SEMIoTICS

Deliverable D5.4: Demonstration and Validation of IWPC - Energy Use Case (Cycle 1)

Deliverable release date	31/07/2020
Authors	Darko Anicic, Arne Bröring, Ermin Sakic (SAG), Ulrich Hansen (BWC), Manolis Michalodimitrakis (FORTH), Kostas Ramantas (IQU)
Responsible person	Ermin Sakic (SAG)
Reviewed by	Ermin Sakic, Vivek Kulkarni (SAG), Nikolaos Petroulakis (FORTH), Konstantinos Fysarakis (STS)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, Ermin Sakic, Mirko Falchetto, Domenico Presenza, Christos Verikoukis) PCC Members (Vivek Kulkarni, Nikolaos Petroulakis, Christos Verikoukis, Georgios Spanoudakis, Domenico Presenza, Danilo Pau, Joachim Posegga, Darek Dober, Kostas Ramantas, Ulrich Hansen)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Table of Contents

Acronyms Table	
Acronym	**Definition**
API	Application Programming Interface
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IIoT	Industrial IoT
IoT	Internet of Things
KPI	Key Performance Indicator
MDSP	MindSphere
MQTT	Message Queuing Telemetry Transport
NFV	Network Function Virtualization
OEDK	Open Edge Device Kit
PE	Pattern Engine
QoS	Quality of Service
REST	Representational State Transfer
SDN	Software Defined Networking
SSC	SEMIoTICS SDN Controller
UC1	Use Case 1
VM	Virtual Machine
W3C	World Wide Web Consortium
WoT	Web of Things

1 INTRODUCTION

Use Case 1 (UC1) is focused on demonstration of various SEMIoTICS components in a utility operator environment. UC1 demonstrates a total of four sub use-cases, focused on local edge analytics for processing unstructured data, interoperability with private and public cloud environment for use case of classic analytics of structured data and field layer monitoring, as well as the interoperability within the field layer in the utility network itself. The orchestration processes and industrial operation occur over an SDN capable of providing the necessary Quality of Service (QoS) guarantees to deployed network flows. Summarized, UC1 highlights the following benefits of using the SEMIoTICS platform:

- Usage of SEMIoTICS for simplified definition of the business logic, distribution and instantiation of applications in an edge device in the field layer;
- Usage of SEMIoTICS to achieve semantic interoperability between a greenfield device executing the business logic and a legacy brownfield environment connected to a wind turbine utilizing a common semantic abstraction layer;
- Usage of SEMIoTICS to achieve efficient “plug and play” on-boarding of a new field layer IIoT devices;
- Usage of SEMIoTICS to integrate field layer devices in interoperable manner with private and public clouds;
- Usage of integrated SEMIoTICS orchestration to deploy network services with deterministic QoS properties and;
- Usage of SEMIOTICS to guarantee Integrity and Connectivity properties in the system, using the pattern-based framework.

The purpose of D5.4 is to describe the business motivation behind the final set of sub-use cases, as well as to discuss their current state of implementation. The information provided in this document is to be updated in later iterations of the deliverable as the demonstrator matures to its final state.

2 USE CASE DESCRIPTION

2.1 Overview of the Use Case

The overall success criteria of UC1 is to demonstrate the coexistence of a highly integrated control system and an agile IIoT ecosystem based on the SEMIoTICS framework; which in return allows service providers to deploy new value-added services faster and provide effective access to data from both an existing 'brownfield' control system and a newly deployed 'greenfield' sensor network. The demonstrator of UC1 in the lab trial addresses some of the common challenges in extending the capabilities of an existing control system in a brownfield environment in the Wind Energy domain.

Control systems for industrial processes are in general terms heavily embedded in nature. Extending the capabilities of such a system typically involves engagement with component manufacturer or service integrator to fuse-in the new functionality that must be tested to validate the new component does not void the integrity and the intended behaviour of the control system. Most of these control systems expose certain high-level machine-readable interfaces for seamless integration with other systems, which is not enough to take full advantage of the value that can be generated.

The SEMIoTICS approach in UC1 provides an end-to-end eco-system allowing service integrators and service operators to build and deploy new value-added services, interfacing directly with the existing assets.

Overall, SEMIoTICS UC1 will demonstrate four concrete sub-use cases in a lab trial showing the IIoT field devices and applications:

- Sub-Use Case 1: Audio Processing Demo – Detection of loose objects;
- Sub-Use Case 2: Video Processing Demo – Oil/Grease leakage detection;
- Sub-Use Case 3: Inclination Measurement of wind turbine tower;
- Sub-Use Case 4: Availability Monitoring of Field Orchestration Layer devices.

The physical components involved in the developed demonstrator are visualized in Figure 1, and comprise:

a) A Wind Turbine model with rotating piece;
b) a SIMATIC S7 programmable logic controller (PLC) acting on the wind turbine rotor;
c) the greenfield sensory devices (microphone, camera, inclinometer);
d) six OpenFlow 1.3 SDN switches;
e) the SEMIoTICS SDN Controller (SSC) hosted on a Siemens Microbox Industrial PC (IPC);
f) the backend components of the SEMIoTICS platform hosted on a Siemens Nanobox Industrial PC;
g) the video and audio analytics software processes hosted on the two IPCs;
h) the integrated SEMIoTICS Gateway solution hosted on another IPC;
i) the LTE Gateway for enabling internet access and connectivity to remote clouds;
j) a management switch for enabling out-of-band control channel over SDN switches and visualization on an external device; and
k) external software application counterparts in the MindSphere cloud and a private cloud at IQU's premises.

The components of the SEMIoTICS platform deployed in UC1 are portrayed on the left in Figure 1 and include the:

- Recipe Cooker for orchestration of business logic;
- Pattern Orchestrator for translation of recipe descriptions into Integrity and Connectivity patterns;
- The Pattern Engine in the Application Orchestration Layer for ensuring Integrity constraints in the private cloud for the purpose of enabling the Sub-Use Case 4.
- The Pattern Engine in the SDN/NFV Orchestration Layer for ensuring Connectivity constraints in the local network for the purpose of enabling the Sub-Use Cases 1 and 2.

- Thing Directories in Application Orchestration Layer and Field Orchestration Layer for registration of field layer devices, their capabilities, and exposure of these to the Recipe Cooker.
- The backend GUI in the Application Orchestration Layer for visualization of the state of Pattern requirements and Thing Directories.
- The majority of sub-components of the SEMIoTICS SDN Controller for enabling end-to-end connectivity with delay, bandwidth, and redundancy constraints for point-to-point network connections;
- The majority of IoT Gateway components for enabling local embedded intelligence, semantic mediation between brownfield and greenfield field components and exposure of their capabilities and actions to external entities using Semantic API and Protocol bindings.

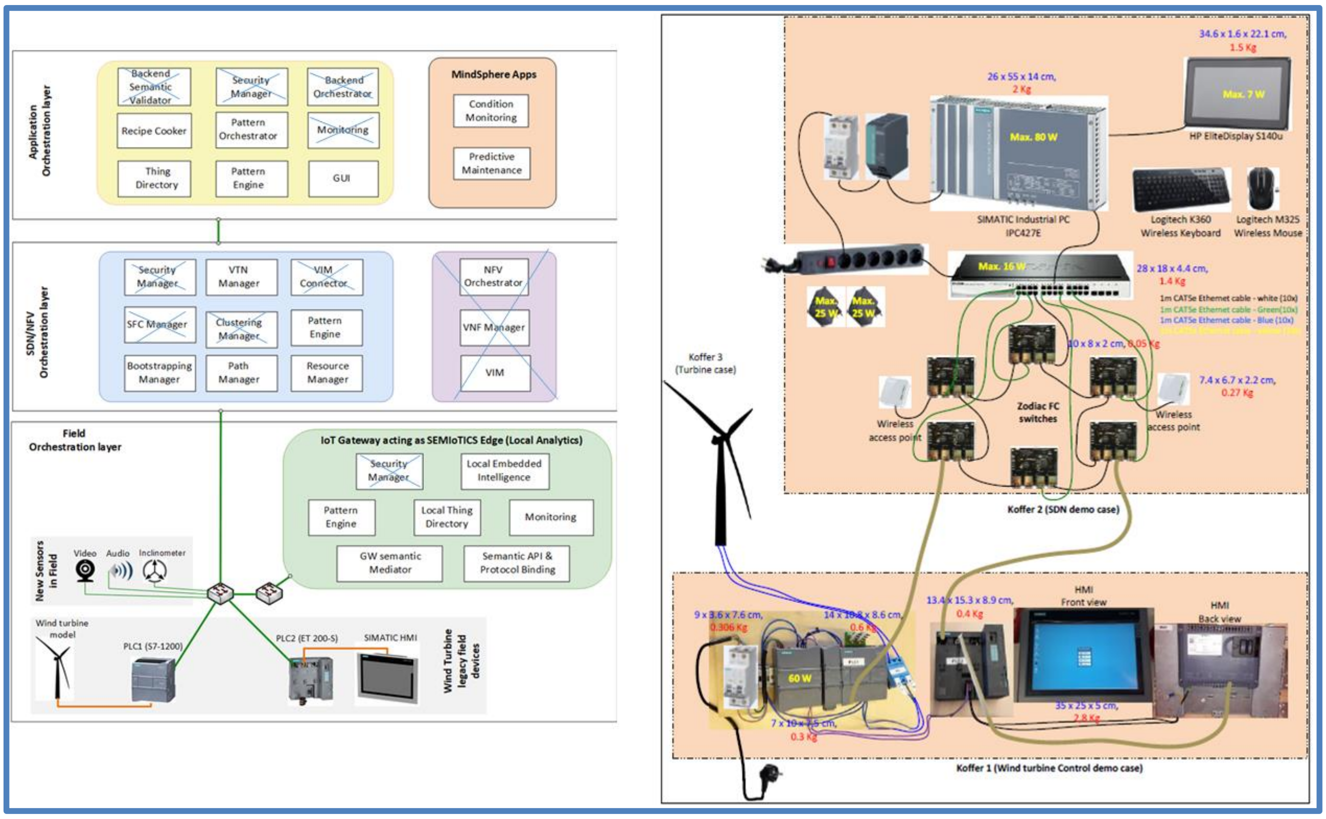


FIGURE 1: HARDWARE SETUP OF THE DEMO BAG

The exact deployment mapping of SEMIoTICS platform components to the underlying hardware resources and layers of SEMIoTICS architecture is depicted in Figure 2.

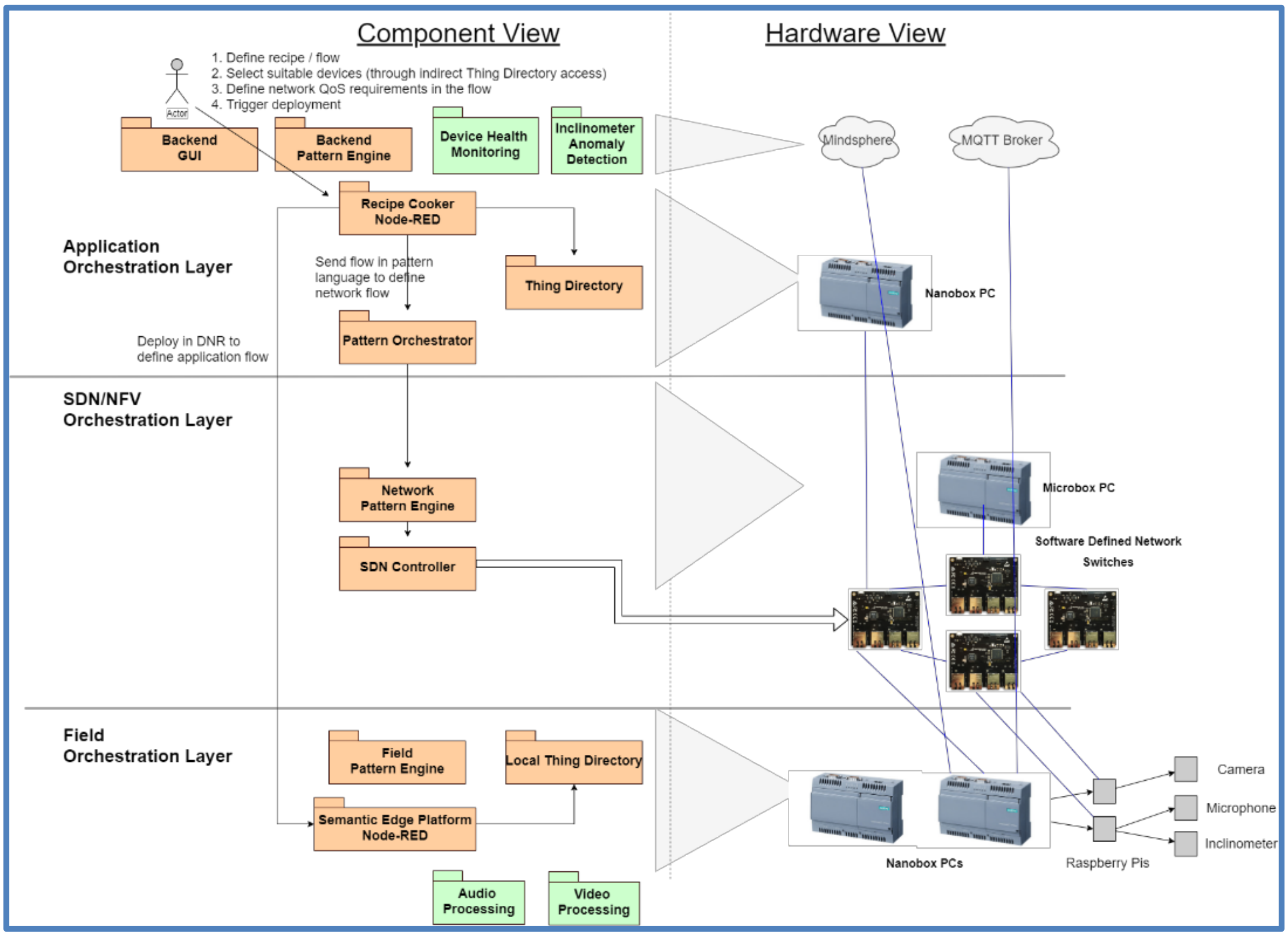


FIGURE 2: OVERVIEW OF KEY COMPONENTS IN USE CASE 1

2.2 Main Objectives of Use Case 1

Current wind turbine controllers in a wind park control network are typically highly integrated operating systems, which follow a long process of thorough development, testing, pre-qualification, and finally deployment in the real world. As a result of this long process, introduction of new features, addition of new sensors, actuators and related advancements require several months or even years to be fully matured and operational in the field. To improve this situation and shorten this development cycle, the UC1 aims and achieves the following objectives:

1) **Local predictive analytics for structured data, smart behavior and monitoring:** We deploy a localized edge to achieve semi-autonomous IIoT behavior where a minimal amount of data is transferred between the wind turbine and the remote control center, while the majority of pre-processing, analytics and triggering of the actuation steps happen at the 'edge' of the system, close to the actual turbine. In Sub-Use Cases 1 and 2 we demonstrate the sensing of unstructured data (e.g. video or audio) and acting locally to prevent any damage to the parts of the turbine in the long run. Additionally, a federation, based on a semantic data model and capabilities, is formed between the IIoT Gateway and the legacy control system in Sub-Use Cases 1, 2 and 3, so to enable the use of data exposed by the sensors connected to the legacy control system. This allows the operators to leverage the existing computing resources to make use of data exposed by an existing sensor infrastructure, in an automated manner.

