SEMIoTICS

Deliverable D2.4.
SEMIoTICS High Level Architecture (Cycle 1)

Deliverable release date	Initial 20.04.2019, revised 25.11.2019
Authors	1. Ermin Sakic (SAG), Darko Anicic (SAG) 2. Eftychia Lakka, Nikolaos Petroulakis (FORTH) 3. Jordi Serra, David Pubill, Angelos Antonopoulos, Christos Verikoukis (CTTC) 4. Danilo Pau, Mirko Falchetto (ST) 5. Domenico Presenza (ENG) 6. Tobias Marktscheffel (UP) 7. Łukasz Ciechomski, Karolina Walędzik, Marcin Zawadzki, Łukasz Kempiński, Urszula Rak (BS) 8. Prodromos-Vasileios Mekikis (IQU)
Responsible person	Łukasz Ciechomski (BS), Mirko Falchetto (ST)
Reviewed by	Mirko Falchetto (ST), Urszula Rak (BS), Łukasz Ciechomski (BS)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, Ermin Sakic, Mirko Falchetto, Domenico Presenza, Christos Verikoukis) 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 revised
Dissemination level	Public

Table of Contents

Acronyms Table	
Acronym	**Definition**
2FA	Two-Factor-Authentication
6LoWPAN	IPv6 over Low-power Wireless Personal Area Network
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programmable Interface
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
AuR	Autonomous Robotics
AWS	Amazon Web Services
BAN	Body Area Network
BIM	Building Information Management
BLE	Bluetooth Low Energy
BSS	Business Support System
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
CPU	Central Processing Unit
DoS	Denial of Service
DPDK	Data Plane Developer’s Kit
DTLS	Datagram Transport Layer Security
E2E	End to End
ETSI	European Telecommunications Standards Institute
FHIR	Fast Healthcare Interoperability Resources
FPGA	Field-Programmable Gate Array
FW	Firmware
GE	Generic Enabler

GW	Gateway
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
IETF	Internet Engineering Task Force
IFTTT	If This, Then That
IHES	Intelligent Heterogeneous Embedded Sensors
IIoT	Industrial Internet of Things
IMU	Inertial Measurement Unit
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process communication
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IT	Information Technology
JSON	JavaScript Object Notation
JSON-LD	1JavaScript Object Notation for Linking Data
KVM	Kernel-based Virtual Machine
LD	Linking Data
LHS	Left Hand Side
LWM2M	Lightweight Machine-to-M
LXD	Linux Containers
M2M	Machine to Machine
MAC	Media Access Control
MANO	Management and Orchestration
MCU	Micro Controller Unit
MQTT	2Message Queuing Telemetry Transport
mW	Milliwatts
NBI	Northbound Interface
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization

1 JavaScript Object Notification for Linking Data

2 Message Queuing Telemetry Transport

NFVI	Network Functions Virtualization Infrastructure
NFVI-RA	Network Functions Virtualization Infrastructure Resource Allocation
NFVO	NFV orchestrator
NGSI	Next Generation Service Interfaces
NGSIv2	Next Generation Service Interfaces version 2
NS	Network Service
O&M	Operations and Maintenance
OASIS	Organization for the Advancement of Structured Information Standards
OEM	Original Equipment Manufacturer
ODL	Open Daylight
OFCONF	OpenFlow Configuration
OGC	Open Geospatial Consortium
OSS	Operations Supports System
OVSDB	Open vSwitch Database Management Protocol
OWL	Web Ontology Language
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PLC	Power Line Controller
PM	Pattern Module
PNF	Physical Network Functions
POP	Point of Presence
QoS	Quality of Service
RA	Robotic Assistant
REST	Representational State Transfer
RHS	Right Hand Size
RO	Resource Orchestrator
RO NBI	Resource Orchestrator Northbound Interface
RPC	Remote Procedure Call
Rpi	Raspberry PI
RR	Robotic Rollator
SARA	Socially Assistive Robotic Solution for Mild Cognitive Impairment or mild Alzheimer's disease
SAREF	Smart Appliance Reference
SASL	Simple Authentication and Security Layer
SAWSDL	Semantic Annotations for WSDL and xml schema
SBI	Southbound Interface

SCADA	Supervisory Control and Data Acquisition
SDN	Software-Defined Networking
SE	Smart Environment
SFC	Service Function Chaining
SLAM	Simultaneous Localization and Mapping
SM	Semantic Mediator
SoS	System of System
SPDI	Security, Privacy, Dependability, and Interoperability
SSC	SEMIoTICS SDN Controller
SSD	Solid State Disk
SSWAP	Simple Semantic Web Architecture and Protocol
TCP	Transmission Control Protocol
TD	Thing Description
TLS	Transport Layer Security
UC	Use Case
UDP	User Datagram Protocol
UI	User interface
URL	Uniform Resource Locator
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VNF-FG	Virtual Network Function-Forwarding Graphs
vSwitch	Virtual Switch
VTN	Virtual Tenant Networks
VXLAN	Virtual Extensible Local Area Network
W3C	World Wide Web Consortium
WP	Work Package
WoT	Web of Things
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
XML	Extensible Markup Language
XSD	XML Schema Definition

1 INTRODUCTION

This deliverable is the first output of Task 2.4 “SEMIoTICS architecture design”. It provides an initial 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 first version of architecture of the pattern-driven SEMIoTICS framework aims to address such challenges of current implementation and deployment stack of IoT applications such as dynamicity, scalability, heterogeneity and end-to-end security/privacy. Design of the SEMIoTICS architecture aims to address the aforementioned challenges. Specifically, the functional components of the proposed architecture are presented to provide an overview of the appropriate realization mechanisms. Finally, there are two verticals mapped in the areas of energy and health care and one horizontal in the area of intelligent sensing use cases scenarios to the suggested architecture in order to demonstrate its applicability in different IoT enabling platforms, types of smart objects, devices and types of networks.

The architectural specification and logical composition of architecture has been built upon the general and use case specific requirements identified during the project. Additionally, considering architecture given in the project proposal, three layers (Field Level, SDN/NFV Orchestration Layer and Application Orchestration Layer) have been included into the architecture diagram. Additionally, layer approach corresponds with the research performed within the project hence abovementioned factors led to the vision of the architecture presented in this document.

It is very important to mention the overall interplay of this task with already ongoing tasks within WP2, WP3 and WP4 and its direct connection with WP5 and WP6. A detailed relation between the described components and specific work packages and tasks addressing the implementation has been provided in section 2. One can find the names of the components, implementation task assignments and information regarding maturity of the specific logical component. Figure 3 showcases indirectly the initial plan for scope of development for each component specifically giving its current status of development. 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 significant number of requirements defined in T2.1 and T2.2, identifying generic framework components able to support diverse use cases, the architecture definition has been a process itself.

Series of face to face and online workshops, triggered many discussions on the framework architecture composition. Since the work has started after WP3 and WP4 were already initiated, specific component and functionalities were identifiable. Once the component identification has been finalized, there has been a series of thorough analysis run along with the requirements identified and work being delivered in entire WP3 and WP4 to ensure including all necessary elements and not to omit any component needed to fulfil defined requirements.

Creation of first version of SEMIoTICS architecture diagram has been followed by vivid discussions within WP2, WP3 and WP4 workshops and further fine tuning has been constantly provided. Final draft version has been discussed in Barcelona during project meeting and approved by entire consortium after some minor changes introduced. Result of that process is demonstrated in this document as first version of the SEMIoTICS architecture.

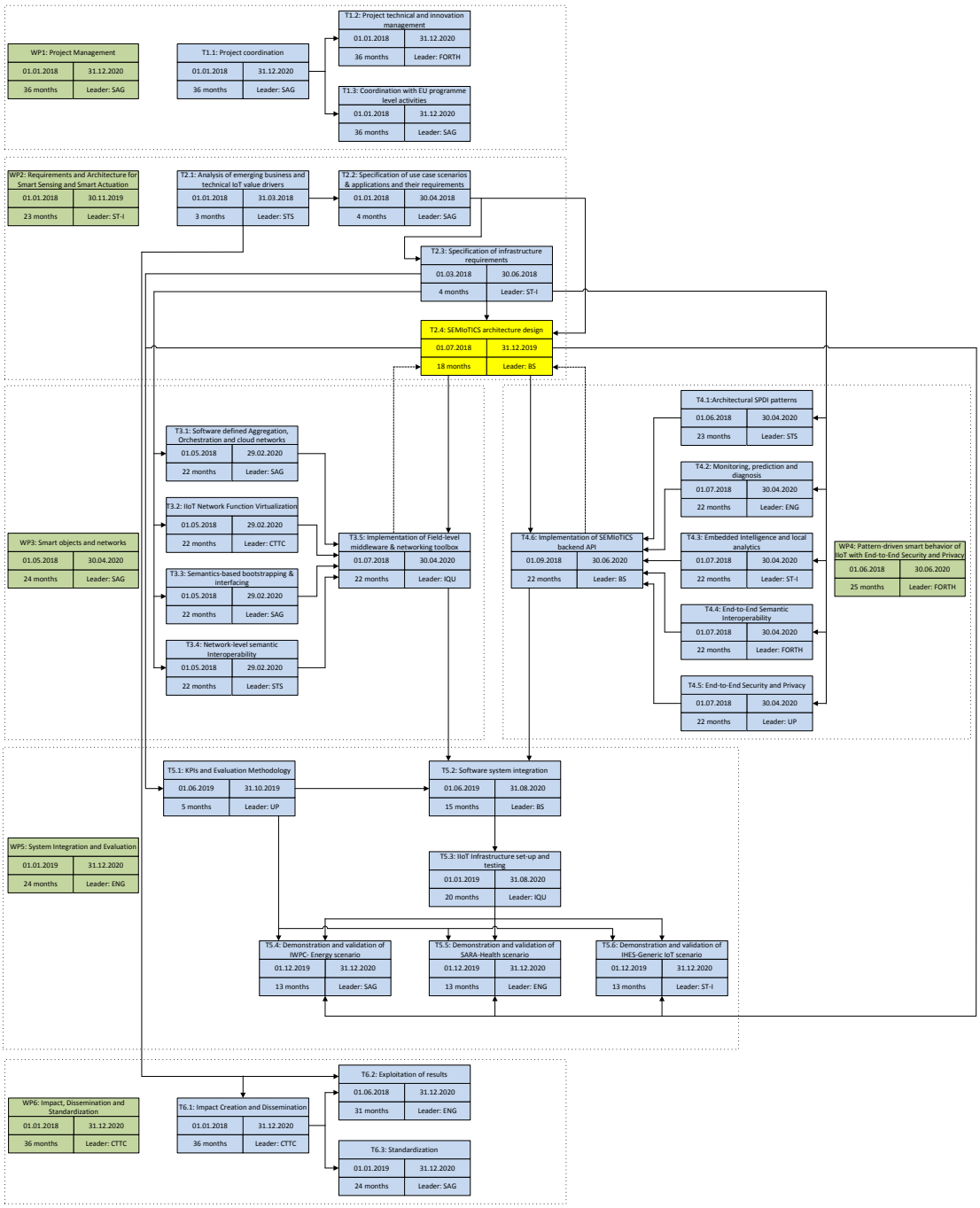
The deliverable is structured as follows: Section 2 presents an overview of the envisaged, deployed and developed SEMIoTICS architectural diagrams. Section 3 provides descriptions of generic components as building blocks of the framework as per architectural layer.

