SEMIoTICS

Deliverable D5.11
Demonstration and validation of IHES - Generic IoT Use Case (Cycle 2)

Deliverable release date	29.12.2020 (revised on 20.04.2021)
Authors	1. Mirko Falchetto, Danilo Pau (ST) 2. Manolis Chatzimpyrros, Othonas Soultatos, Grigoris Kalogiannis, Konstantinos Fysarakis (STS) 3. Kostas Ramantas (IQU) 4. Jordi Serra, Luis Sanabria-Russo, David Pubill, Angelos Antonopoulos, Christos Verikoukis (CTTC)
Responsible person	Mirko Falchetto (ST)
Reviewed by	Mateusz Kaminski, Rafal Szustkiewicz (BS), Jordi Serra (CTTC), Ramantas, Kostis (IQU), Michalodimitrakis, Manolis, Nikolaos Petroulakis (FORTH), Konstantinos Fysarakis (STS), Felix Klement (UP)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, 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
mw	Milli Watts
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 final summary related to reporting all activities within Task 5.6 ("Demonstration and validation of IHES-Generic IoT scenario") in SEMIoTICS.

In more detail, the reporting period covers all technical integration activities done during cycle 1 and cycle 2 integration of SEMIoTICS UC3 (M24-M36). In section 1.1 the updates from D5.6 has been shortly reported.

Section 2 introduces a quick 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 complete presentation of all the sub use case scenarios (section 2.4) derived from the one presented in section 2.1, that has lead the incremental integration from field level to backend level of the UC3 demonstrator.

Section 3 presents the final version of the integrated SEMIoTICS UC3 testbed based on OpenStack and VNF virtualization, that has been used to showcase the three incremental sub use case storylines discussed in sections from section 4 to section 6. Final sections 7 and 8 are dedicated to derive the conclusions of this final cycle of integration and a complete summary with results related to the KPIs associated to the final demo architecture.

The purpose of this document is to describe how the UC3 demonstrator (use case and all associated three sub-use cases) has been incrementally implemented, validated, and tested during Task 5.6 activities from M24 to M36.

1.1 D5.11 Updates vs D5.6 Quick Summary

An overview of the changes and updates to the previous version of this deliverable (i.e., D5.6) are provided below:

- Section 2 "Use Case Description" has been updated to include an additional 3rd sub scenario implemented during cycle 2 integration activities. This 3rd scenario will better demonstrate and support the usage of SEMIoTICS SPDI pattern driven infrastructure to implement three new dependability patterns specific to UC3.
- Section 3 "Testbed setup, integration and validation" has been updated to reflect final UC3 testbed demonstrator.
- Section 4 "Sub Use Case 1: Local vs Global anomalies Detection Demo" has been updated to add details about final integration status.
- Section 5 "Sub Use Case 2: Cloud Level Data Aggregation, Data Analytics and Visualization Demo" has been updated to reflect final integration status.
- Section 6 "Sub Use Case 3: Pattern-Based Sensing Dependability Monitoring" has been updated to reflect final integration status.
- A new section 7 has been added in order to report all UC3 related requirements and KPIs in a single entry point (in section 7.2) and the UC3 related requirements, including implementation status and testing activities (in section 7.1).
- Section 8 "Conclusions" has been further elaborate to include the lessons learnt during this UC3 integration activity, and more specifically the challenges we faced between cycle 1 and cycle 2 integration in order to deploy all the three described sub-use case scenarios.

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, a huge avalanche hits the Hotel Rigopiano in Farindola, near Pescara (central Italy), causing the death of 29 people that were present in the structure for winter holydays.

Those tragedies taught us about the importance of an automatic earthquakes events detection system. If the streak of earthquakes events leading to those disasters had been promptly notified to e.g., the National Protection Department, maybe the early evacuation procedures would have been started in time. This could have saved many precious lives.

Trying to address this scenario, UC3 in SEMIoTICS is aiming to provide an innovative technology for enabling distributed AI at both backend and field level, focusing for the latter in distributed low power IoT field devices. The innovation consists of partitioning the analytics usually centralized at backend/cloud server across several layers of the architecture, up to the sensor devices. This "close to the edge" distributed approach empowers sensors analytics by deploying optimized AI algorithms specifically designed for low power embedded micro-controllers.

UC3 within SEMIoTICS is also a horizontal and generic IoT solution, exploiting SEMIoTICS infrastructure (its pattern design framework and part of its components). UC3 scenario aims to provide an enabling technology that could be used to address a wider range of vertical specific scenarios in which event detection is needed.

The AI Sensing Platform in fact provides a disruptive and innovative technology for enabling AI in distributed low power IoT field devices and in particular low power embedded micro-controllers yet ensuring interoperability and ease of integration as a main concern. This platform is implementing a horizontal technology in the form of SEMIoTICS reusable components. This analytics leverages on optimized, yet powerful AI algorithms tailored for online training and unsupervised model learning on 32bits low-cost low power micro-controller units (MCU). These field device nodes, mapped as specifically developed field layer SEMIoTICS components are referred as "IHES Devices" and "IHES Supervisor" components. UC3 system enables highly scalable distributed intelligence thanks to 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.

On top of those technologies, an additional data aggregation layer for early avalanche warning has been developed in sub scenario2 to complete the scope of the UC3 demonstrator.

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

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

Sub Scenario 1 - Local Vs Global anomalies detection

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 detected at device level the system reports them to local LAN, where they are locally stored, and then, also propagated to the backend level, labelling them as either a local anomaly or a global one. In case of positive feedbacks (either local or global), triggering further layered analytics defined in sub scenario 2, the system reports the alert to owners and operators of the building as well as, in case, to the local authorities. Depending on the specific use case application this capability opens to further actions by integrating further solutions to the system. As an example, it would be possible, by instantiating further SEMIoTICS patterns to 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 also decide after check to issue an evacuation alarm. Further details about this UC3 sub scenario have been reported in section 4.

Sub Scenario 2 – Collaborative Edge-Cloud Avalanche Warning System

Additional outdoor sensors are utilized to track local temperatures and lighting / sun heating condition. Thereby, this sub use case relies on a collaborative edge-cloud data analytics approach. On the one hand, seismic events are detected by the local embedded intelligence of the IHES sensing units, given the accelerometer sensors' data. On the other hand, the temperature data is sent to the backend cloud. Then, to monitor the environmental trends further specific analytics based on linear regressors for the temperature rise warning, have been deployed at UC3 backend level, which allow to detect risky temperature events. Those risky temperature trends are triggered by this additional analytics UC3 service. Namely, they are triggered when the predicted temperature is above the parameter bounds provided by a simulated weather station agency. For instance, this can correspond to the information available on the remote DB of the local ARPA[1] ("*Agenzia Regionale Per l'Ambiente*") or any other openly available 3rd party weather service. Finally, an avalanche warning is raised to authorities when jointly a risky temperature is detected at the cloud and a global abnormal vibration has been reported at the IHES sensing units. Further details about this UC3 sub scenario have been reported in section 5.

Sub Scenario 3 – Pattern-Based Sensing Dependability Monitoring

Focusing on the dependability of the monitoring system deployed within UC3, additional pattern-based capabilities are deployed in the context of the third sub-scenario. This feature has been implemented thanks to the integration of pattern enablers developed within the SEMIoTICS framework. Considering the presence of redundant sensors in UC3, it is possible to monitor the parameters related to the fault tolerance and the reliability of the sensing system. This allows to trigger the associated reasoning of these two properties and the dependability property. These monitoring and reasoning capabilities are achieved through the deployment of a lightweight Pattern Engine at the field layer, integrating monitoring capabilities, and the support by

[1] ARPA https://www.arpalombardia.it (link accessed in December 2020)

instances of the Pattern Orchestrator and the SEMIoTICS GUI at the backend. Further details about this UC3 sub scenario are reported in section 6.

These three sub scenarios have been integrated incrementally in two macro steps that has been implemented in cycle 1 and cycle 2 task 5.6 activities: a first step (cycle 1) where the focus was 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 isolated validation done mainly by ST-I as domain experts, as part of UC3 T5.6 cycle 2 activities, the actual integration of this UC3 analytics in the SEMIoTICS framework has been achieved. Furthermore, this integration has been done by either developing specific new components (IHES LEA on MCU and IHES Supervisor and Local DB, plus the collaborative edge-cloud avalanche warnings system located in UC3 backend), or by adopting existing SEMIoTICS components (e.g., the pattern related ones) enlarging the UC3 testbed environment to dedicated field trials aiming at verification that the all the three sub use cases were realized consistently across the same UC3 testbed demo. As part of the field trial validation, proper end-to-end validation from Field Devices (IHES Sensing Units), to IoT gateway up to SEMIoTICS network and IHES backend/cloud deployed infrastructure where the UC3 App has been 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 distributed end-to-end architecture for detection and validation of critical events that does not require any or very limited prior information the environment to be monitored (IHES, Intelligent Heterogeneous Embedded Sensors + collaborative edge-cloud avalanche warning system).

This system is composed of several capable intelligent nodes (IHES Sensing Units) to obtain environmental data, analyze them, and detect anomalies using both neural networks and autoregressive models. These smart nodes are connected to a supervisor (IHES Supervisor) whose purpose is to coordinate at field level them to propagate relevant seismic events to other system architecture components located in other layers. Its main tasks are to aggregate the data and perform a validation of the local detected changes from any of the individual sensor units. These events are propagated to the UC3 backend level where they are further classified by additional analytics components to achieve the objectives. Scope of Task 5.6 is thus to provide an end to end layered intelligence demo scenario within SEMIoTICS ecosystem. 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 further data and services beyond the scenarios considered in UC3, 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.

Task 5.6 outcomes are complete both as regards contextualization and the KPIs (Key Performance Indicators) defined in SEMIoTICS, both for correctness, robustness, and reliability of the system with the newly introduced dependability and monitoring patterns depicted in sub scenario 3. 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 layered system, making IoT technologies more pervasive and widely adopted.

2.3 UC3 Scenario Stakeholders

The UC3 scenario provides a generic distributed infrastructure for sensing monitoring and anomalies / events detection. In order to provide an intuitive showcasing of the possibilities provided by with self-learning layered intelligence an intuitive use case scenario have been identified in D2.2. That is the UC3 system for earthquakes and temperature trends prediction monitoring for early avalanche detection alerting. In this scenario the main stakeholders/actors are:

1. **IoT stakeholders / system service provider operators:** offer IoT services for early avalanche system monitoring/prediction. The service types include: i) devices deployment and bootstrapping, ii) resource management, iii) physical safety and disaster mitigation, vi) adaptation policy dissemination.
2. **Cloud/data center:** aggregate data coming from field-level via network bridging. Design prediction process temperature trend analysis, to support the avalanche warning scenario. Send information to the stakeholders via UC3 App and pattern GUI for system/alerts monitoring.
3. **IHES IoT gateway:** receives raw-data / events from IHES devices. Correlate and coordinate connected units. Provide relevant information to the IHES backend via network level to the backend/cloud level for prediction and further data aggregation.
4. **IHES sensing units:** sense the physical world and obtain raw data (from either inertial or environmental sensors). Based on the streaming raw data, ESN based prediction, residual computation and statistical tests are done in order to identify potential anomalies mitigating false positive alerts. At this level the units are able to learn from incoming sensed data during a change-free stationary time window of a limited set of samples.

The avalanche warning infrastructure is monitored through a distributed AI layered sensing platform with specific algorithms deployed both at field and UC3 backend level. The sensed data (from accelerometer and environmental sensors) are processed locally at each single IHES IoT Device, as described in sub use case 1 and then only relevant events are transmitted to the upper IoT architecture level through the IoT gateway and MQTT underlying protocol. Next, the data flow can be transmitted via the network to the next upper level which is the data center capable of further analyzing the acquired information (i.e. the temperature trend and global data aggregator described in sub use case 2). Finally, the UC3 stakeholders (i.e. system operators and service vendor providers) can update/deploy the system policies accordingly if needed and disseminate it in a top-bottom fashion though the UC3 system, using the SPDI pattern infrastructure deployed in sub use case 3. Moreover, UC3 stakeholders such as IoT vendors or environmental monitoring system providers can exploit the acquired data and generated events in order e.g. to replace failed or malfunctioned IoT equipment which is critical for the UC3 system monitoring/prediction capabilities (see how this is achieved in UC3 thanks to the deployed dependability patterns in sub use case 3 scenario).

The following figure details the sequence of events associated to the considered avalanche warning scenario (e.g. UC3 layered intelligence, system adaptation policies of IHES units for system reconfiguration etc.).

The UC3 system detects and reports the incident automatically thanks to the UC3 app based on openHAB to the IHES IoT operator, and also guarantee the consistency of overall dependability patterns and proper system functionalities consistency.

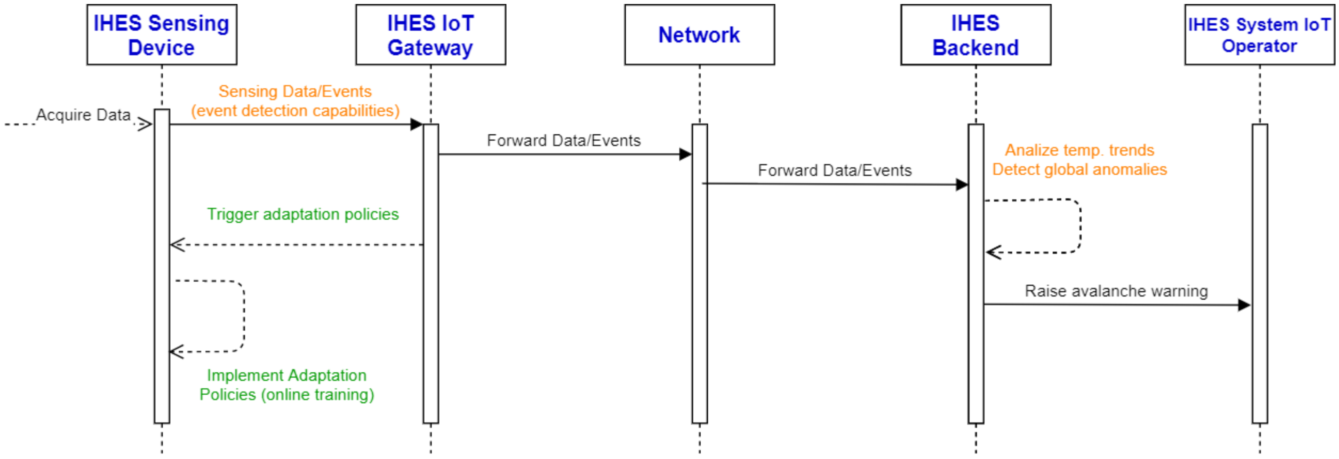


FIGURE 1. UC3 SCENARIO SEQUENCE EVENTS DIAGRAM

2.3.1 TRIGGERING EVENT, PRECONDITIONS, ASSUMPTIONS, SUCCESS CRITERIA/EXPECTED OUTCOME

The IHES sensing units are triggered and collect/process in real-time new measurements corresponding to the qualitative (anomalies/events not conforming to a self-learned nominal state) as well as quantitative characteristics (raw data readings) of the IHES monitoring infrastructure. The collected measurements are locally processed by each single device and transmitted to the IHES IoT gateway where further computations and local monitoring on the state of the system is performed. The outcome of the local computations is propagated to the IHES backend where the provided information are further aggregated and processed (layered intelligence pay a key role here) to assess the potential risk for an avalanche in the monitored area. At backend level a decision is made and eventually an alert is raised to the system operators for further analysis. The expected long-term outcome is to anticipate/report the risk of an avalanche to allow eventually the evacuation of the area or setup any needed emergency plan in advance.

Assumptions: in this scenario, we assume that IHES backend components are already configured and running, i.e. users can interact with pattern GUI and UC3 App. We also assume network connectivity between IHES sensors and backend/cloud platform to be available. This may include networks using NFV or SDN technology, but that is transparent to this use case. Also at field-level there is a IHES IoT gateway up and running with at least six IHES units connected monitoring both environmental (i.e. temperature) and inertial (i.e. vibrations) sensors. Also we assume that upon request or when switched-on the IHES devices are able to learn from the environment (live acquired data) the normality condition (no vibrations and stable temperature). Finally all dependability patterns are verified and no patterns violations are reported by the GUI pattern located at IHES backend.

Triggering Event: we have mainly two triggering events that leads to the reporting of an avalanche alert: on the field device level, abnormal vibrations are reported by more than 50% of connected correlated IHES units (i.e. a global change has been detected. This is realized by the sub use case 1. On the backend level, this use case is triggered when the global aggregation and avalanche warning system receives from the IHES IoT gateway a notification via MQTT for a global change on inertial data and from the readings of temperature sensors and abnormal trend (abnormal heating) is observed as well. These two triggers raise immediately an avalanche warning to the system operation via UC3 App.

Assumptions:

- IHES field components and backend are up and running. Network connection is up and running
- Sensors have completed their local learning and they are in operative phase (anomalies detection mode)
- No dependability patterns are violated (all correlated sensing units are working and are providing consistent data readings).
- Global data aggregation and temperature trend analyzer has been trained on past trends and is ready to report any abnormal temperature trend from sensor readings

Success criteria/expected outcome: This use case is successful when in live condition, at a given instant, the running UC3 system is able to notify the IHES IoT system operator and avalanche warning via UC3 app, when:

1. A global change associated to abnormal vibrations is reported by correlated sensing units at field level (sub use case 1)
2. An anormal temperature trend is detected by the global data aggregation component at IHES backend (sub use case 3)
3. No dependability patterns are detected nor reported by pattern GUI at UC3 backend (sub use case 3)

2.4 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.

In 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 Figure 2. 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. 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".

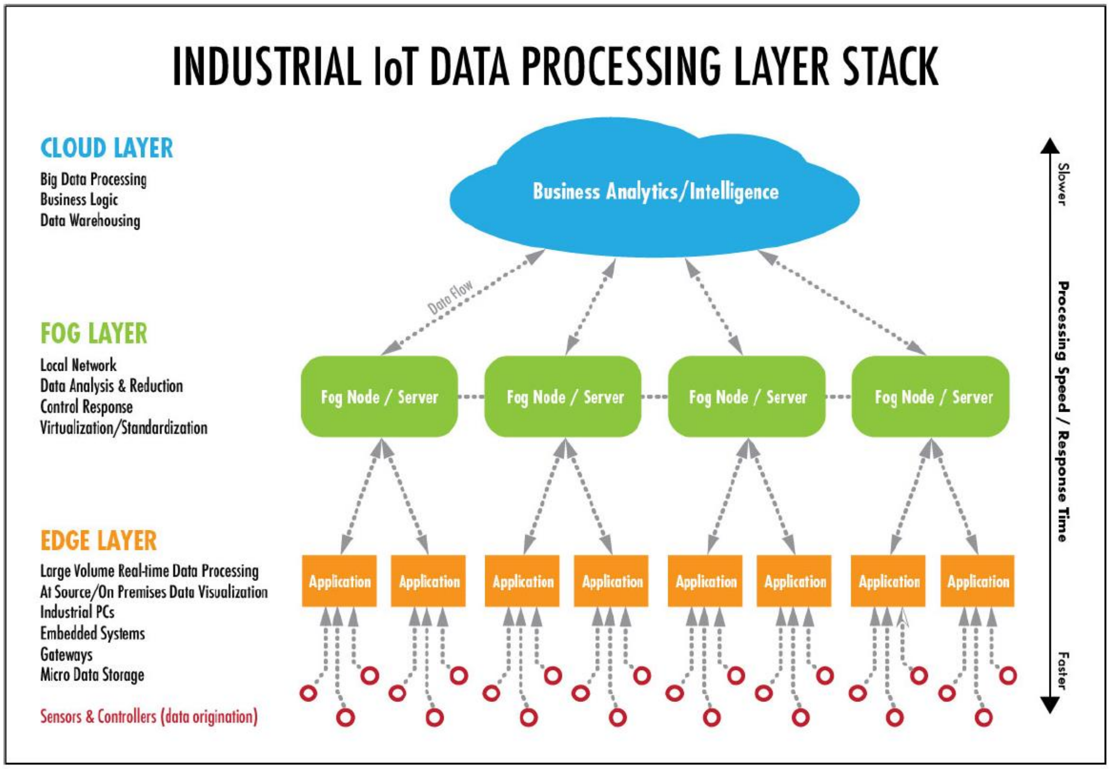

INDUSTRIAL IoT DATA PROCESSING LAYER STACK
CLOUD LAYER
Big Data Processing
Business Logic
Data Warehousing
Business Analytics/Intelligence
Data Flow
FOG LAYER
Local Network
Data Analysis & Reduction
Control Response
Virtualization/Standardization
Fog Node / Server
Fog Node / Server
Fog Node / Server
Fog Node / Server
EDGE LAYER
Large Volume Real-time Data Processing
At Source/On Premises Data Visualization
Industrial PCs
Embedded Systems
Gateways
Micro Data Storage
Application
Sensors & Controllers (data origination)
Slower
Processing Speed / Response Time
Faster

FIGURE 2. DATA ANALYTICS COMPUTING APPROACHES[2]

As can be seen in above figure, "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 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 3. Moreover in Figure 4, 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.

[2] Courtesy from Winsystems

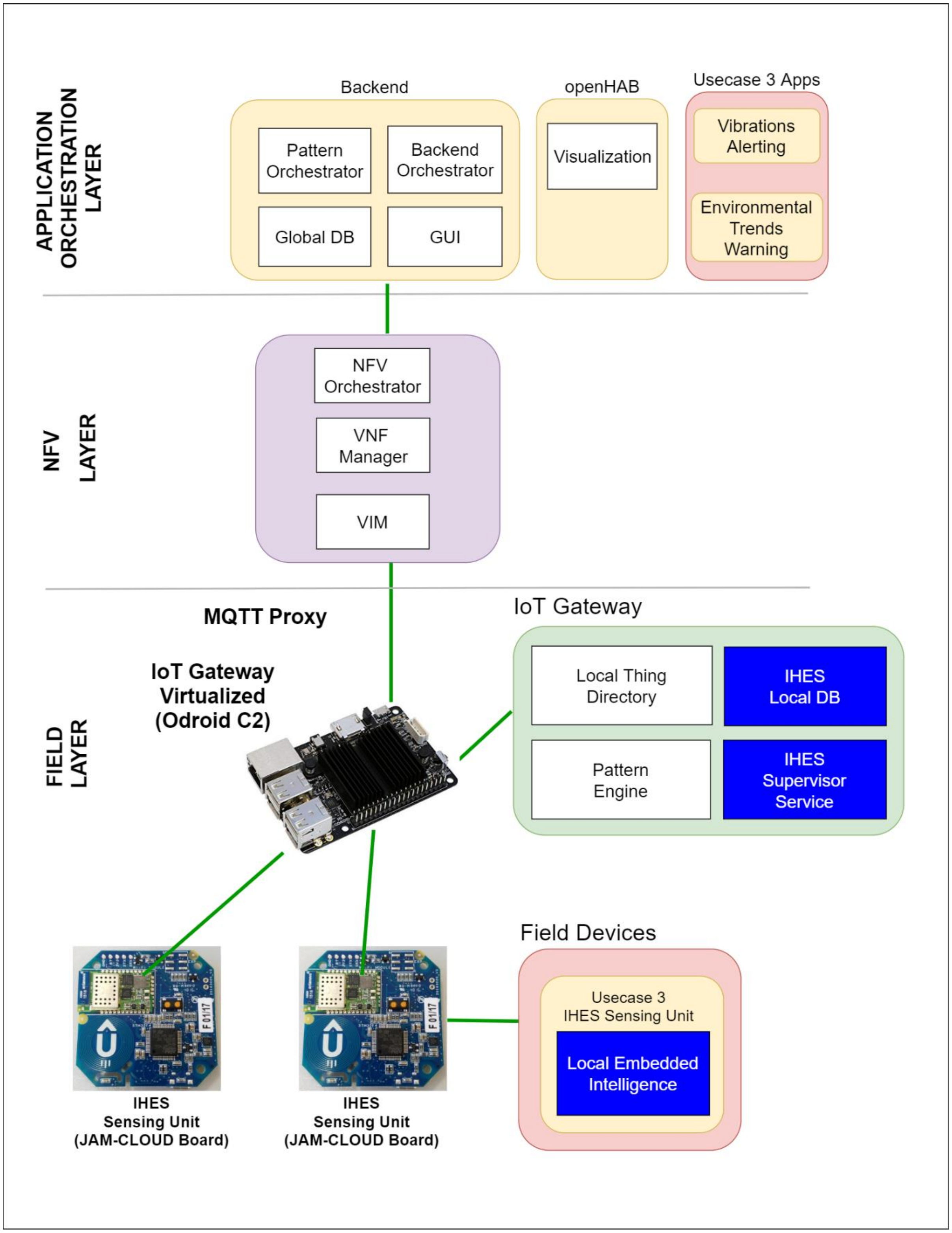


FIGURE 3. UC3 SYSTEM VS COMPONENTS MAPPING

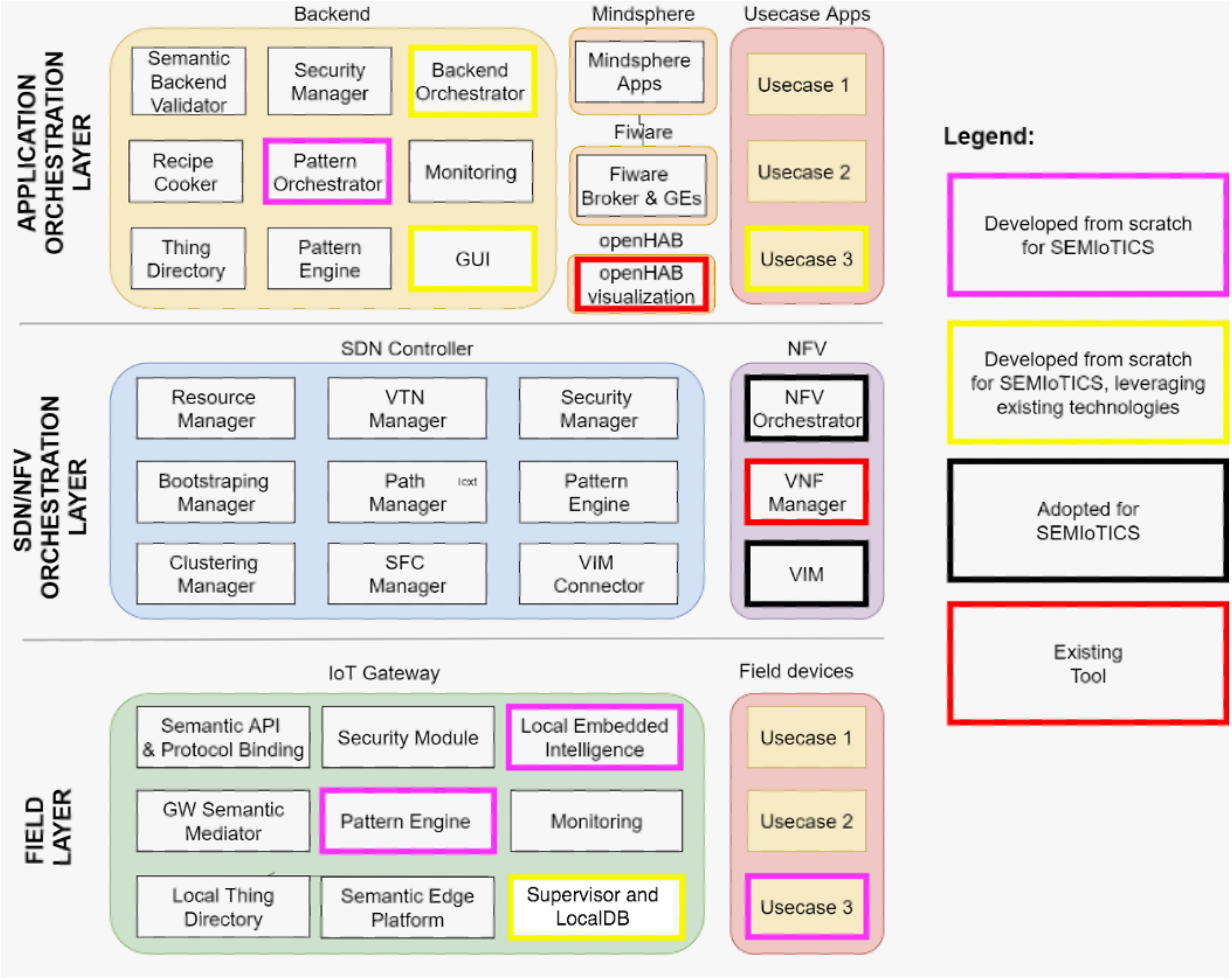


FIGURE 4. UC3 SEMIOTICS COMPONENTS STATIC MAPPING

As stated in general introduction we have consolidated the complete demonstrator along task 5.6 activities by following an incremental approach from bottom to top, and we derived intuitive, self-contained storylines and sub-use cases to showcase the incremental functionalities available.

During task 5.6 activities we realize and implement three incremental sub use cases (from sub use case 1 to sub use case 3) that had been incrementally integrated into UC3 testbed the components highlighted on Figure 4 as part of either cycle 1 (sub use case 1) or cycle 2 activities (sub use cases 2 and 3 and the final validation of the whole UC3 testbed). A video demo has been prepared to showcase the achievements done in task 5.6, and more specifically the description of the UC3 testbed, the presentation of all three UC3 sub-use case scenarios and finally three live demos showing the deployed UC3 testbed at work.

In more details, 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. In this demo scenario (part of it has been already anticipated during mid-term review) it is shown the complete local embedded analytics components deployed at field level at work. The main two goals of this scenario is to show a collaborative self-learning system composed by six connected units (three units monitoring inertial data – vibrations, three other units environmental data). At system startup the six units train themselves in a change-free condition to learn current environmental condition. After the training they enters in operative mode to report any anomaly from this expected state. The Supervisor and LocalDB Service monitors the connected units and is able to classify events as either local to the unit or global observing concurrent anomalies reported by the units. The global / local classification is exploited in sub use case 2 to provide further data aggregation and processing at backend level. The second goal of use case 1 demo is to show the interaction with the UC3 App based on OpenHAB to visualize the status of a UC3 field device system and its connected units. Sub use case 2 focuses on how to bring and further process the results of the data analytics computed in sub use case 1 to convey them to the upper level of the infrastructure (i.e., the UC3 backend), and the scalable infrastructure needed for it in order to open-up/scale to backend added value services. In this demo setup the anomalies from inertial units (i.e. abnormal vibrations) are used together with an analysis on temperature trends in order to estimate the risk of an avalanche, and in case promptly report an alert to operators thanks to the UC3 App. Finally sub use case 3 will deploy end-to-end specific UC3 Dependability patterns exploiting SEMIoTICS ecosystem, by instantiating and adapting its pattern related components, to allow a full integration into SEMIoTICS architecture and a better resilience to system/sensors faults. In particular in this demo it will be shown how it is possible to exploit system redundancy to detect abnormal readings from a sensing unit or detect the malfunctioning of a unit caused by a network disconnection or HW malfunctioning. This will be showcased using SPDI SEMIoTICS pattern infrastructure through visualization thanks to the pattern GUI, and in particular we will the violation of an instantiated pattern, and its restoration to normal conditions by disconnecting / switching off specific units to simulate a malfunctioning of the system.

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. During cycle 2 activities period, we extended the same approach in the sub-use cases 2 and 3 by incrementally adding all the missing features, SW artifacts and missing components, and integrating them under the same UC3 testbed. As a result, a complete end-to-end demonstrator has been provided, ensuring the correct integration from the field layer to the network and finally the backend where the UC3 GUI visualization based on openHAB IoT platform and the global data aggregation and 3rd party services queries has been implemented. The UC3 testbed has been 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 backend server with 6-cores and 32 GB RAM hosts the NFV MANO framework and Network services. To increase portability, the same server 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 and acting as a gateway hypervisor as well.
- A legacy gateway and sensors
- Field layer devices from ST.

