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

D3.6 ASSIST-IoT Architecture Definition – Intermediate

Deliverable No.	D3.6	**Due Date**	31-JAN-2022
Type	Report	**Dissemination Level**	Public
Version	1.0	**WP**	WP3
Description	Intermediate specification of the ASSIST-IoT technical architecture and its components.		

Copyright

Authors

Name	Partner	e-mail
Alejandro Fornés	P01 UPV	alforlea@upv.es
Ignacio Lacalle	P01 UPV	iglaub@upv.es
Carlos E. Palau	P01 UPV	cpalau@dcom.upv.es
Eduardo Garro	P02 PRO	egarro@prodevelop.es
Paweł Szmeja	P03 IBSPAN	pawel.szmeja@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
Fotios Konstantinidis	P11 ICCS	fotios.konstantinidis@iccs.gr
Aristeidis Dadoukis	P11 ICCS	aristeidis.dadoukis@iccs.gr
Tina Katika	P11 ICCS	tina.katika@iccs.gr
Óscar López	P13 S21SEC	olopez@s21sec.com
Zbigniew Kopertowski	P15 OPL	Zbigniew.Kopertowski@orange.com

History

Date	Version	Change
11-Nov-2021	0.1	ToC presented
19-Nov-2021	0.2	ToC updated and assignments of sections identified
23-Dic-2021	0.5	First round of contributions integrated
7-January-2022	0.6	Second round of contributions integrated
13-January-2022	0.7	Version to be changed before IR
14-January-2022	0.8	Version sent to IR with some points pending
21-January-2022	0.9	Version with previous points corrected/included.
27-January-2022	1.0	Version modified after Internal Review reports

Key Data

Keywords	Reference architecture, enablers	**Lead Editor**	P02 PRO – Eduardo Garro
IR(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 second 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.

This deliverable reports the works done towards defining the intermediate version of ASSIST-IoT architecture, including guiding principles and key considerations for its deployment. While the initial version sketched the 2D-structure, dividing the different enablers of the project (i.e., the encapsulated innovative functionalities that ASSIST-IoT provides) into planes and verticals, this intermediate report aims at going more into detail with regards to the use of containerisation and the rationale behind the selection of Kubernetes framework as the main orchestrator of the system.

At first, **Section 2 recaps in a brief overview the main concepts** and architecture paradigms defined in D3.5, including (i) the four main design principles, (ii) the 2D-layered conceptual architecture with the horizontal planes and vertical blocks, and (iii) the enablers abstraction concept of the project for the sake of modularity.

Section 3 aims at reinforcing the message that underlies ASSIST-IoT architecture: we have arrived till here thanks to the previous work of others. In particular, this chapter focuses on the liaison with AIOTI's HLA and the design of their architecture and how it has inspired and maps ASSIST-IoT modules, layers and views.

Section 4 refines the Functional View, Node View and Deployment View but also adds one more: the Data view. The relation among them can be simplified to the following: While the Data View is about the changes over the data processed by the system (sensors, apps data, services data, metadata), the Functional View expresses the actual functionalities, and the Node View represents physical scopes where computation takes place. Finally, the Deployment View provides insights about a system instantiation topological, compositional approach.

The main changes identified in the four horizontal planes of the **Functional View** are:

- The scope of **Device and Edge plane** remains largely unchanged with two main pieces: edge elements, so-called GWEN (Gateway/Edge node), and devices. Nevertheless, some remarks have been stated. On the one hand, GWEN has been enhanced by adding SD cards, new built-in 5G and WiFi6 capabilities, as well as enough processing capabilities for AI analysis support. With regards to devices, anchors and tags are being specifically implemented for the project for the sake of guaranteeing indoor localization as well as fall arrest system, especially needed for Pilot 2.
- The eight enablers that compose the **Smart Network and Control horizontal plane** remain largely unchanged, grouped into four blocks, as in the previous iteration of the architecture: orchestration, SDN, network functions and self-contained network realisation. The main change is the identification of essential enablers. While in D3.5, three enablers of this plane were defined as essential, in this new iteration it has been considered that only the Smart Orchestrator and VPN ones will remain, as the presence of SDN-enabled networks will not be mandatory in an ASSIST-IoT deployment.
- The scope of logically separable data-related functions of ASSIST-IoT architecture remains also largely unchanged, with regards to the previous version presented in deliverable D3.5. Just like before, they can be grouped into semantics and data governance. However, it should be noticed that despite the initial security-related data DLT functions mentioned to be part of the **Data Management Plane**, it was decided that, because of the cross-cutting nature of DLT functionality, it should not be advertised as a set of plane functions, but rather be shifted to a different scope beyond Data Management Plane.
- Still, three main working lines with regards to functional services are envisioned in the **Application and Services plane**: client-side applications, novel AR/MR/Computer Vision interfaces, and graphical toolsets via Open APIs for enabling the open experimentation. However, from the initial 4 enablers envisioned in D3.5, the project has led to the development up to 6 enablers. The main additions are the inclusion of a generic Tactile Dashboard, and the split of originally proposed AR/VR/MR enabler into independent ones: Video Augmentation enabler, and MR enabler.

In this second iteration of the architecture, it has been decided to slightly widen the definition of **Node view.** From this point on, the Node is not restricted to be a hardware element but rather a logical element that is hosted within a specific computing hardware. This subtle point emanates from the refinement of the Deployment View, which expects that enablers are spread among k8s clusters, that are composed of nodes. Considering that one single hardware (e.g., a powerful node in high tiers of the topology) can comprise several nodes, it is more precise to refer to "Nodes" in ASSIST-IoT to these logical units rather than to the complete hardware as a single piece. The updated k8s architecture of the software for an ASSIST-IoT Node can be seen in 4.3.

The **Data View** presents a high-level perspective on data collection, processing, and consumption. It is meant to be a viewpoint on the flow of data, considering the specific actions that a set of sequential enablers perform upon them. This view allows to understand the system without technical details.

The **Deployment View** aims at presenting how an ASSIST-IoT architecture is addressed to actors such as system owners, system administrators and/or developers in order to gain insights about hardware elements deployed, enablers in use, and their instantiation. In accordance with the design principles of the architecture, the following considerations apply to any deployment: (i) the communication between ASSIST-IoT nodes must be IP-based, (ii) nodes must have a k8s distribution installed, (iii) connectivity must be secured considering e.g., VPN tunnelling technologies, and (iv) any ASSIST-IoT deployment must have the current 12 essential enablers catalogued in Section 6 of this document.

Section 5 provides an update of design principles and global guidelines, with special interest over four key points:

1. Some enablers' conventions have been placed with regards to (i) minimum endpoints to be incorporated (four - /health, /api-export, /version, /metrics), (ii) automated monitoring system (PUD) based on Prometheus k8s platform should be supported aiming at collecting data and metrics about the performance and usage of the resources, and (iii) all enablers must contain a logging agent that will collect logs from the components of the enabler and will forward them to the LTSE and potentially to the DLT logging enabler if they fall under the critical logs policies.
2. Encapsulation exceptions over potential developments that are part of ASSIST-IoT (and follow its architecture) but are not encapsulated rather integrated into the project in other ways are highlighted.
3. In order to guarantee a secure development management process, the DevSecOps methodology is reinforced. The underlying aim of this approach is to meet the expected levels of security helping identify, prevent and correct problems that may arise during software development, integration, and deployment.
4. Additionally, DLT and FL design choices are also presented, as they were not included in previous iteration of the architecture. While the DLT has been agreed to be a cornerstone with respect to security, privacy and trust pillars of the project, the FL system has been agreed to be addressed more like a research-oriented platform, so that any ASSIST-IoT platform will rely on the FL enablers for working appropriately in the different operative industrial environments of the project.

Section 6 provides a list of the enablers that have been catalogued as "essential" (Smart Orchestrator, VPN enabler, Edge data broker, LTSE, Tactile dashboard, OpenAPI manager, Identity Manager, Authorisation enabler, DLT logging and auditing enabler, and Enablers manager). This list includes the pieces of the architecture that **are vital to be present in any deployment** while others might be considered complementary (12 out of 41 enablers have been considered essential). This chapter reflects which of them have this consideration and why.

Finally, **section 7 sketches the diverse ASSIST-IoT elements that will be demonstrated and validated in the different pilots of the project**, including the essential enablers as well as diverse specific vertical and core enablers required.

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
AI	Artificial Intelligence
AIOTI	Alliance for the Internet of Things Innovation
API	Application Programming Interface
AR/VR/MR	Augmented/Virtual/Mixed Reality
CA	Certificate Authority
CAD	Computer-Aided Design
CNF	Container Network Function
CPU	Container Network Functions
DApp	Decentralised Application
DLT	Distributed Ledger Technology
ECC	Edge-to-Cloud Computing Continuum

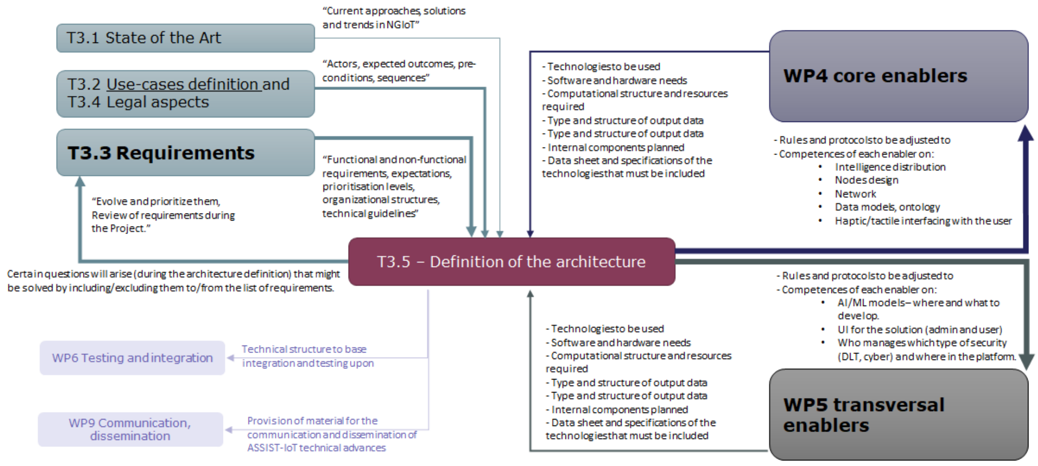
EDBE	Edge Data Broker Enabler
EPC/5GC	Evolved Packet Core /5G Core
FL	Federated Learning
GPU	Graphics Processing Unit
GUI	Graphical User Interface
GWEN	Gateway/Edge Node
HLA	Graphical User Interface
HLF	Hyperledger Fabric
HTTP	HyperText Transfer Protocol
HW	Hardware
IMU	Inertial Measurement Unit
IoT	Internet of Things
JSON	JavaScript Object Notation
K8s	Kubernetes
KPI	Key Performance Indicator
LPWAN	Low Power Wide Area Networks
LTSE	Long-Term Storage Enabler
MANO	Management & Orchestration
ML	Machine Learning
MVP	Minimum Viable Product
NFV	Network Function Virtualisation
NGIoT	Next Generation IoT
NoSQL	Not only SQL
OS	Operative System
PAP	Policy Administration Point
PDP	Policy Decision Point
PIP	Policy Information Point
PUD	Performance and Usage Diagnosis
RA	Reference Architecture
RAM	Random Access Memory
REST	Representative State Transfer
SDN	Software Defined Networks
SDO	Standards Developing Organisation
SD-WAN	Software-Defined Wide Area Network
SOM	System-On-Module
SPA	Simple Page Application

SQL	Structured Query Language
TCP	Transmission Control Protocol
TSN	Time Sensitive Networking
UML	Unified Modelling Language
UWB	Ultra Wide Band
VIM	Unified Modelling Language
VNF	Virtualised Network Function
VPN	Virtualised Network Function
WG[x]	Working Group [number x]
WP	Work Package
XACML	eXtensible Access Control Markup Language
YAML	Yaml Ain't Markup Language

1. About this document

The main objective of this document is **to iterate over the initial specification of 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, D3.6 aims at enhancing the content depicted by D3.5 (delivered in M6) taking advantage of the research conducted in task T3.5 and globally the findings from all technical actions with regards to the architecture during the period M6-M15. At this point, most enablers are being developed (at different speeds), some of them even being delivered as MVP in the first iteration. This has allowed the "architecting team" to realise certain adjustments to be applied to the architecture (e.g., exceptions to the enablers encapsulation principle) as well as to gather more information to complement the initial specification. This deliverable will set the ground for the final delivery of the architecture (D3.7) by M21.

1.1. Deliverable context

Keywords	**Lead Editor**
Objectives	**O1**: D3.6 is the intermediate definition of the architecture, outcome of objective 1.
Work plan	 "Current approaches, solutions and trends in NGIoT" T3.1 State of the Art "Actors, expected outcomes, pre-conditions, sequences" T3.2 Use-cases definition and T3.4 Legal aspects 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
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 in M21.
Deliverables	This deliverable is fed by the elaboration of D3.5 (Architecture definition – Initial) and will serve as the basis for the forthcoming final update of the architecture in D3.7. It will also be feeding into the technical developments to be produced on D4.2 and D5.3.

1.2. The rationale behind the structure

As aforementioned, deliverable D3.6 aims at updating the description of the ASSIST-IoT architecture. This deliverable will be updated in its final version D3.7, that will redound in a formal Reference Architecture (RA) definition document. Therefore, a similar structure as its predecessor has been followed, aligned to an extent with usual formal description of a RA, including, however, specific sections aiming at (i) enhancing previously outlined information, (ii) clarifying certain relevant points/decisions/principles and (iii) adding new information.

It is worth mentioning that the "minimisation principle" has been followed. No verbatim content is replicated from D3.5, only genuine, original (delta from the previous text) has been incorporated to this document.

Section 2 starts by providing a context review of the initial version of the architecture of the project in D3.5, including the principles adopted and the essential cornerstone elements, namely enablers. **Section 3** elaborates (drawing from D3.5) the relation of ASSIST-IoT RA with the trends and recommendations from AIOTI. **In**

Section 4, the Views of the architecture are enhanced with new information to enrich their meaning and description, including the addition of a new View (Data) and certain clarifications. **Section 5** serves as an update for the design principles and deployment guidelines of the architecture, introducing new content related to enablers communication capabilities or relevant encapsulation exceptions. **Section 6** offers the first approach to the list of essential enablers to any ASSIST-IoT deployment, while **Section 7** maps how every enabler will be validated and demonstrated during the project according to their fit into pilots. A Conclusion is included in **Section 8**.

In addition, an Appendix is included containing visual and naming instructions for data pipelines making.

2. ASSIST-IoT Architecture Context Review

This section aims at summarising the principles, concepts and enablers previously defined in D3.5[1].

2.1. Principles

The design principles that are governing the definition of the ASSIST-IoT architecture are

(i) ***The split of the different project developments into microservices:*** It will provide a firm module boundary, allowing for different services being written in different programming languages, being autonomous, independent, and self-contained.

(ii) ***The instantiation of the aforementioned microservices into containers***. It will allow developers to create lightweight ASSIST-IoT microservices decoupled 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

(iii) ***The introduction of "enablers",*** which will consist of a group of microservices, each of them served over a container, and acting towards providing a specific functionality in the architecture. Each enabler provides a single point of entry to communicate with, thus encapsulating the developed microservices.

(iv) ***: The orchestration of ASSIST-IoT enablers using the small K8s footprint.*** It will allow to automate rollouts and rollbacks, to monitor the health of software microservices to prevent bad failures and restarting accordingly, and to scale services up or down based on utilisation, ensuring they are only running what the owner needs.

2.2. ASSIST-IoT Conceptual Architecture

The ASSIST-IoT conceptual architecture is rooted in a multidimensional approach, in which Horizontal planes are intersected by Vertical blocks, allowing for a high level of modularity. The Horizontal lanes represent collections of functions that can be logically layered on top of one another. Verticals, on the other hand, represent functions targeting NGIoT properties. A high-level description of ASSIST-IoT Functional architecture was presented in D3.5, also shown in Figure 1.

The ASSIST-IoT Horizontal planes and Verticals are briefly summarised next:

<u>**Horizontal planes**</u>

- **Device and Edge Plane** describes a 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.
- **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

[1] https://assist-iot.eu/wp-content/uploads/2021/12/ASSIST-IoT_D3.5_Architecture_Definition_Initial_v1.0.pdf

virtualised networks (e.g., SD-WAN, NFV, MANO). The plane follows an access-network-agnostic approach, in which the network connections are highly flexible.

- **Data Management Plane** handles all functions related to a virtual shared data ecosystem, including data interoperability, provenance, fusion and aggregation, as well as semantics empowerment.
- **Application and Services Plane** delivers a layer of abstraction that manages functions offered by lower planes, allowing the creation of advanced and intelligent applications.

<u>**Verticals**</u>

- **Self-*** in order to provide a NGIoT system that is autonomous or semi-autonomous alongside different dimensions. In particular, ASSIST-IoT self-* capabilities involve self-diagnosis and self-healing for autonomously fixing faulty elements, and self-configuration and self-provisioning for autonomously provisioning resources before actual increased demand, based on statistical predictions.
- **Interoperability** by providing a semantic data governance toolset.
- **Security, Privacy and Trust** vertical aims at providing the following functionalities along the ASSIST-IoT architecture: authorised registration of devices, trusted access for sharing data on multiple domains, secure and private storage, and access control mechanisms for responding to potential cyberthreats.
- **Scalability** vertical is a property of the system that is present due to the aforementioned design principles, namely scalable hardware (to allow the modification of the configuration of internal elements such as CPUs, GPUs, RAMs, storage, or network interfaces), scalable software (to enable the efficient deployment and configuration of tens/hundreds of smart applications on computing nodes), and scalable communication capabilities (to permit the modification of network elements, based on different metrics such as performance, reliability, security, etc.).
- **Manageability** for managing nodes and configuration options over any enabler running in a particular deployment of the ASSIST-IoT architecture, covering among others orchestration between enablers, management of enablers' outputs, end-to-end workflows, etc.