Section 4 has been devoted to use case specific architecture and its components hence gives an overview how SEMIoTICS framework support each of the use cases defined in the project.

Section 5 leverages SEMIoTICS for the insertion of new use cases in the framework. Section 6 contains the validation of the framework and the components regarding the requirements and the project KPIs.

Finally, section 7 concludes the deliverable while section 8 is a bibliography.

1.1 PERT chart of SEMIoTICS



Please note that the PERT chart is kept on task level for better readability.

2 SEMIOTICS ARCHITECTURAL FRAMEWORK OVERVIEW

2.1 SEMIoTICS Envisaged Architecture

The main of 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 behaviour 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.

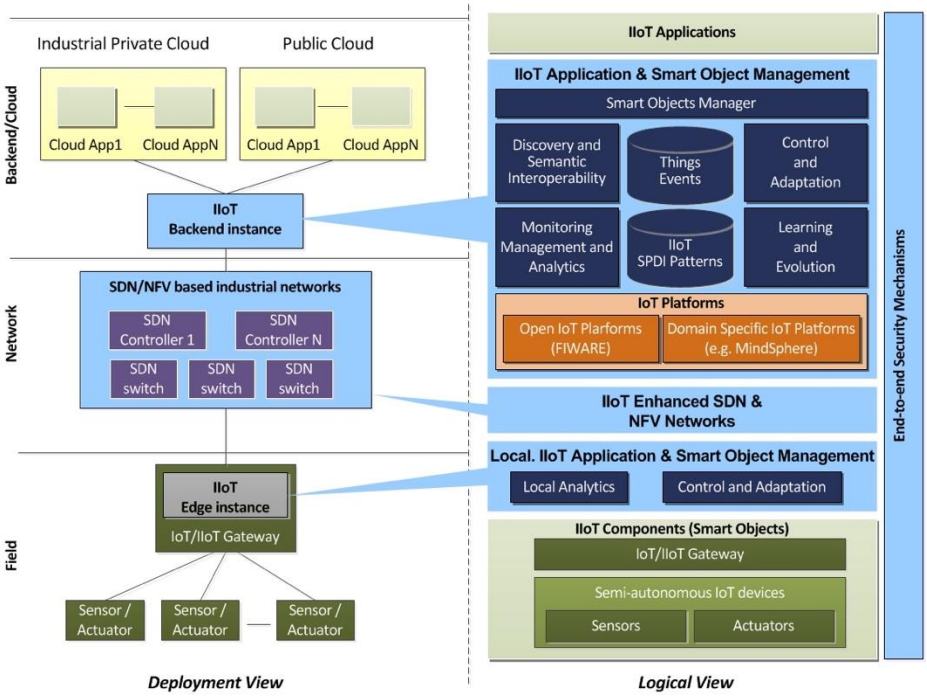


FIGURE 1 ENVISAGED ARCHITECTURE AND DEPLOYMENT OF SEMIOTICS FRAMEWORK

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 . 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.

2.2 SEMIoTICS Architectural Framework Overview

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 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** – consisted of all applications receiving the 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, meet different IoT application requirements in terms of bandwidth, latency and energy efficiency.
- **Field layer** – responsible for hosting heterogenous types of IoT devices. It provides semantic interoperability between IoT devices and seamless flexibility.

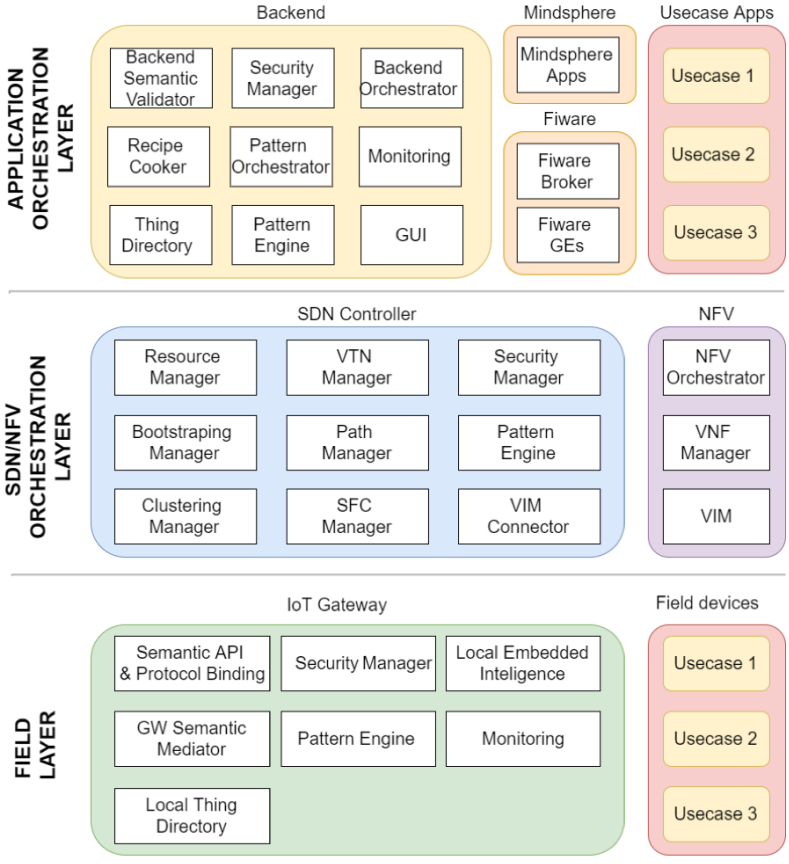


FIGURE 2 - SEMIOTICS ARCHITECTURE DIAGRAM

By designing the architecture this way, all of the requirements and assumptions of the project are fulfilled. Figure 2, 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 is presented further. In the following subsections, the background and the analysis and the component that constitute these layers are described.

2.3 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 is 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 new 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**[5]: The aim 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**.
  - **Agile IoT**[6]: 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**[7]: 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 applications 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.

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

[6] an Adaptive & Modular Gateway for the IoT, IoT EPI, 2016-2018, https://agile-iot.eu/

[7] Bridging the Interoperability Gap of the Internet of Things, IoT EPI, 2016-2018, http://big-iot.eu

- **FIWARE[8]:** 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 SEMIoTICS framework or a **VNF Manager** such as Tacker supported by Openstack.
- Commercial solutions such as **MindSphere** IoT platform.

In Figure 3, a more detailed representation of the proposed architecture regarding the deployment details of each existing adapted component or the new developed components is presented.

