SEMIoTICS
Deliverable D2.5.
SEMIoTICS High-Level Architecture (Final)

Deliverable release date	31.12.2019
Authors	1. Ermin Sakic, Darko Anicic, Arne Broering (SAG) 2. Eftychia Lakka, Nikolaos Petroulakis, Emmanouil Michalodimitrakis (FORTH) 3. Jordi Serra, David Pubill, Angelos Antonopoulos, Christos Verikoukis (CTTC), 4. Konstantinos Fysarakis, Iasonas Somarakis (STS) 5. Danilo Pau, Mirko Falchetto (ST), 6. Domenico Presenza (ENG), 7. Felix Klement, Korbinian Spielvogel, Henrich C. Pöhls (UP), 8. Łukasz Ciechomski, Michal Rubaj, Piotr Kowalski, Marcin Zawadzki, Łukasz Kempiński, Urszula Rak (BS) 9. Prodromos-Vasileios Mekikis, Kostas Ramantas (IQU) 10. Urlich Hansen (BWC)
Responsible person	Łukasz Ciechomski (BS)
Reviewed by	Urszula Rak (BS), Łukasz Ciechomski (BS), Eftychia Lakka, Nikolaos Petroulakis, Emmanouil Michalodimitrakis (FORTH), Konstantinos Fysarakis (STS)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, Ermin Sakic, Mirko Falchetto, Domenico Presenza, Verikoukis Christos) PCC Members (Vivek Kulkarni, Nikolaos Petroulakis, Christos Verikoukis, Georgios Spanoudakis, Domenico Presenza, Danilo Pau, Joachim Posegga, Darek Dober, Kostas Ramantas, Ulrich Hansen)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Table of Content

Acronyms Table	
Acronym	**Definition**
2FA	Two-Factor-Authentication
AI	Artificial Intelligence
API	Application Programmable Interface
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BAN	Body Area Network
BLE	Bluetooth Low Energy
BO	Backend Orchestrator
CAN	Controller Area Network
CAN-BUS	Controller Area Network Protocol
CAPEX	Capital Expenditures
CB	Context Broker
CC	Context Consumer
CI/CD	Continuous Integration / Continuous Delivery
CLoE	CLoud of Engineering
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CP	Context Producer
DB	DataBase
FW	Firmware
GE	Generic Enabler
GW	Gateway
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
IHES	Intelligent Heterogeneous Embedded Sensors
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process communication
IPv4	Internet Protocol version 4
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linking Data
LD	Linking Data
MAC	Media Access Control
MANO	Management and Orchestration
MCU	Micro Controller Unit
MQTT	Message Queuing Telemetry Transport
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
NFVO	NFV orchestrator
NGSI	Next Generation Service Interfaces

NGSIv2	Next Generation Service Interfaces version 2
NS	Network Services
NSO	Nework Service Orchestration
O&M	Operations and Maintenance
ODL	Open Daylight
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PLC	Power Line Controller
QoS	Quality of Service
REST	Representational State Transfer
RO	Resource Orchestrator
SARA	Socially Assistive Robotic Solution for Mild Cognitive Impairment or mild Alzheimer's disease
SBI	Southbound Interface
SCADA	Supervisory Control and Data Acquisition
SDN	Software-Defined Networking
SFC	Service Function Chaining
SM	Security Manager
SPDI	Security, Privacy, Dependability, and Interoperability
SSC	SEMIoTICS SDN Controller
TCP	Transmission Control Protocol
TD	Thing Description
UC	Use Case
UI	User interface
URL	Uniform Resource Locator
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Function
VNFFG	Virtual Network Function-Forwarding Graphs
vSwitch	Virtual Switch
VTN	Virtual Tenant Networks
W3C	World Wide Web Consortium
WP	Work Package
WoT	Web of Things

1 INTRODUCTION

This deliverable is the final output of Task 2.4 “SEMIoTICS Architecture Design”. It provides a final specification of the overall reference architecture and a base-line specification of the interfaces and functionalities of the core components of the SEMIoTICS framework.

The presented final version of the architecture of the pattern-driven SEMIoTICS framework aims to address such challenges of current implementation and deployment stack of IoT applications as dynamicity, scalability, heterogeneity, and end-to-end security/privacy. Specifically, the functional components of the proposed architecture are presented to provide an overview of the appropriate realization mechanisms. Finally, the architecture is mapped in three use cases (two verticals mapped in the areas of energy and health care and one horizontal in the area of intelligent sensing) in order to demonstrate its applicability in different IoT enabling platforms, types of smart objects, devices and types of networks.

The specification and logical composition of the architecture have been built upon the general and Use Case-specific requirements identified during the project. Additionally, considering the initial architecture given in the project proposal, three layers (Field Level, SDN/NFV Orchestration Layer and Application Orchestration Layer) have been included in the SEMIoTICS architecture. Additionally, said layered approach corresponds with the research performed within the project, leading to the vision of the architecture presented in this document.

Given its key role in the project, this architecture task features significant interplay with ongoing tasks within WP2, WP3, and WP4, also featuring a direct connection with WP5 (focusing on integration and demonstration). While the development of the architectural components is planned for WP3 and WP4, final integrations and integrated demos are planned for WP5 as per DoA.

Due to the complexity of the project itself and its goals, the necessity of integration of a significant number of requirements defined in T2.1 and T2.2, and the need to identify interactions between generic framework components that are also able to support diverse use cases, the architecture definition has consumed significant effort. A series of face to face and online workshops triggered many discussions on the framework architecture composition. Since work on architecture definition started after WP3 and WP4 were already initiated, specific components and functionalities were identifiable. Once the component identification had been finalized, thorough analysis followed considering the requirements identified and work being delivered in WP3 and WP4 to ensure all necessary elements are included, without omissions of component needed to fulfill the project’s requirements. Further architecture adjustments and modifications, which were necessary when entering into the intensive development phase, were carried out following direct communications, regular status meetings, and PTC meetings. The differences between the draft and final versions of the architecture are described in this document for a clear understanding of the reader.

The deliverable is structured as follows:

- Section 2 describes the methodology and major steps taken to design the final version of the architecture.
- Section 3 presents an overview of the SEMIoTICS architecture, including static architecture, high-level interaction between the components diagram and descriptions of generic components as building blocks of the framework as per the architectural layer. Moreover, it provides the description of research results and contribution of the project beyond other projects of the consortium.
- Section 4 has been devoted to use-case specific architecture presenting the dynamic architecture for each Use Case hence gives a detailed view of how the SEMIoTICS framework supports each of the Use Cases defined in the project. It also describes how the SEMIoTICS framework could be leveraged for additional use cases
- Section 5 is the validation section where one can see what objectives, KPIs and Requirements are pertinent to the work presented within this deliverable
- Section 6 features the concluding remarks.

1.1 PERT chart of SEMIoTICS

The Task 2.4 is one of the most crucial tasks in SEMIoTICS, related also to most of the other tasks as presented in the PERT chart. Please note that the PERT chart is kept on task level for better readability.

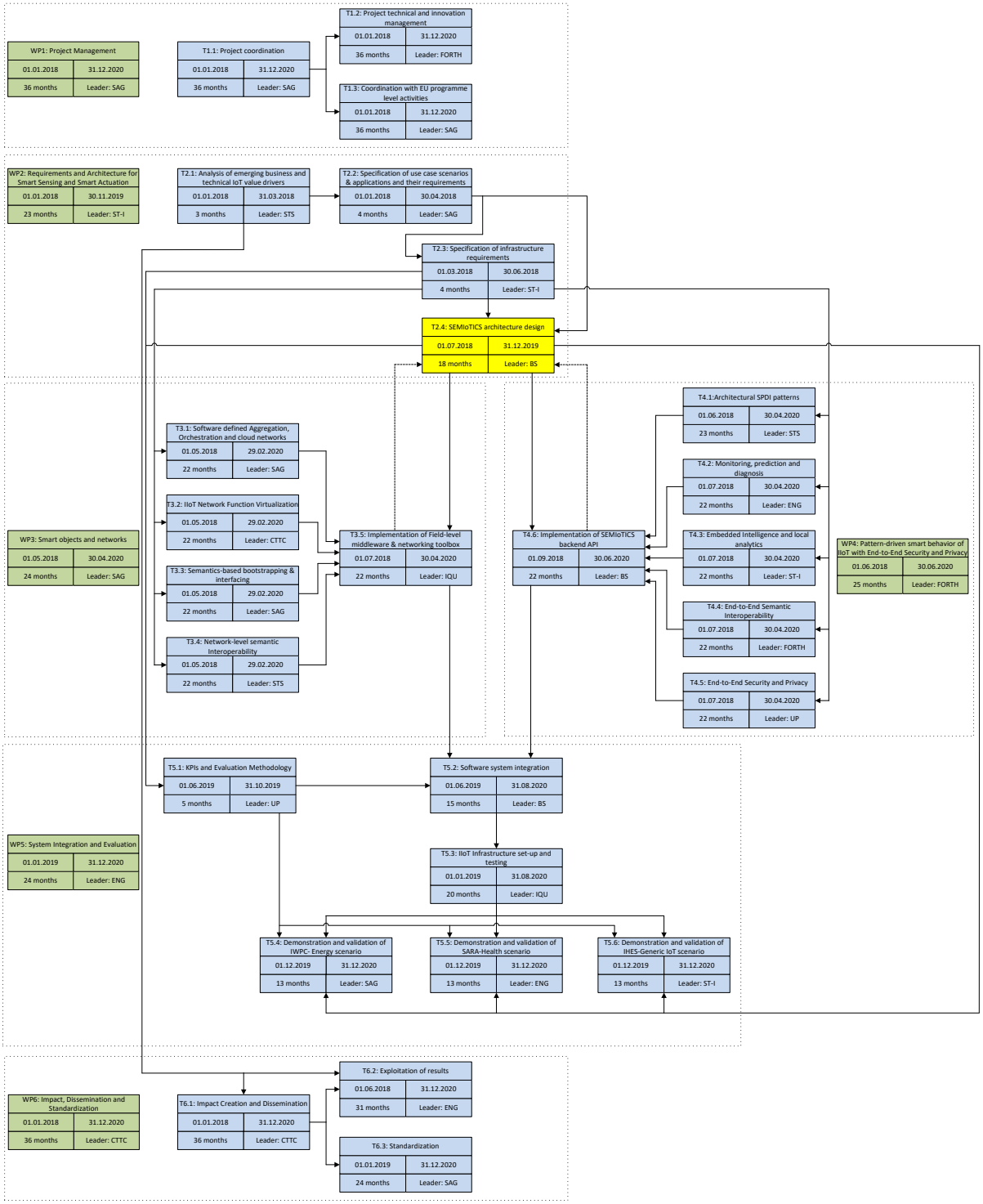


FIGURE 1 PERT CHART

2 METHODOLOGY OF THE ARCHITECTURE CREATION

The development of the SEMIoTICS architectural framework required four major steps in order to finalize it, as visualized in Figure 2.

FIGURE 2 STEPS OF ARCHITECTURE CREATION

- The first step was done in the proposal phase where the envisaged architecture was proposed and described.
- The second step covered the first period of the project in order to cover all implementation building blocks but also achieve the project objectives. The result of this preparation was discussed and agreed in a technical workshop in Barcelona in November 2018. At that stage, most of the components were defined and the owners were assigned.
- The third step was completed during the preparation of deliverable D2.4 which was finalized and submitted in April 2019. The framework was updated based on updates stemming from the detailed description of the project's use cases.
- The final step included the finalization of the framework architecture as presented in this deliverable (D2.5) which is the outcome of the previous steps.

2.1 Envisaged SEMIoTICS Architecture

The main focus of SEMIoTICS is to develop a dynamically configurable and evolvable framework to enable: (a) the integration of heterogeneous smart objects that are available through heterogeneous IoT platforms into IoT applications in a manner that is scalable, secure, privacy-preserving and dependable; (b) the provision of multi-layer intelligence capabilities enabling semi-autonomic smart object behavior and evolution; and (c) the runtime management and adaptation of these objects and the IoT applications that they form to preserve security, privacy, and dependability.

The SEMIoTICS framework is based on the initial vision of the logical architecture of SEMIoTICS framework and how it relates to smart objects, IoT applications, and existing IoT platforms, and how does it map onto a generic deployment infrastructure consisting of private and public clouds, networks, and field devices as depicted in Figure 3. Within the figure, blue boxes show components of the framework that are to be developed by SEMIoTICS; white boxes indicate components of IoT applications managed by the framework. The key role of the SEMIoTICS framework in the IIoT/IoT implementation stack is to support the secure, dependable and privacy-preserving connectivity and interoperability of IoT applications and smart objects used by them, and the management, monitoring, and adaptation of these applications, objects, and their connectivity. The SEMIoTICS vision is articulated around the development of a framework for smart object and IIoT/IoT application management based on trusted patterns, monitoring, and adaptation mechanisms, enhanced IoT centric networks and multi-layered embedded intelligence.

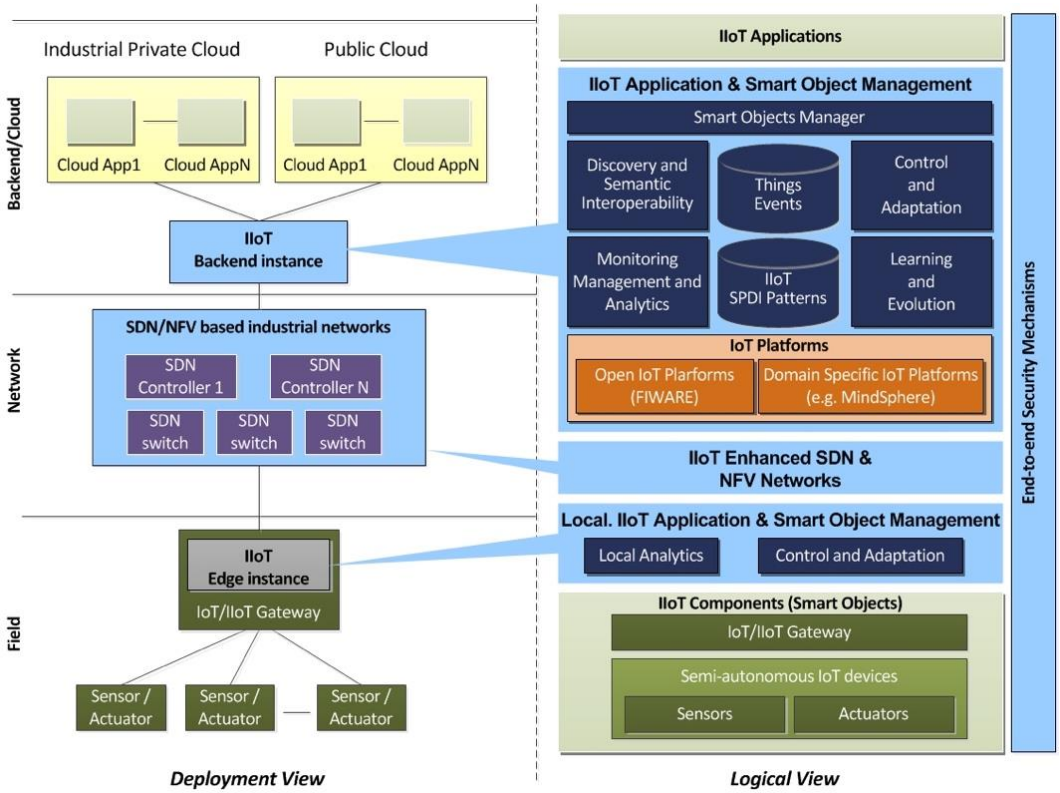


FIGURE 3 ENVISAGED ARCHITECTURE AND DEPLOYMENT OF SEMIOTICS FRAMEWORK

2.2 First draft of SEMIoTICS Architecture

One main goal in the first period of SEMIoTICS was the transition from the conceptual architecture, as described previously, to an open architectural framework including an actual description of components with a single defined goal. In order to provide the required components regarding the construction of the diagram, the outputs and relation between the tasks and project objectives were defined. In addition, the different proposed components as task outputs were considered also in the draft SEMIoTICS architecture, as detailed below:

WP3 - Smart objects and networks

- Task 3.1 Software-defined Aggregation, Orchestration and cloud network
  - **Related Component:** SDN Controller and the containing modules (Resource Manager, Resource Monitor, VTN Manager, Security Manager, Pattern Engine, SFC Manager, Bootstrapping Manager, Clustering Manager, Path Manager)
- Task 3.2 – IIoT Network Function Virtualization
  - **Related Component:** NFV related component (NFV Orchestrator, VNF Manager, VIM) plus the VIM connector located in the controller
- Task 3.3 – Semantics-based bootstrapping & interfacing
  - **Related Component:** Recipe Cooker, Thing Directory
- Task 3.4 – Network-level Semantic Interoperability
  - **Related Component:** Semantic Mediator
- Task 3.5 – Implementation of Field-level middleware & networking toolbox
  - **Related Component:** Semantic API & Protocol Binding

WP4 - Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy

- Task 4.1 – Architectural SPDI patterns
  - **Related Component:** Pattern Orchestrator, Pattern Engine (Backend, SDN, Field)
- Task 4.2 – Monitoring, prediction, and diagnosis

- **Related Component:** Monitoring (Backend, Field),
- Task 4.3 – Embedded Intelligence and local analytics
  - **Related Component:** Local Embedded Intelligence
- Task 4.4 – End-to-End Semantic Interoperability
  - **Related Component:** Semantic Mediator (Backend)
- Task 4.5 – End-to-End Security and Privacy
  - **Related Component:** Security Manager (Backend, SDN, Field)
- Task 4.6 – Implementation of SEMIoTICS backend API
  - **Related Component:** Backend Orchestrator, GUI

Based on the above and the collection of requirements together with the required use case applications/devices, a draft composite diagram of the SEMIoTICS architecture was created containing all the described components and the corresponding functionalities. This architecture was discussed in Barcelona during the technical project meeting in November 2018 and approved by the entire consortium after some minor changes were introduced. Therefore, a first draft architecture including a rough combination of all identified components from different use cases and WP tasks including some generalization was developed as presented in Figure 4.

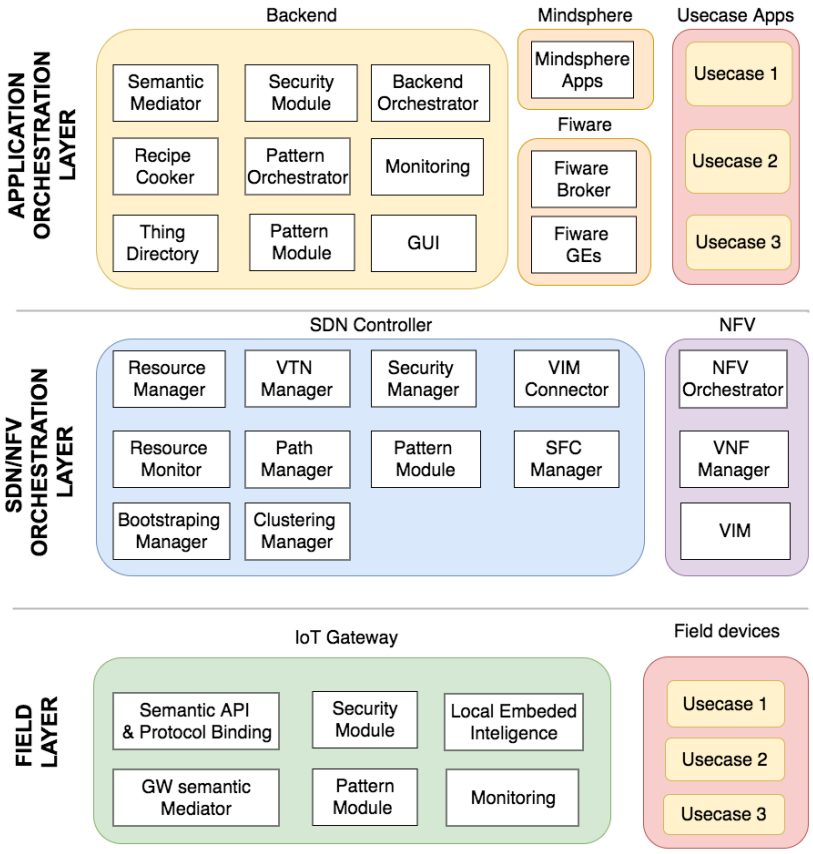


FIGURE 4 FIRST DRAFT OF SEMIOTICS ARCHITECTURE

2.3 Second Draft of SEMIoTICS Architecture

The SEMIoTICS architecture required additional generalization as a general architecture showing all of the components that are shared between use cases. The creation of the first public version of the SEMIoTICS architecture diagram was included in the Deliverable D2.4 SEMIoTICS High-level Architecture. It has been followed by vivid discussions within WP2, WP3 and WP4 workshops and further fine-tuning has been

constantly provided. During the preparation of the Deliverable D2.4 (High-level Architecture) some updates were made to depict the required changes regarding the current status of the Use Cases and the component implementation status. In addition to, the abstract description of the SEMIoTICS architecture as taken from this deliverable was submitted and presented in the Global IoT Summit, in conjunction with IoT week[1].

SEMIoTICS Architecture Updates:

- **Backend Semantic Validator:** During the process, Semantic Mediator in the backend evolved into Backend Semantic Validator – a new component.
- **Recipe cooker**: Extension of the recipe cooker's execution environment for IoT flows to allow their distributed IoT orchestration. Besides enabling the deployment of the components of a flow to different devices, we allow the specification of application-specific constraints (to be auto-translated into patterns to configure the network).
- **Pattern Engine**: An update of the name was done from Pattern Module to Pattern Engine.
- **Security Manager**: An update of the naming was done from the Security Module to Security Manager.
- **Resource Manager**: Logical merge of resource manager and monitoring.
- **Local Thing Directory:** Need for a local version of a component has been identified in order to address possible internet connection losses.

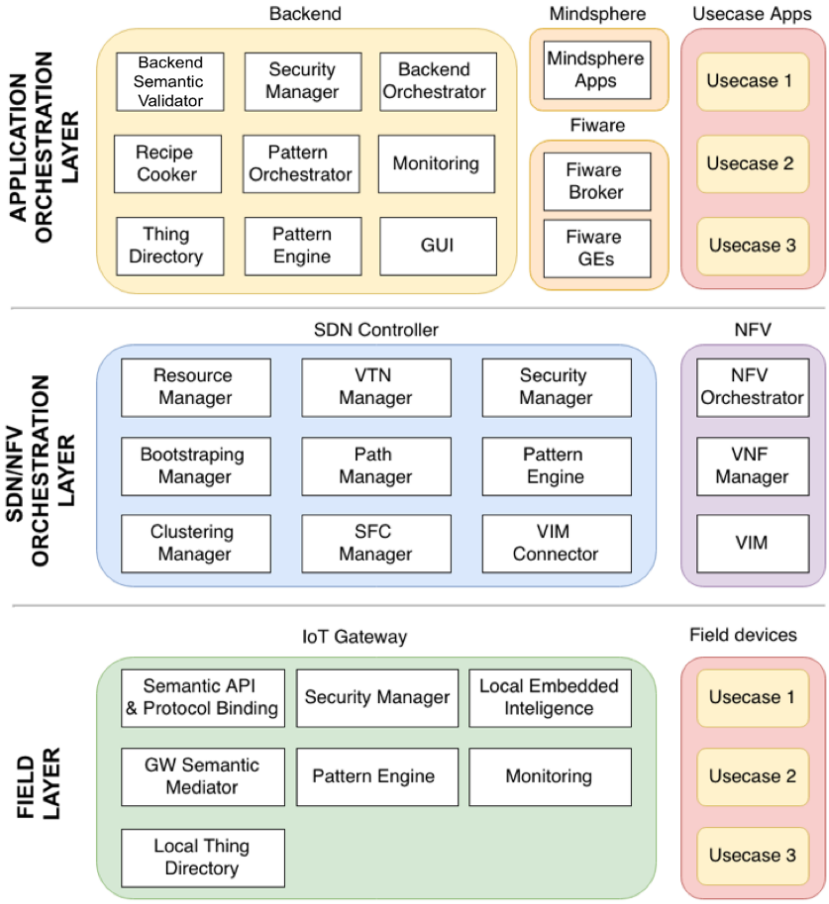


FIGURE 5 SECOND DRAFT OF SEMIOTICS ARCHITECTURE

[1] N. Petroulakis, E. Lakka, E. Sakic, V. Kulkarni, K. Fysarakis, I. Somarakis, J. Serra, L. Sanabria-Russo, D. Pau, M. Falchetto, D. Presenza, T. Marktscheffel, K. Ramantas, P. V. Mekikis, L. Ciechomski, K. Waledzik, SEMIoTICS Architectural Framework: End-to-end Security, Connectivity and Interoperability for Industrial IoT, Global IoT Summit 2019 (GIoTS'19), Aarhus, Denmark, June 17- 21, 2019.

2.4 Final SEMIoTICS Architecture

The final SEMIoTICS Architecture includes the following additional component in order to fulfill the SEMIoTICS objectives and use case requirements.

SEMIoTICS Architecture Updates:

- **Semantic Edge Platform (SEP)** has been introduced as a new component of IoT Gateway (Field Layer). The gateway itself operates in the Field layer. Without the SME, it would be hard for an end-user to scan the network and bootstrap field devices. Thus, SME provides an interface for SEMIoTICS IoT Gateway. For the further role of SEM, please see Section 3.4.8. SEP has been included in IoT Gateway to provide a convenient user interfaces for configuring SEMIoTICS IoT Gateway such as network interface, IP address range when scanning a network for new devices and initiate the device bootstrapping process. In addition, it was required to support a convenient development environment for creating new Apps with a newly bootstrapped device. SEMIoTICS IoT Gateway either provides a uniform API for a new device or re-uses an existing one. This API is automatically exposed over SEP. After the bootstrapping process, there will be created a graphic component (a Node-RED node) based on this API. Thus, the device can be accessed over that node, and the node can be used in new applications right away. These device nodes are automatically created and installed in SPE during the bootstrapping process. Moreover, SEP can provide a mechanism to semantically annotate brownfield devices. So, created semantic descriptions from SEP will be stored in both Local Thing Directory and Global Thing Directory. Furthermore, the SEP can enforce the creation of Edge- and Cloud-based applications in SEMIoTICS. The Recipe-Cooker component in SEMIoTICS will be fully integrated with SEP. That is, it will be possible to instantiate a new application based on a Recipe. The process of discovering field devices, and matching them with affordances from a Recipe will be supported via a machine reasoner that is integrated into SEP.
- **Supervisor and Local DB** component were added also in the architecture since it is required specifically for supporting the Use Case 3 field devices. More specifically, UC3 final architecture required the introduction of a dedicated new component at SEMIoTICS gateway level, the Supervisor and Local DB as presented in section 3.5.7. This component has been defined in order to allow the implementation of Local DB trend analysis (through configurable InfluxDB queries and local data aggregation policies) and local data storing (one of the two envisaged scenarios in UC3). The Supervisor component is also required in order to efficiently handle the communication and the self-adaptive behaviors of a system of intelligent IHES Sensing Units in a real-time environment. Last but not least, this component acts as a logical bridge between the real-time Sensing Node units that continuously pushed new data and events to the local MQTT broker, and the SEMIoTICS backend layer where southbound interfaces are implemented through a GET/POST Rest APIs policy. This facilitates the definition of new derived Use Cases that exploits the edge analytics capabilities of the SEMIoTICS Generic IoT System.
- **OpenHAB Visualisation** component was added as the third additional IoT platform which is necessary for monitoring and visualizing the sensing data and raises alerts. openHAB[2] is a software for integrating different automation systems and technologies into one single solution that allows over-arching automation rules and that offers uniform user interfaces. OpenHAB is a modular, open-source IoT platform, with many functions that are relevant to SEMIoTICS UC3. These functions include the interaction with external sensors, data storage backends and chart libraries for sensor value visualization. Furthermore, openHAB supports a scripting language to implement automation scenarios. OpenHAB functions in UC3 are packaged as VNFs, orchestrated by the MANO framework and placed at either the VIM or the virtualized SEMIoTICS gateway to address low latency monitoring and actuation requirements.
- **Fiware Broker and GEs** were merged into one component.

[2] https://www.openhab.org

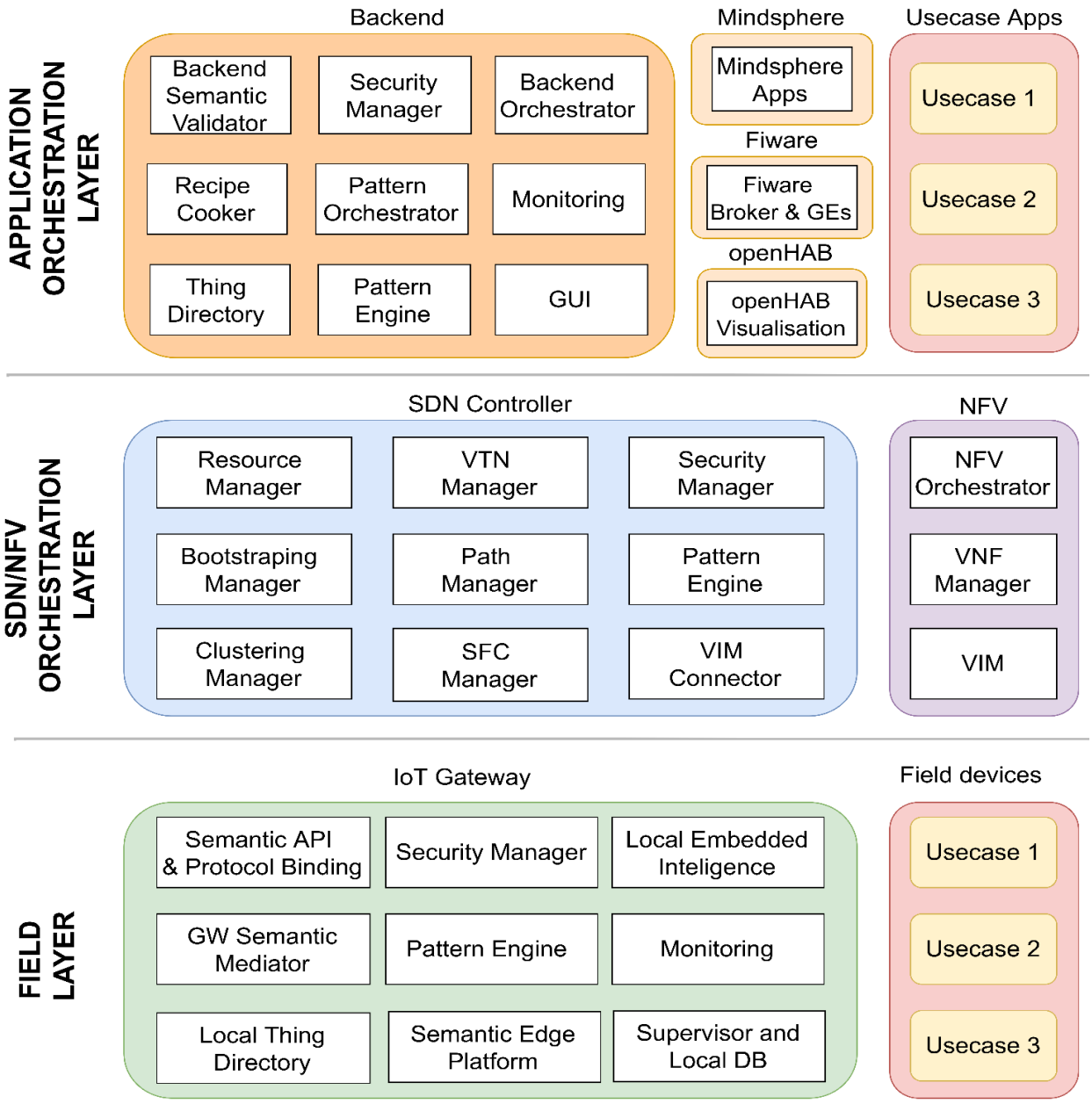


FIGURE 6 FINAL SEMIOTICS ARCHITECTURE

3 ARCHITECTURE OVERVIEW AND COMPONENTS

The SEMIoTICS Architectural Framework aims to leverage generic architecture components combined in layered structure in order to deliver an Embedded Intelligence at all layers of the framework with the mechanisms empowering SPDI patterns verification across all layers as envisioned in the SEMIoTICS envisaged architecture. Created SEMIoTICS pattern-driven framework is capable of supporting diverse scenarios with a specific focus on Smart Energy, Healthcare and Smart Sensing Use Cases. More specifically the SEMIoTICS architecture consists of three layers as follows:

- **Application Orchestration Layer** – consisting of all applications receiving communication from the field layer. The layer provides the framework with security, availability and scalability, privacy, dependability, interoperability as well as intra- and cross-layer monitoring.
- **SDN/NFV Orchestration Layer** – offers flexible, programmable, dynamic and scalable ways to reconfigure network resources in order to provide the QoS demanded by SEMIoTICS Use Cases. It provides end-to-end service connectivity, meets different IoT application requirements in terms of bandwidth, latency, and energy efficiency.
- **Field Layer** – responsible for hosting heterogeneous types of IoT devices. It provides semantic interoperability between IoT devices and seamless flexibility.

