Title: *Case studies implementation – First version*

Authors: *Sergio Jiménez Gómez (INDRA), Rocío Gómez Robledo (INDRA), Jon Colado García (INDRA), Francisco Parrilla Ayuso (INDRA), Erkuden Rios Velasco (TECNALIA), Eider Iturbe Zamalloa (TECNALIA), Angel Rego Fernandez (TECNALIA), Miguel Angel Anton Gonzalez (TECNALIA), Sarah Tiphaine Ada Noye (TECNALIA), Arnor Solberg (TellU), Franck Fleurey (TellU), Uģis Grīnbergs (BOSC), Modris Greitans (EDI), Oscar Zanutto (ISRAA), Emanuela Capotosto (ISRAA).*

Editor: *TECNALIA*
Reviewers: *Nicolas Ferry (SINTEF) and Anne Gallon (Evidian).*
Identifier: *Deliverable # D1.2*
Nature: *Report*
Date: *09 October 2019*
Status: *Final*
Diss. level: *Public*

The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement n° 780351 (ENACT).

Executive Summary

The objective of the deliverable D1.2 is to describe the development and integration of the First Version (M21) of the ENACT Use Cases environment and scenarios performed within Task 1.2 according to the definitions and requirements described in D1.1.

The Use Cases environment comprises all the IoT infrastructures that are needed to perform Development and Operation of the Use Cases, including IoT and edge devices, the integration platforms and the cloud services. In this deliverable, the results of implementing and integrating the developments from WP2 to WP4 tools in the Use Cases are provided, such as the adaptation and monitoring tools, the security and privacy mechanisms, etc. These results in turn provide feedback to the technical WPs validating and demonstrating the ENACT DevOps Framework.

The three different Use Cases implemented in ENACT are:

- **Intelligent Transport System (ITS):** The purpose is to assess the feasibility of IoT services in the domain of train integrity control, in particular for the maintenance and logistics of the rolling stock and the on-track equipment.
- **Digital Health:** The objective of this Use Case is to provide medical device integration and data transport capabilities to external services such as an alarm centre or an electronic patient journal for stakeholders such as patients, doctors etc., exploiting the potential in IoT, edge and cloud services, adding more devices, actuation, distributed processing and better exploitation of collected data.
- **Smart Building:** The aim is to generate Smart Energy Efficiency applications and Smart Elderly Care applications, which will make use of sensors, actuators and services, in order to ensure the safety of the facilities to perform energy efficiency measures and also to support the care-takers in assessing the wellbeing of users.

Tecnalia is leading this task and is responsible for the Smart Building demonstrator, Indra and TellU are responsible for the development and integration of the ITS and eHealth demonstrators, respectively. BOSC and EDI contribute to the implementation of the ITS case study and facilitate the rail test infrastructure and demonstrations.

The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement n° 780351 (ENACT).

Members of the ENACT consortium:

SINTEF AS	Norway
BEAWRE DIGITAL SL	Spain
EVIDIAN SA	France
INDRA Sistemas SA	Spain
Fundacion Tecnalia Research & Innovation	Spain
TellU IOT AS	Norway
Centre National de la Recherche Scientifique	France
Universitaet Duisburg-Essen	Germany
MONTIMAGE EURL	France
Istituto per Servizi di Ricovero e Assistenza agli Anziani	Italy
Baltic Open Solution Center	Latvia
Elektronikas un Datorzinatnu Instituts	Latvia

Revision history

Date	Version	Author	Comments
07.01.2019	0.0	Miguel Angel Anton / TECNALIA	Table of Contents
10.09.2019	0.1	Sergio Jiménez Gómez / INDRA	Template
20.09.2019	0.2	Sergio Jiménez Gómez / INDRA; Uģis Grīnbergs / BOSC; Arnor Solberg / TELLU; Miguel Angel Anton / TECNALIA	INDRA, TELLU and TECNALIA contributions to the deliverable
23.09.2019	0.3	Miguel Angel Anton / TECNALIA	First consolidated version to be reviewed
28.09.2019	0.3.1	Hui Song / SINTEF	Internal Review
30.09.2019	0.3.2	Nicolas Ferry / SINTEF	Internal review
30.09.2019	0.3.3	Anne Gallon / EVIDIAN	Internal review
09.10.2019	0.4	Sergio Jiménez Gómez / INDRA; Arnor Solberg / TELLU; Miguel Angel Anton / TECNALIA	Revised version addressing Internal review feedback
09.10.2019	1.0	Miguel Angel Anton / TECNALIA	Final version for submission