2) **Multi-tenant capable network service establishment:** A wind park network typically comprises multiple stakeholders requiring access to a common network infrastructure. Providing reliable and Quality of Service (QoS)-constrained packet transmission for critical IoT services, in conjunction with non-real-time and bandwidth-heavy services necessitates service differentiation. The SSC support this case. In Sub-Use Cases 1 and 2, the users of the system, i.e. the operators, communicate their network requirements to the SSC by means of a simple service specification interface (Recipe Framework) that is eventually propagated to the SSC for embedding. Thus, all accepted stakeholders in the system are isolated with regards to the network resource utilization and are guaranteed their requested service invariants.
3) **Semantic-based engineering:** In an envisioned scenario, an engineer wants to add a new IIoT device (e.g., a sensor, actuator) or replace existing one with less effort. The device should be bootstrapped in a semi-automated fashion. Further on, it should expose its functionality over an IIoT Gateway via a machine-interpretable interface so that the engineer can develop a new service or application with that device. UC1 demonstrates that the integration and configuration of a new IIoT device in an automation system of a wind park requires very low effort. For this purpose, the greenfield IIoT devices are equipped with a semantic description, which support the process of device discovery, as well as commissioning, engineering and re-engineering of automation functions in an ecosystem of heterogeneous devices and services. The SEMIoTICS semantic models, which enable device descriptions, are based on standardized semantics such W3C Web of Things1 (Thing Description), as well as on the IoT domain semantics created by iot.schema.org2. SEMIoTICS thus supports creation of IIoT applications in a cost-effective manner. The brownfield devices, on the other hand, are similarly supported with their semantics modeled and exposed, by means of a combination of an intermediate semantic API and the semantic binding.

4) **Semantics-based application creation** – Going beyond integration and configuration supported by the “semantic-based engineering” (described above), an application developer needs to be supported in connecting and composing services and devices to Edge-level or Cloud-level applications.

   a. In order to facilitate quicker development of applications, the Recipe Framework allows for compositions by the developer as templates (or “recipes”) that describe how logical links are established and what kind of devices are required in these compositions.

   b. As part of these definitions of compositions, the developer needs to be able to define Security, Integrity, Connectivity, Privacy or Dependability criteria on the connections for the logical links of the distributed application. These criteria are semantically described and are interpretable by the Pattern Engine instances prior to the deployment of the application in order to evaluate whether the infrastructure can meet the requirements of the application. Thus, with the seamless integration of specification of pattern requirements with the application workflow design, the developer is capable of both orchestrating the available resources to deploy and execute the application, but also automatically enabling, validating and monitoring additional security and connectivity invariants from a single interface.

2.3 SEMIoTICS Sub-Use Cases

Prior to discussing the details of the individual sub-use cases, as well as showing the actual state of implementation and integration, we henceforth present a summary of use case workflows.

Sub-Use Case 1 - Audio Processing Demo – Detection of Loose Objects: Sub-Use Case 1 demonstrates the integration of a distributed audio processing function with the remainder of SEMIoTICS framework. The analytics function detects a loose object in the wind turbine by processing a stream of unstructured audio data

[1] https://www.w3.org/WoT/WG/
[2] Currently available from: http://iotschema.org/

and issuing an emergency stop command to the wind turbine if an abnormality is detected while the wind turbine is in operation. Numerous greenfield and brownfield field layer devices are involved in the operation. Furthermore, the analytics function is realized by deploying an application to an edge device using orchestration introduced by SEMIoTICS and involves various other components of the framework i.e., the SSC and IIoT Gateway to guarantee the network invariants, and expose audio state to external applications, respectively.

Sub-Use Case 2 - Video Processing Demo – Oil/grease leakage detection: This sub use case focuses on demonstrating video processing capabilities deployed at the field layer, to detect a grease leakage from a main bearing in a wind turbine, by processing a stream of unstructured video data and, as above, issuing an emergency stop command to the wind turbine if a severe grease leakage is confidently detected. The capability is realized by deploying an application to multiple edge device installed at the field layer utilizing various components of the SEMIoTICS framework.

Sub-Use Case 3 – Inclination Measurement of the Tower: An application deployed in an external IoT platform, i.e., MindSphere, consumes the inclination data measured by an inclinometer in the field layer, along with other operational data from the simulated model wind turbine environment, for visualization and manual decision making. The cloud application furthermore periodically evaluates the reported inclination values and triggers an alarm if a set threshold is exceeded, i.e., enable the operator to act. Only raw data from the inclinometer is being transferred to the cloud and all business logic for decision making is being defined in the cloud application.

Sub-Use Case 4 – Field Devices Availability Monitoring: The final business application is being deployed in the private environment with the purpose of monitoring the deployed inclinometer and other greenfield sensors in the system for availability purposes. As the inclinometer sensor is not natively integrated with the existing control and instrumentation system of the wind turbine, it is not being monitored by the native turbine monitoring system. Instead, a small agent deployed at the field layer continuously monitors the availability of the deployed inclinometer sensor and forward a heartbeat signal to the private cloud for monitoring purposes. In this sub-use case only availability information of the sensor is being transmitted to the private cloud, whereas all operational data recorded is being transmitted to the public cloud application as in the Sub-Use Case 3. Finally, the integrity of data transmitted to the private cloud is guaranteed with the use of pattern framework.

3 SUB USE CASE 1: AUDIO PROCESSING DEMO – DETECTION OF LOOSE OBJECTS

3.1 Scope and Objectives

This sub use case focuses on demonstrating analytical capabilities deployed at the field layer, to detect a loose object in the wind turbine by processing a stream of unstructured audio data and issuing an emergency stop command to the wind turbine if an anomaly is detected while the wind turbine is in operation. The capability is realized by deploying an application to an edge device installed at the field layer utilizing various components of the SEMIoTICS framework.

The analytical capability is implemented, using an approach based on machine learning techniques to detect an abnormal change in the audio landscape. The system, if deployed in real life, is to be installed either inside or in proximity of the wind turbine's hub. The hub is the casted mechanical structure at the very front of the wind turbine, where the three blades are fastened to.

The business objective is to detect a foreign object in hub during operation and stop the wind turbine before any severe damage occurs to the internal parts of the hub. The hub contains several critical hydraulic and electrical systems responsible for controlling the individual blades angle in the wind. This system is known as the pitch control system and is in constant rotation while the wind turbine is in operation. If parts of this critical system are damaged, causing the embedded wind turbine controller to detect an error in the pitch control system, the wind turbine will stop automatically with an error and will be unable to produce electricity until it has been manually inspected and repaired.

Depending on the severity of the damage caused by loose object, the cost of bringing the turbine back into operation can be very costly in terms of direct expenditure related to the repair work, but lost revenue, as the wind turbine is unable to produce until repaired, is likely to account for the largest part of the cost.

Loose objects are typically internal mechanical parts to the hub or the three blades, that have been shook loose during operation or seldom forgotten hand tools by wind turbine technicians. Examples of loose objects could be broken and/or loose bolts in the interface between the hub and the blade bearings, plastic covers or access covers to the blades.

The scope of the demonstrator focuses on addressing the following advancements:

- The use of select SEMIoTICS components to define the business logic, distribute and to instantiate the application in an edge device at the field layer
- To demonstrate the semantic interoperability between a greenfield device executing the business logic and a simulated legacy brownfield environment connected to a model wind turbine utilizing a common semantic abstraction layer
- The implementation of the business logic focusing on the use of unsupervised machine learning to detect an anomaly in the stream of audio. The demonstrator does not focus on extending the capabilities of the business application required for adoption by industry.

3.2 Interaction with SEMIoTICS Framework and Components

Successful deployment of the business application, for detecting a loose object in the wind turbines' hub, relies on several components provided by the SEMIoTICS framework along with components implemented for enabling the demonstrating of the use case specifically. An overview of the components and their use is provided in Table 1.

TABLE 1: INVOLVED SEMIOTICS COMPONENTS IN SUB-USE CASE 1

Component	**SEMIoTICS component**	**Architecture reference**	**Description of use**
Semantic Edge Platform	✓	Field Layer	• Provides capabilities for initial device bootstrapping and discovery • Hosting Local Thing Directory service • Provides API for interaction with other devices enrolled to SEMIoTICS architecture
Semantic API and Protocol Binding	✓	Field Layer	• Process capabilities for interacting with brownfield environments, e.g. the legacy control system
Edge device		Field Layer	• Raspberry Pi device with an embedded microphone for capturing the audio • Anomaly detection process, based on unsupervised machine learning • Executing the business logic
Legacy Control System		Field Layer	• Simulation of Wind Turbine Control System based on Power Logic Controllers and an operational and controllable wind turbine model
Recipe Cooker	✓	Application Layer	• Recipe Cooker framework based on Node Red is used to define and deploy the business logic to the field device in a recipe style format
Pattern Orchestrator	✓	Application Layer	• Pattern Orchestrator is used to translate the recipe (business logic request) into a set of patterns comprising Connectivity requirements.
Network Pattern Engine	✓	SDN/NFV Layer	• Pattern Engine is used to trigger embedding of networking services in the SDN via remaining SDN Controller components, under consideration of delay and bandwidth constraints stemming from Connectivity pattern properties.
SDN Controller	✓	SDN/NFV Layer	• SDN Controller sub-modules, such as Path Finding, Registry Handler, and Resource Manager are used to compute the network paths necessary to fulfill the Connectivity pattern constraints, refer to and keep track of reserved network resources, as well as

			to embed the service via external configuration of SDN switches.
Global Thing Directory	✓	Application Layer	• Orchestration of devices and capabilities registered with the Local Thing Directory • Information made available to the Recipe Cooker through an API

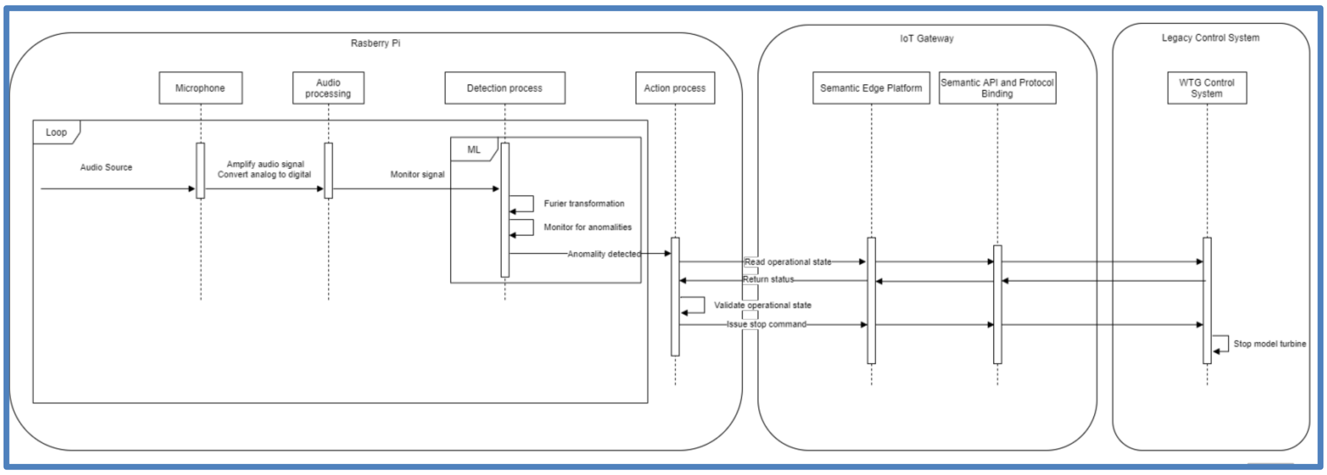


FIGURE 3: OVERVIEW OF DETECTION OF LOOSE OBJECT APPLICATION

Figure 3 shows an overview of the business logic view for detection of loose objects. As a prerequisite, the field device (Raspberry Pi with the microphone attached) has been successfully bootstrapped, is connected to the communication network and on boarded in the SEMIoTICS eco-system. The application is defined in the Recipe Cooker and the business logic is deployed to the field device.

Audio processing: captures and process sound recorded by the microphone. The audio process bridges between the physical and digital world: capturing raw sound, amplifies the signal in hardware and converts the analog signal to a digital format. The digitized raw audio signal is then used by the detection process.

Detection process: relying on Fast Fourier Transformation (utilizing Python library Librosa[3]) of the unstructured audio signal source, the detection processed creates a structured pattern from the audio signal and compares it with those known to the neural network. The machine learning model has been pre-trained with a finite number of audio samples from a normal operational environment. If an anomaly in the audio signal is detected – indicating a detection of a loose object in the wind turbine – the process will forward an event notification to the action process.

Action process: This process is being triggered whenever the detection process detects an anomaly. The process will first determine the running state of the model wind turbine. If the turbine is not in an operational mode (i.e. it's in service mode or not operational for other reasons), it will simply ignore the anomaly detected. If, however, the wind turbine is in an operational mode and producing electricity when the anomaly is detected, the process assumes the anomaly to be legit, and will immediately issue a stop command to the model wind turbine.

[3] https://pypi.org/project/librosa/

Semantic Edge Platform: provides an API that enables the action process to interact with another device enrolled to the SEMIoTICS architecture. The API exposes enrolled devices as Device Nodes and their associated capabilities sourced from Thing Descriptions. It ultimately enables the application to interact with the legacy control system.

Semantic API and protocol binding: responsible for the translation between Web of Things protocol and the last mile connection to the simulated brownfield control system, that is using a legacy industrial protocol. It provides a uniform semantic API for all connected devices. This service is the primary enabler to bridge greenfield edge technology with existing brownfield installations. When the stop command has been received by the action process, the service will forward the stop command to the model wind turbine via the industrial communication protocol that is supported by simulated wind turbine control system.

Wind turbine control system: Is the autonomous simulation of a wind turbine control system. It will take the action to stop the model turbine when the command is received. The simulated control system exposes the operational state of the model wind turbine, which is used by the action process to determine the authenticity of the anomaly detected.

3.3 Setup Testbed and Integration

Prerequisite for this sub use case to work is that all systems are operational and connected to the SEMIoTICS framework, i.e., the Smart Audio Processing device is connected and registered with the Thing Directory.

Furthermore, the capabilities of the legacy control system are loaded to the Thing Directory and the protocol binding service is operational. These Thing capabilities are replicated to the Global Thing Directory so that services can find them. The Backend GUI is capable of visualizing the set of capabilities of involved field layer devices based on the content of the global Thing Directory. The IoT Gateway is operational with excess capacity to execute the application. Finally, the Recipe Cooker framework is operational and the application flow for the audio fault detection is already setup.

Figure 4 shows the deployment setup of Sub-Use Case 1 broken down to actual machines and installed software components on the right.