By designing the architecture this way, all of the requirements and assumptions of the project are fulfilled. Figure 6, presents the component logical architecture consisting of three layers: Field Layer, SDN/NFV Orchestration Layer and Application Orchestration Layer while more details of each layer are presented further. In the following subsections, the background and the analysis of the deployed component that constitute these layers are described. Finally, the high-level interaction between the components of the framework is also presented.

3.1 Deployment of the SEMIoTICS Framework

As detailed previously, the main scope of SEMIoTICS is to provide a framework to ensure project a dynamicity, scalability, heterogeneity, end-to-end security and privacy based on the three main layers of architecture that have been distinguished: the field, the network (SDN/NFV) and the application. The top layer, where the application orchestration takes place, defines the system's backend components that are partly run on the Cloud or on a server within the network of the lower layers.

To deploy the above layers and the required goal per layer, a number of different components on each layer are required to develop the SEMIoTICS framework. Different types of components are proposed and deployed in this architecture. Some components are developed from scratch or leveraging existing technologies. Other components are adapted for SEMIoTICS needs or existing tools are used without any modification. Apart from the newly developed components in SEMIoTICS, regarding the reuse or extension of existing tools SEMIoTICS, the candidate list includes the following:

- Components from other research projects that partners have participated or involved such as:
  - **VirtuWind**[3]: The aim in VirtuWind was to develop and demonstrate an SDN & NFV ecosystem, based on open, modular and secure framework showcasing a prototype for intra-domain and inter-domain scenarios in real wind parks as a representative Use Case of industrial networks, and validate the economic viability of the demonstrated solution. SEMIoTICS reuses the basis of the VirtuWind SDN controller by using (**VTN Manager, Security Manager, Path Manage**r) or adapting (**Resource Manager, Bootstrapping Manager, Clustering Manager,** and **SFC Manager**) existing components or inserting new ones such as **Pattern Engine**.

[3] Virtual and programmable industrial network prototype deployed in operational Wind park, 5G-PPP Phase 1, 2015-2018, http://www.virtuwind.eu

- **Agile IoT**[4]: It builds a modular hardware and software gateway for the Internet of Things with support for protocol interoperability, device and data management, IoT apps execution, and external Cloud communication. SEMIoTICS benefits by this project by enhancing the gained knowledge to the field and application layer located in the cross-layer **Security Managers**.
- **BIGIoT**[5]: SEMIoTICS reuses the **Recipe Cooker** and the **Thing Directory** from the BIGIoT project. The Recipe Cooker module is extended in SEMIoTICS to support the definition of QoS requirements from an application point of view and to produce a conversion from application flows (recipes) that include QoS requirements into SPDI patterns. These patterns are then sent to and interpreted by the **Pattern Orchestrator**. Beyond these conceptual changes to the Recipe Cooker, its implementation is adapted and refactored to the popular Node-RED platform for IoT mash-up building.
- **FIWARE**[6]**:** Core platform project, offering generic enablers (GEs) for a broad range of areas (i.e. Cloud, Apps & Services, IoT, Interfaces to network and devices, Data & Context management and Security), where SEMIoTICS is able to use such as FIWARE Broker and other GEs.

- Tools from the Open Source Community such as VIM (i.e. OpenStack), Kubernetes and Open Source Mano for NFV Orchestrator that can be adapted in the SEMIoTICS framework or a VNF Manager such as Tacker supported by Openstack.
- Commercial solutions such as the MindSphere IoT platform.

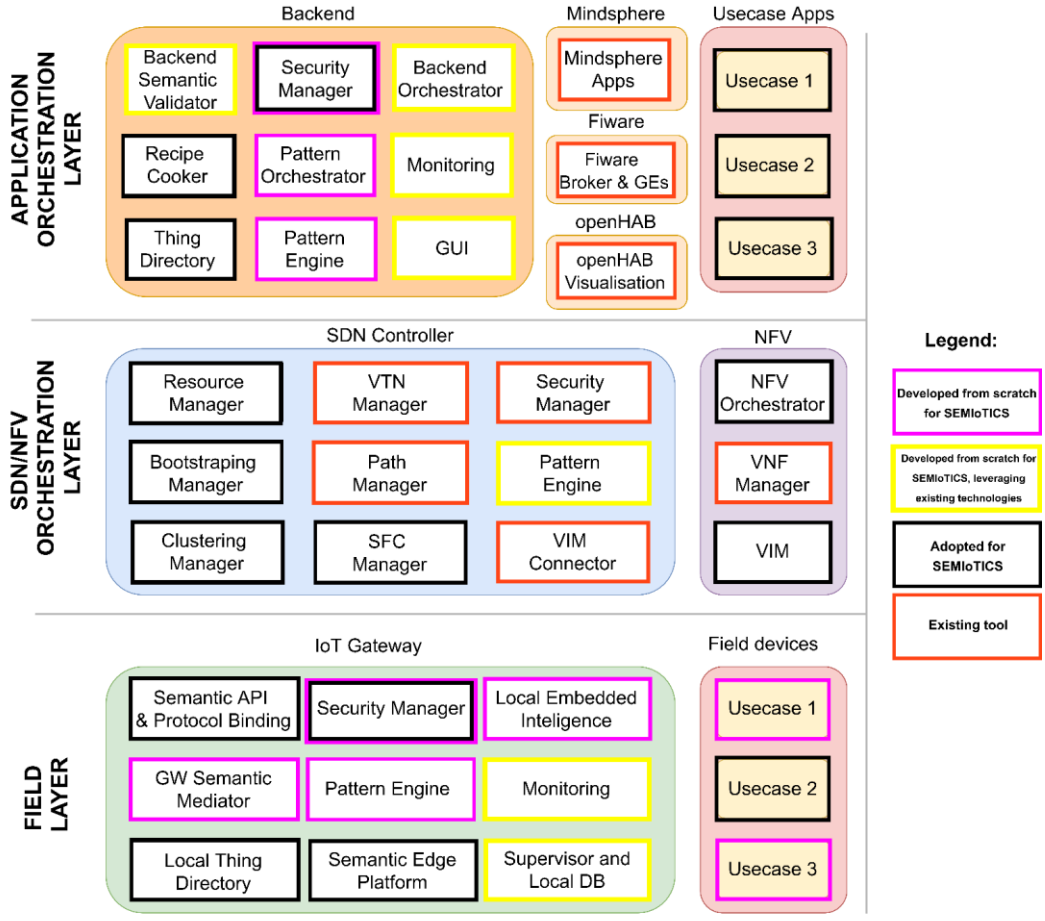


FIGURE 7 ARCHITECTURE DIAGRAM DEPLOYMENT

[4] an Adaptive & Modular Gateway for the IoT, IoT EPI, 2016-2018, https://agile-iot.eu/
[5] Bridging the Interoperability Gap of the Internet of Things, IoT EPI, 2016-2018, http://big-iot.eu
[6] FIWARE: The Open Source Smart Platform, www.fiware.org

Based on the above, a more detailed representation of the proposed architecture providing deployment details for each existing adapted component and newly developed components is presented in Figure 7.

Table 1 presents all components comprising the SEMIoTICS architecture. Moreover, what has been provided is detailed information on component owners as well as the task in the context of which the component will be developed. During the project course, there has been identified a need for a change of the integration approach, hence the specific role of Integrator needs to be defined per Use Case. The definition of the new integrated approach, roles, and scope of work is being identified within WP5 and will be integrated into that workstream.

Further subsections advance the description and deliver more details about each component and its role per layer.

TABLE 1 ARCHITECTURE COMPONENT LIST

Component	Layer	Owner (coordinator)	Implementation task	Maturity level
AREAS	External IoT Platforms	ENG	5.2	Existing tool
Backend orchestrator	Application Orchestration Layer	BS	4.6	Developed from scratch for SEMIoTICS, leveraging existing technologies
Backend Semantic Validator	Application Orchestration Layer	FORTH	4.4	Developed from scratch for SEMIoTICS, leveraging existing technologies
Bootstrapping Manager	SDN Orchestration Layer	SAG	3.1, 3.5	Adapted for SEMIoTICS
Clustering Manager	SDN Orchestration Layer	SAG	3.1, 3.5	Adapted for SEMIoTICS
FIWARE / GE X	External IoT Platforms	IQU	3.5, 5.2	Adapted for SEMIoTICS
FIWARE/ Context Broker	External IoT Platforms	IQU	3.5, 5.2	Adapted for SEMIoTICS
GUI	Application Orchestration Layer	BS	4.6	Developed from scratch for SEMIoTICS, leveraging existing technologies
GW Semantic Mediator	Field Layer	SAG	3.3	Developed from scratch for SEMIoTICS
Local embedded intelligence	Field Layer	ST	4.3	Developed from scratch for SEMIoTICS
Local thing directory	Field Layer	SAG	3.3	Adapted for SEMIoTICS
MindSphere	External IoT Platforms	SAG	5.4	Existing tool
Monitoring	Field Layer	ENG	4.2	Developed from scratch for SEMIoTICS, leveraging existing technologies.
Monitoring	Application Orchestration Layer	ENG	4.2	Developed from scratch for SEMIoTICS, leveraging existing technologies
NFV Orchestrator	NFV Orchestration Layer	CTTC	3.2, 3.5	Adapted for SEMIoTICS
Path Manager	SDN Orchestration Layer	SAG	3.1, 3.5	Existing tool
Pattern Engine	Field Layer	FORTH	4.1	Developed from scratch for SEMIoTICS

Pattern Engine	SDN Orchestration Layer	FORTH	3.1, 3.4, 4.1, 4.5	Developed from scratch for SEMIoTICS, leveraging existing technologies
Pattern Engine	Application Orchestration Layer	STS	4.1	Developed from scratch for SEMIoTICS
Pattern Orchestrator	Application Orchestration Layer	STS	4.1	Developed from scratch for SEMIoTICS
Recipe Cooker	Application Orchestration Layer	SAG	4.4, 4.6	Adapted for SEMIoTICS
Resource Manager	SDN Orchestration Layer	SAG	3.1, 3.5	Adapted for SEMIoTICS
Security Manager	Field Layer	UP	4.5	Some Subcomponents developed from scratch for SEMIoTICS;
Security Manager	SDN Orchestration Layer	FORTH	3.1, 4.5	Existing tool
Security Manager	Application Orchestration Layer	UP	4.5	Some Subcomponents developed from scratch for SEMIoTICS;
Semantic API & Protocol Biding	Field layer	SAG	3.3	Adapted for SEMIoTICS
SFC Manager	SDN Orchestration Layer	FORTH	3.1, 3.5	Adapted for SEMIoTICS
Thing Directory	Application Orchestration Layer	SAG	4.4, 4.6	Adapted for SEMIoTICS
Usecase 1	Field layer	SAG	5.3, 5.4	Developed from scratch for SEMIoTICS
Usecase 1 apps	Application Orchestration Layer	SAG	5.4	Adapted for SEMIoTICS
Usecase 2	Field Layer	ENG	5.3, 5.5	Adapted for SEMIoTICS
Usecase 2 apps	Application Orchestration Layer	ENG	5.5	Adapted for SEMIoTICS
Usecase 3	Field layer	ST	5.3, 5.6	Developed from scratch for SEMIoTICS
Usecase 3 apps	Application Orchestration Layer	IQU/ST	5.6	Adapted for SEMIoTICS
VIM Connector	SDN Orchestration Layer	CTTC	3.1, 3.2, 3.5	Existing tool
Virtualized Infrastructure Manager	NFV Orchestration Layer	CTTC	3.2, 3.5	Adapted for SEMIoTICS
VNF Manager	NFV Orchestration Layer	CTTC	3.2, 3.5	Existing tool
VTN Manager	SDN Orchestration Layer	SAG	3.1, 3.5	Existing tool
Semantic Edge Platform	Field Layer	SAG	3.3	Adapted for SEMIoTICS
Supervisor and Local DB	Field Layer	ST	4.3	Developed from scratch for SEMIoTICS, leveraging existing technologies
openHAB	External IoT Platforms	IQU	5.2	Existing tool

3.2 Application Orchestration Layer

The Application Orchestration Layer consists of all applications receiving communication from the Field Layer. Backend Orchestrator will be leveraged for the application orchestration purposes and to provide common functionalities across all deployed applications.

Additionally, Application Orchestration Layers hold the Use Case flows as well as the SPDI pattern definition.

3.2.1 BACKEND ORCHESTRATOR

Overview: A component responsible for integrating all backend services and exposing API. The technology is to be chosen (OpenShift/Kubernetes/OpenStack).

Core Functionalities:

- Application availability monitoring (health checks).
- Monitoring of application resource consumption.
- Delivering common API for pattern enforcing components.
- Delivering common API for monitoring components.
- Delivering common API for CI/CD tools.
- Providing auto-scaling capabilities for applications ensuring scalability in case of resource saturation.
- Easing application/component deployment.
- Giving one centralized place for backend component management.

Details: The Backend Orchestrator (BO) is a component responsible for provisioning all other applications/components residing in the backend. BO should provide SEMIoTICS Framework with functionalities for application/component development, such as:

- Application availability monitoring (health checks).
- Monitoring of application resource consumption.
- Delivering common API for pattern enforcing components.
- Delivering common API for monitoring components.
- Delivering common API for CI/CD tools.
- Integration with Fiware components.
- Providing auto-scaling capabilities for applications ensuring scalability in case of resource saturation.
- Easing application/component deployment.
- Giving one centralized place for backend component management.

The possible tools for orchestration were revised in detail. While comparing available solutions, out of the box features and restrictions were considered. One of the tasks in the first draft of WP4 implementation was focused on choosing the most suitable tools for backend orchestration. Approaches which were taken into consideration were:

- Kubernetes[7] on bare metal.
- Openstack[8] on bare metal.
- Kubernetes on Openstack.

After analyzing all of the above possibilities, Kubernetes on bare metal was the most appropriate choice for the SEMIoTICS project. This tool allows all partners of a consortium to keep applications in one place that makes them easier to manage and maintain. Each user who has access to BO can manage their application, but only in a specific namespace. It means that interference in the components of other partners is not allowed. This solution ensures that unconscious user action can destroy only one component and does not have any impact on another. Kubernetes uses Docker image to deploy an application, therefore only basic knowledge of using Docker is necessary. Moreover, Docker images are universal and support all environments, so there

[7] https://kubernetes.io
[8] https://www.openstack.org

is no risk that one of the components will not work because of the technology used. BO provides support not only for components residing in the backend layer(e.g. Thing Directory, Pattern Orchestrator, GUI) but also for FIWARE components. All of FIWARE GEs' implementation provides a Docker container that can be used to run own instance.

3.2.2 RECIPE COOKER

Overview: Module responsible for creating recipes reflecting user requirements on different layers (cloud, edge, network) as well as transforming recipes into understandable rules for each layer. It uses Thing Directory with all necessary models to create these rules.

Core Functionalities:

- Creating and instantiating recipes.
- Translates recipes into pattern language and transmits them to Pattern Orchestrator to be executed.

Details: Recipe Cooker (RC) is a module able to instantiate recipes. A recipe is a template for a workflow of interactions between multiple *ingredients*, i.e., devices or services. When a recipe is instantiated, ingredients are replaced with concrete *things*, described with their own respective Thing Description. A draft for a user interface (UI) for the specification of recipes can be seen in Figure 8. Besides the workflow of the recipe, QoS constraints and SPDI patterns can be defined in the interactions.

The user of this tool would be typically an IoT application developer. This user wants to focus on the logic of the application flow. Specifically, the user does *not* have to be an expert in configuring the network and physical connections between the involved IoT devices. The benefit of the recipe approach is that these configurations are automatically done by the tool and the underlying technologies, a user only sets SPDI properties (e.g. latency, rate).

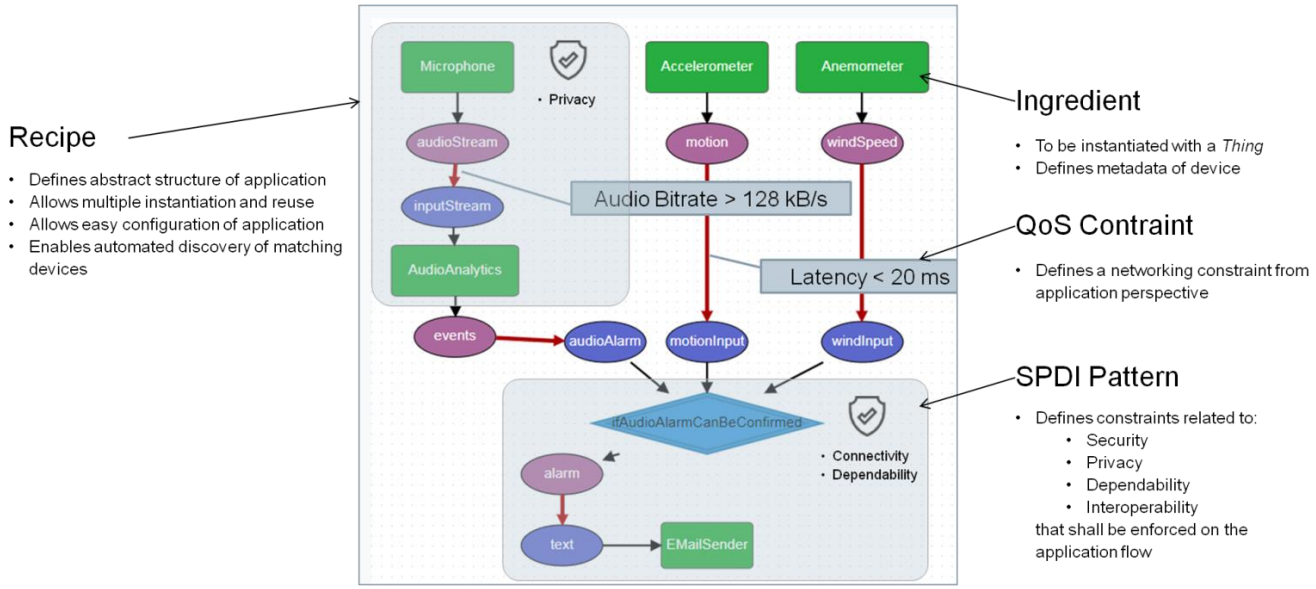


FIGURE 8 DRAFT OF RECIPE COOKER UI

3.2.3 THING DIRECTORY

Overview: The repository of knowledge containing necessary Thing models.

Core Functionalities:

- Searching for a Thing based on its metadata, properties, actions or events.
- Creating a new Thing's TD or updating an existing one.
- Deleting a Thing's TD.

- Generating a runtime environment based on a discovered thing.
- All CRUD operations are supported either over HTTP or CoAP.

Details: The Thing Directory hosts Thing Descriptions of registered *Things*. The Thing Description (TD) model is a recommendation of the W3C Web of Things[9] working group to describe Things. The directory features an API to create, read, update and delete (CRUD) a TD. The directory can be used to browse and discover Things based on their TDs.

3.2.4 PATTERN ORCHESTRATOR

Overview: This module is responsible for translating IoT and service orchestrations (such as cooked recipes from the Recipe Cooker component) into patterns and passing them to pattern engines on each layer.

Core Functionalities:

- Receives recipes in pattern language from Recipe Cooker.
- Breaks down orchestrations in pattern language to rules and facts in a machine-processable format (Drools rules)
- Sends rules and facts to the corresponding Pattern Engines
- Relays orchestration and SPDI properties status to the backend GUI

Details: The Pattern Orchestrator module features an underlying semantic reasoner able to understand the internal components of IoT Service orchestrations expressed using the pattern language (see D4.1, Section 3.3), received from the Recipe Cooker module (see subsection 3.2.2 above) and transform them into architectural patterns. The Pattern Orchestrator is then responsible to pass said patterns to the corresponding Pattern Engines (as defined in the Backend, Network, and Field layers), after translating them to a machine-processable format (in Drools), selecting for each of them the subset of patterns that refer to components under their control (e.g. passing Network-specific patterns to the Pattern Engine present in the SDN controller). Through the above functions, the module achieves automated configuration, coordination, and management of the SEMIoTICS patterns across different layers and service orchestrations. Moreover, the Pattern Orchestrator relays the status of orchestration and the SPDI properties of said orchestrations to the backend GUI.

A high-level view of the operation of the Pattern Orchestrator and its key interactions in IoT Orchestration definition and deployment is depicted in Figure 9.

[9] https://www.w3.org/WoT/

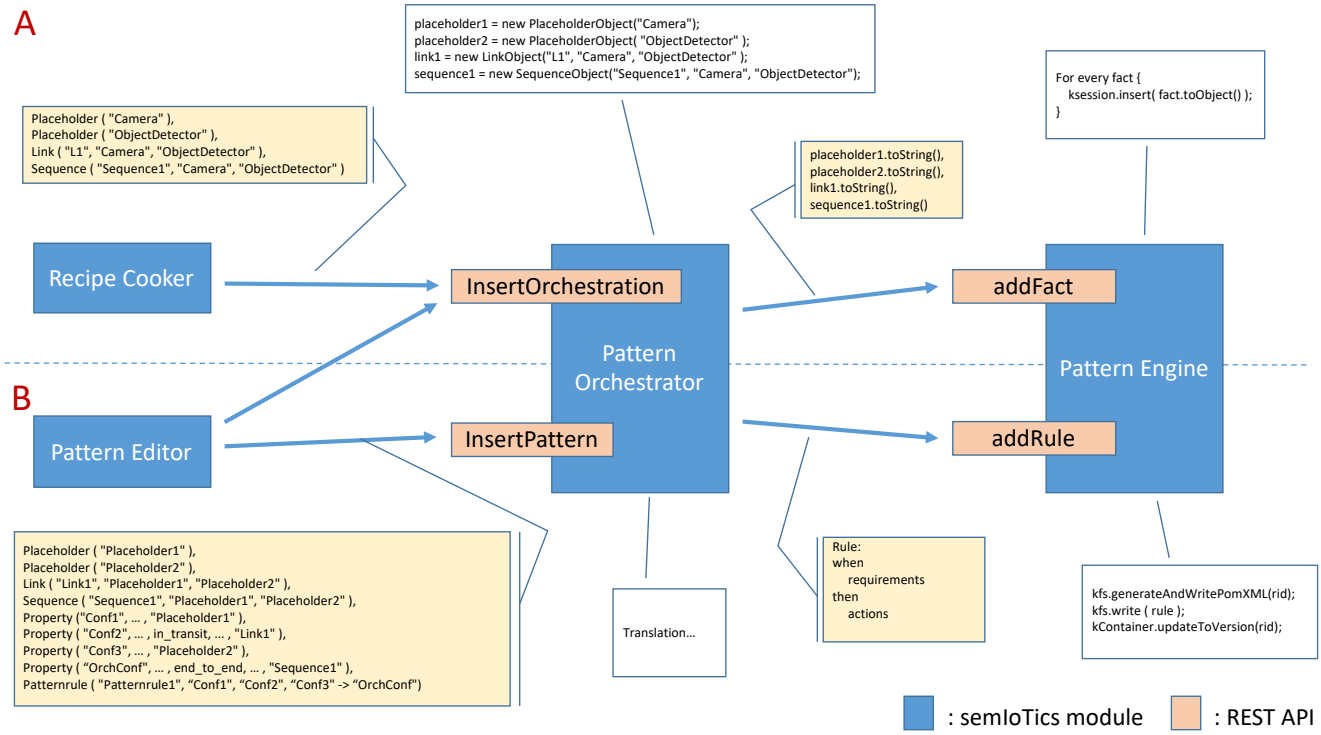


FIGURE 9. PATTERN ORCHESTRATOR; KEY INTERFACES AND INTERACTIONS

3.2.5 (BACKEND) PATTERN ENGINE

Overview: Module responsible for monitoring and reasoning on pattern rules related to SPDI properties of backend components as well as end-to-end orchestration properties.

Core Functionalities:

- Inserting, Modifying, Executing, Retracting patterns at the backend
- (Backend and end-to-end properties') Drools reasoning
- Aggregating properties' status (facts) from the lower layer pattern engines

Details: The Pattern Engine features the pattern engine for the SEMIoTICS framework. Variants of pattern engine can be found at the backend (detailed here), at the network (SDN controller; detailed in subsection 3.3.12) and field (IoT gateway; detailed in subsection 3.4.6) layers. As such, it enables the capability to insert, modify, execute and retract patterns at design or at runtime in the backend; these interactions will typically happen through the interfacing with the Pattern Orchestrator (see subsection 3.2.4 and Figure 9 above) or between Pattern Engines, though additional interfaces can be introduced to allow for more flexible deployment and adjustments if needed.

At the backend, using said patterns and the Drools[10] rule engine, along with monitoring capabilities present at the backend layer, the Backend Pattern Engine is able to reason on the Security, Privacy, Dependability, and Interoperability (SPDI) properties of aspects pertaining to the operation of the SEMIoTICS backend. Moreover, at runtime the Backend Pattern Engine may receive fact updates from the individual Pattern Engines present at the lower layers (Network & Field), allowing it to have an up-to-date view of the SPDI state of said layers and the corresponding components.

[10] Drools Business Rules Management System (BRMS) https://www.drools.org

For example, a security pattern can be used to define integrity protection on a logical communication link, helping at design time to select components that can provide said integrity protection, but also monitoring at runtime that these components indeed do enforce this protection. Moreover, adaptations can be triggered if, e.g. at runtime it is detected that one of the involved components fails to satisfy this requirement, replacing it with an alternative one.

In Figure 10 below high-level representation of the communication of the Backend Pattern Engine, the Pattern Engines of the other layers and the Pattern Orchestrator is presented in the form of a sequence diagram.

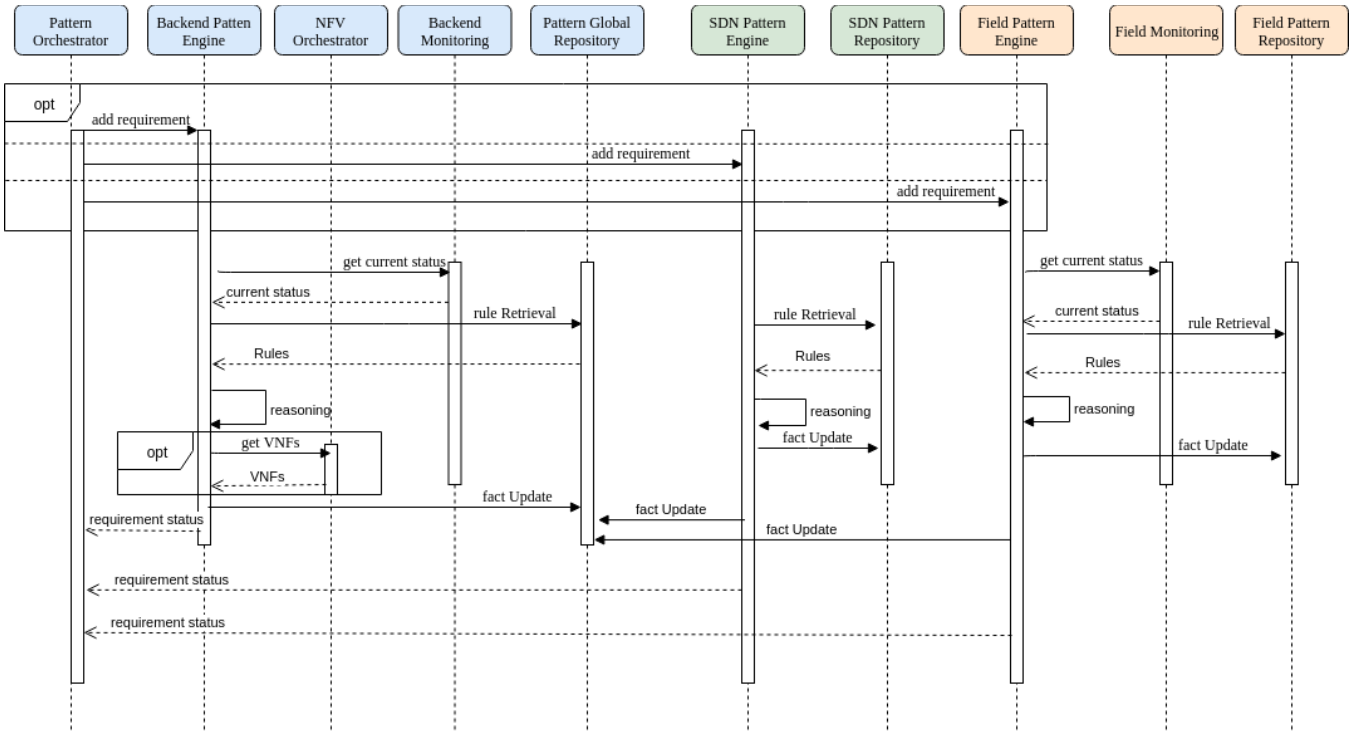


FIGURE 10: COMMUNICATION OF BACKEND PATTERN ENGINE WITH OTHER SEMIOTICS COMPONENTS

3.2.6 MONITORING

Overview: Component responsible for monitoring, learning, and predictive analytics.

Core Functionalities:

- Fusion of intra- and cross-layer monitoring results generated by monitors that may exist on the platforms of different smart objects and components of IoT applications in order to detect violations

Details: The monitoring component in the backend layer has two key requirements:

- To generate specific messages in response to the reception of a set of messages generated by the components of an IoT application and matching some condition specified in the monitoring component by a client application (Monitoring requirement).
- To guarantee that the messages needed to decide whether to generate a message can be produced by an IoT application and received by the monitoring component (Observability property).

Figure 11 presents the main required inputs and outputs of the SEMIoTICS monitoring component.

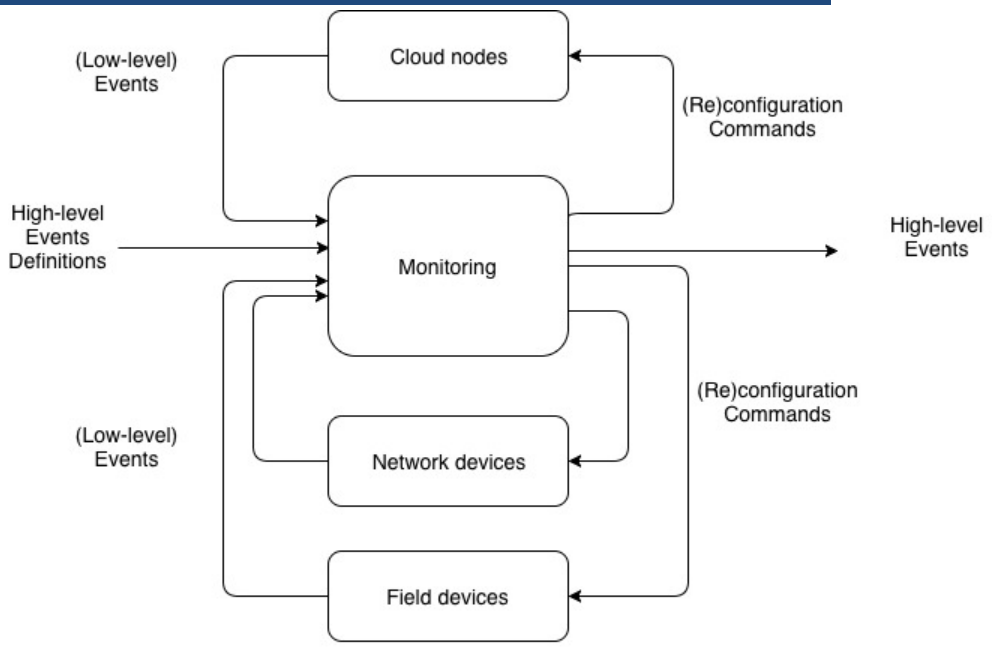


FIGURE 11 MAIN INPUT AND OUTPUT OF THE MONITORING COMPONENT

In particular, the Monitoring component receives as input:

- **Low-level events**: the messages generated by the computational nodes belonging to the three layers identified by the SEMIoTICS architecture: field (e.g. sensors, gateways), network (e.g. routers) and cloud (e.g. FIWARE cloud services, MindSphere services). These low-level events are generated by the computational nodes by means of signaling mechanisms specific to the technology used to implement a computational node.
- **High-level events definitions**: the conditions stating whether a new event should be generated in the response of the reception of a set of low-level events.

The monitoring component emits as outputs:

- **High-level events**: the messages generated by the monitoring component itself in response to the reception of a set of low-level events matching one of the events definitions.
- **Configuration commands**: messages requesting a specific configuration of the mechanisms allowing the computational nodes to generate the low-level events. The possibility to issue these commands allows the monitoring component to properly select and configure the signaling mechanisms needed for the monitoring purpose.

In order to provide the observability property, the Monitoring component embeds learning and predictive analytics components (not shown in Fig. 3) that enables the anticipatory behavior needed to guarantee that the monitoring tasks can continue with the expected QoS despite the failure of some of the components (e.g. event collectors) contributing to the overall monitoring task.

3.2.7 SECURITY MANAGER

Overview: Module responsible for granting access and necessary security checks at the Backend Layer.

Core Functionalities:

- Providing services to Authentication, Key distribution, Management of users, roles, access rights.

Details: The Security Manager (SM) provides the following services to other SEMIoTICS components, devices, and (human) users:

- Authentication.
- Key distribution.
- Management of users, roles, and access rights.