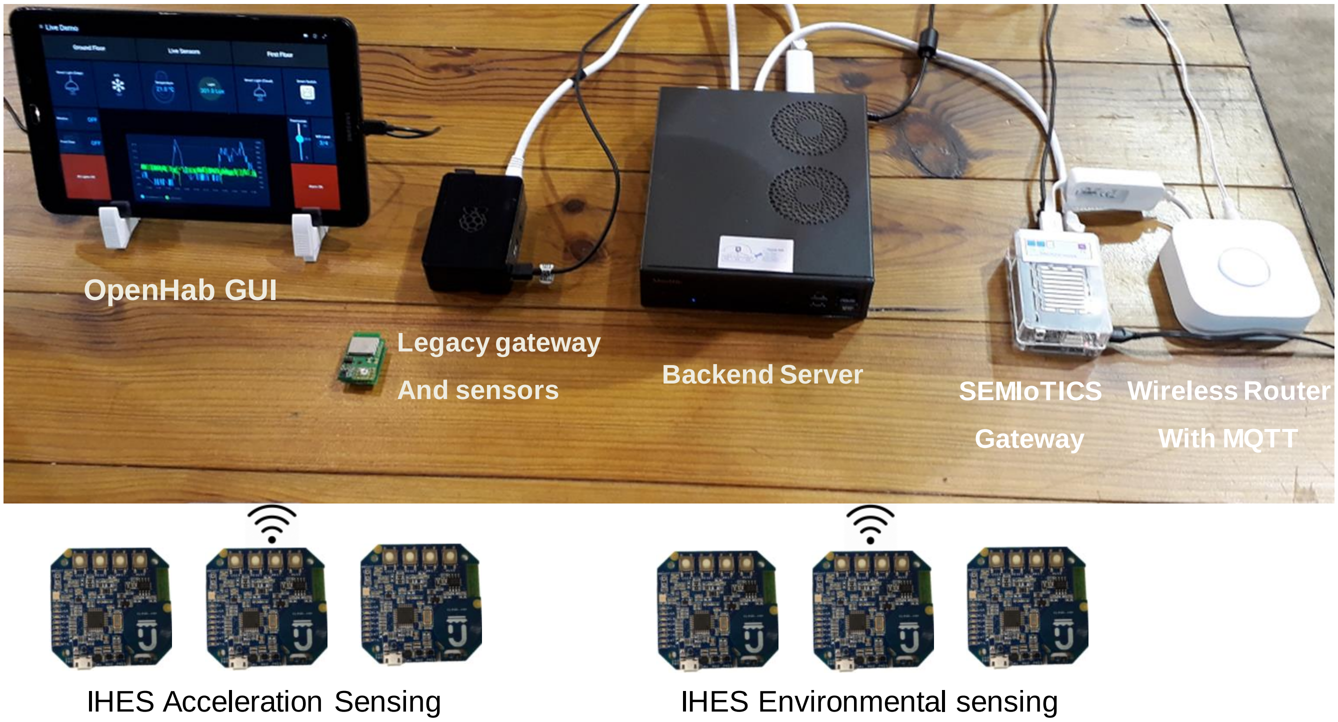


FIGURE 5. PHYSICAL INFRASTRUCTURE OF UC3 TESTBED

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
```

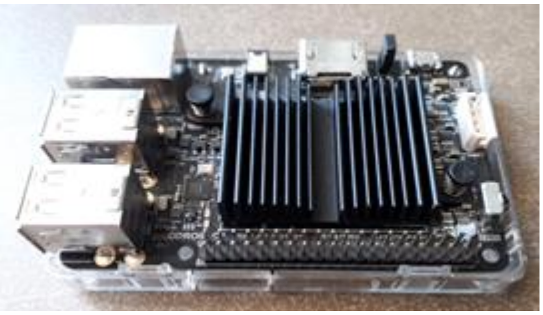


FIGURE 6. VIRTUALIZED IOT GATEWAY

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.

Finally, as part of the UC3 Second Cycle the SEMIoTICS pattern engine was deployed at the IoT gateway, with support added for the reliability pattern. More specifically, the pattern engine keeps track of MQTT messages transmitted by IHES sensors with a timeout value which is triggered when no heartbeat messages are received for a user-defined time threshold. Moreover, the pattern engine can keep track of sensor value disagreement, such that sensor redundancy scenarios can be supported.

3.1.1 BACKEND

As part of the UC3 second cycle, the SEMIoTICS Pattern Orchestrator (PO) and GUIHub were deployed at the UC3 Backend server, while the openHAB IoT platform was already deployed during the First Cycle. The SEMIoTICS PO related to the Field Pattern Engine through TCP port 7080, and the GUIHub through TCP port 9080. Thus, the latter can visualize the status of the Dependability pattern, pulling status updates from the PO in real time. Moreover, the BLS GUI was updated to host the openHAB HabPanel GUI inside the GUIHub module. The following BLS GUI components were deployed as Docker containers at the UC3 backend server:

- **GUI frontend**: This is an Angular app that serves as Graphical User Interface.
- **GUI backend**: This is Spring Boot app that is responsible for all GUI logic and operations. It receives requests from GUI frontend and communicates with integrated components e.g., Thing Directory, Pattern Orchestrator.
- **GUI database**: This is a PostgreSQL database that storages all GUI data.

Moreover, a UC3 recipe for dependability was authored which sets-up a redundancy scenario for UC3 environmental sensors. In this scenario, at least two sensors must be online and transmit measurements of the same value (e.g., temperature) that should not diverge more than a maximum user-defined threshold. Finally, during second cycle a dedicated cloud-based InfluxDB instance was deployed at the Backend to aggregate sensor values and events to be visualized by the openHAB HabPanel GUI. The deployment of a global DB was deemed necessary to free-up resources from the IoT gateway which hosts the local DB and decrease latency in serving sensor values to the visualization sub-system which is also hosted at the Backend. It must be noted that sensor data visualization and aggregation functions are deployed within Virtual Network Functions (i.e., VNFs) hosted by the UC3 NFVI as detailed in Section 3.2.

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. It must be noted that the UC3 Backend server also serves as a Compute Node which provides "*scalable, on demand, self-service access to compute resources*" via the OpenStack Nova hypervisor service and its Compute APIs. 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 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 via a global InfluxDB time series database, allowing post-processing and analysis of sensor data.

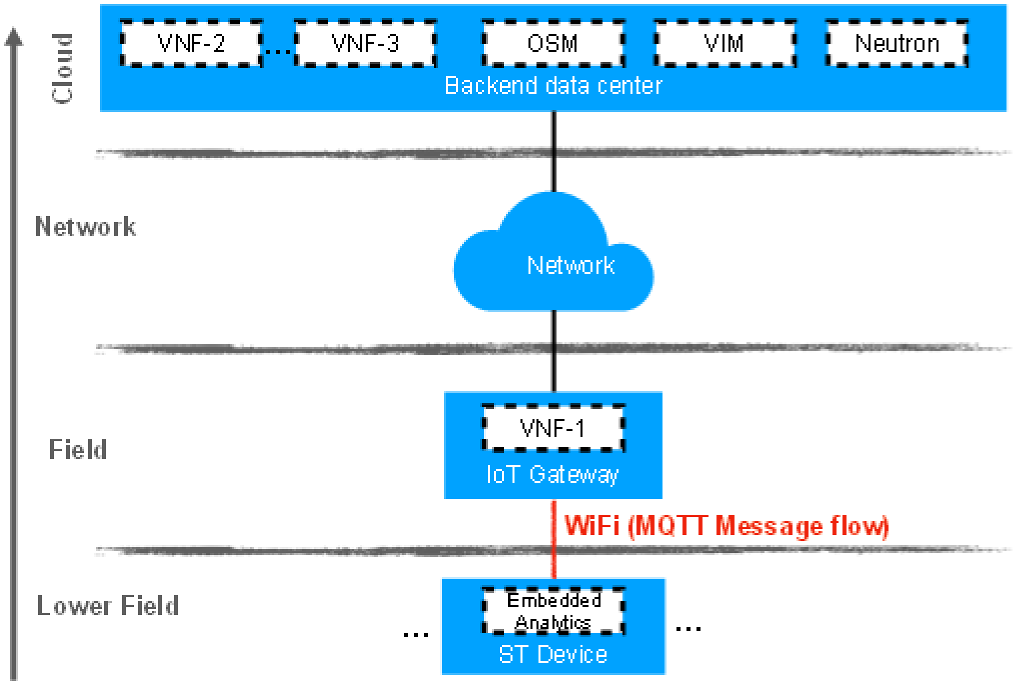
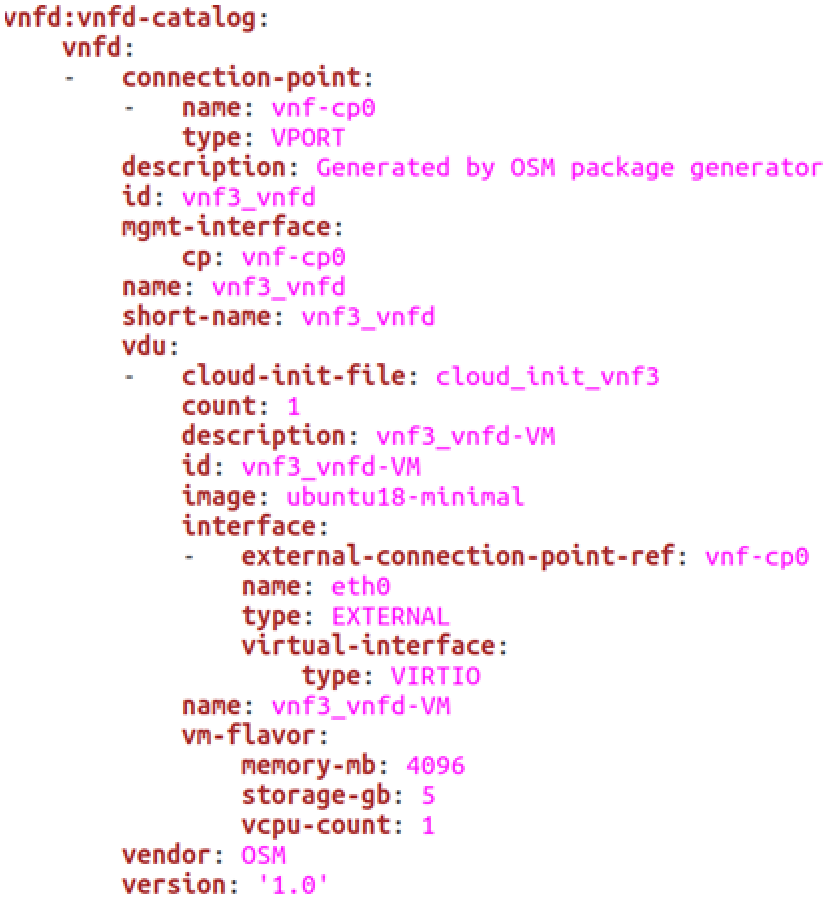


FIGURE 7. UC3 NFVI INCLUDING THE VNF1, VNF2 AND VNF3 AND NFV MANO

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). 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 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 8. 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.

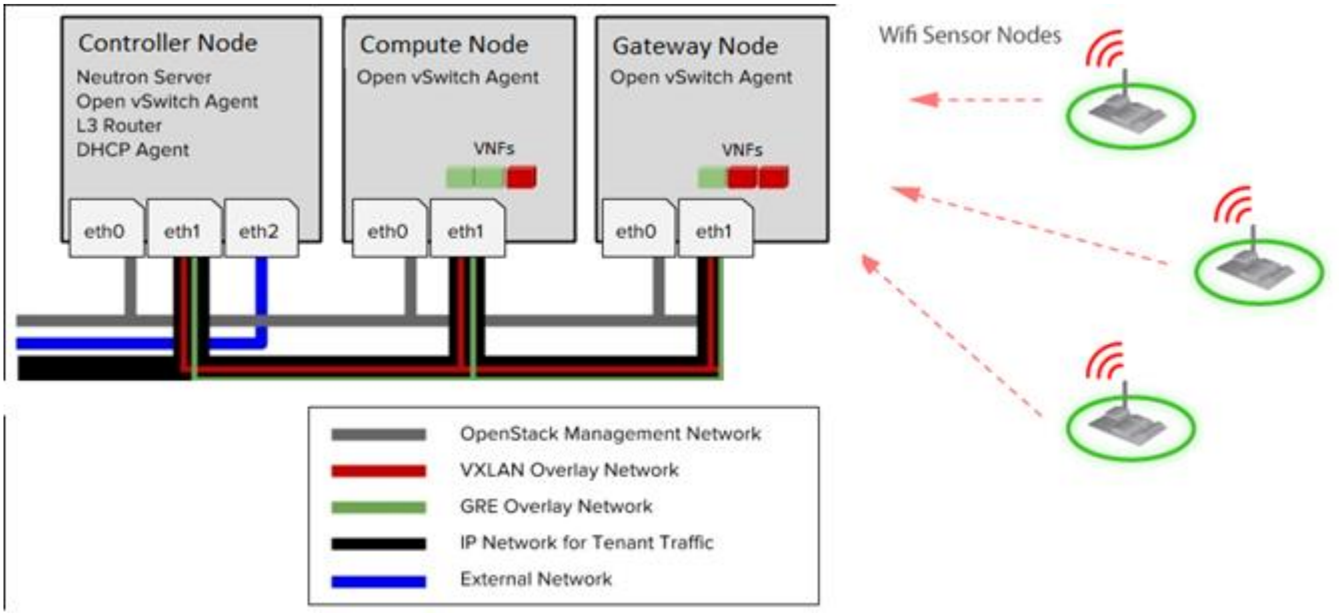


FIGURE 9. UC3 NETWORKING VIA OPENSTACK NEUTR

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. As a result of this, we provided and deployed the complete field level UC3 analytics into UC3 testbed described in section 3, in order to enable the final SEMIoTICS framework integrated UC3 demonstrator. Furthermore, 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 threshold-free analytics in order to detect any anomaly from this acquired nominal state. On top of this it is possible, by using several IHES device 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 belongs to a single node (local change) or has been observed by many (partial change) if not all nodes (global change).

The scope of this sub use case 1 is to provide an integrated infrastructure at field level (devices + gateway) to realize these capabilities. The scenario involves the need for three more Field Level SEMIoTICS components, the instantiation of the MQTT infrastructure needed for message exchange, and the NFV virtualization infrastructure for a reliable, affordable mapping of the UC3 testbed 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, 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 detail 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 laboratory testbed.
- Visualize real-time sensor values, events, as well as global and local changes events though openHAB UC3 App.
- Cross-compile the IHES Supervisor service components for the Odroid C2 virtualized gateway embedded board, and deploy within VNF-1, since 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 to support functionalities described in sub use case scenarios 2 and 3 by designing all components interoperability and implementing dynamic flows architecture as reported in D2.5 – UC3 architecture section.

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 component on MCU (see section 4.3.1).
- Supervisor and Local DB components on gateway (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 – D4.8, D4.11 and D4.13 for details about these two SEMIoTICS components and their interaction with other 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 protocol, it is possible to expose them additionally as standard WoT device node in order to be discoverable as simple sensing units by other systems / components not part of the specific UC3 architecture. This ensures 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 D4.11 subsection 4.3.3.

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 10 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.

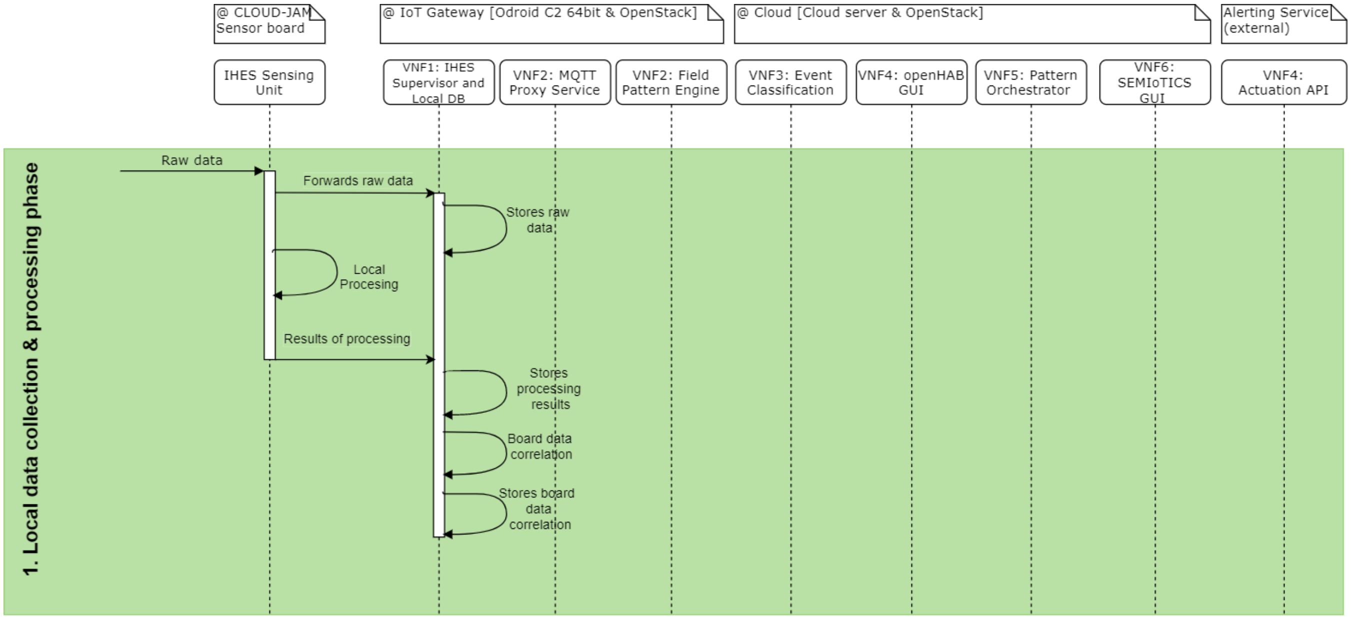

@ CLOUD-JAM Sensor board
@ IoT Gateway [Odroid C2 64bit & OpenStack]
@ Cloud [Cloud server & OpenStack]
Alerting Service (external)
IHES Sensing Unit
VNF1: IHES Supervisor and Local DB
VNF2: MQTT Proxy Service
VNF2: Field Pattern Engine
VNF3: Event Classification
VNF4: openHAB GUI
VNF5: Pattern Orchestrator
VNF6: SEMIoTICS GUI
VNF4: Actuation API
1. Local data collection & processing phase
Raw data
Forwards raw data
Stores raw data
Local Procesing
Results of processing
Stores processing results
Board data correlation
Stores board data correlation

FIGURE 10. UC3 FL MESSAGE FLOW: SENSING UNITS TO IHES SUPERVISOR

On Figure 11 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 11, 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 to realize the complete flow involving the SEMIoTICS infrastructure as well 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.

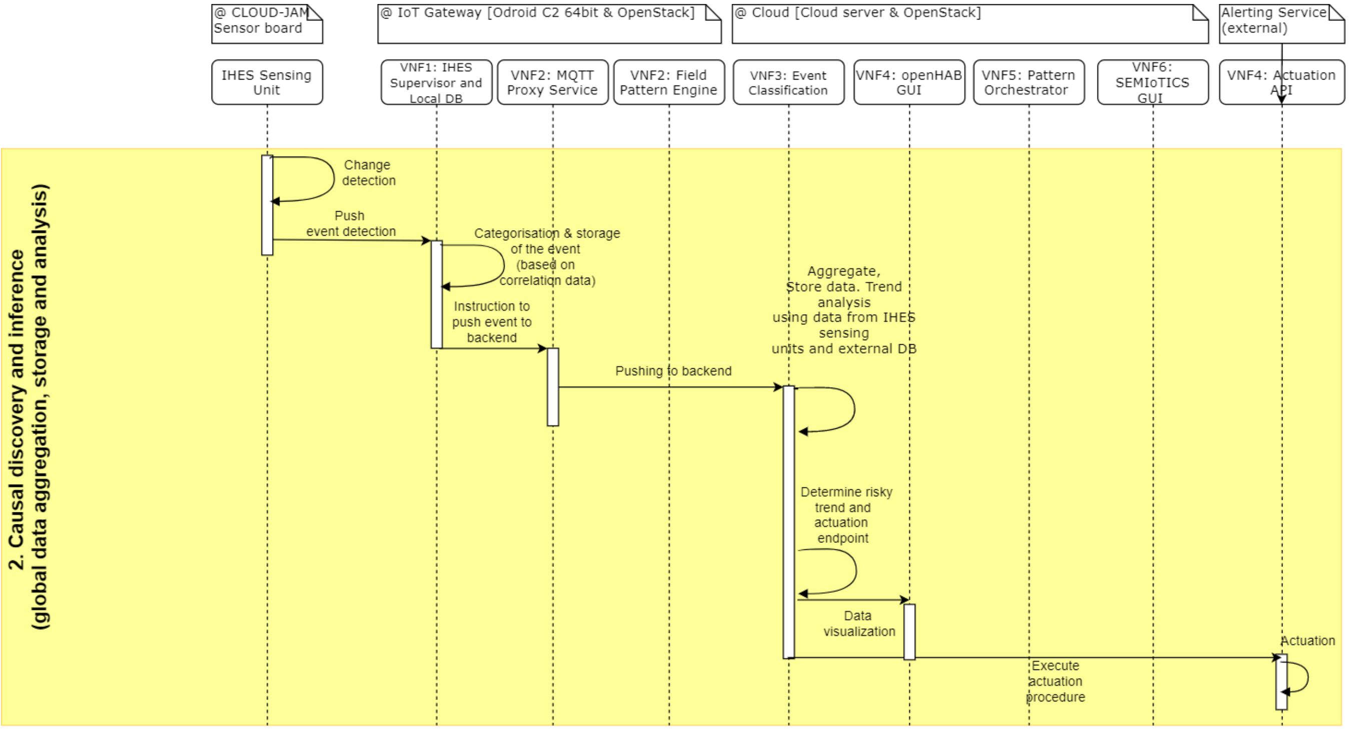

@ CLOUD-JAM Sensor board
@ IoT Gateway [Odroid C2 64bit & OpenStack]
@ Cloud [Cloud server & OpenStack]
Alerting Service (external)
IHES Sensing Unit
VNF1: IHES Supervisor and Local DB
VNF2: MQTT Proxy Service
VNF2: Field Pattern Engine
VNF3: Event Classification
VNF4: openHAB GUI
VNF5: Pattern Orchestrator
VNF6: SEMIoTICS GUI
VNF4: Actuation API
2. Causal discovery and inference (global data aggregation, storage and analysis)
Change detection
Push event detection
Categorisation & storage of the event (based on correlation data)
Instruction to push event to backend
Pushing to backend
Aggregate, Store data. Trend analysis using data from IHES sensing units and external DB
Determine risky trend and actuation endpoint
Data visualization
Actuation
Execute actuation procedure

FIGURE 11. LOCAL VS GLOBAL SCENARIO, FIELD LAYER MESSAGE FLOW

4.3.1 IHES LOCAL EMBEDDED ANALYTICS (ON MCU)

This component has been developed within task 4.3 and it has been integrated on UC3 testbed as a dedicated firmware on a STM32 microcontroller board whose package and libraries are reported in Figure 12. 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.

As a result, 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 a dedicated 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 13Figure 12). 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 sub use case 1 IoT gateway.

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.

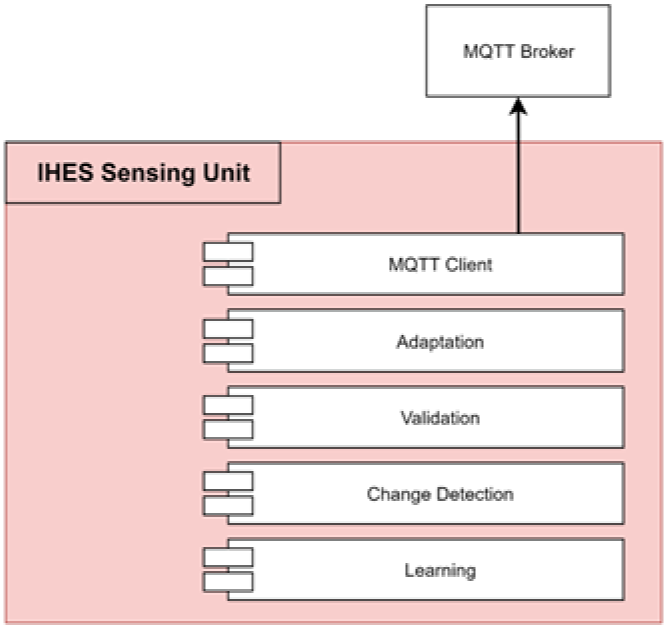


FIGURE 12. IHES SENSING NODE SW ARCHITECTURE: ANALYTICS MQTT JSON FLOW

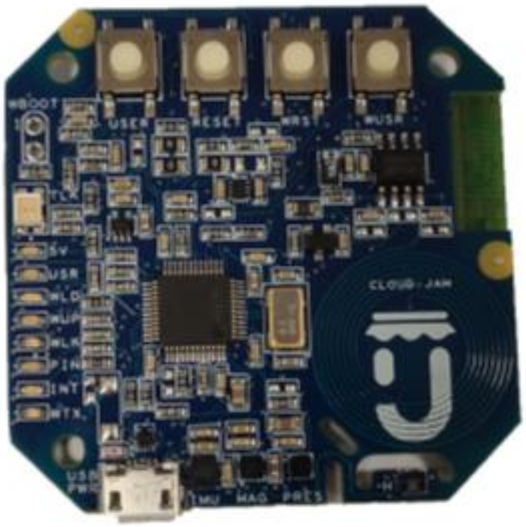


FIGURE 13. IHES LEA ON CLOUD-JAM STM32 BOARD

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 though underlying MQTT protocol and node-RED 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 to implement self-adaptation strategies of the 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 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 14 designed to manage node scalability and seamlessly connection / disconnection to the system.
  On Figure 15 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.

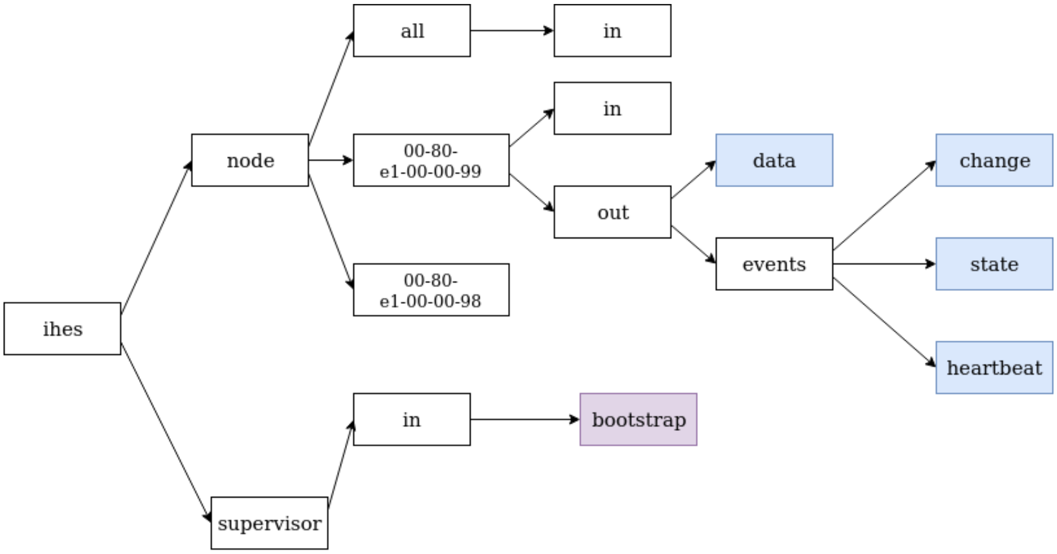


FIGURE 14. UC3 PROTOCOL MQTT TOPICS HIERARCHY

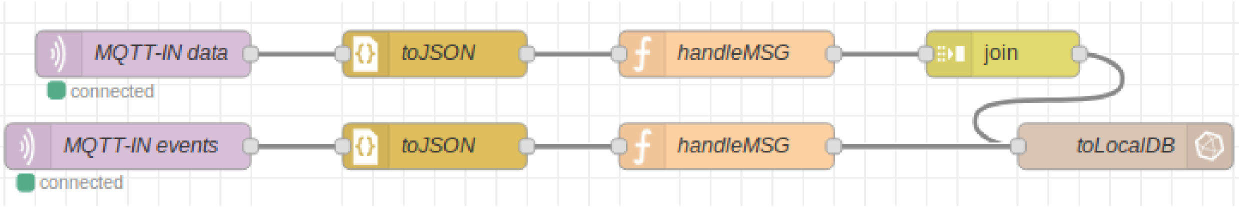


FIGURE 15. NODE-RED MQTT DATAFLOW

4.3.3 IHES LOCAL DB

The IHES local database, is implemented on top of the InfluxDB stack. InfluxDB is a SQL-like time-series database supporting a whole ecosystem for data processing, storage, and data aggregation. All messages sent on local LAN by the IHES sensing units are stored into this database. InfluxDB offers a complete set of standard 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, 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

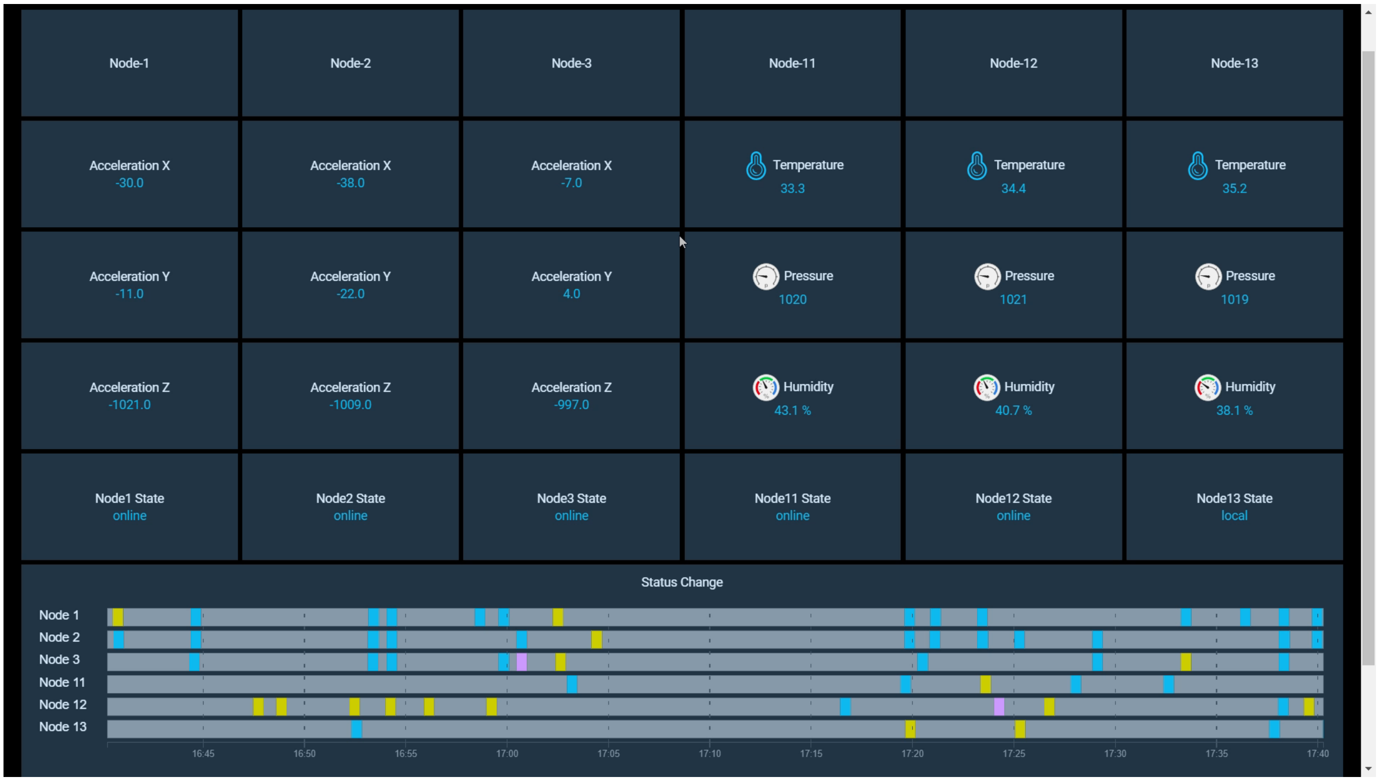
The open-sourced openHAB platform is a flexible, open-source, technology-agnostic automation platform that can integrate a multitude of devices and systems. To achieve this, openHAB segments and compartmentalizes certain functions and operations. openHAB uses Apache Karaf to create an Open Services Gateway initiative (OSGi) runtime environment. Jetty is used as the HTTP server, which implements the Dashboard and Management GUI and 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. More specifically, during the UC3 2nd cycle implementation, a HabPanel Timeline widget was leveraged to visualize global, partial, and local changes. These were represented as purple, yellow, and blue bars respectively.

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 field level dedicated testbed. 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. During this validation great care has been devoted in ensuring that all the technologies characterizing the UC3 scenario, validated in the initial ST proprietary testbed, has been 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 a complete Field Level 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 has been used in cycle 2 to deliver a monitoring GUI of the whole system. The final version of this UC3 app, and a screenshot is reported in Figure 16. 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 anomalies reported by the connected node plus the raw data readings and specific events at each single node. A bar chart located in the bottom 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, in yellow the partial ones, and finally in purple color the global ones. This preliminary classification of the type of abnormal changes done at field device level, that is the main goal of sub scenario 1 presented in section 4 is the key enabler for the demo and in particular of the sub use case 2 described in section 5, to detect and raise potential avalanche warning alerts. In more detail, the avalanche alert is raised only when global changes are reported by the field devices analytics thanks to the analysis on correlated devices done in IHES Supervisor, and when also abnormal temperature trends have been identified by the additional analytics deployed at UC3 backend. This has been designed with the intent to limit as much as possible false positive alerts in the UC3 system.



Node-1	Node-2	Node-3	Node-11	Node-12	Node-13
Acceleration X -30.0	Acceleration X -38.0	Acceleration X -7.0	Temperature 33.3	Temperature 34.4	Temperature 35.2
Acceleration Y -11.0	Acceleration Y -22.0	Acceleration Y 4.0	Pressure 1020	Pressure 1021	Pressure 1019
Acceleration Z -1021.0	Acceleration Z -1009.0	Acceleration Z -997.0	Humidity 43.1 %	Humidity 40.7 %	Humidity 38.1 %
Node1 State online	Node2 State online	Node3 State online	Node11 State online	Node12 State online	Node13 State local

FIGURE 16. UC3 OPENHAB GUI APPLICATION

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

5.1 Story line: Collaborative edge-cloud analytics for early avalanche warning.

This sub use case is in line with the UC3's scenario 2, "Causal discovery and inference", described in section 2.1. Therefore, this sub use case is motivated by the Rigopiano avalanche accident, where both the rise of temperature along with seismic events provoked an avalanche that killed several persons. The aim is to alert proactively on the risk of such avalanches by leveraging the SEMIoTICS technology, as it will be described below. To this end, herein we provide a global data aggregation, data analytics and storage at the cloud level, which is combined with the local intelligence at the field devices. Therefore, in terms of data analytics, herein we leverage the benefits of both the intelligence implemented at the field devices and at the cloud level. Thus, a so-called collaborative edge-cloud approach is proposed herein. This is complemented by a visualization tool that displays the stored data or a processing of this data. The functional block diagram of the scenario is described in Figure 17.

The IHES sensing units by ST are the IoT field devices that gather the environmental data needed to detect the risk of avalanches. Namely, on the one hand they obtain temperature measurements. On the other hand, the accelerometer sensors obtain measurements related to the vibration of the surface where they are placed. The local embedded intelligence at the field devices allows to detect seismic events by relying on the accelerometer sensors' data. The supervisor at the IoT GW controls the training of the embedded local intelligence algorithms. The MQTT protocol is used to transmit sensors' data to the cloud. Thereby, the global data base at the cloud 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) or weather agencies. Thereby, the monitoring of the IHES sensing units can be complemented with atmospheric trends. In this regard, herein we will simulate a weather agency and we will consider that it provides some environmental parameters bounds, related to the temperature. When these parameters are surpassed an early avalanche risk warning can be triggered, as it will be explained in the validation and implementation section.

Besides, the global data aggregation or storage, the cloud also provides data analytics and visualization functionalities. openHAB allows the visualization of the sensors' data and the output of the use case 3 app. In fact, the use case 3 app is the cornerstone of this sub use case, as it implements the data analytics to infer the risk of an avalanche. On the one hand, it implements a temperature predictor, by leveraging the available temperature measurements provided by the IHES field devices. The temperature prediction is compared to the weather agency parameters bounds and if they are surpassed a risky temperature event is triggered. On the other hand, the use case 3 app at the cloud considers the seismic events provided by the local embedded intelligence of the IHES sensing units. Finally, when there is both a risky temperature event and a seismic event, the use case 3 app triggers the early avalanche risk warning. Thereby, a collaborative edge-cloud intelligence is implemented.

Finally, it is worth mentioning that the network function virtualization (NFV) technology controls the virtualization of the computational and storage resources of the cloud, yielding the so called Network Function Virtualization Infrastructure (NFVI). Thereby, a set of Virtual Network Function (VNFs) located at the cloud level can be deployed on top of the NFVI in a flexible, scalable, and isolated manner. This is the case of our use case 3 app, which implements the avalanche risk alert.

In summary, the collaborative-edge cloud approach proposed herein for early avalanche risk warning has the next benefits. It allows the aggregation of local information available at the GW level in a global entity, which is made available for assessment and post-processing by analysts. Thereby, the visualization and data analytics at the cloud 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 Use case 3 app also provides an eastbound interface towards the visualization tool provided by the openHAB to display the aggregated data. For instance, the historic measurements of temperature, the predicted temperature and the avalanche warning can be displayed in openHAB. 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.

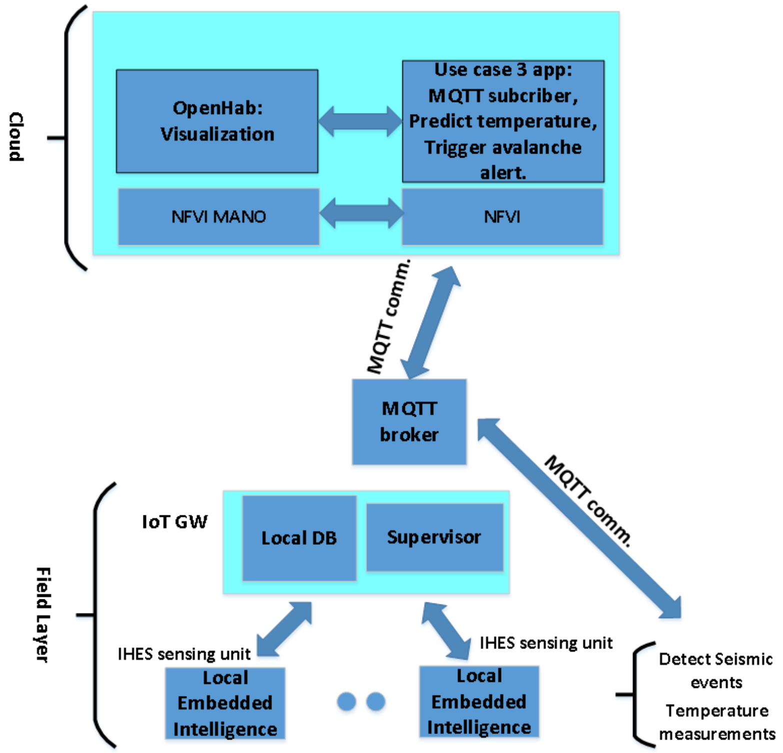


FIGURE 17. FUNCTIONAL BLOCK DIAGRAM OF THE SUB USE CASE 2

5.2 Scope and objectives

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

- Implement an early avalanche warning application that leverages the SEMIoTICS architecture. To this end, rely on a collaborative edge-cloud data analytics approach. The embedded local intelligence at the field devices will detect seismic events. The cloud will implement a temperature predictor given the temperature measurements gathered by the field devices. The early avalanche warning relies on both the seismic events inferred at the edge and the risky temperature events inferred at the cloud.
- Implement in a global entity, at the cloud level, the capability to receive and to aggregate the time series of data transmitted from the IHES sensing units via MQTT. This includes temperature data measurements and seismic events generated by the IHES sensing units.
- Leverage the openHAB visualization tool to display in real-time the temperature data measurements available at the cloud from different IHES sensing units. Also, use openHAB to display in real-time the temperature prediction inferred at the cloud level and the early avalanche warning.
- The data aggregation, visualization and data analytics at the cloud level are implemented within the framework of NFV.
- 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.

5.3 Interaction with SEMIoTICS framework and components

In Figure 18 we display the functional blocks of the sub use case 2, introduced in section 5.1, within the context of the SEMIoTICS architecture. Given this figure, below we explain in more detail the mapping and interaction between the functional blocks of this sub use case and the SEMIoTICS architecture.

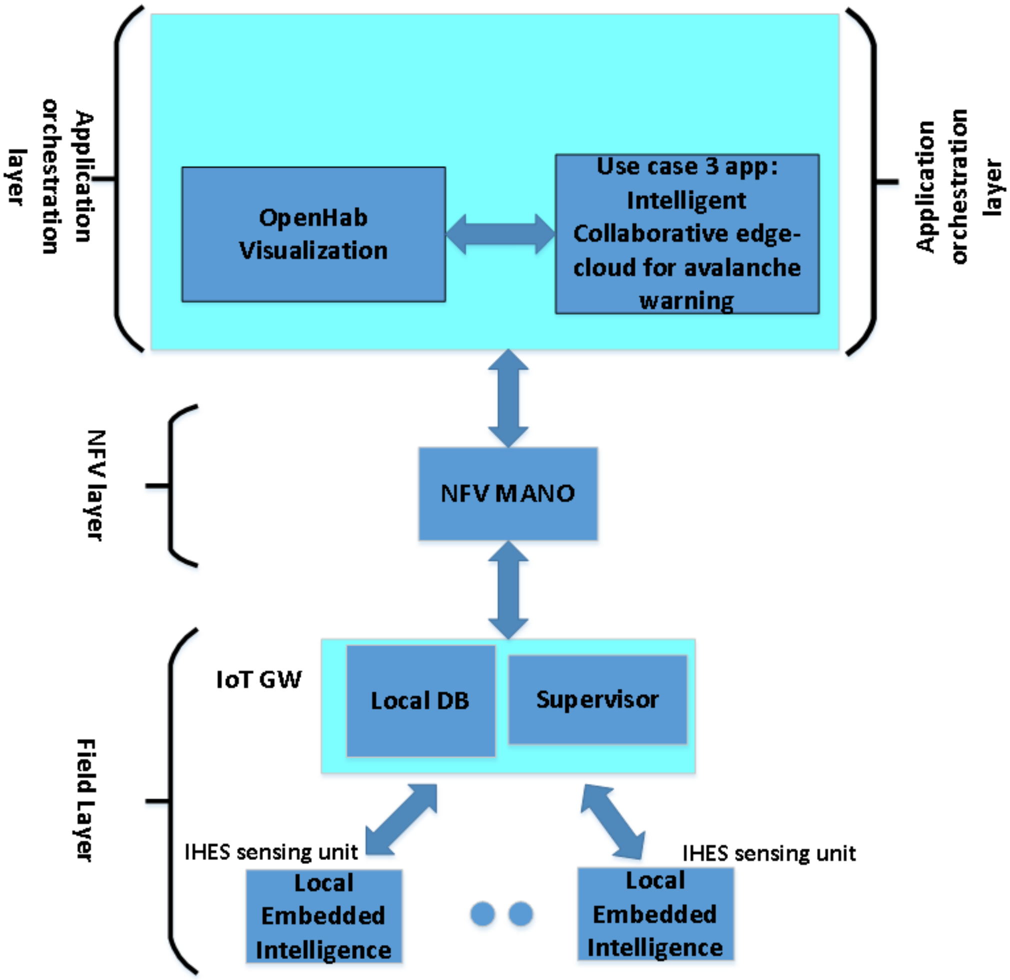

Application orchestration layer
OpenHab Visualization
Use case 3 app: Intelligent Collaborative edge-cloud for avalanche warning
Application orchestration layer
NFV layer
NFV MANO
Field Layer
IoT GW
Local DB
Supervisor
IHES sensing unit
Local Embedded Intelligence
IHES sensing unit
Local Embedded Intelligence

FIGURE 18. FUNCTIONAL BLOCK DIAGRAM OF THE SUB USE CASE 2 AND ITS MAPPING WITHIN THE SEMIOTICS ARCHITECTURE

5.3.1 INTERACTION WITH THE IHES SENSING UNITS AND THE LOCAL EMBEDDED INTELLIGENCE

The IHES sensing units are of paramount importance in the sub use case tackled herein. On the one hand, they obtain the temperature and accelerometer measurements. On the other hand, the accelerometer measurements are used by the local embedded intelligence to infer whether there are seismic events. Moreover, the temperature measurements are sent to the cloud via MQTT. The temperature measurements are the inputs of the temperature predictor implemented at the cloud level. Also, the output of the temperature predictor at the cloud level and the seismic events inferred by the field devices are used as inputs to infer at the cloud level whether there is a risk of an avalanche.

5.3.2 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. Namely, the supervisor at the IoT GW controls the training of the embedded local intelligence algorithms implemented at the IHES sensing units and to this end, it needs the local DB as well. It must be noted that the openHAB visualization tool and the early avalanche warning use case 3 app, located at the cloud-level, interact with the gateway, that hosts the supervisor service and local DB, via an MQTT message flow.

5.3.3 INTERACTION WITH THE NFV COMPONENT

The SEMIoTICS' NFV component is leveraged in 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 18. Thereby, it controls their onboarding within the NFV framework.

5.3.4 INTERACTION WITH THE UC3 APP AT THE APPLICATION ORCHESTRATION LAYER

The use case 3 app of the SEMIoTICS' application orchestration layer is actually the core component of the sub use case presented herein. Namely, first it implements the MQTT receiving chain that allows to receive the MQTT messages sent by the IHES sensing units. Moreover, it extracts the temperature measurements and the seismic events from these MQTT messages. Also, given the temperature measurements, it implements a temperature predictor. The predicted temperature is compared to some parameter bounds provided by a simulated weather agency and if they are surpassed a risky temperature event is generated. Finally, the use case 3 app implements the early avalanche warning system as well. Namely, it leverages the temperature events produced by the temperature predictor along with the seismic events obtained from the IHES sensing units. Thereby, it triggers the avalanche warning when both the temperature and seismic events are identified.

5.3.5 INTERACTION WITH THE OPENHAB VISUALIZATION COMPONENT

This sub-use case provides real-time data visualization by leveraging the openHAB visualization component. More specifically, the global time series database stores the temperature measurements 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 can access the database and visualize the above-mentioned data in the form of charts, at various timescales. openHAB will also visualize in real-time the temperature prediction implemented the use case 3 app mentioned above. Also, the early avalanche warning will be visualized in openHAB.

5.4 Validation

In this section, we show how the sub-use case 2 has been implemented. Namely, according to the previous sections, herein we implement a cloud level data aggregation, data analytics and visualization that tackles the Rigopiano avalanche disaster. To this end, a collaborative edge-cloud approach has been implemented, as it has been introduced above. In Figure 19, for the sake of clarity, we show a block diagram of the implementation of the sub use case 2, which is in the context of the overall use case 3 testbed introduced above.

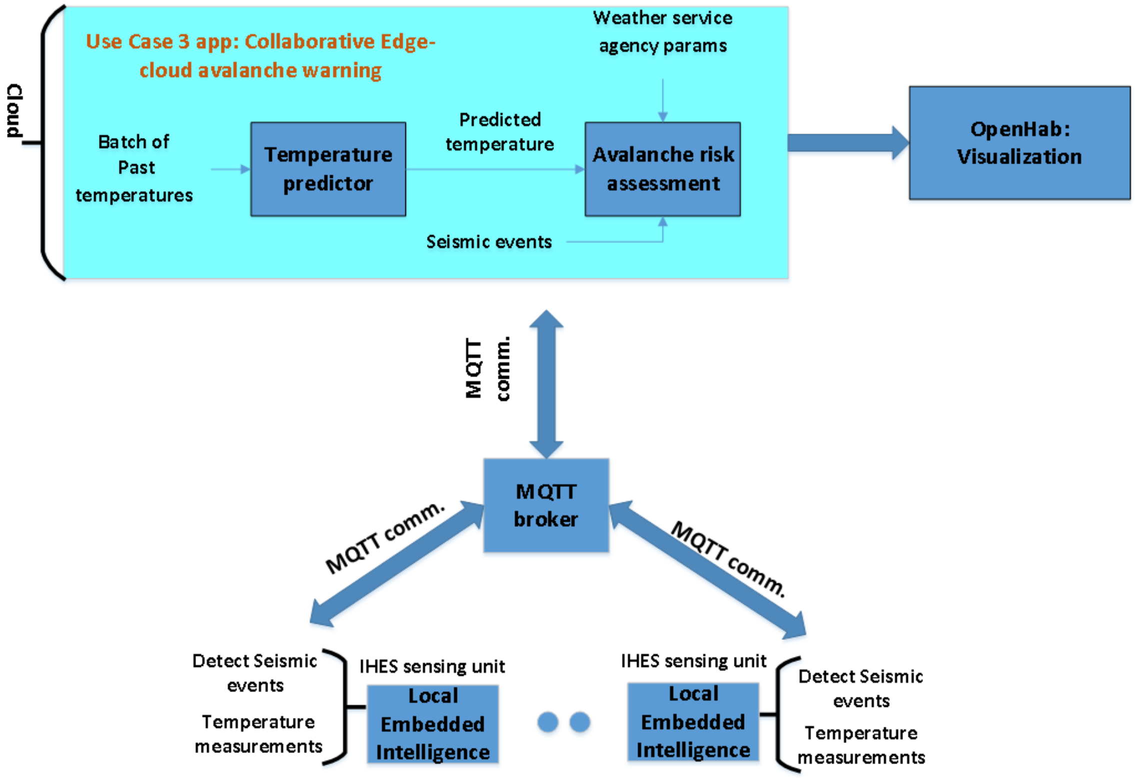


FIGURE 19. FUNCTIONAL BLOCK DIAGRAM OF THE IMPLEMENTATION OF SUB USE CASE 2.

As it can be observed in Figure 19, the IHES sensing units gather temperature measurements. Also, they infer seismic events, given the measurements provided by the accelerometers. This inference is obtained thanks to the local embedded intelligence of the IHES sensing units. Then, the temperature measurements and the seismic events are transmitted to the cloud by using an MQTT communication protocol. Then, at the cloud, the MQTT data is aggregated, analyzed, and visualized. More specifically, the use case 3 app implements the avalanche risk assessment. To this end, it considers the seismic events available from the IHES sensing units and the output of the temperature predictor. The predicted temperature is considered risky if it surpasses the bounds provided by a simulated weather agency. Then, a risk of avalanche is triggered when both the temperature is risky and there are seismic events. Note that the temperature predictor relies on the temperature measurements gathered by the IHES sensing units. Namely, it works recursively and, in each iteration, it considers a batch of past temperatures to make the prediction for the upcoming future time horizon. Also, note that the proposed approach implements collaborative edge-cloud data analytics, as one the one hand the seismic events are inferred at the field devices. On the other hand, the cloud implements a temperature prediction and an avalanche risk assessment that considers both the seismic events, i.e. the intelligence at the edge and the temperature prediction, i.e. the intelligence at the cloud. Finally, the openHAB tool allows to visualize in real-time the temperature measurements available at the cloud, the predicted temperature, and the avalanche risk warning. Observe that the details for the local embedded intelligence and the IHES sensing units have been already provided above in the sub use case 1. Also, the MQTT communication has been detailed above in the testbed setup section. Thereby, herein we focus on providing the details for the use case 3 app, which is the core data analytics component of the collaborative edge-cloud approach. Also, we show the overall integration of this sub-use case in the use case 3 testbed by providing the visualization results displayed in openHAB. These visualizations will show that the temperature measurements that the IHES sensing units transmit via MQTT are received properly and that are displayed in real-time. The visualizations also show that the temperature prediction leverages properly the past temperature measurements. Finally, the openHAB visualizations show that the risk assessment works properly as it raises an alarm when both there are seismic events and the temperature is risky.

5.4.1 TEMPERATURE PREDICTION AND AVALANCHE RISK ASSESSMENT

First, it is worth mentioning that the use case 3 app has been implemented using the next software programming languages and libraries:

- Python. The python 3 programming language has been used to implement the use case 3 app, as it is within the open-source philosophy. Thereby, it can be integrated with the rest of the use case 3 testbed without the need of a license, which is needed e.g., in Matlab.
- NumPy software library. This library has been used as the data analytics requires manipulations of mathematical arrays.
- MQTT paho python library. This is a python library developed by Eclipse that provides APIs that implement the MQTT protocol. We need it as the communications inputs and outputs of the use case 3 app with the rest of the use case 3 testbed are done through MQTT, e.g. the communication between the use case 3 app and the openHAB visualization.

Moreover, as it is highlighted in Figure 19, the use case 3 app implemented for this sub use case 2 consists of the next sub-blocks and interfaces:

- Temperature predictor. It predicts the temperature for a future time horizon periodically, given a batch of past temperature measurements from the IHES sensing units available at the cloud.
- Avalanche risk assessment. Given the temperature prediction and the temperature bounds, provided by a simulated weather agency, it assesses if the predicted temperature is risky. Then, if the temperature is risky and there are seismic events it triggers the avalanche alert.
- Interface with the IHES sensing units. It is based on MQTT, by subscribing to the proper MQTT topics the temperature and seismic events are received, as it is detailed below.
- Interface with the simulated weather agency. It is based on MQTT. As in the previous bullet, the weather station parameter bounds are received by subscribing to the proper MQTT topics, as it is detailed below.
- Interface with the openHAB visualization tool. The use case 3 app publishes MQTT topics containing the predicted temperature and the risk alerts. The openHAB can receive those messages by subscribing to the proper MQTT topics, as it is detailed below.

All these sub-blocks and interfaces will be explained next in more detail.

MQTT communication interfaces

As it is introduced above, the MQTT-based interfaces allow the reception of the temperature measurements and the seismic events from the IHES sensing units. To this end, according to the IHES topic structure introduced in previous sections, we must subscribe to the next MQTT topic:

“ihes/node/+/out”

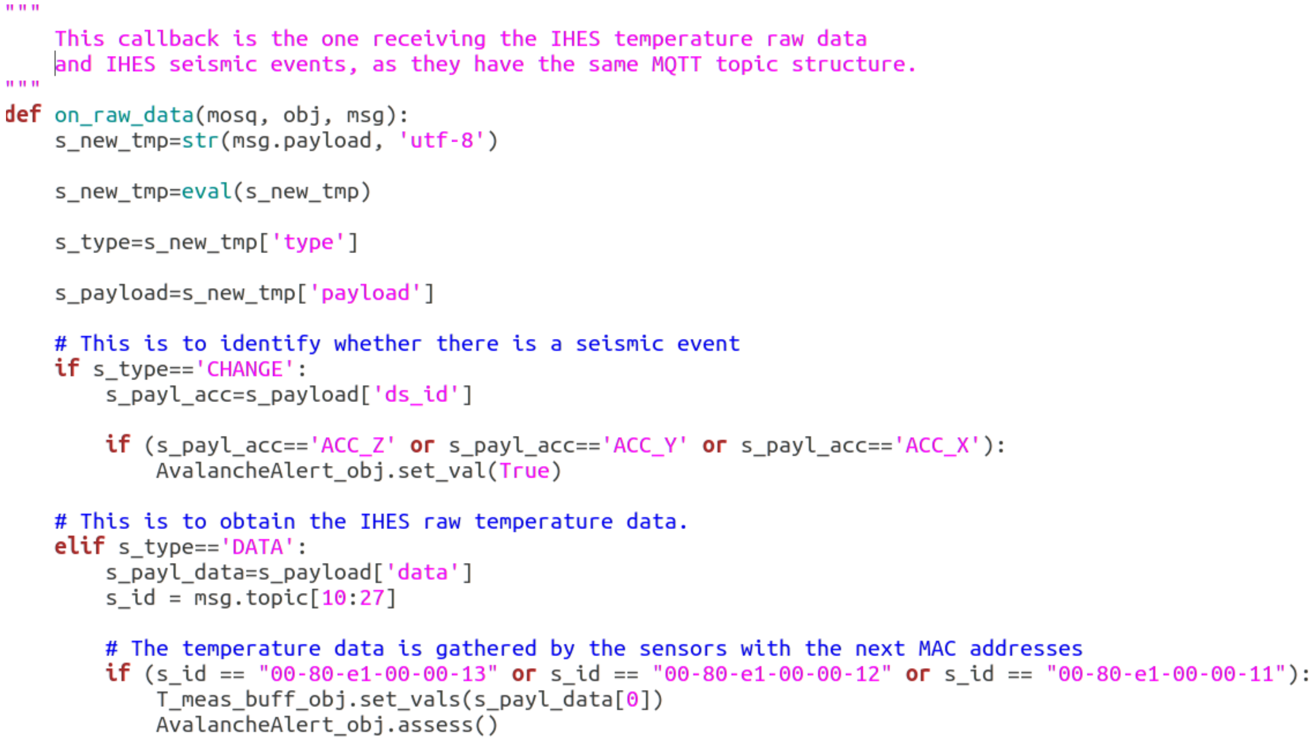
Note, that the wildcard “+” is used, as we do not care on the specific value of that position. When MQTT messages with this topic are received, a callback is called, according to the MQTT paho library, and the user must create the callback to process the MQTT messages with that topic. Thereby, we have created a callback to process the MQTT messages and to extract the temperature measurements and the seismic events embedded within them. This callback is displayed in Figure 20. For the sake of the clarity, note that according to the sections above, an MQTT message containing temperature has a JSON structure of this type:

'{"topic": "ihes/node/00-80-e1-22-11-xx/out", "msg": {"type": "DATA", "ts": 2329865, "seqn": 13518, "payload": {"ds_mask": 7, "sample_id": 11469, "data": [40, 22.1, 1011.32], "residual": [-0.613, 0.353, 0.058]}}

Whereas, an MQTT message containing a seismic event, has a JSON structure of this type:

'{"topic": "ihes/node/00-80-e1-00-00-xx/out", "msg": {"type": "CHANGE", "ts": 2333864, "seqn": 11626, "payload": {"ds_id": "ACC_Z", "tau_ts": 2327264, "thr": 100.0}}}'

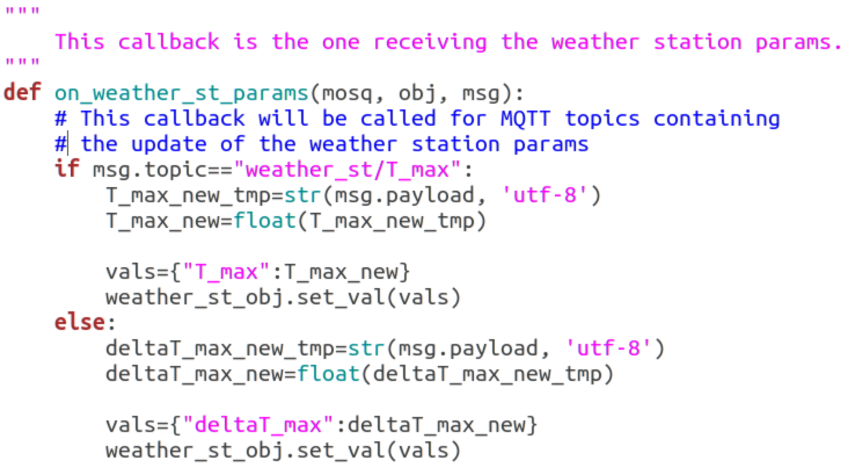
Note that in Figure 20 the callback calls the “AvalancheAlert_obj” object, which implements the avalanche alert assessment.



