3

Tactile IoT Architecture for the IoT–Edge–Cloud Continuum: The ASSIST-IoT Approach

A. Fornes-Leal[1], I. Lacalle[1], C. E. Palau[1], P. Szmeja[2], M. Ganzha[2], F. Konstantinidis[3], and E. Garro[4]

[1]Universitat Politècnica de València, Spain
[2]Systems Research Institute Polish Academy of Sciences, Poland
[3]Institute of Communications and Computer Systems, National Technical University of Athens, Greece
[4]Prodevelop SL, València, Spain
E-mail: cpalau@dcom.upv.es; alforlea@upv.es; iglaub@upv.es; Pawel.Szmeja@ibspan.waw.pl; maria.ganzha@ibspan.waw.pl; fotios.konstantinidis@iccs.gr; egarro@prodevelop.es

Abstract

This chapter describes the ASSIST-IoT approach for tactile IoT, proposing a reference architecture built on cloud-native concepts in which several enabling technologies (AI, cloud/edge computing, 5G, DLT, AR/VR interfaces, etc.) are integrated to implement advanced tactile IoT use cases, providing a set of guidelines, best practices, and recommendations toward this end.

Keywords: Reference architecture, next-generation IoT, cloud-native, Tactile Internet, enablers.

3.1 Introduction

With IoT consolidated in several application domains, the next-generation IoT (NG-IoT) has emerged, aiming at addressing more ambitious and

complex use cases. Different enabling technologies have been identified as key toward this evolution, such as edge computing, 5G, artificial intelligence (AI) and advanced analytics, augmented reality (AR), digital twins, and distributed ledger technologies (DLTs). Still, there is not available any reference architecture (RA) that provides (technical and non-technical) requirements, guidelines, best practices, and recommendations that serve as blueprint for developing and implementing such systems, that being one of the main objectives of the ASSIST-IoT project.

Developing such RA thinking on its further adoption is crucial, and hence it tries to avoid a very high level of abstraction, selecting a set of design principles that consider current and expected trends in the IoT and enabling technologies communities. In this sense, the cloud-native paradigm (based on microservices, containerization, and DevOps practices) is embraced, adapted to the edge-cloud computing continuum as a baseline for its conception. Because of it, the RA will strive to bring flexibility, scalability, and ease of integration, which are crucial for coping with the continuous and fast evolution of the NG-IoT ecosystem as well as to help consolidate the implementation of tactile IoT across different business sectors, which are needed by the industry [1].

3.2 Concepts and Approach

Reference architectures are usually intended to be generic; so they can be applicable to different sectors or application domains. The RA developed in ASSIST-IoT follows the approaches and vocabulary specified in the standard ISO/IEC/IEEE 42010 [2], which is widely used in many modern RAs. Among the vocabulary used, the following terms are key in the conception of the presented architecture:

- **Stakeholder**: Individual, team, organization, or classes thereof, having an interest in a system [2]. They might be technology-focused or not, ranging from developers to testers, maintainers, administrators, and end users, among others [3].
- **Concern**: Topic of interest of one or more stakeholders to the architecture [4]. It includes needs, goals, expectations, requirements, design constraints, risks, assumptions, or other issues belonging to the system-of-interest [2].
- **View**: Work product representing the architecture from the perspective of specific system concerns [2], depicting how the architecture tackles them.

- **Perspective**: Collection of tactics, activities, and guidelines to ensure that a system displays a specific property, which should be considered across the views [3]. Perspectives are also referred to as system characteristics, although in ASSIST-IoT, the term **vertical** is used instead, representing not just properties but also functional blocks solving specific cross-cutting concerns.

IoT applications can be simple, composed of a less number of devices, with a basic frontend−backend-database schema and relaxed communication requirements in terms of latency and bandwidth. However, as IoT systems grow in size and complexity and NG-IoT requirements come to play, software architectures are highly recommended as a starting point to design them as well as to solve the specific needs or issues that may arise. There are different software architecture patterns [5], which could be combined in some cases: layered, event-driven, space-based, serverless, based on services, etc., and among the latter, monolithic, service-oriented architectures (SOA), and microservices. The ASSIST-IoT RA will consist in a layered architecture based on services, which is a result of the influence of cloud-native approaches over typical IoT representations.

3.2.1 Design principles

NG-IoT enables more appealing applications at the expense of complexity. Complementary technologies should be integrated depending on the use case addressed, and hence modularity and agile adaptation cycles must be ensured. The principles that govern the ASSIST-IoT RA are:

1. **Microservices**: The RA, apart from following a layered, multi-dimensional approach (see the next sub-section), proposes following a microservices pattern, allowing independent, self-contained services to be deployed and scaled while specifying boundaries and allowing coding freedom. All services should declare their own, well-defined communication interfaces.
2. **Containerization**: Virtualization, specifically in the form of containers, is key for deploying the services and decoupling them from the underlying hardware resources. They have much larger community support than alternatives such as unikernels or serverless, while being lighter, faster, and more flexible than virtual machines (VMs), and thus they are fostered in the cloud-native paradigm.

3. **Enablers**: An enabler is a collection of software, running on computation nodes, which delivers a particular functionality to the system. It is formed by a set of interconnected components, realized as containers, and exposing a set of well-defined interfaces (internals are not exposed or accessible). They can be essential (needed at all or most deployments), optional, or relevant only for certain use cases. Some features, due to containerization inconvenience or unfeasibility, should be implemented directly as a host operating system's (OS) service.
4. **Kubernetes**: A container orchestration framework provides many benefits, as automation of rollouts/rollbacks, error handling, and resource optimization (upscaling/downscaling) and, most importantly, bridges the gap from development to the market. K8s, although not mandated, is selected for being the *de facto* standard, and some decisions have been made considering it.

3.2.2 Conceptual approach

The conceptual architecture has been envisioned considering not only previous IoT schemas and cloud-native concepts but also the advancements in enabling technologies (e.g., edge computing, AI, SDN/NFV paradigm), outcomes from previous and concurrent projects, partners' expertise, and extensive research, being influenced primarily by the LSP programme [6], the OpenFog consortium [7], and AIOTI HLA [8]. The conceptual architecture is two-dimensional, primarily focused on the functional features, grouped in four layers or **planes** (device and edge, smart network and control, data management, and application and services), representing collections of features that can be logically layered on top of each other, intersected by cross-cutting properties, or **verticals** (self-*, interoperability, scalability, manageability, and security, privacy, and trust), as seen in Figure 3.1.

