Ref. Ares(2020)4057784 - 31/07/2020

SEMIoTICS

Deliverable D5.6
Demonstration and validation of IHES - Generic IoT (Cycle 1)

Deliverable release date	31/07/2020
Authors	1. Mirko Falchetto, Danilo Pau (ST) 2. Konstantinos Fysarakis, Manolis Chatzimpyrros, Michalis Smyrlis (STS) 3. Kostas Ramantas (IQU) 4. Jordi Serra (CTTC)
Responsible person	Mirko Falchetto (ST)
Reviewed by	Michal Rubaj (BS), Jordi Serra (CTTC), Ramantas, Kostis (IQU), Michalodimitrakis, Manolis, Nikolaos Petroulakis (FORTH), Konstantinos Fysarakis (STS)
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
Dissemination level	Public

Table of Contents

Acronyms Table	
Acronym	**Definition**
AI	Artificial Intelligence
CDT	Change Detection Test
CPM	Change Point Method
DB	DataBase
ESN	Echo State Network
FL	Field Layer
FW	FirmWare
GUI	Graphical User Interface
IHES	Intelligent Heterogeneous Embedded Sensors
IIoT	Industrial Internet of Things
IoT	Internet of Things
JSON	JavaScript Object Notation
LEA	Local Embedded Analytics
MAC	Media Access Control Address
MANO	Management and Orchestration
MCU	Micro Controller Unit
MOC	MOCkup
MQTT	Message Queuing Telemetry Transport
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
NFVO	NFV orchestrator
RC	Reservoir Computing
SBC	Single Board Computer
SPDI	Security, Privacy, Dependability, and Interoperability
UC	Use Case
UI	User Interface

1 INTRODUCTION

This deliverable is the first summary related to reporting all activities done so far within Task 5.6 ("Demonstration and validation of IHES-Generic IoT scenario"), focusing on Cycle 1 of said activities.

In more detail, the reporting period covers all technical integration activities done during cycle 1 integration, while these will be later finalized in D5.11 that will cover the complete UC3 demonstrator platform as it will be delivered at the end of cycle 2 integration. A short overview on next activities related to cycle 2, i.e. in D5.11 – "Demonstration and validation of IHES- Generic IoT (Final)" will be reported along this deliverable sections depending on the technical topics discussed to provide to the reader on anticipation of what has been planned for cycle 2 integration period and to provide a clear understanding of the undergoing technical activities as a whole.

Section 2 introduces a short recap of the target motivating scenarios to be showcased by the UC3 demonstrator within SEMIoTICS, a quick overview on the challenges associated to UC3 scenario and a presentation of the sub use case scenarios (section 2.3) derived from the one presented in section 2.1, that will lead the incremental integration from field level to backend level of the demonstrator.

Section 3 presents the SEMIoTICS UC3 testbed based on OpenStack and VNF virtualization, that will be used to showcase the three incremental sub use case storylines that are discussed in sections from section 4 to section 6. Final sections are dedicated to derive the conclusions of this 1st cycle of integration and a summary related to the KPIs associated to the final demo architecture whose complete results will be presented in D5.11.

The purpose of this document is to describe how the UC3 demonstrator (use case and associated sub-use cases) will be incrementally implemented along whole Task 5.6. The consortium maintains this as a living document, that is updated recurrently as the demonstrator mature.

2 USE CASE DESCRIPTION

2.1 UC3 storyline and scenarios

On April 6th, 2009, Student's House in L'Aquila (Italy) was devastated by a strong earthquake and 8 young students died. On January 18th 2017, 2017 a huge avalanche hit Hotel Rigopiano in Farindola, close to Pescara (Italy), causing the death of 29 people present in the structure for winter vacation. Those tragedies taught us that about the importance of an automatic earthquakes events detection system. If the streak of earthquakes events that occurred before these two disasters had been notified to the National Protection Department, maybe the early evacuation procedures would have been started. This could have saved many precious lives.

UC3 aiming to provide an innovative technology for enabling AI in distributed low power IoT field devices. The innovation consists on moving the analytics from the cloud server directly as close as possible to the sensors, "close to the edge", empowering sensors analytics by powerful yet optimized AI algorithms specifically designed for low power embedded micro-controllers.

UC3 is also a horizontal UC because it aims to provide an enabling technology that could be thus used to address a wider range of vertical specific scenarios in which event detection is needed.

The AI Sensing Platform use case is a horizontal use case aiming at providing a disruptive and innovative technology for enabling AI in distributed low power IoT field devices and in particular low power embedded micro-controllers. This platform is implementing a horizontal technology because it brings a complete different approach on conceiving the nature of distributed IoT devices that could be thus used to address a wider range of vertical specific scenarios in which event detection is needed. The innovation consists on moving the analytics from the cloud server directly as close as possible to the sensors, i.e. "close to the edge", empowering sensors analytics by powerful yet optimized AI algorithms specifically designed for online training and unsupervised model learning on 32bits low cost low power micro-controllers. These nodes enable devices communication and coordination on event-driven patterns instead of data-centric driven approach mostly used on nowadays cloud IoT centralized systems.

STMicroelectronics AI Sensing platform adopts the core principles of the edge/pervasive computing paradigm as opposed to cloud-centric approach, where the intelligent processing of sensed data is moved and distributed to the leaves of the system, at field level sensing devices, embedding algorithms based on the highly nonlinear approximation capabilities of artificial neural networks, statistical analysis, distributed computation for increased system scalability, safety and robustness. In particular, the core functionalities of the system are moved at lower levels of the platform and two key aspects are therefore implemented:

1. Local predictive analytics for environmental and inertial data streams: In the AI Sensing use case, localized edge analytics will be applied which will result in unsupervised IoT behavior where only results and events triggered by these analytics will be propagated to the upper level on the infrastructure to the IoT Gateway locally and to enable a seamless deployment on a Vertical Application at network layer.

2. Local AI Sensing behavior and monitoring: The sensing of this environmental and inertial data streams acting locally at sensor level allows to process more data in real-time and to precisely identify relevant events by modeling the characteristics of the acquired data thanks to neural network self-learning algorithms. Security is increased as well because complete raw data are almost rarely propagated to the IoT Gateway, but just those identifying anomalies are transmitted for complex processing. This makes the system highly scalable, robust, and largely autonomous on its local behavior.

A significant and meaningful incremental demonstration of the technology will be implemented and validated during the project as described in the following sections.

Two different scenarios have been identified of a relevance for demonstrating UC3 using SEMIoTICS framework in a lab environment:

Scenario 1 - Local Vs Global anomalies

A series of inertial intelligent IoT Nodes are deployed in the public building under control analyzing the vibrations at different places and floors. Equivalent systems are deployed in nearby buildings. When abnormal vibrations are reported the system analyzes whether similar anomaly is reported by the neighborhood and if observed whether similar events have been reported by area services (e.g. by the high precision seismographs belonging to the INGV[1] - "*Istituto Nazionale di Geofisica e Vulcanologia*" web service). In case of positive feedbacks, the system reports the alert to owners and operators of the building as well as the local authorities while automatically send the command to the elevators control system of the building to reach level zero and stop working preventing people from use them until the alert is released. Authorities may decide after check to issue an evacuation alarm.

Scenario 2 - Causal discovery and inference

Additional outdoor sensors are utilized to track local temperatures and lighting / sun heating condition. This time the validation is done vs the remote DB of the local ARPA[2] ("*Agenzia Regionale Per l'Ambiente*") reporting area expected trend and other useful information such as avalanche or flood risks for the zone. In this case there is no "event" triggered as such but rather a more detailed evaluation of the evolution in the local area given the local condition enforced by the area expected trend. Again, owners and operators of the structure would be alerted when in front of potential risky trends better evaluated by authorities leveraging historical statistics and associated safety procedures would be activated as precautionary measures.

These two scenarios will be demonstrated incrementally in two macro steps: a first stage focused on testing the system in isolation on a controlled environment (i.e. in a simulated laboratory environment) in order to effectively demonstrate the effectiveness of the Local Embedded Analytics (LEA) and edge computing core capabilities at work. After this laboratory validation by ST-I domain experts, a second phase mainly developed during Task 5.6 activities, will be focused on the integration of this platform in the SEMIoTICS framework, enlarging the test environment to dedicated field trials, including proper end-to-end validation from Field Devices (IHES Sensing Units), to IoT gateway up to SEMIoTICS network and backend/cloud infrastructure where the UC3 App will be deployed.

2.2 Challenges and objectives

The system presented in this deliverable, that in essence is the UC3 as it is defined in SEMIoTICS technical annex, is an intelligent end-to-end architecture for detection and validation of critical events that does not require any prior information the environment to be monitored (IHES, Intelligent Heterogeneous Embedded Sensors).

This system is composed of several capable intelligent nodes (IHES Sensing Units) to obtain environmental data, analyze it and detect anomalies through the use of both neural and autoregressive models. These smart nodes are connected to a supervisor (IHES Supervisor) whose purpose is to coordinate them in order to propagate relevant events to other system architecture components. Its main tasks are to aggregate the data and perform a validation of the local detected changes from any of the individual sensor units. Scope of Task 5.6 is thus to provide an end to end UC3 demo scenario within SEMIoTICS ecosystem that will be incrementally

[1] INGV http://cnt.rm.ingv.it/ (link accessed on July 2020)
[2] ARPA https://www.arpalombardia.it (link accessed on July 2020)

consolidated. This effort provides a complete end-to-end SEMIoTICS infrastructure integration at field, network and backend level. Those additional levels of integration, will open up the possibility to aggregate data from multiple local IHES supervisor's analytics cluster, carrying out HTTP queries towards 3rd party open cloud services and, in case of re-detection of any anomaly confirmed event, promptly notify it (for example by email) to the service manager monitoring.

The results of the work are complete both as regards contextualization and the KPIs (Key Performance Indicators) defined in SEMIoTICS, both for correctness, robustness and reliability of the system. The solution presented in this deliverable as a whole, is an excellent technological driver that will enable thanks to the SEMIoTICS open infrastructure the derivation of new use cases able to exploit this intelligent distributed system, making IoT technologies more pervasive and widely adopted.

2.3 SEMIoTICS UC3 sub use cases

Nowadays we live surrounded by many devices connected to the network capable to acquire data on the environment and on those who live the environment. Using this huge amount of data in the correct way, opens the door to new technologies and construction of systems with purposes and purposes unthinkable even just a few years ago. SEMIoTICS and UC3 IoT Generic scenario within it, aims to explore the possibilities offered by diffusion of the Internet of Things (IoT), from the data and resources that this dissemination brings to have, with the will to explore new ways of thinking, designing and building pervasive systems. In particular, the main UC3 scenario objective is to create a horizontal enabling technology aimed at automatic detection and validation local critical events, which does not require any prior knowledge of the environment to monitor. In addition, this technology will be integrated in this task 5.6 into an end-to-end project implemented with the aim of demonstrating the feasibility of this new disruptive Intelligent IoT technology proposal within the SEMIoTICS framework for intelligent IoT and IIoT (Industrial IoT) applications.

On IoT domains there are various approaches to handle such a large data from massively deployed IoT things.

But in essence they could be classified into "Cloud", "Fog" or "Edge" computing depending on where actually these data are processed and aggregated into the IoT ecosystem, e.g. on backend, gateway or IoT device node. A summary picture is shown on **Error! Reference source not found.**. The main issue with "Cloud Computing" [1] data aggregation approach is mainly poor scalability and system resiliency.

Another aspect to consider, especially in their simpler form concerns the data. Very often these IoT cloud based solutions use a lot of sensors that, very often, are unable to perform computations but only to acquire and send raw data. In order to get information from these data, therefore, they must arrive in some centralized unit with the aim to carry out appropriate analyzes depending on the type of data and the purpose of the system. This creates a huge amount of data traffic that needs to be transferred within the system and requires to have a reliable, responsive and high-bandwidth communication. Moreover, data security poses relevant challenges on such scenario. To remedy, or at least mitigate, these problems, there are at least others

two well-known approaches to solve the problem: "Fog Computing" and "Edge Computing".

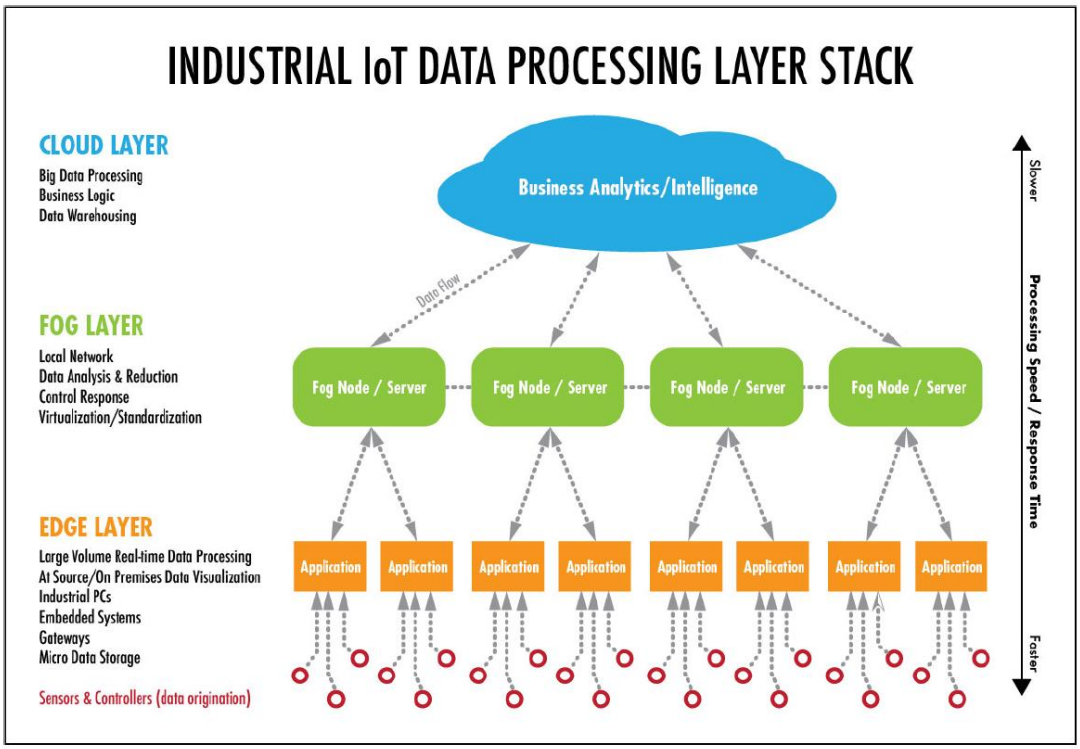

FIGURE 1. DATA ANALYTICS COMPUTING APPROACHES[3]

As can be seen in Figure 1, "Fog Computing" consists in using an intermediate level with computational ability to receive the raw data of the that derive from the sensors and, even before sending them to the cloud, make part of the computing in order to reduce the amount of data exchanged and sent to the cloud.

Finally, the paradigm opposite to "Cloud Computing", is the so-called "Edge Computing": in this case the computation takes place directly at the lowest level (on the IoT sensing device), reducing considerably the band of data transferred to the highest levels of architecture. Obviously, even this latter solution is not without any problem or cons: by carrying out the computing closer to where the data is collected greatly reduces the problems related to bandwidth and security of raw data (which is no longer necessary to transmit at other architectural levels), but there are severe hardware limitations. In fact, the more the computation moves down, the more difficult it will be to implement complex and intelligent algorithms.

In UC3 scenario we tried to provide to map on SEMIoTICS architecture the “edge computing” approach by tailoring specific scenarios in order to demonstrate its feasibility.

From the main storyline presented in section 2.1 we derived three sub use cases to incrementally demonstrate the capabilities of the Generic IoT System within SEMIoTICS technology, and these latter ones will be analyzed in sections 3-5 that follow.

This deliverable describes the specific UC3 SEMIoTICS infrastructure adopted in order to bring “Edge computing” approach into reality and the actual mapping of the SEMIoTICS components vs the HW platform testbed developed in task 5.6 is presented in Figure 2. Moreover in Figure 3, the SEMIoTICS components considered in the scope of the UC3 – Generic IoT scenario - are represented, with new features specifically developed, existing technologies re-adapted, and existing IoT technologies.

