This project has received funding from the European's Union Horizon 2020 research innovation programme under Grant Agreement No. 957258

Architecture for Scalable, Self-human-centric, Intelligent, Secure, and Tactile next generation IoT

D3.5 ASSIST-IoT Architecture Definition – Initial

Deliverable No.	D3.5	**Due Date**	30-APR-2021
Type	Report	**Dissemination Level**	Public
Version	1.0	**WP**	WP3
Description	Initial specification of the ASSIST-IoT technical architecture and its components.		

Copyright

UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Spain
PRODEVELOP S.L.	Spain
SYSTEMS RESEARCH INSTITUTE POLISH ACADEMY OF SCIENCES IBS PAN	Poland
ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS	Greece
TERMINAL LINK SAS	France
INFOLYSIS P.C.	Greece
CENTRALNY INSTYUT OCHRONY PRACY	Poland
MOSTOSTAL WARSZAWA S.A.	Poland
NEWAYS TECHNOLOGIES BV	Netherlands
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS	Greece
KONECRANES FINLAND OY	Finland
FORD-WERKE GMBH	Germany
GRUPO S 21SEC GESTION SA	Spain
TWOTRONIC GMBH	Germany
ORANGE POLSKA SPOLKA AKCYJNA	Poland

Disclaimer

Authors

Name	Partner	e-mail
Alejandro Fornés	P01 UPV	alforlea@upv.es
Ignacio Lacalle	P01 UPV	iglaub@upv.es
César López	P01 UPV	csalpepi@upv.es
Carlos E. Palau	P01 UPV	cpalau@dcom.upv.es
Eduardo Garro	P02 PRO	egarro@prodevelop.es
Maria Ganzha	P03 IBSPAN	maria.ganzha@ibspan.waw.pl
Paweł Szmeja	P03 IBSPAN	pawel.szmeja@ibspan.waw.pl
Piotr Lewandowski	P03 IBSPAN	piotr.lewandowski@ibspan.waw.pl
Georgios Stavropoulos	P04 CERTH	stavrop@iti.gr
Iordanis Papoutsoglou	P04 CERTH	ipapoutsoglou@iti.gr
Theoni Dounia	P06 INFOLYSIS	tdounia@infolysis.gr
Nick Vrionis	P06 INFOLYSIS	nvrionis@infolysis.gr
Alex van den Heuvel	P09 NEWAYS	alex.van.den.heuvel@newayselectronics.com
Ron Schram	P09 NEWAYS	ron.schram@newayselectronics.com
Konstantinos Naskou	P11 ICCS	konstantinos.naskou@iccs.gr
Óscar López	P13 S21SEC	olopez@s21sec.com
Zbigniew Kopertowski	P15 OPL	Zbigniew.Kopertowski@orange.com

History

Date	Version	Change
26-Feb-2021	0.1	ToC presented
4- Mar-2021	0.2	ToC updated and assignments of sections after discussion in 2nd Plenary
17-Mar-2021	0.3	Merged first round of contributions
30-Mar-2021	0.4	Merged second round of contributions
9-Apr-2021	0.5	Version sent for Internal Review
27-Apr-2021	0.6-7-8	Iterative versions during review processes
28-Apr-2021	0.9	Version sent to PIC for last check
30-Apr-2021	1.0	Final version submitted to EC

Key Data

Keywords	Reference architecture, enablers
Lead Editor	P01 UPV - Ignacio Lacalle
Internal Reviewer(s)	P04 CERTH, P12 FORD-WERKE

Executive Summary

This deliverable is written in the framework of WP3 – Requirements, Specification and Architecture of **ASSIST-IoT** project under Grant Agreement No. 957258. The document is the first of a series that are devoted to formalising ASSIST-IoT technological architecture. This document aims at outlining the guiding principles of ASSIST-IoT architecture, altogether with the identification of main elements (Views) and enablers to be part of it.

Next Generation Internet of Things field is urgently requiring the creation of a robust, formal, valid, useful reference architecture to build deployments upon. Several initiatives are detected, as well as classic concepts of architecture definition are varyingly meeting requirements. However, the blueprint is yet to come.

This deliverable reports the works done towards defining ASSIST-IoT architecture guiding principles and key considerations for its deployment, which has redounded in a two-dimensional structure divided in planes and verticals within which the actual technological assets will be included. These assets have been defined (novel approach) as enablers, which aim at encapsulating the innovative functionalities that ASSIST-IoT's provides.

Throughout the core sections of the document, there can be found the motivated decisions that will guide the architecture in ASSIST-IoT: the selection of service-based philosophy, the use of containerisation and the rationale behind the selection of Functional, Node and Deployment View, among others.

In addition, deep study of each plane and vertical has been included. The results of the research have allowed to describe the key enablers, digging into their objective and candidate technologies (e.g., custom components on Apache Kafka, Scala, Akka, Apache Jena to create the Semantic translation enabler in the Data Management plane).

In order to establish a language for a common understanding of the terms, a Glossary has been created and accompanies this document as appendix.

This document aims at being the starting point for further improvements and refinements that will come later during the project. Thus, to finalise it, a clear roadmap of forthcoming actions is included, altogether with some early conclusions obtained.

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
3D	Three Dimensional
5GC	5G Core
AI	Artificial Intelligence
AIOTI	Alliance for the Internet of Things Innovation
API	Application Programming Interface
AR	Augmented Reality
BSS	Business Support Systems
CLI	Command Line Interface
CNF	Cloud-Native Network Functions
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DAG	Directed Acyclic Graph
DHCP	Dynamic Host Configuration Protocol
DLT	Distributed Ledger Technology
ECC	Edge Computing Consortium
ETSI	European Telecommunications Standards Institute
FL	Federated Learning
FPGA	Field Programmable Gate Arrays
GA	Grant Agreement
GKE	Google Kubernetes Engine
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HLA	High Level Architecture
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IBN	Intent-Based Networking
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
ISO	International Organization for Standardization

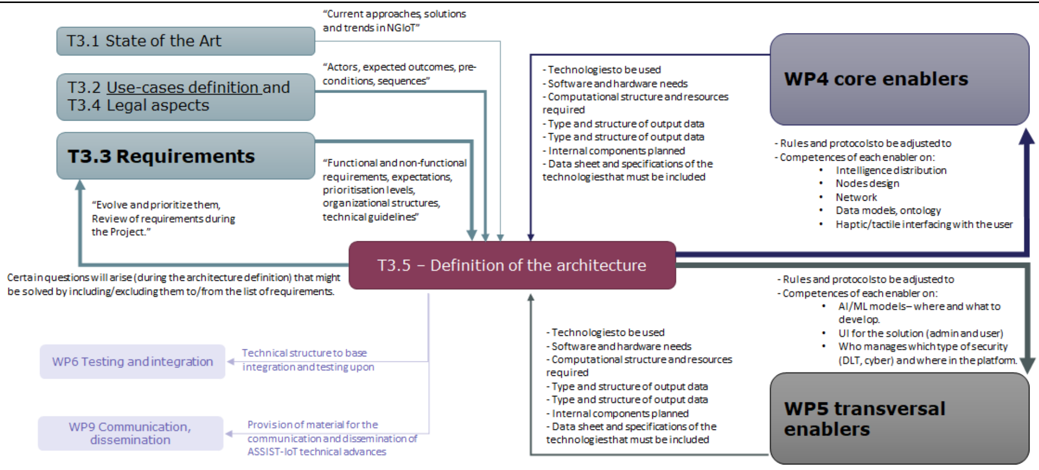
ITU	International Telecommunication Union
KNF	Kubernetes-based VNFs
KPI	Key Performance Indicator
LIDAR	Light Detection and Ranging
LSP	Large-Scale Pilots
LTE	Long Term Evolution
MANO	Management and Network Orchestration
MPLS	Multi-Protocol Label Switching
MR	Mixed Reality
NFV	Network Function Virtualisation
NFVI	Network Function Virtualisation Infrastructure
NFVO	Network Function Virtualisation Orchestrator
NGIoT	Next Generation Internet of Things
NS	Network Service
OHS	Occupational Health and Safety
ONAP	Open Network Automation Platform
OPNFV	Open Platform for NFV
OS	Operative System
OSS	Operations Support Systems
OSI	Open Systems Interconnection
OSM	Open Source MANO
PAP	Policy Administration Point
PDP	Policy Decision Point
PIP	Policy Information Point
PLC	Programmable Logic Controller
PNF	Physical Network Function
PoP	Point of Presence
QoE	Quality of Experience
QoS	Quality of Service
RA	Reference Architecture
RAMEC	Reference Architecture Model Edge Computing
REST	Representational State Transfer
RPM	Revolutions per Minute
SaaS	Software-as-a-Service
SDN	Software-Defined Networking
SDO	Standards Developing Organisations

SOA	Service Oriented Architecture
SD-WAN	Software-Defined Wide Area Network
TCP	Transmission Control Protocol
TSN	Time-Sensitive Networking
TPU	Tensor Processing Unit
UI	User Interface
UML	Unified Modelling Language
USB	Universal Serial Bus
VHDL	Very High-Speed Integrated Circuit (VHSIC) Hardware Description Language
VIM	Virtualised Infrastructure Manager
VM	Virtual Machine
VNF	Virtualised Network Function
VNFM	Virtualised Network Function Manager
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
VR	Virtual Reality
WAN	Wide Area Network
WIM	WAN Infrastructure Manager
WP	Work Package
XaaS	Anything-as-a-Service
XACML	eXtensible Access Control Markup Language
YAML	Yaml Ain't Markup Language

1. About this document

The main objective of this document is **to set the foundations of the ASSIST-IoT architecture**. Considering that the most prominent outcome of the project will be its blueprint architecture, this deliverable must be key for understanding the rest of technical content of ASSIST-IoT. More specifically, D3.5 aims at being a first step in the description of the architecture. While deep understanding of the diverse elements of the ASSIST-IoT architecture will need further elaboration and joint interpretation with WP4, WP5 and WP6, this deliverable (D3.5) depicts the guiding principles, the base structure, and the main elements (with their role and fit) of the architecture. In other words, the backbone of the technical provision of the project is defined in this deliverable.

1.1. Deliverable context

Keywords	**Lead Editor**
Objectives	**O1**: D3.5 is the first definition of the architecture, that is the outcome of objective 1. **O2**: Smart network's Functional View is provided with overview of its core enablers. **O3**: Security and Privacy vertical capabilities are described with overview of main enablers. **O4**: Federation of smart AI enablers is preliminary outlined.
Work plan	
Milestones	This deliverable does not mark any specific milestone completion; however it contributes towards *MS5 – Final architecture defined*, that will be achieved by submitting D3.7.
Deliverables	This deliverable is fed by the elaboration of D3.2 (use-cases and requirements) and will serve as the basis for the forthcoming updates of the architecture in D3.6 and D3.7. It will also be feeding developments in WP4 and WP5, inspiring the production of D4.1 and D5.1.

T3.1 State of the Art
"Current approaches, solutions and trends in NGIoT"
T3.2 Use-cases definition and T3.4 Legal aspects
"Actors, expected outcomes, pre-conditions, sequences"
T3.3 Requirements
"Functional and non-functional requirements, expectations, prioritisation levels, organizational structures, technical guidelines"
"Evolve and prioritize them, Review of requirements during the Project."
Certain questions will arise (during the architecture definition) that might be solved by including/excluding them to/from the list of requirements.
T3.5 – Definition of the architecture
- Technologies to be used
- Software and hardware needs
- Computational structure and resources required
- Type and structure of output data
- Type and structure of output data
- Internal components planned
- Data sheet and specifications of the technologies that must be included
WP4 core enablers
- Rules and protocols to be adjusted to
- Competences of each enabler on:
• Intelligence distribution
• Nodes design
• Network
• Data models, ontology
• Haptic/tactile interfacing with the user
- Rules and protocols to be adjusted to
- Competences of each enabler on:
• AI/ML models – where and what to develop.
• UI for the solution (admin and user)
• Who manages which type of security (DLT, cyber) and where in the platform.
- Technologies to be used
- Software and hardware needs
- Computational structure and resources required
- Type and structure of output data
- Type and structure of output data
- Internal components planned
- Data sheet and specifications of the technologies that must be included
WP5 transversal enablers
WP6 Testing and integration
Technical structure to base integration and testing upon
WP9 Communication, dissemination
Provision of material for the communication and dissemination of ASSIST-IoT technical advances

1.2. The rationale behind the structure

As mentioned in the previous section, deliverable D3.5 aims at laying the first grounds in the description of the ASSIST-IoT architecture. This deliverable D3.5 will be updated in D3.6 and D3.7, that will redound in a formal Reference Architecture (RA) definition document. Therefore, it has been adopted a structure aligned with usual formal description of a RA, including the definition of guiding principles and views of the architecture.

Section 2 serves as an overview of RA design, comparing the different approaches. **Section 3** starts by mapping those approaches with ASSIST-IoT, defining the principles adopted and the overall structure of ASSIST-IoT architecture and explaining the essential cornerstone elements (enablers). **In Section 4**, the different transversal NGIoT concerns devised for ASSIST-IoT are tackled, including the description of main enablers. **Sections 5, 6 and 7** are devoted to explaining the four different views existing in ASSIST-IoT: Functional View (including main enablers), Node View and Deployment View.

2. Reference Architecture Design Principles

For the ASSIST-IoT architecture to be built upon, key principles of established Reference Architectures (RA) will be leveraged. This section's main objective is to provide an overview of the main concepts exploited to conceive RAs, as well as to introduce the main architecture paradigms, available in the market, for building systems intended for different scenarios. The selected design choices, abstractions and methodology of the concepts and approaches upon which ASSIST-IoT will be built are presented in detail in Section 3, while the main outcomes lie in the subsequent sections.

2.1. Concepts

Applied to the IoT ecosystem, a RA can be described as a useful model that can serve as a set of guidelines to implement an IoT system. Their objective consists of assessing a set of requirements in order to provide a set of functionalities, information structures and mechanisms [1], serving as a blueprint for developing and implementing IoT architectures. Reference Architectures usually have a high level of abstraction, so they can be applied for different domains or application, setting the ground for discussing under a common vocabulary. The standard ISO/IEC/IEEE 42010 [2], which is leveraged in many modern RAs as was reviewed in deliverable D3.1, specifies a conceptual model to aid in the description of architectures:

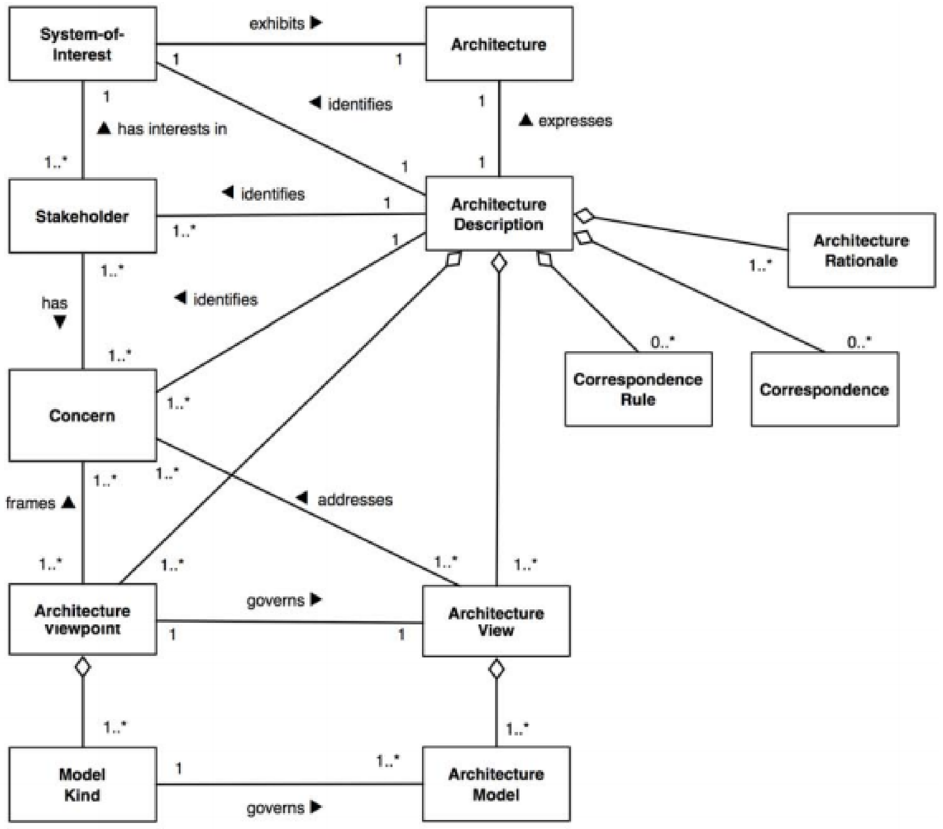


Figure 1. Conceptual model of an architectural description defined in ISO/IEC/IEEE 42010 [2]

The most vital concepts in defining an architecture from the standard are summarised below:

- A **stakeholder** in the architecture of a system is an individual, team, organisation, or classes thereof, having an interest in a system [2]. Interests also referred to as concerns are defined the next point. The architect must ensure that there is adequate stakeholder representation across the board, including nontechnology stakeholders (such as acquirers and users) and technology-focused ones (such as developers, system administrators, and maintainers). Stakeholders can range from developers, testers and maintainers to support staff, administrators, product engineers and end users, among others [3].

- A **concern** is a topic of interest to one or more stakeholders belonging to an architecture [4]. A concern could be manifest in many forms, such as in relation to one or more stakeholder's needs, goals, expectations, responsibilities, requirements, design constraints, assumptions, dependencies, quality attributes, architecture decisions, risks or other issues pertaining to the system [2].
- A **viewpoint** is a work product establishing the conventions for the construction, interpretation and use of architecture Views to frame specific system concerns [2]. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views [3]. The functional viewpoint is always present in architecture descriptions (which describes the functional elements of a system, along with their responsibilities, interfaces, and main interactions). Because of the wide range of opinion, standards do not require a specific set of viewpoints, they expect that they are selected appropriately depending on the system-of-interest.
- A **view** is a work product expressing the architecture of a system from the perspective of specific system concerns [2], illustrating how the architecture addresses them. An architecture description shall include exactly one architecture view for each architecture viewpoint used. According to [3], the core views of an architecture are the Functional, the Information, the Concurrency, The Development, the Deployment and the Operational views.
- A **perspective** is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views [3]. Although not formally defined by the aforementioned standards, this reference is included in [2] as a note. In some architectures, perspectives are referred to as system characteristics, however, in ASSIST-IoT the term **"Vertical"** will be used instead since apart from inherent properties, specific software will be included under its scope to meet or solve specific cross-cutting concerns.

2.2. Architecture paradigms

Almost every software application written today can be broken into three elements: a front-end or client-side, a backend, and some type of database. While requests are made to the developed application via the frontend interface, the backend code does all the heavy lifting, and any relevant data that needs to be stored or accessed is sent to or retrieved from the database (see Figure 2).

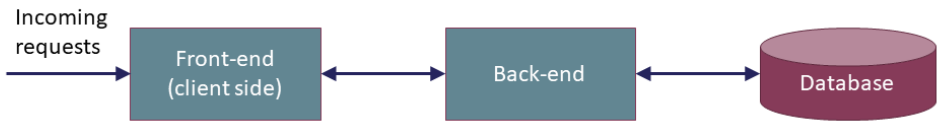


Figure 2. Three-tier architecture

In the early deployment stages, IoT applications were simple, and the number of IoT elements involved were small, so that IoT developers typically shared the burden of contributing to and maintaining the codebase. As the Next Generation Internet of Things (NGIoT) grows, new features shall be added to the applications, leading to (i) an increase in the operational workload, and (ii) a necessary horizontal and/or vertical scaling, requiring that more servers host the application. The complexity of the NGIoT applications is growing steadily, and hundreds of tests shall be carried out to guarantee that any minimum change made does not compromise the integrity of the existing code.

To cope with the increase of the number of elements present and workload required in IoT systems, software architectures are the starting point to design them and to solve a specific problem or adapt to a need. Some approaches have been proposed to address the complexity, at different levels, of software architectures. The asset of plans that form the software architecture guide the management of the information infrastructure to enable the desired state. Architectural patterns define how to organise system components when building a complete system and meeting the requirements set by an activity [5]. Architecture is not just about design, it also involves coding, abstraction, standards, formal training (of software architects), and style. The architecture deals with the interactions and constraints on those elements.

The variety on architectural styles and patterns available in the software industry makes it necessary to understand the particularities that suit in each situation. The **most relevant software architectures and patterns**[1] are listed below [6]:

- **Layered architecture pattern** closely matches the traditional IT communication, organised in horizontal layers. Each layer (commonly: User-Business-Infrastructure) has a specific role within the application and can communicate with each other through defined interfaces and different topologies (e.g.: OSI model).
- **Microkernel (plug-in architecture pattern)** is a natural pattern for implementing product-based applications, that are packaged and made available for download in versions as a typical third-party product. The core component contains the minimal functionality required to make the system functional, and plug-in modules contain specialised processing (e.g. Eclipse and extra compilers).
- **Event-driven** architecture is a distributed asynchronous pattern that manages data processing timeouts by building a central unit that accepts all data and then delegates it to the decoupled components that asynchronously receive and process events. Consists of two main topologies: the mediator (used when need to orchestrate multiple steps within an event through a central mediator) and the broker (used when you want to chain events together without the use of a central mediator - e.g., JavaScript web page).
- **Space-based architecture** is specifically designed to address and solve scalability and concurrency issues, being a better architectural approach than trying to scale a database or adapt caching technologies to a non-scalable architecture. It is suitable for applications that have variable and unpredictable concurrent user volumes. The space-based pattern works around the idea of distributed shared memory, and try to minimise the factors that limit application scaling. There are two main components within this architecture pattern: a processing unit and virtualised middleware. (e.g. cloud).
- **Serverless architecture** is a cloud computing approach to building and running apps and services without the need for infrastructure management [7]. In serverless applications, code execution is managed by a server, allowing developers to implement code setting without taking care of server maintenance, while a third-party cloud service takes full responsibility. This configuration eliminates the need for extra resources, application scaling, server maintenance, and database and storage systems.
- The architecture paradigm **based on services** is focused on performing functions that break complex problems into a series of simpler ones [8], executing tasks through a communication protocol over a network. Services are designed to be separately deployable, allowing to build highly scalable and resilient systems, in order to respond to hardware events, or listening for data requests from other software [9]. Service-based architectures vary in terms of services characteristics, service taxonomy and granularity [10]. There are two main approaches of the architecture paradigm based on services:
    - **Monolithic architecture** is a traditional software model that aims to build a single-tiered software application in which different components are combined into an indivisible program or platform formed by a code base with several modules [11].
    - **Service-oriented architectures (SOA)** can be seen as a natural evolution of monolithic pattern by decoupling an application into smaller modules. All the services would then work with an aggregation layer which can be termed as bus. This architecture can still be seen as a monolith from the deployment perspective, while microservices lead towards independent deploys.
    - **Microservices architecture** style structures a robust solution as a set of lightly coupled small services which are isolated in small coherent and autonomous units, to solve the problem of complex architecture and code redundancies

Architectures based on services:

Since ASSIST-IoT will follow one of the architecture paradigms based on services, here we delve into them to motivate the choice made in the next section. **Monolithic** architectures are the first approach of architectures based on services, grouping systems that are implemented as single unit of deployment. It has been the benchmark operating model for many years and countless applications have been successfully built. Its simplicity allows avoiding many of the problems associated with distributed systems, resulting in simpler developer workflows, monitoring and testing. The most common point of view about monolithic architectures

[1] https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice

refers to single process monolith, where all the code is packed into a single process. However, it may be the case in which a series of modules function as a single indivisible system, summarising monoliths as architectures where all functionalities in the system have to be deployed together [12].