Contents

Table of Figures

1 Abbreviations and Definitions

Term	Definition
ACM	Actuation Conflict Manager
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARM	Advanced RISC Machine
BDA	Behavioural Drift Analysis
BLE	Bluetooth Low Energy
CTC	Centralized Traffic Control
DevOps	Development and Operations
eGW	Enhanced Gateway
EPJ	Electronic Patient Journals
ERTMS	European Rail Traffic Management System
GNSS	Global Navigation Satellite System
GRDP	General Registry of Data Protection
GSM-R	Global System for Mobile Communications – Railway
GUI	Graphical User Interface
GW	Gateway
HLA	High Level Architecture
HVAC	Heating, Ventilation and Air Conditioning
ID	Identification
IoT	Internet of Things
ISO	International Organization for Standardization
ITS	Intelligent Transport Systems
JSON	JavaScript Object Notation
KP	Knowledge Processor
KPI	Key Performance Indicator
L&M	Logistics and Maintenance
LED	Light-Emitting Diode
MQTT	Message Queue Telemetry Transport
MW	Middleware
OPC-UA	OPC Unified Architecture
OPEX	OPerating EXpense
OSI	Open System Interconnection
OTI	On-board Train Integrity
PLC	Programmable Logic Controller
QoS	Quality of Service
RCA	Root Cause Analysis
RFID	Radio-Frequency Identification
SIB	Semantic Information Broker
SIS	Smart IoT Systems
STCC	Smart Train Composition Coupling
SW	Software
TCP/IP	Transmission Control Protocol / Internet Protocol
TDS	Train Detection Systems

V&V	Validations and Verification
VPN	Virtual Private Network
WSN	Wireless Sensor Network

2 Introduction and Objectives

The goal of this document is to report the work performed in first phase of Task 1.2 towards the construction and implementation of the IoT environment for each of the three use cases of the ENACT project: Intelligent Transport System (ITS), Digital Health, and Smart Building. The IoT environment comprises the equipment and infrastructures that are needed to develop and operate each use case: the IoT and edge devices, the integration platforms and the cloud services. A set of DevOps Enablers that belong to the initial ENACT DevOps Framework will be demonstrated in each of the three use cases.

In Section 3 “Implementation of End-to-End Functionalities”, the deployment of each IoT environment is described for each of the three use cases. This includes general features of each Smart IoT System as the final architecture for the use case available at M21, the related schematics and the description of each device. This section also provides the objectives to provide to end users with the end-to-end functionalities as well as the details and status information of IoT services and applications.

In Section 4 “Testing of DevOps Scenarios”, specific scenarios for testing the DevOps practice in general are defined. In that way, the End-to-End Functionalities in each of the three ENACT use cases are broken into smaller DevOps scenarios in which the tests and operations happen simultaneously with design and development. These procedures allow for rapid product innovation cycles and ensure the trustworthy operation of Smart IoT Systems (SIS). The DevOps enablers developed in the ENACT project are novel concepts that offer a complete DevOps support for SIS. These Enablers are realised as open software engineering methods and tools that allow to continuously transform the IoT environments, deploy software, test and run scenarios in a loosely couple architecture. Each DevOps enabler is tested in at least one of the three use cases. The DevOps scenarios linked to the ITS use case cover the security, the failures mitigation, the monitoring, and the upgrade of the system. The DevOps scenarios linked to the Digital Health use case ensure security, privacy and in general trustworthiness. And finally, the DevOps scenarios applied to the Smart Building use case will be focused on the improvement of the design of both energy efficiency and user comfort IoT applications when they share the same IoT system infrastructure.