[3] Courtesy from Winsystems

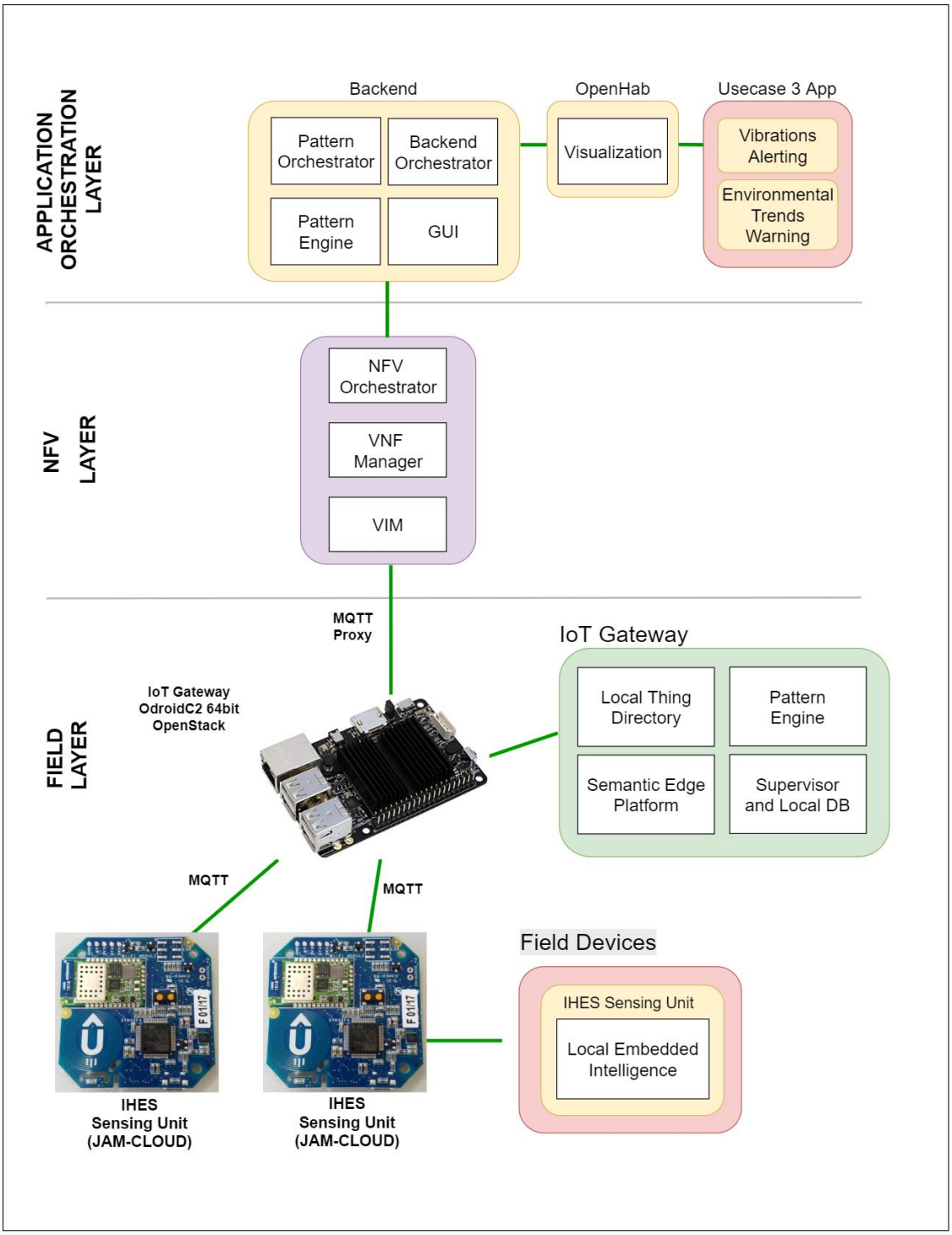

FIGURE 2. UC3 SYSTEM VS COMPONENTS MAPPING

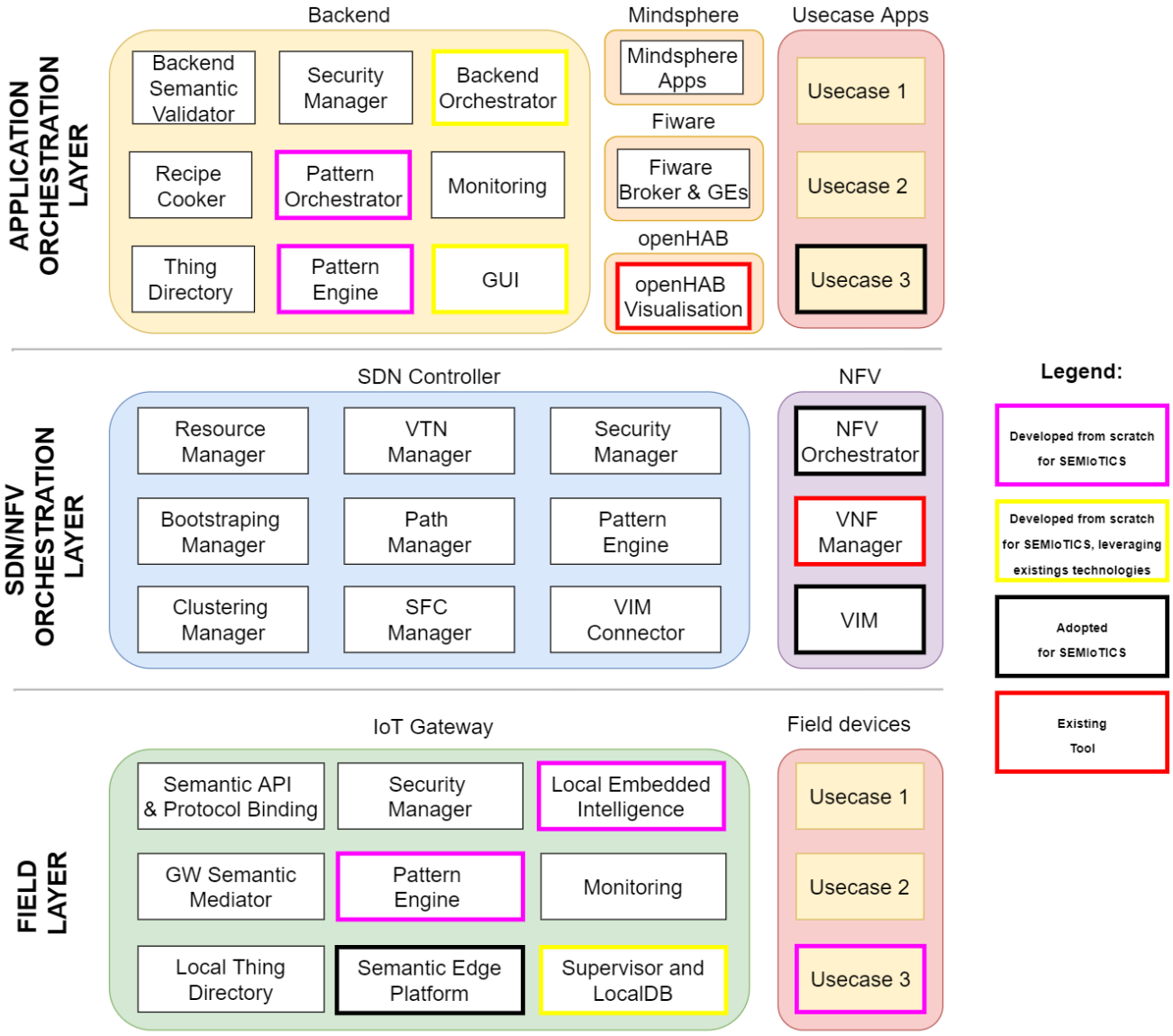

FIGURE 3. UC3 SEMIOTICS COMPONENTS STATIC MAPPING

As stated in general introduction we will consolidate the complete demonstrator along task 5.6 activities by following an incremental approach from bottom to top, and we associate intuitive, self-contained storylines and sub-use cases to showcase the incremental functionalities available.
We identify three incremental sub use cases (from sub use case 1 to sub use case 3) that will incrementally integrate the components showed on **Error! Reference source not found.** on cycle 1 and cycle 2 activities.
Sub use case 1 is devoted to show specific local analytics deployed at device level to enable smart autonomous learning devices following the “Edge Computing” approach. Sub use case 2 will be focused on how to bring the results of the data analytics on the upper level of the infrastructure, and the scalable infrastructure needed for it in order to scale / open-up/scale to backend services. Finally sub use case 3 will deploy end-to-end specific UC3 SPDI patterns exploiting SEMIoTICS ecosystem, to allow a full integration into SEMIoTICS architecture.

3 TESTBED SETUP, INTEGRATION AND VALIDATION

UC3 is validated in a SEMIoTICS testbed environment, being composed of a Backend layer, an NFV Infrastructure layer, and a Field layer. This is the 1st validation testbed and infrastructure, which was integrated during this cycle 1 period, focusing on the first sub-use case which is detailed in Section 4, where UC3 applications that implement smart monitoring and analytics functions, are managed autonomously by the SEMIoTICS framework. Thus, functionalities such as establishing connectivity, negotiating transport protocols and networking paths, as well as service scale-out and load balancing functions will be totally transparent for IoT applications. Moreover, they are handled by the respective frameworks of the SEMIoTICS infrastructure under the control of the Pattern Orchestrator. Now, as we are entering on cycle 2 activities period, we intend to extend the same approach in the sub-use cases 2 and 3 by incrementally adding all the missing features, and integrating all components under the common UC3 testbed. So, we are moving towards a complete end-to-end demonstrator moving the integration from the field layer to the network and finally the backend where the GUI visualization and the global data aggregation and 3rd party services queries will be implemented. The UC3 testbed will be incrementally enhanced by providing the infrastructure needed to support the validation and inclusion of all the yet missing components needed by the full storyline and expected for the final demo. More specifically, the UC3 testbed is currently composed of:

- One 6-core 64-bit server with 32 GB RAM hosts the OpenStack Controller and Network services, related to Management, Orchestration and SDN control.
- One 6-core 64-bit server with 32 GB RAM acts as the Compute Node, or Cloud hypervisor, that hosts all IIoT services and VNFs in dedicated Virtual Machines (VMs).
- A virtualized IoT gateway based on a 64-bit ARMv8 Single Board Computer (i.e., an Odroid C2) which is capable of hosting VNFs, acting as a VIM hypervisor.
- Field layer devices from ST.

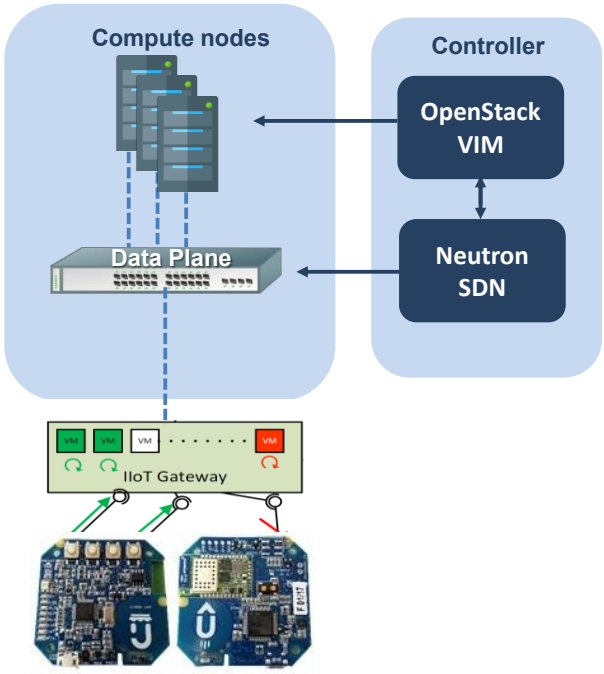

FIGURE 4. PHYSICAL INFRASTRUCTURE OF UC3 TESTBED

3.1 Field layer

The UC3 Field Layer testbed includes a virtualized IIoT gateway that interconnects a set of sensors and actuators with the backend cloud. The IoT gateway supports KVM virtualization, enabling us to push VNFs down to the gateway tier. This allows services with ultra-low latency requirements to be pushed in very close proximity to the Field devices, hence minimizing latency. The relatively modest resources available at the gateway, which is implemented with a Odroid 64-bit ARM-based Single-Board Computer (SBC), means that it must be used for a minimum number of VNFs with low processing needs. Crucially, the IoT gateway hosts the IHES supervisor service, which is implemented as a .Net console application and thus requires the mono or .Net Core runtime frameworks to operate. Since both runtimes require hundreds of megabytes of storage space, the mkbundle tool was leveraged, which is a cross-compiler tool which produces a native Linux executable an initial .Net assembly, packaging any .NET dependencies and any additional assemblies that the application requires. Moreover, since the original application supported the Raspberry Pi 3 environment, the cross-compilation capabilities of the mkbundle were leveraged:

First, the cross-compiler tools of the 64-bit target platform (i.e., the Odroid C2) were fetched:

```
$ mkbundle --fetch-target mono-5.18.0-ubuntu-16.04-arm64
```

And finally, the native Odroid C2 executable was simply cross-compiled with the following command, and deployed within the VNF-1:

```
$ mkbundle -o slim_coordinator --cross mono-5.18.0-ubuntu-16.04-arm64 --deps
slim_coordinator.exe --machine-config /usr/etc/mono/4.5/machine.config
```

FIGURE 5. VIRTUALIZED IOT GATEWAY

The UC3 LEA component mapped directly into STM32 MCU has been integrated into the UC3 testbed as it was designed released in task 4.3 with minimal adaptions to it for the specific MCU mapped components. Please refer to section 4.3.1 for an overview about it.

3.2 NFV Infrastructure (NFVI)

The UC3 NFVI includes the set of NFV controllers and managers (i.e., VIM, Neutron SDN, ETSI OSM), and the set of hardware used for virtualizing network functions (also referred to as compute nodes). Together, SDN and NFV are able to realize customizable isolated network environments (termed Virtual Tenant Networks, or VTNs), where processing endpoints (i.e. VNFs) are dynamically instantiated at appropriate Compute Nodes. Network traffic is then directed towards such VNFs, which may be standalone or part of a custom Service Function Chain (SFC) to reach a desired endpoint, be processed or consumed. The UC3 IIoT applications are implemented as a collection of 3 such VNFs:

- VNF-1 includes the Supervisor and Local DB functions and is deployed at the IoT gateway, simplifying deployment and ensuring minimal latency of the aforementioned functions
- VNF-2 includes the OpenHAB Visualization service, and is leveraged to display sensor data, as well as global and local status changes in real time and generate charts.
- VNF-3 implements global data aggregation and storage at a global time series database, allowing post-processing and analysis of sensor data.

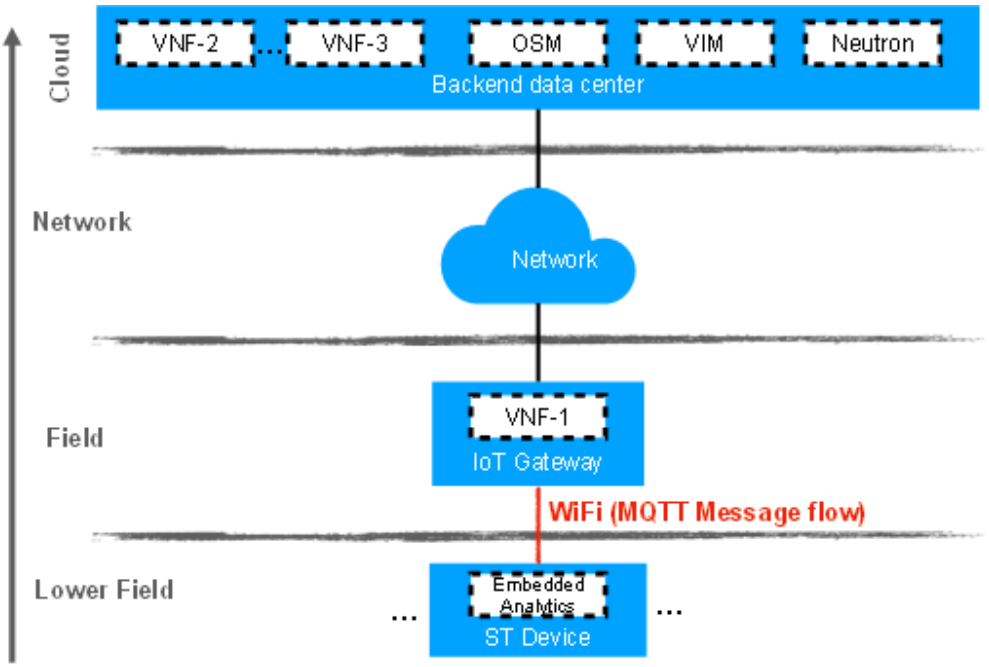

FIGURE 6. UC3 NFVI INCLUDING THE VNF1, VNF2 AND VNF3 AND NFV MANO

The whole lifecycle of the VNFs is managed by the NFV MANO framework which is composed of the NFVO and VNFM modules. More specifically, note that the user has to interact with the OSM to manage the whole lifecycle of the VNF. Moreover, the OSM interacts internally with the OpenStack, which manages the deployment of the VNFs on top of the NFVI, encapsulated within VMs. More insights on NFV for the SEMIoTICS purposes are given in the SEMIoTICS' deliverable D3.8 [2]. Next, we list the configuration files required for VNF deployment and we explain their role:

- **VNF descriptor** (VNFd). This is the configuration file that describes the VNF, which has a yaml format. It describes the features of the VM that will host the VNF, the links that the VNF exposes externally to connect to e.g. NS and other features that we will see below.
- **Network Service descriptor** (NSd). VNFs are part of an overall network service (NS). That is to say a NS contains at least one VNF. Moreover, the NS is described by a configuration file in yaml format that is so-called NSd. It describes among other features the constituent VNFs of the NS or the links between the NS and the VNF. The tag "constituent-vnfd" indicates which VNFs are part of the NS and the "vld" tag defines the virtual links used by the NS for to interconnect VNFs
- **Cloud init file**. This is a configuration file that will be used by the VM that will host the VNF. It specifies the initial behavior that the VM needs to provide. For instance, installation of software packages, execution of software applications, among other features.