3.3 Architecture Views

The views described in the following sections compose, altogether, the whole scope of the architecture; separately, they represent an observation prism of the whole specification fit to the wills of a group of stakeholders. Five views have been developed: functional, development, node, deployment, and data, following a customized Kruchten's model [9], coined "$4\frac{1}{2}$+1," after splitting its development view into two (development and deployment). The relation among them can be seen in Figure 3.2.

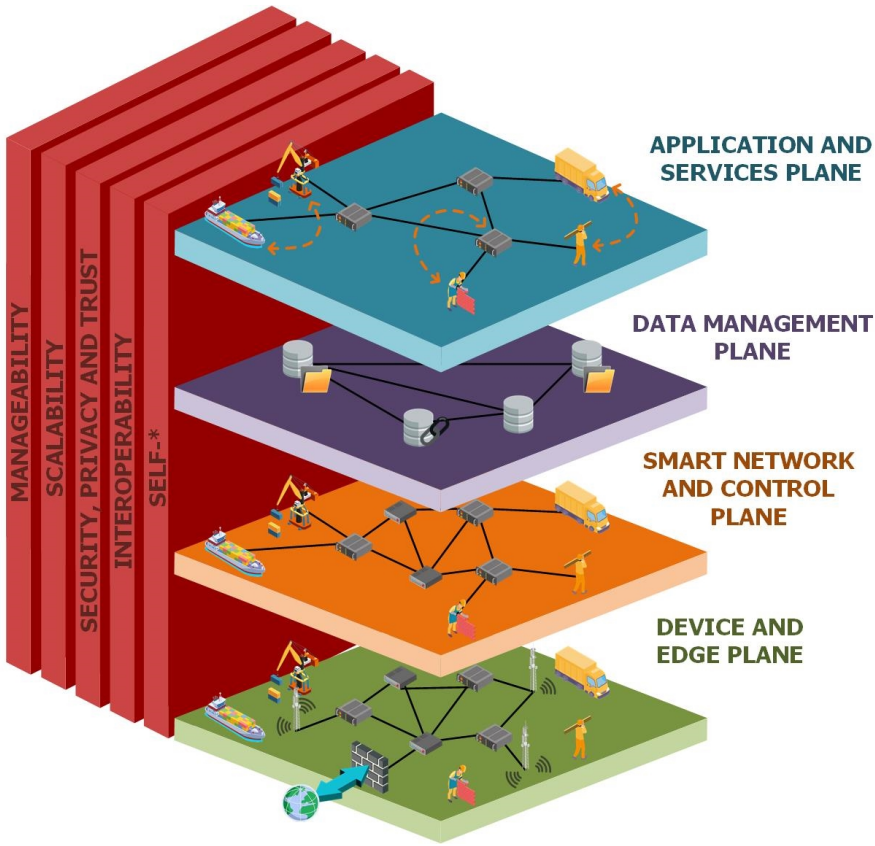

Figure 3.1 ASSIST-IoT conceptual architecture.

3.3.1 Functional view

This view, sometimes referred to as logical, represents the functionalities provided by a system, which is crucial for developers and maintainers as well as users and acquirers of the solution. A set of enablers are introduced for each of the planes (Figure 3.3), always considering that a system realization could require only a subset of them and/or include additional ones tailored to it.

3.3.1.1 Device and edge plane

This plane is in charge of (i) providing the infrastructure elements (e.g., computing nodes, networking elements, etc.) needed for interacting with end

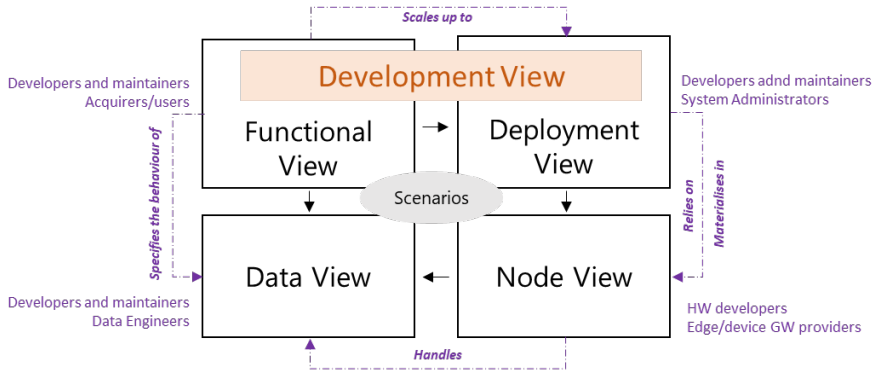

Figure 3.2 Custom $4\frac{1}{2}$+1 model of relation among views in ASSIST-IoT RA.