Finally, the Conclusion section summarizes the work done for the development and integration of each of the three use cases. The efforts to integrate DevOps Enablers in each use case and the lessons learned are also addressed, as well as pending implementations to be carried out in the second half of the project.

3 Implementation of End-to-End Functionalities

This section explains the implementation of each of the three solutions from a use case perspective, that is, the general features of each system (ITS, Digital Health and Smart Building) and the objectives to provide to end users. For each use case, the general architecture of the Smart IoT System is presented and the end-to-end functionalities that belong to each use case are described.

3.1 ITS Domain (Rail)

The deliverable D1.1 states that the Rail infrastructure requires expensive resources that makes the progress on the sector expensive and difficult to plan and execute. The architecture and development that have being built during the ENACT first period are intended to establish a rail platform to deal with those objectives. The main development is the provisioning of a safety and secure communication infrastructure that makes possible the integration with the current and future IoT and Cloud technologies with the rail systems to be controlled and/or improved.

As it is mentioned, the Use Case is designed to get attached different DevOps tools that permits increasing its value thanks to the DevOps philosophy. The flexibility of the ITS platform permits scaling the system indefinitely, this factor generates the need to check that the system is working in a coordinate manner, check that the software is the correct one in every device, ensure that every interface is safe and secure or that system is stable itself. Treating logistic and maintenance is critical task to check that every train is located and in a proper state. Therefore, updating every device to provide these logistic and maintenance data in a synchronized manner or ensure the different wireless interfaces requires and create a challenge for the tools that are intended to be used in the Use Case.

These tasks can be solved using the ENACT tools. As it is planned at this stage of the project, this functionality is intended to be integrated with the GeneSIS tool to deal with the rail devices orchestration and update processes, besides the RCA and the Security and Monitoring and Control tools to supervise the events that may occur and the security failures.

The following Figure 1 shows the ITS architecture.