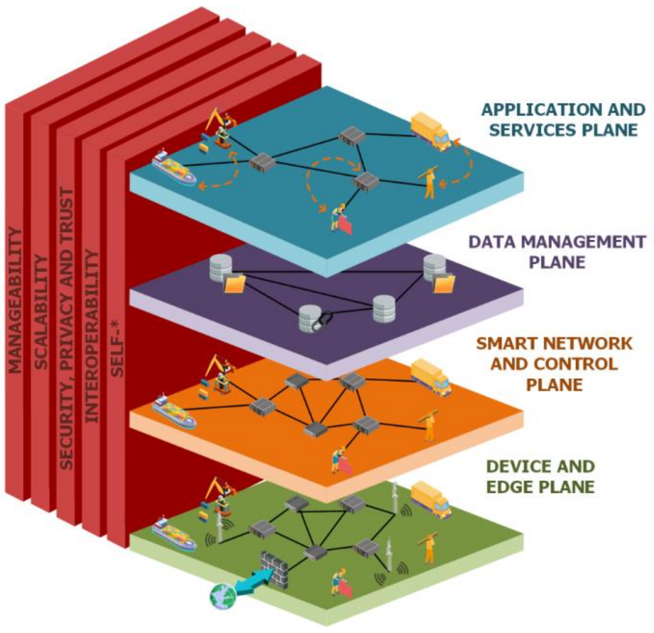

MANAGEABILITY
SCALABILITY
SECURITY, PRIVACY AND TRUST
INTEROPERABILITY
SELF-*
APPLICATION AND SERVICES PLANE
DATA MANAGEMENT PLANE
SMART NETWORK AND CONTROL PLANE
DEVICE AND EDGE PLANE

Figure 1. ASSIST-IoT Conceptual Architecture

For more information about the ASSIST-IoT Horizontal planes and Verticals, please refer to deliverable D3.5.

Since it is not possible (nor recommended) to capture all the features in a single, all-encompassing model, the ASSIST-IoT architecture resulted in not only a Functional View, but also others: Deployment View and Node View. Their previous description and their refinement since the first ASSIST-IoT architecture presented in D3.5 are detailed in Section 4.

2.3. Enablers

Enablers are the cornerstone of ASSIST-IoT architecture. The introduction of the "enablers abstraction" responds to the realisation of a modular architecture, facilitating the delivery not only of the functions promised by ASSIST-IoT, but also the addition of 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, this is, ASSIST-IoT enablers are not atomic but presented as a set of interconnected components. However, not all enablers are mandatory to be deployed to be compliant with ASSIST-IoT architecture: some may be optional, instantiated only when needed, while others are critical for setting the basis of an ASSIST-IoT deployment.

In another vein, it should be noticed that multiple enablers may be used in a system to deliver a more complex *service* (see concept in Section 4.4), leveraging features of the involved enablers. 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. Following the previous description, an enabler diagram is shown in Figure 2.

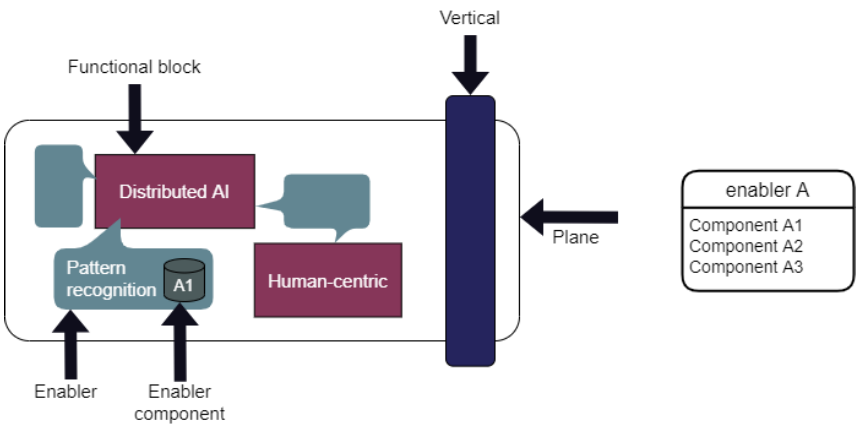


Figure 2. ASSIST-IoT enabler diagram

3. Relevant role of AIOTI's HLA

During the period M6-M15, the analysis of different organisations and standardisation initiatives has become more and more relevant up to making it the baseline to refine ASSIST-IoT's detailed solution specification. Close cooperation with AIOTI is allowing ASSIST-IoT's architecture designers to be in line with current standardisation works, giving the team the chance to contribute to the evolution of the architecture in the standardisation domain.

The Alliance for Internet of Things Innovation (AIOTI) was initiated by the European Commission in 2015, with the aim to strengthen interaction among Internet of Things (IoT) players in Europe and to speed up the creation of a dynamic European IoT ecosystem. The AIOTI activities focus on well-defined areas of development:

- Horizontal: research, innovation eco-systems, standardisation, testbeds, policies & strategies, urban society, DLT and digital for green,
- Vertical/cross-disciplinary IoT subjects: (e.g.,) agriculture, buildings, energy, health, manufacturing and mobility & logistics.

The activities of AIOTI are amongst the most relevant for ASSIST-IoT in terms of standardisation. In specific, out of the broad scope of subjects that the Alliance develops, their works on the creation of a **High Level Architecture (HLA)** stand out. In this activity, AIOTI focuses on the specification and consolidation of architectural frameworks, reference, architectures, and architectural styles in the IoT & Edge Computing space.

In ASSIST-IoT, partners OPL and UPV are the members of AIOTI with interest especially in this standardisation work.

With regards to links with the project's architecture, cooperation with AIOTI brings the possibility to build a new concept of IoT architecture based on knowledge provided by the Alliance and make new contributions to standardisation work and AIOTI reports. One of the most relevant documents is the HLA Release 5.0 in WG3 Standardisation group [1]. In this report, some relevant points related to the project highlight:

Domain model:

Whereas AIOTI's approach is focused on the concept of "*a thing*" (classical term in IoT), ASSIST-IoT (due to its new-born nature) takes the "thing" for granted and aims at innovating in upper levels of the stack (network, data, semantic interoperability, etc.).

Functional model:

Figure 3 depicts the functional model of AIOTI's HLA, consisting mainly of three *layers*, namely Application, IoT and Network. It is based on ISO/IEC/IEEE 42010 and has evolved along releases, addressing concerns such as privacy, virtualisation, Big Data and AI in IoT, and systems' autonomy in the components of the layers rather than as addition of them. Security and management are also considered, but intrinsic to interface specifications instead of represented as cross-cutting layers.

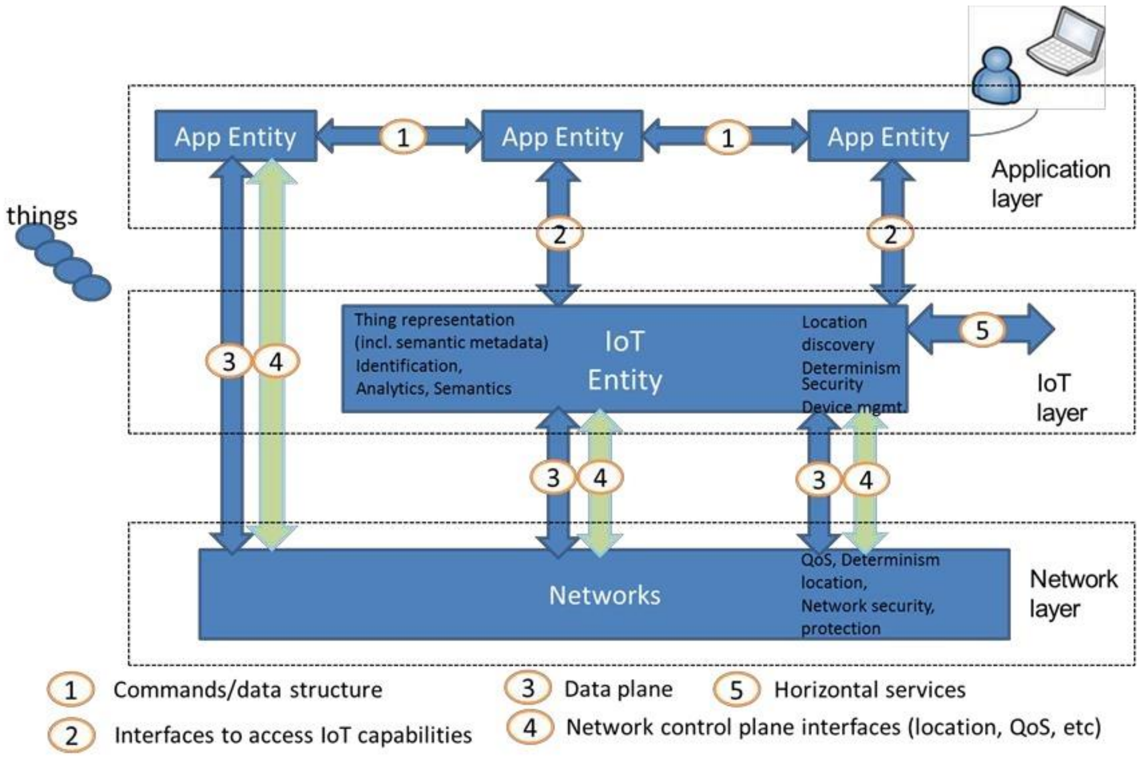

App Entity
1
App Entity
1
App Entity
Application layer
things
2
2
Thing representation (incl. semantic metadata) Identification, Analytics, Semantics
IoT Entity
Location discovery Determinism Security Device mgmt
5
3
4
IoT layer
3
4
3
4
Networks
QoS, Determinism location, Network security, protection
Network layer
1 Commands/data structure
2 Interfaces to access IoT capabilities
3 Data plane
5 Horizontal services
4 Network control plane interfaces (location, QoS, etc)

Figure 3. High Level Architecture (Source: AIOTI report)

The baseline of the architecture relies on the following:

- **App Entity** – implementation of IoT application logic, which can reside in devices, gateways and servers.
- **IoT Entity** – exposure of IoT functions to App Entities or to other IoT entities; communication to Networks layer using data and control plane interfaces.
- **Networks** – realised by different network technologies. Typically, interconnections are IP-based.

A first commonality found between the two approaches is that, although describing a layered structuration, it must not be understood as a "protocol stack" or topological mandate. Both distil from a **software perspective**. That way, any of the elements in "layers" can live in a same device (or not). This is a **major influence** that AIOTI has had in the design of ASSIST-IoT architecture.

Another relevant point that has **hugely emanated from AIOTI's HLA** is the "interfacing". AIOTI established a solid, sound mechanism of interfacing between their *layers*, consisting of different interfaces with different purposes (e.g., network control, data plane, commands…) that vary depending on which layer are peer-to-peer being communicated. In ASSIST-IoT, the question of crossed interfaces among equivalent elements has been followed, establishing conventions to interface enablers with other enablers (see Section 5.1).

In addition, the concept of **allencompassing security and management** has also been leveraged in ASSIST-IoT project. Interfaces must intrinsically entail security (tackled in ASSIST-IoT via Security Enablers and protection of APIs), supporting Authentication and Authorisation at "every-hop" level. This is followed in ASSIST-IoT as it can be seen in Section 0. Management, southbound protocols, digital rights, data governance principles and data lifecycle (pipelines in ASSIST-IoT see 4.5 and Appendix A -) are also considered in the project.

However, ASSIST-IoT decided to take a slightly different path in terms of naming and concept with regards to **functionalities**. AIOTI conceives the intermediate layer (IoT entity – the closest to a physical entity) as the so-called "usually exposed IoT functions" (data storage, discovery, device management, location, etc.), whereas it creates an upper layer for those applications more focused on the business deployment (IoT) logic. ASSIST-IoT has decided to determine that "any functionality" - be it "usual" or "business-oriented" - is represented (and materialised) **as an enabler**.

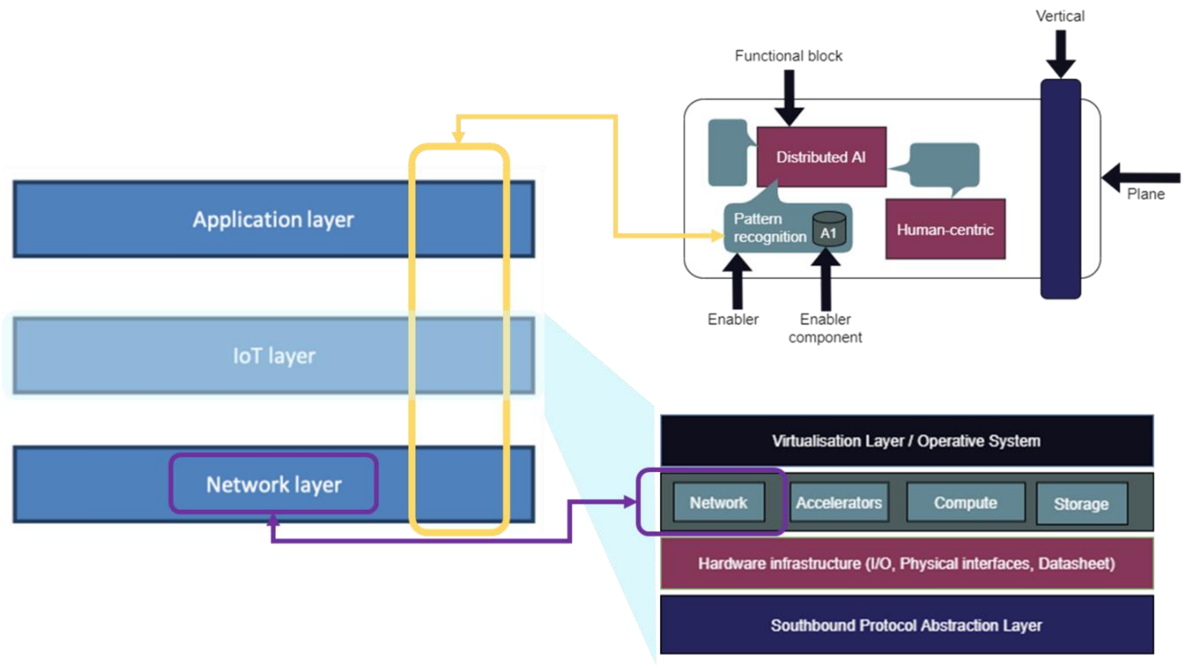


Figure 4. AIOTI HLA comparison with ASSIST-IoT approach – Conceptual model

This special level of abstraction rotates the design towards a more "cloud-native-ish" structure of the architecture, assuming that (AIOTI's) Network, App Entity and IoT Entity layers can be encapsulated as "services" able to be executed in any spot through the edge-cloud-computing continuum. While AIOTI puts its focus on IoT capabilities, ASSIST-IoT adopts ideas from Cloud Computing trends to hybridise IoT modern deployments.

In addition, ASSIST-IoT has decided to separate specific domains (or functionalities) into Horizontal planes (not the same *layer* concept as in AIOTI) and in Vertical capabilities.

With regards to the Horizontal planes, the following can be argued:

- Application plane encompasses a set of functionalities (in form of enablers) that would be represented as App Entities in AIOTI's approach. However, App Entities would also implement functions that are spanned through Horizontal planes in ASSIST-IoT (e.g., MR equipment works).
- Data Management plane would represent (encapsulated as enablers) a sub-set of the functionalities (those related to data transformation and analytics) in the IoT Entity of AIOTI's architecture.

- Smart Network and Control plane would deal with the management of Network layer features.
- Device and Edge plane – A set of the "intrinsic" characteristics in AIOTI's IoT entity are embedded in this ASSIST-IoT's plane, being the closest expression of the built-in traits of physical elements.

With regards to the Verticals, 5 "columns" are proposed in ASSIST-IoT, where specified transversal functionalities, capabilities and properties require the introduction of NGIoT features. This architecture extension creates another level of specialisation over the HLA architecture, with more detailed specification of capabilities devoted to specific "technological domains" (e.g., management, privacy, self-*…) and higher level of modularity.

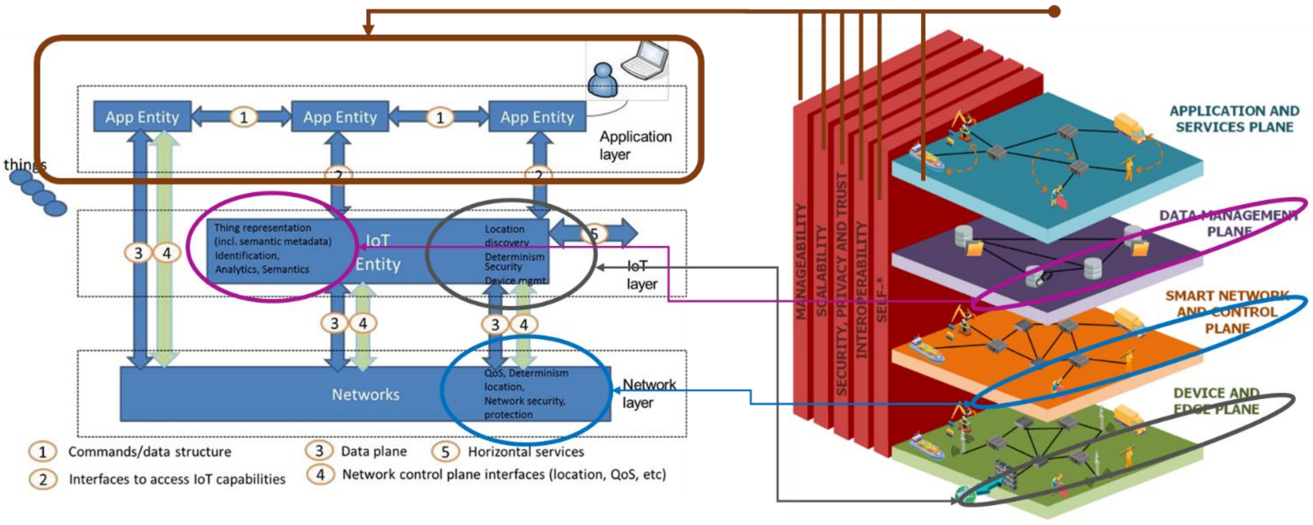

App Entity
App Entity
App Entity
Application layer
things
Thing representation (incl. semantic metadata) Identification, Analytics, Semantics
IoT Entity
Location discovery Determinism Security Device mgmt
IoT layer
Networks
QoS, Determinism location, Network security, protection
Network layer
1 Commands/data structure
2 Interfaces to access IoT capabilities
3 Data plane
4 Network control plane interfaces (location, QoS, etc)
5 Horizontal services
MANAGEABILITY
SCALABILITY
SECURITY, PRIVACY AND TRUST
INTEROPERABILITY
SELF-*
APPLICATION AND SERVICES PLANE
DATA MANAGEMENT PLANE
SMART NETWORK AND CONTROL PLANE
DEVICE AND EDGE PLANE

Figure 5. AIOTI HLA comparison with ASSIST-IoT approach –Functional model

Figure 5 aims at representing the exposed relations between the two functional model approaches. It is worth mentioning that ASSIST-IoT has drawn from high level assumptions posed by AIOTI, trying to satisfy its own requirements (those of NGIoT) and prepared to be validated in the project's own pilots.