Furthermore, it stores and takes decisions on security policies across all SEMIoTICS components. The security manager at the backend layer is the Policy Decision Point (PDP). In contrast, the security managers at SDN and edge level are Policy Enforcement Points (PEP). The security managers at SDN and edge levels only have a local view on security policies and authentication, whereas the security manager at the backend has a global view. Therefore, the following case can happen: The security manager at the edge level (or SDN level respectively) might not have the information required to decide whether to grant or deny an action; it then queries the security manager at the backend layer on what decision to take.

The security manager is responsible for all authentication decisions. It supports both local authentications as well as relying on external identity providers using OAuth2. By means of OAuth2, particularly strong authentication mechanisms such as two-factor-authentication (2FA) are also supported. As a result, the security manager shares a long-term key with each component and device.

Another service provided by the security manager is key distribution. Whenever two components or devices want to protect the confidentiality of a direct communication link, they require keys for encryption. As the security manager shares a long-term key with both components, it can use this key for securely exchanging a session key with both components. Additionally, the security manager may have better means for securely generating keys, for example by using hardware support; in contrast, in particular sensors and actuators with limited computational power may lack the resources to securely generate keys at all.

The security manager is also responsible for managing roles, users, and access rights across SEMIoTICS. Users may assume one or more roles such as regular users, security analysts, etc. Access rights are defined by security policies that are stored in the security manager.

Finally, the security manager offers cryptographic functionality to combine the last two mentioned functionalities: managing the access control via roles and managing keys: it supports the functionality of attribute-based encryption, where a role can be set as an attribute, e.g. user can have the attribute is_doctor and by generating a key for the encryption of data with this attribute the Security Manager is able to ensure that only users who have a decryption key containing that role can later decrypt it.

3.2.8 BACKEND SEMANTIC VALIDATOR

Overview: Module responsible for semantic validation mechanisms at the Backend Layer.

Core Functionalities:

- Detection of any potential semantic conflicts in Things Description.
- Resolving semantic conflicts.
- Development of data transformation techniques and validation mechanisms to ensure end-to-end semantic interoperability.

Details: The aim of the Backend Semantic Validator component (see SEMIoTICS deliverable D4.4) is to tackle the semantic interoperability issues that arise in the SEMIoTICS framework, at the Application Orchestration Layer. The Backend Semantic Validator can receive a request from IoT application for interaction between two Things (i.e. sensor, actuator), which are described with two different TDs (based on W3C Thing Descriptions that are serialized to JSON-LD standard format), respectively. The functionality of this component consists of:

- Searching for the necessary Thing models in Thing Directory component (Section 3.2.3), in order to detect any potential semantic conflicts between the interacting domains.
- Connecting with Recipe Cooker (Section 3.2.2) and Semantic Edge Platform (in the field) to resolve these semantic conflicts using the Adaptor Nodes that configure an Interaction Pattern in accordance with the application's requirements. The Adaptor Nodes and their functionality will be described and analyzed in D4.11.

- Transferring the translated request to the Semantic API & Protocol Binding component (Section 3.4.1) which is responsible to trigger the GW Semantic Mediator (Section 3.4.4) in the Field Layer, in order to send the request in an appropriate format to the target Thing (actuator).

3.2.9 GRAPHICAL USER INTERFACE

Overview: A component responsible for the presentation layer.

Core Functionalities:

- Visualization of SPDI pattern monitoring, pattern details, recipes.
- Interacting with smart devices (Things)Things.
- Performing CRUD operations on Thing Descriptions.
- Collecting data gathered from IoT devices.
- Providing user dashboard interface.
- Providing routing to other SEMIoTICS' components.
- Providing visualization of monitoring data (leveraging FIWARE Knowage GE).

Details: Graphical User Interface is a component responsible for providing one centralized point of interaction with the platform while giving meaningful insights into the platform and centralized visualization of the whole framework as well as a layer of presentation for specific Use Cases.

The following approaches are to be considered:

- GUI that communicates through the API with an external application.
- GUI that loads the view itself from the external application.
- GUI that is dedicated to the given backend application.

Further definition of GUI is subject to Task 4.5 within the project.

3.2.10 USE CASE 1 APPLICATIONS

Overview: Use Case 1 backend application including all necessary underlying components. It applies UC1 business logic to the platform.

Core Functionalities:

- Translation of application-level network requirements to network constraints and monitoring and enforcing them in a decentralized edge infrastructure of a wind turbine.
- The distributed application transfers video data to the analytics engine (for oil detection) and stops the wind turbine in a detected emergency case.

Details: The backend/cloud module, to demonstrate UC1, consists of a variety of modules supported by a single web-UI:

- **Application definition:** flow-based programming tool used in the creation of abstract applications allowing multiple instantiation and reuse. Applications definition is based on pre-defined models & capabilities derived from sensors and actuators, i.e. video and audio feeds, inclinometers, temperature sensors, etc. The module supports regular functions such as if/else and for-loops.
- **Application deployment:** UI supporting the execution of defined applications. The module facilitates the configuration, validation, and execution of an application. Examples of functionalities include a selection of ingredients required by the application to be executed, and instantiations parameters, e.g. if the application shall be run in the cloud or at the IIoT gateway. Furthermore, this module validates if all pre-requisites, e.g. QoS constraints, can be met by the architecture prior to deployment and execution of the application.
- **MindSphere Apps:** There are Use Case specific Apps in MindSphere cloud platform used for time-series data processing and/or visualization. These apps take the current and historical data from the field layer.
- **Edge Apps:** Not all field data needs to be transferred to the Cloud. Apps can also run on the Edge. These Apps typically operate on data, which is accessible from SEMIoTICS IoT Gateway. For apps

that should operate on a global view (e.g., data from a whole plant) Cloud Apps (MindSphere Apps) are more suited.

3.2.11 USE CASE 2 APPLICATIONS

Overview: Use Case 2 backend application including all necessary underlying components (Daily Activity Monitor, Patient Activities Scheduler and HR Interaction Manager).

Core Functionalities:

- Enforcement of Security, Privacy, Dependability and Safety properties stemming from GDPR.
- Enforcement of access control policies.
- Detection of anomalies within patterns of discrete events generated by the Smart Environment.
- Standardized access to CoAP and ZigBee device.
- Discovery of heterogeneous (CoAP and ZigBee) devices.
- Aggregate and analyze data from heterogeneous devices.
- Distributed AI at the edge (e.g. Gait Analysis).
- Manage the optimal configuration of networking resources w.r.t. uncertainty and unpredictability in the distributed computational loads.

Details: The SARA module represents the backend of the SARA solution. It consists of a set of services providing all the functionalities required to fulfill the requirements presented in deliverable D2.1. Examples of functionalities provided by the SARA module include the Daily Activity Monitor, the Patient Activity, Scheduler, the Tele-monitoring service, the Localization, and Mapping service, the Human-Robot Dialog Manager.

Some of the above services (e.g. he Human-Robot Dialog Manager) require access to third party AI services (e.g. IBM Watson, Google) to provide advanced functionalities.

Moreover, the SARA solution includes a web application providing the Graphical User Interfaces supporting the various user roles envisaged for the solution:

- **Call-center operator:** access patient details for incident appraisal and handling, record incident reports in the Patient Diary, update the Patient Diary with incident tracking/outcome info, access first responder details for incident handling.
- **Medical expert:** query patient's monitored daily activities, perform statistical analysis of Patient data, manage Patient records, manage Patient-specific calendar of scheduled activities, and manage first responder records.
- **Technician:** access patient-specific service configuration, maintain the Technical Inventory (e.g. replacement of a battery in a Robotic Rollator).

The SARA module, as part of the AREAS® software suite, interfaces other modules of AREAS® through the AREAS® service bus. The AREAS® service bus allows the SARA solution to access the AREAS® Patient Health Record service.

The SARA module relies on the CLoE-IoT platform (see Section 3.4.4) for what concerns the management of IoT requirements.

3.2.12 USE CASE 3 APPLICATION

Overview: Use Case 3 backend application including underlying components. UC3 Application is a specific component developed in SEMIoTICS leveraging existing IoT technologies that will be responsible for the visualization, management, data aggregation, system monitoring of the IHES Edge Analytics System.

Core Functionalities:

- Visualization of the global system status (I.e. the status up-to-single IHES node) of the IHES System.
- Visualization of the relevant events (I.e. anomalies, etc.) generated by each IHES unit or cluster of units.
- Visualization of the correlation graph, the status of aggregated IHES Units by the aggregated information provided by each Supervisor Service deployed at the Field layer.

- Exposing interface with Local DB that stores all events and the related event generated by a set of IHES Sensing Units associated with a specific Supervisor on a given gateway.
- Visualization and management of alerts generated by the IHES system.

Details: As a testbed application to validate the generic IHES demonstrator and shows its capabilities will be developed as part of the WP5 activities within the project. A dedicated web app will be defined and implemented as part of T5.6 activities with the goal of demonstrating the system at work. The UC3 app will report in a simple GUI the environmental processes monitored by the IHES system in order to demonstrate in a straightforward way the two main scenarios declared in D2.2 (section 2.3.2.2). The application will report any (relevant) anomaly detected by the system at single/multiple nodes level and will provide at a glance a complete overview of the relevant events globally aggregated by the IHES system. Since the proposed enabling technology has no major limitations on data series processing, the system can deal with any kind of time-series signal, with no specific a priori knowledge on its dynamic. IHES system will be able to deal with most environmental sensors that will be incrementally demonstrated during the project lifespan: initially by supporting low-frequency signals like temperature/humidity/pressure to move in a 2nd phase to the deployment of more complex inertial sensors management (i.e. 3-axis accelerometers and gyroscopes).

3.3 SDN/NFV Orchestration Layer

SDN and NFV offer flexible, programmable, dynamic, scalable to reconfigure network resources in order to provide the QoS demanded by SEMIoTICS IoT Use Cases. To this end, the SEMIoTICS SDN approach is as follows: network control assumes a centralized network decision making entity. It is in charge of configuring (optionally) QoS-constrained paths required by the field layer devices as well as the control flows required for VNF and device management throughout all layers of the architecture. Additionally, the SDN controller is in charge of isolating the interactions between various tenants of the infrastructure by means of virtual tenant networks. The SDN controller components are prevalent both in the backend and intra-site infrastructure, so to provide for interconnection of virtualization services (i.e., VNF interconnection) as well as for field-layer and field-backend layer interactions. NFV deployment in SEMIoTICS assumes general-purpose hardware devices deployed throughout different parts of the network. In the SEMIoTICS architecture, this corresponds to the IoT Gateway, network nodes such as switches, and compute nodes at the backend cloud. Moreover, it is assumed that these machines allow the virtualization of their resources in terms of e.g. virtual machines (VM) or containers, yielding a pool of virtual computing, storage and communication resources available to deploy virtual Network Services (NS). The virtualization of the hardware resources is managed by a so-called virtualization layer. As can be seen in Figure 12, the set of physical hardware resources, the virtualization layer and the virtualized computing storage and networking resources is the so-called NFV Infrastructure (NFVI). Thereby, NFVI contains all the available resources available in the network. NFVI paves the way to obtain a flexible, programmable, dynamic and scalable network, as the virtual network resources exposed to the network services can be dynamically assigned or released in different parts of the network to meet the required QoS requirements.

We next discuss the main components of the SDN/NFV layer in the SEMIoTICS architecture.

3.3.1 VIRTUALIZED INFRASTRUCTURE MANAGER

Overview: Virtual Infrastructure Manager (VIM) is a component that is aware of the physical infrastructure (compute, storage and networking resources). Its main task is to control the virtualization of those resources and to expose them to the services that run on top of the virtualized infrastructure. VIM also enables communication with SDN Controllers to provide network resources.

Core Functionalities:

- Providing communication with SDN Controllers.
- Monitoring of the physical infrastructure,
- Managing of software resources,

Details: NFVI defines two Administrative Domains [11] namely, the Infrastructure and Tenant domains. The former contemplates the physical infrastructure upon which virtualization is performed and therefore it is application agnostic. The latter makes use of virtualized resources to spawn VNFs and create NS. Unlike resource allocation in other virtualized environments, in NFVI requests simultaneously ask for computing, storage, and network resources. Moreover, NS could be composed of VNFs with hardware affinity/anti-affinity or require specific latency/bandwidth constraints in virtual links connecting VNFs. Such demands occur dynamically, allocating or freeing resources that could be used for other NS, e.g. scaling up VNF's compute.

A VIM lies in the Infrastructure Domain. It takes care of abstracting the physical resources of the NFVI and making them available as virtual resources for VNFs. This is achieved through the reference point **Nf-Vi**, which interconnects the VIM and NFVI (see Figure 12). It allows the VIM to acknowledge the physical infrastructure (compute, storage) as well as enabling communication with network controllers (SDN Controllers) to provide virtual network resources to NS. Even-though VIMs could well control all resources of the NFVI (compute, storage and network), they could also be specialized in handling only a certain type of NFVI resource (e.g. compute-only, storage-only, network-only)[11].

Beyond the already-mentioned, functions carried on by the VIM are the following:

- Orchestrate requests made to the NFVI from higher layers (NFVO), e.g. allocation/update/release/reclamation of resources.
- Keep an inventory of allocated virtual resources to physical resources.
- Ensure network/traffic control by maintaining virtual network assets, e.g. virtual links, networks, subnets, ports.
- Provide network-level security functions via VNFs such as Honeypots, Intrusion Detection/Prevention Systems, Firewalls
- Management of VNF Forwarding Graphs (VNFFG) by guaranteeing their compute, storage and network requirements.
- Management and reporting of virtualized resources utilization, capacity, and density (e.g. virtualized to physical resources ratio).
- Management of software resources (such as hypervisors and images), as well as the discovery of capabilities of such resources.

As detailed in [11] other relevant VIM responsibilities within the NFVI network are:

- Provide "Network as a Service" northbound interface to the NFVO (realized via the **Or-Vi** reference point, see Figure 12).
- Abstract the various southbound interfaces (SBI) and network overlay mechanisms exposed by the NFVI network.
- Invoke SBI mechanisms of the underlying NFVI network.
- Establish connectivity by directly configuring forwarding instructions to network VNFs (e.g. vSwitches), or other VNFs not in the domain of an external network controller.

3.3.2 NFV ORCHESTRATOR

Overview: A component responsible for the orchestration of Network Function Virtualization. Combined with the other VIM Manager Component creates the so-called Management and Orchestration (MANO) framework.

Core Functionalities:

- Providing VNF with the NFVI resources,
- Registration of the available VNF and Network Service.

[11] ETSI. (2014, December). *ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001).* Retrieved November 2018, from https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

Details: NFV MANO framework is composed of a Virtualized Infrastructure Manager (VIM), VNF Manager (VNFM), and NFV Orchestrator (NFVO) (see Figure 12). This section deals with the functional description of the NFVO, particularly, the Network Service and Resource Orchestration functions, and the related Information Models (IM) used to build descriptors that help spawn NS.

Management and Orchestration of VNF relate to providing each VNF with the NFVI resources they need. But also, other aspects such as registering available VNFs or NS, scaling in/out each VNF according to policies or load, lifecycle management, snapshots, modifying the network interconnection among VNFs, modifying the VNFs in a VNFFG, creation, and termination of NS. These are potentially complex tasks, primarily because VNF's NFVI resource requirements and constraints need to be satisfied simultaneously on top of a very dynamic environment (VNFs are instantiated or terminated, changing the pool of available resources). To leverage this, the NFV MANO (VIM+VNFM+NFVO) should expose services that support accessing these resources, preferably using standard APIs[11]. The NFVO performs two main functions, called Network Service and Resource Orchestration functions (NSO and RO, respectively). Capabilities of each function are exposed via standard interfaces consumed by other elements of the NFV MANO.

The following non-exhaustive list gathers some of the functionality performed by the NFVO employing the NSO function:

- Checks that VNF or NS descriptors include all mandatory information for onboarding.
- Through VIM's exposed services, NSO checks that the software images specified in the descriptors are available at the targeted VIM.
- NS lifecycle management, that is instantiation, update, scaling, event collection and correlation, and termination.
- Collects performance metrics from NS.
- Management of the instantiation of VNFs (alongside VNFM).
- Validation and authorization of NFVI requests from VNFM.
- Management of the relationship between NS instances and VNF instances.
- NS automation management based on triggers specified in the NS descriptors.

On the other hand, the RSO function interfaces with the NFVI to make sure resources are available for the instantiation of VNF/NS. The following non-exhaustive list gathers some of the services provided by the RSO function:

- Validation and authorization of NFVI requests from VNFM.
- NFVI resource management (distribution, reservation, and allocation) by maintaining an NFVI repository.
- Leverages resource utilization information gathered from VIMs to manage the relationship between VNF instances and NFVI resources.
- Policy management and enforcement, e.g.: NFVI resource access control, affinity/anti-affinity rules, resource usage, among others.
- Collects usage information of NFVI resources by VNF instances.

Apart from APIs exposed by VIMs (which are triggered through the Or-Vi reference point, see Figure 12, descriptors are the main element in the instantiation of NS. In them, administrators specify details about VNFs, as well as Virtual Links (VL), VNFFG, and the NS as a whole (even PNFs). All descriptors should be on-boarded to the NFVO in order for the NSO function to verify them (e.g.: checking the validity of all fields, checking the availability of software images at VIMs, among others). The following is a list of descriptors and a short description of their functionality:

- NS descriptor (NSd): used by the NFVO to instantiate an NS which would be formed by one or several VNFFG, VNF, PNF, and VL. It also specifies the deployment flavors of NS.
- VNF descriptor (VNFd): describes a VNF in terms of deployment and operation behavior. It includes network connectivity, interfaces and KPIs requirements that can be used by NFV-MANO functional blocks to establish appropriate VL within the NFVI.

- VL descriptor (VLd): provides information about each virtual link. It is used by NFVO to determine the appropriate placement of a VNF instance, and by the VIM to select a host with adequate network infrastructure. The VIM or external SDN controller may use this information to establish the appropriate paths and VLANs.
- VNFFG descriptor (VNFFGd): it includes metadata about the VNFFG itself, that is, VL, VNFs, PNFs, and policies (e.g.: MAC forwarding rules, routing entries, firewall rules, etc.).
- PNF descriptor (PNFd): is used by NFVO to create links between VNFs and PNFs. It includes information about connection points exposed by the PNF, and VLs that such physical connection points should be attached to.

3.3.3 VNF MANAGER

Overview: This module focuses on all virtualization-specific management tasks necessary in the NFV framework. This component is responsible for the lifecycle management of Virtual Network Functions.

Core Functionalities:

- Creating and managing of the needed virtualized resources for the VNF as well as Fault, Configuration, Accounting, Performance, Security Managing.

Details: VNF lifecycle management refers to the creation and management of the needed virtualized resources for the VNF[11], as well as the traditional Fault Management, Configuration Management, Accounting Management, Performance Management and Security Management (FCAPS).

By making use of the information stored in VNF descriptors (VNFd) during onboarding, VNF Management functions make sure such requirements are met at the moment of instantiation. Furthermore, VNFd may also contain information relevant for the lifecycle management (e.g.: constrains, KPIs, scale factors, policies, etc.). Such lifecycle management information may be used for scaling operations, adding a new virtualized resource, shutting down an instance, or terminating it.

VNF Management maintains the virtualized resources that support the VNF functionality, without interfering with the VNFs' logical functions. Like NFVO, its functions are exposed through APIs as services to other functions. Each VNF instance is assumed to have an associated VNF Manager, and a VNF Manager could handle several VNFs. The following non-exhaustive list gathers the functions implemented by the VNF Manager[11]:

- VNF instantiation (based on-boarded VNFd).
- VNF instantiation feasibility checking.
- Scale VNFs (increase or decrease the resources of a VNF).
- Software Update/Upgrade on VNFs.
- Correlation between NFVI measurement results and faults/events, and the VNF instances.
- VNF instance assisted or automated healing.
- Terminate VNF (releasing the VNF-associated NFVI resources).
- Management of the VNF instance's integrity during its lifecycle.

3.3.4 VTN MANAGER

Overview: Responsible for assignment of individual network services to various network tenants. It further ensures a separation of L2 traffic in the scope of a virtual tenant network.

Core Functionalities:

- Providing multi-tenancy in the network,
- Enforcing of the isolation of the tenant networks in the infrastructure

Details: VTN Manager is a component of the SEMIoTICS SDN Controller that will provide for the multi-tenancy functionality in the network. It realizes logical slices ("virtual tenant networks") for per-application mapping and enforcement of isolation of the tenant networks in the infrastructure. Using the exposed northbound interface, VTN Manager must thus allow for the creation of tenant networks and translation of pattern requests into path-

request calls to Path Manager in the scope of its VTN. VTN Manager will store all resulting data structures containing information about reservations and established VTNs in the centralized data store.

3.3.5 PATH MANAGER

Overview: Main network path computation engine of the SDN Controller, responsible for the identification of nodes and ports combined into a path that fulfills the pattern requirements (i.e., on fault-tolerance or bandwidth/delay constraints).

Core Functionalities:

- Translating pattern requests into path-request calls to Path Manager.
- Storing data structures containing information about reservations and established VTNs.

Details: Path Manager guarantees the industrial QoS, i.e. the bandwidth provisioning, flow isolation and worst-case delay estimation for individual per-application flows. As described in D3.1, instead of basing its routing decision on a reactive control loop of network observations, Path Manager provides for real-time constraints by mechanisms for admission (and rejection) of flows. Namely, by maintaining an accurate model of the network state and service embedding in the control plane, Path Manager ensures per-flow isolation and worst-case guarantees at all times. In addition to providing for constrained QoS path computation, it is capable of computing resilient flows for incorporation of seamless redundancy in flow transmission, i.e., in the case of data-plane failures (i.e., a switch/port/link failure), given sufficiently disjointness of the paths in the physical network topology, resilient transmission of flows, with no packet loss can be provided.

3.3.6 RESOURCE MANAGER

Overview: Provides Path Manager with a resource view of the network (i.e., the available topology resources, port speed, no. of queues metrics, etc.) exposing the metrics observable using the standardized OpenFlow 1.3 interface.

Core Functionalities:

- Embedding of real-time flows, best-effort flows, the meter structures for policing purposes.

Details: Resource Manager is responsible for configuration management and network control tasks, i.e. embedding of L2/L3 OpenFlow flow rules into the network. Resource Manager will provide for the embedding of: i) real-time flows that require dedicated per-queue flow assignments; ii) best effort flows, without queue considerations; and iii) the meter structures for policing purposes. Furthermore, the Resource Manager exposes its internal data-store using a REST-based approach, so to allow access to internal data-state of the controller, free to use in higher layer Network Management System or monitoring components – i.e., the number of admitted flows, the computed paths for the admitted flows, the occupied resources, the admitted Virtual Tenant Networks as well as the general topology and network device capabilities.

3.3.7 SECURITY MANAGER

Overview: The security component responsible for the administration of tenants and assignment of applications with respective tokens used for fast authentication during runtime.

Core Functionalities:

- Authenticating and accounting services to the rest of the SDN controller,
- Administrating of local SDN controllers accounts

Details: The main role of the Security Manager (SM) component is the support for authentication and accounting services. SM should accomplish the authentication and corresponding services to the rest of the SDN Controller as well as the users and applications that interact with the controller. Moreover, it exposes interfaces for the administration of local SDN Controller accounts, in order to achieve authentication. The necessary methods for C.R.U.D (Create, Read, Update, and Delete) Users, Roles, and Domains are developed by the SM and also made available to other controller components as well.

To support the Use Case requirements, two operating modes are possible scenarios for the SM component:

- User/application authentication can be based on a local set of entered policies/users
- User/application authentication can be based on an external set of entered policies/users, i.e., using the OAuth2 authentication protocol. In the case of distributed authentication, the SM presents the tokens to the external server for validation.

3.3.8 VIM CONNECTOR

Overview: The component responsible for connecting with the backend VIM component.

Core Functionalities:

- ODL-OpenStack integration,
- Passing OpenStack's Neutron API calls to ODL's VTN manager via REST calls

Details: This section describes relevant reference points in the ETSI NFV architecture[12], as well as the set of compatible APIs employed for the realization of Network Services (NS). Specifically, the interfaces or plugins used by external SDN Controllers that allow them to interact with the Virtualized Infrastructure Manager (VIM).

The diagram shown in Figure 12 depicts a VNF instantiation example on a virtualization capable node, such as the ones composing the Network and Field layers of the SEMIoTICS infrastructure. Such message flow and triggered reference points follow the standard procedure defined by ETSI NFV Management and Orchestration.

In SEMIoTICS, the SDN Controller is considered an external entity to the NFV Management and Orchestration (MANO) framework presented in Figure 12. That is, the management of virtual network resources (e.g.: Virtual Tenant Networks), and the control of the underlying physical network are tasks handled by the SEMIoTICS SDN Controller. This brings benefits in terms of outage/saturation resilience, primarily due to the isolation of network services to separate hosts. But also allows for joint optimization of both overlay and physical network paths/resources, which could help satisfy SEMIoTICS Use Case requirements/constraints.

Infrastructure flexibility is one of the most relevant features provided by Network Functions Virtualization (NFV)[12] (either at the compute, storage or networking level), and network overlays play a crucial role in network virtualization. Through overlays, the SDN Controller is able to create different network topologies for each project[13], dubbed Virtual Tenant Networks (VTN), which are effectively isolated from each other.

[12] ETSI. (2014, February). ETSI.org: Network Functions Virtualisation (NFV); Architectural Framework (ETSI GS NFV 002 V1.2.1). Retrieved November 2018, from https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[13] Project, Use Case, or tenants refer to the same thing, and will be used interchangeable throughout this document.

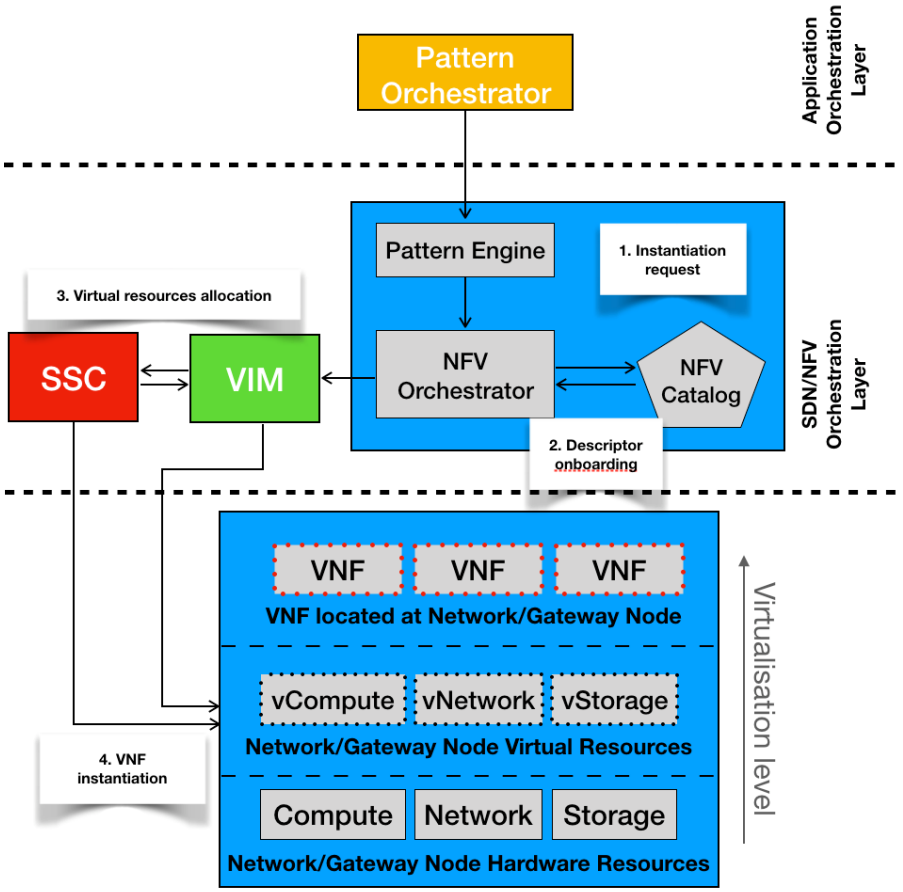


FIGURE 12 VNF INSTANTIATION EXAMPLE ON A VIRTUALISATION CAPABLE NODE WITHN SEMIOTICS FRAMEWORK

VTN allows the creation of network functions as virtual entities without having to consider the physical network. OpenDaylight[14] (ODL, the reference SDN Controller for SEMIoTICS) is equipped with a VTN module for interfacing with Virtualized Infrastructure Managers (VIM) such as OpenStack. The ODL VTN module is a policy manager that registers any tenant resource in the VIM via ODL's ML2 plugin[15,16], so any tenant configuration modification at the VIM is reflected in ODL, too. That is, by analyzing the information gathered for each tenant (network topologies, VNFs, MAC, IPv4 addresses, and so forth), VTN is able to replicate such a policy[17] using the VIM's exposed networking APIs and ODL's SBIs.

Referring to Figure 12, the reference point through which an external SDN Controller gathers tenants' information from the VIM is **Nf-Vi**. Moreover, the NFVO may also request Network as a Service (NaaS) for the

[14] OpenDaylight. OpenDaylight Lithium. Retrieved January 2019, from https://www.opendaylight.org/what-we-do/current-release/lithium
[15] OpenStack. ML2 plug-in. Retrieved January 2019, from https://docs.openstack.org/newton/networking-guide/config-ml2.html
[16] The ML2 plugin was created for ODL-OpenStack integration. It passes all OpenStack's Neutron API calls to ODL's VTN manager via REST calls (Toghraee, R. (2017). Learning OpenDaylight: The art of deploying successful networks. Birmingham: Packt Publishing Ltd)
[17] I.e. What nodes should be able to communicate with which ones.

instantiation of an already on-boarded Network Service (NS). Such API calls (NFVO-VIM) are realized through the **Or-Vi** reference point using the corresponding VIM APIs[18].

3.3.9 CLUSTERING MANAGER

Overview: A component with underlying Registry Handler used in state-keeping of other component's knowledge base, as well as for its strong consistent replication across the SDN controller instances for the purpose of fault-tolerance and high-availability.

Core Functionalities:

- Stores and replicates the knowledge state of the stateful controller components in a YANG-modelled data-store for purpose of highly available SEMIoTICS SDN Controller (SSC) operation.
- Enables the backup instances to operate as leader instances in case of a leader instance failure. Extended with support for Byzantine Fault Tolerance in SEMIoTICS.

Details: The issue of the SDN controller's single point of failure is resolved by means of state replication and fail-over to one of the backup controllers on failure. Network application relies on the availability of up-to-date state of the network controller replicas. The requirement of strong consistency is addressed by incorporating RAFT consensus in the synchronization procedure of the controller state across redundant controller instances. To ensure fault-tolerance even in the face of Byzantine/malicious adversaries, this component is also capable of operation in the Replicated State Machine mode. Namely, each instance of the SDN Controller executes the client operation and propagates its result to the underlying configuration targets that are capable of comparison of the resulting messages and thus allow for guaranteeing the integrity of the correct configuration. While unavailable in off-the-shelf OpenDaylight releases, this Byzantine Fault Tolerance mode of operation of the SDN Controller was investigated and will be discussed in more detail in D3.2.

3.3.10 SFC MANAGER

Overview: Service Function Chaining Manager used in Service Function Chains given the ordering and IP addresses of the nodes that are to be traversed by a tenant's traffic.

Core Functionalities:

- Chaining of network functions.
- Identifying an abstract set of service functions and their constraints that should be applied to packets.

Details: SFC Manager handles the service function chaining of network functions. It identifies an abstract set of service functions and their ordering constraints that should be applied to packets and/or frames selected as a result of classification. In the SEMIoTICS cases, service instances in service chains may include Firewall, IDS, DPI, and HoneyPot. These services can be deployed as the physical appliances or virtual machines running in network function virtualization infrastructures. They may be composed of one or multiple instances. SFC Manager is responsible for administrating the services chain and mapping the operator's/tenant's/application's requirements into service chains.

3.3.11 BOOTSTRAPPING MANAGER

Overview: Component used in initial flow configuration of just-connected switches, so to allow for seamless interaction with IoT devices (i.e., to enable flow rules for propagation of unmatched application packets up to the controller for the purposes of ARP-based end-device discovery, MAC Learning for best-effort services or similar).

Core Functionalities:

- Deploys the initial OpenFlow rules necessary to provide for in-band / out-of-band switch-controller connectivity.

[18] *OpenStack Docs: Networking API v2*. Retrieved from https://developer.openstack.org/api-ref/network/v2/

Triggers the installation of basic, non-QoS-guaranteeing flows for non-critical and basic infrastructural services where traffic specification is not available.

Details: Industrial SDN networks require a highly available control plane. The control plane may require an in-band or out-of-band realization depending on the exact Use Case. The wind park Use Case 1 assumes an in-band deployment, so to minimize the CAPEX related to out-of-band cabling requirements. By means of an automated network bootstrapping procedure, this component guarantees a robust and resilient control plane configuration at network runtime. To handle the impact of the data plane failures on the control plane flows, redundant control flow embedding is leveraged. While recent works propose slower, restoration-based techniques in industrial scenarios, industrial scenarios typically use 1+1 protection by duplicating controller-to-controller and controller-to-switch TCP-based flows on maximally disjoint paths, thereby ensuring zero packet loss for control flows.