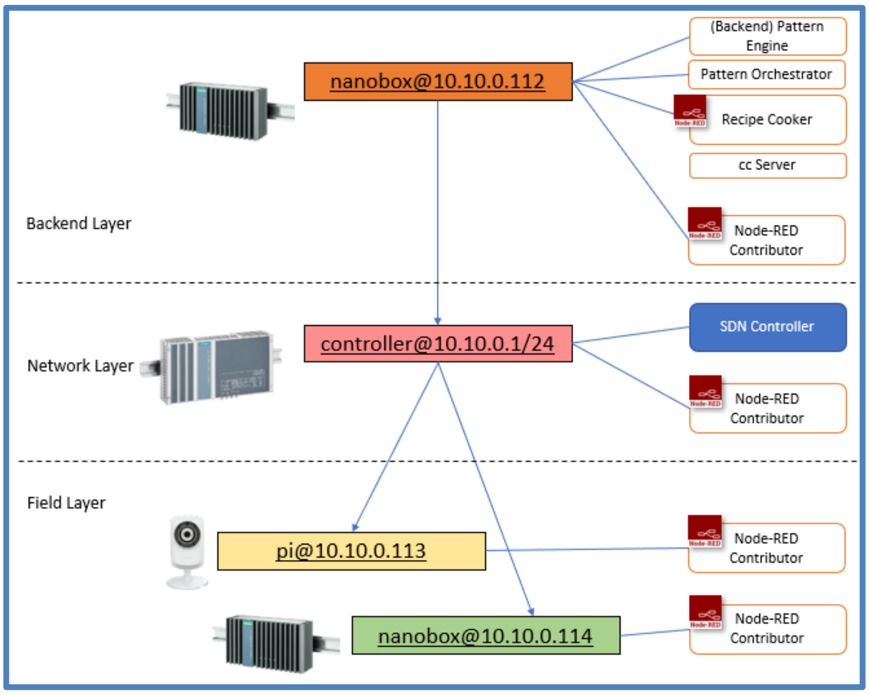


FIGURE 4: DEPLOYMENT SETUP

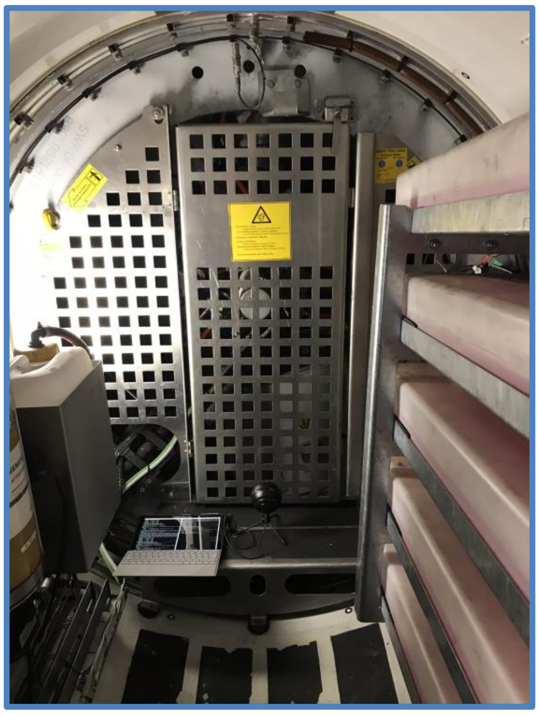


FIGURE 5: MICROPHONE (AT THE BOTTOM OF THE DOOR) INSTALLED IN WIND TURBINE

Figure 5 shows the microphone mounted within the housing of a wind turbine. The flow of the audio processing that is setup in the Recipe Cooker is shown in Figure 6. After triggering the application, recording using the microphone is done for a defined timeframe. Afterwards the recorded audio is sliced into samples of predefined length. The audio samples are then transformed into images using Fourier transformation. These images are then fed into the AI model for inference to identify whether it is an anomaly (e.g., loose objects making tumbling noise) or a regular sound.

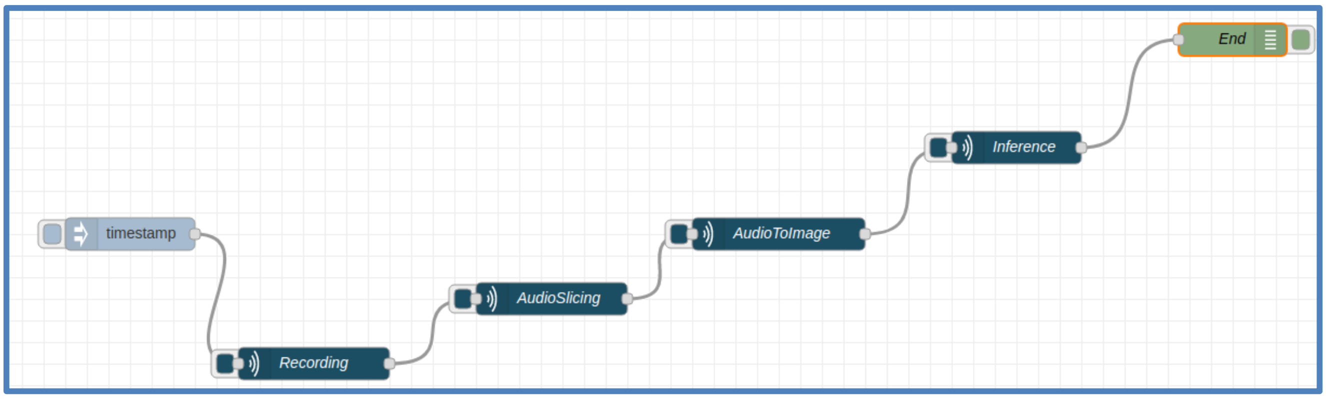


FIGURE 6: FLOW OF THE AUDIO PROCESSOR

Figure 7 shows example images resulting from Fourier Transformation of audio samples: on the left is a normal sound sample transformed into an image and on the right is an abnormal sound sample.

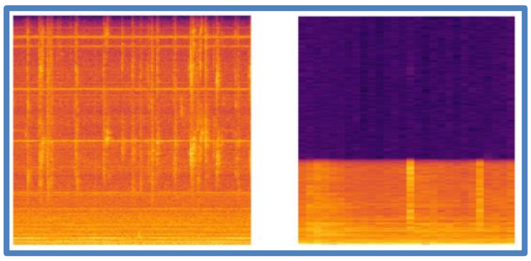


FIGURE 7: AUDIO FILES AFTER FOURIER TRANSFORMATION

4 SUB USE CASE 2: VIDEO PROCESSING DEMO – OIL/GREASE LEAKAGE DETECTION

4.1 Scope and Objectives

This sub use case focuses on demonstrating analytical capabilities deployed at the field layer, to detect a grease leakage from a main bearing in a wind turbine, by processing a stream of unstructured video data and issuing an emergency stop command to the wind turbine if a severe grease leakage is confidently detected. The capability is realized by deploying an application to multiple edge device installed at the field layer utilizing various components of the SEMIoTICS framework.

The analytical capability is implemented is using an approach based on Machine Learning techniques to detect grease leakages in the video feed images. A large set of training images of a small-scale mock-up has been created and used to train the model in advance of deploying the business logic to the field.

The business objective is to prevent an uncontrolled grease leakage from the main bearing by monitoring and eventually stopping the wind turbine as a preventive maintenance action.

Main bearings on large modern wind turbines are designed to consume and loose some grease during normal operation and are fitted with automating greasing systems that ensure the bearing is re-greased with a regular interval. Dependent on the implementation by the wind turbine manufacturer, the bearings are re-greased either at fixed time intervals or ad-hoc by use of more advanced monitoring techniques involving continuously monitoring the pressure inside the bearings. If a lower limit threshold is exceeded the system will re-grease the bearing.

Larger grease leakages could indicate an issue with the seal of the bearing that needs repair or replacement. An early detection of the issue enables the operator to monitor the development of the leakage and plan appropriate measures remedy the issue before stopping the turbine becomes as necessity. A stopped turbine means loss of revenue for the operator.

Other industrial implementations may not have an automated re-greasing and monitoring system put in place; thus, early detection is critical to avoid mechanical failure.

The scope of the demonstrator focuses on addressing following advancements:

- The use of select SEMIoTICS components to define the business logic, distribute and to instantiate the application in an edge device at the field layer
- To demonstrate the semantic interoperability between a greenfield device executing the business logic and a simulated legacy brownfield environment connected to a model wind turbine utilizing a common semantic abstraction layer
- The implementation of the business logic focuses on the use of supervised machine learning to detect a leak in a mock-up of a bearing. The demonstrator does not focus on extending the capabilities of the business application required for adoption by industry.

4.2 Interaction with SEMIoTICS Framework and Components

Successful deployment of the business application, for detecting a loose object in the wind turbines' hub, relies on several components provided by the SEMIoTICS framework along with components implemented for enabling the demonstrating of the use case specifically. An overview of the components and their use are listed the table below.

TABLE 2: INVOLVED SEMIOTICS COMPONENTS IN SUB-USE CASE 2

Component	**SEMIoTICS component**	**Architecture reference**	**Description of use**
Semantic Edge Platform	✓	Field Layer	• Provides capabilities for initial device bootstrapping and discovery • Hosting Local Thing Directory service • Provides API for interaction with other devices enrolled to SEMIoTICS architecture
Semantic API and Protocol Binding	✓	Field Layer	• Process capabilities for interacting with brownfield environments, e.g. the legacy control system
Edge device		Field Layer	• Raspberry Pi device with an external camera connected for capturing the video stream • Anomaly detection process, based on machine learning • Executing the business logic
Legacy Control System		Field Layer	• Simulation of Wind Turbine Control System based on Power Logic Controllers and an operational and controllable wind turbine model
Recipe Cooker	✓	Application Layer	• Recipe Cooker framework based on Node Red is used to define and deploy the business logic to the field device in a recipe style format
Global Thing Directory	✓	Application Layer	• Orchestration of devices and capabilities registered with the Local Thing Directory • Information made available to the Recipe Cooker through an API
Pattern Orchestrator	✓	Application Layer	• Pattern Orchestrator is used to translate the recipe (business logic request) into a set of patterns comprising Connectivity requirements.
Network Pattern Engine	✓	SDN/NFV Layer	• Pattern Engine is used to trigger embedding of networking services in the SDN via remaining SDN Controller components, under consideration of delay and bandwidth constraints stemming from Connectivity pattern properties.
SDN Controller	✓	SDN/NFV Layer	• SDN Controller sub-modules, such as Path Finding, Registry Handler, and

			Resource Manager are used to compute the network paths necessary to fulfill the Connectivity pattern constraints, refer to and keep track of reserved network resources, as well as to embed the service via external configuration of SDN switches.

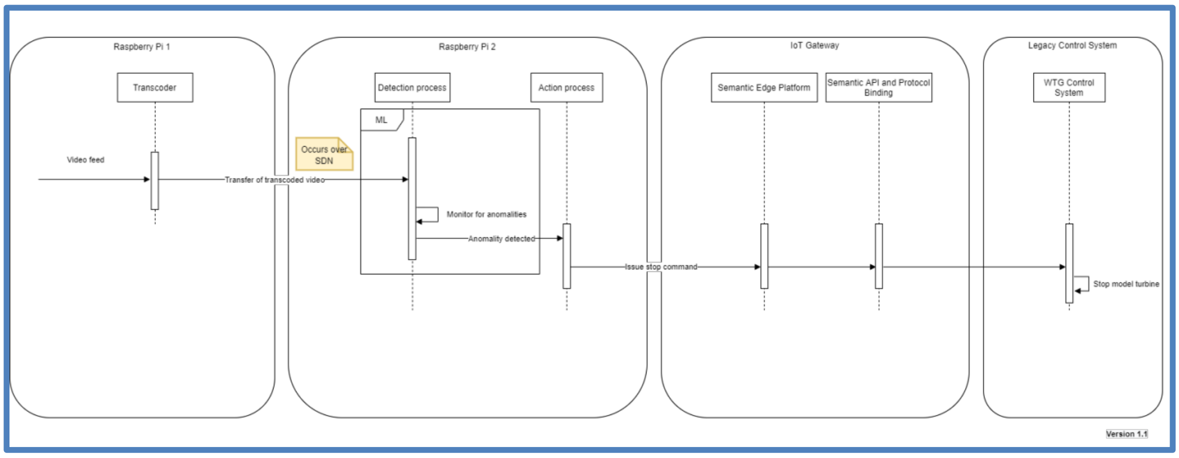


FIGURE 8: OVERVIEW OF GREASE LEAKAGE DETECTION APPLICATION

Figure 8 shows an overview of the business logic view for grease leakage detection. As a prerequisite, the field devices (consists of Raspberry Pi with the camera attached and another NanoBox running the anomaly detection application) has been successfully bootstrapped, is connected to the communication network and on boarded in the SEMIoTICS eco-system. The application is defined in the Recipe Cooker and the business logic is deployed to the field device.

Transcoder: captures the video feed from the camera and encodes the signal to reduce the size of the video when transmitted via the SDN network to the detection process, which is hosted and executed on a different edge device in this demonstrator.

Detection process: processes the encoded video feed in a machine learning model with the purpose of detecting an anomaly. The machine learning model has been pre-trained with a finite number of image samples from a simulated environment, enabling the model to determine when a leakage occurs with a high accuracy. If the model determines - with significant confidence - that a leakage under development the process will forward an event notification to the action process.

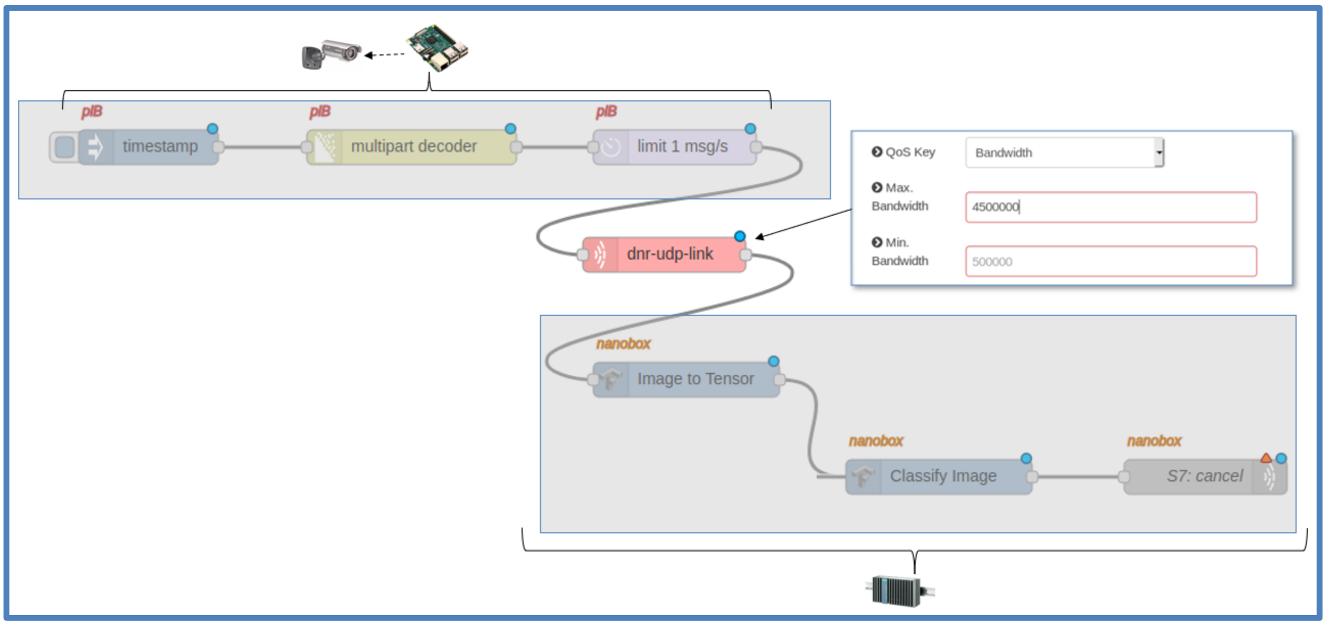
Action process: This process is being triggered whenever the detection process detects an anomaly. The process will immediately issue a stop command to the model wind turbine.
Semantic Edge Platform: provides an API that enables the action process to interact with another device enrolled to the SEMIoTICS architecture. The API exposes enrolled devices as Device Nodes and their associated capabilities sourced from Thing Descriptions. It ultimately enables the application to interact with the legacy control system.