AIOTI WG3 has included (in its latest release) a description of the comparison with other relevant IoT architectures in the state of the art (e.g., oneM2M, AUTOPILOT, RAMI4.0, 3D-layered from CREATE-IoT…). ASSIST-IoT has also been inspired (see deliverable D3.5 and deliverable D3.1) by those initiatives, with which architectural comparisons have also been made in the first months of the project.

It is in ASSIST-IoT's intentions **to push forward the inclusion of the comparison in Figure 5** as a part of the **Release 6** of AIOTI. The different partners of the project that are members of AIOTI are already working in this line within the corresponding working group.

Deployment considerations:

AIOTI is also serving as deployment model example. Although AIOTI prefers not to be sticking to specific materialisation technologies, its HLA defends the adoption of strategies that ASSIST-IoT has aimed to mirror:

- **Microservices-based**: As per words of HLA document: "*The possibility to define architectural layers and group them in a functional architecture for IoT virtualization may allow for the most effective selection and combination of microservices-based components*". In this regard, ASSIST-IoT follows those principles to the letter, devising a decentralised architecture that relies on the execution of combined microservices (services in ASSIST-IoT are a composition of various enablers interacting with each other). However, AIOTI warns that such an approach "*goes with the risk of a lack of structure of the resulting implementation*". Thus, it is key to effectively select and combine the elements under a clear structure to guarantee consistency. To solve this, AIOTI makes a theoretical proposal (see in the left image below) that ASSIST-IoT translates to the field (see in the right image below).

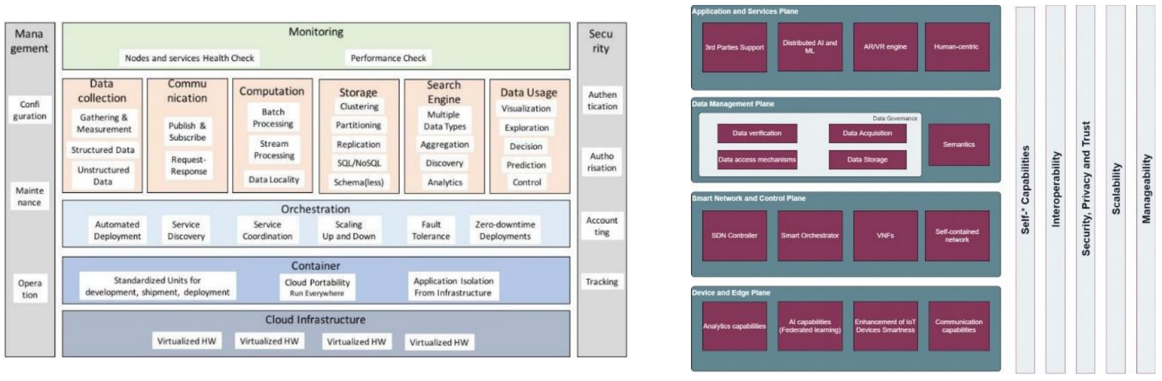


Figure 6. Microservices-based structure approach in (left) AIOTI and (right) ASSIST-IoT

- **ECC**: While AIOTI provides the general concept of cloud and edge computing solutions, ASSIST-IoT proposes a more fluid perspective, allowing enablers to be deployed (via the Smart Orchestrator) throughout various logical and topological spots located somewhere in the ECC.
- **Network virtualisation**: AIOTI presents the architectural framework of NFV ISG as a potentially valid IoT virtualization framework as long as VNF are replaced for IoT Virtualised functions. To achieve this, ASSIST-IoT has leveraged the NFV ISG orchestrator concept (using OSM) and to implement that replacement via CNF (Container Network Functions) to actually instantiate IoT network features.
- **Security** and **Privacy**: ASSIST-IoT's security enablers (4 have been defined) deal with availability, authentication, authorisation, cyberthreats/cybersecurity analysis and global encryption of messages. The previous complies with the relevant metaphor of the "packaged gift" exposed in AIOTI's document. On the other hand, confidentiality and accountability (key aspects in IoT systems) are handled by DLT enablers in the architecture proposed in the project. Besides, privacy and protection of personal data is a matter of special emphasis in ASSIST-IoT that is generating an "artifact" within task T4.3 for all enablers to comply with. ASSIST-IoT's advances will be submitted for consideration for the next release of AIOTI's HLA, that has already promised to work in this direction.

Finally, it is worth mentioning that ASSIST-IoT is intensively following the list of gaps in IoT standardisation identified by AIOTI, conducting a specific task (T9.3) in the workplan aimed at tackling these aspects.

4. Views refinement

4.1. ASSIST-IoT architecture views – practical application

According to D3.5, ASSIST-IoT (following the indications of ISO/IEC/IEEE 42010) defines its **Views** as a work product to express its architecture from the perspective of specific system concerns, illustrating how the architecture addresses them. Subsequently, it declares a "concern" as a topic of interest to one or more stakeholders belonging to an architecture. As a conclusion, a View of the ASSIST-IoT architecture aims at representing an observation prism of the whole specification fit to the wills of a determined stakeholder group.

Considering the 2D-layered approach of ASSIST-IoT, D3.5 specified the usual Views (Logical/Functional and Deployment) and included the Node view as a specificity of the project. In this intermediate version of the architecture the (also usual) Data (or Information) View has been included.

Whereas all the previous seek to align with the guidelines of the building of a reference architecture according to ISO/IEC/IEEE 42010, its interpretation in pragmatic terms might seem a bit loose. Note that this specification is the one recommended by AIOTI to capture relevant views and models in IoT architectures [1].

The goal of this page **is to illustrate how those Views** (and what they represent) **have an actual impact in the creation of the architecture and also in its usage**.

Borrowing a sentence from TOGAF documentation[2], "*just as a building architect might create wiring diagrams, floor plans, and elevations to describe different facets of a building to its different stakeholders (electricians, owners, planning officials), so an IT architect might create* ***Views*** *of an IT system for the stakeholders who have*

[2] https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap31.html

concerns related to these aspects". The different Views described in D3.5 and here compose, altogether, the whole scope of the architecture while, separately, are useful to understand a concern/domain of use.

In the next table, a summary of the stakeholder-concern-View pointing is provided:

Table 1. Practical application of architecture Views- a relations summary

View of the Architecture	Stakeholder to whom is useful	Concern of interest & msg. transmitted
Functional/Logical View	Developers and maintainers	Enablers to be developed and how are they catalogued/framed.
	Acquirers/users	Understand the functionalities (core and vertical) which the arch. is composed of.
Node View	HW developers	Create/enhance a hardware to deliver a suitable ASSIST-IoT edge node.
	Edge device/gateways providers	Select a hardware to deliver a suitable ASSIST-IoT edge/node.
Deployment View	System administrators	Put in place network topology and k8s' distributions properly.
	Developers and maintainers	Understand relation among enablers facilitating the development of APIs and interfaces.
Data View	Data engineers	Understand data transformations to be happening within the architecture and plan their enablers accordingly.
	Developers and maintainers	

To further illustrate the practical meaning of each View, below some examples (applied to real uses of the RA in ASSIST-IoT) are outlined:

<u>Practical examples of Functional view:</u>

One team must develop an enabler to allow multi-link network connections among k8s (or similar) clusters in an ASSIST-IoT deployment. The team will need to know: (i) the enablers related to network management that are being put in place as per the architecture (e.g., orchestrator), (ii) the description of the enablers of the same layer and their interfaces (in order to properly interact with them), (iii) the inter-layer connectors to allow enablers from other layers to interact. In addition, this View is also the tangible space where the enabler must be properly informed towards the architecture in global (components used, technologies, API description, etc.). An acquirer of the solution will also be able to observe, through the Functional view, the layers (and the enablers per layer) that the architecture is composed of, in order to analyse whether all their necessities are covered or to compare it to other RAs.

<u>Practical example of Node view:</u>

A company is interested to manufacture a novel edge device that may become an element of an ASSIST-IoT deployment. The Node view will allow the company to know: (i) the minimum performance and characteristics to be included, (ii) the software layers that must be considered, (iii) the physical interfaces to be equipped, (iv) the mandatory elements that must run upon it (e.g., k8s or similar distribution), (v) the operating systems that it can/must support, thus the processor architecture to be selected, (vi) protocols that must be supported natively in the edge device/gateway and (vi) which built-in processes (like monitoring, healing, etc.) need to be embedded. All the previous information is dismissed in all Views but in the Node view, whose essential goal is to provide these kinds of information. Knowing the previous, the company will be able to purchase/manufacture hardware based on ASSIST-IoT developments accordingly, or a non-HW specialised entity could select the most suitable device out of a catalogue.

Practical example of Deployment view:

The System administrator of a company willing to deploy ASSIST-IoT (i.e., in the IT department of a stakeholder) wishes to know where to install a specific enabler within the network topology of the company. They may also wish to, for instance, (i) realise how many -and which- enablers are necessary to provide a specific service and ensure that enough nodes are put in place, (ii) analyse whether new edge equipment must be purchased to support the vertical functionalities, (iii) identify where AI inference is taking place in the topology, (iv) manually balance the distribution of the enablers among the available nodes, (v) identify which services should be activated (ports open) in each machine/domain of the network. The necessary information to take those decisions will be provided by the Deployment view.

Practical example of Data view:

As the engineer that is going to develop an AI algorithm (a set of scripts that will be encapsulated as one enabler) to predict the position of cranes (or any element) in the yard (or any space), they would wish to know the details of the data collection in the ASSIST-IoT deployment as well as where, why and how any data transformation (semantic translation, data model harmonisation, etc.) is taking place along the information path to reach that specific enabler.

4.2. Functional View

As per deliverable D3.5, the Functional 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.

Figure 7 shows a summary of the Functional View, containing the different Planes and the enablers that are included in every of them.

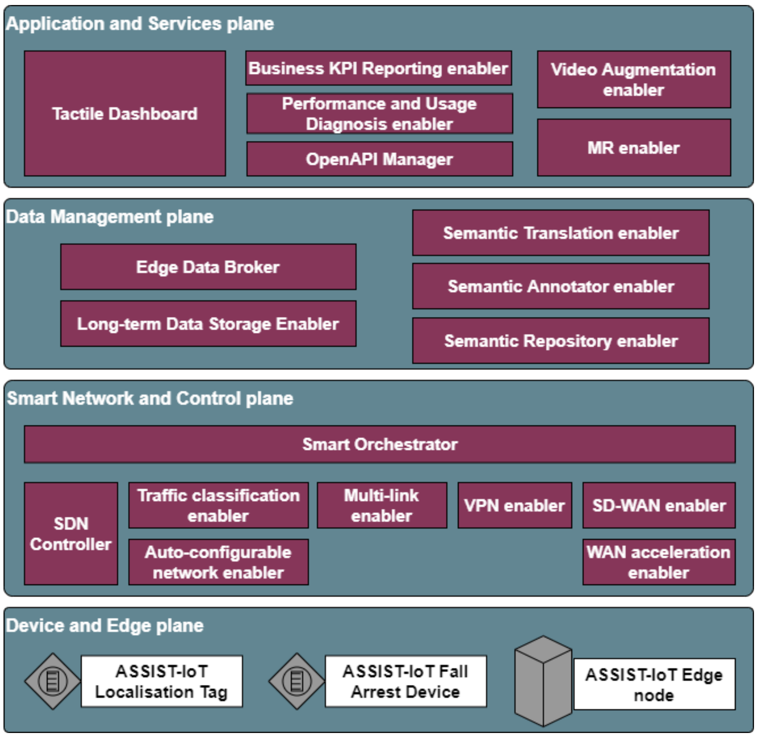

Application and Services plane
Tactile Dashboard
Business KPI Reporting enabler
Performance and Usage Diagnosis enabler
OpenAPI Manager
Video Augmentation enabler
MR enabler
Data Management plane
Edge Data Broker
Long-term Data Storage Enabler
Semantic Translation enabler
Semantic Annotator enabler
Semantic Repository enabler
Smart Network and Control plane
Smart Orchestrator
SDN Controller
Traffic classification enabler
Multi-link enabler
VPN enabler
SD-WAN enabler
Auto-configurable network enabler
WAN acceleration enabler
Device and Edge plane
ASSIST-IoT Localisation Tag
ASSIST-IoT Fall Arrest Device
ASSIST-IoT Edge node

Figure 7. Functional View summary representation

On the other hand, and although not being part of the Functional View, vertical capabilities address all-encompassing concerns, properties and transversal functionalities such as Security, Interoperability or Manageability among others, and are worth to be reminded here to contextualise the rest of the document:

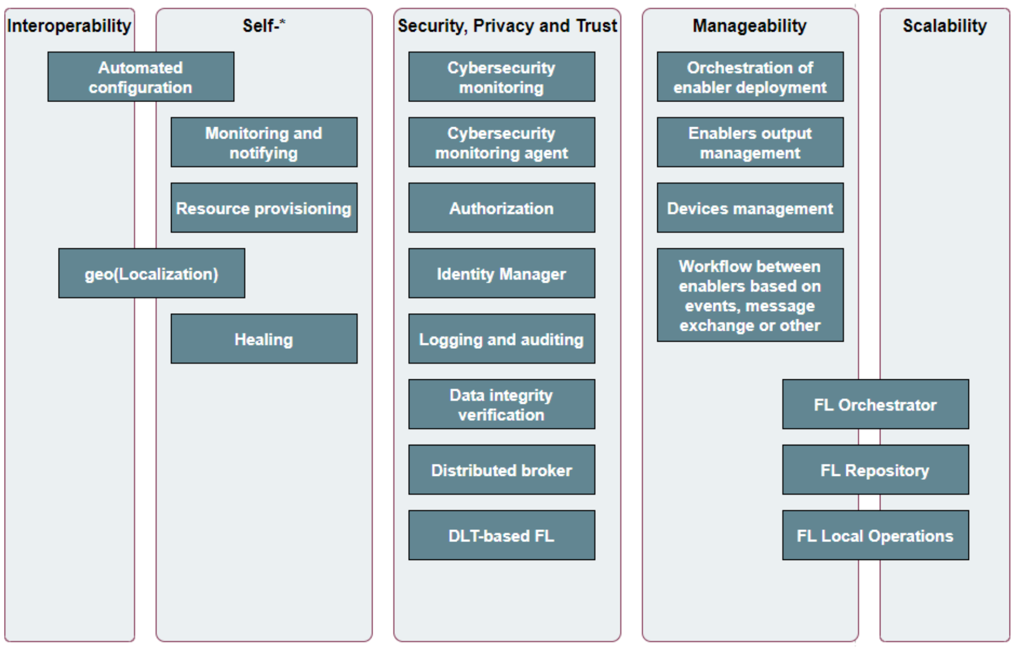


Figure 8. Vertical capabilities summary representation

4.2.1. Device and Edge Plane

The scope of logically separable device and edge plane functions of ASSIST-IoT architecture remains largely unchanged, with regards to the previous version presented in deliverable D3.5.

The materialisation of this Plane is separated in two main pieces: (1) the so-called edge element: ASSIST-IoT Gateway/Edge node (**GWEN**) and (2) devices (or far-edge elements) that are more specific computation equipment instances that act directly over physically related parameters (sensors, localisation…).

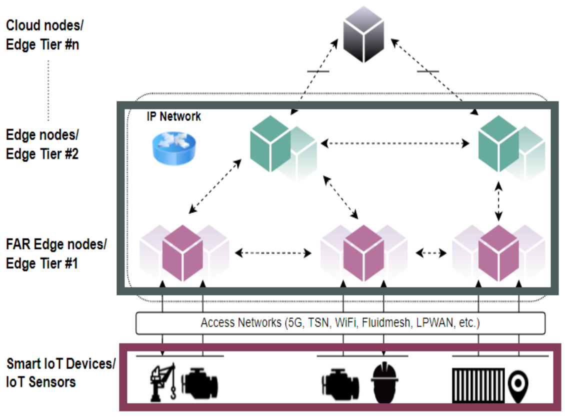

Cloud nodes/ Edge Tier #n
Edge nodes/ Edge Tier #2
FAR Edge nodes/ Edge Tier #1
Smart IoT Devices/ IoT Sensors
IP Network
Access Networks (5G, TSN, WiFi, Fluidmesh, LPWAN, etc.)

Figure 9. Deployment (from D3.5) and Edge and Device Plane actuation spots

The **GWEN** is the element in charge of connecting the bottom layer of the architecture (in terms of equipment) with the upper stride. It provides a set of connectivity, communication, processing, storage, actuation, etc. by default, and can be extended to fit different use-cases. A first design was depicted in D3.5 (Figure 19), and during this M6-M15 period, the capabilities schematic and "reality" of the element have been enhanced. It is observable at Figure 10.

