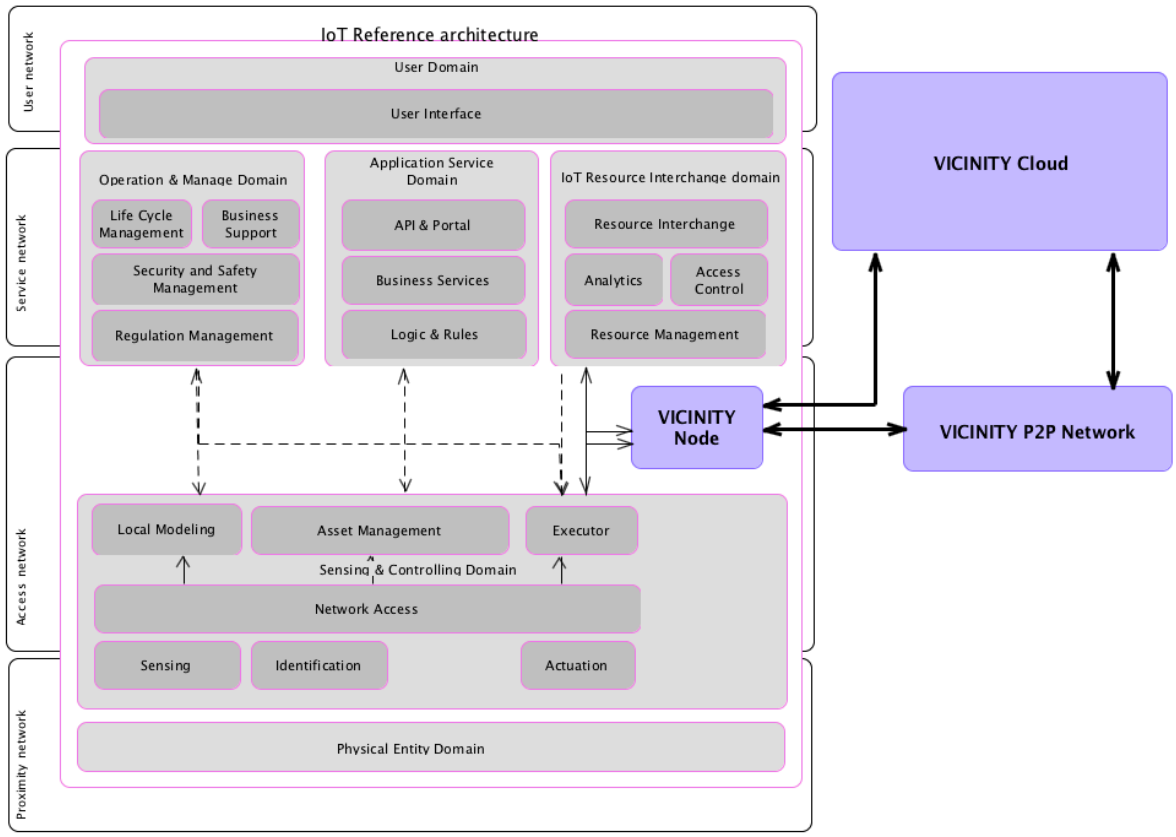

Figure 37 IoT Reference architecture in relation to VICINITY Architecture

From the IoT Reference architecture point of view, VICINITY interacts with the following domains through VICINITY Agents or Adapters in VICINITY Nodes:

- IoT Resource Interchange domain to access resources of integrated IoT infrastructure or value added service;
- Sensing & controlling domain to access or virtualize devices by VICINITY Agents and Adapters.

Appendix B.2. Introduction to IoT Reference architecture

The drafted standard covers the generalised Reference architecture of IoT, which is to serve as base when to develop (specify) context specific IoT architectures and then to actual systems. The contexts can be of different kinds, e.g., industry verticals or national specific requirement sets, see figure below.

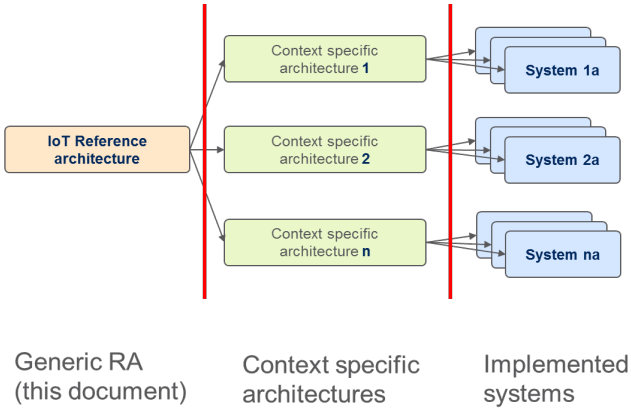

Figure 38 Context of IoT architectures

The IoT Reference Architecture (RA) describes a Conceptual Model (CM) containing common entities and their relations, and a Reference Model (RM) and different architecture views.