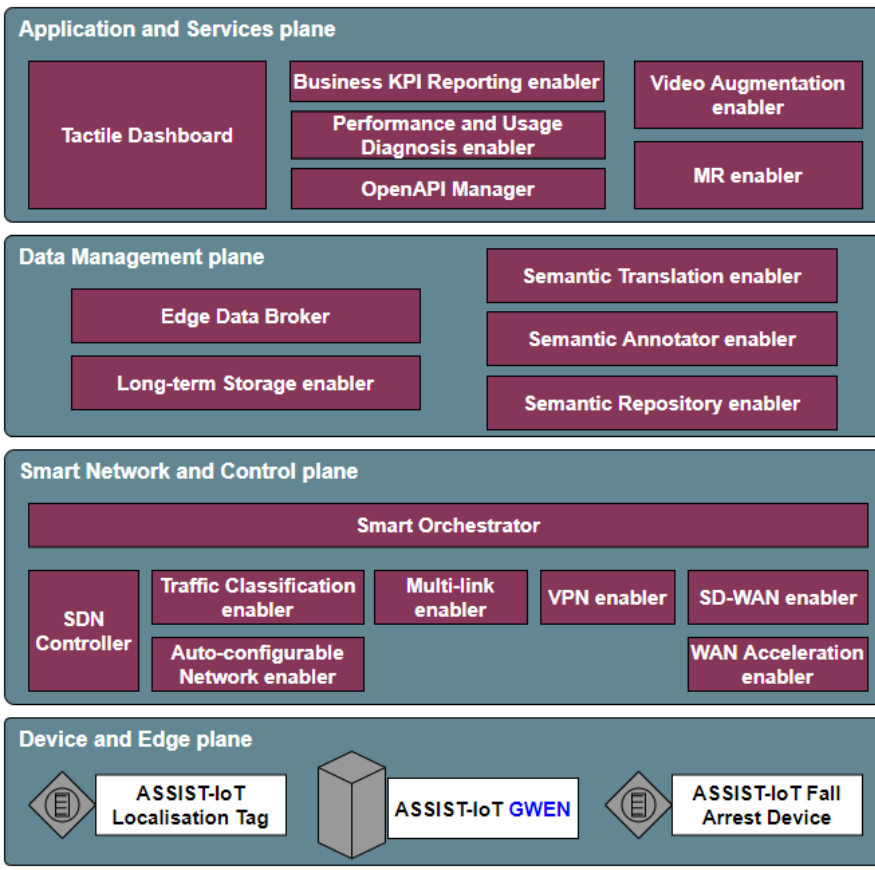

Figure 3.3 Functional view summary representation.

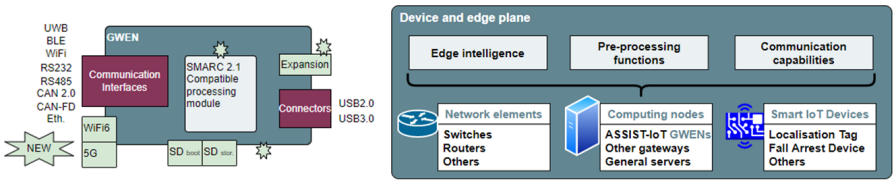

Figure 3.4 ASSIST-IoT's GWEN (left); infrastructure elements and functional blocks of the device and edge plane (right).

devices, sensors, and actuators, and integrating them with the rest of the architecture, and (ii) offering a set of (hardware/software/firmware) features that help realizing intelligence, pre-processing, and communication operations at the edge (e.g., from GPUs/FPGAs to AI frameworks, local processing functions, protocol adapters, specific extensions for communication protocols like LoRa, and ZigBee; whatever needed for a specific system realization). As a matter of fact, the project has designed (and developed) its own gateway/edge node (GWEN), which, apart from the required processing and storage resources, implements common interfaces and baseline functions needed for the RA to perform (firmware, OS, container engine, K8s, and pre-installed plugins). It is modular, meaning that features can be extended via expansion boards and SD slots. Regarding actual enablers, none has been defined in advance, as they are expected to be tightly coupled to the actual needs of the use cases addressed by a given system realization. Figure 3.4 presents a high-level schema of the GWEN, as well as the infrastructure and functional blocks of the plane.

3.3.1.2 Smart network and control plane

This plane hosts different communication and orchestration features, for deploying and connecting virtualized functions. A set of enablers have been selected as relevant (or, at least, interesting) for NG-IoT system realizations, grouped into four functional blocks: "orchestration," "software-defined networks," "self-contained networks realization," and "access networks management," as depicted in Figure 3.5. The smart orchestrator, designed considering ETSI MANO specifications [10], is the main enabler of the plane. It is in charge of controlling the lifecycle of other enablers (network and non-network-related) to be deployed on top of the virtualized infrastructure, managed by K8s, selected for being *de facto* standard toward cloud-native

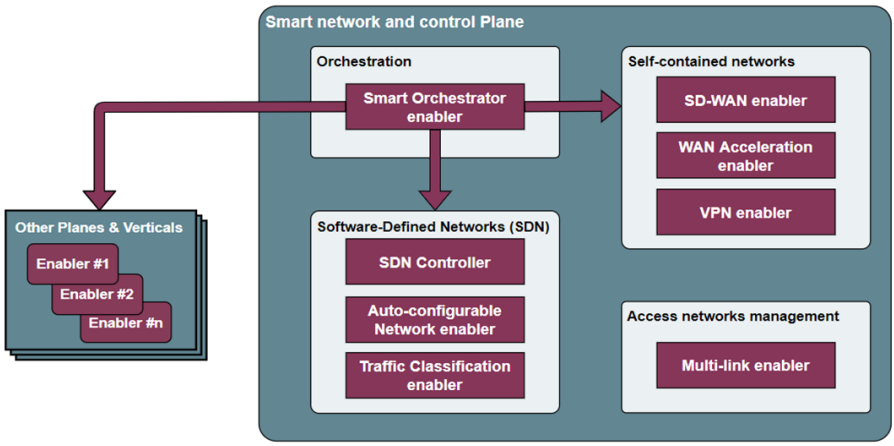

Figure 3.5 Enablers and functional blocks of the smart network and control plane.

paradigm [11]. Additionally, it performs intelligent resource scheduling operations (i.e., selecting the optimal place of the continuum for deploying an enabler) and applies communication rules and encryption among enablers.

Different adoption strategies for SDN in NG-IoT have been studied [12]. This RA presents a block devoted to it, consisting of (i) a controller, which manages the underlying SDN-enabled network; (ii) an auto-configurable network enabler, which acts over the controller to set policies optimally; and (iii) a traffic classification enabler, which identifies the type of traffic so networking rules are applied properly. This functional block is complemented by the programmatic rules that the orchestrator applies over the virtualized network.

The functional block related to self-contained networks includes enablers that provision secured channels over public or non-trusted networks. Three enablers are envisioned: one for establishing VPN tunnels for connecting isolated devices to a managed network and two for implementing SD-WAN, which follows a controller-agent schema to connect delocalized networks and to enable firewalling or application-level prioritization functions. Finally, within the access network management block, a multi-link enabler has been formalized, providing mechanisms for bonding different access networks to work as a logical, single one, thus bringing redundancy and reliability features.

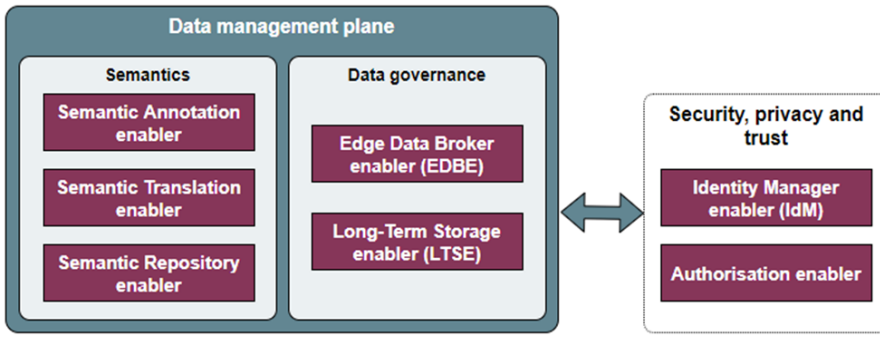

Figure 3.6 Enablers and functional blocks of the data management plane.