Semantic API and protocol binding: responsible for the translation between Web of Things protocol and the last mile connection to the simulated brownfield control system, that is using a legacy industrial protocol. It provides a uniform semantic API for all connected devices. This service is the primary enabler to bridge greenfield edge technology with existing brownfield installations. When the stop command has been received by the action process, the service will forward the stop command to the model wind turbine via the industrial communication protocol that is supported by simulated wind turbine control system.

Wind turbine control system: Is the autonomous simulation of a wind turbine control system. It will take the action to stop the model turbine when the command is received.

4.3 Setup Testbed and Integration

Prerequisite for this sub use case to work is that all systems are operational and connected to the SEMIoTICS framework, i.e. the video processing unit is connected and registered with the Thing Directory. The capabilities of legacy control system are registered at the Thing Directory. The protocol binding service is operational, and the thing capabilities are replicated at the Global Thing Directory. Furthermore, the IoT Gateway is operational with excess capacity to execute the application. Finally, the Recipe Cooker framework is operational, and the application flow is designed and ready to be deployed.



QoS Key: Bandwidth
Max. Bandwidth: 4500000
Min. Bandwidth: 500000

FIGURE 9: APPLICATION FLOW FOR THE VIDEO PROCESSING AND ANALYSIS

5 SUB USE CASE 3: INCLINATION MEASUREMENT OF TOWER

5.1 Scope and Objectives

This sub use case focuses on “plug and play” onboarding capabilities provided by the SEMIoTICS framework, demonstrating integration of a third-party inclination sensor for deployment in a wind turbine. An inclination sensor is used to measure the orientation angle of an object in relation to the gravitational field of the earth.

Furthermore, it focuses on publishing selected data to both public and private cloud environment, demonstrating the use of various SEMIoTICS components for interoperability from field layer to cloud. Two business applications deployed in the cloud, to make use of the data, is being created for this purpose.

One business application, deployed in the public cloud, will consume the measured inclination data along with other operational data from the simulated model wind turbine environment, for visualization and manual decision making. The cloud application will monitor the inclination values transmitted and enable the operator to act if a set threshold is exceeded. Only raw data from the inclinometer is being transferred to the cloud and all business logic for decision making is being defined in the cloud application.

The second business application is being deployed in the private environment with the purpose of monitoring the deployed inclinometer sensor for availability purposes. As the inclinometer sensor is not natively integrated with the existing control and instrumentation system of the wind turbine, it is as well not being monitored. The part of the application, which is deployed at the field layer, will continuously monitor the availability of the deployed inclinometer sensor and forward a heartbeat signal to the private cloud for monitoring purposes. In this example, only availability information of the sensor is being transmitted to the private cloud, whereas all operational inclination data recorded is being transmitted to the public cloud application.

The business objective is to enable continuous monitoring of the structural health of the wind turbine tower, by continuously monitoring the X/Y sway of the tower. It is common in the wind industry to assess and offer lifetime extending upgrade packages for brownfield installations when they are nearing the end of their original design life, e.g. by performing extended service, inspections and upgrades that would extend the lifetime of a wind turbine by example from 25 to 30 years.

The tubular steel tower of the wind turbine is not easy to or economically viable to exchange for such an upgrade. Inclination sensors can then be fitted at certain locations inside the tower, to continuously monitor the sway of the tower in relation to the current operational and weather conditions, to aid structural engineers to determine to health of the structure.

Steel towers loose its strength over time due to fatigue caused by the cycling loads, when the turbine is both in stand-still and in operational mode, therefore it’s important to monitor the condition of the tower when the structure is nearing the end of its designed life.

The scope of the demonstrator focuses on addressing following advancements:

- The use of select SEMIoTICS components to define the business logic, distribute and to instantiate the application in an edge device at the field layer
- To demonstrate the efficient “plug and play” onboarding of a new field layer IIoT device utilizing the SEMIoTICS framework
- Integration of field layer devices and exposure of data to both private and public cloud.

- Example of a business applications, deployed at the field layer, in public and private cloud environment, to monitor the inclination of the wind turbines tower and the availability of the inclinometer sensor.

5.2 Interaction with SEMIoTICS Framework and Components

The primary components used will be the Recipe Cooker eco system, backend GUI, the network layer and the data forward application running on the SEMIoTICS IoT Gateway. Furthermore, the connectivity to the cloud is demonstrated. An application to visualize the incoming flow of data from the sensor is defined in MindSphere. The Backend GUI visualizes the set of and capabilities of newly detected field layer devices, based on the content of local / global Thing Directory.

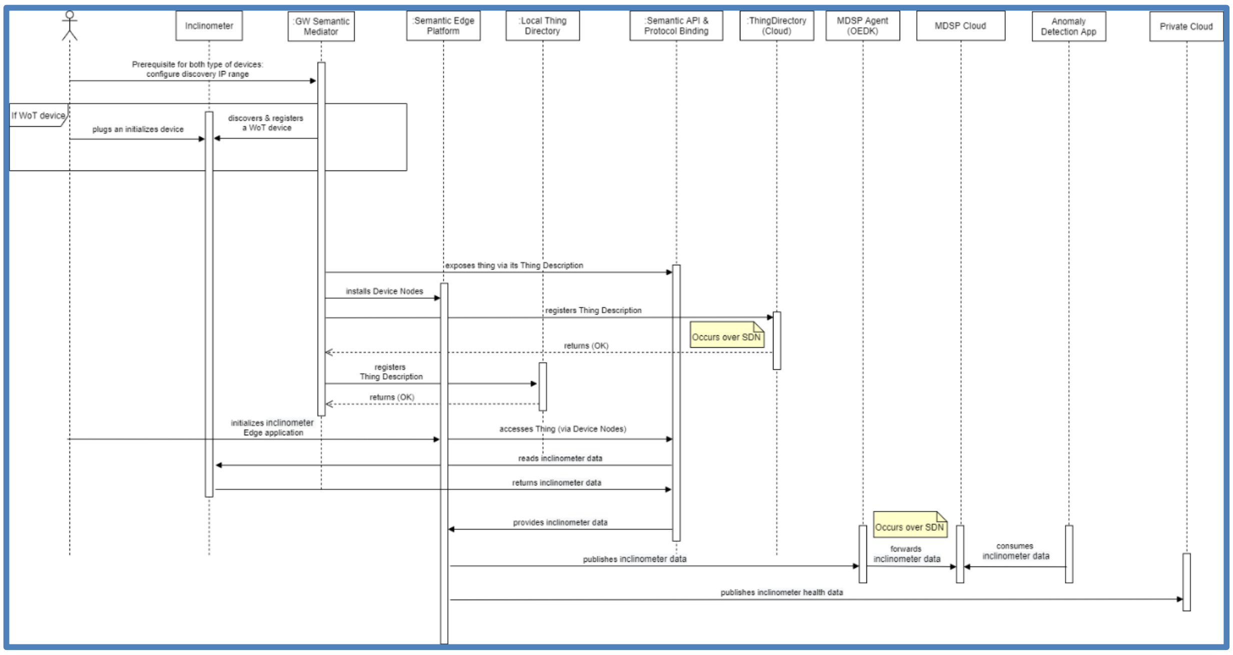

Inclinometer
:GW Semantic Mediator
:Semantic Edge Platform
:Local Thing Directory
:Semantic API & Protocol Binding
:ThingDirectory (Cloud)
MDSP Agent (OEDK)
MDSP Cloud
Anomaly Detection App
Private Cloud
Prerequisite for both type of devices: configure discovery IP range
If WoT device
plugs an initializes device
discovers & registers a WoT device
exposes thing via its Thing Description
installs Device Nodes
registers Thing Description
Occurs over SDN
returns (OK)
registers Thing Description
returns (OK)
initializes inclinometer Edge application
accesses Thing (via Device Nodes)
reads inclinometer data
returns inclinometer data
provides inclinometer data
Occurs over SDN
publishes inclinometer data
forwards inclinometer data
consumes inclinometer data
publishes inclinometer health data

FIGURE 10: OVERVIEW OF INCLINATION MEASUREMENT OF TOWER

Figure 10 shows an overview of the inclination measurement sub-use case, starting from the bootstrapping process of an inclinometer up to the process of use of data in two Clouds. An operator installs a n inclinometer in the tower and plugs it in the communication network where IIoT Gateway operates. Semantic Mediator will fetch discover and register the device, using the Thing Description of the device. It will store the description in Local and Global Thing Directory. Further on, it will expose the device over the gateway's API and generate Device Node to interact with the device (based on information from Thing Description). Once the bootstrapping process is completed, the device can be used in an application. For the purpose of this sub-use case we will provide two applications, one deployed in MindSphere Cloud and another one in a private Cloud (refer to Sub-Use Case 4 as well). We will also provide an Edge application that just puts the inclinometer data in a data format suitable for Clouds and in addition creates device-health data (e.g., hart-beat data stating that the inclinometer is alive). The Edge application will publish this data to both Clouds. MindSphere Cloud requires that the inclinometer data is sent to the Cloud over a special agent (MDSP Agent). We implanted the agent by using Open Edge Device Kit (OEDK)[4]. Device-health data is sent to Private Cloud directly (with no agent) over MQTT.

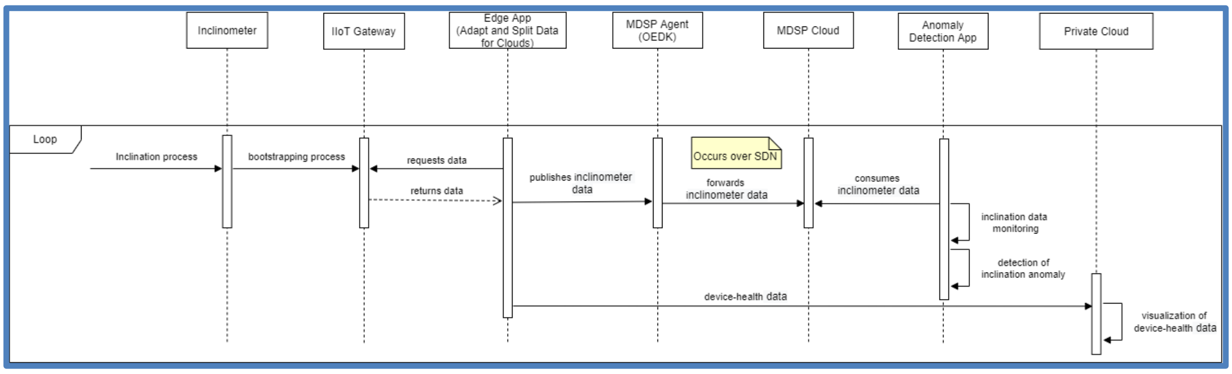

Inclinometer
IIoT Gateway
Edge App (Adapt and Split Data for Clouds)
MDSP Agent (OEDK)
MDSP Cloud
Anomaly Detection App
Private Cloud
Loop
Inclination process
bootstrapping process
requests data
returns data
publishes inclinometer data
Occurs over SDN
forwards inclinometer data
consumes inclinometer data
inclination data monitoring
detection of inclination anomaly
device-health data
visualization of device-health data

FIGURE 11: APPLICATIONS RELATED TO INCLINATION MEASUREMENT OF TOWER

Figure 11 depicts details about the two applications that are based on data from the inclinometer. The user initializes both applications over Semantic Edge Platform (IIoT Gateway) as shown in Figure 10. The first application continuously monitors the sway of the tower and detects anomaly (the tower inclination exceeds the allowed threshold). This application is deployed in MindSphere Cloud. The second application monitors the health status of the inclinometer and visualizes this data. The application is deployed in a private Cloud.

5.3 Setup Testbed and Integration

In the following paragraphs, we describe the testbed setup and integration results for the Sub-Use Case 3. In this test we successfully on-boarded an inclinometer sensor via SEMIoTICS IIoT Gateway and showed its data in MindSphere Cloud. The test succeeded via communication enabled by SEMIoTICS SDN network.

Figure 12 shows the inclinometer, which is hosted by a Web of Things (WoT) Device. This device represents a greenfield device that is added to brown-field installations with the goal of continuously monitoring the condition and health of the wind turbine tower. The inclinometer is expected to become fully functional in the overall automation system with minimal effort, i.e., to operate in plug-and-play manner.

SEMIoTICS IIoT Gateway on-boards the inclinometer. This process includes discovery of the newly plugged device, registration of its semantic meta data (Thing Description) with the Local and Global Thing Directory, and the exposure of the sensor's data over unified (WoT) API. IIoT Gateway and the newly plugged device are not directly connected. The gateway scans the network and discovers the device. Based on Thing Description, the gateway is enabled to perform the onboarding procedure in an automated fashion.