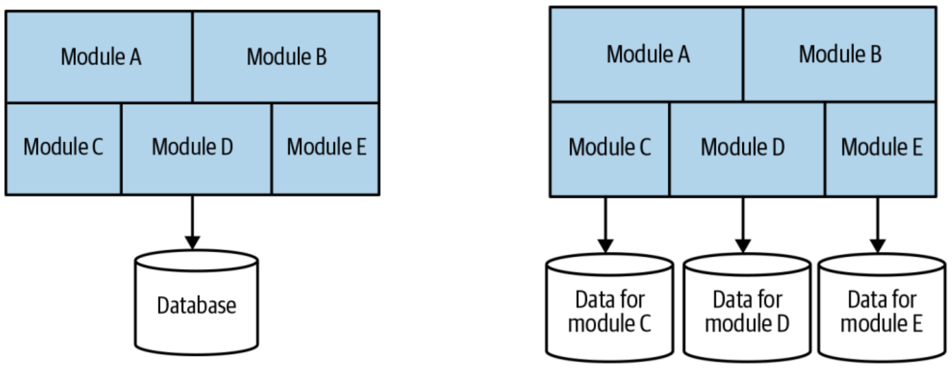


Figure 3. Modular monolith with single database (left) and decomposed database (right)

The opposite approach to monolithic architectures would be to divide this complex structure into smaller services. In **SOA**, application components provide services to other components through communication protocols, usually over a network [8]. The architecture includes a messaging middleware component that allows mediation and routing, message enhancement, message transformation and protocol transformation [10]. Service-oriented architecture principles are independent of any product, vendor or technology, enabling interoperability between systems, services and applications [13], [14]. The service bus acts as an orchestrator for complex event interactions, while the integration hub handles protocols and other transformations. The enterprise services must call code to implement their behaviour based on business processes [15].

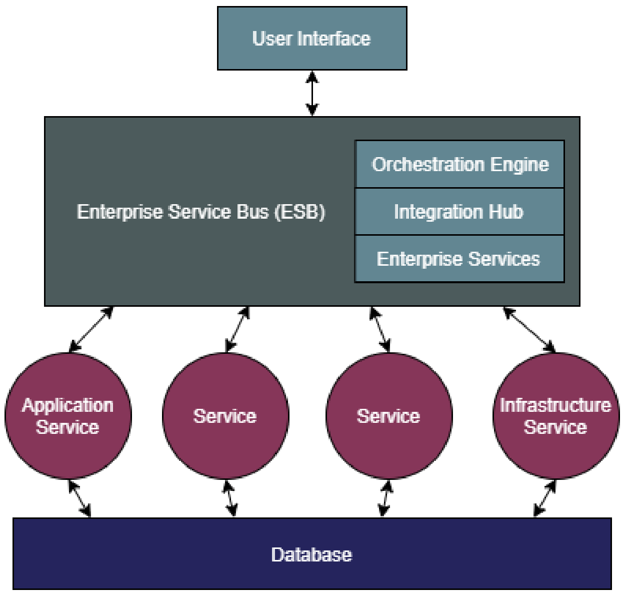


Figure 4. SOA architecture

To simplify services development, an architectural style that focuses on the design and development of software systems as a set of small independent services has emerged [10]. Microservices also bring some complexity in understanding the call chain that will happen for any given request, and the performance implications of all the additional network calls [16], [17]. A single microservice can be small and easy to understand, both in terms of business domain and performance [18], in contrast to a constellation of microservices.

The microservices architecture pattern does not support the messaging middleware concept, so it typically has an API layer between services and the consumers [10], using less elaborate and simple messaging systems and lightweight protocols as HTTP, REST or thrift APIs [19]. Microservices architecture makes scaling and adding new capabilities much easier, being suitable to develop a large application with multiple modules and user journeys [18]. Also, each microservice can be reused as a part of a different application. The granularity on the design results in a performance negative side effect of the distributed nature of microservices, as network calls and security verification at every endpoint adds additional processing time [20], [21].

The architecture of a microservice is not that different from the standard three-tier application architecture of Figure 2, as each microservice will be formed by three components: a frontend, some backend code, and a way to store or retrieve any relevant data in a database. However, there are some differences, as shown in Figure 5:

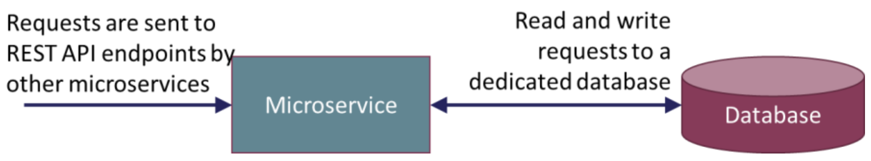

Requests are sent to REST API endpoints by other microservices
Microservice
Read and write requests to a dedicated database
Database

Figure 5. Elements of microservice architecture

1. The frontend of a microservice will be an API with static endpoints that will allow microservices to easily and effectively interact and send requests to the relevant API endpoint(s). For example, a common type of API endpoints for microservices is via HTTP over REST endpoints.
2. Although the API endpoints may be theoretically separated from the architectural point of view, in practice, they live alongside and as part of all the backend code that processes every request.
3. Most microservices will store some type of data, either in memory or in an external database.

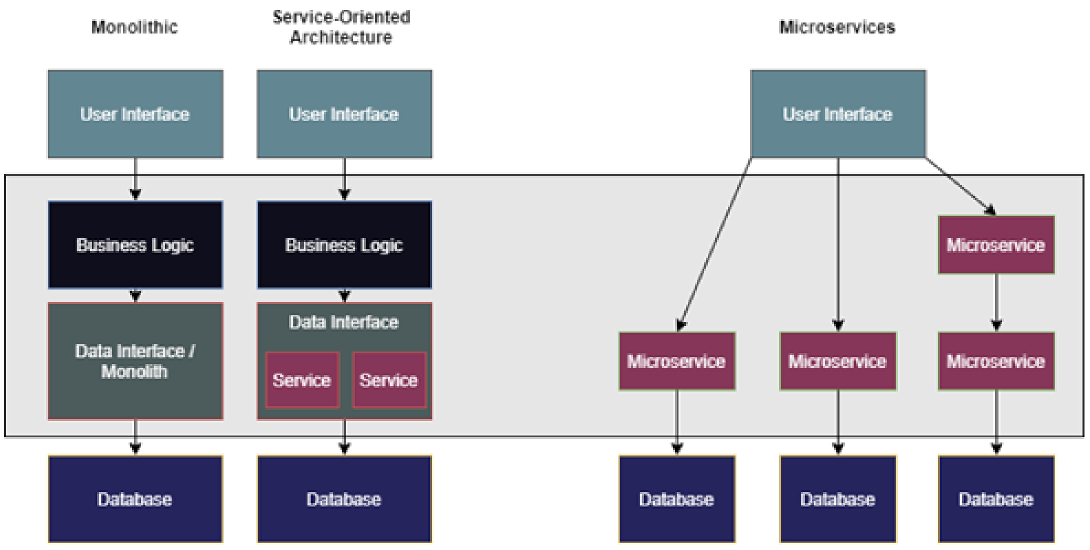


Figure 6. Monolithic vs SOA vs Microservices architecture

Usage comparison between architecture paradigms based on services.

Monolithic architectures are suitable for small team or simple applications that require a quick launch such as activities that do not demand much business logic, superior scalability, or flexibility have no need to deal with the complexity of the microservices architecture. If needed, a monolithic model is a choice that requires less initially spending resources, allowing to develop and launch it as soon as possible.

On the other hand, distributed architectures try to maximise application service reusability, by avoiding modifying the entire monolith when a systematic change is required. The more services are reused, the lower is the cost of implementing software development and management, so SOA is an ideal architecture method for large and complex business applications. It is the most suited architecture for environments that require integration with many diverse applications.

SOA and microservices are architectures built on different component-sharing concepts. SOAs try to avoid functionalities duplication by sharing as much as possible, generating tightly coupled components, which increases their difficulty to be changed. On the contrary, microservices only expose a defined interface to communicate its single closed units [10]. Minimal dependencies allow easier deployment and less risk while changing modules or services. Microservices architectures are focused on decoupling, so a systematic change implies creating a new service [18]. They require expertise, with the proper skills and knowledge, and it is focused on teams' collaboration and freedom of choice. The following table summarises the main concepts of the architectures described in this section.

Table 1. Comparative among architecture paradigms

	Monolithic	SOA	Microservices
Suitable	Small team or simple applications. Low initially spending resources.	Web-services. Widely used in distributed computing, SAAS based cloud computing models	Websites with small components. Corporate data centres with well-defined boundaries. Development teams that are spread out, often across the globe.
Not suitable	Applications that require scalability. Not suitable for applications that require complex development before executing. Not easy to adapt in applications that require changes.	Not ideal for small applications as they do not require middleware messaging components.	The services must be largely independent or else interaction can cause the cloud to become imbalanced. Websites with complex components that compromise performance. For instance, too many microservices can cause that parts of the web page appear much later than others.

As a lead paragraph towards the next section, and before digging deeper into ASSIST-IoT paradigm selection, it is worth reflecting on the information provided by the previous table.

Per definition, (Next-Generation) IoT deployments must be ready to adapt to changes, either in terms of adding new devices to the installation, applying new services over the data provided by those or changing the security policy for managing them, among others. Besides, scalability is also key in such deployments. With ever-increasing computing capacity in always-smaller miniaturised devices, the structure of an IoT deployment must from now on be ready to spot processing and capabilities throughout a wider range of elements/network spots.

Finally, an off-topic but relevant aspect is to keep in mind the COVID-19 pandemic has contributed to a major shift towards the teleworking paradigm [22]. This will, most likely, mean that for the future years (when NGIoT will finally take off globally) the development, integration and maintenance staff teams will be distributed throughout the globe. Therefore, a proper NGIoT blueprint architecture should allow an easy DevOps flow targeting such working teams.

3. ASSIST-IoT Approach

3.1. Design Principles

If the traditional monolith software architecture style was selected for ASSIST-IoT, the development and deployment of NGIoT applications would become consequently a burden and a blocker for even the most crucial fixes. The main reason most monoliths are susceptible to scalability problems is that their nature does not address partitioning (i.e., each task can be broken up into smaller pieces) and concurrency (i.e., is not only broken up into small pieces but can be processed in parallel). For instance, a software upgrade scalability requires scaling of the entire application rather than parts of it. Hence, in this section are detailed the design principles that will govern ASSIST-IoT architecture and solutions, namely (i) the use of microservices, (ii) their instantiation in containers, (iii) their grouping into "enablers", that will be introduced in this section and further described in Section 3.4, (iv) and their further orchestration using Kubernetes technology.

Microservices

To cope with these anticipated challenges, ASSIST-IoT architecture proposes to **follow a microservice software architecture**, which pursues building applications as suites of services. Following this approach, it will allow beyond the fact that microservices can be independently deployable and scalable, to also provide a firm module boundary, allowing for different services being written in different programming languages, and being managed by different project partners. The goal of the microservice architecture of ASSIST-IoT will be to build a set of small applications (called enablers) that are each responsible for executing one function (as opposed to the monolithic way of building one application that executes everything), and to let each microservice be autonomous, independent, and self-contained.

Containerisationg

As IoT deployments become more numerous, scalable architectural solutions are crucial for meeting and sustaining the demand of large, expanding, and elastic device networks. While former IoT applications simply control and analyse signal data from edge devices in a centralised manner, NGIoT applications are moving towards smart analytical applications, requiring horizontal scalability of device additions, including a reduction of instantiation times, or migrating the actual computing and intelligence towards the edge.

To promote rapid onboarding, ASSIST-IoT foresees a NGIoT architecture that ensures that the instantiation of the first service (called enabler) will be as simple as the instantiation and addition of the 1000th. To enable this seamless scalability in NGIoT infrastructure deployments, **ASSIST-IoT proposes to employ a *containerised* approach** that will allow developers to create each microservice over the most fitting OS and language.

Like virtual machines, containers allow to package for and decouple applications from the environment in which they will be running. This decoupling will allow container-based ASSIST-IoT microservices to be deployed easily and consistently, regardless of the target environment. However, **containers offer several benefits with respect to VMs**. Instead of virtualising the hardware stack (as VMs), containers virtualise at the OS level, i.e., multiple containers can be running over the OS kernel directly. Hence, containers are far more lightweight than VMs, as they can be run much faster, and use a fraction of the memory. Other benefits are listed in the Table 2 below:

Table 2. Containers benefits versus Virtual Machines benefits[2]

	Container benefits	VMs benefits
Consistent runtime environment	✓	✓
Application sandboxing	✓	✓
Small size on disk	✓	
Low overhead	✓	

[2] https://cloud.google.com/containers

As all components on the edge appliance will be containerised, it will allow to search and infer which equipment can handle which containers, and enables applications to be dynamically deployed and moved, and their resource utilisation to be monitored.

Enablers

Since ASSIST-IoT targets software products for covering many different functional domains, the project introduces the abstraction term "enablers", which will consist of a group of microservices, each of them served over a container, acting towards a single goal (i.e., to provide a specific functionality) in the architecture. Each enabler provides a single point of entry (interface) to communicate with it, without exposing the internal communication mechanisms between its components, thus having an "encapsulation" of microservices. As a result, the security of the software is improved by design, and in addition, potential software deployments and upgrades are facilitated by (i) grouping microservices according to the provided functionality, and by (ii) reducing the level of granularity offered by using just standalone microservices, without losing the advantages provided by microservices (e.g. distributed computation). In such ways, ASSIST-IoT ends up presenting an architecture that, although based on microservices, has a granularity that falls in the middle of a "pure" microservice architecture and a SOA. In some cases, this kind of architectures is also referred to as Service-based, although according to different sources this may not be fully correct if presented as an opposite to microservices or SOA, as described in Section 2.2. Additional information regarding enablers is presented in Section 3.4.

Kubernetes

Kubernetes[3], Docker Swarm[4] and Apache Mesos[5] are examples of technologies for service virtualisation and container orchestration. They facilitate the execution of different tasks related to deploying and managing applications, and are specially needed for managing systems with a large number of containers instantiated in different servers (e.g., a cluster environment). These solutions are in charge of (i) managing the creation of containers, (ii) verifying their operation, and (iii) offering correct management of errors. In ASSIST-IoT, **Kubernetes has been selected as the main technology for containers orchestration, and, therefore, for enablers orchestration.** The main motivation for choosing this option lies in its large adoption in current trends in cloud native systems in contrast to other alternatives [23]. In addition, there are modified distributions of Kubernetes that target constrain devices, which are part of the Device and Edge plane of ASSIST-IoT as will be explained later in this document.

Kubernetes, also referred to as k8s, automates rollouts and rollbacks, monitors the health of software microservices to prevent bad rollouts before things go bad. It also enables to continuously run health checks against deployed services, restarting containers that fail. Additionally, Kubernetes will automatically scale services up or down based on utilisation, ensuring they are only running what the owner needs.

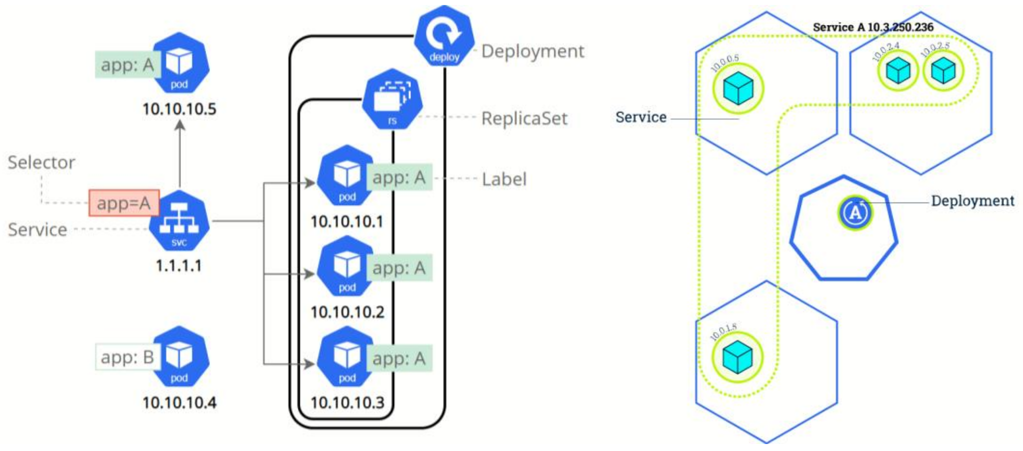


Figure 7. Services exposed (left) and scalability approach (right) for container orchestration with Kubernetes[6]

[3] https://kubernetes.io/
[4] https://docs.docker.com/engine/swarm/
[5] http://mesos.apache.org/
[6] https://kubernetes.io/docs/tutorials/kubernetes-basics/scale/scale-intro/

Still, k8s initial suites were not suitable for small form-factor devices, such as the ones initially considered in a current and NGIoT deployment. In particular, NGIoT deployments at the edge typically are expected to work with smaller footprint of devices that do not have enough capacity to run Kubernetes effectively, such as sensors. Additionally, connectivity issues related with e.g., latency, bandwidth limitations or disconnection shall be solved in an automated way. Because of these and other requirements related to public or private access to the Internet, or cybersecurity, **containers orchestration at the edge in general, and Kubernetes modifications** are started to get the attention of NGIoT alliances and open-source communities and will become the basis of scalability solutions of ASSIST-IoT. Recent projects include K3s[7], microk8s[8], or FLEDGE [24], among others.

3.2. ASSIST-IoT Conceptual Architecture

The ASSIST-IoT conceptual architecture is a reference to a high level of abstraction, on which more concrete designs may be implemented, and specific applications realised. The architecture is rooted in a multidimensional, conceptual approach, in which architectural layers (called "Planes", see Appendix A -) are intersected by vertical blocks ("Verticals") presented in Figure 8. This approach captures horizontal and vertical functions of an IoT system, as well as the intersections between them and allows for higher level of modularity and adaptability in compliant systems.

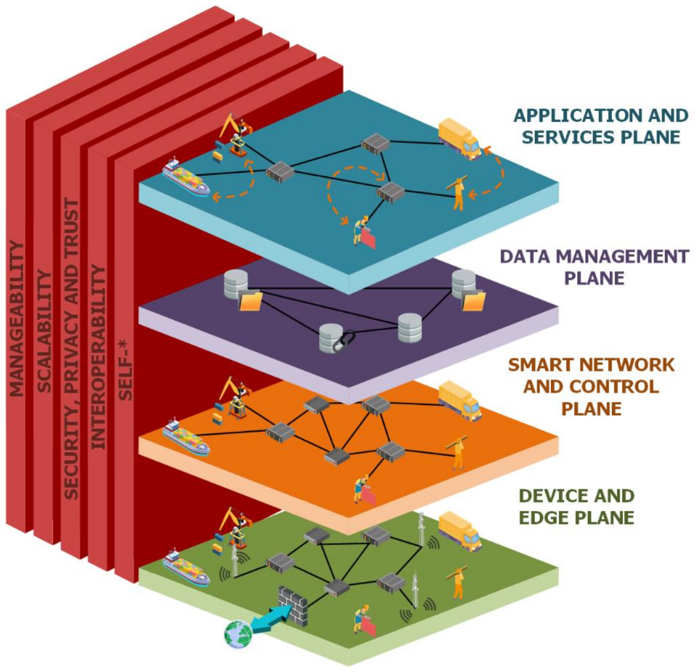

APPLICATION AND SERVICES PLANE
DATA MANAGEMENT PLANE
SMART NETWORK AND CONTROL PLANE
DEVICE AND EDGE PLANE
MANAGEABILITY
SCALABILITY
SECURITY, PRIVACY AND TRUST
INTEROPERABILITY
SELF-*

Figure 8. ASSIST-IoT Conceptual Architecture

The horizontal Planes represent collections of functions that can be logically layered on top of one another. For example, observation data originating from a sensor must pass through the Smart Network and Control plane and be processed on the Data Management plane, before being presented to an end-user in a GUI application on the Applications and Services plane. Not all information must always pass through all planes – in fact, one common function of an edge device is to filter out only necessary data, or to aggregate data, so that only relevant information is passed on. In principle, however, software (and hardware) on any level can rely on, and abstract away, any functions performed on planes below. It is worth mentioning that ASSIST-IoT concept of Planes must not be confused with the traditional (OSI-like) approach of protocol stack, but rather as a smart classification of logical functions that fall under the scope of diverse plane domains.

Verticals, on the other hand, represent functions targeting NGIoT properties that exist either independently on different planes or require the cooperation of elements from multiple planes. For example, although one may implement a fully secure networking solution, the property of being secure could also be extended to any system that uses such networking, provided that security measures are also implemented on other planes. Cooperation

[7] https://k3s.io/
[8] https://microk8s.io/

of plane functions may be required to secure e.g., data access with biometrics measured by IoT devices – if access to data (from the Data Management plane) depends on signals from a device (Device and Edge plane) analysed on the Applications and Services plane, then system security must consider all those planes.

Since it is not possible to capture all the functional features in a single model, the ASSIST-IoT architecture will **result in different Views and** a set of **Verticals** (addressing properties and cross-cutting concerns). In Sections 4 and 5, these **Verticals** and **Views (Functional, Node and Deployment)** are introduced, considering that their number and extent may change along with further iterations of the architecture.

3.2.1. Functional architecture

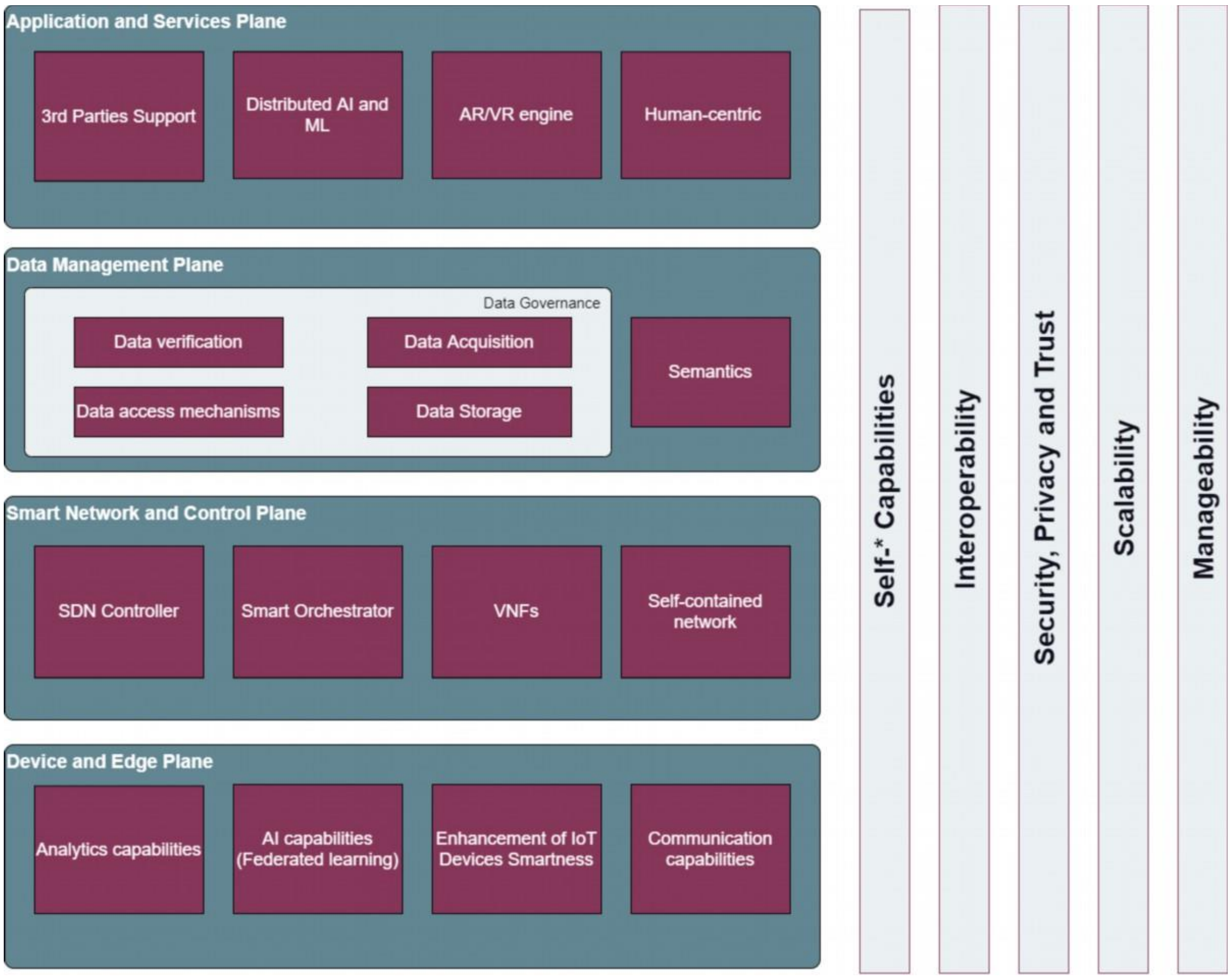


Figure 9. ASSIST-IoT Functional Architecture

A high-level Functional View of the ASSIST-IoT is presented on Figure 9. This View will be extended in Section 5. While the Vertical functions are described in detail in Section 4, the Planes of the ASSIST-IoT architecture are as follows:

Device and Edge plane describes the collection of functions that can be logically appointed to physical components of IoT, including, but not limited to, smart devices, sensors and actuators, wearables, edge nodes, as well as network hardware, such as hubs, switches and routers. Note that this plane, like all the others, represents a Functional View. So even though e.g., functions related to self-contained network could be naturally associated with network devices, there is a group of functions that can be identified and separated into functional blocks that belong squarely on the Device and Edge plane. The aforementioned functions include any physical connectivity and interfaces (e.g., Ethernet), low-level security functions (e.g., firewalling). This plane directly interfaces the hardware capable of executing specific functions designed on higher planes.