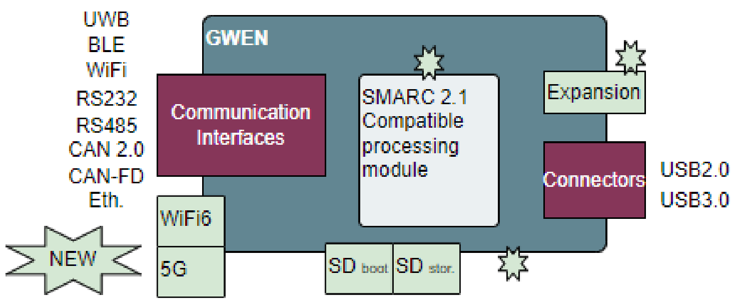


Figure 10. GWEN design updated to fulfil architecture needs

The interfaces included are needed to support different pilot and use cases, not all having to be used in every case. To be flexible, a modular approach has been determined to implement all interfaces. Drawing from D3.5, the design has been enhanced now to include two SD cards, one for booting (OS) and one for storage. In addition, to implement the required compute power a System-On-Module (SOM) will be used, enabling future upgrades of the SOM without redesigning the board. Besides, communication capabilities have also been improved due to the incorporation of built-in 5G and WiFi6 network interfaces modules. Moreover, following the "modularity" and "scalability" philosophy of ASSIST-IoT architecture, the GWEN has included a connector to add carrier boards with expansion modules. This has been mainly designed to allow "embedding" on top of the GWEN additional computing capabilities, specific connection interface modules (e.g., TSN protocol interface) or on-purpose boards (e.g., AI, GPUs). The GWEN is fully equipped with AI processing power to run smart analyses algorithms, which can remain state-of-the-art through distributed and/or Federated Learning update processes. This way, if a configuration adjustment from a smart IoT device or an update of an algorithm is required, this can be passed on to the GWEN. Following the architecture structure, this element is an essential part of the Edge and Device Plane and falls under the **functional blocks:** Analytic capabilities, AI capabilities and Communication capabilities.

With regards to the devices, the ASSIST-IoT architecture does not force any particular implementation, as it is conceived to be open to the specific use-cases. In the case of ASSIST-IoT project, the expressed needs during the first 15 months of the project are:

i. To create (via a device Plane technology) a mechanism through which **indoor localization** of moving objects is enabled in real-time making use of low-latency, state-of-the-art techniques and protocols. To realise this mechanism, ASSIST-IoT team is developing two type of hardware pieces (anchors and tags) able to calculate 2D and 3D positioning using trilateration method, reporting it back to further analysis. This element falls under the **functional blocks**: Enhancement of IoT devices smartness and Analytic capabilities.
ii. To deliver a compact fall detection system, by integrating a microcontroller with a fall arrest sensor, altogether sending a signal via UWB transceivers to the GWEN whenever needed. This element falls under the **functional blocks**: Enhancement of IoT devices smartness and Communication capabilities.

The previous two cases do not preclude the potential creation and addition of other devices into ASSIST-IoT architecture. As part of the design of this plane, it has been specified that any custom (or off-the-shelf) smart computing equipment can act as a far-edge element in ASSIST-IoT architecture as long as it follows the indications provided in D3.5 and in this document.

4.2.2. Smart network and Control Plane

The Smart Network and Control Plane is responsible for connectivity aspects as well as orchestration of virtualised functions, not just network-related ones (e.g., VNFs for delivering services such as load balancing, firewall, packet inspection, etc.) but also for Next-Generation functions belonging to other planes and verticals. The eight enablers that compose this horizontal plane remain largely unchanged, grouped into four blocks, as in the previous iteration of the architecture: orchestration, SDN, network functions and self-contained network realisation. In any case, some considerations regarding the scope of these enablers within the architecture are worth to be revisited.

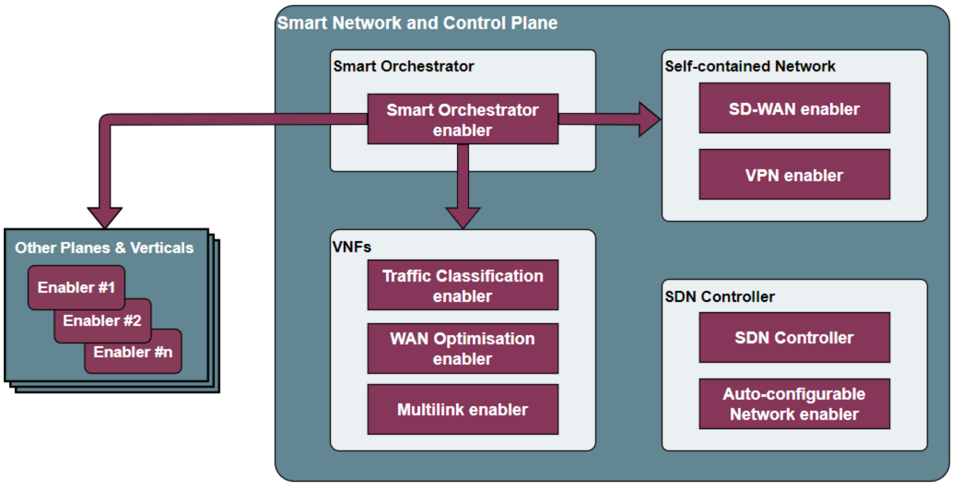


Figure 11. Smart Network and Control Plane functional blocks and enablers

The essential element of the plane is the **smart orchestrator**, which is being designed considering NFV MANO[3] specifications. It is in charge of managing the lifecycle of Virtualised Network Functions (VNFs), still, as the architecture is based on microservices on top of containers, these will be mostly realised as Containers Network Functions (CNFs). Controlling their lifecycles includes positioning them on the optimal place of the managed topology, as well as updating resources as needed, capabilities that can be of utility as well for non-network related functions. For this reason, the orchestrator will be extended to also manage the lifecycle of functions from other planes/verticals (i.e., other enablers), targeting intent-based deployments. It should be mentioned that being k8s selected as the main Virtualised Infrastructure Manager (VIM) technology, some additional features will be included to control network policies and service meshes to facilitate Layer 3 and Layer 4-7 communication among enablers. In any case, the use of k8s as main VIM technology does not preclude the coexistence with additional alternatives for Virtual Machines (such as OpenStack), as currently most available network functions do not yet come in the form of containers. In this way, both virtual machine and container-based network functions could be interconnected and chained for composing network services.

Apart from SD-WAN (which despite belonging to the same paradigm has been included within the block devoted to self-network realisation), the second functional block is devoted to **SDN**. Two forms of SDN are present in the architecture: on the one hand, traffic between containers managed by k8s is controlled programmatically, leveraging networking controllers and plugins, that can be configured via software applying a set of k8s manifests. This feature will be managed by the smart orchestrator. On the other hand, if the hosts that contain k8s nodes (masters and workers) as well as the devices belong to an SDN-enabled network, managed by a controller, typical operations of this paradigm can be applied. Although an enabler that will come in the form of an SDN controller will be part of the plane, it will not be considered as essential for the realisation of the architecture, as many vertical deployments lack SDN-enabled equipment to perform its intended operations.

In the third block the **VNFs** are grouped. These will be managed by the smart orchestrator and will come primarily in the form of containerised VNFs, also known as CNFs. A set of functions, such as traffic classification, multi-link and wan optimisation will be realised as enablers in the project. Other networking functions from external parties that also belong to this group and could be tested include firewalls, load balancers, link aggregators, EPC/5GC, virtual switches, etc.

The last building block is related to **self-contained network** realisation. This block responds to the potential necessity of provisioning a private network over a public channel, ensuring anonymisation and security of communication. Two technologies are envisioned for fulfilling this functionality, VPN and SD-WAN.

In the previous iteration of the architecture, three enablers of this plane were defined as essential. However, as the presence of an SDN-enabled network will not be a mandatory requirement of an ASSIST-IoT deployment, this has left the orchestrator as the only essential enabler for the realisation of an ASSIST-IoT architecture. A high-level diagram of the plane can be seen in the previous figure, and technical details and designs of all mentioned enablers can be found in relevant deliverables.

4.2.3. Data Management Plane

The scope of logically separable data-related functions of ASSIST-IoT architecture remains also largely unchanged, with regards to the previous version presented in deliverable D3.5. Just like before, they can be grouped into semantics and data governance.

Semantic descriptions are supported, although not required, for all data in ASSIST-IoT. The Data Management Plane proposes specific enablers to process, share and present the data, with focus on its semantics. For instance, two solutions are posed to offer data (packets) interoperability: semantic lifting and semantic translation. Semantic lifting, performed by the Syntactic Annotation enabler, is the process of changing the data format to fit a specific semantic form, regardless of whether the original format contained any semantic (meta)data. Semantic translation involves mediating the "meaning" of data, so that it is understood for its consumer. It requires input that is already in a semantic format. The Semantic Translation enabler delivers this functionality and can be configured on-the-fly to support mediation between data packets for any set of data models or

[3] https://www.etsi.org/technologies/nfv

schemas. Semantic data interoperability functions can be used in streaming, or for bulk data.

To support the usage of semantic interoperability, the Data Management Plane includes a Semantic Repository Enabler. This hub of data models, schemas and ontologies enables sharing of common data models with relevant documentation (e.g., usage examples), so that the meaning of data is understood not just by semantically enabled software, but also by users. This supporting function is important to promote the understanding of semantic data, and to help explain how the semantic data interoperability mechanisms are used in practice.

A separate group of features are delivered by the Edge Data Broker, which facilitates the creation of pipelines, in which data flow can be controlled. Scriptable broker nodes enable control over the conditional paths that the data will take, and the side-effects that it may cause. Depending on particular needs, data may be inspected to cause alert events, directed to specific sinks depending on pre-set conditions, or be blocked from further transmission (e.g., for security reasons). The data broker is scriptable with a highly expressive language to enable a wide range of uses.

Finally, the needs for a dependable data storage in a dynamic, distributed and highly virtualized environment are answered by the Long-Term Storage Enabler (LTSE). It supports SQL and NoSQL data models for purposes of secure and highly available archiving and temporary data persistence. A single LTSE can be used independently by many clients (users, services, enablers, etc). The data functions delivered by this solution are aimed at alleviation of the risk of loss of data, in face of hardware failure, or any other unforeseen circumstances that may lead to otherwise unrecoverable data.

In the initial designs, the Data Management Plane of ASSIST-IoT promised to include security-related data functions, implemented with the use of distributed ledger technologies. Those include, for instance, non-repudiation and integrity verification, auditing, tamper detection, etc. It was decided that, because of the cross-cutting nature of DLT functionality, it should not be advertised as a set of plane functions, but rather as a vertical capability, spanning multiple planes and interacting with enablers both on and outside of the Data Management Plane. All functionalities, features, and services offered by DLT enablers were moved to ASSIST-IoT Verticals. Consequently, relevant functions were shifted to a different scope, and are laid out in Section 0 of this document, and relevant WP5 deliverables, which includes D5.2 [2] and, in the future D5.3 and D5.5.

The technical details and designs of all mentioned enablers can be found in relevant deliverables.

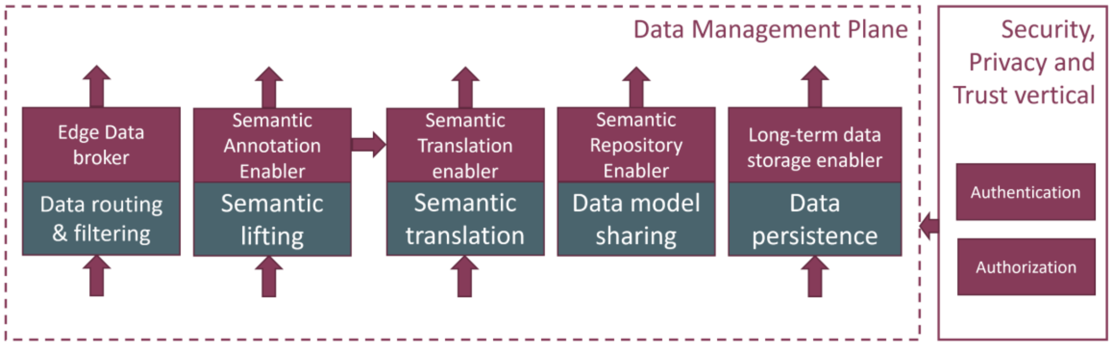


Figure 12. Data Management Plane main functions

As a supplement to the data processing functions, ASSIST-IoT also proposes a set of data processing rules and guidelines, aimed at data administrators. This artifact is intended to be a reference of how to design specific data-related processes whenever sensitive (e.g., private or confidential) data is involved. Because the data processing rules are not architectural or technical in nature, they will be presented in detail in deliverable D2.4.

4.2.4. Application and Services Plane

The ASSIST-IoT Application and Services plane is intended to provide access to data via human-centric visualisation enablers. As initially foreseen in D3.5, three main working lines with regards to functional services are envisioned in this horizontal plane: the development of client-side applications (also called frontends or dashboards), the introduction of novel AR/MR/Computer Vision interfaces (taking into account potential hardware encapsulation requirements as highlighted in Section 5.2), and the provision of graphical toolsets via Open APIs for enabling the open experimentation over ASSIST-IoT deployments. Nevertheless, relevant changes with respect to the expected enablers that can support the previous working lines have been raised. In particular, from the initial 4 enablers envisioned in D3.5 [3], the project has identified two more enablers, leading to the development up to 6 enablers, as shown in Figure 13.

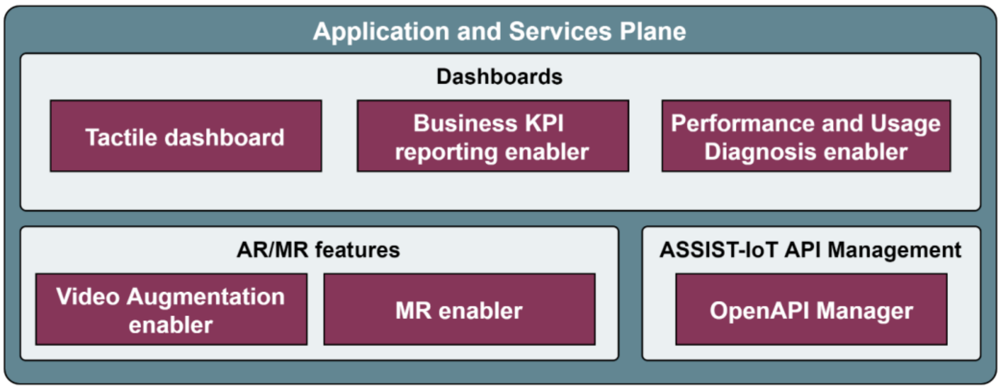

Application and Services Plane
Dashboards
Tactile dashboard
Business KPI reporting enabler
Performance and Usage Diagnosis enabler
AR/MR features
Video Augmentation enabler
MR enabler
ASSIST-IoT API Management
OpenAPI Manager

Figure 13. Application and Services plane functional blocks and enablers.

Although they were widely detailed in deliverable D4.1 [4], and will be further extended in D4.2 [5], as well as their deployment documented in D7.2[6], they are again briefly summarised:

Tactile dashboard enabler: The Tactile dashboard enabler will provide: (i) a frontend framework for allowing the creation of fully reusable web components that can be leveraged to create Simple Page Application (SPA) webpages, or complex web applications and (ii) a backend, that handles all the services needed for generating the frontend dashboard interfaces. The Tactile dashboard enabler will be based on the JavaScript VueJS[4] frontend framework, which provides an approachable, versatile, and performant framework that will help ASSIST-IoT practitioners to create more maintainable and testable code base and backend, that handles all the services needed for generating the frontend dashboard interfaces. It is expected that the Business KPI reporting enabler, and the Performance and Usage Diagnosis enabler are integrated within the tactile dashboard web pages as plugged-in widgets.

Business KPI reporting enabler: Its objective remains the same as described in D3.5. It will provide a web graphical and intuitive interface to make custom graphics, allowing to choose, in a very intuitive way among several types of graphs the collected data from one or more storages and databases of ASSIST-IoT deployments in order to extract business insights, and allow to focus on only the KPI that are relevant to the organisation.

Performance and usage diagnosis enabler (PUD): Its objective remains the same as in D3.5. In short, it will collect performance metrics trying to maintain operational simplicity while being able to adapt to varying scales/levels of the ASSIST-IoT infrastructure. Integrating many service-discovery systems via REST APIs, PUD enabler will stay synchronised with the ASSIST-IoT infrastructure that it is actually monitoring.

OpenAPI management enabler: Its objective remains the same as described in D3.5. It will provide a unified and open API access, consisting of an API design document for each ASSIST-IoT enabler based on Swagger[5] definitions, followed by an API publisher interface. Hence, any type of user will be able to subscribe to the APIs through the API subscription web GUI, namely API Portal, according to his/hers agreed access rights (configured via the Cybersecurity enablers).

Regarding the originally proposed AR/VR/MR enablers in D3.5, two enablers have been finally identified:

Video augmentation enabler: It will allow ASSIST-IoT deployments to perform computer vision functionalities over captured images or video streams either from ASSIST-IoT Edge nodes, or from ASSIST-IoT databases. The enabler consists of a Machine Leaning training block for object detection, and an Inference Engine, aiming to overlap new unseen data as input.

MR enabler: It will provide immersive experience to practitioners of ASSIST-IoT, transforming collected data into more suitable formats for visualisation capabilities over head-mounted MR displays.

[4] https://vuejs.org/
[5] https://swagger.io/

4.3. Node View

Node definition

The Node in ASSIST-IoT, as per Node View in D3.5, is defined as "*a hardware element within the deployment network that can provide computation capabilities by running some ASSIST-IoT enabler execution*".

In this second iteration of the architecture definition, it has been decided to **slightly widen the definition of Node.** From this point on, the Node is not restricted to be a hardware element **but rather a logical element** that is (of course) hosted within a specific computing hardware. This subtle point emanates from the refinement of the Deployment View. According to this refinement, enablers are now spread among k8s clusters, that are composed of nodes. Considering that one single hardware (e.g., a powerful node in high tiers of the topology) can comprise several nodes, it is more precise to refer to "Nodes" in ASSIST-IoT to these logical units rather than to the complete hardware as a single piece. This mental exercise is alike hardware virtualisation in the virtual machines field.

Therefore, ASSIST-IoT Node is assimilable now to the concept of "node" in Kubernetes[6].

Node's baseline technology for enablers deployment

As per architecture design decision (see Deployment View and D3.5), enablers in ASSIST-IoT will materialise in k8s (or assimilable) applications, packaged as Helm charts, to be instantiated in a cluster (composed of one or more nodes). Therefore, every ASSIST-IoT Node must be prepared to execute k8s applications (through pods) as requested, as well as to be addable onto a higher cluster.

Thus, 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. During M6, successful MVPs for the sake of design of the architecture have been tested, using other (k8s-assimilable) technologies. In this period, **k3s**[7], **microk8s**[8] and **k0s**[9] have been tested (in different hardware, e.g., Rpis, low-resources servers…) and have been determined as **valid** to be used as baseline k8s technologies for ASSIST-IoT Nodes.

Notwithstanding the previous, every physical Node will still have its own operative system (Linux OS, Windows, other – Yocto[10] in the case of GWEN).