```
"""
    This callback is the one receiving the IHES temperature raw data
    and IHES seismic events, as they have the same MQTT topic structure.
"""
def on_raw_data(mosq, obj, msg):
    s_new_tmp=str(msg.payload, 'utf-8')

    s_new_tmp=eval(s_new_tmp)

    s_type=s_new_tmp['type']

    s_payload=s_new_tmp['payload']

    # This is to identify whether there is a seismic event
    if s_type=='CHANGE':
        s_payl_acc=s_payload['ds_id']

        if (s_payl_acc=='ACC_Z' or s_payl_acc=='ACC_Y' or s_payl_acc=='ACC_X'):
            AvalancheAlert_obj.set_val(True)

    # This is to obtain the IHES raw temperature data.
    elif s_type=='DATA':
        s_payl_data=s_payload['data']
        s_id = msg.topic[10:27]

        # The temperature data is gathered by the sensors with the next MAC addresses
        if (s_id == "00-80-e1-00-00-13" or s_id == "00-80-e1-00-00-12" or s_id == "00-80-e1-00-00-11"):
            T_meas_buff_obj.set_vals(s_payl_data[0])
            AvalancheAlert_obj.assess()
```

FIGURE 20. CALLBACK FUNCTION THAT EXTRACTS THE TEMPERATURE AND SEISMIC EVENTS FROM THE MQTT MESSAGES.

Similarly, in Figure 21 we present the callback that we have created to extract the weather station parameters embedded within an MQTT topic with a structure *“weather_st/#”*. Namely, there are two kind of topics. “*weather_st/T_max”* has an associated value that indicates the upper bound on temperature. If this value is surpassed then a risky temperature is triggered, as a risk of avalanche arises. In a similar manner, the topic “*weather_st/deltaT_max”* has an associated value that indicates an upper bound on the difference of temperatures, above which an avalanche may occur.



```
"""
    This callback is the one receiving the weather station params.
"""
def on_weather_st_params(mosq, obj, msg):
    # This callback will be called for MQTT topics containing
    # the update of the weather station params
    if msg.topic=="weather_st/T_max":
        T_max_new_tmp=str(msg.payload, 'utf-8')
        T_max_new=float(T_max_new_tmp)

        vals={"T_max":T_max_new}
        weather_st_obj.set_val(vals)
    else:
        deltaT_max_new_tmp=str(msg.payload, 'utf-8')
        deltaT_max_new=float(deltaT_max_new_tmp)

        vals={"deltaT_max":deltaT_max_new}
        weather_st_obj.set_val(vals)
```

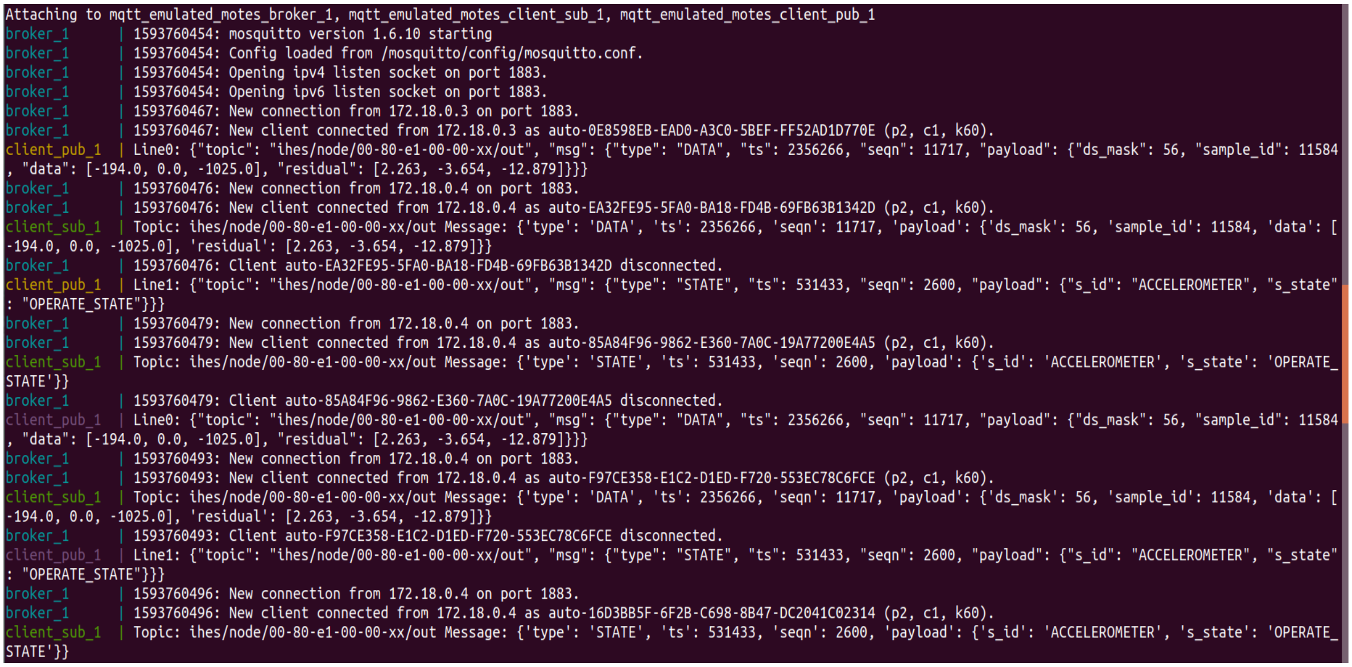
FIGURE 21. CALLBACK FUNCTION THAT EXTRACTS THE WEATHER AGENCY PARAMETERS.

Also, regarding the MQTT-based interfaces, it is worth mentioning that the temperature prediction and the avalanche risk alert, inferred within the use case 3 app, are transmitted to the openHAB via MQTT. To this end, we use the MQTT paho client and we have created three MQTT topics:

- "backend_analytics/avalanche_alert". It has an associated value that indicates whether there is an avalanche alert risk.
- "backend_analytics/temperature_prediction". It has an associated value that contains the predicted temperature.
- "backend_analytics/temperature_diff_prediction". It has an associated value that contains the predicted difference of temperatures.

Finally, we would like to mention that all the MQTT-based interfaces described in this section have been tested before their integration within the overall use case 3 testbed presented below. To this end, we followed the same approach that was suggested in our previous deliverable [5]. Namely, a docker container approach was followed, as it allows to isolate each software block [6]. Thereby, we encapsulated in one docker container the software code that implements the "use case 3 app" of Figure 19, i.e. the temperature predictor, the risk assessment block and the MQTT receiving chains described above. In another docker container we encapsulated an MQTT Eclipse mosquitto broker [7], and finally, in another docker container we encapsulated the MQTT publisher that was emulating the IHES sensing units by transmitting real MQTT messages. Thereby, the MQTT messages stemming from the generated data are published and conveyed to an MQTT broker. Then, the MQTT subscriber callbacks explained above receive the data, i.e. the MQTT topics. To deploy the docker containers we use Docker compose [8], which is a tool to orchestrate multi-container Docker applications.

In Figure 22 we show an example, where we test that an MQTT message that contains the temperature measurements is properly received by the MQTT callbacks explained above. 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. 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 22, 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.



```
Attaching to mqtt_emulated_motes_broker_1, mqtt_emulated_motes_client_sub_1, mqtt_emulated_motes_client_pub_1
broker_1       | 1593760454: mosquitto version 1.6.10 starting
broker_1       | 1593760454: Config loaded from /mosquitto/config/mosquitto.conf.
broker_1       | 1593760454: Opening ipv4 listen socket on port 1883.
broker_1       | 1593760454: Opening ipv6 listen socket on port 1883.
broker_1       | 1593760467: New connection from 172.18.0.3 on port 1883.
broker_1       | 1593760467: New client connected from 172.18.0.3 as auto-0E8598EB-EAD0-A3C0-5BEF-FF52AD1D770E (p2, c1, k60).
client_pub_1  | Line0: {"topic": "ihes/node/00-80-e1-00-00-xx/out", "msg": {"type": "DATA", "ts": 2356266, "seqn": 11717, "payload": {"ds_mask": 56, "sample_id": 11584
, "data": [-194.0, 0.0, -1025.0], "residual": [2.263, -3.654, -12.879]}}}
broker_1       | 1593760476: New connection from 172.18.0.4 on port 1883.
broker_1       | 1593760476: New client connected from 172.18.0.4 as auto-EA32FE95-5FA0-BA18-FD4B-69FB63B1342D (p2, c1, k60).
client_sub_1  | Topic: ihes/node/00-80-e1-00-00-xx/out Message: {'type': 'DATA', 'ts': 2356266, 'seqn': 11717, 'payload': {'ds_mask': 56, 'sample_id': 11584, 'data': [
-194.0, 0.0, -1025.0], 'residual': [2.263, -3.654, -12.879]}}
broker_1       | 1593760476: Client auto-EA32FE95-5FA0-BA18-FD4B-69FB63B1342D disconnected.
client_pub_1  | Line1: {"topic": "ihes/node/00-80-e1-00-00-xx/out", "msg": {"type": "STATE", "ts": 531433, "seqn": 2600, "payload": {"s_id": "ACCELEROMETER", "s_state"
: "OPERATE_STATE"}}}
broker_1       | 1593760479: New connection from 172.18.0.4 on port 1883.
broker_1       | 1593760479: New client connected from 172.18.0.4 as auto-85A84F96-9862-E360-7A0C-19A77200E4A5 (p2, c1, k60).
client_sub_1  | Topic: ihes/node/00-80-e1-00-00-xx/out Message: {'type': 'STATE', 'ts': 531433, 'seqn': 2600, 'payload': {'s_id': 'ACCELEROMETER', 's_state': 'OPERATE_
STATE'}}
broker_1       | 1593760479: Client auto-85A84F96-9862-E360-7A0C-19A77200E4A5 disconnected.
client_pub_1  | Line0: {"topic": "ihes/node/00-80-e1-00-00-xx/out", "msg": {"type": "DATA", "ts": 2356266, "seqn": 11717, "payload": {"ds_mask": 56, "sample_id": 11584
, "data": [-194.0, 0.0, -1025.0], "residual": [2.263, -3.654, -12.879]}}}
broker_1       | 1593760493: New connection from 172.18.0.4 on port 1883.
broker_1       | 1593760493: New client connected from 172.18.0.4 as auto-F97CE358-E1C2-D1ED-F720-553EC78C6FCE (p2, c1, k60).
client_sub_1  | Topic: ihes/node/00-80-e1-00-00-xx/out Message: {'type': 'DATA', 'ts': 2356266, 'seqn': 11717, 'payload': {'ds_mask': 56, 'sample_id': 11584, 'data': [
-194.0, 0.0, -1025.0], 'residual': [2.263, -3.654, -12.879]}}
broker_1       | 1593760493: Client auto-F97CE358-E1C2-D1ED-F720-553EC78C6FCE disconnected.
client_pub_1  | Line1: {"topic": "ihes/node/00-80-e1-00-00-xx/out", "msg": {"type": "STATE", "ts": 531433, "seqn": 2600, "payload": {"s_id": "ACCELEROMETER", "s_state"
: "OPERATE_STATE"}}}
broker_1       | 1593760496: New connection from 172.18.0.4 on port 1883.
broker_1       | 1593760496: New client connected from 172.18.0.4 as auto-16D3BB5F-6F2B-C698-8B47-DC2041C02314 (p2, c1, k60).
client_sub_1  | Topic: ihes/node/00-80-e1-00-00-xx/out Message: {'type': 'STATE', 'ts': 531433, 'seqn': 2600, 'payload': {'s_id': 'ACCELEROMETER', 's_state': 'OPERATE_
STATE'}}
```

FIGURE 22. THE MQTT TOPICS AND VALUES ARE RECEIVED PROPERLY BY THE MQTT SUBSCRIBER

Temperature predictor

Next, we provide the details for the temperature predictor introduced above in Figure 19. Herein, for demonstration purposes we will assume that the prediction is short time horizons. In this setting, we will assume that the temperature at a given time instant depends linearly on its own previous values and on a stochastic term. Thereby, we assume that the temperature evolution is modelled as an autoregressive (AR) mode of order 1, which has the next expression [9]:

$$\boxed{T_m(n) = aT_m(n-1) + \gamma + w(n), \quad 2 \leq n \leq N.} \tag{1}$$

Where, $T_m(n)$ is the temperature measurement gathered by a IHES sensing unit at the time instant *n*. The terms a and γ are the parameters that define the AR model and that must be learned or estimated. Finally, $w(n)$ is a stochastic term that is modeled as white noise.

To predict the temperature for a future time horizon, we need to learn the model parameters a and γ. To this end, it is assumed that a batch of past temperature measurements from the IHES sensing units is available at the cloud level. More specifically, we assume that the batch is of size N and that it can contain measurements from different sensing units if they are correlated. For the learning or estimation of the parameter models, we leverage a Least Squares (LS) method. Namely, first we initialize the estimation of the parameter a to a given value $\breve{a}$. Note that, $\breve{a}$ can be simply the estimated value of a in the previous prediction or that it can be initially estimated using the LS criterion, which yields,

$$\breve{a} = \boldsymbol{x}^{\#}\boldsymbol{y}.$$

Where, bold-face expression denotes vectors and # denotes the pseudo-inverse operator, i.e. $\boldsymbol{x}^{\#} = (\boldsymbol{x}^{T}\boldsymbol{x})^{-1}\boldsymbol{x}^{T}$. T denotes the transpose operator of a vector and $\boldsymbol{x}, \boldsymbol{y}$ have the next expressions:

$$\boldsymbol{x} = [T_m(1), \cdots, T_m(N-1)\,]^T, \qquad \boldsymbol{y} = [T_m(2), \cdots, T_m(N)\,]^T$$

Then, given the initial estimation of $\breve{a}$ we can learn $\hat{\gamma}$. We use a LS regression and thereby $\hat{\gamma}$ has the next expression,

$$\boxed{\hat{\gamma}|\breve{a} = \mathbf{1}^{\#}\boldsymbol{z}} \tag{2}$$

Being, $\mathbf{1}$ a vector of ones of size *N-1* and $\boldsymbol{z}$ has the next expression,

$$\boldsymbol{z} = [T_m(2), \cdots, T_m(N)\,]^T - \breve{a}[T_m(1), \cdots, T_m(N-1)\,]^T$$

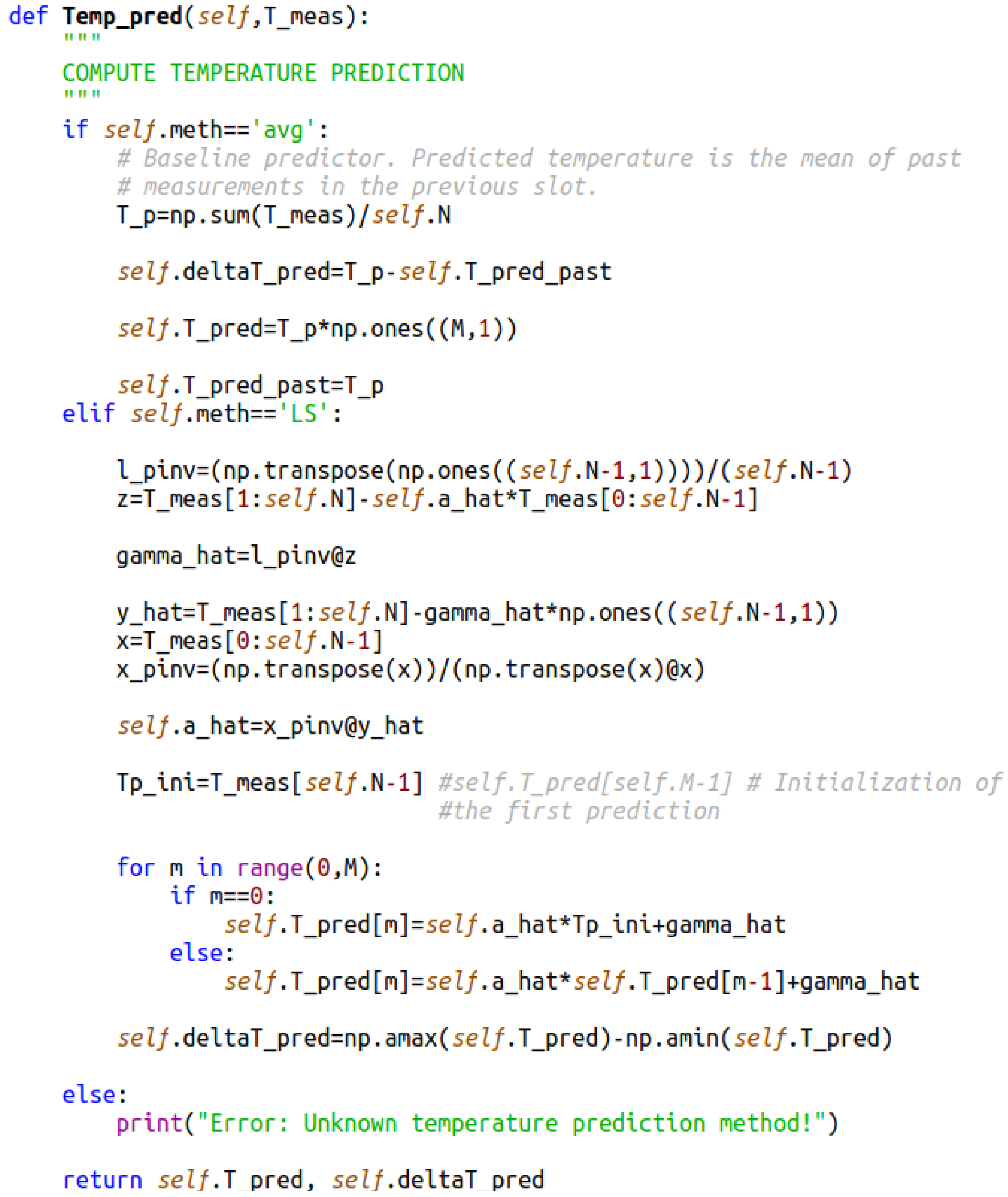
And finally, given $\hat{\gamma}$ we can update our estimation of a by using a LS algorithm:

$$\boxed{\hat{a}|\hat{\gamma} = \boldsymbol{x}^{\#}\widetilde{\boldsymbol{y}}} \tag{3}$$

Where $\widetilde{\boldsymbol{y}} = [T_m(2) - \hat{\gamma}, \cdots, T_m(N) - \hat{\gamma}]^T$. Finally, we can obtain the expression of the predicted temperature, given the values of the AR model that we have learnt in equations (2) and (3). Thereby, the predicted temperature for a future time horizon M is given by the next equation,

$$\boxed{T_p(n) = \hat{a}T_p(n-1) + \hat{\gamma}, \quad 2 \leq n \leq M} \tag{4}$$

Also, to complete our exposition, we provide in Figure 23 the python code that we coded to implement the temperature predictor explained above in equations (1)-(4).



