



SEMIoTICS

Deliverable D5.5

Demonstration and validation of SARA-Health (Cycle 1)

Deliverable release date	30/07/2020
Authors	<ol style="list-style-type: none">1. Domenico Presenza, Philip Wright (ENG),2. Emmanouil Michalodimitrakis, Nikolaos Petroulakis (FORTH),3. Felix Klement, Korbinian Spielvogel, Henrich C. Pöhls (UP),4. Łukasz Ciechomski, Michal Rubaj (BS),5. Jordi Serra (CTTC)
Responsible person	Domenico Presenza (ENG)
Reviewed by	Felix Klement (UP), Nikolaos Petroulakis (FORTH), Vivek Kulkarni (SAG)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, Ermin Sakic, Mirko Falchetto, Domenico Presenza, Verikoukis Christos) PCC Members (Vivek Kulkarni, Nikolaos Petroulakis, Verikoukis Christos, 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

1. Introduction	6
1.1. Methodology and document structure	7
2. Use Case Description	8
2.1. Generic storylines	9
2.1.1. Fall Management Assistive Task	10
2.1.2. Remote Gait Analysis Assistive Task	10
2.2. Challenges and objectives	11
2.3. SEMIoTICS sub use cases	12
3. Sub Use Case 1: Moving Intelligence to the Edge	14
3.1. Scope and objectives	14
3.2. Interaction with SEMIoTICS framework	14
3.3. Setup testbed and integration	17
4. Sub Use Case 2: Automated, trustworthy healthcare connectivity	20
4.1. Scope and objectives	20
4.2. Interaction with SEMIoTICS framework	22
4.2.1. Interaction with NFV	23
4.2.2. NFV Orchestrator and VNF manager	23
4.2.3. Virtual Infrastructure Management	24
4.2.4. Pattern Orchestrator	24
4.2.5. Pattern Engine	24
4.2.6. Thing Directory	24
4.2.7. Backend Semantic Validator	24
4.3. Setup testbed and integration	24
4.3.1. NFV testbed	24
4.3.3. Service Function Chaining Patterns Validation	27
4.3.5. Backend Semantic Validator and Thing Directory Validation	33
5. Sub Use Case 3: Enforcement and Monitoring of GDPR-compliant access Control to confidential and sensitive data	36
5.1. Scope and objectives	36
5.2. Interaction with SEMIoTICS framework	37
5.2.1. Authorization Policy Management	37
5.2.2. Authentication Management	38
5.3. Setup testbed and integration	39
5.3.1. Setup steps	40
5.3.2. Enforcing access control	40
5.3.3. Dynamic Policy Adaptation	41
5.3.4. Monitoring compliance	41

6. Sub Use Case 4: Controlling bulbs and robots	43
6.1. Scope and objectives	43
6.2. Interaction with SEMIoTICS framework	44
6.3. Setup testbed and integration	45
6.3.1. Virtualization of The Robotic Rollator	45
6.3.2. Virtualization of ZigBEE Light bulbs	47
7. Overall KPIs and Requirements	49
7.1. Related KPIs	49
7.2. Related framework requirements	49
8. Conclusion	54
9. References	55
Appendix A - UC2 SEMIoTICS Requirements	56
Appendix B - Network Infrastructure Validation	61

Acronym	Definition
AE	Auto Encoder
AI	Artificial Intelligence
AWS	Amazon Web Services
BPTT	Back Propagation Through Time
CEP	Complex Event Processor
CGNN	Causal Generative Neural Network
DoA	Description of Action
eBPF	extended Berkley Packet Filter
ETC	Event Triggered Causality
FPR	False Positive Rate
GAN	Generative Adversarial Network
GUI	Graphical User Interface
IIoT	Industrial Internet of Things
IoT	Internet of Things
KPI	Key Performance Indicator
LLDP	Link Layer Discovery Protocol
LOOCV	leave-out-one-device cross validation
LSTM	Long-Short-Term-Memory network
MC	Monitoring Component
MCU	Micro Controller Unit
ML	Machine Learning
NMS	Network Management System
NSGI	Next Generation Service Interface
ODL	OpenDayLight
OF	OpenFlow
OMP	Orthogonal Matching Pursuit algorithm
PaaS	Platform-as-a-Service
PO	Pattern Orchestrator

QoS	Quality of Service
RC	Recipe Cooker
RNN	Recurrent Neural Network
SDK	Software Development Kit
SDN	Software Defined Network
SNMP	Simple Network Management Protocol
SNS	Simple Notification Service
SPDI	Security, Privacy, Dependability, Interoperability
SR	Sparse Representation
SVDD	Support Vector Data Description
TPR	True Positive Rate
VM	Virtual Machine
WoT	Web of Things

1. INTRODUCTION

This deliverable provides the initial validation of the SEMIoTICS technologies in the context of the development of a solution in the domain of Ambient assisted living.

The aims of SARA, this is the name of the solution, is to integrate a number of assistive technologies to address the diverse problems that older people face when living independently with the support of their community.

In particular, the SARA solution is being developed around the concept of Assistive Task (AT). An Assistive Task is a coherent set of SARA functionalities addressing a specific risk related to the independent living of elderly people affected by Mild Cognitive Impairment (MIC) or mild Alzheimer's disease (AD).

The Assistive Tasks being developed in SARA address risks related to four areas: physical decline, cognitive decline, medication management, and psychological needs.

We selected two assistive tasks to validate how and to what extent the SEMIoTICS technologies can facilitate the implementation of a system like SARA. The selected ATs are the Fall Management and the Remote Gait Analysis. The *Falls Management Assistive Task* aims to reduce the consequences of fall events of elderly living at home. The *Gait Analysis Assistive Task* aims to support remote gait quality assessment by means of specific exercises executed by the patient using a Robotic Rollator and under the remote supervision of a Doctor.

The implementation of each SARA Assistive Task requires the integration of multiple mobile (mobotic) and static (domotic, or home automation) robotic elements working in tandem with wearable health monitors, personal smart devices (phones and tablets) and various (proprietary and third party) computational and information services. All these devices that are not specifically designed to work together.

This integration effort requires to cope with a number of challenges arising in each of the main areas addressed by the SEMIoTICS project:

- **security, privacy, dependability and safety:** all stored personal data must be securely encrypted with protected access only by authorized and authenticated individuals and/or system processes; all communications with personal data as content must be securely encrypted with authenticated endpoints; the treatment of patient health data must comply with healthcare domain standards and national and international regulations concerning security and privacy; all actuator control signals (whether originating 'on device' or 'over the network') must be protected from malicious interference;
- **semantic interoperability** the different devices consume and produce different kinds and quantities of data, in diverse formats, and support different modes of network communication (Bluetooth, Wi-Fi, wired ethernet, can-bus, etc.). Moreover: for reliability, critical data is collated from multiple redundant sources (several components monitoring the same features) and shared through multiple redundant channels;
- **embedded intelligence** to ensure timely responses of robotic elements, AI intense computations are executed concurrently throughout the system, with operations dynamically dispatched to the nearest available computing resources (including edge devices);
- **trustworthy network management:** system components and technologies must be configured on a reliable network to avoid service interruptions; high-priority communications, e.g. incident alerts, must be transmitted by reliable means (e.g. multiple identical transmissions over multiple channels) and given network priority; network configurations are not static: mobile devices move around and may temporarily go out of range of Wi-Fi; mains powered devices (e.g. Wi-Fi base stations, domotic servers) are lost during power outages, while battery powered devices (e.g. robots and phones) will need periodic recharging; and wearable health monitors can be removed (e.g. when taking a bath).

In this document the validation of the benefits brought by the SEMIoTICS technologies to the development and operation of SARA is presented by means of four sub use cases. Each sub use case focuses a subset of the challenges in one the areas presented and represents a different view on the very same system. The four sub use cases are:

- **Enforcement and Monitoring of GDPR-compliant access Control to confidential and sensitive data** presenting how SEMIoTICS can be used to address security, privacy, dependability and safety challenges.
- **Moving Intelligence to the Edge** demonstrating how SEMIoTICS Local Embedded Intelligence can be deployed on the SARA Robotic Rollator and utilized in the implementation of the SARA Remote Gait Analysis Assistive Task.
- **Automated, trustworthy healthcare connectivity** show how NFV along with the pattern related components, offer a flexible solution to deal with the heterogeneous traffic flows of SARA. Also, how they reduce the manual intervention in the networking configuration and management, thus yielding a network management automation solution.
- **Controlling bulbs and robots** demonstrates how the SEMIoTICS Semantic Interoperability technologies can help the developers of the SARA solution to interface heterogeneous devices ranging from light bulbs to semi-autonomous robots.

1.1. Methodology and document structure

The overall development of SARA in SEMIoTICS proceeded in three phases:

- the initial phase took place during the first year of the project and concerned the establishment of both business requirements and requirements of SARA towards the SEMIoTICS framework.
- the second phase, during the second year of the project, saw the setup of the initial infrastructure, the selection of the ATs to be used for the evaluation, and the development of a first version of the Fall Management AT without the use of SEMIoTICS technologies (still under development at that time). This development served to establish a baseline for the subsequent developments based on the SEMIoTICS technologies. This first implementation was shown during the first formal review of the project.
- the third phase started with the third year of the project and is concerning both the re-engineering of Fall Management AT and the development of the new Remote Gait Analysis AT. Both these developments are based on the adoption of the SEMIoTICS technologies as illustrated by means of four sub use cases presented in the present deliverable.

This deliverable D5.5 accounts for the intermediate results achieved during the third phase of the development and is organised as follows: section 2 presents the Assistive Tasks selected for the evaluation of the SEMIoTICS technologies (section 2.1), the challenges subsumed by the implementation of selected ATs (section 2.2), and introduces the aspects addressed by each of the four sub use cases (section 2.3). Section 3, 4, 5, and 6 goes in the details of each sub use case.

This deliverable D5.5 is the result of the joint effort by all contributing partners during the development of the SARA prototype. At the beginning of the third phase bi-weekly meetings were run to keep aligned each partner about the work of the others, share results of experimentations and take design decisions. Later on, the meetings were run weekly.

2. USE CASE DESCRIPTION

As described in Deliverable 2.2, the use case 2 is from the domain of Ambient Assisted Living and focuses on the development of the SARA distributed application running on the CLOE-IoT platform.

The global trend for increased life expectancy is expected to be accompanied by a rapid rise in the number of people affected by Mild Cognitive Impairment (MIC) or mild Alzheimer's disease (AD). Aging and dementia lead to a significant chronic loss in physical/cognitive capacities and a decline in functional ability, resulting in increased dependency and need for caregiving, with substantial medical, social, psychological, and financial burdens placed on patients, their families, and their communities. Indeed, the kind of continuous, holistic and integrated care required by people with dementia cannot be satisfied by the current 'specialist' model, and the gap between the number of people in need of care and those able to provide it is expected to grow. People also prefer to live in their own homes for as long as possible rather than being institutionalized in sheltered/nursery homes when age-related problems appear. Leading, for example, to propose "Aging in place" - allowing older adults to age in the least restrictive environment of their choice - as a more desirable and viable care model.

Home automation and robotics systems can help relieve the caregiving burden and make 'ageing in place' a reality by providing physical and cognitive assistance to the elderly in the common tasks of daily home living, helping them maintain their autonomy longer and delay the need for social/health service interventions. Advances in artificial intelligence (AI) and IoT technologies are rapidly converging to make such solutions both technically and economically feasible. In this light, Engineering is extending its current commercial AREAS® suite - an Enterprise Resource Planning solution for the Health sector - with SARA (Socially Assistive Robotic Solution for Ambient assisted living), a robotic system aimed at providing automated health/incident monitoring and home assistance to elderly people with mild cognitive impairments.

As its name indicates, SARA is a Socially Assistive Robotics (SAR) solution. SAR solutions reside at the intersection of Assistive Robotics (AR) - typically robots that provide physical assistance (e.g. rehabilitation robots, wheelchair robots, mobility aids, manipulator arms) - and Socially Interactive Robotics (SIR) - whose main task is some form of social interaction with humans (e.g. engaging in conversation, understanding gestures, role taking, task collaboration). SAR aims to assist humans through social interaction - an ambitious goal given that social interaction is arguably the most complex of all human behaviours. A socially competent robot, for example, should be able to: communicate with high-level dialog; use natural cues (gaze, gesture, etc.); express and/or perceive emotions; learn/recognise models of other agents; exhibit distinctive personality and character; establish and maintain social relationships; (possibly) learn/develop social competences.

Achieving human level competence in these areas is well beyond even the best of contemporary AI technologies, so with SARA our goal is more pragmatic. We aim to develop an expandable platform that initially exhibits only trivial social capabilities (well within current technological limits), while yet providing a solid foundation for future expansion to more sophisticated human-like interactions.

The design of SARA integrates a number of assistive technologies to address the diverse problems that older people face when living independently with the support of their community. More specifically, it comprises multiple mobile (mobotic) and static (domotic, or home automation) robotic elements working in tandem with wearable health monitors, personal smart devices (phones and tablets) and various (proprietary and third party) computational and information services, to continuously monitor and assist elderly subjects (and their caregivers) in their normal daily activities in and around the home. The technical components of SARA are as follows:

- backend **AREAS Cloud Services** (ACS): serving primarily as a repository of medical records and an offline store for monitored bio-medical data;
- a **Body Area Network** (BAN) comprising wearable health monitors (e.g. for heart-rate, blood pressure, breathing rate, stress levels, balance and fall detection) worn by the elderly subject and communicating via Bluetooth to their personal device (phone or tablet);

- a **Robotic Rollator** (RR): a smart wheeled walking frame providing physical support for standing, sitting and walking, and capable of monitoring its user's posture and gait, and of (limited) autonomous navigation;
- a **Robotic Assistant** (RA): a mobile humanoid robot - specifically Softbank's humanoid "Pepper" robot¹ - capable of autonomous navigation, face recognition, scripted dialog and adopting life-like human poses (e.g. to perform arm/hand gestures);
- various **AI Services**: computational resources providing machine learning (ML), mapping, route planning and reasoning capabilities (among others);
- a **Smart Home**: a home incorporating intelligent objects, utilizing wired or wireless networks and having the capacity to analyse patterns of activities to manage appliances in anticipation of human needs or to provide adaptive cues for human occupants.

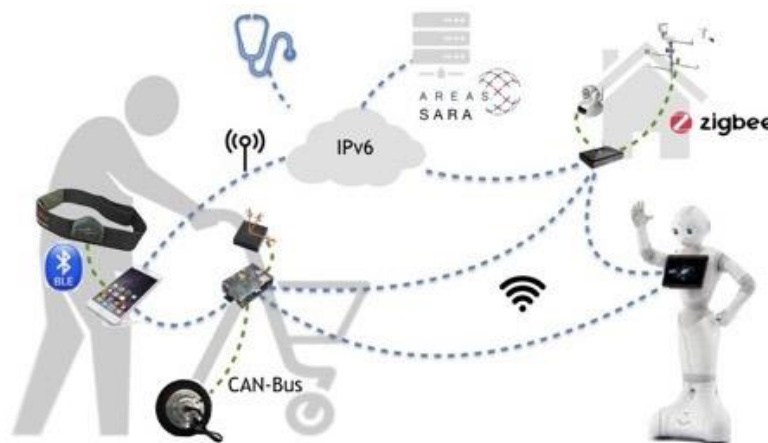


FIGURE 1: SARA MAIN TECHNICAL COMPONENTS AND COMMUNICATION SYSTEM

With the exception of the ACS and RR (both developed in house by Engineering) these components are all off-the-shelf devices that are not specifically designed to work together. The different devices consume and produce different kinds and quantities of data, in diverse formats, and support different modes of network communication (Bluetooth, Wi-Fi, wired ethernet, can-bus, etc.). Moreover: for reliability, critical data is collated from multiple redundant sources (several components monitoring the same features) and shared through multiple redundant channels; to ensure timely responses of robotic elements, AI intense computations are executed concurrently throughout the system, with operations dynamically dispatched to the nearest available computing resources (including edge devices); network configurations are not static: mobile devices move around and may temporarily go out of range of Wi-Fi; mains powered devices (e.g. Wi-Fi base stations, domotic servers) are lost during power outages, while battery powered devices (e.g. robots and phones) will need periodic recharging; and wearable health monitors (BAN devices) can be removed (e.g. when taking a bath).

The SARA solution is built on top of on the CloE-IoT platform. The CloE - IoT platform² aims to simplify the integration of highly distributed, complex and robust IoT solutions exploiting computational resources both in the cloud and at the edge.

2.1. Generic storylines

¹ <https://www.softbankrobotics.com/emea/en/robots/pepper>

² Starting from January 2020 the CloE-IoT platform is part of the Digital Enabler ecosystem. <https://www.eng.it/en/our-platforms-solutions/digital-enabler>

The SEMIoTICS technologies will be validated in the context of a re-engineering process of the SARA prototype being developed by ENG. This re-engineering process will concern both the business logic layer (i.e. the assistive tasks provided by SARA to its end users) and the middleware layer (i.e. the CloE-IoT services).

At the business logic layer two assistive tasks are going to be considered: Fall Management and Gait Analysis.

2.1.1. FALL MANAGEMENT ASSISTIVE TASK

The Fall Management Assistive Task is one of the major tasks of the SARA Use case and described in deliverable D2.2 - "SEMIOTICS usage scenarios and requirements".

The *Falls Management Assistive Task* aims to reduce the consequences of fall events of elderly living at home. In particular:

1. SARA continuously monitors the patient by means of the devices (e.g. smart phone/smart watch) within her Body Area Network;
2. whenever the fall detection software hosted by the BAN detects a possible fall is detected it raises (both via cellular and Internet connectivity) an early warning towards a remote assistance Call Centre. This warning message carries also information about the location/area where the event was detected. The fall detection software identifies the location of the event using iBeacon technology.
3. using the local Wi-Fi network, the warning message is also forwarded to the Robotic Assistant (i.e. a Pepper robot in the current prototype).
4. as soon as the Robotic Assistant receives a fall event message it starts to navigate toward the location indicated in the message. The Robotic Assistant relies on the navigation and localization cloud services of the SARA solution to autonomously reach the location of the event.
5. once reached the location of the event the Robotic Assistant may interact with the Smart Environment to better prepare the environment for the telepresence session initiated by the Call Centre operator later on (see next steps in this scenario). As an example, the Robotic Assistant might decide to turn on the lights or open the shades if it's on board sensors report a low level of enlightenment in the room.
6. the call distributor software operating at the Call Centre dispatches the message to one of the operators logged into the system. The call distributor software forwards to an operator only those messages originating from a certified device. Moreover, the call distributor also filters out duplicated messages related to the same event.
7. the Operator Console web application retrieves from the Electronic Patient Healthcare record service of the AREAS® platform the relevant clinical data of the patient indicated in the warning message. The Operator Console web application presents, in a coherent way, both the information carried by the warning message and that retrieved from the Electronic Patient Health Record.
8. once all the relevant information has been presented, the Call Centre operator can assess the actual situation at patient's home starting a telepresence session with the Robotic Assistant using the cameras on board of it.
9. if the position reached autonomously by the Robotic Assistant is not satisfactory for an assessment of the scene of the event, the Call Centre operator can also initiate a teleoperation session to move the Robotic Assistant to a better point of view.

2.1.2. REMOTE GAIT ANALYSIS ASSISTIVE TASK

The Remote Gait Analysis Assistive Task is one of the other important tasks of the SARA Use case and described in deliverable D2.2 - "SEMIOTICS usage scenarios and requirements".

The Gait Analysis Assistive Task aims to support remote gait quality assessment by means of specific exercises executed by the patient using the Robotic Rollator and under the remote supervision of a Doctor. In particular:

1. The Doctor schedules the timeframe for the execution of the assessment session using the SARA Agenda service. As an example, a Doctor could schedule a periodic assessment session to be executed during the third week of every month.
2. At the start the period schedule by the Doctor (e.g. the third week of the month) the SARA Agenda service notifies a remainder to the Robotic Assistant.
3. once received the notification the Robotic Assistant autonomously navigates to the location where the patient is. For the navigation and localization, the Robotic Assistant relies on the navigation and localization cloud services of the SARA solution (see also previous Fall Management scenario).
4. once reached the patient the Robotic Assistant initiates a dialog with the patient to engage her in the execution of the gait assessment session. For the execution of the dialog the Robotic Assistant relies on the Dialog Manager cloud service of SARA and other third-party AI services (e.g. speech to text).
5. if the patient accepts to do the gait assessment session a notification is sent to the Doctor. If the patient refuses the session is rescheduled.
6. upon the reception of the acceptance by the patient the Doctor starts a remote supervision session using the Gait Analysis web application from the SARA solution.
7. at the beginning of the session the Doctor starts a telepresence session using the camera on board of the Robotic Assistant. Using this session, the Doctor can do a visual assessment of the gait of the patient. The doctor can use the Gait Analysis web application also to teleoperate the Robotic Assistant and move it to a better point of view if the current one does not allow a clear view of the scene. The Doctor may also decide to keep a video recording of the execution of the exercises for subsequent analysis.
8. the patient executes the exercises as requested by the Doctor. During the execution of the Gait Encoder software hosted by the controller of the Robotic Rollator segments and encode the time series generated by various sensors embedded within the Robotic Rollator (e.g. IMU, Lidar, handlebars).
9. as the encoded time series are produced by Gait Encoder, they are sent to the Gait Clustering cloud service part of the SARA solution.
10. the Gait Clustering service clusters the time series received by the Gait Encoder and presents the resulting clusters to the Doctor using the Gait Analysis web application. The Doctor uses this clusters to make her assessment about the condition of the patient. The Gait Clustering service also attaches the result of the clustering to the Health record of the patient.