The updated architecture of the software of the Node can be seen in Section 4.2.1. In comparison to D3.5, a layer with pre-installed software is added and next to the container runtime test software is added (to comply with the previous), and the test software has been included to test the hardware during development of the GWEN.

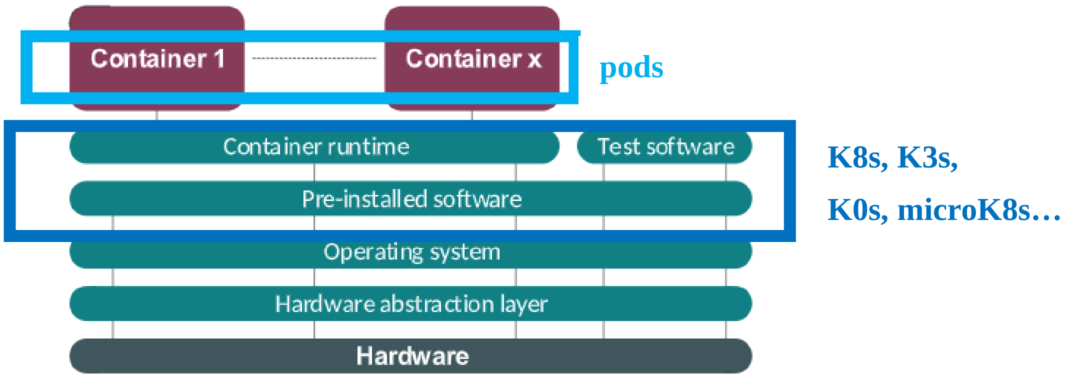


Figure 14. Software architecture of the ASSIST-IoT Node

[6] https://kubernetes.io/docs/concepts/architecture/nodes/
[7] https://k3s.io/
[8] https://microk8s.io/
[9] https://k0sproject.io/
[10] https://www.yoctoproject.org/

4.4. Deployment View

The Deployment View aims at presenting how an ASSIST-IoT architecture is deployed to address the specific uses cases and business scenarios. The first iteration of the architecture focused on two aspects: (i) distribution of physical nodes connected to form a system, and (ii) characteristics related to the deployment and combination of enablers to build a *service*. It should be reminded that, in this context, services must not be confused with the Application and Services plane division (see section 4.2.4). Here, it refers to a specific application offered by the technology providing measurable actions/functionalities to the final user, which may consist of the combination of, for instance, three enablers spotted in the Device and Edge plane and Application and Services plane. This iteration of the architecture reviews and extends these aspects, with the aim of facilitating the realization of an ASSIST-IoT deployment.

From a bird's eye, the Deployment View consists of a set of software and hardware pieces that are used in an ASSIST-IoT deployment. It is addressed to actors such as stakeholders (or system owners), system administrators and developers to gain insights about:

- The hardware elements present within a deployment. It can be used to assess the necessity of additional equipment to extend the already provided services or to offer new ones.
- The enablers utilised/combined to provide a service.
- The instantiation of these enablers, and their components, within the hardware infrastructure.

As it can be seen in Figure 15, an ASSIST-IoT deployment is composed of a set of different IoT **nodes, distributed within** different **edge tiers**, which are **connected to other physical elements to create an interconnected system.** In accordance with the design principles of the architecture, the following considerations apply to any deployment:

1. The **communication between** ASSIST-IoT **nodes** must be **based on IP**. This does not preclude that end-devices (e.g., actuators, sensors, or other lower elements of Figure 15) make use of non-IP networks, if nodes are capable of handling this type of communication, as ASSIST-IoT is access network agnostic. Access networks have to be selected according to the requirements of use cases.
2. To enable the deployment enablers, nodes must have a **Kubernetes distribution installed** (acting as master or as worker). From a user perspective, having one or more clusters in a deployment should be transparent, as the smart orchestrator will allocate enablers over the available nodes according to a set of policies. However, some recommendations are given for system administrators:
    a. A k8s master should control nodes with similar performance (e.g., a high-performance cloud node and a constrained IoT edge one should not be part of the same cluster). See Figure 15, where "M" corresponds to a Master node in a k8s cluster and "S" to a slave node.
    b. If new nodes are to be added to a deployment, it is preferable to include them as workers of an existing cluster, since masters dedicate more resources to control aspects.
    c. **A deployment logically divided in different edge tiers suggest having a master controlling the nodes available in each of them**, as this separation is likely related to different hardware resources.
3. If ASSIST-IoT nodes belong to different networks (e.g., on-premises nodes communication with cloud ones), connectivity must be secured, considering tunnelling technologies such as VPN.
4. Any ASSIST-IoT deployment must have a set of **essential enablers installed** prior to the deployment of services. More information is provided in Section 6.

The first 3 considerations can be important in those deployments in which hardware, communication and Kubernetes infrastructure are managed by different actors, which may likely occur in large environments. In a normal process, ASSIST-IoT dedicated tools and enablers are built on top of these others, which can be considered as prerequisites of an ASSIST-IoT deployment. Apart from these considerations, the **characteristics and design criteria of an ASSIST-IoT deployment** remain mostly unchanged:

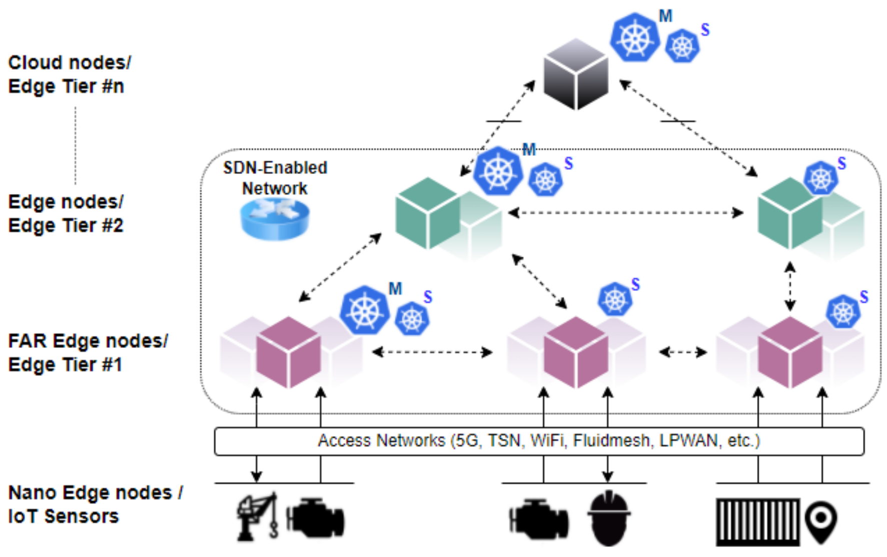

Cloud nodes/ Edge Tier #n
Edge nodes/ Edge Tier #2
FAR Edge nodes/ Edge Tier #1
Nano Edge nodes / IoT Sensors
SDN-Enabled Network
M
S
Access Networks (5G, TSN, WiFi, Fluidmesh, LPWAN, etc.)

Figure 15. Deployment View of ASSIST-IoT architecture

- 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.**
- Communication among nodes of the same tier is considered to support a larger number of use cases.
- The communication between devices and (far) Edge nodes is access network agnostic (5G, WiFi, TSN, LPWAN, etc.).
- Each enabler has its own "scope", and it is formed of different components, which might be 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 service is the combination of various enablers (at least, one). One service may use enablers belonging to different layers (from a Functional View perspective).

4.5. Data View

The Data View presents a high-level perspective on data collection, processing, and consumption. It is meant to be a viewpoint on the flow of data, considering the specific actions that a set of sequential enablers perform upon them. This view allows to understand the system without technical details (usually required to implement designed software, such as description of every message in an authentication scheme), proposing an overview meant to expose a high-level view on the flow of data within the system, using abstractions, such as ASSIST-IoT enablers.

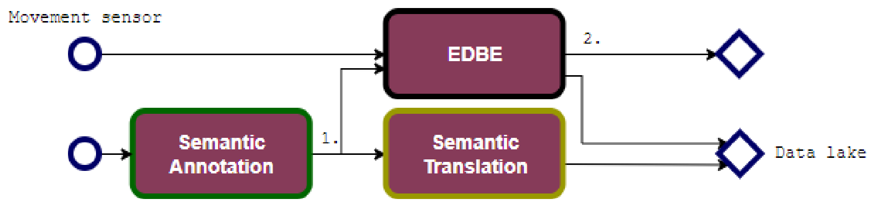
As a highly decentralized architecture, ASSIST-IoT does not mandate any particular flow of data. Instead, it allows to connect data processing functions, spread across the system, into concrete flows that answer particular requirements and needs of concrete deployments. A design abstraction introduced by ASSIST-IoT to describe high-level data flows is called ***data pipeline***.

Data pipelines are descriptions with visual and textual components, of how data flows in distinct parts of ASSIST-IoT deployments, being more abstract than typical, dedicated UML diagrams (e.g., activity diagram, sequence diagram, communication diagram). Any pipeline should describe a process in which data is generated at a **source**, **processed by** some **enablers**, and output at a **sink**, from the point of view of a message travelling through a "virtual *stream*". Because data pipelines are a design tool, and not a software specification standard, there is no requirement to use streaming technologies or asynchronous processing. Data flowing within pipelines is divided into **messages**, which are distinguishable pieces of said data that travel through different **data paths**. It is important to note, that in this context a message must contain concrete data that is directly relevant to the described pipeline. Items, such as performance logs, even though technically may also be named messages, do not need to be included if they do not affect the understanding of the flow. Again, a data pipeline is a high-level abstraction, so gratuitous details are to be avoided.

Every pipeline includes a number of inputs, which are data-generating elements (also called sources). Those are services, endpoints, sensors, etc. at which the data originates, usually constantly, periodically, or as a result of a data-generating event measured or perceived at some sensor. Data is generated at sources in the form of messages, containing information about events, entities, measurements, and any other relevant artifact. Messages travel along directional data paths between enablers, that process the data, i.e., decide where to forward it, change or verify its contents, split the message into two or more, or remove parts of it before sending it further. What kind of processing is done depends on the enabler. Eventually, any data path is terminated in a data sink, where the data is finally consumed, i.e., stored in a database, displayed in a dashboard, added to a log, or simply deleted, if it is no longer needed. In complicated systems, consumption of data may result in starting another flow. In such cases, it is important to clearly separate concerns and functions described in data pipelines, so that independent data processes are described in separate pipelines. Even though, technically, data sinks of one pipeline may be visually (and logically) connected to sources of another pipeline, it is recommended to use simpler and detached diagrams.

In order to document the data pipelines under ASSIST-IoT architecture, the previous abstract concepts have materialised in a set of icons and visual instructions that will help illustrating them. A brief layout of the visual instructions is outlined in Appendix A - .

One example of data pipeline has been created during this period in order to set the foundations for this exercise, that should also be done by external adopters of the solution (beyond ASSIST-IoT) in order to understand enablers' interaction and data flows to be existing:



For the next iteration of this deliverable, this **set of visual instructions will be enhanced** including allowed and forbidden connections and pipeline restrictions. In addition, during the next period M15-M21, this section will be enlarged including the mandatory **data pipelines that an end-to-end data flow** in ASSIST-IoT **may require**. Up to now, those identified data pipelines are as follows:

- Acquisition, storage and visualization of historic data.
- User identification, authorisation and access to the management interface of the platform.
- Transformation of data passing through different enablers.
- Register of critical events in the DLT.
- Flow of IoT data (coming from sensors and distributed among enablers).
- An example of an end-to-end data flow scenario.

4.6. Relation among Views

One of the aspects that complete an architecture, from what concerns the Views of it, is the relation among those particular viewpoints. While the previous sections aimed at explaining the (enhanced status of the) Views of ASSIST-IoT, these paragraphs will reflect on the relation held among them.

As ASSIST-IoT intends to be a blueprint, reference architecture for Next Generation IoT deployments, it aims at following well-known (or de facto) standards for architecture specification creation. In the case of Views relation, the team has selected the 4+1 model published by Kruchten [7] as a valid reflect of its intention. In this reference, a structure of four views is exposed while an additional one (representing the "scenario" or use-case validation) is included where all of them converge. Figure 16 represents how the 4+1 approach fits ASSIST-IoT architecture, which the team of T3.5 decided to extend by tailoring it to the project' solution, adding relevant agents involved in them and specifying heritage relations. This approach has also been followed in previous works [8].

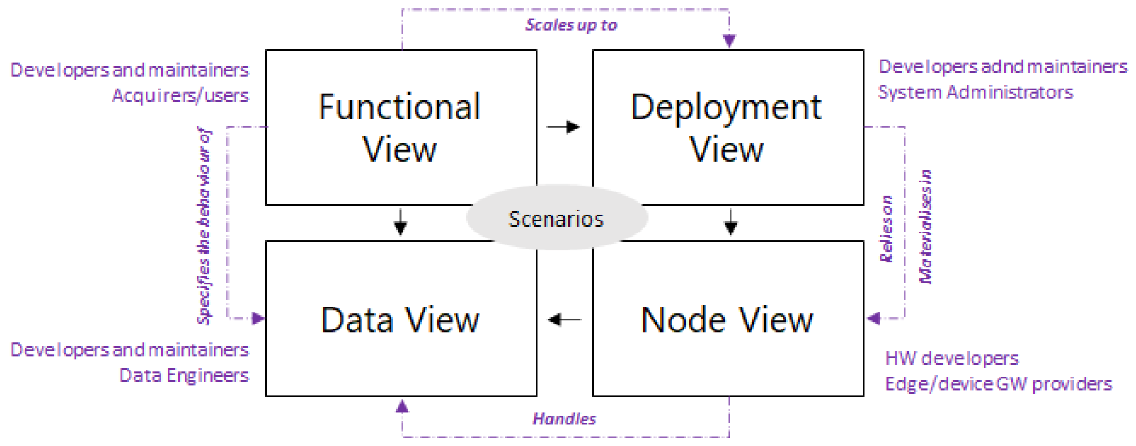
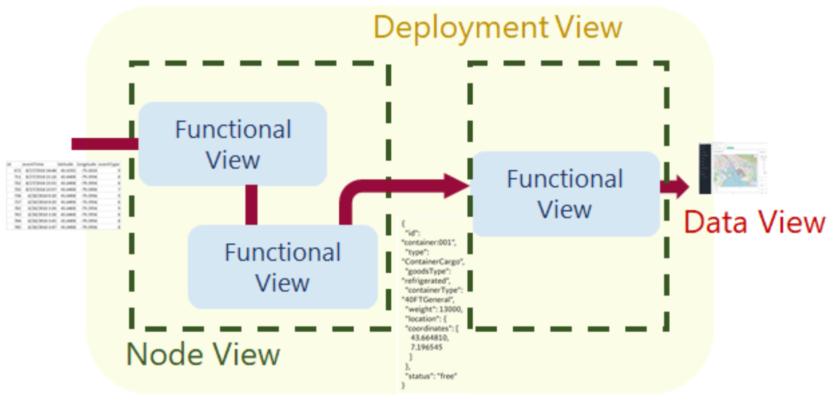


Figure 16. Relation among architecture views following 4+1 model

Comparing to canonical Kruchten's division, Logical View corresponds to ASSIST-IoT's Functional view, aiming at describing the functionality of the system. ASSIST-IoT's Deployment View maps to original Development View, intended to the administration of software artifacts. Project's Node View matches with Kruchten's Physical View, useful for System Administrators relating to the computation infrastructure. Finally, ASSIST-IoT's Data View maps to Process View, trying to represent the dynamic, flowing aspects of the system in terms of processing.

Notwithstanding the previous, and for the sake of clarity, Figure 17 represents a more natural way of expressing their relation. Data View is about the changes over the data processed by the system (sensors, apps data, services data, metadata); Functional View expresses the actual functionalities, while Node View represents physical scopes where computation takes place; and Deployment View joins everything together in an all-encompassing, global, topological, composition approach.



```
{
  "id":
"container:001",
  "type":
"ContainerCargo",
  "goodsType":
"refrigerated",
  "containerType":
"40FTGeneral",
  "weight": 13000,
  "location": {
  "coordinates": [
    43.664810,
    7.196545
   ]
  },
  "status": "free"
}
```

Figure 17. Alternative representation of the relation among ASSIST-IoT architecture Views

5. Update of design principles and global guidelines

5.1. Enablers' conventions

Enablers provide functionalities that belong to different planes and verticals, and hence development and internals can be technologically very different among them. So far, besides their realisation as a set of containers, the main requirement that has been defined is an API as the mechanism that has to be present in every enabler to facilitate the communication between them, and with external services (i.e., single point of exposure). In this manner, internal components of different enablers should not communicate with each other (nor have the possibility). In this second iteration of the architecture, additional conventions are specified to be followed during the design and implementation of enablers as well as for the overall architecture.

It is not the intention of these conventions to hamper developments introducing worthless restrictions, but rather to follow a set of minimum best practices that ease their integration and monitoring in a unified platform, while also facilitating their further utilisation beyond the scope of ASSIST-IoT, in other Cloud-native environments. These conventions, which can be understood as requirements for enablers' design and implementation, are related to three different aspects: endpoints, monitoring and logging.

5.1.1. Endpoints

A first convention that should be followed by all the enablers is API versioning. This practice is required if (typically major) changes may break consumer services or applications that make use of the API. Versioning is realised by adding /v1/, /v2/, etc. at the start of the API paths. Other practices such as using nouns instead of verbs in endpoint paths, handling errors with status codes and informative messages, enable filtering and pagination, and caching are encouraged although not required. Besides, four minimum endpoints have been defined to be incorporated in all the enablers.

- **/health** (GET method). In microservices architectures, it is considered a good practice to incorporate three kinds of probes in the deployments, in particular: (i) *startup probes*, that is used when the container is up and ready, (ii) *liveness probes*, which detect if the application process has crashed; and (iii) *readiness probes*, which ensure that the application is ready to handle requests. A probe can be of type HTTP, which outcome depends on the response code (most common), container command, or TCP probe, where a TCP connection on a specified port is tried to be established.