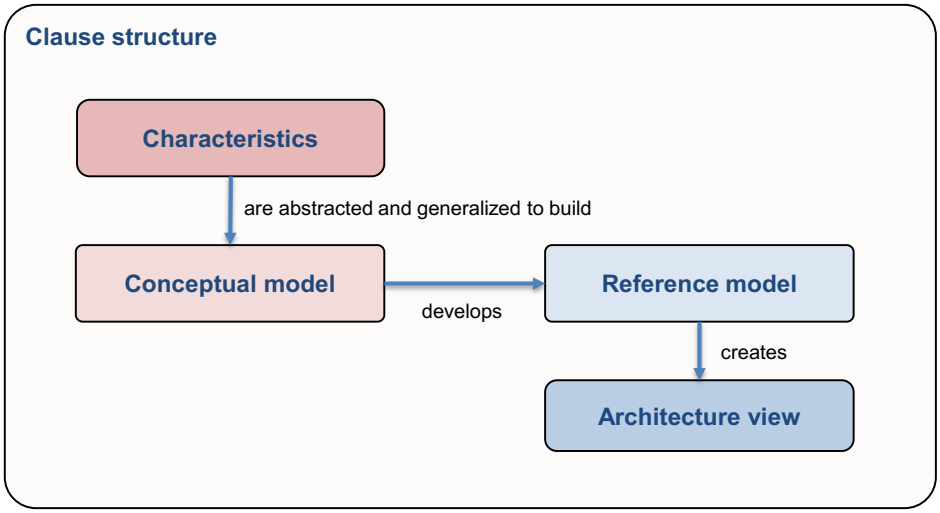

Figure 39 Conceptual model of IoT reference architecture

CM contains the following elements

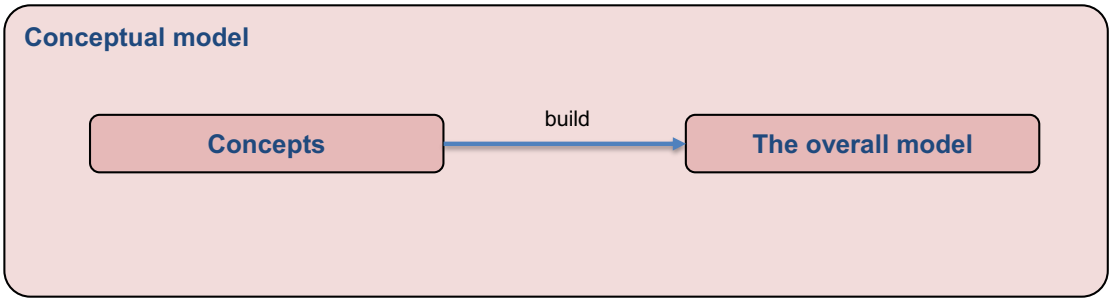

Figure 40 Relation between overall model and architecture concepts

The RM contains the following parts:

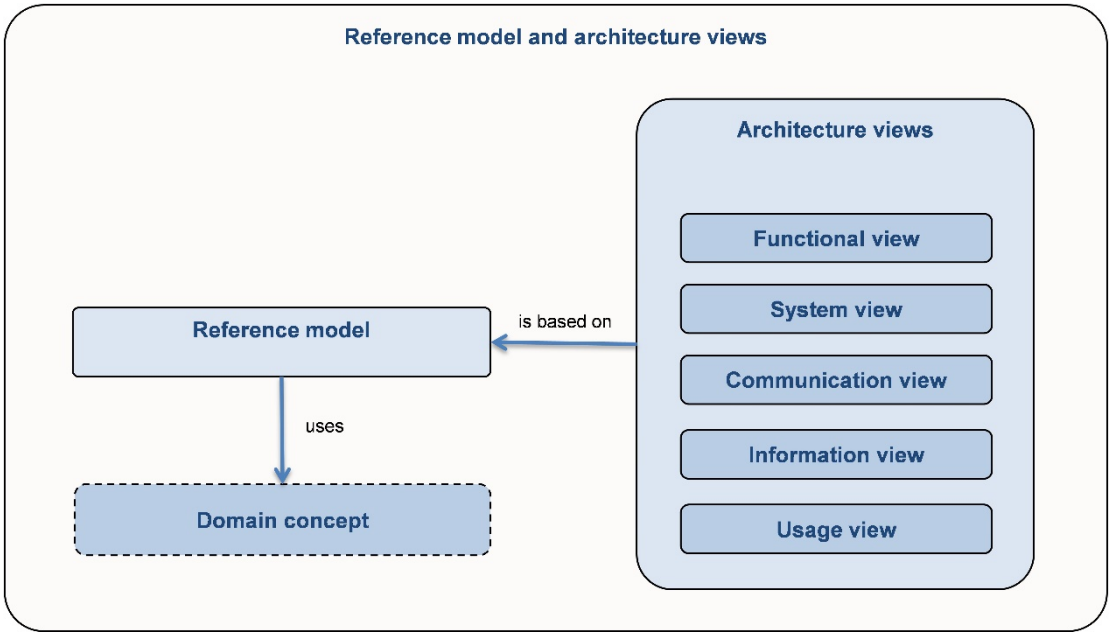

Figure 41 IoT architecture reference model of Architecture views

An IoT system is interoperable to other IoT systems. For this purpose, functions based on all or a part of these characteristics can be implemented in IoT systems according to services and operations. The characteristics of IoT systems is taken from ISO/IEC 30141:2017 and defined as

Auto-configuration is the automatic configuration of devices based on the interworking of predefined rules (associated algorithms based on data inputs). Auto-configuration includes automatic networking, automatic service provisioning and plug & play. Auto-configuration allows an IoT system to react on conditions and the addition and removal of components such as devices and networks. Auto-configuration needs security and authentication mechanisms to make sure that only authorised components can be auto-configured into the system. Security mechanism need to be organized appropriately for each market segment.

Auto-configuration is useful for IoT systems where there are many and varied components that can change over time and it benefits those users who expect robust systems because auto-configuration can allow automatic elimination of faulty components and maintenance of a working system.

Examples of auto-configuring devices and protocols include DHCP, Zero Configuration Networking (Zeroconf), Bonjour, UPnP ISO/IEC 29341 series etc.

Separation of **functional and management capabilities** means that the functional interfaces and capabilities of an IoT component, such as an IoT device, are cleanly separated from the management interfaces and capabilities of that component. This typically means that the management interface is on a different endpoint from that of the functional interface and the management capabilities are handled by different software components than the functional interfaces.

Management capabilities and functional capabilities have logically different

- purposes (execution/action vs information/description),
- user roles (control and modify behaviour vs transfer or consume facts and information),
- classification and types of data (technical or system specific vs personal/sensitive/public),

- access (e.g. an operator may access system configuration, but not gathered personal data; while the user can access the personal data but not access and modify system configuration)

- protocols, formats and lifecycle (e.g. support multiple control protocols vs meta-data/structure of the transferred information, which is particularly important considering interoperability and co-existence of multiple versions and variants of management capabilities)

Usually, the differences have associated specific risks and require special security (and other) controls, e.g. retention policy is applicable while dealing with functional data, but might not apply to management data; access control may be weaker for a user and stronger for an administrator).