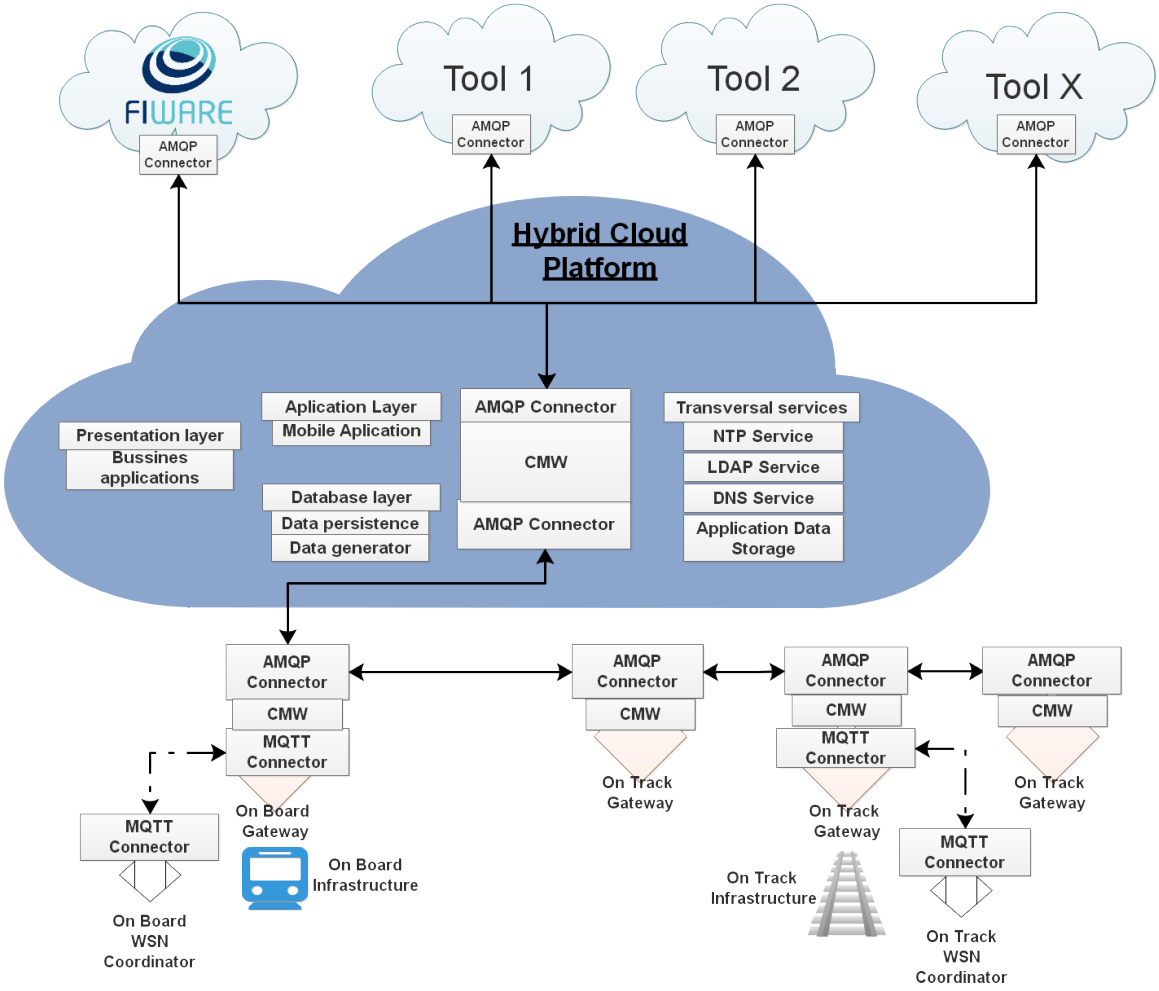


Figure 1: ENACT ITS Architecture. Source: INDRA.

The architecture presented is aligned with the trustworthiness definition stated along D2.1 and D1.1 as it is designed to preserve the security, privacy, safety, reliability, and resilience of the system. Moreover, the architecture design criteria decrease the gap between the operation and development in rail and becomes aligned with the current technology trends.

The architecture implemented is divided into three main sections: the thing, the gateway, and the Indra Hybrid Cloud Platform.

- **Thing section:** This section is formed by the Things deployed on the Rail Infrastructure without considering the gateways. It is divided into an On Board and On Track part and it covers the possibility to be connected to other Rail Systems (Signalling, Maintenance,) but this is not the focus of the ENACT project. In any case, it is designed to be rail agnostic as any Rail Thing could be connected.

  The On Board part is formed by the On Board nodes. These nodes provide the capacity to allocate the geolocalised composition (set of wagons and loco that form the train) continuously with a 0.25 seconds refresh rate. Moreover, the composition and all the wagons that build it can be identified wirelessly in a flexible manner discriminating other compositions. Finally, each node provides Logistics and Maintenance information (location and power consumption) that is used into the ENACT environment. The On Track part provides information to locate the composition at specific points over the track even with no On Board gateway.

- **Gateway section** : This section represents the main integration core between the Use Case and the DevOps part. This section coordinates the information exchange between the Rail section (Things + Business layer) and the Cloud services ensuring proper communications. The gateway connects the Edge section, the business layer, and the Indra Hybrid Cloud Platform in order to coordinate the system logic, the data acquisition, and the data storage.

- **Indra Hybrid Cloud Platform section:** This section is in charge of storing the information, adapting it to other protocols, connecting the ENACT tools, and providing the connection with other Cloud platforms to generate a scalable system. Moreover, the security mechanisms are applied at this stage.

These three sections are linked and coordinated throw several protocols at different levels. The edge and gateway sections communicate throw MQTTS in order to transmit lightweight information in a secure manner (encryption and authentication) providing identification to all the elements of the edge part authorized to be involved into the system. The gateway and the Cloud communicate throw the AMQP protocol that guarantees higher security levels even for aggregated traffic. Moreover, it is an extended protocol that has a connector in the most relevant open and private Cloud platforms. The communication among the three sections is ensured by the security services centralized at the Cloud.

The platform development is already adapted to the ENACT need stated into the D1.1 as it is flexible and independent to be integrated with other systems, FIWARE and ENACT tools in the ENACT project case. The relevancy for the ENACT project and of the ENACT project for the rail sector is that it is an industrial environment that offers a high regulation procedure. It also offers the possibility to print a high-quality improvement into the development and operation of the rail sector updating the last century systems with current technologies in a secure and safe manner. In the context of ENACT, the secure communication infrastructure will be used in the following logistic and maintenance functionality.