```
def Temp_pred(self,T_meas):
    """
    COMPUTE TEMPERATURE PREDICTION
    """
    if self.meth=='avg':
        # Baseline predictor. Predicted temperature is the mean of past
        # measurements in the previous slot.
        T_p=np.sum(T_meas)/self.N

        self.deltaT_pred=T_p-self.T_pred_past

        self.T_pred=T_p*np.ones((M,1))

        self.T_pred_past=T_p
    elif self.meth=='LS':

        l_pinv=(np.transpose(np.ones((self.N-1,1))))/(self.N-1)
        z=T_meas[1:self.N]-self.a_hat*T_meas[0:self.N-1]

        gamma_hat=l_pinv@z

        y_hat=T_meas[1:self.N]-gamma_hat*np.ones((self.N-1,1))
        x=T_meas[0:self.N-1]
        x_pinv=(np.transpose(x))/(np.transpose(x)@x)

        self.a_hat=x_pinv@y_hat

        Tp_ini=T_meas[self.N-1] #self.T_pred[self.M-1] # Initialization of
                                #the first prediction

        for m in range(0,M):
            if m==0:
                self.T_pred[m]=self.a_hat*Tp_ini+gamma_hat
            else:
                self.T_pred[m]=self.a_hat*self.T_pred[m-1]+gamma_hat

        self.deltaT_pred=np.amax(self.T_pred)-np.amin(self.T_pred)

    else:
        print("Error: Unknown temperature prediction method!")

    return self.T_pred, self.deltaT_pred
```

FIGURE 23. PYTHON CODE THAT IMPLEMENTS THE TEMPERATURE PREDICTOR

Below, we will present how the temperature predictor was properly integrated within the whole use case 2 testbed, as we will show the real-time prediction in openHAB. Moreover, for the sake of completeness, we present next more insights on the temperature prediction that has been just explained. To this end, we did a simulation, where we generated the temperature measurements by using the equation (1), with $a = 1$, $\gamma = 0.1$, $N = 10$, the initial temperature is 18 Celsius degrees and a uniform distribution was assumed for the noise term i.e. $w(n) \sim U(-0.08, 0.08)$. Moreover, the real temperature is assumed to be given by (1) when the noise term is not present. The predicted temperature is obtained by using equation (4) with a horizon $M = 5$. The results are presented in Figure 24. It can be observed that the predicted temperature represents a good estimation of the real temperature. In order to complete the validation, we also present some experiments with real data that we did with this algorithm in Figure 25.

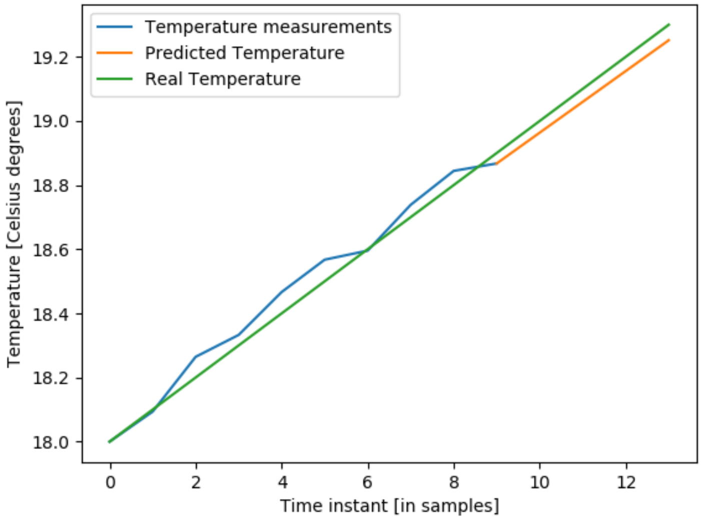
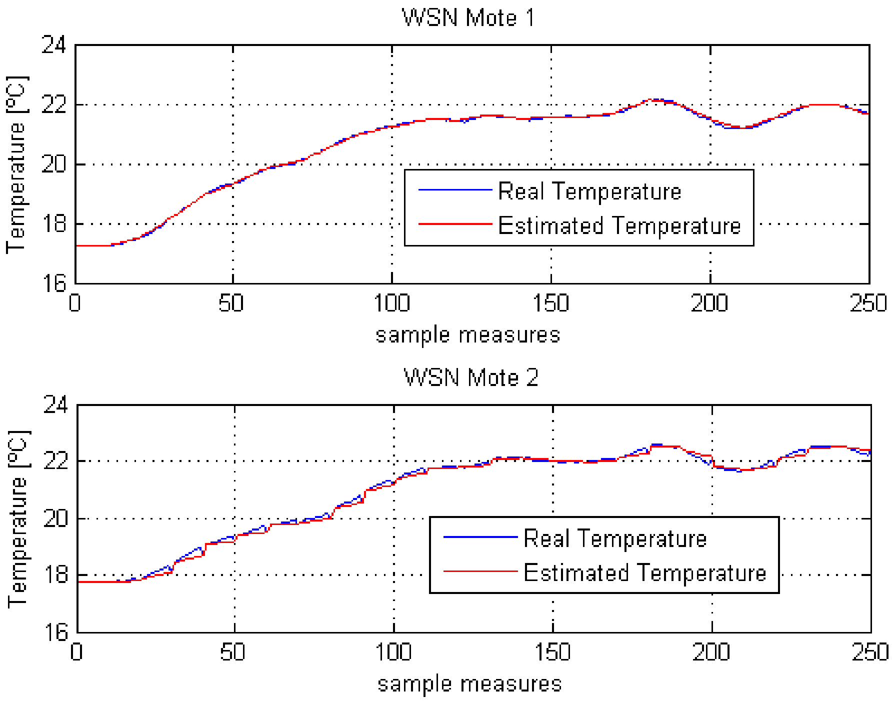


FIGURE 24. VALIDATION OF THE TEMPERATURE PREDICTOR WITH SIMULATED DATA.



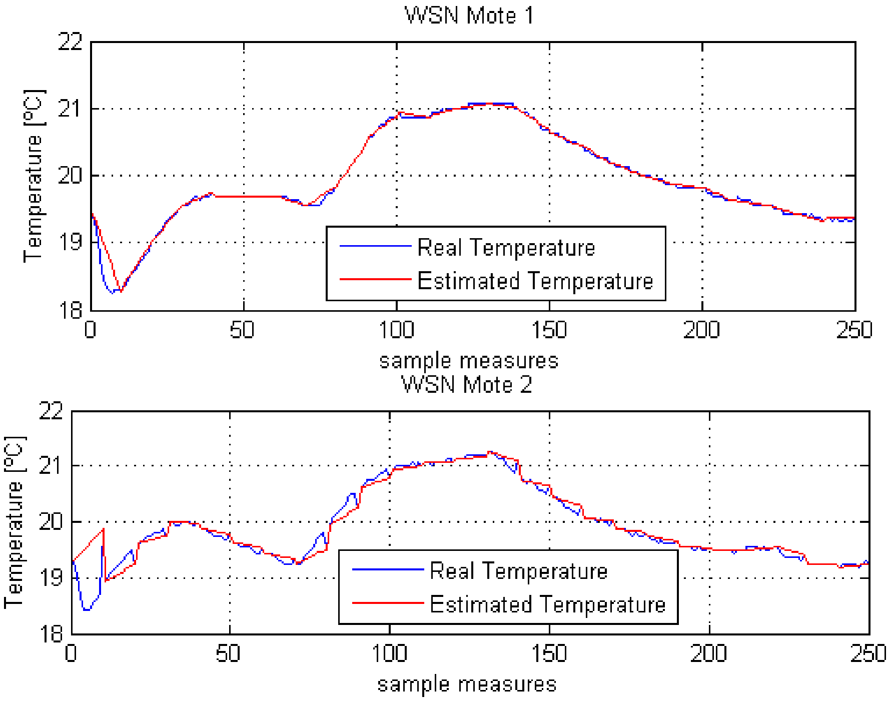


FIGURE 25.VALIDATION OF THE TEMPERATURE PREDICTOR WITH REAL DATA.

Avalanche risk assessment

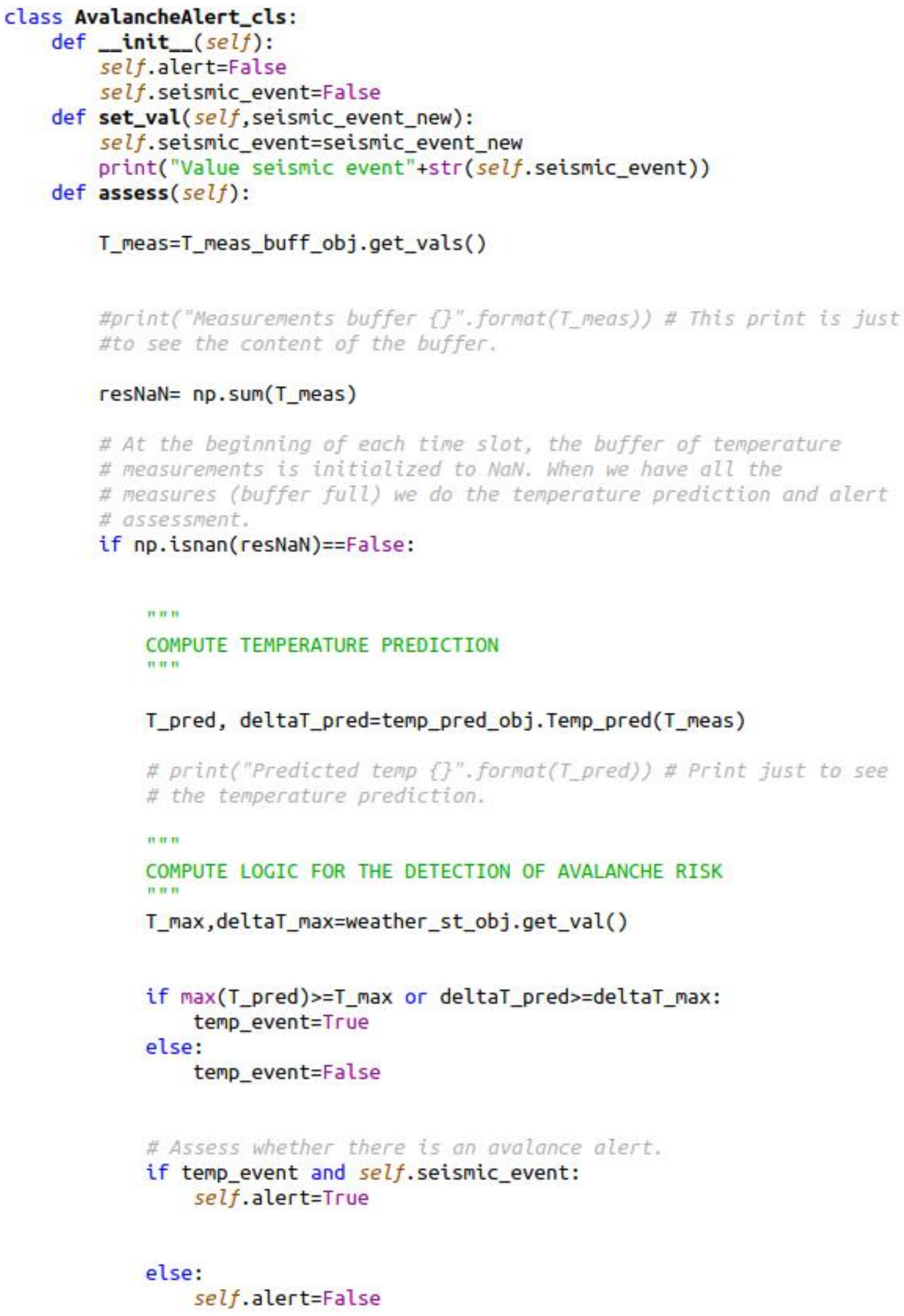
Next, we explain how the avalanche risk assessment sub block depicted in Figure 19 has been implemented. To this end, in Figure 26 we present the python code that was developed. Herein, a collaborative edge-cloud approach is proposed to detect the risk of avalanches, as it has been mentioned above in the context of e.g. Figure 19. Thereby, the avalanche risk assessment requires two main inputs:

- The output of the temperature predictor at the cloud level. This is provided by the code of Figure 23 and corresponds to the equations (1)-(4).
- The seismic events inferred by the IHES sensing units that are available at the cloud via MQTT. This is the interface explained above in Figure 20.

Namely, as it can be observed in Figure 19, when the buffer of temperature measurements is filled, the temperature predictor is called. Then, the predicted temperature is compared to the parameter bounds provided by the weather agency, i.e. T_max and deltaT_max. Recall that these parameters are obtained via MQTT, as it was discussed in the context of Figure 21. If those bounds are exceeded, then a risky temperature event is activated. Then, it is checked if there are both a seismic event and a risky temperature event. In that case, a risk of avalanche is triggered. Therefore, the logic to trigger the avalanche is mathematically expressed as:

$$\boxed{\alpha = \begin{cases} 1, & [\left(\max(\boldsymbol{T}_p) \geq T_{max}\right) OR \left(\Delta \boldsymbol{T}_p \geq \Delta T_{max}\right)] AND(\sigma) \\ 0, & otherwise. \end{cases}} \quad (5)$$

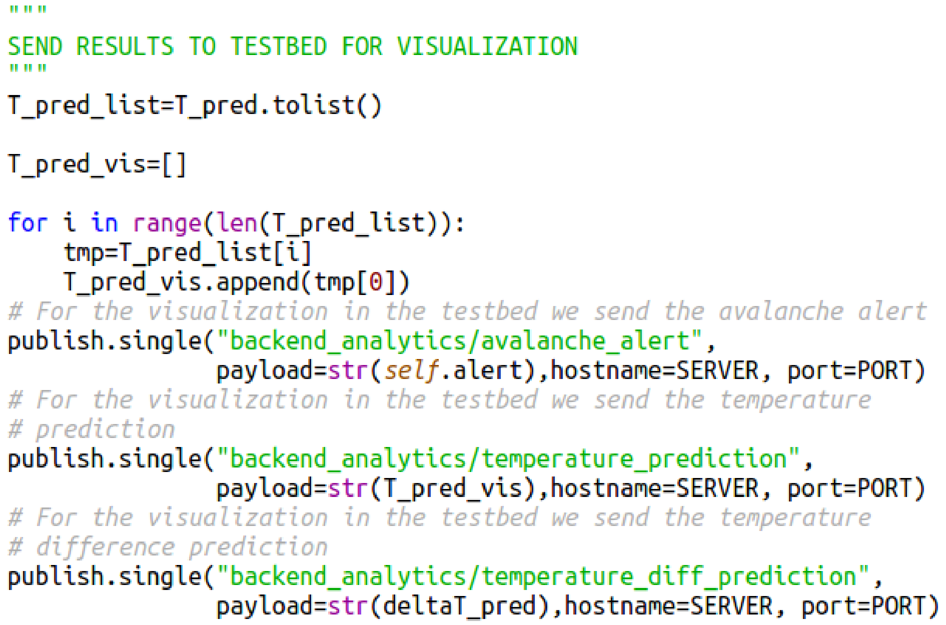
Where, α denotes the avalanche warning, $\boldsymbol{T}_p$ is the vector of the predicted temperatures obtained by stacking the values in (4), $\Delta\boldsymbol{T}_p$ is the difference between the maximum and minimum in $\boldsymbol{T}_p$. The parameters T_{max} and ΔT_{max} are the upper bounds on the temperature and difference of temperatures given by the weather agency. Finally, σ denotes whether there are seismic events detected by the IHES sensing units.



```
class AvalancheAlert_cls:
    def __init__(self):
        self.alert=False
        self.seismic_event=False
    def set_val(self,seismic_event_new):
        self.seismic_event=seismic_event_new
        print("Value seismic event"+str(self.seismic_event))
    def assess(self):

        T_meas=T_meas_buff_obj.get_vals()


        #print("Measurements buffer {}".format(T_meas)) # This print is just
        #to see the content of the buffer.

        resNaN= np.sum(T_meas)

        # At the beginning of each time slot, the buffer of temperature
        # measurements is initialized to NaN. When we have all the
        # measures (buffer full) we do the temperature prediction and alert
        # assessment.
        if np.isnan(resNaN)==False:


            """
            COMPUTE TEMPERATURE PREDICTION
            """

            T_pred, deltaT_pred=temp_pred_obj.Temp_pred(T_meas)

            # print("Predicted temp {}".format(T_pred)) # Print just to see
            # the temperature prediction.

            """
            COMPUTE LOGIC FOR THE DETECTION OF AVALANCHE RISK
            """
            T_max,deltaT_max=weather_st_obj.get_val()


            if max(T_pred)>=T_max or deltaT_pred>=deltaT_max:
                temp_event=True
            else:
                temp_event=False


            # Assess whether there is an avalance alert.
            if temp_event and self.seismic_event:
                self.alert=True


            else:
                self.alert=False
```

FIGURE 26. AVALANCHE RISK ASSESSMENT PYTHON CODE

Finally, the avalanche risk warning and the temperature predictions are sent via MQTT to the openHAB. The code to do so is displayed in Figure 27.



```
"""
SEND RESULTS TO TESTBED FOR VISUALIZATION
"""
T_pred_list=T_pred.tolist()

T_pred_vis=[]

for i in range(len(T_pred_list)):
    tmp=T_pred_list[i]
    T_pred_vis.append(tmp[0])
# For the visualization in the testbed we send the avalanche alert
publish.single("backend_analytics/avalanche_alert",
               payload=str(self.alert),hostname=SERVER, port=PORT)
# For the visualization in the testbed we send the temperature
# prediction
publish.single("backend_analytics/temperature_prediction",
               payload=str(T_pred_vis),hostname=SERVER, port=PORT)
# For the visualization in the testbed we send the temperature
# difference prediction
publish.single("backend_analytics/temperature_diff_prediction",
               payload=str(deltaT_pred),hostname=SERVER, port=PORT)
```

FIGURE 27. PYTHON CODE THAT SENDS THE AVALANCHE WARNING AND PREDICTED TEMPERATURES TO OPENHAB FOR VISUALIZATION.

5.4.2 OPENHAB VISUALIZATION: VALIDATION OF THE INTEGRATION OF THE COLLABORATIVE EDGE-CLOUD ANALYTICS WITHIN THE UC3 TESTBED

In this sub-section, we present the integration of the sub use case 2 explained above within the use case 3 testbed by means of the visualizations provided by openHAB. To this end, first in Figure 28 we highlight the next functionalities. On the bottom left, we display the IHES sensing units that gather the temperature measurements and seismic events. As it was mentioned above in this section, the IHES sensing units send the data to the cloud via MQTT, where the data is stored, it is analyzed, and it is displayed. Thereby, on the top of Figure 28 we show the temperature measurements from the IHES sensing units are displayed properly in real-time by means of the openHAB tool. This figure also validates the MQTT communication chain between the IHES sensing units and the cloud. Moreover, on the right bottom of Figure 28 we display the real-time prediction of the temperature predictor explained in the previous sub-section. Recall, that this predictor takes periodically a batch of temperature measurements to perform the prediction. Namely, in the case of Figure 28, the predictor is taking the temperature measurements that are displayed in the figure. On the bottom of Figure 28 we also show the avalanche warning, which is tackle in more detail in the figures that are explained below.

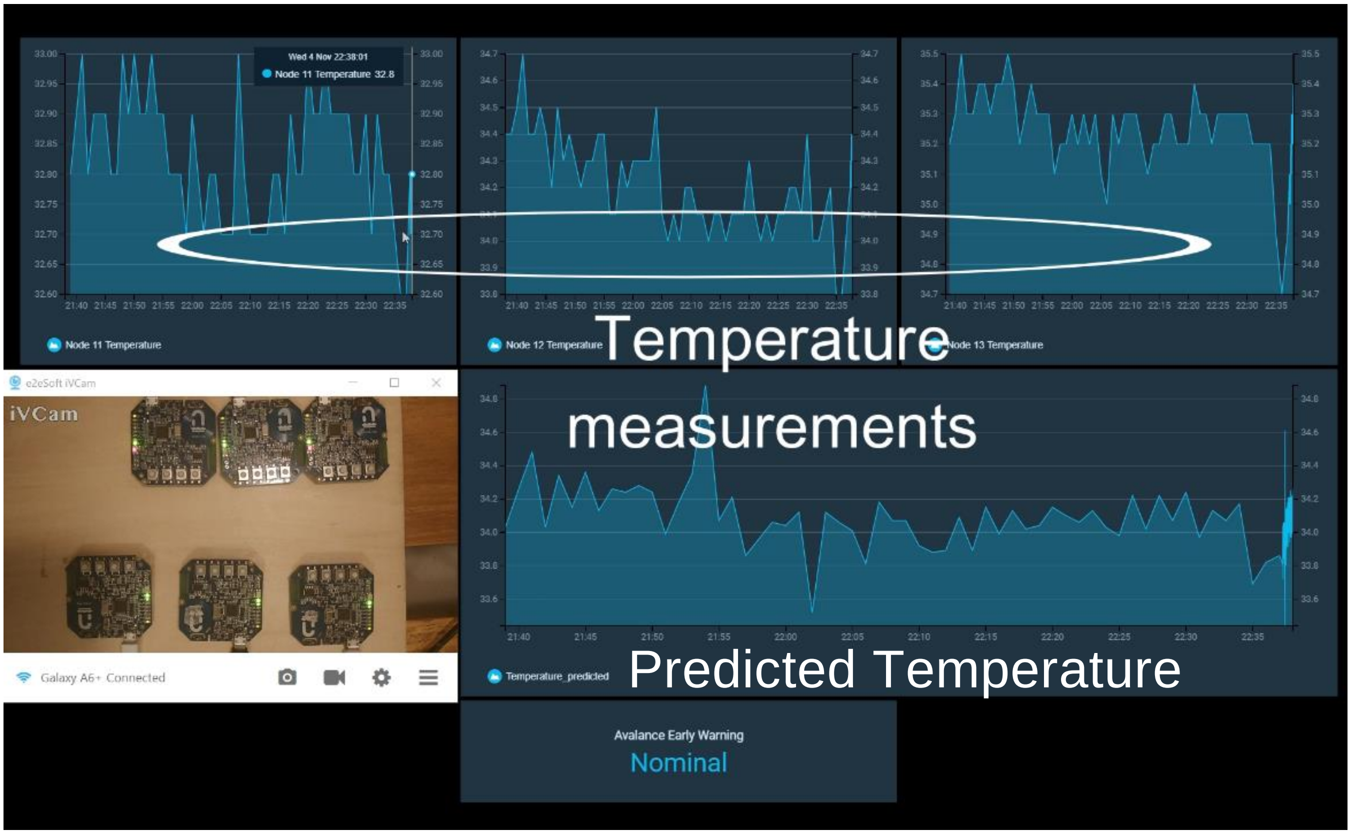
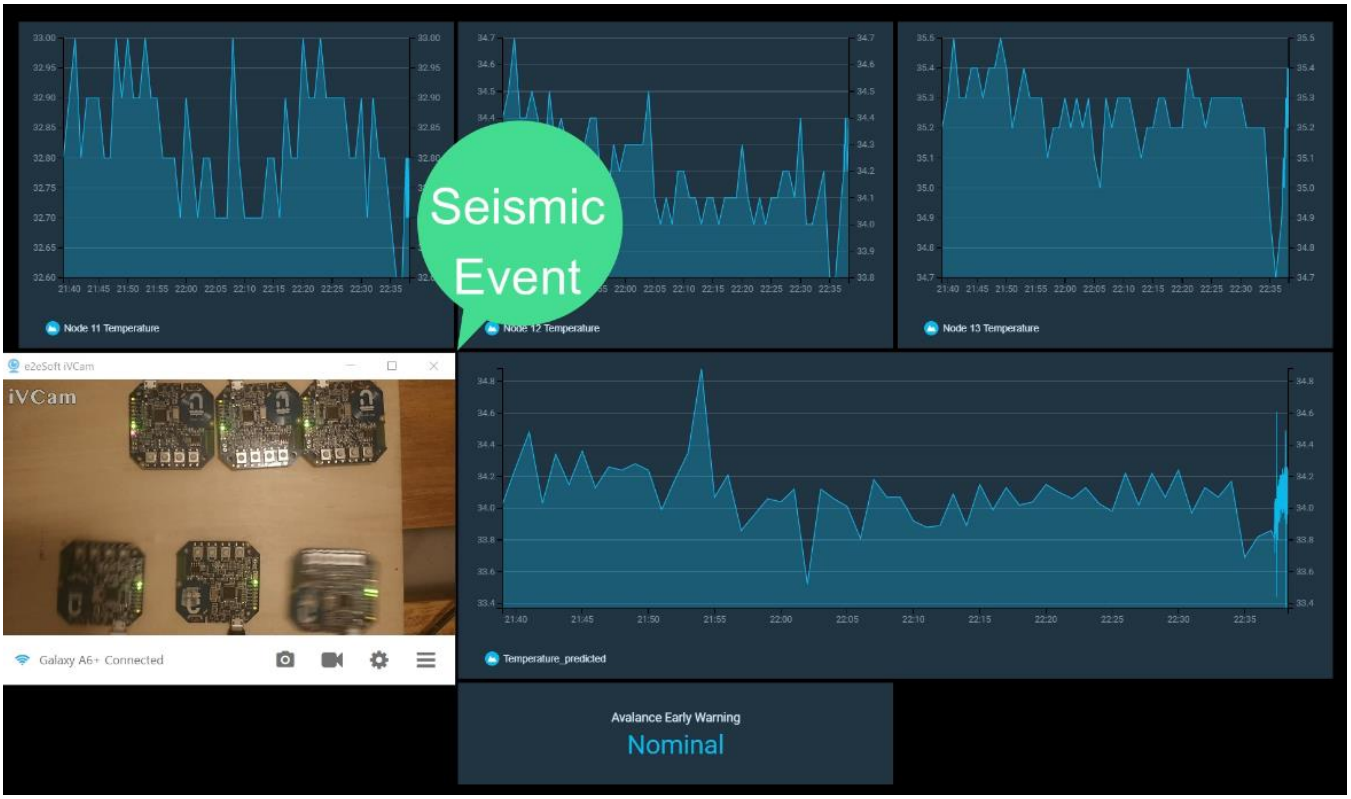


FIGURE 28. INTEGRATION IN UC3 TESTBED. OPENHAB DISPLAY TEMPERATURE MEASUREMENTS AND THE PREDICTED TEMPERATUTRE.

In Figure 29, we generated a seismic event by moving some of the IHES sensing units. This event is detected by the local embedded intelligence of the IHES sensing units and it is transmitted to the cloud via MQTT. Then, as it was explained in the previous sub-section, the avalanche risk assessment block of Figure 19 considers this seismic event along with the output of the temperature predictor. Note that in this case, the temperature is not risky, as compared to the weather station bounds. Thereby, the avalanche risk assessment block considers that there is not an avalanche risk. This is properly displayed in the bottom of Figure 29.



Avalance Early Warning
Nominal

FIGURE 29. INTEGRATION IN UC3 TESTBED: SEISMIC EVENT GENERATED AT THE IHES SENSING UNITS.

In Figure 30, we provoked a risky temperature event by lighting up some of the IHES sensing units. The temperature measurements are sent to the cloud via MQTT and are displayed in real-time in openHAB. We can see on the top right of Figure 30 how the temperature raises suddenly. The temperature measurements are the inputs of the temperature predictor, which predicts that the temperature will keep increasing. Then, this output is sent to the avalanche risk assessment block. However, as there is not a seismic event, the risk assessment block considers that there is not an avalanche risk.

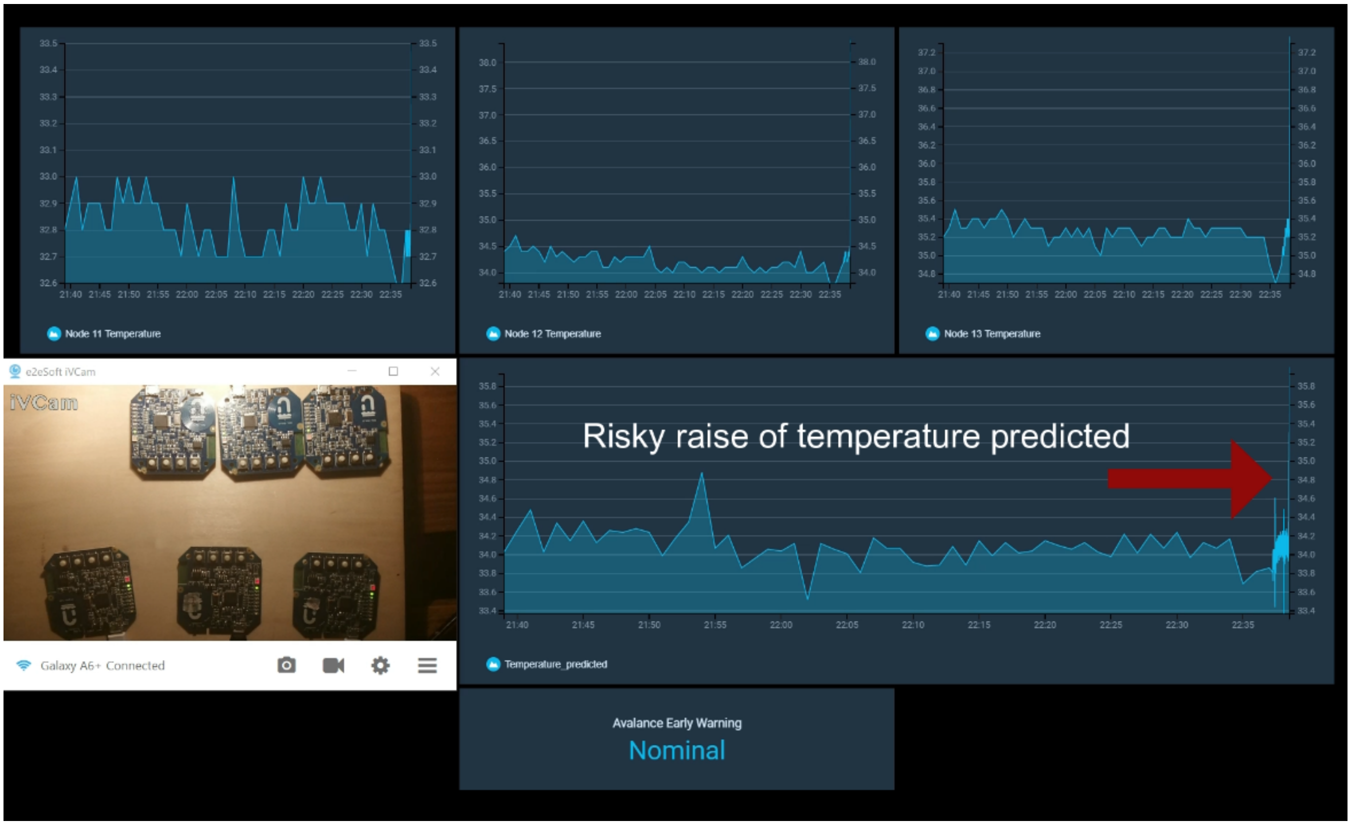
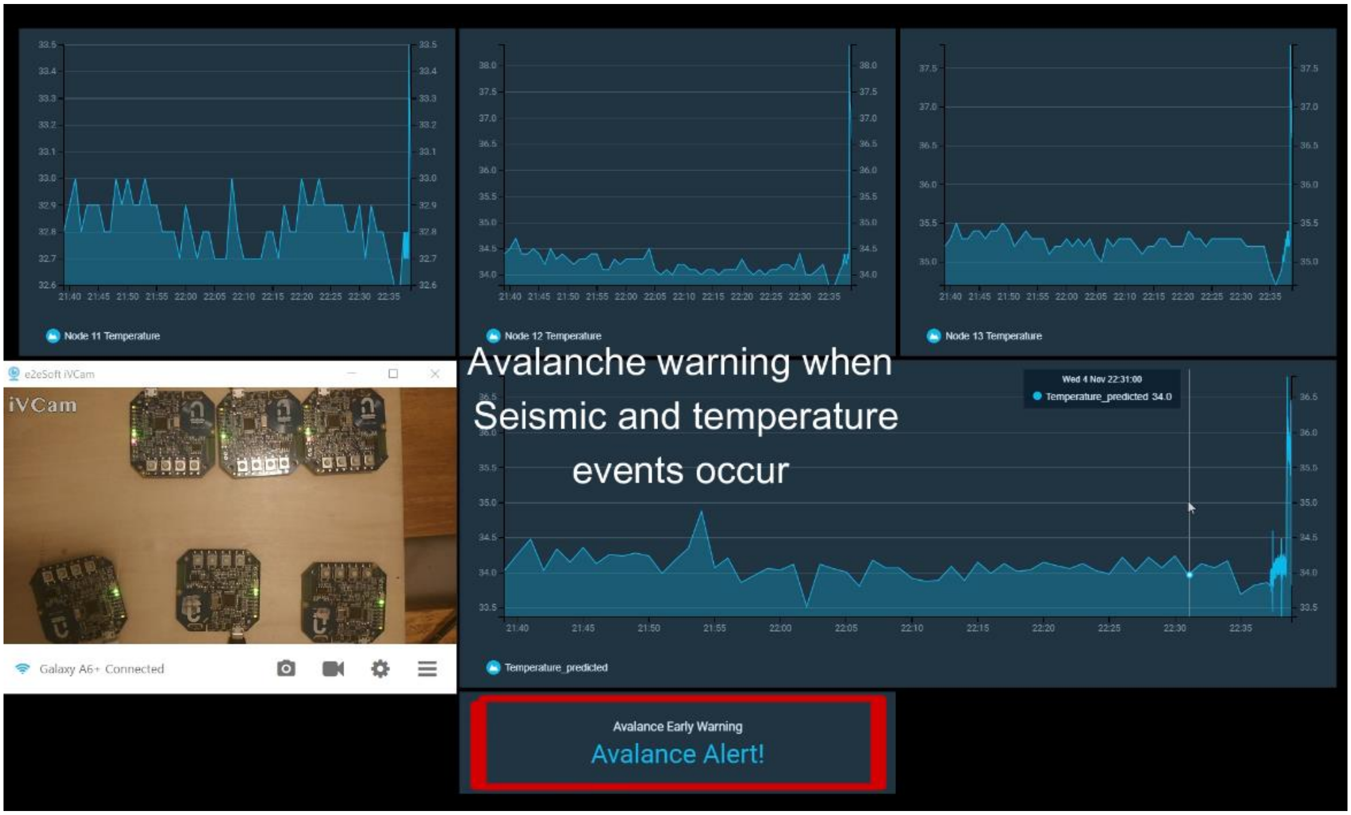


FIGURE 30. INTEGRATION IN UC3 TESTBED: RISKY TEMPERATURE EVENT PREDICTED.

Finally, in Figure 31, we generated both a seismic event and a rise of the temperature measurements, by following the same procedure than the ones explained for Figure 29 and Figure 30. The temperature measurements are sent to the cloud via MQTT and are the input of the temperature predictor, which displays the trend of an increasing temperature. This prediction is the input of the risk assessment block along with the seismic event sent by the IHES sensing units. The risk assessment block evaluates the predicted temperature, and it considers that is risky as it surpasses the weather agency parameter bounds. At the same time, it realizes that there is a seismic event. Thereby, according to the logic explained in equation (5) it raises an avalanche warning, which is sent to openHAB via MQTT. As we can observe in Figure 31 the avalanche risk warning is properly displayed in openHAB.



Avalanche warning when Seismic and temperature events occur

Avalance Early Warning

Avalance Alert!

FIGURE 31. INTEGRATION IN UC3 TESTBED: AVALANCHE WARNING WHEN THERE ARE BOTH SEISMIC EVENTS AND RISKY TEMPERATURE EVENTS.

6 SUB USE CASE 3: PATTERN-BASED SENSING 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 must be deployed to ensure that even if some sensors fail, the system will continue to operate. Moreover, the reliability of the measurements must be monitored, to ensure that, in case of sensor(s) malfunction, this is detected on time.

In this context, the 3rd sub-use case revolves around dependable 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). The SEMIoTICS patterns are leveraged in this case to provide real-time situational awareness regarding the dependability posture of the monitoring setup.

In Figure 32 a topology is depicted that corresponds to the scenario of sub use case 3, and more specifically the 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. In the case of the specific testbed used to demonstration and validation, N=3, i.e., 3 identical sensors. At any time, all of them are deployed in the same vicinity and are simultaneously operational to provide the needed redundancy.

A key SEMIoTICS enhancement in this regard is the use of the pattern-driven capabilities of SEMIoTICS to autonomously, at the field layer, reason on sensing system‘s dependability posture, informing the operators at the backend in real time. The **Dependability** property reasoning is decomposed to:

1) a **Fault Tolerance** property, based on monitoring the availability of redundant sensors (i.e., the availability of at least 2 active sensors), and
2) a **Reliability** property, monitoring sensor readings’ reliability (i.e., consistency between received sensor readings, to detect malfunctioning sensors)

To showcase the above, the demonstration and validation scenario features sensor failures in terms of both the sub-properties (Fault Tolerance & Reliability) and a combination of those, with intermittent sensors’ restoration, visualising the consequent changes in the dependability posture of the deployment for the operator at the SEMIoTICS backend GUI.

In terms of implementation, the above is achieved by deploying a lightweight Field Pattern Engine with MQTT integration that runs on Gateway, and which integrates pattern rules allowing to reason locally about the dependability properties of the anomaly detection setup. The backend visualisation of the Dependability state is enabled by the deployment of the Pattern Orchestrator & GUI components. More details on the implementation aspects are provided in the subsections that follow.

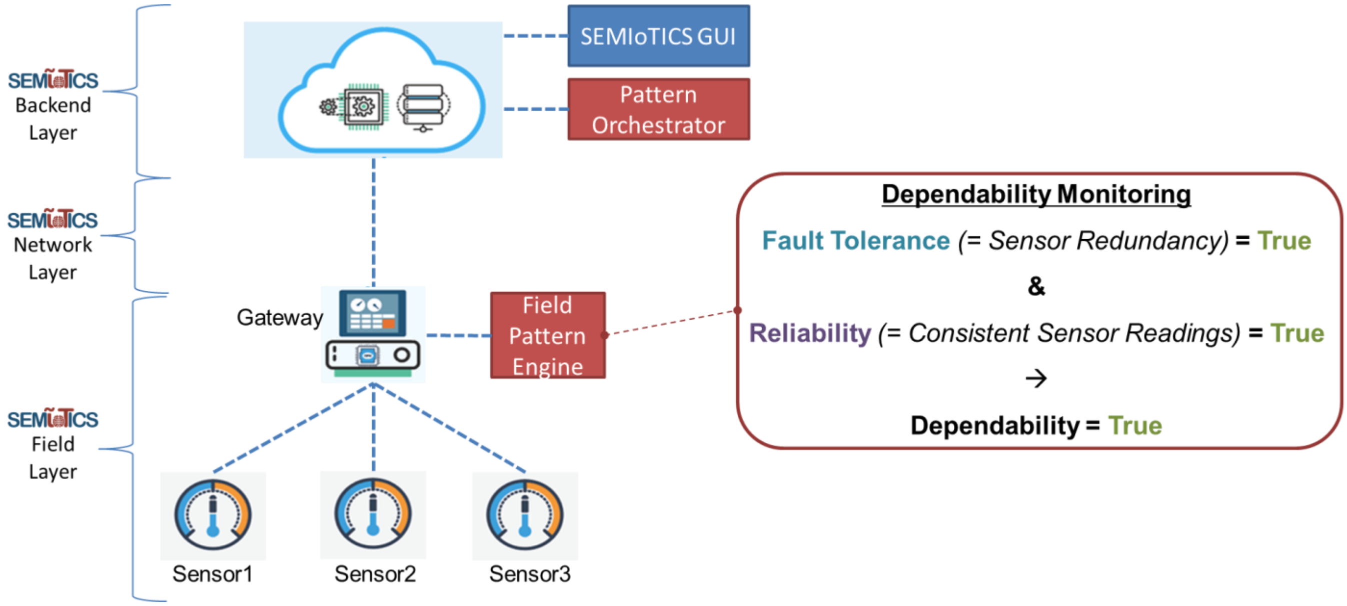


FIGURE 32. SUB USE CASE 3 TOPOLOGY AND KEY ELEMENTS

6.2 Scope and objectives

The overarching aim and objective of the sub-use case is to demonstrate in practice how the pattern-driven monitoring capabilities of SEMIoTICS can be used to autonomously, locally at the field layer (the layer at the focus of UC3), reason on sensing system‘s dependability posture, providing a real-time view of the sensing dependability posture of the system to the SEMIoTICS operators at the backend.

Specific objectives include:

- Provide autonomous & lightweight pattern reasoning capabilities at the field layer
- Define and use sensing dependability-focused pattern rules that combine sensor reliability and fault-tolerance aspects
- Define and integrate fault-tolerance -related monitoring capabilities
- Define and integrate reliability -related monitoring capabilities
- Integrate with field layer SEMIoTICS deployment devices and associated messaging bus
- Provide connectivity with SEMIoTICS GUI, enabling the visualization of sensing dependability posture in real time.

6.3 Interaction with SEMIoTICS framework and components

A sequence diagram detailing the sub use case 3 interactions realising the field layer pattern-based sensing dependability monitoring is provided in Figure 33 and Figure 34.

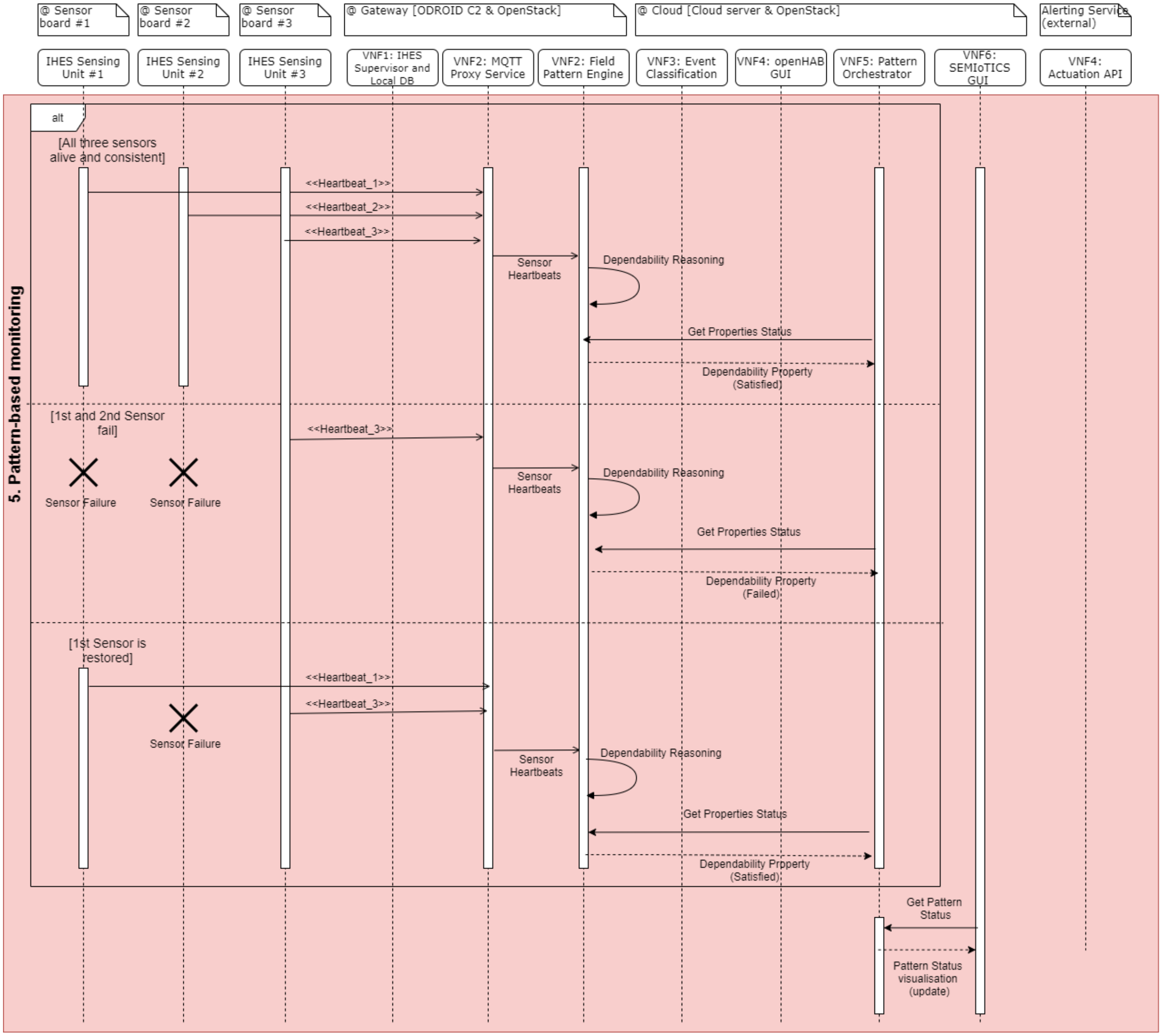


FIGURE 33. PATTERN-BASED SENSING DEPENDABILITY MONITORING SEQUENCE (1/2)

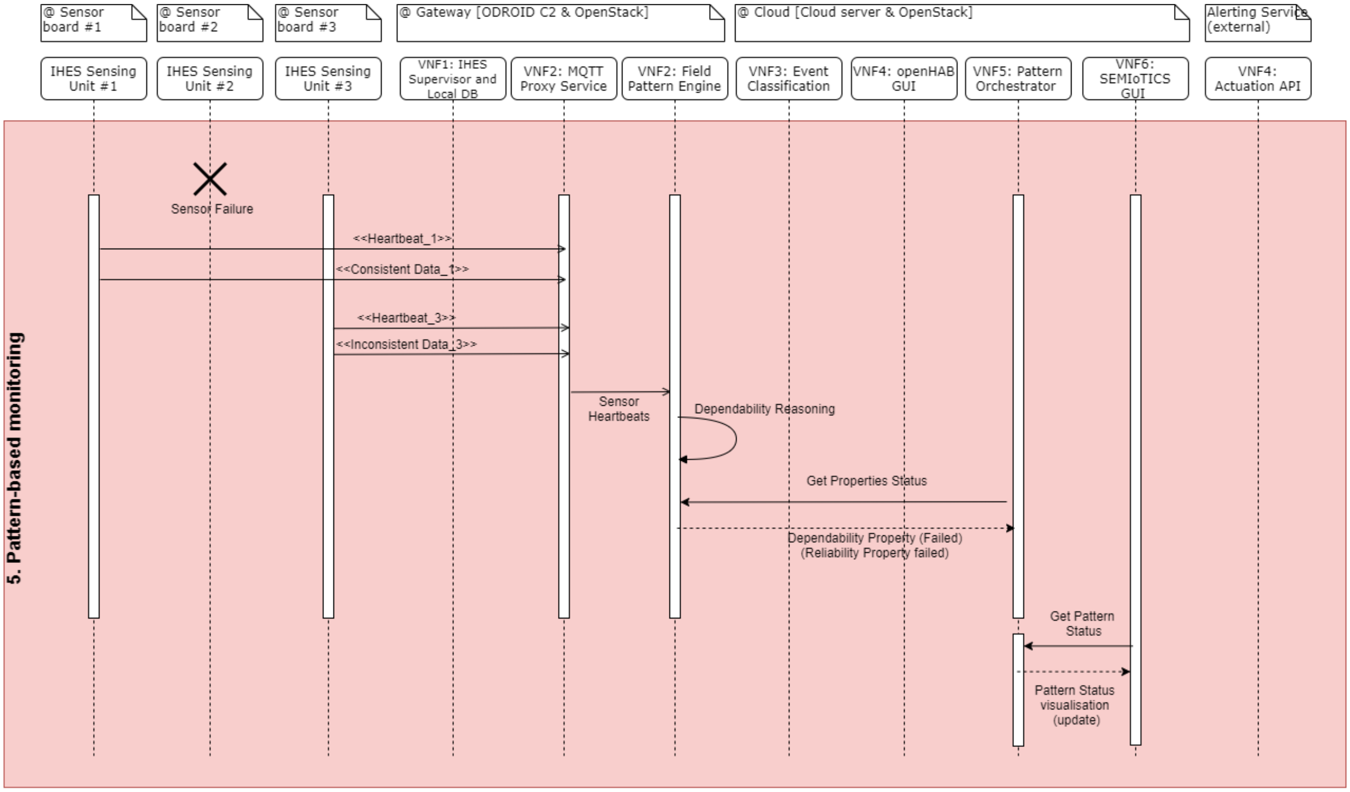


FIGURE 34. PATTERN-BASED SENSING DEPENDABILITY MONITORING SEQUENCE (2/2)

As shown in the above figure, the lightweight Field Pattern Engine monitors (through the MQTT broker):

- the liveness messages from the three sensors ("heartbeats"), leveraging this information to reason about the fault tolerance of the sensing system.
- The readings provided by the three sensors, assessing the deviation between said readings, and leveraging this information to reason about the Reliability of the sensing system.

The above messages are then used to reason on the overall dependability posture of the sensing system. The status of the pattern reasoning and the associated properties is relayed to the SEMIoTICS backend, via interaction between the Field Pattern Engine and the Pattern Orchestrator. The latter then relays that information to the SEMIoTICS GUI for visualization.

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 can locally reason 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 MQTT 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.

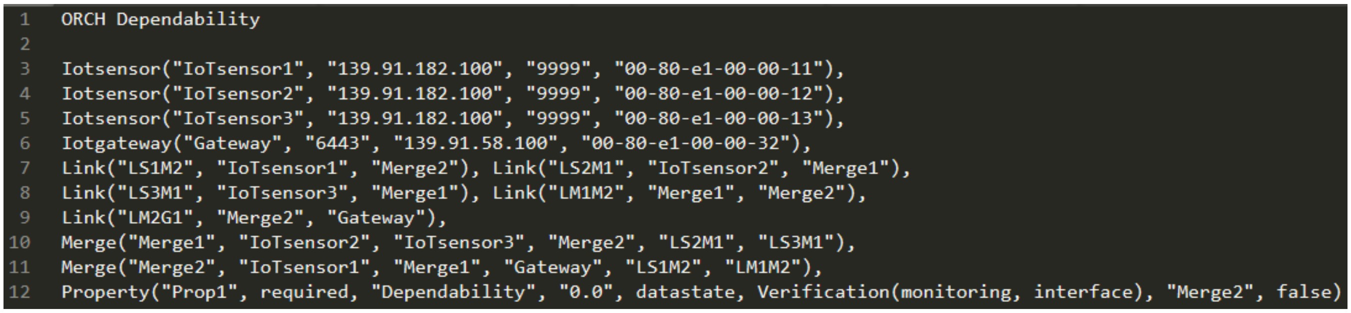
While Cycle 1 efforts focused on the integration of the field-level components (i.e., integrating the Pattern Engine with the Gateway and MQTT Broker, reasoning on the sensor properties), Cycle 2 focused on also integrating (ii) and (iii) above in the sub use case, followed by the validation and demonstration of the sub use case, and UC3 as a whole.

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 rules, leveraging the SEMIoTICS pattern language (see D4.8 – "SEMIoTICS SPDI Patterns (final)").

In specific, the topology depicted in Figure 32 is described using the SEMIoTICS orchestration specification language as shown in Figure 35.