Ubiquitous penetration of IoT into virtually all areas of life increases the attack surface, multiplying the number of potential attack targets and often making ineffective measures such as physical security controls. The key value of IoT – the connection of numerous edge components to each other and to IoT service components – increases the security concerns, since adding a weak link makes whole chain weak. Applications and systems previously running in well-protected data centres may become exposed to additional threats via connected IoT components.

Separation of management from functional capabilities enables or strengthens the ability to apply different authorization, authentication and protection mechanisms or constraints to management as opposed to functional capabilities. Broad sharing of data from an IoT system might be useful or desirable, and yet there are many circumstances where it is necessary to limit control of an IoT system or component to only a subset of the entities with which the data from that IoT system is shared.

If an IoT system is used to provide sensors and data for HVAC or other building management systems, it might be desirable to share data with other inter-related systems (alarms, access control, power management or auxiliary power, etc.), while still retaining management of the system to ensure system constraints are respected.

Distributed systems are the systems which, while being functionally integrated, consists of sub-systems which may be physically separated and remotely located from one another. These sub-systems are normally connected by a communication link (e.g. data bus). (ISO 3511-4)

IoT systems can span whole buildings, span whole cities, and even span the globe. Wide distribution can also apply to data – which can be stored at the edge of the network or stored centrally or a combination of the two. Distribution can also apply to processing – but processing can also takes place centrally (in cloud services), but processing can take place at the edge of the network, either in the IoT gateways or even within (more capable types of) sensors and actuators. Today there are officially more mobile devices than people in the world.

For industry 4.0, manufacturing can be done using smart manufactory systems which have distributed assembly lines across many factories and closely integrated with 3rd party suppliers, logistics companies, market providers and customers etc. all located far away from the assembly line" after.

Network communications – IoT systems depend on **network communications** of a number of different types. There are often limited range, low power networks collectively termed proximity networks that form the local connections for IoT devices. There are the wide area networks that connect the proximity networks to the internet, which can take wired and wireless forms and which may be dedicated to the IoT system or which may be shared general purpose networks.

Communication protocols used can vary between the different network types. It is common for proximity networks to use specialized protocols suited to the specialized nature of these networks. IP is more typically used for the wide area networks, although the higher levels in the protocol stack can vary, with HTTP being used in some cases, and messaging protocols being used in other cases. Some networks are deliberately intermittent in nature and the protocols used for such networks reflect the intermittent transmission pattern.

IoT systems rely on the ability to exchange information units in a structured manner based upon different but interoperable kinds of network types. Devices need to both transmit and receive data and need to communicate with software services that may be located nearby or in a remote location.

Gateways may be employed to connect networks of different types, typically between the proximity networks and the wide area. Network structure may need to be dynamic and needs to consider properties such as QoS, resilience, security and management capabilities.

In a proximity network, IoT devices can be connected by wireless technology, e.g., IEEE 802.15.4 and IEEE 802.11 in communication protocols on physical and data link layers. Data may be transported by 6LowPAN which is IoT specific IP and UDP. The IoT devices are then connected to a dedicated or general purpose wide area network via area a Gateway which routes data between the proximity network and the wide area network as necessary.

Network management - IoT systems require network management. The form and purpose of network management and operation depend on the network type and network ownership and the type of communication taking place over the network. Management is required during the setting up of a network, including the handling of device identity and addresses, profiles for the usage of the network and the inclusion of dynamic management capabilities. Management of the networks involves control over QoS, dynamic extension of the networks (for new or updated IoT devices), fault handling and security control. Networks must also handle dynamic and transitory membership of the network by mobile devices as those devices move into or out of the range of the network.

Some networks are managed as part of the IoT system – particularly the proximity networks connecting the IoT devices. Other networks, particularly the wide area networks, may not be managed as part of the IoT system, since they are general purpose networks often run by other organizations (e.g. mobile phone networks).

IoT network management has to span both kinds of networks and assemble them into a coherent system that can serve the purposes of the IoT system. Where IoT systems make use of third party general purpose communication networks, their management and operational interfaces can be used, where available.

Energy monitoring by smart meters is an example of a context where strict operation and management will be likely, since there is a commercial interest in such an IoT system being free of unauthorized activity. In such a context, all of the IoT devices, communication networks and information processing platforms are managed.

On the other hand, in case of home energy management, it is not necessary that the individual device be managed strictly. The management of the networks and information processing platforms of the vendor's support infrastructure will be done more as a means of selling more devices than as a profit-generating service in itself.

Real time capability is pertaining to a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results can be used to control, monitor, or respond in a timely manner to the external process. (ISO/IEC/IEEE 24765).

IoT systems often function in real time; data flows in continually about events in progress and there can be a need to produce timely responses to that stream of events. This may involve stream processing; acting on the event data as it arrives, comparing it against previous events and also against static data in order to react in the most appropriate way.

In process control systems, process parameters like temperature, flow, or pressure or status of a device are continuously monitored by sensors and instant actions are initiated.

Self-description is process by which components of an IoT system define their capabilities in order to inform other IoT components or other IoT systems for the purposes of composition and interoperability. Self-description includes interface specification, the capabilities of the IoT component, what types of devices can be connected to an IoT system, what kinds of service are made available by the IoT system, and the current state of the IoT system.

Self-description is needed for composability and interoperability for IoT systems and IoT devices. Self-description is of most benefit for those use cases where an IoT system needs to be interconnected with other IoT systems or those use cases where an IoT system benefits from being extended by the addition of new IoT devices. **Self-description** is also necessary for mobile devices and also for devices that hibernate – both of which join and leave networks on a regular basis.