2.2. Challenges and objectives

The smart end-to-end IoT interoperability, connectivity and security technologies developed by the SEMIoTICS project can play a critical role in addressing the key challenges entailed in the development of SARA.

In particular, already the implementation of the two assistive tasks described in the section 2.1 requires to address a number of challenges arising in the area of:

- **End-to-end security, privacy, dependability and safety:** the development and operation of SARA need means to accurately assess the impact of varying system configurations on the security, privacy, dependability and safety properties of the system: to verifiably ascertain, for example, if using a particular device for a particular purpose, or a particular configuration of devices, exposes the system to an unacceptable level of risk.
- **IoT Semantics Interoperability:** SARA system relies on a distributed network of sensors and actuators, employing diverse wired and wireless communications technologies. The nodes in the network are highly heterogeneous, ranging from basic inertial measurement units (IMU - e.g. for balance and fall detection), presence sensors (e.g. cameras equipped with motion detection and face/object recognition), to sophisticated robotic devices with their own internal sensor/actuator systems and onboard computers. The sensor/actuator network also communicates with backend cloud services to access and store events

(e.g. 'fall detected', 'telepresence session initiated') and monitored health data (e.g. detailed time-sequenced measurements from BAN devices). Accordingly, the system must deal with a wide range of device semantics (different kinds of devices with different functions), data formats (syntactic representations), measurement unit conventions (for sensor readings), and communications protocols (e.g. Wi-Fi, ZigBee, Bluetooth). Various key aspects of SARA functionality, moreover, require that data from multiple sources is collated, aggregated and/or analysed in a coherent collective fashion. The reliable detection of 'fall incidents', for example, may involve the continuous comparative evaluation of data from wearable IMU devices, RR handle-mounted pressure sensors and RA video cameras (among others).

- **Embedded Intelligence and local analytics:** SARA is an application of Socially Assistive Robotics (SAR) and is thus fundamentally an artificial intelligence system. SARA requires the continuous operation of a range of perceptual systems. This means multiple AI/ML processes working concurrently to extract different features from the sensory input in a timely fashion - i.e. without any significant network latency. Thus, there is a strong need for an IoT infrastructure that supports computationally intensive, distributed AI processing at the edge of the IoT network
- **Network Management:** SARA is a distributed application where different computing entities continuously communicate and synchronize their activities to collectively generate the overall functionality of the system. Many of these distributed computations, due to their near real-time nature, are subject to strong time constraints – which can only be met by a network infrastructure capable of guaranteeing low latency, reliable communications between all participating components. The SARA robotic nodes, moreover, have (quasi)autonomous behaviours driven to a large extent by uncertain and unpredictable events in the environment – with a corresponding uncertainty and unpredictability in the generated computational load. Such variability prohibits the advance determination of optimal configurations of networking resources. Indeed, dealing with such dynamic run-time variability entails adopting correspondingly dynamic and flexible mechanisms for network management.

2.3. SEMIoTICS sub use cases

In this document the validation of the benefits brought by the SEMIoTICS technologies to the development and operation of SARA is presented by means of four sub use cases. Each sub use case focuses a subset of the challenges in one the areas presented in section 2.2 and represents a different view on the very same system. The four use cases are:

- **Moving Intelligence to the Edge** demonstrates how the time series clustering algorithm part of the SEMIoTICS Local Embedded Intelligence component developed in the context of Task 4.3, can be deployed on the SARA Robotic Rollator and utilized in the implementation of the SARA Remote Gait Analysis Assistive Task.
- **Automated, trustworthy healthcare connectivity** shows how NFV along with the pattern related components, offer a flexible solution to deal with the heterogeneous traffic flows of SARA. Also, how they reduce the manual intervention in the networking configuration and management, thus yielding a network management automation solution.
- **Enforcement and Monitoring of GDPR-compliant Access Control to Confidential and Sensitive Data** shows how the interaction of the SEMIoTICS Security & Privacy components is able to address, realize and monitor the enforcement of all security- and privacy-relevant properties of user-related data. Especially in this UC the data relates to the medical conditions and thus is privacy sensitive data. SEMIoTICS enforces the GDPR-compliant controlled access to that data by enforcing and monitoring access control through policies as well as through encryption to enable GDPR-compliant data handling also during transfer and storage even beyond the system's boundary.
- **Controlling bulbs and robots** demonstrates how the SEMIoTICS Semantic Interoperability technologies developed in the context of Task 3.3 can help the developers of the SARA solution to

interface heterogeneous devices ranging from light bulbs to semi-autonomous robots while keeping the application logic independent from the underlying hardware and transport protocols as much as possible.

The reader can find in Appendix A the mapping between the sub use cases introduced by this section and the requirements posed by the SARA-Health use case towards the SEMIoTICS framework. This set of requirements was initially presented in deliverable D2.3 - "Requirements specification of SEMIoTICS framework".

3. SUB USE CASE 1: MOVING INTELLIGENCE TO THE EDGE

3.1. Scope and objectives

This sub use case aims to demonstrate how the SEMIoTICS Local Embedded Intelligence technologies developed in the context of Task 4.3 can help to reduce the communication bandwidth required for transmitting raw data from filed devices to cloud services.

In particular it demonstrates how the time series clustering algorithm part of the SEMIoTICS Local Embedded Intelligence component developed in the context of Task 4.3, can be deployed on the SARA Robotic Rollator and utilized in the implementation of the SARA Remote Gait Analysis Assistive Task (see Section 2.1.2).

This sub use case contributes to evaluate the fulfilment by the SEMIoTICS platform of functional requirement R.UC2.1

Table 3 SEMIoTICS Requirement evaluated by Sub use case 1

SEMIOTICS Req.-ID	SARA Req-ID	Functional	Description	Req. level
R.UC2.1	R2.13	Yes	The SEMIoTICS platform SHOULD support time- and safety-critical requirements by allowing SARA application logic to be deployed on resource-constrained edge gateways (e.g. smartphones, vehicles, mobile robots). SEMIoTICS platform functionalities SHOULD be locally available even in case of failure of communication with the SEMIoTICS cloud nodes.	SHOULD

3.2. Interaction with SEMIoTICS framework

Figure 2 shows the Embedded Intelligence Components involved in the implementation of the SARA solution.

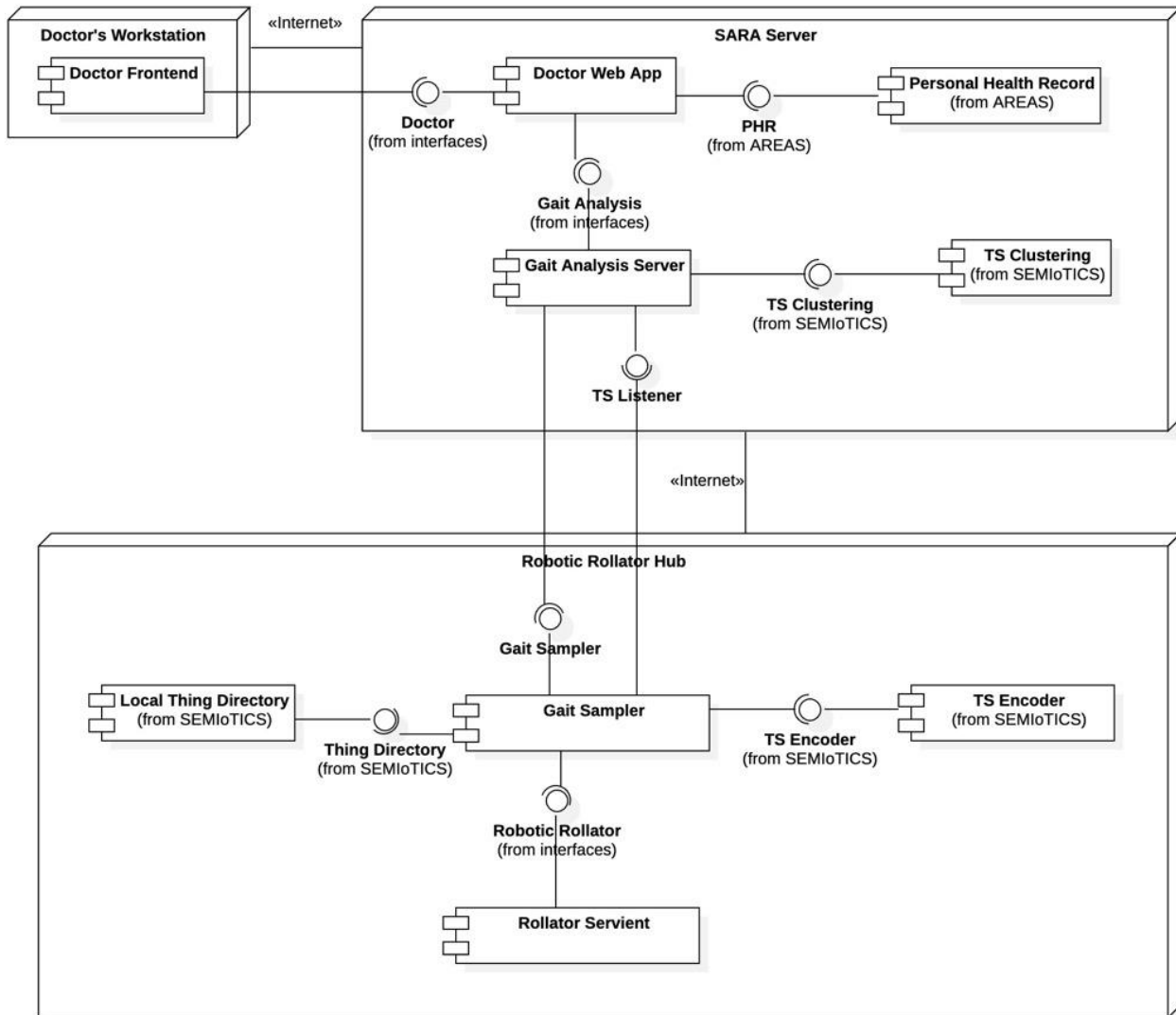


FIGURE 2: EMBEDDED INTELLIGENCE COMPONENTS IN SARA

More specifically:

- the **Local Thing Directory** allows the Gait Sampler deployed on board of the Robotic Rollator to retrieve the Rollator Servient. The Rollator Servient is the SARA component that virtualizes the Robotic Rollator as a “Thing” in the sense defined by the WoT standard. The Rollator Servient allows the Gait Analysis App to observe the distances of the patients’ legs from the rollator’s frame (Figure 3).

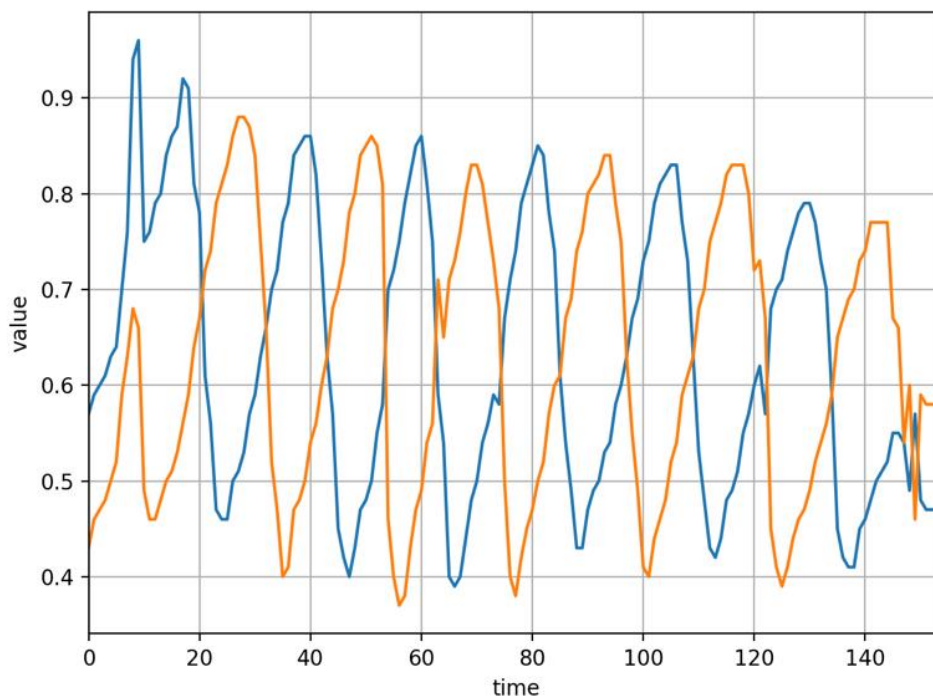


FIGURE 3: AN EXAMPLE OF TIME SERIES RECEIVED BY THE GAIT SAMPLER

- the **TS Encoder** allows the Gait Sampler to encode each gait time series (i.e. a time series representing the distance of a patient's leg from the rollator during a complete gait cycle) (Figure 4).

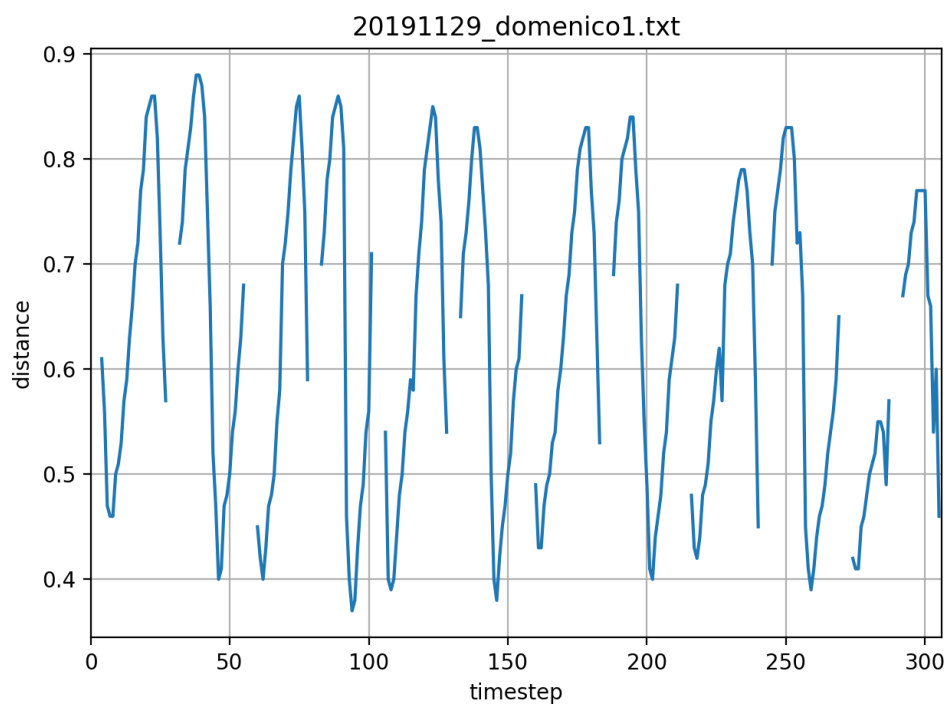


FIGURE 4: EXAMPLE OF TIME SERIES SENT TO THE TS ENCODER

- the **TS Classifier** allows Gait Analysis Server to compute the probability that a gait cycle belongs to one of the possible predefined cluster (Figure 5).

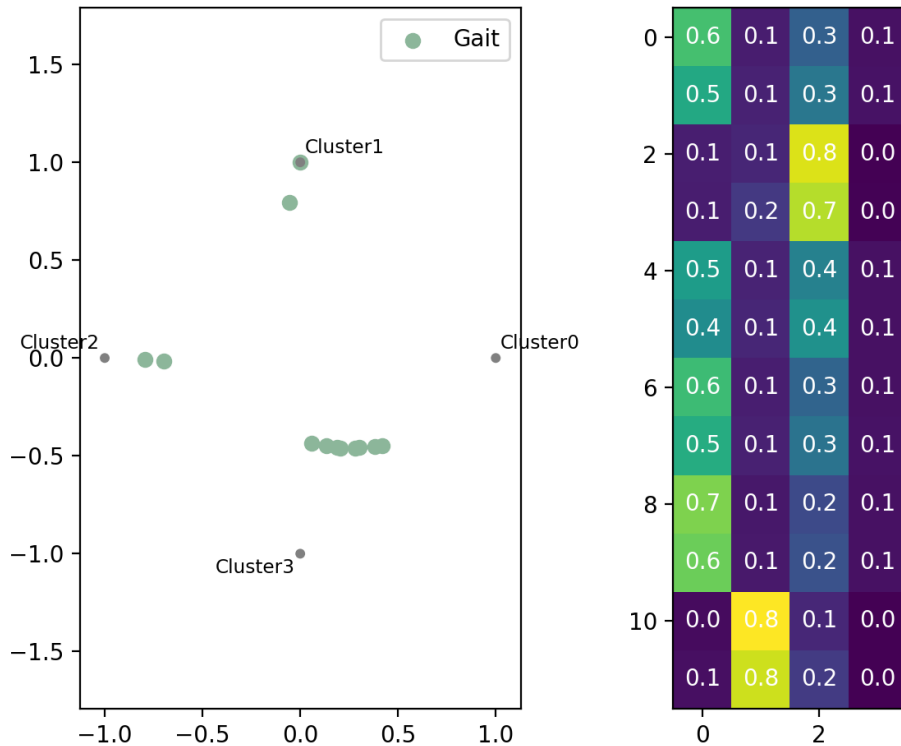


FIGURE 5: AN EXAMPLE OF ASSIGNMENT PRODUCED BY THE TS CLASSIFIER

3.3. Setup testbed and integration

The tested of sub use case 1 utilizes three hosts (see also Figure 2):

- Doctor's workstation** implemented by a standard PC running Window 10 OS
- SARA server** implemented by a Virtual Machine running Linux OS and hosted by the FIWARE Lab infrastructure³.
- Robotic Rollator Hub** implemented by a Raspberry Pi 3 running Raspbian OS. This hosts is the same host used by sub use case 4 (see section 6).

The components presented in figure 2 are implemented as follows:

- Doctor Frontend** is realized by a Chrome Web browser executing the JavaScript served by the Doctor Web App hosted by the SARA server. This script implements the GUI of the Doctor Web Application.
- Doctor Web App** is implemented by a web application written in Angular 8 and Typescript running on a Tomcat web server.
- Personal Health Record** is realized by a database from AREAS@.

³ <https://www.fiware.org/developers/fiware-lab/>

- **TS Clustering** is realized by the Keras library and the SEMIoTICS time series clustering model described in deliverable D4.10 and trained using data collected by means of the Robotic Rollator before the Covid-19 lockdown (i.e. same data used by sub use case 4 described in section 6).
- **Gait Analysis Server** is realized by a REST service implemented in Python 3 and the Flask framework.
- **TS Encoder** is realized by the Keras library and the SEMIoTICS time series encoder model described in deliverable D4.10 and trained using data collected by means of the Robotic Rollator before the Covid-19 lockdown (i.e. same data used by sub use case 4 described in section 6).
- **WoT_RR_SARA** is the WoT servient virtualizing the Robotic Rollator as a “Thing” in the sense of the WoT standard and is described more in details in section 6).
- **Gait Sampler** is realized by a REST service implemented in Python 3 and the Flask framework.

The Figure 3 shows the interactions occurring between components during the execution of Gait Analysis Task.