Next, we present an example snapshot of the yaml file that we have used to define the VNFd of VNF3 (similar yaml files are prepared for VNF-1 and VNF-2). First, note that the “connection-point” tag indicates the external connection points of the VNF. Its value corresponds with the one assigned above in the NSd yaml file, within the “vnf-connection-point-ref” tag of the “vld” list. The tag “id” is the unique identifier for the VNF and it is important to recall it, as it is used in the NSd. The tag “mgmt-interface” is the interface over which the VNF is managed. Moreover, the “cp” within it just specifies the type of management endpoint, in our case “cp” means that we will use a connection point. Another important tag is the “vdu”, which stands for virtual description unit, and it specifies the features of the VM that will host the VNF. Thereby, the “cloud-init-file” indicates the cloud init file that will be used by the VM. The snapshot for this cloud init file is actually described below. The tag “image” indicates the image that will be used to create the VM, in our case we will have an Ubuntu OS. The “interface” tag within the “vdu” tag specifies the interfaces for the vdu. Note that we define an external connection point that corresponds with the connection point defined for the vnfd. Last, but not least, the “vm-flavor” indicates the computing, memory and storage features of the VM that will host the VNF. Thereby, note that we define a VM with 1 virtual CPU, 4 GB of RAM and 5 GB of storage.

```
vnfd:vnfd-catalog:
    vnfd:
    -   connection-point:
        -   name: vnf-cp0
            type: VPORT
        description: Generated by OSM package generator
        id: vnf3_vnfd
        mgmt-interface:
            cp: vnf-cp0
        name: vnf3_vnfd
        short-name: vnf3_vnfd
        vdu:
        -   cloud-init-file: cloud_init_vnf3
            count: 1
            description: vnf3_vnfd-VM
            id: vnf3_vnfd-VM
            image: ubuntu18-minimal
            interface:
            -   external-connection-point-ref: vnf-cp0
                name: eth0
                type: EXTERNAL
                virtual-interface:
                    type: VIRTIO
            name: vnf3_vnfd-VM
            vm-flavor:
                memory-mb: 4096
                storage-gb: 5
                vcpu-count: 1
        vendor: OSM
        version: '1.0'
```

FIGURE 7. VNFD TO CONFIGURE AND TO DEPLOY THE VNF RELATED TO VNF3

3.3 UC3 Networking

In UC3 a Virtual Tenant Network (VTN) is deployed to interconnect VNFs1-3, leveraging OpenStack Neutron Networking, which is hosted within the main Controller node. OpenStack Neutron is an SDN controller which is part of the OpenStack networking project and provides the virtual networking resources expected in the SEMIoTICS NFVI, offering control over L2/L3 networking parameters, security policies, resource management

and QoS over a simple REST API. Moreover, VTN offers Layer 2 isolation of virtual network participants, compatible with the Network Slicing (NS) concept. In UC3 VTNs are implemented with VXLAN, supporting unlimited network overlays on top of the provider network which provides external access to UC3 VNFs. The UC3 VTN is bridged via an Open vSwitch bridge with the Wi-Fi network, allowing sensor nodes to directly interact with the VNFs, through an MQTT based message bus. Moreover, DHCP and Firewall functionality is offered by Neutron.

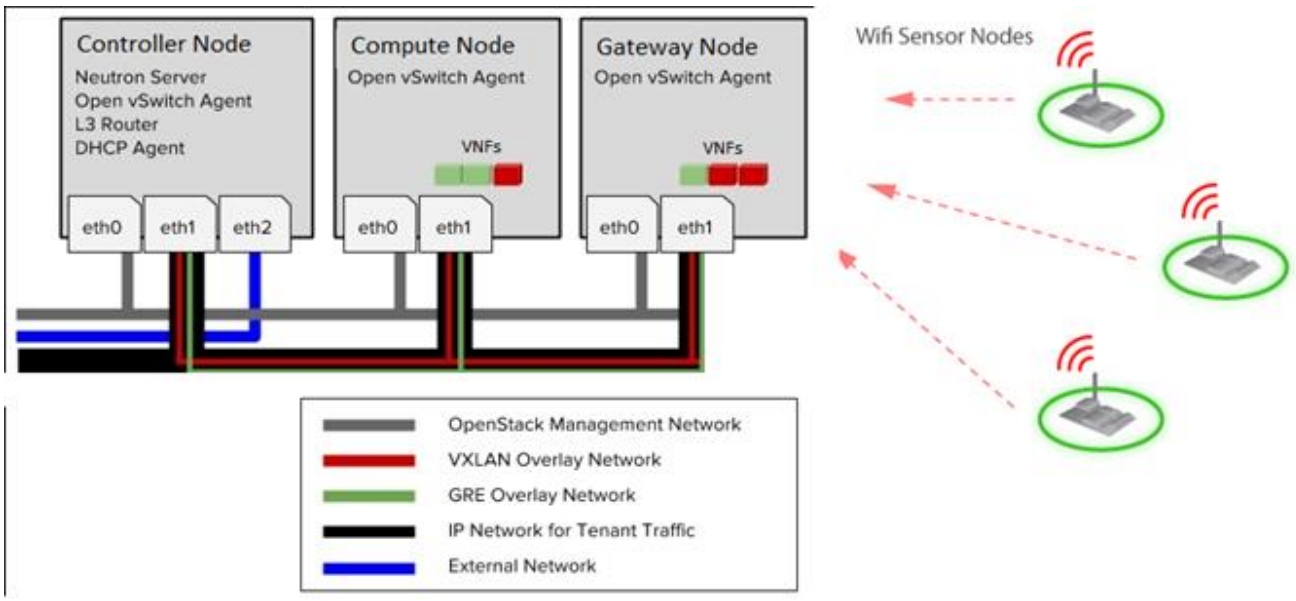

FIGURE 8. UC3 NETWORKING VIA OPENSTACK NEUTRON

4 SUB USE CASE 1: LOCAL VS GLOBAL ANOMALIES DETECTION DEMO

4.1 Story line

This sub use case is in line with the UC3's scenario 1, "Local Vs Global anomalies", described in section 2. Thereby, we are going to provide the (almost) complete field level UC3 deployment to enable the final SEMIoTICS framework integrated UC3 demonstrator. For instance, as stated in the scenario 1 of UC3, a key capability of the UC3 distributed intelligence system is the ability to self-learn from the environment any nominal state from a given input sensor and set-up autonomously a threshold-free analytics in order to detect any anomaly from this acquired nominal state. On top of this it is possible, by using several nodes deployed in the environment, to estimate the relations between the sensing device in order to understand at a higher level of the architecture, if a particular anomaly is of a single node (local change) or has been observed by many (partial change) if not all nodes (global change).

The cornerstone of this sub use case 1 is to provide an integrated infrastructure at field level (devices + gateway) in order to realize these capabilities. The scenario involves two key FL SEMIoTICS components, the instantiation of the MQTT infrastructure needed for message exchange, and the NFV virtualization infrastructure for a reliable, affordable mapping of the infrastructure.

4.2 Scope and objectives

The main objectives of this sub use case are summarized as follows:

- Integrate into the testbed discussed in section 3.1 the three UC3 Local Embedded Analytics (LEA) related components at field level, as it has been designed and developed from AI self-learning / statistical algorithms characterized in task 4.3 and described in details on deliverable D4.10 [3]
- Integrate and deploy UC3 MQTT infrastructure at field level as it has been designed and discussed in task 2.4, D2.5 [4]
- Validate the integrated functionalities verifying that the VNF UC3 testbed works as expected compared to the ST initial testbed
- Visualize real-time sensor values, as well as global and local changes events though OpenHAB
- Cross-compile the IHES Supervisor service components for the Odroid C2 virtualized gateway embedded board, and deploy within VNF-1, as in ST testbed it was validated on a Raspberry Pi3 board.
- Integrate the IHES local DB service exploiting InfluxDB for local data and events storage
- Set the requirements and derive the functional blocks in order to support functionalities described in sub use case scenarios 2 and 3 by designing all components interoperability

4.3 Interaction with SEMIoTICS framework and components

In this the section we detail SEMIoTICS the set of components leveraged by UC3 or that could interoperate with UC3 infrastructure within the SEMIoTICS framework architecture, and explain their interactions during the UC3 sub use case 1 storyline presenting sequence diagrams for common procedures and APIs and their relation to other SEMIoTICS components are overviewed.

Field level SEMIoTICS components needed for UC3 scenario:

- Local Embedded Analytics (see section 4.3.1).
- Supervisor and Local DB (see sections 4.3.2 and 4.3.3).
- The OpenHAB platform(see section 4.3.4).

Field level SEMIoTICS components interoperable with UC3 designed sub-system (please refers do related deliverables in WP4 for details about these two SEMIoTICS components):

- Semantic Edge Platform: interoperability with this component is possible by exploiting common underlying Node-RED infrastructure. This allows the possibility to have an open flexible ecosystem able to easily map and sustain new use case scenarios.
- Local Thing Directory: on UC3 scenario the coordination and management of the nodes are ensured by the interaction between the LEA component on the MCU and the IHES supervisor service. Anyhow, since the IHES sensing devices relies on standard MQTT + JSON communication it is possible to export them as a WoT device node in order to be discoverable as sensing units by other systems / components not part of UC3 architecture. This ensure the interoperability and facilitate integration at gateway level with virtually any 3rd party services. A TD (Thing Descriptor) exposing underlying MQTT protocol of the UC3 sensing devices has been documented and released as reference on D4.4

The sub use case 1 scenario basically implements two flow diagrams that has been declared in D2.5 as part of the UC3 ecosystem. The one presented in Figure 9 represents the communication flow and processing between the IHES sensing nodes and the IHES supervisor, where a generic raw data signal is locally processed to detect anomalies, stored in a local DB together with events generated by the LEA component embedded on the MCU device. Finally, on bottom left part the correlation between gathered data is computed so to enable at IoT gateway the possibility to discriminate among local or global anomalies.

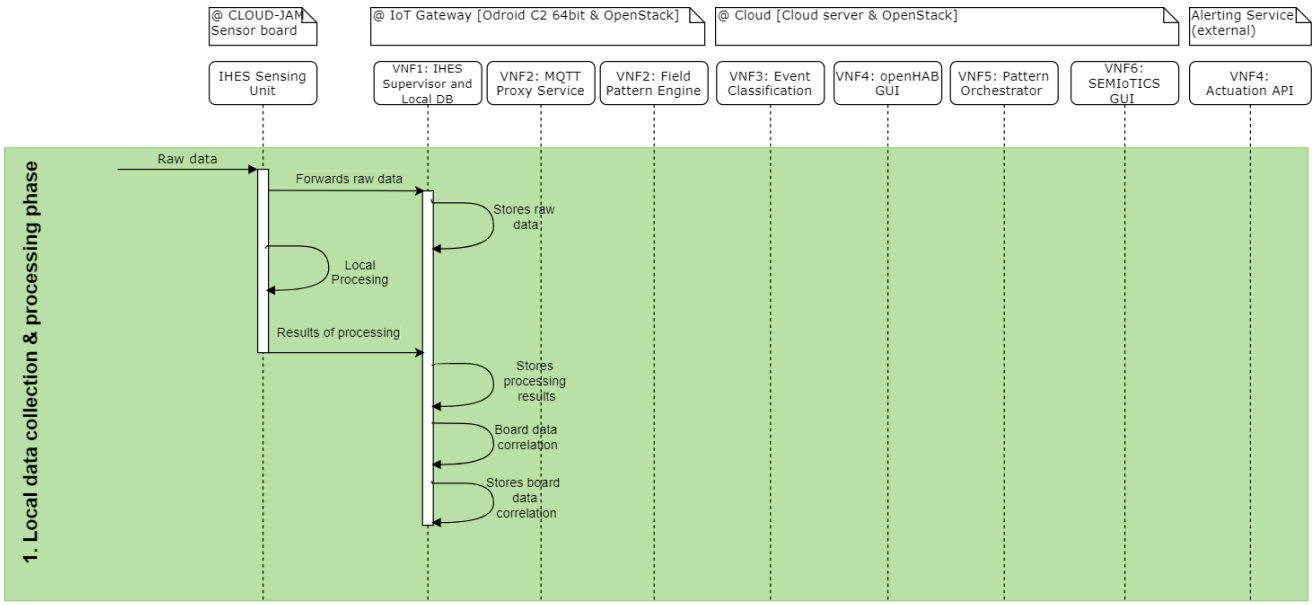

FIGURE 9. UC3 FL MESSAGE FLOW: SENSING UNITS TO IHES SUPERVISOR

On Figure 10 instead, it is reported part of the flow that we implement as part of cycle 1 activities where the results (i.e. events) reported by the system at FL are propagated to the upper level of the UC3 infrastructure exploiting the MQTT communication and the VNF virtualization infrastructure. As today, we realized only part of the diagram flow (the left part on Figure 10, involving mainly the interaction between the sensing board and the gateway declared in this sub use case 1). On cycle 2 integration the full diagram will be integrated in order to realize the complete flow involving as well the SEMIoTICS infrastructure at network and cloud level. Please refer to section 5 (sub use case 2) for a preliminary discussion about that specific part of the UC3 demonstrator.

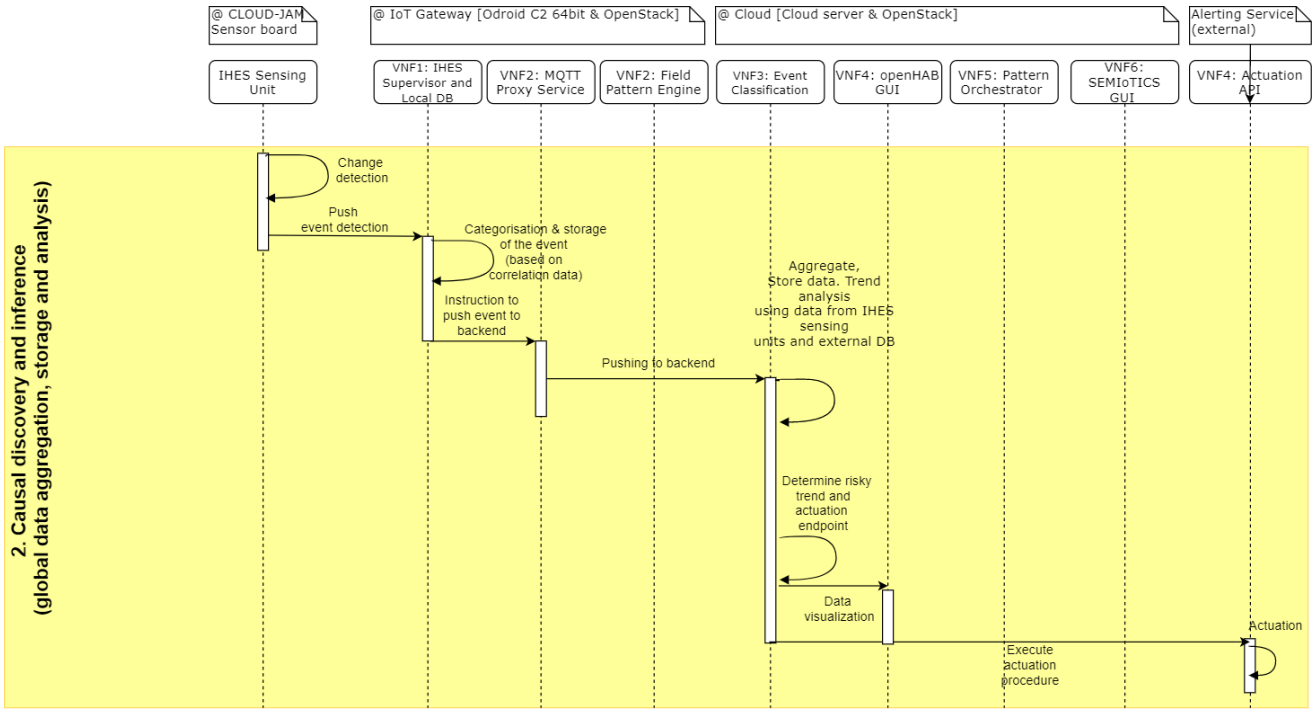

FIGURE 10. LOCAL VS GLOBAL SCENARIO, FIELD LAYER MESSAGE FLOW

4.3.1 IHES LOCAL EMBEDDED ANALYTICS (ON MCU)

This component, developed within task 4.3 has been integrated on UC3 testbed as a dedicated firmware on a microcontroller board. It includes all the analytics local algorithms (tiny AI for signal modelling prediction and online training, statistical algorithms like the CDT and CPM for anomalies detection and validation), and finally the MQTT + JSON client needed to interface the other components in the scenario.