Example of **self-description** for an IoT system and protocols: A system which uses Bluetooth in its proximity networks provides device name and supported service list to each other when connecting.

Wifi access points broadcast the SSID. Wi-Fi devices send passwords and MAC addresses to an access point when connecting to it.

Service subscription – It is often the case that IoT users subscribe to IoT services made available by IoT service providers. In this case, the IoT service providers make available a subscription process by which the IoT users can subscribe to a particular IoT service. The subscription process can include payments, plus a clear statement of any pre-requisites that apply to the IoT user. It can be the case that the IoT service involves the installation of IoT devices and the installation and configuration of software components – these are typically provided or specified by the IoT service provider.

In some alternative cases, the IoT user can establish their own IoT service, but in this case the IoT user has the burden of acquiring the necessary equipment and software and has the subsequent responsibilities for operating and maintaining the IoT service.

Some IoT systems are established on the basis of a subscription model where the IoT users pay for their use of the IoT system – in these cases, the IoT service provider must establish clear mechanisms for establishing and maintaining the subscriptions.

An example of a **subscription-oriented** IoT service is the provision of personal fitness monitoring, where the IoT user must purchase a wearable IoT device that is then connected to an IoT service that monitors their activity using the collected data and provides analysis and advice on how their activity is helping the user achieve life goals.

Content-Awareness is the property of having sufficient knowledge of the information in an IoT component and its associated meta-data. Devices and services with content-awareness are able to adapt interfaces, abstract application data, improve information retrieval precision, discover services, and enable appropriate user interactions.

Content-Awareness facilitates appropriate functional operations, such as data routing, speed of delivery, security capabilities such as encryption, based on factors such as location, quality of service requirements and sensitivity of data.

This capability can be essential in many applications including health services, broadcasting, surveillance systems and emergency services where some types of information or data flows have specific requirements with respect to timeliness, security and privacy.

Context-Awareness is the property of an IoT device, service or system being able to monitor the environment in which it is operating environment and events within that environment to determine information such as when (time awareness), where (location awareness), or in what order (awareness of sequence of events) one or more observations occurred in the physical world.

Context-Awareness enables flexible, user-customized and autonomic services based on the related context of IoT components and/or users. Context information is used as the basis for taking actions in response to observations, possibly through the use of sensor information and actuators. To fully utilize an observation and effect an action, the understanding of context is often critical.

An example of **context awareness** would be location-based services, such as a system in which different services are presented according to the location of a use

In cases of an emergency like a fire, the arrival of the fire service requires that the doors to a building shall be unlocked. The security policy that governs the door's access can be enhanced with context. The context here is that an emergency situation is currently happening and that the emergency services are in the vicinity. Based on these two contextual inputs the policy could enable the system to unlock the door automatically and provide access without the need for further authorisation.

Timeliness is the property of performing an action, function, or service within a specified period of time, which supports deterministic operations.

As IoT systems act on the physical world, some events initiated by the IoT systems need to occur at certain times, or within certain intervals of time, or in a particular sequence in relation to other events. To achieve this, the actions, functions, and services that lead to such events need to happen within specific time constraints. Timeliness in IoT includes not only latency related issues, but other aspects such as jitter, frequency/sampling rate, and phase.

An example in an industrial manufacturing process context might involve some sensors monitoring the quality of items flowing down an automated production line

In an industrial manufacturing process, an example is where some sensors are monitoring the quality of items flowing down an automated production line. Any items which are considered below the required quality must be removed from the line. The removal is performed by some actuators that divert the relevant items off the line. To achieve this, there is a strict time limit on commanding the actuators to perform the diversion – all the processing of sensor information and other relevant data must be completed within the time limit. Where IoT entities are part of any kind of control loop, overall processing time for the loop is critical.

Composability is the ability to combine discrete IoT components into an IoT system to achieve a set of goals and objectives.

System integration, interoperability and **composability** deal with how the functional components are assembled to form a complete IoT system and how the functional components connect to each other and the binding mechanisms which are used (e.g. dynamic or static, agent-based or P2P). Interoperability and composability are important topics in both the cyber and physical spaces. Composability imposes a stronger requirement than interoperability in that it requires components not only compatible in their interfaces but exchangeable with other components. At a minimum share similar components and provide improvements on any of the characteristics such as timing behaviours, performance, scalability and security. When a component is replaced by another of the same kind that is compostable and compatible the overall system functions should, at a minimum,

remain unchanged but consideration can be given to permitting improvements in system functions and characteristics.

An example of **composability** might be the ability to swap out sensor components from one vendor and replace them with sensor components produced by a different vendor. In this example, there might be two levels of composability.

First would be complete interchangeability of "commodity" functionality, such as an IoT device from Vendor A being fully replaceable with one from Vendor B.

A second level of composability (or possibly interoperability) might be an IoT control that is vendor-specific at the interface between the IoT component and a physical process device being controlled (a valve, motor, switch, pump or fan, for example), but is still fully interchangeable at the interface between the IoT device and the rest of the IoT system. In this sort of example, the IoT device would serve as a kind of "middleware" between the vendor-agnostic IoT infrastructure, and the vendor-specific physical devices or mechanisms being controlled.

Discoverability allows users, services, and other devices, to find not only devices on the network but also the capabilities and services they offer at any particular time. Discovery services allow IoT users, services, devices and data from devices to be located, identified, and accessed according to different criteria, such as geographic location, criteria, such as capabilities and geographic location.

Services connected with an IoT system can indicate what information can be found by a Discovery/Lookup service in accordance with predefined rules for each market segment. Discovery/Lookup services allow IoT systems to locate other devices, services or systems based on parameters such as geographical location, capabilities, interfaces, accessibility, ownership, security policy, operational configuration, data provided, data consumed, or other relevant factors.