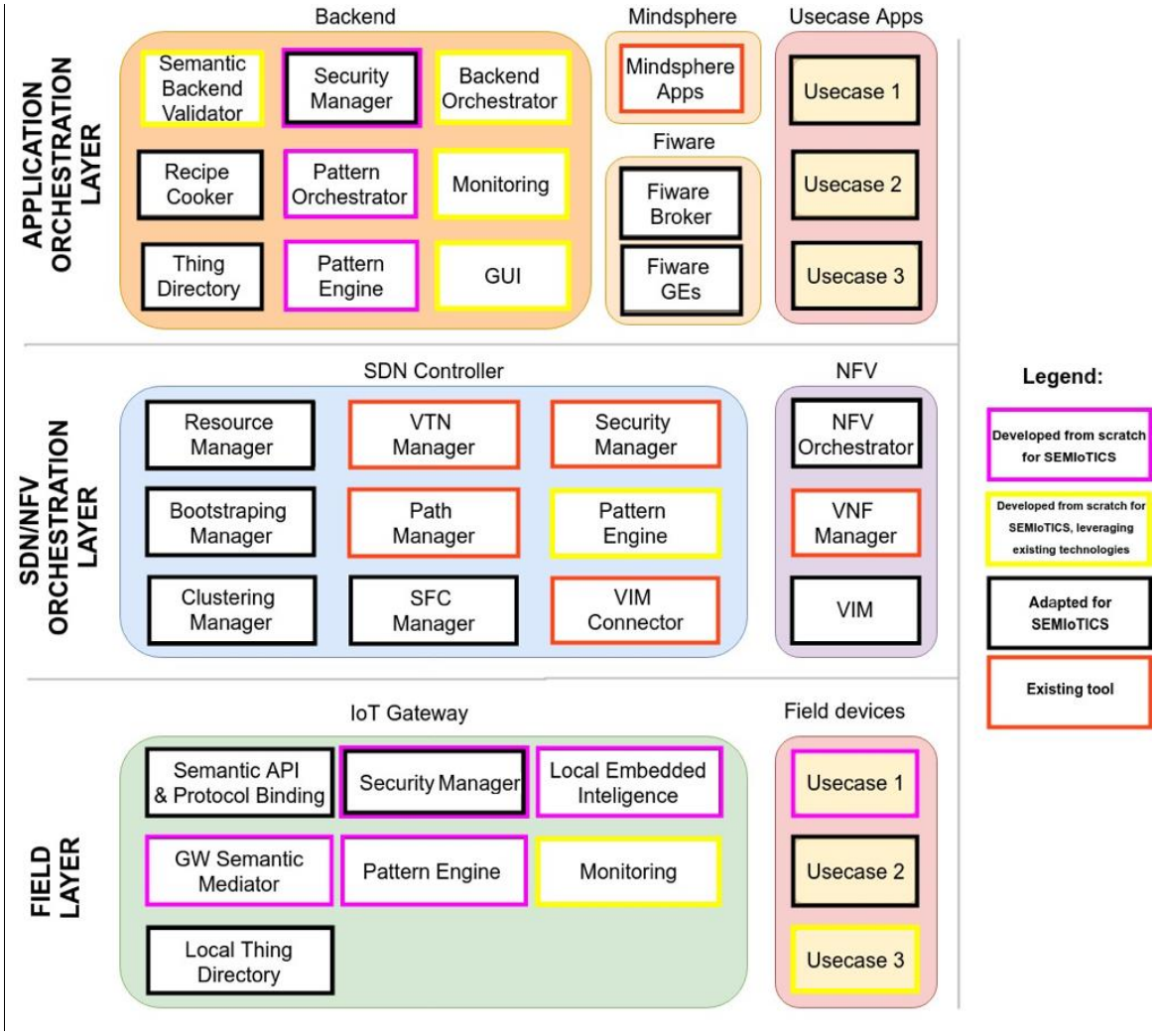


FIGURE 3 ARCHITECTURE DIAGRAM DEPLOYMENT

[8] FIWARE: The Open Source Smart Platform, www.fiware.org

In the following Table 1 there have been presented 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. Further subsections advances the description of the role, core functionalities and deliver more details about each component per layer. This way once can understand rationale of including each component into SEMIoTICS architecture.

TABLE 1 COMPONENT LIST

Component	Layer	Owner (coordinator)	Implementation task	Maturity level
Semantic API & Protocol Biding	Field layer	SAG	3.3	Adapted for SEMIoTICS
Security Manager	Field layer	UP	4.5	Some Subcomponents developed from scratch for SEMIoTICS; some subcomponents adapted for SEMIoTICS
GW Semantic Mediator	Field layer	SAG	3.3	Developed from scratch for SEMIoTICS
Monitoring	Field layer	ENG	4.2	Developed from scratch for SEMIoTICS, leveraging existing technologies.
Pattern Engine	Field layer	FORTH	4.1	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
Use case 1	Field layer	SAG	5.3, 5.4	Developed from scratch for SEMIoTICS
Use case 2	Field layer	ENG	5.3, 5.5	Adapted for SEMIoTICS
Use case 3	Field layer	ST	5.3, 5.6	Developed from scratch for SEMIoTICS, leveraging existing technologies
Security Manager	SDN orchestration layer	FORTH	3.1, 4.5	Existing tool
VIM Connector	SDN orchestration layer	CTTC	3.1, 3.2, 3.5	Existing tool
Clustering Manager	SDN orchestration layer	SAG	3.1, 3.5	Adapted for SEMIoTICS

SFC Manager	SDN orchestration layer	FORTH	3.1, 3.5	Adapted for SEMIoTICS
Bootstrapping Manager	SDN orchestration layer	SAG	3.1, 3.5	Adapted for SEMIoTICS
Resource Manager	SDN orchestration layer	SAG	3.1, 3.5	Adapted for SEMIoTICS
Pattern Engine	SDN orchestration layer	FORTH	3.1, 3.4, 4.1, 4.5	Developed from scratch for SEMIoTICS, leveraging existing technologies
NFV Orchestrator	NFV orchestration layer	CTTC	3.2, 3.5	Adapted for SEMIoTICS
VNF Manager	NFV orchestration layer	CTTC	3.2, 3.5	Existing tool
Virtualized Infrastructure Manager	NFV orchestration layer	CTTC	3.2, 3.5	Adapted for SEMIoTICS
Backend orchestrator	Application orchestration layer	BS	4.6	Developed from scratch for SEMIoTICS, leveraging existing technologies
Recipe Cooker	Application orchestration layer	SAG	4.6	Adapted for SEMIoTICS
Thing directory	Application orchestration layer	SAG	4.6	Adapted for SEMIoTICS
Pattern Orchestrator	Application orchestration layer	STS	4.1	Developed from scratch for SEMIoTICS
Pattern Engine	Application orchestration layer	STS	4.1	Developed from scratch for SEMIoTICS
Monitoring	Application orchestration layer	ENG	4.2	Developed from scratch for SEMIoTICS, leveraging existing technologies

Security Manager	Application orchestration layer	UP	4.5	Some Subcomponents developed from scratch for SEMIoTICS; some subcomponents adapted for SEMIoTICS
Backend Semantic Validator	Application orchestration layer	FORTH	4.4	Developed from scratch for SEMIoTICS, leveraging existing technologies
Use case 1 apps	Application orchestration layer	SAG	5.4	Adapted for SEMIoTICS
Use case 2 apps	Application orchestration layer	ENG	5.5	Adapted for SEMIoTICS
Use case 3 apps	Application orchestration layer	IQU/ST	5.6	Adapted for SEMIoTICS
GUI	Application orchestration layer	BS	4.6	Developed from scratch for SEMIoTICS, leveraging existing technologies
FIWARE/ Context Broker	External IoT Platforms	IQU	3.5, 5.2	Adapted for SEMIoTICS
FIWARE / GE X	External IoT Platforms	IQU	3.5, 5.2	Adapted for SEMIoTICS
MindSphere	External IoT Platforms	SAG	5.4	Existing tool
AREAS	External IoT Platforms	ENG	5.2	Existing tool

3 SEMIOTICS FRAMEWORK AND COMPONENTS

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. Created SEMIoTICS pattern-driven framework will be capable of supporting diverse scenarios with specific focus on Smart Energy, Healthcare and Smart Sensing use cases.

The main scope of SEMIoTICS is to provide a framework to ensure project Dynamicity, Scalability, Heterogeneity, End-to-end Security and Privacy, three main layers of architecture have been distinguished:

- Application orchestration layer – consisted of all applications receiving the 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, meet different IoT application requirements in terms of bandwidth, latency and energy efficiency.
- Field layer – responsible for hosting heterogenous types of IoT devices. It provides semantic interoperability between IoT devices and seamless flexibility

In the next subsections, the description with more details of each layer and deployed component is presented further.

3.1 Application orchestration layer

Application orchestration layer consist of all applications receiving the communication from 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 SPDI patter definition.

3.1.1 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 resources 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. Currently the possible tools for orchestration are revised. While comparing available solutions, their out of the box features and restrictions are considered. One of the tasks in the first draft of WP4 implementation will be focused on choosing most suitable tools for backend orchestration. Approaches which are taken into consideration are:

- Kubernetes[9] on bare metal
- Openstack[10] on bare metal
- Kubernetes on Openstack.

3.1.2 RECIPE COOKER

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

Core Functionalities:

- Creating recipes,
- Transforming recipes into understandable rules

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 4.. Besides the workflow of the recipe, QoS constraints and SPDI patterns can be defined on 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, user only sets SPDI properties (e.g. latency, rate).

9 https://kubernetes.io

10 https://www.openstack.org

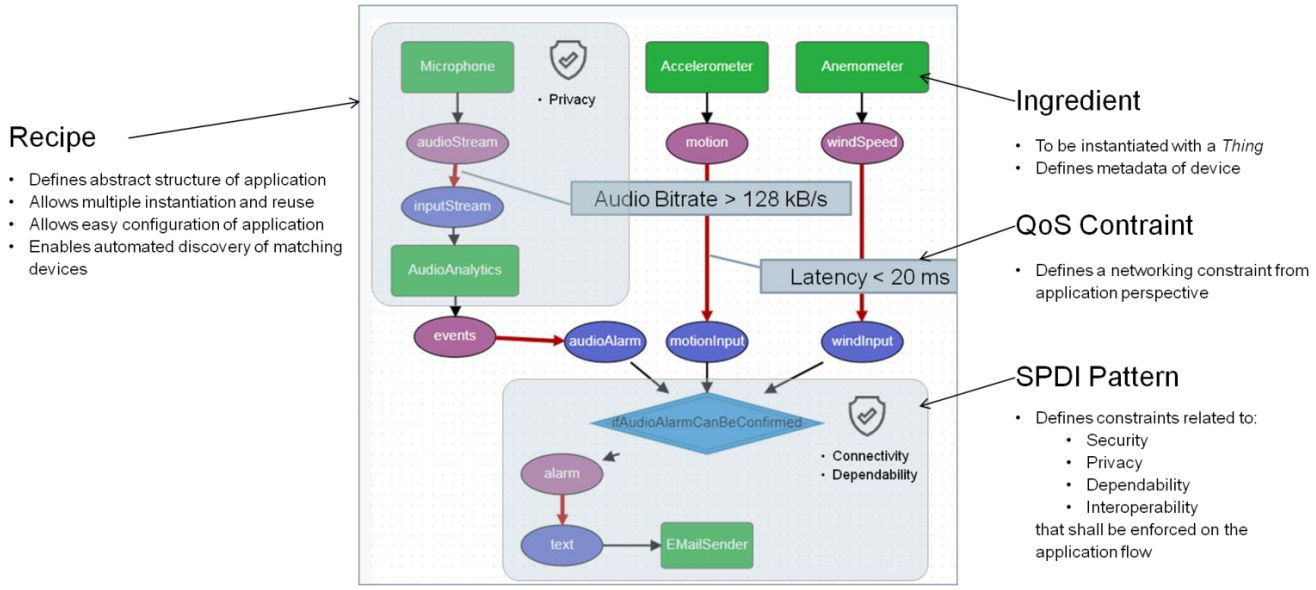


FIGURE 4: DRAFT OF RECIPE COOKER UI

3.1.3 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 [11]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.