3.1.1 ITS Functionality: Logistics and Maintenance

The Logistics and Maintenance is a well-known service within the transport environment especially into the rail infrastructure. The constant location of the rolling stock and the cargo is critical and there are current systems designed to accomplish these functionalities. However, they have several limitations that do not permit checking the status and the location of the rolling stock constantly.

The main objective of the Logistics and Maintenance service is to provide information about the train's location and their status. The Location information is extracted from the Train Integrity and Train Inauguration services detailed in the Annex A.1.1 The Train Inauguration service is in charge of identifying a composition and validates that the wagons that form it are the correct ones discriminating other compositions, and the Train Integrity service indicates the completeness of the composition during the operation. The Train Inauguration service provides information about the train and wagon identification and the Train Integrity service about the position.

The maintenance is covered with energy status information provided by the edge part. This information can be used in order to infer several maintenance parameters that can be used by the Rail Operator to track the wagons state.

As it was stated, these functionalities can be widely expanded connecting other rail systems to the infrastructure. The main point is to provide a safe and secure system to connect them as it is done in the current system.

The procedures that enable the system are explained in the Figure 2. The entities represented are:

- **WSN Coordinator:** the Logistic and Maintenance On Board data source

- **RFID tag:** represents the On Track data source as it provides information about the train location (based on the known presence of the RFID reader) as a complement in case of no On Board infrastructure.
- **Gateway:** represented at this instance as the communication middleware with the business layer that treats the Logistic and Maintenance data
- **Cloud:** It is represented in this scheme as the source of the orders to interact with the functionality.

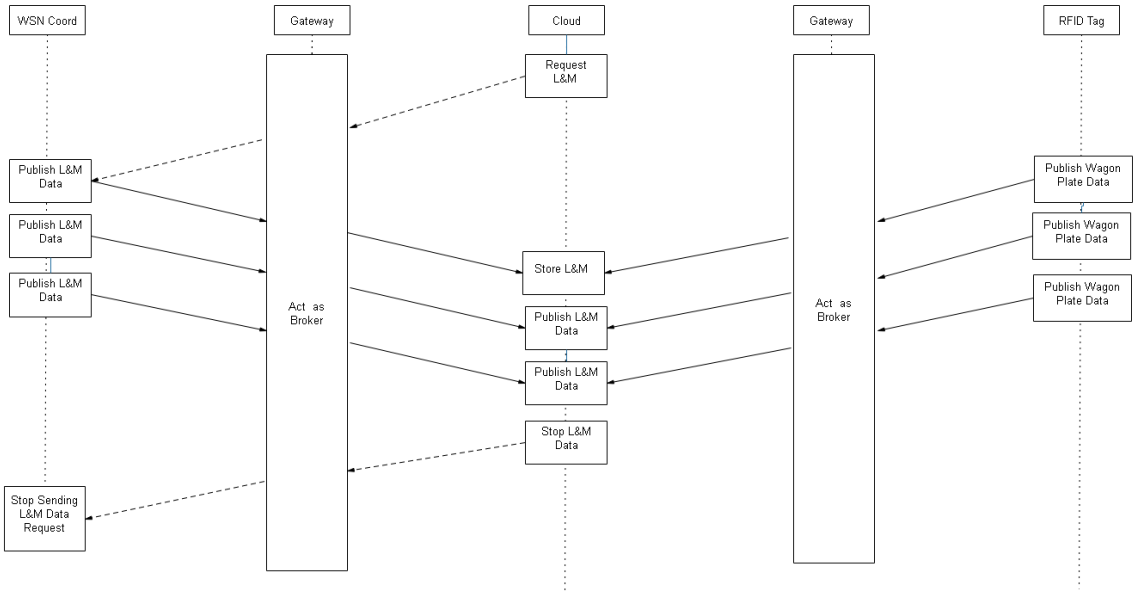


Figure 2: ENACT L&M Procedures. Source: INDRA.