3.3.1.3 Data management plane

Traveling over the network layer, data will be shared, processed, and stored to be later on consumed by business/administrator applications and services. Five enablers are defined in this plane, separated into semantics and data governance blocks as one can see in Figure 3.6, supported by security, privacy, and trust mechanisms provided by enablers of such verticals. Data governance enablers include (i) a long-term storage enabler (LTSE), offering a dynamic, distributed space for highly available data persistence, accessible via API; and (ii) an edge data broker enabler (EDBE), key element for realizing data pipelines (Section 3.3.5), providing mechanisms for distributing data, filtering, and alerting, following scalable publication−subscription schemas aligned with current IoT trends.

Complementary to the data storage and transportation, a semantic framework is proposed to process (streaming and bulk), share, and present data. This framework includes (i) semantic annotation, for lifting data to fit a specific semantic format; (ii) semantic translation, for mediating between data that follow different ontologies or data models; and (iii) semantic repository, as a "hub" of data models, schemas, and ontologies, complemented with relevant documentation.

3.3.1.4 Applications and services plane

The upper plane is devoted to provide human-centric, user-friendly access to data, for both administrators and end users, including externals to a system realization. Three functional blocks have been identified (see Figure 3.7), with a set of enablers that, as occurs with the rest of the planes, could be extended.

The enablers allocated within the dashboards functional block are (i) the tactile dashboard, i.e., the main entry point to the system that will be used by administrators and users, with spaces accessible according to the provided credentials. It can provide mechanisms to add graphical interfaces to such application; (ii) the business KPI enabler, which allows administrators to prepare representation figures for metrics and indicators to be consumed by the stakeholders; and (iii) the performance and usage diagnosis enabler, which collects both system and enablers performance-related metrics.

Then, for tactile applications, two enablers are defined: the mixed reality and the video augmentation enabler. The former offers mechanisms for human-centric interaction, based on real-time and visual feedback and data from/to the system and the environment. It works jointly with hardware equipment; so the provided features and representations are largely influenced by it. The latter performs real-time computing vision functionalities over images or video streams (particularly, object detection and recognition), with recommended support from acceleration hardware. It should be highlighted that these enablers focus on particular augmentation capabilities, and additional ones could be thought for providing additional features from tactile and/or haptic interfaces.

Lastly, the OpenAPI manager allows exposing and monitoring API interfaces so that users and third-party systems can consume deployed enablers of the system. This enabler should be properly integrated with security enablers (i.e., identity manager and authorization server) to ensure that only rightful users/systems have access and to expose documentation to ease their respective usage.

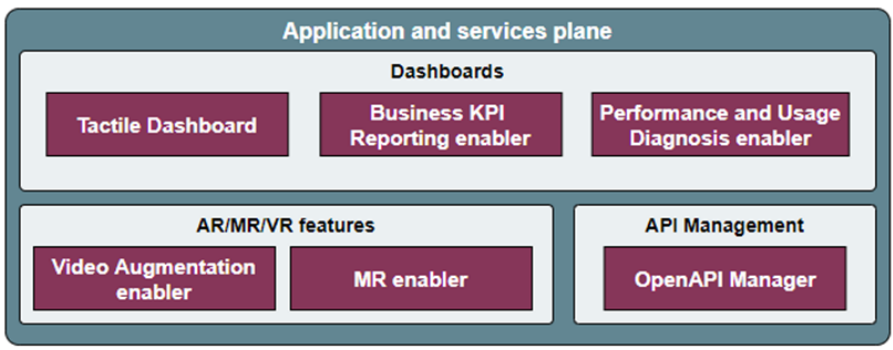

Figure 3.7 Enablers and functional blocks of the applications and services plane.

3.3.2 Node view

This view presents structural guidelines and recommendations to provision nodes that can be later on leveraged in NG-IoT systems. The provided data can be useful for stakeholders like hardware developers, edge devices, or gateway providers, as well as developers and maintainers of an ASSIST-IoT system. Nodes should not be understood as physical equipment but as a virtualized resource (e.g., a powerful physical server might host several virtual nodes); they can be placed on different tiers of the continuum (edge, fog, and cloud) and will likely have varying computing capabilities. Thus, to be ready as a node, a set of pre-requisites must be fulfilled:

- A K8s distribution must be installed (kubeadm and K3s encouraged), with a compatible container runtime (Docker recommended) and a Linux OS (the latter is not needed if K8s on bare metal is installed).
- A set of plugins for managing packages (Helm), storage classes (OpenEBS), and local and multi-cluster networking (Cilium). These specific plugins mentioned are not mandated but are compatible with the enablers developed in ASSIST-IoT.

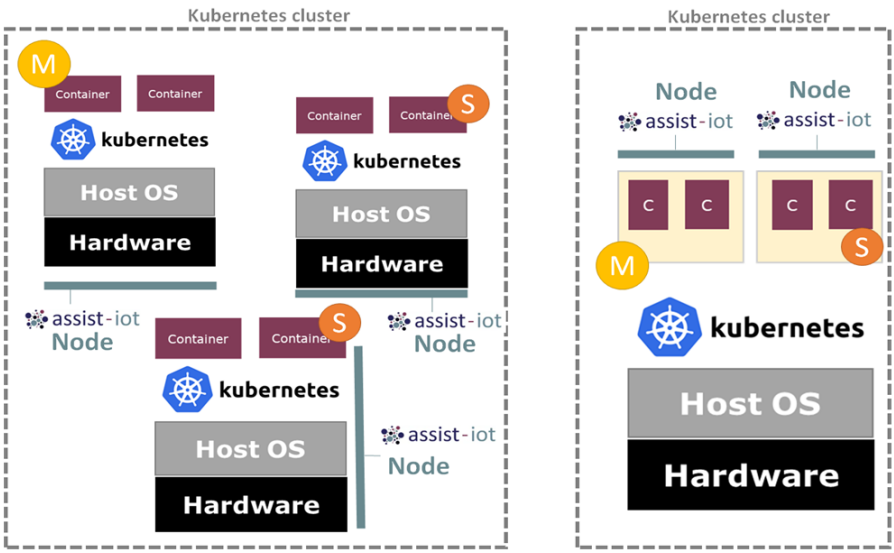

Figure 3.8 ASSIST-IoT node concept and spots.