Smart Network and Control plane manages virtual and wireless aspects of network connectivity. The key functions handled on this plane are encompassed by technologies that deliver software-related and virtualised networks, such as SDN (SD-WAN), NFV, MANO, and anything related to virtualised or self-contained networking. Any direct and logical connection in the communication infrastructure is provided on this plane. The functions on this plane follow the access-network-agnostic approach, in which the network connections are highly flexible. Features, such as dynamic configuration, routing and addressing, and high-level intelligent firewalling help deliver the required flexibility.

Data Management plane handles all functions related to a virtual shared data ecosystem, in which data are acquired, delivered and processed to provide key data-related functions. Those include data interoperability, provenance, fusion and aggregation, but also content-independent functions, such as resilience (e.g., redundancy). Security functions for access grants and trust management also belong to this plane. Moreover, this plane is empowered by semantics and might be supported by judiciously selected DLT.

Application and Services plane crowns the Functional View with end-user and administrative functions and services. It delivers a layer of abstraction that manages functions offered by lower planes. Moreover, it combines them to provide synergistic value for the whole system. Its functions, aided by the Verticals, aim to offer a unified point of access, and provide system-wide intelligence and configuration capabilities. Because of the high level of abstraction, this plane enables the creation of advanced and intelligent applications, including configurable autonomous systems, that benefit from the lower planes, and their interconnection.

3.3. Relation with other IoT-related RAs

ASSIST-IoT Conceptual Architecture has not been created from scratch, but rather designed considering multiple inputs including (i) the current trends towards integration between IoT-related technologies with NGI technologies (such as Edge Computing, Artificial Intelligence and SDN/NFV Paradigms), (ii) the expertise of the Consortium partners in different technological areas, (iii) the outcomes of previous and concurrent projects, as well as of Standards Developing Organisations (SDOs), and an (iv) extensive research of innovative concepts to improve current performance and scalability of IoT architectures or integrate novel functionalities.

As presented in D3.1, the number of Reference Architectures (RAs) related to IoT is quite large, and hence starting a completely new architecture without evaluating or considering the results and insights provided by (at least some of) them would not be the right decision. Considering those facts, ASSIST-IoT is influenced mostly by two RAs, namely the ones provided by the IoT European Large-Scale Pilots (LSP) programme [25] and the OpenFog consortium [26].

LSP architecture

The 3D architecture presented within the framework of the LSP programme is composed of 8 layers, with 8 cross-cutting functions and 8 properties. The "Layer" dimension supports the Functional View of the system in a technology-agnostic way, while the "Cross-Cutting Functions" dimension considers transversal technologies to different layers, whereas "Properties" addresses the global properties of the IoT system that are (or not) provided by a proper implementation of combination between layer-level and cross-cutting functions. In ASSIST-IoT, we have considered a similar layered concept, which differs in grouping them into four main horizontal Planes to facilitate further developments and implementations. ASSIST-IoT Conceptual Architecture maps the LSP RA as shown in Figure 10. Regarding the provided **cross-cutting functions and system properties**, in ASSIST-IoT they are considered in a **single dimension (Verticals)** since both concepts are transversal to the horizontal Planes of ASSIST-IoT, and as with most available RAs, no strong reason to add a third dimension was found.

Mapping LSP layers and ASSIST-IoT planes is quite straightforward. "Edge devices" is included in ASSIST-IoT's Device and Edge Plane, since it is the plane in direct contact with elements of the edge (including in the latter Smart IoT Devices, which in the case of LSP are outside the scope of the architecture). The following two LSP layers can be mapped to both lower layers of ASSIST-IoT: (i) "Connectivity" responds to the physical gateways and forwarding devices (e.g., switches), which in ASSIST-IoT are located in the lower plane, and routing decisions as well as other virtualised networking functionalities which are managed by the Smart Network and Control plane; (ii) "Edge Computing" can be applied in elements of the Device and Edge plane, however, virtualised computing capabilities are orchestrated by the second plane. Lastly, the two layers devoted to data within LSP match the Data Management plane of ASSIST-IoT, whereas the three upper layers devoted to Services, Applications and Business Processes can be included in the upper plane of ASSIST-IoT.

ASSIST-IoT Verticals can gather the functionalities stated in the other two dimensions of the LSP architecture. The vertical related to Security, Privacy and Trust can be mapped to the cross-cutting functions related to "Identifiability", "Trustworthiness", "Security", "Safety" and "Privacy" as well as with the property of "Dependability", which is related to trust and resilience of system and applications. The resilient aspect of "Dependability" could be included also with Self-* vertical, since this one includes functionalities related to guarantee "Reliability", "Resilience", "Availability" and, most importantly, "Intelligence" to ensure all the former properties and cross-cutting functions in an autonomous way. Interoperability vertical, apart from its homonymous property in LSP RA, can be related to the "Integrability" property and "Connectivity" cross-cutting functions (for instance, facilitating the connectivity of external devices, sensors or network elements, either physical or virtual). Manageability vertical can be mapped to its homonymous property in LSP architecture and also to Composability one, whereas Scalability is mapped one to one. In summary, despite being split within sixteen properties and cross-cutting functions, the transversal functionalities included by them can be gathered within the five ASSIST-IoT Verticals.

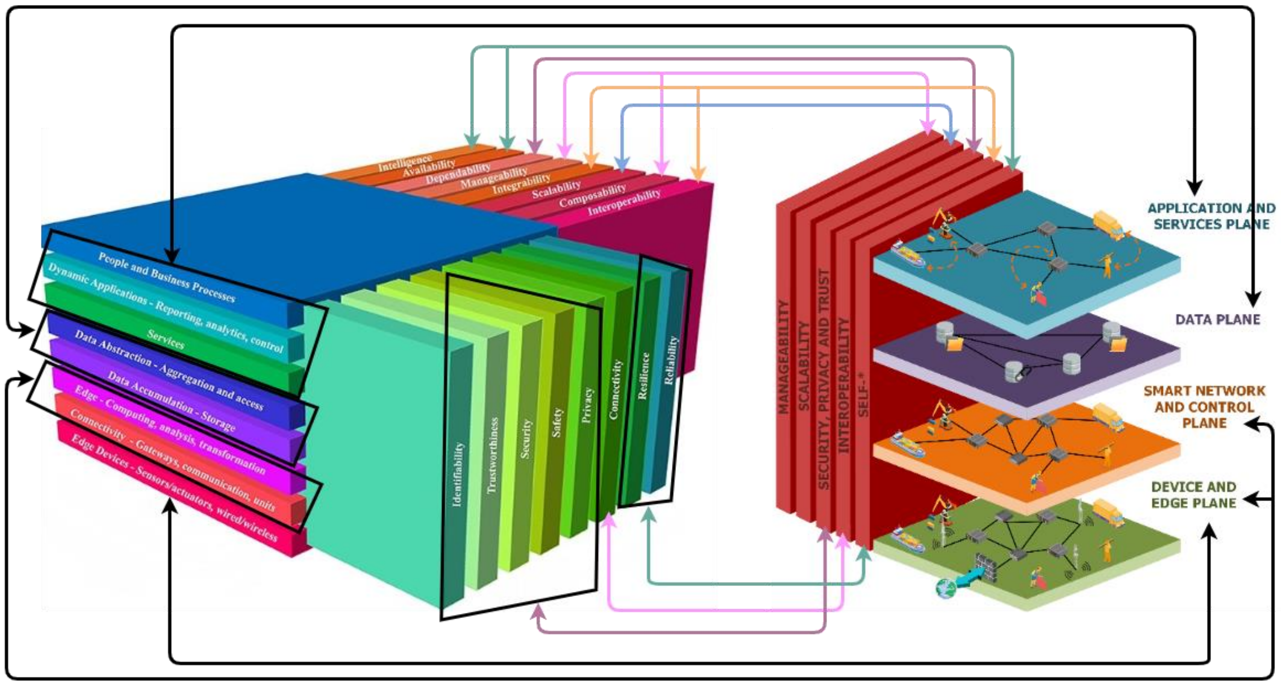

Intelligence
Availability
Dependability
Manageability
Integrability
Scalability
Composability
Interoperability
People and Business Processes
Dynamic Applications - Reporting, analytics, control
Services
Data Abstraction - Aggregation and access
Data Accumulation - Storage
Edge - Computing, analysis, transformation
Connectivity - Gateways, communication, units
Edge Devices - Sensors/actuators, wired/wireless
Identifiability
Trustworthiness
Security
Safety
Privacy
Connectivity
Resilience
Reliability
MANAGEABILITY
SCALABILITY
SECURITY, PRIVACY AND TRUST
INTEROPERABILITY
SELF-*
APPLICATION AND SERVICES PLANE
DATA PLANE
SMART NETWORK AND CONTROL PLANE
DEVICE AND EDGE PLANE

Figure 10. Comparative between ASSIST-IoT Conceptual Architecture and LSP 3D Architecture

OpenFog architecture

OpenFog RA is defined by its authors as a medium- to high-level view of system architectures for fog nodes and networks. Its conceptual architecture is presented in a two-dimensional model, consisting of "Layers" and "Perspectives". Despite not plotting a third dimension, OpenFog RA is driven by a set of principles named "Pillars": Security, Scalability, Open, Autonomy, RAS (from Reliability, Availability and Serviceability), Agility, Hierarchy and Programmability. Hence, although most of these pillars are not directly addressed within a specific perspective or functionality, they represent a set of properties that should be inherently present in an OpenFog RA instantiation. The mapping between ASSIST-IoT and OpenFog's Architecture Description is illustrated in Figure 11.

Comparing OpenFog's perspectives with ASSIST-IoT Verticals is quite straightforward. "Manageability", "Security" and "Scale" are quite self-explanatory, so despite some of the included functionalities within OpenFog's perspectives vary in comparison to their homonymous ASSIST-IoT's Verticals, the core ones are shared. The latter is included within the perspective named "Performance & Scale", dealing mostly with scalability and isolation aspects. "Data Analytics and Control" has been mapped with Self-* vertical since both are responsible for extracting knowledge for performing actions, mostly at node level, although in OpenFog it goes beyond the extent of a single node to also send results to higher ones for further business or operation-specific analysis. "IT Business & Cross-fog Applications" is understood in OpenFog as the flexibility of distributing data or knowledge among different nodes, mostly in multi-vendor ecosystems. Although mapped to the Interoperability vertical, the latter expands to other domains. In any case, ASSIST-IoT project has provisioned a task for cross-context federated enablers, so the concerns of that particular OpenFog's perspective will be also covered.

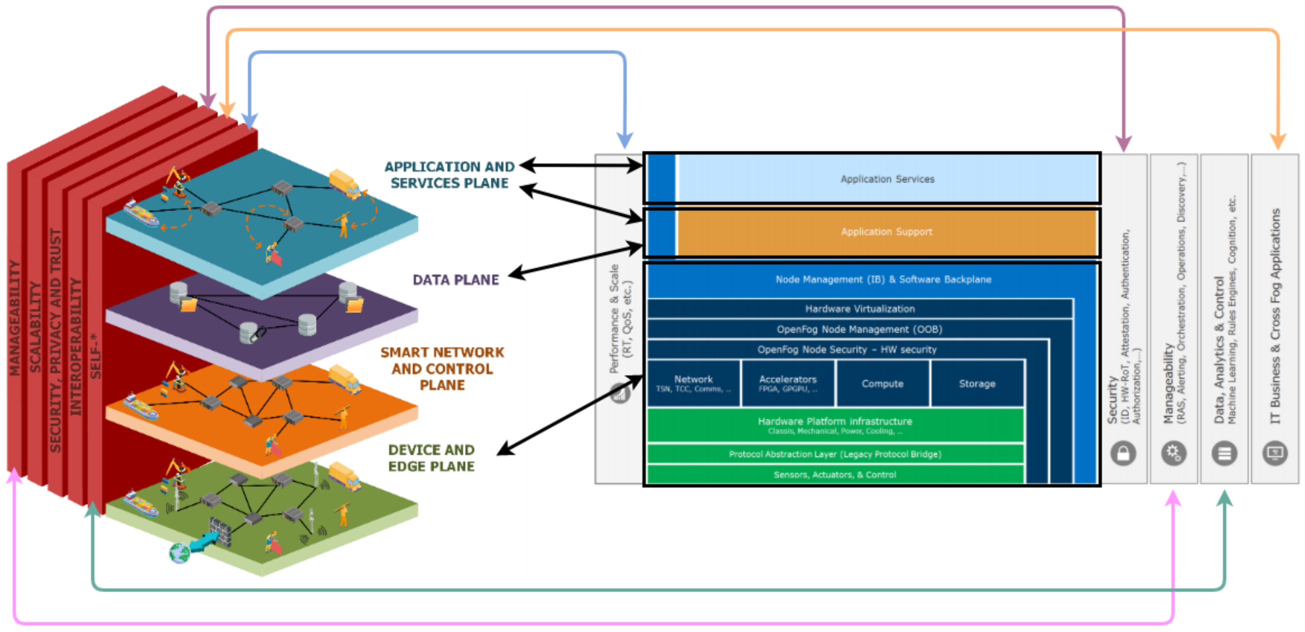

MANAGEABILITY
SCALABILITY
SECURITY, PRIVACY AND TRUST
INTEROPERABILITY
SELF-*
APPLICATION AND SERVICES PLANE
DATA PLANE
SMART NETWORK AND CONTROL PLANE
DEVICE AND EDGE PLANE
Performance & Scale (RT, QoS, etc.)
Application Services
Application Support
Node Management (IB) & Software Backplane
Hardware Virtualization
OpenFog Node Management (OOB)
OpenFog Node Security – HW security
Network TSN, TCC, Comms, ...
Accelerators FPGA, GPGPU, ...
Compute
Storage
Hardware Platform infrastructure Classis, Mechanical, Power, Cooling, ...
Protocol Abstraction Layer (Legacy Protocol Bridge)
Sensors, Actuators, & Control
Security (ID, HW-RoT, Attestation, Authentication, Authorization,...)
Manageability (RAS, Alerting, Orchestration, Operations, Discovery,...)
Data, Analytics & Control Machine Learning, Rules Engines, Cognition, etc.
IT Business & Cross Fog Applications

Figure 11. Comparative between ASSIST-IoT Conceptual Architecture and OpenFog Architecture Description

Regarding OpenFog layers, the lower ones (from "Sensors, Actuators & Control" to "Hardware security") have been related to ASSIST-IoT's Device and Edge plane. This mapping is direct as they refer to this group of layers as "Node View". In any case, ASSIST-IoT's lower plane covers not only fog (in our case, edge) nodes, but also innovations framed close to the physical network elements (routers, switches, etc.) as well as potential Smart IoT Devices (i.e., devices with certain processing capabilities). The following layers, related to virtualisation and software to facilitate node-to-node communications (e.g., "Software Backplane" includes Operative Systems, software virtualisation, containerisation) have been also mapped to the lower layers, since apart from hardware elements, ASISST-IoT edge nodes will need software container orchestration systems to be fully functional. "Application support" has been split between the two upper layers of ASSIST-IoT because data-related services belong to the third plane, although non-related ones would lie within the upper one (related also to OpenFog's "Application services" layer).

The larger divergence comes with ASSIST-IoT's Smart Network and Control plane. On the one hand, ASSIST-IoT has a plane devoted to network services and management influenced by the SDN/NFV paradigm, hence representing a system rather than a node. On the other hand, OpenFog's description is a blueprint that could be instantiated in any fog node (with its hardware, hardware and software virtualisation, hidden and exposed applications and services), and no further specification is given regarding how to address the orchestration of Network Services nor the system's network.

Other Architectures

Although ASSIST-IoT Conceptual Architecture is inspired mostly by the two former RAs, some aspects have been modified based on other existing architectures. For instance, most RAs do not specify the number of horizontal Planes or layers, since it may hinder further implementations rather than facilitating it. For this reason, ASSIST-IoT groups their related functionalities in 4 main domains, being more inspired from pre-normative activities and standards issued by entities like ITU-T Rec. Y.2060 [27] (which defines Device, Network, Service and Application Support, and Application layer) and AIOTI HLA [28] (Network, IoT and Application, omitting the Device layer as also done in different software architectures), as well as edge-centric RAs like ECC RA 2.0 [29] (Edge Computing Node, Connectivity and Computing Fabric, Service Fabric and Smart Service). In those IoT/edge-centric RAs, the scopes of the layers vary to a greater or lesser extent when compared to ASSIST-IoT, and especially in the layer between network/connectivity and application. Although the number and functionalities included in the Verticals (and mentioned properties) change, Management and Security are common to all of them. Also, despite not being depicted in the conceptual representation of the architecture, ASSIST-IoT also considers possible hierarchy levels in the Edge Continuum, as Edge Computing RAs such as RAMEC [30] or OpenFog.

3.4. Enablers

Section 3.1 presented the concept of "enablers", which are the cornerstone of ASSIST-IoT architecture. These pieces aim at encapsulating (containerised) microservices to achieve a functionality of the system. The introduction of the "enablers abstraction" responds to the "realisation of a modular architecture to deliver the functions promised by ASSIST-IoT innovations and future capabilities". In essence, an enabler is a collection of software (and possibly hardware) components - running on nodes - that work together to deliver a specific functionality of a system. Although an enabler can be abstracted away as a distinct module (and must be logically separable from the rest of the system in which it is deployed), enablers may depend on one another to deliver what they promise (see Section 7).

In ASSIST-IoT the functionalities of enablers are divided into Verticals and Planes (see Figure 8), which also presents a natural way to categorise enablers. For example, one may talk about a "semantic interoperability enabler", if the enabler delivers functionalities that fall under the "semantics" functional block, and Interoperability vertical.

Although ASSIST-IoT delivers concrete enablers that will be deployed in specific pilot implementations, its architecture is defined on multiple levels, each more formal and lower in the abstraction level. Following this approach, the complexity of an enabler, as an architectural concept, is not mandated, and depends on the function that it delivers. For example, a geofencing AR system that analyses and displays area information on AR goggles, may be designed as a single enabler. It may also be built out of several enablers, e.g., one that gathers geographical data and delivers it to some persistent storage, and a separate one, that displays any geocoded information on AR goggles. The AR solution, whether implemented as one, or many enablers, may be used by yet another enabler to deliver information about restricted areas, or persons without proper authorisation.

Enablers are not atomic but presented as a set of interconnected components. An enabler component is a software or hardware artifact that can be viewed as an internal part of an enabler, and that performs some action necessary to deliver the functionality of an enabler as a whole.

An enabler component may be logically assigned to a functional block from a specific plane. For example, any data persistence solution (e.g., a database instance) "lives" on the Data Management plane. This property is inherited by any enabler that uses such component as their own. In effect, such enabler can also be logically placed on the Data Management plane. Because enablers may deliver complex functionalities, they may include components from multiple planes, i.e., enablers can be transversal.

In short, a critical part in describing any enabler is the *functionality*, that it *enables*. In order to deliver the functionality, an enabler employs components. Multiple enablers may be used in a system to deliver a complex application or service, exhibiting features of all components included in those enablers.

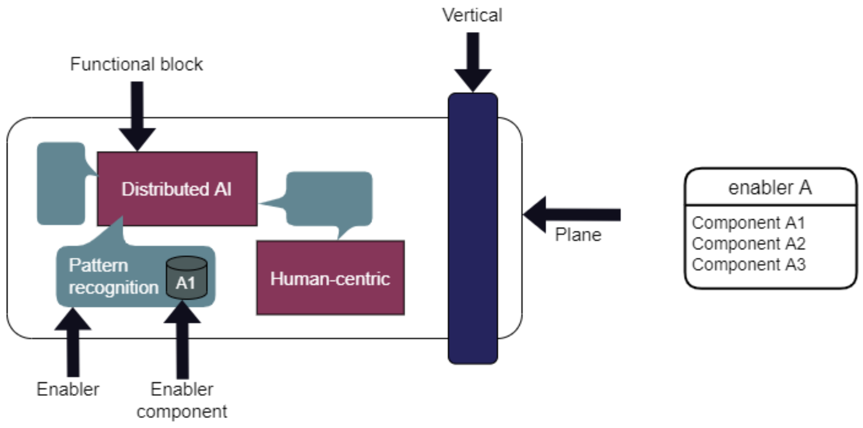


Figure 12. ASSIST-IoT enabler diagram

High variance in scope and weight of the functions that enablers may deliver, implies, that some enablers are optional, and should be deployed when needed, and others are critical delivering the full scope of ASSIST-IoT. Following this idea, enablers in ASSIST-IoT may be "obligatory" i.e., required for any ASSIST-IoT to be fully compliant with the architecture, while others are "complimentary" and deliver optional functions. Enablers, such as the SDN orchestrator form the "skeleton" of the architecture, and must be present in any ASSIST-IoT deployment. On the other hand, the geofencing enabler described in previous paragraphs is optional and can be added later, if a use-case requires it.

Another important property of an enabler is its deployability. As seen in Figure 12, an enabler may be described as a set of components, and categorised with Verticals, Planes, and functional blocks. In a real, physical system, however, the components of an enabler may be physically separated i.e., deployed on different premises (hardware components), or different machines/virtual containers (in the case of software components).

This means, that an enabler must define and control communication between its components. ASSIST-IoT does not mandate any particular interface of communication between enabler components in order to allow for flexible implementations, depending on the needs for performance and throughput of the communication. The communication between enabler components, and enablers, however, should be separated. In fact, one of the most important design principles that distinguish components from enablers, is that enablers should not directly communicate with components of other enablers, unless explicitly allowed by the enabler, to which they belong. Concrete deployments of components should also not be shared between enablers. That means, that even if two enablers use the same database software, they should not use the same database instances as their components. Under the ASSIST-IoT approach, if a shared database is needed, it should be offered as an enabler. Any communication between enablers should take place using the dedicated interfaces that an enabler exposes, which are separate from internal inter-component communication. This separation delivers the property of *encapsulation*. By architectural definition, ASSIST-IoT enablers are encapsulated.

Appendix B - provides the template for describing an enabler. This descriptor has been designed to represent functionally (and ultimately technologically) the different elements and properties of an enabler.

3.5. Methodology

The ASSIST-IoT architecture design is based on several inputs. On the one hand, the perspectives and objectives of the NGIoT must be met. On the other hand, the requirements of early-adopters/stakeholders must be considered. At the same time, the different points of view and particularities of the core technological blocks call for further analysis and consideration towards shaping the final specification of the ASSIST-IoT blueprint architecture.

First, the purposes of the reference architecture of ASSIST-IoT are mainly guided by the **official proposal** as set out in the Grant Agreement (GA). However, the definitions outlined in the GA are not precise enough to completely drive the definition of a NGIoT architecture**. Further research** has been conducted in the context of Task 3.5 in order to elaborate the ASSIST-IoT architectural framework and guiding principles that are described in this document.

Second, as the ASSIST-IoT architecture will be a human-centric and user-friendly architecture that must be effective and straightforward at the same time, its definition must rely on enabling the successful deployment of real-world pilots. This can only be achieved by mapping the stakeholders' **concerns** (i.e., essentially their requirements at this point) to the abstract concepts of the reference architecture specification. As Tasks 3.2, 3.3 and 3.5 are running concurrently and both Deliverables 3.2 and 3.5 are due on Month 6 of the project, an iterative approach has been followed in order to translate the evolving user stories, business scenarios and stakeholders' requirements into architectural elements. In order to make all the information elicited from the stakeholders available and transparent to the entire ASSIST-IoT consortium during the evolution of use cases and gathering of requirements, Miro[9] was used as an online whiteboard for visual collaboration. At this moment, **stakeholders** include **final users** from ASSIST-IoT pilots, **software architects and developers,** as well as production engineers and assessors (overseeing system's conformance to standards and regulations).