3.1.4 PATTERN ORCHESTRATOR

11 https://www.w3.org/WoT/

Overview: This module is responsible for translating cooked recipes into patterns and passing them to pattern engines on each layer.

Core Functionalities:

- Translating recipes from Recipe Cooker
- Sending recipes to the according Pattern Engine

Details: The Pattern Orchestrator module features an underlying semantic reasoner able to understand "cooked" recipes, as received from the Recipe Cooker module (see 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), 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.

3.1.5 PATTERN ENGINE

Overview: Module responsible for enforcing patterns as provided by pattern orchestrator.

Core Functionalities:

- Enforcing patterns provided by Pattern Orchestrator

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

Using said patterns and the Drools[12] rule engine, along with monitoring capabilities present at the backend layer, the Pattern Engine will be able 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.

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,

12 Drools Business Rules Management System (BRMS) https://www.drools.org

e.g. at runtime it is detected that one of the involved components fails to satisfy this requirement, replacing it with an alternative one.

3.1.6 MONITORING

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

Core Functionalities:

- Generating high-level events

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).

Error! Reference source not found. presents the main required inputs and outputs of the SEMIoTICS monitoring component.

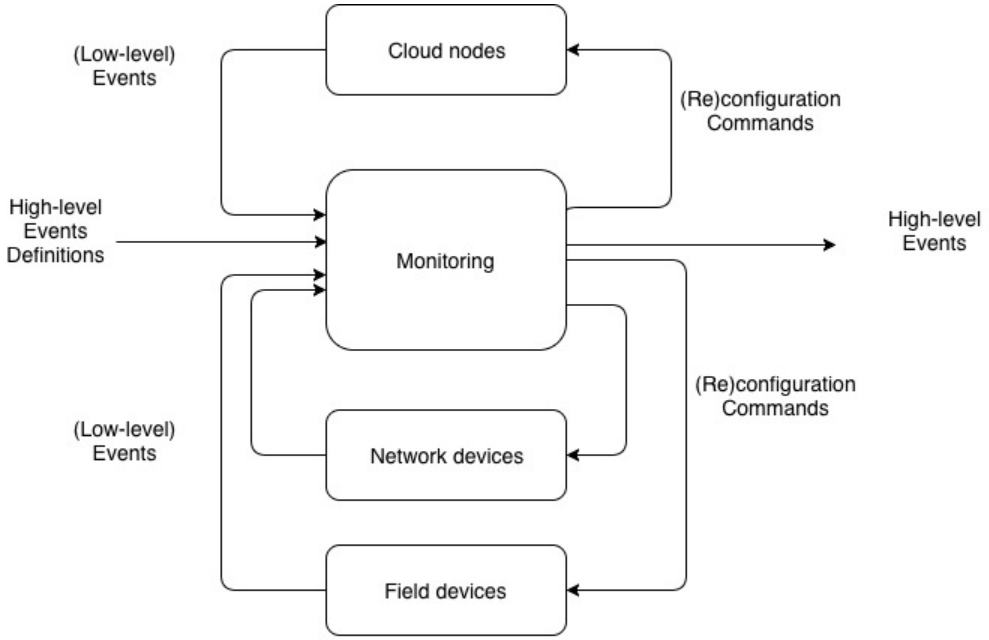


FIGURE 5 - MAIN INPUT AND OUTPUT OF THE MONITORING COMPONENT

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 of the technology used to implement a computational node.
- **High-level events definitions**: the conditions stating whether a new event should be generated in 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.

3.1.7 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: Security Manager 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) does 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 authentication 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 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 on or more roles such as regular user, security analyst, etc. Access rights are defined by security policies which are stored in the security manager. This includes generating keys for attribute-based encryption.

3.1.8 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,
- Transferring requests to Semantic API & Protocol Binding

Details: The aim of 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 3.1.3), in order to detect any potential semantic conflicts between the interacting domains.
- Connecting with Recipe Cooker (Section 3.1.2) and Semantic Edge Platform (in the field) to resolve these semantic conflicts using the Adaptor Nodes that configure an Interaction Pattern in accordance to the application's requirements.
- Transferring the translated request to the Semantic API & Protocol Binding component (Section 3.3.1) which is responsible to trigger the GW Semantic Mediator (Section 3.3.4) in the filed layer, in order to send the request in appropriate format to the target Thing (actuator).

3.1.9 GRAPHICAL USER INTERFACE

Overview: A component responsible for the presentation layer

Core Functionalities:

- Visualization of SPDI pattern monitoring, pattern details, recipes
- Interacting with Things,
- Performing CRUD operations on Thing Descriptions,
- Collecting data gathered from IoT devices,
- Providing user dashboard interface,
- Providing routing to other SEMIoTICS's components

Details: Graphical User Interface is a component responsible for giving meaningful insights into the platform and centralized visualization of the whole framework as well as is 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 a subject to specific task within the project.

3.1.10 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 constrains and monitoring and enforcing them in a decentralized edge infrastructure of a wind turbine,
- The distributed application transfers video data to 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, motors, strain gauges, 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 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 namely Condition monitoring and Predictive maintenance. These apps take the current and historical data of Cloud Thing Directory in order to monitor the condition/predictive maintenance of the Thing/s.
- **Edge Apps:** IoT Gateway (SEMIoTICS Edge) apps can be run from the backend.
- **Data meant for 3rd party OEM vendor:** 3rd party OEM vendors can get access to their data through MindSphere Apps installed at their location.

3.1.11 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 heterogenous (CoAP and ZigBee) devices
- Aggregate and analyze data from heterogeneous devices
- Distributed AI at the edge (e.g. Gait Analysis)
- Manage 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) requires the 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 CIoE-IoT platform (see Section 3.4.4) for what concerns the management of IoT requirements.

3.1.12 USE CASE 3 APPLICATIONS

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 conveniently 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 IHES Supervisor Service deployed at the Field layer.
- Exposing interface with global database that stores all events and related event generated by a set of IHES Sensing Units associated with a specific IHES Supervisor on a given gateway.
- Visualization and management of alerts generated by the IHES system.

Details: As a testbed application for 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 app will report in a simple GUI the environmental processes monitored by the system and will report any (relevant) anomaly detected by the system at single / multiple nodes level. The application will provide at a glance a complete overview of the relevant events clustered 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).

3.2 SDN/NFV orchestration layer

NFV offers flexible, programmable, dynamic, scalable and easy ways to reconfigure network resources in order to provide the QoS demanded by SEMIoTICS IoT use cases. To this end, the SEMIoTICS NFV approach is as follows: general purpose hardware devices are considered in 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 6, the set of physical hardware resources, the virtualization layer and the virtualized computing storage and networking resources is 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.

3.2.1 VIRTUALIZED INFRASTRACTURE MANAGER

Overview: Virtual Infrastructure Manager (VIM) is a backend component which is aware of the physical infrastructure (compute, storage and network). 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 (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001), 2014) namely the Infrastructure and Tenant domains. The former contemplates the physical infrastructure upon which virtualization is performed, and therefore application agnostic; while the latter makes use of virtualized resources to spawn VNFs and create Network Services (NS). Unlike resource allocation in other virtualized environments, in NFVI requests simultaneously ask for compute, storage and network resources. Moreover, NS could be composed of VNFs with hardware affinity/anti-affinity or require specific latency/bandwidth constrains in virtual links connecting VNFs. Such demands occur dynamically, allocating or freeing resources that could then be used for other NS, e.g. scaling up VNF's compute.

A Virtualized Infrastructure Manager (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 6). 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) (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001), 2014).

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 discovery of capabilities of such resources.

As detailed in (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001), 2014) 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 6).
- Abstract the various southbound interfaces (SBI) and network overlays 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.

3.2.2 NFV ORCHESTRATOR

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

Core Functionalities:

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

Details: NFV MANO framework is composed of a Virtualized Infrastructure Manager (VIM), VNF Manager (VNFM), and NFV Orchestrator (NFVO) (see Figure 6). 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 relates to providing each VNF with the NFVI resources they need[13]. 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 (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001), 2014). 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.

13 NFVI resources are those that can be consumed by virtualization containers, such as compute (CPU, virtual machines, bare metal hosts, memory), storage (volumes of storage), and network (networks, subnets, ports, addresses, forwarding rules, links).

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 a 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 6), descriptors are a 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 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 a NS which would be formed by one or several VNFFG, VNF, PNF, and VL. It also specifies 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 of 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.

3.2.3 VNF MANAGER

Overview: This module focuses on all virtualization-specific management tasks necessary in NFV framework. This component is responsible for 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 (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001), 2014), 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 a 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 factor, 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 (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001), 2014):

- 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.

3.2.4 VTN MANAGER

Overview: Responsible for assignment of individual network services to various network tenants. It further ensures a separation of L2 traffic in 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 creation of tenant networks and translation of pattern requests into path-request calls to Path Manager in scope of its VTN. VTN Manager will store all resulting data structures containing information about reservations and established VTNs in the centralized data store.

3.2.5 PATH MANAGER

Overview: Main network path computation engine of the SDN Controller, responsible for 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 will provide 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 will ensure per-flow isolation and worst-case guarantees at all times.

3.2.6 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 embedding of: i) real-time flows which require dedicated per-queue flow assignments; ii) best effort flows, without queue considerations; and iii) the meter structures for policing purposes.

3.2.7 SECURITY MANAGER

Overview: The security component responsible for 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 accounting 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 exposed them to other controller components as well.

Depending on the requirements of any use case, we can describe two possible scenarios for the SM component:

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

3.2.8 VIM CONNECTOR

Overview: The component responsible for connecting with 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 (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Architectural Framework (ETSI GS NFV 002 V1.2.1), 2014), 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 6 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 6. That is, the management of virtual network resources (e.g.: Virtual Tenant Networks, Virtual Network Functions), 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 cases requirements/constraints.