3.3.3 Development view

This view aims at offering some guidelines and recommendations to ease the design and development of enablers, useful for developers and maintainers. Despite the project developing enablers for the different planes and verticals, it is possible that future systems rooting from ASSIST-IoT might require additional features. Enablers are to be designed respecting the principles of the RA, following some common conventions and considering the DevSecOps methodology [13] from the project. In particular, we have the following:

- **Virtualization**: Enablers should be deployable independently, and each of its components (inner functions) should be delivered in containers.
- **Encapsulation**: Enablers can communicate between each other, and with external clients, via explicitly defined and exposed interfaces (typically but not limited to REST APIs). Enablers' internal components are not exposed by default.
- **Manageability**: Enablers should expose a set of basic endpoints and logs (through **stderr** and **stdout** interfaces), following standard conventions for providing their status (e.g., with HTTP response codes and considering all the inner components), version (considering SemVer specifications), API documentation (Open API specifications), and relevant metrics (Prometheus-compatible format encouraged).

The process for designing and developing enablers is depicted in Figure 3.9, consisting in six main steps: (i) definition and formalization of requirements, considering its key features, main (software and hardware) constrains, and applicable use cases; (ii) breakdown of internal components, including its exposed interfaces and its internal communication; (iii) initial design of the endpoints to expose, including the manageability ones previously mentioned (i.e., /health, /version, /api, /metrics); (iv) baseline technologies and programming languages to leverage, avoiding reinventing the wheel while focusing on decentralization and resource optimization; (v) if data are involved, (sector, regional, and national) privacy regulations and ethical aspects should be considered; and once development starts, (vi) DevSecOps methodology should be followed [13], ensuring that the final result is secure by design. As additional tips, the use of verified container images, initial proofs of concept considering Docker compose tool before moving to K8s, and the provisioning of CI/CD pipelines for automating DevSecOps processes, including unit, functional, and security testing, are encouraged.

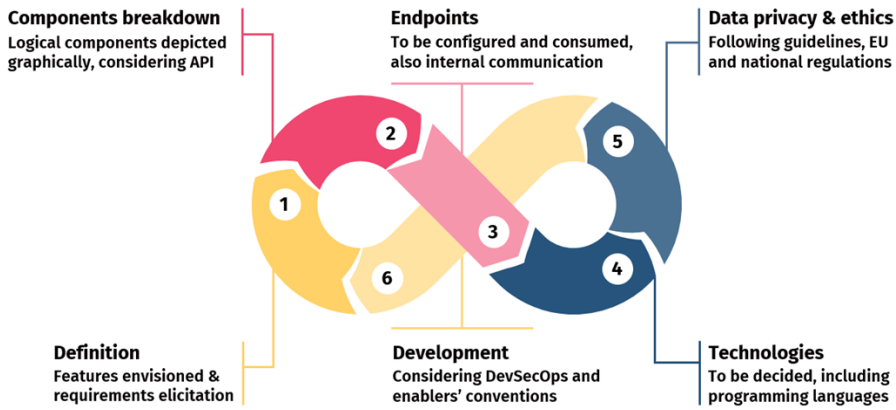

Figure 3.9 Continuous enablers' development process.

3.3.4 Deployment view

The deployment view addresses different concerns that can be useful, especially for network and system administrators (as well as developers and maintainers), such as hardware provisioning (computation nodes and networking elements), K8s setup, and the deployment of enablers and their integration to address a use case or business scenario.

3.3.4.1 Infrastructure and Kubernetes considerations

The computing continuum can be decomposed into tiers, each of them consisting of a set of nodes, extending from end devices (smart IoT devices, and MR/AR/VR interfaces) to edge (with one or multiple levels) and cloud, if needed. Being K8s strongly encouraged as virtualized infrastructure manager, the underlying connection among nodes must be IP-based (with the exception of the access network, as interfaces between end devices and edge gateways might involve other forms of communication, e.g., LPWAN and BLE). A generic topology is presented in Figure 3.10. A system topology design will strongly depend on the business scenario, security, and decentralization aspects, as well as economic reasons.

Regarding K8s, computing nodes are grouped into clusters, where at least one acts as **master**, in charge of control plane actions, and the rest as **workers** (which execute workloads). Some aspects that should be considered for a proper implementation include: (i) clusters should consist of nodes with similar performance; (ii) a multi-tier topology suggests having a master in control, rather than a master in charge of different tiers; (iii) if new nodes are

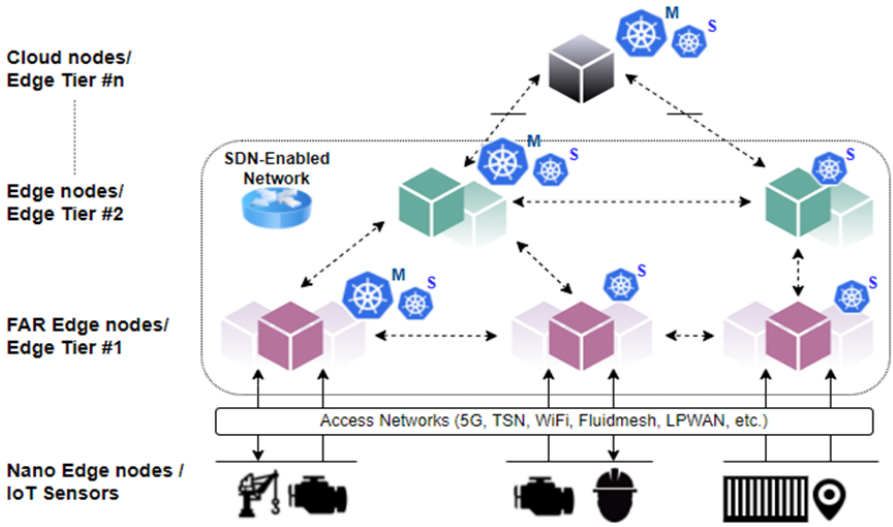

Figure 3.10 Generic topology of an NG-IoT system.

added to an existing tier, better to do it as workers to avoid devoting more resources to control plane tasks (unless done for high availability strategies); and (iv) K8s management is outside the scope of the architecture, despite requiring some minimum requirements (e.g., plugins and add-ons) that should be provisioned by network administrators. In any case, best practices for K8s security are encouraged [14] for any system realization.