The LEA component interacts with the SEMIoTICS framework thanks to a specifically designed MQTT JSON protocol that delivers to other components in the infrastructure the anomalies detected in the form of relevant events.

In more detail, each IHES sensing unit is composed by an ST X-NUCLEOF401RE Board equipped with an ARM Cortex M4 80Mhz MCU, an X-NUCLEO-IDW01M1 Wi-Fi adapter and an X-NUCLEO IKS01A2 environmental + inertial sensors expansion board, all stacked together on a single PCB board named "CLOUD-JAM" (see Figure 11). Each MCU is programmed with a dedicated FW stack implementing the UC3 designed analytics algorithms discussed in D4.3 and D4.10 together with the communication stack (Wi-Fi + MQTT client) based on STMicroelectronics legacy middleware SW stack.

These devices were integrated into the described sub use case 1 IoT gateway testbed described above.

A detailed overview and characterization of them has been provided as part of final D4.10 [3]. Please refer to this deliverable for any detail about it.

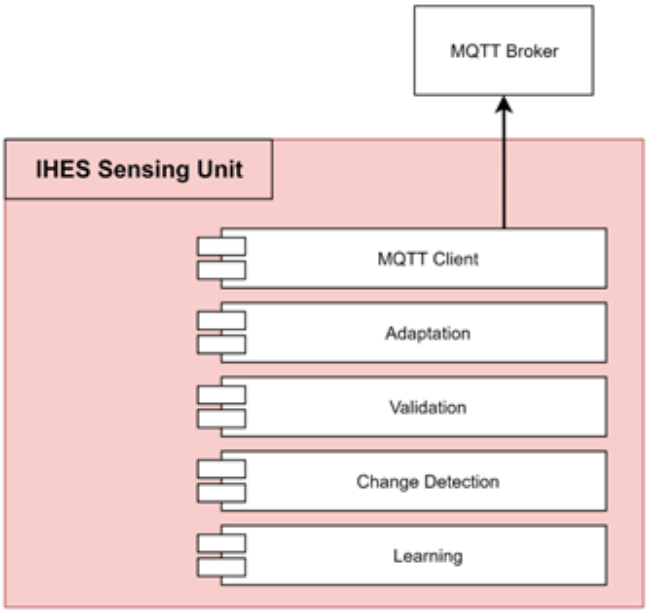

FIGURE 11. IHES SENSING NODE SW ARCHITECTURE: ANALYTICS MQTT JSON FLOW

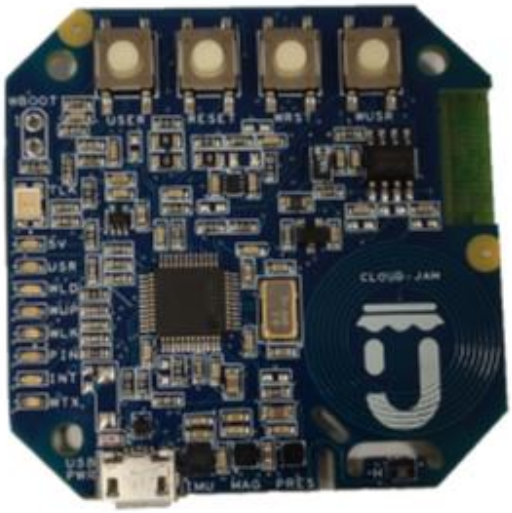

FIGURE 12. IHES LEA ON CLOUD-JAM STM32 BOARD

4.3.2 IHES SUPERVISOR SERVICE

The IHES supervisor service is a specifically designed SW service mapped on UC3 IoT gateway as part of the virtualized VNF-1 instantiation. It is a key-components at gateway level to interface the IHES Sensing unit and allows interoperability with the SEMIoTICS infrastructure.

It is mainly divided into following software modules briefly described:

MQTT CLIENT: the purpose of this module is to allow communication through the MQTT protocol and, therefore, to exchange messages with the IHES Sensing Unit connected to the IoT gateway. In addition, the algorithms are contained here for parsing JSON messages sent by nodes, together with APIs for writing to the local DB.

- CORE MODULE: all the functions are contained in this main software module. They include the complete management of IHES Sensing Unit connected to the system, the implementation of the adaptation strategies. Also these functions define all the parameters that allow connection to the local MQTT broker.

- DEPENDENCY GRAPH: the purpose of this last module is to compute the dependency graph. It is a graph G = {V, E} with V that represents the set of graph nodes (the datastream connected to the system) and the set of arches (relationships between datastream). We will say, therefore, that two data flows X and Y are related to each other if and only if there is an arc connecting them. An arc from datastream X to datastream Y creates a direct relationship when X and Y are datastream associated with sensors of the same type. If instead, these data streams are associated with different types of sensors, the report says live. The IHES supervisor service contains all the algorithms for the calculation of the correlation matrix between different data series and allows to obtain the dependency graph from the latter.
  The purpose of the Dependency Graph is to autonomously estimate which (and how) the datastream they are related to each other in order to implement self-adaptation strategies of the more or less intelligent system using this information.

- NODE-RED dataflow frontend: this module is responsible to dynamically manages the MQTT dataflow connections between connecting IHES nodes and the IHES Local DB component in order to ensure data connection properties and safe data storage for later monitoring and data / events analysis. The MQTT data flows are organized in a hierarchy of topics depicted in Figure 13 designed to manage node scalability and seamlessly connection / disconnection to the system.
  Also on it is reported an example of the Node-RED dataflow instantiated to store into the local DB the message MQTT payloads sent by a node device.

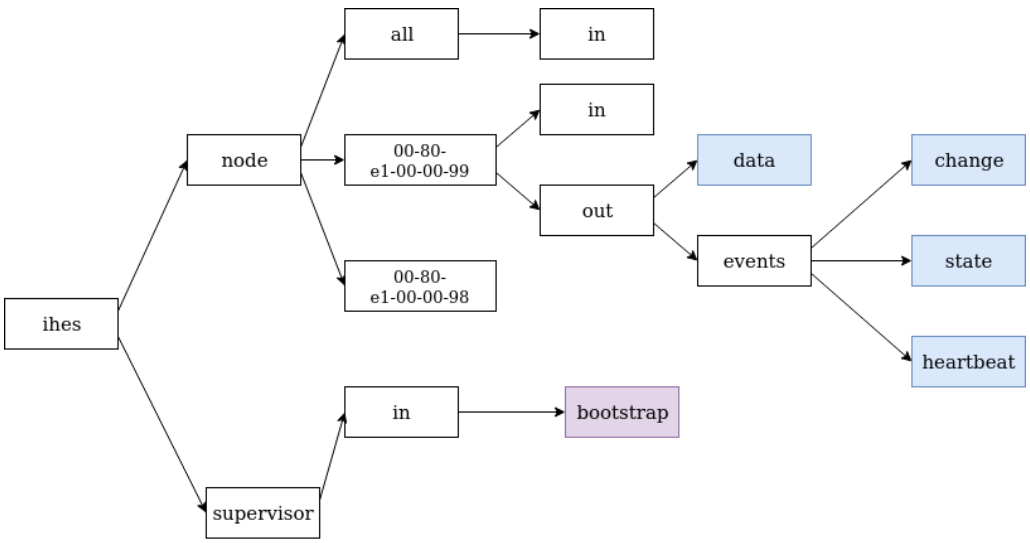

FIGURE 13. UC3 PROTOCOL MQTT TOPICS HIERARCHY

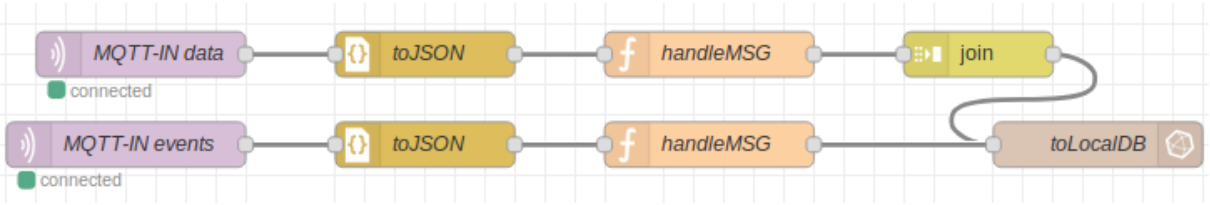

FIGURE 14. NODE-RED MQTT DATAFLOW

4.3.3 IHES LOCAL DB

The IHES local database, is implemented using InfluxDB stack. InfluxDB is a SQL-like time-series database supporting a whole ecosystem for data processing, storage and aggregation. All messages sent on local LAN by the IHES sensing units are stored into this database. InfluxDB offers a complete set of HTTP Rest APIs to query and manage the database. The interaction between the IHES local DB component, the IHES supervisor and other components in case (e.g. the Sematic Edge Platform), is guaranteed thanks to the adoption of the Node-RED infrastructure as the enabling technology for real-time data flows instantiations.

4.3.4 OPENHAB PLATFORM

The OpenHAB external platform is a flexible, open-source, technology-agnostic automation platform that is able to integrate a multitude of devices and systems. To achieve this, OpenHAB segments and compartmentalizes certain functions and operations. OpenHAB uses Apache Karaf to create an Open Services Gateway initiative (OSGi) runtime environment. Jetty is used as the HTTP server, which implements the Dashboard and Management GUI and also hosts the OpenHAB REST API. In UC3, the OpenHAB HabPanel GUI is leveraged for visualization purposes, to display in real-time field sensor values, as well as global and local changes.

4.4 Validation

The validation activities related to the development and later deploying of the two components supporting sub use case 1 story line (i.e. the IHES Sensing Device and IHES Supervisor service) has been done as part of both WP4 and WP5 activities, mainly on task 4.3 and task 5.6.

We could ideally identify two separate phases of the whole validation step of the components: a 1st initial development / tuning phase of the analytics algorithms and their efficient partitioning and mapping on the different part of the system, carried out in ST labs by setting-up a dedicated simplified testbed not encompassing the whole SEMIoTICS infrastructure. This part of the validation has been done during task 4.3 by mainly ST while planning the integration infrastructure needed for task 5.6 with the other UC3 partners. A second phase validation, aiming at providing a reliable, interoperable UC3 testbed integrated in SEMIoTICS has been done as part of the task 5.6 activities thanks to the cooperation of all partners involved in the UC3. The focus of this validation was the deployment of a SEMIoTICS integrated testbed fully functional to incrementally support the three main sub use cases along the whole task 5.6 activities. During this validation great care has been devoted in ensuring that all the technologies characterizing the UC3 scenario, validated in the initial ST testbed, will be integrated into existing SEMIoTICS ecosystem in order to enable all requirements and needs related to the distributed edge computing approach.

As a result of all these activities we achieved an almost complete FL UC3 deployment at device and gateway level, together with a preliminary vertical integration with the network and backend components and the OpenHAB IoT platform that will be used in cycle 2 to deliver a monitoring GUI of the system. A preliminary version of the GUI is already available and a screenshot is reported in Figure 15**Error! Reference source not found.**. It shows the GUI panel showing the live data acquired form six connected IHES sensor devices (three

inertial sensors for vibration plus three more for environmental monitoring) with a temporal plot of the raw data acquired and the anomalies reported by the connected node. A bar chart located in the bottom right part of the panel is instrumental to this and the color of the bar identify the type of change. In blue color are reported the local changes and in red color the global ones.

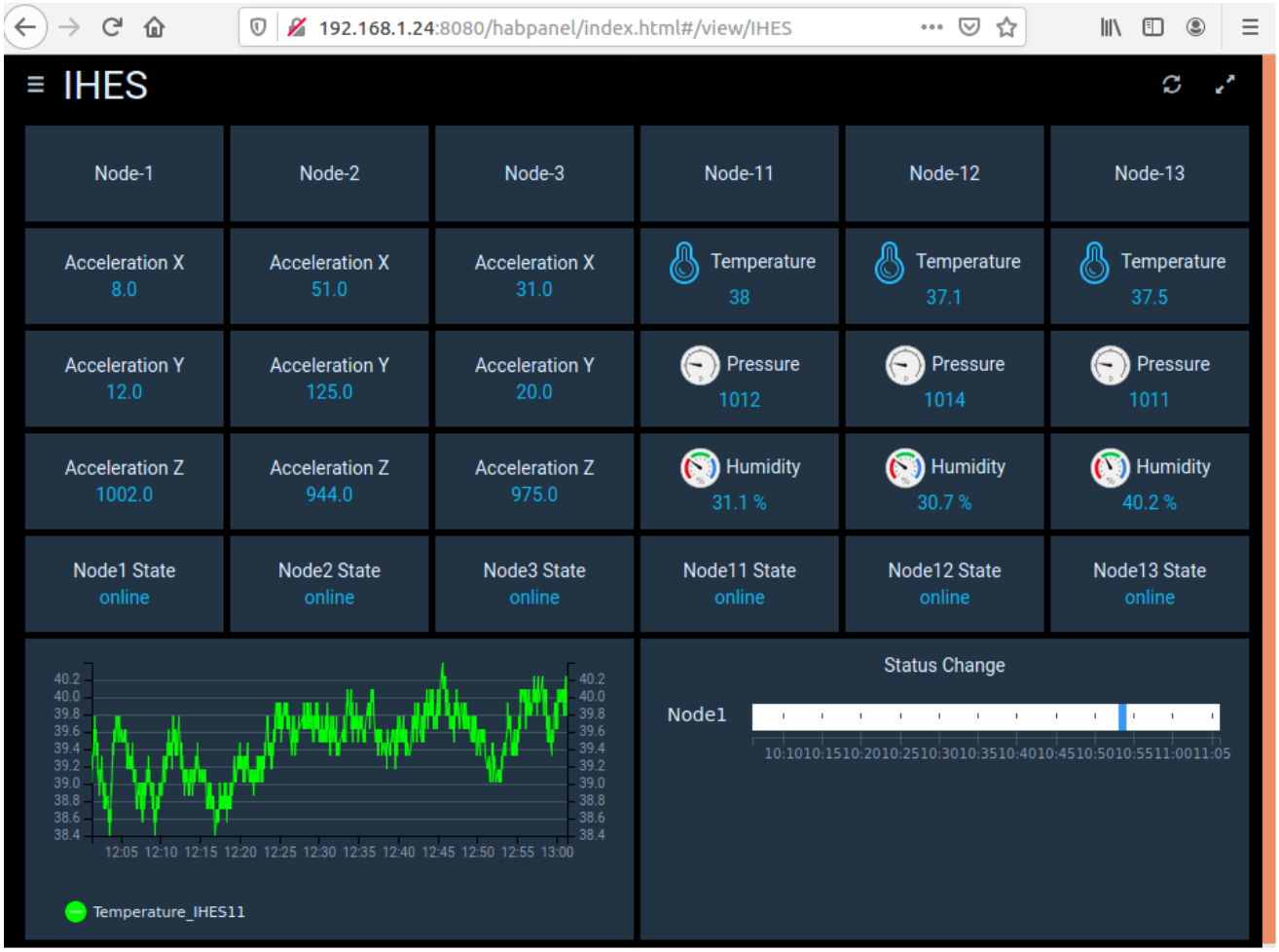

FIGURE 15. UC3 OPENHAB GUI MOC

4.5 Related KPIs

In reported Table 1 has been summarized all the related KPIs involving the UC3 sub scenario 1 – field level deployment of LEA component. In following sub sections, a clear overview of each KPI has been reported with the relevant measures and results obtained, as well as the methodology used to asses them.