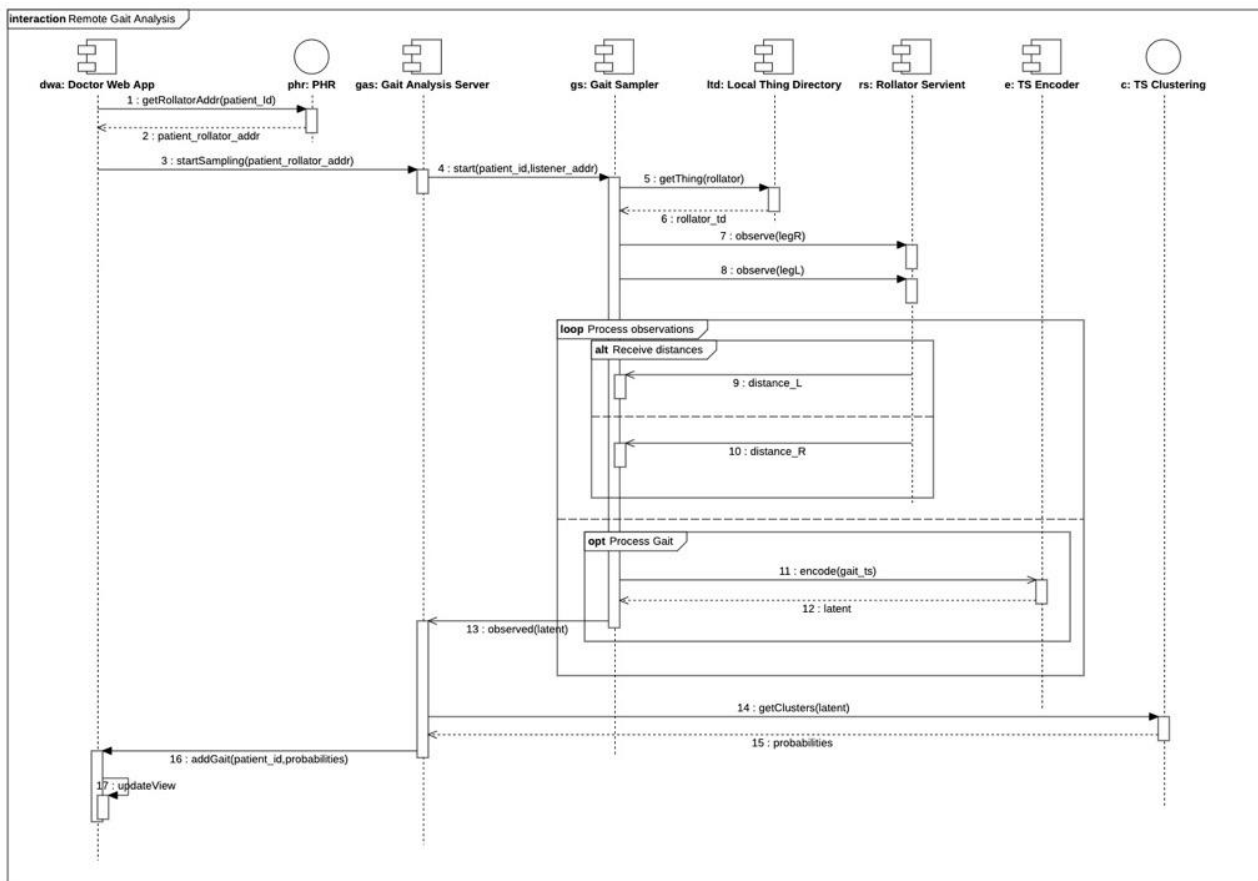


FIGURE 3: INTERACTIONS BETWEEN COMPONENTS

In particular:

1. The Doctor Web Application requests the address of the patient’s rollator to the Patient Health Record (PHR).
2. The PHR return the rollator’s address to the Doctor Web Application.
3. The Doctor Web Application requests to the Gait Analysis Server to start the sampling of the patient’s gaits.

4. The Gait Analysis Server requests to the Gait Sampler to start sampling process.
5. The Gait Sampler requests to the Local Thing Directory (a SEMIoTICS component) the descriptor of the Rollator
6. The Local Thing Directory returns the descriptor of the Rollator to the Gait Sampler component
7. the Gait Sampler requests to the Rollator Servient to notify periodically the distance of the right leg
8. the Gait Sampler requests to the Rollator Servient to notify periodically the distance of the left leg. Once both observations (left leg and right leg) are in place the Gait Sampler repeats continuously the steps 9-13 described below:
9. the Gait Sampler receives (asynchronously) from the Rollator Servient the distance of the left leg, or
10. the Gait Sampler receives (asynchronously) from the Rollator Servient the distance of the right leg,
11. the Gait Sampler, whenever detects that a gait cycle is complete, requests to the TS Encoder (a SEMIoTICS component) the encoding of the two times series accounting for the distance of the left and right legs from the rollator.
12. the Gait Sampler receives from the TS Encoder the encoded versions (latent representations) of the two time series
13. the Gait Sampler forwards to the Gait Analysis Server the time series received by the TS Encoder
14. the Gait Analysis Server requests to the TS Clustering component (a SEMIoTICS component) to assign to each time series the probability to be in one of the predefined gait clusters.

4. SUB USE CASE 2: AUTOMATED, TRUSTWORTHY HEALTHCARE CONNECTIVITY

4.1. Scope and objectives

The SARA system generates traffic with heterogeneous features, stemming from the diversity of IoT devices embedded in the different SARA components such as the robotic rollator, the robot or the Body Area Network (BAN). Namely, the traffic will have different requirements in terms of priority:

- Low priority. For instance, traffic from appliances within the smart home during ordinary activities.
- High priority. Traffic generated by those Assistive Tasks like the Fall Management AT described in section 2.1.1, which are related to potential health problems of the elderly people under control, e.g. abnormal change in his breathing, in his heart rate or in his gait.

Moreover, the SARA use case involves data related to a health application. Thereby, it involves several trust levels that have to be considered for their processing at the networking layer:

- Low trust. For instance, the traffic generated by the mobile phone.
- Medium trust. For instance, the traffic from sensors at the smart home.
- High trust. For instance, the traffic generated by the robot assistant.

The heterogeneity in the traffic requires a flexible network that is capable to process the traffic accordingly. Moreover, the heterogeneity of the traffic leads to an increased network complexity, as it can lead to more configurations to adapt the network to the traffic features. In order to cope with the traffic heterogeneity and with the increased network complexity we adopt herein the Network Function Virtualization (NFV) framework. NFV provides a flexible network, as it relies on the virtualization of the computing, storage and networking resources yielding the so-called Network Function Virtualization Infrastructure (NFVI). Thereby, the network functions can be deployed on top of this NFVI in the form of the so-called Virtual Network Functions (VNF). This provides a flexible networking approach as the virtual resources can be allocated easily according to the network services 'needs. Moreover, the VNFs can be chained to form the so-called Service Function Chains (SFC). For instance, one can form an SFC consisting of a Firewall (FW) and a Load Balancer (LB). Another example is an SFC consisting of a FW, a Deep Packet Inspection (DPI) and Intrusion Detection System (IDS). Thereby, NFV provides flexibility and the proper means to deal with traffic heterogeneity. Last, but not least, the configuration and lifecycle management of the VNFs along with the virtualized infrastructure management, are automated thanks to the NFV Management and Orchestration (NFV MANO) sub block within the NFV framework.

Considering the above, there is significant motivation to leverage the flexibility provided by SFC to define specific service chains for each type of traffic. By applying the previous described procedure of chain instantiation, the SARA solution can support traffic forwarding through specific service functions. That includes the traffic forwarding for the different type of traffic exchanged between the different actors as following:

- *Chain 1 – Mobile Phone:* Firewall -> DPI -> IDS -> Output.
- *Chain 2 – Robotic Rollator:* Firewall -> IDS -> Load Balancer -> Output.
- *Chain 3 – Smart Home:* Firewall -> IDS -> Output.
- *Chain 4 – Robotic Assistant:* Firewall -> Load Balancer -> Output.
- *Chain 5 - Malicious:* Firewall -> Honeypot.

The above scenario sketches a complex environment, requiring support for integration of heterogeneous devices and communication protocols, high degrees of interoperability, and support for distributed services and applications (each with its own set of intrinsic requirements), while guaranteeing the safety of the patient and

the security and privacy of her patient data. This use case is visualized in Figure 6, which depicts the various types of devices, their interactions, and the involved communication technologies.

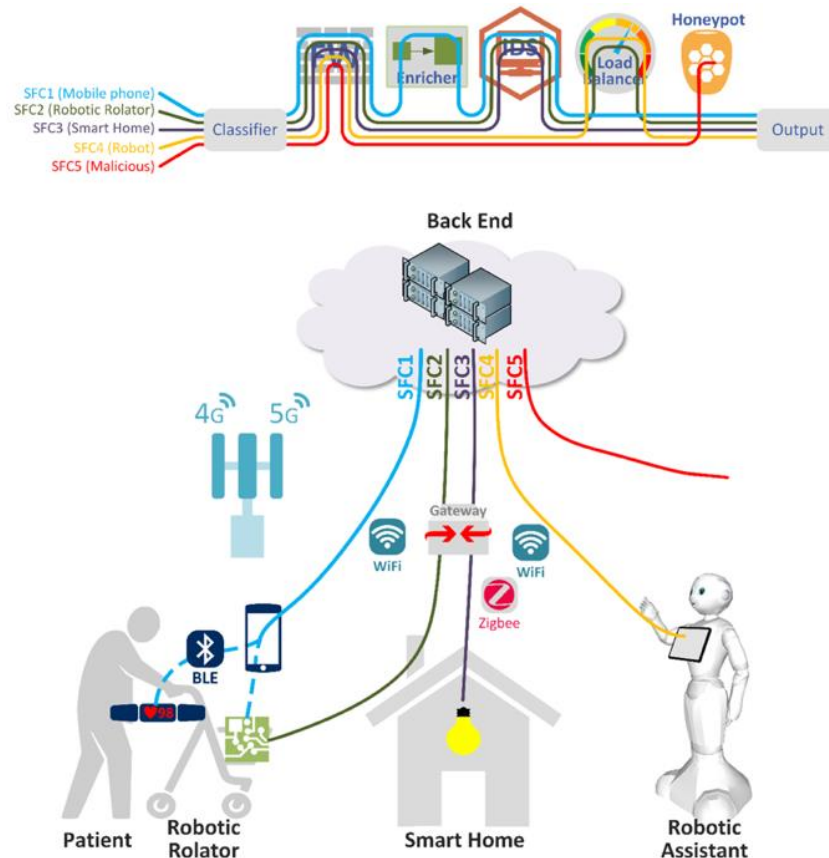


FIGURE 6 SARA-HEALTH SCENARIO AND TRAFFIC CLASSIFICATION

The instantiation of a sequence of functions can constitute a service chain. Similar to the insertion of service functions in the SFC manager through the exposed service function REST interface, service chains can be inserted. In the list of the service functions, the firewall, the DPI, the IDS and the Load Balancer have been defined as the most crucial ones to enable the SPDI properties required by each chain to guarantee. Each VNF has a unique IP address which is required for the configuration and integration with the other functions interacting also with the use case devices and apps. Pattern Orchestrator is responsible to forward the SFC request to the Pattern Engine in order to verify or instantiate SFC requests.

By leveraging the NFV framework, herein several SFC will be deployed on top of the virtual resources exposed by the NFVI. Namely, first a traffic classifier splits the incoming traffic from the SARA IoT field devices into several data flows. Then, these data flows are processed by the most proper SFC according to the traffic features.

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

- Show how NFV along with the pattern related components, offer a flexible solution to deal with the heterogeneous traffic flows of SARA. Also, how they reduce the manual intervention in the networking configuration and management, thus yielding a network management automation solution.
- Provide an NFV testbed to process, at the networking level, the different data flows of the SARA use case, according to their heterogeneous requirements.

- Provide the NFV MANO components that control the lifecycle management of the VNF along with the management of the virtualized infrastructure to deploy the VNFs. Thus, the NFV MANO leads to the automation of the virtual resources and the network functions management.

This sub use case contributes to evaluate the fulfilment by the SEMIoTICS platform of functional requirement R.UC2.3, R.UC2.15 and RUC2.17.

Table 4 SEMIoTICS Requirement evaluated by Sub use case 2

SEMIOTICS Req.-ID	SARA Req-ID	Functional	Description	Req. level
R.UC2.3	R2.5, R2.13	Yes	The SEMIoTICS platform SHOULD guarantee proper connectivity between the various components of the SARA distributed application. The SARA solution is a distributed application not only because it uses different cloud services (e.g. AREAS Cloud services, AI services) from different remote computational nodes, but also because the SARA application logic itself is distributed across various edge nodes (SARA Hubs).	SHOULD
R.UC2.15	R2.14 R2.15 R2.21	No	<p>The SEMIoTICS platform SHOULD provide low latency connectivity between the SARA hubs and cloud services (i.e. AREAS cloud services and AI services) to allow offloading of near real-time computation intensive tasks to the cloud. Examples include:</p> <ul style="list-style-type: none"> • the robotic assistant (RA) employing AI services to analyse Patient's speech (audio) and body language (video) to identify significant events – e.g. "Patient requests an escort", "Patient asks where his glasses are" <p>R.UC2.16 R.UC2.17</p> <ul style="list-style-type: none"> • mobile robotic Devices (RA/RR) exploiting cloud resources for simultaneous localization and mapping (SLAM) <p>Therefore, SARA hubs need to send with minimal delay:</p> <ul style="list-style-type: none"> • raw range data (e.g. from Lidar sensors) to identify proximal objects/objects, • real-time audio stream for speech analysis, • and real-time raw video stream (object/people recognition, gesture recognition, posture analysis). 	SHOULD
R.UC2.17	R2.15 R2.21	No	The SEMIoTICS connectivity SHOULD support real time No exchange of raw sensor data among sensors/actuators and SARA Hubs.	SHOULD

4.2. Interaction with SEMIoTICS framework

The components of the SEMIoTICS 'framework that are used in this sub use case are highlighted in Figure 2 and their interaction is detailed in the following subsections.

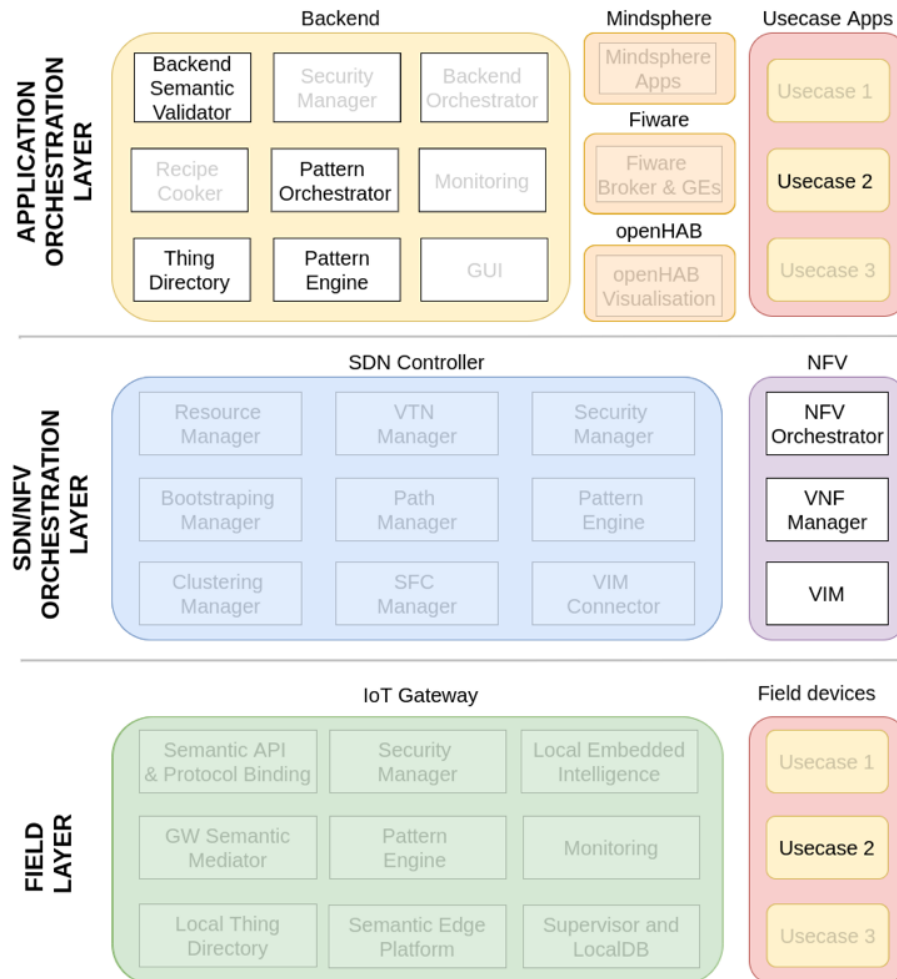


FIGURE 7 SEMIoTICS 'COMPONENTS USED IN SUB USE CASE 2

4.2.1. INTERACTION WITH NFV

The SEMIoTICS 'NFV component is a fundamental part of the sub use case presented in this section. The NFV component allows to virtualize the computing, the networking and the storage resources at the cloud level yielding the so-called NFVI. Thereby, the VNFs that compose the SFC can be deployed as VMs that leverage those virtual resources. Moreover, the NFV component has the role to manage the whole lifecycle of the VNFs, through the NFV MANO subcomponent, see Figure 2. Thereby, it controls their onboarding within the NFV framework. Also, it provides configuration files that allow to automatize the instantiation of the VNFs, e.g. by defining the features and configuration of the VMs that hold the VNFs. Moreover, it allows the termination of the VMs that hold the VNFs. Next, more details on the role of the NFV component are given.

4.2.2. NFV ORCHESTRATOR AND VNF MANAGER

These are the sub blocks of the NFV component that manage the lifecycle of the VNF. Namely, the NFV orchestrator provides templates of configuration files to allow the end-user to describe the features and functionalities of the Network Services (NS) and their associated VNFs. These configuration files are so-called VNF descriptors and NS descriptors. Note that this explanation is assuming that we use an OSM implementation of the NFV MANO, thereby a VNF is always within the context of a NS. Also, a cloud init file can be included to indicate the initial behaviour, configuration and software installations of the VM that will hold

the VNF. Once these configuration files are edited, we can onboard to the library of the OSM the VNF and NS packages that define the VNFs and the NS. Then, one can trigger the instantiation of the NS and their associated VNFs. This provokes that the NFV orchestrator communicates internally with the VNF manager, note that both blocks are implemented by the OSM. Then, the VNF Manager communicates internally with the VIM to trigger the instantiation or termination of the VNF. Thereby, it triggers the management of the virtual resources in the VIM to deploy the VMs that hold the VNFs.

4.2.3. VIRTUAL INFRASTRUCTURE MANAGEMENT

This block is the responsible for managing the NFVI, i.e. the virtual computing, storage and networking resources that allow to deploy the VNFs at the cloud level. The VNF manager and the NFV orchestrator manage the communication with the VIM internally. In other words, the end-users or applications that are external to the NFV framework do not need to communicate with the VIM, as the NFV orchestrator provides a high-level interface through the configuration files mentioned above and through REST API interfaces. Thereby, as an example, the VIM creates or terminates the VM that holds the VNF, when the instantiation or termination of the VNF is triggered at the NFV orchestrator. The instantiation or termination process implies the allocation or release of the virtual resources from the NFVI to create or terminate the VM. Note that herein the VIM and NFVI are implemented by means of the OpenStack software, see section 4.4.

4.2.4. PATTERN ORCHESTRATOR

Pattern Orchestrator is responsible for automated configuration, coordination, and management of different patterns and their deployment. Moreover, it is used for processing a received requirement in order to translate it to Drools facts. The result of the said processing enables the Pattern Orchestrator to choose which Pattern Engine should receive the corresponding Drools facts in order to reason with them.

4.2.5. PATTERN ENGINE

Pattern Engine at the application layer, includes the pattern rules in the form of Drools rules, responsible for verifying and instantiating VNFs and SFCs. With the said rules the Pattern Engine is able to reason whether a new VNF is needed to be instantiated in order to complete a specific SFC. Moreover, if an SFC is not instantiated at all, the rules allow the instantiation of the SFC after having gathered all the necessary VNFs. Due to the fact that the Pattern Engine at the application layer, has a global view of the pattern facts across all layers, it makes it appropriate for providing up to date information on a graphical interface designed specifically for SFC management in a higher level. This interface is called SFC GUI and extracts all the information depicted directly from the Pattern Engine.

4.2.6. THING DIRECTORY

Thing Directory at the application layer represents a global repository for all the registered Things and is used to browse Things based on their Thing Description. The information extracted from Thing Directory is used from the Backend Semantic Validator to verify the interoperability between two Things. Moreover, it is used from the SFC GUI to populate the available nodes that can be used as source and destination of a flow.

4.2.7. BACKEND SEMANTIC VALIDATOR

The Backend Semantic Validator component is responsible for semantic validation mechanisms at the backend layer. It receives from the Pattern Engine the source and destination of a flow in order to verify the interoperability between them. When the interoperability between source and destination is not present, The Backend Semantic Validator informs the Pattern Engine so that the latter can take appropriate actions.

4.3. Setup testbed and integration

4.3.1. NFV TESTBED

Herein, we leverage CTTC's NFV testbed to host the VNFs that process the traffic stemming from the SARA's IoT GW. More specifically, this testbed consists of the next functional blocks according to the ETSI's NFV standard [1].

First, the VNFs are deployed on top of a virtualized infrastructure that exposes virtual computing, networking and storage resources. This infrastructure is so-called Network Function Virtualization Infrastructure (NFVI) and is one of the fundamental blocks of the NFV framework. Second, the whole NFV framework is managed by the so-called NFV Management and Orchestration (NFV MANO). More specifically, the NFV MANO is responsible for the whole lifecycle of the VNFs, i.e. their creation, deployment and termination. The NFV MANO consists of three sub blocks. The NFV Orchestrator (NFVO), the VNF Manager (VNFM) and the Virtualized Infrastructure Manager (VIM). The virtualized resources and the whole NFVI are controlled by the VIM. The whole lifecycle of the VNF is managed by the NFVO and the VNFM, which have the responsibility to provide northbound API interfaces to interact with the end-user. This allows the end-user to edit the VNF features in terms of computing, storage or networking requirements. Also, it permits to onboard the VNF within the NFV framework, i.e. as part of the library of available VNFs. Moreover, the NFVO/VNFM blocks let the end-user to trigger the VNF instantiation or termination on top of the NFVI. In this regard, it is worth mentioning that the NFVO communicates downwards with the VNFM and the VNFM in turn communicates downwards with the VIM. More insights on NFV for the SEMIoTICS purposes are given in the SEMIoTICS 'deliverable D3.8 [2].

The above-mentioned NFV blocks are implemented in CTTC's NFV testbed by using the next software:

- OSM (Open Source MANO) [3]. It implements the NFVO along with the VNFM blocks.
- OpenStack [4]. The OpenStack controller implements the VIM, the OpenStack compute node implement the NFVI.