3.3.4.2 Enablers deployment

The main and recommended tool for instantiating enablers over the managed computing continuum is the smart orchestrator, considered an essential enabler of the system. After provisioning the infrastructure, network, and K8s clusters, the latter must be registered in the orchestrator (individually, or as part of a group of clusters), and from this moment, a platform administrator can deploy enablers over them, either manually or automatically based on a desired policy. The orchestrator developed within ASSIST-IoT considers Helm as the packaging technology, but other formats (custom-made based on K8s manifests, Juju, or Kustomize) could have been used (and thus an orchestrator designed based on the selected one/s).

It is perfectly possible to deploy enablers directly over the managed continuum via Helm commands or utilizing third-party management software like Rancher Fleet. However, some of the additional features provided by the orchestrator, such as automatic application of networking rules, policy-based

automatic scheduling, and mobility mechanisms for clusters without public or/nor fixed IP addresses are no longer supported.

Regarding enablers integration, ASSIST-IoT does not force that every expected or possible integration be realized, as there are just many artifacts, data schemas, technologies, etc. Still, when deemed necessary, some enablers' implementations made within the scope of the project have been integrated. On the one hand, some cases were evident, like the semantic suite (Section 3.3.1.3), the federated learning enablers (Section 3.4.3.2), or the security enablers (Section 3.4.3.1) related to identity management and authorization. On the other hand, other interactions were evaluated and, in some cases, integrated requiring higher or lower effort, like the OpenAPI manager with identity management, the manageability enablers with the smart orchestrator, or the tactile dashboard with BKPI and PUD enablers, among others. Besides, manageability enablers provide some mechanisms to provision agents within the right spot of the continuum as "integration bridges," providing translation of transport protocols (e.g., MQTT to MQTT) and basic data formatting capabilities.

3.3.5 Data view

This view, useful for data engineers as well as developers and maintainers, provides an overview of the flow of data within a system, with respect to their collection, processing, and consumption, specifying the actions made by the enablers (and other artifacts) over them. ASSIST-IoT introduces **data pipelines** as an abstraction design that represents such flows, avoiding information related to the underlying hardware infrastructure or network-related enablers. In essence, these pipelines present a linear sequence of steps where data are transmitted between data processing elements, from a given source/s (e.g., services, endpoints, devices, sensors, and outputs from another pipeline/s) to an output/s or sink/s (e.g., database, dashboard, log gatherer, source of following data pipeline/s, or simply deleted). Data travel as messages, through different paths, and having a specific format and content. An example of data pipeline is presented in Figure 3.11. In such representations, data sources, protocols and payload types, as well as enablers and services involved should be easily identifiable, accompanied with dedicated textual explanation when needed but trying to keeping them readable (i.e., avoiding gratuitous details). In any case, these representations do not aim to substitute other typical, dedicated UML or similar diagrams, but rather complement them.

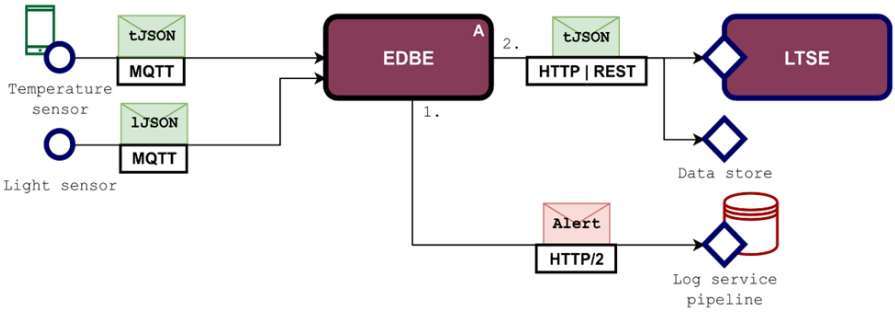

Figure 3.11 Data pipeline example.

3.4 Verticals

Verticals represent or provide NG-IoT properties and capabilities that (should) exist on different planes, either in an independent way or requiring cooperation from them. Five verticals have been identified as crucial for the development of NG-IoT systems, namely self-*, interoperability, scalability, manageability, and security, privacy, and trust. In some cases, capabilities are implemented through dedicated enablers (see Figure 3.12), while in others, they are the result of the design principles embraced. As with planes, some of them might not be needed for addressing certain use cases, and

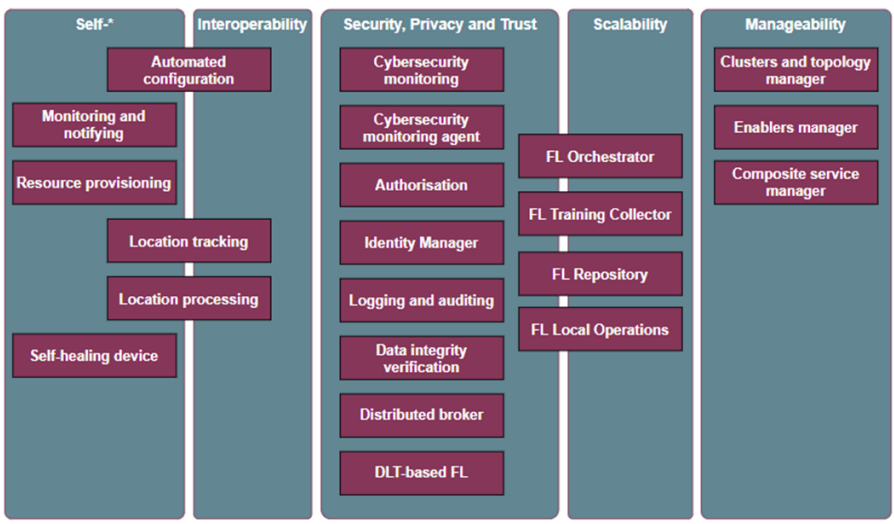

Figure 3.12 Enablers addressing verticals.

envisioning additional ones for bringing unavailable capabilities to a system is not precluded.