TABLE 1. SUB USE CASE 1 KPIS

KPI	Description	Related Task	Goal
KPI-4.1	Delivery of lightweight ML algorithms	4.3, 5.1	To demonstrate Edge computing approach thanks to Local Embedded Analytics UCs components within SEMIoTICS framework

KPI-4.4	Detection time of less than 10 ms	4.3, 5.1	Enable real-time Local Embedded Analytics processing on Field Device level
	Delivery of repeatable Change Detector	NEW	Detector trained in similar condition should provide similar thresholds
KPI-4.5	Comparison false positive rate of auto regressive (AR) models vs neural network models for a defined set of changes	4.3, 5.1	Improvement of at least 20% in false positive rate – ESN vs AR[4]
	Complexity of back propagation on feed forward autoencoder neural network[5] vs ST-I proprietary adaptation mode[6] based on NN		Improvement of at least 20% in minimum adaptation time – ESN vs Autoencoder

4.5.1 KPI-4.1 - DELIVERY OF LIGHTWEIGHT ML ALGORITHMS

LOCAL EMBEDDED ANALYTICS COMPONENT

As part of Task 4.3 efforts, the UC3 Local Embedded Component wrapping all UC3 Generic IoT needed functionalities. This component has been deployed in task 5.6 as a dedicated MCU device firmware including these new algorithms:

- An online training for neural model (ESN).
- An online predictor using (online) trained neural model (ESN).
- A model-free change detection (CDT[7]) test applied to residual signal.
- A change-point method to (CPM[8]) validate the detected change

OUTCOMES

In SEMIoTICS UC3 scenario edge computing could be enabled thanks to the LEA components leveraging SEMIoTICS framework.

4.5.2 KPI-4.4 – DETECTION TIME LESS THAN 10MS

TEST CONDITIONS

Detection Time is defined as the time in which LEA component in MCU firmware performs all the needed phases to identify from the input data series if a potential change in the signal could be reported as a (local) change or not.

For this experiment the Detection Time was profiled on the ST X-NUCLEO F401RE board using developed analytics firmware encompassing the ESN model developed in task 4.3.

RESULTS

[4] Both ESN (Echo State Network) and AR are models developed by STMicroelectronics specifically for UC3
[5] Autoencoder reference model has been developed by ST specifically for UC3 as initial reference for self-learning design
[6] Use case specific adaptation modes have been developed by ST-I as part of UC3 T4.3 activities
[7] CDT models are developed by ST specifically for UC3 scenarios
[8] CPM models are developed by ST specifically for UC3 scenarios

The measured Detection Time is on average around 8/9 ms for each processed sample from the sequence (acquisition data rate was set to 5Hz, thus on average less than 50ms on a second are consumed by this stage). Thus the LEA component is deployed in ST MCU FW enables real time processing at Field Device level.

4.5.3 KPI 4.4 – DELIVERY OF REPEATABLE CHANGE DETECTOR

TEST CONDITIONS

The Change Detector repeatability is defined as the ability of the CDT algorithm to compute same thresholds during the CDT thresholds estimations when the same predictive models are trained in the same conditions. In this experiment the goal is to achieve a Standard Deviation / Mean below 10%. This means that, on average, thresholds should vary less than 10% if the same models are trained in the same conditions.

For this experiment the thresholds has been computed using the UC3 LEA component mapped on X-NUCLEO F401RE MCU Board, retrieving data from the 3-axes accelerometer stacked on the MCU board leaning on a desk.

RESULTS

In table below, the experimental results are reported.

Numbers in the table refers to mean values of 20 different trainings done using a 3-axis accelerometer setup.

TABLE 2. KPI 4.4 RESULTS REPEATABLE CDT

	X	Y	Z
CDT Threshold (Mean)	52.65	43.32	144.26
CDT Threshold (Std. Dev)	4.91	3.6	17.6
(St. Dev / Mean) %	9.33%	8.3%	12.27%

The mean percentage over the three accelerometer axes is **9.96%.**

CONCLUSIONS

The CDT deployed algorithm on the UC3 Local Embedded Intelligence component is able to estimate equivalent thresholds from same model under same conditions.

4.5.4 KPI 4.5 – DETECTED CHANGES FALSE POSITIVE RATE COMPARISON (AR MODELS VS ESN MODELS)

CONSIDERED MODELS

In the experiment ST-I compare the following models:

- Autoregressive model:
  - order: 5
  - regression: least squares
- (ESN) Neural model:
  - model class: Echo State Network
  - reservoir: 35 neurons

DATASET

A dedicated dataset composed of several sequences from accelerometer data captured at 5Hz for 70 seconds. We consider the time instant T* a change if in that time instant we add an exogenous signal to the original signal. The exogenous signal has been defined as:

- fault: adding random noise *ns*, with $ns \in U(0, 70)$ (uniform distribution)
- incremental: adding ramp noise with a coefficient $\beta = 4 \cdot (r - 2)$ with $r \in U(0, 4)$
- stuck-at: signal became value c, with c equal to the last input value before the change

THRESHOLD based CDT

A threshold-based CDT (Change Detection Test) has been developed to find the changes in the acquired data signals. The threshold-based CDT is configured on an initial training sequence to automatically define its threshold value with no hand-crafted thresholds.

Given K as the residual and σ_k as the sample standard deviation over K, the threshold has been defined as follows:

$$Th = p\,\sigma_k$$

where *p* is a multiplicative factor.

CHANGES definition

During the operational phase, a change is detected when the CDT algorithm verifies whether the residual between the measurement and the prediction provided by the learned AI nominal model overcomes the computed threshold.

According to this experimental setup the following variables has been considered (see Figure 16):

- T_0: time instant the experiment starts
- T*: time instant the change happens
- T ref: time instant the model detects the change
- $\hat{T}$: time instant the experiment ends

TEST CONDITIONS

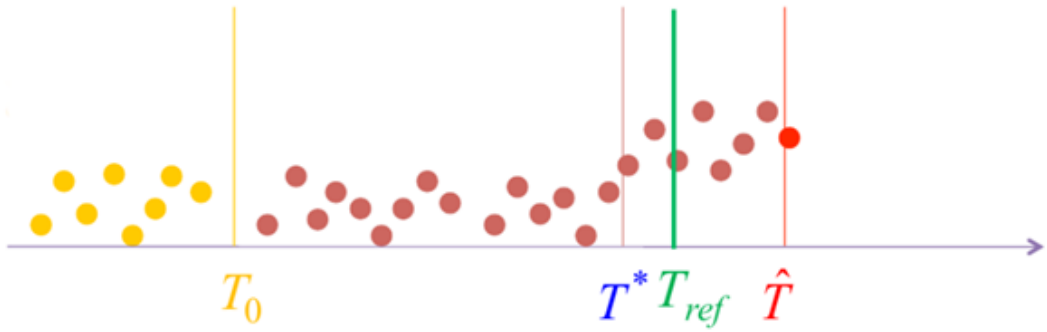

FIGURE 16. KPI 4.5 TEST CONDITIONS

During the experiment, when the model detects a change at time instant T*ref*, there are 3 different possible outcomes that could happen:

- False positive when: $T_0 < Tref < T^*$
- Correct Prediction when: $T^* < Tref < \hat{T}$ (with a detection delay (dd) equal to $t - T^*$)
- False negative when: $Tref > \hat{T}$

With these new definitions, the False Positive Rate (FPR) is computed as follows:

$$FPR = FP / M$$

where M that represents the number of experiments.

For this experiment the two models (AR vs ESN) has been benchmarked on the X-NUCLEO F401RE[9] MCU board computing the FPR.

RESULTS

In this section the results of the experiments performed are reported using:

- Training samples = 100 samples
- p = 2.5 (Table 3) and p = 5 (Table 4)
- M = 200
- $\hat{T}$ = 350

TABLE 3. KPI 4.5 RESULTS (P = 2.5)

	Fault			Incremental			Stuck-at		
	FPR	FNR	dd	FPR	FNR	dd	FPR	FNR	dd
AR	100%	-	-	100%	-	-	100%	-	-
ESN	5%	-	40,7	0,8%	-	97,3	13%	29%	21,7

TABLE 4. KPI 4.5 RESULTS (P = 5.0)

	Fault			Incremental			Stuck-at		
	FPR	FNR	dd	FPR	FNR	dd	FPR	FNR	dd
AR	81%	-		70%	-		63%	-	50,5
ESN	-	-	40,7	-	-	49,5	-	32%	21,7

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

CONCLUSIONS

According to the experimental findings the ESN model used for prediction modelling improves the FPR more than 20% with respect to the AR model as required by KP-I definition.

4.5.5 KPI 4.5 – AUTOENCODER BACK PROPAGATION COMPLEXITY VS ESN ONLINE TRAINING ADAPTION MODEL

CONSIDERED MODELS

In this experiment the following models has been taken into account for signal prediction tasks:

- Neural model 1:
  - type: Autoencoder
  - neurons: 18 (for the encoder)
  - inputs: 50 (this is set as observation window size since autoencoder is a non-recursive neural network)

- Neural model 2:
  - type: Echo State Network
  - neurons: 35 (for the ESN reservoir)
  - inputs: 3 (three axes accelerometer (X, Y, Z) input samples)

TEST CONDITIONS

The minimum adaptation time is defined as the time in which a model can make the inference process from the acquired (X, Y, Z) input data samples.

The two models have been mapped on the X-NUCLEO F401RE MCU board, and code has been instrumented in order to measure the acquisition time from the input data sample to the computation of the associated output data predictions (X', Y', Z').

RESULTS

In this section we reported the results of the experiments measured by using X-CUBE-AI tools v4.1.0 integrated into STM32CubeMX 5.3:

TABLE 5. KPI 4.5 RESULTS

	Adaptation Time (ms)	Input
Auto Encoder Model	0.428	50
ESN Model	0.0273	3

CONCLUSIONS

The ESN models improve the minimum adaptation time more than 20% with respect to the Auto Encoder model. The maximum adaptation time is not reported in table above, as it's well known by an expert in the field of Deep Learning that back propagation learning based procedure applied to an autoencoder vs training ESN model double at least by 2 the training complexity compared to ESN models.

4.6 Related requirements

In this section we consider the set of requirements that were described in deliverable D2.3 [5] within the context of the SEMIoTICS's framework. Thereby, we list the subset of requirements from D2.3 that are relevant for this sub use case 1 and we indicate in which part of the functional block architecture they are needed.

TABLE 6. UC3 COMPONENTS VS RELATED REQUIREMENTS

Component	Layer	Owner	Related Task	Related Requirements
VIM Connector	SDN orchestration layer	CTTC	3.1, 3.2, 3.5	R.GP.2 to 3, R.BC.12 to 13, R.NL.8 to 9, R.NL.11, R.NL.12
NFV Orchestrator	NFV orchestration layer	CTTC	3.2, 3.5	R.GP.2 to 3, R.BC.1 to 4, R.BC.14 to 15, R.NL.1 to 4, R.NL.10 to 11, R.S.4, R.UC2.12, R.UC3.9, R.UC3.12 to 14
Virtualized Infrastructure Manager	NFV orchestration layer	CTTC	3.2, 3.5	R.GP.2 to 3, R.BC.1 to 4, R.BC.14, R.NL.1 to 4, R.NL.10, R.UC2.12, R.UC3.9, R.UC3.12 to 14
VNF Manager	NFV orchestration layer	CTTC	3.2, 3.5	R.GP.2 to 3, R.BC.1 to 9, R.BC.11 to 15, R.NL.1 to 6, R.NL.8 to 11, R.UC2.12, R.UC3.9, R.UC3.12 to 14
Local embedded intelligence	Field layer	ST	4.3	R.FD.1 to 2, R.FD.4 to 7, R.FD.10 to 11, R.FD.3 to 9, R.FD.11 to 18
Use case 3	Field layer	ST	5.3, 5.6	R.UC3.1 to R.UC1.18

5 SUB USE CASE 2: CLOUD LEVEL DATA AGGREGATION AND VISUALIZATION DEMO

5.1 Story line

This sub use case is in line with the UC3's scenario 2, "Causal discovery and inference", described in section 2.1. Thereby, herein we provide a global data aggregation and storage at the cloud level. This is complemented by a visualization tool that displays the stored data or a processing of this data. On the one hand, the global data base is populated by the data transmitted by the IHES sensing units. On the other hand, the data base can contain data from third party databases of local authorities, e.g. the National Institute of Geophysics and Volcanology (INGV). Thereby, the monitoring of the IHES sensing units can be complemented with atmospheric trends (e.g., air quality metrics) collected from the aforementioned online databases. Information from local sensors and online services can be leveraged trigger alarms to alert people that are in danger, and to prevent the occurrence of tragedies. For instance, as stated in the scenario 2 of UC3, local earthquakes in conjunction with the temperature rise provoked an avalanche.

The cornerstone of this sub use case is a set of Virtual Network Function (VNFs) located at the cloud level, i.e., VNF2 which offers visualization though OpenHAB, and VNF3, which offers global data aggregation and storage of sensor data and events, as it is illustrated in Figure 10. This approach allows the aggregation of local information available at each GW in a global entity, which is made available for assessment and post-processing by analysts. Thereby, VNF2 and VNF3 offer a global view of sensor data and events, which complements the local view offered by GWs. Moreover, the global database can contain data from third party databases. The consolidation of this data allows local authorities to act proactively and avoid disasters. The VNF3 also provides an eastbound interface towards VNF2 to expose the aggregated data to the OpenHAB visualization tool. Thereby, the time-series of data from the IHES sensing units and from third party authorities can be presented in charts at various timescales upon request (e.g., when global alerts from the gateways are received)

Note that the SEMIoTICS NFV component is another important ingredient of this demo, which showcases the functionalities of the NFV Cloud, including the virtualized infrastructure (or NFVI), VIM, Cloud and Edge hypervisors, and the ETSI MANO which is the responsible of the lifecycle management of the VNF and the control of the virtualized resources at the cloud level. Thereby, a flexible, adaptable and virtualized infrastructure control technology is provided to deploy the functionalities of all VNFs, thanks to the SEMIoTICS NFV component.

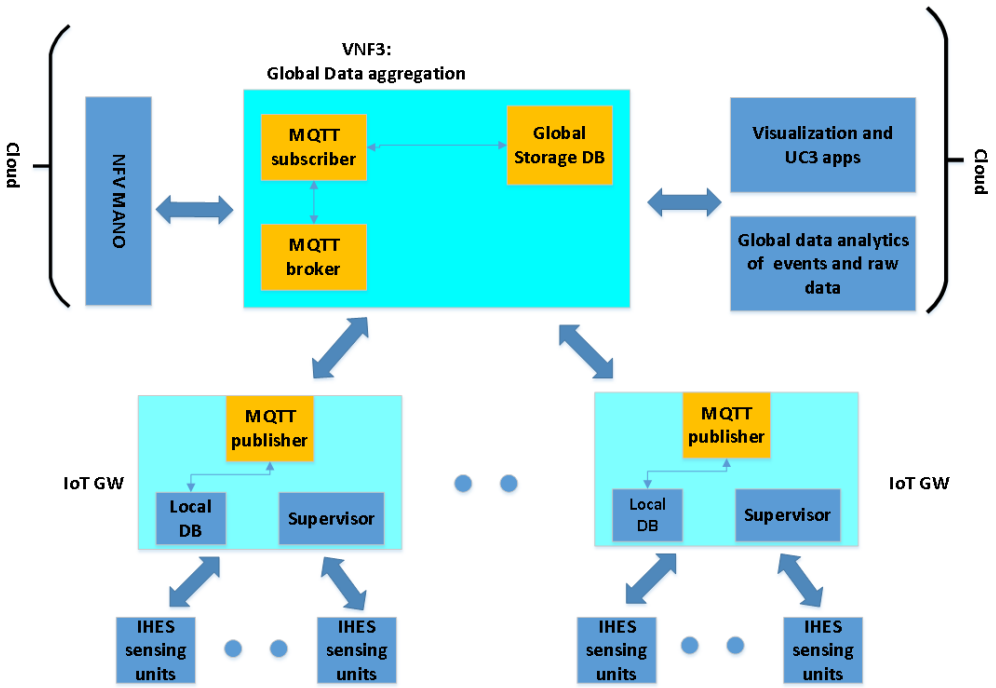

FIGURE 17. FUNCTIONAL BLOCK DIAGRAM OF THE DATA AGGREGATION DEMO

5.2 Scope and objectives

The main objectives of this sub use case are summarized as follows:

- Implement in a global entity, at the cloud level, the capability to receive, aggregate, visualize and store the time series data received from IHES GWs. This includes sensor data and events generated by the IHES sensing units and gateway. The global data base can also store data from third party databases, e.g. authorities supervising the environmental activity.
- The global data aggregation and visualization components are implemented within the framework of NFV, i.e. in the form of VNFs.
- Provide the NFV MANO components that control the lifecycle management of the VNF along with the management of the virtualized infrastructure to deploy the VNF.
- Provide a southbound interface for the GWs to send the IHES sensing unit data to the cloud level. The information model is analogous to the one used between the IHES sensing units and the GW. Thereby, the raw data and relevant events are expressed by means of a JSON model and an MQTT protocol is used for communication purposes.
- Provide an eastbound interface to let other external blocks to access the global data stored in the VNF3, e.g. the visualization component. To display the time-series of data as well as its trends, which allow external units to trigger alarms.