IoT systems which support dynamic configuration, such as the addition of new devices and services to the IoT system, have a requirement for some form of discoverability, since there is a need to identify and characterize new components added to the system. So the addition of a new temperature sensor in a building monitoring IoT system is an example, where it is necessary to bring the new sensor into the existing system with minimum effort. Various protocols and software solutions exist to provide discovery in IoT systems, with a variety of architectures, some server based others being P2P. Examples include Hypercat, Alljoyn and Consul.

Modularity is when a component is a distinct unit that can be combined with other components or removed cleanly from the system and replaced with another module which occupies the same space and conforms to the same physical and logical interfaces.

Modularity allows components to be combined in different configurations to form systems as needed. By focusing on standardized interfaces and not specifying the internal workings of each component, implementers have flexibility in the design of components and IoT systems.

An example of **Modularity** in an IoT system might be a smart thermostat. Because the interface to an HVAC system and the interface to a larger IoT infrastructure could both be defined in compliance with open interface standards, there is nothing to prevent a thermostat from Vendor A being replaced by one from Vendor B. Furthermore, it is not important how the functionality of the device is implemented. Vendor A might provide the capability in the form of an ASIC-based state machine, while Vendor B's design might be based on a microcontroller. As long as both devices perform the same functions in response to the same inputs, and they are both compliant with open standard interfaces without imposing any proprietary constraints, there is nothing to prevent one from being replaced by the other.

Network connectivity – In IoT systems, components communicate with each other across network links. The connections between components are established using either wired or wireless media. Networked IoT devices that originate, route and terminate communications are described as (network) nodes. Endpoint network devices are the source or destination of any kind of information. Any IoT related networking communications protocol is layered onto more specific or more general communications protocols, down to the physical layer that directly deals with the transmission media at every network node.

IoT systems rely on the ability to exchange information in a structured manner based upon multiple different but interworkable network topologies – all within a physical, wired or wireless network. IoT devices are called "networked" when one device is able to exchange information with other devices whether or not they have a direct connection to each other. IoT network structure can be static or dynamic and may have capabilities such as QoS, resilience, encryption, authentication and authorisation.

The scale of an IoT network can vary substantially, from local proximity networks connecting a handful of devices over a limited distance, to global scale networks operating at Internet scale and connecting very large numbers of devices and service components.

It is typical for the networks in IoT systems to be heterogeneous and connected to each other via gateways or equivalent components.

Shareability is the ability to share the use of an individual component between multiple interconnected systems.

Many IoT components are underutilized since a single system often uses only a fraction of a component's capabilities. Resources can be used more efficiently if the functionalities or outputs of components can be shared among multiple systems.

The motion detection capabilities of a lighting control system could be leveraged by the security system to increase the security systems capability.

Temperature sensing for heating control could be used by the security system for fire detection.

Unique identification is the characteristic of an IoT system to unambiguously and repeatedly associate the entities within the system with an individual name, code, symbol, or number, and to interact with the entities, or trace or control their activities, by referencing that name, code, symbol or number. These entities include the components of the IoT system itself, such as the software components, the sensors and actuators and the network components.

It is essential that the entities in an IoT system can be distinguished from each other. This enables interoperability and global services across heterogeneous IoT systems. It is important for entities to be uniquely identifiable so that IoT systems can monitor and communicate with specific entities. A variety of identification schemes may be supported in specific implementations of IoT systems to meet the application requirements.

IPv4, IPv6, URI, and FQDNs are used as unique, unambiguous identification of network endpoints in internet applications. Individual hardware devices, software etc. may have unique manufacturer's IDs, OIDs, UUIDs or other identifiers, which similarly allow unique, unambiguous identification and can be used to tag data from those entities or direct commands to them.

Physical entities are often given unique identifiers in the form of RFID tags, barcodes and equivalent labelling technologies. For humans, biometric information can be used to provide unique identification.

Ownership – Asset management, according to ISO 55000, is the "coordinated activity of an organization to realise value from assets", where an asset is defined to be "… an item, thing or entity that has potential or actual value to an organization." Management implies responsibility and ownership.

From ISO 17799: "The asset owner is the person or group of people who have been identified by management as having responsibility for the maintenance of the confidentiality, availability and integrity of that asset. The asset owner may change during the lifecycle of the asset. The owner does not normally or necessarily personally own the asset. In most cases the employing organisation, its customers or suppliers will be the entity with property rights to the asset."

Ownership is also compassable. For example, an operational asset may incorporate assemblies or subassemblies that have different owners, but work in conjunction with each other. Using the concept of domain, ISO 27001:2005 states that the "… typical objectives of the asset management domain is to identify and create an inventory of all assets, establish an ownership on all assets identified, establish a set of rules for the acceptable use of assets, establish a framework for classification of assets, establish an asset labelling and handling guideline." An asset management domain represents a boundary where responsibility may transfer from one owner to another.

Legacy support is the concept that an IoT system might need to incorporate existing installed components even where these components embody technologies that are no longer standard or approved. A service, a protocol, a device, system, component, technology, or standard that is outdated but which is still in current use, may need to be incorporated into an IoT system.

Support of **legacy** component integration and migration can be important, although when supporting legacy components, it is also important to ensure that the design of new components and systems does not unnecessarily limit future system evolution. To prevent prematurely stranding legacy investment, a plan for adaptation and migration of legacy systems is important. Care ought to be taken when integrating legacy components to ensure that security and other essential performance and functional requirements are met. Legacy components may increase risk and vulnerabilities. Since current technology becomes legacy technology in the future it is important to have a process in place for managing legacy aspects of IoT. The different lifecycles of physical systems and information systems also creates additional challenges for managing legacy aspects in IoT.

One example of transition from legacy to future compatibility is the current slow rollover from IPv4 compliance to IPv6 compliance. The limits of the IPv4 address space and of the IPv4 protocol are known, and the transition to IPv6 is clearly the way of the future, but the varying pace of the transition, depending on the context, makes it a topic which can be very complex.