3.4.1 Self-*

This vertical refers to enablers implementing system autonomy capabilities, able to make intelligent decisions, where humans just manage them by policies (or just installing them) rather that manually acting over the involved mechanisms. According to IBM, eight conditions should be fulfilled to consider a mechanism as self-* [15].

Particularly, five enablers have been identified to extend the capabilities already present due to architecture design choices. (i) Self-healing enabler: considering K8s provides a set of healing mechanisms over managed services, this enabler extends them to the node itself, collecting data (e.g., battery and CPU usage, memory access, and network state), evaluating them, determining, and applying the optimal healthy remediation process. (ii) Self-resource provisioning: K8s also provides resource autoscaling mechanisms for services – still, these are static, meaning that once their behavior is set, they can only be changed manually. This enabler develops models of other deployed enablers to forecast their behavior, modifying the performance of these scaling mechanisms on-the-fly, without human intervention. (iii) Location tracking and (iv) processing enablers: these enablers work together for bringing contextual location data, with dedicated hardware and firmware extensions, and providing configurable and flexible geofencing capabilities based on such data, using either batch or streaming approaches. (v) Monitoring and notifying enabler: responsible for monitoring devices and notifying malfunctioning incidents, ensuring that telemetry data are sent and presented while validating its own performance. (vi) Automated configuration enabler: it allows users to define requirements (resources) needed to meet specific functionalities, and reactions over external actions or change of the pool of resources, all in an abstract way. In case of limited resources, the system will automatically decide which functionalities are kept, modifying existing configurations and emitting related messages and logs.

3.4.2 Interoperability

This vertical represents the ability of systems, devices, and/or applications to communicate together on the same infrastructure or on another while roaming. The IoT field, and by extension, NG-IoT systems, are heterogeneous

in terms of hardware, data, and applications, and, hence, mechanisms that ease devices connection, data sharing, and service communication are needed to facilitate the adaptation of novel technologies and services. Interoperability is present at different levels, rather than addressed as enablers: (i) technical, when two or more information and communication technologies are able to accept data from each other; (ii) syntactic or structural, when two systems communicate and exchange data using compatible formats and protocols; and (iii) semantic, which entails applications and systems sharing the same understanding of the exchanged data.

Although any enabler has been directly assigned to this vertical, it is present through different enablers of the architecture. For instance, SDN-related enablers allow governing a networking infrastructure with hardware from different vendors autonomously, based on policies; the semantic suite brings processing and translation capabilities to store data and share them following specific ontologies or data models, enabling effective cooperation among IoT artifacts; also, the use of smart contracts coming from DLT-related enablers of the next vertical, allowing metadata management and non-repudiation from different sources or systems; in self-* localization tracking, various geospatial data sources can be combined (UWB and GPS); in federated learning (FL) suite (Section 3.4.3.2), enablers provide mechanisms to run on different clients and to accommodate training data from different formats, etc. Besides, although an interoperability suite like that in [16] could have been defined, its usage would require extensive knowledge of the several available mechanisms, and, hence, implementing them as independent enablers when needed is preferred instead.

3.4.3 Security, privacy, and trust

This vertical should be considered meticulously, as perceiving a system as unsecure, untrusted, or privacy-disrespectful would destine it to fail. Many mechanisms can be grouped under this vertical; the provided content is extended in [17].

3.4.3.1 Security

This pillar involves several aspects, from good practices for development and data access by design (e.g., DevSecOps) to enablers that provide confidentiality, integrity, availability, authentication, and authorization. Here, the following enablers have been defined:

- **Identity manager (IdM) and authorization enablers**: They offer (i) access control based on user/devices/system identity, and (ii) authorization over protected resources. The ASSIST-IoT RA depicts a decentralized approach, where decision-making features can be moved from a central point to distributed endpoints, sharing security policies (previously set by an administrator) to apply.
- **Cybersecurity monitoring enabler, and monitoring agents**: The central enabler consolidates the data collected by the agents distributed through the continuum to provide cybersecurity awareness, visibility, and response against detected threats.

Besides, additional considerations are presented. First, the OpenAPI management enabler from the applications and services plane includes a gateway, envisioned as primary access mechanism for (HTTP) third-party access. In this way, a single point should be exposed, secured, and documented, reducing the number of attack surfaces; and hence the IdM and authorization enablers must be integrated with this one. Second, when MQTT is the main communication protocol, different security mechanisms should be assessed, especially when the network is not considered secured or trusted. These include protection at (i) network level, providing tunnels between clients and brokers; (ii) transport level, encrypting data using SSL/SSL and certificates; and (iii) application level, considering user-password credentials (or Access Control List files) to grant or deny access, including the possibility of allowing nodes to publish or be subscribed only to specific topics.

3.4.3.2 Privacy

Privacy aims at protecting the information of individuals or private data from exposure in NG-IoT environments. In ASSIST-IoT, a set of rules for preserving it during development has been made, and, in addition, an FL suite for training ML algorithms in a decentralized environment has been designed, in which actual data is not exchanged but only the trained models. This suite is composed of (i) the central orchestrator, responsible for declaring the FL pipeline and control the overall lifecycle, including job scheduling and error handling; (ii) a repository, providing storage for ML algorithms, aggregation approaches, and trained models, supporting the rest of the enablers; (iii) local operations component, installed in the distributed nodes to perform data format verification and transformation, local model training, and inference, among other tasks; and (iv) training collector, which aggregates the models updated by the managed nodes and redistributes the combined model.

3.4.3.3 Trust

Trusts represents the level of confidence of the devices, data, or system of an ecosystem [18]. The RA does not delve on best practices and recommendations related to this, as there exist dedicated projects focusing on it [19]; still, it defines a set of enablers, based on DLT, for easing the implementation of trusted decentralized ecosystems: (i) logging and auditing, for storing critical actions and having a trusted source of truth; (ii) data integrity verification, based on hashed data; (iii) distributed broker, to facilitate data sharing of devices from different edge domains; and (iv) DLT-based FL enabler, an auxiliary component of the FL suite to manage ML contextual information, preventing any data alteration. Before using them, it is important to decide which events and data are critical, as data are replicated on the ledgers, and storing many data can cause performance issues.