[4] https://support.industry.siemens.com/cs/document/109766079/open-edge-device-kit-(oedk)?dti=0&lc=en-YE

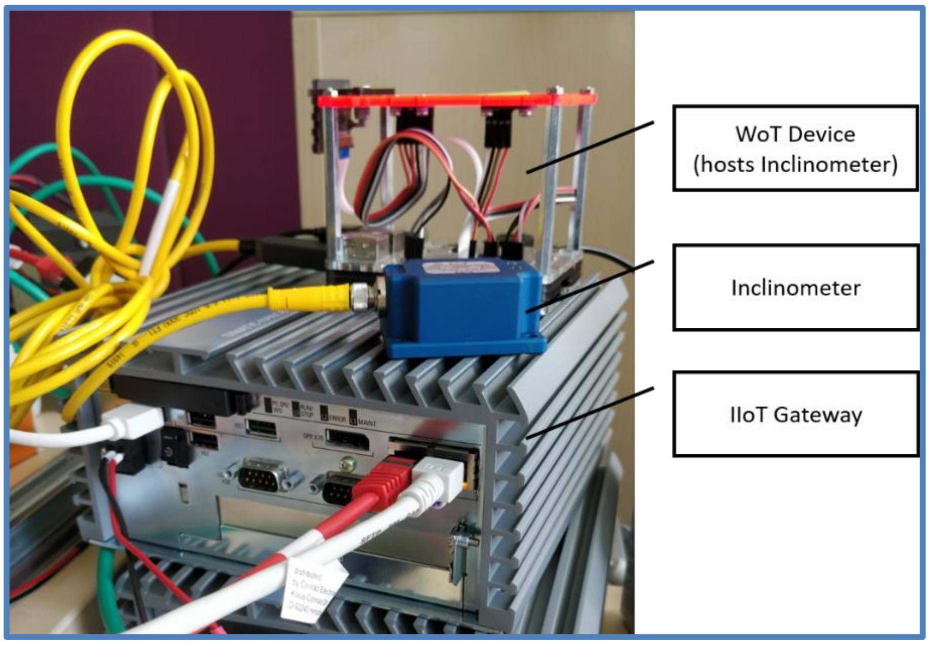


FIGURE 12: HARDWARE FOR INCLINOMETER SUB-USECASE

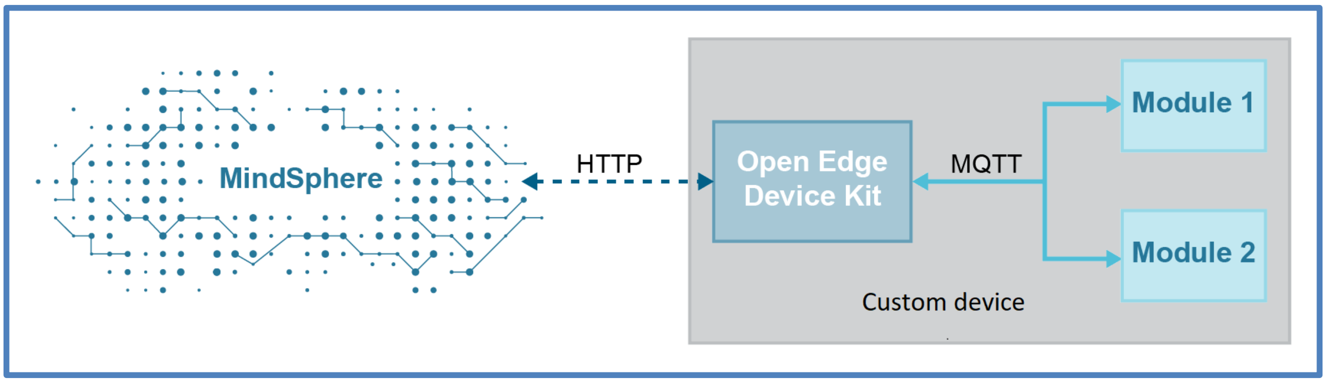
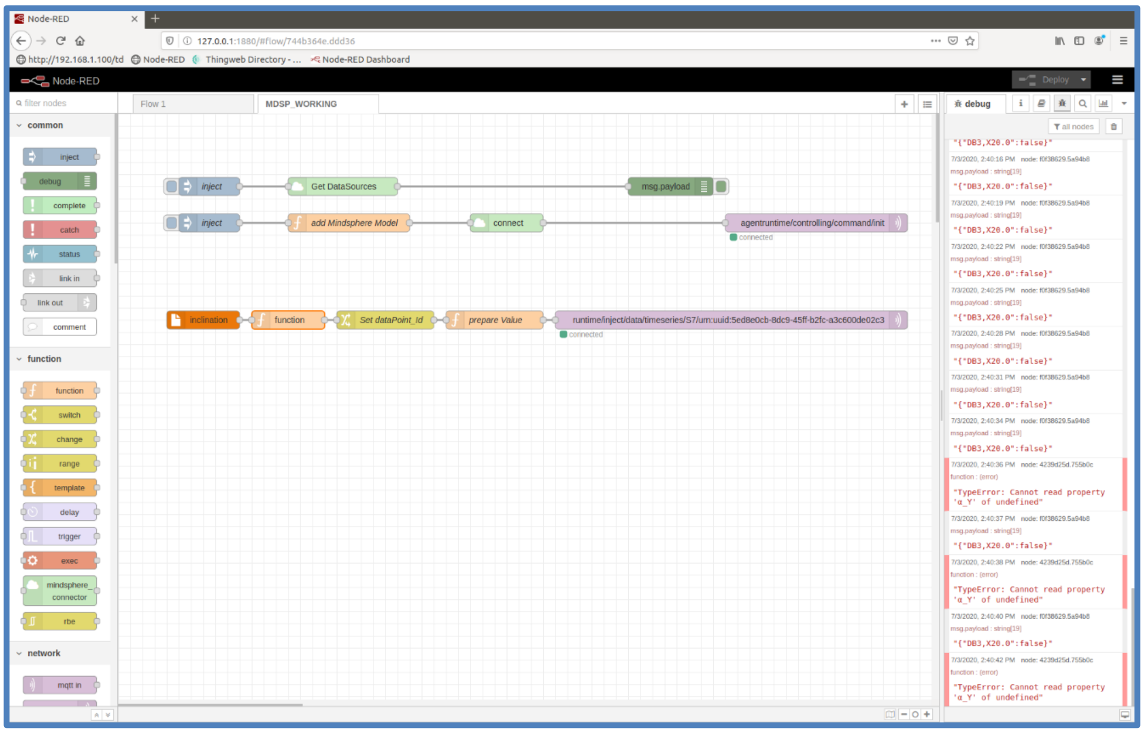


FIGURE 13: INTEGRATION OF IIOT GATEWAY WITH MINDSPHERE CLOUD

IIoT Gateway integrates field-level data and enables an easy development of Edge applications. But the data of inclinometer sensor in Sub-Use Case 3 is supposed to be used in MindSphere Cloud (MDSP Cloud App). Therefore, we have extended IIoT Gateway to meet this requirement. Open Edge Device Kit (OEDK)[5] has been used for this purpose. Figure 13 depicts how a custom device (on the Edge level) may communicate with MindSphere Cloud over OEDK. For example, the inclinometer data can be sent to OEDK via MQTT protocol and further from OEDK to MindSphere Cloud over HTTP (essentially over HTTPS protocol). This communication pattern works for any field device that is registered by IIoT Gateway.

[5] OEDK: https://developer.mindsphere.io/resources/openedge-devicekit/index.html

The data of an on-boarded field device can be forwarded to MindSphere Cloud via Semantic Edge Platform[6]. Semantic Edge Platform provides a convenient user interface for interacting with field devices, IIoT Gateway, and Cloud (via OEDK). Figure 14 shows a Node-RED flow for testing the Sub-Use Case 3, i.e., provision of inclinometer (field-level) data via gateway in Cloud (MDSP Cloud).



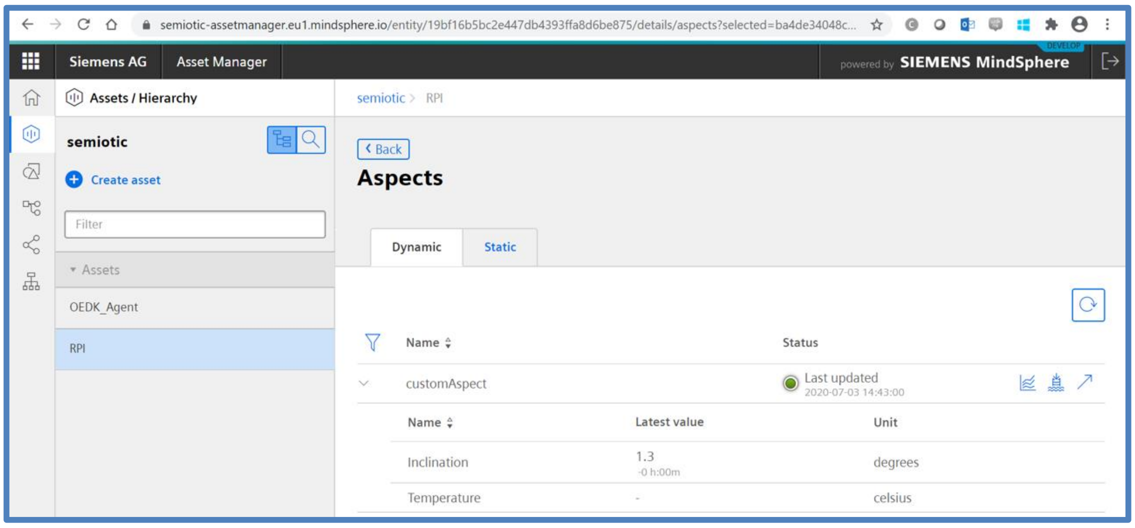
```
"{"DB3,X20.0":false}"
7/3/2020, 2:40:16 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:19 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:22 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:25 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:28 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:31 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:34 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:36 PM   node: 4239d25d.755b0c
function : (error)
"TypeError: Cannot read property
'α_Y' of undefined"
7/3/2020, 2:40:37 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:38 PM   node: 4239d25d.755b0c
function : (error)
"TypeError: Cannot read property
'α_Y' of undefined"
7/3/2020, 2:40:40 PM   node: f0f38629.5a94b8
msg.payload : string[19]
"{"DB3,X20.0":false}"
7/3/2020, 2:40:42 PM   node: 4239d25d.755b0c
function : (error)
"TypeError: Cannot read property
'α_Y' of undefined"
```

FIGURE 14: PROVISION OF INCLINOMETER DATA IN MINDSPHERE CLOUD OVER SEMANTIC EDGE PLATFORM

In general, Cloud has capability to integrate all kinds of data from the field. We can imagine thousands of inclinometers connected from a wind farm to MindSphere (together with other types of sensors, e.g., for audio, video etc.). This brings up the question how to interpret all this different kind of data in Cloud. MindSphere requires a user to create an information model (meta-data)[7] for an asset before the data of the asset can be sent to the Cloud.

IIoT Gateway has the goal to provide the semantic integration of field-level data. Thus, the semantic meta-data, which the gateway integrates and provides at the Edge level, is used in the Cloud integration too. We have extended IIoT Gateway to automatically create MindSphere information model based on Thing Description. Figure 15 shows an automatically created MindSphere information model based on Thing Description for inclinometer and a temperature sensor.

[6] Semantic Edge Platform is a SEMIoTICS extension of Node-RED, see deliverables D3.3 and D3.9.
[7] https://documentation.mindsphere.io/resources/html/asset-manager/en-US/113537583883.html


semiotic-assetmanager.eu1.mindsphere.io/entity/19bf16b5bc2e447db4393ffa8d6be875/details/aspects?selected=ba4de34048c...
Siemens AG
Asset Manager
powered by SIEMENS MindSphere
Assets / Hierarchy
semiotic
Create asset
Filter
Assets
OEDK_Agent
RPI
semiotic > RPI
Back
Aspects
Dynamic
Static

Name	Status
customAspect	Last updated 2020-07-03 14:43:00

Name	Latest value	Unit
Inclination	1.3 -0 h:00m	degrees
Temperature	-	celsius

FIGURE 15: IIOT GATEWAY AUTOMATICALLY CREATES MINDSPHERE INFORMATION MODEL BASED ON W3C THING DESCRIPTION

Finally, Figure 16 shows a successful test, i.e., field data of the inclinometer is available in Cloud. The test demonstrates the integration of a field device, IIoT Gateway, SDN network, and MindSphere Cloud. With this infrastructure in place a SEMIoTICS user can by simple interaction via Semantic Edge Platform to onboard a field device and access its data in MindSphere Cloud.

The last remaining step in Sub-Use Case 3 is to develop a Cloud app (MDSP App), which will consume the inclinometer data and trigger an alert when data exceeds a predefined threshold.

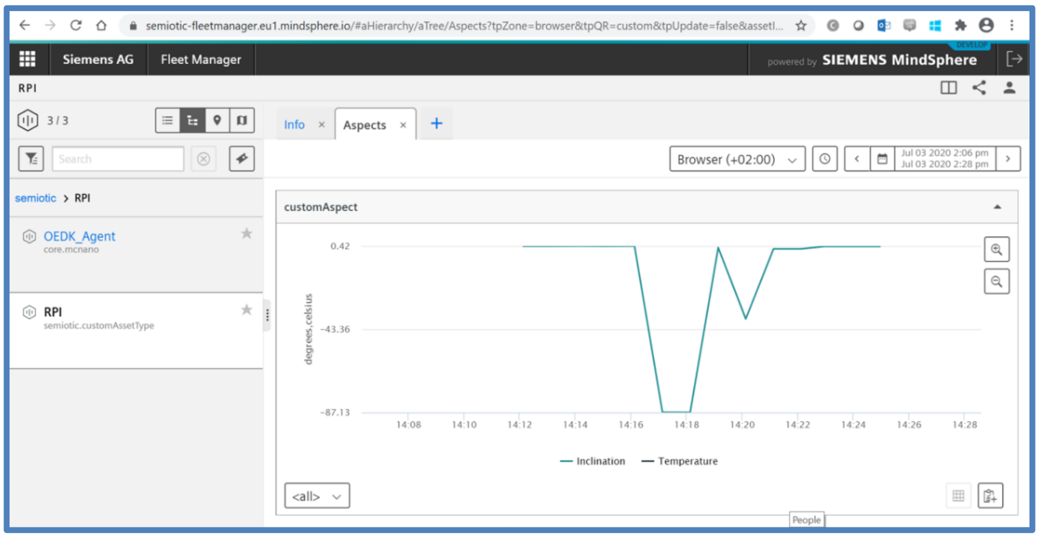


FIGURE 16: DATA OF INCLINOMETER AVAILABLE IN MINDSPHERE CLOUD

6 SUB USE CASE 4: FIELD DEVICES AVAILABILITY MONITORING

6.1 Scope and Objectives

Continuous availability of field devices, such as sensors and actuators, enrolled to the SEMIoTICS eco-system, is a prerequisite for applications that rely on them – whether instantiated in cloud or at the edge.

In this scenario, it's foreseen that the integrator or operator of the wind farm is responsible for the availability and integrity of the field devices enrolled, whereas the applications deployed, that rely on the interaction with the field devices to generate value, is defined by others, e.g., by example business intelligence expert, data scientists or and application that is instantiated on license from a market place.

This sub-use case is focusing on demonstrating interoperability and seamless integration with an 'availability monitoring application', defined and instantiated in the private cloud space, simulating a third-party integrator responsible for the availability monitoring of field devices enrolled to the SEMIoTICS eco-system.

All field devices used throughout UC1 demonstrator are being monitored and the availability status us displayed in an orchestrated user interface and hosted in the private cloud. Additionally, in order to guarantee that data in-transit is not modified in an unauthorized manner thus preserving information integrity, involved components are forced to use encryption when encryption is deemed unavailable.

Due to the fact that the field devices are registered to an MQTT Broker as per Figure 17, enforcing encryption is possible by interacting with the Broker.

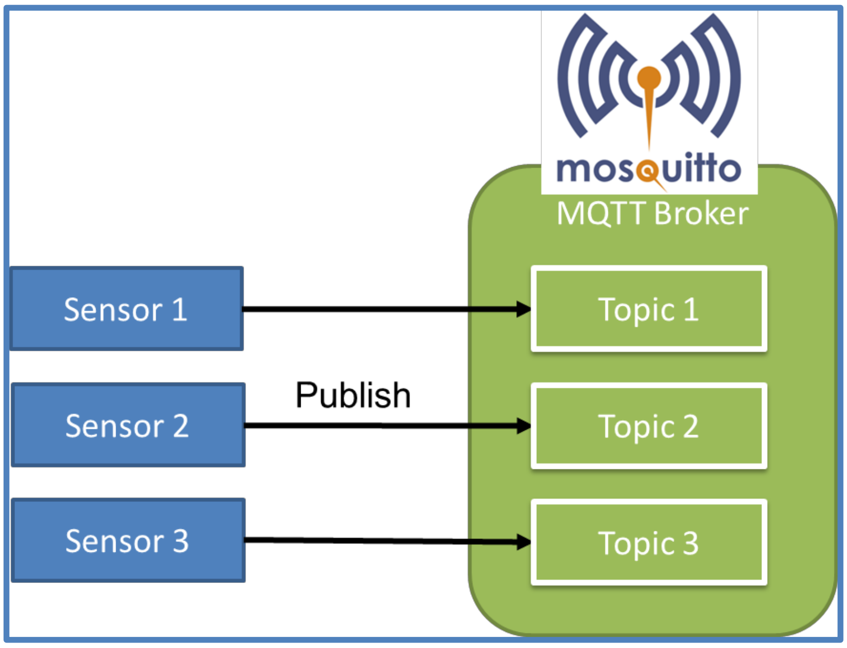


FIGURE 17: INTERACTION OF FIELD DEVICES IN SUB-USE CASE 3

Cycle 1 focuses on the integrity of the field devices enrolled and Cycle 2 will focus on the availability.