```
ORCH Dependability

Iotsensor("IoTsensor1", "139.91.182.100", "9999", "00-80-e1-00-00-11"),
Iotsensor("IoTsensor2", "139.91.182.100", "9999", "00-80-e1-00-00-12"),
Iotsensor("IoTsensor3", "139.91.182.100", "9999", "00-80-e1-00-00-13"),
Iotgateway("Gateway", "6443", "139.91.58.100", "00-80-e1-00-00-32"),
Link("LS1M2", "IoTsensor1", "Merge2"), Link("LS2M1", "IoTsensor2", "Merge1"),
Link("LS3M1", "IoTsensor3", "Merge1"), Link("LM1M2", "Merge1", "Merge2"),
Link("LM2G1", "Merge2", "Gateway"),
Merge("Merge1", "IoTsensor2", "IoTsensor3", "Merge2", "LS2M1", "LS3M1"),
Merge("Merge2", "IoTsensor1", "Merge1", "Gateway", "LS1M2", "LM1M2"),
Property("Prop1", required, "Dependability", "0.0", datastate, Verification(monitoring, interface), "Merge2", false)
```

FIGURE 35. SUB USE CASE 3 ORCHESTRATION SPECIFICATION IN SEMIOTICS LANGUAGE

Other than the orchestration specification, several pattern rules are needed to verify the dependability property within the sub use case 3 setup as described above. Following the approach detailed within D4.8, these patterns are expressed in the form of Drools rules, while some additional rules are necessary for the verification of the desired property. The whole dependability property verification property is shown in Figure 36.

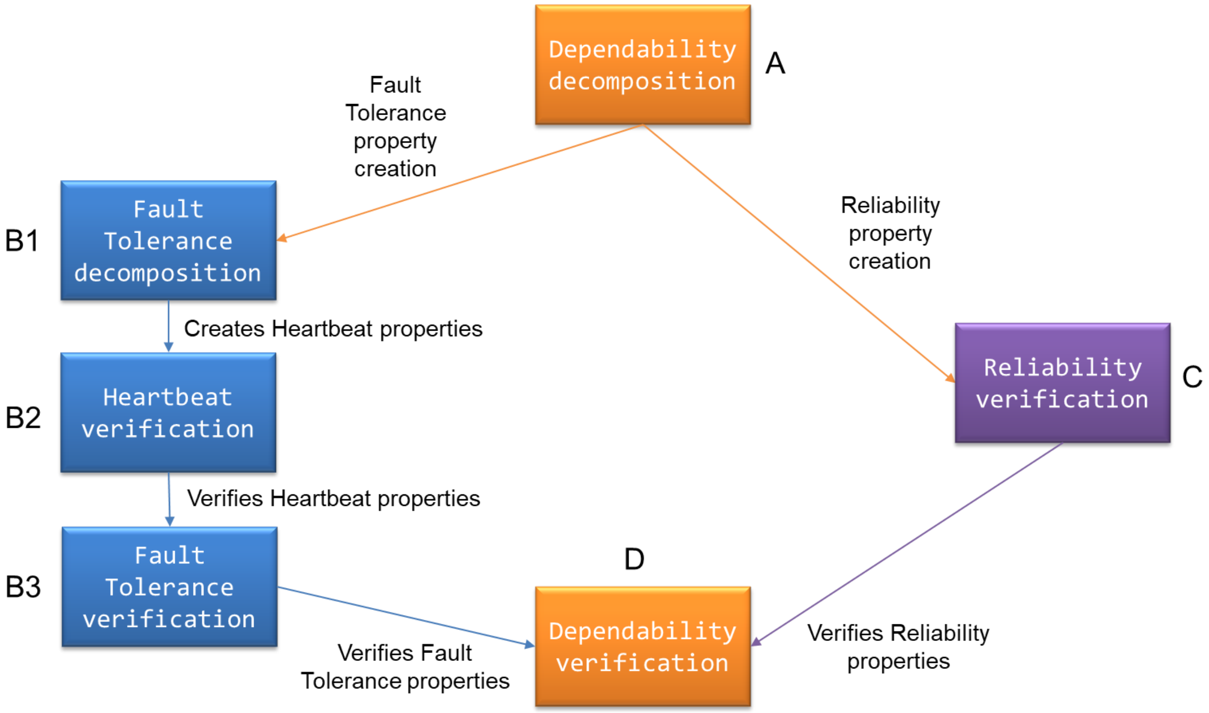
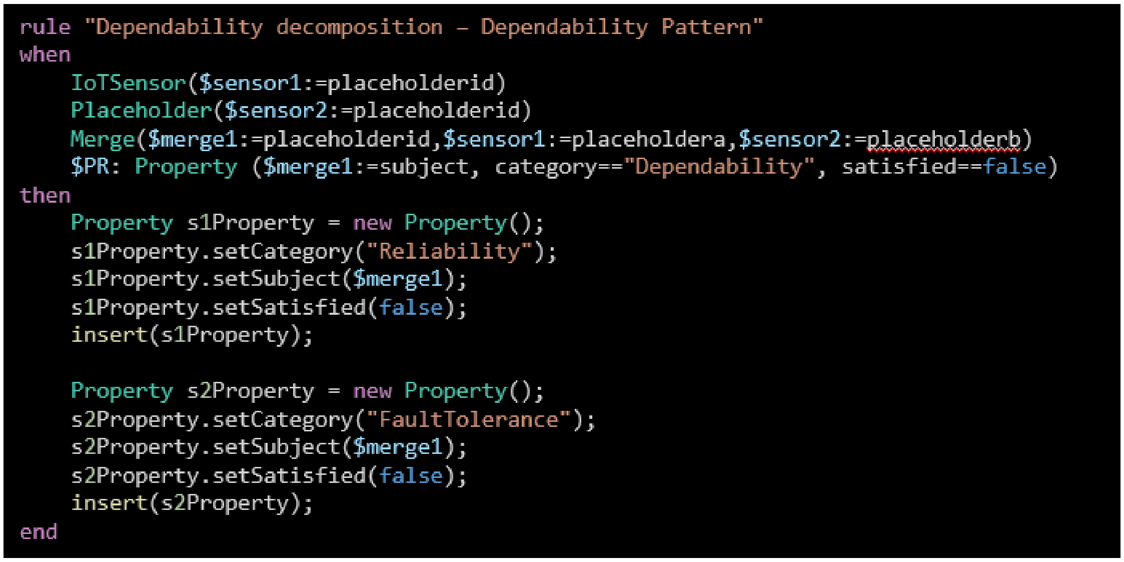

Dependability decomposition
A
Fault Tolerance property creation
Reliability property creation
B1
Fault Tolerance decomposition
Creates Heartbeat properties
B2
Heartbeat verification
Verifies Heartbeat properties
B3
Fault Tolerance verification
Reliability verification
C
D
Dependability verification
Verifies Fault Tolerance properties
Verifies Reliability properties

FIGURE 36. PATTERN-BASED DEPENDABILITY REASONING PROCESS (PROPERTY DECOMPOSITION & VERIFICATION)

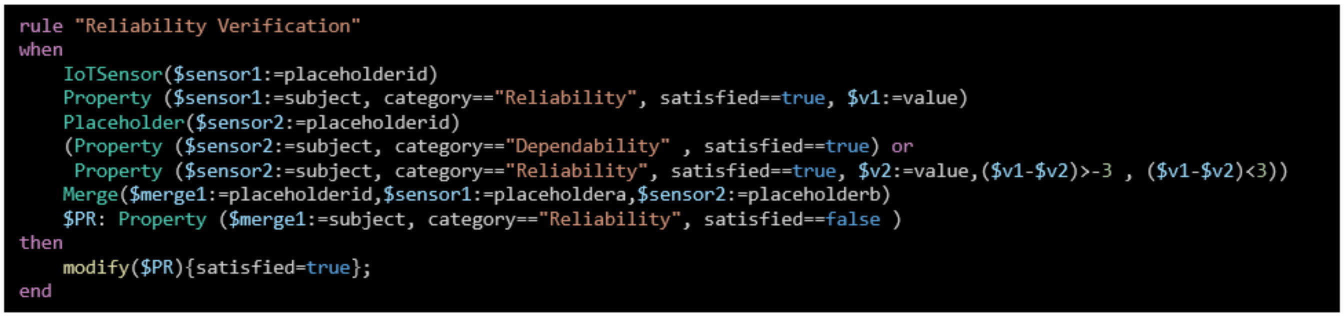
As shown in the figure above, the first pattern-Drools rule that is triggered is the Dependability Decomposition ("A" in Figure 36), through which two new properties are created: Fault Tolerance and Reliability. This rule is shown in Figure 37.



```
rule "Dependability decomposition - Dependability Pattern"
when
    IoTSensor($sensor1:=placeholderid)
    Placeholder($sensor2:=placeholderid)
    Merge($merge1:=placeholderid,$sensor1:=placeholdera,$sensor2:=placeholderb)
    $PR: Property ($merge1:=subject, category=="Dependability", satisfied==false)
then
    Property s1Property = new Property();
    s1Property.setCategory("Reliability");
    s1Property.setSubject($merge1);
    s1Property.setSatisfied(false);
    insert(s1Property);

    Property s2Property = new Property();
    s2Property.setCategory("FaultTolerance");
    s2Property.setSubject($merge1);
    s2Property.setSatisfied(false);
    insert(s2Property);
end
```

FIGURE 37. DEPENDABILITY DECOMPOSITION DROOLS RULE

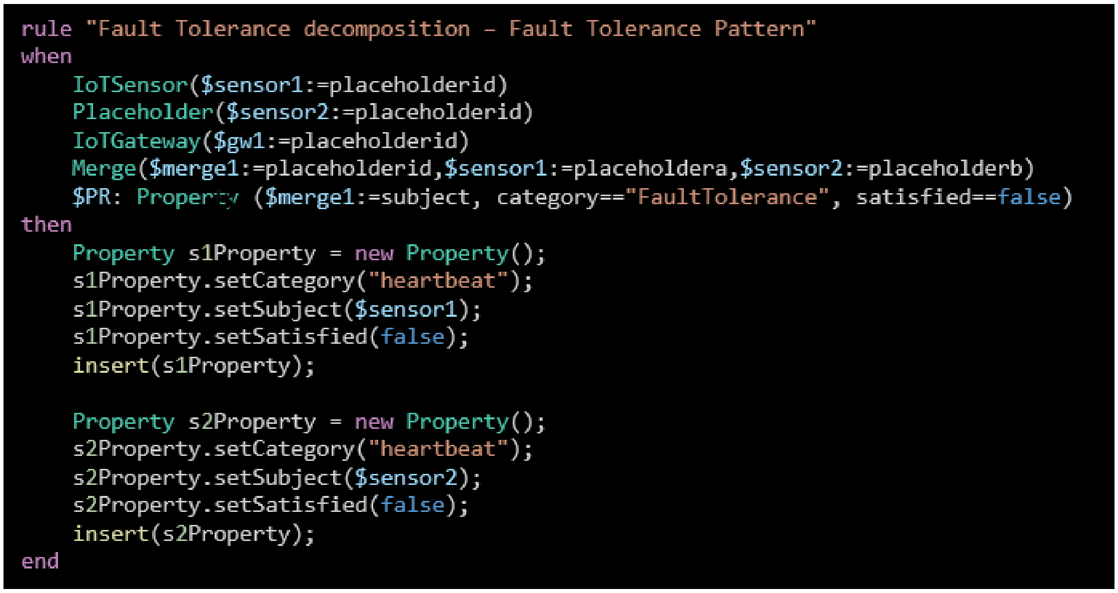
Subsequently, for the Reliability property a verification rule is needed ("C" in Figure 36), and one was defined accordingly. As shown in Figure 38, the defined rule applies on a merge orchestration of two sensors and assesses the deviation of their readings.



```
rule "Reliability Verification"
when
    IoTSensor($sensor1:=placeholderid)
    Property ($sensor1:=subject, category=="Reliability", satisfied==true, $v1:=value)
    Placeholder($sensor2:=placeholderid)
    (Property ($sensor2:=subject, category=="Dependability" , satisfied==true) or
     Property ($sensor2:=subject, category=="Reliability", satisfied==true, $v2:=value,($v1-$v2)>-3 , ($v1-$v2)<3))
    Merge($merge1:=placeholderid,$sensor1:=placeholdera,$sensor2:=placeholderb)
    $PR: Property ($merge1:=subject, category=="Reliability", satisfied==false )
then
    modify($PR){satisfied=true};
end
```

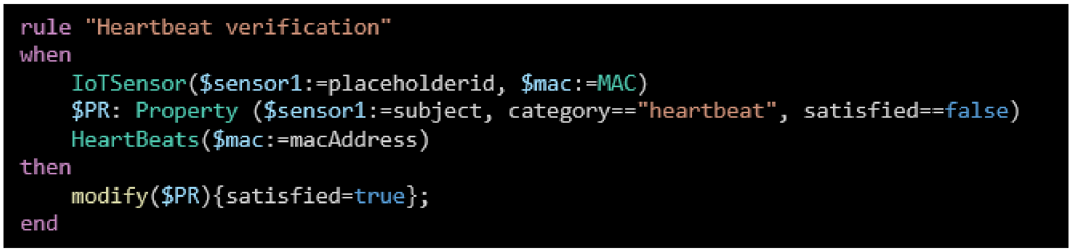
FIGURE 38. RELIABILITY VERIFICATION DROOLS RULE

The Fault Tolerance decomposition ("B1" in Figure 36), on the other hand, dictates the creation of 2 more heartbeat properties (as shown in Figure 39) and, therefore, two more verification rules are needed for Heartbeat verification ("B2" in Figure 36) and Fault Tolerance verification ("B2" in Figure 36). These two rules are shown in Figure 40 and Figure 41 respectively.



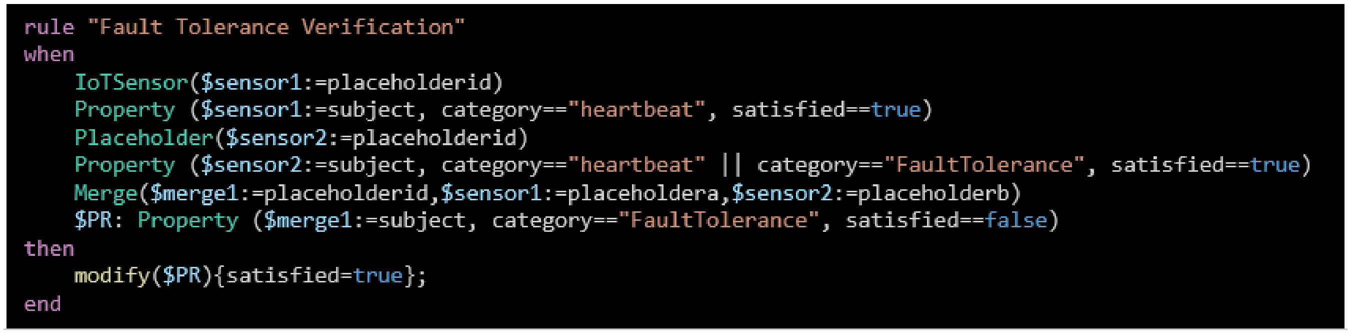
```
rule "Fault Tolerance decomposition - Fault Tolerance Pattern"
when
    IoTSensor($sensor1:=placeholderid)
    Placeholder($sensor2:=placeholderid)
    IoTGateway($gw1:=placeholderid)
    Merge($merge1:=placeholderid,$sensor1:=placeholdera,$sensor2:=placeholderb)
    $PR: Property ($merge1:=subject, category=="FaultTolerance", satisfied==false)
then
    Property s1Property = new Property();
    s1Property.setCategory("heartbeat");
    s1Property.setSubject($sensor1);
    s1Property.setSatisfied(false);
    insert(s1Property);

    Property s2Property = new Property();
    s2Property.setCategory("heartbeat");
    s2Property.setSubject($sensor2);
    s2Property.setSatisfied(false);
    insert(s2Property);
end
```

FIGURE 39. FAULT TOLERANCE DECOMPOSITION DROOLS RULE



```
rule "Heartbeat verification"
when
    IoTSensor($sensor1:=placeholderid, $mac:=MAC)
    $PR: Property ($sensor1:=subject, category=="heartbeat", satisfied==false)
    HeartBeats($mac:=macAddress)
then
    modify($PR){satisfied=true};
end
```

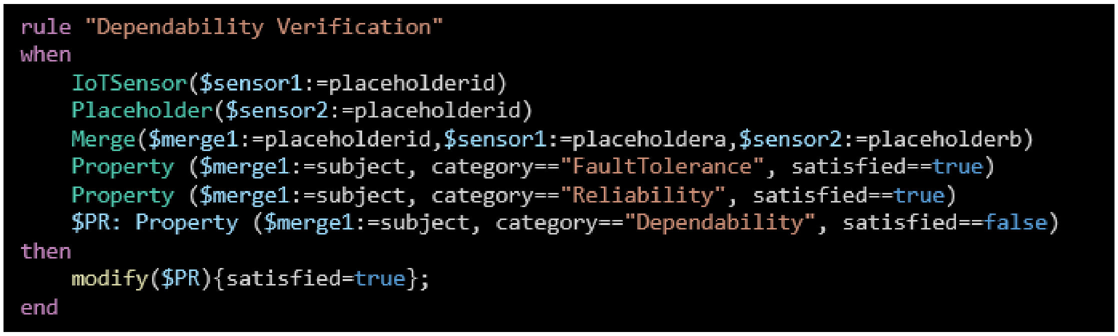
FIGURE 40. HEARTBEAT VERIFICATION DROOLS RULE



```
rule "Fault Tolerance Verification"
when
    IoTSensor($sensor1:=placeholderid)
    Property ($sensor1:=subject, category=="heartbeat", satisfied==true)
    Placeholder($sensor2:=placeholderid)
    Property ($sensor2:=subject, category=="heartbeat" || category=="FaultTolerance", satisfied==true)
    Merge($merge1:=placeholderid,$sensor1:=placeholdera,$sensor2:=placeholderb)
    $PR: Property ($merge1:=subject, category=="FaultTolerance", satisfied==false)
then
    modify($PR){satisfied=true};
end
```

FIGURE 41. FAULT TOLERANCE VERIFICATION DROOLS RULE

The verification of the Heartbeat property takes place whenever there is a heartbeat originated from our sensors. The verification of a Fault tolerance property of a merge of 2 sensors takes place whenever both have a heartbeat property that is valid. These rules lead to the final Dependability Verification rule ("D" in Figure 36) which, if triggered, verifies the dependability property. For this to happen, both Fault Tolerance and Reliability properties must be valid, as shown in Figure 42.



```
rule "Dependability Verification"
when
    IoTSensor($sensor1:=placeholderid)
    Placeholder($sensor2:=placeholderid)
    Merge($merge1:=placeholderid,$sensor1:=placeholdera,$sensor2:=placeholderb)
    Property ($merge1:=subject, category=="FaultTolerance", satisfied==true)
    Property ($merge1:=subject, category=="Reliability", satisfied==true)
    $PR: Property ($merge1:=subject, category=="Dependability", satisfied==false)
then
    modify($PR){satisfied=true};
end
```

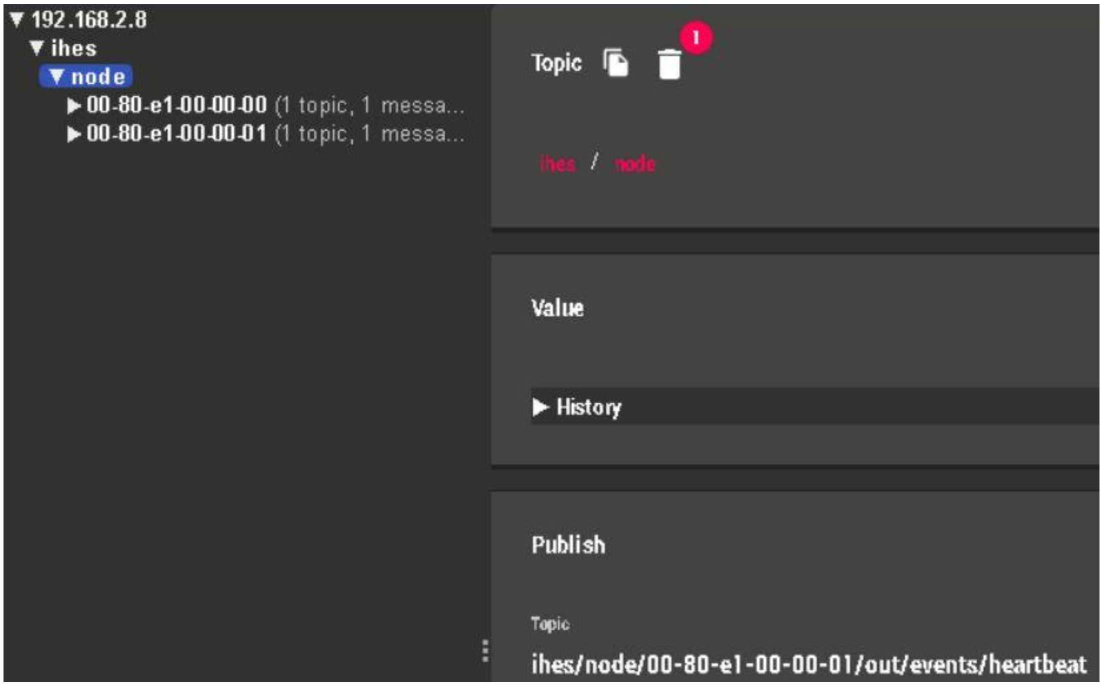
FIGURE 42. DEPENDABILITY VERIFICATION DROOLS RULE

6.4 Validation

6.4.1 LOCAL TESTBED VALIDATION

A local testbed was setup to integrate and test the sub use case 2, prior to integration with the main UC3 testbed, emulating the topology shown in Figure 32. 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 Field Pattern Engine include Java, Maven and Spring Boot. 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 33 and Figure 34, excluding interactions with backend components (which were implemented in Cycle 2).

Figure 43 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.



▼ 192.168.2.8
  ▼ ihes
    ▼ node
      ▶ 00-80-e1-00-00-00 (1 topic, 1 messa...
      ▶ 00-80-e1-00-00-01 (1 topic, 1 messa...

Topic

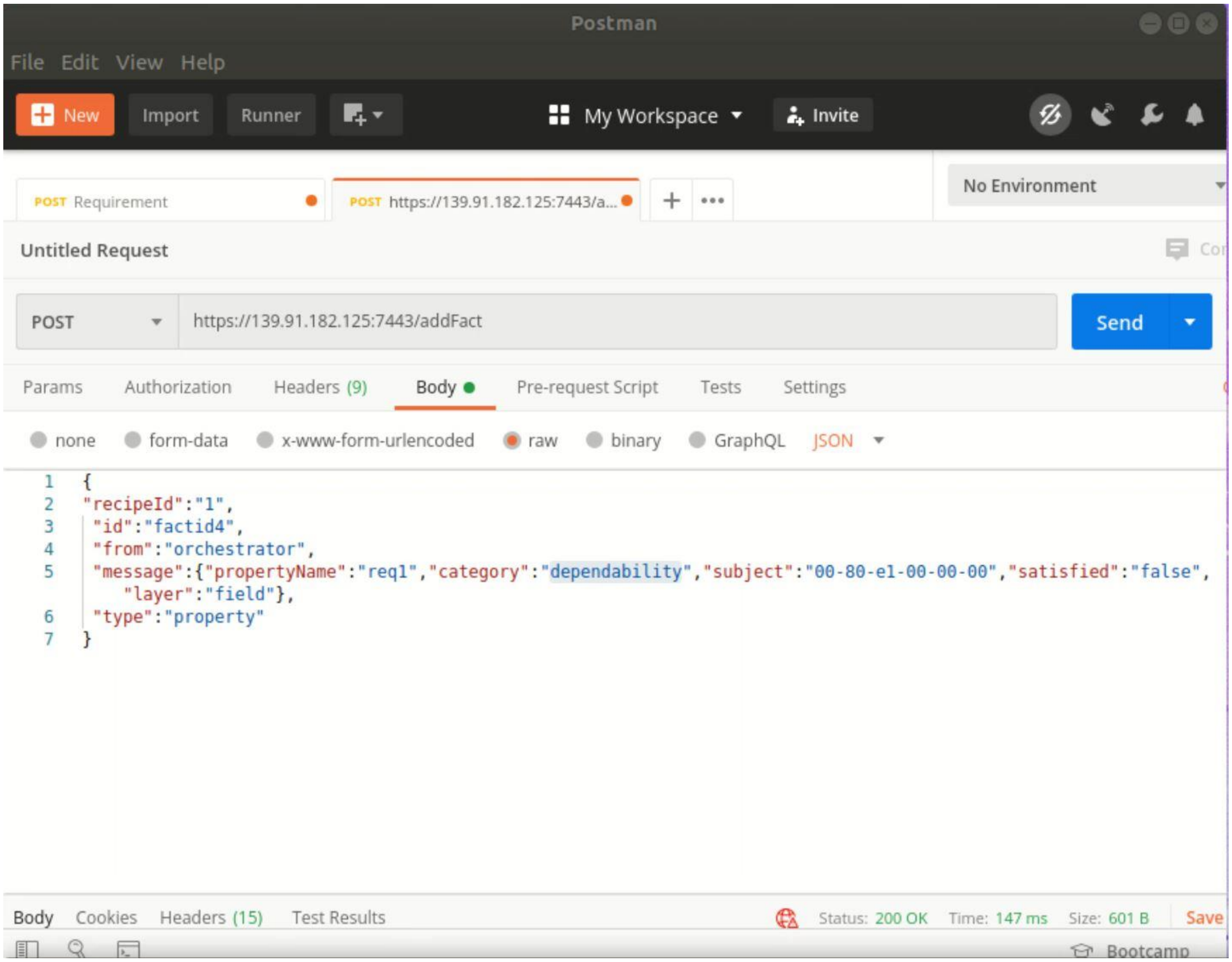
ihes / node

Publish

Topic
ihes/node/00-80-e1-00-00-01/out/events/heartbeat

FIGURE 43. SIMULATED IHES NODES (PUBLISHING HEARTBEATS AT IHES/NODE/$MAC/OUT/EVENTS/HEARTBEAT TOPIC)

Figure 44 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 compo nent 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.



POST https://139.91.182.125:7443/addFact

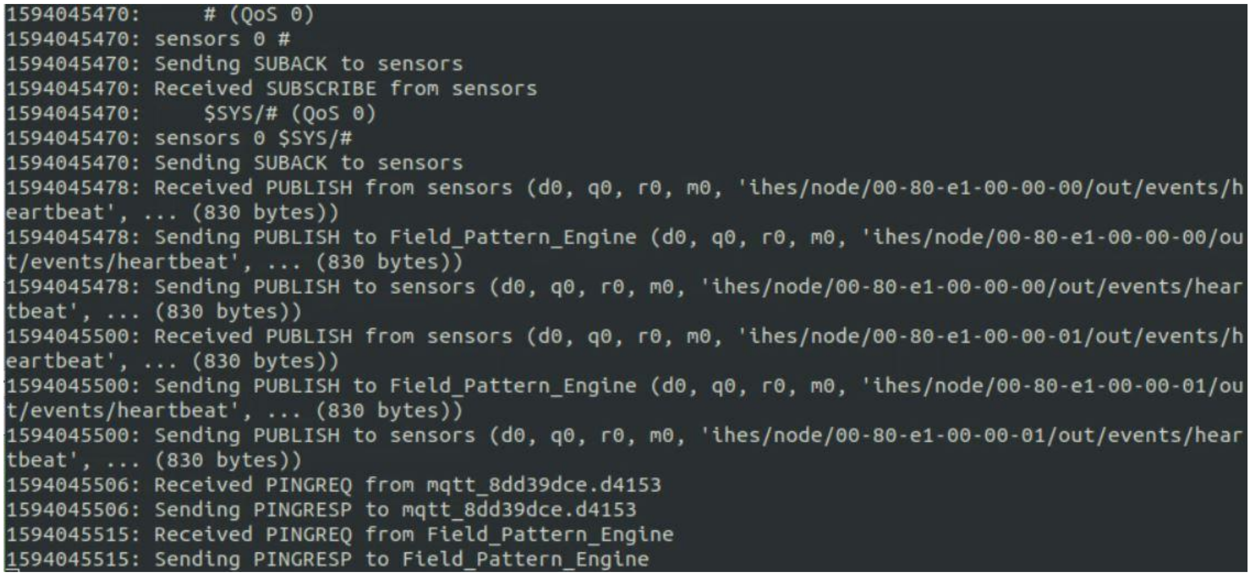
Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
"recipeId":"1",
 "id":"factid4",
 "from":"orchestrator",
 "message":{"propertyName":"req1","category":"dependability","subject":"00-80-e1-00-00-00","satisfied":"false",
    "layer":"field"},
 "type":"property"
}
```