Infrastructure flexibility is one of the most relevant features provided by Network Functions Virtu alization (NFV) (ETSI, ETSI.org: Network Functions Virtualisation (NFV); Architectural Framework (ETSI GS NFV 002 V1.2.1), 2014) (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[14], dubbed Virtual Tenant Networks (VTN), which are effectively isolated from each other.

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

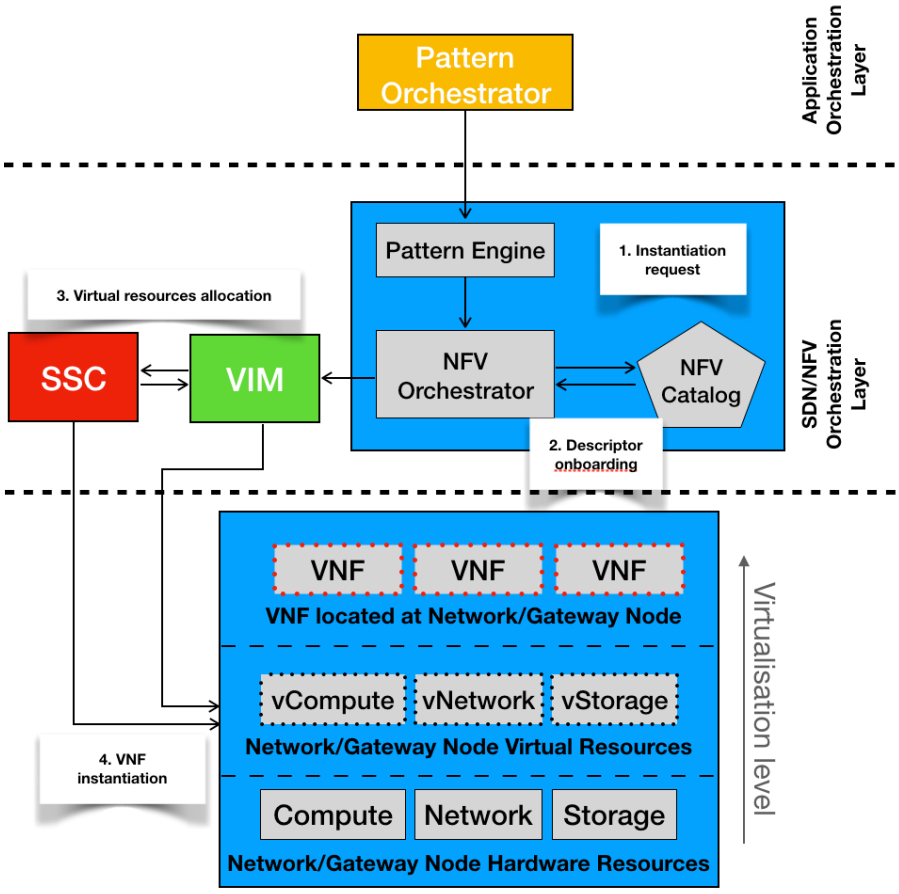


FIGURE 6 - VNF INSTANTIATION EXAMPLE ON A VIRTULISATION CAPABLE NODE WITHIN SEMIOTICS FRAMEWORK

VTN allows creation of network functions as virtual entities without having to consider the physical network. OpenDaylight (ODL, the reference SDN Controller for SEMIoTICS) is equipped with a VTN module for interfacing with Virtualized Infrastructure Managers (VIM) such as OpenStack (OpenDaylight, n.d.). The ODL VTN module is a policy manager that registers any tenant resource in the VIM via ODL's ML2 plugin[15] (OpenStack, ML2 plug-in, n.d.), 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 policy[16] using the VIM's exposed networking APIs and ODL's SBIs.

15 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, 2017).

16 I.e. What nodes should be able to communicate with which ones.

Referring to Figure 6, 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 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 (OpenStack, OpenStack Docs: Networking API v2, n.d.).

3.2.9 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 controller's single point of failure is resolved by means of state replication and fail-over to one of the backup controllers on failure. We plan to address the requirement of strong consistency by basing the implementation of this module on the RAFT consensus-based OpenDaylight Clustering project. To ensure fault-tolerance even in the face of Byzantine/malicious adversaries, this component must be capable of operation in the Replicated State Machine mode. Namely, each instance of the SDN Controller will execute the client operation and propagate its result to the underlying configuration targets that are capable of comparison of the resulting messages and will 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 will be investigated and discussed in more detail in D3.1.

3.2.10 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 constrains that should be applied to packets.

Details: SFC Manager will handle 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 may be 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.

3.2.11 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.
- 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 will guarantee 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 can be 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.

3.2.12 PATTERN ENGINE

Overview: Module responsible for retrieving network specific rules and verifying them.

Core Functionalities:

- Inserting, Modifying, Executing, Retracting patterns in the SDN controller

Details: Pattern Engine (PE) will enable 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. PM can be based on a rule engine which will be 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[17] rule engine appears to be a suitable solution to support design patterns by applying and extending the Rete algorithm[18]. More specifically, since Drools rule engine is based on Maven, it can support the integration of all required dependencies with the ODL controller, as well as the integration of the entities that interact with the controller to run Drools at design and at runtime. Finally, PE will also enable 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 7.

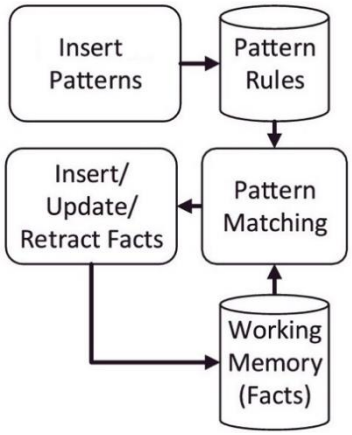


FIGURE 7 - PATTERN ENGINE PROCESS

3.3 Field layer

Field layer is responsible for hosting all types of IoT devices such as sensors and actuators as well as IoT gateway which provides 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 what ensures ability to deliver diverse use cases in various sectors.

3.3.1 SEMANTIC API & PROTOCOL BINDING

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

Core Functionalities:

- Semantic Mapping of brownfield semantic models into IoT semantic models,

17 Drools Business Rules Management System (BRMS) https://www.drools.org

18 Charles Forgy: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: Artificial Intelligence, vol. 19, pp. 17–37, 1982.

- Semantic configuration,
- Providing a 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 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[19], Modbus[20] 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 a 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 Semantic Edge Platform. The platform will be based on 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.

3.3.2 SECURITY MANAGER

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

Core Functionalities:

- Enforcing security policy decision 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.

Sensors and actuators in many cases will be connected to the gateway using low-level protocols and technologies such as MQTT, Bluetooth, etc.; in these cases it simplifies authentication if the gateway contains its own security manager.

19 https://www.profibus.com/

20 https://en.wikipedia.org/wiki/Modbus

3.3.3 LOCAL THING DIRECTORY

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 Local Thing Directory is to store semantic description of Things locally in the Generic IoT Gateway. Semantic Backend Validator will be used to provide these descriptions in accordance to 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 sematic 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, feature of interest etc.). The whole content of a Local Thing Directory will be synchronized with the Thing Directory, running in the Backend.

3.3.4 GW SEMANTIC MEDIATOR

Overview: Module responsible for semantic mappings between different data models.

Core Functionalities:

- Integrating brownfield semantics with IIoT semantics

Details: The goal of semantic integration (see SEMIoTICS deliverable D3.3) is to enable 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 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.

3.3.5 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:

- Generating high-level events

Details: This module is part of 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 field level. The consequent 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 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.

3.3.6 PATTERN ENGINE

Overview: Module responsible for retrieving gateway specific rules and verifying them by interaction with monitoring module.

Core Functionalities:

- Enforcing patterns provided by Pattern Orchestrator

Details: Pattern engine in the 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, the module will be able to host patterns in an executable form compared to the pattern rules as provided in the other layers. The executable 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 and sensors and actuators on the field. Finally, the Pattern Engine in gateway will keep stored the patterns executable in a local knowledge base that will be updated from by the pattern orchestrator as needed and requested.

3.3.7 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:

- Executing use case specific application logic

Details: A 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.

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.

3.3.8 USE CASE 1

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

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

- An IIoT Gateway with compute capacity to instantiate IIoT applications as VMs. The IIoT gateway will 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 the 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.

3.3.9 USE CASE 2

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

Details: The SARA field devices include:

- A smartphone (iOS or Android) acting as hub for the Bluetooth wearable devices forming the Body Area Network subsystem of the SARA solution.
- A Robotic Rollator which 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 board computer acting as 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 hub for the ZigBee devices instrumenting the Smart Environment subsystem of the SARA solution.

3.3.10 USE CASE 3

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

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[21] 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 the IHES Supervisor Service and the local MQTT broker are deployed. Part of the analytics will run on that IHES Supervisor Service that will have also a MQTT client interface to interoperate with the GW Semantic Mediator of the SEMIoTICS architecture

3.4 External platforms’ components

3.4.1 FIWARE CONTEXT BROKER GE

21 https://www.st.com/en/evaluation-tools/nucleo-f401re.html

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

Core Functionalities:

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

Details: In FIWARE, the Orion Context Broker fulfils 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 is 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.

3.4.2 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 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 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.

3.4.3 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[22] is a Cloud platform customized for industrial IoT applications, which is developed by Siemens.. The data is 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 3.3). The gateway also provides the semantics of this data. The same semantics will be used to create 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.

3.4.4 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.

22 www.mindsphere.io

4 USE CASE SPECIFIC ARCHITECTURE

This chapter showcases three demonstration scenarios (use cases) to be presented within SEMIoTICS framework. Each use case is described from the perspective of the generic SEMIoTICS architecture with special focus on showing 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. In further project progress, within further SEMIoTICS framework and use case development, it will be identified which components explicitly will be used for the final demonstration purposes and what functionalities will be leveraged. Outcome of such analysis and research will be documented within dynamic architecture delivered in D2.5 as well as WP5 validation tasks (T5.3, T5.4 and T5.5).