5.3 Interaction with SEMIoTICS framework and components

5.3.1 INTERACTION WITH THE SUPERVISOR AND LOCAL DB

The sub use case presented herein has a clear interaction with some of the SEMIoTICS' architectural components at the IoT GW level and at the field devices level. In fact, the local DB is leveraged as the source of sensor data which is aggregated within VNF3. Furthermore, the Supervisor service is the source of local and global events. It must be noted that the cloud-level VNF2 and VNF3 interact with the gateway level VNF-1 that hosts the supervisor service and local DB via an MQTT message flow.

5.3.2 INTERACTION WITH THE NFV COMPONENT

The SEMIoTICS' NFV component is a fundamental part of the sub use case presented in this section. The NFV component allows the virtualization of the computing, networking and storage resources at the cloud and at the gateway level yielding the so-called NFVI. Thereby, all VNFs can be deployed within VMs that leverages those virtual resources. Moreover, the NFV component has the role to manage the lifecycle of VNFs, through the NFV MANO subcomponent, see Figure 6. Thereby, it controls their onboarding within the NFV framework. Also, it provides configuration files that allow to automatize the instantiation of the VNFs, e.g. by defining their configuration files (VNFs and NSD files).

5.3.3 INTERACTION WITH THE OPENHAB VISUALIZATION COMPONENT

This sub-use case provides VNF3 for global data aggregation and storage, and VNF2 for data visualization leveraging the OpenHAB visualization component. More specifically, a global time series database at VNF3 stores events and relevant data stemming from several IHES sensing units associated to the corresponding IoT GW. Thereby, the database contains relevant global information generated by the IHES sensing units. The OpenHAB visualization component within VNF2 can access the database contained in the VNF3 to visualize the aforementioned data in the form of charts, at various timescales.

Based on the diagram flow reported in Figure 10, we derived a sequence flow diagram for the sub use case 2 that exemplifies the relation between the SEMIoTICS components involved in this sub use case. It also highlights the steps described in the story line of this sub use case. As it can be observed, the data generated at the IHES sensing units is processed locally by the embedded intelligence. Then, it is sent to the GW and afterwards to the cloud, where the VNF3 is located. The VNF3 provides the global data aggregation and storage capabilities of the IHES sensing units' data. Moreover, the VNF3 has access to external data bases that provide further environmental data. As a consequence, a trend analysis can be done at the VNF3 by leveraging the data from the IHES sensing units' along with the data provided by external data bases. Thereby, potential risky data trends can be identified, which triggers and alarm. That trend analysis will be visualized thanks to the OpenHAB capabilities at VNF4.

5.4 Validation

In the scope of this first cycle deliverable, global data aggregation was validated at CTTC premises with a dump file containing snapshots of the data generated by the real IHES sensing units, while end-to-end validation and integration with the VNF-2 (i.e., OpenHAB visualization) will be implemented as for the upcoming deliverable D5.11. Thereby, is this section we explain the implementation of an emulated MQTT communication chain. This allows the generation of emulated IHES sensing fata by reading a dump file provided by ST. Then, MQTT messages stemming from the generated data are published and conveyed to an MQTT broker. Finally, an MQTT subscriber receives the data, i.e. the MQTT topics. Note that, the MQTT broker and MQTT subscriber will be part of the functionalities of the VNF3 in the final setup of D5.11, whereas the MQTT publisher will be part of the GW. However, we consider the MQTT publisher herein to have a source of data.

For the implementation purposes we implement each of the components, i.e. the MQTT publisher, MQTT broker and MQTT subscriber as Docker containers [6]. This approach allows that each component is isolated in containers, with the bundle of software and libraries that they need. This is a modular approach that will allow us to easily drop the MQTT publisher component in the final D5.11 and integrate the one of the IoT GW. Moreover, it paves the way to automate the installation, configuration and deployment of each of the MQTT components. Namely, to this end, we will use Docker compose [7], which is a tool to orchestrate multi-container

Docker applications. That is, Docker compose allows to configure and to run each of the Docker containers. In **Error! Reference source not found.**, we display a functional block diagram of the implementation that we have just introduced. Note that it is in line with Figure 18, and then we will implement this MQTT E2E chain as a VNF on the NFV testbed described above. Next, we explain in more detail the implementation of the MQTT functional blocks:

- "MQTT publisher" Docker container: Within this container, a python script reads a dump file, which contains the data that was extracted from the IHES sensing units. This emulates the IHES sensing units' data. As it was introduced above, this dump file contains data in JSON format that embeds hierarchical MQTT topics and their associated values. The python code converts the data from JSON format to an MQTT message to be sent. Afterwards, the MQTT message is sent or published to the broker.
- "MQTT broker" Docker container: This container will deploy and will run the functionality of an MQTT broker. To this end, an open source broker will be leveraged, it is called eclipse mosquitto [8]. As it will be shown below, Docker compose will create a container with this functionality by using the latest Docker image of the eclipse mosquitto.
- "MQTT subscriber" Docker container: The aim of this container is to implement an MQTT subscriber that is able to receive the MQTT topics that are sent by the MQTT publisher described above. That is, all the MQTT topics generated by the IHES sensing units

All Docker containers that include the aforementioned E2E MQTT chain are embedded within the VNF3. Recall that the received side of the MQTT chain corresponds to the global data aggregation and the MQTT transmitting side emulates the IHES sensing units. Moreover, the VNF3 is managed by the NFV MANO, which is implemented by the OSM and OpenStack controller nodes. Finally, note that the VNF3 is deployed on top of the virtualized resources exposed by the OpenStack compute node, which represents the NFVI. It is important to note that the CTTC NFV testbed is used to test the VNF3 in this cycle 1. In the cycle 2, the VNF3 will be integrated within the overall UC3 testbed, which contains the NFV MANO and NFVI, but where the OSM is not necessary. Here we just use OSM as a high-level interface, to ease the testing of VNF3.

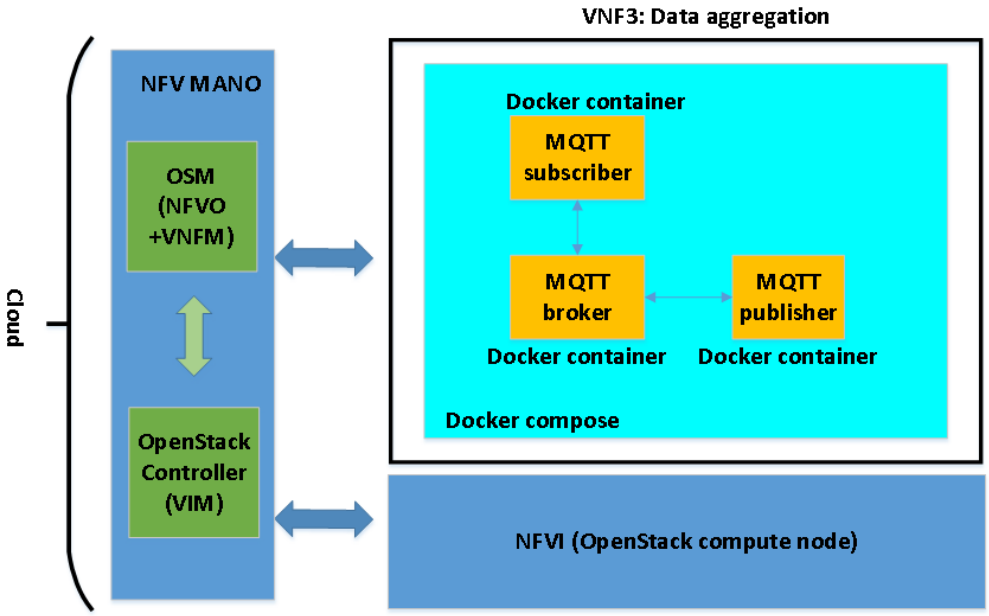

FIGURE 18. ILLUSTRATION OF THE VNF3 WITHIN THE CTTC'S NFV TESTBED USED IN CYCLE 1 FOR TESTING PURPOSES OF THE VNF3

Next, in Figure 19**Error! Reference source not found.**, it is demonstrated that we are able to onboard properly the NSd and VNFd packages into the library of OSM. Recall that these are a set of configuration files that describe the properties of our VNF3, the requirements in terms of computing and networking that it has. Also, they describe the features of the VM that will host the VNF3 and the initial configuration and software packages installations that we need in the VM. We have called the NSd and VNFd as vnf3_nsd and vnf3_vnfd, respectively. It can be observed that OSM has been on boarded properly the NSd and VNFd, as they appear on the list of available packages in the OSM library. Note that the osm instruction "osm nsd-list" and "osm vnfd-list" were used.

Then, we trigger the instantiation of the NS that we have on boarded in OSM for our VNF3. This means basically that OSM will communicate with the OpenStack controller, which creates the VM that will host our VNF3 on top of the virtual resources exposed by the OpenStack compute node. For that, obviously, the OSM takes into account the information embedded within the NSd and VNFd. Note that to trigger the NS instantiation we used the OSM command "osm ns-create –ns_name semiotics_vnf3 –nsd_name vnf3_nsd". Observe that you must specify for the nsd_name option the name of the NSd that you want to instantiate, otherwise the NS will not be instantiated.

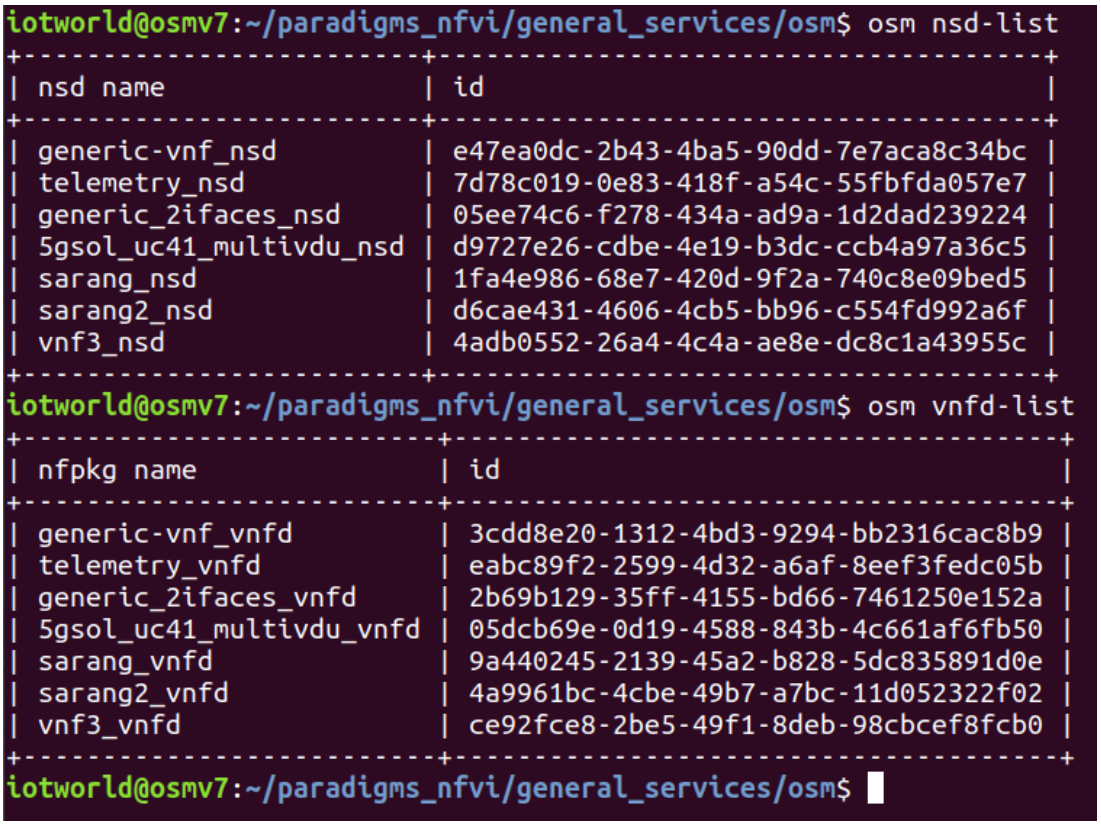

FIGURE 19. NSD AND VNFD PACKAGES ARE ONBOARDED PROPERLY TO OSM

Once the NS instantiation is performed, the VM for our VNF3 is created. Next, in Figure 20, we show how the MQTT publisher, MQTT broker and MQTT subscriber containers, deployed within VNF3, work as expected.

First, note how the MQTT broker, whose tag is “broker_1”, opens its port 1883 to listen new connections. And then, it detects a new connection from the MQTT publisher, whose IP is 172.18.0.3. Then, we can see how the MQTT publisher prints the line that it has read from the dump file that contains the snapshots of the IHES sensing units. This is the JSON data that embeds the MQTT topic and value that are transmitted by the MQTT publisher. Then, we can observe in Figure 20, that the MQTT broker detects a new connection from 172.18.0.4, this is the MQTT subscriber. Afterwards, the MQTT subscriber displays the MQTT topics and the associated value that it has received. Note, that the MQTT subscriber receives properly the MQTT message sent by the MQTT publisher and extracts the MQTT topic and value as it was expected. Then, the MQTT publisher sends a new MQTT message, which corresponds to the next line of the dump file, and the MQTT subscriber receives it properly. And this process is iterated. Observe that herein for the sake of the clarity we have just put two lines in the dump file, which correspond to two MQTT messages. Also, note that the container for the MQTT publisher is restarted periodically. This is needed because the python code that implements the MQTT publisher finishes after reading all the content of the dump file that contains the snapshots of the IHES sensing units.

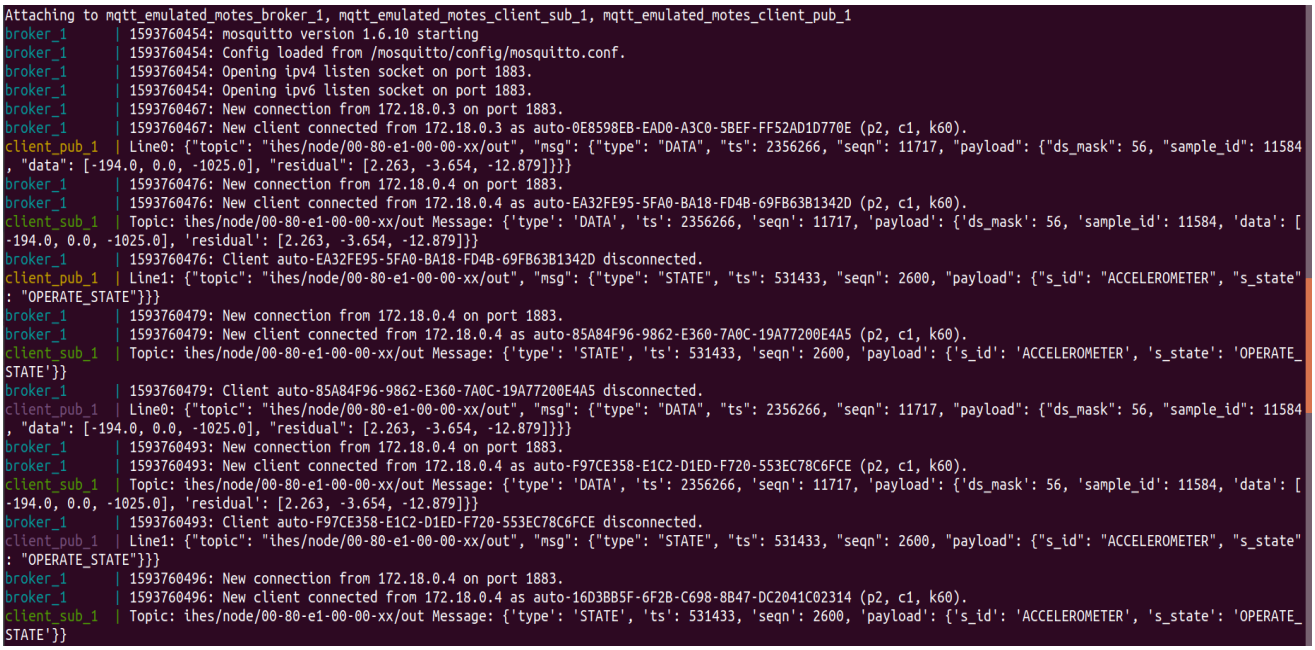

FIGURE 20. THE MQTT TOPICS AND VALUES ARE RECEIVED PROPERLY BY THE MQTT SUBSCRIBER

In this sub section we list the D5.1 SEMIoTICS’ KPIs [9] related to the sub use described in this section 5. Also, we describe the role of this sub use case in this KPI.