The NFV testbed, mentioned above, has the topology described in Figure 8. More specifically, the NFV MANO consists of two functional blocks. Thus, on the one hand, the OSM implements the NFVO and VNFM. On the other hand, the NFV MANO contains another sub block, the OpenStack controller node, which implements the VIM. The NFVI is composed of one functional block, an OpenStack compute node that exposes its virtualized resources to instantiate the VNFs. A switch is leveraged for the communications between all the functional blocks mentioned above.

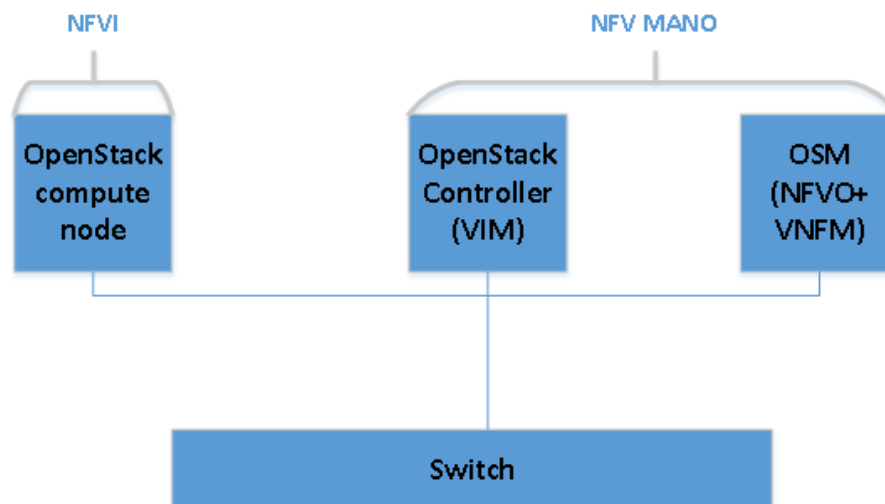


FIGURE 8 TOPOLOGY OF THE CTTC's NFV TESTBED.

In order to implement the NFVI and NFV MANO functional blocks described in Figure 8 we have deployed VMs within the servers available at the CTTC premises. Namely, each functional block of Figure 8 is a VM. Moreover, the computing, storage and RAM memory of the VM have been set up according to the requirements of each of the software components. Next, we describe the features of each VM and the switch:

- VM containing the OSM.

- Storage of 98 GB.
- RAM of 8 GB.
- 4 CPU cores.
- Operative system is the Ubuntu 18.04.4 LTS.
- VM containing the OpenStack controller node.
 - Storage of 98 GB.
 - RAM of 32 GB.
 - 6 CPU cores.
 - Operative system is the Ubuntu 18.04.4 LTS.
- VM containing the OpenStack compute node.
 - Storage of 10 GB.
 - RAM of 8 GB.
 - 4 CPU cores.
 - Operative system is the Ubuntu 18.04.4 LTS.
- The switch is a 10GBASE-T programmable switch.

Moreover, it is important to note that OSM version 7 has been installed, following the guidelines in [5]. Regarding the OpenStack, the Train version has been installed and set up in our NFV testbed. To this end, a devOps approach has been adopted. More specifically, OpenStack Ansible has been leveraged to ease and to automate the installation of the application and the configuration process of the underlying infrastructure that will support the OpenStack. We have followed the OpenStack Ansible guidelines in [6] to install and configure OpenStack Train in our NFV testbed. It is worth noting that Ansible is a type of configuration-management software, which follows an infrastructure as code approach. Thereby, it relies on an inventory, i.e. a set of configuration files that describe the nodes that can be accessed by Ansible. Moreover, Ansible relies on a set of files that permit to carry out operations on the managed nodes, thus these files are so-called Playbooks. Note that each Playbook maps a group of hosts to a set of roles.

Additionally, in FORTH's premises, the Pattern Orchestrator, Pattern Engine and SFC GUI are deployed also in VMs. An Intel NUC (Figure 9) 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 SFC GUI.

- Storage of 20 GB.
- RAM of 2 GB.
- 4 CPU cores.
- Operative system is the Ubuntu 18.04.4 LTS



FIGURE 9 INTEL NUC

The detailed procedure used to validate the NFV testbed described in this section can be found in Appendix A.

4.3.3. SERVICE FUNCTION CHAINING PATTERNS VALIDATION

Service Function Chaining Patterns provide the ability to define an ordered list of security network services (e.g. firewalls, DPIs, IDS) for security in network infrastructures by creating chains at design and by updating function in chains based on available ones at runtime. SFC patterns should cover the placement, security and scalability aspect of SEMIoTICS network infrastructures. The deployment of network service functions can be used to guarantee specific network security and dependability properties. However, stitched them into chains can satisfy more than one SPD properties. One of the innovative approaches supported by this work, is the dynamic instantiation of SFCs based on the predefined SFC patterns. When there is a request for an SFC instantiation containing service functions, the depicted in Figure 10 procedure should be followed. If the SFC does not exist, the instantiation of the respective SFC is deployed through the identification of the requested VNFs. If the VNFs exist in the service nodes, the SFC is updated including these VNFs. If the VNFs do not exist, the service node with the available resources is requested to instantiate the respective VNFs. The procedure is ended when all the requested VNFs are included in the SFC.

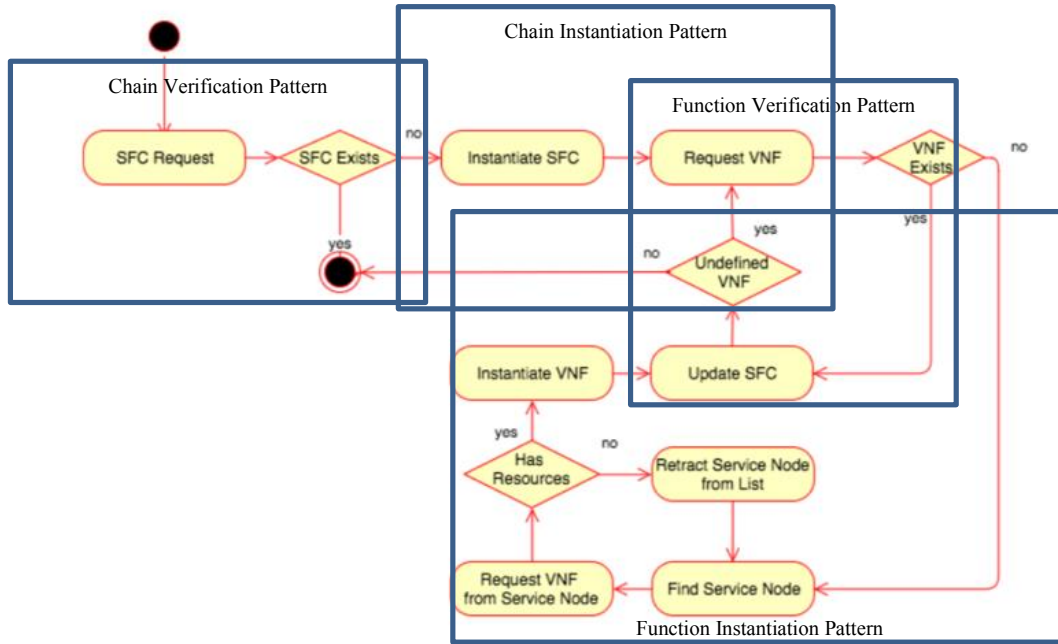


FIGURE 10 VNF INSTANTIATION BASED ON SFC REQUEST

To proceed to the above description, SFC patterns are developed to achieve the requirement for end to end guarantees by the traffic forwarding through different security service functions. The patterns can be expressed as Drools rules⁴ to enforce the following requirements:

- Verify service function chains on chain requests.
- Instantiate service function chains, if the required functions have been already instantiated.
- Verify functions to insert the in the request chain.
- Instantiate not defined service functions to satisfy service function chain requests.

The ultimate goal of the SFC GUI is to enable the Administrators of the SARA solution to interact with the system that provides a secure networking infrastructure, via the associated proactive and reactive security mechanisms, such as the deployment of network security services and the continuous monitoring and intrusion detection. In Figure 11 the SFC GUI is presented which gathers in one screen all the necessary information.

⁴ Drools Business Rules Management System <https://www.drools.org>

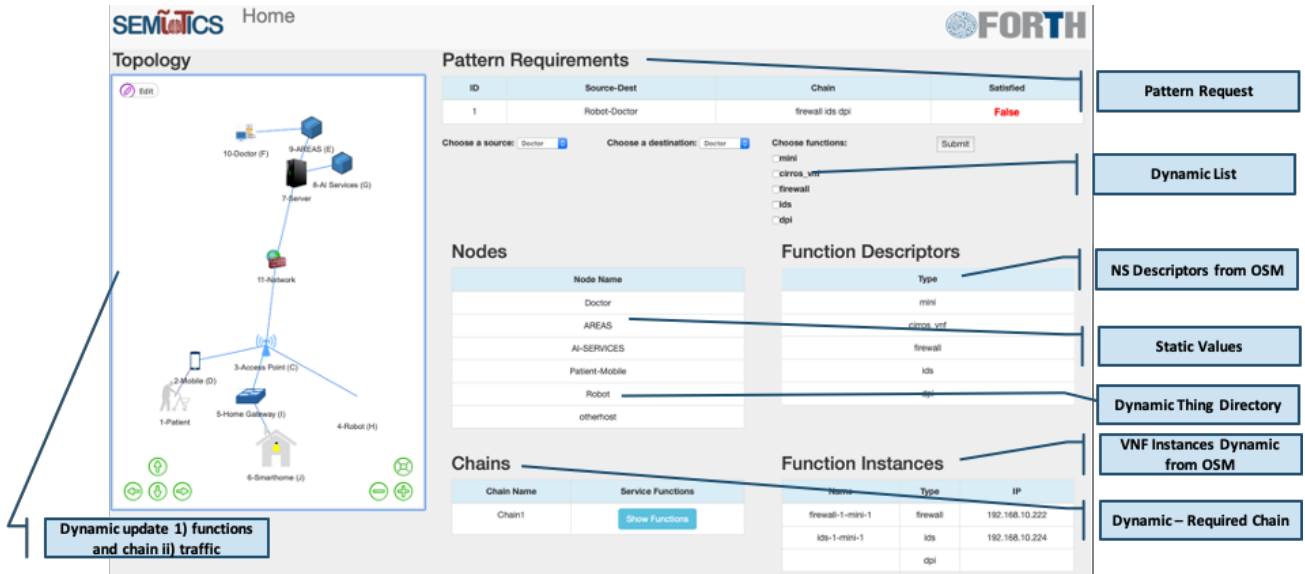


FIGURE 11 SFC GUI

All information depicted in SFC GUI is gathered from Backend Pattern Engine with a simple http-GET request (Figure 12).

- Backend Pattern Engine gathers information from:
 - Thing Directory (Nodes)
 - OSM (Function Descriptors and Function Instances)
 - SFC Requests (Chains)
 - Pattern Requirement (user input)



FIGURE 12 PATTERN ENGINE REST FOR SFC GUI

A user can request for specific VNFs to be applied in a flow of a specific source and destination such as the end hosts such the robot, patient or backend services such as AREAS, AI services. The list of sources and destination will be populated on the next cycle based on the information retrieved from the Thing Directory or manually in case of backend services. The VNFs that the user can choose from, is dynamically populated with information provided by the OSM.

Upon clicking on the submit button a new Pattern Requirement is created that is sent directly to the Pattern Engine and is consumed by the addSFCReq endpoint (Figure 13).

POST

/addSFCReq2

addSFCReq2

Response Class (Status 200)

string

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
factString	<pre>{ "sfcReqID":0, "src":"Doctor", "dst":"AREAS", "chain":[{"name":"firewall","type":"firewall","instantiated":"false"}, {"name":"dpi","type":"dpi","instantiated":"false"}], "satisfied":"false" }</pre>	factString	body	string

Parameter content type: application/json

Response Messages

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

Try it out!

Hide Response

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{"sfcReqID":0,"src":"Doc'
```

Request URL

https://139.91.68.107:9443/addSFCReq2

Request Headers

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

Response Body

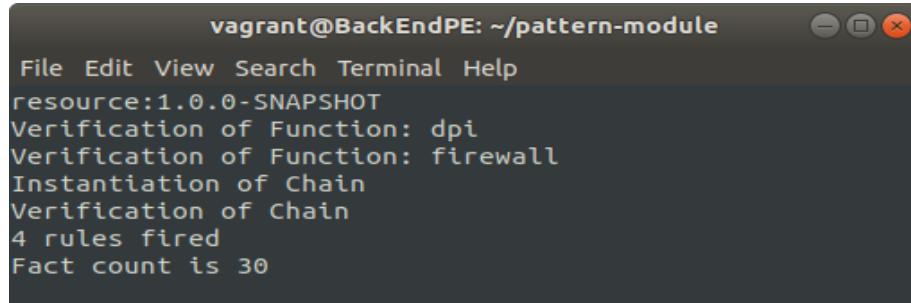
no content

Response Code

200

FIGURE 13 PATTERN ENGINE REST FOR SUBMIT BUTTON OF SFC GUI

After the new requirement has been consumed by the Pattern Engine, the pattern rules (Figure 15, Figure 16, Figure 17, Figure 18) are fired causing the verification and instantiation of VNFs and SFCs (Figure 14)



```

vagrant@BackEndPE: ~/pattern-module
File Edit View Search Terminal Help
resource:1.0.0-SNAPSHOT
Verification of Function: dpi
Verification of Function: firewall
Instantiation of Chain
Verification of Chain
4 rules fired
Fact count is 30

```

FIGURE 14 PATTERN ENGINE FIRING SFC RELATED RULES

```

1. rule "Service Function Chaining Chain Verification"
2. salience 10
3. ruleflow-group "SFC"
4. when
5.     $src: Placeholder($srcName: name)
6.     $dst: Placeholder ($dstName: name)
7.     $function: Function($type:=type, instantiated==true)
8.     $chain: Chain($functions: functions, $functions contains
    $function, instantiated==true)
9.     $PR: Property($src:=src, $dst:=dst,
    $reqFunctions:=property.chain.functions, satisfied== false)
10. then
11.     System.out.println("Verification of Chain");
12.     modify($PR){satisfied=true};
12. and

```

FIGURE 15 CHAIN VERIFICATION

```

1. rule "SFC Chain Instantiation"
2. ruleflow-group "SFC"
3. salience 20
4. when
5.     $src: Placeholder($srcName: name)
6.     $dst: Placeholder($dstName: name)
7.     $function: Function($type:=type, instantiated==true)
8.     $chain: Chain($functions: functions, $functions contains
    $function, instantiated==false)
9.     $PR: Property($src:=src, $dst:=dst,
    $reqFunctions:=property.chain.functions, satisfied== false)
10.    Function($type==type) from $reqFunctions
11. then
12.    System.out.println("Instantiation of Chain");
13.    modify($chain){instantiated=true};

```

FIGURE 16 CHAIN INSTANTIATION

```

1. rule "Virtual Service Network Function Verification"
2. ruleflow-group "SFC"
3. salience 30
4. when
5.     $src: Placeholder($srcName: name)
6.     $dst: Placeholder($dstName: name)
7.     $function: Function($type:=type, instantiated==true)
8.     $chain: Chain($functions: functions, $functions not contains
    $function, instantiated==false)
9.     $PR: Property($src:=src, $dst:=dst,
    $reqFunctions:=property.chain.functions, satisfied== false)
10.    Function($type==type) from $reqFunctions
11. then
12.    System.out.println("Verification of Function");
13.    $chain.addFunction($function);
14.    update($chain);
15. end

```

FIGURE 17 NETWORK FUNCTION VERIFICATION