Status: 200 OK Time: 147 ms Size: 601 B

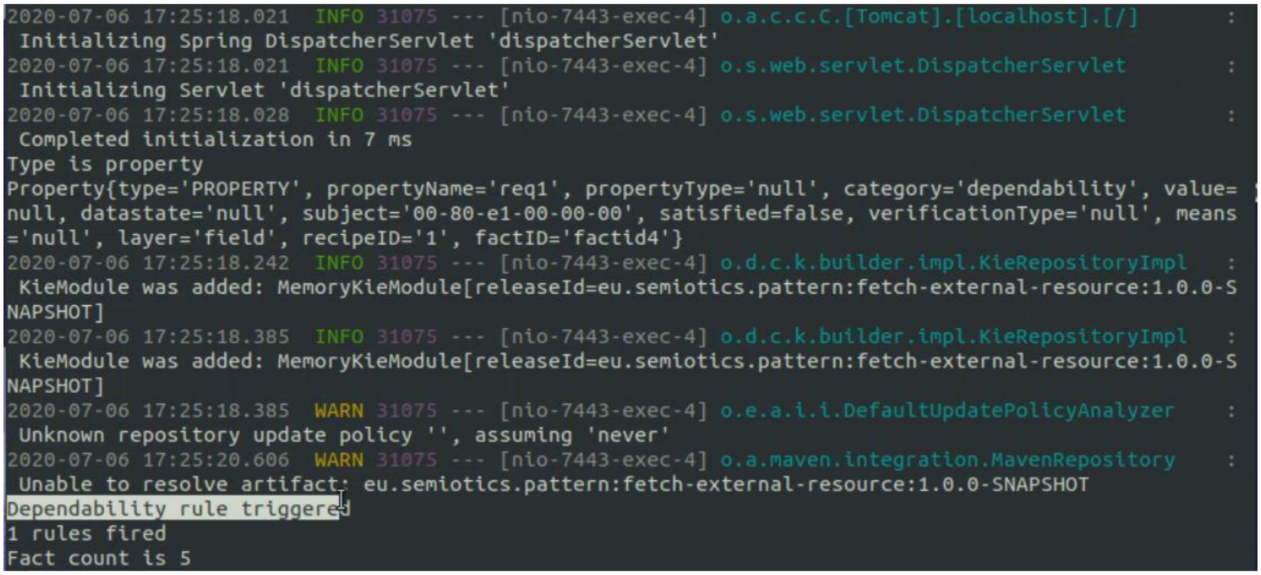
FIGURE 44. PATTERN ORCHESTRATION INFORMATION RELAYED TO FIELD PATTERN ENGINE

Finally, Figure 45 and Figure 46 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.



```
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 45. 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
```

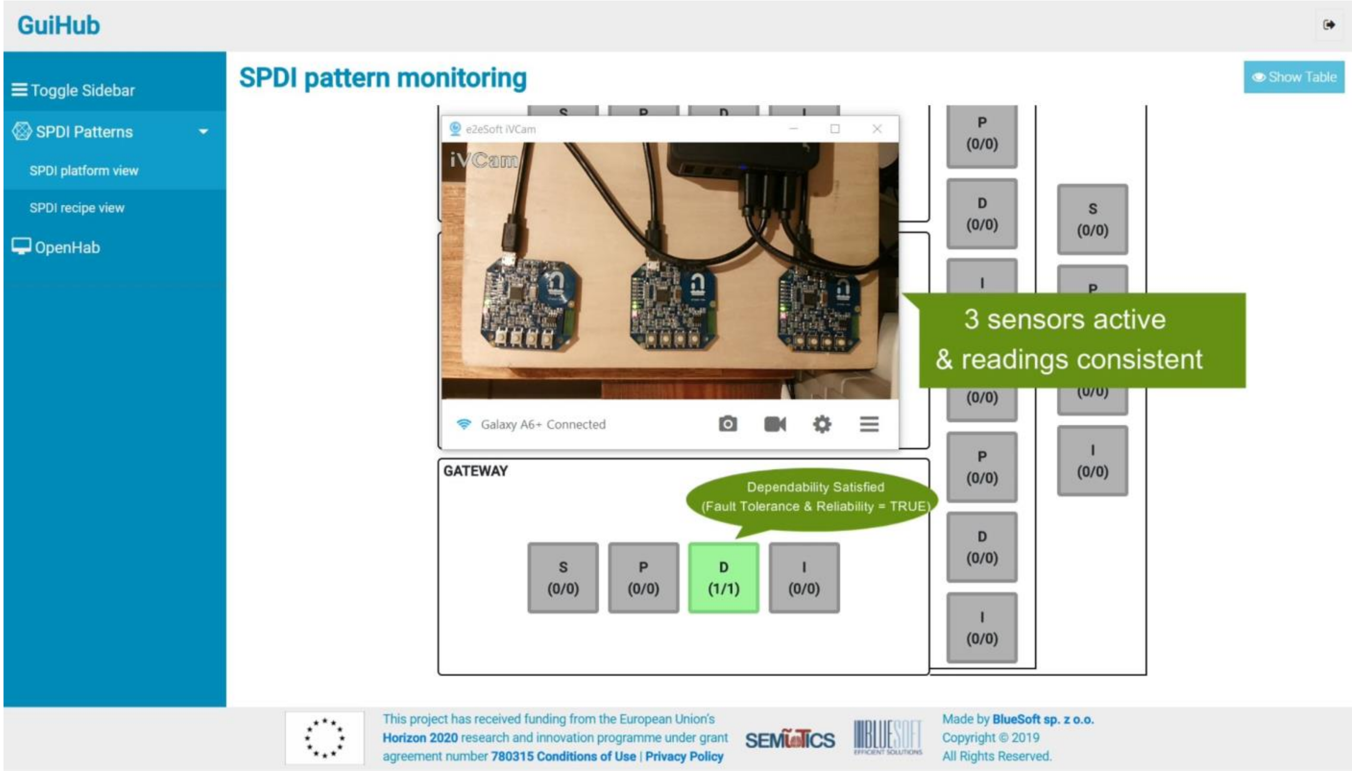
FIGURE 46. FIELD PATTERN ENGINE (SUBSCRIBED TO BROKER; RECEIVING UPDATES & REASONING ON DEPENDABILITY PROPERTY)

6.4.2 INTEGRATED TESTBED VALIDATION & DEMONSTRATION

Following the local testing, the implemented pattern components were transferred and integrated into the main UC3 testbed, while elaborating on the capabilities both from the perspective of the Dependability property verification and the associated pattern rules (to support the full set of reasoning capabilities presented in subsection 6.3.2), as well as from the perspective of the deployment and integration with the SEMIoTICS backend components involved in the sub use case (namely, the Pattern Orchestrator and the SEMIoTICS GUI). The final testbed configuration was as defined in section 3 above.

The use of the said testbed implements a scenario, aiming to demonstrate and validate the full coverage of the interactions specified in subsection 6.3, the reasoning capabilities presented in subsection 6.3.2, and the satisfaction of the sub use case 3 objectives as defined in subsection 6.2. This validation and demonstration process has been recorded in the form of a video integrating both the testbed view, as well as the GUI view into a single screen (to facilitate presentation). The key steps of this process are presented below, including the associated scenes from said video recording.

As shown in Figure 47 and Figure 48, at the starting phase ("Phase 0") all 3 sensors are active (see insert of actual setup in said figure) and their readings are consistent; therefore, the GUI shows the Dependability property is satisfied (shown with green color and marking 1/1 properties as satisfied).



This project has received funding from the European Union's **Horizon 2020** research and innovation programme under grant agreement number **780315 Conditions of Use** | **Privacy Policy**

Made by **BlueSoft sp. z o.o.**
Copyright © 2019
All Rights Reserved.

FIGURE 47. STARTING PHASE – ALL 3 SENSORS ACTIVE & READINGS CONSISTENT (DEPENDABILITY SATISFIED)

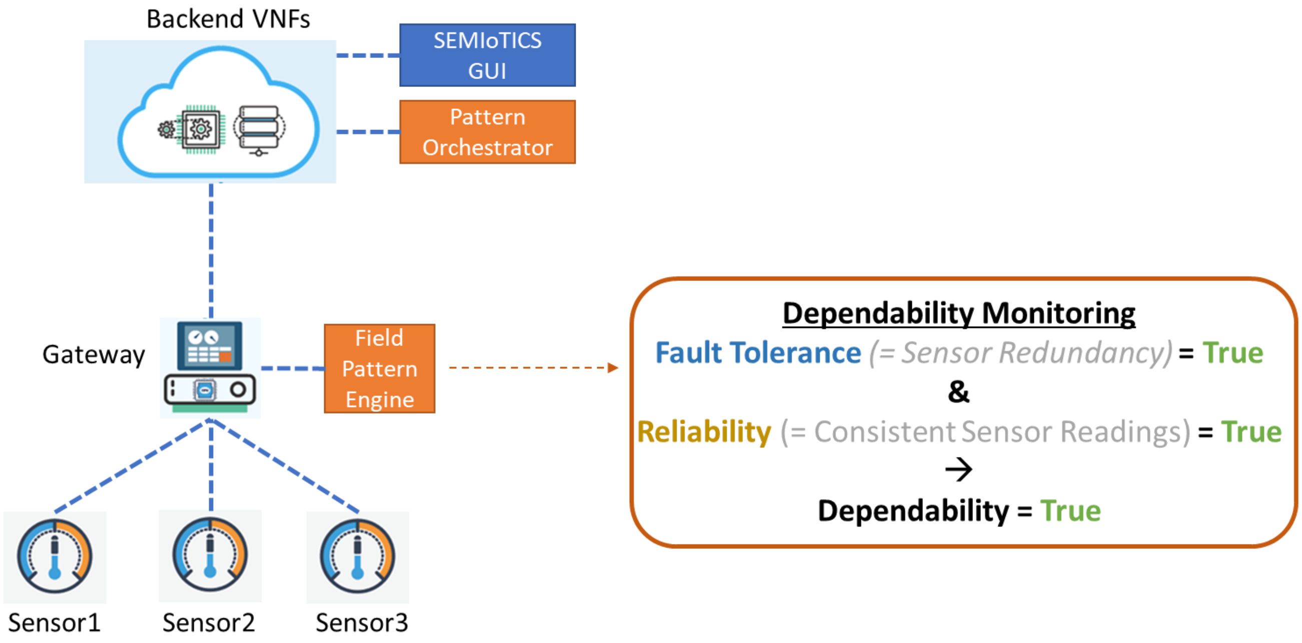
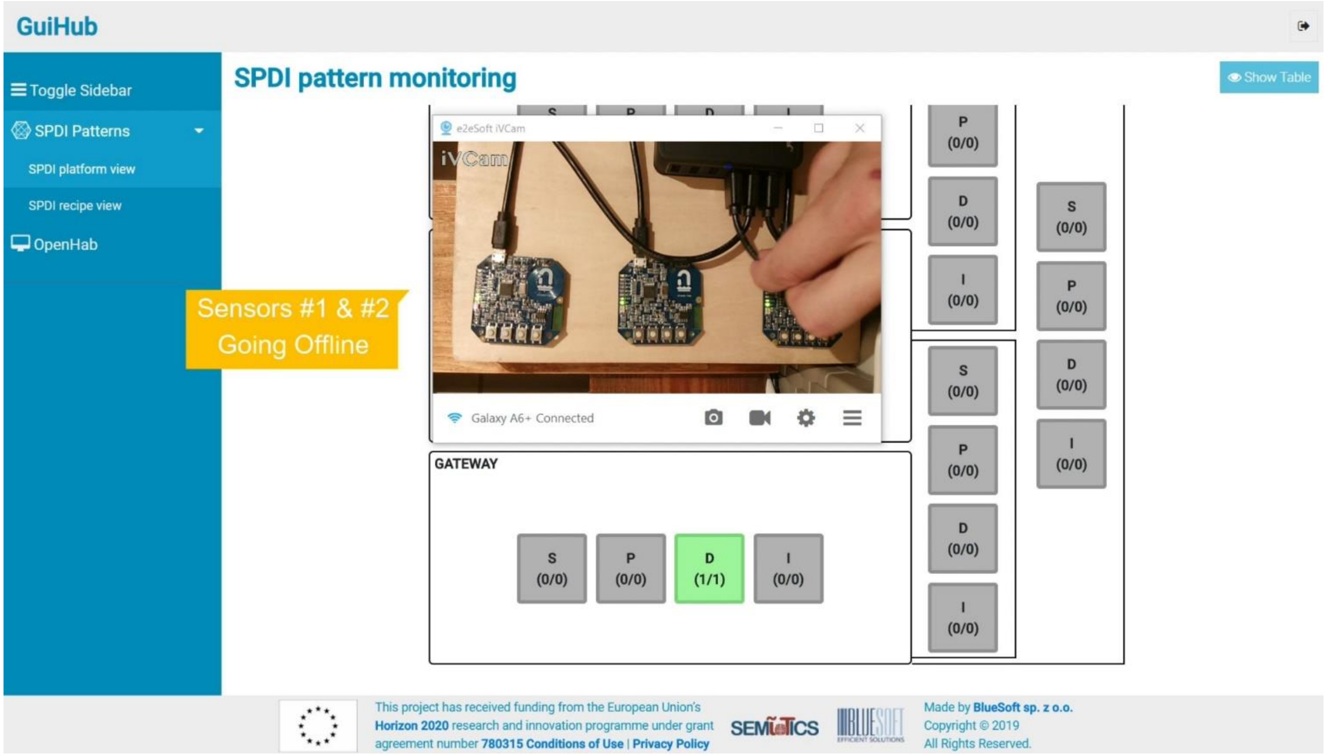


FIGURE 48. STARTING PHASE (PHASE 0) OVERVIEW AND PROPERTY REASONING

As a first step post-instantiation (Phase 1; see Figure 49), sensors #1 and #2 are disconnected to emulate the sensors' failure (e.g., batteries are depleted, or the sensors are destroyed, either simultaneously or within a relatively short timeframe). Once a number of "heartbeats" are not received by those two sensors at the Pattern Engine (through monitoring the corresponding topics at the MQTT Broker), a reasoning process is triggered to re-assess the satisfaction of the Fault Tolerance property (and by extension the Reliability and Dependability properties). Since the setup now only has one operational sensor, neither fault tolerance nor reliability can be guaranteed. Therefore, the Dependability fails. An overview of this Phase and the associated properties' status reasoning are shown in Figure 50. The view of this property change, as visualized on the SEMIoTICS GUI, is depicted in Figure 51.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 Conditions of Use | Privacy Policy

Made by BlueSoft sp. z o.o. Copyright © 2019 All Rights Reserved.

FIGURE 49. PHASE 1 – SENSORS #1 & #2 GOING OFFLINE

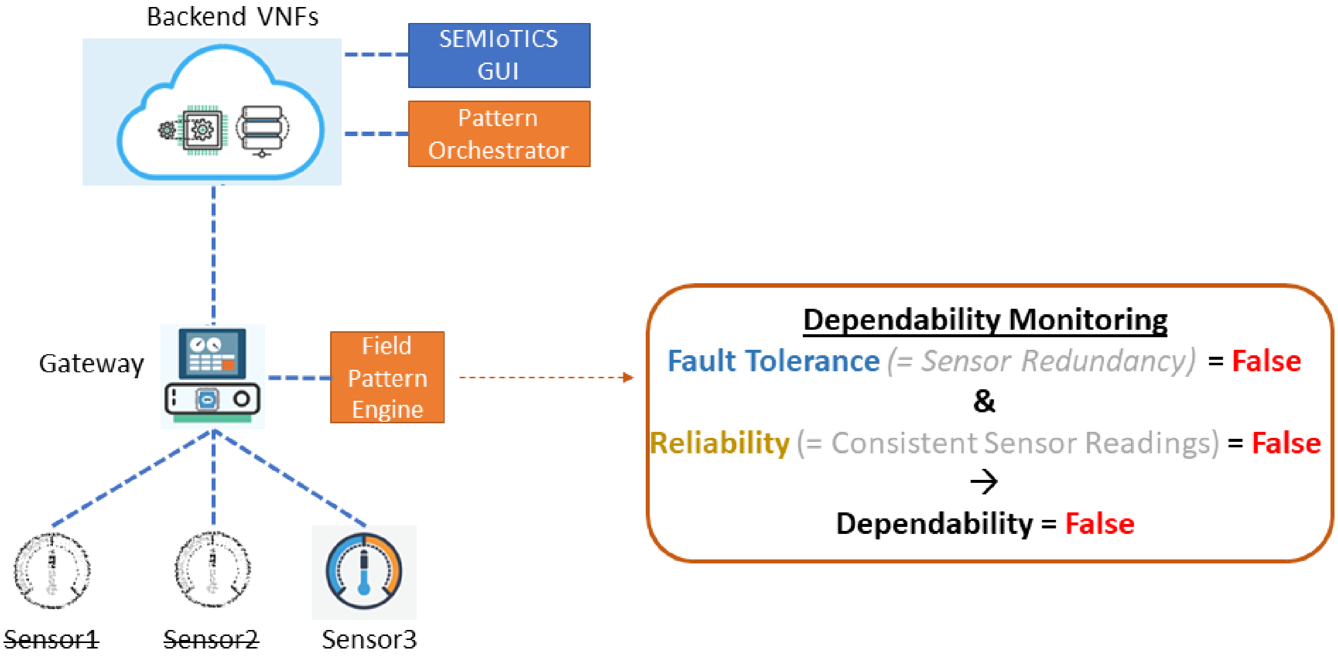
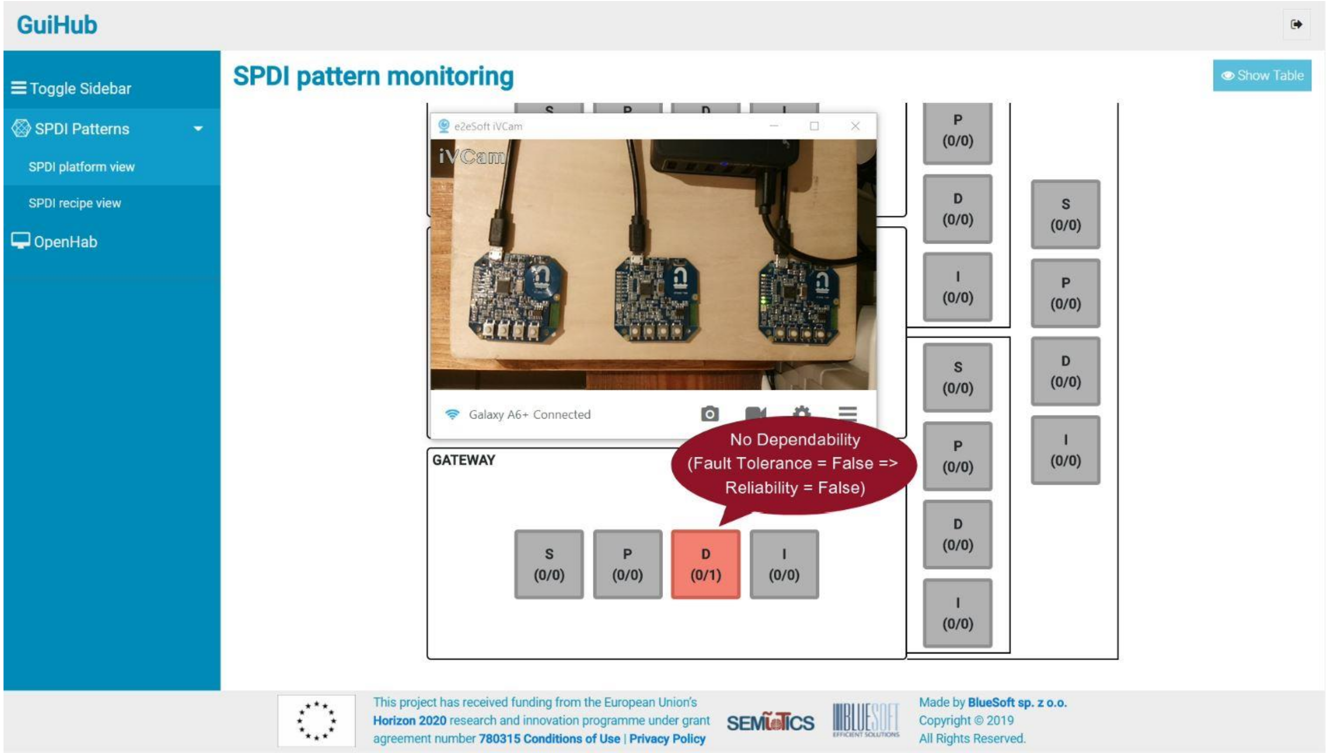


FIGURE 50. PHASE 1 OVERVIEW AND PROPERTY REASONING

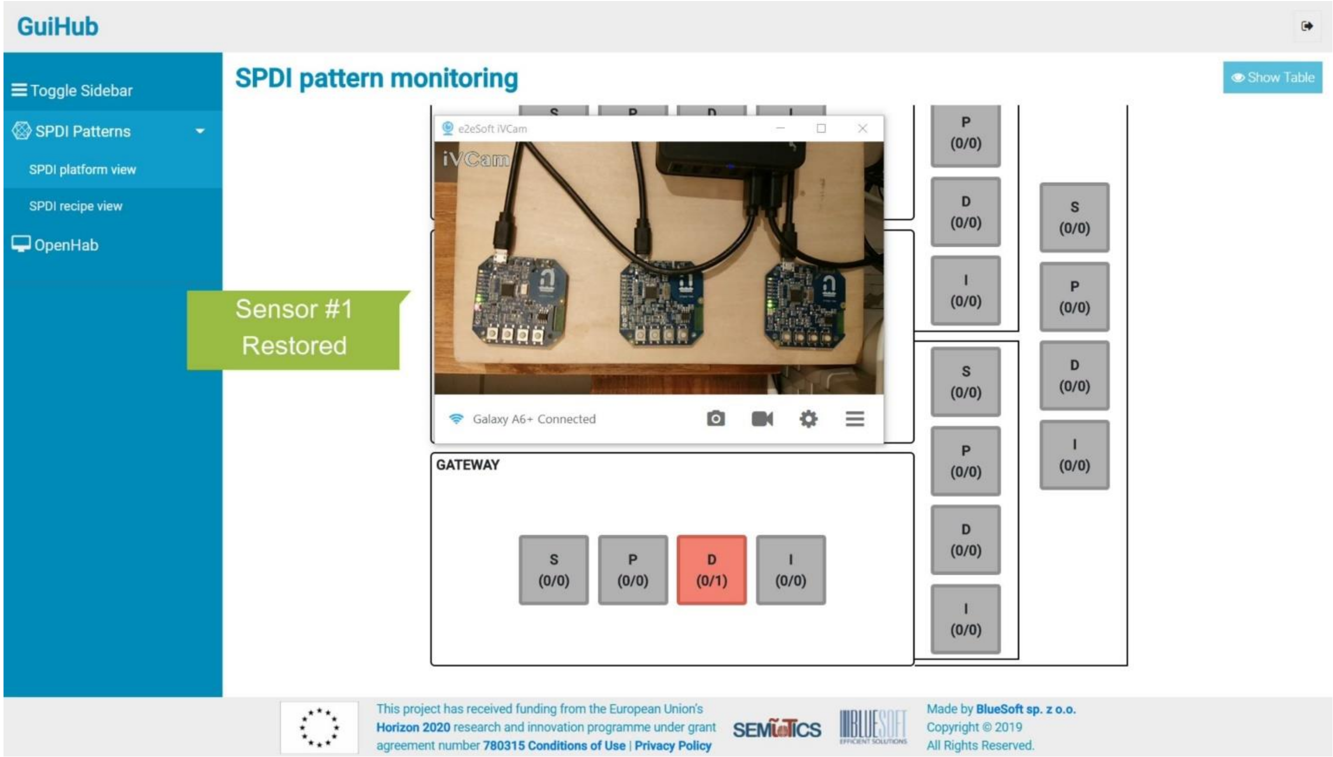


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 Conditions of Use | Privacy Policy

Made by BlueSoft sp. z o.o.
Copyright © 2019
All Rights Reserved.

FIGURE 51. PHASE 1 – DEPENDABILITY PROPERTY FAILURE (GUI VISUALISATION)

On Phase 2, Sensor #1 is restored (see Figure 52). As the Field Pattern Engine resumes receiving “heartbeats” from that sensor, reasoning is triggered and the Fault Tolerance (and, consequently, the Dependability property) is restored, as two sensors are now active (Sensors #1 and #3); see the properties’ status in Figure 53 and the associated GUI visualization in Figure 54.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 Conditions of Use | Privacy Policy

Made by BlueSoft sp. z o.o.
Copyright © 2019
All Rights Reserved.

FIGURE 52. PHASE 2 – SENSOR #1 RESTORED

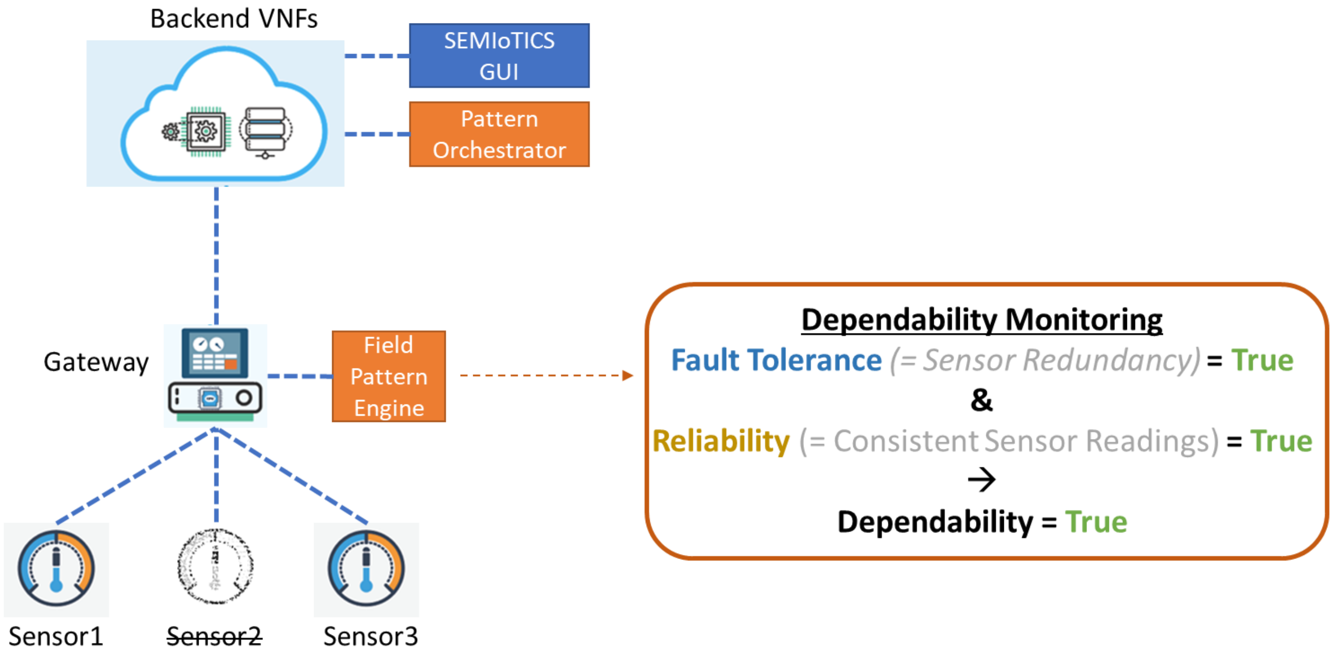
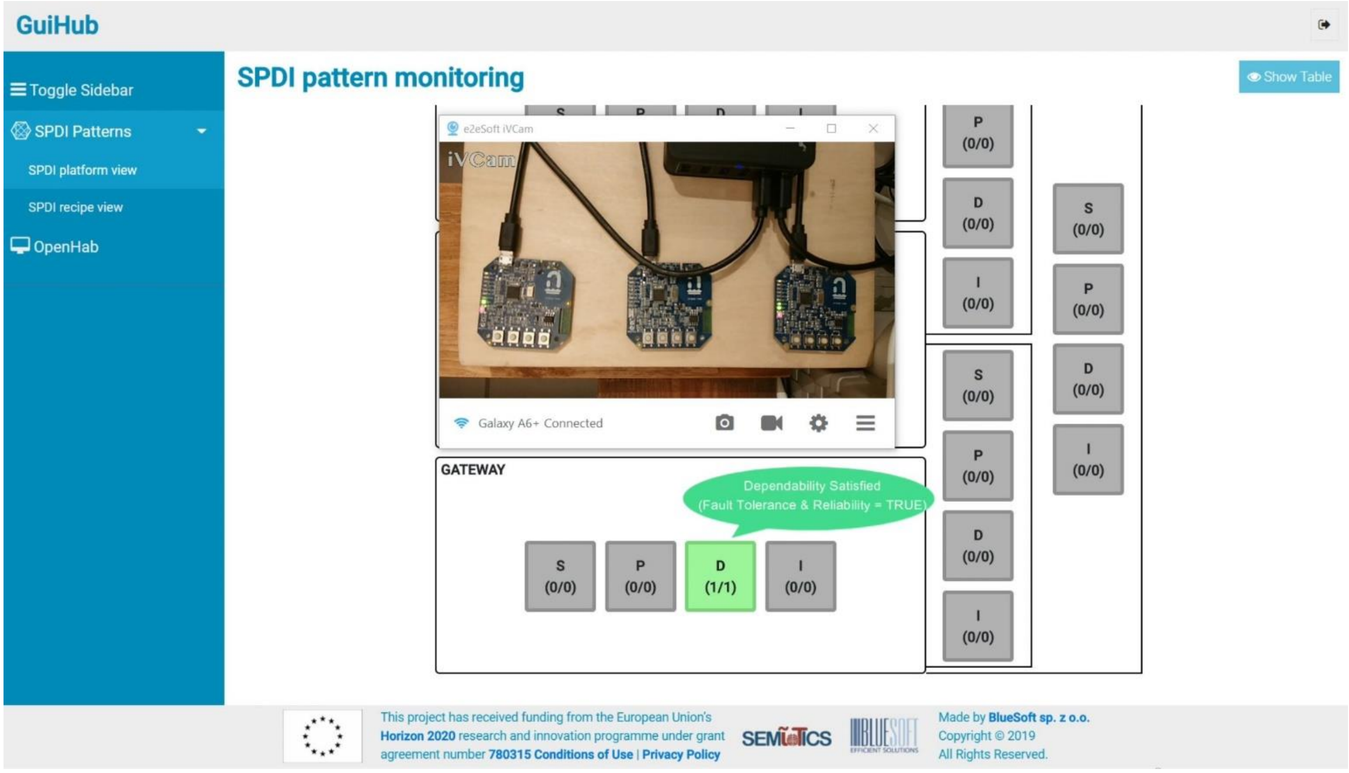


FIGURE 53. PHASE 2 OVERVIEW AND PROPERTY REASONING



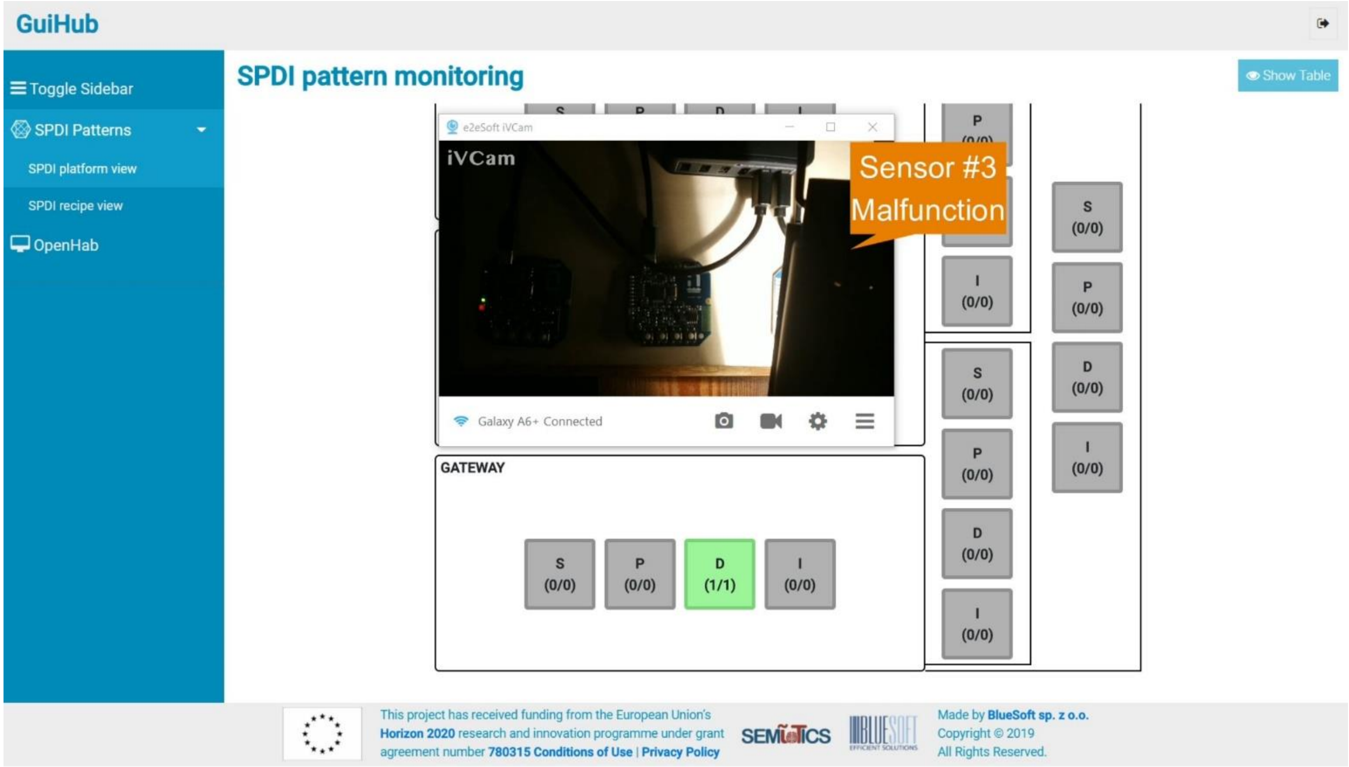
Dependability Satisfied
(Fault Tolerance & Reliability = TRUE)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 Conditions of Use | Privacy Policy

Made by BlueSoft sp. z o.o.
Copyright © 2019
All Rights Reserved.

FIGURE 54. PHASE 2 – DEPENDABILITY PROPERTY RESTORED (GUI VISUALISATION)

In the 3rd and final phase, one of the two remaining operational sensors begins to malfunction (see Figure 55). More specifically, we introduce a fault in the sensing of Sensor #3, causing a deviation between the readings between said active sensors. This triggers a reasoning on the Reliability property (and consequently the Dependability properties) at the Field Pattern Engine. As shown in Figure 56, due to the significant deviation in the sensor readings, the Reliability property is not satisfied. Moreover, as one sensor is offline (Sensor #2) and another sensor is malfunctioning (Sensor #3), only one sensor remains operational (Sensor #1). Consequently, the Dependability property is not satisfied either. This change is promptly reflected on the SEMIoTICS GUI, as shown in Figure 57.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 Conditions of Use | Privacy Policy

Made by BlueSoft sp. z o.o.
Copyright © 2019
All Rights Reserved.