Many existing standards and application environments still assume and depend on IPv4, and yet it's clear that continuing to use IPv4 forever is not a viable strategy since there are insufficient addresses and running multiple devices behind a single IP address is not always appropriate. Deciding how and when to make the transition, however, is a topic that nobody has a universal answer to.

Well-defined components – IoT entities are deemed to be well-defined when an accurate description of their capabilities and characteristics is available, including any associated uncertainties. Capability information includes not only information about the specific component functionality, but configuration, communication, security, reliability and other relevant information.

Many components are used to assemble an IoT system. They are typically discovered through an information system interface. Without understanding the capabilities of each component that will be used within a system it is difficult to understand whether the system meets its design goals.

An example of an implementation of a **well-defined component** is: A particular IoT component is available with varying amounts of memory or support for various RF frequencies, waveforms and

protocols. Such a device has a baseline information interface which all the variants make use of to inform other IoT components of the list of capabilities possessed by the device. Once the devices' respective configurations have been exchanged, each device's software or applications can then self-adjust to take into account the capabilities of the other devices.

Flexibility is the capability of an IoT system, service, device or other component to provide a varied range of functionality, depending on need or context.

History and experience tell us that while there are exceptions, the economic and functional sweet spot for flexibility is usually somewhere in the middle, between the extremes of a dedicated single purpose component on one end of the spectrum, and a massively capable, programmable, extensible, "all things to all people" general purpose component at the other end.

It is possible to break down the general concept of flexibility into different sub-categories or dimensions.

One dimension of **flexibility** is the distinction between IoT capabilities hosted on a platform powered by a general purpose computing core and a similar capability implemented in the form of state machines implemented using discrete components, programmable FPGAs, or a purpose-specific ASIC. The state machine versions tend to be smaller, faster, more power efficient, and potentially more secure (due to a more limited range of capability). The general purpose version trades off speed, size, power consumption and other traits to gain more generalized capabilities, and a greater ability to adapt to meet unanticipated future requirements.

A second dimension of **flexibility** is illustrated by the distinction between the following kinds of device:

1) A device which has fixed, nonprogrammable, non-extensible functionality – "hard wired, single purpose".

2) A device which has fixed H/W capability, but which provides some amount of configurability within the single available format.

3) A device which is both programmable and expandable in the hardware domain – such as adding memory, adding more computational capability or adding RF channel capability.

4) A family of devices, each of which might fall into categories 1-3, from which an integrator can select the ones) which are appropriate for a given context.

5) A family of devices such as in 4, where some of the options provide different amounts of composability or modularity, at different levels of abstraction.

A third dimension of **flexibility** might involve the range of standards, protocols, formats, and interfaces which an IoT component is designed to support, where that support might then be designed and implemented taking the factors above into account.

Aside from the IoT component, there is another dimension of **flexibility** that involves the overall design of the IoT system. As in other domains, there will likely be open IoT ecosystems, and proprietary IoT ecosystems, with varying amounts of overlap between the two.

An example of differences regarding **flexibility** in the context of a sensor device is such as a thermostat. The simplest devices may only offer simple temperature control and reporting of temperature. More sophisticated and flexible thermostats allow for remote control via smartphone, can be connected to other IoT devices in the building to detect occupancy, to gain information about

the weather and so on – and these more capable devices typically have software components that can themselves be upgraded to offer newer capabilities.

Manageability addresses aspects of IoT systems such as device management, network management, system management, and interface maintenance and alerts. Manageability is important to meet IoT system requirements. Components capable of monitoring the system and changing configurations are needed for manageability of the IoT device, network and system.

Many IoT devices, networks, and systems are unmanned and run automatically. Special care must be taken to ensure that such systems remain manageable even when parts of the system malfunction, become unstable or mis-calibrated in the course of operation. Even in circumstances where individual IoT entities are accessible, the potentially large scale and geographic span of IoT systems argues for the ability to manage IoT entities remotely to the greatest extent possible, to increase both convenience and operational effectiveness.

IoT devices such as smoke sensors are deployed in various locations in buildings. These devices are often hard to maintain because of their locations. Any type of malfunction could cause undesirable events and consequences. Thus, remote manageability should be a system design consideration and goal from the beginning of specification, and throughout the development, and deployment, and operational lifecycle of the IoT system.

Additionally, software updates are necessary to ensure that devices and systems maintain functionality and the latest security vulnerabilities are patched. The manageability capabilities of an IoT entity might include device state monitoring capability, the link monitoring, calibration, etc. Update servers should be able to authenticate the IoT component and vice versa (i.e., mutual authentication). Updates should be digitally signed to ensure its authenticity and integrity. Updates should be transmitted over a secure channel, where possible

In the context of reliability, **accuracy** is the capability of an IoT device, service or system to provide sensor output calculations or actions within the expected range of acceptable precision in absolute terms, relative terms, or both.

An appropriate level of accuracy is essential to some IoT system deployments and applications. Depending on the context, differing degrees of accuracy might be required.

In a medical or manufacturing context, it might be critical for an IoT Device, application or system providing temperature information or control to be accurate to within a tenth of a degree Fahrenheit, while in a home HVAC context, accuracy to plus or minus two degrees might be adequate.

Reliability is the trait of a system exhibiting consistent behaviour – preferably the intended behaviour. An appropriate level of reliability in capabilities such as communication, service and data management capabilities is important to meet system requirements.

An appropriate level of **reliability** is essential in diverse IoT system deployments and applications. Reliability can be highly critical in some applications, e.g. for specific health related applications, industrial manufacturing operations and time-critical applications.

Reliability of data is of great importance for the decision-making processes of many IoT systems. The absence of data or data corruption can lead to incorrect decisions or the failure to make decisions. Reliability of communications networks is important for ensuring the availability and correct operation of IoT systems, particularly in mission-critical use cases.

Medical devices are one potential IoT application area where the specifications for mean time between failure might be quite stringent, due to the possibility of injury or death if an IoT device, application or system providing medical capability were to fail while a patient is being treated.