```

1. rule "Virtual Service Network Function Instantiation"
2. ruleflow-group "SFC"
3. salience 40
4. when
5.     $src: Placeholder($srcName: name)
6.     $dst: Placeholder ($dstName: name)
7.     $function: Function($type:=type, instantiated==false)
8.     not Function($type:=type, instantiated==true)
9.     $chain: Chain($functions: functions, $functions not contains
    $function, instantiated==false)
10.    $PR: Property($src:=src, $dst:=dst,
    $reqFunctions:=property.chain.functions, satisfied== false)
11.    Function($type==type) from $reqFunctions
12. then
13.    System.out.println("Instantiation of Function");
14.    Function function = new Function($function.type);
15.    modify($function){instantiated=true};
16. end

```

FIGURE 18 NETWORK FUNCTION INSTANTIATION

4.3.5. BACKEND SEMANTIC VALIDATOR AND THING DIRECTORY VALIDATION

Backend Semantic Validator (BSV) is able to verify whether two endpoints, a.k.a. source and destination such as end hosts (robot, patient, smart home) or backend services (doctor, AREAS, AI Services), are semantically interoperable. Figure 19 shows how the Backend Semantic Validator is involved in the flow and Figure 20 show a closer look to that process. The main idea here is to feed the BSV with a pair of end nodes that represent the source and destination. Their Thing Description is looked up in the Thing Directory and based on the result the BSV is capable to respond whether interoperability exists between them.

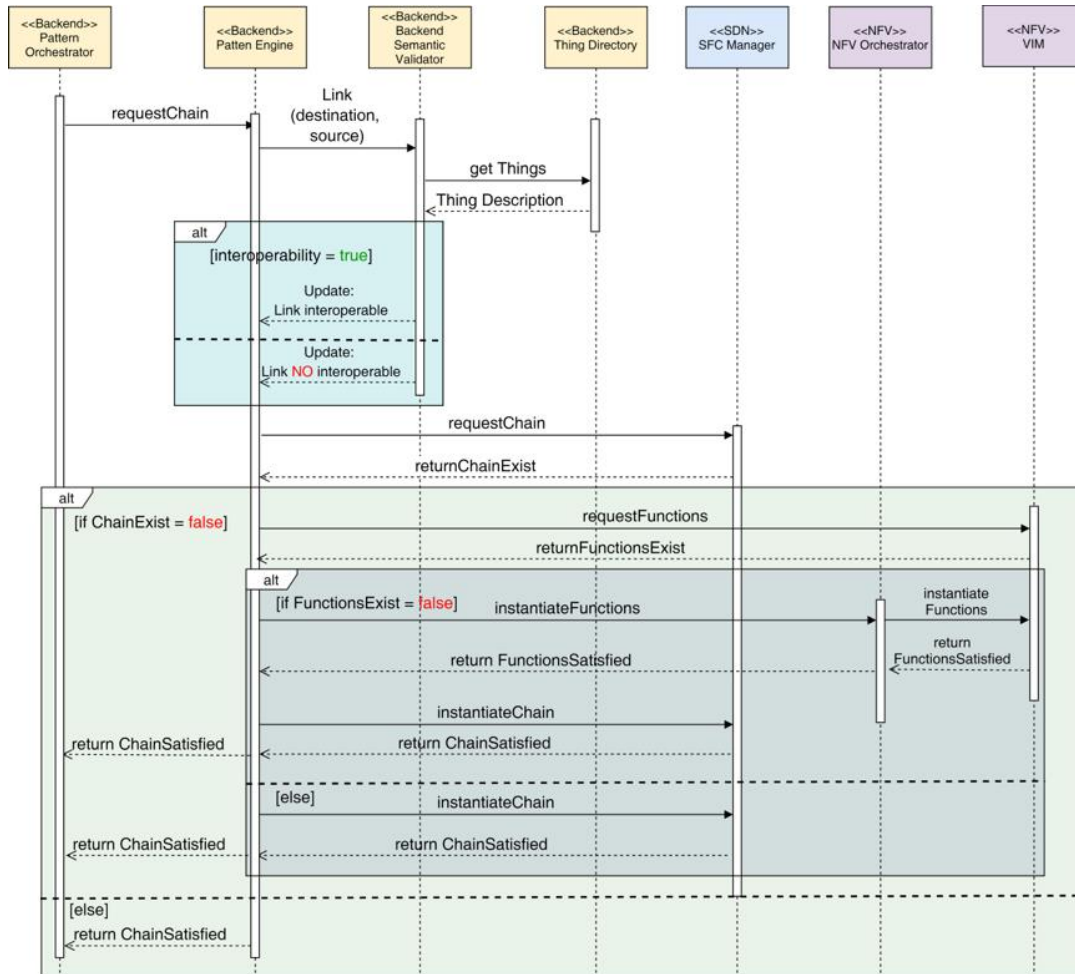


FIGURE 19 BSV INTEGRATION

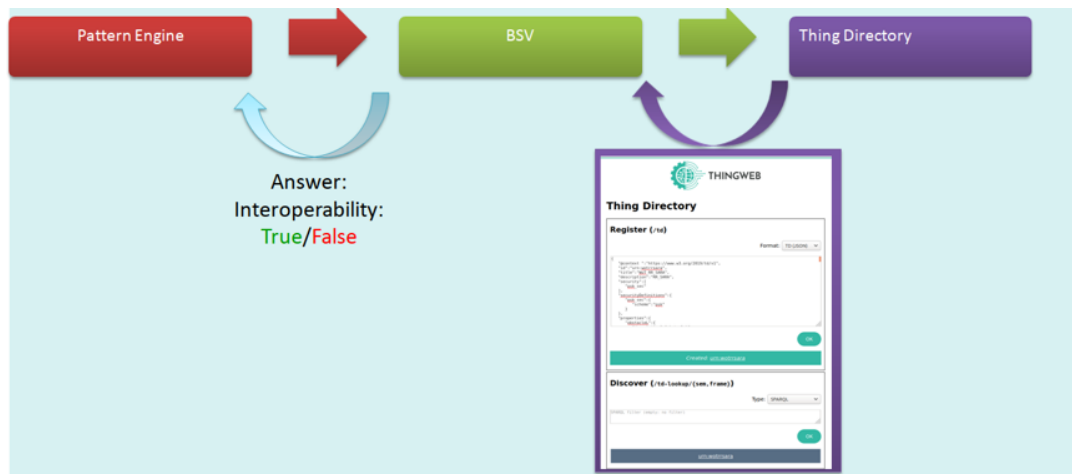


FIGURE 20 BSV AND THING DIRECTORY INTERACTION

The above are possible by the REST endpoint “validateData” (Figure 21) of the BSV which receives the source and destination in order to lookup their Thing Description in Thing Directory

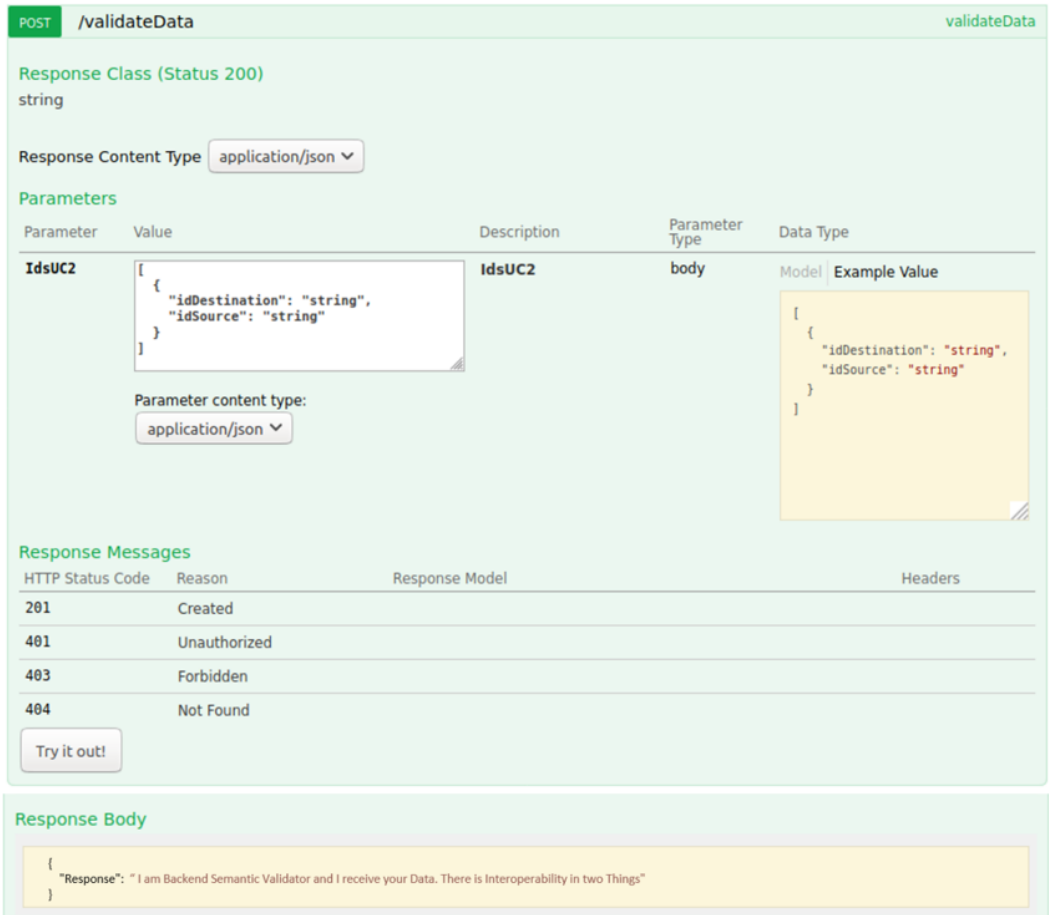


FIGURE 21 BSV “VALIDATEDATA” ENDPOINT

The evaluation of the SFC approach in the SARA use case was presented in this section. The SDN/NFV-enabled test-bed setup for service function chaining was tested in the NFV-enabled architecture for the dynamic VNF instantiation enabling the interaction between the pattern engine to instantiate VNF through the OpenSource MANO (OSM) and OpenStack.

5. SUB USE CASE 3: ENFORCEMENT AND MONITORING OF GDPR-COMPLIANT ACCESS CONTROL TO CONFIDENTIAL AND SENSITIVE DATA

5.1. Scope and objectives

We would like to highlight a very important topic of security and privacy: the protection of personal data. Not only with the advent of the GDPR, the protection of confidentiality and also integrity of personal data has become of paramount importance. Especially in a medical use case, such as UC2. However, we would like to note that when we speak of confidentiality protection for GDPR-compliance we can as well map the technical protection mechanisms of SEMIoTICS to protect the confidentiality of trade-secrets. Following the definition in TRIPS from the World Trade Organization, trade secrets describe information that is considered to have a commercial value due to its secrecy and that its secrecy must be protected against third parties. So, when we can prohibit that information is made available or disclosed to unauthorized individuals, which is the more technical definition of confidentiality from ISO 27000, then we can protect both personal data to comply with the GDPR or trade-secrets. In use case 2 the former is the more important case, due to which we decided to put GDPR in the title.

The challenges we face are that the GDPR requires you to limit the amount of data you collect to what you actually need and further limit the access to the information already collected to the authorized entities.

The access to data needs to be managed in a very generic fashion to make it usable at large scale but also at very fine-grained levels in order to allow the SEMIoTICS framework to be adapted to many different usage scenarios. To fulfil this objective the access control is attribute-based which allows Role-based access control (RBAC), e.g. you allow or deny access based on an attribute that specifies a role, e.g. like entity with the attribute “doctor” which shall be given to those with the role of a medical doctor in our hospital use case. Using attributes enables fulfilling two GDPR-requirements, first and most obvious they allow to specify policies which control and thus allow to restrict access to the sensitive data. Second, it allows that selected individuals, which have the attribute, could gain access without having to reveal more than their attribute, i.e. a more privacy-friendly form of access control can be implemented. So instead of having to tell the system that one has to state that one is the doctor “Max Mustermann” one would only need to reveal to the system that one has the correct set of attributes, e.g. being a “doctor”. This limits the data being collected.

Another challenge was to ensure that the confidentiality of medical data even when that data becomes stored or transported outside of SEMIoTICS perimeter. It shall stay protected for confidentiality. To address this challenging objective, SEMIoTICS facilitates a cryptographic mechanism, known as attribute-based encryption (ABE). This is like normal cryptographic encryption, e.g. we encrypt data and unless you have the correct decryption key you cannot read the data. However, in many applications it is not a-priori known who will be the authorized recipient. Instead one might only know certain attributes of authorized entities and all those entities who possess the right attributes would form a group. This means that encryption can happen, e.g. data should only be read by entities that are doctors, without knowing nor without needing to know who is a doctor at the time of retrieval. SEMIoTICS achieves this loose coupling and still enforces confidentiality for data even if it is stored on systems outside the realm of SEMIoTICS.

To enforce one of the most sought-after features of GDPR-compliance, that of retention times, the attribute-based encryption capabilities of SEMIoTICS are combinable with a time-variant attribute. That means that the patient’s system might generate data, e.g. a video stream or a location, which is encrypted not only for doctors (see our previous example), i.e. using an attribute “doctor” during encryption, but with an additional attribute that encodes that this data can only be accessed for a certain time frame, e.g. 48h is an often used retention period for videos. The technical details have been discussed in D 4.12.

Last but not least, the challenge is to monitor that the system correctly enforces the intended behaviour, for this we describe how SEMIoTICS monitoring is used as a solution for periodical self-audit capabilities by monitoring for pattern-compliance.

This sub use case contributes to evaluate the fulfilment by the SEMIoTICS platform of functional requirement R.UC2.16.

Table 5 SEMIoTICS Requirement evaluated by Sub use case 3

SEMIOTICS Req.-ID	SARA Req.-ID	Functional	Description	Req. level
R.UC2.16	R2.2 R2.22 R2.23	Yes	The SEMIoTICS platform SHOULD support secure bootstrap, configuration and diagnostic of SARA hubs and devices.	SHOULD
R.GSP.9	R2.24	No	<p>The SARA system should provide robust mechanisms to protect Patient-related data - i.e. to:</p> <ul style="list-style-type: none"> • Authenticate the sources of Patient data – in particular: to avoid the case that data monitored from one Patient is incorrectly ascribed to another. • Prevent unauthorised access to or manipulation of stored Patient data. • Secure the transmission of Patient data. <p>Such data includes (but is not limited to):</p> <ul style="list-style-type: none"> • System configuration data for the Patient • Daily activity rules for the Patient • Monitored daily activity data for the Patient • Audio-visual streams transmitted during telepresence 	SHALL
R.GSP.10	R2.28	No	The SARA system MUST fully comply with all relevant Italian laws governing the privacy, security and storage of sensitive Patient health-related data.	MUST

5.2. Interaction with SEMIoTICS framework

5.2.1. AUTHORIZATION POLICY MANAGEMENT

Figure 22 shows the workflow of the Remote Gait Analysis Assistive Task (see section 2.1.2): the workflow starts with the scheduling of the Gait Analysis session. The data shared by devices can only be provided to systems or users who have the appropriate privileges at the moment, specifically, the patient's data can only be accessible by his or her doctor during the specified period (e.g. during the exercises). Changing policies for accessing specific devices will take place before and after exercise. The design shown in Figure 22 fulfils data protection principles like purpose limitation – the data is processed only for the legitimate purposes specified explicitly to the patient, the data processing is done in such a way which ensures appropriate security, integrity and confidentiality.

The exercise can be either postponed or started. If patient postpones the exercise, it is rescheduled (i.e. for the next day) and it's one end of the flow. If the patient chooses otherwise, SARA sends an HTTP request to Pattern Orchestrator informing that the exercises have already started and the doctor should get permission to get the patient's data for the time of exercises. Pattern Orchestrator transforms the request and sends it to Backend Pattern Engine. Then, using the Security Manager's API Backend Pattern Engine updates the policies to give the doctor access to the data. As soon as the patient finishes exercises, the flow is repeated but the only difference is the update of policies takes away privileges to inspect the patient's data instead of giving them.

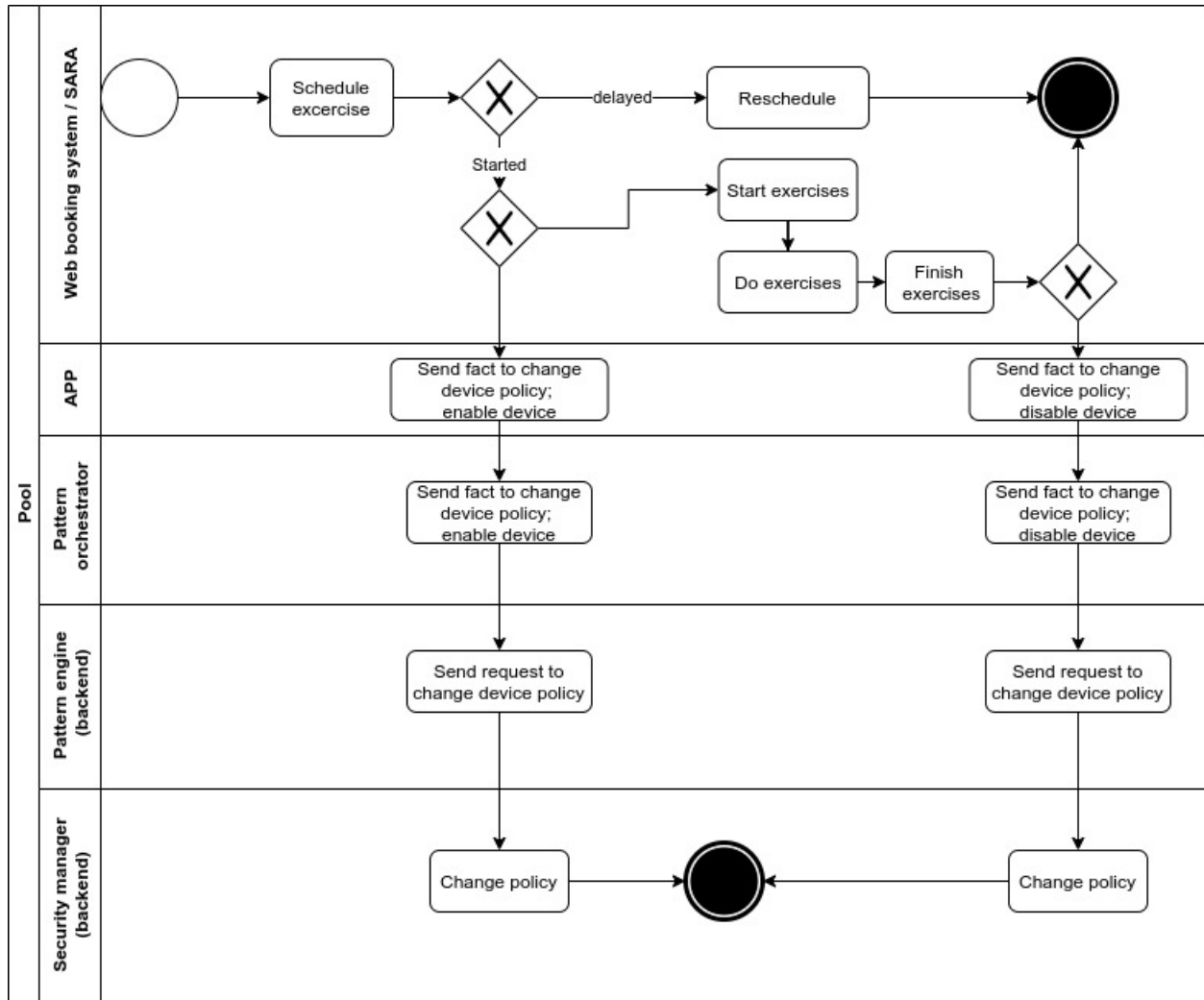


FIGURE 22 REMOTE GAIT ANALYSIS WORKFLOW

- **SARA** The component is responsible for interaction with the patient. SARA's main role in the use case will be to trigger the request to update the doctor's policies after the patient starts exercises and as soon as they are finished.
- **Security Manager (backend)** The component will be used in the sub-use case both as the security provider and policy management point. It will serve to generate an authentication token using SARA's credentials and later on, to verify entities based on their token. The second role of Security Manager will be to alter doctor's permission, either to allow him to get access to the patient's data during the exercises or to take away privileges after the exercises are finished.
- **Pattern Orchestrator** Pattern Orchestrator receives the HTTP request from SARA and transforms it into the pattern, and sends it to the Backend Pattern Engine.
- **Backend Pattern Engine** Backend Pattern engine transform the pattern into rules and sends HTTP request to Security Manager to change the doctor's privileges.

5.2.2. AUTHENTICATION MANAGEMENT

To authenticate SARA, to every HTTP request an authorization token is added. The Token is generated based on SARA's unique client identification number and client secret. All incoming request to the device will be intercepted by Policy Enforcement Point which verifies whether the owner of the request has sufficient permission to get access to the data. If it does, the request is forwarded to its original destination, if not an error is thrown due to lack of privileges (according to standard RFC 7519).

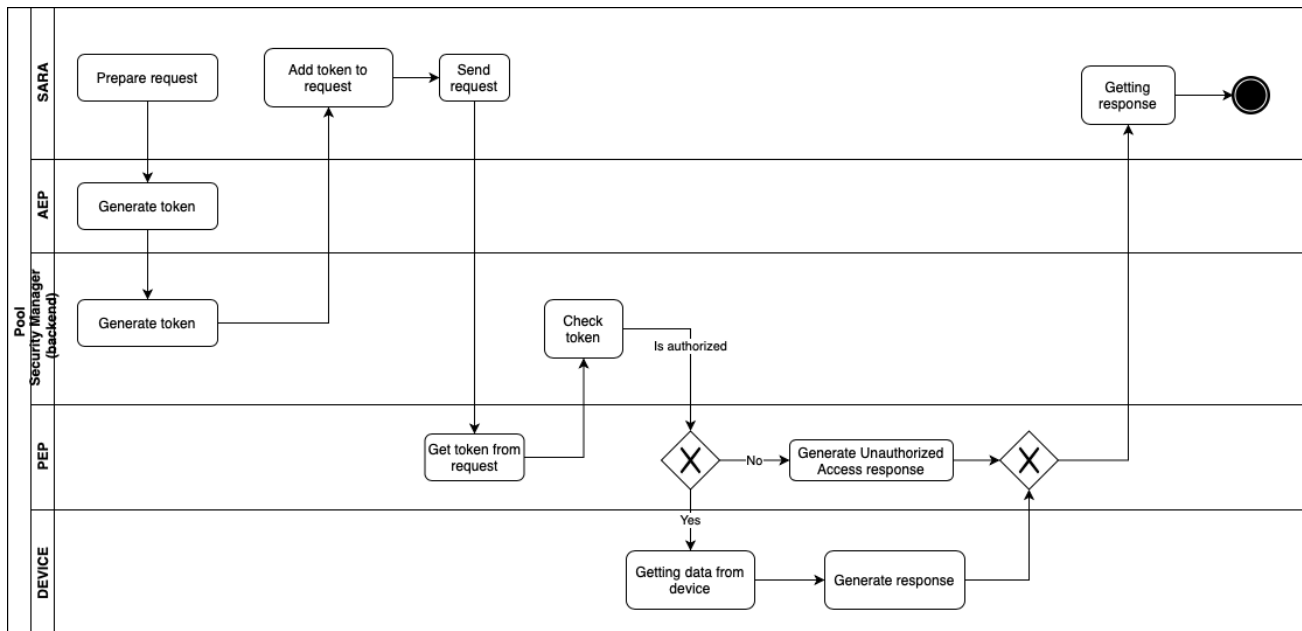


FIGURE 23 AUTHENTICATION MANAGEMENT PROCESS

- **AEP (Authentication Enforcement Point)** Authentication Enforcement Point it is a stand-alone application which adds an authentication token to every HTTP request that is coming out of SARA. To apply a valid token, AEP uses Security Manager's OAuth 2.0 Clients Credentials Grant flow using SARA's client-id and client secret. Using these values, Security Manager can generate a unique token for the component, which is added to the HTTP request header in order to not be rejected by Policy Enforcement Point.
- **PEP (Policy Enforcement Point)** In order to secure access to the components' resources, the APIs of a device is protected by Policy Enforcement Point. PEP provides security in two ways. Firstly, it is deployed as a sidecar along with the primary application. They both share the same network allowing them to communicate through localhost. The main application is only accessible from within local network, because it does not expose its port outside the device, only PEP does that, meaning that in order to get the resource of the main application all HTTP traffic must go through PEP. Secondly, the HTTP request must bear a valid authorization token and the client which makes an HTTP call has to be authorized to do so. To evaluate client authorization, PEP uses the Security Manager's Policy Decision Point. After the evaluation of the client's policy, HTTP call is either allowed to reach the main application or it is rejected due to the insufficient permission.