TABLE 7. SUB USE CASE 2 RELATED KPIS

SEMIoTICS’ KPI ID	KPI description	Role in this KPI
KPI-6.1	Reduce manual interventions required for bootstrapping of smart object in each use case domain by at least 80%.	The bootstrapping service for the smart objects involves the availability of other functional blocks, such as the supervisor, the local DB, the MQTT broker/publisher/subscriber at the

		IoT GW. And also, the MQTT subscriber, MQTT broker and global data base at the cloud level. An implementation of these functional blocks in the form of VNFs automates its availability for the bootstrapping process. In this regard, note that the NFV MANO automatizes the creation of the VNFs and its deployment on top of the NFVI in the form of VMs. Also, it automatizes the configuration, software installation and programs executions in the VM at boot time. Therefore, such operations are expected to eliminate user intervention completely.

5.5 Related requirements

In this section we consider the set of requirements that were described in deliverable D2.3 [5] within the context of the SEMIoTICS's framework. Thereby, we list the subset of requirements from D2.3 that are relevant for this sub use case and we indicate in which part of the functional block architecture they are needed.

TABLE 8. SUB USE CASE 2 RELATED REQUIREMENTS

SEMIoTICS' Req-ID	Req-ID description	Requirement context
General platform requirements		
R.GP.2	Scalable infrastructure due to the fast-paced growth of IoT devices	As it is shown in **Error! Reference source not found.** the architecture tackled herein is highly scalable. Namely, on the one hand, the number of IoT GWs can scale to face the growth of IoT devices. Moreover, the global data aggregation and storage implemented at the cloud level by VNF3 provides more scalability, as it resides on top of virtual resources controlled by the VIM at the NFV MANO. Thereby, more virtual computing and storage resources can be easily allocated when needed for the scalability purposes.
Network layer and Backend/Cloud Layer Requirements		

R.NL.1/R.BC.1	Controller Node requirement: At least 6 CPU cores and 32 GB RAM	This is in line with the requirements that have already reported above for e.g. the OpenStack controller, see section 5.4. Note that, in addition, it is required that the controller nodes have around 100 GB of storage memory, see section 5.4.
R.NL.2/R.BC.2	Controller Node requirement: At least 2 Network interfaces	This is a requirement applicable to the computer holding the controller, e.g. the OpenStack controller, see section 5.4
R.NL.3/R.BC.3	Controller Node Requirement: Linux OS	The controller nodes at the NFV MANO context, e.g. the OpenStack controller, require Linux OS.
R.NL.5/R.BC.5/ R.BC.6/ R.BC.7	Hypervisor Nodes Requirement: At least 4 CPU cores and 8 GB RAM, at least 2, 1Gbps Network interfaces.	This is fulfilled or in line with the requirements for the compute nodes of the NFVI, see section 5.4
R.NL.6/R.BC.8/ R.BC.9	Hypervisor Nodes: KVM and Linux Containers (LXD) must be supported by the Hypervisor Linux OS	This is supported in the compute nodes that form the NFVI, e.g. at the cloud level.
UC3 Sensing		
R.UC3.9	IoT Sensing gateway shall support 1 to many standard IP based (i.e. TCP transport) M2M communication protocol to interface a number N of connecting Sensing units (e.g. broadcast type).	This is a requirement at the IoT GW, see **Error! Reference source not found.**, which is fulfilled in the implementation, as already discussed in section 0.
R.UC3.12	IoT Sensing gateway shall be capable to run Linux (e.g. Ubuntu OS) and standard graphics and browser libraries.	This is a requirement at the IoT GW, see **Error! Reference source not found.**, which is fulfilled in the implementation, as already discussed in section 0.
R.UC3.13	IoT Sensing gateway should be able to support http and standard protocols for cloud interfacing.	This is a requirement at the IoT GW, see **Error! Reference source not found.**, which is fulfilled in the implementation, as already discussed in section 0.
R.UC3.14	The specific M2M protocol adopted on UC3 is based on MQTT. A MQTT broker service will be available to dispatch messages between the coordinating Sensing	As it has been explained above in this section, the communication between the IHES sensing units and the IoT GW is based on MQTT. In the same vein, the

	gateway and its associated Sensing units.	communication between the IoT GW and the VNF3 at the cloud, i.e. the global data aggregation and storage, is based on MQTT as well.

6 SUB USE CASE 3: SYSTEM RELIABILITY – PATTERN-BASED SENSOR DEPENDABILITY MONITORING

6.1 Story line

Considering the criticality of the use case application, revolving around earthquake monitoring in public areas, sensor dependability is of very high importance. Therefore, and to avoid system downtime, redundant sensors will be deployed to ensure that even if some sensors fail, the system will continue to operate.

Therefore, this sub-use case revolves around unsupervised monitoring from environmental sensors with anomaly detection (from temperature, pressure, humidity), as well as unsupervised monitoring from inertial sensors with anomaly detection (from accelerometer and gyroscope), with dependability and reliability guarantees.

In Figure 21, a topology is depicted that corresponds to the scenario of sub use case 3, and more specifically distributed anomaly vibration monitoring for earthquake detection. In this scenario, we consider that a Gateway is connected to N vibration sensors (where N>=2 for redundancy), which are identical. At any time, all of them are up and running, for redundancy in the monitoring. This redundant topology can be modelled and monitored as the Dependability property, through the appropriately defined pattern rule. Therefore, the infrastructure owner will be able to monitor in real-time the dependability status of his/her deployment, potentially triggering adaptations (e.g., the disabling of one sensor while waiting for a replacement).

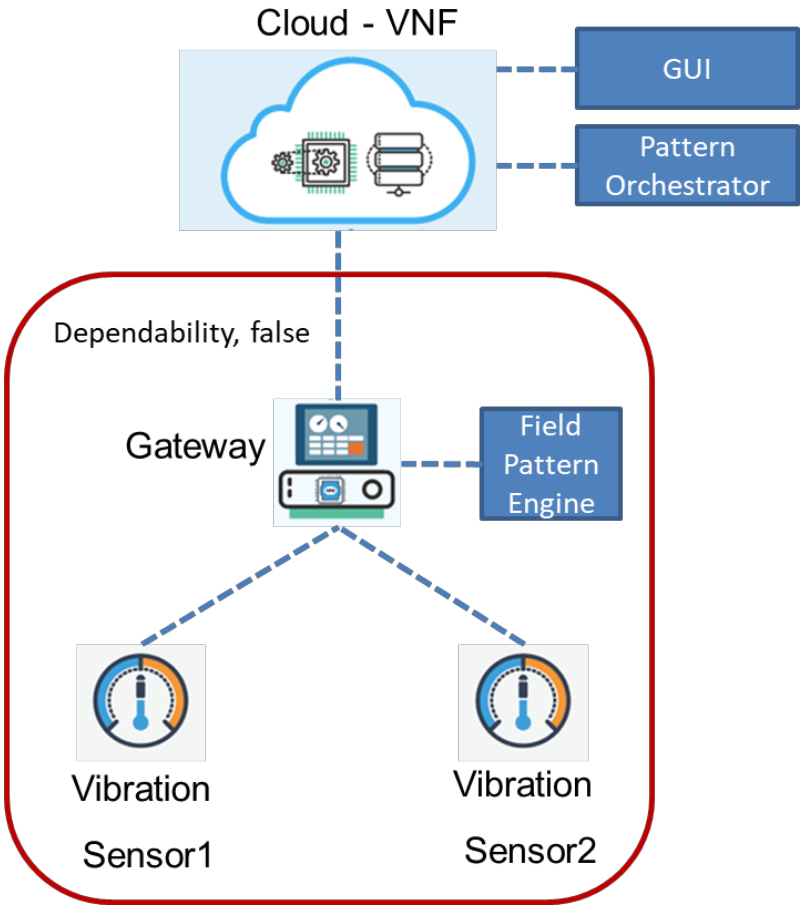

FIGURE 21. SUB USE CASE 3 TOPOLOGY

6.2 Scope and objectives

The overarching aim and objective of the sub-use case is to demonstrate in practice how the pattern-based monitoring capabilities of SEMIoTICS can be used to provide a real-time view of the sensing dependability posture of the system.

In this context, pattern components are deployed at the field layer (which is the focus of UC3), with the support of components deployed at the backed for visualisation purposes. Regarding the former, a pattern engine runs on the Gateway, able to reason locally about the dependability properties of the anomaly detection setup.

Said Pattern Engine will be a lightweight version of the engines deployed in other scenarios, and will feature appropriate Dependability Pattern Rule to verify that this Dependability property is satisfied and, in case that a sensor fails, will be able to reason and report the failure of the property to the backend. When the sensor is restored, reasoning will verify that the dependability property is restored.

The subsections that follow present more details into the building blocks and mechanisms enabling the pattern-driven monitoring and adaptation in the context of this scenario.

6.3 Interaction with SEMIoTICS framework and components

A sequence diagram depicting the involved entities as well as the events taking place in the scenario in terms of the pattern-based dependability monitoring is provided in Figure 23.

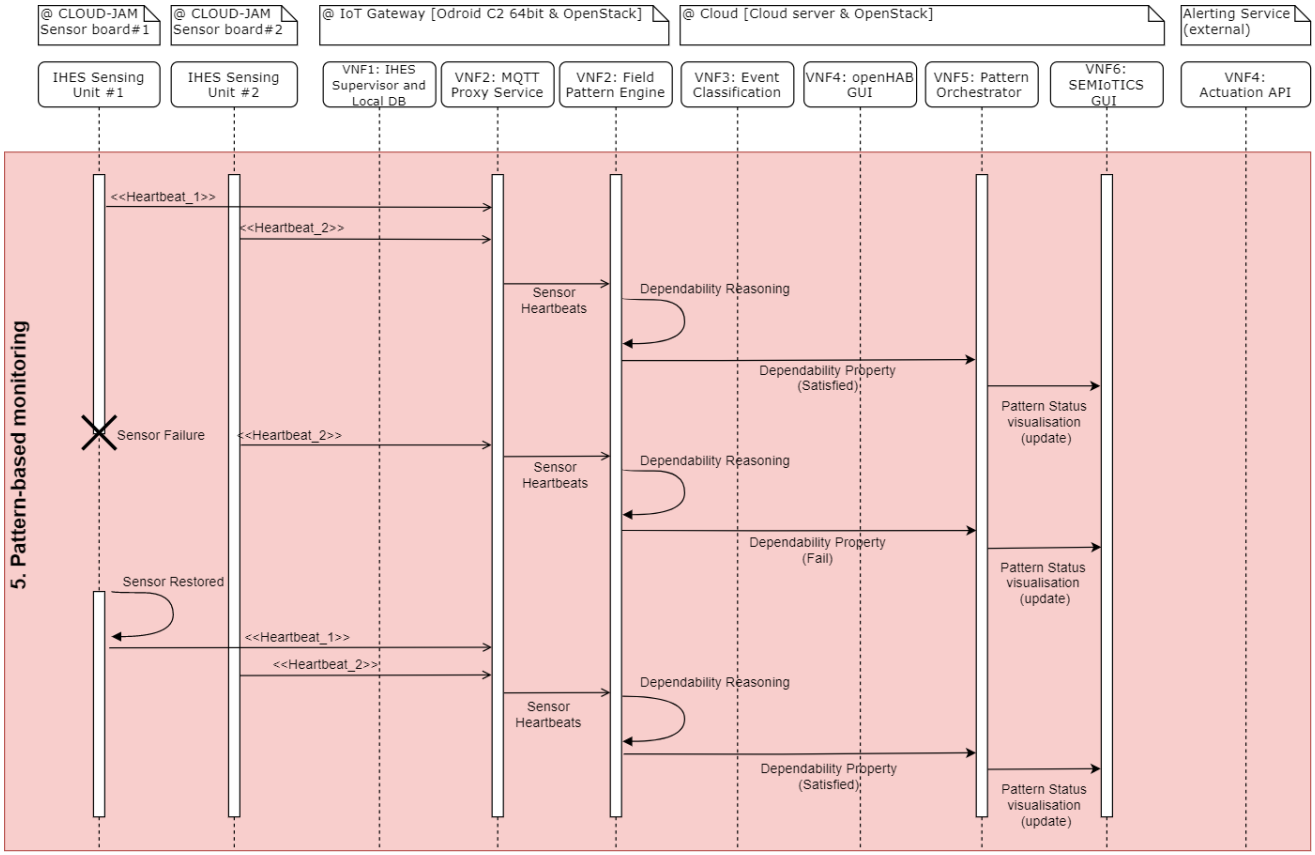

FIGURE 22. REAL-TIME PATTERN-BASED SENSING DEPENDABILITY MONITORING SEQUENCE DIAGRAM

As shown in the above figure, the pattern engine monitors the liveness messages from the two sensors ("heartbeats"), via integration with the MQTT broker, leveraging this information to reason about the redundancy of the monitoring system. When one of the sensors fails, the redundancy requirement is violated, and when it is restored (e.g., the sensor comes back online or is restored), the property is satisfied once again.

6.3.1 COMPONENTS

Other than the sensors and field gateway itself, with the various capabilities described in the previous subsections, the specific sub-use case is implemented through the deployment of:

(i) the Field Pattern Engine which is able to locally reasoning about the dependability properties of the anomaly detection setup, comprising a lightweight instance of Pattern Engine with MQTT integration to be able to interact with the Broker;

(ii) the Pattern Orchestrator component, with a supporting role at the SEMIoTICS Backend (for bootstrapping and visualisation proxy purposes);

(iii) the SEMIoTICS GUI, and the pattern visualisation part in specific, deployed at the backend for pattern status visualisation (and UC homogeneity) purposes.

While Cycle 1 efforts will focus on the integration of the field-level components (i.e., integrating the pattern engine with the Gateway and MQTT Broker, reasoning on the sensor properties), Cycle 2 will focus on also integrating (ii) and (iii) above in the sub use case.

It should be noted that while components (ii) and (iii) are included to facilitate deployment and provide a more homogeneous setup and use of SEMIoTICS components across use cases (even in the case of use case 3, which is horizontal, focusing on the field layer), these are optional; the full sequence and associated dependability monitoring capabilities could be implemented without these components. Such a simpler deployment could be achieved if the orchestration and pattern rule are preconfigured at the field pattern engine. In both cases, though, the dependability reasoning happens independently at the field layer, without any knowledge or reasoning capabilities assumed of other layers, thus demonstrating SEMIoTICS' approach of semi-autonomous operation of each layer.

6.3.2 PATTERNS SPECIFICATION

Other than the components themselves, an important part in realising the sub-use case is the configuration of the pattern components through the specification of the orchestration and associated pattern rule, leveraging the SEMIoTICS pattern language (see D4.8 – "SEMIoTICS SPDI Patterns (final)").