  Thanks to these probes, containers can be re-deployed automatically by the container orchestrator system, and in case of failure, retrieve some information for later debugging processes. In the scope of ASSIST-IoT, as a minimum requirement, this endpoint will be responsible for **collecting and providing information about the liveness status** of the internal of the components, without the need of the ASSIST-IoT administrator to interact with Kubernetes.
- **/api-export** (GET method). This endpoint will provide a list of methods and endpoints that can be served by the enabler, following OpenAPI specifications[11]. This is typically served in JSON or YAML format. This endpoint fulfils two objectives: on the one hand, to be used as documentation, so end-users can gain knowledge about the usage of each of the provided resources, as they are accompanied by examples; on the other hand, they can be used in API managers or API frameworks (e.g., Swagger[12]) to publish an access API resources, so users with proper rights can make use of them.
- **/version** (GET method). This endpoint will return the current version and patch of the enabler. The versioning format should follow the Semantic Versioning Specification (SemVer)[13].
- **/metrics** (GET method). The objective of this endpoint is to expose relevant metrics of the enabler, which should be determined by its responsible team. The format in which these metrics should be prepared and provided needs to be compliant with the monitoring tool, to be introduced in the following subsection.

[11] https://www.openapis.org/
[12] https://swagger.io/
[13] https://semver.org/

5.1.2. Monitoring

The successful data-driven decision-making processes in a system that consists of a variety of devices and services require addressing important issues of collecting data and metrics about the performance and usage of the resources involved in the lifecycle management of the various services and devices. A monitoring system is called upon resolving this critical task. As various services and devices function on multiple layers, the traditional method in which system administrators, network administrators and application developers are collecting manually metrics on a regular basis is not an acceptable solution. New technologies enable a more automated solution for this problem.

Value of Monitoring in ASSIST-IoT

In the context of a scalable, trustworthy, domain agnostic and interoperable IoT ecosystem, created with several building blocks and enablers, that is proposed by the ASSIST-IoT architecture; the need of a monitoring service is intensified, since in such infrastructures the need to predict failures is considered critical. In the ASSIST-IoT architecture, the Performance and Usage diagnosis enabler (PUD) -more information can be found in deliverable D4.1- will be filling the role of a monitoring service, collecting data and metrics and making it available to other enablers.

Monitoring Architecture

The PUD enabler (based on Prometheus – consolidated k8s standard) is a service that is capable of gathering data and metrics from endpoints made available by other enablers within the ASSIST-IoT ecosystem. In order to achieve that, all ASSIST-IoT enablers need to expose two endpoints to publish their status to the PUD enabler (see 5.1.1 before). The first endpoint is a health endpoint that provides a heartbeat status, while the second one is a metrics endpoint that provides the metrics data. The PUD enabler then stores the received data on the Long Time Storage enabler (LTSE) on a time series database that implements a highly dimensional data model, where the time series entry is identified by a metric name and a set of key-value pairs. The architectural design of the PUD enabler is presented in Figure 18.

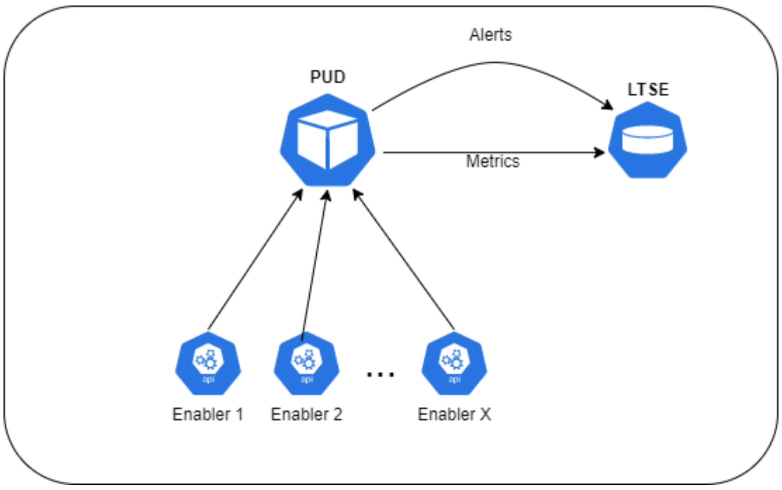


Figure 18. PUD Monitoring Architecture

Monitoring tool on ASSIST-IoT

The tool to be used to implement the “monitoring” in ASSIST-IoT must have the following capabilities:

- Support for a multi-dimensional and flexible data model
- A query language to aggregate time series data in real time.
- HTTP for pulling data and metrics.
- Pushing data to an external time series database.
- Targets can be discoverable via service discovery or static configuration.
- Alert creation with predefined rules.

5.1.3. Logs

Logging is a relevant feature in every distributed system that will help administrators (and in ASSIST-IoT, the system itself) to debug failures and act in advance. As per the current design, all enablers **will contain a logging agent** that will collect logs from the components of the enabler and will forward them to **a log collector and forwarder** (independent enabler) that will be in charge of the distribution of those logs. The logging agent will collect the logs from the enabler components that will be encapsulated as containers within the nodes of the Kubernetes cluster. Detailed **packaging nuances** (e.g., pods strategy) and identification of the appropriate **technology stack** that can facilitate the logging process is **on-going**. The high-level functionality of the logging agent appears in Figure 19.

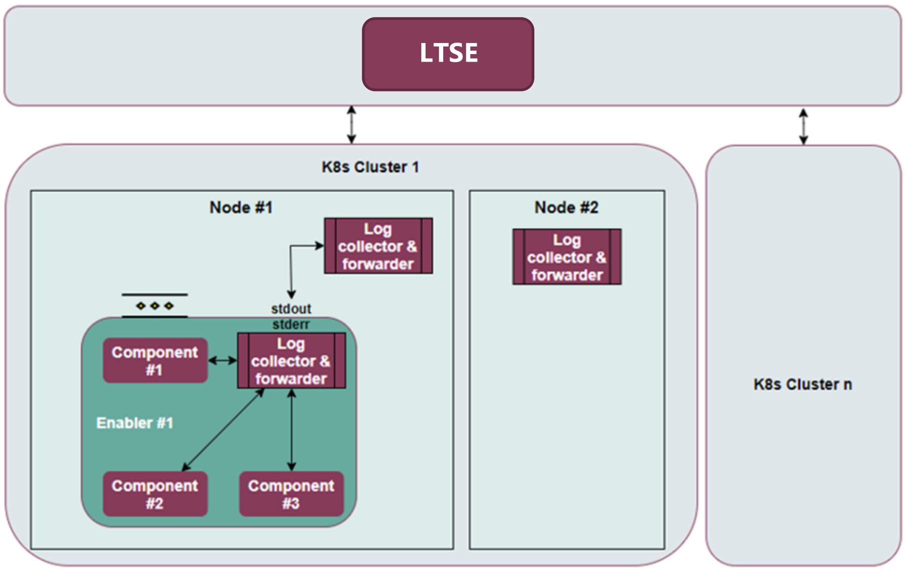


Figure 19. High-level functionality of the logging agent

Critical logs captured by the logging agent will also be forwarded to the DLT logging and auditing enabler. A **set of policies will define which logs will be considered critical**, and an appropriate filtering mechanism will be embedded in the logging agent to output the subset of logs that will need to be forwarded to the DLT logging and auditing enabler via a proper middleware. These logs will not be routed directly to the LTSE -which will be hosted on a centralised server, but **channelled through the EDBE**, in order to avoid having a single point of failures. This way, the decentralisation property of the DLT will also be leveraged.

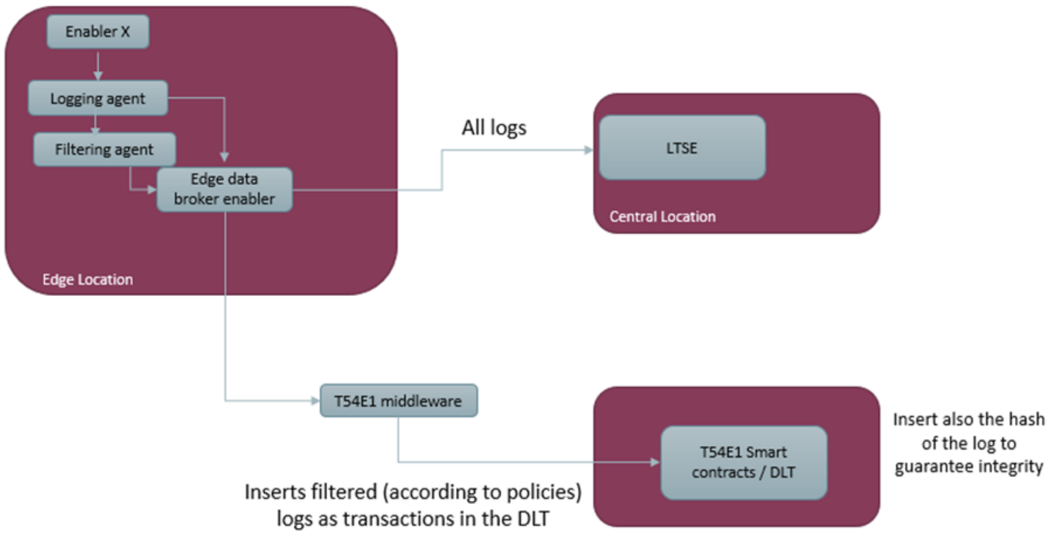

Enabler X
Logging agent
Filtering agent
Edge data broker enabler
Edge Location
All logs
LTSE
Central Location
T54E1 middleware
T54E1 Smart contracts / DLT
Insert also the hash of the log to guarantee integrity
Inserts filtered (according to policies) logs as transactions in the DLT

Figure 20: High-level pipeline of storing critical logs to the DLT

Apart from saving the critical logs, a fingerprint (***hash***) of the log will also be saved to the DLT, allowing for log integrity verification by any involved stakeholder. Special care will take place to store identified sensitive fields (anonymisation, encryption, or avoidance of saving fields of the sensitive logs and whose storage would not provide any extra value). The high-level pipeline of storing critical logs to the DLT appears in Figure 20 above.

5.2. Encapsulation exceptions

The encapsulation approach makes specific assumptions and places requirements on software, that may not be feasible or even possible to implement in practice. Although, in principle, any combination of hardware and software can be virtualised (e.g., emulated), practical demands placed on latency and efficiency may make the cost of encapsulation prohibitive in practice. This section illustrates and justifies examples, where developments that are part of ASSIST-IoT (and follow its architecture) are not encapsulated but integrated into the project in other ways. The provided descriptions highlight problematic areas, explain the design decisions, and propose alternative solutions. They may also guide the design of future software, whose inherent properties make it difficult to fit into the encapsulation principles.

5.2.1. Specific hardware requirements and constraints

Any application that requires specific hardware (whether highly complicated, with restricted resources, or vendor-locked to the used software, e.g., by licensing) can face difficulties in encapsulation. Because the process aims to package the software in a deployable bundle that requires the underlying virtual environment, it puts additional requirements that may be difficult to meet on a given hardware. One example of such issue is apparent in the mixed reality (MR) solution used in ASSIST-IoT.

Novel interfaces, that are used in the MR enabler, offer human-centric interaction through better cooperation of their end-users with the IoT environment. Through the MR enabler, the human effort and decision can be introduced, if and when needed, in the loop of every critical action. The MR enabler aids human-friendly haptics and through MR interfaces, the end-users can receive and provide tactile, real-time and visual feedback as well as data capable of identifying critical improvements, preventions and triggers in long-, short-term, or real-time. Through reporting functions, the MR enabler will gather reliable data to extract information and perform analytics. Decision-making is improved as human flexibility, creativity and expertise interact with IoT platforms and devices.

As mentioned before, in general, the enablers that are deployed in specific devices, which are not designed for high resource consuming applications, face technical constrains that limit their capabilities. The ASSIST-IoT MR enabler, through its interface (the Microsoft HoloLens 2[14] was chosen for the pilot implementation), consumes its processing power to visualise CAD-based models and interact with real-time data through wireless services. Furthermore, any AR application, in order to operate properly, requires direct access to device's hardware such as the cameras, the device sensors (Head Tracking, Eye tracking, Depth and IMU) and specific features such as 6DoF tracking[15] and Spatial Mapping. Considering the above-mentioned energy-demanding processes, we cannot utilize a fully-fledged virtual environment (such as Kubernetes), as the deployment of containerized services reduce the efficiency and at the same time, running multiple replicas on one device will cause a variety of parallelization-related problems. Apart from that, the HoloLens built-in software does not currently fit the encapsulation principles, as there is, at this point, no known (official or otherwise) K8s support for the Windows Holographic Operating System used by the HoloLens Devices. Summarising, the Microsoft HoloLens 2 is a low-resource node and, at the same time, an AR application which already drains most of those resources. Containerising a service, even if it would technically be possible, it is not supported and would aggravate the mentioned resource usage problems.

In theory, in order to overcome the aforementioned obstacles, a potential workaround would be to use the Windows Device Portal in order to manage device status, the app status and some configuration aspects, along with the use of the central management system that will provide a layer of data containing information on how the device will communicate with the ASSIST-IoT. In such design, **the enabler itself would consist of a**

[14] https://www.microsoft.com/es-es/hololens/hardware?SilentAuth=1&wa=wsignin1.0
[15] https://www.ar.rocks/glossary/6dof-tracking

virtualised service to manage the MR device running non-encapsulated software. In other words, the hardware itself would be just a component, and not the whole enabler. Such solution fits, in general, any enabler with specific hardware requirements or constraints. However, this potentiality introduces a complication in the form of additional software (and some hardware node to run it), potential network and computing delays, and with that a dependency that can be avoided, if the encapsulation is not strictly required.

5.3. Special design: choice of Security, DLT and FL strategies

5.3.1. Security management in an architecture based on enablers

When proposing an architecture that is intended to be secure as much as functional, the owner must take into account some principles to guide the design. In that sense, the first principle would be to assume that security is not an absolute situation, but a way of approaching the provided solution with a reasonable level of protection against different threats. The application of this approach can be structured in three main steps: (i) Design, (ii) Implementation and (iii) Monitoring and mitigation.

In the ASSIST-IoT project, these steps are covered by the usage of a secure development process, incorporating security tools and creating a detection-and-response environment.

In order to guarantee a secure development process, the team has proposed to follow a specific DevcSecOps methodology, described in deliverable D6.1 . The underlying aim of this approach is to meet the expected levels of security as it helps identify, prevent, and correct problems that may arise during software development. A set of tools and methodologies will also help provide quality code, identify possible problems and correct them.

What concerns securing the access of an enabler to the global solution, the chosen approach consists of using extended identity and authorization tools that are incorporated into enablers regardless its own scope and functionality. These tools will come as autonomous enablers, responding, and providing service to the requesters, to facilitate the implementation of the access control features.

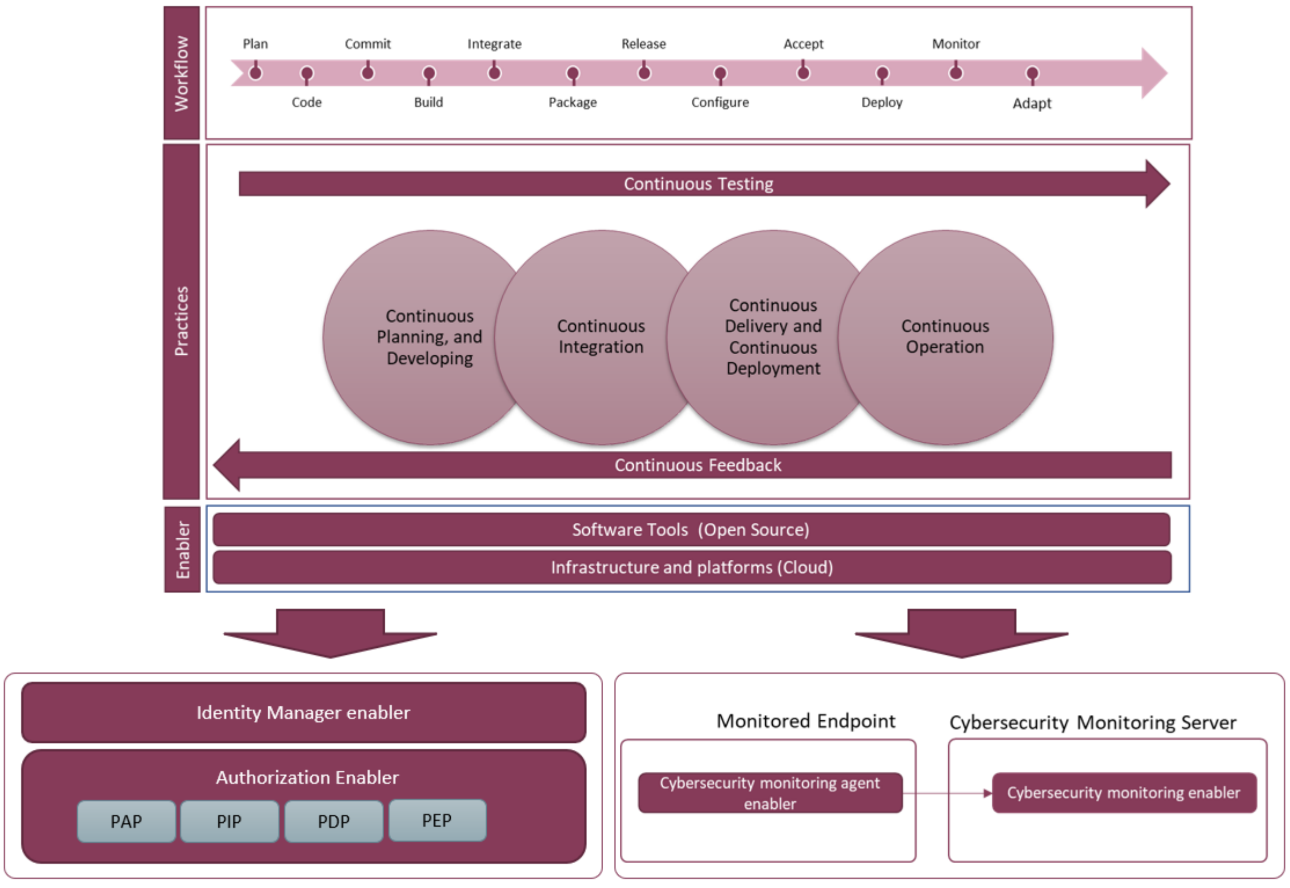


Figure 21. Security enablers contribution to DevSecOps methodology

Lastly, for the monitoring step, the proposed enablers will offer awareness and reaction on situations that may affect the service. Monitoring enablers have been divided in an agent and server architecture to provide an adaptable way of interaction to fit most possibilities. All enablers existing in an ASSIST-IoT deployment should include, in one way or another, a mechanism to feed the threat identification system, and, if possible, provide mitigation mechanisms for those threats.