FIGURE 55. PHASE 3 – SENSOR #3 MALFUNCTION (ERRONEOUS READINGS)

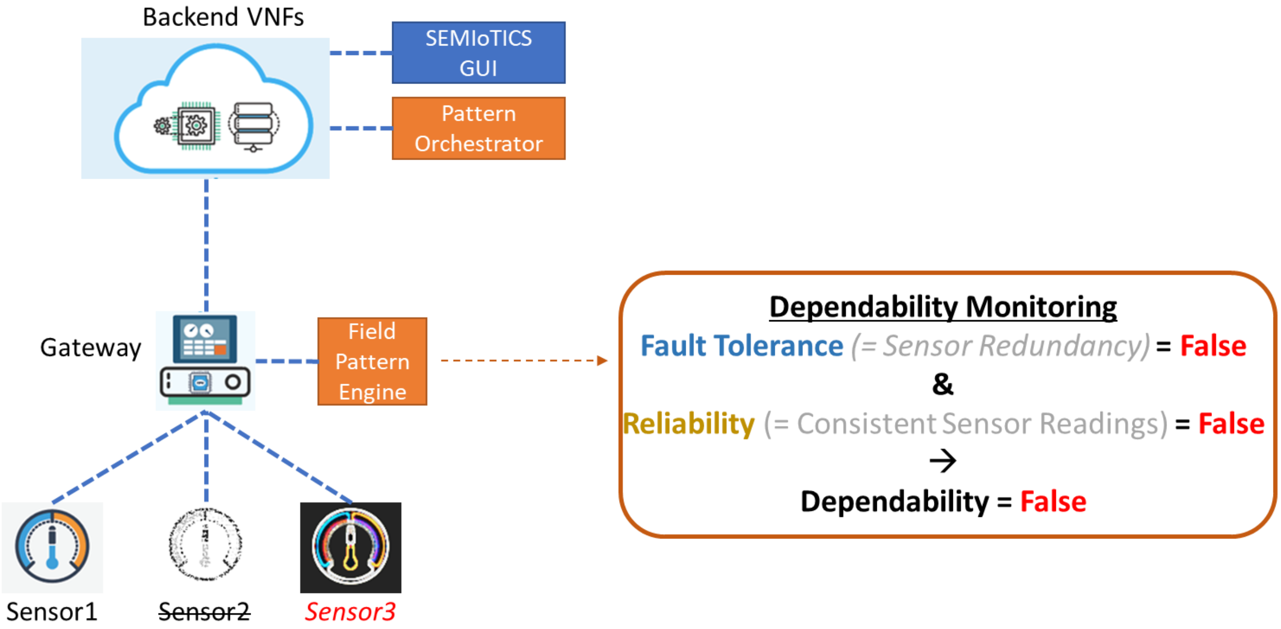
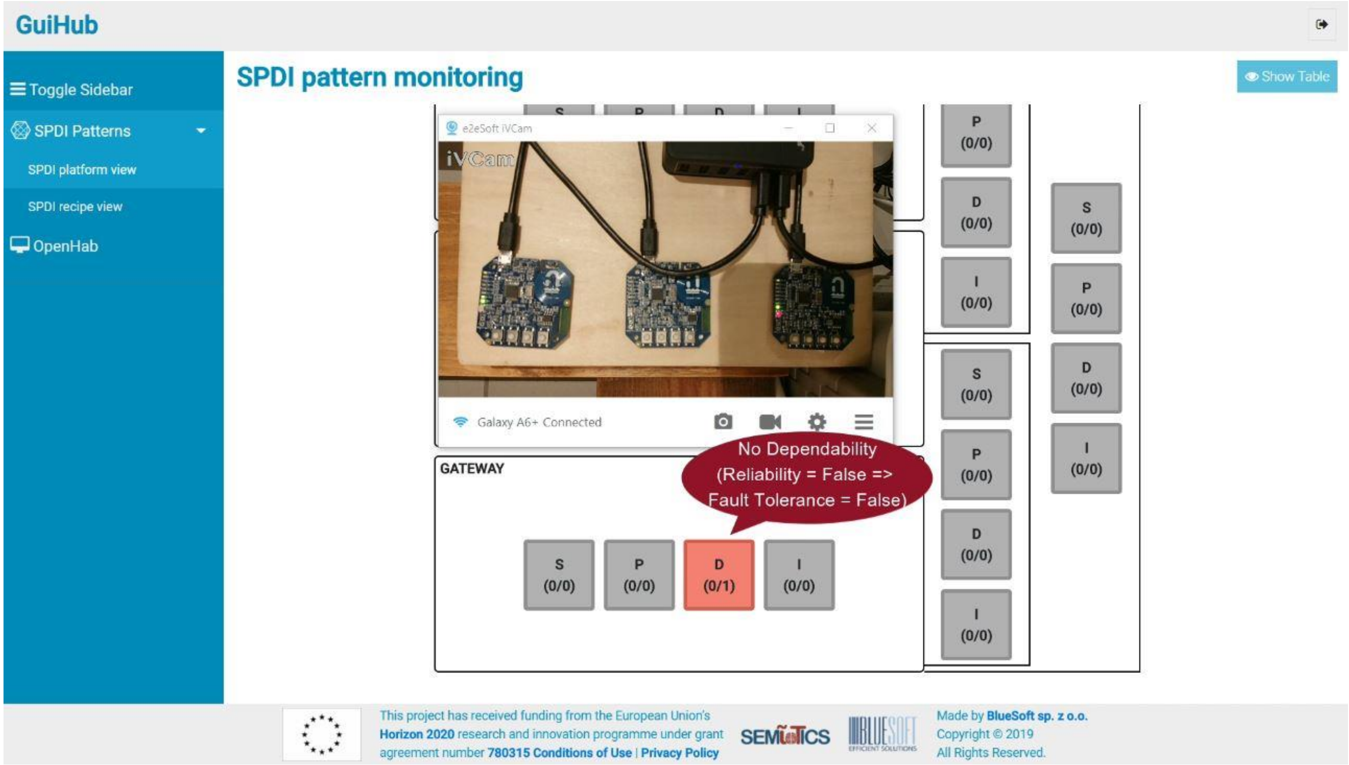


FIGURE 56. PHASE 3 OVERVIEW AND PROPERTY REASONING



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 Conditions of Use | Privacy Policy

Made by BlueSoft sp. z o.o.
Copyright © 2019
All Rights Reserved.

FIGURE 57. PHASE 3 – DEPENDABILITY PROPERTY FAILURE (GUI VISUALISATION)

7 OVERALL KPIS AND REQUIREMENTS VALIDATION

In this section are reported the UC3 applicable KPIs derived from the project wide table in section 3 of deliverable D5.1. In Table 1 all the requirements impacting UC3 are reported as an extract from the project wide one. All together these KPIs contributes to 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.

The planned methodology of KPI evaluation is contained in D5.1 and is omitted here for brevity.

7.1 UC3 Related Requirements

In this section we consider the set of requirements that were originally described in deliverable D2.3 [10] within the context of the SEMIoTICS framework. Furthermore, some key requirements pertinent to the pattern-driven operation demonstrated within UC3 are also included.

TABLE 1. REQUIREMENTS PLACED ON USE CASE 3 AS PER D2.3

Req-ID	Description	Reference	Fulfilment
R.GP.1	End-to-end connectivity between the heterogeneous IoT devices (at the field level) and the heterogeneous IoT Platforms (at the backend cloud level)	D2.5, D5.11	This has been achieved by deriving the UC3 architecture from SEMIoTICS one has it has been described in D2.5. In particular it was implemented and validated in sub-use case 2 (Section 5.3) and sub-use case 3 (Section 6.3) by means of the MQTT infrastructure added to pattern infrastructure (mainly pattern engine) and the pattern GUI and the global MQTT broker on the Cloud Level Data Aggregation component. It was implemented, tested and validated during deployment of sub-usecase 2 and sub-usecase 3 in UC3 testbed.
R.GP.2	Scalable infrastructure due to the fast-paced growth of IoT devices	Section 3	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 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. This requirement has been implemented as part of the UC3 testbed derived

			from SEMIoTICS Architecture. It has been implemented in a scalable manner thanks to virtualization provided on top of the VIM and NFV components by virtualizing at field and backend level the required UC3 components (i.e. pattern-engine, IHES Devices, IHES Supervisor And LocalDB, at field level and Pattern Orchestrator, GUI, OpenHAB + UC3 GUI at backend level). Also local embedded analytics contributes to fulfill this requirements due to the distributed by design topology. It was implemented, tested and validated during deployment of sub-usecase1, sub-usecase 2 and sub-usecase 3 in UC3 testbed.
R.GP.3	High adaptation capability to accommodate different QoS connectivity needs (e.g., low latency, reliable communication)	D3.1, D3.2, D3.5	This has been achieved thanks to the VIM Connector, NFV Orchestrator, Virtualized Infrastructure Manager, and VNF Manager components introduced in D3.1 and D3.2 and integrated into UC3 testbed. It was implemented, tested and validated as part of the deployment of the UC3 testbed.
R.NL.1/R.BC.1	Controller Node requirement: At least 6 CPU cores and 32 GB RAM	Section 3.2	This is in line with the requirements that have already reported above for e.g., the OpenStack controller. Note that, in addition, it is required that the controller nodes have around 100 GB of storage memory see section 3.2. It was verified, tested and validated as part of the deployment of the UC3 testbed.
R.NL.2/R.BC.2	Controller Node requirement: At least 2 Network interfaces	Section 3	This is a requirement applicable to the computer holding the controller, e.g., the OpenStack controller, see section 3.3 It was verified, tested and validated as part of the deployment of the UC3 testbed.
R.NL.3/R.BC.3	Controller Node Requirement: Linux OS	Section 3	The controller nodes at the NFV MANO context, e.g., the OpenStack controller, require Linux OS.

			UC3 testbed runs on Linux OS
R.BC.4	Controller Node Requirement: Solid State Disk (SSD) of at least 256 GB	Section 3	SSD storage used exclusively in UC3 testbed
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.	Section 3.2	This is fulfilled or in line with the requirements for the compute nodes of the NFVI, see section 3.2. It was verified, tested and validated as part of the deployment of the UC3 testbed.
R.NL.6/R.BC.8/ R.BC.9	Hypervisor Nodes: KVM and Linux Containers (LXD) must be supported by the Hypervisor Linux OS	Section 3.2	This is fulfilled in the compute nodes that form the NFVI, e.g., at the cloud level. It was implemented, tested and validated as part of the deployment of the UC3 testbed.
R.BC.11	Virtual Network requirement: Support for GRE, VLAN, and VXLAN tunnels for virtual tenant networking.	Section 3.3	UC3 leverages Virtual Tenant Networking through Neutron APIs, which support GRE, VLAN and VXLAN protocols. VXLAN tunnels were leveraged in UC3. It was implemented, tested and validated as part of the deployment of the UC3 testbed.
R.BC.12/R.NL.8	The VIM and Virtual Network frameworks must support Interfaces that enable VM tenant networking	Section 3.3	UC3 Testbed supports Virtual Tenant Networking. It was implemented, tested and validated as part of the deployment of the UC3 testbed.
R.BC.13/R.NL.9	Interface between the VIM and the SDN controller to allow Tenant Network Slicing	Section 3.3	Interfacing between the VIM and SDN is accomplished through Neutron APIs, which facilitates Virtual Tenant Networking in UC3. It was implemented, tested and validated as part of the deployment of the UC3 testbed.
R.BC.14/R.NL.10	Interfaces among the MANO entities (NFO, RO, NFVO) and the VIM must ensure seamless interoperability among different entities of the Backend Cloud	Section 3.3	Interfacing between backend cloud VNFs (e.g., for data visualization, aggregation, and analytics) and gateway VNFs is accomplished via Virtual Tenant Networking. It was implemented, tested and validated as part of the deployment of the UC3 testbed. In particular for the realization of sub-usecase 2 and sub-usecase 3 for dependability

			patterns and cloud level data aggregation facilities.
R.BC.16	Prediction mechanism based on the data stored in the cloud	Section 5	This has been addressed in UC3 sub scenario 2 “collaborative edge cloud for avalanche warning” It was implemented, tested and validated as part of the deployment sub-usecase 2 scenario. In particular the prediction mechanism was tested by verifying the algorithms to the specific case of computing temperature trends at cloud level, from information provided by devices at IHES field-level.
R.BC.17	Quantitative/statistical analysis of the cloud data	Section 5	This has been addressed in UC3 sub scenario 2 “collaborative edge cloud for avalanche warning” by predicting temperature trends from IHES device readings from UC3 field level. It was verified as part of the avalanche warning scenario.
R.BC.19	The backend layer should feature pattern-driven cross-layer orchestration capabilities	D4.8	UC3 – and sub-use case 3 in specific, focuses on Field Layer and only involves a simple pattern-driven orchestration that is defined at bootstrap phase; nevertheless, the Pattern Orchestrator component deployed within said sub use case to enable this functionality can be leveraged to provide more complex cross-layer orchestration capabilities, if needed. It was implemented, tested and validated as part of the deployment sub-usecase3 and by implementing the 3 specific UC3 dependability patterns including interaction with the backend GUI.
R.BC.20	The backend layer must aggregate intra-layer as well as inter-layer SPDI status information to enable local and global intelligence reasoning and adaptation	D4.8, D4.2	The Pattern Orchestrator component deployed at the Backend layer (see sub use case 3) aggregates SPDI-related information from the Field layer. While this is only used for visualisation purposes (as the UC focused on Field layer autonomous reasoning), this aggregation of SPDI status information is a necessary enabler for the inter-layer reasoning

			(along with the deployment of a Backend pattern engine). It was implemented, tested and validated as part of the deployment sub-usecase3 and by implementing the 3 specific UC3 dependability patterns including interaction with the backend GUI.
R.FD.1	Field devices SHOULD be able to get data from the environment through sensors (sensors).	Sections 3 and 4	Sensing Devices based on STM32 CLOUD-JAM boards used in UC3 are equipped with both environmental and inertial sensors. This requirement is fulfilled thanks to the middleware firmware implemented in the STM32 Cloud-Jam devices and tested as part of the deployment of the sub-usecase1 (see also KPIs 4.1, 4.2, 4.4)
R.FD.2	Field devices SHOULD be able to process data in near real time (process units).	Section 4 D4.10	This has been achieved by IHES Devices thanks to the LEA component deployed within them. Please refer also to KPIs 4.4 and 4.5. It has been tested verifying that the device firmware of the LEA component is able to detect changes in less than 10ms (KPI 4.4) and with an online learning of about 250ms on average. This are coherent with real-time scenarios for a microcontroller running at 80Mhz.
R.FD.4	Field devices SHOULD use a global clock for time synchronization.	Section 4 D4.10	This has been achieved by using global clock provided by epoch time with millisecond precision that has been implemented in UC3 LEA components. It relies on STM32 MCU clock. It has been implemented by implementing specific MQTT messages exchanged via MQTT low level protocol between the IHES Device and the IHES supervisor at gateway level. As part of initial handshaking also channel latencies are estimated.
R.FD.5	Field devices SHOULD be able to interact with SEMIoTICS IIoT/IoT gateway dedicated components	Section 6	IHES Devices interacts with Pattern Engine and IHES Supervisor MQTT Proxy, LocalDB SEMIoTICS gateway components.

			It has been implemented and validated by bootstrapping those components, as part of the deployment of sub-usecase1 (LEA Components) , and sub-usecase3 (interaction between IHES devices and the pattern engine component via local MQTT infrastructure).
R.FD.6	Field devices MUST interoperate using a standard communication protocol like Rest APIs, COAP, MQTT.	Section 4.3 D4.10	UC3 Testbed uses MQTT as communication protocol at Field Level. This requirement has been implemented, validated and tested as part of the integration and deployment of the UC3 testbed field-level components. MQTT has been integrated into key SEMIoTICS components like the LEA and the pattern related components (i.e. pattern-engine)
R.FD.7	Field devices MUST use standardize interoperable message format (e.g., JSON, etc.).	Section 4.3 D4.10	UC3 Devices uses standard JSON serialized packets to exchange messages with other Field components. This requirement has been achieved by designing a JSON protocol for interacting with IHES devices that uses interoperable JSON+MQTT infrastructure. That protocol has been semantically described to be interoperable with the IEEE WoT Standard in order to expose at field-level the IHES devices as WoT devices.
R.FD.9	Field devices MUST be able to communicate with the IIoT Gateway / other architectural components.	Sections 3 and 4	This requirement is fulfilled and demonstrated by UC3 IoT Gateway components and the UC3 testbed as part of the deployment of sub-usecase1, sub-usecase2 and sub-usecase3. The requirement was tested and validated both at intra-layer level (communication and interactions btw the devices and the pattern engine and IHES supervisor), and also cross-layer in the realization of the avalanche warning scenario and the integration with the Pattern and UC3 GUI visualization app located at backend level.

R.FD.10	Field devices SHOULD minimize data traffic.	Section 4 D4.10	This requirement is fulfilled and demonstrated by Local Analytics component in IHES Devices. This requirement was addressed by UC3 sub-usecase1 thanks to the local embedded analytics for anomalies detection located within the sensing units. This only relevant events (clock synchronization, anomalies, hearth-beat messages) are exchanged between the layer infrastructure (UC3 field vs backend services).
R.FD.11	Field devices SHOULD minimize energy consumption.	Section 4 D4.10	The actual distributed processing deployed on STM32 MCU sensing devices ensure a power consumption in the milliwatts range where centralized backend analytics is on the range of hundreds of Watts. Also reducing data traffic over the network thanks to the local processing ensure further power savings. This requirement was fulfilled by mapping the distributed LEA component directly on the MCU sensing device that has a much lower power consumption vs the other components running on the gateway, network and backend level.
R.FD.14	The field layer must feature SPDI pattern reasoning local embedded intelligence capabilities	D4.8, D4.10	Implemented, validated through sub-use case 3, whereby Field-layer SPDI pattern reasoning is the focus. Showcased through the deployment of lightweight Pattern Engine at the Field Gateway, providing Dependability property monitoring and autonomous, through monitoring and reasoning on the sub-properties of Fault Tolerance and measurement Reliability.
R.FD.15	The field layer must aggregate intra-layer monitored information to enable local intelligence reasoning and adaptation	D4.8, D4.2	Implemented, validated through sub-use case 3, whereby Field-layer Dependability reasoning is showcased. The lightweight Field Pattern Engine deployed at the Field Gateway integrates monitoring capabilities allowing it to aggregate evidence regarding the fault

			tolerance and reliability of the sensing system (which, in turn, enables reasoning on said properties).
R.P.1	The collection of raw data MUST be minimized.	Section 4 D4.10	This has been implemented thanks to the local processing done by LEA module directly mapped on IHES Sensing Devices. It has been verified as part of the development of the LEA firmware and the LocalDB service where thanks to the InfluxDB policies it is possible to locally (at gateway) aggregate collected data in order to minimize the DB storage. Only raw data associated to relevant events (i.e. anomalies) are kept.
R.P.2	The data volume that is collected or requested by an IoT application MUST be minimized (e.g., minimize sampling rate, amount of data, recording duration, different parameters).	Section 4.3 D4.10	This is implemented by UC3 layered intelligence where all data and recordings are kept / stored in IoT Gateway (InfluxDB) and available to the backend services on request. Thanks to the InfluxDB it is possible to define use case specific retaining policies to efficiently manage DB storage (e.g. keep only raw data associated to reported anomalies).
R.P.3	Storage of data MUST be minimized.	Section 4	LocalDB Component on IoT Gateway allows configurable policies to aggregate data thanks to the InfluxDB infrastructure. It was validates as part of system testing by collecting data over time and monitoring the size of the local DB.
R.P.5	As much data as possible MUST be processed at the edge to hide data sources and not reveal user related information to adversaries (e.g., user's location).	Section 4 D4.10	This has been achieved thanks to the local processing done by LEA module directly mapped on IHES Sensing Devices and the deployment at gateway level of LocalDB service that avoids to send raw data to backend level during system lifetime. Relevant raw data are still available at gateway level and available through standard SQL query language of InfluxDB infrastructure.
R.UC3.1	IoT Sensing unit shall be able to embed environmental (e.g., temperature, pressure, humidity, light) and inertial sensors (accelerometer, gyroscope).	D4.10	IHES Sensing device has built-in sensors compliant with this requirement. This requirement has been implemented as part of sub-

			usecase1 scenario during the development of the LEA component on STM32 equipped IHES devices. It has been validated as part of KPI4.2 to 4.4 verification and in the UC3 testbed integrated environment.
R.UC3.2/ R.UC3.15	IIoT Sensing unit shall be able to interface to the IIoT Sensing gateway to coordinate with it. A standard IP based (i.e., TCP transport) 1 to many M2M communication protocol must be adopted to properly handle node communication with components in the gateway.	D4.10	This has been implemented by implementing the UC3 LEA Components at Field / Gateway level in UC3 sub scenarios 1, 2 and 3. It has been achieved as part of sub-usecase1 scenario during the development of the LEA component on STM32 equipped IHES devices and the IHES Supervisor Service. Also it has been tested and validated as part of the interaction with the pattern-engine as part of sub-usecase3.
R.UC3.3	IIoT Sensing unit shall be able to learn a model from observed data in an unsupervised manner. In particular, IoT Sensing unit shall be equipped with a low power (tens/hundreds of mw range) 32 bits MCU to support unsupervised learning and unsupervised statistical processing.	D4.10	This has been implemented by implementing the UC3 LEA Components at Field / Gateway level in UC3 sub scenario 1. It has been implemented and tested as part of sub-usecase1 scenario during the development of the LEA component on STM32 equipped IHES devices and the IHES Supervisor Service.
R.UC3.4	IIoT Sensing unit shall be able to detect relevant changes from the learned model and report them to IIoT Sensing gateway.	D4.10	This has been implemented by implementing the UC3 LEA Components at Field / Gateway level in UC3 sub scenario 1. It has been implemented and tested as part of sub-usecase1 scenario during the development of the LEA component on STM32 equipped IHES devices and the IHES Supervisor Service.
R.UC3.5	IIoT Sensing unit shall be able to adapt to a new model if IIoT sensing gateway requires this.	D4.10	This has been implemented by implementing the UC3 LEA Components at Field / Gateway level in UC3 sub scenario 1. It has been implemented and tested as part of sub-usecase1 scenario during the development of the LEA component on STM32 equipped IHES devices and the IHES

			Supervisor Service. Adaptation have been tested by instructing the IHES devices to train themselves as part of the MQTT protocol validation.
R.UC3.6	IIoT Sensing gateway shall be able to coordinate a set of IIoT sensing units by finding any correlation btw them according to observed data, models	D4.10	This has been implemented by implementing the UC3 LEA Components at Gateway level (IHES Supervisor Component) and in UC3 sub scenario 1. It has been implemented and tested as part of sub-usecase1 scenario during the development of the IHES Supervisor Service. Expected behaviour have been verified by collecting data acquired by sensing units and computing the correlation graph between them, in order to instruct proper local trainings on correlated device units.
R.UC3.7	IIoT Sensing gateway shall be able aggregate relevant events (i.e., changes) coming from whichever of connected IIoT sensing units deciding if they are global or local changes	Section 4 and D4.10	This has been implemented in UC3 sub scenario 1 and visualized thanks to the UC3 GUI at backend level. It have been tested during the development of the MQTT infrastructure and the interaction of the IHES devices with the UC3 GUI based on OpenHAB to verify semantic interoperability. Also local and global changes have been simulated and tested in laboratory conditions by feeding the analytics algorithms with known data patterns and directly on the field as part of the UC3 testbed deployment. Finally it was demonstrated in live demos of the system in the scenario of the avalanche warning detection.
R.UC3.8	IIoT Sensing gateway may have the capability to exchange relevant information (i.e., events) between itself, the cloud, and the sensing units with some connectivity capabilities	Section 5 and 6	This has been implemented and validated in UC3 sub scenarios 2 and 3 thanks to the integration on UC3 testbed of the required components to enable sub-usecase2 and sub-usecase3.
R.UC3.9	IIoT Sensing web GUI may be able to display correlations between connected IIoT Sensing units and the status related to each IIoT sensing unit.	Section 4.3.4	This has been implemented in UC3 openHAB UC3 App and tested as part of the verification of the GUI components.

R.UC3.10	IIoT Sensing web GUI may be able to display logging about relevant events detected by connected IIoT Sensing units reporting info about unit ID, type of data and type of event detected.	Section 4.3.4	This has been implemented in UC3 openHAB UC3 App. It has been verified and tested during deployment of the UC3 App in sub-use case 2 within the avalanche warning demo scenario.
R.UC3.11	IoT Sensing unit shall be able to run Artificial neural networks on the MCU in real time at the sensor data rate of choice.	Section 4 and D4.10	This has been implemented in UC3 sub scenario 1 by providing the LEA component as part of the sensing devices firmware. It has been tested also as part of the KPIs 4.2 to 4.4 verification and with dedicated field trials to test the correct deployment of the UC3 testbed on the avalanche warning scenario.
R.UC3.12	IoT Sensing unit shall be able to run lightweight statistical model analysis algorithms on the MCU not in real time at the sensor data rate of choice.	Section 4 and D4.10	This has been implemented in UC3 sub scenario 1 by providing the LEA component as part of the sensing devices firmware. It has been tested also as part of the KPIs 4.2 t 4.4 verification and with dedicated field trials to test the correct deployment of the UC3 testbed on the avalanche warning scenario.
R.UC3.13	MCU into IoT Sensing unit should be associated with a high-level tool for automatic generation of optimized code to support pre trained neural networks.	Section 4 and D4.10	This has been implemented in UC3 sub scenario 1. The STM32 Cube.AI tool was used to facilitate the development of part of the analytics firmware that runs on the device. The validation facilities provided by Cube.AI tool have been used to validate the functional equivalence of the mapped models.
R.UC3.14	MCU IoT Sensing unit shall be able to run neural network online training at the sensor data rate of choice.	Section 4 and D4.10	This has been implemented in UC3 sub scenario 1 by providing the LEA component as part of the sensing devices firmware. It has been tested also as part of the KPIs 4.2 to 4.4 verification and with dedicated field trials to test the correct deployment of the UC3 testbed on the avalanche warning scenario.
R.UC3.16	IoT Sensing gateway shall be equipped with a low power embedded CPU (100s of mW to W).	Section 4 And D4.10	This has been addressed in UC3 sub scenario 1. It was achieved by using a low power STM32 MCU equipped device with embedded sensors (i.e. accelerometer and environmental sensors).

R.UC3.17	IoT Sensing gateway shall be able to negotiate capabilities, notify, start and shutdown any Sensing unit at any point in time.	Section 4 and D4.10	This has been implemented in UC3 sub scenario 1 in particular as part of the developed IHES Supervisor Service Locate at Gateway and validated as part of its deployment.
R.UC3.18	IoT Sensing gateway shall be capable to run Linux (e.g., Ubuntu OS) and standard graphics and browser libraries.	Section 4	This has been implemented in UC3 testbed that runs on Linux OS.
R.UC3.19	IoT Sensing gateway should be able to support http and standard protocols for cloud interfacing.	Section 5	This has been implemented in UC3 sub scenario 2 global data aggregation and avalanche warning. It has been tested while deploying the storyline associated to that scenario.
R.UC3.20	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 gateway and its associated Sensing units.	Section 3	This approach has been implemented and adopted in UC3 testbed. A local MQTT Broker at gateway and a global MQTT broker at backend have been instantiated in order to ensure components interoperability. It has been tested as part of the UC3 testbed integration as part of sub-usecase2 and sub-usecase 3 demo deployments.
R.UC3.21	A use case specific serialized message protocol is required to coordinate the gateway and its associated units and exchange data / events / anomalies between them. JSON will be the preferred serialization format adopted.	Sections 4, 5 and 6	This has been implemented in UC3 sub scenarios 1, 2 and 3.
R.UC3.22	Each connected IHES sensing unit should send to the gateway a keep alive signal on a specified period (e.g., few seconds) to notify the gateway it is correctly working. The sensing gateway should detect by this mean any non-working sensing unit and reconfigure the system accordingly.	Section 6	This has been implemented and tested in UC3 sub scenario 3 – dependability monitoring and demonstrated during live demos.
R.UC3.23	Sensing units and sensing gateway should share a common clock (i.e., global reference time), precise up to milliseconds, to properly classify events and data acquired during the processing.	Section 3	This approach has been implemented and adopted in UC3 testbed and has been tested as part of the UC3 testbed deployment activities.

	This global reference time will be negotiated when a sensing unit node will join a given gateway. Internally the system will work scheduling activities according to this global reference time.		
R.UC3.24	Sensing units may be equipped with dedicated FW to detect relevant sensors malfunctioning and report that to the gateway	Section 6	This has been implemented in UC3 sub scenario 3 through instantiation of specific patterns instead of mapping on FW device.

7.2 UC3 KPIs Summary

In reported Table 2 have been summarized all the related KPIs involving the UC3 scenario inclusive of the sub-usecase1 – global/local changes detector (KPIs 4.4.X and 4.5), sub-usecase2 (KPI2.3, KPIs 3.1.X), sub-usecase3 (KPI 1.1, 1.2, 4.2) and overall UC3 testbed (KPI 2.1, 2.3, 6.1, 6.3). In following sub sections, a clear overview of each KPI has been reported, where needed, with the relevant measures and results obtained, as well as the methodology used to assess them.

TABLE 2. USE CASE 3 KPIS SUMMARY

KPI-ID	KPI Description	Reference	Fulfilment
KPI-1.1	Number of SPDI Patterns	D4.8	In UC3 the focus was on showcasing the Dependability monitoring and verification, introducing a new complex rule that considered Fault Tolerance and measurement Reliability. A total of 6 pattern rules are used (Dependability, Fault Tolerance, Reliability composition and decomposition rules), as detailed in subsection 6.3.2. These are adapted and enhanced from the pattern specification approach and full list of SPDI patterns specified within D4.8.
KPI-1.2	Pattern Language	D4.8	Pattern-based Dependability monitoring and reasoning capabilities showcased within UC3 (sub UC3) leverage the SEMIoTICS pattern language and associated IoT orchestration specification approach defined within D4.8.
KPI-2.1	Semantic descriptions for objects	D4.1	Semantic description of IHES devices sensors for interoperability has been provided in D4.11 Section 4.3.3
KPI-2.3	Semantic interoperability with IoT platforms	Section 5	UC3 leverages on standard MQTT infrastructure to handle real-time data

		D4.11 Subsection 4.3.3	and events communication between different used components and moreover as a proxy to handle communication between the field layer and the services located in UC3 backend. Furthermore, the IHES devices have been semantically described by providing the TD also as proper WoT devices to allow improved interoperability with potentially any 3rd party services that relies on this W3C open standard. In this particular scenario the WoT servient must be hosted in IoT gateway since for legacy SW middleware constraints it could not be deployed on IHES devices directly.
KPI-3.1.1	Generating monitoring strategies in the 3 targeted IoT platforms	Section 5	UC3 leverages openHAB IoT platform for cloud-based visualization of time series data, to allow monitoring of environmental and acceleration parameters. UC2 and UC1 leverage on Chloe IoT and MindSphere respectively, amounting to 3 IoT platforms in total.
KPI-3.1.3	Performing predictive monitoring with accuracy > 80%	Section 5.4.1	Highly accurate temperature predictor at the backend cloud, within the context of avalanche early warning system.
KPI-4.1	Delivery of lightweight ML algorithms	Section 7.2.3 D4.10	A total of 5 different lightweight algorithms has been deployed on IHES sensing units as part of the LEA component mapped on MCU. These algorithms have been detailed and characterized in D4.10
KPI-4.2	Delivery of adaptation mechanisms that support proactive and reactive, as well as horizontal and vertical adaptation actions, related to network, smart objects and IoT platforms with an adaptation time of 15ms	D4.8	Pattern rules used within sub use case 3 (see subsection 6.3.2), along with the associated pattern components able to reason on said pattern rules and facts are key enablers for triggering adaptations to changes in the fault tolerance, reliability and dependability posture of the smart objects implementing the sensing system. The actual adaptation time will depend on the adaptation strategy adopted and can be from near-real time (e.g., for adaptation to ignore inputs from malfunctioning sensors) to

			longer adaptation times (e.g., when a manual sensor replacement is required). Testing indicates that the minimal reasoning delays allow for automated adaptation within 15ms.
KPI-4.4	Detection time of less than 10 ms	Section 7.2.4 D4.10	This 10ms figure is to ensure the real-time Local Embedded Analytics processing on Field Device level on UC3. This KPI has been verified as part of the validation of the LEA component (CDT test) detailed in D4.10
KPI-4.4.1	Delivery of repeatable Change Detector	Section 7.2.6	This is an additional KPI identified during T5.6 integration introduced to ensure that similar input data on different devices or at different time produce an equivalent estimation on the thresholds used by CDT tests. Thus, the detector, trained in similar condition should provide similar thresholds. This is a key point to enable system capabilities reported in UC3 sub use case 1.
KPI-4.4.2	Comparison false positive rate of auto regressive (AR) models vs neural network models for a defined set of changes	Section 7.2.7	This is an additional KPI identified during T5.6 integration needed to better characterize the analytics at field layer in UC3. Improvement of at least 20% in false positive rate – ESN vs AR[3]
KPI-4.5	Complexity of back propagation on feed forward autoencoder neural network[4] vs ST-I proprietary adaptation mode[5] based on NN	Section 7.2.8	Derived from generic Improvement of at least 20% in minimum adaptation time – ESN vs Autoencoder
KPI-6.1	Reduce manual interventions required for bootstrapping of smart object in each use case domain by at least 80%.	Appendix 9, see also Sections 3 and 5.	The bootstrapping service for the smart objects involves the availability of other functional blocks, such as the MQTT broker/publisher/subscriber at the IoT GW/backend. An implementation of these functional blocks has been made within as as part of the UC3 testbed deployment in the form of VNFs automates its availability for the bootstrapping

[3] Both ESN (Echo State Network) and AR are models developed by STMicroelectronics specifically for UC3

[4] Autoencoder reference model has been developed by ST specifically for UC3 as initial reference for self-learning design

[5] Use case specific adaptation modes have been developed by ST-I as part of UC3 T4.3 activities

			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. These implemented functionalities has been tested during the deployment of sub-usecase2 and sub-usecase3 scenarios.
KPI-6.3	Delivery of 3 prototypes of IIoT/IoT applications	Section 4.3.4	openHAB platform has been adopted in UC3 as the IoT Platform to implements the system visualization UC3 smart sensing application
KPI-7.1	Provision the SEMIoTICS building blocks	Section 3, 4, 5 and 6	UC3 has provided 3 specific new components to the SEMIoTICS framework (thanks to sub-usecase1 scenario), i.e they are the IHES local embedded analytics components hosted directly on IHES sensing devices units and the IHES Supervisor and LocalDB hosted at gateway level, 2 components have been improved to integrate for interoperability the specific MQTT flow of UC3 (in sub-usecase 3 with adaption of the field pattern engine and the GUI). Finally, several other components (UC3 App, Backend orchestrator, Pattern orchestrator, NFV Orchestrator, VNF manager, etc.) have been used to implement UC3 testbed functionalities with an overall TRL 4 maturity level being the UC3 testbed validated in laboratory environment. The TRL level of each component used in UC3 are reported there for completeness: UC3 App - TRL4 Backend orchestrator – TRL6 Pattern Orchestrator – TRL4 GUI – TRL6 NFV Orchestrator – TRL6 VNF Manager – TRL6

			Virtualized Infrastructure Manager – TRL6 Pattern Engine – TRL4 Local embedded intelligence – TRL4 Supervisor and Local DB – TRL4

7.2.1 KPI 1.1 - NUMBER OF SPDI PATTERNS

Pattern-driven SPDI management is at the core of the SEMIoTICS security-by-design approach. This KPI has been satisfied by the delivery of the final set of SPDI patterns in Section 4 of deliverable D4.8. As aggregated in subsection 4.6 of said deliverable, 49 patterns are delivered in total, covering all key SPDI properties and different data states and cases of platform connectivity.

Furthermore, in UC3 the focus was on showcasing the Dependability monitoring and verification, introducing a new complex rule that considered Fault Tolerance and measurement Reliability. To implement and showcase the associated Dependability monitoring and reasoning, a total of 6 pattern rules are defined and used (Dependability, Fault Tolerance, Reliability composition and decomposition rules), as detailed in subsection 6.3.2.

7.2.2 KPI-1.2 – PATTERN LANGUAGE

This KPI is satisfied through the delivery of the final version of the pattern language as presented in detail in Section 3 of D4.8, developed to accommodate all the needs of the SEMIoTICS pattern definition and reasoning.

The Dependability reasoning capabilities demonstrated in UC3 leverage the SEMIoTICS pattern language and the associated IoT orchestration specification approach.

7.2.3 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[6]) test applied to residual signal.
- A change-point method to (CPM[7]) validate the detected change
- A fine-tuning algorithm tailored to the online training stage, to compute automatic thresholds for triggering the CDT and CPM stages

7.2.4 KPI-4.2 - ADAPTATION MECHANISMS WITH AN ADAPTATION TIME OF 15MS

PATTERN-DRIVEN ADAPTATIONS

Pattern rules used within sub use case 3 (see subsection 6.3.2), along with the associated pattern components able to reason on said pattern rules and facts, are key enablers for triggering adaptations to changes in the fault tolerance, reliability and dependability posture of the smart objects implementing the sensing system.

Considering the above, testing indicates that the minimal reasoning delays of the pattern mechanisms allow for automated adaptation within 15ms. In more detail, in the testbed setup detail in section 6, testing over a

[6] CDT models are developed by ST specifically for UC3 scenarios
[7] CPM models are developed by ST specifically for UC3 scenarios

number of 100 reasoning events (including sensor failures, sensor malfunction and sensor recovery events) yield a reasoning time of ~12ms on average, with a typical delay of ~10ms for most cases, and some outlier events at ~15ms (which affected the average time). These outlier events are attributed to the timing of the interrupts that the MQTT client embedded into the field-layer Pattern Engine introduces when it receives data and could, potentially, be further tuned.

Nevertheless, it is evident from said testing that automated, pattern-driven adaptations within the limit of 15ms are possible in most instances. Nevertheless, it should be noted that the actual adaptation time will depend on the adaptation strategy adopted and can be from near-real time, within the 15ms limit (e.g., for adaptation to ignore inputs from malfunctioning sensors), to longer adaptation times for more complex scenarios and e.g., when a manual sensor replacement is required – in which case, adaptation times of 15ms are anyway not realistically possible.

7.2.5 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

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.

7.2.6 KPI 4.4.1 – 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 have 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 3. 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 can estimate equivalent thresholds from same model under same conditions.

7.2.7 KPI 4.4.2 – 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 58):

- T_0: time instant the experiment starts
- T*: time instant the change happens
- T ref: time instant the model detects the change
- T̂: time instant the experiment ends

TEST CONDITIONS

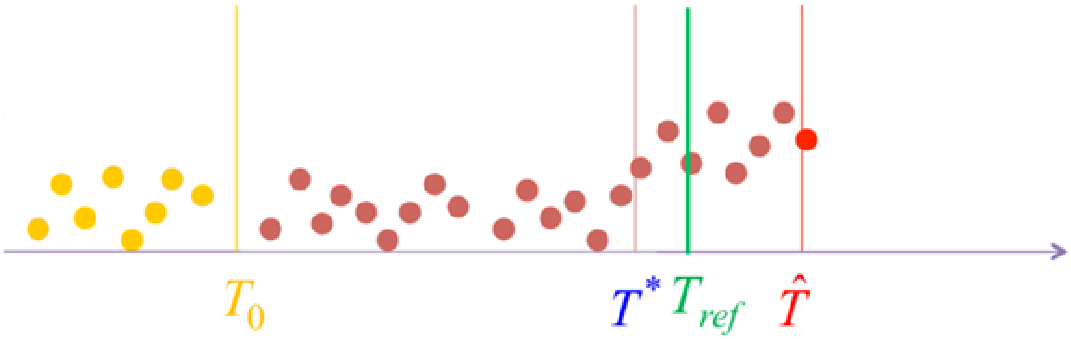


FIGURE 58. KPI 4.5 TEST CONDITIONS

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 < T*ref* < T*
- Correct Prediction when: T* < T*ref* < T̂ (with a detection delay (dd) equal to t – T*)
- False negative when: T*ref* > T̂

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) have been benchmarked on the X-NUCLEO F401RE[8] 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 4) and *p* = 5 (Table 5)
- M = 200

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

- T̂ = 350

TABLE 4. 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 5. 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

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.