6.2 Interaction with SEMIoTICS Framework and Components

The components of the SEMIoTICS' framework that are used in this sub use case are highlighted in **Error! Reference source not found.**. In particular these components are Recipe Cooker, Pattern Orchestrator, Pattern Engine in the backend and GUI from the application layer and Pattern Engine at the field layer.

Recipe Cooker is used to inform the Pattern Orchestrator with a specific recipe which contains the sensors, the MQTT Broker and the requirement for encrypted communication between them.

Pattern Orchestrator is responsible for automated configuration, coordination, and management of different patterns and their deployment. Additionally it is used for proccessing the received recipe. The result of the said processing enables the Pattern Orchestrator to choose which Pattern Engine should receive the corresponding Drools facts. Additionally the processing includes the preparation of the REST endpoint required by the GUI to visualize the status of the pattern properties.

TABLE 3: SEMIOTICS COMPONENTS USED IN SUB USE CASE 4

Component	**SEMIoTICS component**	**Architecture reference**	**Description of use**
Recipe Cooker	✓	Application Layer	• Recipe Cooker framework based on Node Red is used to define and deploy the business logic to the Gateway and Backend Pattern Engines using Pattern Orchestrator as a proxy
Pattern Orchestrator	✓	Application Layer	• Pattern Orchestrator is used to translate the recipe (business logic request) into a set of patterns comprising Integrity requirements.
Backend Pattern Engine	✓	Application Layer	• Pattern Engine at the backend layer is used to toggle encryption on the MQTT broker in the public cloud based on triggered Integrity pattern by the Pattern Orchestrator.
Backend GUI	✓	Backend Layer	• Used to visualize the status of Pattern Orchestrator and in, particular, the adaptation of patterns applied to health monitoring connection.
Field Pattern Engine	✓	Field Layer	• The Pattern Engine at thefField layer is used for informing the Pattern Engine at the application layer about any changes on the pattern properties regarding its laye

Pattern Engine at the application layer is used for gathering all the facts from the Pattern Orchestrator regardless which layer are referred to. This enables the Pattern Engine at the application layer to have a complete view of the current status of the requirements. All the application-related pattern rules also are hosted at this engine and specifically in this subuse case, the encryption rules that are to interact with the application of the MQTT Broker.

The Pattern Engine at the Field layer is used for informing the Pattern Engine at the application layer about any changes on the pattern properties regarding its layer. Additionally, all the field related pattern rules also are hosted at this engine and specifically in this sub-use case, the availability rules that are to reason based on the field devices. Finally, the GUI is used for visualizing the status of the pattern properties with the use of the Pattern Orchestrator's corresponding REST endpoint.

Figure 18 shows the interactions of the components arranged in time sequence involved in the current scenario and the messages exchanged between the components needed to carry out the functionality of the sub use case in terms of integrity.

Recipe Cooker initiates the flow by sending the recipe to the Pattern Orchestrator. As already described, the recipe contains the field devices, the MQTT Broker and the requirement for encryption between the communication of the sensors and the MQTT Broker. Pattern Orchestrator after processing the recipe, translates the ingredients to Drools facts and sends them to the appropriate Pattern Engine. For every fact the Pattern Engine receives, tries to find a rule that can be triggered.

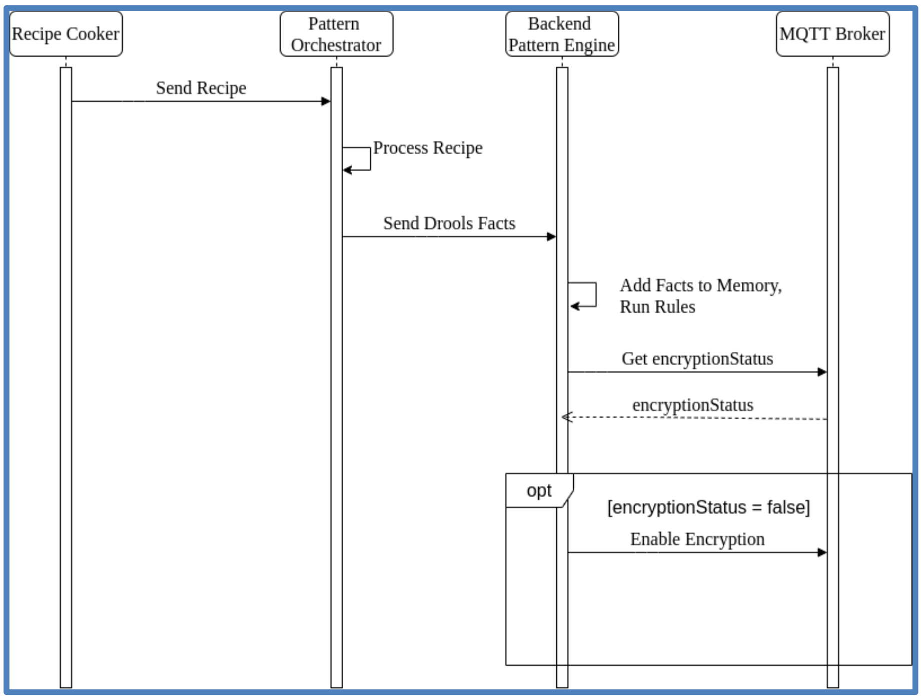
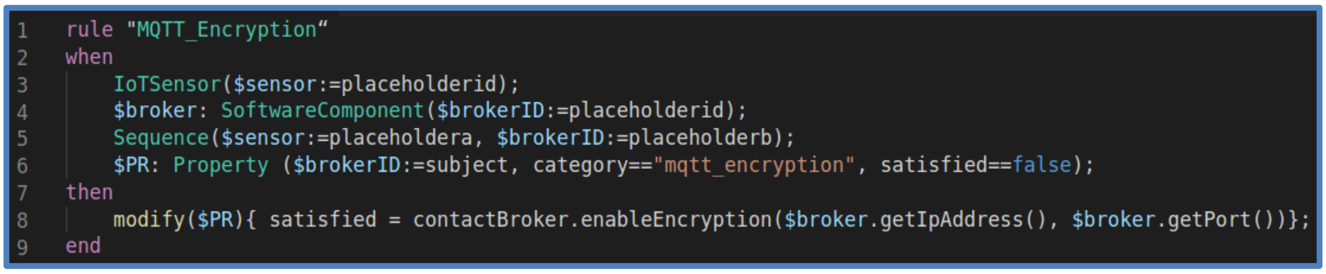


FIGURE 18: SEQUENCE DIAGRAM OF THE INTEGRITY ASPECT

As soon as all the necessary facts are received, the encryption rule is triggered as per Figure 19.



```
rule "MQTT_Encryption"
when
    IoTSensor($sensor:=placeholderid);
    $broker: SoftwareComponent($brokerID:=placeholderid);
    Sequence($sensor:=placeholdera, $brokerID:=placeholderb);
    $PR: Property ($brokerID:=subject, category=="mqtt_encryption", satisfied==false);
then
    modify($PR){ satisfied = contactBroker.enableEncryption($broker.getIpAddress(), $broker.getPort())};
end
```

FIGURE 19: MQTT ENCRYPTION RULE

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

1. An IoT sensor.
2. A software component that represents the MQTT Broker.
3. A sequence of the sensor and the MQTT Broker indicating their interaction.
4. The orchestration property that can be guaranteed through the application of the pattern, i.e., the encryption property in this case ($PR).

The **then** part triggers adaptation actions by contacting the Broker and forcing him to use encryption.

6.3 Setup Testbed and Integration

A testbed was set up to integrate and test the integrity aspect of the sub use case, emulating the sequence diagram shown in Figure 18. The technologies used to implement this include Java, Maven and Spring Boot for the development and creation of the Pattern Engine. Moreover, as detailed in deliverable D4.8, the Pattern Engine integrates the Drools Business Rules Management System (BRMS) solution to implement its reasoning capabilities. Finally, the Eclipse Mosquitto was extended with a restful service based also in Java, Maven and Spring Boot.

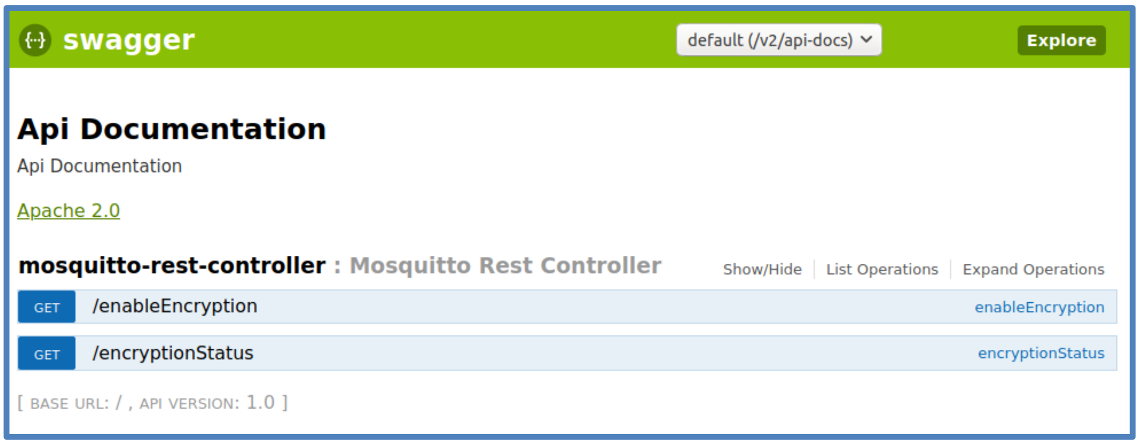
In FORTH's premises the Pattern Orchestrator, Backend Pattern Engine and MQTT Broker are deployed in VMs. An Intel NUC (Figure 20) is used with 32GB RAM, 500GB storage and a CPU i7-6770HQ 2.60GHz with 8 cores. Proxmox Virtual Environment is installed in the Intel NUC which hosts the aforementioned VMs:

- VM containing the Pattern Orchestrator.
    - Storage of 10 GB.
    - RAM of 4 GB.
    - 4 CPU cores.
    - Operative system is the Ubuntu 18.04.4 LTS.
- VM containing the Pattern Engine.
    - Storage of 10 GB.
    - RAM of 4 GB.
    - 4 CPU cores.
    - Operative system is the Ubuntu 18.04.4 LTS.
- VM containing the MQTT Broker.
    - Storage of 10 GB.
    - RAM of 2 GB.
    - 4 CPU cores.
    - Operative system is the Ubuntu 18.04.4 LTS

FIGURE 20: THE INTEL NUC

6.4 Validation

The aforementioned testbed allowed validating the functionality of the components by partially implementing the sequence diagram in Figure 18. Figure 21 shows the two REST endpoints implemented on the Broker. In the configuration of the MQTT Broker, all the related information for incoming connections such as port, protocol certificates etc., is grouped and the terminology used to describe that group is “listener”. The “enableEncryption” alters the configuration of the Broker in such a manner that it will include certificates for the available listener for incoming connections, whereas the “encryptionStatus” verifies whether such certificates are present in the configuration.



Api Documentation

Api Documentation

Apache 2.0

mosquitto-rest-controller : Mosquitto Rest Controller

Show/Hide | List Operations | Expand Operations

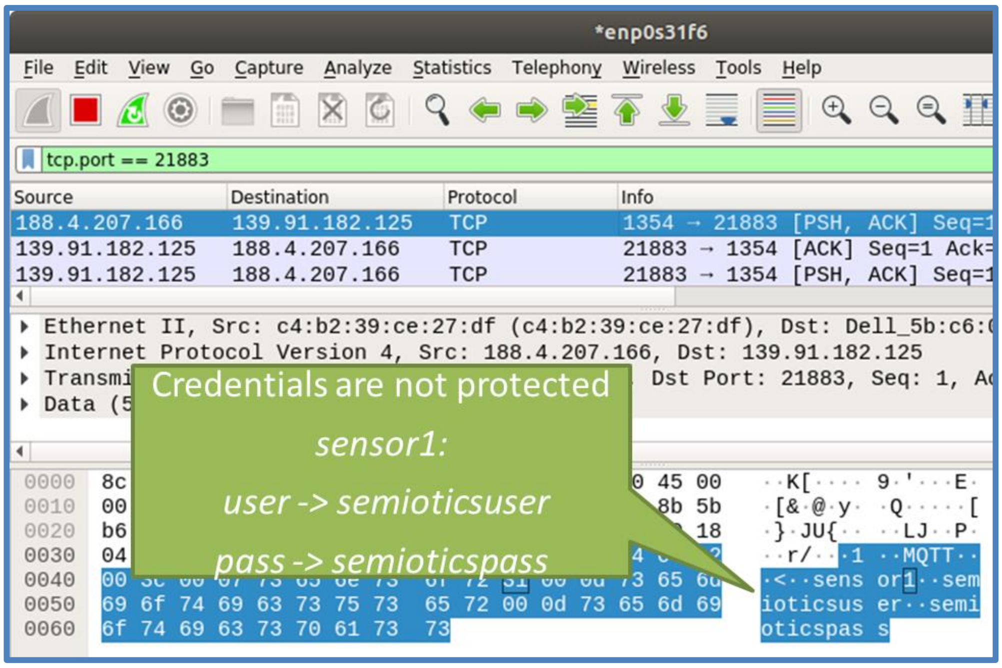
GET /enableEncryption enableEncryption

GET /encryptionStatus encryptionStatus

[BASE URL: / , API VERSION: 1.0]

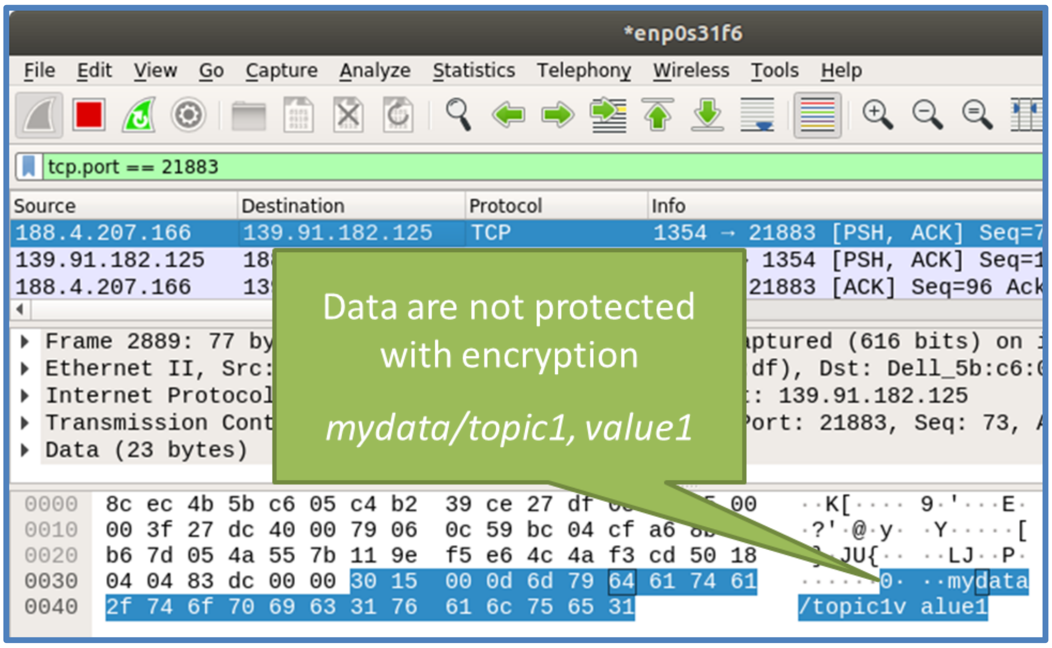
FIGURE 21: MQTT BROKER REST APIS

Figure 22 and Figure 23 shows the capture of packets from the Broker side before the encryption is enabled. At this point all the communication is unencrypted making visible the credentials exchange as well as any data exchange readable.