3.3.12 (SDN) PATTERN ENGINE

Overview: Module responsible for retrieving network-specific rules and reasoning on them.

Core Functionalities:

- Inserting, Modifying, Executing, Retracting patterns in the SDN controller
- (Network properties') Drools reasoning

Details: The SDN Pattern Engine enables the capability to insert, modify, execute and retract patterns at design or at runtime in the SDN controller, ensuring the Secure, Privacy-aware, Dependable and Interoperable operation of the SEMIoTICS network layer at design and runtime. The Pattern Engine is based on a rule engine able to express design patterns as production rules. Enabling reasoning, driven by production rules, appeared to be an efficient way to represent SEMIoTICS patterns. For that reason, a rule engine is required to support backward and forward chaining inference and verification. Drools[19] rule engine appears to be a suitable solution to support design patterns by applying and extending the Rete algorithm[20]. More specifically, since the Drools rule engine is based on Maven, it supports the integration of all required dependencies with the ODL codebase, as well as the integration of the entities that interact with the controller to run Drools at design and at runtime. The Pattern Engine may send at runtime fact updates to the backend Pattern Engine, allowing the latter to have an up-to-date view of the SPDI state of the SDN layer and the corresponding components. Finally, PE enables the support of different components as required by the rule engine such as the knowledge base, the core engine, and the compiler. The procedure of the pattern module is depicted in the following Figure 13..

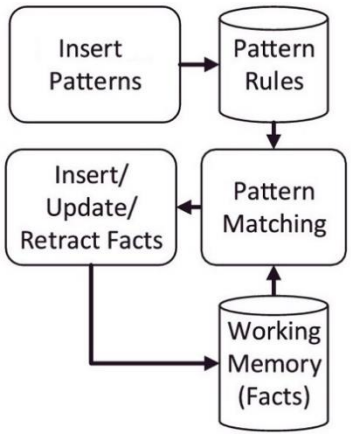


FIGURE 13 PATTERN ENGINE PROCESS

3.4 Field Layer

The field layer is responsible for hosting all types of IoT devices such as sensors and actuators as well as IoT gateway which provides a common way for communication and ensures enforcement of SPDI patterns in this layer. Generic gateway components are capable to work with any set of IoT devices that ensures the ability to deliver diverse Use Cases in various sectors.

3.4.1 SEMANTIC API & PROTOCOL BINDING

[19] Drools Business Rules Management System (BRMS) https://www.drools.org
[20] Charles Forgy: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: Artificial Intelligence, vol. 19, pp. 17–37, 1982.

Overview: Module responsible for binding different protocols and exposing common semantic API located at the Generic IoT Gateway layer.

Core Functionalities:

- Semantic Mapping of brownfield semantic models into IoT semantic models.
- Semantic configuration.
- Providing uniform standardized access to Thing's and their data.

Details: This functionality is needed in order to integrate brownfield devices into a common IoT access layer. Technology-wise, the functionality will be realized based on W3C Web of Things (WoT) building blocks, i.e., Thing Description, Binding Templates, and the Wot Scripting API.

Thing Description will be used to semantically describe field device resources, their interfaces, security meta-data, and so forth. For some of the brownfield devices there exist already various kinds of device descriptions. Therefore, in order to reuse existing semantics, we will need to provide a semantic mapping from brownfield semantic models into IoT semantic models as expected by W3C TD and iot.schema.org.

The mechanism of Binding Templates we will use in SEMIoTICS in order to provide bindings for various brownfield protocols (e.g., Profibus[21], Modbus, [22] etc.) into common Web application layer (e.g., HTTP, CoAP, etc.).

In SEMIoTICS we can use the WoT Scripting API to expose Things (field devices) that have been integrated over Binding Templates and described with Thing Descriptions. In this way, we can provide uniform standardized access to Thing's and their data, which can greatly reduce development effort for IoT applications at the Edge and in the Cloud.

The complete functionality of this component including also semantic configuration will be accessible over the Semantic Edge Platform. The platform will be based on the Node-RED tool, and thus will on one hand-side provide a graphic user interface for this component, and on the other hand, it can be used for developing Edge-level applications.

3.4.2 SECURITY MANAGER

Overview: Module responsible for granting access and necessary security checks at the IoT gateway.

Core Functionalities:

- Enforcing security policy decisions locally.
- Facilitating authentication of sensors and actuators.

Details: The Security Manager (SM) at the edge level serves as local frontend for the security manager at the backend layer; confer also to Section 3.2.7 for a more detailed explanation of the services provided by all security managers.

The two main purposes of the additional security manager in the gateway are:

- Facilitating authentication of sensors and actuators towards SEMIoTICS
- Enforcing security policy decisions locally

Sensors and actuators in many cases will be connected to the gateway using low-level protocols and technologies such as MQTT, Bluetooth or other protocols. In such cases, it simplifies the authentication of the clients if the gateway contains its own security manager which handles all relevant security concerns.

3.4.3 LOCAL THING DIRECTORY

[21] https://www.profibus.com/
[22] https://en.wikipedia.org/wiki/Modbus

Overview: The local repository of knowledge containing necessary Thing models.

Core Functionalities:

- Storing semantic description of Things locally.
- Providing an interface for semantic queries.
- Keeping all semantic meta-data up to date.
- Providing a digital representation of all physical assets.

Details: The purpose of the Local Thing Directory is to store a semantic description of Things locally in the Generic IoT Gateway. Backend Semantic Validator will be used to provide these descriptions in accordance with W3C WoT standard and iot.scheman.org. Once created, an application developer needs a tool to discover Thing Descriptions, and to easily find out whether a Thing can be used for a new Edge application that she wants to develop. Not only humans will use Thing Directory. Software components or machines may query Local Thing Directory too, e.g., when automatically generating a user interface for a Thing or when matching Recipe requirements with capabilities of Things. Local Thing Directory provides an interface for semantic queries and access to all Thing Descriptions stored locally. The directory keeps all semantic meta-data up to date. Thus, it provides a digital representation of all physical assets, accessible from a gateway. This includes device capabilities, configuration parameters of devices, contextual information (e.g., location, a feature of interest, etc.). The whole content of a Local Thing Directory will be synchronized with the Thing Directory, running in the Backend.

3.4.4 GW SEMANTIC MEDIATOR

Overview: Module responsible for the integration of brownfield semantics with IIoT semantics.

Core Functionalities:

- Integrating brownfield semantics with IIoT semantics.

Details: The goal of semantic integration (see SEMIoTICS deliverable D3.3) is to enable the realization of new IoT applications that have not been envisioned at the time of engineering of an existing automation system. To this goal, we work on a common semantic access layer between brownfield devices and new IoT devices. In order to integrate devices from both layers, we need to map and integrate semantics from existing brownfield devices into IoT or IIoT application semantics. Only then it will be possible to discover required Things when developing an application and to put them into semantically correct interactions.

Semantic Mappings is a layer that we introduce in the SEMIoTICS project in order to map and integrate brownfield semantics with IIoT semantics. In this layer, we have to provide a mapping knowledge, e.g., *Knowledge Packs*, which can be used to map semantics from a particular brownfield semantic standard into another IIoT standard. SEMIoTICS IoT Gateway will be able to install these Knowledge Packs and thus get enabled to integrate data and metadata from appropriate field device into a harmonized IoT access layer, based on the W3C WoT standard.

GW Semantic Mediator will be realized with W3C Thing Descriptions, which are serialized to JSON-LD standard format. Thing Descriptions will be semantically enriched with application-level, domain-specific semantics from iot.schema.org, and will be accessible over a local and backend semantic repository.

3.4.5 MONITORING

Overview: Module responsible for monitoring and predictive analytics at the IoT gateway level. This module interacts with bottom layer devices as well as with pattern engine and GW Semantic Mediator.

Core Functionalities:

- Fusion of intra- and cross-layer monitoring results generated by monitors that may exist on the platforms of different smart objects and components of IoT applications in order to detect violations

Details: This module is part of the SEMIoTICS Monitoring component. The SEMIoTICS Monitoring component is indeed a distributed computational entity having its modules distributed both in the cloud and at the edge. This section describes the peculiarities of modules available at the edge.

An edge monitoring module contributes to the overall objectives of the SEMIoTICS Monitoring component (see also section 3.6.6):

- To generate specific messages in response to the reception of a set of messages generated by the components of an IoT application and matching some condition specified in the monitoring component by a client application (Monitoring requirement).
- To guarantee that the messages needed to decide whether to generate a message can be produced by an IoT application and received by the monitoring component (observability property).

The specific contribution of an edge monitoring module is to allow the execution of part of the overall monitoring tasks close to the field devices generating the low-level events which are aggregated by the monitoring component. This strategy allows, hence, to send to a cloud monitoring module only the result of aggregations and not all the raw events generated at the field level. The consequently reduced number of transmissions provides a saving of those resources (i.e. energy, bandwidth) which are scarce within edge nodes (e.g. the mobile phone acting as a hub for the Body Area Network present within the SARA UC).

In general, an edge monitoring component will aggregate the low-level events generated by sensors directly connected to it (e.g. the devices connected via Bluetooth in the case of the above-mentioned SARA Body Area Network). However, if required, an edge monitoring module could aggregate also events generated by other edge monitoring modules.

3.4.6 (FIELD) PATTERN ENGINE

Overview: Module responsible for retrieving gateway specific rules and reasoning on them.

Core Functionalities:

- Inserting, Modifying, Executing, Retracting patterns at the field layer
- (Field layer properties') Drools reasoning

Details: The Pattern engine at the field gateway is able to host design patterns as provided by the Pattern Orchestrator located in the Application Orchestration Layer. Since the capabilities of the gateway are limited, this module will be a lightweight version of the backend and network reasoning engine. The patterns will be able to monitor and guarantee the Security, Privacy, Dependability and Interoperability properties locally, based on the data retrieved and processed by the monitoring module, the thing directory in the IoT gateway and based on the interaction as well with other components in the field layer. As such, it would be used to monitor and guarantee, for example, that secure communications are enforced between the IoT Gateway, sensors, and actuators on the field. The Pattern Engine in the gateway, similarly to the Pattern Engine in the SDN, may send at runtime fact updates to the backend Pattern Engine, allowing the latter to have an up-to-date view of the SPDI state of field layer and the corresponding components. Finally, the Pattern Engine in the gateway will keep stored the patterns in a local knowledge base that will be updated by the Pattern Orchestrator as needed and requested.

3.4.7 LOCAL EMBEDDED INTELLIGENCE

Overview: Module with Use Case specific logic (e.g. body area network GW in SARA UC) and embedded intelligence at the IoT gateway level.

Core Functionalities:

- Exposes the capabilities of the local analytics provided in SEMIoTICS at Gateway / Field Devices Level.
- Wrap and abstract Use Case specific logic and algorithms (enabling the Gait Analysis and the Generic IoT appliances set).

Details: The local embedded intelligence module is any software entity (i) executing a Use Case specific application logic (ii) relying on at least one of the services provided by the SEMIoTICS framework and (iii) deployed on a field device. In SEMIoTICS the specific application logic that has been identified according to the requirements are:

The Controller on board of the Robotic Rollator part of the SARA UC is an example of Local Embedded Intelligence since: (i) it address a requirement specific of the UC (i.e. to power the hub wheels in order to balance the user's weight) (ii) it relies on the GW Semantic Mediator to discover how to address the hub wheels available on the specific rollator (iii) it is deployed on the Single Board Computer (SBC) (i.e. a Raspberry Pi 3) on board of a Robotic Rollator.

The Supervisor component in the UC3 is another example of Local Embedded Intelligence because it (i) execute all the intelligent validations on changes (e.g. local vs global) specifically for the UC3; (ii) it relies on the Local Embedded Analytics component deployed on the IHES Sensing Unit and (iii) is deployed on a Raspberry Pi 3 at Field Device Level.

3.4.8 SEMANTIC EDGE PLATFORM

Overview: Module that enables a user to interact with SEMIoTICS IoT Gateway at the Field Level.

Core Functionalities:

- A user interfaces for configuring IoT Gateway.
- Development environment for creating new Apps.
- Semantic annotation of brownfield devices.

Creating Edge- and Cloud-based applications in SEMIoTICS.

Details: Semantic Edge Platform (SME) has been introduced in IoT Gateway (Field Layer). SME has multiple purposes in SEMIoTICS architecture. First, SME provides a convenient user interface for configuring SEMIoTICS IoT Gateway. That is, a user can choose a network interface, define an IP address range when scanning a network for new devices, and initiate the device bootstrapping process.

Second, SME provides a convenient development environment for creating new Apps with a newly bootstrapped device. SEMIoTICS IoT Gateway either provides a uniform API for a new device or re-uses an existing one. This API is automatically exposed over SME. After the bootstrapping process, there will be created a graphic component (a Node-RED node) based on this API. Thus, the device can be accessed over that node, and the node can be used in new applications right away. These device nodes are automatically created and installed in SME during the bootstrapping process.

Third, SME will provide a mechanism to semantically annotate brownfield devices. So created semantic descriptions from SME will be stored in both Local Thing Directory and Global Thing Directory.

Fourth, SME will be used as a framework for creating Edge- and Cloud-based applications in SEMIoTICS. The Recipe-Cooker component in SEMIoTICS will be integrated with SME. That is, it will be possible to instantiate a new application based on a Recipe. The process of discovering field devices and matching them with affordances from a Recipe will be supported via a machine reasoner that is integrated into SME.

3.4.9 SUPERVISOR AND LOCAL DB

Overview: Module responsible for data persistence and self-adapting mechanism of the Generic IoT System.

Core Functionalities:

- Manage a set of IHES Sensing Units
- Store data and events
- Implements part of Field Device-level intelligence

Details: The Supervisor component in SEMIoTICS is a dedicated module that implements the intelligent behaviors and policies required to manage a generic set of N Intelligent IoT IHES Sensing Units. Its main features are:

- To manage the IHES Sensing Units by interfacing with them through a dedicated MQTT Broker system and dedicated JSON protocol.
- To define and manage a common reference clock for time synchronization between all connected IHES devices, the Supervisor itself and the local storage DB.
- To implement dedicated frontend APIs to store data and events into the Local DB: to enable this feature the component parses and stores MQTT messages in the Local DB. This database has been implemented exploiting the time-series database InfluxDB[23].
- To dynamically allocate on-demand communication pipelines exploiting Node-RED[24] infrastructure.
- To implement some of the Field Device Level local embedded intelligence algorithms aimed at supervising the connected IHES nodes (e.g. for the UC3, when a change is detected by one more device, the Supervisor manages all the operations for the local/global validation).

All the specific Generic IoT communications at field device level use MQTT + JSON wrapped protocol redirected to a local MQTT broker (Eclipse Mosquitto[25]) running on the IoT gateway as an additional mapped service. On the contrary, all the external communication and DB queries are performed through standard HTTP GET/POST RestAPI. This has been specifically designed in SEMIoTICS framework in order to provide an abstraction interface and semantic bridging between the field device physical level where data and events are always pushed on a real-time basis and the upper backend services where typically the IoT platforms requires them "on-demand" through dedicated web service APIs or ad-hoc queries to the local DB.

3.4.10 USE CASE 1 FIELD DEVICES

Overview: Field devices and components related to UC1: "Local smart behavior in a wind turbine to provide value-added services".

Core Functionalities: Includes all the field devices related to UC1.

Details: The wind turbine use-case includes the following devices:

- An IIoT Gateway with computing capacity to instantiate IIoT applications as VMs. The IIoT gateway will be connected to various sensors as well as the simulated legacy wind turbine control system.
- A Power Line Controller (PLC) simulating a wind turbine control system. The PLC will facilitate connectivity to the legacy sensors and actuators.
- A small-scale wind turbine which will be directly controlled by the legacy control system. The small-scale wind turbine is used to visually demonstrate the Use Case.

3.4.11 USE CASE 2 FIELD DEVICES

Overview: Field devices and components related to UC2: "Socially Assistive Robotic Solution for Ambient assisted living".

Core Functionalities: Includes all the field devices related to UC2.

Details: The SARA field devices include:

- A smartphone (iOS or Android) acting as a hub for the Bluetooth wearable devices forming the Body Area Network subsystem of the SARA solution.
- A Robotic Rollator is a standard rollator frame equipped with hub motors and various sensors (e.g. Inertial Measurements Units, Laser Range finder). The Robotic Rollator hosts a Raspberry Pi 3 single-

[23] https://www.influxdata.com/products/influxdb-overview/
[24] https://nodered.org/docs/
[25] https://mosquitto.org/

board computer acting as a hub for the onboard devices connected via a Controller Area Network (CAN) Bus.
- An Aldebaran Pepper Robot which is a humanoid robot materializing the SARA Robotic Assistant subsystem. The Pepper Robot hosts an ARM computer dedicated to the management of the robot hardware and the execution of the software implementing the behaviors of the robot. Moreover, the robot hosts an Android tablet available to host applications having the need to present a graphical user interface (GUI) to the users.
- A Raspberry Pi 3 acting as a hub for the ZigBee devices instrumenting the Smart Environment subsystem of the SARA solution.

3.4.12 USE CASE 3 FIELD DEVICES

Overview: Field devices and components related to UC3: “Artificial Intelligent Embedded Sensing Platform”.

Core Functionalities: Includes all the field devices related to UC3.

Details: The IHES Generic IoT field devices are composed by:

- A set of N IHES sensing units mapped to an STM32 MCU prototype board equipped with a Wi-Fi and a sensor shield expansion board. During Cycle 1 demonstrator deployment an X-Nucleo-F401RE[26] board will be used. The board will run a bare-metal firmware that includes a library for the local analytics mapping and MQTT communication with a dedicated MQTT broker running on the IoT gateway
- A Raspberry Pi3 (or similar) ARM Board equipped with Embedded Linux OS where local MQTT broker, the Supervisor Service, and the Local DB are deployed. Part of the analytics will run on that Supervisor Service that will have also an MQTT client interface to interoperate with the GW Semantic Mediator of the SEMIoTICS architecture

3.5 External platforms’ components

3.5.1 MINDSPHERE

Overview: MindSphere platform component enhancing SEMIoTICS possibilities.

Core Functionalities:

- Enabling industrial customers to connect various automation systems and devices to the platform.

Details: MindSphere[27] is a Cloud platform customized for industrial IoT applications, which is developed by Siemens. MindSphere enables industrial customers to connect various automation systems and devices to this platform. The data is then accessible over the MindSphere Asset model and MindSphere API. Customers may apply different Apps in order to make decisions based on valuable factual information, e.g., predictive maintenance, automated production, vehicle fleet management, and so forth.

In the context of the Use Case 1 implementation (see Section 4.1), the MindSphere platform will be used as the backend system. The Backend/Cloud system will gather data from field devices over Generic IoT Gateway (see Section 3.4). The gateway also provides the semantics of this data. The same semantics will be used to create the MindSphere Asset model. Pre-processed data from wind turbines will be sent from the IoT Gateway, over Wind Park Control Network, to the MindSphere Backend. MindSphere Apps can then be applied to further analyze this data and visualize it. Depending on detailed analysis of the Use Case we may apply different Apps, e.g. for event-driven alarm detection and visualization of time series data.

[26] https://www.st.com/en/evaluation-tools/nucleo-f401re.html

[27] www.mindsphere.io

3.5.2 FIWARE

The interaction of SEMIoTICS framework with the FIWARE external IoT platform allows manage context information, incorporate generic enablers by simplifying the integration of IoT solutions.

3.5.2.1 CONTEXT BROKER GE

Overview: The main component of FIWARE allowing to manage context information.

Core Functionalities:

- Managing the lifecycle of context information (updates, queries, registrations, subscriptions).

Details: In FIWARE, the Orion Context Broker fulfills the pub/sub Message Broker functionality and must be federated with SEMIoTICS. FIWARE leverages the NGSIv2 Data Model and API, which relies on JSON representation to make data from multiple providers accessible for data consumers. The interaction with both data providers and data consumers are taking place via the FIWARE NGSI 10 context data API. SEMIoTICS must leverage the API for context queries, context subscription, and context updates to interact with the respective context elements (i.e., sensors and actuators) in a FIWARE domain.

3.5.2.2 FIWARE GES

Overview: Set of FIWARE components enhancing SEMIoTICS possibilities.

Core Functionalities:

- Accessing context elements in other domains.
- Generating context.
- Exploiting context information.

Details: *Set of FIWARE components for SEMIoTICS interoperability*

- *A **Context Provider*** is employed by FIWARE to access context elements in other domains (in this case SEMIoTICS). It can be registered via its URL as the source of context information for specific entities and attributes included in that registration, using the ORION NGSIv1 and NGSIv2 APIs. If FIWARE Orion fails to find a context element locally (i.e. in its internal database) for a query or update operation but a Context Provider is registered for that context element, then it will forward the query or update request to the respective Provider. In this case, Orion acts as a proxy, while the client that issues the request, the process is transparent. SEMIoTICS must implement the respective NGSI10 API (at least partially) to support query/update operations from FIWARE to a context element in the SEMIoTICS domain.
- **Context Producer**. A Context Producer (CP) is an actor (e.g., a temperature sensor) able to generate context. The basic Context Producer is the one that spontaneously updates context information, about one or more context attributes according to its internal logic. This communication is between CS and CB is in push mode, from the CP to the CB.
- **Context Consumer.** A Context Consumer (CC) is an entity (e.g. a context-based application) that exploits context information. A CC can retrieve context information sending a request to the CB or invoking directly a CP over a specific interface. Another way for the CC to obtain information is by subscribing to context information updates that match certain conditions (e.g., are related to a certain set of entities). The CC registers a call-back operation with the subscription for the purpose, so the CB notifies the CC about relevant updates on the context by invoking this call-back function.

A number of other FIWARE GE's are available. Variety of GE's functionalities are giving numerous opportunities to integrate FIWARE with SEMIoTICS framework depending on the maturity level of specific GE's.

3.5.2.3 CLOE-IOT

Overview: CLoE-IoT platform component enhancing SEMIoTICS possibilities.

Core Functionalities:

- Simplifying the integration of IoT solutions.

Details: The CLoE-IoT platform is part of Engineering's cloud offering (CLoE) and aims to simplify the integration of highly distributed, complex and robust IoT solutions exploiting computational resources both in the cloud and at the edge. The CLoE-IoT platform is intended to support the Engineering's products facing common IoT requirements (connectivity, device management, device security, data storage, etc.). CLoE-IoT embeds some of the FIWARE technologies (a.k.a. Generic Enablers) like the ORION Context Broker and the PROTON Complex Event Processor.

3.5.3 OPENHAB

Overview: OpenHAB is a flexible automation tool for integrating a multitude of devices that enhances SEMIoTICS possibilities.

Core Functionalities:

- interaction with external sensors
- data storage backends and chart libraries for sensor value visualization
- support for a scripting language to implement automation scenarios

Details: openHAB[28] is a flexible, open-source, technology-agnostic automation platform that is able to integrate a multitude of devices and systems. It includes various technologies into one single solution, in order to provide a uniform user interface and a common approach for automation rules across the entire framework, regardless of manufacturers and sub-systems involved. It communicates electronically with smart and not-so-smart devices, performs user-defined actions and provides web-pages with user-defined information as well as user-defined tools to interact with all devices. To achieve this, openHAB segments and compartmentalizes certain functions and operations. openHAB uses Apache Karaf to create an Open Services Gateway initiative (OSGi) runtime environment. Jetty is used as the HTTP server, which implements the Dashboard and Management GUI and also hosts the openHAB REST API. openHAB is extended through "add-ons" that handle the interaction with external sensors, data storage backends and chart libraries for sensor value visualization.

3.6 High-level component interactions

In this section, we present a high-level view of the interaction between the components. This is given within the layers as well all the communication between the layers. The diagram allows to identify the integration interfaces. Any details related to the specific endpoints, direction, and scope of communication will be given in the WP5 deliverables, which are related to the integration.

[28] openHAB official website: https://www.openhab.org

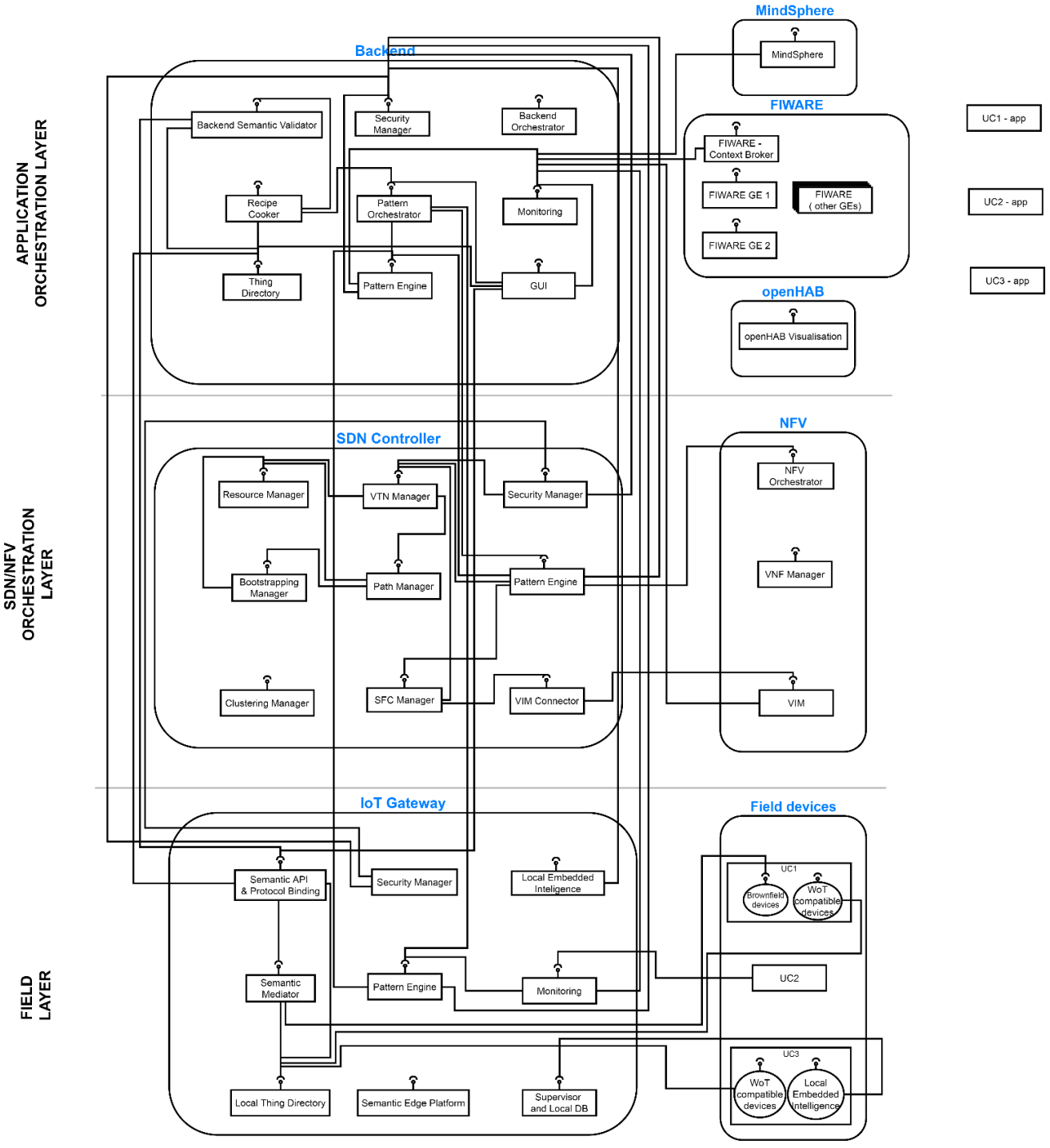


FIGURE 14 HIGH-LEVEL COMPONENT INTERACTION

In Table 2, an analysis of the interaction between the components of the architecture is given.

TABLE 2 HIGH-LEVEL COMPONENT INTERACTIONS

LAYER OF ENDPOINT	FROM	TO
BACKEND	Recipe Cooker	Backend Semantic Validator
	Backend Semantic Validator	Semantic API & Protocol Binding (Field)
	Pattern Engine	Security Manager
	Security Manager (Field)	Security Manager
	Local Embedded Intelligence (Field)	Security Manager
	Security Manager (SDN)	Security Manager
	Pattern Engine (Field)	Security Manager
	Pattern Engine (SDN)	Security Manager
	Recipe Cooker	Pattern Orchestrator
	GUI	Pattern Orchestrator
	GUI	Monitoring
	Pattern Engine	Monitoring
	Monitoring (Field)	Monitoring
	VIM (NFV)	Monitoring
	Context Broker (Fiware)	Monitoring
	MindSphere (MindSphere)	Monitoring
	VIM	Monitoring
	Backend Semantic Validator	Thing Directory
	Semantic API & Protocol Binding	Thing Directory
	Recipe Cooker	Thing Directory
	GUI	Thing Directory
	Pattern Orchestrator	Pattern Engine (Backend)
	Pattern Engine (SDN)	Pattern Engine (Backend)
	Pattern Engine (Field)	Pattern Engine (Backend)
SDN/NFV	Bootstrapping Manager	Resource Manager
	Path Manager	Resource Manager
	VTN Manager	Resource Manager
	SFC Manager	VTN Manager
	Pattern Engine (SDN)	VTN Manager
	Security Manager (SDN)	VTN Manager
	Security Manager (Field)	Security Manager
	Path Manager	Bootstrapping Manager
	VTN Manager	Path Manager
	Pattern Orchestrator	Pattern Engine (SDN)
	Pattern Engine (SDN)	SFC Manager
	SFC Manager	VIM Connector
	Pattern Engine (Backend)	NFV Orchestrator
	VIM Connector	VIM
Field	GUI	Semantic API & Protocol Binding
	Semantic API & Protocol Binding	Semantic Mediator
	Monitoring (Field)	Pattern Engine (Field)
	Pattern Orchestrator	Pattern Engine (Field)
	UC2	Monitoring (Field)
	Semantic Mediator	Local Thing Directory
	UC3	Local Thing Directory
	UC1	Local Thing Directory
	Semantic API & Protocol Binding	Local Thing Directory
	UC3	Supervisor and Local DB

4 USE CASE SPECIFIC ARCHITECTURE

This chapter showcases three demonstration scenarios (Use Cases) to be presented within the SEMIoTICS framework. Each Use Case is described from the perspective of the generic SEMIoTICS architecture with a special focus on showing the specificity of each Use Case and how dedicated components are leveraged in order to follow generic architectural guidelines. It is important to note that Use Case-specific architectures shown and described below, depict components which have been found useful from the general perspective of the respective Use Cases. Dynamic architecture sections presented per Use Case, are describing the goal for a final demonstration of the Use Cases. Within tasks T5.4, T5.5 and T5.6 feasibility study and validation of taken approach will be presented with a detailed description of the choice of components and functionalities which proves to be valid for the relevant Use Case. Finally, a methodology on how SEMIoTICS framework can support additional use cases is presented.

4.1 Use Case 1 – Wind Energy

This Use Case will showcase IIoT integration in Wind Park Control Network providing value-added services such as Local smart behavior and Monitoring etc. The current state of the art of Wind Turbine Controller in a Wind Park control network is typically an embedded or highly integrated operating system, which follows rigorously development and pre-qualification prior to deployment in the real world. Because of this slow process, new features, adding new sensors, actuators, and related advancements require several months or even years to be fully matured and operational in the field.

There are two sub-Use Cases, namely:

1) Embedded Intelligence on structured data: It refers to taking local action on sensing and analyzing structured data to find the inclination of a steel tower. When the nacelle is turned during a cable untwisting event (Sensing), the gravity acceleration (Ag) component measured by an accelerometer in longitude direction (Ay) will vary as a function of the inclination (Inc) of the steel tower. O&M personnel in remote control center wants to know the inclination of all the steel towers on a number of specific wind farms, as these details will have to be shared with the customer to monitor the deformation and fatigue of the steel. To find the inclination of a steel tower, a full cable-untwist procedure has to be activated. This happens, depending on wind conditions, 3-4 times a month. It is also possible to manually instruct the wind turbine to perform the unwind procedure. At the time of the unwinding-procedure a hi-frequency set of data is recorded. A relatively large amount of data is required to calculate the inclination. This datasheet needs to be sent back to the remote-control center to model and calculate the inclination. In SEMIoTICS, localized edge analytics will be applied which will result in semiautonomous IIoT behavior as only the container containing the algorithm and result of the inclination calculation is transferred to between the wind turbine and the remote-control center. The unnecessary data traffic between each turbine and remote-control center is greatly reduced.
2) Smart Actuation by sensing unstructured video/audio data: Within the turbine, there are many events that can be captured by IIoT sensors such as Grease leakage detection during normal operation or unintended noise detection when the turbine rotor is changing the direction in the line of wind to maximize energy production. The sensing of this unstructured data and acting locally to prevent any damage to the parts of the turbine, in the long run, will be of key importance. Localized analytics, as proposed in SEMIoTICS, will lead to smart actuation to protect the critical infrastructure of renewable energy resources.