[9] https://miro.com/app/

Third, the **technological blocks** of ASSIST-IoT (corresponding to Planes or Verticals) will have a mutually dependant relationship with the architecture definition, as one will serve to shape the other. The different enablers (encapsulating the capacity of the architecture to provide certain functionalities) will be both guided by the architecture principles and by the objectives and particularities of each technology. In ASSIST-IoT, the work to be conducted in WP4 and WP5 will need to be aligned with the architecture, so certain interaction rules must be set. In order to facilitate and formalise the basic communication among them, the architecture principles (set out in this document) establish a common template to be followed by WP4 and WP5 activities (*enabler template*) that will serve to improve the architecture if needed.

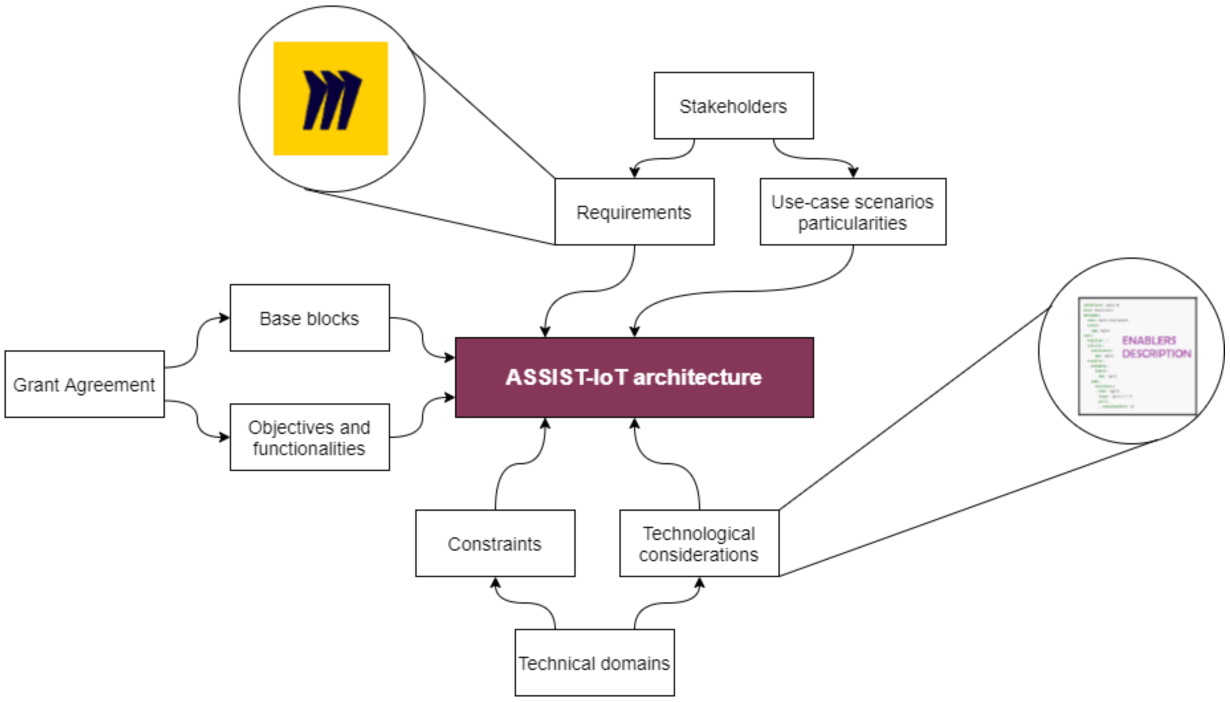


Figure 13. Inputs needed for formally describing the ASSIST-IoT architecture

Formally, the definition of ASSIST-IoT architecture must be conducted within the period between Month 3 and Month 21 of the project (corresponding to January 2021-July 2022). During that 18-month span, the mentioned inputs will be feeding the architecture introducing new functionalities that will need to be timely incorporated.

Therefore, to align the evolution of the architecture definition with the pace of inputs provision, a careful analysis has been done. The approach will be to “freeze” the first version of the architecture description with this deliverable (D3.5). The next version will be ready in Month 15; in the meantime, core and transversal enablers will have been initially defined and the requirements will be further evolved including elaborate considerations on legal aspects and regulatory constraints. By Month 19, all the requirements will have been taken into account and the architecture will be ready to be introduced to Open Call participants (see Figure 14).

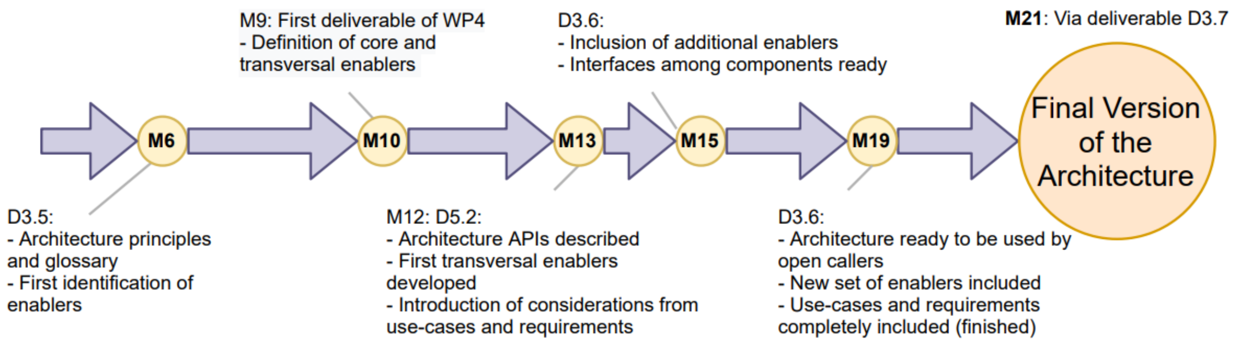

M9: First deliverable of WP4
- Definition of core and transversal enablers
D3.6:
- Inclusion of additional enablers
- Interfaces among components ready
M21: Via deliverable D3.7
M6
M10
M13
M15
M19
Final Version of the Architecture
D3.5:
- Architecture principles and glossary
- First identification of enablers
M12: D5.2:
- Architecture APIs described
- First transversal enablers developed
- Introduction of considerations from use-cases and requirements
D3.6:
- Architecture ready to be used by open callers
- New set of enablers included
- Use-cases and requirements completely included (finished)

Figure 14. Timing and methodology for evolution of the architecture definition

4. Vertical Capabilities

The Next Generation Internet of Things is a term that has been used to refer to the new wave of functionalities, properties, traits and features that forthcoming IoT deployments will need to support. The volume of initiatives targeting IoT is ever-growing, with many technologies, ideas and fields playing their role. While the challenge of overcoming the current barriers is enormous, ASSIST-IoT is being designed to support the upcoming needs.

Apart from the capabilities associated with specific "classic domains", the NGIoT will need to be characterised by a series of wide features targeting cross-domain aspects. While the former (horizontal) capabilities aim at covering the basic, functional aspects of an IoT deployment (network, communication, devices interaction, application, human interfaces, data processing, data storage, services, etc.), the latter (called "Vertical" in the ASSIST-IoT world) address **all-encompassing concerns, properties and transversal functionalities** such as Security, Interoperability or Manageability among others. The most prominent characteristic of these Verticals is that all of them may apply (individually or jointly) to different horizontal domains (or "Planes"). For instance, interoperability can be understood as an "edge/device layer" asset when allowing diverse sensors and gateways to co-exist in the same deployment, whereas interoperability at the application layer may mean allowing the user to execute services using the same UI and framework although leveraging various underlying platforms (e.g., Watson IoT[10], OneM2M[11], UniversAAL[12]).

Since the proposal stage, five Verticals have been identified to be driving ASSIST-IoT in this matter: (i) Self-* capabilities, (ii) Interoperability, (iii) Security, Privacy and Trust, (iv) Scalability and (v) Manageability.

As mentioned, ASSIST-IoT architecture must support the introduction of functionalities framed within these 5 vertical categories. Following the design principles of ASSIST-IoT architecture, the conceptual abstraction that must be internalised is the following: *ASSIST-IoT architecture will allow (and put the means)* ***to define enablers and describe properties*** *targeting vertical capabilities in an IoT deployment. Additionally, a specific set of enablers are defined as inherent, meaning that those will be present in any ASSIST-IoT deployment in order to provide a determined suite of ASISST-IoT essential innovations.*

Therefore, the objective in this section is three-fold:

- To define what each of the verticals means. This entails describing the overall concept, showcasing relevant examples and outlining the potential for NGIoT deployments that those capabilities cater.
- To describe the inherent enablers or the cross-cutting properties that will be needed per each vertical. This explanation aims at crystallising the abstraction of each vertical into particular implications when devising an ASSIST-IoT deployment.
- To propose an initial list of potential enablers to be created. This part will consider the different expectations during ASSIST-IoT in terms of pilots, tasks' scope, particular needs, etc.

4.1. Self-*

Self-* System is a system that is autonomous or semi-autonomous alongside some dimension. Autonomous in this context means that there is no need for constant overview from human operators. In order to realise Self-* capabilities, the functionalities with focus on Self-*- system defined in terms of ASSIST-IoT architecture must be realised. To do that, the State-of-the-Art analysis has been carried out, altogether with ASSIST-IoT's Pilots (stakeholder concerns – see Section 2.1) and professional experiences.

In the Figure 10 it can be seen that ASSIST-IoT's Self-* correlates to "cross-product" of cross-cutting functions related to Resilience and Reliability with Intelligence and Availability which might help narrowing down the *initial* scope of Self-* enablers to those closely related to broadly understood DevOps and software reliability. After some thought, this conclusion is not that surprising – as one could argue that the basic feature of Self-* system is that of ensuring its reliable operation now and in future.

[10] https://www.ibm.com/es-es/cloud/watson-iot-platform
[11] https://onem2m.org/developers-corner/tools/open-source-projects
[12] https://www.universaal.info/

One of the most basic and important features of any network is reliability. First step towards that direction is for the system to be able to *diagnose* a fault. This is where **self-diagnosis** is introduced**.** Self-diagnosis is a capability of a system to detect erring behaviour within itself. Counterpart to fault detection is *fault localisation* as usually it is not enough to say that a system has faulty components - it might be necessary to pinpoint which component (or a set of those) is responsible.

Building on top of self-diagnosis, the next logical step is to enrich the system's ability to fix faulty elements (either a group of faulty elements, fix them partially to restore expected behaviour or as a last resort notify human operators). That capability is called **self-healing**.

Another capability is called **self-awareness.** A system is self-aware when it is able to interpret its own state based on some internal domain knowledge. This implies that self-awareness is a very high-level term and might be realised in various forms (self-healing is one of them). Due to the broad scope of this term many classifications of those systems were introduced, pertaining to levels, scope, span and even type of objects, that the system works with.

Self-organisation is the ability of a system to adapt to changing conditions and various problems that the system faces in a given moment. It is often realised by monitoring itself and reacting appropriately to signals to maintain usefulness. An example would be a swarm of robots coordinating work among each other to execute a complex task that would be impossible to realise by a single robot.

Self-configuration allows the system to autonomously configure and reconfigure itself and its resources when faced with changing environments to maintain its functionalities. An example would be a personal fan which would decide RPMs based on environmental temperature.

The previous characteristics (properties of the NGIoT deployment to be achieved by ASSIST-IoT) will redound on core enablers that may fall under various planes of the architecture. A thorough description of those will be provided in further versions of this deliverable (D3.6 and D3.7).

Enablers of the Self-* vertical

All aforementioned Self-* capabilities will be realised via enablers. At the initial stage of architecture definition, a final clear-cut list of Enablers targeting this vertical is unfeasible to be closed, but a general direction of where our works should be directed towards is given below. Two enablers have been identified as **inherent** of this vertical of the architecture, meaning that they should be present in every non-trivial ASSIST-IoT deployment:

- Resource Provisioning and Coordination.
- Automated Device Connection and Configuration.

Besides, a list of enablers identified so far that will be potentially developed or adapted throughout the execution of ASSIST-IoT project to provide the functionalities expected for this vertical are listed in Table 3. This list is preliminary and thus it is expected to change with the refinement of requirements and in the process of realising the architecture.

Table 3. Preliminary list of potential enablers targeting Self- capabilities*

Enabler Name	Self-* Capabilities	Horizontal Plane Crossed	Description
Reliability and Communication	Self-diagnosis, Self-healing	Smart Network and Control Device and Edge	All pilots and most practical problems lean on reliable communication between its nodes. Moreover, depending on consequences of the faulty behaviour, we will need to ensure that particular components work as intended (for example cranes providing its localisation reliably). This enabler could be extended with capability to either perform self-healing actions (for example: send a request to restart component) or notify human operator that human intervention is required.

Resource Provisioning and Coordination	Self-Awareness Self-organisation	Data Management Application and Services	To tackle problems in a changing environment we might need to increase or decrease resourcing trying to solve them. Well-known example would be scaling up or down software component instances depending on the pressure put on the system. Additional feature might include provisioning resources before actual increased demand based on statistical predictions.
(geo) Localisation	Self-awareness	Device and Edge Data Management	To solve challenges of pilots we need to localise physical objects (containers in ports, workers on construction sites), some devices should be aware of their position in relation to each other (aligning cranes and tractors). We might need to realise localisation using absolute coordinates (GPS) or relative (coordinates in a port). Another form of this enabler might be a logical localisation of data or data lineage.
Monitoring and Notifying	Self-awareness	Device and Edge Application and Services	This enabler could be viewed as a general purpose by representing it as a combination of high-level monitoring module (which would allow to monitor devices, logs, etc.) and notifying module that could send custom messages to predefined system components. For example, on construction sites we will monitor health signals of workers. Those signals should be monitored and in case of breaching some threshold notification should be sent and action might be taken.
Automated Device Connection and Configuration	Self-awareness, Self-configuration	Device and Edge Smart Network and Control	Various devices will be joining networks (cranes, tractors, wristbands). After joining this network, the process of configuring and assigning work to a device should be automated.

4.2. Interoperability

Interoperability is the ability of equipment from different manufacturers (or different systems) to communicate together on the same infrastructure (same system), or on another while roaming. The implementation of the project is about to take place in three different pilots. Considering this, interoperability will play an especially important role in the fruitful completion of each of the pilots, regardless the systems used in each separate case. Interoperability will be undertaken at three levels:

- **Technical interoperability –** means the ability of two or more information and communication technology applications, to accept data from each other and perform a given task in an appropriate and satisfactory manner without the need for extra operator intervention.
- **Syntactic interoperability** – allows two or more systems to communicate and exchange data in case that the interface and programming languages are different (e.g. by using of a standardisation of the communication between a software client and a server).
- **Semantic interoperability** – is the highest level of interoperability which denotes the ability of different applications/artefacts/systems/… to understand exchanged data in a similar way, implying a precise and unambiguous meaning of the exchanged information.

On the software perspective, the term interoperability is used to describe the technical capability of different programs to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols.

Interoperability will be addressed in terms of scalability, security, privacy and heterogeneity of **data sources**. ASSIST-IoT will support **data interoperability** by proposing a semantic data governance toolset, offering data sharing, privacy, security and trust enablers. Another possibility to support the interoperability approaches in ASSIST-IoT is the adoption of DLT. Recognizing the promises that DLT brings to the table, associated enablers will be investigated for the project, especially on which benefits (vis-á-vis interoperability) may Smart Contracts bring. As the technology has passed its novelty phase, hurdles have surfaced during the adoption of DLT in projects. Considering the previous fact, certain components within the DLT scope seem appropriate to be potentially leveraged (after consideration and only to the most essential areas that will harness the benefits of DLT) towards achieving interoperability. In particular, semantic enablers might be supported in ASSIST-IoT by judicious use of DLT-based provenance management gand included in pertinent (established on the basis of requirements analysis) components of the architecture. Hence, DLT may be used (after study and investigation) as a supporting mechanism for enabling semantic interoperability between different IoT networks/platforms..

Interoperability will be a property of the ASSIST-IoT architecture that will be more thoroughly addressed in the next iteration of this document (D3.6). This feature will be tightly related with the development of the first enablers in WP4 and WP5, as well as with the adoption of the design principles outlined in this deliverable.

4.3. Security, Privacy and Trust

Security:

Security applied in NGIoT architectures is highlighted in different building blocks, as already mentioned in ASSIST-IoT deliverable D3.1 (State of the Art). More specifically, references architectures such as IoT-A Project [31] and OpenFog [26] have great influence over the one proposed by ASSIST-IoT.

In general, an NGIoT architecture like ASSIST-IoT's, requires a multilevel and multi-plane approach when it comes to security. Not only all the security, privacy and trust requirements should be guaranteed by the mechanisms provided by this vertical, but they have to act within the horizontal planes and be part of transversal enablers. These specific enables will enforce security, privacy and trust on all the planes of the architecture.

More specifically, the Security, Privacy and Trust vertical will provide the following **functionalities** along the ASSIST-IoT architecture:

- Authorised registration of the IoT devices joining the network.
- Security, privacy, and trust on access and when sharing data for multiple domains.
- Security and privacy for data storage.
- Security monitoring and incident response to avoid cyberthreats.

Access control mechanisms are core security mechanisms for implementing security features that require identification with proper authentication and authorisation. The following terms describe different entities and actions that take part in access control processes.

- Access: is an operation that allows an entity to view or modify information resources.
- Resources: any service, knowledge, or information which is published, shared, or registered.
- Resource provider: any entity or organisation which provide resources in a connected environment.
- Client: any user or organisation which request a specific resource provided by a resource provider.
- Client profile: the identity of a client which provides information about the client and the purpose of requesting a specific resource.
- Permission: a special authorisation rule which govern how a resource is being accessed by the client.
- Capability: a mechanism that contains resource access permissions which is entitled to each profile.
- Authentication: is a process by which the credentials provided by an identified entity are compared with those memorised/created in the system to ensure that it is effectively who or what it claims to be.
- Authorisation is a process of granting, or automatically verifying, permission to an entity to access to the requested information after the entity has been authenticated.

Enablers of the Security part of the vertical:

In order to fulfil the aforementioned requirements within the ASSIST-IoT architecture, the following **inherent** key enablers have been identified so far:

- Identity Manager enabler.
- Authorisation enabler.
- A collection of monitoring and incident response enablers (see Table 4 below).
- Distributed Ledger Technology DLT related enablers (see the reasoning in the Trust and Privacy part and the list in Table 4).

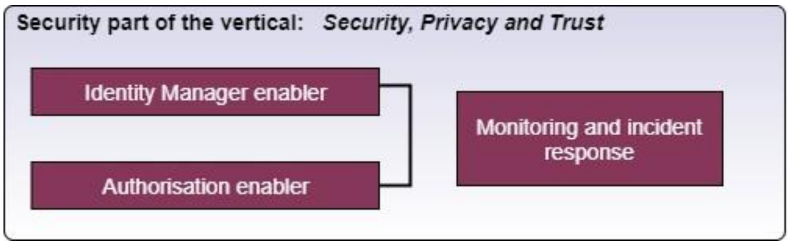

Security part of the vertical: Security, Privacy and Trust
Identity Manager enabler
Authorisation enabler
Monitoring and incident response

Figure 15. Security enablers for ASSIST-IoT architecture

ASSIST-IoT architecture will require a multilevel security monitoring solution that will need to adapt to different services and server instances in all the horizontal planes. One of the major difficulties will be the different objects to manage, and the capacity of those to generate more pieces of information. Security monitoring mechanisms, which basic architecture is shown in Figure 16, will provide security awareness and infrastructure monitoring for threat detection and incident response to the architecture deployed.

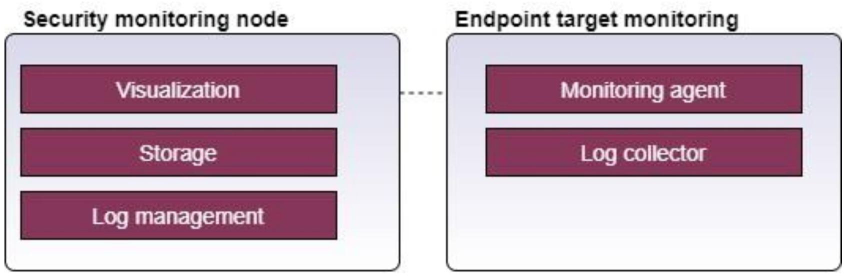


Figure 16. Structure and mechanisms for security monitoring enablers

A tentative list of enablers of this vertical targeting security is presented in Table 4. It should be highlighted that this is a preliminary list and it is expected to change during the execution of the project.

Table 4. Preliminary list of potential enablers targeting Security

Enabler name	Description	Planes involved	Candidate technological components
Identity Manager enabler	Identity manager enabler will be responsible for identifying and authenticating to have access to the resources by associating user rights with established identities. Identity manager enabler will perform authentication phase of access control process. Identity manager will process and validate the identity for later control the access to the resources by the authorisation enabler.	Data Management Application and Services	OAuth2, Federated identity, W3C VCs
Authorisation enabler	Authorisation enabler. Identity manager enabler is focused on authentication while access management is aimed at authorisation. Both processes compose the access control workflow and process. Authorisation enabler is based on XACML standard [32] security	Data Management Application and Services	Software implementations for XACML existing components to be

	policies, results on obligations actions to be deployed after the evaluation process • PAP (Policy Administration Point). Offers the interface for the security policy definition. • PDP (Policy Decision Point) • PIP (Policy Information Point)		evaluated or other modules developed ad-hoc.
Security agent enabler	Perform functions of an endpoint detection and response system, monitoring and collecting activity from end points that could indicate a threat	All horizontal planes	Wazuh agent (Wazuh)
Security monitoring and threat detection enabler	Security monitoring enabler for threat detection and incident response. Provides security awareness and visibility and infrastructure monitoring.	All horizontal planes	Wazuh (Wazuh)
Collector enabler	Collector enabler will perform data shipper function and will enable transference and collection of logs and metrics. Software component based on lightweight agents and send data from machines	All horizontal planes	Beats (Elastic)
Storage and search analytics enabler	Search engine and distributed storage for data collected form the infrastructure under study	All horizontal planes	Elasticsearch (Elastic)
Log management enabler	Log collection and log aggregation enabler to further analysis and process	All horizontal planes	Logstash (Elastic)
Visualisation enabler	Analytics and visualisation platform	All horizontal planes	Kibana (Elastic)

Privacy and trust.

In ASSIST-IoT, privacy and trust per design will be addressed by the introduction of DLT-related enablers. DLT is a novel technology that has numerous uses. DLT is known for the opportunity to decentralise procedures, resilience to changes, anonymity, and immutability to data. The implementation of carefully selected DLT mechanisms within ASSIST-IoT will be tackled from various viewpoints, both purely technological (for the architecture) and dependent on the use cases.

However, privacy and trust are somehow inter-twinned. After an initial research in the literature, a basic structure figure has been envisioned to harmonise the adoption of traditional security elements (see previous Security part) with a DLT infrastructure.

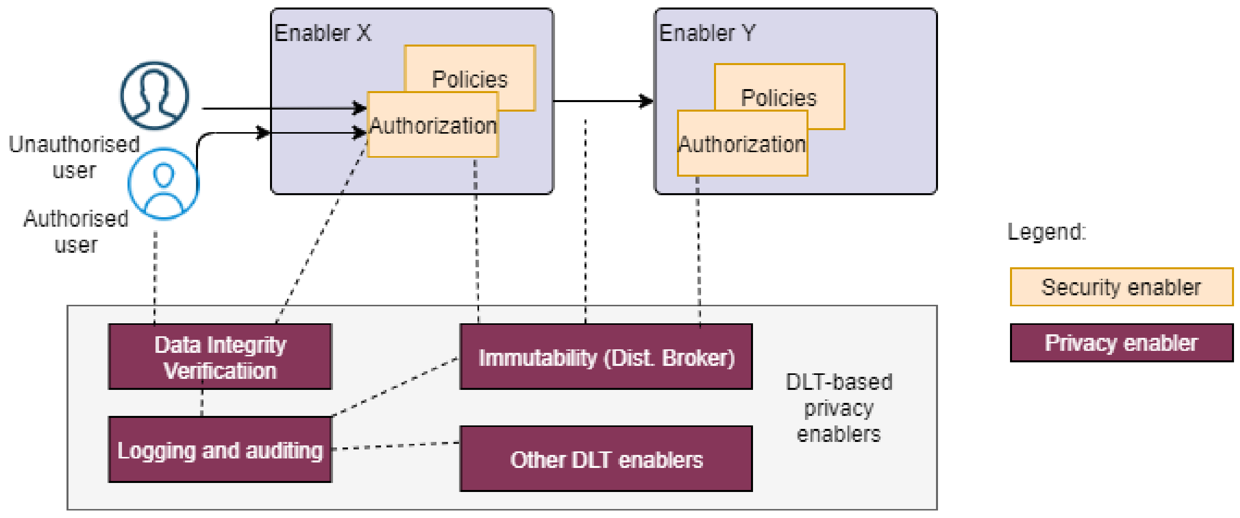


Figure 17. Harmonisation of DLT and Security enablers in ASSIST-IoT

This initial concept is showcased in Figure 17, where the PAP policy (security-wise) can be central, controlled by an admin team and replicated in the DLT - probably using Smart Contracts. The DLT can be used to provide resistance to unauthorised changes to policies.