*enp0s31f6
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Credentials are not protected
sensor1:
user -> semioticsuser
pass -> semioticspass

```
tcp.port == 21883

Source          Destination     Protocol   Info
188.4.207.166   139.91.182.125  TCP        1354 → 21883 [PSH, ACK] Seq=1
139.91.182.125  188.4.207.166   TCP        21883 → 1354 [ACK] Seq=1 Ack=
139.91.182.125  188.4.207.166   TCP        21883 → 1354 [PSH, ACK] Seq=1

▸ Ethernet II, Src: c4:b2:39:ce:27:df (c4:b2:39:ce:27:df), Dst: Dell_5b:c6:(
▸ Internet Protocol Version 4, Src: 188.4.207.166, Dst: 139.91.182.125
▸ Transmi                                      Dst Port: 21883, Seq: 1, Ac
▸ Data (5

0000  8c                                       0 45 00    ··K[····  9·'···E·
0010  00                                        8b 5b    ·[&·@·y·  ·Q·····[
0020  b6                                           18    ·}·JU{··  ··LJ··P·
0030  04                                                 ··r/··.1  ··MQTT··
0040  00                                   73 65 6d    ·<··sens or1··sem
0050  69 6f 74 69 63 73 75 73  65 72 00 0d 73 65 6d 69    ioticsus er··semi
0060  6f 74 69 63 73 70 61 73  73                         oticspas s
```

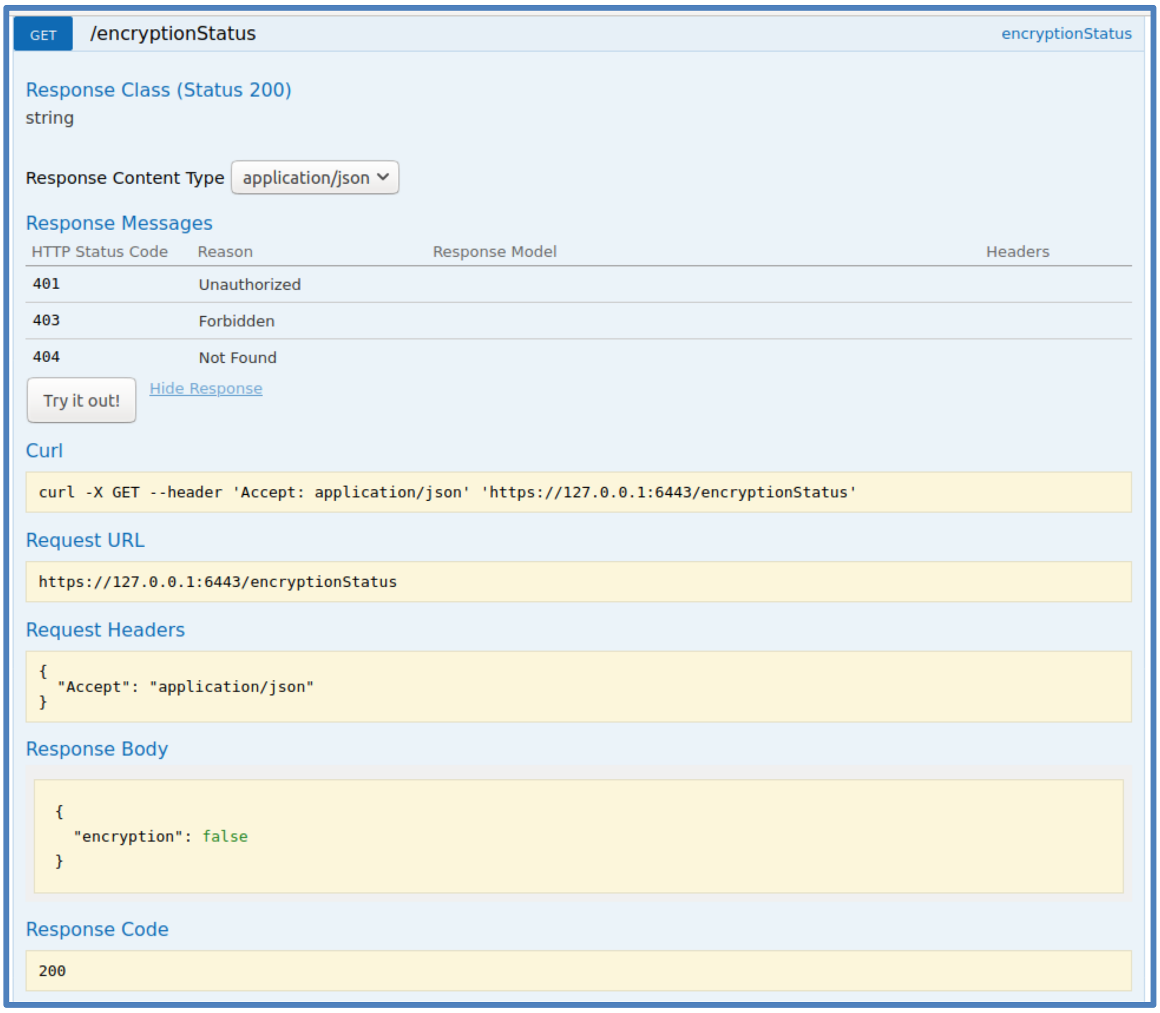
FIGURE 22: CREDENTIALS EXCHANGE


*enp0s31f6
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Data are not protected
with encryption
mydata/topic1, value1

```
tcp.port == 21883

Source          Destination     Protocol   Info
188.4.207.166   139.91.182.125  TCP        1354 → 21883 [PSH, ACK] Seq=7
139.91.182.125  18                         1354 [PSH, ACK] Seq=1
188.4.207.166   13                         21883 [ACK] Seq=96 Ack

▸ Frame 2889: 77 by                        ptured (616 bits) on i
▸ Ethernet II, Src:                        df), Dst: Dell_5b:c6:(
▸ Internet Protocol                        : 139.91.182.125
▸ Transmission Cont                        Port: 21883, Seq: 73, A
▸ Data (23 bytes)

0000  8c ec 4b 5b c6 05 c4 b2  39 ce 27 df          00   ··K[····  9·'···E·
0010  00 3f 27 dc 40 00 79 06  0c 59 bc 04 cf a6         ·?'·@·y·  ·Y·····[
0020  b6 7d 05 4a 55 7b 11 9e  f5 e6 4c 4a f3 cd 50 18      JU{··  ··LJ··P·
0030  04 04 83 dc 00 00 30 15  00 0d 6d 79 64 61 74 61   ······0·  ··mydata
0040  2f 74 6f 70 69 63 31 76  61 6c 75 65 31            /topic1v alue1
```

FIGURE 23: DATA EXCHANGE

Also, in Figure 24 we see the REST “encryptionStatus” that we created gives a response *encryption:false.*



HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

```
curl -X GET --header 'Accept: application/json' 'https://127.0.0.1:6443/encryptionStatus'
```

```
https://127.0.0.1:6443/encryptionStatus
```

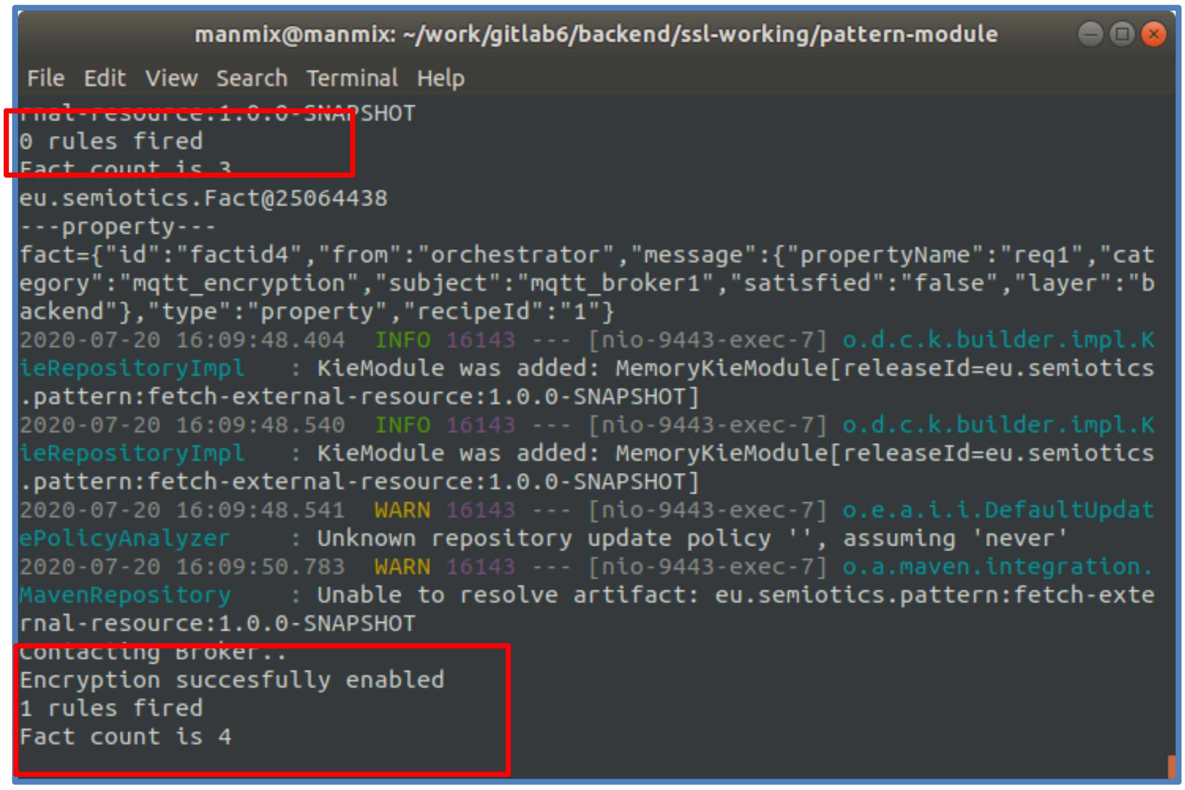
```
{
  "Accept": "application/json"
}
```

```
{
  "encryption": false
}
```

```
200
```

FIGURE 24: ENCRYPTION STATUS BEFORE PATTERN REASONING

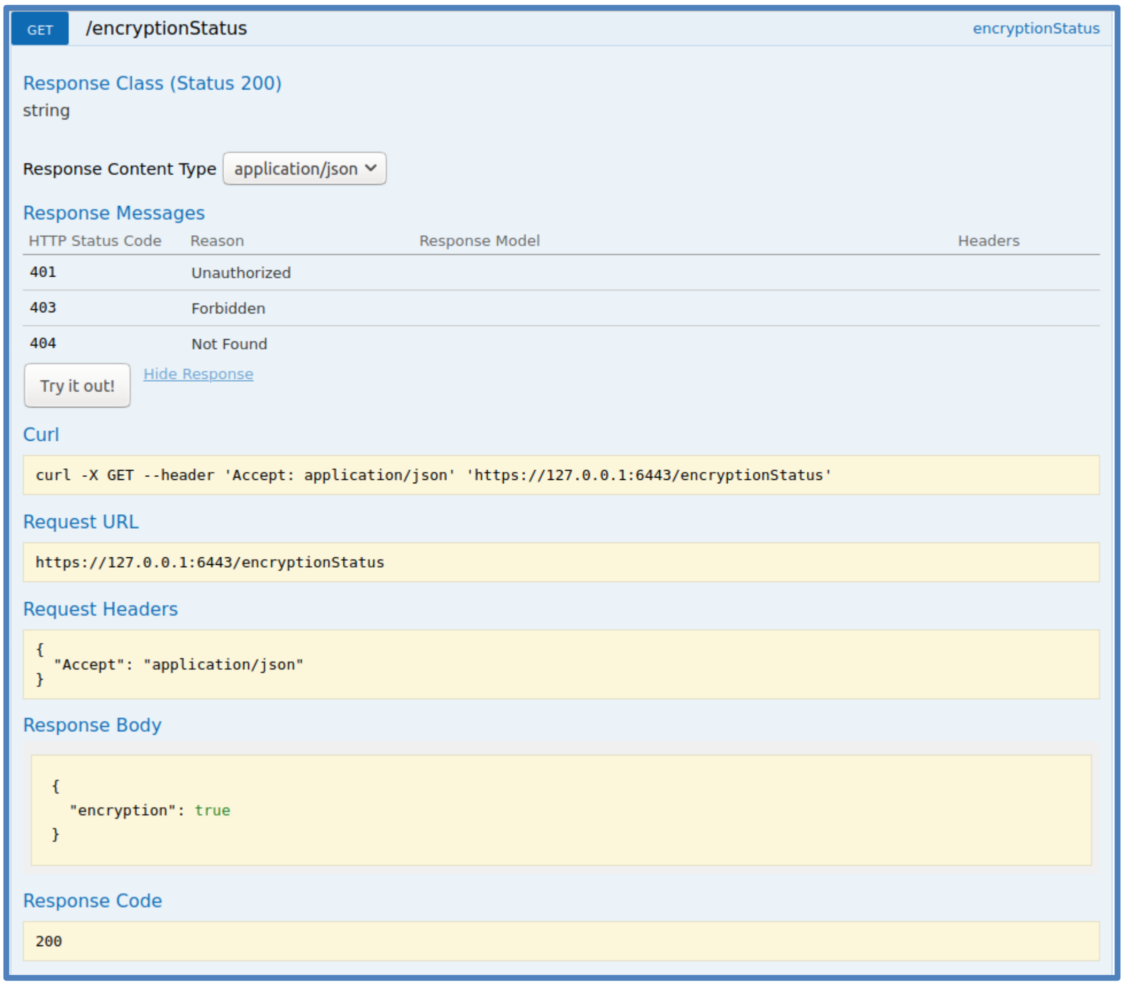
While the facts inserted to the Pattern Engine at the application layer, do no satisfy any rule, the output of the Pattern Engine is “0 rules fired” as we see in Figure 25. As soon the final fact required is inserted, one rule is triggered, and the Pattern Engine initiates communication with the MQTT Broker. After that, the application on the Broker side makes the appropriate changes to the configuration and encryption is enabled.


manmix@manmix: ~/work/gitlab6/backend/ssl-working/pattern-module
File Edit View Search Terminal Help

```
rnal-resource:1.0.0-SNAPSHOT
0 rules fired
Fact count is 3
eu.semiotics.Fact@25064438
---property---
fact={"id":"factid4","from":"orchestrator","message":{"propertyName":"req1","cat
egory":"mqtt_encryption","subject":"mqtt_broker1","satisfied":"false","layer":"b
ackend"},"type":"property","recipeId":"1"}
2020-07-20 16:09:48.404  INFO 16143 --- [nio-9443-exec-7] o.d.c.k.builder.impl.K
ieRepositoryImpl    : KieModule was added: MemoryKieModule[releaseId=eu.semiotics
.pattern:fetch-external-resource:1.0.0-SNAPSHOT]
2020-07-20 16:09:48.540  INFO 16143 --- [nio-9443-exec-7] o.d.c.k.builder.impl.K
ieRepositoryImpl    : KieModule was added: MemoryKieModule[releaseId=eu.semiotics
.pattern:fetch-external-resource:1.0.0-SNAPSHOT]
2020-07-20 16:09:48.541  WARN 16143 --- [nio-9443-exec-7] o.e.a.i.i.DefaultUpdat
ePolicyAnalyzer     : Unknown repository update policy '', assuming 'never'
2020-07-20 16:09:50.783  WARN 16143 --- [nio-9443-exec-7] o.a.maven.integration.
MavenRepository     : Unable to resolve artifact: eu.semiotics.pattern:fetch-exte
rnal-resource:1.0.0-SNAPSHOT
Contacting Broker..
Encryption succesfully enabled
1 rules fired
Fact count is 4
```

FIGURE 25: PATTERN ENGINE OUTPUT

When we call the REST */encryptionStatus* again (Figure 26), we get a different response this time indicating that the encryption is enabled on the MQTT Broker.