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, Predictive Maintenance and Monitoring etc. 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 which 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, which will lead in smart actuation to protect the critical infrastructure of renewable energy resources.

4.1.1 USE CASE 1 ARCHITECTURE

Use case specific components/hardware are shown in the following SEMIoTICS architecture layers namely, Field, Network and Application orchestration layer. The exact interfaces of different architectural components will be detailed in WP3 and WP4 deliverables respectively and UC1 specific architecture will be detailed in the final high-level architecture of SEMIoTICS (D2.5).

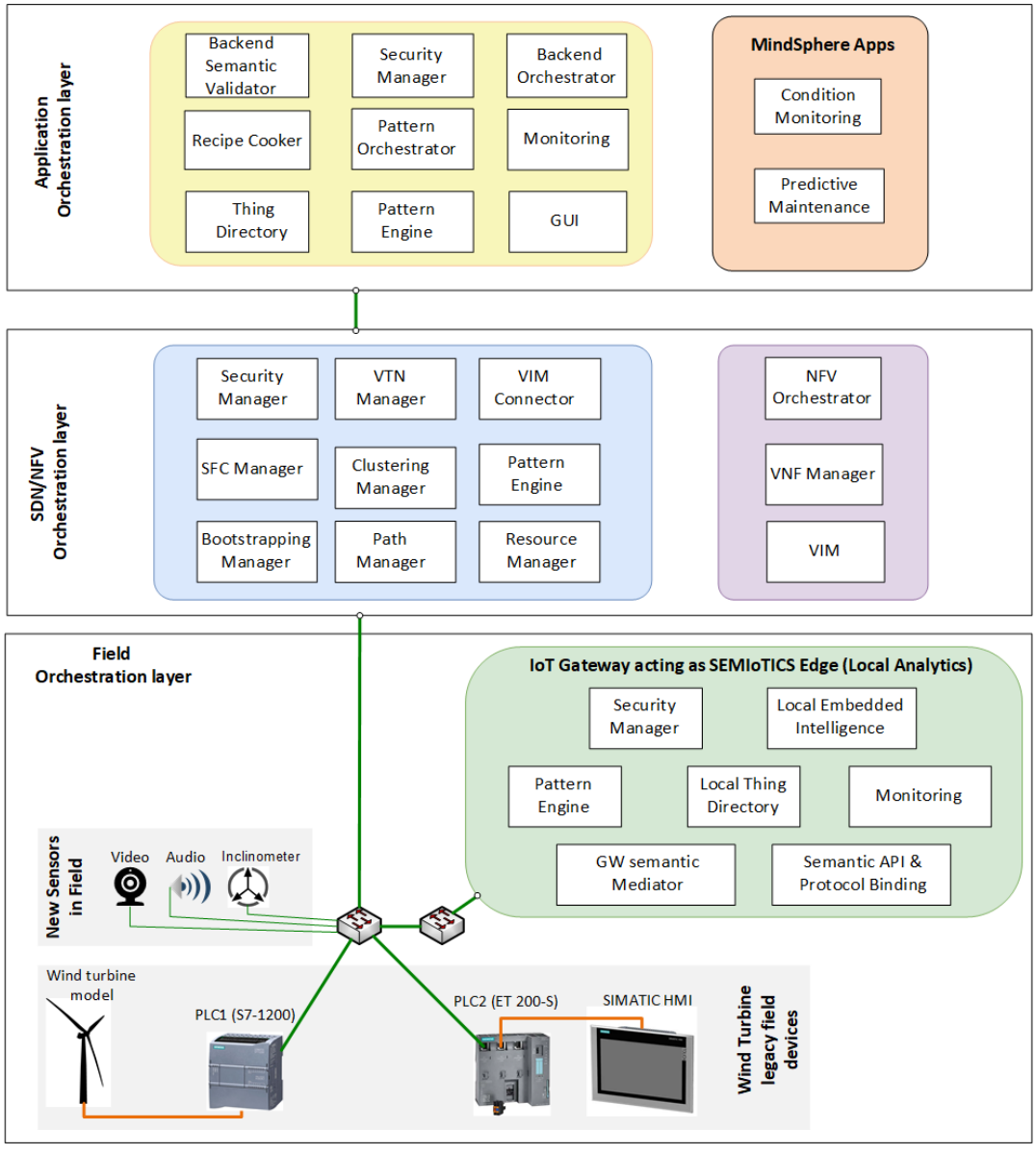


FIGURE 8 - USE CASE 1 - ARCHITECTURE

As depicted in the Figure 8, 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 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.

4.1.2 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 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.

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 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 CIoE-IoT platform since it uses
- **Gait analysis:** utilizes the CIoE-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 are 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 Tacking:** 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 comprises of:

- **Fall detection:** is the module responsible for the detection of patients' falls.
- **Gait Analysis:** is the module to perform 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 try 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 CIoE-IoT platform.
- **HR Dialog Management:** is the counterpart of the HR Dialog Management module available in the cloud. It is intended to support 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.

4.2.2 MESSAGE FLOW AND COMMUNICATION

The Figure 9 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 a backend services by means of LTE connectivity.
- The Home Gateway supports the communication between the field nodes a 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)

Figure 9 depicts the main software modules envisaged in the preliminary design for the SARA UC.

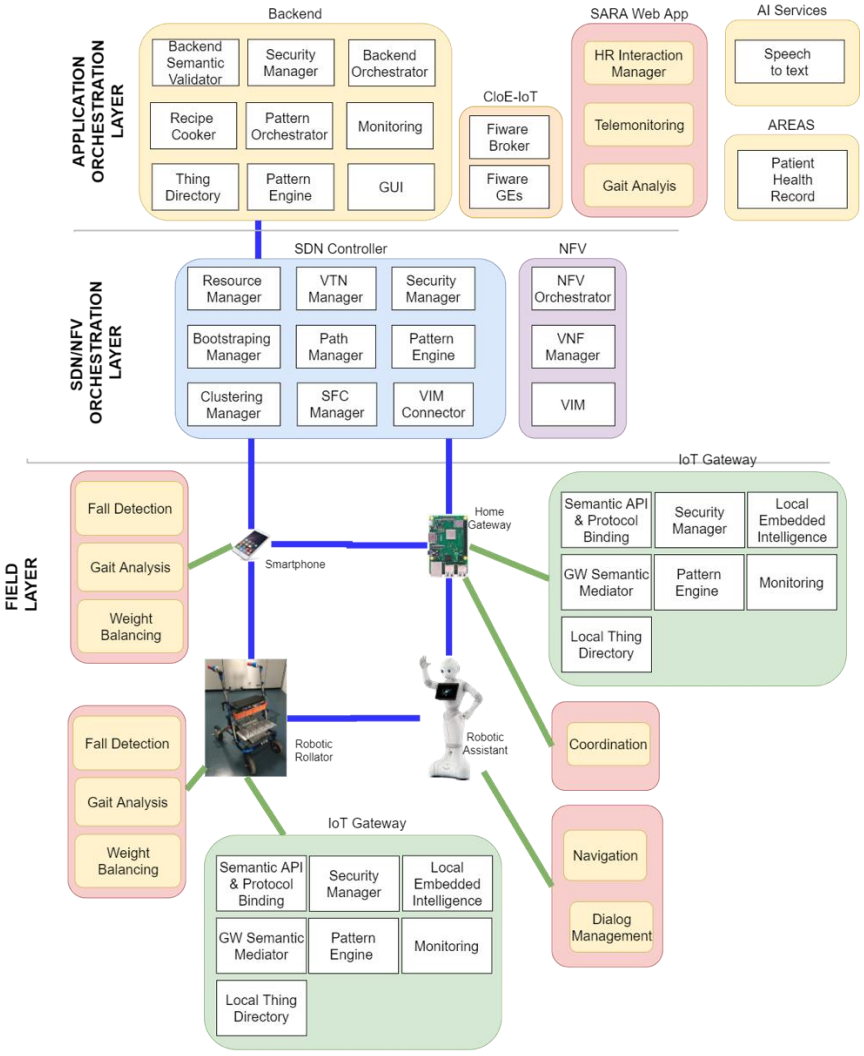


FIGURE 9 USE CASE 2 - MAIN MODULES OF THE SARA UC

4.3 Use case 3 – Smart Sensing

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

The IHES use case offers an interesting specular approach to this scenario (somehow influenced by “Edge Computing” or “Pervasive Computing”). Main assumption is that intelligent data processing shall take place at sensor level, and that distributed data classification and clustering 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 on 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 (C. Krupitzer, 2015) (Roveri, 2017), 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 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.

4.3.1 USE CASE 3 ARCHITECTURE

Local Embedded Intelligence is a key aspect in SEMIoTICS. It will 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 algorithms are moved to the edge: in SEMIoTICS they will be mainly deployed in the IoT Gateway and the Field level Node Devices. These aspects will be considered and covered in full details as part of the deliverable D4.3 activities. These algorithms will be deployed and instantiated within a specific architectural component named “*Local Embedded Intelligence Component*” (see Figure 2 of the general architecture) that will implement all of those algorithms and will 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 field devices level of UC3 will be the deployment of such

Local Embedded Intelligence component directly down to each single field device node, tightly coupled with data gathered from sensors.

Local analytics algorithms will be 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 them relevant info sent as events to other layers. The communication interfaces will be 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 nodes of a data-driven unsupervised (data) monitoring infrastructure. This will be achieved as part of WP4 activities by defining the specific low-level architecture and incremental mapping of all required local analytics algorithm to sustain those technologies that will be integrated as part of WP3 activities.