4.1.1 USE CASE 1 – STATIC ARCHITECTURE

Use case specific components/hardware are shown in the following SEMIoTICS architecture layers namely, Field, Network and Application Orchestration Layer. The extensive description of the final interfaces of different architectural components will be detailed in final WP3 and WP4 deliverables respectively.

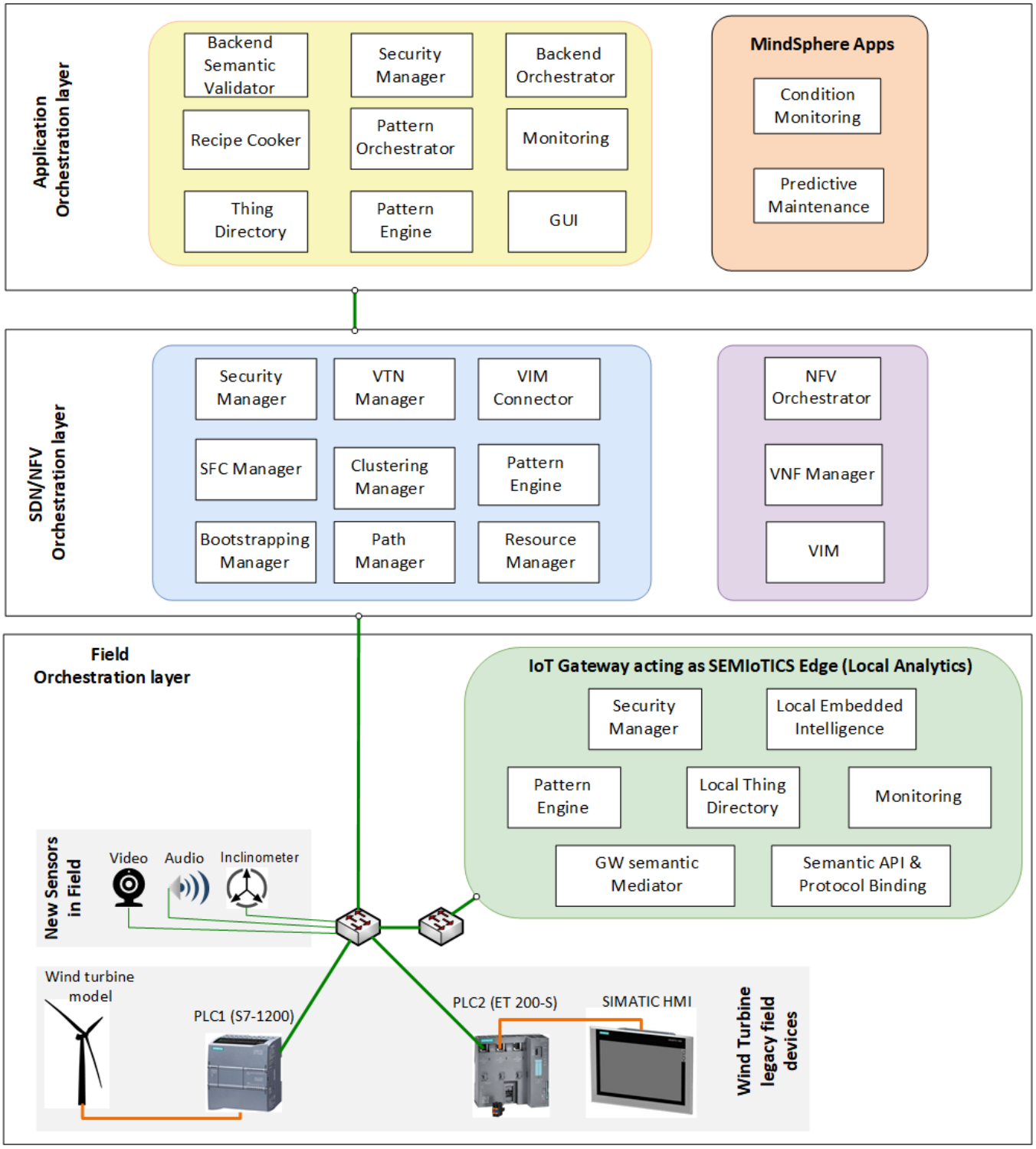


FIGURE 15 USE CASE 1 ARCHITECTURE

As depicted in Figure 15, the new sensors will be used in UC1 namely Video, Audio and Inclination measuring sensor for additional data. The IoT Gateway with its different components will do local analytics on the colle cted

data through different sensors and will take local action to avoid any damage to the rotating parts in the turbine in the case where the observed variables cross a certain threshold value. The details of the message flow can be seen in D3.1, D3.3 and D4.1 respectively. Table 3 lists the components and their role in the context of this UC1 -specific architecture.

TABLE 3 UC1 COMPONENTS

Architectural Components Used in UC1	Purpose of Deployment
Recipe Cooker	The Recipe Cooker is used in this Use Case to build the application flow of the AI pipeline for grease leakage detection. I.e., appropriate nodes are selected to (1) read video stream from a camera, (2) transmit images of stream via the network, (3) read in image-by-image, (4) transform image to tensor, (5) classify tensor using underlying neural network model (with 2 classes: “no grease” or “grease detected”). The transmission via the network needs to be configured according to minimum QoS.
Thing Directory	The Recipe Cooker selects devices for deploying the above-described application flow according to the metadata received from the Thing Directory.
Security Manager	Security Manager is used to authenticating external users and assign access roles to internal module functions to external interfacing components (i.e., the Pattern Orchestrator).
VTN Manager	VTN Manager is used to deploying the virtual network used to isolate basic connectivity services (e.g., Local Thing Directory – backend Thing Directory), as well as to enable QoS-constrained service deployment in dedicated virtual tenant networks.
Clustering Manager	While not used to showcase the resilient SSC functionality in UC1 specifically, the Clustering Manager is generally used to host individual components’ state of reservations in the YANG-modeled data-store.
Pattern Engine	Pattern Engine is used to parse the connectivity and SPDI-related pattern instances provided by the Pattern Orchestrator and enforce and monitor their validity at runtime.
Bootstrapping Manager	Used to bootstrap the network from an initially unconfigured state using in-band or out-of-band control plane channel. The Use Case 1 demonstrator will showcase the out-of-band network deployment.
Path Manager	Used to identify network paths fulfilling the QoS requirements required by the application service. Alternatively, if QoS requirements are left unspecified, the Path manager identifies the shortest paths computed using Dijkstra algorithm. Path Manager will be used to compute all paths and provide them to Resource Manager for embedding for interconnecting all Use Case 1 distributed components, e.g., Greenfield Sensors <-> IoT

	Gateway <-> Brownfield Actuator or PLC1 <-> PLC2 communication as per Figure 15.
Resource Manager	Resource Manager is used for embedding all paths computed by the Path manager into the physical / data plane, using OpenFlow 1.3.1 flow syntax.
Brownfield and greenfield devices (video camera, microphone, inclinometer, controllers, etc.)	Field devices provide and consume data in UC1 (sensors, actuators, and controllers). For example, a camera is a device that we will bootstrap and make it available (via IIoT Gateway) for the application flow by Recipe Cooker (the AI pipeline for grease leakage detection). The controller is another field device that will be interfaced over the gateway. For instance, the application for detection of grease leakage will be able to stop a wind turbine over the controller that is exposed via the gateway.
Semantic API & Protocol Binding	Semantic API & Protocol Binding integrates brownfield devices into a common IoT access layer. The interface will be accessible over IIoT Gateway, and mostly it will be used via Semantic Edge Platform (for Edge applications) and via Recipe Cooker (for Cloud applications).
GW Semantic Mediator	GW Semantic Mediator integrates brownfield semantics with IIoT semantics. The mediator provides a means to configure brownfield devices with IIoT semantics. In UC1 the examples of brownfield devices will be data points to control a wind turbine, i.e., start the turbine, stop the turbine, etc. The IIoT semantics (iotschema.org) is created with common ontology editors, e.g. Protégé. Once the semantic model has been created it can be used by the mediator to configure the brownfield devices.
Local Thing Directory	Local Thing Directory makes the knowledge about Things (e.g. Field devices) discoverable for applications. Local Thing Directory will be typically accessed by Semantic Edge Platform (for Edge applications).
Semantic Edge Platform	Semantic Edge Platform provides a user interface for the functionality of IoT Gateway. In the scope of UC1, it will be possible to scan the network and bootstrap a video camera, microphone, inclinometer, controllers, etc. Further on, we will semantically configure brownfield device with GW Semantic Mediator (integrated into Semantic Edge Platform), and to create a Thing Description (TD). TD for the wind turbine will be stored in the Local Thing Directory, which is also configurable over the Semantic Edge Platform.

4.1.2 USE CASE 1 - DYNAMIC ARCHITECTURE – MESSAGE FLOW AND COMMUNICATION

The new sensors will be used in UC1 namely Video, Audio and Inclination measuring sensor for additional data. The IoT Gateway with its different components will do local analytics on the collected data through different sensors and will take local action to avoid any damage to the rotating parts in the turbine in the case

where the observed variables cross a certain threshold value. The details of the message flow can be seen in D3.1, D3.3, and D4.1 respectively.

In the bootstrapping process, we distinguish two different classes of devices. The first class consists of devices that already have a Web-based RESTful interface and are described by W3C Thing Description. The second class comprises of all other devices that yet need to be made accessible over a Web-based RESTful interface. These devices do not have a semantic description, or it exists, but needs to be mapped to standardized semantic IoT models. This is a case, for example, with brownfield devices, see Figure 16. In order to realize IoT applications, it is convenient to map these brownfield descriptions into descriptions based on standardized IoT semantic models.

Let us consider now a sequence diagram of activities that occur during the bootstrapping of the WoT device, see Figure 16. The user performs the first step during the initialization of a new device. This assumes the provision of information such as an IP address, device capability, domain of use, location, etc. Since the device already has a Thing Description (TD), this information is directly put in its TD. The device can then be registered with SEMIoTICS IIoT Gateway (with GW Semantic Mediator, which is an internal component of the Gateway).

If a brownfield device needs to be initialized, then a user in addition to previously mentioned information needs to specify metadata related to the communication protocol and the encoding format. This information will be an important part of a Thing Description and is used by SEMIoTICS IIoT Gateway to realize a protocol binding.

In comparison to the sequence diagram for bootstrapping and interfacing field devices in deliverable D3.3, here we have added a component called Semantic Edge Platform (SME), see Section 3.4.8. SME eases the interaction with SEMIoTICS IoT Gateway, including the interaction with the Local Thing Directory too.

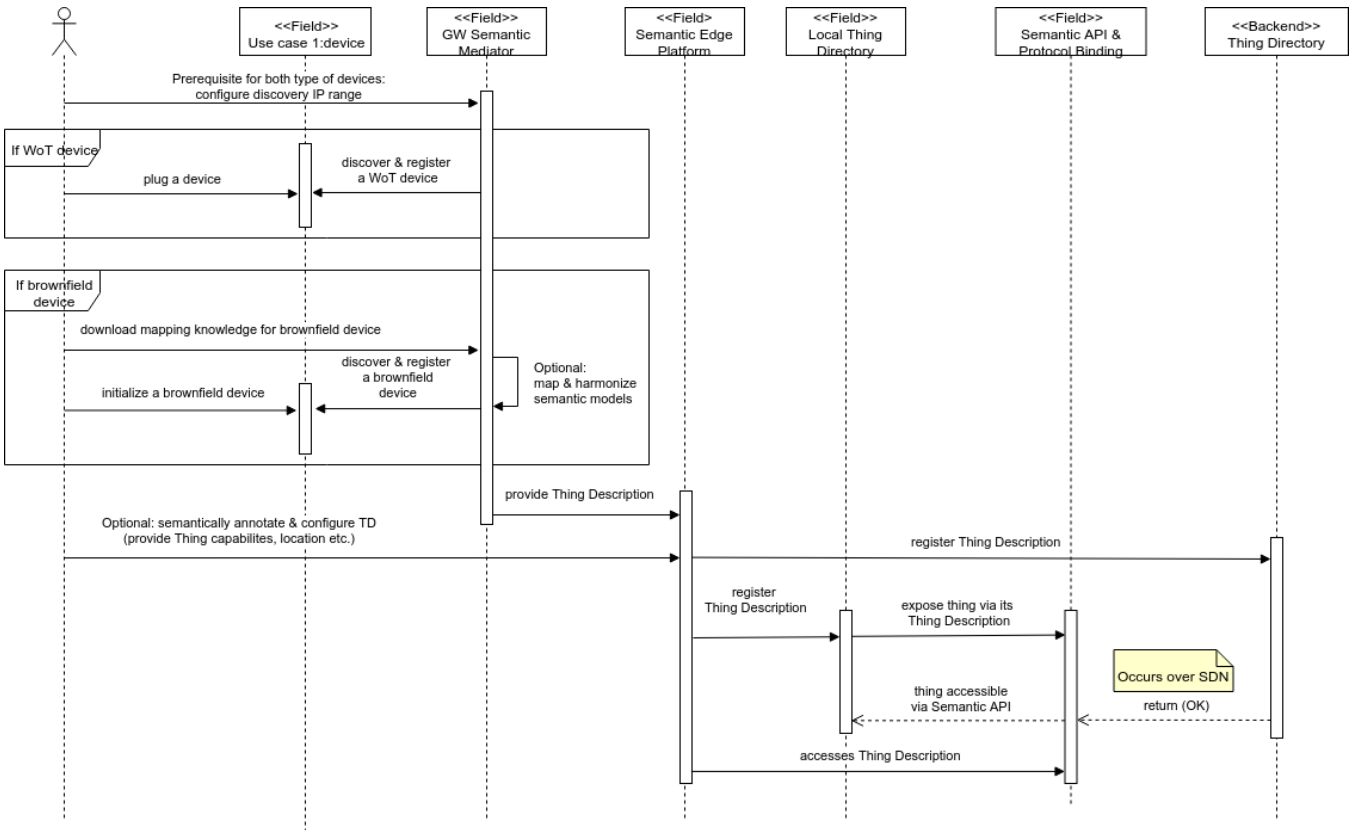


FIGURE 16: SEQUENCE DIAGRAM FOR BOOTSTRAPPING AND INTERFACING SEMIOTICS FIELD LEVEL DEVICES

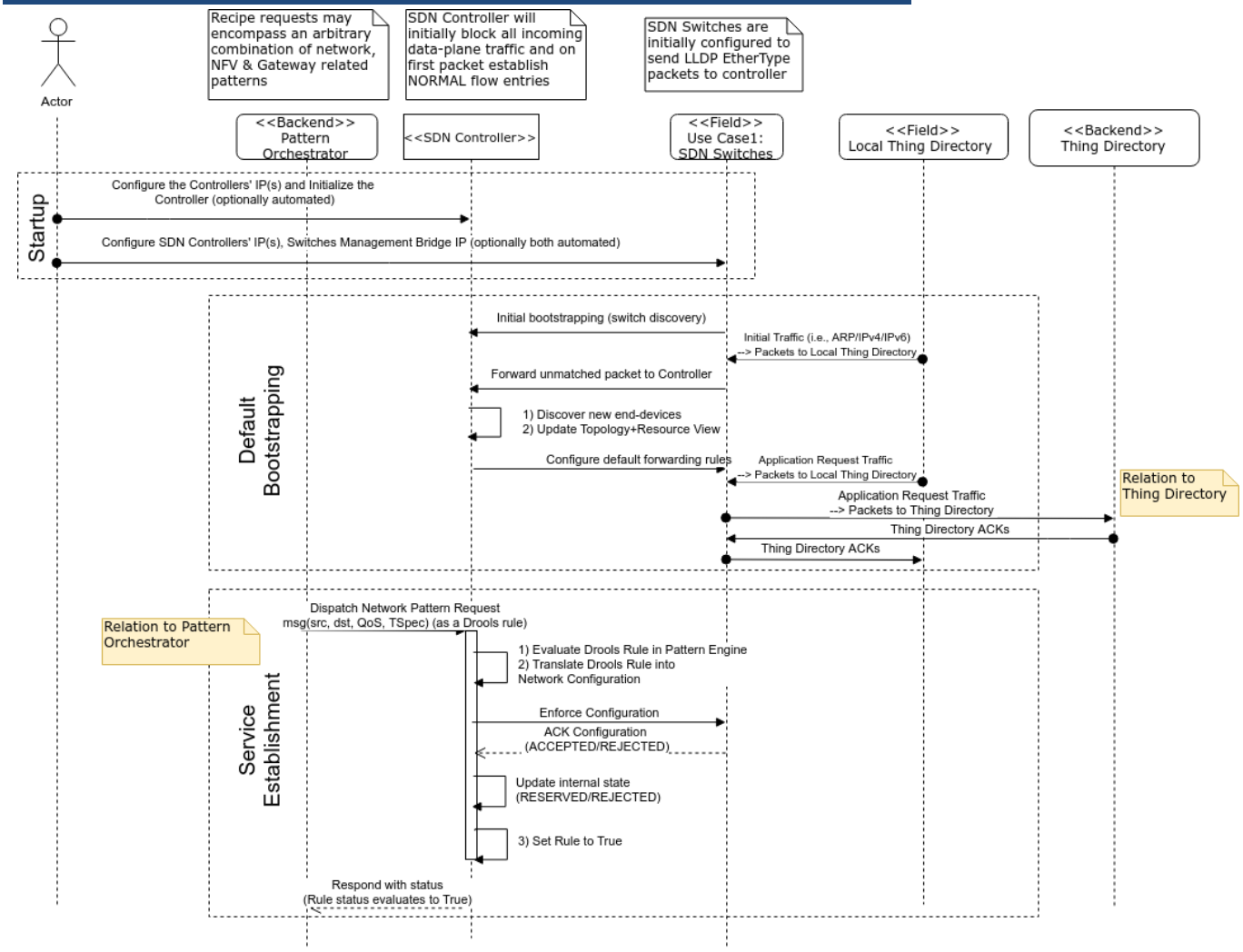


FIGURE 17 OPERATION OF SSC IN USE CASE 1 (GENERALIZED)

SSC is used to bootstrap the network and enable connectivity between infrastructural services deployed in the Use Case 1 as well as the pattern evaluation and its enforcement. Figure 17 showcases the operation of SSC during bootstrapping time and runtime operation, to be fully showcased in Use Case 1 as well. We encompass all granular components of the SSC in a single actor (SDN Controller) for the brevity of visualization.

SSC discovers the devices in its network using the OpenFlow protocol. Following the discovery and establishment of control sessions, SSC listens for incoming packets from end-devices (hosts), e.g., the IIoT Gateway and SCADA application and updates its host database with the corresponding attachment points behind which the hosts are located. The SSC then proceeds to install the required flow rules, so to enable basic infrastructural services, i.e., a network connection between the IoT Gateway and backend, in order to provide for a possibility of Local Thing Directory of the IoT Gateway to report its status and the capabilities of its field devices (i.e., sensors and actuators) in the Thing Directory in the Backend.

To embed QoS-constrained services, SSC is enabled to parse, evaluate and enforce connectivity patterns specifying a set of connection properties, encompassed in invariants, that describe the intent which is to be fulfilled by the underlying data substrate. The SSC subsequently processes the connectivity requests, specified as Drools rules at its northbound interface and validates the viability of its enforcement in its internal modules.

To adhere to the rule specification, the controller internally evaluates the topology state for a path that would fulfill the named criteria, i.e., using a combination of routing algorithms, designed to consider a set of constraints (including delay and bandwidth). If the connectivity is possible given the current amount of resources, the controller propagates the enforcement request for individual flow rules to the lower-level southbound interface (i.e., the OpenFlow plugin). The OpenFlow plugin is then in charge of sending the according to flow modification messages to the corresponding data plane switches.

4.2 Use Case 2 – Assisted Living

This Use Case employs the SEMIoTICS technologies to develop an Information and Communication Technology (ICT) solution aimed at sustained independence and preserved quality of life for elders with Mild Cognitive Impairment or mild Alzheimer's disease, with the overall goal of delaying institutionalization: supporting both 'aging in place' (individuals remain in the home of choice as long as possible) and 'community care' (long-term care for people who are mentally ill, elderly, or disabled provided within the community rather than in hospitals or institutions).

A detailed description of the requirements for the SARA solution can be found within deliverable D2.2 - "SEMIoTICS usage scenarios and requirements".

4.2.1 USE CASE 2 – STATIC ARCHITECTURE

The SARA UC design envisages two groups of modules: cloud modules (deployed in the cloud and drawn above the SEMIoTICS Platform box) and field modules (deployed on field nodes and drawn above the SEMIoTICS Platform box). In this design, the SEMIoTICS Platform is envisaged to offer the services (e.g. networking, monitoring, security) facilitating the integration of the modules belonging to the two groups.

Cloud modules include:

- **Localization and Mapping:** providing the services for localization and mapping especially needed by the mobile robots (i.e. the Robotic Rollator and the Robotic Assistant). The Localization and Mapping service is hosted by the CLoE-IoT platform since it uses it.
- **Gait analysis:** utilizes the CLoE-IoT platform to access the measurements taken via the Body Area Network and the Robotic Rollator. The result of the analysis performed by this component is stored in the Patient Health record via the AREAS Service Bus.
- **Head Pose and Gaze Estimation:** providing the services for the estimation of pose and gaze needed by the Human-Robot Dialog Manager to support the interaction between humans and the Robotic Assistant.
- **Object Detection and Tracking:** providing the services for detection and tracking of objects needed by the Human-Robot Dialog Management.
- **Human-Robot Dialog Management:** manages the interaction between the Robotic Assistant and humans. It relies on the information provided by other cloud components (Localization, Object Detection/Tracking and Pose and Gaze Estimation) to decide which behavior of the Robotic Assistant should be activated/deactivated in order to support a smooth interaction with the user.
- **Assistive Tasks Management:** represents the core of the SARA solution since it is responsible to orchestrate and, if needed, to configure the processes providing the Assistive Tasks aimed at Patients and Caregivers. The Assistive Tasks Management takes its decisions relying on the information produced by other modules (e.g. Fall Detection Head & Pose and Gaze Estimation).
- **Tele-monitoring:** provides the services enabling an operator of the Call-center Operator or a Medical Expert, in case of emergency, to rely on the video cameras of the Robotic Assistant to set up a real-time video connection to inspect the scene of a possible incident.
- **AI Services:** represent the collection of third parties cloud platform offering AI services (e.g. IBM Watson) needs by some of the modules within the SARA UC (e.g. speech-to-text service needed by the Robotic Assistant).

Field modules comprise of:

- **Fall detection:** is the module responsible for the detection of patients' falls.

- **Gait Analysis:** is the module to perform a preliminary analysis of the measurements concerning the gait and to forward the result of that analysis to the corresponding cloud service. The Gait Analysis module is deployed both within the hub of the Body Area Network (i.e. a smartphone) and the Robotic Rollator since both devices can take measure relevant of the analysis.
- **Weight Balancing:** is the module that tries to balance the patient weight by controlling the hub motors of the Robotic Rollator. The Weight Balancing module is deployed both in the BAN and the Robotic Rollator to implement a dual redundant control scheme providing fault tolerance and contributing to patient safety.
- **Navigation:** is the module responsible for providing the robotic components (i.e. Robotic Rollator and Robotic Assistant) with navigation capabilities. This module relies on the localization and mapping service available from the CLoE-IoT platform.
- **HR Dialog Management:** is the counterpart of the HR Dialog Management module available in the cloud. It is intended to support a simple form of dialog not requiring access to extended computational resources available in the cloud.
- **Human Activity Monitoring:** is the module deployed within the Smart Environment gateway and is responsible for monitoring the occupant movements and locations (e.g. by tracking the entrance of people in and out from rooms). Results from monitoring can trigger automated actions like entering security mode if there are no occupants. The monitoring may concern also the outside (e.g. garden) for privacy and security.

Figure 18 highlights (in blue color) the main possible interactions between the edge nodes and between end nodes and backend services:

- The smartphone (BAN gateway) supports the communication between the field nodes backend services by means of LTE connectivity.
- The Home Gateway supports the communication between the field nodes backend services by means of IP connectivity.
- the smartphone (BAN gateway) communicates with Home Gateway to access the services provided by the Smart Environment subsystem and, for reliability purposes, provide additional connectivity to between the field nodes a backend service.
- The smartphone (BAN gateway) and the Robotic Rollator communicates (via Wi-Fi) to support joint functionalities (e.g. to enable redundant weight balance control there is the need to exchange the inertial data between the smartphone and the Robotic Rollator).
- The Robotic Assistant (Pepper robot) communicates with the Home Gateway to access the backend services (e.g. to request the execution of compute-intensive AI task), to access the Smart Environment services (e.g. to increase the luminosity of the environment to facilitate computer vision tasks) and coordinate (via the coordination service) its activity with those of the other field devices.
- The Robotic Assistant (Pepper robot) and the Robotic Rollator communicate to coordinate their activities in the context of specific task (e.g. during navigation).

The specific components leveraged in UC2 are listed in Table 4.

TABLE 4 UC2 COMPONENTS

Architectural Components Used in UC1	**Purpose of Deployment**	
Recipe Cooker	SPDI Patterns - Compliance with GDPR	Description of SARA IoT orchestrations
Pattern Orchestrator		Translation of SARA IoT Service orchestrations into SPDI patterns
Pattern Engine		Enforcement of SPDI patterns relevant for SARA
Security Manager		Enforcement of security policies
Monitoring		Monitoring events as requested by the Pattern Engine
Thing Directory	Semantic Interoperability Mechanisms – Uniform	Discover devices registered in the system
Semantic API & Protocol Binding		Accessing the resources of devices in a uniform way

GW Semantic Mediator	access to heterogeneous devices	Map and integrate semantics from ZigBee devices into iot.schema.org semantics
Backend Semantic Validator		Detect and resolve possible semantic conflicts that may exist within SARA IoT orchestrations
Local embedded intelligence	Embedded Intelligence and local analytics – Uniform access to heterogeneous devices	Extraction and encoding of gait features collected by means of the Robotic Rollator
SDN Controller	Network Management - Dynamically adapt the network to the unpredictable computational load	Management of network routes to fulfill requests from pattern engine.
NFV Infrastructure		Chaining of Virtualised Network Functions to fulfill requests from pattern engine.

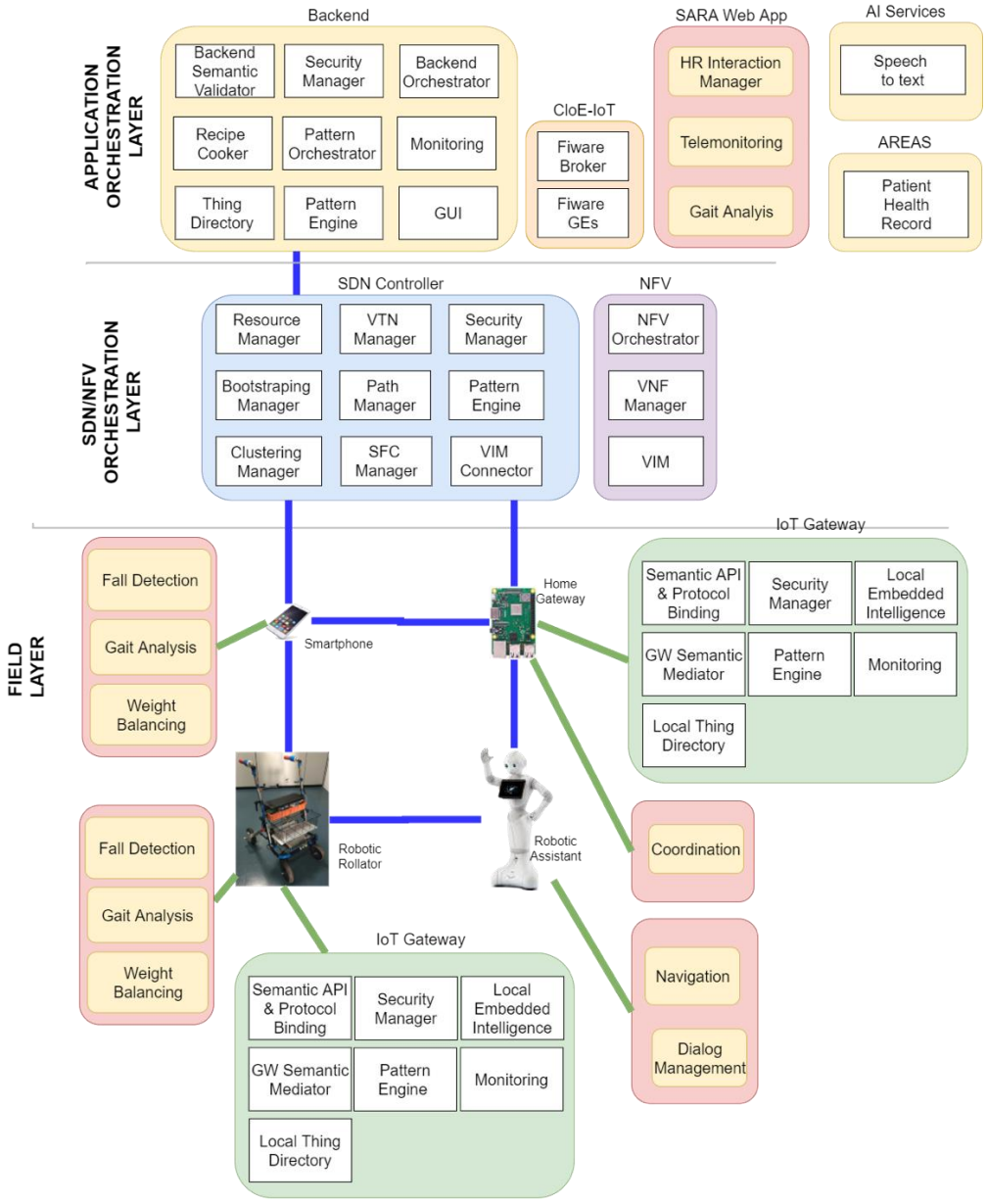


FIGURE 18 UC2 - MAIN MODULES OF THE SARA UC

4.2.2 USE CASE 2 – DYNAMIC ARCHITECTURE - MESSAGE FLOW AND COMMUNICATION

4.2.2.1 USE CASE 2 DYNAMIC ARCHITECTURE – SEARCH AND RESCUE INTERACTION

The Search and Rescue interaction (Figure 19) occurs whenever SARA detects a possible Fall Event. The interactions develop across three main phases: Early Warning, Search, and Rescue.