3.4.4 Scalability

The RA pretends enabling elastic implementations, where hardware (node and system level), software, and services can be scaled up/down as seamlessly for adopters as possible. In ASSIST-IoT, this vertical does not result from the action of specific enablers, but rather due to design choices made and the use of K8s (or any similar container orchestrator framework), primarily. In this RA, we can find first hardware scalability, contemplating (i) processing, having nodes with constrained resources, like PLCs, to high-performance servers with large GPU arrays; (ii) storage, from simply flash chips to large arrays of storage clusters; (iii) network interfaces, with nodes having a single (wired and wireless) interface to others having several of them, with aggregation capabilities; and (iv) system, considering small-size to large-size, decentralized topologies, including possible business scenarios with thousands of clusters. The GWEN also incorporates mechanisms fostering this scalability dimension, through the expansion boards and modules through which storage, access networks, and computation capabilities could be expanded, if needed.

Besides, software scalability is also crucial, involving not just the deployment of services and applications but also the optimal scheduling of the managed resources. Regarding the latter, the use of K8s distributions, along with the smart orchestrator and resource provisioning enabler, guarantees that once a (software) feature is deployed over the infrastructure as an enabler, it (i) has its required resources, and (ii) can be scaled up/down automatically based on current and forecasted usage. Besides, being a microservices-based

RA and using containers as virtualization paradigm, features are decoupled by nature and can be developed and integrated quite easily, thanks to their respectively exposed interfaces. Additional guidelines and best practices can be found in [17].

3.4.5 Manageability

The last vertical responds to concerns related to the management of the overall system and the required enablers. Tools are needed to register and manage (large volume of) K8s clusters and enablers, including mechanisms to (i) detect and inform about faults, (ii) allow configuration options of the enablers to be deployed, (iii) enabling processes for storing and sharing logs and metrics of enablers and clusters, and everything (iv) in a secure and user-friendly manner. Along with the implementation of common endpoints for enablers depicted in Section 3.3.3, the following manageability enablers have been defined:

- **Clusters and topology manager**: It allows to register/delete clusters to the system, ensuring that they are working properly and providing graphical data of their distribution and hosted enablers.
- **Enabler manager**: Eases the management of enablers, in a graphical way, from the registration of enabler repositories to their instantiation (also configuration), logs consumption, and deletion.
- **Composite services manager**: It eases the flow of data between enablers, by provisioning interoperability agents that provide protocol (e.g., MQTT-HTTP) and basic payload translations. It includes a graphical interface to configure these agents, which are then deployed optimally within the continuum.

3.5 Conclusion

This chapter describes the reference architecture developed within the framework of H2020 ASSIST-IoT project, following cloud-native principles adapted to the edge−cloud computing continuum for next-generation, tactile IoT, providing a set of guidelines, best practices, and recommendations. It is based on microservices, using containers and Kubernetes as main virtualization technologies, as well as coining the concept of an enabler. Functionalities will be delivered in the form of these, which will belong to one of the planes (device and edge, smart network and control, data management, and application and services) or verticals (self-*, interoperability, scalability,

manageability, and security, privacy, and trust) of the multi-dimensional approach of the architecture.

Since providing such information in an all-encompassing model would hinder its comprehension, the architecture has been divided in five separated views, namely functional, node, development, deployment, and data, each of them of utility for a particular group of stakeholders. It should be highlighted that this document, as well as the outcomes presented as the project's deliverables and code, is a reference for building tactile IoT systems, and, thus, it should be taken as such, rather than a platform ready to be deployed and used without performing any tailoring to the targeted business scenario.

Acknowledgements

This work is part of the ASSIST-IoT project, which has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 957258.

References

[1] I. Lacalle et al., "Tactile Internet in Internet of Things Ecosystems," in International Conference on Paradigms of Communication, Computing and Data Sciences (PCCDS 2021), 2021, pp. 794–807.
[2] ISO/IEC/IEEE 42010, "ISO/IEC/IEEE 42010 - Systems and software engineering - Architecture description," 2011.
[3] N. Rozanski and E. Woods, Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison Wesley, 2011.
[4] M. W. Maier, D. Emery, and R. Hilliard, "Software architecture: Introducing IEEE standard 1471," Computer (Long. Beach. Calif)., vol. 34, no. 4, pp. 107–109, 2001.
[5] M. Richards, Software Architecture Patterns. O'Reilly Media, 2015.
[6] CREATE-IoT Project, "D6.3. Assessment of convergence and interoperability in LSP platforms," 2020.
[7] OpenFog Consortium, "OpenFog Reference Architecture for Fog Computing," 2017.
[8] AIOTI WG Standardisation, "High Level Architecture (HLA) Release 5.0," 2020.

[9] P. B. Kruchten, "The 4+1 View Model of Architecture," IEEE Softw., vol. 12, no. 6, pp. 42–50, 1995.
[10] ETSI, "GS NFV-MAN 001 Network Functions Virtualisation (NFV); Management and Orchestration," 2014.
[11] A. Fornes-Leal et al., "Evolution of MANO Towards the Cloud-Native Paradigm for the Edge Computing," in International conference on Advanced Computing and Intelligent Technologies (ICACIT 2022), 2022, vol. 914, pp. 1–16.
[12] C. Lopez et al., "Reviewing SDN Adoption Strategies for Next Generation Internet of Things Networks," in International Conference on Smart Systems: Innovations in Computing (SSIC), 2021, vol. 235, pp. 619–631.
[13] O. López et al., "DevSecOps Methodology for NG-IoT Ecosystem Development Lifecycle - ASSIST-IoT Perspective," J. Comput. Sci. Cybern., vol. 37, no. 3, pp. 321–337, 2021.
[14] The Kubernetes Authors, "Kubernetes documentation - Security." Online: https://kubernetes.io/docs/concepts/security/.
[15] IBM, "An architectural blueprint for autonomic computing," 2005.
[16] C. I. Valero et al., "INTER-Framework: An Interoperability Framework to Support IoT Platform Interoperability," Interoperability of Heterogeneous IoT Platforms. Springer, pp. 167–193, 2021.
[17] ASSIST-IoT Project, "D3.7 - ASSIST-IoT Architecture Definition – Final," 2022.
[18] NIST, "Internet of Things (IoT) Trust Concerns," 2018.
[19] TIoTA Alliance, "Trusted IoT Alliance Reference Architecture," 2019.