As described in the Figure 17, the principles and practices applied with the use of DevSecOps, will implement appropriate tools and monitoring that will be incorporated to the core of all the solutions with the use of the described elements.

5.3.2. DLT design and DLT-based access control

DLT and blockchain can create a trust in a trust-less environment of distinct stakeholders due to the immutability of data. As such it will be leveraged to apply the security, privacy, and trust vertical of the ASSIST-IoT conceptual architecture.

It is advised to not store all the data on DLT and blockchains, but **only the record of the most critical events** (off-chain). This is a best practice for DLT, as the ledger is reproduced across several nodes. The critical events are defined as: *"those more relevant to the enablers to implement (logging auditing, verification data, data broker, FL DLT, XACML)"*.

In the design of ASSIST-IoT architecture, DLT enablers are standalone enablers that will be deployed either on the cloud or at the edge depending on the privacy concerns that derive from the data that will be stored on each of these enablers. Therefore, as an example, DLT logging and auditing enabler (see deliverable D5.2 [2] for more) will be cloud based while the FL DLT enabler will have DLT nodes installed at the edge to avoid any leakage of the FL model metadata. Due to the rather limited storage scalability of blockchain and to privacy concerns, it is advised to not store all the data on DLT and blockchains, **but rather opt to include only the most critical events** [9].

In ASSIST-IoT, only a certain number of transactions will be stored via the blockchain:

- Critical logs, via the DLT logging and auditing enabler (see previous section)
- Other specific relevant transactions defined in D5.1 [10]: integrity verification hashes, data source registries to enable interoperability, FL model metadata and XACML policies.

During this process, **Smart Contracts** will be used to deliver functionalities. The usage of Smart Contracts in ASSIST-IoT was also described in described in D5.1

In addition, there will be cases where blockchain nodes will be executed at the edge (FL combined with DLT, and XACML combined with DLT) to compose a fully decentralised approach, while other times there will be hosted in the cloud side. Whenever the latter applies (logging and auditing, integrity verification and broker enabler) the consortium blockchain will depend both on networking capabilities and the performance of the cloud itself. The Consortium blockchain is based on limited and pre-selected participants to construct the network and exhibits characteristics from both private and public blockchains.

DLT-based access control

DLT is a general-purpose technology that has applications in numerous fields and applications. Authorisation is an application that can benefit from DLT implementation to deliver a robust application. For that reason, a novel adaptation for decentralising the Authorization enabler from task T5.3 is suggested to be researched in the project.

Components for the authorisation enabler are the DApp (Decentralised Application) and CA (Certificate Authority) Service. The access control will be supported by community management DApp with two parts. The first one appeals to administrators as a secured backend to manage identities via APIs. A User Interface for handling the access management tasks comprises the second part. The other component in the access control mechanism is the CA Service, a component based on HLF (Hyperledger Fabric) stack that is an authority for certifying endpoints in the network. Essentially, CA Service is responsible for publishing credentials to users. The endpoints of service are restricted to administrators authenticated via the community management DApp's functionalities. A service will expose endpoints that are invoked as part of a user's registration process via the community management DApp and involve the generation and signing of X.509.

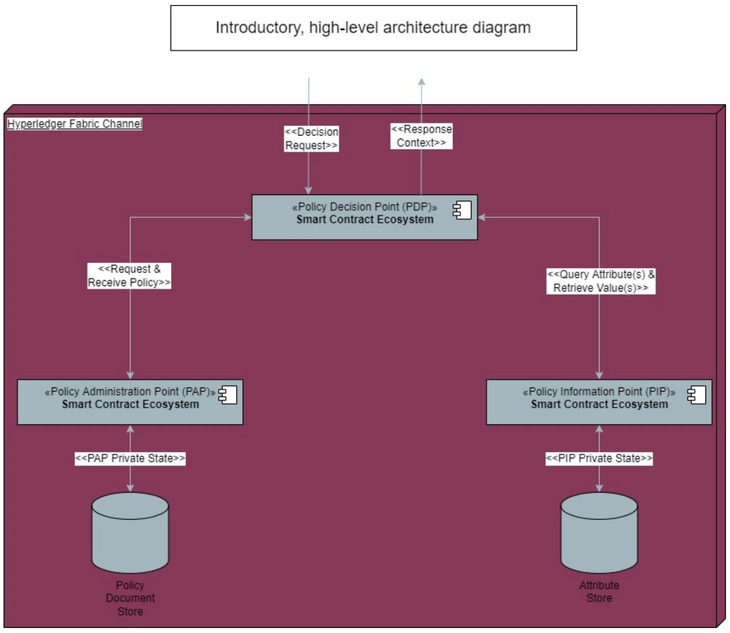

Introductory, high-level architecture diagram
Hyperledger Fabric Channel
<<Decision Request>>
<<Response Context>>
«Policy Decision Point (PDP)» Smart Contract Ecosystem
<<Request & Receive Policy>>
<<Query Attribute(s) & Retrieve Value(s)>>
«Policy Administration Point (PAP)» Smart Contract Ecosystem
«Policy Information Point (PIP)» Smart Contract Ecosystem
<<PAP Private State>>
<<PIP Private State>>
Policy Document Store
Attribute Store

Figure 22: High-level architecture diagram for DLT-based access control

An initial high-level view of the authorisation management is included in the above figure showcasing that the components can run in a decentralised way with the use of HLF. The process commences once a request for a decision reaches the Policy Decision Point (PDP). The PDP is a set of smart contracts deployed on DLT that receives requests and evaluates them according to the policies to reach a decision. As the evaluation is an integral process to reach a decision, there is a need to retrieve values and attributes. The Policy Information Point (PIP) is the source that provides the attributes from identities. Essentially, it acts as the mediator for PDP's access to make decisions. The structure for the authorisation concludes with another integral that is the Policy Administration Point (PAP). The main objective of PAP is to manage policies and other operational aspects for decision making on the access. The following paragraph indicates a number of benefits in adopting DLT for implementing the authorisation compared to a centralised approach.

As ASSIST-IoT strives to push intelligence towards the edge, the idea is to deploy the DLT nodes to the edge devices. The deployment of the authorisation in DLT has various benefits and there are included in research in the field. ASSIST-IoT architecture innovates with the proposal of a DLT-based solution, as the implementations are scarce [11]. The DLT-based approach for the XACML policies can alleviate the malicious denying rights bestowed upon users by policies [12]. Furthermore, actions and events like storing, querying, or updating policies are to be stored as events on the ledger to permit auditability on the nodes behaviour [12]. Blockchain has the potential to be adopted in the IoT sector [13]. Essentially, a blockchain-based framework can achieve high availability, autonomy, and traceability by offloading the policy administration and decision-making to blockchain nodes on the edge. Moreover, centralised systems run the risk of a single point of failure [14]. In contrast, the blockchain's decentralised nature ensures the constant availability of resources, both in computing power and data. This is a characteristic of the blockchain, as the resources are available even with the existence of a single node in the network. The node has a replica of the distributed ledger and is in place to perform tasks with its computing power. Finally, the implementation on public blockchain should avoided to reduce overheads in computations and communication [15].

5.3.3. Federated Learning design choice

According to AIOTI's roadmap [1], the next release (v6) of its HLA will focus on Artificial Intelligence for IoT, where Federated Learning (i.e., improving accuracy by distributed training and inference through various dispersed nodes) is called to play a major role. ASSIST-IoT aims at moving forward in this direction by implementing state-of-the-art solutions into an NGIoT architecture. It is within the plans of the project to actively contribute in the next release on AI-FL topic.

In this sense, ASSIST-IoT aims at developing a standalone FL system, as several use cases of the project are candidates for its use. Due to still the infancy of FL approach into the AI/ML research ecosystem, it should be noticed that the FL system to be developed in the project is considered more like a research-oriented platform than the rest of the enablers of ASSIST-IoT platform. Hence, it is expected that ASSIST-IoT platform will not rely on the contemplated enablers of FL for working appropriately in the different operative industrial environments of the project. Nevertheless, in case any given pilot demands a specific AI/ML functionality, which, in turn, rely on some specific ML model allocated in the FL repository (see its description below), the latter will be accessible beyond the FL environment of the project.

In order to successfully design the appropriate ASSIST-IoT FL System, the team has followed the FL taxonomy proposed in [16]. A detailed explanation was included in deliverable D5.1 [10] as an Appendix.

Following that taxonomy, the partners of the project responsible of its development ended with the below block diagram and flow chart for the ASSIST-IoT Federated Learning process, including its associated FL enablers.

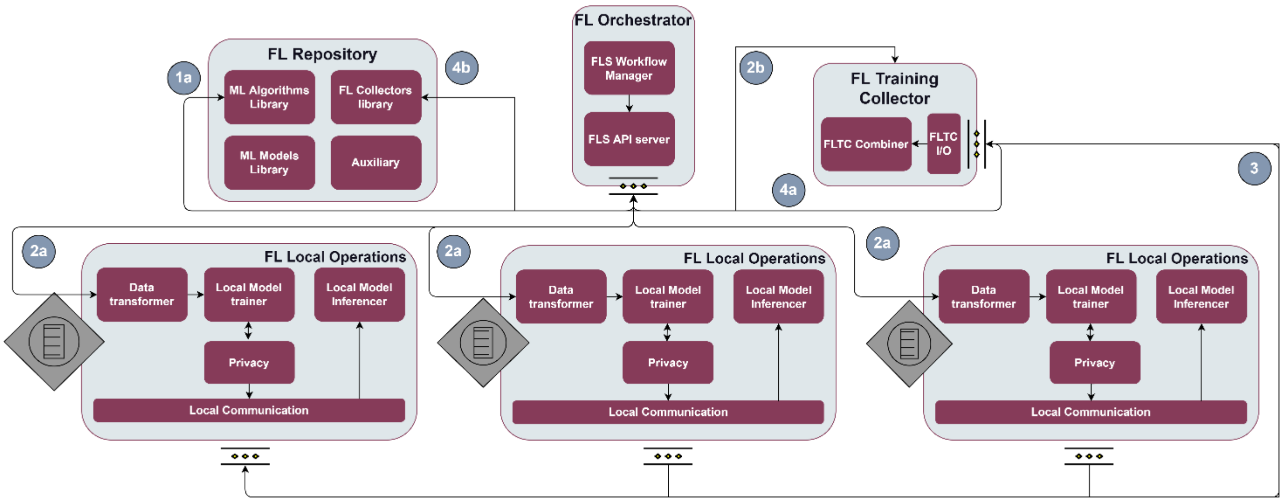


Figure 23. ASSIST-IoT FL system formed by five enablers.

As it can be seen, the independent FL framework of ASSIST-IoT will be mainly formed by four enablers:

- **FL Orchestrator:** responsible for specifying details of FL workflow(s)/pipeline(s), such as FL job scheduling, managing the FL lifecycle, selecting, and delivering initial version(s) of the shared algorithm, as well as handling different “error conditions” that may occur during the FL process.
- **FL Repository:** provides storage for FL related data like initial ML algorithms, already trained ML models suitable for specific data sets and formats, averaging approaches, and auxiliary repositories for other additional functionalities that may be needed. In addition, as mentioned previously, the FL repository will be the only FL enabler that may be accessible for other ASSIST-IoT enablers beyond FL environment, in order to provide specific AI/ML models demanded by some ASSIST-IoT pilot.
- **FL Local Operations:** This enabler will be embedded in each involved party performing local FL training and will be in charge of data formats compatibility verification and potential transformations, local model training, send encrypted recommendations to the FL training collector for averaging (see below), and inferencing the model over new arrival data.
- **FL Training Collector:** responsible of aggregating local updates of the ML model prepared by independent parties as part of a model enhancement process, as well as its delivery via Pickle files back to the FL repository in order to be retrieve back by the FL orchestrator to the FL Local operations in further training rounds.

The development design of all the FL enablers is mainly following the successfully framework used Flower[16] [17], a friendly FL research framework that allows the use of TensorFlow, PyTorch and MXNet models, and provides an easy-to-enter. It is expected that the ASSIST-IoT FL framework may enable a smooth transition

[16] https://flower.dev/

from simulation research to pre-production system research over real edge devices deployed in ASSIST-IoT pilots.

6. Catalogue of essential enablers

ASSIST-IoT architecture is composed of enablers, that are divided in either core (corresponding to horizontal planes) or transversal (corresponding to vertical capabilities). Besides, ASSIST-IoT has built the foundations of a robust system that will be able to be expanded/extended via **other enablers** (e.g., a web application server for notification of certain events or new network functionalities like packet encryption). This way, anything installed over the ASSIST-IoT base architecture will be inserted as an enabler (following guidelines to be delivered in the work of this task jointly with WP6).

However, depending on the characteristics and goals of an IoT deployment, some enablers might be strictly necessary while others might not be. In other words, some pieces of the architecture **are vital to be present in any deployment** while others might be considered complementary. The scope of this section is to describe which enablers (out of all those developed in the project) are **essential** (thus, mandatory) to exist in an IoT deployment for it to be considered an "ASSIST-IoT-based deployment".

The catalogue of enablers listed below is the result of thorough analysis aimed at reducing the number of components required for an ASSIST-IoT system to run. By M15 (due date of this document), several enablers have already produced MVP versions and the Team of T3.5 is in a good position to envision a preliminary list of **essential** enablers. It is believed that this reduced list will also help teams (of the project and, in the future, adopters of the solution) prioritise the deployment in real scenarios. Up to now, **12 enablers** (**out of the 41 planned** to be delivered in the project) have been considered essential. Hence, **29.27%** of them.

Some of the essential enablers will be **pre-installed** in any ASSIST-IoT deployment. This means that, in the moment that a user will start being able to deploy enablers, the previous (pre-installed) will already be existing, as they are the ones that facilitate all the process. Those (pre-installed) enablers do not follow the encapsulation principle, neither are subject to be managed by the Manageability enablers. Therefore, those (pre-installed) can be considered the "pre-requisites" for an ASSIST-IoT deployment to be usable

Essential enablers:

- **Smart Orchestrator (pre-installed)**

This enabler, through a set of descriptors, connection to APIs and taking advantage of the *kubectl* proxy functionality, has been designed to effectively deploy all the rest of enablers in an ASSIST-IoT deployment. It keeps an updated list of the k8s (or assimilable) clusters that are part of the system infrastructure, selects the spot and deploys enablers as CNFs (Container Network Functions).

This enabler is considered essential as it will be the genesis for the rest of the enablers of the deployment. The smart orchestrator must exist in all ASSIST-IoT instances, and it is the **#1 priority** in terms of development and maintenance, as it can be considered the "brain" and "engine" of the whole solution.

- **VPN enabler**

This enabler is in charge of facilitating the access to a node or device from a different network to the site's private network using a public network (e.g., the Internet) or a non-trusted private network. The goal is to allow k8s (or assimilable) clusters hosted in remote machines to become part of the trusted ASSIST-IoT network thanks to the establishment of a VPN tunnel with a "local" (of ASSIST-IoT network) server.

This enabler is considered essential as it is assumed that edge, far-edge nodes and intermediate machines where k8s clusters will live **might not be in the same local network** than the central management location (logical) of ASSIST-IoT (where the Orchestrator and the Manageability backend will live).

- **Edge data broker**

The Edge data broker oversees the distribution of data among nodes assuming that an edge node is involved in the communication. Its role as data router is based on a publish/subscribe schema (data demand and data supply from/to nodes) and takes into account load balancing criteria.

This enabler is considered essential only in those deployments that involve IoT sensoring and the existence of IoT gateways as edge nodes of the solution. The rationale is that these kinds of deployments require a dispatching element to move data from/to edge nodes upwards/downwards.

- **Long-term data storage enabler (LTSE)**

LTSE enabler persists data. While enablers, in general, are free to incorporate databases within their components, the information stored in them will always be referred to the own (logical) scope of the enabler, without keeping data or metadata related to the whole deployment. In addition, LTSE has been designed to cover any kind of storage requested by the enablers in the deployment (relational, non-relational, encrypted, logs, sensor data, configuration data…).

This enabler is considered essential for three main reasons: (i) it keeps information about enablers' context (e.g., volumes, historic, connections) that will be crucial to re-initialise them in case of shutdown, (ii) it allows enablers to rely on a "centric", "cloud" storage so they can save space, and (iii) this element manages access to the platform and enablers, based on roles and user profiles (see below security enablers).

- **Tactile Dashboard enabler (pre-installed)**

The central GUI of the system. Although ASSIST-IoT deployments will require frequent System Administrator (console-based) interventions, the goal of the solution is to allow user-friendly configuration, management of devices, network, services, results and, globally, consultation of parameters at many different levels via a UI.

Therefore, the existence of such centralised user interface is considered essential.

- **OpenAPI management enabler**

It is seldom the case where (in a company) one system performs it all and does not need to retrieve/push information from/to others. Although ASSIST-IoT has been conceived as a holistic NGIoT solution, covering most needs appearing in associated use-cases, the truth that must be faced is that it will live together with other apps and systems that might require interacting with its enablers. ASSIST-IoT acknowledges this reality and has designed as an essential enabler the OpenAPI management enabler to materialise this. Technically, it consists of an API Proxy (with certain rules and configuration) to corresponding interfaces of underlying enablers so that external systems (e.g., Open Call winners' IT tools) can interact with them.

- **Basic Security enablers**

Security is a basic feature that all (modern) IoT systems must equip. In ASSIST-IoT, the **essential** security will be tackled by: (1) the Identity Manager, for validating the identity -after a resource request- against a trusted central server (storing credentials based on OAuth2 protocol), and (2) the Authorisation enabler, that decides (based on XACML policies) whether to grant access to such a resource or not.

- **DLT Logging and auditing enabler**

Transparency, non-repudiation and action accountability have been considered essential when designing ASSIST-IoT solution. Whereas passing all data transactions through blockchains in a large, scalable IoT deployment does not seem the best approach (due to constrained resources and energy consumption), the critical events happening throughout the system must be logged and properly registered via ASSIST-IoT's auditing enabler.