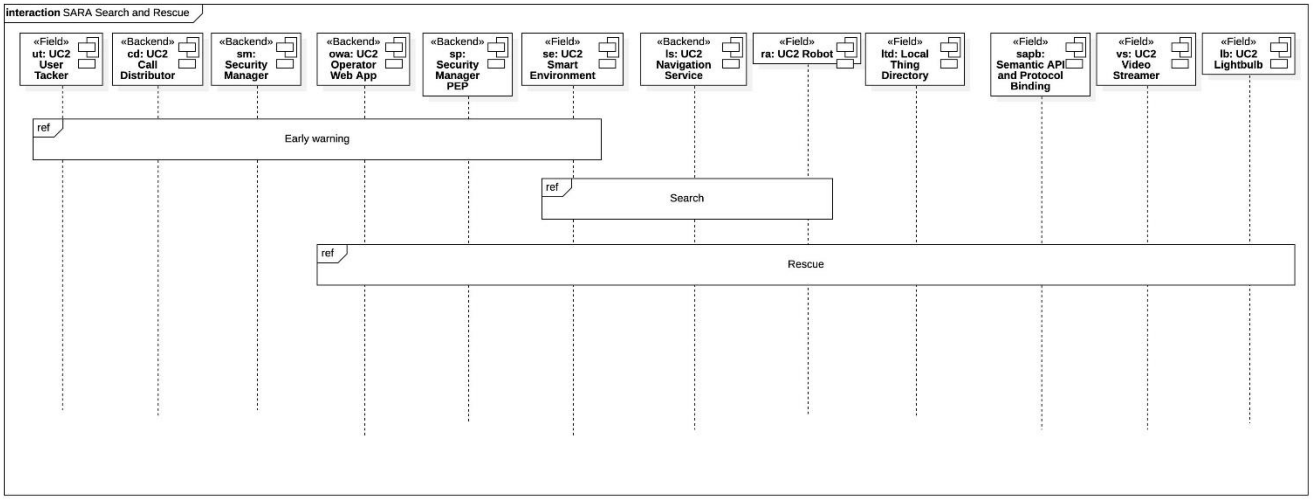


FIGURE 19 SARA SEARCH AND RESCUE INTERACTION

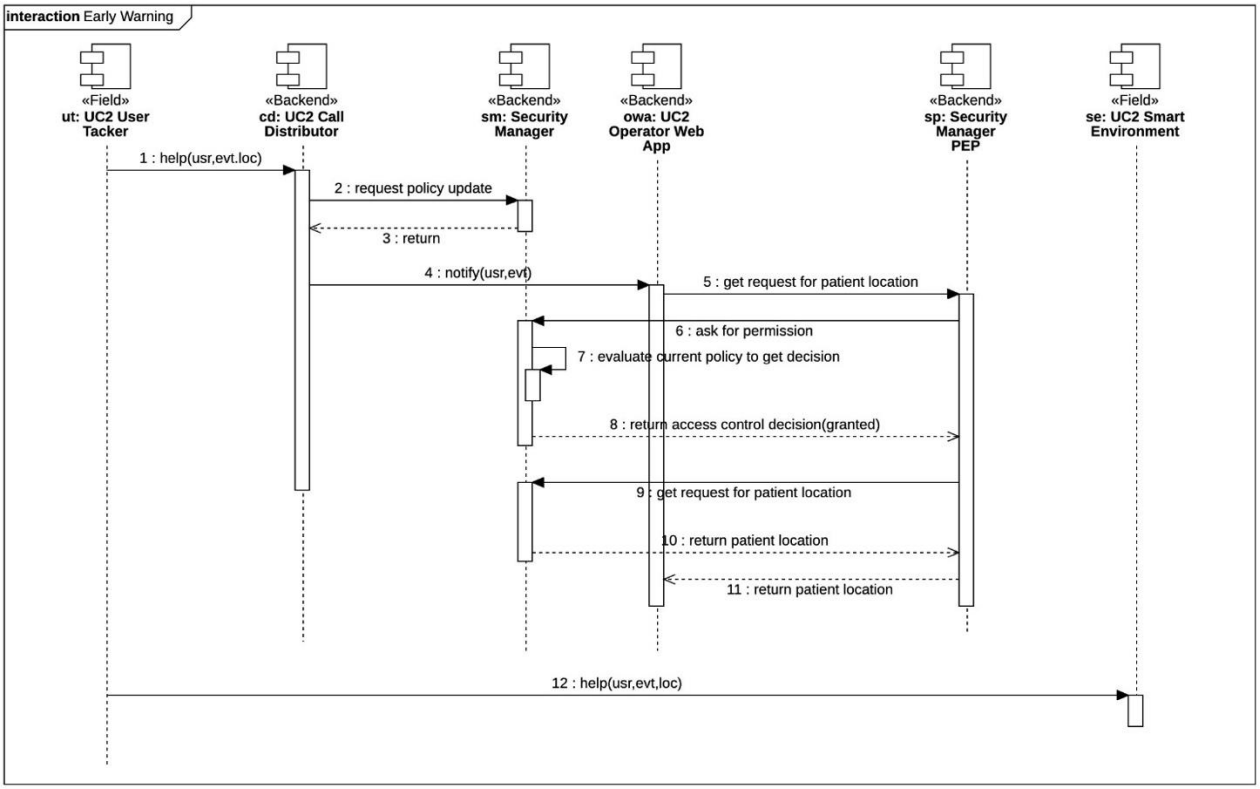


FIGURE 20 EARLY WARNING INTERACTION

The Early Warning phase (Figure 20) initiates with the User Tacker sending and help request to the Call Distributor service. The Call Distributor requests to the Security Manager to update the current security policy to enable the Operator Web Application to access user location. Once the update is done the Call Distributor selects the operator responsible to manage the call and notifies her the help request via the Web App instance she is using. Since the patient's location is sensible information the Operator Web Application requests to the Sidecar Proxy to retrieve the patient's location stored within the Call Distributor. The Sidecar Proxy asks the Security Manager permission to access the patient's location on behalf of the Operator Web App. Since the security policy has been changed during the previous steps the permission is granted. Once received the permission the Sidecar Proxy access and return to the Operator Web App location of the user.

The objective of the Search phase (Figure 21) is to allow the Robotic Assistant to reach the location where the (possible) fall event occurred. This phase is initiated by the Smart Environment sending to the Robot the request to move to the location of the event. The Robot requests to the backend Navigation Service to be guided to the target location indicated by the Smart Environment. Once the Navigation Service is ready the Robot starts to notify the readings from the onboard depth camera. Using this information, the Navigation Service estimates the current position of the Robot and returns to it the indication of the new velocities. When the Navigation Service estimates that the Robot has reached the desired location, it notifies to the Robot the event. The Robot, in turn, notifies the achievement of the navigation goal to the Smart Environment.

It is worth noting that even if the description of the Search interaction does not mention any SEMIoTICS component the interaction between the SARA system and SEMIoTICS indeed exists. In fact, it is the SEMIoTICS SDN Controller that takes care of the timely delivery of the messages exchanged between the Robot and the Navigation Server. The point is that the proper configuration of the network transport infrastructure is done by the SDN Controller implicitly without explicit request by the application logic. That's why in Figure 21 the SDN Controller is not shown.

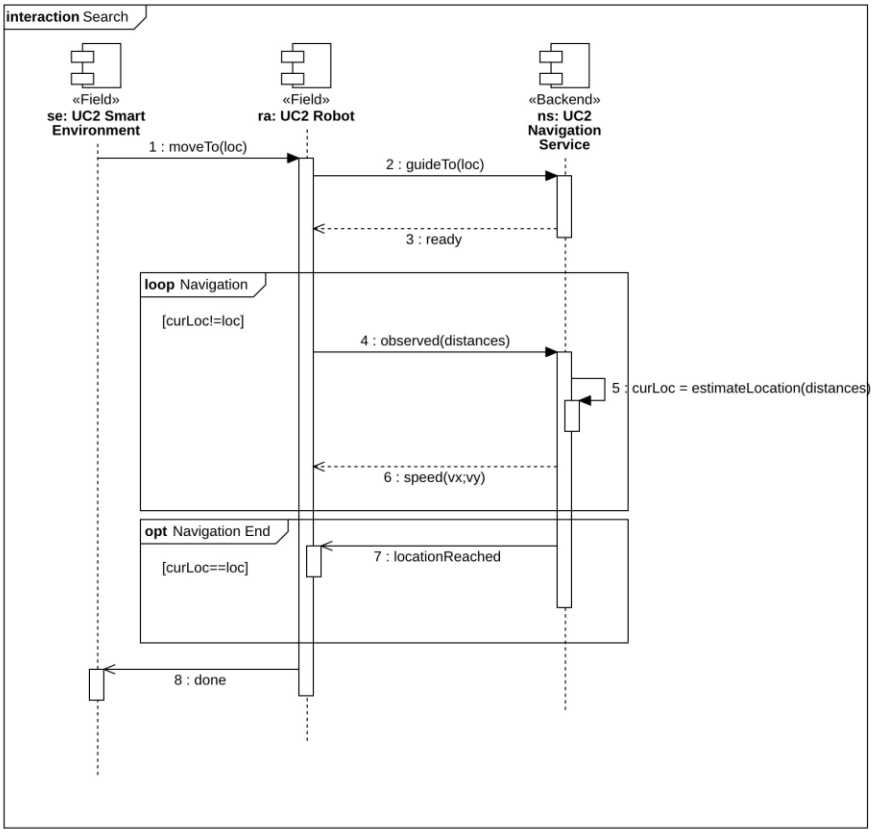


FIGURE 21 SEARCH INTERACTION

The objective of the Rescue interaction is to allow the Call Center operator (selected during Early Warning) to directly see which is the current situation at the location of the fall event. Once the Operator Web App has received the user location from the Sidecar Proxy (see Early Warning interaction) the Web App uses the Thing Directory to retrieve the video cameras available at the location of the event. The Thing Directory returns the identifier of the Robot. The Operator Web App requests to be notified when the Robot will reach the location.

As soon as the Robot receives the notification that it is at the event location, the Robot updates its current location in the Local Thing Directory. The Local Thing Directory takes care to communicate to the Backend Thing Directory the updated location of the Robot. This update results in the Thing Directory notifying the new location to the Operator Web App. At this point, the Operator Web App can subscribe to the video stream originating from the onboard camera of the Robot. The result of this subscription is that the Video Streamer will start to push video frames towards the Operator Web App. Also, in the case of this interaction, it is worth to note that the timely delivery of video frames from the Video Streamer to the Operator Web App is guaranteed by the SEMIoTICS SDN Controller. Since, as explained in the case of the Search interaction, the configuration of the network transport infrastructure is done without explicit request by the SARA solution, the SDN Controller is not shown in Figure 21.

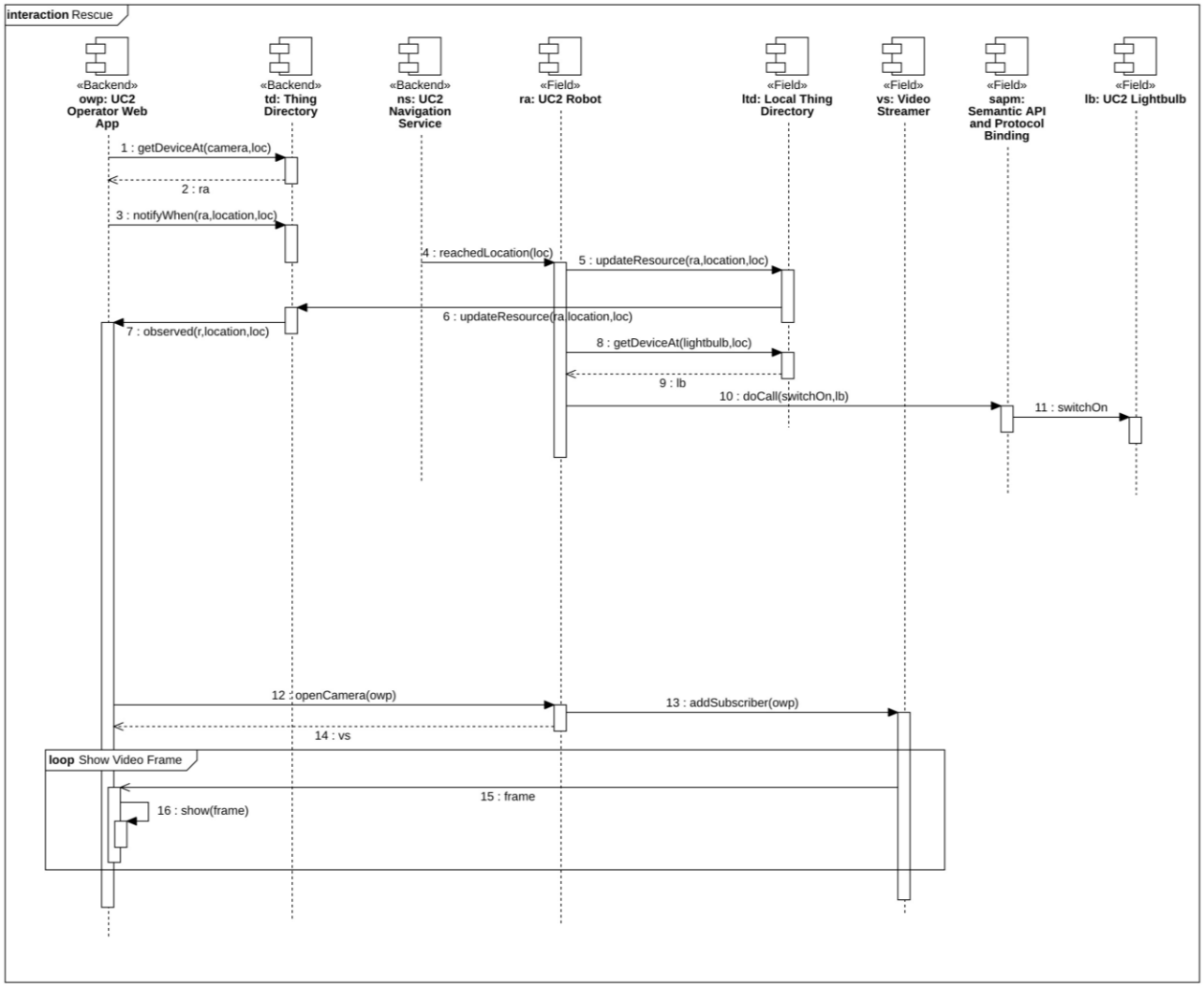


FIGURE 22 RESCUE INTERACTION

4.2.2.2 USE CASE 2 DYNAMIC ARCHITECTURE – REMOTE GAIT ANALYSIS INTERACTION

The Remote Gait Analysis Interaction Figure 23 is started whenever a Doctor schedules by means of the AREAS® Web Booking System a new gait assessment session for a patient. A gait assessment schedule prescribes a time period for the execution of the assessment session. During this period (usually a week) the Web Booking System sends a reminder message to the User Tracker App hosted by the mobile phone of the user. If the user decides to delay the execution of the exercise the decision is notified to the Web Booking System which re-schedule the exercise. In the case, the user agrees to start the exercise the decision is notified to the Web Booking System and the data collection phase is initiated. The data collection phase is described by the Gait Data Collection Interaction.

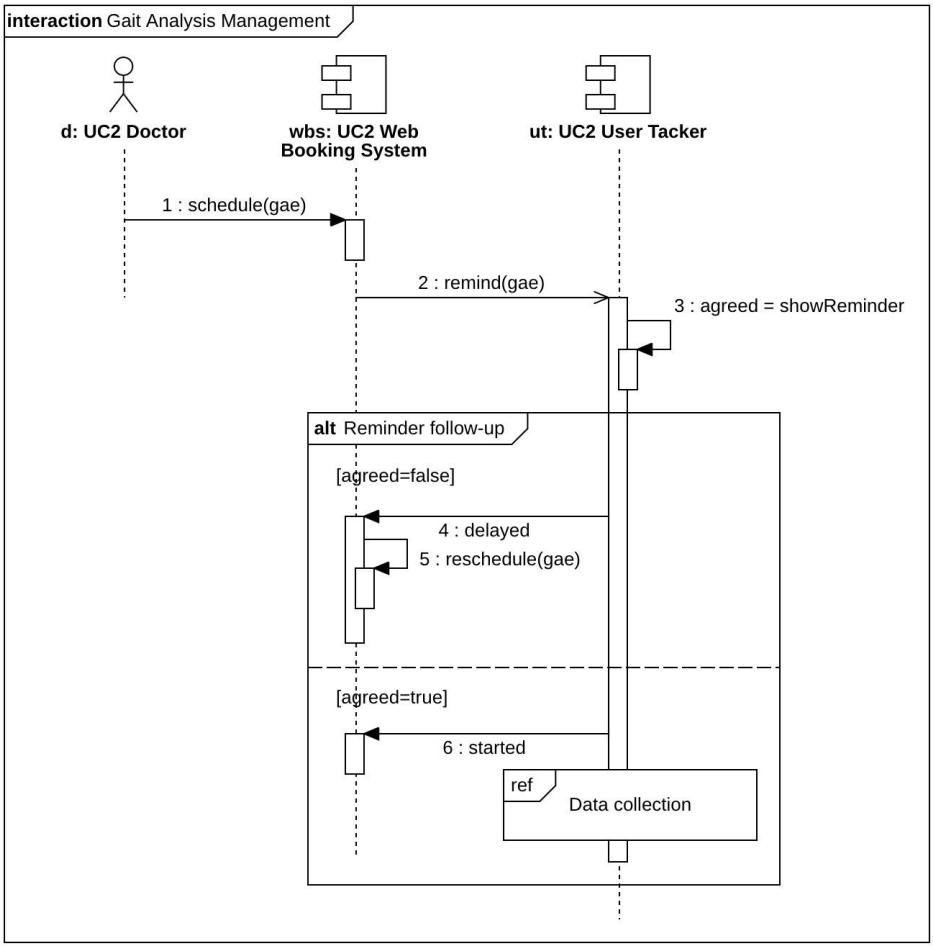


FIGURE 23 GAIT ANALYSIS MANAGEMENT

The Gait Data Collection Interaction Figure 24 occurs during the execution of a physical exercise (e.g. five meters walk) done using the SARA Robotic Rollator. The interaction is initiated by the User Tacker which is an App hosted on the Hub (typically the patient's mobile phone) of the Body Area Network. The User Tracker signals to the Gait Analyzer (a SARA application hosted by the Robotic Rollator) the start of the exercise. Consequently, the Gait Analyzer puts an observation on the Range Sensor.

Whenever the Range Sensor sends a new measure to the Gait Analyzer, the Gait Analyzer checks whether this represents the end of a gait. If this is the case the Gait Analyzer encodes the time series representing a single gait and stores it in the Local Storage Service. The Local Storage Service is a service of the CLoE-IoT platform hosted by the Robotic Rollator. The "Observe and Store" loop is interrupted by the arrival of an

endExercise message from the UserTacker. Once the “Observe and Store” loop is terminated, the Gait Analyzer clusters the time series collected by the “Observe and Store” loop. The clustering is done using the SEMIoTICS Local Embedded Intelligence component also hosted on the Robotic Rollator. The Local Embedded Intelligence component retrieves the times series to be clustered from the Local Storage Service. The Gait Analyzer stores in the backend CLoE-IoT Storage Service the clusters created by the Local Embedded Intelligence.

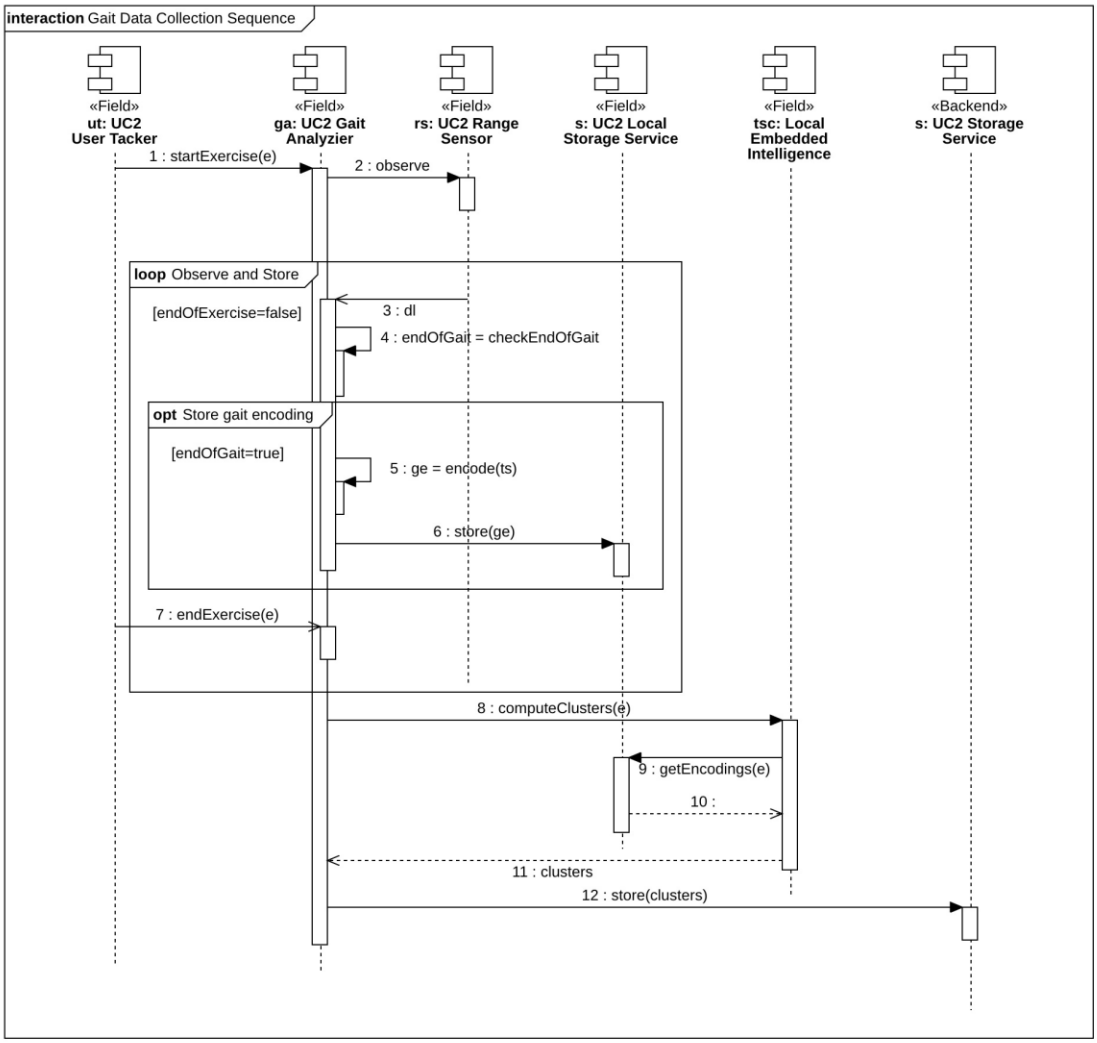


FIGURE 24 GAIT DATA COLLECTION

4.2.2.3 USE CASE 2 DYNAMIC ARCHITECTURE – SERVICE FUNCTION CHAINING

Smart assisted living monitoring systems situated at homes can facilitate the monitoring of patients’ activities and enable the remote provision of assisted services. They improve the quality of elder population well-being in a non-obtrusive way, allowing greater independence, maintaining good health, preventing social isolation for individuals and delay their placement in institutions such as nursing homes and hospitals. In this context, one of the scopes of the second use case of SEMIoTICS is to provide security guarantees through the traffic forwarding via different

network security functions by applying the Service Function Chaining (SFC; as detailed in deliverable D3.2) concept on an ambient assisted living scenario, whereby a smart home environment.

Investigating this use case, and considering the different types of traffic reaching the backend where the chaining of services will take place, the following intricacies are observed: traffic originating from the mobile phone is of low trust and low priority, as the mobile device is not trusted (e.g., can be easily targeted by malicious software) and the reporting from the BAN devices has low bandwidth and latency requirements; traffic from the Robotic Rolator are of medium trust (relatively restricted devices) but high priority, as messages need to arrive in a timely fashion (e.g., in case a patient fall is detected); the smart home traffic is of medium trust (commercial devices which may be vulnerable to, e.g., incorrect configuration) and of low priority, and finally; traffic from the robot are of high trust (closed/restricted device) and high priority, as low latency and relatively high bandwidth is required to enable seamless interactions with the robot.

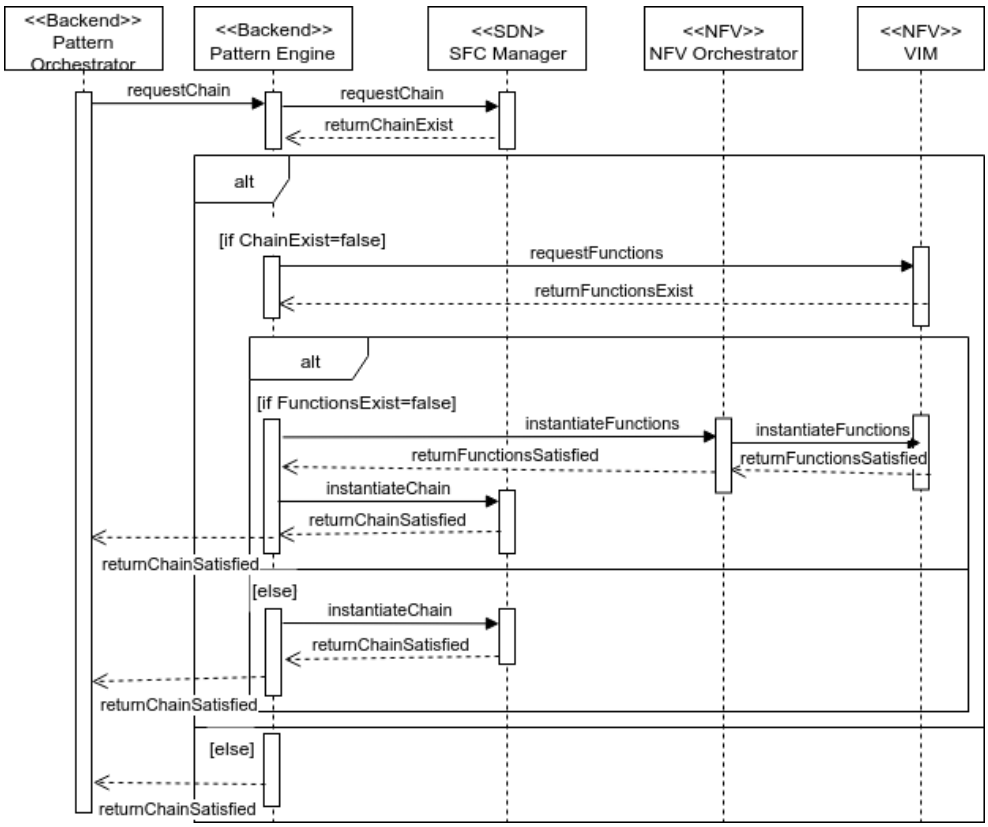
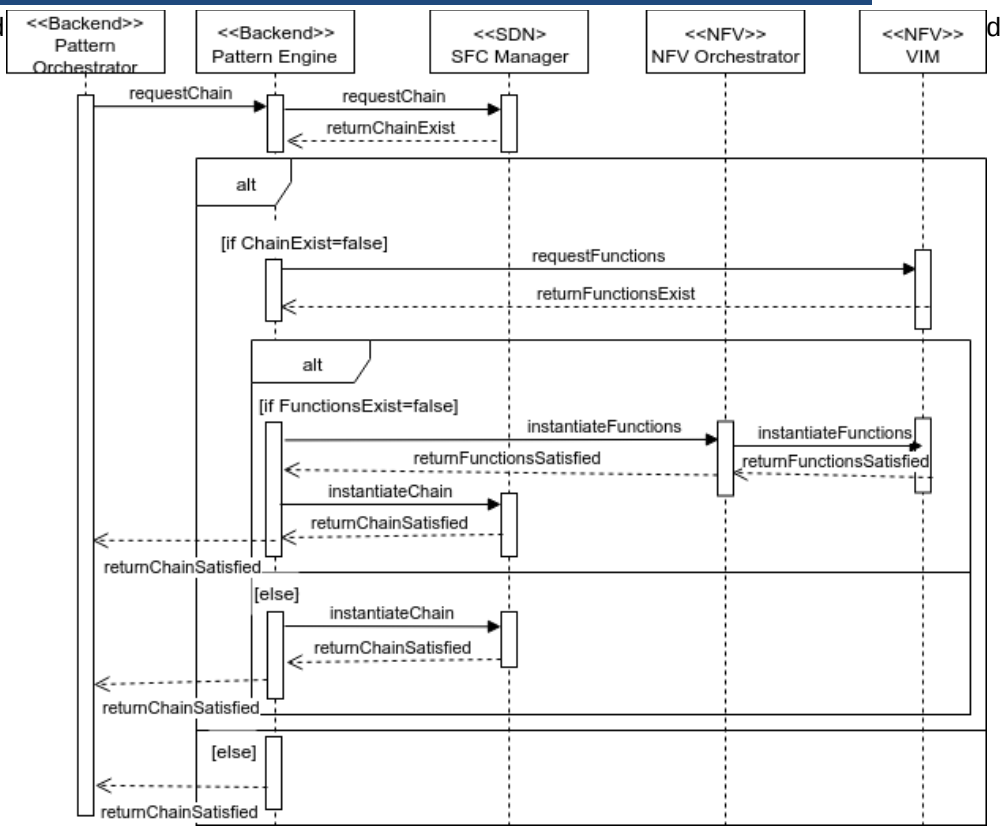


FIGURE 25 SEMIOTICS SARA SFC CONTROL FLOW

The proced 

d on the patterns is depicted in

Figure 25 including the following interactions with the components of the SEMIoTICS architecture:

- The Pattern Orchestrator forwards a specific chain request to the pattern engine for forwarding the traffic between entities through a specific chain of functions.
- The Backend Pattern Engine forwards this request to the SFC manager which is located in the SDN Controller responding to the pattern engine whether the chain exists or not.
    - If the chain exists, then a respond of the chain satisfaction is returned to the pattern orchestrator.
    - If the chain does not exist, then a requested is forwarded to the VIM asking whether the service functions exist or not.
        - If functions exist in the VIM, then
            - the chain can be instantiated in the SFC Manager,
            - a respond of the chain satisfaction is returned to the pattern orchestrator.
        - If functions do not exist in the VIM then,
            - function instantiation request is forwarded to the NFV Orchestrator, which is responsible to instantiate them in the VIM,
            - the chain can be instantiated in the SFC Manager,
            - a respond of the chain satisfaction is returned to the pattern orchestrator.

Considering the above, there is significant motivation to leverage the flexibility provided by SFC to define specific service chains for each type of traffic. By applying the previously described procedure of chain instantiation, the SARA Use Case can be extended to support traffic forwarding through specific service functions. That includes traffic forwarding for the different type of traffic exchanged between the different actors as following and depicted in Figure 26:

- **Chain1: Mobile phone** traffic having low trust and low priority e.g. delay of exercise in the Gait Analysis Interaction (SFC1 – *Phone:* FW -> DPI -> IDS -> Web Booking System)
- **Chain2: Mobile phone** traffic having high trust and high priority e.g. emergency call in the Early Warning phase of the Fall Management Interaction (SFC2 – *Phone*: FW -> Load Balancer -> Call Distributor)
- **Chain3: Call Center Operator** traffic having of medium trust but high priority e.g. access to the onboard camera of the robot in the Rescue phase of the Fall Management Interaction (SFC3 – *Operator:* Load balancer -> FW -> Robot)

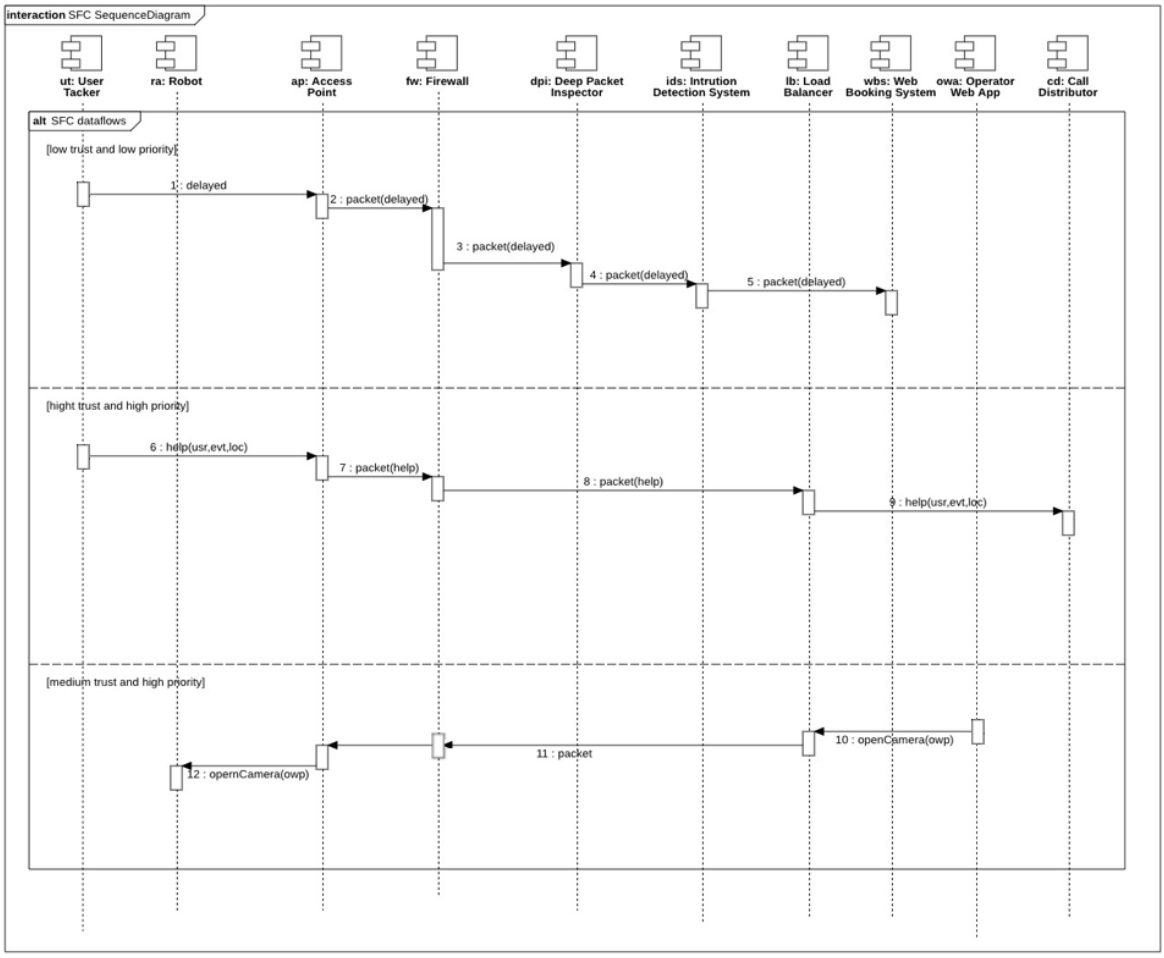


FIGURE 26 SEMIOTICS SARA SFC DATA FLOW

Based on the provided dynamic instantiation of service chains and service functions through the pattern engine, the potentiality within the SARA use case can be increased and extended by the support of additional service chains to enable traffic classification through different combinations of service functions to guarantee different secure end to end traffic forwarding.