The Train Inauguration and Train Integrity services are launched before the operation is started as they will be required for the Logistics and Maintenance functionality. Once the train starts, the operation and those services are already launched, the information needed to start the Logistics and Maintenance functionality is available.

The order to start the Logistics and Maintenance services is requested by the Rail Operator, then, the sensors provide the information to be stored at the Cloud in the On Board case. However, in the On Track case the information is reported with no order as it is generated by a RFID tag, this case will provide only the location and identification of the composition and its wagons.

All these generated data are reported to the Indra Hybrid Cloud Platform only by authenticated elements into the system and encrypted. These data are available and stored in the Cloud for all the tools and platforms authenticated to be connected to the Indra Hybrid Cloud Platform. The usage of these data is introduced into the sections 4.1.

3.2 Digital Health

TellU has implemented the remote patient monitoring system, which is an innovative digitalisation solution for efficient and cost effective monitoring and following up of patients that want to live at home. To be able to scale this eHealth solution, new DevOps tools that supports software systems that are deployed across the IoT, edge and cloud space is needed. In particular we need more advanced tools for managing large fleet of Personal Health Gateways (the core edge component), that again manages a set of sensors and devices residing in the IoT layer. Moreover, it is crucial that the operation and the evolution of the system is trustworthy and adhere to strict security and privacy requirements. Thus, the remote patient monitoring system will be well suited for validating a set of the ENACT results.

This subsection provides an overall description of the system and of the main application functionalities that are implemented. In Section 4.2 the architectural design is described more in detail and the current status of the application of the ENACT enablers is reported.

Figure 3 provides an overview of the overall remote patient monitoring use case. It includes a health provider application where the health providers can specify the follow up plan for each individual patient and check the status of patient tasks (e.g., providing measurements applying medical devices) as well as checking the status of the patient health and in general see information gathered from various sensors (including cameras).

The patient app is applied by the patient to help him follow up his tasks (e.g., replying to a questionnaire, follow up reminders etc.)

The administration application provides the tool for managing devices, personal health gateways, patients and health providers in the remote patient monitoring system. There is a separate module in the Health provider application that can be accessed from there for the persons that have the proper access rights.

The system is developed do be compliant with relevant standards in the Health domain, in particular with HL7 FHIR (https://www.hl7.org/fhir/), OpenID Connect Health Relationship Trust (HEART) recommendations (https://openid.net/wg/heart/), and it supports 2 factor access control according to national standards such as BankID in Norway. The development has been conducted according to ISO27001 policies.

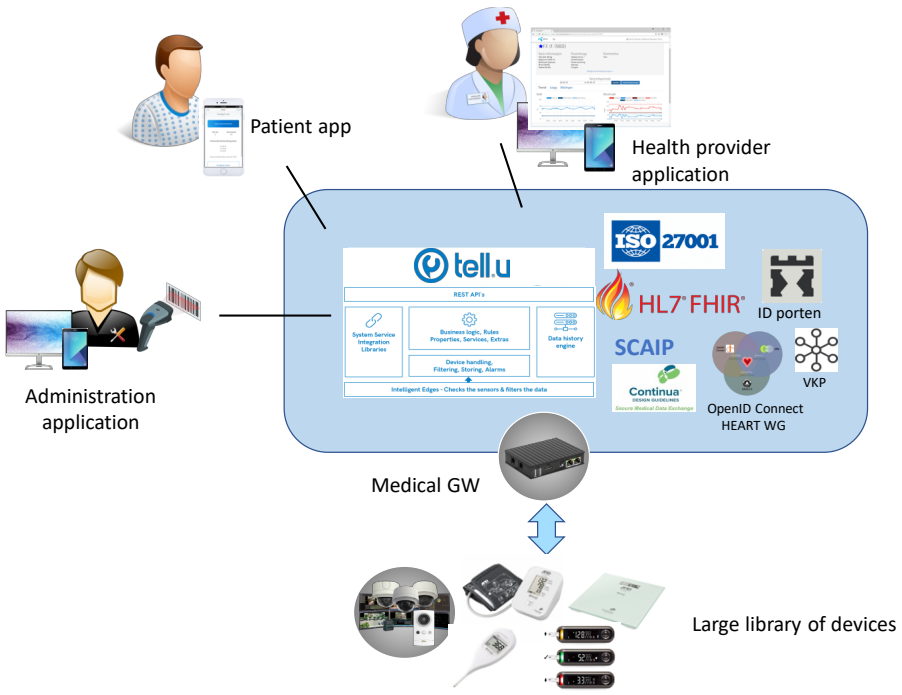


Figure 3: Overall eHealth application. Source: TellU.