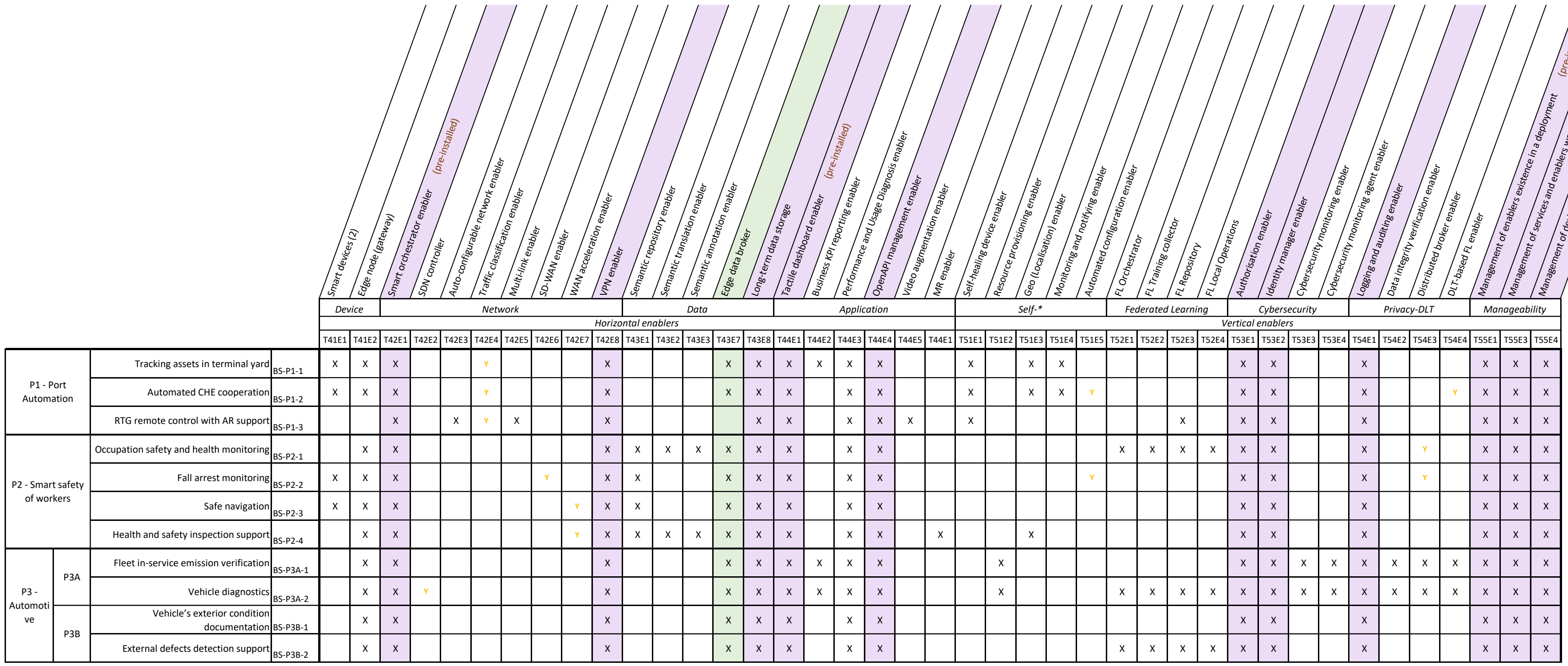
- **Enablers to manage the deployment (pre-installed)**

As per D3.5 (document that sets the foundations of ASSIST-IoT architecture), the essence of the solution is to combine enablers to form services, whose execution is scattered through diverse nodes. Once the first installations will be performed in ASSIST-IoT (the enablers listed above), the last functionality will be the one that will allow such combination of enablers in the infrastructure. This functionality is provided by the Services and enablers workflow, that will become a centrepiece of the management of an ASSIST-IoT deployment (where System owners will spend more time in). This requires, first, the management of the enablers existing in the deployment, allowing their download, configuration, deployment, modification and removal; and, secondly, the handling of devices and nodes (k8s clusters), via the Devices management enabler.

.

7. Architecture instantiation in Pilots (intermediate approach)

This section aims at representing how the diverse elements of the architecture defined (and developed) in ASSIST-IoT **will be demonstrated and validated in use-cases of the different pilots**. As it can be seen, the essential enablers (see previous section) will be present in all implementations, whereas diverse specific vertical and core enablers will be present (or not) whenever required depending on the characteristics of the IoT deployment and the goal of the experimentation.



			Horizontal enablers																				
			Device		Network								Data					Application					
			Smart devices (2)	Edge node (gateway)	Smart orchestrator enabler (pre-installed)	SDN controller	Auto-configurable network enabler	Traffic classification enabler	Multi-link enabler	SD-WAN enabler	WAN acceleration enabler	VPN enabler	Semantic repository enabler	Semantic translation enabler	Semantic annotation enabler	Edge data broker	Long-term data storage	Tactile dashboard enabler (pre-installed)	Business KPI reporting enabler	Performance and Usage Diagnosis enabler	OpenAPI management enabler	Video augmentation enabler	MR enabler
			T41E1	T41E2	T42E1	T42E2	T42E3	T42E4	T42E5	T42E6	T42E7	T42E8	T43E1	T43E2	T43E3	T43E7	T43E8	T44E1	T44E2	T44E3	T44E4	T44E5	T44E1
P1 - Port Automation		Tracking assets in terminal yard	BS-P1-1	X	X	X			Y				X				X	X	X	X	X	X		
P1 - Port Automation		Automated CHE cooperation	BS-P1-2	X	X	X			Y				X				X	X	X		X	X		
P1 - Port Automation		RTG remote control with AR support	BS-P1-3			X		X	Y	X			X					X	X		X	X	X	
P2 - Smart safety of workers		Occupation safety and health monitoring	BS-P2-1		X	X							X	X	X	X	X	X	X		X	X		
P2 - Smart safety of workers		Fall arrest monitoring	BS-P2-2	X	X	X					Y		X	X			X	X	X		X	X		
P2 - Smart safety of workers		Safe navigation	BS-P2-3	X	X	X						Y	X	X			X	X	X		X	X		
P2 - Smart safety of workers		Health and safety inspection support	BS-P2-4		X	X						Y	X	X	X	X	X	X	X		X	X		X
P3 - Automotive	P3A	Fleet in-service emission verification	BS-P3A-1		X	X							X				X	X	X	X	X	X		
P3 - Automotive	P3A	Vehicle diagnostics	BS-P3A-2		X	X	Y						X				X	X	X	X	X	X		
P3 - Automotive	P3B	Vehicle's exterior condition documentation	BS-P3B-1		X	X							X				X	X	X		X	X		
P3 - Automotive	P3B	External defects detection support	BS-P3B-2		X	X							X				X	X	X		X	X		

			Vertical enablers																			
			Self-*					Federated Learning				Cybersecurity				Privacy-DLT				Manageability		
			Self-healing device enabler	Resource provisioning enabler	Geo (Localisation) enabler	Monitoring and notifying enabler	Automated configuration enabler	FL Orchestrator	FL Training collector	FL Repository	FL Local Operations	Authorisation enabler	Identity manager enabler	Cybersecurity monitoring enabler	Cybersecurity monitoring agent enabler	Logging and auditing enabler	Data integrity verification enabler	Distributed broker enabler	DLT-based FL enabler	Management of enablers existence in a deployment (pre-installed)	Management of services and enablers workflow (pre-installed)	Management of devices in an ASSIST-IoT deployment (pre-installed)
			T51E1	T51E2	T51E3	T51E4	T51E5	T52E1	T52E2	T52E3	T52E4	T53E1	T53E2	T53E3	T53E4	T54E1	T54E2	T54E3	T54E4	T55E1	T55E3	T55E4
P1 - Port Automation		Tracking assets in terminal yard	BS-P1-1	X		X	X						X	X			X				X	X	X
P1 - Port Automation		Automated CHE cooperation	BS-P1-2	X		X	X	Y					X	X			X			Y	X	X	X
P1 - Port Automation		RTG remote control with AR support	BS-P1-3	X							X		X	X			X				X	X	X
P2 - Smart safety of workers		Occupation safety and health monitoring	BS-P2-1						X	X	X	X	X	X			X		Y		X	X	X
P2 - Smart safety of workers		Fall arrest monitoring	BS-P2-2					Y					X	X			X		Y		X	X	X
P2 - Smart safety of workers		Safe navigation	BS-P2-3										X	X			X				X	X	X
P2 - Smart safety of workers		Health and safety inspection support	BS-P2-4			X							X	X			X				X	X	X
P3 - Automotive	P3A	Fleet in-service emission verification	BS-P3A-1		X								X	X	X	X	X	X	X	X	X	X	X
P3 - Automotive	P3A	Vehicle diagnostics	BS-P3A-2		X				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
P3 - Automotive	P3B	Vehicle's exterior condition documentation	BS-P3B-1										X	X			X				X	X	X
P3 - Automotive	P3B	External defects detection support	BS-P3B-2						X	X	X	X	X	X			X				X	X	X

Figure 24. Preliminary validation scenarios for all ASSIST-IoT enablers

It is worth mentioning that this is only a preliminary exercise and that this is currently under thorough study in WP7, where the actual deployment of ASSIST-IoT will take place. Therefore, this table will be completely filled after the delivery of D7.2 (M18), where the usage of enablers per pilot will be clarified. In addition, it is also remarkable the fact that this table only contemplates Business Scenarios (BS), while it is most likely that for next iteration of the architecture (D3.7), this table will add a new granularity, per use case.

To interpret the table, it must be considered that: (i) columns in purple correspond to essential enablers (therefore present in all scenarios), (ii) enablers with "X" in particular scenarios are identified to be potentially demonstrated there (**subject to changes**), and (iii) enablers with "**Y**" in particular scenarios represent those cases for which demonstration is expected to be conducted only under specific theoretical/demo conditions.

8. Conclusions and Future Work

This document presents the intermediate definition of ASSIST-IoT architecture, being the second of a series of three iterations and hence to be further refined during the final version, scheduled for M21. 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 version defined in D3.5. The main objective of this document is to provide insights with respect to the technical outcomes to be produced on WP4, and WP5.

After recapping a brief overview of the main concepts and architecture paradigms defined in D3.5, including (i) the main design principles, (ii) the conceptual architecture, and (iii) the methodology that guide the definition of the architecture, D3.6 outlines the following advances expected over the architecture that will be finally formalised on updated content in deliverable D3.7.

- **Refinement of Views**: The Views of ASSIST-IoT architecture have been enhanced in this deliverable (Functional, Node, Deployment). Furthermore, a new Data View that represents the architecture from the viewpoint of end-to-end data flow has been introduced.
- **Enhancement of currently identified enablers**: The descriptions of the key enablers have been refined, based on the insights and problems identified during their development in WP4 and WP5, including the extension of previously defined ones, as well as the addition and removal of others.
- **Settlement of architectural decisions**, indicating the selected technologies that is guiding the materialisation of ASSIST-IoT architecture pilots 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.
    - Security: how the security is being tackled in ASSIST-IoT – via monitoring agents, access control and competences separated per enablers.
    - Strategy on Federated Learning: how it has been decomposed in various parts (repository, orchestrator, training collector and local operation), their role and how they interact among each other.
    - Which data, when and how must go through ASSIST-IoT’s Distributed Ledger Technology nodes (and whether this take place in cloud-assimilable nodes, in edge nodes, etc.).
    - Some **common conventions for enablers,** which include their endpoints (following a specific structure and a set of mandatory features to be included), how to retrieve and store logs and how to monitor the status of all enablers.

It is important to remember that the goal of this document is to enhance the delivered 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 intermediate definition, so that further refinement is expected in the final version to be produced in M21.

References

[1] O. Elloumi, M. Carugi, J. P. Desbenoit, G. Karagiannis, and P. Murdock, “AIOTI-WG3-High Level Architecture (HLA) Release 5.0,” 2020, Online: https://aioti.eu/wp-content/uploads/2020/12/AIOTI_HLA_R5_201221_Published.pdf.

[2] K. Wasielewska-Michniewska *et al.*, “D5.2 - Traversal Enablers Development Preliminary Version,” Oct. 2021, Online: https://assist-iot.eu/wp-content/uploads/2021/12/ASSIST-IoT_D5.2-Transversal-Enablers-Development-Preliminary-Version-v1.0.pdf.

[3] A. Fornés *et al.*, “D3.5 - ASSIST-IoT Architecture Definition – Initia,” Apr. 2021, Online: https://assist-iot.eu/wp-content/uploads/2021/12/ASSIST-IoT_D3.5_Architecture_Definition_Initial_v1.0.pdf.

[4] A. Fornés *et al.*, “D4.1 Initial Core Enablers Specification,” Jul. 2021, Online: https://assist-iot.eu/wp-content/uploads/2021/12/ASSIST-IoT_D4.1_Initial_Core_Enablers_Specification_v1.pdf.

[5] ASSIST-IoT Project, “D4.2 - Core Enablers Specification and Implementation,” Apr. 2022.

[6] ASSIST-IoT Project, “D7.2 - Pilot Scenario Implementation – First Version,” Apr. 2022.

[7] P. B. Kruchten, “The 4+1 View Model of architecture,” *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[8] Q. Tu, M. W. Godfrey, and X. Dong, “Modelling and Extracting the Build-Time Software Architecture View,” 2003.

[9] J. Chen, Z. Lv, and H. Song, “Design of personnel big data management system based on blockchain,” *Future Generation Computer Systems*, vol. 101, pp. 1122–1129, Dec. 2019.

[10] K. Wasielewska-Michniewska *et al.*, “D5.1 - Software Structure And Preliminary Design,” Jul. 2021, Online: https://assist-iot.eu/wp-content/uploads/2021/12/ASSIST-IoT_D5.1_Software_Structure_and_Preliminary_Design_v1.pdf.

[11] I. Achour, S. Ayed, and H. Idoudi, “On the Implementation of Access Control in Ethereum Blockchain,” in *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)* , 2021, pp. 483–487.

[12] D. di Francesco Maesa, P. Mori, and L. Ricci, “Blockchain Based Access Control,” 2017, pp. 206–220.

[13] E. Chen, Y. Zhu, Z. Zhou, S.-Y. Lee, W. E. Wong, and W. C.-C. Chu, “Policychain: A Decentralized Authorization Service with Script-driven Policy on Blockchain for Internet of Things,” *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[14] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, “Distributed Attribute-Based Access Control System Using a Permissioned Blockchain,” Jun. 2020.

[15] “Private Blockchain-Based Secure Access Control for Smart Home Systems,” *KSII Transactions on Internet and Information Systems*, vol. 12, no. 12, Dec. 2018.

[16] Q. Li *et al.*, “A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection,” Jul. 2019.

[17] D. J. Beutel *et al.*, “Flower: A Friendly Federated Learning Research Framework,” Jul. 2020.

Appendix A - Visual instructions for documenting data pipelines

The following image represent the instructions that have been designed to define data pipelines in ASSIST-IoT architecture.

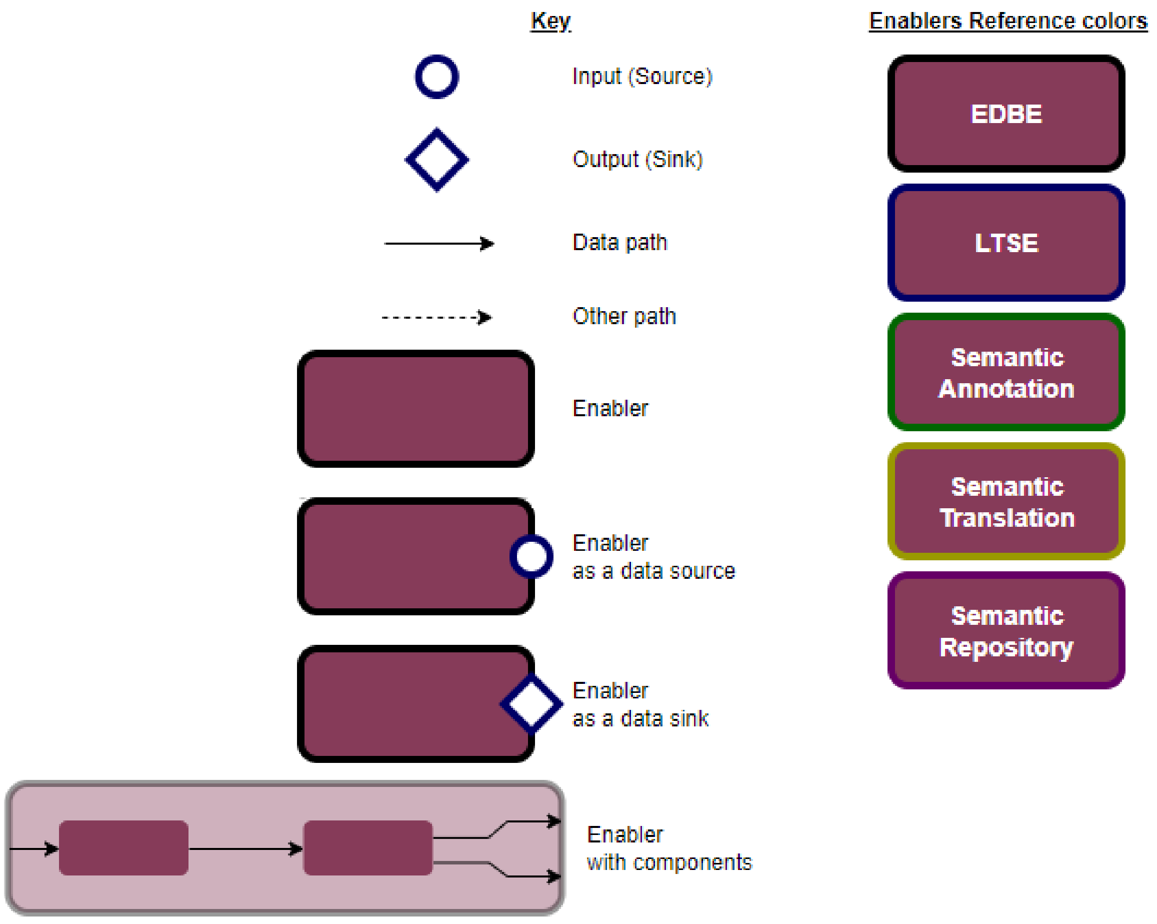


Figure 25. Visual instructions for data pipelines