4.3 Use case 3 – Smart Sensing

IoT embedded Things are more and more named as being smart devices. “Smart” usually is associated with some Things that show some form of intelligence or adaptability, being able to better interoperate in the environment in which they are in. Unfortunately, the current way of making these objects “smart” is through a quite naive approach where the physical device is locally executing dummy local algorithms and is always connected to some cloud infrastructure were more complex algorithms are running for providing the feeling of

being “smart”. Therefore. these devices transmit sensed data to the cloud without any analytic being performed locally and without showing remarkable forms of computational intelligence. An example is Microsoft Azure or Amazon AWS cloud platforms and the related ecosystem. The major weakness of these solutions is that they are poorly scalable and rely on the main assumption the connection is always present. An IoT thing is “really” intelligent if it has local capabilities to learn from, and act upon the environment it is sensing.

The IHES use case offers an interesting specular approach to this scenario (somehow influenced by “Edge Computing” or “Pervasive Computing”). The main assumption is that intelligent data processing shall take place at the sensor level, and that distributed data classification and data aggregation is a key aspect for massive system scalability. Moreover, in this use case algorithms derived from AI techniques will be deployed at Gateway, down to MCU level, allowing as well to online/self-learn from the environment: this latter a quite challenging aspect by itself on the AI field. On these systems, distributed data computing and semantic interoperability are key aspects of design and in this respect, SEMIoTICS offers the perfect deployment testbed. Research on this field, especially for the self-learning distributed part, is highly fragmented with solutions exploring different but specific aspects of the problem[29] [30], e.g., the properties or the architecture of the system, the challenges or the adaptation mechanisms. A holistic view of the problem and a mainstream methodology for the design are still missed. These systems are distributed intelligently interacting devices in which physical and software components are deeply intertwined, each operating on different spatial and temporal scales, exhibiting multiple and distinct behavioral modalities. Such systems consist of intelligent heterogeneous sensor networks, monitoring physical processes and processing real-time data to extract relevant information with very limited supervision, learning from them and aggregating compact information related to their time-varying nature. Intelligent data processing can happen at the single sensor, group of sensors or at the server level, to learn from time-varying heterogeneous data, trigger events on them, take decisions on what type of intelligent behaviors must be adapted to new conditions so adapting themselves. The main characteristic of this new generation of distributed intelligent systems is the ability to closely interact with the environment, in which they operate, learn from it (without human supervision), and enable automatically self-adaption to new time-varying operating conditions at different levels of the architecture. For a better understanding of the scenario description, and the two sub UCs, please refer to D2.2 (section 2.3.2.2).

4.3.1 USE CASE 3 – STATIC ARCHITECTURE

Local Embedded Intelligence is a key aspect of SEMIoTICS. It can enable the infrastructure to migrate from the cloud-centric computation-intensive mainstream approach to a more scalable one where some part of the currently used AI/ML algorithms are moved to the edge (i.e. at Field Device-level): in SEMIoTICS, they are mainly deployed in the IoT Gateway and the Field level Node Devices. These aspects are considered and covered in full details as part of the task 4.3 activities and will be reported in the final deliverable D4.10. These algorithms can be deployed and instantiated within a specific architectural component named “*Local Embedded Intelligence”* component (see Figure 6, SEMIoTICS general architecture) that implement all of those algorithms and interoperate with all the other components identified at IoT gateway level primarily to make those capabilities available to the other network and backend layers of the architecture. The major challenge at the field devices level of UC is the deployment of such a *Local Embedded Intelligence* component directly down to each single field device node, tightly coupled with data gathered from sensors.

Local analytics algorithms are adapted as well at this level of the architecture mainly for data-reduction purposes: the raw data acquired from the sensor are processed locally in order to derive from the relevant info sent as events to other layers. The communication interfaces are designed in order to ensure event-driven end-to-end semantic interoperability by adopting widely used standards such as iot.org, JSON data format. The focus of the horizontal generic IoT UC3 is thus to provide a specific working deployment at IoT sensing

[29] C. Krupitzer, F. M. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing, vol. 17*, pp. pp. 184–206.

[30] Roveri, C. A. (2017). The (not) far-away path to smart cyber-physical systems: An information-centric framework. Computer, vol. 50, no. 4, (pp. 38–47).

nodes of a data-driven unsupervised (data) monitoring infrastructure. This can be achieved as part of WP4 activities by defining the specific low-level architecture and incremental mapping of all required local analytics algorithms to sustain those technologies that is integrated as part of WP3 activities.

The developed algorithms are derived from well-known approaches in the field of AI, Statistical Analysis, Causal inference, and prediction analysis. Differently from widely used algorithms deployed at the central Cloud level, light-weight versions of those are derived for accounting the specific needs of these constrained domain architectures. Most of those algorithms are implemented as key components of the IHES generic Use Case demonstrator whose major goal is to provide those enabling new local analytics-enabled technologies to SEMIoTICS architecture. Depending on the specific device that host this component there are different deployments of the same component functionalities done by exploiting their specific device capabilities/limitations/ecosystems. The main factors that drive light-weight porting are mainly due to limitations in real-time constraints, memory, computation, power consumption, and existing legacy software middleware support. In particular, for the IHES tailored component a specifically designed version of a subset of those named algorithms made available as part of the bare metal firmware (FW) at microcontroller sensing unit directly (i.e. a set of dedicated STM32 MCUs tightly coupled with communication and sensing capabilities expansion board shields – the IoT Thing).

A monitoring sample app deployed at Raspberry Pi3 level or aside PC for monitoring the status of the whole IHES system and to be used for supporting the specific demo that is implemented as a reference implementation to demonstrate the system capabilities. The monitoring web app basically reports the status of the system in a specific deployment of the technology, in order to report environmental relevant events (anomalies on temperatures and humidity, abrupt changes on luminosity). From this web app template, other 3rd party apps could be derived by interfacing the IHES system through other SEMIoTICS components. An overview of the envisaged system architecture at the field level is reported in Figure 27.

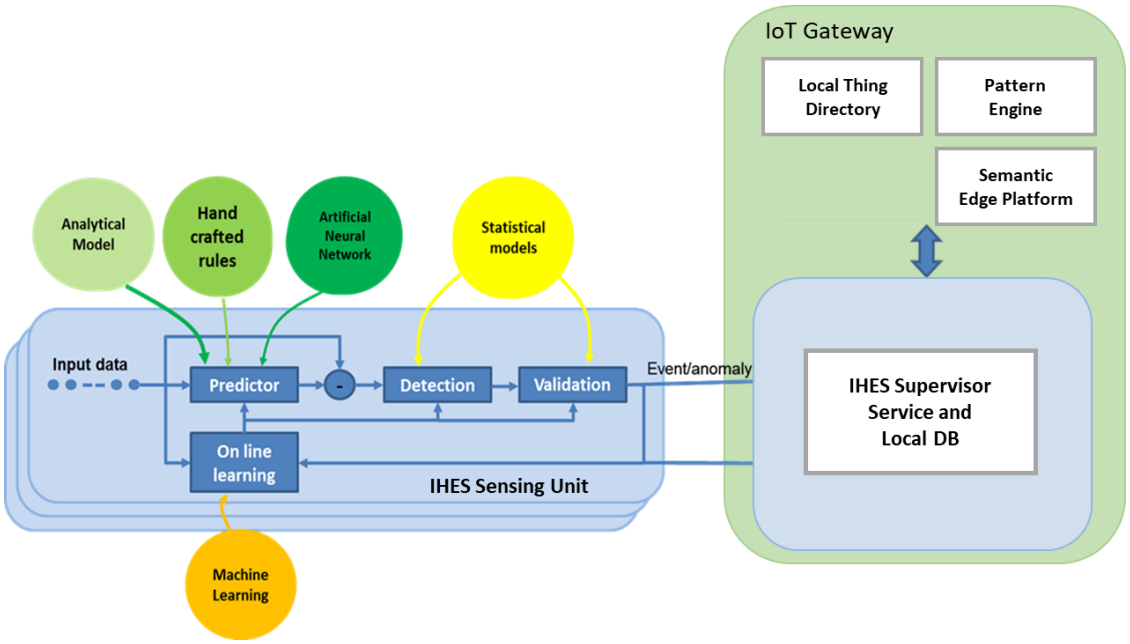


FIGURE 27 IHES LOCAL ANALYTICS (IOT GW + DEVICE NODE)

Moreover, an overview of the envisaged end-to-end system architecture is reported in Figure 28. NFV has a prominent role in UC3 architecture. As it is described in this section, UC3 involves SEMIoTICS components at the IoT GW level (Supervisor and Local DB component, Pattern Engine, Local Thing Directory, etc.) and at the Application Orchestration Layer (GUI, Monitoring, Pattern Orchestrator). These functionalities need to be deployed in a flexible manner, for instance by allowing software updates. Also, they require the dynamic provision of storage and processing resources to face the scaling nature of IoT. Fortunately, this flexibility,

programmability and dynamic resource provisioning is provided by NFV. More specifically, NFV is composed of several components. First, a network infrastructure that allows the virtualization of its computing, storage and networking resources, which is so-called NFVI. In our case, this is the IoT GW and the backend. Then, on top of the NFVI, one can deploy network functionalities that are so-called VNFs. Finally, the NFVI is managed by the VIM and the lifecycle of the VNFs is managed jointly by the VNF manager and the NFV Orchestrator, see Figure 28.

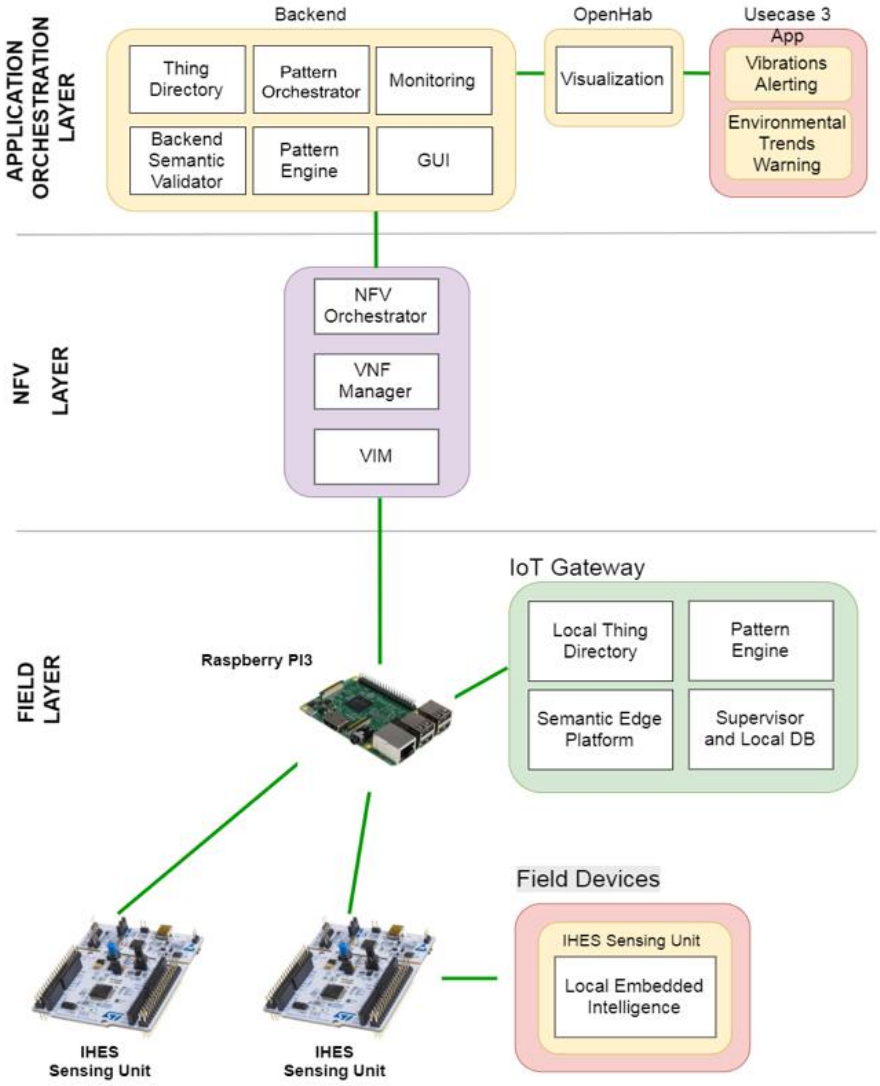


FIGURE 28 USE CASE 3 SYSTEM ARCHITECTURE

In the SEMIoTICS UC3, different functionalities are deployed in the form of VNFs:

- VNF1: This VNF contains the “Supervisor service” and the “Local DB” component in the IoT GW as it is described in Figure 28. That is, basically VNF1 gathers data from different Field devices, which are stored and then correlated to generate events.

- VNF2: This VNF is also deployed at the IoT GW and opens API endpoints for reporting data analytics events to the backend via MQTT.
- VNF: This is the global data aggregator located at the backend. Its main function is to receive and to store the data sent by the VNF2 of several IoT GW. To this end, it exposes API endpoints. Also, it paves the way for further processing of the events generated at the IoT GW, e.g. providing a global view rather than a local context. And last but not least, it permits to trigger external endpoints, e.g. VNF4 for data visualization. VNF3 can be considered as the SEMIoTICS "Monitoring" component of the backend.
- VNF4: This VNF contains the SEMIoTICS GUI component located at the backend, allowing data visualization.
- VNF5: This VNF contains the SEMIoTICS pattern engine at the IoT GW.

As part of the core functionalities of the IHES system, a lightweight version of algorithms focused on data monitoring and data model prediction is derived from the generic SEMIoTICS components and deployed on the IHES Sensing node. This functionality is responsible at the edge device level to support monitoring of the relevant events generated by the data-reduction algorithms developed. These algorithms are deployed likely in different instances due to the specific constraints and middleware's available for a given device/target ecosystem. In the case of UC3, the devices are low power STM32 MCUs units, so very different from the raspberry Pi3 ones adapted e.g. in UC1 and UC2. Considering the heavily constrained domain imposed by those MCUs units a subset of the functionalities deployed in the generic Gateway component will be mapped. Anyhow interoperability of the different modules will be ensured by the interoperable semantic patterns identified on WP3 activities.

Focusing on the UC3 scenario, The IHES Generic IoT Use Case the monitoring is intended on real-time time-variant generic signals that are locally processed in order to self-learn a predictive model and based on this prediction monitor any relevant deviation from the estimated model. In case of anomalies, these will be reported to the IHES service deployed in the IoT Gateway that in the case will propagate them to the upper level of SEMIoTICS architecture. Moreover, the Supervisor subcomponent will be interfaced to a Local DB subcomponent that will collect all relevant events accumulated by the system during its operations, in order to make them available to the other components of SEMIoTICS by mean of the GW Semantic Mediator and/or legacy Rest APIs exposed by the database.

The generic architecture of the IHES Use Case and mapped algorithms is shown in Figure 28. In blue are reported the components that are specific of the UC3 or that have been derived from other components but has been adapted to the specific field device platform. The system is composed ideally of two different kinds of functional modules: a set of IHES Sensing Units nodes and a set of Supervisor subcomponent. They act as asynchronous coordinated communicating using specific defined JSON messages sent to an MQTT broker deployed at IoT local gateway. Both modules will implement a local analytics processing pipeline composed of several algorithms in order to realize a generic unsupervised sensing node data monitoring facility. At the very end of this edge computing-oriented ecosystem there is a small, power-efficient sensing IoT node composed by an STM32 MCU equipped with Wi-Fi expansion board and a sensor shield board equipped with the following sensors:

- Environmental Sensors: temperature, humidity, pressure, luminosity
- Inertial Sensors: Accelerometer, Gyroscope, Magnetometer

The MCUs device maps both AI algorithms and hand-crafted algorithm (e.g. linear predictors) for implementing a predictive model estimation close to the source of data to process. The device node functionalities will be mapped on top of legacy ST middleware Software to manage the communication with the IoT gateway from one side and the acquisition of data from the sensor board on the other side. Each node will be able to monitor several sensors in the same device: as an example, it will be possible to instantiate a bare metal FW encompassing an accelerometer and another node implementing the monitoring of temperature/pressure/humidity at once. Similar algorithms will be mapped using different Middleware at IoT gateway level where the gateway likely will have an Embedded Linux OS Available. To enable interoperability

between the different algorithms and deployment on these heterogeneous devices a specific EAL (Embedded Analytic Library) analytic library will be developed with portability over Linux and STM32 in mind. Due to these technological constraints, only a subset of the functionalities of relevant identified components will be deployed in UC3, from the generic ones described in the SEMIoTICS reference architecture.

UC 3 architecture presented in Figure 28 that highlights the distributed nature of this system (many intelligent nodes connected to a local gateway, and potentially many gateways interfacing the backend layer) and how the analytics are distributed at several levels (mainly field devices and IoT Gateway) of the architecture. The architecture will be flexible in order to support a generic number of connected IHES devices with different capabilities (i.e. environmental vs inertial sensors). For this reason, the bootstrap interfaces have been carefully designed as part of WP3 activities (Task 3.3). A detailed discussion about the specific algorithms used to support this Use Case (a brief rationale on the technical choices made) will be made available in D4.3 that will cover all the aspects related to the local embedded analytics lightweight algorithms in SEMIoTICS.

A quick summary table of the components suitable for adoption in UC3 and their scope is presented in Table 5.

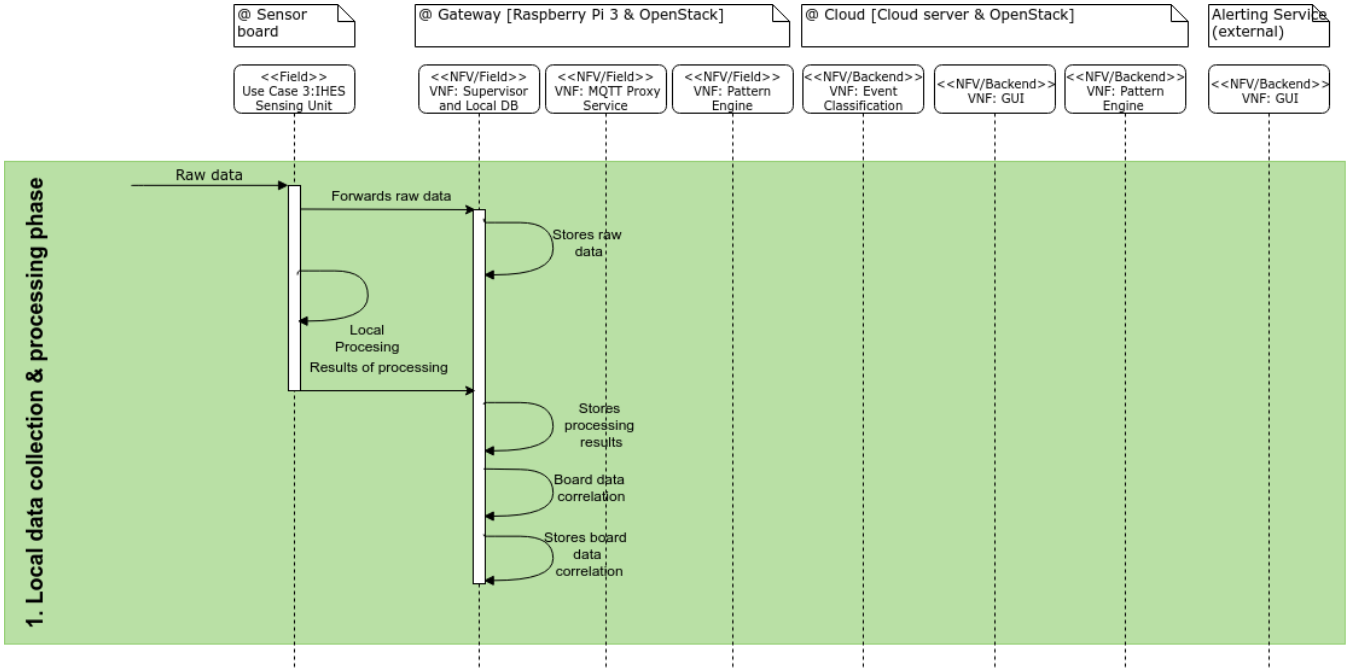
TABLE 5 UC3 COMPONENTS

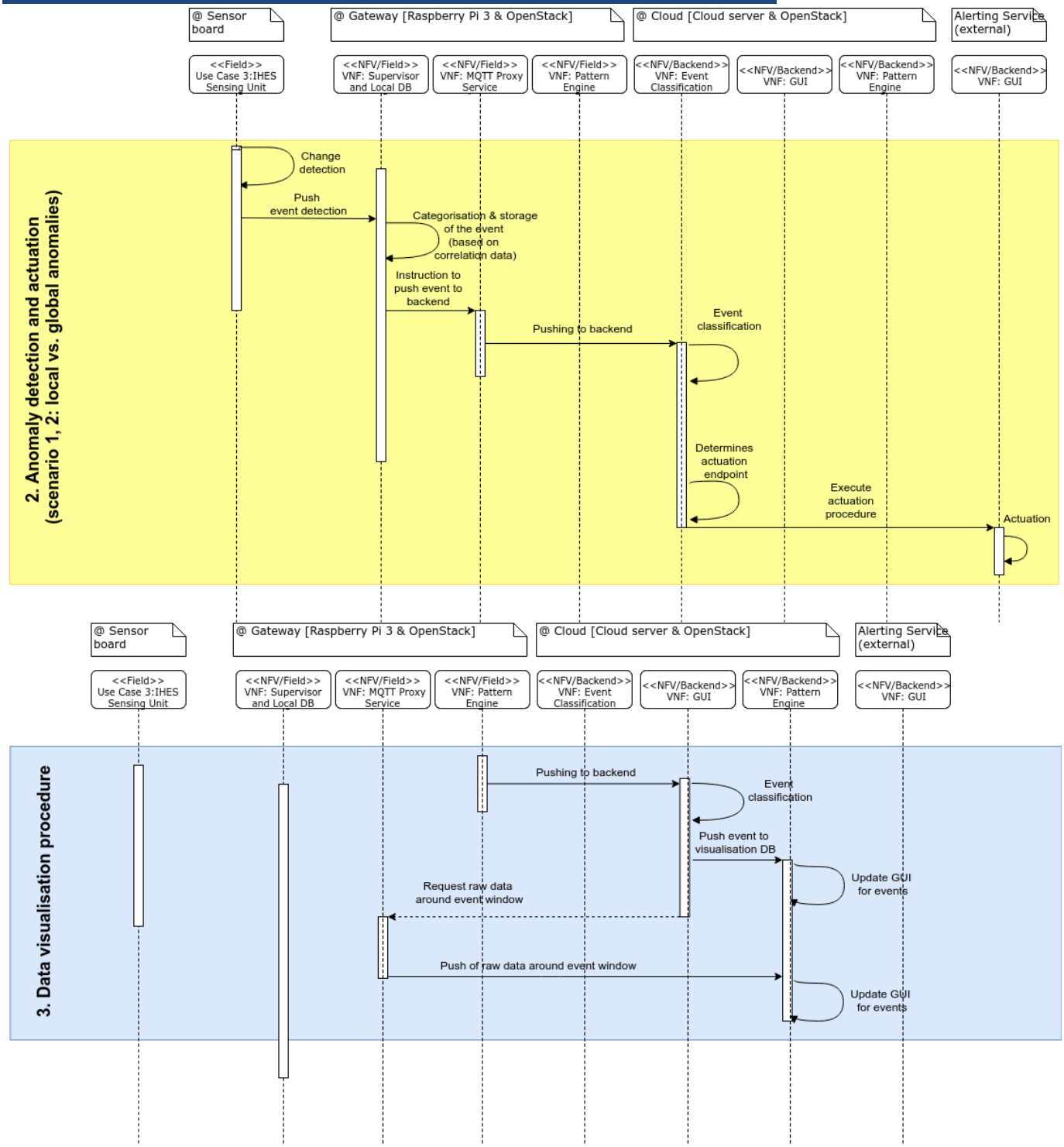
Architectural Components Used in UC3	Purpose of Deployment
openHAB Visualization	This openHAB IoT platform component is used in UC3 to realize the GUI frontend to monitor/manages the IHES system from an operator.
Thing Directory	Thing Directory contains the metadata of all the Local Things Directory entries to make all Things discoverable to the backend level as well.
Pattern Orchestrator	Pattern orchestrator is responsible to forward the respective UC3 pattern to the field layer pattern engine.
Pattern Engine	Pattern Engine is used to parse the connectivity and SPDI-related pattern instances and enforce and monitor their validity at runtime as received by the pattern orchestrator.
Local Thing Directory	Local Thing Directory makes the knowledge about Things (e.g. Field devices) available for applications. Local Thing Directory is typically accessed by Semantic Edge Platform (for enabling Edge applications).
Semantic Edge Platform	Semantic Edge Platform provides a user interface for the functionality of IoT Gateway. It can scan the network and discover sensing units and their capabilities. Moreover, the Local Thing Directory will be configurable over the Semantic Edge Platform.
Supervisor and Local DB	This component can supervise and manages the IHES Sensing units by either change their inner states (e.g. analytics processing) and collects data and events from them. It is able to store that data end events on a proper local specialized database in order to enable data aggregation and trend analysis at the backend level.
Local Embedded Intelligence	This component is the key one in UC3 wrapping the majority of local analytics tasks by implementing a set of ML / AI unsupervised algorithms (see D4.3). It provides interface the Supervisor in order to be coordinated by it.

4.3.2 USE CASE 3 DYNAMIC ARCHITECTURE – MESSAGE FLOW AND INTEROPERABILITY

The communication pattern in UC3 is heavily impacted by the underlying computational paradigm: distributed computing systems need to have a requisite strong communication capability in terms of QoS and associated semantic, usually relying on very complex message patterns. This is already complex in a cloud-dependent thing like most of today's devices, but it becomes a key aspect when intelligence is massively deployed (and distributed) at the level of the node. Semantics interoperability among heterogeneous devices and consistent message pattern flow, needs to be carefully designed when devices start to exchange not only raw data or simple events, but more complex message patterns used to describe more interaction between intelligent things. IHES devices are able to join/detach from a local cluster computation network coordinated by a local Supervisor subcomponent.

This Use Case deploys this distributed communication pattern by relying on two powerful available communication infrastructures: the MQTT protocol and the JSON data format for data interoperability. These infrastructures are used both for handling custom message patterns between the IHES nodes and the Supervisor and Local DB and for interfacing this same component with some other field device components of SEMIoTICS. More details about these semantic communication patterns and a subset of those used during the bootstrap interfacing phase have been provided in T3.3.




@ Sensor board
@ Gateway [Raspberry Pi 3 & OpenStack]
@ Cloud [Cloud server & OpenStack]
Alerting Service (external)
<<Field>> Use Case 3:IHES Sensing Unit
<<NFV/Field>> VNF: Supervisor and Local DB
<<NFV/Field>> VNF: MQTT Proxy Service
<<NFV/Field>> VNF: Pattern Engine
<<NFV/Backend>> VNF: Event Classification
<<NFV/Backend>> VNF: GUI
<<NFV/Backend>> VNF: Pattern Engine
<<NFV/Backend>> VNF: GUI
2. Anomaly detection and actuation (scenario 1, 2: local vs. global anomalies)
Change detection
Push event detection
Categorisation & storage of the event (based on correlation data)
Instruction to push event to backend
Pushing to backend
Event classification
Determines actuation endpoint
Execute actuation procedure
Actuation
@ Sensor board
@ Gateway [Raspberry Pi 3 & OpenStack]
@ Cloud [Cloud server & OpenStack]
Alerting Service (external)
<<Field>> Use Case 3:IHES Sensing Unit
<<NFV/Field>> VNF: Supervisor and Local DB
<<NFV/Field>> VNF: MQTT Proxy Service
<<NFV/Field>> VNF: Pattern Engine
<<NFV/Backend>> VNF: Event Classification
<<NFV/Backend>> VNF: GUI
<<NFV/Backend>> VNF: Pattern Engine
<<NFV/Backend>> VNF: GUI
3. Data visualisation procedure
Pushing to backend
Event classification
Push event to visualisation DB
Update GUI for events
Request raw data around event window
Push of raw data around event window
Update GUI for events

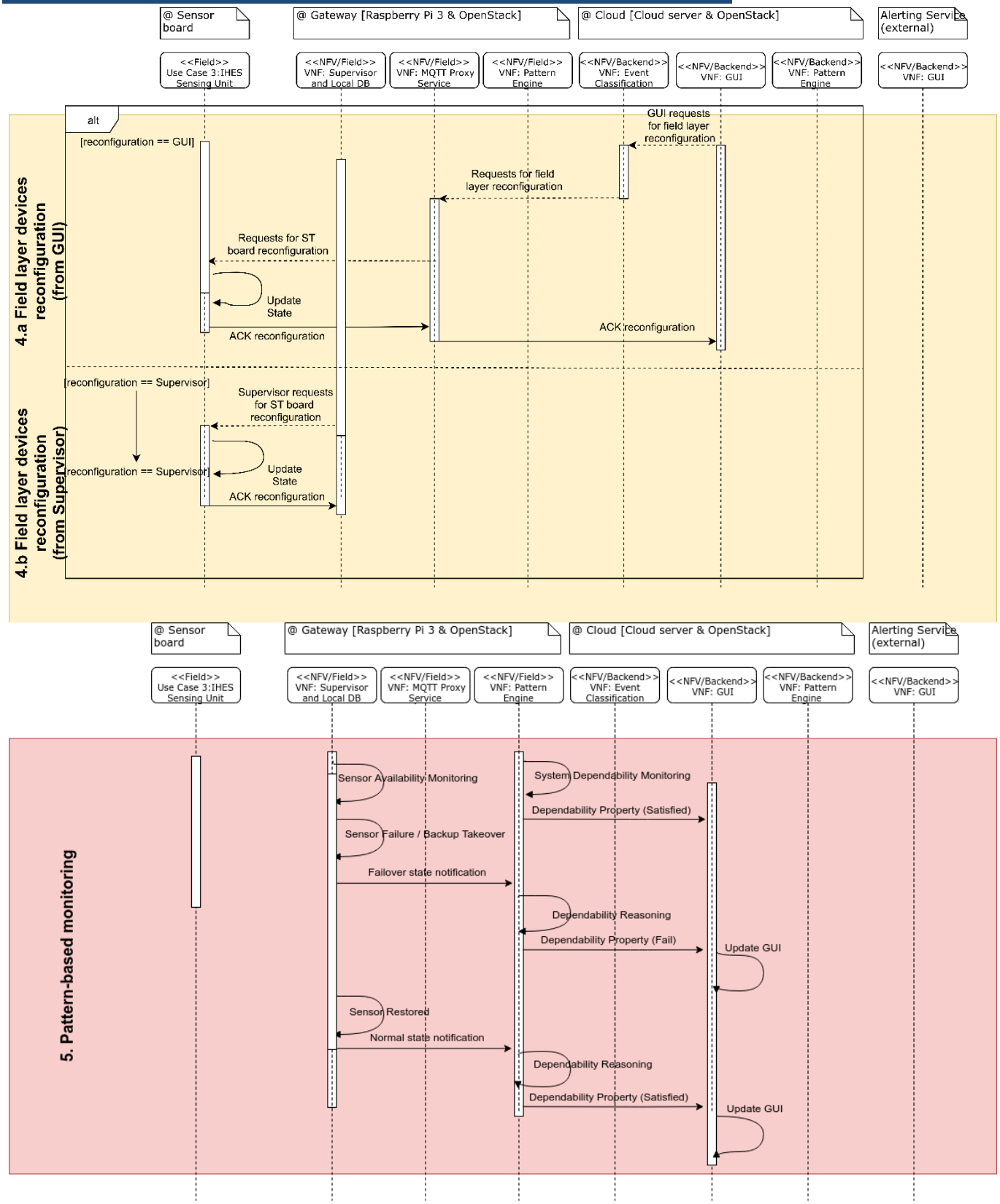


FIGURE 29 UC3 SEQUENCE FLOWS ENABLING D2.2 TARGET SCENARIOS

The UC3 dynamic architecture, derived from Figure 29, could be described as composed by the following functionalities:

Local data collection & processing phase

Raw data are collected by the IHES Sensing Unit and forwarded to the Supervisor subcomponent that stores it in the Local DB part. At the same time, the sensing unit starts the computation of the local embedded analytics algorithms in order to find out anomalies and local changes. these distribute analytics are sent to the Supervisor and stored into the Local DB. At the same time, the Supervisor computes and saves the Node Dependency Graph exploiting data correlations between actual working nodes.

Anomaly detection and actuation

When during the operative phase a change is detected by an IHES Sensing Unit, an alert is sent to the Supervisor. Then the supervisor, based on Node Dependency Graph information about the correlation of connected Sensing Units, categorizes the event and push it to the backend. This event is classified, and the correct actuation endpoint is selected. Then, the actuation procedures start.

Data visualization procedure

MQTT Proxy Service, when receiving an IHES Sensing Units change the message, stores the message in the visualization database. It also requests raw data around the event windows to Local DB. Requested data and received change messages are used to update the openHAB visualization GUI.

Field layer device reconfiguration (a)

At this point the openHAB visualization GUI request for field layer reconfiguration. This request is conveyed to the IHES Sensing Units and after the reconfiguration procedures, an ACK is sent back to the GUI.

Field layer device reconfiguration (b)

In this case, the reconfiguration request is sent to IHES Sensing Units from the Supervisor. After the reconfiguration procedures, an ACK is sent back to Supervisor.