The possibility of adopting DLT both as a dedicated enabler and as a component of other enablers to handle devices and software inside an enabler will be explored. Regardless of the approach chosen, judiciously applied DLT techniques will enhance the security in sharing data, enforce access control mechanisms, enhance data integrity verification, allow auditing, and support federated learning (to be conceptualised) with its decentralisation.

To sum up, the DLT adoption is envisioned to offer the opportunity for protecting the data along with their transmission by contributing to access control, data integrity verification, and auditing.

Enablers of Privacy and Trust part of the vertical:

About **secured data sharing** in ASSIST-IoT, blockchain and/or DLT-related enablers (e.g., Directed Acyclic Graph -DAG- and/or Blockchain – to be investigated) will be used to guarantee data sharing among heterogeneous devices. A set of enablers within ASSIST-IoT architecture that could be acting as data producers and consumers will be identified and authenticated using DLT-based mechanisms (e.g., using certificates tied with Hyperledger Fabric Certificate Authorities) or Verifiable Credentials (VCs) generated in accordance with W3C specific standards), (see also Identity Manager in the security part of the Section). These mechanisms might be inter-twinned with the distributed broker identified in the Data Management plane of the ASSIST-IoT architecture.

DLT in ASSIST-IoT will allow IoT devices at the Devices and Edge plane to establish secure communication channels and exchange verifiable credentials that act as means of their certification by a root of trust (see also Identity Manager in the security part of the Section). At Data Management plane level, data providers and consumers (potential DLT component to be studied) might have appropriate applications to authenticate themselves using certificates or attribute-provisioning services allowing a control of with whom their data will be shared.

Access control mechanisms in an NGIoT deployment system must be conceived from two perspectives. On the one hand, the security viewpoint was addressed in the previous part of the section. On the other hand, privacy and trust also play a role in their definition. Data access mechanisms based on Smart Contracts deployed on a Peer-to-Peer permissioned network (e.g., Hyperledger Fabric) may further enhance the secure data sharing by controlling the access to (refined) sets of data. Moreover, the usage of Smart Contracts can enable translation of conventional agreements into automated transactions, providing transparency, assurance and provenance, and as such a better and more trusted collaboration among the entities that exchange data.

Another relevant aspect in the Privacy and Trust vertical of ASSIST-IoT will be the **data integrity verification.** DLT-based technologies enable immutability of data kept on the ledger. Proper data integrity verification mechanisms employed in ASSIST-IoT architecture will allow data consumers to verify the integrity of the exchanged data.

Furthermore, the specific property of the immutability of the data kept on the ledger can be leveraged for **logging and auditing selected data sharing transactions** in an immutable way allowing for transparency, auditing, non-repudiation and accountability of actions during the data exchange related actions from the part of the involved stakeholders, i.e., for the enablers of the ASSIST-IoT architecture that are involved in data exchanges.

Besides, an intriguing aspect that this vertical (in the Privacy and Trust part) could cover is **securing decentralised intelligence**. Decentralisation and federation (Federated Learning – FL) are interesting key novel concepts in the ASSIST-IoT technological proposition, therefore the architecture should research most appropriate ways and include enough provisions to allow the "conceptualization and testing" of DLT-powered Federated Learning in the project.

With regards to ASSIST-IoT, the DLT-based FL techniques will be implemented within the architecture for representative scenarios. This approach could enhance the privacy of data exchanged among the edge nodes when they execute AI functions to extract knowledge from contextual and streaming data within the ASSIST-IoT architecture. More specifically, ASSIST-IoT architecture will foster the use of DLT-related components to exchange the local, on-device models (or model gradients) in a decentralised way avoiding single point of failures acting as a component to manage AI contextual information in an immutable form, and avoiding as well alteration to the data.

In general, it should be noted that usage of DLT imposes scalability and performance overheads which may cause hurdles in the use case related requirements for (near) real-time provision of results.

Table 5. Preliminary list of potential enablers targeting Privacy and Trust

Enabler name*	Description	Planes involved	Preliminary candidates
Logging and Auditing enabler	Allow the documentation of data usage and data usage billing, when applicable	Data Management	IDS (Blockchain-based) Clearing House, Hyperledger Fabric Chaincode (Smart Contracts), cryptographic techniques.
Data integrity Verification enabler	Provide mechanism for Data Integrity Verification	Data Management	Hyperledger Fabric Chaincode, cryptographic techniques.
Distributed Broker service enabler	Support immutability and non-repudiation of selected aspects of connections between enablers	All horizontal planes	IDS Clearing House, Hyperledger Fabric Chaincode, cryptographic techniques.
DLT-based Federated Learning enabler	Facilitates exchanges of parameters of on-device local data models in an immutable and decentralised way	Device and Edge	Hyperledger Fabric clients - light nodes, openDSU), DAG (IoTa), cryptographic techniques.

*Please, note that this table is quite preliminary. DLT-based enablers are prone (more than others in ASSIST-IoT architecture) to be changed/re-focused due to their dependency on forthcoming design principles. Extensive discussions are on-going between task leaders, architecture designers and also stakeholder representatives to fine-tune the approach of these (and further) enablers.

4.4. Scalability

Scalability vertical in ASSIST-IoT is a **property** of the system that is present due to (i) the design principles followed, (ii) the container orchestration technologies leveraged, and (iii) the functionalities covered by the Planes and other verticals of the architecture. This means that no specific enablers are provided to guarantee this property, but rather comes implicitly from all the former.

The Scalability vertical addresses the dynamic technical and business needs behind NGIoT deployments in general, and ASSIST-IoT in particular. Because of the variability of edge/fog continuum scenarios for the NGIoT, the ASSIST-IoT architecture envisions to enable elastic scaling deployments ranging from modest barely local operations up to large heterogeneous deployments based on demand features and functionalities. This scalability is essential for in order to adapt to different workloads, performance, costs, and other business needs. From ASSIST-IoT, and following OpenFog RA [26], Scalability will involve three main dimensions: software, hardware and communication capabilities.

Scalable hardware

It involves the ability to add and modify the configuration of the internal elements (either sensors, actuators, or edge/fog nodes) of an NGIoT deployment, as well as the numbers of and relationships between them, including:

- *Computation scalability*: from single core CPUs on PLCs to specialised GPUs with thousands of cores required for AI model training.
- *Network interfaces scalability*: from a single wireless (or wired) interface to large arrays of wireless (or wired) interfaces with aggregate capacities of many Gbps.
- *Storage scalability:* from simple flash memory chips to large arrays of cluster disks.

This particular scalability dimension will be provided mostly by the capabilities of ASSIST-IoT Nodes and Smart IoT Devices, which final extent will be determined during their specification and design stage. The provisioning of new sensors, actuators, interfaces, etc. will be acknowledged by the system, requiring the participation of functionalities from the Application and the Network planes.

Scalable software

It will be of paramount importance, as it will not only include applications, but also the infrastructure to run them, as well as optimal management of those resources.

- The management infrastructure of ASSIST-IoT must scale to enable the efficient deployment and ongoing operation of tens/hundreds of computing/processing nodes in support of thousands of smart and connected things. To do so, a scalable orchestration will manage the partitioning, balance, and allocation of resources across the ASSIST-IoT network. As business data analytics algorithms will handle data of several orders of magnitude, it will also have a particularly aggressive scalability target. To do so, composability and modularity are key aspects of software scalability, where individual hardware and software components are assembled into a NGIoT node optimised to run the specific applications required (e.g., microservices running over basic containers).

To cope with this **scalability dimension, Kubernetes and related technologies for container orchestration at the edge** will be leveraged to schedule and monitor containers in ASSIST-IoT, as explained in Section 3.1. They try to provide these functionalities so developers can focus only on running in the most reliable and safe manner their NGIoT applications. However, although Kubernetes is great at offering a common layer of abstraction across different environments, many companies are looking at it mostly for its extensibility, portability and scalability. Three k8s architectural patterns will be considered, which application will depend on the constrains and available hardware in a particular ASSIST-IoT scenario:

1. ***Kubernetes Clusters at the edge:*** instead of deploying a high availability cluster, a minimal version of k8s in a single-server machine shall be used. Then, platforms to manage and orchestrate container workloads on multiple clusters may be used.
2. ***Kubernetes Nodes at the edge:*** For those cases where the type of infrastructure is limited at the edge, so that it is not possible to consider the installation of a cluster, a minimal version of k8s node at the edge can be deployed, while the main k8s cluster can be placed at a cloud provider or in a colocation data center. Networking may become even more important, so that the k8s control plane can reside in the cloud and k8s nodes, or even in devices at the edge, with an agent to interact with the k8s API.
3. ***Kubernetes Devices at the edge***: The open-source Akri project allows registering native k8s resources leaf devices such as IP cameras and USB devices at the edge. It is a similar pattern as the previous one (a k8s node at the edge is still needed), but it does not need to install Kubernetes on a device.

The use of container orchestration systems is not the only way that addresses software scalability in ASSIST-IoT. Both microservices architecture and containerisation **design principles** contribute to it, bringing the benefits specified in Section 3.1 in comparison to the use of Virtual Machines or the adoption of SOA or monolithic architectures.

Scalable communication capabilities

It involves the ability to modify the configuration of the network elements, including among others:

- *Nodes scalability:* that allows changes in size as more applications, or objects are added or removed from the network (see Section 6). To do so, it is envisioned that the scalability can range from adding capacity to individual nodes by adding hardware like equipment, or by adding software and/or pay-as-you-grow licensing (e.g., X-as-a-service – XaaS).
- *Performance scalability:* that enables growth of capabilities in response to application performance demands (e.g., reducing round-trip-time latencies between sensors and actuator).
- *Reliability scalability:* that permits the inclusion of optional redundant capabilities to manage faults or overloads, as well as to ensure an integrity and reliability at scale.
- *Security scalability*: achieved through the addition of security modules (HW and/or SW) to a basic node based on the stringent security needs (scalable rights access, crypto processing capabilities, or autonomous security features). See Section 4.3 for more details.

As aforementioned, Scalability is a property of ASSIST-IoT that is addressed by the inherent characteristics of enablers from other planes and the design principles adopted. For instance, scalability related to both communication reliability and performance are guaranteed by the joint effort of (i) specific Self-* enablers to be developed/adapted, (ii) the utilisation of SDN paradigm, and (iii) specific data enablers such as the Edge data broker (to be presented in Section 5.3).

4.5. Manageability

ASSIST-IoT will consider the introduction of certain components that involve autonomous decision making (see previous verticals). These components may rely on complex observations (human-centric, image) that require advance management characteristics. Additionally, drawing from the decentralisation approach of ASSIST-IoT, special manageability traits must be researched so that the control of this autonomy and reconfiguration can be done in a distributed way. Besides, the different outcomes of the enablers may feed other enablers' parameters located at diverse locations. All the previous drives the need of introducing manageability features beyond the classic centralised approach (controlling the deployment from a single, cloud data centre).

The purpose of the Manageability vertical in ASSIST-IoT goes beyond the classic management lifecycle of the nodes in an IoT deployment. Manageability in ASSIST-IoT refers **to managing nodes** and every configuration option **over any enabler** running in a particular deployment of the architecture.

Manageability enablers will be developed in ASSIST-IoT architecture to cover the following functionalities:

- Allowing cross-cutting coordination and **orchestration between enablers**. This may be introduced by enablers whose components help reduce complexity and improve configuration capabilities.
- Enhancing the **management of enablers' outputs** (e.g. AI methods results) towards self-autonomy and self-awareness. This will mean providing for (semi-)autonomous actions undertaken by the ecosystem always with the intervention of the human in the process (via these Manageability enablers).
- Creating **end-to-end workflows** involving all pertinent sub-systems. This will mean the creation of enablers (aligned with other verticals such as Security, Interoperability and Scalability) for re-deploying the architecture depending on enablers' outcomes throughout the Edge-cloud Continuum approach. This will also mean the potential creation of **flow orchestrator enablers** coordinating the exchange of information, messages and alerts between enablers (aligned with DLT instructions) and services (see Section 7). Workflow orchestration will be strongly related to decentralisation mechanisms, as those must consider management across multiple nodes (located at various tiers – Section 7) and devices.
- Supporting dynamic ecosystem **re-configurability** (possibly without the need for a restart, or only local restart) available to users, but also supporting Self-* enablers. Re-configuration (in the ASSIST-IoT context) will take place based on the needs of the different deployed enablers and the global system performance (enablers, nodes, network, etc.).

The creation of enablers targeting this vertical will be mainly measured (in terms of benefit for a NGIoT deployment) in uptime. The fact of introducing Manageability vertical (responding to the previous aims) **will optimise the uptime of the different enablers** (other planes and verticals) running in the ASSIST-IoT deployment. This is directly related to the capacity of re-configuration and adaptability of the system.

As an architectural decision, the first design mandate with regards to Manageability will **be to force all enablers to include (in their interfaces) a baseline set of methods/attributes to ensure manageability**. This will mean including a specific series of API functions that all enablers will need to comply with.

Finally, any Manageability enabler to be deployed in an ASSIST-IoT instance must consider the plurality of devices for interaction. This will mean not restricting the management from a central entity and a console, but taking into account different access methods (UI, APIs) from different devices (smartphone) connecting to different nodes of the architecture.

Enablers of the Manageability vertical

The **inherent** enabler for Manageability in ASSIST-IoT will be the **orchestrator of enablers deployment**. While this enabler will be a matter of study within the DevSecOps procedure, it is clearly targeting manageability features in the architecture, therefore will become the crucial element of this vertical.

DevSecOps will be aimed at designing the end-to-end flow of deployment in ASSIST-IoT, considering security, agile integration and methodological concerns. However, the orchestration enabler (of study here) consists of the software mechanism through which one user of the system will actually be able to analyse the Deployment View and **order** the creation/edition/removal of enablers (where, how, when, with which resources, etc.). Additionally, this inherent enabler of the architecture may automatically re-configure the enablers deployed based on a series of rules, events, KPIs, user-settings, etc.

This enabler will be formed of various “enabler components”, which could be a combination of different technologies or be instantiated by a single building piece. This decision will mostly depend on the will of the system owner of each ASSIS-IoT deployment. Initially, the schema of this enabler might be as follows (to be fine-tuned):

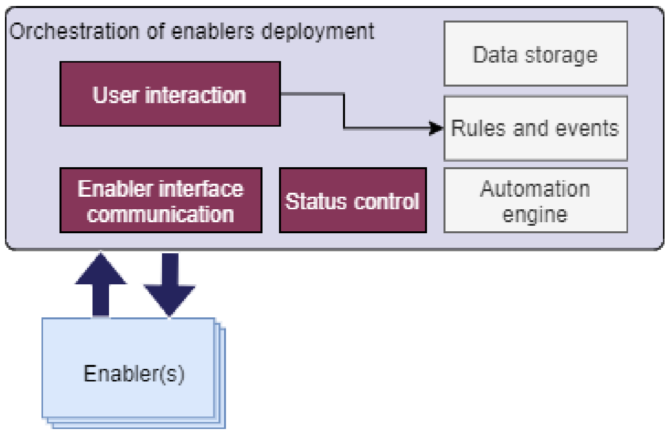


Figure 18. Inherent enabler of Manageability vertical

Particularisation of this enabler, as well as selecting preferred technologies (one per component) will be a matter of study for the next version of this deliverable (D3.6). The enabler template will also be completed by then. Besides, a list of enablers identified so far that will be potentially developed or adapted throughout the execution of ASSIST-IoT project to provide the functionalities expected for this vertical are listed in Table 6. This list is preliminary and thus it is expected to change with the refinement of requirements and in the process of realising the architecture.

Table 6. Preliminary list of potential enablers targeting Manageability vertical

Enabler name	Description	Planes involved	Candidate technological components
Orchestration of enablers deployment	User of the system able to create, edit and remove enablers.	Smart Network Devices and Edge	ISTIO, GKE, MANO, Others from T4.2.
Enablers’ outputs management	Management of the results of the different enablers. UI or tool for allowing the user forward/customise the use of those results.	Data Management Application and Services	-
Workflow between enablers based on events, messaging exchange, or others	UI-based tool for representing nodes, enablers and enabler components, allowing their interconnection (graph-like), following architecture principles.	Devices and Edge Data Management Application and Services	Argo, Apache Airflow, Plynx, Brigade, Dagster, Node-red, Custom enabler components over those.
Devices management	Enabler for monitoring devices and nodes in a deployment, allowing to monitor status and current work (in terms of enabler components).	Devices and Edge	Upswift, openBalena, Particle, DataV, QuickLink Others from T4.1 analysis.

5. Functional View

As aforementioned, the Functional View, also sometimes referred to as Logical View, has the primary objective of showing the functionality required to fulfil the user needs and address the stakeholders' concerns. It describes the main system's functional elements, their responsibilities, interfaces, and primary interactions [3]. Apart from the main functional elements that have to be present within an ASSIST-IoT architecture, the project targets the design, development (or adaptation) and implementation of a set of "core" enablers to enhance the functionalities of the elements of the different planes of the architecture. Hence, the objective is three-fold:

- To define the main functionalities that each plane has to provide in an ASSIST-IoT system, jointly with a functional diagram showing the main interactions between the different identified elements composing the plane.
- To describe the inherent enablers that will be developed or adapted in each plane, aiming at enhancing the basic functionalities provided by the elements of the plane.
- To propose an initial list of potential enablers to be created. This part will consider the different expectations during ASSIST-IoT in terms of pilots, tasks' scope, particular needs, etc.

In the following subsections, a functional decomposition is presented for each of the horizontal Planes of the conceptual architecture. It should be reminded that this is a first version of the architecture and it will be refined in two following iterations, so some interfaces may not be present in this document or may suffer changes during the execution of the project.

5.1. Device and Edge Plane

The Device and Edge plane is the logical abstraction of ASSIST-IoT for the functionalities that will interface with sensors and actuators along with network functions. The innovations to be carried out in the architecture of ASSIST-IoT associated with this plane will fall under four different functional blocks (see Section 3.2.1): (i) Analytics capabilities, (ii) AI capabilities (federated learning), (iii) enhancement of IoT devices smartness and (iv) communication capabilities.

The functionalities of the Device and Edge plane are executed by nodes. Nodes provide the hardware bedrock for the NGIoT architecture and contain hardware and firmware (hardware-specific software) to support the containerisation of enablers.

Nodes (see also Section 6).

In general, a node contains the hardware bedrock on top of which ASSIST-IoT enablers run. To support containerisation, as described in section 3.1, the general architecture of a node consists of hardware, a hardware abstraction layer, an Operating system and a Container runtime (e.g. Docker) on top of which containerised enablers can be operated (see Figure 19).

In ASSIST-IoT, a node can be connected to sensors and actuators (what is called a Far-Edge Node). Sensors might be very basic, like temperature and humidity sensors or might be very complex like LIDAR and cameras, which generate huge amounts of data. Actuators span from traffic lights, fans, etc. to high data consuming displays. The captured sensor data can be made available to the network (through the aforementioned nodes) by means of the communication capabilities and the physical network interface or can be stored in memory. In-memory stored data is also used by the Analytics and AI capabilities. Memory can also be used to prevent data loss in case the network connection is lost (e.g., when a wireless communication interface is used). Sensors and Actuators can be interfaced to an Edge Node or could even be part of it (e.g. embedded sensors), extending the possibilities of the Edge Node. In addition, each node may manage one or several Smart IoT device interfaces. A smart IoT device has the same architecture as an Edge Node. A Smart IoT device can either be connected to an Edge Node or be connected to a network directly through the Physical network interface. The physical network interface can be a wired (e.g. Ethernet) or wireless (e.g. WiFi, Bluetooth, Lora, etc) interface.

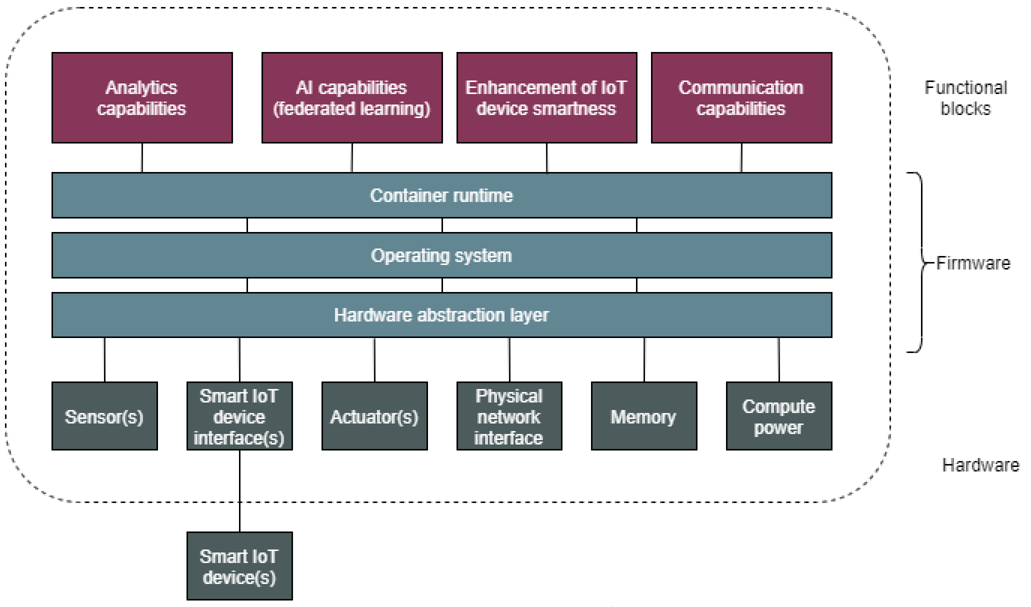

Analytics capabilities
AI capabilities (federated learning)
Enhancement of IoT device smartness
Communication capabilities
Functional blocks
Container runtime
Operating system
Hardware abstraction layer
Firmware
Sensor(s)
Smart IoT device interface(s)
Actuator(s)
Physical network interface
Memory
Compute power
Hardware
Smart IoT device(s)

Figure 19. General node functions

An example of an IoT device smartness enabler can be the annotation of images for AR purposes. In this case, only annotated data needs to be communicated to the node instead of sending huge amounts of image data to the cloud to be annotated and sent back to the node. Analytics capabilities can analyse/filter/process data. The results can be communicated to the network, stored, can be used to control an actuator or can be used as input for the AI capabilities enabler e.g. for federated learning.

Depending on the node application the node can be battery-powered. This means that the balance between energy consumption and compute power needs to be taken care of depending on the required autonomous operating time. Another potential enabler in this regard could be using AI for ensuring energy balance in Edge Nodes.

Functional blocks

Table 7 relates the four functional blocks that have been identified for this plane. The key concept here is that in an ASSIST-IoT deployment, enablers will be developed falling under one of the following building blocks, that serve as an abstraction of the functionalities/innovations of the NGIoT that can be included in this plane.

Table 7. Functional blocks of the Device and Edge plane

Building block	Description	Potential enablers
Analytic capabilities	Analyse data to make decisions based on captured data, vision, audio, text, etc.	Custom-based Python enabler, FPGA and VHDL for latency optimisation.
AI capabilities (Federated Learning)	Framework to implement AI based models and federated learning.	AI capable micro controller or processor from ST, NXP, Microchip, Renesas, etc. to implement compute power. Custom Python component(s) to implement algorithms. Akka for distributed systems.
Enhancement of IoT devices smartness	Application specific intelligence, data processing to enhance data that can be used by e.g. the AI capabilities.	Image annotation, AR/VR engine, signal processing algorithms.
Communication capabilities	Interface with the network and/or Smart IoT device.	Extension in the node for using the following protocols: Ethernet, WiFi, Bluetooth, ZigBee, 5G, RS232, RS485, DHCP, TCP/IP.