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

This is a specific adaption from generic KPI as originally stated in D5.1. This KPI is defined as “The adaptation response time should bring at least a 20% improvement over the baseline of each domain”. In the context of UC3 the adaptation has been implemented as part of the unsupervised learning implemented at field device level as part of the LEA component. The complexity has been evaluating by comparing the adopted UC3 model ESN training vs a standard autoencoder backpropagation process.

CONSIDERED MODELS

In this experiment the following models has been considered for signal prediction tasks:

- Neural model 1:
  - type: Autoencoder (Reference SOTA)
  - 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 (UC3 adopted):
  - 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 STM32 JAM-CLOUD 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 6. KPI 4.5 RESULTS

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

CONCLUSIONS

The ESN models adopted in UC3 improve the minimum adaptation time more than 20% with respect a standard 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.

7.2.9 KPI-6.3 DELIVERY OF 3 PROTOTYPES OF IIOT/IOT APPLICATIONS

OpenHAB platform has been adopted in UC3 as the IoT Platform to implements the system visualization UC3 smart sensing application as reported in Table 2 – see KPI 6-3.

7.2.10 KPI-7.1 PROVISION THE SEMIOTICS BUILDING BLOCKS

All components deployed in UC3 reaches a TRL4 level maturity since the UC3 testbed have been deployed in a lab environment. No matter, some of the used UC3 components are at a higher stated TRL level since they are based on technology that has been tested also in real conditions. The complete overview of each single component used in UC3 are reported in Table 2 – see KPI 7-1.

8 CONCLUSION

8.1 Lessons Learned

8.1.1 OBSERVED OBSTACLES

During the consolidation of the UC3 we realize how critical in such scenario is the availability of methodologies and tools for deploying local embedded intelligence to the device level, where the devices are extremely low power MCU units. Nowadays there are not yet any de-facto framework that covers that area. The complex cloud infrastructures solutions providers such as Amazon with their analytics tools, Microsoft Azure, or IBM do not currently have any solution for integration seamlessly the local embedded intelligence with existing backend services. In this layered and distributed intelligence scenario the communication and networking of the results of the analytics is of paramount importance. The right communication patterns and an open framework where the outcomes of the local analytics (together with the ones already widely available at cloud infrastructure) could be easily integrated to be exploited coherently according to the desired scenario are somehow a key technical challenge not yet fully addressed in the IoT community.

In this respect we think that in SEMIoTICS we were able to provide a solution to the problem by providing a general methodology for the deployment of AI analytics on devices (see D4.10 section about DL tools overview), a layered, scalable, and interoperable communication infrastructure based on MQTT publish/subscribe fully integrated with SEMIoTICS SPDI pattern design and finally a vertical integration of the system with its monitorability thanks to the designed dependability patterns and the VNF NFV virtualization infrastructure of the UC3 testbed that simplify all needed components integration and reuse.

8.1.2 POTENTIAL IMPROVEMENTS

As described in this UC3 scenario, the developed UC3 testbed is one of the possible verticalization of the generic IoT edge computing infrastructure provided in SEMIoTICS, addressing a very specific scenario, complementary to e.g. UC1 and UC2. It was developed to propose an understandable scenario offering a solution to the challenges mentioned in section 8.1.1. We know that thanks to the SEMIoTICS modular infrastructure it is possible to quickly instantiate other use cases in not so-far considered application fields area such as smart building management and IIoT and industries 4.0 use cases. One possible improvement is, for example, to add semi-autonomous actuation upon a relevant event detected by the layered intelligence (e.g., to stop elevators in a building where an alert has been triggered). Another potential improvement is a tighter integration with 3rd party services like weather forecasting services, or earthquakes monitoring, or also the possibility to cluster together more local LAN IHES services to not monitor a single building but also a district area, or even possibly a city with a fine-grained understanding of the events generated by all of these distributed subsystems adding further trend analytics on top of what has been done in the sub use case 2. The potential applications are endless.

8.2 Final recommendations for UC3 Generic IoT Deployment

Although D5.11 implements and demonstrates three (3) real-world IoT incremental applications, it does not aim to provide a generic vertical deployment for “any” distributed analytics and monitoring for the general usage of framework. In fact, the goal of UC3 remains to provide, within SEMIoTICS SPDI pattern driven approach and final reference architecture, a subset of dedicated designed components that enabling distributed self-learning capabilities for anomalies detection to the framework. Anyhow, the specific application field, domain scenario and application field impose peculiar trade-off, and somehow different/additional requirements on the final system architecture and deployment: in this respect the proposed avalanche warning system could not be considered anyway generic, usable “as it is” into another use case.

Still, it is definitely a good architecture reference for any similar deployment.

System IoT integrators stakeholders have to fine-tune their desired system derived from UC3 for their specific goals.

In order to implement this fine-tuning a system integrator should take care of:

1. Deploy at field level the desired number of devices so that a certain redundancy in the system is achieved (depending on the use case) in order to exploit consistent dependability patterns for their scenario.
2. Fine-tune the IHES Supervisor component configuration in order to define the level of correlation between units and the local field level system topology (number of inertial unit's vs number of environmental units deployed).
3. Define specific use case coherent data aggregation policies on Influx LocalDB through fine-tuning reference configuration setup.
4. Adapt / define /instantiate new patterns to the field pattern engine / pattern orchestrator according to the specific scenario following implementation examples provided in sub use case 3 section.
5. Adapt the UC3 application according to the specific use case need (no need to adapt pattern GUI once new patterns will be correctly deployed).
6. Test the complete integrated system on real conditions and eventually go again to step 2 for final tuning / verification.
7. Extend the system with new data aggregation policies or new cloud services according to desired targeted scenario.

8.3 Concluding Remarks

This deliverable D5.11 presented the final version of the UC3 testbed demonstrator alongside the complete presentation of the executive status activities done in task 5.6 “Demonstration and validation of IHES- Generic IoT (Final)”. This document is intended in this respect as a self-comprehensive executive summary and report all the relevant architectural / implementation details related to the UC3 integration activities done in order to finalize the integration of the whole UC3 testbed demonstrator within SEMIoTICS architecture, as originally designed and declared in D2.5 final architecture design.

For convenience, we had identified three sub use case scenarios to drive efficiently the integration of the system during cycle 2 activities that have been incrementally integrated in task 5.6 and that, all together, demonstrate the whole set of capabilities of the UC3 scenario.

The three sub use cases have been declared by providing for each one of them the links with relevant design documentation already discussed in former deliverables, and their relationship with the SEMIoTICS components and infrastructure, both static and dynamic, has been provided as well.

This document concludes the cycle 1) and cycle 2) integration activities done in task 5.6. Cycle 2) activities were defined and consolidated by leveraging on the initial demonstration and validation of the UC3 scenario done during cycle1) in relation to the correct integration of sub use case 1 into the UC3 testbed. The complete functional characterization and testing of the sub use case 1) scenario withing the final UC3 testbed done in cycle 1) allowed to complete without any delay nor issues the complete vertical integration of the demos thanks to the sub use case 2 and 3 deployment during cycle 2).

9 APPENDIX TESTING OF THE END-TO-END MQTT COMMUNICATION IN NFV

In this section, we explain the testing and implementation of an end-to-end MQTT communication system in the framework of NFV, which is depicted in FIGURE 59. This was used to test the MQTT communication at the backend cloud in section 5. That is, we create a VNF that contains the functionalities required for an end-to-end MQTT communication: the MQTT publisher, MQTT broker and MQTT subscriber, which we call VNF3. Then, this VNF3 is deployed within the NFV testbed. Moreover, the VNF3 is managed by the NFV MANO, which is implemented by the OSM[9] 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.

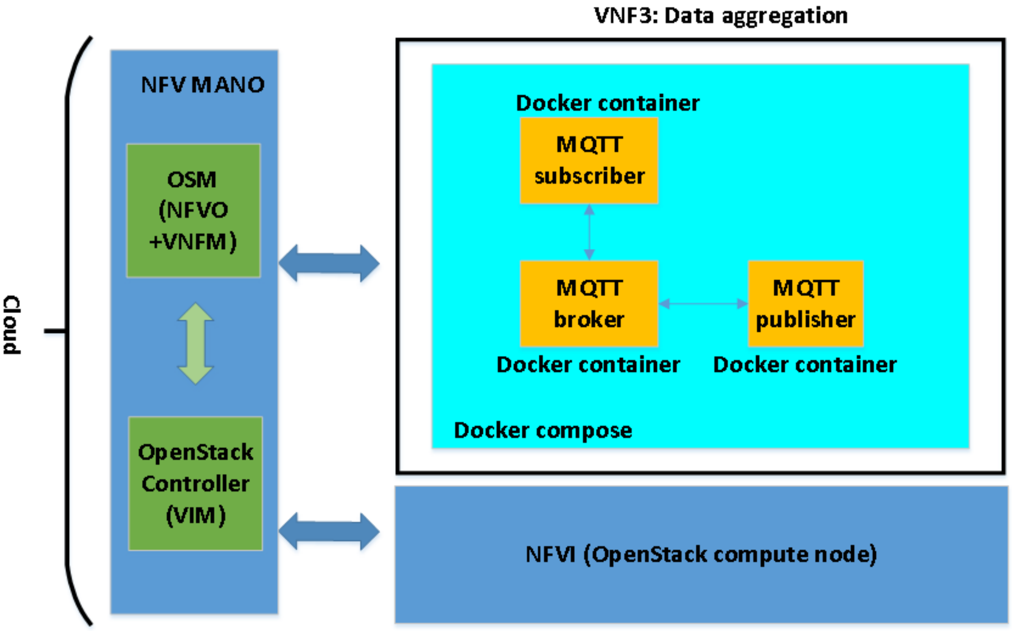


FIGURE 59. ILLUSTRATION OF THE IMPLEMENTATION OF THE VNF3 WITHIN THE NFV TESTBED

Note that the user has only 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 VNF on top of the NFVI in the form of a VM. Therefore, our task to set up the system of FIGURE 59.is focused to define a set of configuration files required by the OSM to manage the VNF lifecycle and to interact with the OpenStack. Next, we list these configuration files, and we explain its role:

- **Network Service descriptor** (NSd). In OSM a VNF is always part of a network service (NS). 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. We will see more details below.

- **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.

[9] https://osm.etsi.org/

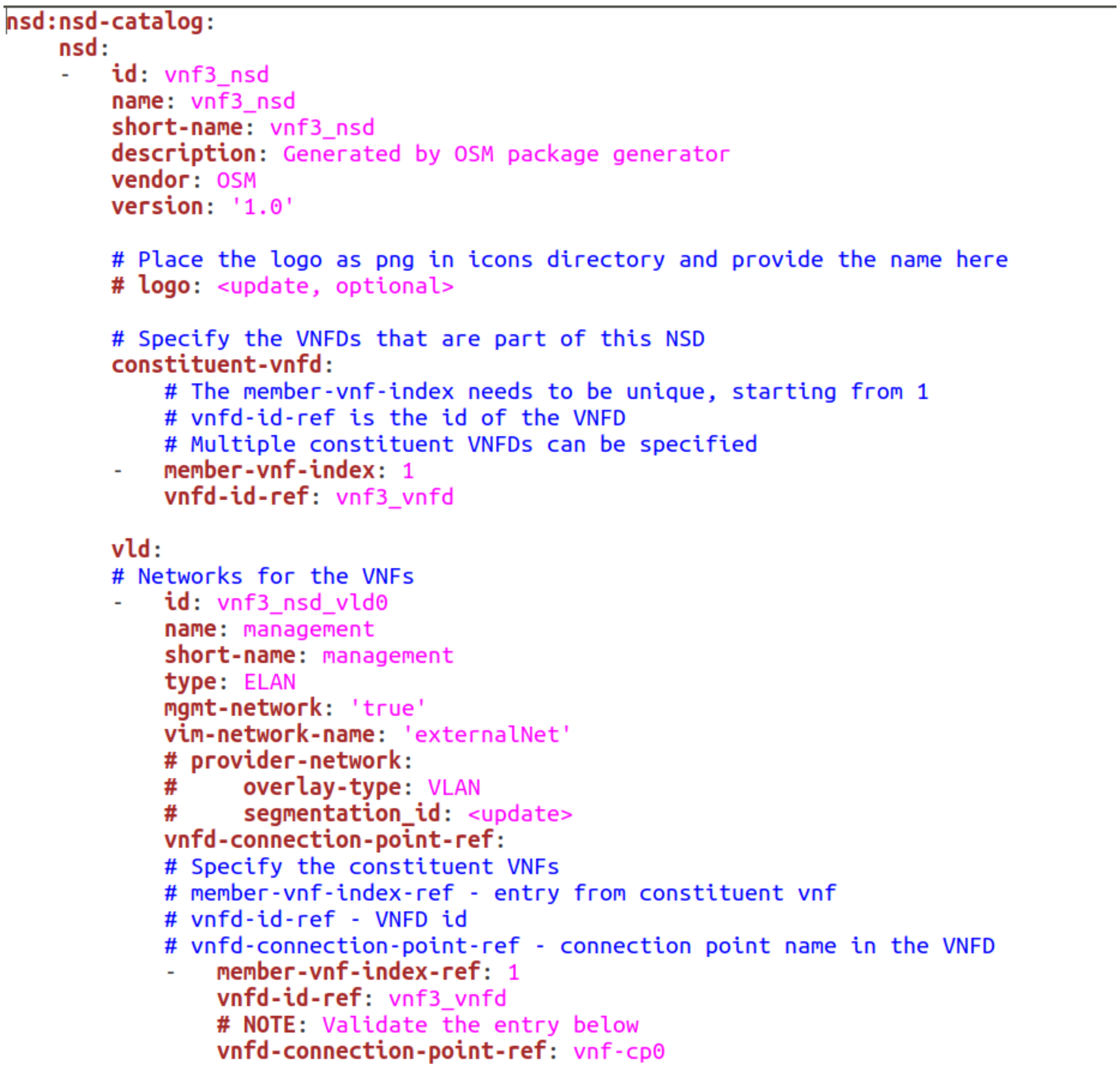
- **Cloud init file**. This is a configuration file that will be used by the VM that will host the VNF. It specifies the initial behavior that the VM needs to provide. For instance, installation of software packages, execution of software applications, among other features.

Next, we describe how we defined the NSd, VNFd and cloud init file to achieve the implementation of our system depicted in **FIGURE 59**. The explanations for the NSd and VNFd rely on the information models provided by OSM, whereas the explanations for the cloud init file rely on the cloud init documentation[10].

- *NSd specification*

Next, we display a snapshot of the yaml file that we edited to specify the NSd. The relevant tags are described in the sequel. First, note that the tag “id” determines the unique identifier for the Network Service. The tag “constituent-vnfd” indicates which VNFs are part of the NS. In our case we have just the VNF3, whose identification is “vnf3_vnfd” and is specified through the tag “vnfd-id-ref”. Note that, in the vnfd the “id” tag must correspond with that value. Then, we have the tag “vld”, which is a description of the virtual links used by the NS for networking connections. In our case, note that the tag “type” is set to “ELAN”. This indicates that the virtual link is a service to connect VNFs. The tag “mgmt.-network” set to “true” means that this is a VIM management network. The tag “vim-network-name” describes the name of the network in the VIM account, in our case “externalNet” is the name that was given such network in the OpenStack framework. Finally, the tag “vnfd-connection-point-ref” describes the connection points for the virtual links towards a vnf. We can see that this is a connection towards our VNF3 as the tag “vnd-id-ref” is set to “vnf3_vnfd”. It is important to note that the connection point for the VNF is set through the tag “vnf-connection-point-ref”, whose value must correspond with the connection point set in the vnfd, as you can check below in the vnfd snapshot.

[10] “Cloud init documentation”. [Online]. Available: https://cloudinit.readthedocs.io/_/downloads/en/latest/pdf/ [Accessed December 2020].

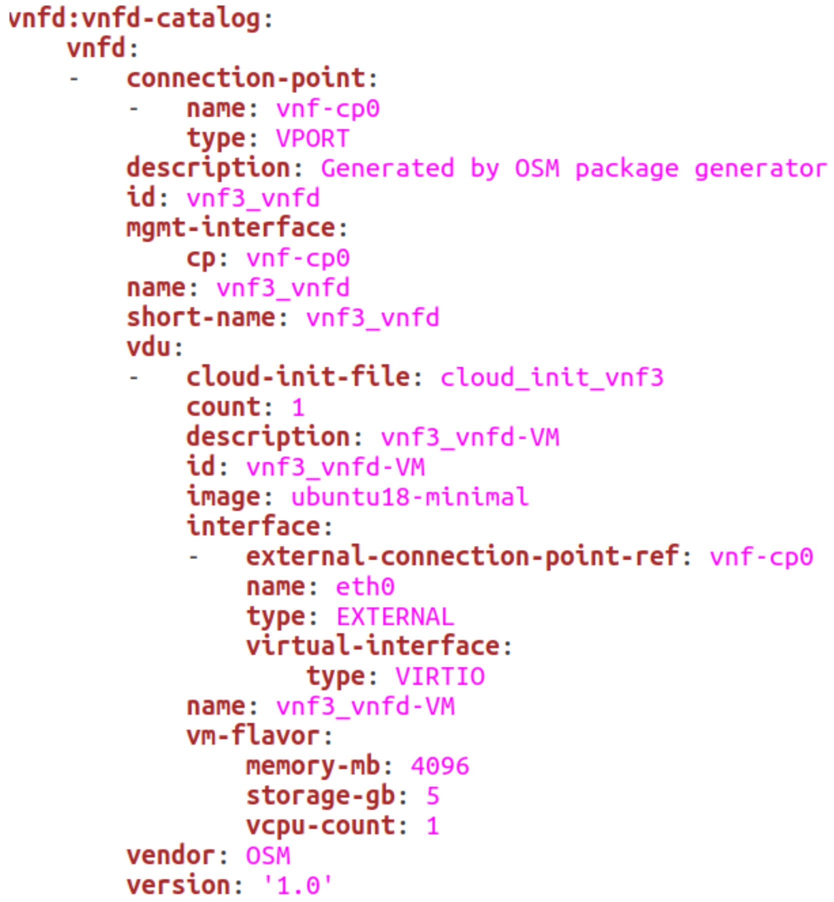


```
nsd:nsd-catalog:
    nsd:
    -   id: vnf3_nsd
        name: vnf3_nsd
        short-name: vnf3_nsd
        description: Generated by OSM package generator
        vendor: OSM
        version: '1.0'

        # Place the logo as png in icons directory and provide the name here
        # logo: <update, optional>

        # Specify the VNFDs that are part of this NSD
        constituent-vnfd:
            # The member-vnf-index needs to be unique, starting from 1
            # vnfd-id-ref is the id of the VNFD
            # Multiple constituent VNFDs can be specified
        -   member-vnf-index: 1
            vnfd-id-ref: vnf3_vnfd

        vld:
        # Networks for the VNFs
        -   id: vnf3_nsd_vld0
            name: management
            short-name: management
            type: ELAN
            mgmt-network: 'true'
            vim-network-name: 'externalNet'
            # provider-network:
            #     overlay-type: VLAN
            #     segmentation_id: <update>
            vnfd-connection-point-ref:
            # Specify the constituent VNFs
            # member-vnf-index-ref - entry from constituent vnf
            # vnfd-id-ref - VNFD id
            # vnfd-connection-point-ref - connection point name in the VNFD
            -   member-vnf-index-ref: 1
                vnfd-id-ref: vnf3_vnfd
                # NOTE: Validate the entry below
                vnfd-connection-point-ref: vnf-cp0
```

FIGURE 60. NSD TO CONFIGURE AND TO DEPLOY THE NS RELATED TO VNF3 IN THE NFV TESTBED

- *VNFd specification*

Next, we present the snapshot of the yaml file that we have used to specify the VNFd. We discuss in the sequel the important parts of this file. 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 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.

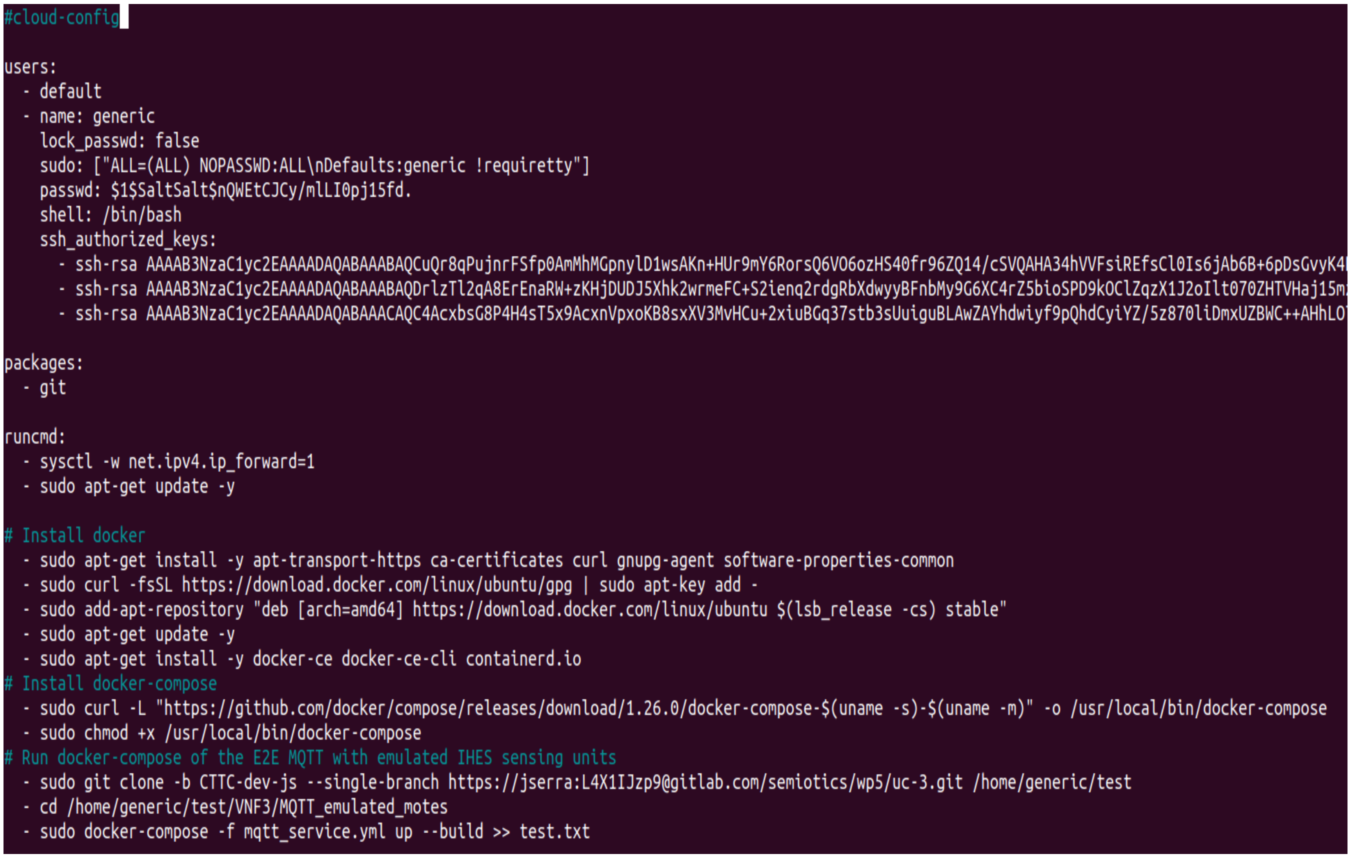


```
vnfd:vnfd-catalog:
    vnfd:
    -   connection-point:
        -   name: vnf-cp0
            type: VPORT
        description: Generated by OSM package generator
        id: vnf3_vnfd
        mgmt-interface:
            cp: vnf-cp0
        name: vnf3_vnfd
        short-name: vnf3_vnfd
        vdu:
        -   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 61. VNFD TO CONFIGURE AND TO DEPLOY THE VNF RELATED TO VNF3 IN THE NFV TESTBED

- *Cloud init file specification*

This file is the one that specifies the initial configuration that we aim for the VM that will host the VNF3. Note that its name is specified in the vnfd, as commented above. Next, we provide the snapshot of this cloud init file and describe its functionalities. First, note that we have a field called “users”. This allows to add users to the system. Note that we have added a user called “generic”. Within this user, there is an important field to be added, the “ssh_authorized_keys”. This is important, because here we add the public ssh keys of the users that will access the VM that hosts the VNF3. Afterwards, there is a field called “packages”, which indicates which software packages we would like to install. In our case, we indicate git to be able to download software from the gitlab repository. Then, we have a field called “runcmd”, which runs commands during the first boot of the VM. First, note that we update the database of available software packages that can be installed, through the “sudo apt-get update -y” command. Note that it is important to put the “-y” option otherwise, the shell will get stacked waiting that we say “yes” to make the update. After that, we put a set of commands to install docker. These commands are based on the guidelines in the official docker webpage[11]. Herein docker is needed to implement the functionality of the VNF3. Namely, each block of the E2E MQTT chain is a docker container. Then, we install docker compose, as it is needed to automate the deployment, configuration, and execution of the docker containers that implement the E2E MQTT chain. Afterwards, we clone the gitlab repository that contains all the software of the E2E MQTT chain. Finally, we leverage docker compose to build and run all the dockers and their networking interactions that implement the E2E MQTT chain. Note that we store the output in a text file. This allows us to test whether the received MQTT topics correspond to the ones generated by the emulated IHES sensing units at the MQTT publisher side.



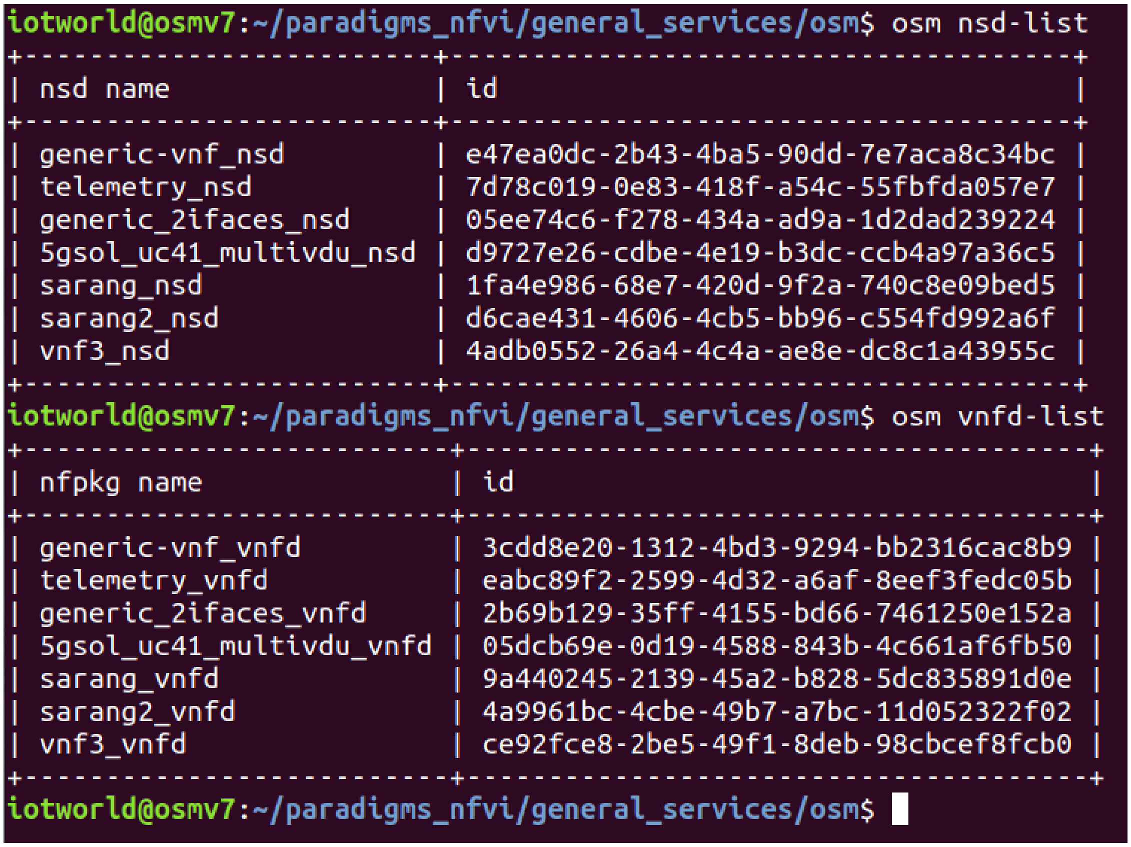
```
#cloud-config

users:
  - default
  - name: generic
    lock_passwd: false
    sudo: ["ALL=(ALL) NOPASSWD:ALL\nDefaults:generic !requiretty"]
    passwd: $1$SaltSalt$nQWEtCJCy/mlLI0pj15fd.
    shell: /bin/bash
    ssh_authorized_keys:
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCuQr8qPujnrFSfp0AmMhMGpnylD1wsAKn+HUr9mY6RorsQ6VO6ozHS40fr96ZQ14/cSVQAHA34hVVFsiREfsCl0Is6jAb6B+6pDsGvyK4
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDrlzTl2qA8ErEnaRW+zKHjDUDJ5Xhk2wrmeFC+S2ienq2rdgRbXdwyyBFnbMy9G6XC4rZ5bioSPD9kOClZqzX1J2oIlt070ZHTVHaj15m
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQC4AcxbsG8P4H4sT5x9AcxnVpxoKB8sxXV3MvHCu+2xiuBGq37stb3sUuiguBLAwZAYhdwiyf9pQhdCyiYZ/5z870liDmxUZBWC++AHhLO

packages:
  - git

runcmd:
  - sysctl -w net.ipv4.ip_forward=1
  - sudo apt-get update -y

# Install docker
  - sudo apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
  - sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
  - sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
  - sudo apt-get update -y
  - sudo apt-get install -y docker-ce docker-ce-cli containerd.io
# Install docker-compose
  - sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  - sudo chmod +x /usr/local/bin/docker-compose
# Run docker-compose of the E2E MQTT with emulated IHES sensing units
  - sudo git clone -b CTTC-dev-js --single-branch https://jserra:L4X1IJzp9@gitlab.com/semiotics/wp5/uc-3.git /home/generic/test
  - cd /home/generic/test/VNF3/MQTT_emulated_motes
  - sudo docker-compose -f mqtt_service.yml up --build >> test.txt
```

FIGURE 62. CLOUD INIT FILE THAT DEFINES THE CONFIGURATION AND EXECUTIONS OF THE VM THAT WILL HOST THE VNF3

Next, in Figure 63 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 onboarded 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.

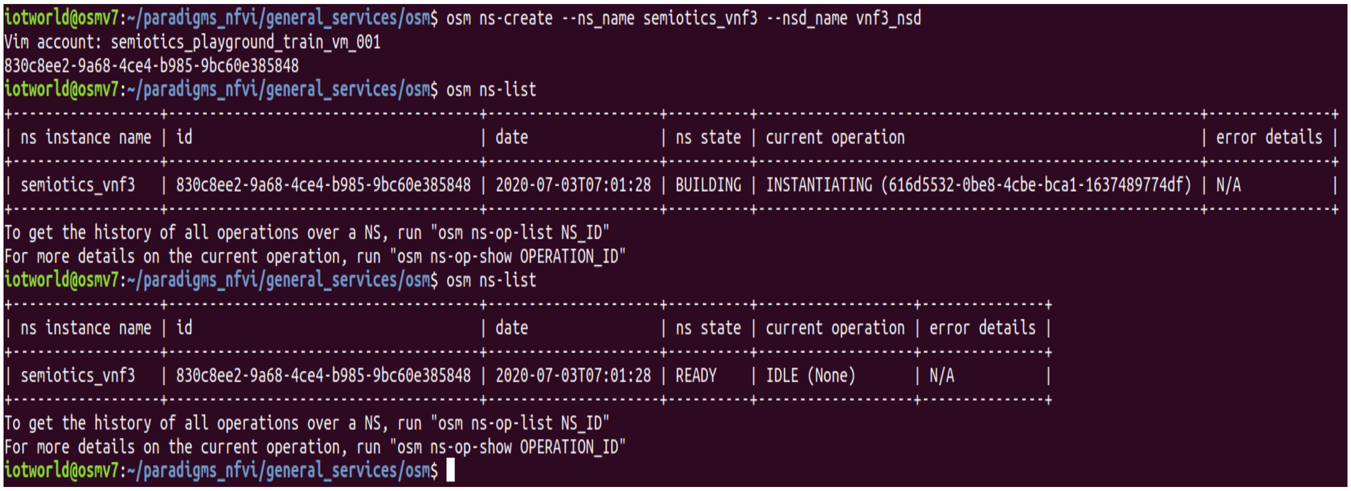
[11] “Docker”. [Online]. Available: https://www.docker.com/. [Accessed July 2020].

Then, in Figure 64, we trigger the instantiation of the NS that we have onboarded 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.



```
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$ osm nsd-list
+------------------------+--------------------------------------+
| nsd name               | id                                   |
+------------------------+--------------------------------------+
| generic-vnf_nsd        | e47ea0dc-2b43-4ba5-90dd-7e7aca8c34bc |
| telemetry_nsd          | 7d78c019-0e83-418f-a54c-55fbfda057e7 |
| generic_2ifaces_nsd    | 05ee74c6-f278-434a-ad9a-1d2dad239224 |
| 5gsol_uc41_multivdu_nsd | d9727e26-cdbe-4e19-b3dc-ccb4a97a36c5 |
| sarang_nsd             | 1fa4e986-68e7-420d-9f2a-740c8e09bed5 |
| sarang2_nsd            | d6cae431-4606-4cb5-bb96-c554fd992a6f |
| vnf3_nsd               | 4adb0552-26a4-4c4a-ae8e-dc8c1a43955c |
+------------------------+--------------------------------------+
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$ osm vnfd-list
+-------------------------+--------------------------------------+
| nfpkg name              | id                                   |
+-------------------------+--------------------------------------+
| generic-vnf_vnfd        | 3cdd8e20-1312-4bd3-9294-bb2316cac8b9 |
| telemetry_vnfd          | eabc89f2-2599-4d32-a6af-8eef3fedc05b |
| generic_2ifaces_vnfd    | 2b69b129-35ff-4155-bd66-7461250e152a |
| 5gsol_uc41_multivdu_vnfd | 05dcb69e-0d19-4588-843b-4c661af6fb50 |
| sarang_vnfd             | 9a440245-2139-45a2-b828-5dc835891d0e |
| sarang2_vnfd            | 4a9961bc-4cbe-49b7-a7bc-11d052322f02 |
| vnf3_vnfd               | ce92fce8-2be5-49f1-8deb-98cbcef8fcb0 |
+-------------------------+--------------------------------------+
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$
```

FIGURE 63. NSD AND VNFD PACKAGES ARE ONBOARDED PROPERLY TO OSM



```
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$ osm ns-create --ns_name semiotics_vnf3 --nsd_name vnf3_nsd
Vim account: semiotics_playground_train_vm_001
830c8ee2-9a68-4ce4-b985-9bc60e385848
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$ osm ns-list
+------------------+--------------------------------------+---------------------+----------+------------------------------------------------------+---------------+
| ns instance name | id                                   | date                | ns state | current operation                                    | error details |
+------------------+--------------------------------------+---------------------+----------+------------------------------------------------------+---------------+
| semiotics_vnf3   | 830c8ee2-9a68-4ce4-b985-9bc60e385848 | 2020-07-03T07:01:28 | BUILDING | INSTANTIATING (616d5532-0be8-4cbe-bca1-1637489774df) | N/A           |
+------------------+--------------------------------------+---------------------+----------+------------------------------------------------------+---------------+
To get the history of all operations over a NS, run "osm ns-op-list NS_ID"
For more details on the current operation, run "osm ns-op-show OPERATION_ID"
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$ osm ns-list
+------------------+--------------------------------------+---------------------+----------+-------------------+---------------+
| ns instance name | id                                   | date                | ns state | current operation | error details |
+------------------+--------------------------------------+---------------------+----------+-------------------+---------------+
| semiotics_vnf3   | 830c8ee2-9a68-4ce4-b985-9bc60e385848 | 2020-07-03T07:01:28 | READY    | IDLE (None)       | N/A           |
+------------------+--------------------------------------+---------------------+----------+-------------------+---------------+
To get the history of all operations over a NS, run "osm ns-op-list NS_ID"
For more details on the current operation, run "osm ns-op-show OPERATION_ID"
iotworld@osmv7:~/paradigms_nfvi/general_services/osm$ 
```

FIGURE 64. INSTANTIATION OF THE NS THAT REPRESENTS VNF3 INTO THE NFVI

10 REFERENCES

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

[2] J. Serra, N. Petroulakis, E. Sakic, P. Wright and T. Marktscheffel, "Network Functions Virtualization for IoT (final)”," SEMIoTICS deliverable D3.8, 2020.

[3] M. Falchetto, D. Pau, A. Broering , P. Wright, D. Parra, L. Ciechomski, P. Gawron, C. Tzagkarakis and M. Michalodimitrakis, "Embedded Intelligence and Local Analytics (Final)," SEMIoTICS deliverable D4.10, April 2020.

[4] E. Sakic, E. Lakka, J. Serra, K. Fysarakis, M. Falchetto, D. Presenza, F. Klement, Ł. Ciechomski, K. Ramantas and U. Hansen, "SEMIoTICS High-Level Architecture (Final)," SEMIoTICS deliverable D2.5, December 2019.

[5] M. Falchetto, J. Serra, K. Ramantas and K. Fysarakis, "Demonstration and validation of IHES- Generic IoT (Cycle 1)," SEMIoTICS - Deliverable D5.6, Agusut 2020.

[6] "Docker," [Online]. Available: https://www.docker.com/. [Accessed April 2021].

[7] "Eclipse mosquitto," [Online]. Available: https://mosquitto.org/. [Accessed April 2021].

[8] "Docker Compose," [Online]. Available: https://docs.docker.com/compose/. [Accessed April 2021].

[9] R. Shumway, Time series analysis and its applications : with R examples, Springer, 2010.

[10] G. Hatzivasilis, J. Serra, K. Fysarakis, D. Presenza, M. Falchetto, T. Marktscheffel, Ł. Kempiński and P.-V. Mekikis, "Requirements specification of SEMIoTICS framework," SEMIoTICS deliverable D2.3, November 2018.