GET /encryptionStatus encryptionStatus

Response Class (Status 200)
string

Response Content Type application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

Curl

```
curl -X GET --header 'Accept: application/json' 'https://127.0.0.1:6443/encryptionStatus'
```

Request URL

```
https://127.0.0.1:6443/encryptionStatus
```

Request Headers

```
{
  "Accept": "application/json"
}
```

Response Body

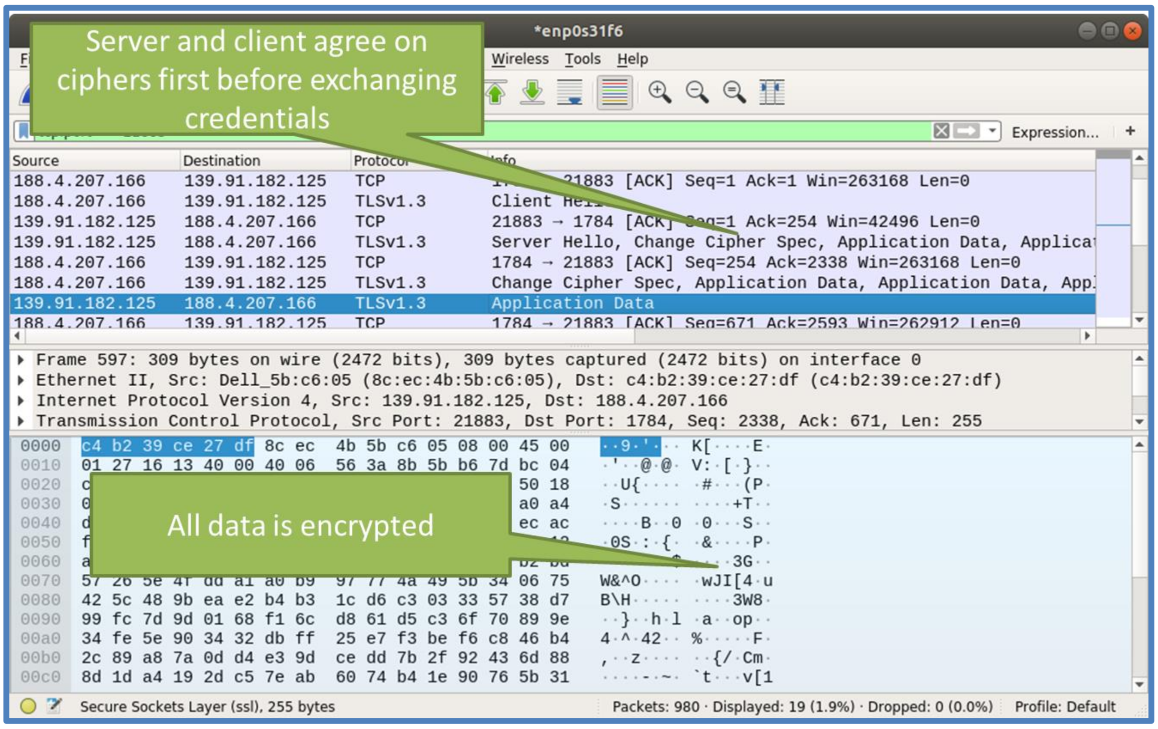
```
{
  "encryption": true
}
```

Response Code

```
200
```

FIGURE 26: ENCRYPTION STATUS AFTER PATTERN REASONING

Also, we verify the actual communication with the capture of packets from the Broker side after the encryption is enabled. At this point all the communication is encrypted making the credentials exchange as well as any data exchange unreadable as per Figure 27.


*enp0s31f6
Wireless Tools Help
Expression...
Server and client agree on ciphers first before exchanging credentials
All data is encrypted

```
Source           Destination      Protocol   Info
188.4.207.166    139.91.182.125   TCP        1… 21883 [ACK] Seq=1 Ack=1 Win=263168 Len=0
188.4.207.166    139.91.182.125   TLSv1.3    Client Hel…
139.91.182.125   188.4.207.166    TCP        21883 → 1784 [ACK] Seq=1 Ack=254 Win=42496 Len=0
139.91.182.125   188.4.207.166    TLSv1.3    Server Hello, Change Cipher Spec, Application Data, Applicat
188.4.207.166    139.91.182.125   TCP        1784 → 21883 [ACK] Seq=254 Ack=2338 Win=263168 Len=0
188.4.207.166    139.91.182.125   TLSv1.3    Change Cipher Spec, Application Data, Application Data, Appl
139.91.182.125   188.4.207.166    TLSv1.3    Application Data
188.4.207.166    139.91.182.125   TCP        1784 → 21883 [ACK] Seq=671 Ack=2593 Win=262912 Len=0

▸ Frame 597: 309 bytes on wire (2472 bits), 309 bytes captured (2472 bits) on interface 0
▸ Ethernet II, Src: Dell_5b:c6:05 (8c:ec:4b:5b:c6:05), Dst: c4:b2:39:ce:27:df (c4:b2:39:ce:27:df)
▸ Internet Protocol Version 4, Src: 139.91.182.125, Dst: 188.4.207.166
▸ Transmission Control Protocol, Src Port: 21883, Dst Port: 1784, Seq: 2338, Ack: 671, Len: 255

0000  c4 b2 39 ce 27 df 8c ec  4b 5b c6 05 08 00 45 00   ··9·'···K[····E·
0010  01 27 16 13 40 00 40 06  56 3a 8b 5b b6 7d bc 04   ·'··@·@·V:·[·}··
0020  c                                            50 18   ··U{·····#···(P·
0030  0                                            a0 a4   ·S··········+T··
0040  d                                            ec ac   ····B··0·0···S··
0050  f                                                    ·0S·:·{··&····P·
0060  a                                                    ············3G··
0070  57 26 5e 4f dd a1 a0 b9  97 77 4a 49 5b 34 06 75   W&^O·····wJI[4·u
0080  42 5c 48 9b ea e2 b4 b3  1c d6 c3 03 33 57 38 d7   B\H·········3W8·
0090  99 fc 7d 9d 01 68 f1 6c  d8 61 d5 c3 6f 70 89 9e   ··}··h·l·a··op··
00a0  34 fe 5e 90 34 32 db ff  25 e7 f3 be f6 c8 46 b4   4·^·42··%·····F·
00b0  2c 89 a8 7a 0d d4 e3 9d  ce dd 7b 2f 92 43 6d 88   ,··z······{/·Cm·
00c0  8d 1d a4 19 2d c5 7e ab  60 74 b4 1e 90 76 5b 31   ····-·~·`t···v[1

Secure Sockets Layer (ssl), 255 bytes    Packets: 980 · Displayed: 19 (1.9%) · Dropped: 0 (0.0%)    Profile: Default
```

FIGURE 27: DATA EXCHANGE AFTER REASONING

7 OVERALL KPIS AND REQUIREMENTS VALIDATION

The following table contains the brief summary of requirements and KPIs relevant for evaluation in Use Case 1. The four presented sub-use-cases, together with the remaining Use Cases 2 & 3 aim to address and validate each of the below KPIs.

Fulfillment of the project KPIs will, however, be documented in the final version of this deliverable, i.e., the D5.9 and is hence omitted at this time.

Further information on use case - specific requirements related on Use Case 1 are contained in D2.3. The planned methodology of KPI evaluation is contained in D5.1 and is omitted here for brevity.

TABLE 4: REQUIREMENTS PLACED ON USE CASE 1 AS PER D2.3

Req-ID	Description	Req. level	Comment / Fulfilment
R.UC1.1	Automatic establishment of networking setup MUST be performed to establish End-to-end connectivity between different stakeholders	MUST	This requirement is fulfilled and demonstrated by the interaction of Pattern Engine and Path Finding and reservation components in the SSC, in Sub-Use Cases 1 and 2.
R.UC1.2	Automatic establishment of computing environment MUST be performed in IIoT Gateway for the minimum operation of the IIoT devices through network controller based on SDN/NFV	MUST	Dynamic orchestration of computing resources is the focus of Use Case 2, SFC being the main highlighted benefit of the approach. Nevertheless, computing resources in UC1 are orchestrated by the Recipe Framework when deploying machine learning workloads on Nanobox devices in Sub-UC1/Sub-UC2.
R.UC1.3	There MUST be enabled the definition of network QoS on application-level and automated translation into SDN controller configurations.	MUST	This requirement is fulfilled and demonstrated by the interaction of Pattern Engine, Path Finding and Resource Manager components in the SSC, in Sub-Use Cases 1 and 2.
R.UC1.4	Network resource isolation MUST be performed for guaranteed Service properties – i.e. reliability, delay and bandwidth constraints.	MUST	This requirement is fulfilled and demonstrated by the Path Finding / Resource Manager modules of SSC, in Sub-Use Cases 1 and 2.
R.UC1.8	Semantic and robust bootstrapping/registration of IIoT sensors and actuators with IIoT gateway MUST be supported.	MUST	This requirement is fulfilled and demonstrated by the SEMIoTICS components of the IIoT Gateway and is demonstrated in Sub-Use Cases 1, 2, 3.
R.UC1.9	Semantic interaction between use-case specific application on IIoT Gateway and legacy turbine control system MUST be supported.	MUST	This requirement is fulfilled and demonstrated by the SEMIoTICS components of the IIoT Gateway and is demonstrated in Sub-Use Cases 1 and 2 - automated turbine rotor stopping (using a brownfield PLC) based on greenfield sensor inputs.
R.UC1.10	Local analytical capability of IIoT Gateway to run machine learning algorithms (e.g. specific to 2 specific sub-use cases)	MUST	This requirement is fulfilled and demonstrated by the video and audio analytics applications demonstrated in Sub-Use Cases 1 and 2.

R.UC1.11	Device composition and application creation SHALL be supported through template approach.	SHALL	This requirement is fulfilled and demonstrated by the Recipe Cooker as Recipe (Template) orchestration tool and is demonstrated in Sub-Use Cases 1, 2 and 4.
R.UC1.12	Standardized semantic models for semantic-based engineering and IIoT applications MUST be utilized.	MUST	This requirement is fulfilled and demonstrated by the integration of Thing descriptions (WoT), i.e., by semantic modelling in IIoT Gateway is demonstrated in Sub-Use Cases 1, 2 and 3.
R.UC1.13	Middleware functionality MUST be supported on IIoT gateway, to deal with termination of IIoT sensors, signal processing and termination of interfaces to legacy systems to provide prioritization and QoS for IIoT applications.	MUST	The requirement of sensor flow termination is fulfilled and demonstrated by the scenario of information splitting in the IIoT Gateway and is demonstrated in Sub-Use Cases 3 and 4. QoS and prioritization are demonstrated in Sub-Use Cases 1 and 2.
R.UC1.5	Fail-over and highly available network management SHALL be performed in the face of either controller or data-plane failures.	SHALL	This (SHALL) requirement is fulfilled, demonstrated and evaluated in a demonstrator and paper presented at ACM SIGCOMM 2019. Due to numerous other highlights of UC1, it has been excluded from demonstration.
R.UC1.6	Decisions made by unreliable, i.e. faulty or malicious SDN controllers, SHALL be identified and excluded.	SHALL	This (SHALL) requirement is fulfilled, demonstrated and evaluated in a demonstrator and paper presented at ACM SIGCOMM 2019. Due to numerous other highlights of UC1, it has been excluded from demonstration.
R.UC1.7	The operation of the SDN control SHALL be scalable to cater for a massive IoT device integration and large-scale request handling in the SDN controller(s) using a (near-) optimal IoT client – SDN controller assignment procedure.	SHALL	This (SHALL) requirement is fulfilled, demonstrated and evaluated in a demonstrator and papers presented at IEEE GLOBECOM 2019 and ACM SIGCOMM 2019. Due to numerous other highlights of UC1, it has been excluded from demonstration.

TABLE 5: KPIS TO BE VALIDATED / MEASURED BY USE CASE 1

KPI-ID	Name	Leader
KPI-2.1	Semantic descriptions for 6 types of smart objects	SAG/ENG/ST
KPI-2.2	Data type mapping and ontology alignment	FORTH/SAG/ENG
KPI-2.3	Semantic interoperability with 3 IoT platforms	FORTH/STS
KPI-3.1.1	Generating monitoring strategies in the 3 targeted IoT platforms	ENG
KPI-3.1.2	Fuse results from these monitors	ENG
KPI-3.1.3	Performing predictive monitoring with an average accuracy of 80%	FORTH/ENG
KPI-3.2	Delivery of a monitoring language	ENG/STS/FORTH
KPI-4.2	Delivery of mechanisms with adaptation time of 15ms	ENG/FORTH
KPI-4.6	Development of new security mechanisms/controls	UP/FORTH/STS/CTTC

KPI-6.1	Reduce manual interventions required for bootstrapping of smart object in each use case domain by at least 80%	SAG/CTTC
KPI-6.3	Delivery of 3 prototypes of IIoT/IoT applications	SAG/ENG/IQU

8 CONCLUSION

Main goal of UC1 is to demonstrate the coexistence of a highly integrated control system and an agile IIoT ecosystem based on the SEMIoTICS framework; which in return allows service providers to deploy new value-added services faster and provide effective access to data from both existing control system and the newly deployed edge sensor networks. The demonstrator of UC1 addresses some of the common challenges in extending the capabilities of an existing control system in a brownfield environment in the Wind Energy domain while highlighting the general benefits of the SEMIoTICS platform.

Deliverable D5.4, in particular, gives an initial insight into the status of development of the four sub-use cases for SEMIoTICS platform. It presents the current state of the integration of physical equipment comprising industrial programmable logic controllers, actuation, the greenfield and brownfield sensory and actuation devices, private and public clouds environments and hosted applications, the orchestration mechanisms introduced in SEMIoTICS, as well as the resources orchestrated, and applications dynamically instantiated and interconnected by our framework.

The subsequent deliverable D5.9 will address the final status of implementation and integration of the UC1 components and elements of the SEMIoTICS platform and will detail the methodology used to quantify the reaching of KPIs assigned to UC1, as per D5.1.