- **Device** The device is a generic component which is done according to WoT standard and have API. The device shares the data through the interface.

5.3. Setup testbed and integration

There are 4 main steps for enforcing and monitoring that are taking place in several components spread over all layers of the SEMIoTICS architecture:

- First, setup tasks --as well as later management tasks-- establish the policies and users as well as their attributes;
- Second, the actual enforcement of access control;
- Third, we dynamically adapt the access control to react, e.g. to emergency situations like in the fall assistance task;
- Forth, monitoring for compliance, e.g. GDPR-conformance.

In the remaining text we will detail how these main steps are happening within the generic storyline of UC2. In section 5.3 we will then highlight which components' interactions are facilitated to reach our objectives.

5.3.1. SETUP STEPS

SEMIOTICS is built upon the concept that each participating entity has their own identity and their own set of attributes. If you think about human users, the name would be "Max Mustermann" and an attribute could be "doctor". To ensure that only particular entities are able to obtain certain attributes, the identity management must initially setup administrative users who can change those attributes or create new users, and the system could also gradually enforce administrative restrictions, by enforcing access control on the ability to modify (create, change, delete) attributes or new entities. Further there are policies which govern which access is allowed and which access is restricted, this follows the general principle of access control policies, e.g. an entity is given the right to access an object in a certain manner, e.g. an entity with the attribute doctor is allowed to read Hans Meier's medical records. These policies are created initially as part of the system's setup. The use of patterns which allow to create those even faster. We explained the usage and technical backgrounds of attribute-based policies as well as entity related policies in D4.12 Section 2.2.4.1. These policies are defined individually as required by the respective application or use case. In order to provide a more user-friendly way to define the policies defined by config files, there is the possibility to set the policies through the currently deployed graphical user interface (GUI). Our GUI is implemented as a node.js based application, which facilitates the Security Manager API and can also be used to showcase how to interact with the Security Manager. All necessary functionality of the Security Manager's API is made available via the GUI and can be accessed via the interface. This removes the hurdle of having to write complicated configuration files to initially setup the Security Manager.

Furthermore, one of the initial steps during the system setup is to define the patient's privacy settings. As each patient may have other privacy concerns, we can individually enforce different privacy settings for each patient. This can range from defining how long each private and sensitive data should be accessible, by whom it should be accessible to defining several characteristics of the person that can access the data (e.g. gender, age, language, country). These settings can, of course, be later changed manually by the patient him-/herself.

5.3.2. ENFORCING ACCESS CONTROL

The SEMIoTICS framework allows us to enforce access control on privacy sensitive data in two different ways:

- **By means of access control:** When an entity with the role "doctor" tries to access the patient's data, the request is intercepted by the Policy Enforcement Proxy (PEP) (see Section 5.2.6 and D4.12 Section 3.2.4.1 for detailed information) which then forwards this request to the Security Manager. In this context the Security Manager functions as a Policy Decision Point (PDP), that then either allows or rejects the access to the patient's data, based on the entity's attributes (see Section 5.2.6 and D4.12 Section 3.2.1.1). For Use-Case 2, this mechanism is especially used for step 7 of the Remote Gait Analysis Assistive Task.
- **By means of encryption:** During an emergency case, the patient's data is accessible by anyone of the role "doctor" (enforced by the Security Manager and PEP). However, the SEMIoTICS framework is also able to enforce the patient's previously defined privacy settings. This is done by encrypting the patient's data with the help of Attribute Based Encryption (see D4.12 Section 3.2.1.4). Due to the encryption, the

privacy sensitive data is now only accessible by an entity that fulfils all conditions that the patient wants to be enforced (e.g. gender of the doctor, language, age). Additionally, the maximum availability of the data can also be enforced by this technique. E.g. the patient can control how long the data can be decrypted, by defining a maximum lifetime of the data which is then part of the encryption. When this time-window expires nobody will be able to decrypt this data anymore. For Use-Case 2, this mechanism will be especially helpful for step 7 of the Fall Management Assistive Task.

5.3.3. DYNAMIC POLICY ADAPTATION

We assume the following case:

Alice is a Call Centre Operator who normally is not able to get Bob's location. In an emergency (e.g. during the execution of Fall Management AT), however, Alice should be able to retrieve it. The SM offers the possibility to dynamically adjust access rights very quickly through the policies. If Bob falls, the application changes the policy via the API of the Security Manager dynamically. Now it is possible for Alice to get the location (see [Figure 24](#)).

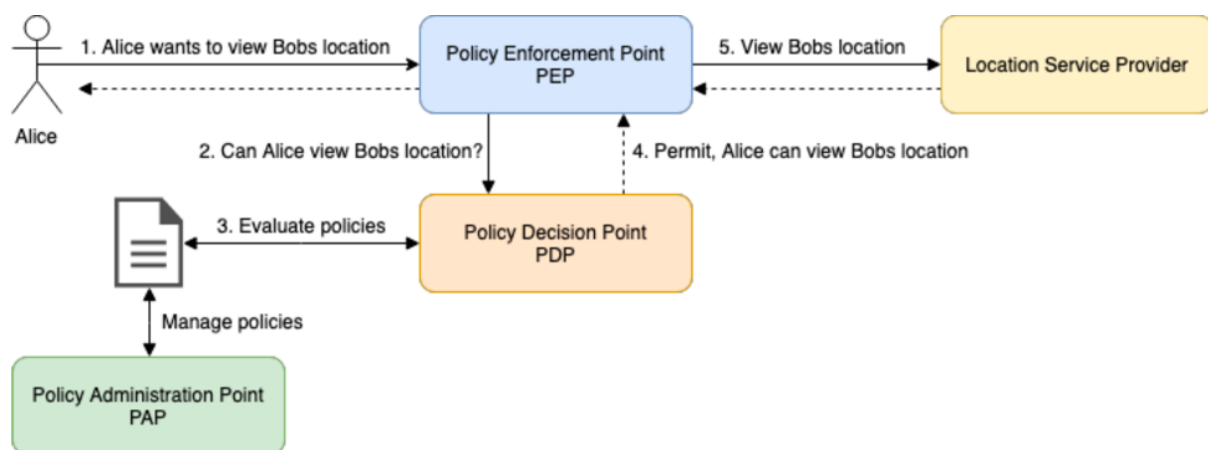


FIGURE 24 DYNAMIC POLICY ADAPTATION

5.3.4. MONITORING COMPLIANCE

It is important that in complex systems we not only set policies and enforce them, but that we also –at least– periodically monitor that the system is still conforming to the intended behaviour; this is especially true in a system where the policies will get dynamically updated to adapt to changes in the environment as supported by SEMIoTICS. In use case two this dynamicity happens to adapt to medical emergency situations, in which case access is granted to aid in such emergency situations.

As an example, let us explain a bit more details about the Fall Management Assistive task of UC 2. Here we allow logging of and access to a patient's geo location, in order to correctly assist the patient, but only in case the patient fell down (dynamic policy update).

In order to ensure compliance SEMIoTICS is capable, and we take advantage of this functionality in UC 2, to check actively if the intended policies are enforced. This allows to detect suspicious access capabilities or changes to the policy which were not reverted. This self-audit capability is based on checking periodically the current access capabilities and compare them with expected values, and this checking is based on the SPDI patterns, here especially the security (S) and privacy (P) related patterns are of concern. To stay within the general storyline, there is a pattern on privacy that states that only patients normally have access to their location. Thus, in SEMIoTICS there is a periodic question to the security manager asking to list all entities who are not patients which have access to location data, and if this answer is not an empty list, then the privacy pattern is violated and an alarm is raised. Of course, in emergency cases, like the fall of a patient, this default

privacy setting is dynamically overruled. But, only for those patients that fell and only for the duration of the emergency. Thus, the monitoring component should not have to constantly alert the data protection officers.

So, by creating patterns that implement GDPR-compliant behaviour SEMIoTICS allows the systems administrators to enable the system to self-check itself.

6. SUB USE CASE 4: CONTROLLING BULBS AND ROBOTS

6.1. Scope and objectives

This sub use case aims to demonstrate how the SEMIoTICS Semantic Interoperability technologies developed in the context of Task 3.3 can help the developers of the SARA solution to interface heterogeneous devices ranging from light bulbs to semi-autonomous robots.

As should be evident from the description of the Assistive Tasks given in section 2.1, in general, a SARA Assistive Task is realized through the interaction of various devices. As an example the Fall Management AT requires that the Robotic Assistant interacts with the lighting system of the Smart Home whilst the Gait Analysis AT requires the Doctor Web Application, interacts across the Internet, with a remote Robotic Rollator (figure 25).



FIGURE 25 THE ROBOTIC ROLLATOR BEING USED BY THE SARA PROTOTYPE

In the Fall Management AT the Robotic Assistant needs to discover which are the lights available in the room where the possible fall event occurred and then request their activation to illuminate the scene. Once the available lights are discovered the interaction between the robot and the lights bulbs may occur using different protocols (e.g. ZigBee, Bluetooth, Z-wave). This need is reflected in the SARA requirements R.UC2,5 and R.UC2.11.

In the development of SARA there is the general requirement to keep the application logic independent from the underlying hardware as much as possible. As an example, in the context of the Gait Analysis AT, we wish

to keep the application logic deployed on board of the Robotic Rollator independent from the specificities of the rollator used for the prototyping. This general requirement is reflected in the requirement R.UC2.6.

The table 5 presents functional requirements of the SEMIoTICS platform evaluated by means of sub use case 6.

Table 5 SEMIoTICS Requirement evaluated by Sub use case 4

SEMIOTICS Req.-ID	SARA Req.-ID	Functional	Description	Req. level
R.UC2.5	R2.23	Yes	The SEMIoTICS platform should allow the SARA solution to discover the IoT devices that are registered in the system. IoT devices deployed by the SARA solution are expected to register themselves into the system using various standard protocols (e.g. LwM2M, MQTT, Bluetooth LE, ZigBee, etc.).	SHOULD
R.UC2.6	R2.22	Yes	The SEMIoTICS platform SHOULD allow the SARA solution to retrieve the resources exposed by registered devices via their object model (i.e. a data structure wherein each element represents a resource, or a group of resources, belonging to a device). The SEMIoTICS platform SHOULD support at least the OMA LWM2M object model.	SHOULD
R.UC2.11	R2.30	Yes	The SEMIoTICS platform SHOULD allow a SARA component to request a group of devices to take/initiate an action (e.g. turn on/off a light bulb).	SHOULD

6.2. Interaction with SEMIoTICS framework

In relation to interconnection between different devices, and the requirements previously described, we followed the KPI2-1, that it is the semantic descriptions for different types of smart objects, that we described on the D5.1. The semantic descriptions for all the types of smart objects are based on W3C Web of Things (WoT) standard; in particular we used the WoT Thing Description standardized format for describing IoT things. Each sensor, actuator or thing from all SEMIoTICS use cases were identified and for each smart object one Thing Description (TD) was provided. We used iotschema.org to semantically annotate each TD.

The goal was to enable smart objects to become interoperable. For the UC2 we focused on the sensors/actuators on board of the Robotic Rollator (RR), such as the Handlebar, Inertial Measurement Unit (IMU), LiDAR, Range Sensors (obstacle sensors) and Motorised Hub Wheels.

Figure 26 shows the SEMIoTICS Semantic Interoperability technologies involved in the implementation of the SARA solution.

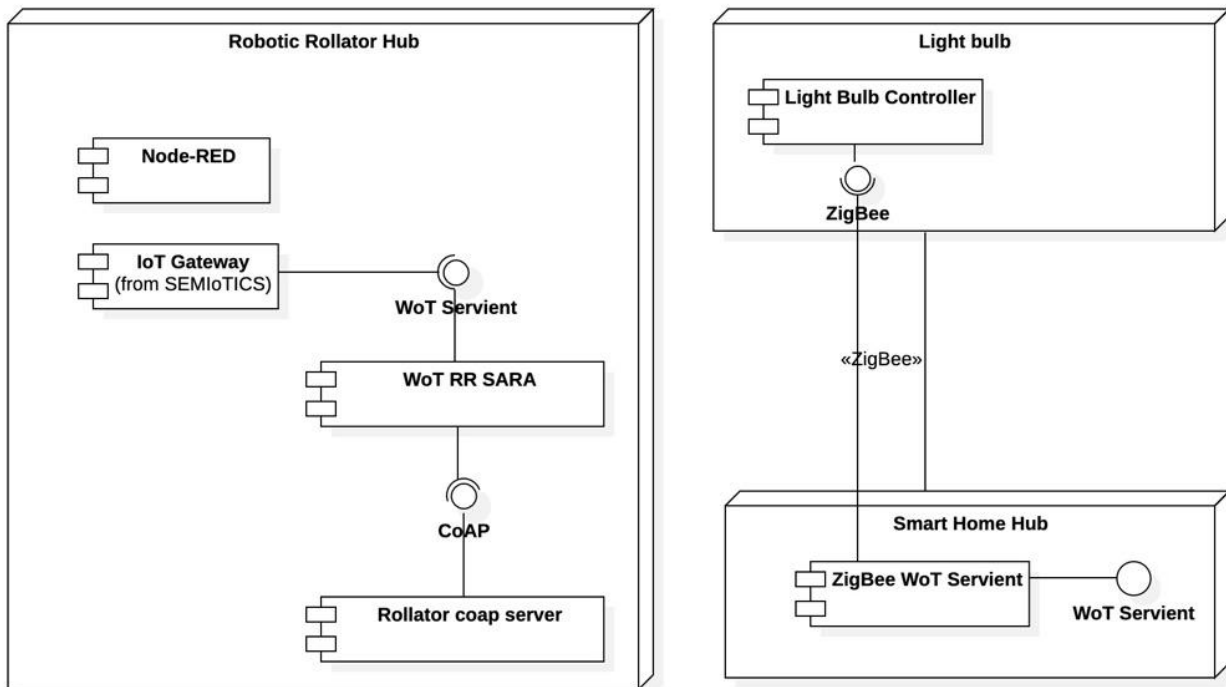


FIGURE 26 SEMANTICS INTEROPERABILITY COMPONENTS IN SARA FIELD DEVICES

6.3. Setup testbed and integration

6.3.1. VIRTUALIZATION OF THE ROBOTIC ROLLATOR

The first step that we did to make the Robotic Rollator available as a “Thing” (in the sense of the WoT standard) it was to expose all the sensors onboard of the RR, through the CoAP protocol. This because CoAP is one of the protocols supported from the WoT (as described in detail in D3.9). To do that, we implemented a CoAP server running on the Raspberry Pi that it is the device acting as Robotic Rollator Hub, i.e. in charge of controlling all the sensors/actuators. The Raspberry Pi used by the Robotic Rollator is a Raspberry Pi 3b with 1Gb RAM, 16GB SSD and Raspian OS.