Resilience is the ability of an IoT system or its components to continue to perform their required function in the presence of faults and failures.

Communication, device or software component failures are to be expected in IoT systems and without appropriate design, they can escalate quickly causing the global failure of the system. IoT systems need to be designed for resilience, incorporating self-monitoring and self-healing techniques to improve the system resilience.

An IoT system has to be resilient to gateway failures to ensure continuing communications paths between software components and IoT devices.

One approach to resiliency is to adopt a master-slave design where if the master unit fails then a redundant device is available to assume the master role.

For networks, a mesh network design is resilient to the failure of one link or one node - data can still flow from source to sink through an alternative route.

Security – Availability is the ability of a system to be accessible and usable on demand by an authorized entity. IoT systems can include both human users and service components as "authorized entities".

In IoT systems availability can be seen in terms of devices, data and services. Availability of a device is related both to its inherent properties of operating correctly over time and to the network connectivity of the device. Availability of data is related to the ability of the system to get the requested data to and from a system component. Availability of services is related to the ability of the system to provide the requested service to users with a pre-defined QoS.

In some critical applications, e.g. health monitoring or intrusion detection, devices and data have to be always available so that alarms can be sent to the system immediately when raised. In these cases, system design must take into account potential failure modes and provide means of continuing operations, such as power supply backups, redundant devices, multiple instances of a service.

Confidentiality is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes.

In an IoT system confidentiality protection policies and mechanisms are responsible for prohibiting people or systems from reading data or control messages when they are not authorized to do so.

Confidentiality is a pre-requisite for a secure operation especially when the data to be transmitted contains secret tokens, e.g. for access control. Confidentiality is also required to protect sensitive data, which may include PII (e.g. financial information) or personal data (see the clause on Privacy).

Many items of data flowing an IoT system need to be treated as confidential – in the hands of the wrong recipient the data could be used for criminal acts or represent inappropriate use of personal data. For example, IoT motion detection sensors could reveal whether a property is occupied or not – which could be used by thieves to target the property.

Similar concerns relate to IoT smart meters – where even the frequency of messages transmitted should not depend on the rate of electricity use, since this could reveal whether a property is occupied or not.

Data integrity is the property that data has not been altered or destroyed in an unauthorized and undetected manner. [ISO_19790:2012, 3.58] Given that data is the basis on which IoT systems operate, tampering or destruction of data flowing or stored in the system could compromise the operation of the system and lead to highly undesirable outcomes.

Data integrity is vital for IoT systems to ensure that the data used for decision-making processes in the system and executable software has not been altered by faulty or unauthorized devices or by malicious actors. The protection of the integrity of the data and the executable software is a key requirement to ensure the security of the IoT system.

In IoT deployments that comprise of multi-hop wireless sensor networks there is a risk that intermediate nodes may alter the data and this can have impact on the functioning of the system. For example, an intermediate node may increase the value of the temperature of a room but this should not cause the air-conditioning system to increase the amount of cooling.

Safety is the freedom from risk which is not tolerable. Risk is the combination of probability of occurrence of harm and the severity of that harm. Harm includes injury or damage to the health of people, or damage to property or the environment. Harm can be due to malfunction, failure, or accident. While prior traits describe the desired behaviour of the system when operating correctly, Safety includes the consideration of failure modes with the intent of preventing, reducing or mitigating the potential for undesired outcomes; specifically, damage, harm or loss.

Many IoT systems are deployed in contexts or operational environments where damage, loss, injury or death might result if failure modes are not adequately addressed. In many operational contexts, approval to operate or approval to connect will not be granted if safety requirements are not met.

Even in contexts where compliance with safety standards is optional or voluntary rather than mandatory, proper consideration of safety factors may have significant impact on aspects such as: continuity of operations, reduction of loss, prevention of injury or death, insurance premiums, torts and liability, and other issues.

IoT contexts where safety standards or requirements might need to be considered include medical or health care applications, transport such as aviation and automotive applications, consumer products, buildings, and environment monitoring. Many countries will have specific regulations related to such applications.

Protection of personally identifiable information (PII) is a legal or regulatory requirement in most jurisdictions whenever an IoT system involves personally identifiable information anywhere in its operation.

Privacy is the right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed. (Based on ISO/TS 17574:2009, 3.16) The concept of privacy overlaps, but does not completely coincide, with the concept of data. With respect to data protection it ensures that PII is not processed without the informed consent of the, individual, unless otherwise permitted by law, and is not and is not disclosed to unauthorized entities. For IoT systems, entities include both people, machines and processes.

The principle of data minimisation applies to PII: the quantity of PII collected is the minimal necessary to support the application. The PII which is necessarily present should be securely deleted when no longer needed. This protects the individual and minimizes legal risk to the organization using the PII. If PII is disclosed it must be based on prior informed consent given by the PII principal for the intended purpose.

Meta-data may include PII. As such, organizations processing such data must take steps to ensure that any such processing conforms to applicable privacy/data protection legislation or regulation. This includes taking steps to ensure that all such data is adequately safeguarded

Many IoT systems do not collect or interchange PII. However, any IoT system which does collect, receive and/or interchange personal information needs to ensure that such IoT systems and their interactions with other IoT systems (or IT systems in general) are in full compliance with privacy protection requirements of applicable jurisdictions domains.

One aspect of IoT systems is that the nature of the data handled by the system can be unclear. For example, a home automation IoT system may appear to be dealing in data that is not PII, but if (say) the electrical usage data of a house is present in the system, if the data can be connected with a specific house, it is likely that the data is connected with specific people and can be regarded as PII.

IoT systems need careful analysis to understand if any of the data they handle is or is potentially PII. If PII is present, then the IoT system must be designed to meet appropriate data protection regulations and laws in the relevant jurisdiction(s).