Pattern-based monitoring

Within the UC3 scenario scope, the focus of the pattern-driven monitoring and adaptation will on the dependability property. Considering the criticality of the monitoring application, it is considered that redundant sensors will be used in the deployment to ensure that, even in the case of failures, another sensor will always be available to provide inputs. In this context, relevant pattern rules have been defined some and will be included as part of the demonstrating scenario. Leveraging said pattern rules, there is continuous monitoring of the monitor nodes that are connected and their current status. Thus, when a node fails and a redundant one takes over, the field pattern engine reasons on the reduced dependability condition, informing the backend for this change to a non-desired state in order to derive recover actions (node replacement, system reconfiguration, alerting, etc.); equivalently, when a node is restored and is back in action, this is detected and reasoned upon, informing that the system is back in the normal (dependable monitoring) state.

4.4 Leveraging the SEMIoTICS Framework for New Use Cases

The SEMIoTICS framework offers multiple core functionalities that can be used to support a variety of IoT use cases in addition to the ones already detailed in project use cases. In order to be able to fully utilize the proposed functionalities of the framework, there is a number of steps that need to be followed. The key areas of which need to be taken into considerations and analysis when implementing new use cases with SEMIoTICS framework are the following:

- **IoT device – WoT description**: SEMIoTICS can support IoT devices which are described according to WoT schema. WoT schema can be generated for existing devices using the Recipe Cooker app. Moreover, SEMIoTICS is capable to cooperate with "brownfield devices", however, to do that some additional implementation is required in the component called Semantic API & Protocol Binding.

- **Use case apps (field and backend level):** Use case business logic must be implemented by a new Use Case owner. SEMIoTICS doesn't provide ready-to-use applications logic. For new Use Cases, dedicated apps containing specific business logic must be created either as separate applications or as blocks in Recipe Cooker. The Recipe cooker delivers the functionality of modeling many different scenarios and can be easily enhanced with new logical blocks that can represent various functionalities of outside applications, platforms or components.
- **SPDI patterns – Patterns description:** One of the key features of SEMIoTICS is seamless orchestration through SPDI patterns. SEMIoTICS can set up cross-layer guarantees (backend, network, field layer) for a particular new Use Case with different properties in place such as QoS (i.e. bandwidth-delay, etc.) or SPDI properties (security dependability, etc.). These patterns can interact with the Recipe Cooker via the deployed recipe as received by the Pattern Orchestrator. Moreover, Pattern Orchestrator is responsible to pass the information to the respective Pattern Engines at all layers (Backend, Network and Field level). In this context, it is important to review the intrinsic architecture and SPDI requirements of the new use cases where SEMIoTICS is to be deployed and elaborate on the IoT system model (if needed), while also defining additional pattern rules, if the ones provided by SEMIoTICS are not adequate.
- **Monitoring – intelligence**: SEMIoTICS offers sophisticated monitoring component which collects, and monitors events form all of the components present in the platform. Platform users can subscribe to chosen complex events in order to get notified as soon as they occur. This gives enormous opportunities for platform monitoring at a central point with no information dispersion and scarcity. This monitoring procedure can enable the detection and prediction capabilities of SEMIoTICS.

High-level steps which are required for a newly approached Use Case owner are:

1. Describe the devices with WoT schema
2. Create field and backend applications with business logic (optional)
3. Create a recipe in Recipe Cooker
    a. Auto discover WoT devices
    b. Model a recipe
    c. Add patterns
4. Validate/instantiate the required pattern to guarantee the SPDI or QoS properties
5. Deploy a recipe and forward it to the pattern orchestrator.
6. Configure complex alerts

5 VALIDATION

This chapter summarizes the validation features of SEMIoTICS that are related to the platform architecture delivery and the various topics that are covered in this deliverable.

5.1 Related Project Objectives and Key Performance Indicators (KPIs)

The T2.4 related objectives and their mapping to D2.5 content is summarized in Table 6.

TABLE 6 TASK'S OBJECTIVES

T2.4 Objectives	D2.5 Chapters
• Specification of the overall reference architecture and a base-line specification of the interfaces and functionalities of the core components of the SEMIoTICS framework	2, 3
• Architectural and functional specification driven by the requirements identified in Tasks 2.1 and Task 2.2	3
• The reference architecture will contain the logical decomposition of SEMIoTICS to specific components with assigned roles, functionality and short description of the interaction between them.	3
• User-centric approach to design to ensure that user requirements are addressed by it.	4

The overall deliverable constitutes the initial contribution towards fulfilling the project's requirements regarding **SEMIoTICS' objectives**, and the associated KPIs, as shown in Table 7.

TABLE 7 TASK'S KPIS

Objective		KPI-ID	Description
2	Semantic interoperability	KPI-2.3	Semantic interoperability with 3 IoT platforms
5	IoT-aware Programmable Networks	KPI-5.1	Deployment of a multi-domain SDN orchestrator
6	Development of a Reference Prototype	KPI-6.3	Delivery of 3 prototypes of IIoT/IoT applications
7	Promote the adoption of EU technology offerings internationally	KPI-7.1	Provision of the SEMIoTICS framework and building blocks

5.2 End-user involvement in the Use Cases requirements elicitation

Process of Use Case-specific requirements gathering, for all three SEMIoTICS requirements, involved domain experts or end-users. Series of interviews and discussions with relevant stakeholders allowed to Use Case owners to identify and define the Use Case specific requirements.

For Use case 1 (Wind Energy), the detailed requirement descriptions were defined through an interview process with subject matter experts within the wind industry. BWC (Wind park operator) has verified the scope and details of these requirements, which will be considered for the lab trials where it is not planned to involve further end-users (e.g. other wind park operators). Furthermore, BWC as a consortium partner is involved in the project within the implementation and verification of UC1.

For Use case 2 (Assisted living) the elicitation of the SARA requirements was performed by the Engineering's Business Unit taking care of the evolution of the AREAS® E-Health Integrated Platform. This elicitation process

relied on the network of customers and partners that Engineering has in the Healthcare market. The requirements for the SARA component were collected through stakeholder interviews and focus groups.

Finally, use case 3 (Smart Sensing) has been deriving the Use Case specific requirements via domain expert interviews and based on extensive ST-I expertise in the field.

5.3 Project requirements mapping to Tasks and Architectural Components

Apart from the previously described role of the developed components, one additional scope of them is to satisfy the SEMIoTICS project requirements which were derived and documented in D2.3. More specifically, the procedure followed to provide this final correlation is based on the relationships between the different architectural logical components as being mapped in the tasks, KPIs or through the individual direct relationships between the requirements. In Table 8, the correlation between components and related requirements is presented. Finally, the full list of the mapping between the requirements, the Tasks and the Components and how they have derived can be found in Appendix 7.1.

TABLE 8 CORRELATION BETWEEN COMPONENT AND REQUIREMENTS

Component	Layer	Owner	Related Task	Related Requirements
Resource Manager	SDN orchestration layer	SAG	3.1, 3.5	R.GP.1 to 3, R.GP.6, R.BC.10, R.BC.12, R.NL.7, R.UC1.1, R.UC2.3, R.UC3.6 to 7
VTN Manager	SDN orchestration layer	SAG	3.1, 3.5	R.GP.1 to 3, R.NL.8 to 9, R.S.11, R.UC1.1, R.UC1.4
Bootstrapping Manager	SDN orchestration layer	SAG	3.1, 3.5	R.GP.2
Path Manager	SDN orchestration layer	SAG	3.1, 3.5	R.GP.1 to 3, R.S.11, R.UC.1, R.UC1.3 to 5, R.UC2.3, R.UC2.15, R.UC2.17, R.UC3.6 to 7
Clustering Manager	SDN orchestration layer	SAG	3.1, 3.5	R.GP.2, R.UC1.5 to 7
SFC Manager	SDN orchestration layer	FORTH	3.1, 3.5	R.GP1, R.NL.11, R.S.7 to 16, R.GSP.1 to 2, R.UC1.1, R.UC2.3, R.UC2.15, R.UC3.6 to 7
VIM Connector	SDN orchestration layer	CTTC	3.1, 3.2, 3.5	R.GP.2 to 3, R.BC.12 to 13, R.NL.8 to 9, R.NL.11, R.NL.12
NFV Orchestrator	NFV orchestration layer	CTTC	3.2, 3.5	R.GP.2 to 3, R.BC.1 to 4, R.BC.14 to 15, R.NL.1 to 4, R.NL.10 to 11, R.S.4, R.UC2.12, R.UC3.9, R.UC3.12 to 14
Virtualized Infrastructure Manager	NFV orchestration layer	CTTC	3.2, 3.5	R.GP.2 to 3, R.BC.1 to 4, R.BC.14, R.NL.1 to 4, R.NL.10, R.UC2.12, R.UC3.9, R.UC3.12 to 14
VNF Manager	NFV orchestration layer	CTTC	3.2, 3.5	R.GP.2 to 3, R.BC.1 to 9, R.BC.11 to 15, R.NL.1 to 6, R.NL.8 to 11, R.UC2.12, R.UC3.9, R.UC3.12 to 14
Semantic API & Protocol Biding	Field layer	SAG	3.3	R.GP.1, R.FD.5 to 8, R.FD.11 to 13, R.UC1.1, R.UC1.8 to 9, R.UC2.2, R.UC3.9, R.UC3.13 to 15
GW Semantic Mediator	Field layer	SAG	3.3	R.UC1.9 to R.UC1.12
Semantic Edge Platform	Field layer	SAG	3.3	R.UC1.9 to R.UC1.12
Local thing directory	Field layer	SAG	3.3	R.UC1.11, R.UC2.5 to 6
Pattern Orchestrator	Application Orchestration Layer	STS	4.1	R.GP.1, R.GP.4, R.S.1. R.P.12, R.GSP.10
Pattern Engine	Application Orchestration Layer	STS	4.1	R.GP.1, R.S.1, R.S.4 to 5, R.P.1, R.P.3 to 4, R.P.6, R.P.8 to 9, R.P.12, R.GSP.4, R.GSP.10, R.GSP.7, R.UC1.5 to 6, R.UC2.8
Pattern Engine	SDN orchestration layer	FORTH	3.1, 3.4, 4.1, 4.5	R.GP.1, R.GP.4, R.GP.7, R.NL.11 to 13, R.S.1 to 2, R.S.4 to 5, R.S.7, R.S.17 to 18, R.P.1, R.P.3 to 4, R.P.6, R.P.8 to 9, R.P.12, R.GSP.1, R.GSP.4, R.GSP.10, R.UC1.1, R.UC1.3 to 6
Pattern Engine	Field layer	FORTH	4.1	R.GP.1, R.GP.4, R.S.1, R.S.3 to 7, R.P.1, R.P.3 to 4, R.P.6, R.P.8 to 9, R.P.12, R.GSP.4, R.GSP.10, R.UC1.6, R.UC2.8, R.UC3.16

Monitoring	Field layer	ENG	4.2	R.GP.4, R.BC.20, R.NL.13, R.P.4, R.GSP.7
Monitoring	Application Orchestration Layer	ENG	4.2	R.FD.15, R.P.4, R.GSP.2, R.GSP.7
Local embedded intelligence	Field layer	ST	4.3	R.FD.1 to 2, R.FD.4 to 7, R.FD.10 to 11, R.FD.3 to 9, R.FD.11 to 18
Recipe Cooker	Application Orchestration Layer	SAG	4.4, 4.6	R.GP.1 to 3, R.UC1.1, R.UC1.3
Thing directory	Application Orchestration Layer	SAG	4.4, 4.6	R.GP.2, R.P.3 to 4, R.P.9, R.UC1.1, R.UC1.3
Backend Semantic Validator	Application Orchestration Layer	FORTH	4.4	R.GP.1, R.S.1, R.UC1.8 to 9, R.UC1.12, R.UC3.3, R.UC2.6, R.UC2.11, R.UC3.1, R.UC3.15
Security Manager	Application Orchestration Layer	UP	4.5	R.BC.15, R.S.1, R.P.3 to 5, R.P.8, R.P.12, R.GSP.9,
Security Manager	SDN orchestration layer	FORTH	3.1, 4.5	R.S.2, R.S.7, R.P.1 to 13, R.GSP.9, R.UC1.6
Security Manager	Field layer	UP	4.5	R.P.5, R.P.12, R.GSP.9
Backend orchestrator	Application Orchestration Layer	BS	4.6	R.GP.1, R.GP.2, R.BC.15
GUI	Application Orchestration Layer	BS	4.6	R.P.1 to 4, R.P.9
Use case 1	Field layer	SAG	5.3, 5.4	R.UC1.1 to R.UC1.13
Use case 1 apps	Application Orchestration Layer	SAG	5.4	R.UC1.1 to R.UC1.13
Use case 2	Field layer	ENG	5.3, 5.5	R.UC2.17 to R.UC2.17
Use case 2 apps	Application Orchestration Layer	ENG	5.5	R.UC2.17 to R.UC2.17
Use case 3	Field layer	ST	5.3, 5.6	R.UC3.1 to R.UC1.18
Use case 3 apps	Application Orchestration Layer	IQU/ST	5.6	R.UC3.1 to R.UC1.18

6 CONCLUSION

This deliverable described the SEMIoTICS architectural framework, which addresses the complicated requirements of IoT/IIoT applications such as security, privacy, dependability, and interoperability. It documented the core mechanisms of the SEMIoTICS framework and presented their mapping to the architecture structure. The functional components of the proposed architecture are illustrated in detail. Finally, the representation of use case scenarios is described and presented in the context of the SEMIoTICS Framework and its presented architecture.

To optimize time and resources, work was carried out in parallel with tasks carried out within WP3, WP4, and WP5. All results from said tasks have provided feedback and led to continuous updates of the SEMIoTICS architecture. Since through research and development new concepts and needs arose, the architectural approach had to be changed and some modifications needed to be introduced. The evolution of architecture was presented herein to highlight the changes made at every stage of the project.

The final version of the SEMIoTICS high-level architecture deliverable (i.e., D2.5) is focused on the dynamic architecture aspects considering the design decisions made in WP3 and WP4. Each component was analyzed in detail to ensure satisfaction of the project's requirements is achieved. Moreover, to fulfill the assumptions about the use of external platform components, the consortium made every effort to examine the intrinsic requirements of external IoT platforms and the use case environments. The output of these efforts in the context of the architecture definition, i.e., the detailed specification of components and their interactions, was documented herein.

Based on the above, and per the deliverable's scope, D2.5 covers use case -specific message flows and diagrams with connections to modules and layers. To facilitate readers' understanding, the architecture is presented for each specific use case with the actual devices and sensors present in each of these contexts (e.g. robots, sensors, actuators) and the scenarios that are applicable within said use cases.

Through this interaction with the implementation tasks (i.e. WP3, WP4) and the use case owners, work on this task has also resulted in a useful exchange of knowledge regarding specific components and technologies between partners, also enabling the consortium to gain a common and homogeneous vision about the role of each components and the SEMIoTICS framework as a whole. Armed with this knowledge and common understanding, this deliverable and the resulting architecture will drive the integration and demonstration efforts of WP5.

7 APPENDIX

7.1 Mapping between requirements, architectural component and tasks.

7.1.1 GENERAL PLATFORM, BACKEND, NETWORK, FIELD AND PROJECT REQUIREMENTS MAPPING TO TASKS AND ARCHITECTURAL COMPONENTS

SDN Layer		NFV Layer		WP3 - Smart objects and networks (SAG)																				WP4 - Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy (FORTH)																														WP5 (ENG)		
Backend Layer		Field Layer		T3.5 - Implementation of Field-level middleware & networking toolbox (IQU)																				T4.6 - Implementation of SEMIoTICS backend API (BS)																														T5.2-T5.3		
Topic	Req-ID	Functional	Req Level	T3.1 - SDN	Resource Manager	VTN Manager	Bootstrapping Manager	Path Manager	Clustering Manager	SFC Manager	VIM Connector	T3.2 - NFV	NFV Orchestrator	VNF Manager	VIM	T3.3 - Bootstraping	Semantic API & Protocol Binding	GW Semantic Mediator	Local Thing Directory	T3.4 - Interoperability	Pattern Engine NBI	T3.5 - Implementation	T4.1 SPDI	Pattern Orchestrator	Pattern Engine (Backend)	Pattern Engine (SDN)	Pattern Engine (Field)	T4.2 -Monitoring	Monitoring (Field)	Monitoring (Backend)	T4.3 - Local Intelligence	Local Embedded Intelligence	T4.4 - Semantic interoperability	Recipe Cooker	Thing Directory	Backend Semantic Validator	T4.5 - Security	Security Manager (Network)	Security Manager (Field)	Security Manager (Backend)	T4.6 - Implementation	Backend Orchestrator	GUI	T5.4- UC1	T5.5-UC2	T5.6 - UC3
				SAG	SAG	SAG	SAG	SAG	SAG	FORTH	CTTC	CTTC	CTTC	CTTC	CTTC	SAG	SAG	SAG	SAG	STS	FORTH	IQU	STS	STS	STS	FORTH	FORTH	ENG	ENG	ENG	ST	ST	FORTH	SAG	SAG	FORTH	UP	FORTH	UP	UP	BS	BS	BS	SAG	ENG	ST
General Platform Requirements	R.GP.1	Yes	MUST	o	o	o		o		o						o	o		o	o	o	o	o	o	o	o	o				o		o			o					o	o		o	o	o
	R.GP.2	Yes	MUST	o	o	o	o	o	o			o	o	o	o	o						o									o			o	o						o	o				
	R.GP.3	Yes	MUST	o	o	o		o			o	o	o	o	o					o	o	o	o											o								o	o			
	R.GP.4	Yes	MUST	o																o	o	o	o	o		o		o	o													o	o			
	R.GP.5	Yes	MUST	o																o	o	o																				o				
	R.GP.6	Yes	MUST	o	o															o	o	o																				o				
	R.GP.7	Yes	MUST	o																o	o	o	o			o																				
Backend/Cloud Layer Requirements	R.BC.1	Yes	MUST									o	o	o	o							o																								
	R.BC.2	Yes	SHOULD									o	o	o	o							o																								
	R.BC.3	Yes	MUST									o	o	o	o							o																								
	R.BC.4	Yes	MUST									o	o	o	o							o																								
	R.BC.5	Yes	MUST									o			o							o																								
	R.BC.6	Yes	SHOULD									o			o							o																								
	R.BC.7	Yes	MUST									o			o							o																								
	R.BC.8	Yes	MUST									o			o							o																								
	R.BC.9	Yes	MUST									o			o							o																								
	R.BC.10	Yes	MUST		o																	o																								
	R.BC.11	Yes	MUST																			o																								
	R.BC.12	Yes	MUST		o						o	o			o							o																								
	R.BC.13	Yes	MUST		o						o	o			o							o																								
	R.BC.14	Yes	MUST									o	o	o	o							o																								
	R.BC.15	Yes	MUST									o	o		o							o																		o	o	o				
	R.BC.16	Yes	MUST																			o																								
	R.BC.17	Yes	MUST																			o																								
	R.BC.18	Yes	MUST																			o	o		o																				
	R.BC.19	Yes	SHOULD																			o	o	o																						
	R.BC.20	Yes	MUST																			o	o	o			o	o																		
Network Layer Requirements	R.NL.1	Yes	SHOULD	o								o	o	o	o							o																								
	R.NL.2	Yes	SHOULD	o								o	o	o	o							o																								
	R.NL.3	Yes	MUST	o								o	o	o	o							o																								
	R.NL.4	Yes	SHOULD	o								o	o	o	o							o																								
	R.NL.5	Yes	MUST	o								o			o							o																								
	R.NL.6	Yes	MUST	o								o			o							o																								
	R.NL.7	Yes	SHOULD	o	o																	o																								
	R.NL.8	Yes	MUST	o		o					o	o			o							o																								
	R.NL.9	Yes	MUST	o		o					o	o			o					o	o	o																								
	R.NL.10	Yes	MUST	o								o	o	o	o							o																								
	R.NL.11	Yes	MUST	o						o	o	o	o		o					o	o	o	o			o																				
	R.NL.12	Yes	MUST																	o	o	o	o			o																				
	R.NL.13	Yes	MUST																	o	o		o			o		o	o																
Field Layer Requirements	R.FD.1	Yes	SHOULD																												o	o												o
	R.FD.2	Yes	SHOULD																												o	o												o
	R.FD.3	Yes	SHOULD																																									o
	R.FD.4	Yes	SHOULD																												o	o												o
	R.FD.5	Yes	SHOULD														o														o	o												o
	R.FD.6	Yes	MUST														o														o	o											o	o
	R.FD.7	Yes	MUST														o														o	o											o	o
	R.FD.8	Yes	MUST														o																										o	
	R.FD.9	Yes	MUST																																								o	
	R.FD.10	No	SHOULD																												o	o											o	o
	R.FD.11	No	SHOULD																												o	o											o	o
	R.FD.12	Yes	MUST														o																										o	
	R.FD.13	Yes	MUST														o																										o	
	R.FD.14	Yes	MUST																				o				o																o	o
	R.FD.15	Yes	MUST																				o				o	o	o															

7.1.2 SECURITY AND PRIVACY PROJECT REQUIREMENTSMAPPING TO TASKS AND ARCHITECTURAL COMPONENTS

				WP3 - Smart objects and networks (SAG)																				WP4 - Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy (FORTH)																																		WP5 (ENG)		
				T3.5 - Implementation of Field-level middleware & networking toolbox (IQU)																				T4.6 - Implementation of SEMIoTICS backend API (BS)																																		T5.2-T5.3		
Topic	Req ID	Functional	Req Level	T3.1 - SDN	Resource Manager	VTN Manager	Bootstrapping Manager	Path Manager	Clustering Manager	SFC Manager	VIM Connector	T3.2 - NFV	NFV Orchestrator	VNF Manager	VIM	T3.3 - Bootstraping	Semantic API & Protocol Binding	GW Semantic Mediator	Local Thing Directory	T3.4 - Interoperability	Pattern Engine NBI	T3.5 - Implementation	T4.1 SPDI	Pattern Orchestrator	Pattern Engine (Backend)	Pattern Engine (SDN)	Pattern Engine (Field)	T4.2 -Monitoring	Monitoring (Field)	Monitoring (Backend)	T4.3 - Local Intelligence	Local Embedded Intelligence	T4.4 - Semantic interoperability	Recipe Cooker	Thing Directory	Backend Semantic Validator	T4.5 - Security	Security Manager (Network)	Security Manager (Field)	Security Manager (Backend)	T4.6 - Implementation	Backend Orchestrator	GUI	T5.4- UC1	T5.5-UC2	T5.6 - UC3
				SAG	SAG	SAG	SAG	SAG	SAG	FORTH	CTTC	CTTC	CTTC	CTTC	CTTC	SAG	SAG	SAG	SAG	STS	FORTH	IQU	STS	STS	STS	FORTH	FORTH	ENG	ENG	ENG	ST	ST	FORTH	SAG	SAG	FORTH	UP	FORTH	UP	UP	BS	BS	BS	SAG	ENG	ST
IoT Security and Privacy Requirements	R.S.1	Yes	MUST																	o	o		o	o	o	o									o				o							o		
	R.S.2	Yes	MUST	o																o			o																o							o		
	R.S.3	No	SHALL																	o						o																			o		
	R.S.4	Yes	MUST									o	o							o	o		o		o	o	o											o							o		
	R.S.5	Yes	SHALL																				o		o	o	o											o							o		
	R.S.6	No	SHALL																				o				o																			o		
	R.S.7	Yes	SHALL	o						o										o	o		o			o												o									
	R.S.8	No	SHALL							o																																						
	R.S.9	No	SHALL							o																																						
	R.S.10	No	SHALL							o																																						
	R.S.11	No	MUST			o		o		o																																						
	R.S.12	No	MUST							o																																						
	R.S.13	No	SHALL							o																																						
	R.S.14	Yes	SHALL							o																																						
	R.S.15	No	MUST							o																																						
	R.S.16	Yes	MUST							o																																						
	R.S.17	Yes	MUST																				o			o																						
	R.S.18	Yes	SHALL																				o			o																						
	R.S.19	Yes	SHALL																																												
	R.S.20	Yes	MUST																																												o		
	R.P.1	Yes	MUST																				o		o	o	o											o	o			o		o		o		
	R.P.2	Yes	MUST																																			o	o			o		o		o		
	R.P.3	Yes	MUST																				o		o	o	o							o		o		o	o		o	o		o		o		
	R.P.4	Yes	MUST																				o		o	o	o	o	o	o				o		o		o	o		o	o		o		o		
	R.P.5	Yes	MUST																																			o	o	o	o				o		
	R.P.6	Yes	MUST																				o		o	o	o											o							o		
	R.P.7	Yes	MUST																																			o	o						o		
	R.P.8	Yes	MUST																				o		o	o	o											o	o		o				o		
	R.P.9	Yes	SHALL																				o		o	o	o							o		o			o					o		o		
	R.P.10	Yes	SHALL																																				o							o		
	R.P.11	No	SHALL																																				o							o		
	R.P.12	Yes	MUST																	o	o		o	o	o	o	o											o	o	o	o				o		
	R.P.13	Yes	SHALL																																				o							o		
Generic Security and Privacy Requirements for the SEMIOTICS Use CASES	R.GSP.1	Yes	MUST							o													o			o																						
	R.GSP.2	No	MAY							o																				o																		
	R.GSP.3	Yes	SHALL																																													
	R.GSP.4	No	SHALL																				o		o	o	o																						
	R.GSP.5	No	SHALL																																													
	R.GSP.6	Yes	MUST																																													
	R.GSP.7	Yes	SHALL																				o		o			o	o	o																		
	R.GSP.8	Yes	MUST																																												o		
	R.GSP.9	No	SHALL																				o																o	o	o	o				o		
	R.GSP.10	No	MUST																				o	o	o	o	o																				o		

7.1.3 USE CASE SPECIFIC PROJECT REQUIREMENTS MAPPING TO TASKS AND ARCHITECTURAL COMPONENTS

SDN Layer	NFV Layer
Backend Layer	Field Layer

WP3 - Smart objects and networks (SAG) — T3.5 - Implementation of Field-level middleware & networking toolbox (IQU)

Topic	Req-ID	Functional	Req Level	T3.1 - SDN	Resource Manager	VTN Manager	Bootstrapping Manager	Path Manager	Clustering Manager	SFC Manager	VIM Connector	T3.2 - NFV	NFV Orchestrator	VNF Manager	VIM	T3.3 - Bootstraping	Semantic API & Protocol Binding	GW Semantic Mediator	Local Thing Directory	T3.4 - Interoperabilty	Pattern Engine NBI	T3.5 - Implementation
				SAG	SAG	SAG	SAG	SAG	SAG	FORTH	CTTC	CTTC	CTTC	CTTC	CTTC	SAG	SAG	SAG	SAG	STS	FORTH	IQU
UC1 Wind Energy	R.UC1.1	Yes	MUST	o	o	o		o		o						o	o			o	o	
	R.UC1.2	Yes	MUST																			
	R.UC1.3	Yes	MUST	o				o												o	o	
	R.UC1.4	Yes	MUST	o		o		o												o	o	
	R.UC1.5	Yes	SHALL	o				o	o											o	o	
	R.UC1.6	Yes	SHALL	o					o													
	R.UC1.7	Yes	SHALL	o					o													
	R.UC1.8	Yes	MUST													o	o					
	R.UC1.9	Yes	MUST													o	o	o				
	R.UC1.10	Yes	MUST													o						
	R.UC1.11	No	SHALL													o			o			
	R.UC1.12	No	MUST													o		o		o	o	
	R.UC1.13	Yes	MUST													o						
UC2 SARA	R.UC2.1	Yes	SHOULD	o																		
	R.UC2.2	Yes	SHOULD	o		o										o				o	o	
	R.UC2.3	Yes	SHOULD	o	o	o		o		o												
	R.UC2.4	Yes	SHOULD																			
	R.UC2.5	Yes	SHOULD	o		o										o			o			
	R.UC2.6	Yes	SHOULD													o			o			
	R.UC2.7	Yes	SHOULD																			
	R.UC2.8	Yes	SHOULD																			
	R.UC2.9	Yes	SHOULD																			
	R.UC2.10	Yes	SHOULD																			
	R.UC2.11	Yes	SHOULD																			
	R.UC2.12	Yes	SHOULD								o	o	o	o	o							
	R.UC2.13	Yes	MAY																			
	R.UC2.14	Yes	MAY																			
	R.UC2.15	No	SHOULD	o				o		o										o	o	
	R.UC2.16	No	SHOULD																			
	R.UC2.17	No	SHOULD	o				o														
UC3 Sensing	R.UC3.1	Yes	SHOULD																			
	R.UC3.2	Yes	MUST													o	o					
	RUC3.3	Yes	MUST																			
	R.UC3.4	Yes	MUST																			
	R.UC3.5	Yes	SHOULD																			
	R.UC3.6	Yes	MAY	o	o	o		o		o												
	R.UC3.7	Yes	MUST	o	o	o		o		o												
	R.UC3.8	Yes	MUST																			
	R.UC3.9	Yes	MUST									o	o	o	o	o	o					
	R.UC3.10	Yes	MUST																			
	R.UC3.11	Yes	SHOULD																			
	R.UC3.12	Yes	MUST									o	o	o	o	o						
	R.UC3.13	Yes	SHOULD									o	o	o	o	o	o					
	R.UC3.14	Yes	MUST	o		o						o	o	o	o	o	o					
	R.UC3.15	Yes	SHOULD													o	o					
	R.UC3.16	Yes	SHOULD													o						
	R.UC3.17	Yes	SHOULD													o						
	R.UC3.18	Yes	MAY																			

WP4 - Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy (FORTH) — T4.6 - Implementation of SEMIoTICS backend API (BS)

Req-ID	T4.1 SPDI	Pattern Orchestrator	Pattern Engine (Backend)	Pattern Engine (SDN)	Pattern Engine (Field)	T4.2 -Monitoring	Monitoring	Monitoring	T4.3 - Local Intelligence	Local Embedded Intelligence	T4.4 - Semantic interoperability	Recipe Cooker	Thing Directory	Backend Semantic Validator	T4.5 - Security	Security Manager (Network)	Security Manager (Field)	Security Manager (Backend)	T4.6 - Implementation	Backend Orchestrator	GUI
	STS	STS	STS	FORTH	FORTH	ENG	ENG	ENG	ST	ST	FORTH	SAG	SAG	FORTH	UP	FORTH	UP	UP	BS	BS	BS
R.UC1.1	o			o								o	o								
R.UC1.2																					
R.UC1.3	o			o								o	o								
R.UC1.4	o			o																	
R.UC1.5	o		o	o																	
R.UC1.6	o		o	o	o											o					
R.UC1.7																					
R.UC1.8											o			o	o						
R.UC1.9											o			o							
R.UC1.10																					
R.UC1.11																					
R.UC1.12											o			o	o						
R.UC1.13																					
R.UC2.1																					
R.UC2.2																					
R.UC2.3											o			o							
R.UC2.4																					
R.UC2.5																					
R.UC2.6											o			o							
R.UC2.7	o																				
R.UC2.8	o		o		o																
R.UC2.9																					
R.UC2.10						o															
R.UC2.11											o			o							
R.UC2.12						o															
R.UC2.13																					
R.UC2.14																					
R.UC2.15	o																				
R.UC2.16																					
R.UC2.17																					
R.UC3.1											o			o							
R.UC3.2																					
RUC3.3									o	o											
R.UC3.4									o	o											
R.UC3.5									o	o											
R.UC3.6									o	o											
R.UC3.7	o								o	o											
R.UC3.8									o	o											
R.UC3.9									o	o											
R.UC3.10																					
R.UC3.11									o	o											
R.UC3.12									o	o											
R.UC3.13									o	o											
R.UC3.14									o	o											
R.UC3.15	o								o	o	o			o							
R.UC3.16	o				o				o	o											
R.UC3.17									o	o											
R.UC3.18	o								o	o											

WP5 (ENG) — T5.2-T5.3

Req-ID	T5.4- UC1	T5.5-UC2	T5.6 - UC3
	SAG	ENG	ST
R.UC1.1	o		
R.UC1.2	o		
R.UC1.3	o		
R.UC1.4	o		
R.UC1.5	o		
R.UC1.6	o		
R.UC1.7	o		
R.UC1.8	o		
R.UC1.9	o		
R.UC1.10	o		
R.UC1.11	o		
R.UC1.12	o		
R.UC1.13	o		
R.UC2.1		o	
R.UC2.2		o	
R.UC2.3		o	
R.UC2.4		o	
R.UC2.5		o	
R.UC2.6		o	
R.UC2.7		o	
R.UC2.8		o	
R.UC2.9		o	
R.UC2.10		o	
R.UC2.11		o	
R.UC2.12		o	
R.UC2.13		o	
R.UC2.14		o	
R.UC2.15		o	
R.UC2.16		o	
R.UC2.17		o	
R.UC3.1			o
R.UC3.2			o
RUC3.3			o
R.UC3.4			o
R.UC3.5			o
R.UC3.6			o
R.UC3.7			o
R.UC3.8			o
R.UC3.9			o
R.UC3.10			o
R.UC3.11			o
R.UC3.12			o
R.UC3.13			o
R.UC3.14			o
R.UC3.15			o
R.UC3.16			o
R.UC3.17			o
R.UC3.18			o