```
pi@sara:~/CoAPthon $ python coapserverJSONlegL.py
CoAP server started on 0.0.0.0:5555
['/obstacleC', '/obstacleL', '/', '/motorL', '/obstacleR', '/legL']
```

Following an example of REQUEST to the CoAP server previously shown:

```
pi@sara:~/CoAPthon $ python coapclient.py -o GET -p coap://127.0.0.1:5555/obstacleR
Source: ('127.0.0.1', 5555)
Destination: None
Type: ACK
MID: 5706
Code: CONTENT
Token: Qq
Payload:
value: 3.23 [m]
```

FIGURE 27 A SENSOR VALUE RETURNED BY THE COAP SERVER FO THE ROBOTIC ROLLATOR

After that, we created the node-wot servient; this one allows to show the RR as a Thing. To do that we utilized the ThingWeb's node-wot implementation of the WoT standard⁵.

We create a TD for the RR, linking all the sensors to the related CoAP connection; so that, if we want the value of the obstacleR (obstacle sensor put on the right front on the RR), the node-wot servient provide a GET REQUEST to the related CoAP link, in this case *"coap://127.0.0.1:5555/obstacleR"*.

To summarise we developed two main programs running on the RR:

- the CoAP server, that exposed all the sensors compliant with the CoAP protocol and
- the node-wot servient, that made the RR as a Thing in the WoT, for this reason we named it as *"WoT_RR_SARA"*.

If both those programs are running properly on the RR, it is possible to open a browser in a PC connected on the same networks, pointing to the RR's IP address followed by the port 8080 (where the WoT Servient is listening for incoming requests) and all the sensors are been shown as a property of the RR (figure 28).

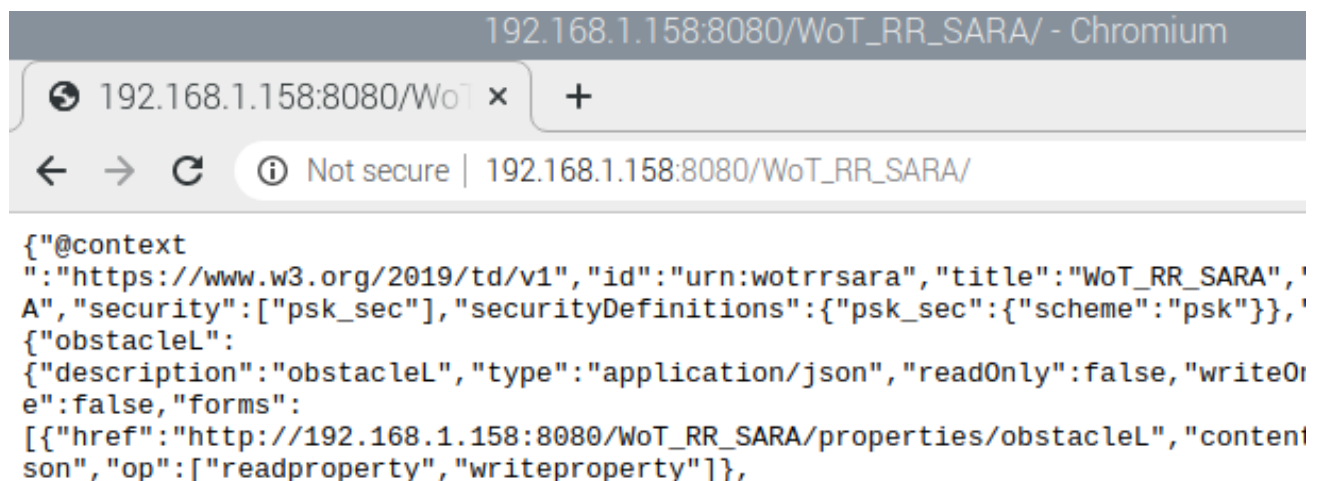


FIGURE 28 THE TD OF ROBOTIC ROLLATOR AS RETURNED BY THE WOT_RR_SARA SERVIENT

As it is possible to see from figure 28, the RR is described as a Thing and exposed all the sensors following the iotschema.org, that it was our scope.

If we wish to get the data of a single sensor, we need to add *"/properties/obstacleR"* to the completed link previously explained as following:

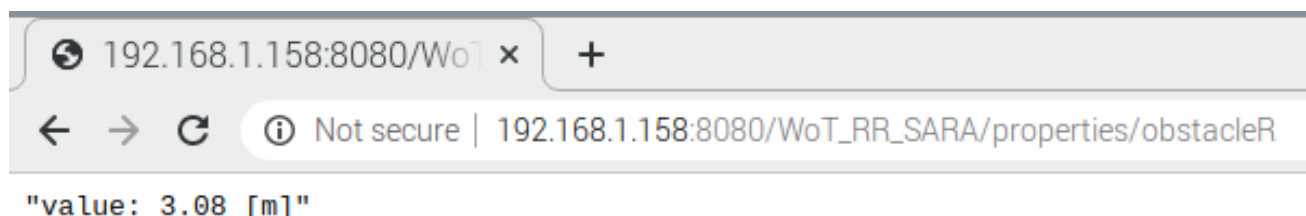


FIGURE 29 A SENSOR VALUE RETURNED BY THE WOT_RR_SARA SERVIENT

In addition to that, we deployed the Semantic-IoT-Gateway developed by Siemens in the context of the SEMIoTICS project. Using this component, it is possible to make the Robotic Rollator available as a node within a Node-RED instance running on the Robotic Rollator hub. This enable the possibility to use the Node-

⁵ <https://github.com/eclipse/thingweb.node-wot/>

RED graphical environment to develop node.js applications that make use of the vitalized sensors and actuators of the Robotic Rollator.

In the figure 30 it is shown a simple Node-RED flow that visualizes the value of the obstacleR sensor every 10 seconds by means of a GET request.

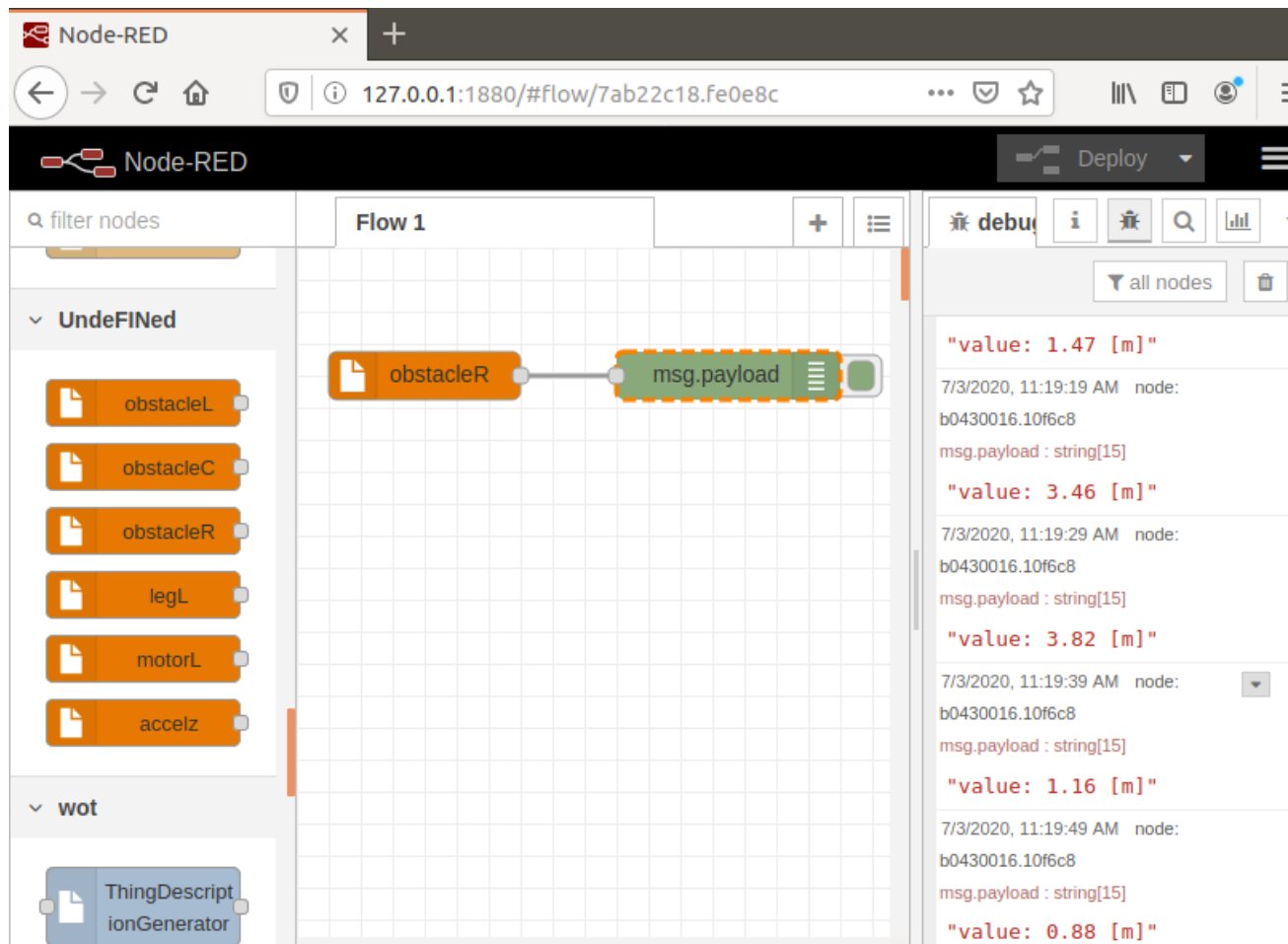


FIGURE 30 A SIMPLE NODE-RED FLOW USING A SENSOR OF THE ROBOTIC ROLLATOR

To the date, due to the impossibility to access the actual Robotic Rollator because of the Covid-19 pandemics, all the steps previously listed in detail have been carried out with a Raspberry Pi simulator. All the validations steps will be repeated with the actual physical device as soon as it will be possible to access again the ENG premise where the Robotic Rollator is.

The data used for this first validation round comes from traces recorded with the actual rollator before the Covid-19 lock down.

6.3.2. VIRTUALIZATION OF ZIGBEE LIGHT BULBS

The Smart Home Hub is realized using a Raspberry 4. The connectivity with ZigBee devices enabled by an XBee module connected to the Raspberry via USB. The light bulb used for the validation is a Philips Hue lamp (figure 30).

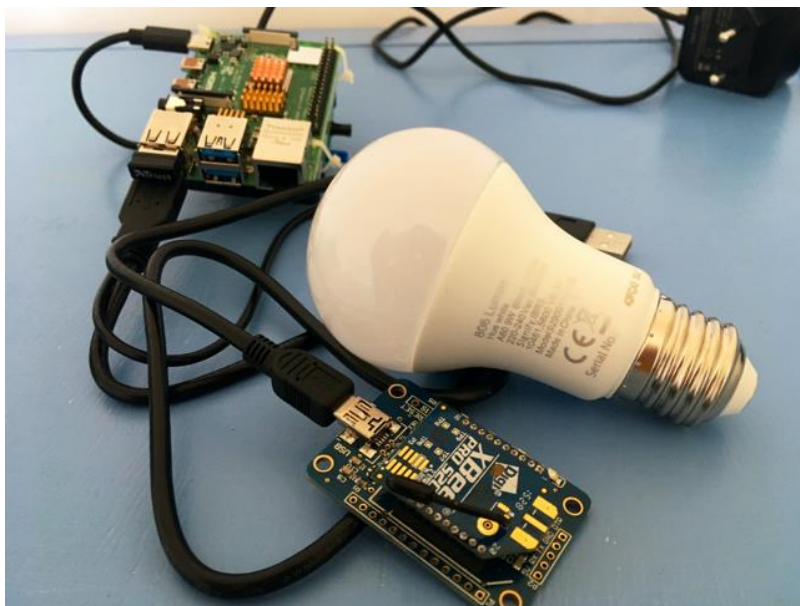


FIGURE 31 THE HARDWARE USED FOR THE VALIDATION OF THE SMART HOME HUB

The WoT Servient is implemented using the SANE Web of Things Servient⁶. The SARA ZigBee Servient uses DigiKey's library⁷ to control the XBee module.

The SARA ZigBee Servient creates a http binding for the Hue hence it is possible to control the light bulb using standard REST call over http for turning the light on (e.g. `http://10.0.1.29:8080/ZigBeeBulb/actions/on`) and off (e.g. `http://10.0.1.29:8080/ZigBeeBulb/actions/off`).

⁶ <https://github.com/sane-city/wot-servient>

⁷ <https://www.digi.com/resources/documentation/digidocs/90001438/Default.htm>

7. OVERALL KPIS AND REQUIREMENTS

7.1. Related KPIS

In this sub section we list the SEMIoTICS 'KPIs [7] related to the sub use described in this section 4. Also, we describe the role of this sub use case in this KPI.

SEMIoTICS' KPI ID	KPI description	Role in this KPI
KPI-4.6	Development of new security mechanisms/controls	From an NFV perspective, SFC is leveraged to guarantee security procedures for each kind of traffic in UC2. This is done by concatenating different security enforcers (firewalls, Intrusion Detection Systems, Honeypots) and forcing traffic to travers them. As each element is configured with specific security rules according to the expected traffic, only authorized packets are expected to go through to the services' endpoints.
KPI-5.2	Service Function Chaining (SFC) of a minimum 3 VNFs	This KPI aims at the orchestration of SFC able to provide security by the chaining of at least 3 VNFs. That is, from a centralized position in the SEMIoTICS architecture, the NFV component should be able to build and configure the SFC for each kind of traffic.
KPI-6.1	Reduce manual interventions required for bootstrapping of smart object in each use case domain by at least 80%.	<p>The bootstrapping service for the smart objects involves the availability of other functional blocks, e.g. at the networking level the VNFs and SFC that allow to forward the traffic from the smart objects towards the backend.</p> <p>An implementation of these functional blocks, at the networking level, in the form of VNFs automates its availability for the bootstrapping process. In this regard, note that the NFV MANO automatizes the creation of the VNFs and its deployment on top of the NFVI in the form of VMs. Also, it automatizes the configuration, software installation and programs executions in the VM at boot time.</p> <p>Therefore, such operations are expected to eliminate user intervention completely.</p>

7.2. Related framework requirements

In this section we consider the set of requirements that were described in deliverable D2.3 [8] within the context of the SEMIoTICS's framework. Thereby, we list the subset of requirements from D2.3 that are relevant for this sub use case and we indicate in which part of the functional block architecture they are needed.

SEMIoTICS' Req-ID	Req-ID description	Requirement context
-------------------	--------------------	---------------------

General platform requirements		
R.GP.2	Scalable infrastructure due to the fast-paced growth of IoT devices.	The architecture tackled herein is highly scalable. Thanks to the NFV framework, more virtual computing and storage resources can be easily allocated when needed for the scalability purposes.
Network layer and Backend/Cloud Layer Requirements		
R.NL.1/R.BC.1	Controller Node requirement: At least 6 CPU cores and 32 GB RAM.	This is in line with the requirements that have already reported above e.g. the OpenStack controller or the OSM, see section 4.4. Note that, in addition, it is required that the controller nodes have around 100 GB of storage memory, see section 4.4.
R.NL.2/R.BC.2	Controller Node requirement: At least 2 Network interfaces.	This is a requirement applicable to the computer holding the controller, e.g. the OpenStack controller, see section 4.4.
R.NL.3/R.BC.3	Controller Node Requirement: Linux OS.	The controller nodes at the NFV MANO context, e.g. the OpenStack controller, require Linux OS.
R.NL.5/R.BC.5/ R.BC.6/ R.BC.7	Hypervisor Nodes Requirement: At least 4 CPU cores and 8 GB RAM, at least 2, 1Gbps Network interfaces.	This is fulfilled or in line with the requirements for the compute nodes of the NFVI, see section 4.4.
R.NL.6/R.BC.8/ R.BC.9	Hypervisor Nodes: KVM and Linux Containers (LXD) must be supported by the Hypervisor Linux OS.	This is supported in the compute nodes that form the NFVI, e.g. at the cloud level.

IoT Security and Privacy Requirements		Evaluation	Reference	Status ^[1]
Req. ID	Description			

R.S.1	The confidentiality of all network communication MUST be protected using state-of-the-art mechanisms.	SDN connection with ovs switches by enabling SSL, + Communication between Pattern Orchestrator and Pattern Engines with SSL	D3.7 and Section 3.2.2, Section 3.4.2	achieved
R.S.2	Authentication and authorization of the stakeholders MUST be enforced by the Network controller, e.g. through access and role-based lists for different levels of function granularities (overlay, customized access to service, QoS manipulation, etc.)	The SDN Security Manager can provide authorization to the users entering the controller	Section 3.3.1 and D3.2	achieved
R.S.3	Sensors SHALL be identifiable (e.g. by a TPM module/smartcard) and authenticated by the gateway.	/	Will be discussed in D5.10	deferred
R.S.20	Cloud platforms MUST be protected by a firewall against network-based attacks.	/	Will be discussed in D5.10	deferred
R.P.1	The collection of raw data MUST be minimized.	/	Will be discussed in D5.10	deferred
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).	/	Will be discussed in D5.10	deferred
R.P.3	Storage of data MUST be minimized.	/	Will be discussed in D5.10	deferred
R.P.4	A short data retention period MUST be enforced and maintaining data for longer than necessary avoided	Attribute-Based Encryption can be used for enforcing a maximum access-time to stored data	Sub-usecase 3	achieved

R.P.5	As much data as possible MUST be processed at the edge in order to hide data sources and not reveal user related information to adversaries (e.g. user's location).	With the help of local analytics this can be facilitated	Will be discussed in D5.10	deferred
R.P.6	Data MUST be anonymized wherever possible by removing the personally identifiable information in order to decrease the risk of unintended disclosure.	/	Will be discussed in D5.10	deferred
R.P.7	Data granularity MUST be reduced wherever possible, e.g. disseminate a location-related information (i.e. area) and not the exact address	/	Will be discussed in D5.10	deferred
R.P.8	Data MUST be stored in encrypted form.	ABE as described in Sub-usecase 3, is used for encrypting user-specific data	Sub-usecase 3	achieved
R.P.9	Repeated querying for specific data by applications, services, or users that are not intent to act in this manner SHALL be blocked.	The Policy Enforcement Points (PEP) will be seeing all requests and can thus detect and mitigate such attempts	Will be discussed in D5.10	deferred
R.P.10	Wherever possible, information over groups of attributes or groups of individuals SHALL be aggregated	/	Will be discussed in D5.10	deferred
R.P.11	The data principal SHALL be sufficiently informed regarding which data are collected, processed, and disseminated, and for what purposes	/	Will be discussed in D5.10	deferred
R.P.12	During all communication and processing phases logging MUST be performed to enable the examination that the system is operating as promised	/	Will be discussed in D5.10	deferred

R.P.13	The user SHALL be able to control the privacy mechanisms (i.e. redemption period, data granularity and dissemination, and anonymization technique)	This can be enforced by using Attribute-Based Encryption as described in Sub-usecase 3. The user is then able to enforce his/her privacy-concerns and wishes by encrypting the user specific data with the help of ABE.	Sub-usecase 3	achieved
--------	--	---	---------------	----------

Finally, a list with all the use case 2 specific requirements and the related sub-use case is presented in Appendix A.

8. CONCLUSION

This deliverable presented the initial validation of the SEMIoTICS technologies in the context of the development of SARA a solution in the domain of Ambient assisted living.

This document identified the challenges subsumed by the implementation of two of the SARA assistive tasks (*Falls Management Assistive Task* and *Gait Analysis Assistive Task*).

This document presented the SEMIoTICS-based implementation of SARA by four points of view (sub use cases) each focusing one of main areas addressed by the SEMIoTICS project: (a) security, privacy, dependability and safety; (b) IoT semantic interoperability; (c) embedded intelligence and (d) automated trustworthy network management.

Being published halfway the development of the SEMIoTICS-based SARA this document does not yet presented full evaluation of the benefits gained by the adoption of the SEMIoTICS framework. However, it also anticipates some encouraging results.

The main developments that remain to be done concerns:

- the connection, via the networking infrastructure presented in section 4 and whose validation is presented in Appendix A, of the field devices presented in the section 6 with the SEMIoTICS backend services deployed at the infrastructure made available by the BLS partner.
- the integration of the Gait Analysis AT Web App with the security mechanisms presented in section 5.
- the further training of the time series clustering algorithm with additional data samples collected by means of the robotic rollator (restrictions due to covid-19 allowing)
- the development of parts of the Gait Analysis pipeline (e.g. the time series segmentation stage) using the NODE-Red environment and the SEMIoTICS IoT Gateway described in section 6.

These developments will allow to complete the evaluation of the SEMIoTICS framework in the context of the SARA-Health use case and draw the conclusions in the forthcoming deliverable D5.10 - "Demonstration and validation of SARAHealth (Cycle 2)".

9. REFERENCES

1. ETSI, "ETSI.org: Network Functions Virtualisation (NFV); Architectural Framework," 2013. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf. [Accessed July 2020].
2. J. Serra et al. "Network Functions Virtualization for IoT (final)", SEMIoTICS deliverable D3.8, February 2020.
3. "Open Source MANO (OSM)" [Online]. Available: <https://osm.etsi.org/>. [Accessed July 2020].
4. "OpenStack" [Online]. Available: <https://www.openstack.org/>. [Accessed July 2020].
5. "OSM installation". [Online]. Available: <https://osm.etsi.org/docs/user-guide/03-installing-osm.html>. [Accessed July 2020].
6. "OpenStack Ansible, Train version" [Online]. Available: <https://docs.openstack.org/openstack-ansible/train/>. [Accessed July 2020].
7. F. Klement et al. "SEMIOTICS KPIs and Evaluation Methodology". SEMIoTICS deliverable D5.1, January 2020.
8. M. Falchetto et al. "Requirements specification of SEMIoTICS framework", SEMIoTICS deliverable D2.3, November 2019.

APPENDIX A - UC2 SEMIOTICS REQUIREMENTS

This appendix maps SEMIoTICS requirements to the sub use cases presented in this deliverable. Table A.1 indicates for each requirement which is the sub use case addressing it.

TABLE A.1 SEMIoTICS requirement evaluated by each sub use case.

Req-ID	Description	SU1	SU2	SU3	SU4
R.UC2.1	The SEMIoTICS platform SHOULD support time- and safety-critical requirements by allowing SARA application logic to be deployed on resource-constrained edge gateways (e.g. smartphones, vehicles, mobile robots). SEMIoTICS platform functionalities SHOULD be locally available even in case of failure of communication with the SEMIoTICS cloud nodes.	X			
R.UC2.2	The SEMIoTICS platform SHOULD support the SARA solution to manage the trade-off between different requirements (e.g. reliability, power consumption, latency, fault-tolerance) by allowing both SARA application logic and platform features to be distributed over a cluster of gateways (SARA Hubs).				
R.UC2.3	The SEMIoTICS platform SHOULD guarantee proper connectivity between the various components of the SARA distributed application. The SARA solution is a distributed application not only because it uses different cloud services (e.g. AREAS Cloud services, AI services) from different remote computational nodes, but also because the SARA application logic itself is distributed across various edge nodes (SARA Hubs).		X		

R.UC2.4	<p>The SEMIoTICS platform SHOULD provide services to synchronize and coordinate the activities of the various components of the SARA application. The SARA components synchronize and coordinate their activities by relying on the "shared event pools" paradigm.</p> <p>Through a "shared event pool":</p> <ul style="list-style-type: none"> • All devices contribute events to the pool – e.g. "Patient has requested escort to kitchen", "Patient is in distress". • The AREAS Cloud Service (ACS) manages consistency of data in the pool. • The pool ensures that all devices share the same operational context. • The pool allows multiple devices to respond independently to the same events (redundancy). • The design & development of the high-level system functions is simplified: abstracts away from low-level device implementation & communication details 				
R.UC2.5	The SEMIoTICS platform should allow the SARA solution to discover the IoT devices that are registered in the system. IoT devices deployed by the SARA solution are expected to register themselves into the system using various standard protocols (e.g. LwM2M, MQTT, Bluetooth LE, ZigBee, etc.).				X
R.UC2.6	The SEMIoTICS platform SHOULD allow the SARA solution to retrieve the resources exposed by registered devices via their object model (i.e. a data structure wherein each element represents a resource, or a group of resources, belonging to a device). The SEMIoTICS platform SHOULD support at least the OMA LWM2M object model.				X
R.UC2.7	The SEMIoTICS platform SHOULD notify periodically the SARA solution about the state of the resources hosted by registered IoT field devices.				
R.UC2.8	The SEMIoTICS connectivity SHOULD keep track of the field device connectivity state (e.g. to detect anomalies, but also required for higher-level (cognitive) control algorithms).				

R.UC2.9	The SEMIoTICS platform SHOULD allow the SARA solution to persist and, subsequently, retrieve the notifications received by IoT field devices. The SARA components expect the values to be stored as key-value pairs within collections belonging to data stores.				
R.UC2.10	The SEMIoTICS platform SHOULD allow the SARA components (e.g. SARA Hubs) to query and aggregate (e.g. to average) the values of a resource (e.g. current measured temperature) hosted by a group of field devices. The SARA solution defines a group of devices by specifying filtering criteria over the set of registered devices. These filtering criteria may concern the location of the device, the Quality of Service (QoS) offered (e.g. its precision) and their ownership. The SARA components MAY submit three classes of queries: one-time, periodic, and conditional. One-time queries are evaluated once. Their evaluation starts at the time of the submission and their results are retrieved asynchronously by the application. With periodic query, a SARA component requests periodic evaluation and reporting. With conditional queries, the application is notified whenever a Boolean condition, defined over the value computed by the query, changes its truth value.				
R.UC2.11	The SEMIoTICS platform SHOULD allow a SARA component to request a group of devices to take/initiate an action (e.g. turn on/off a light bulb).				X
R.UC2.12	The SEMIoTICS platform SHOULD allow SARA components to delegate to the platform the computation of complex functions over the data received by field devices. These computations may result either in the generation of higher-level observation events (e.g. significant Patient events abstracted from sensor data) towards the ACS or in sensors configuration parameters (including actuators command). The SARA components MAY specify computations either as Dataflow or as Finite State Machine.				

R.UC2.13	The SEMIoTICS platform MAY allow the SARA solution to synchronize the clocks of the mobile robotic devices hosting the components of the solution. This service, hence, would enable the SARA solution to properly order the events generated by its components (e.g. to generate a single coherent map of the environment starting with data from multiple hubs). Moreover, the SEMIoTICS platform MAY allow the SARA components to delegate the management of timers relying on the synchronized clocks.				
R.UC2.14	The SEMIoTICS platform MAY allow the SARA solution to access information concerning the location of mobile robotic nodes and to maintain SARA specific location information (e.g. collected map data). This would allow the SARA hub or cloud to send to robotic devices high level navigation related commands (e.g. "go to location").				
R.UC2.15	<p>The SEMIoTICS platform SHOULD provide low latency connectivity between the SARA hubs and cloud services (i.e. AREAS cloud services and AI services) to allow offloading of near real-time computation intensive tasks to the cloud. Examples include:</p> <ul style="list-style-type: none"> the robotic assistant (RA) employing AI services to analyse Patient's speech (audio) and body language (video) to identify significant events – e.g. "Patient requests an escort", "Patient asks where his glasses are" the robotic rollator (RR) exploiting AI Services to analyse Patient's gait and posture to identify significant events – e.g. "Patient has fallen". mobile robotic Devices (RA/RR) exploiting cloud resources for simultaneous localization and mapping (SLAM) <p>Therefore, SARA hubs need to send with minimal delay:</p> <ul style="list-style-type: none"> raw range data (e.g. from Lidar sensors) to identify proximal objects/objects, real-time audio stream for speech analysis, and real-time raw video stream (object/people recognition, gesture recognition, posture analysis). 		X		
R.UC2.16	The SEMIoTICS platform SHOULD support secure bootstrap, configuration and diagnostic of SARA hubs and devices.			X	

R.UC2.17	The SEMIoTICS connectivity SHOULD support real time exchange of raw sensor data among sensors/actuators and SARA Hubs.		X		
R.GSP.8	The SARA system MUST periodically log (Hub monitored) Patient bio-medical data to the ACS.				
R.GSP.9	<p>The SARA system SHALL provide robust mechanisms to protect Patient-related data - i.e. to:</p> <ul style="list-style-type: none"> · Authenticate the sources of Patient data – in particular: to avoid the case that data monitored from one Patient is incorrectly ascribed to another. · Prevent unauthorised access to or manipulation of stored Patient data. · Secure the transmission of Patient data. <p>Such data includes (but is not limited to):</p> <ul style="list-style-type: none"> · System configuration data for the Patient (UC 1) · Daily activity rules for the Patient (UC 2) · Monitored daily activity data for the Patient (UC 2) · Audio-visual streams transmitted during telepresence (UC 6 & 7) 			X	
R.GSP.10	The SARA system MUST fully comply with all relevant Italian laws governing the privacy, security and storage of sensitive Patient health-related data.			X	

APPENDIX B - NETWORK INFRASTRUCTURE VALIDATION

It is worth mentioning that a VPN has been configured at CTTC to let external users to access the NFV testbed described above. In Figure B.1, we show that the execution of the VPN connection instruction, based on a configuration file, is successful. Then, in Figure B.2 and Figure B.3, it is shown that the VPN allows to access the OSM and OpenStack GUIs, respectively. Moreover, below it will be shown, that the external VPN users can use properly the OSM and OpenStack services. To this end, a generic VNF and they corresponding Network Service (NS) will be created, onboarded to OSM and instantiated on top the NFVI, which is managed by the OpenStack. Observe that in OSM a VNF is always within the context of NS, which can be interpreted as a higher layer entity. In fact, a NS can contain multiple VNFs.