At this point, no inherent enablers have been defined. Instead, during the first moths of the project, this plane is focusing on the **design of Technological Components to carry such enablers**. These the two Technology Components in the plane are: (i) a Gateway/Edge Node and (ii) a Smart IoT Device Node. It is the objective of the project to, apart from devising a set of innovative enablers in this plane, to create two novel nodes that will be, per design, compliant with ASSIST-IoT architecture:

- Novel Edge ASSIST-IoT Node.
- Novel Smart IoT Device Node.

These nodes will be prepared to connect with other nodes compliant with ASSIST-IoT and to directly execute enablers being scheduled and orchestrated by ASSIST-IoT structure.

5.2. Smart network and Control Plane

The Smart Network and Control plane is in charge of key aspects of the ASSIST-IoT architecture. On the one hand, it is responsible for the connectivity among network elements, aiming at ensuring low latency and resiliency. On the other hand, this plane covers as well the orchestration of virtualised functions, not only for network-related functions (e.g., VNFs for delivering services such as load balancing, firewall, packet inspection, etc.) but also for Next-Generation functions such as data governance, interoperability, privacy, security, and intelligence, among other functionalities.

This plane has been designed following the SDN/NFV paradigm, considering auto-configuration capabilities to provide continuous support for real-time applications. To this end, the plane is composed of four functional blocks, namely (i) smart orchestrator, (ii) SDN controller, (iii) VNFs and (iv) self-contained network. The functionalities offered by each one of the functional blocks are explained hereinafter, which general components are presented in Figure 20.

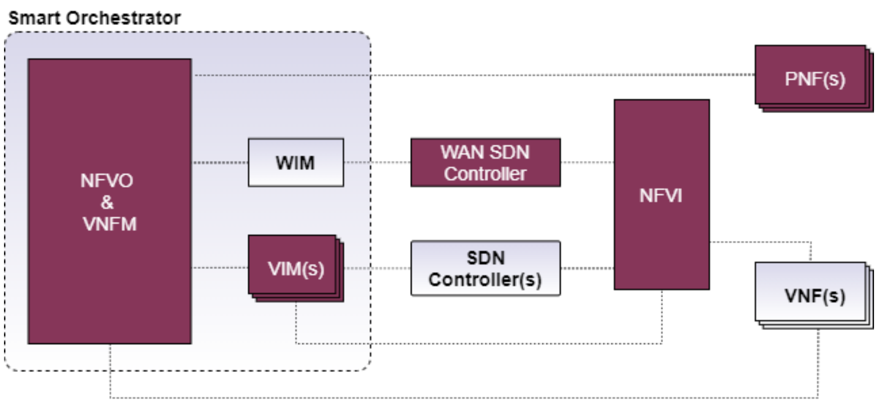


Figure 20. Functional View of the Smart Network and Control plane

Smart Orchestrator

The orchestrator is the central component of a system based on NFVs, being responsible of both resource and network service orchestration. It ensures that network services have their needed computational, memory, network and storage resources, and manages the lifecycle of the NSs deployed over the virtualised infrastructure (NFVI). ETSI has developed a standard for NFV Management and Orchestration, known as ETSI NFV MANO, being the most leveraged architectural framework for NFV orchestration. The NFV MANO (hereinafter referred to as just MANO) is composed of three main components: the NFV Orchestrator (NFVO), the VNF Manager (VNFM) and the Virtualised Infrastructure Manager (VIM).

The NFVO is responsible for different functions, including (i) the onboarding of NSs and VNF packages, (ii) the management of NS lifecycles (from instantiation and scaling to performance measurements and termination) and (iii) global resource management, validation and authorisation of NFVI resource requests. The VNFM is in charge of the lifecycle of VNF instances (instantiation, in/out-scaling, updating and/or upgrading, and termination), and lastly the VIM controls and manages the NFVI resources, while collecting performance measurements and events. Additional functionalities of these components can be found in [33], although new ones are being presented in newer releases of the NFV reference framework.

MANO solutions are usually composed of the NFVO and VNFM entities, thus controlling both VNFs and NSs onboarding and lifecycles, whereas VIMs technologies are installed independently. There are a plethora of solutions, including OSM, ONAP, Open Baton, OPNFV, Cloudify, Tacker, Open-O, etc. ASSIST-IoT will address this functional block not only by implementing it, but the project will develop specific enablers based on two principles. The principles are the **smart auto-configuration** capabilities, which aims at ensuring QoS demanded by end-user services, and **agnosticism of the underlying orchestration solution**.

ASSIST-IoT orchestrator will be initially based in MANO specifications, although keeping a close look towards current trends in case the NFV orchestration paradigm is shifted towards k8s-centric [23]. At the moment, following the general tendency towards containerisation, k8s and related technologies for low-resource devices (e.g., k3s) will be the primary selection for the instantiation of VNFs rather than their deployment on top of Virtual Machines (VMs), this is, leveraging Kubernetes technology as a VIM integrated in MANO.

SDN Controller

This network component is in charge of connecting different nodes of the infrastructure through dedicated switches, or routers, with SDN capabilities. Since SDN-related equipment has the control plane decoupled from the data plane, its functionalities (e.g., traffic prioritisation, output ports, etc.) are taken over by the Controller, which can configure them through dedicated interfaces after making decisions based on information gathered from the network. A general architecture of SDN controllers is presented in Figure 21. In summary, controllers are composed of a core, which contains the main functions related to network configuration and monitoring, and a set of interfaces, which are needed (i) to allow applications to interact will the controller and the data plane of devices (northbound interfaces), (ii) to configure and monitor physical and virtual network devices (southbound interface, typically with OpenFlow controller), and (iii) for communicating with other controllers or with legacy equipment (east/west bound interfaces). It should be highlighted that the exposed interfaces vary among the different controllers available, and that although most of them have the same basic functionalities, different network operations are integrated in them.

Controllers can be applied in different parts of the NFV architecture shown in Figure 21. According to [34], the controllers can be positioned in:

- The VIM, merged with it,
- a virtualised VNF,
- as a part of the NFVI (without being a VNF),
- as part of the OSS/BSS,
- the physical plane, realised as a PNF.

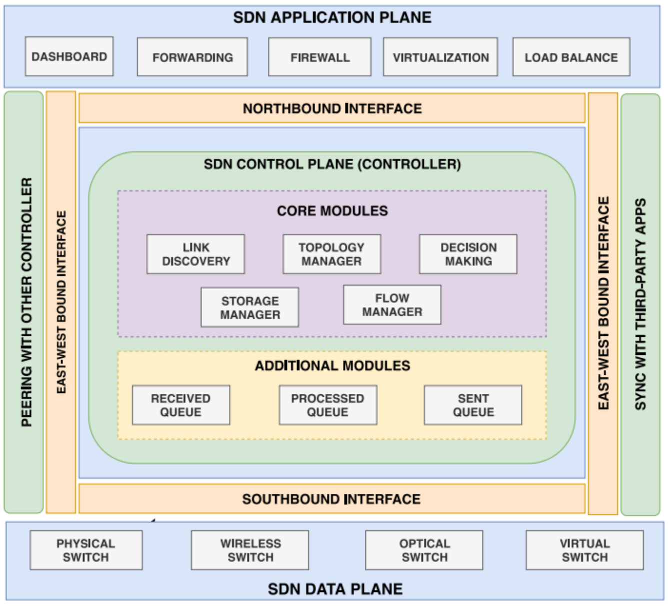


Figure 21. General overview of SDN Controller [35]

Regardless of the position as well as the number of Controllers within an ASSIST-IoT supported deployment, they must ensure connectivity not only among the VNFs deployed on a particular NFVI Point of Presence (PoP, for instance, an edge server with virtualisation capabilities), but also among different NFVI PoPs that may be available in the same site (i.e., among those nodes with virtualisation capabilities). An ASSIST-IoT SDN Controller must provide at least **services** (a) for routing data based on usual routing protocols, (b) for topology discovery and management, (c) for tracking of the elements, and (d) for capturing packets metrics to have information of the traffic of the network, having all this information stored within (e) a storage manager.

Regarding its interfaces, **they must at least provide OpenFlow 1.3 as one of the southbound interfaces**, and a REST northbound interface accessible through a command line interface (CLI) for facilitating the access of SDN applications.

In ASSIST-IoT, apart from selecting and deploying an SDN Controller (or many), it will be supported by **enablers** to **improve the performance of SDN-based networks, leveraging telemetry and historic information and/or network-specific AI models,** in terms of routing, filtering, access, etc., targeting NGIoT deployments. Hence, a set of enablers will be developed to align ASSIST-IoT with the concepts of intent-based networking (IBN[13]). Besides, it should support decentralised topologies and high availability, so in case of failure of a Controller, its managed nodes can be controlled by another. The latter can be achieved in different ways, for instance, by either using a slave Controller instantiated in a different node which monitors the master Controller and take over only in case of failure, or by having a clustering of Controllers within the same node [36]. As with the MANO component, the necessity of an abstraction layer will be explored aiming at decoupling the actual selection of the SDN Controller from the provided technology.

It should be highlighted that despite the fact that SD-WAN solutions also make use of a dedicated SDN Controller, they fall within the responsibility of the self-contained network functional block and hence it is outside of the scope of this building block.

VNFs

VNFs are the cornerstone of any NFV deployment, since the former are the logical result of the latter. VNFs execute particular network functionalities usually over generic virtualised hardware, functionalities that outside this paradigm have to be performed by dedicated hardware. Although they were originally envisaged to be instantiated on top of Virtual Machines, current trend is moving towards the use of containers. Thus, it is not unusual to refer to the later as Cloud-Native Network Functions (CNFs) or Kubernetes-based VNFs (KNFs). In any case, a Network Service can be composed of a combination of one or different types of virtualised functions, and even also Physical Network Functions (PNFs), which refer to classical dedicated hardware that performs a specific network function. In ASSIST-IoT, with the exception of PNFs, the rest of virtualised functions will be referred as VNFs.

VNFs aims at providing various network functionalities such as load balancing, firewalling, WAN acceleration, packet inspection, etc. In any case, software solutions from non-network scopes can be virtualised, so VNFs for providing other functionalities such as data governance, scalability, security, intelligence, etc. will be integrated so they can be managed as well by the Smart Orchestrator of ASSIST-IoT. In any case, the latter functionalities will be provided by the different software enablers envisioned in ASSIST-IoT, and hence this section will be focused only in VNFs that provide network-related functionalities.

In principle, any kind of network functionality falls under the scope of this building block, however, only a set of them will be implemented within the project, according primarily to their relevance in the considered pilots. These VNFs include: (i) WAN acceleration, which aims at optimising data transfer efficiency in Wide Area Networks; (ii) v5GC, to have a virtualised Core for private and custom 5G access networks; (iii), traffic classification, to categorise the network traffic into a number of application classes; (iv) virtual switching, to have the possibility of having not only physical but also virtual switches in the network; (v) load balancing, to distribute workloads according to the resources of the physical nodes of the Device and Edge plane and (vi) link aggregation functions, to combine different access networks for transmitting and receiving data. If additional functions are needed either for the particularities of the use cases or required for implementing the listed ones, they will be developed and indicated in the next version of the architecture (i.e., D3.6).

Self-contained network

This building block response to the necessity in some potential deployments of provisioning a private network that works over a public one, ensuring anonymisation and security of communication. Different technologies can be leveraged to realise this kind of private Wide Area Networks (WAN), which selection depends on the actual requirements of the deployment: Virtual Private Networks (VPNs) and Software-Defined WAN (SD-WAN).

[13] https://www.cisco.com/c/en/us/solutions/intent-based-networking.html

On the one hand, VPNs are secured networks that transmit data in an encrypted form between two network elements, usually to facilitate the connection of a device to a network different to the one it belongs to (remote access), although it can also be used to connect two networks (site-to site VPN). They can be implemented by means of technologies such as IPSec or SSL. On the other hand, SD-WAN is proposed to apply software-defined techniques in networking connections over wide geographic areas [37]. It simplifies the connections between different physical sites, while providing centralised monitoring and control with flexibility and low cost. The main advantages of this network are that (i) similarly to SDN, if allows defining network policies and manage traffic from a centralised location without having to configure each device, thus simplifying network management tasks, (ii) it considers application-level requirements to guarantee QoE for particular users, locations or applications [37] and, in addition, (iii) it can manage different underlying networks (e.g., MPLS, LTE, Internet, satellite…) as well as aggregating all the link bandwidth to increase the total throughput.

ASSIST-IoT will comprise both types of technologies: VPN and SD-WAN. The VPN connections are to grant access of external nodes to an ASSIST-IoT deployment, while SD-WAN technology will allow connectivity between different sites and enabling the orchestration of NSs with VNFs located in different PoP of the NFVI. To realise the latter, ETSI MANO specifies an additional component named WAN Infrastructure Manager (WIM). It can interact with the NFVO to control its dedicated SDN Controller, which will prepare the NFVI PoP network gateways of the different sites to allow their connectivity. The WIM component will be part of ASSIST-IoT, interacting with the rest of components of the plane as can be seen in Figure 22 below. Initially it is plan to leverage Internet for the realisation of SD-WAN within ASSIST-IoT, however, it should be extended to support other underlying networks such as the aforementioned.

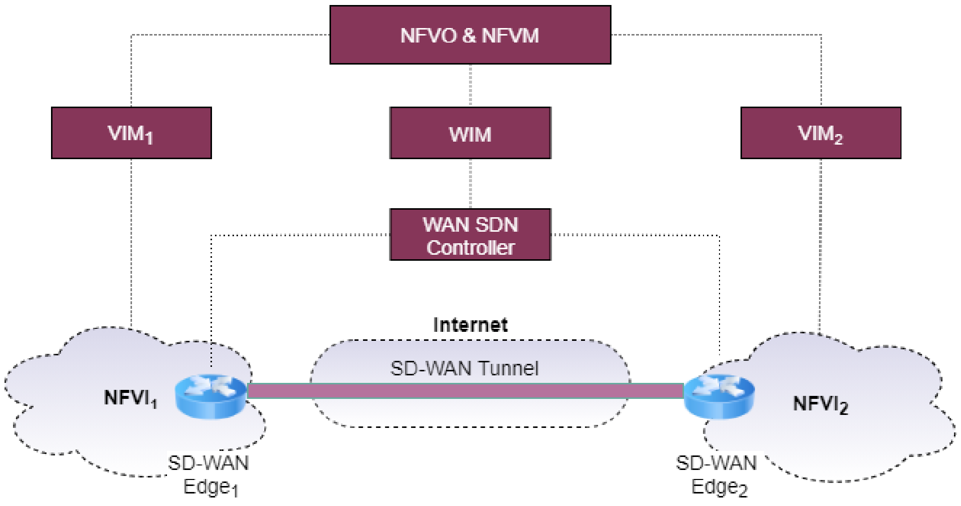


Figure 22. Interaction between ASSIST-IoT components

Enablers of the Smart Network and Control Plane

According to the layered approach in ASSIST-IoT architecture, the main features of this plane will be covered with the following **inherent** key enablers, which will be aided with a set of VNFs for providing specific network functionalities:

- Smart Orchestration enabler.
- SDN Controller.
- Auto-configurable network enabler.

Other enablers have been identified to be developed, however, they are not inherent to any ASSIST-IoT deployment since they use depend on the particular network topology of a deployment scenario (for instance, enablers related to WAN or VPN only make sense in multi-site environments, or a link aggregator enabler is only needed if there are multiple access networks and applications needs it). These other identified enablers are listed jointly with the inherent ones in Table 8, aiming at providing the functionalities expected from this plane. This list is preliminary and thus it is expected to change with the refinement of requirements and in the process of realising the architecture.

Table 8. Preliminary list of potential enablers targeting the Smart Network and Control plane

Enabler name	Description	Candidate technological components
Smart orchestration enabler	This enabler will facilitate the interaction with MANO components (NFVO and VIM). It will provide functionalities as VNFD and NSD validation while serving also as abstraction layer (decoupling actual MANO selection from ASSIST-IoT architecture).	OSM, ONAP, OpenVIM, Openstack, custom components.
SDN Controller	This enabler will manage the forwarding plane of the physical and virtual switches of the ASSIST-IoT architecture, based on the business logic applied through its northbound APIs.	OpenDayLight, ONOS
Auto-configurable network enabler	Enabler to generate and apply policies for the SDN controller to improve and optimise the network performance, based on AI mechanism, for support QoS in multi-application traffic scenarios.	Tensorflow, Keras (AI platform), AI network models.
Traffic classification enabler	Enabler to classify network traffic into a number of application classes (video, VoIP, etc.), based on ML algorithms.	ETSI ENI specification, custom classifier, AI platform
Multi-link enabler	This enabler aims at selecting among the available wireless access network technologies (cellular, WiFi, fluidmesh, etc.) for transmitting data based on the target application. Additionally, this enabler will provide reliability mechanisms, so in case one access technology stop working it would switch to another	Custom component
SD-WAN enabler	Enabler to provide access between nodes or devices from different sites based on SD-WAN technology. It will consist of three elements, a dedicated node (SD-WAN edge), an SD-WAN controller and a configuration server.	FlexiWAN, Nante-WAN, Custom WIM component, SDN Controller (does not have to be the same as the one considered as enabler)
WAN acceleration enabler	Enabler to increase the efficiency of data transfer in Wide Area Network. It will leverage different techniques such as compression, latency optimisation and traffic shaping, among others.	Custom component
VPN enabler	Enabler to provide access to a node or device from a different site considering VPN technologies in a secure and seamless way, outside the scope of the SD-WAN enabler. It will consist of a server and a client.	Wireguard

5.3. Data Management Plane

The Data Management plane encompasses any process, in which data is processed to deliver features concerning data interoperability, annotation, security, acquisition, provenance, aggregation, fusion, etc.

The functionalities of this plane very often have a supporting role for other processes or applications. For example, data communication channels in a heterogeneous fog environment have additional properties or constraints that need to be observed. They pertain to the dynamically changing needs, but also momentary capabilities of delivering and consuming data. This dynamic process of data supply and demand introduces new challenges, in delivering data, where and when it is needed, taking into account security, network performance, processing capabilities of the receivers, and even predicting demand, before it is explicitly reported. This series of challenges are addressed by a data broker enabler.

A separate concern is data interoperability i.e. the capability of the data to be understood by multiple cooperating systems. ASSIST-IoT approaches data interoperability with the semantic approach. It addresses the steps needed to produce and consume so-called self-describing data, such as semantic annotation, translation and harmonisation. In controlled systems, semantic annotation can, by itself, introduce interoperability by enriching data with commonly understood and shared schemas, ASSIST-IoT goes a step further in using a broader approach of semantic translation, in which data can be made interoperable, even if annotated with different schemas. A number of supporting solutions, such as a repository of ontologies, is also planned.

The Data Management plane also contemplates leveraging DLT enablers (see 4.3) to enhance data security mechanisms. DLT possesses characteristics that could assist the secure storing, transfer, and handling of IoT. In more details, DLT can allow the decentralisation of the data management and can secure the data as the immutability of data is one of the DLT characteristics. This immutability property will be applied to a set of data integrity verification DLT-based mechanisms as well as to a set of DLT-based communication, among different architectural layers, auditing mechanisms. Semantic data interoperability between the different layers of the assist-IoT architecture may be supported by a DLT-based mechanism. Besides, the decentralisation of the data has the potential to support federated learning, for example a distributed ledger can accommodate hashes that points to the data that are stored in devices. The immutability of the data in DLT ensures that the data have not been tampered with. Moreover, there is the opportunity to manage and control the ownership of the data in a DLT environment. In a previous section dedicated to Interoperability, mechanisms leveraging selected DLT technologies have been described to showcase the support of those enablers to data interoperability in ASSIST-IoT.

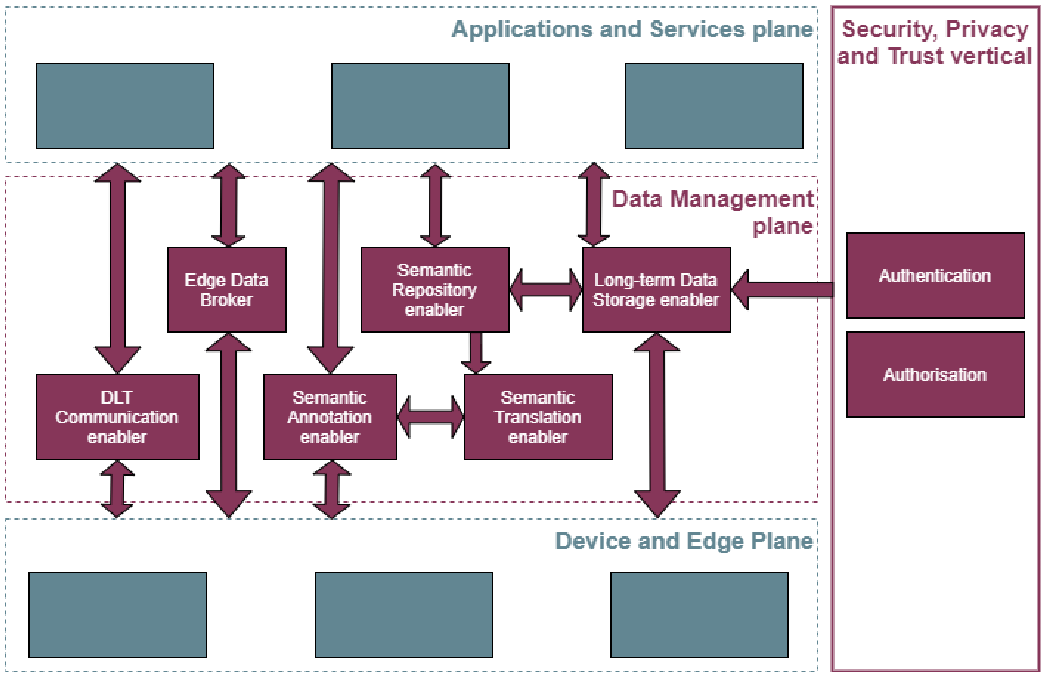

Applications and Services plane
Security, Privacy and Trust vertical
Data Management plane
Edge Data Broker
Semantic Repository enabler
Long-term Data Storage enabler
Authentication
Authorisation
DLT Communication enabler
Semantic Annotation enabler
Semantic Translation enabler
Device and Edge Plane

Figure 23. ASSIST-IoT Data Management plane draft interconnections diagram

Additionally, the rules for processing of data on this plane will be prepared as a supporting work for processing heterogeneous data of varying degree of confidentiality. Although this action will not result in a technological (software or hardware) enabler, it will deliver guidelines and descriptions of pitfalls and rules, that must be observed and attended to, when handling sensitive and non-public data.

Enablers of the Data Management plane in ASSIST-IoT:

The main functions of this plane will be covered with the following **inherent** enablers, with key support from security functions provided by enablers in the Security, Privacy and Trust vertical, in particular the authorisation and authentication enablers:

- Long-term data storage enabler
- Semantic translation enabler
- Edge data broker

In Table 9, the initial list of enablers to be developed or adapted to provide the functionalities expected from this plane are listed (see also Figure 23). Similarly to the lists presented for the former planes and verticals, this list is preliminary and thus it is expected to change with the refinement of requirements and in the process of realising the architecture.

Table 9. Preliminary list of enablers on the Data Management plane

Enabler name	Description	Candidate technological components
Semantic repository enabler	A database of data models and ontologies offered publicly. This enabler provides access to data models as a service. The key features include versioning (different versions of data models), ownership (only the data model owner may update a data model), provision & search (data models are public and searchable) and documentation (documentation provided by data model owner is served with the data model itself). This enabler supports data interoperability through shared data models and semantics.	Custom component on CMS/Django, reSpec
Semantic translation enabler	An intermediary in communication capable of translating streaming messages or batch data, preserving their semantics. It enables semantic interoperability, even in a system without a central shared data model.	Custom component on Apache Kafka, Scala, Akka, Apache Jena
Semantic annotation enabler	An enabler that annotates data with data model or ontology information, in order to bring it to the semantic level. It has a supporting role in data interoperability, and is dedicated to data that wants to benefit from semantic technologies, but is not yet represented in a semantic (i.e. self-describing) way.	Custom component on Scala, Apache Jena
DLT communication enabler	Will contemplate how to leverage DLT-enablers distributed across many nodes, in order to enhance data integrity verification, as well as communication auditing mechanisms. The use of DLT in this enabler is meant to provide mechanisms that can detect data tampering in any context – whether changing or sending a sensitive piece of data, or logging data access and resource requests. Moreover, supporting Semantic Interoperability mechanisms could be incorporated into this enabler.	Hyperledger fabric, OpenDSU
Long-term data storage enabler	Provides dedicated storage space for users, services and other enablers. The role of this enabler is to serve as a secure and resilient storage, offering different storage sizes, individual storage space for separate users, and the promise, that the data will be kept safe, in face of various kinds of unauthorised access requests, or hardware failures.	Custom component on MongoDB, Neo4j
Edge data broker	Enables the efficient management of data demand and data supply from/to the Edge Nodes. It optimally distributes data where it is needed for application, services and further analysis. Data distribution is based on reported demand and available resources at the Edge Nodes. It provides: subscriptions and messages between the broker and the Edge Nodes; management of message scheduling, routing and delivery; common interfaces for searching and finding information.	Custom component on RabbitMQ, Apache Camel

5.4. Application and Services Plane

The three ASSIST-IoT horizontal Planes discussed above facilitate the collection of real-time data from massively distributed sets of sensors and heterogeneous networks. However, all these data shall be exposed in a human-centric approach to NGIoT end users.