The algorithms that will be developed will be derived from well-known approaches in the field of Artificial Intelligence (AI), Statistical Analysis, Causal inference, and prediction analysis. Differently from widely used algorithms deployed at central Cloud level, light-weight versions of those will be derived for accounting the specific needs of these constrained domain architectures. Most of those algorithms will be 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 will host this component there will be different deployments of the same component functionalities done by exploiting their specific device capabilities / limitations / ecosystems. The main factors that will drives the light-weight porting is 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 will be 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 will be implemented as a reference implementation to demonstrate the system capabilities. The monitoring web app will basically report 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 is reported in Figure 10.

As part of the core functionalities of the IHES system, a lightweight version of algorithms focused at data monitoring and data model prediction will be derived from the generic SEMIoTICS components and will be deployed on the IHES Sensing unit. This functionality is responsible at edge device level to support monitoring of the relevant events generated by the data-reduction algorithms developed. These algorithms will be deployed likely in different instances due the specific constraints and middleware's available for a given device / target ecosystem. In the case of UC3 the devices will be 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.

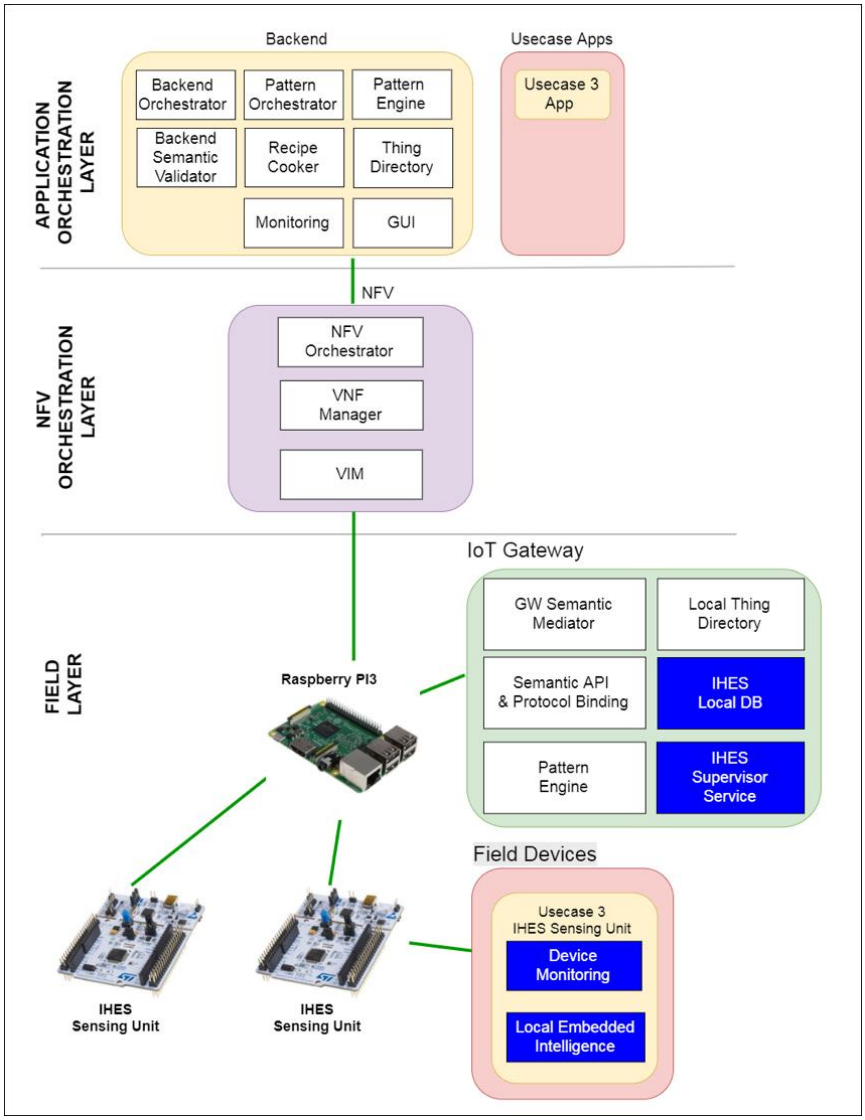


FIGURE 10: USE CASE 3 SYSTEM ARCHITECTURE

Specifically for UC3, IHES Generic IoT use case the monitoring is intended on real-time time-variant generic signal 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 case will propagate them to the upper level of SEMIoTICS architecture by leveraging on the GW Semantic Mediator. A functional flow with a very high-level overview of the envisaged architecture and algorithms mapping are shortly presented in Figure 11. Data monitoring will be locally done at field device level by analyzing the event patterns generated at device level evaluating them by statistical methods.

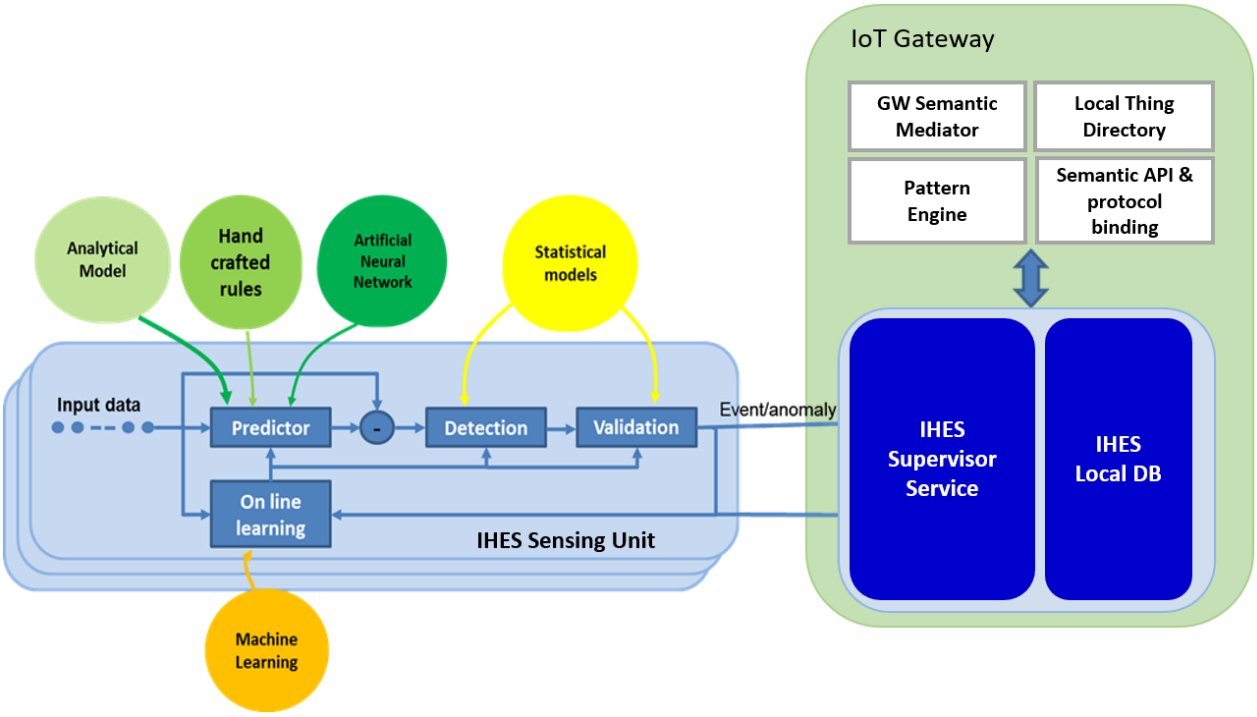


FIGURE 11 - IHES LOCAL ANALYTICS (IOT GW + DEVICE NODE)

The generic architecture of the IHES use case is shown in Figure 11. In blue are reported the components that are specific of the UC3 or that has been derived from other components but has been adapted to the specific field device platform. The system is composed ideally by two different kind of functional modules: a set of IHES Intelligent sensing nodes and a set of IHES Supervisor Service. They act as asynchronous coordinated communicating using a specific defined JSON messages sent to a MQTT broker deployed at IoT local gateway. Both modules will implement a local analytics processing pipeline composed by 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 a 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 algorithm 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. As an example the Recipe cooker mentioned in section 3.1.2 will not be relevant for the IHES system because the Recipe cooker is more oriented to target a Cloud Centric Approach, so we do not envisage its adoption in UC3. Moreover the IHES Supervisor Service module mapped onto the IoT Gateway will be interfaced to a NoSQL database that will collects all the relevant events (i.e. anomalies) collected by the system during its operations, in order to make them available to the other components of SEMIoTICS interfacing the ecosystem at gateway level by a subset of SEMIoTICS components available at this level (e.g. the Gateway Semantic Mediator, etc.) as reported in Figure 10 that highlights also the distributed nature of this system and how the analytics is 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 rational on the technical choices made) will be made available in D4.3 that will cover all the aspects related to the local embedded analytics in SEMIoTICS.

4.3.2 USE CASE 3 MESSAGE FLOW AND INTEROPERABILITY

The pattern communication on UC3 is heavily impacted by the underlying computational paradigm: distributed computing systems need to have as a requisite strong communication capability in term 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 nodes level. Semantics interoperability among heterogeneous devices and consistent message pattern flows need to be carefully design when devices start to exchange not only raw data or simple events, but more complex messages patterns, used to describe more interaction between intelligent things. IHES devices will be able to join / detach from a local cluster computation network coordinated by a local IHES Supervisor Service node.

Our use case will deploy 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 will be used both for handling custom message patterns between the IHES nodes and the IHES Supervisor Service, and for interfacing the IHES Supervisor Service with the GW semantic mediator and Semantic API & protocol binding components of SEMIoTICS. More details about these semantic communication patterns and a subset of those used during the bootstrap interfacing phase has been provided in D3.3.

5 LEVERAGING SEMIOTICS FRAMEWORK FOR NEW USE CASES

SEMIoTICS can framework offer multiple core functionalities that can be used to support a variety of IoT use cases. In order to be able to fully utilize proposed functionalities of the framework, there is number of steps which needs to be followed. The key areas of SEMIoTICS which needs to be taken into considerations and analysis run when implementing new use case within SEMIoTICS framework are 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 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. Recipe cooker delivers functionality of modelling 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).
- **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 of platform monitoring in 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 filed 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

6 VALIDATION

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

6.1 Related Project Objectives and Key Performance Indicators (KPIs)

The objectives of the related T2.4 and their mapping to D2.4 content is summarized in the following table.