```
jserra@jserra-Latitude-5480:~$ sudo openvpn --config semiotics_vpn.ovpn
Thu Jun 25 16:50:11 2020 OpenVPN 2.4.4 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4]
Thu Jun 25 16:50:11 2020 library versions: OpenSSL 1.1.1 11 Sep 2018, LZO 2.08
Thu Jun 25 16:50:11 2020 Outgoing Control Channel Authentication: Using 256 bit message
Thu Jun 25 16:50:11 2020 Incoming Control Channel Authentication: Using 256 bit message
Thu Jun 25 16:50:11 2020 TCP/UDP: Preserving recently used remote address: [AF_INET]84.
Thu Jun 25 16:50:11 2020 Socket Buffers: R=[212992->212992] S=[212992->212992]
Thu Jun 25 16:50:11 2020 UDP link local: (not bound)
Thu Jun 25 16:50:11 2020 UDP link remote: [AF_INET]84.88.61.201:3000
Thu Jun 25 16:50:11 2020 NOTE: UID/GID downgrade will be delayed because of --client, --
Thu Jun 25 16:50:11 2020 TLS: Initial packet from [AF_INET]84.88.61.201:3000, sid=0ce72
Thu Jun 25 16:50:11 2020 VERIFY OK: depth=1, CN=SMARTECH CA
Thu Jun 25 16:50:11 2020 VERIFY KU OK
Thu Jun 25 16:50:11 2020 Validating certificate extended key usage
Thu Jun 25 16:50:11 2020 ++ Certificate has EKU (str) TLS Web Server Authentication, ex
Thu Jun 25 16:50:11 2020 VERIFY EKU OK
Thu Jun 25 16:50:11 2020 VERIFY OK: depth=0, CN=openvpn-server
Thu Jun 25 16:50:11 2020 Control Channel: TLSv1.3, cipher TLSv1.3 TLS_AES_256_GCM_SHA38
Thu Jun 25 16:50:11 2020 [openvpn-server] Peer Connection Initiated with [AF_INET]84.88
Thu Jun 25 16:50:12 2020 SENT CONTROL [openvpn-server]: 'PUSH_REQUEST' (status=1)
Thu Jun 25 16:50:12 2020 PUSH: Received control message: 'PUSH_REPLY,route 10.1.14.2 25
10.8.0.21,peer-id 4,cipher AES-256-GCM'
Thu Jun 25 16:50:12 2020 OPTIONS IMPORT: timers and/or timeouts modified
Thu Jun 25 16:50:12 2020 OPTIONS IMPORT: --ifconfig/up options modified
Thu Jun 25 16:50:12 2020 OPTIONS IMPORT: route options modified
Thu Jun 25 16:50:12 2020 OPTIONS IMPORT: peer-id set
Thu Jun 25 16:50:12 2020 OPTIONS IMPORT: adjusting link_mtu to 1624
Thu Jun 25 16:50:12 2020 OPTIONS IMPORT: data channel crypto options modified
Thu Jun 25 16:50:12 2020 Data Channel: using negotiated cipher 'AES-256-GCM'
Thu Jun 25 16:50:12 2020 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 2
Thu Jun 25 16:50:12 2020 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 2
Thu Jun 25 16:50:12 2020 ROUTE_GATEWAY 192.168.0.1/255.255.255.0 IFACE=wlp2s0 HWADDR=68
Thu Jun 25 16:50:12 2020 TUN/TAP device tun0 opened
Thu Jun 25 16:50:12 2020 TUN/TAP TX queue length set to 100
Thu Jun 25 16:50:12 2020 do ifconfig, tt->did_ifconfig_ipv6_setup=0
Thu Jun 25 16:50:13 2020 /sbin/ip link set dev tun0 up mtu 1500
Thu Jun 25 16:50:13 2020 /sbin/ip addr add dev tun0 local 10.8.0.22 peer 10.8.0.21
Thu Jun 25 16:50:13 2020 /sbin/ip route add 10.1.14.2/32 via 10.8.0.21
Thu Jun 25 16:50:13 2020 /sbin/ip route add 172.113.0.0/16 via 10.8.0.21
Thu Jun 25 16:50:13 2020 /sbin/ip route add 10.8.0.1/32 via 10.8.0.21
Thu Jun 25 16:50:13 2020 GID set to nogroup
Thu Jun 25 16:50:13 2020 UID set to nobody
Thu Jun 25 16:50:13 2020 WARNING: this configuration may cache passwords in memory -- u
Thu Jun 25 16:50:13 2020 Initialization Sequence Completed
```

FIGURE B.1 VPN THAT ENABLES EXTERNAL USERS TO ACCESS THE CTTC's NFV TESTBED.

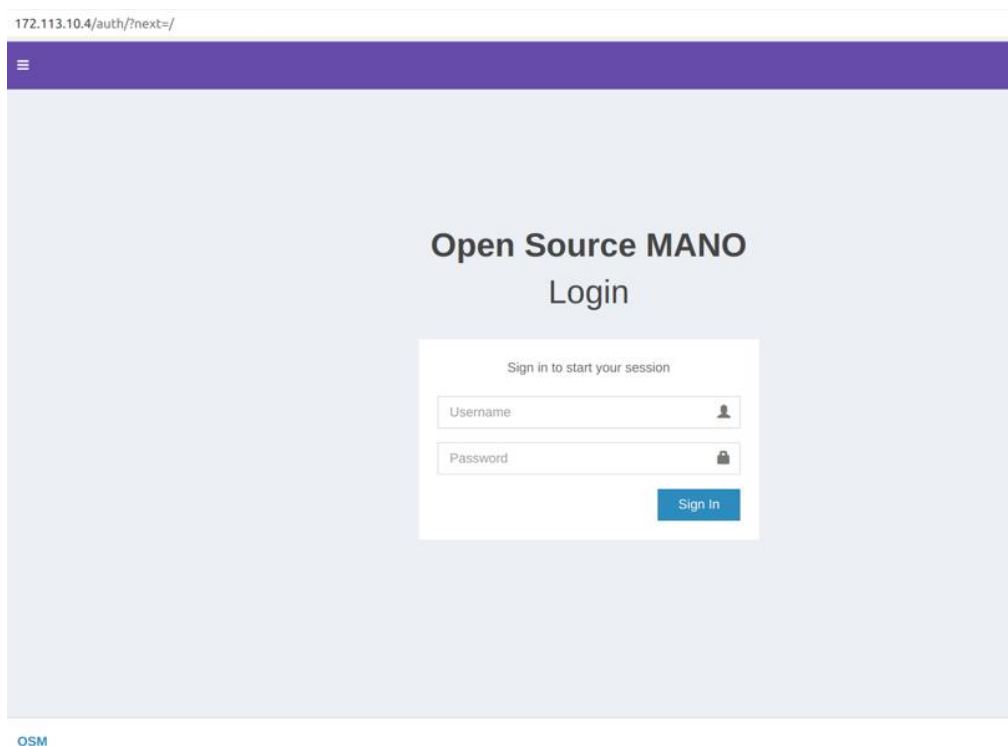


FIGURE B.2 VPN ALLOWS ACCESS TO OSM AT CTTC's NFV TESTBED.



FIGURE A.3 VPN ALLOWS ACCESS TO OPENSTACK AT CTTC's NFV TESTBED.

Next, as it was mentioned above, a generic VNF and its corresponding NS are created, onboarded to OSM and instantiated on top of the NFVI. This shows that the external VPN users can access properly the NFV testbed and use its NFV services, i.e. OSM and OpenStack. To this end, we use the OSM command line interface (cli). Thereby, first it is shown in Figure 6 that the external users can enter via ssh to the VM that contains the OSM.

```
jserra@jserra-Latitude-5480:~$ ssh iotworld@172.113.10.4
iotworld@172.113.10.4's password:
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-106-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Thu Jun 25 14:54:54 UTC 2020

System load:  0.68           Users logged in:           0
Usage of /:   27.0% of 90.09GB IP address for enp5s0:      172.113.10.4
Memory usage: 35%           IP address for lxdbr0:      10.181.16.1
Swap usage:   0%            IP address for docker0:     172.17.0.1
Processes:    302           IP address for docker_gwbridge: 172.18.0.1

=> There is 1 zombie process.

 * "If you've been waiting for the perfect Kubernetes dev solution for
   macOS, the wait is over. Learn how to install Microk8s on macOS."

   https://www.techrepublic.com/article/how-to-install-microk8s-on-macos/

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

13 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Thu Jun 25 11:26:01 2020 from 172.113.10.1
iotworld@semiotics-osm-big:~$
```

FIGURE B.4 SSH TO THE VM THAT CONTAINS THE OSM.

After entering to the VM that hosts the OSM, we will follow a set of steps to create, instantiate and terminate the generic VNF and the generic NS. Note that, the end user only needs to interact with the OSM, which communicates internally with the OpenStack. Moreover, the interaction with the OSM is through a set of configuration files that define the computing, storage and networking features of the VNF and the NS. These configuration files are so-called VNF descriptor (VNFd) and Network Service descriptor (NSd), respectively. Thereby, the next steps will be followed below to create, instantiate and terminate the VNF and NS.

- Create generic NSd and VNFd.
- Generate VNF/NS packages.
- Onboard the VNF/NS packages to OSM library.
- Instantiate the NS (and the VNF).
- Check that we can access the VM created for the VNF instantiation.
- Terminate the NS.

Create generic NSd (and VNFd).

- Create VNFd, NSd folder structure

First, OSM needs to create a folder structure for the NSd, VNFd and its related files, such as the cloud init, which defines the initial configuration of the VM that will hold the VNF. This is accomplished by using a shell script provided by OSM, it is called “generate_descriptor_pkg”. In Figure 7, we show how that folder structure is created correctly.

```
iotworld@semitotics-osm-big:~/osm/vnfd$ generate_descriptor_pkg.sh -t vnfd --image ubuntu18-minimal -c generic
iotworld@semitotics-osm-big:~/osm/vnfd$ ls
generic_vnfd
iotworld@semitotics-osm-big:~/osm/vnfd$ cd generic_vnfd/
iotworld@semitotics-osm-big:~/osm/vnfd/generic_vnfd$ ls
README charms cloud_init generic_vnfd.yaml icons images scripts
iotworld@semitotics-osm-big:~/osm/vnfd/generic_vnfd$
```

```
iotworld@semitotics-osm-big:~/osm/nsd$ ls
iotworld@semitotics-osm-big:~/osm/nsd$ generate_descriptor_pkg.sh -t nsd -c generic
iotworld@semitotics-osm-big:~/osm/nsd$ ls
generic_nsd
iotworld@semitotics-osm-big:~/osm/nsd$ cd generic_nsd/
iotworld@semitotics-osm-big:~/osm/nsd/generic_nsd$ ls
README charms generic_nsd.yaml icons ns_config scripts vnf_config
iotworld@semitotics-osm-big:~/osm/nsd/generic_nsd$
```

FIGURE B.5 CREATION OF THE FOLDER STRUCTURE NEEDED TO CREATE VNFd and NSd.

- Edit VNFd

After creating the folder structure for the VNFd and NSd, we can edit the VNFd, which is a yaml configuration file. In Figure A.6 we present the snapshot of the yaml file that we have used to specify the VNFd. In the sequel, the important parts of this file are discussed. 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. 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, 1 GB of RAM and no persistent storage, as the “image” that we discuss next defines enough storage for this test. The tag “image” indicates the image that will be used to create the VM, in our case we will have an Ubuntu OS. The “cloud-init-file” indicates the cloud init file that will be used by the VM. The snapshot for this cloud init file is actually described below. The “interface” tag within the “vdu” tag specifies the interfaces for the vdu.

- Cloud init file specification

This file is the one that specifies the initial configuration that we aim for the VM that will host the generic VNF. Note that its name is specified in the vnfd, as commented above. In Figure A.7, 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”. Finally, 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 VNF.

```
vnfd:vnfd-catalog:
  vnfd:
  - id: generic_vnfd
    name: generic_vnfd
    short-name: generic_vnfd
    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>

    # Management interface
    mgmt-interface:
      cp: vnf-cp0

    # Atleast one VDU need to be specified
    vdu:
    # Additional VDUs can be created by copying the
    # VDU descriptor below
    - id: generic_vnfd-VM
      name: generic_vnfd-VM
      description: generic_vnfd-VM
      count: 1

      # Flavour of the VM to be instantiated for the VDU
      vm-flavor:
        vcpu-count: 1
        memory-mb: 1024
        storage-gb: 0

      # Image including the full path
      image: 'ubuntu18-minimal'

      # Cloud init file
      cloud-init-file: 'generic'

    interface:
    # Specify the external interfaces
    # There can be multiple interfaces defined
    - name: eth0
      type: EXTERNAL
      virtual-interface:
        type: VIRTIO
        external-connection-point-ref: vnf-cp0
```

FIGURE B.6 VNFD THAT DESCRIBES A GENERIC VNF.

- Edit NSd

Next, in Figure B.8 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 NS. The tag “constituent-vnfd” indicates which VNFs are part of the NS. In our case we have just the generic VNF, whose identification is “generic_vnfd” and is specified through the tag “vnfd-id-ref”. Note that, in the vnfd the “id” tag has to 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 generic VNF as the tag “vnd-id-ref” is set to “generic_vnfd”.



```
GNU nano 2.9.3 generic
#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 AAAAB3NzaC1yc2EAAAADAQABAAQCuQr8qPujnrFSfp0AmMhMGpnylD1wsAKn+HUr9mY6RorsQ6V
  - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDrLzTL2qA8ErEnaRW+zKHjDUDJ5Xhk2wrmeFC+S2ienq2rdg
  - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC4AcxbsG8P4H4sT5x9AcxnVpxoKB8sxXV3MvHCu+2xiuBGqj
runcmd:
- systemctl -w net.ipv4.ip_forward=1
```

FIGURE B.7 CLOUD INIT FILE ASSOCIATED TO THE GENERIC VNF.

```
GNU nano 2.9.3 generic_nsd.yaml

nsd:nsd-catalog:
  nsd:
    - id: generic_nsd
      name: generic_nsd
      short-name: generic_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: generic_vnfd

      vld:
        # Networks for the VNFDs
        - id: generic_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 VNFDs
            # 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: generic_vnfd
              # NOTE: Validate the entry below
              vnfd-connection-point-ref: vnf-cp0
```

FIGURE B.8 NSd ASSOCIATED TO THE GENERIC VNF.

- Generate VNFD/NSd packages.

At this point, the VNFD and NSd are already edited. Thereby, we can generate the NSd, and VNFD packages, which are required in the onboarding process of the OSM. This means, the process of having our NS and VNF packages available in the OSM library. To generate those packages we only need to execute the shell script “generate_descriptor_pkg” provided by the OSM, which needs the NS and VNF folder structure that we created above in Figure 7.

```
iotworld@semiotics-osm-big:~/osm/vnfd$ ls
generic_vnfd
iotworld@semiotics-osm-big:~/osm/vnfd$ generate_descriptor_pkg.sh -t vnfd -N generic_vnfd/
iotworld@semiotics-osm-big:~/osm/vnfd$ ls
generic_vnfd  generic_vnfd.tar.gz
iotworld@semiotics-osm-big:~/osm/vnfd$
```

```

iotworld@semitotics-osm-big:~/osm/nsd$ ls
generic_nsd
iotworld@semitotics-osm-big:~/osm/nsd$ generate_descriptor_pkg.sh -t nsd -N generic_nsd/
iotworld@semitotics-osm-big:~/osm/nsd$ ls
generic_nsd  generic_nsd.tar.gz
iotworld@semitotics-osm-big:~/osm/nsd$ █

```

FIGURE B.9 GENERATION OF VNFd AND NSd PACKAGES.

- Onboard the VNF/NS packages to OSM library.

Next, in Figure B.10 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 VNF and the requirements in terms of computing and networking that it has. Also, they describe the features of the VM that will host the VNF and the initial configuration and software packages installations that we need in the VM. We have called the NSd and VNFd as `generic_nsd` and `generic_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.

- Instantiate the NS.

Then, in Figure B.11 we trigger the instantiation of the NS that we have onboarded in OSM for our generic VNF. This means that OSM will communicate with the OpenStack controller, which creates the VM that will host our VNF 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 generic -nsd_name generic_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@semitotics-osm-big:~/osm$ osm nsd-list
+-----+-----+
| nsd name | id |
+-----+-----+
+-----+-----+
iotworld@semitotics-osm-big:~/osm$ osm vnfd-list
+-----+-----+
| nfpkg name | id |
+-----+-----+
+-----+-----+
iotworld@semitotics-osm-big:~/osm$ osm vnfd-create ./vnfd/generic_vnfd.tar.gz
b0fe4c07-0c8f-4b80-a5b8-25920ebd9538
iotworld@semitotics-osm-big:~/osm$ osm nsd-create ./nsd/generic_nsd.tar.gz
3de46d77-78f5-45bd-8242-0597922b7ea7
iotworld@semitotics-osm-big:~/osm$ osm nsd-list
+-----+-----+
| nsd name      | id |
+-----+-----+
| generic_nsd   | 3de46d77-78f5-45bd-8242-0597922b7ea7 |
+-----+-----+
iotworld@semitotics-osm-big:~/osm$ osm vnfd-list
+-----+-----+
| nfpkg name    | id |
+-----+-----+
| generic_vnfd  | b0fe4c07-0c8f-4b80-a5b8-25920ebd9538 |
+-----+-----+
iotworld@semitotics-osm-big:~/osm$ █

```

FIGURE B.10 ONBOARD VNFD AND NSd PACKAGES TO OSM.

```
iotworld@semiotics-osm-big:~$ osm ns-list
+-----+-----+-----+-----+-----+-----+
| ns instance name | id | date | ns state | current operation | error details |
+-----+-----+-----+-----+-----+-----+
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@semiotics-osm-big:~$ osm ns-create --ns_name generic --nsd_name generic_nsd
Vim account: semiotics_playground_train_vm_001
8fb2ac7a-53d4-4237-8a78-1d7c6e98324d
iotworld@semiotics-osm-big:~$ osm ns-list
+-----+-----+-----+-----+-----+-----+
| ns instance name | id | date | ns state | current operation | error details |
+-----+-----+-----+-----+-----+-----+
| generic | 8fb2ac7a-53d4-4237-8a78-1d7c6e98324d | 2020-06-25T20:10:17 | BUILDING | INSTANTIATING (b295e6e5-b623-493b-ac53-3e3c9f277c99) | 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@semiotics-osm-big:~$ osm ns-list
+-----+-----+-----+-----+-----+-----+
| ns instance name | id | date | ns state | current operation | error details |
+-----+-----+-----+-----+-----+-----+
| generic | 8fb2ac7a-53d4-4237-8a78-1d7c6e98324d | 2020-06-25T20:10:17 | 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@semiotics-osm-big:~$
```

FIGURE B.11 INSTANTIATE THE NS RELATED TO THE GENERIC VNF.

- Check that we can access the VM created for the VNF instantiation.

The NS instantiation creates the VM that holds the VNF, on top of the NFVI. Thereby, it is important to check if we have access to such VM. In order to obtain the IP of the VM we can just execute the openstack command "openstack server list". OSM also provides this IP in the information of the NS instance. In Figure B.12 we show that we can access the VM that holds the generic VNF.


```
jserra@jserra-Latitude-5480:~$ ssh generic@172.113.40.28
The authenticity of host '172.113.40.28 (172.113.40.28)' can't be established.
ECDSA key fingerprint is SHA256:pKnp+EqPs+dWKAqQqs0vxnsQHwwCFvg8+EVD8Jonq9c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.113.40.28' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-1053-kvm x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

0 packages can be updated.
0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

generic@generic-1-generic-vnfd-vm-1:~$ pwd
/home/generic
generic@generic-1-generic-vnfd-vm-1:~$ █
```

FIGURE B.12 CHECK THE ACCESS TO THE VM CREATED TO HOLD THE GENERIC VNF.

- Terminate the NS

Finally, we show that we can terminate the NS, which shows the overall lifecycle of a NS and its associated VNFs. To this end, we use the OSM command “osm ns-delete” and its argument must specify the id of the NS instance that we want to terminate. In Figure B.13 we show that we can terminate the NS instance associated to the generic VNF.

```
iotworld@semitotics-osm-big:~$ osm ns-list
+-----+-----+-----+-----+-----+-----+
| ns instance name | id | date | ns state | current operation | error details |
+-----+-----+-----+-----+-----+-----+
| generic | 8fb2ac7a-53d4-4237-8a78-1d7c6e98324d | 2020-06-25T20:10:17 | 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@semitotics-osm-big:~$ osm ns-delete 8fb2ac7a-53d4-4237-8a78-1d7c6e98324d
Deletion in progress
iotworld@semitotics-osm-big:~$ osm ns-list
+-----+-----+-----+-----+-----+-----+
| ns instance name | id | date | ns state | current operation | error details |
+-----+-----+-----+-----+-----+-----+
| generic | 8fb2ac7a-53d4-4237-8a78-1d7c6e98324d | 2020-06-25T20:10:17 | TERMINATING | TERMINATING (f252f714-eae9-4554-b37c-db5043c5ff10) | 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@semitotics-osm-big:~$ osm ns-list
+-----+-----+-----+-----+-----+-----+
| ns instance name | id | date | ns state | current operation | error details |
+-----+-----+-----+-----+-----+-----+
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@semitotics-osm-big:~$
```

FIGURE B.13 TERMINATE THE NS RELATED TO THE GENERIC VNF.