The ASSIST-IoT Application and Services plane is intended to provide access to data via human-centric configuration enablers. These enablers can be seen as the App Entities envisioned in the AIOTI HLA [28], i.e., features to provide application logic, which may include data visualisation and user interaction services, data analytics capabilities, various kinds of data processing capabilities, data protection support and/or data management logic[14]. To do so, ASSIST-IoT will develop and deploy software platforms for helping human decision-makers to coordinate operations with special focus on sharing visibility of what is happening in various parts of the NGIoT deployments. In particular, three main working lines are envisioned in this horizontal plane:

- The development of client-side applications, also called frontend applications or dashboards, will enable users to see and interact with content in a user-friendly interface, including the support of business analytics that makes use of Intelligent Decision-Making services. To do so, a mixture of HTML, CSS, JavaScript, and ancillary libraries is foreseen.
- ASSIST-IoT will not only address web-based or mobile-based dashboard applications, but the project will also evaluate innovative interaction mechanisms like AR/MR interfaces, considering real time requirements and human-centricity. This type of interfaces has the potential to make an immense impact on how people work and interact in the future, boosting operational performance efficiency (such as cranes remote operating stations), or guaranteeing workers' safety (such as identification of abnormalities in workers' physiological parameters by OHS managers), as well as end-users' safety (such as augmented vehicle maintenance support).
- Finally, as an open platform, ASSIST-IoT system will expose its applications capabilities via open APIs to boost open experimentation. Therefore, the ASSIST-IoT Applications and Services plane will also provide open APIs, over which more advanced external functionalities coming from third parties can be onboarded, demonstrated and validated in pilot facilities.

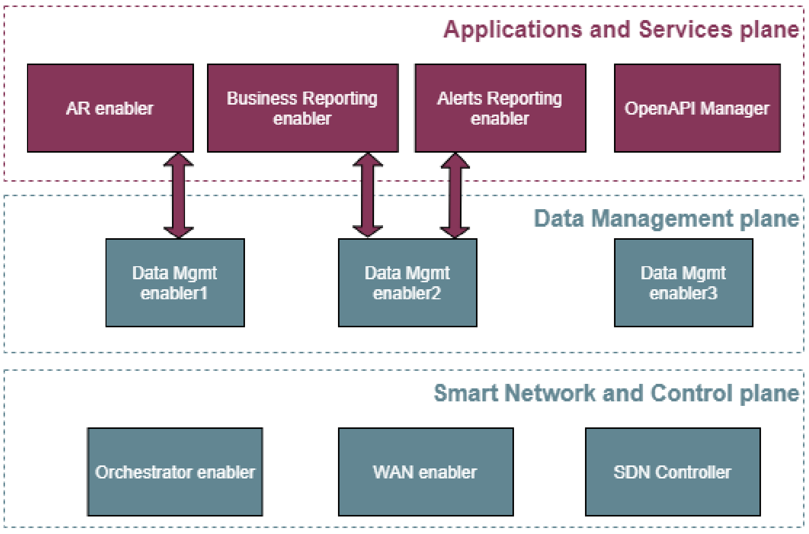


Figure 24. ASSIST-IoT Applications and Services functional model.

Enablers of the Application and Services plane in ASSIST-IoT:

In this particular plane, classifying an enabler as **inherent** is a challenging task since the presence of them is strongly dependent on the actual use case. In any case, the following enablers have been identified as such, following the principles of (i) monitoring being a key functionality that should be present in any novel NGIoT environment, and (ii) human-centricity being one of the cornerstones of ASSIST-IoT:

- Business KPIs reporting enabler.
- Performance and usage diagnosis enabler.
- AR/VR/MR enabler.

The full list of enablers identified so far for this plane is listed in Table 10. This list is preliminary and it is prone to changes during the execution of the project, based on the evolution of requirements and in the actual realisation of the architecture.

[14] In addition, the App Entities may include support for cybersecurity and trust.

Table 10. Preliminary list of enablers on the Applications and Services plane

Enabler name	Description	Candidate technological components
Business KPIs reporting enabler	All valuable metrics and figures for the industrial stakeholder/end user to be available for representation in dashboards, reports, etc.	Grafana, Freeboard, Mozaïk, Kibana, Tipboard, Smashing, Dashing, ELK stack, MS Power BI, Microstrategy, Tableau
Performance and usage diagnosis enabler	Performance and diagnosis numbers to be collected, so the system could highlight some problems and act in accordance (notify to the admin, automated recovery, fine tuning machine resources).	Grafana, Freeboard, Mozaïk, Kibana, Tipboard, Smashing, Dashing, ELK stack, MS Power BI, Microstrategy, Tableau, Prometheus
AR/VR/MR enablers	An AR vision and 3D rendering engine that will mix the virtual content with the real world (including localising the AR device relative to the real-world reference, and represent digital augmented scenes).	Holo Lens SDK, Unreal Engine, Unity3D, Google ARCore, AR.js, ARToolKit+, DrokidAR, Vuforia
OpenAPI Management	The open call experimenters should subscribe to the different Open APIs (like API creation, publication, securing, and monitoring) via an API manager.	Apigee, 3scale, IBM API management, Akana, Kone, MuleSoft, MS Azure API management, Postman, WSO2

6. Node View

The View of the Node in ASSIST-IoT responds to **a structural aspect** of the architecture. Per definition (see in Section 3.2), one *enabler of ASSIST-IoT instances its enabler components in one or various nodes of the infrastructure of the deployment*. This chapter aims at describing and characterising what exactly conforms a node, how it is expressed in the architecture and its role in the whole play.

Simply put, an ASSIST-IoT node is a **hardware element** within the deployment network that can provide computation capabilities by running some ASSIST-IoT enabler execution. These **nodes might vary** on geographical location (close to the source, same building, same city, remote…), on topological spot (see Far-Edge, different Edge tiers, etc. in Figure 26 in the next section) on processing capabilities (datasheets, restrictions, technical specifications) and on application domain (applying only a certain subset of potential enablers, for instance, to an SDN-enabled switch, a Smart Gateway realised by a Raspberry Pi or a full-fledged server able to train complex AI models).

The main role of the nodes in ASSIST-IoT architecture **consists of running (executing) enabler components**. These components may also be quite varied (e.g., databases, specific server endpoints) and will base their application on the node characteristics. Typically, the nodes closer to data sources (e.g., Smart IoT devices) will focus on I/O processes for monitoring and actuation, whereas remote nodes (e.g., cloud server in a data centre) will focus on centralised, massive computing components (e.g., inference engines, Big Data storage).

On the other hand, being the essential piece of ASSIST-IoT communication, the nodes in the architecture will need to be able to **interact hierarchically (north-south) and laterally (east-west) with other nodes**. The latter will be the cornerstone for building decentralisation of the system upon. This communication (in ASSIST-IoT) will be realised via REST API mechanisms, through which the different nodes will share status and components' data to allow enablers properly function. As a matter of fact, the functioning of enablers in an ASSIST-IoT deployment may set this interaction as their essential feature. As outlined before, these communications (north-south and east-west) will take place providing that the instantiation of ASSIST-IoT already possesses sufficient network connections. The Node View **takes the underlying network connectivity for granted**.

The Node View is represented in ASSIST-IoT using the following abstraction. Figure 25 represents the basic capabilities that are considered in the Node View. This schema covers the communication, processing, storage and virtualisation properties that are needed for characterising a Node and for spotting it within the system.

First, the two blocks at the bottom of the figure aim at depicting the southbound communication capabilities of a Node and the direct connections with the physical world, correspondingly. These will be especially relevant in low-tier nodes (such as Smart IoT devices and Far-Edge nodes) for monitoring IoT devices and acting over them.

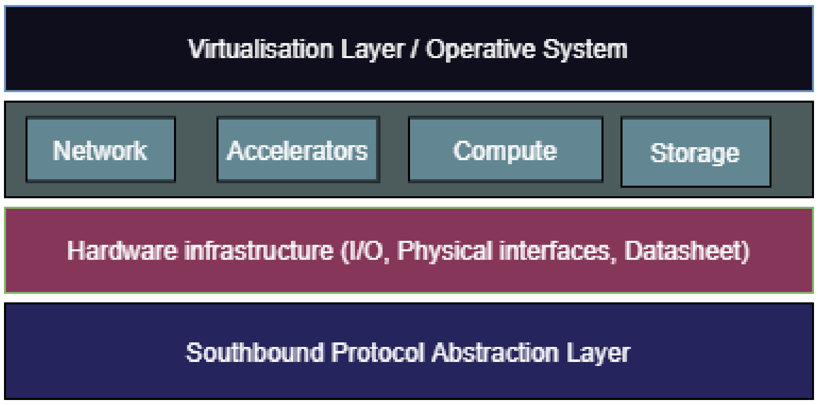

Virtualisation Layer / Operative System
Network
Accelerators
Compute
Storage
Hardware infrastructure (I/O, Physical interfaces, Datasheet)
Southbound Protocol Abstraction Layer

Figure 25. Node View in ASSIST-IoT architecture

Scaling up in the abstraction, the most relevant properties of a node will be its network capacities (network connections, IP address, etc.), its accelerators (number and type of GPUs, TPUs, FPGAs, etc.), computing capabilities (CPUs, cores) and its storage capacity. This information will be the baseline to build the enabler components. Depending of these capacities, a Node will have different roles and will be able to support a certain number of enablers.

At the top of the schema, it represents the operative system and/or virtualisation/containerisation option selected. Any node in ASSIST-IoT will have (most likely) its own operative system (Linux OS, Windows, other) or its own virtualised/containerised environment (VMWare, OpenStack, Kubernetes) allowing them to be sub-divided in smaller computing entities. This representation will be very useful from ASSIST-IoT perspective as this **will highly drive the different techniques/enablers that will be able to be used for that node**. In particular (see Section 5), ASSIST-IoT will base the enabler deployments on the containerisation of "enabler components" and its distribution as a Kubernetes service among nodes. Following this approach, ASSIST-IoT nodes will function as K8s nodes (acting either as master or slave nodes). Therefore, K8s will need to run over the current OS of the node. In the case that this would not be feasible (operative restrictions, logistic restrictions, low resources availability, etc.), the system will need to know which environment exists in that node to provide an alternative mechanism for running "enabler components" upon it.

The previous form the **"static" View** of one Node in ASSIST-IoT. Apart from those, each Node will have general attributes for identifying and handling them within an ASSIST-IoT deployment. These will be an identification number, hostname, spot, current location, and the type of node per each (see Appendix B). These data will be used by different enablers to manage the nodes, including manageability features and operative processes enabling the execution/scheduling of "enabler components" on them.

Additionally, Nodes will also need to be **dynamically monitored by and interacted with enablers**. With that purpose, an additional set of abstract properties have been defined. It has been considered that, at any moment, one Node **will be running a series of enabler components** (normally more than one). Some traits of ASSIST-IoT architecture are self-reconfigurability, manageability and adaptation. To materialise those, the owner of the deployment (and the system itself) will need to know exactly which "enabler components" are being executed in each node. This also opens the possibility to monitor system's performance and facilitates starting/stopping/interacting with those enabler components. It has been designed that every node must keep a constant record of those and must make that record accessible.

For describing all the previous, one Node will be continuously described within ASSIST-IoT **using the template that is provided in the Appendix B - of this document**.

7. Deployment View

At this point, all basic elements of the ASSIST-IoT architecture have been explained. On the one hand, the enablers (framed within Planes, functional blocks, Verticals, etc.) form the key asset the other elements orbit around. On the other hand, the nodes (HW) hold the computing load in the architecture. However, a crucial aspect is missing for gluing all the previous together. The way that nodes and enablers are scattered through a specific infrastructure in the deployment form the **Deployment View**.

The Deployment View of the architecture aims at presenting how an ASSIST-IoT architecture is deployed to address the specific uses cases and business scenarios. It consists, from one side, of a set of different **ASSIST-IoT nodes** (presented in the previous section), distributed within different edge tiers, which are **connected to other physical elements to create an interconnected system.** Systems can range from small embedded to large fully connected ones, responding to the specific scenarios and use cases that the system aims at addressing. Figure 26 represents an edge deployment of different tiers, considering Far-Edge nodes, Edge Nodes and Cloud nodes. From the other side, it covers the aspects of the architecture **relevant for building, testing and maintaining services in the system**. The different enablers active in an ASSIST-IoT deployment will target specific objectives/functionalities. However, the architecture foresees **the combination of enablers to build services**. Here (for ASSIST-IoT), the concept of service refers to a specific application offered by the technology providing measurable actions/functionalities to the final user. Services (in this context) must not be confused with the Application and Services plane division (see section 5.4). While the Application and Services plane is a Functional View of the traits of ASSIST-IoT spotted in the higher layer of the IoT, one service in the Deployment View may consist of the combination of, for instance, three enablers spotted in the Device and Edge plane and Data Management plane.

The Deployment View is mainly addressed to the "system owners/developers" actors. This View will allow all of the actors to have an all-encompassing sight of a particular service: which enablers it is consisted of; which nodes are being used, how the enablers are interacting among them, etc. However, this View is also useful for an end user to observe at. For a stakeholder (that just wishes to obtain a functionality from the IoT system), this View provides a "snapshot" of where and how is ASSIST-IoT being instantiated on their infrastructure. As a joint of both, the information managed in a Service Development View will allows those actors to take specific actions such as maintenance, introduction of new equipment, resizing, etc. From a bird's eye, this View will consist of a set of software and hardware pieces that are used by ASSIST-IoT. These pieces will be completely adjusted (and restricted) to the network infrastructure that each deployment can provide. Finally, this View will also allow those two actors build on two directions: (i) analysing the introduction of new services (e.g. monitoring of sensor data and applying AI models over it), and (ii) hardware technologies (e.g. AR Glasses, new network elements adjusted to improve latency, bandwidth, etc. or to connect to external systems).

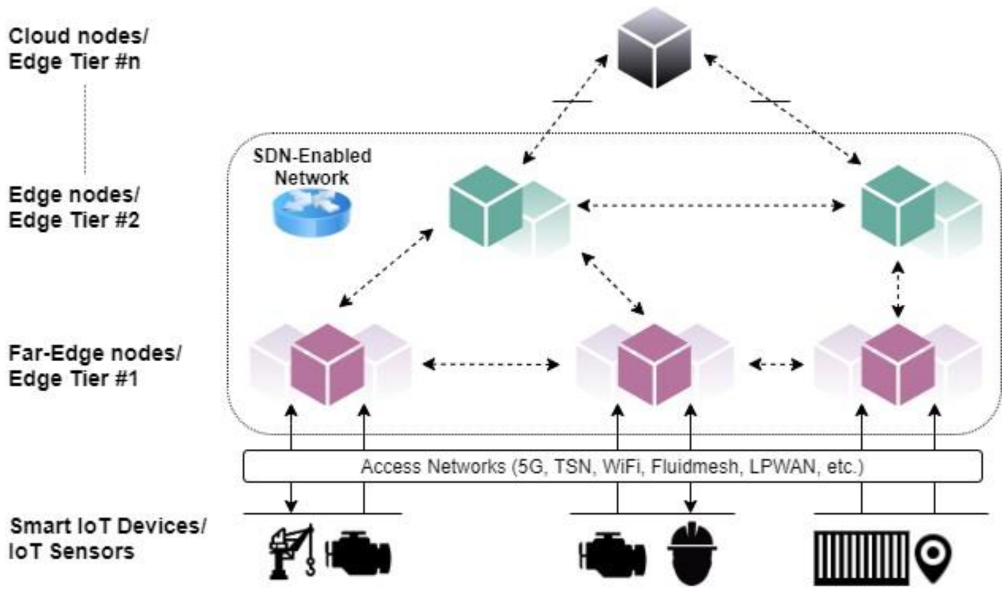


Figure 26. Deployment View of ASSIST-IoT architecture

The Deployment View represents a typical hierarchical deployment of nodes of the Edge Continuum paradigm. However, some considerations have to be explained to avoid potential misleading in the interpretation of this View. The **characteristics and design criteria of an ASSIST-IoT deployment** are driven by the following:

- ASSIST-IoT aims at extending the **capabilities of decentralised, distributed architectures**: it must support specific use cases that fall under the scope of just IoT Devices and Far-Edge tiers, without involving upper tiers or even a central one.
- Many of the (software) enablers provided by ASSIST-IoT can be part of not only decentralised but also centralised topologies, and for this reason the Deployment View represents this possibility.
- Regarding the hierarchy, a central node might be part of the physical site (e.g., higher edge tier) or be part of a cloud infrastructure (which would prevent it to be directly involved in low-latency communications).
- **The number of tiers and nodes is dictated by the requirements of a particular scenario.** To design it, it is needed (i) to know the hardware capabilities of the available nodes, (ii) to evaluate the workload that will be executed in each tier, (iii) to evaluate the number of devices involved, (iv) to define the workloads to be executed by each tier, and (v) to have the requirements in terms of latency between nodes and with IoT devices [26].
- Communication among nodes of the same tier is considered to support a larger number of use cases (e.g., through northbound or dedicated east-west bound interfaces for distributed ones).
- Communication between the different nodes of ASSIST-IoT will be realised through SDN physical or virtual devices (mostly switches) and different access networks (5G, WiFi, TSN, etc.).
- One service is the combination of (at least, one) various enablers. One service may use enablers targeted to different layers (from a Functional View perspective).
- Each enabler has its own “scope” that contains its components, possibly spread out across different places of deployment (devices, Edge Nodes, cloud, etc.).
- The communication between “enabler components” and any other element of ASSIST-IoT architecture will also take place via the enabler interface.
- One “enabler component” might be used by different enablers. The mechanisms for allowing this “share” of components will be a matter of design and discussion (for deliverable – D3.6).

A real topology depends on the vertical industry in which it is expected to be instantiated, since the system topology is strongly influenced by the physical location and the actual requirements of the use cases and scenarios considered. In an ASSIST-IoT hierarchical system, the nodes of each tier have different scopes: (i) Smart IoT Devices, which are differentiated from “dummy” IoT sensors and actuators as they are designed considering enough computing capabilities to assist in decision making and data filtering and processing tasks; (ii) Edge Nodes of the first tier, also referred to as Far-Edge Nodes, which are responsible primarily for data gathering, real time processing and first location of intelligence; (iii) nodes in upper tiers are responsible for the orchestration of services, the coordination of distributed and/or federated operations, and for applying intelligence based on inputs from different nodes of the lower tier (e.g., applications that require inputs from nodes that manage different types of devices); and lastly, in those use cases that require it, (iv) a cloud backend is in charge of deep analysis of the captured data to extract knowledge as well as for long-term storage.

Figure 27 represents an example of the deployment of one service. In this example, one relevant actor is designing a service that combines five enablers distributed among a wide variety of nodes, targeting various functional blocks, and making use of diverse components.

The design, in which enablers are treated as modules, whose internals do not need to be of concern to other enablers, is called *encapsulation*. Any encapsulated enabler should also be secure, which requires that all communication channels between its components are secured, and components are not directly accessed from outside. Encapsulation principle proposed by ASSIST-IoT enables features that are important in the IoT domain, such as modularisation and virtualisation, and aims to deliver more abstract features, such as security by design. It also provides a fruitful ground for separation of scopes for enablers, that aim to deliver functionalities promised by ASSIST-IoT.

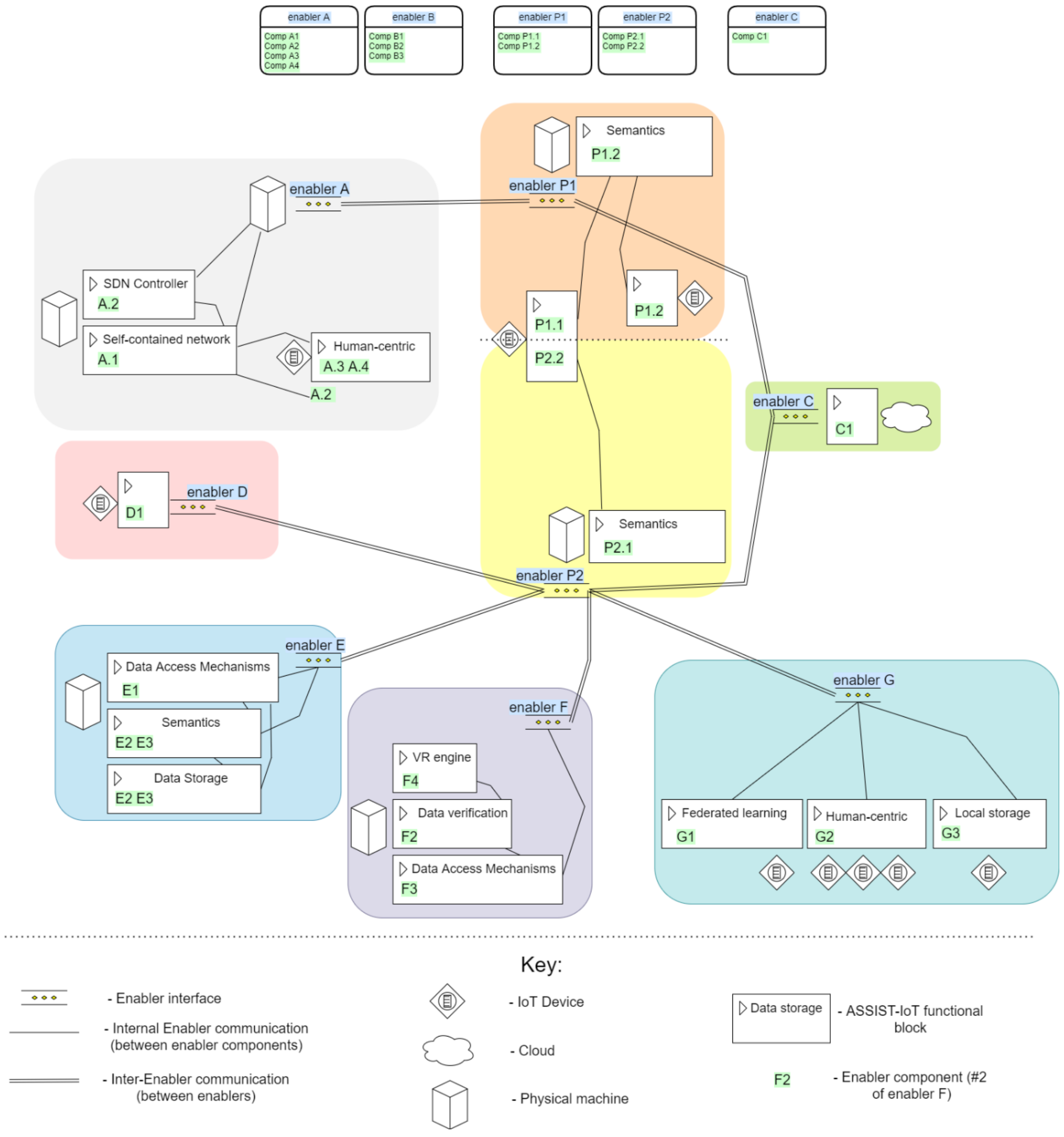


Figure 27. Example of service deployment in ASSIST-IoT

8. Future Work

As advanced in Section 3.5, the architecture of ASSIST-IoT will be delivered via several iterations aligned with different milestones of the project. This document (see Conclusions –9) has provided the starting point that will be used in forthcoming deliveries to build upon.