TABLE 2 TASK'S OBJECTIVES

T2.4 Objectives	D2.4 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
• Architectural and functional specification driven by the requirements identified in Tasks 2.1 and 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.	3.1, 3.2, 3.3, 3.4
• 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 **SEMIOTIC's objectives:**

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 the SEMIoTICS framework and building blocks

6.2 Project requirements mapping to Tasks and Architectural Components

In this section one can find all requirements which were derived in the project and documented in D2.3 being mapped to the Tasks which addresses those requirements as well as mapping to the architectural logical components. A more detailed presentation of the correlation between the project requirements and the project tasks and components can be found below.

6.2.1 GENERAL PLATFORM, BACKEND/NETWORK/FIELD PROJECT REQUIREMENTS MAPPING TO TASKS 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	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	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	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	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																	

6.2.2 SECURITY AND PRIVACY PROJECT REQUIREMENTS MAPPING TO TASKS ARCHITECTURAL COMPONENTS

SDN Layer		NFV Layer		WP3 - Smart objects and networks (SAG)																		
Backend Layer		Field Layer		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 - Interoperability	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
IoT Security and Privacy Requirements	R.S.1	Yes	MUST																	o	o	
	R.S.2	Yes	MUST	o																		
	R.S.3	No	SHALL																			
	R.S.4	Yes	MUST									o	o							o	o	
	R.S.5	Yes	SHALL																			
	R.S.6	No	SHALL																			
	R.S.7	Yes	SHALL	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																			
	R.S.18	Yes	SHALL																			
	R.S.19	Yes	SHALL																			
	R.S.20	Yes	MUST																			
	R.P.1	Yes	MUST																			
	R.P.2	Yes	MUST																			
	R.P.3	Yes	MUST																			
	R.P.4	Yes	MUST																			
	R.P.5	Yes	MUST																			
	R.P.6	Yes	MUST																			
	R.P.7	Yes	MUST																			
	R.P.8	Yes	MUST																			
	R.P.9	Yes	SHALL																			
	R.P.10	Yes	SHALL																			
	R.P.11	No	SHALL																			
	R.P.12	Yes	MUST																	o	o	
	R.P.13	Yes	SHALL																			
Generic Security and Privacy Requirements for the SEMIOTICS Use CASES	R.GSP.1	Yes	MUST																			
	R.GSP.3	No	SHALL																			
	R.GSP.4	No	MUST																			
	R.GSP.6	Yes	MUST																			
	R.GSP.7	Yes	MUST							o												
	R.GSP.8	No	MAY							o												
	R.GSP.9	Yes	SHALL																			
	R.GSP.10	No	SHALL																			
	R.GSP.11	No	SHALL																			
	R.GSP.12	Yes	MUST																			
	R.GSP.13	Yes	SHALL																			

SDN Layer		NFV Layer		WP4 - Pattern-driven smart behavior of IIoT with End-to-End Security and Privacy (FORTH)																					WP5 (ENG)		
Backend Layer		Field Layer		T4.6 - Implementation of SEMIoTICS backend API (BS)																					T5.2-T5.3		
Topic	Req-ID	Functional	Req Level	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	T5.4- UC1	T5.5-UC2	T5.6 - UC3
				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	
	R.S.2	Yes	MUST	o			o												o							o	
	R.S.3	No	SHALL	o				o																		o	
	R.S.4	Yes	MUST	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								
	R.S.8	No	SHALL																								
	R.S.9	No	SHALL																								
	R.S.10	No	SHALL																								
	R.S.11	No	MUST																								
	R.S.12	No	MUST																								
	R.S.13	No	SHALL																								
	R.S.14	Yes	SHALL																								
	R.S.15	No	MUST																								
	R.S.16	Yes	MUST																								
	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		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	
	R.P.13	Yes	SHALL																o							o	
Generic Security and Privacy Requirements for the SEMIOTICS Use CASES	R.GSP.1	Yes	MUST																							o	
	R.GSP.3	No	SHALL	o														o	o	o	o					o	
	R.GSP.4	No	MUST	o	o	o	o	o																		o	
	R.GSP.6	Yes	MUST																								
	R.GSP.7	Yes	MUST	o			o																				
	R.GSP.8	No	MAY								o																
	R.GSP.9	Yes	SHALL																								
	R.GSP.10	No	SHALL	o		o	o	o																			
	R.GSP.11	No	SHALL																								
	R.GSP.12	Yes	MUST																								
	R.GSP.13	Yes	SHALL	o		o			o	o	o																

6.2.3 USE CASE 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 - Bootstrapping	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	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	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	BTS	FORTH	IQU	BTS	BTS	BTS	FORTH	FORTH	ENG	ENG	ENG	ST	ST	FORTH	SAG	SAG	FORTH	UP	FORTH	UP	UP	BS	BS	BS	SAG	ENG	ST
UC1 Wind Energy	R.UC1.1	Yes	MUST	o	o	o		o		o						o	o			o	o		o			o																			o		
	R.UC1.2	Yes	MUST																																										o		
	R.UC1.3	Yes	MUST	o				o												o	o		o			o								o	o									o		
	R.UC1.4	Yes	MUST	o		o		o												o	o		o			o																			o		
	R.UC1.5	Yes	SHALL	o				o	o											o	o		o		o	o																			o		
	R.UC1.6	Yes	SHALL	o					o														o		o	o	o													o					o		
	R.UC1.7	Yes	SHALL	o					o																																			o		
	R.UC1.8	Yes	MUST													o	o																o			o	o						o		
	R.UC1.9	Yes	MUST													o	o	o															o			o								o		
	R.UC1.10	Yes	MUST													o																												o		
	R.UC1.11	No	SHALL													o			o																									o		
	R.UC1.12	No	MUST													o		o		o	o												o			o	o						o		
	R.UC1.13	Yes	MUST													o																												o		
UC2 SARA	R.UC2.1	Yes	SHOULD	o																																									o	
	R.UC2.2	Yes	SHOULD	o		o										o				o	o																								o	
	R.UC2.3	Yes	SHOULD	o	o	o		o		o																								o			o								o	
	R.UC2.4	Yes	SHOULD																																										o	
	R.UC2.5	Yes	SHOULD	o		o										o			o																											o	
	R.UC2.6	Yes	SHOULD													o			o														o			o									o	
	R.UC2.7	Yes	SHOULD																				o																						o	
	R.UC2.8	Yes	SHOULD																				o		o		o																		o	
	R.UC2.9	Yes	SHOULD																																										o	
	R.UC2.10	Yes	SHOULD																									o																	o	
	R.UC2.11	Yes	SHOULD																														o			o									o	
	R.UC2.12	Yes	SHOULD								o	o	o	o	o													o																	o	
	R.UC2.13	Yes	MAY																																										o	
	R.UC2.14	Yes	MAY																																										o	
	R.UC2.15	No	SHOULD	o				o		o										o	o		o																						o	
	R.UC2.16	No	SHOULD																																										o	
	R.UC2.17	No	SHOULD	o				o																																					o	
UC3 Sensing	R.UC3.1	Yes	SHOULD																														o			o										o
	R.UC3.2	Yes	MUST													o	o																													o
	RUC3.3	Yes	MUST																												o	o													o
	R.UC3.4	Yes	MUST																												o	o													o
	R.UC3.5	Yes	SHOULD																												o	o													o
	R.UC3.6	Yes	MAY	o	o	o		o		o																					o	o													o
	R.UC3.7	Yes	MUST	o	o	o		o		o													o								o	o													o
	R.UC3.8	Yes	MUST																												o	o													o
	R.UC3.9	Yes	MUST									o	o	o	o	o	o														o	o													o
	R.UC3.10	Yes	MUST																																										o
	R.UC3.11	Yes	SHOULD																												o	o													o
	R.UC3.12	Yes	MUST									o	o	o	o	o															o	o													o
	R.UC3.13	Yes	SHOULD									o	o	o	o	o	o														o	o													o
	R.UC3.14	Yes	MUST	o		o						o	o	o	o	o	o														o	o													o
	R.UC3.15	Yes	SHOULD													o	o						o								o	o	o			o									o
	R.UC3.16	Yes	SHOULD													o							o				o				o	o													o
	R.UC3.17	Yes	SHOULD													o															o	o													o
	R.UC3.18	Yes	MAY																				o								o	o													o

7 CONCLUSION

In this work, we described the SEMIoTICS architectural framework, which addresses the complicated requirements of IoT/IIoT applications such as security, privacy, dependability and interoperability. There has been clarified 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 SEMIoTICS Framework.

As the first draft of the architecture, the work will be continued in parallel with works carried within WP3, WP4 and WP5. Every result of project finding will be consulted and aligned with the initial architectural concepts of the framework architecture. However, if research and development will bring some new concepts, architectural approach will be discussed, and some modifications introduced if necessary.

In the final version of the SEMIoTICS high level architecture deliverable (D2.5), focus will be on the dynamic architecture aspects considering the design decisions made in WP3 and WP4. D2.5 Deliverable will cover topics such as specificity of components interactions, detailed use case massage flows and diagrams describing such interactions together with description of data exchanged.

8 BIBLIOGRAPHY

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

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

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

OpenDaylight. (n.d.). *OpenDaylight Lithium*. Retrieved January 2019, from https://www.opendaylight.org/what-we-do/current-release/lithium

OpenStack. (n.d.). *ML2 plug-in*. Retrieved January 2019, from https://docs.openstack.org/newton/networking-guide/config-ml2.html

OpenStack. (n.d.). *OpenStack Docs: Networking API v2*. Retrieved from https://developer.openstack.org/api-ref/network/v2/

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).

Toghraee, R. (2017). *Learning OpenDaylight: The art of deploying successful networks.* Birmingham: Packt Publishing Ltd.