In specific, the topology depicted in Figure 21 is described using the SEMIoTICS orchestration specification language as shown in Figure 23.

```
ORCH "Dependability"

Iotsensor ("VibrationSensor1", "activityaddress", "activityport"),

Iotsensor ("VibrationSensor2", "activityaddress", "activityport"),

Iotgateway ("Gateway", "activityaddress", "activityport"),

Link("Link1", "VibrationSensor1", "Gateway"),

Link("Link2", "VibrationSensor2", "Gateway"),

Merge("Merge1", "VibrationSensor1", "VibrationSensor2", "Gateway",
"Link1", "Link2"),

Property("Prop0", required, dependability, "1", end_to_end, "Merge1",
false)
```

FIGURE 23. SUB USE CASE 3 ORCHESTRATION SPECIFICATION IN SEMIOTICS LANGUAGE

In terms of the pattern rule to be employed in the specific scenario and since dependability is the focus, the Redundancy Pattern will be leveraged, as defined in deliverable D4.8. Adapting said pattern to the specific use case would result in a Drools rule as the one shown in Figure 24.

```
1.  rule "Dependability Rule"
2.  when
3.  IoTSensor($sensor1:=id)
4.  Property ($sensor1:=subject, category==“heartbeat”,
    satisfied==true)
5.  IoTSensor($sensor2:=id, $sensor1!=$sensor2)
6.  Property ($sensor2:=subject, category==“heartbeat”,
    satisfied==true)
7.  $PR: Property ($sensor1:=subject, category==“dependability”,
    satisfied==false)
8.  then
9.  modify($PR){satisfied=true};
10. end
```

FIGURE 24. SENSING REDUNDANCY PATTERN AS DROOLS RULES

The **when** part of the rule specifies:

1. the placeholders $sensor1, $sensor2;
2. the extra condition that the sensors re not the same type;
3. the checks that each sensor transmits a heartbeat in a timely manner;
4. the orchestration property that can be guaranteed through the application of the pattern, i.e., the dependability property in this case ($pr).

The **then** part verifies that the orchestration property holds since every essential component is included in the when part (satisfied=true).

6.4 Validation

A local testbed at STS was setup to integrate and test the above sub use case, emulating the topology shown in Figure 21. The core component, other than the sensors, gateway, MQTT broker and other parts common to the other sub-use cases, is the field pattern engine. The technologies used to implement this include Java, Maven and Spring Boot for the development and creation of the field pattern engine. Moreover, as detailed in deliverable D4.8, the pattern engine integrates the Drools Business Rules Management System (BRMS) solution to implement its reasoning capabilities. Moreover, the Field Pattern Engine is extended for UC3 via integration of the Eclipse Paho Java Client, an MQTT client library written in Java. Finally, the Docker platform is used for packaging and running the pattern engine application. Leveraging the testbed deployed for this purpose, the integration and functionality of the components was validated through implementation of the sequence diagram shown in Figure 23, excluding interactions with backend components (which will be implemented in Cycle 2).

Figure 25 Figure 26 shows a screenshot of the simulated sensors (IHES nodes) transmitting their heartbeats, once instantiated, to the *ihes/node/$mac/out/events/heartbeat* topic of the MQTT broker, where *$mac* is the MAC address of each of the sensing nodes.

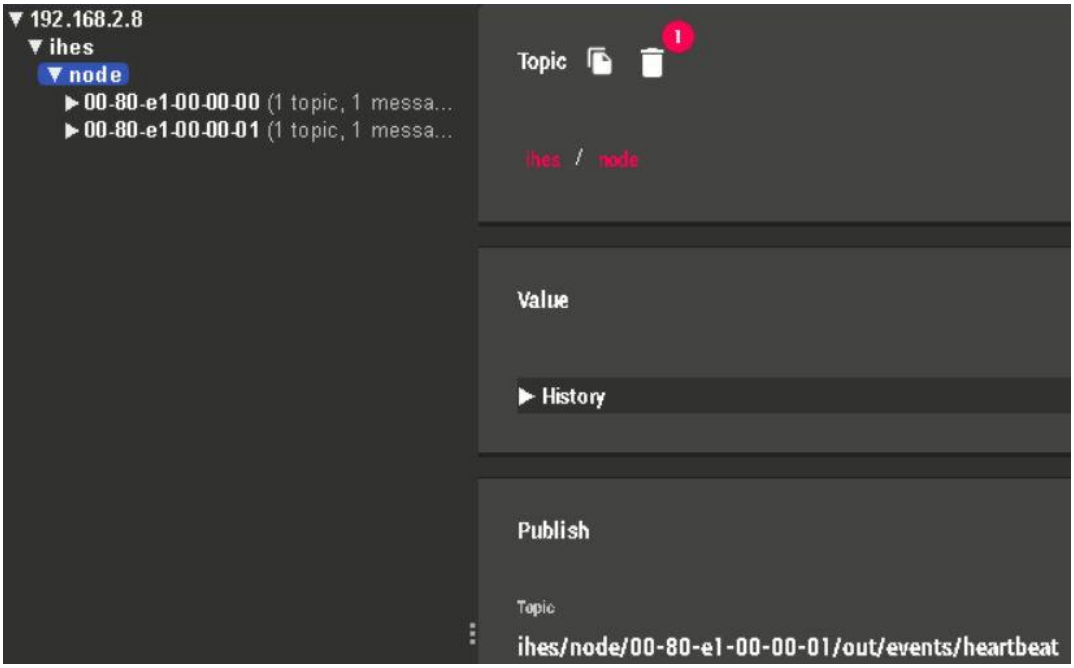

FIGURE 25. SIMULATED IHES NODES (PUBLISHING HEARTBEATS AT IHES/NODE/$MAC/OUT/EVENTS/HEARTBEAT TOPIC)

Figure 26 shows the orchestration information, as relayed to the field pattern engine for bootstrapping purposes. In the final scenario (Cycle 2), these will be relayed by the Pattern Orchestrator component at the backend, though they can also be sent via a script or be pre-encoded into the pattern engine, if a completely autonomous operation is envisioned.

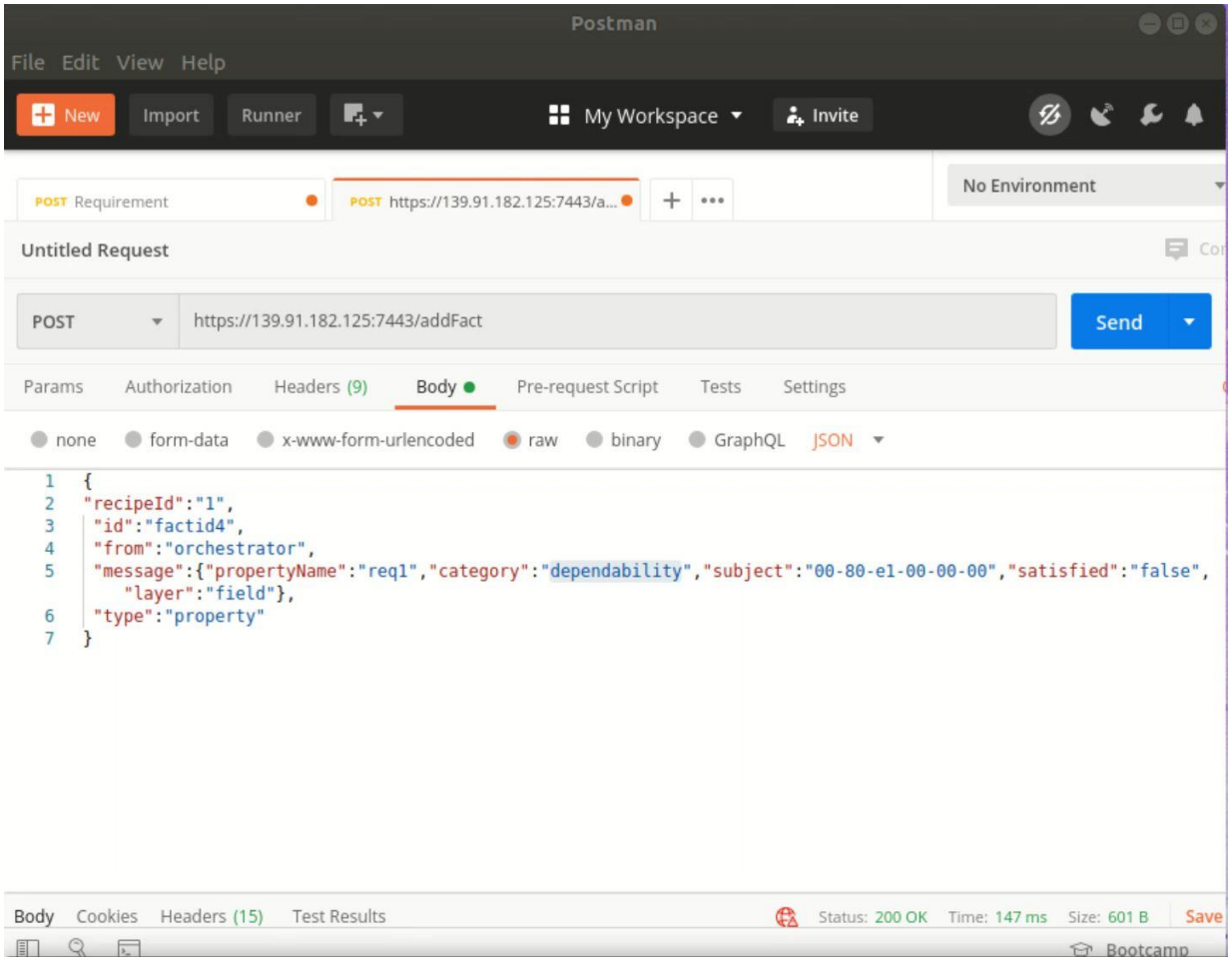

FIGURE 26. PATTERN ORCHESTRATION INFORMATION RELAYED TO FIELD PATTERN ENGINE

Finally, Figure 27 and Figure 28 showcase the heartbeat messages as received and relayed by the broker, and how these are received and trigger reasoning actions at the field pattern engine, respectively. 0

```
1594045470:     # (QoS 0)
1594045470: sensors 0 #
1594045470: Sending SUBACK to sensors
1594045470: Received SUBSCRIBE from sensors
1594045470:     $SYS/# (QoS 0)
1594045470: sensors 0 $SYS/#
1594045470: Sending SUBACK to sensors
1594045478: Received PUBLISH from sensors (d0, q0, r0, m0, 'ihes/node/00-80-e1-00-00-00/out/events/h
eartbeat', ... (830 bytes))
1594045478: Sending PUBLISH to Field_Pattern_Engine (d0, q0, r0, m0, 'ihes/node/00-80-e1-00-00-00/ou
t/events/heartbeat', ... (830 bytes))
1594045478: Sending PUBLISH to sensors (d0, q0, r0, m0, 'ihes/node/00-80-e1-00-00-00/out/events/hear
tbeat', ... (830 bytes))
1594045500: Received PUBLISH from sensors (d0, q0, r0, m0, 'ihes/node/00-80-e1-00-00-01/out/events/h
eartbeat', ... (830 bytes))
1594045500: Sending PUBLISH to Field_Pattern_Engine (d0, q0, r0, m0, 'ihes/node/00-80-e1-00-00-01/ou
t/events/heartbeat', ... (830 bytes))
1594045500: Sending PUBLISH to sensors (d0, q0, r0, m0, 'ihes/node/00-80-e1-00-00-01/out/events/hear
tbeat', ... (830 bytes))
1594045506: Received PINGREQ from mqtt_8dd39dce.d4153
1594045506: Sending PINGRESP to mqtt_8dd39dce.d4153
1594045515: Received PINGREQ from Field_Pattern_Engine
1594045515: Sending PINGRESP to Field_Pattern_Engine
```

FIGURE 27. MQTT BROKER RELAYING HEARTBEATS

```
2020-07-06 17:25:18.021  INFO 31075 --- [nio-7443-exec-4] o.a.c.c.C.[Tomcat].[localhost].[/]       :
 Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-07-06 17:25:18.021  INFO 31075 --- [nio-7443-exec-4] o.s.web.servlet.DispatcherServlet        :
 Initializing Servlet 'dispatcherServlet'
2020-07-06 17:25:18.028  INFO 31075 --- [nio-7443-exec-4] o.s.web.servlet.DispatcherServlet        :
 Completed initialization in 7 ms
Type is property
Property{type='PROPERTY', propertyName='req1', propertyType='null', category='dependability', value=
null, datastate='null', subject='00-80-e1-00-00-00', satisfied=false, verificationType='null', means
='null', layer='field', recipeID='1', factID='factid4'}
2020-07-06 17:25:18.242  INFO 31075 --- [nio-7443-exec-4] o.d.c.k.builder.impl.KieRepositoryImpl   :
 KieModule was added: MemoryKieModule[releaseId=eu.semiotics.pattern:fetch-external-resource:1.0.0-S
NAPSHOT]
2020-07-06 17:25:18.385  INFO 31075 --- [nio-7443-exec-4] o.d.c.k.builder.impl.KieRepositoryImpl   :
 KieModule was added: MemoryKieModule[releaseId=eu.semiotics.pattern:fetch-external-resource:1.0.0-S
NAPSHOT]
2020-07-06 17:25:18.385  WARN 31075 --- [nio-7443-exec-4] o.e.a.i.i.DefaultUpdatePolicyAnalyzer    :
 Unknown repository update policy '', assuming 'never'
2020-07-06 17:25:20.606  WARN 31075 --- [nio-7443-exec-4] o.a.maven.integration.MavenRepository    :
 Unable to resolve artifact: eu.semiotics.pattern:fetch-external-resource:1.0.0-SNAPSHOT
Dependability rule triggered
1 rules fired
Fact count is 5
```

FIGURE 28. FIELD PATTERN ENGINE (SUBSCRIBED TO BROKER; RECEIVING UPDATES & REASONING ON DEPENDABILITY PROPERTY)

6.5 Related KPIs

The pattern-related work (carried out mostly within Task 4.1), the output of which forms the basis for this sub-use case, directly targets Objective 1 of the project. This objective states: “Objective 1: Development of patterns for orchestration of smart objects and IoT platform enablers in IoT applications with guaranteed security, privacy, dependability and interoperability (SPDI) properties”. Moreover, this work is also relevant to

various aspects of other overarching objectives of the project as well; e.g., the patterns specified herein are important enablers of the multi-layered embedded intelligence and semi-autonomic operation of SEMIoTICS, as sketched in Objective 4 of the project.

In this context, a detailed mapping of the pattern-related work to the overarching project objectives and the associated KPIs have been presented in subsection 2.6 of deliverable D4.8.

6.6 Related requirements

Some key requirements pertinent to the SPDI pattern-driven operation of SEMIoTICS, also including non-SPDI specific requirements, such as underlying/global requirements, functional requirements etc., that directly or indirectly affected the design of SPDI patterns and which were considered during the pattern language definition are presented in subsection 2.5 of deliverable D4.8.

7 OVERALL KPIS AND REQUIREMENTS VALIDATION

As long as the integration of the UC3 demonstrator goes on, we will also present and analyze the relevant KPIs that has been reported as of interest for each one of the sub use cases storylines (see sections 4.5 and 6.5). For some of them the analysis and results has been already presented (sub use case 1), whereas for other sub use cases (namely 2 and 3) they will be reported as part of D5.11.
All together these KPIs forms the complete set of KPIs identified in task 5.1 as of relevance for the specific UC3 scenario or for the SEMIoTICS components that this scenario will use.

8 CONCLUSION

This deliverable describes the initial draft of the executive status activities done in task 5.6 “Demonstration and validation of IHES- Generic IoT (Cycle 1)”. This document is an executive summary and report of the initial integration activities that will lead during next cycle 2 integration to complete the integration of the whole UC3 demonstrator within SEMIoTICS architecture, as originally designed and declared in D2.5.
For convenience, we had identified three sub use case scenario that incrementally will be integrated in task 5.6 and that, all together, will demonstrate the whole set of capabilities of the UC3 scenario.
In this initial version of the document the three sub use cases have been declared by providing for each one of them the links with relevant design documentation already documented in former deliverables, and their relation with the SEMIoTICS components and infrastructure.
In particular, we focused in documented all achieved cycle 1) integration activities by providing as well and anticipation of next planned cycle 2 ones.
These latter ones will be the focus of next D5.11 that will be the final updated version of this document.

9 REFERENCES

[1] T. G. e. a. P.Mell, "The NIST Definition of Cloud Computing," *NIST,* vol. 800, p. 145, 2011.

[2] J. S. e. al., "Network Functions Virtualization for IoT (final)"," SEMIoTICS deliverable D3.8, 2020.

[3] M. F. e. al., "Embedded Intelligence and Local Analytics (Final)," SEMIoTICS deliverable D4.10, April 2020.

[4] M. f. e. al., "SEMIoTICS High-Level Architecture (Final)," SEMIoTICS deliverable D2.5, December 2019.

[5] M. F. e. al., "Requirements specification of SEMIoTICS framework," SEMIoTICS deliverable D2.3, November 2018.

[6] "Docker," [Online]. Available: https://www.docker.com/. [Accessed July 2020].

[7] "Docker Compose," [Online]. Available: https://docs.docker.com/compose/. [Accessed July 2020].

[8] "Eclipse mosquitto," [Online]. Available: https://mosquitto.org/. [Accessed July 2020].

[9] F. K. e. al., "SEMIoTICS KPIs and Evaluation Methodology," SEMIoTICS deliverable D5.1, January 2020.