Although some thorough definitions area already put in place in D3.5 (e.g., architectural decisions such as enablers approach or Views specification), much work is still to be done from different perspectives. This section aims at outlining the following advances expected over the architecture that will be materialised on updated content in deliverables D3.6 and D3.7.

- **Refinement of Views**: The basic Views of ASSIST-IoT architecture have been described in this deliverable (Functional, Node, Deployment). Those descriptions will be enhanced drawing from the research in WP4 and WP5, as well as from the requirements (D3.2 and T3.4 advances). Additionally, T3.5 team envisions potential additions in form of new Views. To mention one, there is the plan of introducing a Data View that will represent the architecture from the viewpoint of end-to-end data flow: from its generation, how it leaves the IoT device, how passes through different nodes and the processing to which is exposed at each step.
- **Enhancement of currently identified enablers**: In D3.5, the key (inherent) enablers per plane have been identified. In addition, some useful (already planned) enablers have been described. However, those explanations have been done from a merely theoretical/intentional point of view. As developments in further tasks advance, the descriptions will be refined, including problems found at their development, additional considerations, among other.
- **Inclusion of new enablers per plane and new enablers associated to verticals**: Extension of the tables presented in the sub-sections of Sections 4 and 5.
- **Settlement of architectural decisions**, clearly indicating the mechanisms and selected technologies that will be guiding the materialisation of ASSIST-IoT architecture in pilots (and future deployments)
    - Deployment of enablers: Specification of the mechanisms (e.g., using service YAML descriptors) for deploying the enablers. Establishment of the tool to “run up” enablers and selection of: K8s, K8s in combination with K3s, Akri, FLEDGE, others…
    - Federated Learning and whole decentralisation approach. Some options seem valid at this point (on how to distribute and orchestrate such intelligence) that are under discussion within ASSIST-IoT technical team.
    - Introduction of global approach to Tactile Internet as structural part of ASSIST-IoT architecture. Although specific tactile applications are a matter of design under the Application and Service plane, there has been observed the need to introduce certain considerations in the architecture for allowing such technologies (that have strong, explicit infrastructure and performance requirements) to run under ASSIST-IoT structure.
    - Transversal introduction of DLT throughout the architecture: Security, Privacy and Trust is a defined vertical in ASSIST-IoT architecture. However, its influence as “operative, practical consideration” reaches beyond the usual scope of delivering enablers. Software Architects of ASSIST-IoT are already in discussions on how the different DLT-related mechanisms should be tackled as structural inclusions (e.g., within each enabler components, as separated enablers, leveraging a somehow centralised facility, relying on replication, etc.).
- Depict a **mapping of requirements** (set out in T3.4) to the different architecture properties, traits and to different enablers (depending on the case).
- Inclusion of additional formal specifications to align the “architecture asset” with pre-normative activities and procedures documentation. These specifications may include (among others) UML-based diagrams of software engineering.

9. Conclusion

This document presents the initial definition of ASSIST-IoT architecture, being the first of a series of three iterations and hence to be further refined during the next versions. The architecture responds to the perspectives and objectives identified for the Next Generation IoT and it is based on the expertise of the technical partners of the project as well as on the initial requirements of the stakeholders involved. Although the main objective of this document is setting the foundations of the ASSIST-IoT architecture, it also aims at providing insights with respect to the technical outcomes to be produced later on in WP4, WP5 and WP6 (what will be actually delivered, and under which principles).

After presenting a brief overview of the main concepts and architecture paradigms available in the market for building systems, this document exposes (i) the main design principles, (ii) the conceptual architecture and (iii) the methodology that guide the definition of the architecture. Conceptually, ASSIST-IoT architecture is a multidimensional architecture which consists of horizontal layers called "Planes", which represent collections of functionalities that can be logically layered on top of one another, and "Verticals", which represent cross-cutting functions and properties of NGIoT that exist on different planes or require coordination among them. Besides, the architecture introduces the abstract term "enabler", which aims at delivering the functions promised by ASSIST-IoT innovations and future capabilities within the different Planes and Verticals. In essence, an enabler is a collection of software (and possibly hardware) components, running on one or different nodes, that work together to deliver a specific functionality of a system.

Dedicated sections for the Verticals and for the Functional View of the Planes have been included, along with the enablers expected to be delivered during the execution of the project. On the one hand, the 5 Verticals of ASSIST-IoT comprise (i) Self-*, (ii) Interoperability, (iii) Security, Privacy and Trust, (iv) Scalability and (v) Manageability. On the other hand, the Planes of ASSIST-IoT are (a) Device and Edge, (b) Smart Network and Control, (c) Data Management, and (d) Application and Services. In order to address additional concerns of stakeholders of a Next Generation IoT architecture, two additional Views have been included, namely "Node" and "Deployment", aiming at identifying those aspects related to the characteristics of nodes and devices, as well as actual implementations of an ASSIST-IoT architecture.

It is important to remind that the goal of this document is to deliver a blueprint, thus presenting a set of characteristics and functionalities that should be present in a Next Generation IoT system, but providing a degree of freedom with respect to technological choices and deployment strategy. Lastly, highlighting that this is an initial definition, meaning that further refinement is expected and hence the presented views, the functionalities and enablers identified will potentially change, both in number and description, in the following versions.

10. References

[1] M. Weyrich and C. Ebert, “Reference architectures for the internet of things,” *IEEE Softw.*, vol. 33, no. 1, pp. 112–116, Jan. 2016.

[2] ISO/IEC/IEEE 42010, “ISO/IEC/IEEE 42010 - Systems and software engineering - Architecture description,” 2011, Online: https://www.iso.org/standard/50508.html.

[3] N. Rozanski and E. Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison Wesley, 2011.

[4] M. W. Maier, D. Emery, and R. Hilliard, “Software architecture: Introducing IEEE standard 1471,” *Computer (Long. Beach. Calif).*, vol. 34, no. 4, pp. 107–109, Apr. 2001.

[5] A. Sharma, M. Kumar, and S. Agarwal, “A Complete Survey on Software Architectural Styles and Patterns,” in *Procedia Computer Science*, 2015, vol. 70, pp. 16–28.

[6] M. Richards, *Software Architecture Patterns*. O’Reilly Media, 2015.

[7] R. A. P. Rajan, “Serverless Architecture - A Revolution in Cloud Computing,” in *2018 10th International Conference on Advanced Computing, ICoAC 2018*, 2018, pp. 88–93.

[8] C. M. Mackenzie, F. Mccabe, P. F. Brown, P. Net, R. Metz, and A. Hamilton, “Reference Model for Service Oriented Architecture 1.0,” 2006, Online: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[9] S. T. Alanazi, N. Abdullah, M. Anbar, and O. A. Al-Wesabi, “Evaluation Approaches of Service Oriented Architecture (SOA) - A Survey,” in *2nd International Conference on Computer Applications and Information Security, ICCAIS 2019*, 2019.

[10] M. Richards, *Microservices vs. Service-Oriented Architecture*. O’Reilly Media, 2016.

[11] J. Prathap Irudayaraj and Mp. Research Scholar, “Comparative study on Cloud Software Architecture: Monolithic, SOA,” *J. Appl. Sci. Comput.*, vol. 6, pp. 2257–2261, May 2019.

[12] S. Newman, *Monolith to Microservices*. O’Reilly Media, 2019.

[13] T. Cerny, M. J. Donahoo, and J. Pechanec, “Disambiguation and comparison of SOA, microservices and self-contained systems,” in *Proceedings of the 2017 Research in Adaptive and Convergent Systems, RACS 2017*, 2017, vol. 2017-Janua, pp. 228–235.

[14] Z. Wu, S. Deng, and J. Wu, “Service-Oriented Architecture and Web Services,” in *Service Computing*, Elsevier, 2015, pp. 17–42.

[15] N. Ford, R. Parsons, and P. Kua, *Building Evolutionary Architectures*. O’Reilly Media, 2017.

[16] E. Berrio-Charry, J. Vergara-Vargas, and H. Umana-Acosta, “A Component-Based Evolution Model for Service-Based Software Architectures,” in *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, 2020, vol. 2020-Octob, pp. 111–115.

[17] J. Ghofrani and D. Lübke, “Challenges of Microservices Architecture: A Survey on the State of the Practice,” in *10th Central European Workshop on Services and their Composition*, 2018.

[18] E. Wolff, *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016.

[19] P. Di Francesco, I. Malavolta, and P. Lago, “Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption,” in *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 2017, pp. 21–30.

[20] M. Richards and N. Ford, *Fundamentals of Software Architecture*. O’Reilly Media, 2020.

[21] S. Hassan, R. Bahsoon, and R. Kazman, “Microservice transition and its granularity problem: A systematic mapping study,” *Softw. - Pract. Exp.*, vol. 50, no. 9, pp. 1651–1681, Sep. 2020.

[22] The European Comission’s science and knowledge service - Joint Research Centre, “Telework in the EU before and after the COVID-19: where we were, where we head to,” 2020, Online: https://ec.europa.eu/jrc/sites/jrcsh/files/jrc120945_policy_brief_-_covid_and_telework_final.pdf.

[23] 5G-PPP Software Network Working Group, “Cloud-Native and Verticals’ services,” 2019, Online: https://5g-ppp.eu/5g-ppp-phase-3-projects.

[24] T. Goethals, F. De Turck, and B. Volckaert, “FLEDGE: Kubernetes Compatible Container Orchestration on Low-Resource Edge Devices,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2020, vol. 11894 LNCS, pp. 174–189.

[25] CREATE-IoT Project, “D6.3. Assessment of convergence and interoperability in LSP platforms,” 2020.

[26] OpenFog Consortium, “OpenFog Reference Architecture for Fog Computing,” 2017, Online:

http://site.ieee.org/denver-com/files/2017/06/OpenFog_Reference_Architecture_2_09_17-FINAL-1.pdf.

[27] ITU-T, "Y.2060: Overview of the Internet of things," 2012, Online: https://www.itu.int/rec/T-REC-Y.2060-201206-I.

[28] AIOTI WG03-loT Standardisation, "High Level Architecture (HLA) Release 4.0," 2018, Online: https://aioti.eu/wp-content/uploads/2018/06/AIOTI-HLA-R4.0.7.1-Final.pdf.

[29] ECC and AII, "Edge Computing Reference Architecture 2.0," 2017, Online: http://en.ecconsortium.net/Uploads/file/20180328/1522232376480704.pdf.

[30] A. Willner and V. Gowtham, "Towards a Reference Architecture Model for Industrial Edge Computing," *Comput. Sci.*, 2020.

[31] M. Bauer *et al.*, "Final architectural reference model for the IoT," 2013, Online: https://www.researchgate.net/publication/272814818_Internet_of_Things_-_Architecture_IoT-A_Deliverable_D15_-_Final_architectural_reference_model_for_the_IoT_v30.

[32] E. Rissanen, "eXtensible Access Control Markup Language (XACML) Version 3.0," Jan. 2013, Online: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[33] ETSI, "GS NFV-MAN 001 Network Functions Virtualisation (NFV); Management and Orchestration," 2014, Online: http://portal.etsi.org/chaircor/ETSI_support.asp.

[34] ETSI, "GS NFV-EVE 005 Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework," 2015, Online: http://portal.etsi.org/tb/status/status.asp.

[35] L. Zhu *et al.*, "SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study," *ACM Comput. Surv.*, vol. 53, no. 6, Feb. 2021.

[36] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, pp. 333–354, Jan. 2018.

[37] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (SD-WAN): architecture, advances and opportunities," in *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 2019, vol. 2019-July.

Appendix A - Glossary

This section contains the summary of important terminology used to describe the ASSIST-IoT architecture.

Table 11. Architectural terms

Term	Description	Examples
Plane	Abstract concept that logically groups system parts working in similar contexts and environments. A plane corresponds to a horizontal layer.	ASSIST-IoT defines 4 Planes: • Application and Services • Data Management • Smart Network and Control • Device and Edge
Transversal (adjective)	Intersecting more than one plane, or involving more than one plane.	
Vertical (noun)	A concept that groups together logically connected features and functionalities of a system, regardless of the plane on which they may be implemented.	ASSIST-IoT defines 5 Verticals: • Manageability • Scalability • Security, Privacy and Trust • Interoperability • Self-* (autonomy)
Vertical capability and feature	Every vertical involves capabilities. A concretisation of a capability is called a feature. Capabilities and features are further subdivisions of the categories defined by Verticals. Although, in principle, a feature is a part of a capability, not every capability is required to be subdivided into features. Capabilities (and features) are abstractions that are independent from Planes. The collections of Planes on which features are delivered depends on a given enabler.	Example capabilities and features: • User management capability (Security, Privacy and Trust vertical) • Authentication and authorisation feature • Self-healing capability (Self-* vertical) • Auto-recovering, fault resistant communication network • Semantic interoperability capability (Interoperability vertical) • Semantic translation • Compliance with standard data model
Functional block	A logical part of a plane. It represents a separable family of functionalities, that fit directly into one plane.	• Semantics (Data Management plane) • Self-contained networking (Smart Network and Control plane)
Enabler	A configurable and deployable collection of software and/or hardware that enables a specific set of features (or functionalities) of an IoT system, and can be interfaced with. An enabler should be separable from the rest of the system, in which it is deployed (including other enablers), but does not need to be independent. It may require other enablers to deliver a promised feature. Enablers can be unmanaged (fully autonomous), semi-autonomous, or fully managed. Enablers must provide a system feature. A computer vision ML algorithm is not an enabler, but it may be a component of a security enabler that employs face recognition.	• Secure storage enabler on the Data Management plane (Security, Privacy and Trust vertical) - provides a data storage interface for the whole system. May require an authorisation and authentication enabler to deliver security. • Self-healing network enabler on the Smart Network and Control plane (Self-* vertical) - employs multiple network components that autonomously try to recover any lost connection, (e.g., find alternative network routes, use different protocols etc). • Resource scaling enabler on the Device and Edge plane (Scalability and Self-* verticals) - enables autonomous facilitation of scalable number of hardware resources to best support workloads without unnecessary resource usage.

	Enablers may require specific software or hardware components to function (e.g., a secure storage enabler may require a database component). Enablers should abstract their operations, so that they may be viewed as coherently delivering a specific functionality, without unnecessary technical details. In ASSIST-IoT the enabled features are grouped into Verticals. An enabler is transversal, if it involves components on more than one plane.	• Resilient communication enabler on the Smart network and Control, and Data Management planes (Security, Self-* verticals) - provides data and communication channel redundancy, to automatically recover from data corruption. • Core ontology semantic translation enabler on the Applications and Services, and Data Management plane (Interoperability vertical) - provides a semantic annotation; semantic repository; and translation services (that use a “core ontology” translation model).
Enabler component	A piece of software or hardware that is a part of an enabler, and is required for an enabler to function and deliver a feature. A component may have a number of descriptive properties (e.g., an instance of an ACID-compliant database) and belong to a functional block (and therefore also to a plane). Enablers are, in principle, not enabler components, but components may be reused between enablers.	• A specific ML algorithm implementation • A specific ML model • An ontology • A LoRa-enabled edge device • An encryption algorithm • An internet camera • A computer vision software delivered as SaaS • A database instance • A user registry

Table 12. Autonomic computing terms

Term	Description	Examples
Self-*	A property of a system that makes it autonomous, i.e., capable of identifying problems or tasks, and solving or carrying them out on its own. Self-* systems may also be semi-autonomous; in which case they require some instructions or supervision. If a system requires supervision of every step, or complete instructions, it is not autonomous. In ASSIST-IoT autonomic systems can exhibit properties in the following categories: self-healing, self-protection, self-awareness, self-organisation, self-synchronisation, self-configuration. Delivery of specific self-* properties may involve multiple self-* categories, or other properties. In particular self-awareness and context-awareness are often helpful or required for other self-* properties.	
Self-healing	Self-healing systems are reliable, highly available and dependable. This is achieved through the ability to monitor and analyse data about self to autonomously predict, detect, prevent and heal faults. It seems to be important to define (non)functional requirements of self-healing solution (where we can apply self-healing, what to do in case of and how to measure the self-healing ability).	• A network tool that uses redundant, parallel channels, caching, and intelligent prediction to ensure recovery of any network faults outside of long-term full connectivity loss (e.g., due to physical damage to all network interfaces). • An autonomous robot capable of returning to upright position (almost) always (e.g., Boston Dynamics “Spot”).

Self-protection	A system is self-protecting, if it continuously (actively or passively) maintains a safety property and defends itself in the presence of potential threats by applying privacy policies, trust mechanisms, and threat detection to secure itself and its data.	• A physical FDR device (flight data recorder) that protects itself from data loss or physical damage. • A system, that uses an ML-powered semantic data parser to redact any sensitive or protected information from all outgoing communication.
Self-awareness	Context-awareness, where the context pertains to the state of the autonomic system itself. It involves the ability of a system to interpret its state by monitoring and analysis based on given domain knowledge.	• A physical robot that has information about its position in space relative to any object in the surrounding area. • A ML training software that is aware of different kinds of models, that it can train, currently trained model, as well as purpose and shape of training, verification and testing data.
Self-organisation	The collective ability of a system to maintain, improve, or restore components in a persistent way under appropriate conditions. It is often characterised by the ability to continuously perceive its own state and the state of its environment (self- and context-awareness) and react to certain events in order to maintain a high degree of usefulness without a human in the loop.	• A drone swarm capable to pick (provision) and position specific machines to carry out a given task (e.g., move a heavy object). • A bit-torrent network that uses throttling and spoiler isolation (shadow ban/ shadow greylisting) to ensure fairness of resource distribution.
Self-configuration	The ability of a system to (re)configure its parameters and resource assignments to maintain the quality of performance metrics based on requirements or to adapt to changing conditions.	• A software container that automatically deploys itself on any system and configures itself to work within it (e.g., a virus). • A music software that applies relevant copyright laws based on location information.

Table 13. General terms

Term	Description	Examples
Artifact	A catch-all term to describe any kind of non-living, tangible or intangible element. Artifacts are usually either parts of a bigger whole, or outputs of some actions.	• An IoT device • An architecture • A diagram • A song
Context-awareness	A property of any system that has access to information about the context, in which it operates. Context-awareness is practically never full. The degree of awareness varies and is usually limited to information required to deliver some functionality without compromising security. E.g., a web server operates in a context of an operating system, but for security reasons is not aware of every process in the system, or system-wide resource usage.	• A physical robot that has information about spatial position of any object in the immediate surrounding area. • A load balancer aware of resource usage on managed nodes (e.g., naive round-robin balancer is not context-aware). • Context awareness is broad term used in FIWARE environments.
Tactile	A property of any system, controller or HMI, which makes its user able to interact with the physical world, either directly by controlling physical actuators, or indirectly, by changing the state of software through the means of a physical controller. The interaction may be two way: from the user to actuators (control), or from the actuators to the user (feedback).	• Touch screen. • Virtual glove with gesture transfer to physical actuators. • Haptic feedback controller.

Tactile internet	A communication network enabling low-latency tactile applications.	• A network enabling real time virtual building tour via wireless VR glasses with head position and orientation support. • Wireless touch controller for vehicle diagnostics in a specialised facility, with real time AR glasses feedback.
Methodology	A system of practices, techniques, procedures and rules used by those who work in a discipline. Applying different principles, themes, frameworks, processes and standards to help provide structure to the way we deliver projects. Methodology can be represented as: documentation (guidelines, best practices), modelled processes, templates. A methodology often defines some design patterns.	• Methodology for requirements gathering. • Methodology for algorithm adaptation to be executed on an edge device. • Steps methodology for delivering the architecture jointly with other assets of the project (e.g. requirements). • Methodology for new enabler development and deployment. • “How to” style.
Design pattern	General reusable recipe for solving a commonly occurring problem.	• Publisher-subscriber communication pattern. • Purely functional programs. • Relational data modelling. • Interoperability patterns.
DevOps	A set of practices that work to automate and integrate the processes between software development and IT teams, so that they can build, test, and release software faster and more reliably.	• Providing CD/CI infrastructure. • Versioning. • Resources management.
DevSecOps	An extension of DevOps, in which security decisions are distributed among participating entities, and made by those, that hold the highest level of context without sacrificing safety. DevSecOps aims to achieve security and privacy by design. To this end security is an aspect at every level of the design and development process and is not delegated or encapsulated as a separate and independent feature.	• DevSecOps pipeline: code review, automatic security testing, vulnerability scanning. • The pipeline has to be linked with DevOps.

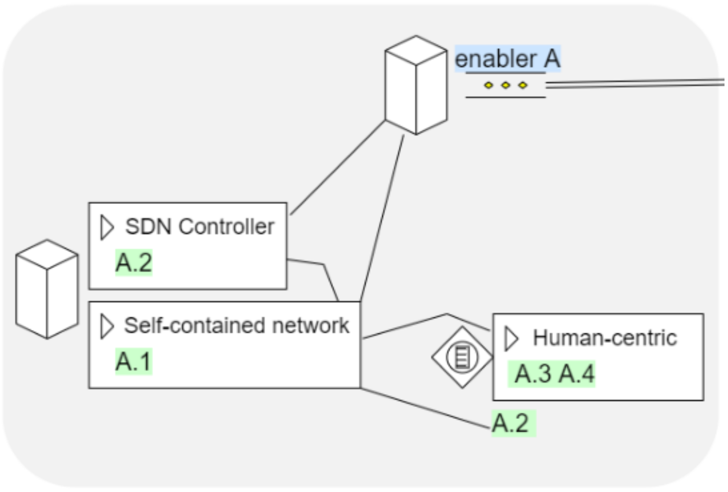
Appendix B - Enabler template

B.1 - General information of the enabler

Enabler	***Name of the enabler (follow glossary guidelines to name it)***
id	*Short unique identifier/acronym*
Owner and support	*Lead and supporting beneficiaries*
Description and main functionalities	*Functional description of the enabler (description paragraph and bullet points for describing its functionalities)*
Plane/s involved	*Horizontal plane or planes in which it acts*
Vertical	*To specify to which vertical it belongs to (mostly for transversal enablers)*
Relation with other enablers	*List of enablers (core or vertical) that interact with this one*
Requirements mapping	*List of the IDs of the requirements addressed or considered*

B.2 - Basic visual diagram

Including arrows of the enabler, with its components, and ids of internal (between its components) and external interfaces.



B.3 - Enabler endpoints

The enabler should have a primary interface for communicating with other enablers or applications (its components communicate through internal communication mechanisms). REST API is assumed for the template, may be others.

Method	**URL**	**Payload (if needed)**	**Description**	**Response format**
GET/POST/ PUT/DELETE	**/{something}/…**			

B.4 - Description of its enabler components

For each component of the enabler:

Enabler component	***Name of the enabler component***
id	*Short unique identifier*
Rationale	*Necessity of the component in the enabler*
Node type/s*	*Physical device in which it can be installed (edge node, smart IoT, gateway, cloud…)*
Implementation technologies	*Technologies to implement it*
HW Requirements	*Memory, storage and computation power needed*
SW Requirements	*Execution environment and/or other project/third-party requirements*

Appendix C - Node template

C.1 - General information of the node

Node	***Name of the node (e.g. Node1TL)***
id	*Short unique identifier/acronym*
Hostname	*Hostname of the node (virtual or phy.)*
Type	*IoT Gateway, Cloud, Edge (selectable)*
Operative System	*Auto-explanatory*
MAC address	*Auto-explanatory*
IP address	*Auto-explanatory*

C.2 - Specific information of the HW, computing resources and network capacities of the Node

Field	**Explanation**	
HW interfaces	*Network and communication interfaces (serial, ethernet, usb)…*	
CPU	*Auto-explanatory*	
GPU	*Auto-explanatory*	
Cores	*Auto-explanatory*	
HW model and year	*(only of application if it is a complete phy HW bare metal being used)*	
Storage capabilities	*Auto-explanatory*	
Acceleration capabilities	*Auto-explanatory*	
Southbound protocols accepted (node to device)	**Access Layer Protocols**	*(e.g. Bluetooth, Serial, WiFi, LoraWAN)*
	Application Layer Protocols	*(e.g. MQTT, HTTP, CoAP)*
East-West protocols accepted (node to node)	*(e.g. RESTful, HTTP/COAP, MQTT/AMQP/RTS…)*	
Can it be further virtualised?	*Yes/No*	

C.3 - Dynamic fields about status of the node

Field	***Explanation***				
System tier	*Spot in the Deployment View of the architecture in which this node currently is.*				
List of components being executed in this node					
Id_component	**Component_name**	**Technology**	**Id_enabler**	**Enabler**	**Created**