Many IoT applications involve end-users and the collection of specific data relating to them. For example, traffic speed cameras record a number plate and often an image of the driver's face. This information is correlated with licensing records to allow fines to be levied. However, such data cannot be retained beyond a pre-defined time and should not be made available for other purposes. Mobile phone location can be tracked and while this can be useful for a user to receive information about facilities in the area, access to such information should be controlled; it may be required for police investigations but users may not want to receive adverts for local venues.

In particular, with healthcare monitoring and other such monitoring of specific individuals there is a need for the data to be provided only for the agreed purpose for example to update a GP or personal healthcare record and not for use by other institutions such as insurance companies.

Driver may be providing data for traffic monitoring systems (location and speed) allowing traffic congestion to be reported but would not necessarily expect this data to be linked to an employer's system. Similarly, tracking people movement in offices may be possible with a building surveillance and access system but noting times of rest breaks etc., may not have been part of the purpose of the data collection.

Smart metering applications are another example where an individual may grant access to data for a particular purpose. The smart meter is collecting real-time information about electricity usage in the home and transmitting it to the electricity utility, who may use the data for a variety of purposes, including demand management and differential pricing. It is clear that the data relates to the people living in that home and may reveal significant details about their lives. It is necessary for the electricity utility organization to inform the householder about the PII they are gathering and to be clear about its use. The electricity utility also needs to apply appropriate protection to the data (e.g. encrypt data streams flowing from the smart meter), and apply privacy principles to the processing of the data, including minimising the data, anonymizing the data and deleting the data as soon as possible.

Several governments and group of states has issues laws based on the European directive 2016/680 on Privacy, especially to protect citizens from the increasing exposures of their PII in the daily digital life on the net. A set of laws to be implemented to provide rules for the business how to handle PII and make it certain that PII is not exposed more than the business relation requires.

Other characteristics – The "Data 5Vs" of volume, velocity, veracity, variability and variety often apply to IoT systems. The Data 5Vs derive from Big Data systems – but it is often the case that IoT

systems are the source of data which is large in volume, delivered at speed across network links, whose veracity needs to be validated (e.g. due to malfunctioning sensors), which can vary over time and can contain a wide variety of different data types from different IoT components.

IoT Systems are also expected to generate large amounts of data from diverse locations. The data may be aggregated into centralized locations or it may be stored in distributed locations (depending on the nature of the data, the processing required on the data and the communication link characteristics), which generates a need to appropriately index, store, process and secure the data.

A logistics company uses big data analytics for an On-Road Integrated Optimization and Navigation service. The system uses numerous address data points, plus other data collected during deliveries, to optimize delivery routes.

Heterogeneity – An IoT system typically is composed of a diverse set of components and physical entities that interact in various ways.

IoT is typically cross-system, cross-product, and cross-domain. Realizing the full potential of IoT requires interoperability between heterogeneous components and systems. This heterogeneity creates numerous challenges for the resulting IoT systems.

A smart container using RFID tags for identity and related RFID sensors needs interworking of RFID systems and sensor network systems.

Regulatory compliance – IoT systems, services, components and applications can be deployed in circumstances which require adherence to a variety of laws, policies or regulations. Such support might be inherent in the IoT device or system, or might require specific configuration, programming, modification or extension to ensure compliance.

Additionally, there might be a range of different granularity or levels of abstraction at which the regulations are applied or enforced.

Regulations of relevance to IoT systems might take many forms, including regulations to assure interoperability, to mandate or constrain functionality or capability, to assess the ability of the IoT device or system to function in a certain usage context without causing damage, and to impose at least minimal balance between contribution to the collective good and self-interest on the part of system owners or operators.

Regulations which might apply to an IoT context include one or more of the following categories:

1) Safety regulations – These might include flight safety standards for IoT devices operating in aircraft, or regulations covering the manufacture and sale of devices intended for consumer use in the home, regulations for automotive systems, or regulations for devices or systems used in a medical context.

2) RF related regulations – This category might include national or international regulations governing RF emanations, adherence to frequency band restrictions, signal strength, spurious signals (such as side channels, noise, or harmonics produced outside of the device's nominal frequency allocation), etc.

3) Consumer protection regulations– These might include national and international regulations invoked whenever an IoT system involves a consumer anywhere in its operation.

In some IoT contexts, such as home automation, HVAC, etc. another layer of regulations might be imposed in the form of building codes in various jurisdictions.

While the area is still developing, it is quite possible that at some point, there will be regulations imposed or referenced by insurance companies as part of their risk models for pricing coverage of structures, vehicles, systems, or businesses incorporating IoT systems and devices.

Scalability is the characteristic of a system to continue to work effectively as the size of the system, its complexity or the volume of work performed by the system is increased.

IoT systems involve various elements such as devices, networks, services, applications, users, stored data, data traffic, event reports. The amount of each of these elements can change over time and it is important that the IoT system continues to function effectively when the amounts increase.

One example of **scalability** is when the number of sensor devices attached to an IoT system is increased. If a system changes from monitoring temperature sensors in a single building to monitoring temperature sensors on all buildings in a city there will be a significant increase in the volume of sensor data flowing in the system, in the volume of data being stored in databases, in the number of devices handled by the management system, and in the number of temperature readings processed by services and applications.

Trustworthiness is the degree to which a user or other stakeholder has confidence that a product or system will behave as intended.

Device, data and service trustworthiness is of utmost importance for IoT systems to ensure that only trusted devices participate in the decision-making process of the system, resulting in the provision of trustworthy applications. Device executable processes and data must be trusted to ensure that the device/system operates as intended.

Devices may become untrusted due to compromise, outdated software or firmware, or other reasons. IoT systems must be able to identify untrusted devices and must implement policies, processes, procedures and/or mechanisms (e.g., quarantine) to address the issue of untrusted devices on the system

Where an IoT system that monitors the average measurement of a room taking the mean value reported by x sensors, if y sensors report false values, due to a fault or malicious programming the resulting mean measurement will be false. Detection, assessment and potential exclusion of anomalous readings is necessary to ensure trustworthy data.

Appendix C. Register