3.2.1 Digital Health Functionality 1: Daily measurements

The Remote patient monitoring system is generic in the sense that it can be applied for patients with various set of diseases (e.g., Diabetes, Kidney, COPD, Cancer, etc.). Based on individual needs the patients get a package of devices in their home, together with the Personal Health Gateway that manages and controls the set of devices and ensures that the measurements are sent to the right health provider(s) and stored in the electronic patient archives. The functionality presented below illustrates such kind of a functionality for a diabetes patient.

Actors	**Primary user (Anders), administrator, Health Provider**
Description	Daily measurements of Glucose level and blood pressure along with a questionnaire
Trigger	Anders measures his blood glucose and blood pressure every morning after waking up and every night before going to bed. Along with the measurements he answers a set of questions put up by his doctor in a questionnaire using the app. This functionality is triggered two times a day, by Anders himself. He will also get reminders to fulfil his tasks.
Normal flow	1. Anders logs in to the app. 2. He replies to questionnaire and submits it. 3. He makes measurements of blood glucose and blood pressure. Anders pricks his finger for blood and makes a measurement with his blood glucose meter. Then he does the blood pressure measurement. 4. The Personal Health Gateway immediately receives the measured values and transmits them to the backend FHIR database

3.2.2 Digital Health Functionality 2: Status Check and Alert on Bad measurements

In general, the health providers will regularly check the status of the patient according to planned intervals and they get prioritized tasks to follow up. These can be triggered based on measured values and/or responses on the questionnaire. The patient can also follow his own health status using the app. The functionality presented below illustrates such kind of a functionality.

Actors	**Primary user, Health Provider**
Description	Health personnel of the health provider follows up the patient's measurement and replies on questionnaires according to the patient care plan. This is provided as tasks in the system. In case of bad measurements or trends, there will be an alert and following up tasks get higher priority
Trigger	The following up tasks are triggered according to the care plan as well as a result of alerts in case of measurements outside specified thresholds or bad trends (e.g., in replies of questionnaires)
Normal flow	1. Health personnel logs in to the application. 2. A set of prioritized tasks is listed for them to follow up.

	3. High priority tasks are caused by measurement or questionnaire replies that exceed thresholds or indicate abnormal health issues. 4. The Health personnel checks the health status and follows up accordingly. 5. When a task is completed it is marked accordingly. 6. Health personnel logs out.

3.2.3 Digital Health Functionality 3: Administration of Devices and Patients

Actors	**Primary user, Administrator**
Description	Personnel of the health provider with administrator rights register and manage devices and patients.
Trigger	A new patient shall be registered with a device package according to a defined care plan.
Normal flow	1. Administrator logs in to the application. 2. An existing Personal Health Gateway is selected or a new one is registered in the system. This is done most efficiently using a bar code reader. 3. The package of devices required for the actual patient is selected or new ones are registered in the system. 4. The set of devices selected are associated with the Personal Health Gateway (in order for the Gateway to connect pair when online with the right devices). 5. A patient is registered/selected in the system. 6. Personal Health Gateway is associated with the patient. 7. Package of devices are shipped to Patient. 8. Personal Health Gateway and devices are turned on at Patient home. 9. PHG looks for devices and pair when devices are in paring mode. 10. PHG and devices are onboarded and managed by the system.

3.3 Smart Building

The Smart Building use case will develop Smart Energy Efficiency applications and Smart User Comfort applications to be tested in KUBIK building to validate the ENACT DevOps framework. The continuous development and deployment of these applications for concurrent execution using common sensors and actuators presents many challenges that jeopardize the trustworthiness of the whole IoT environment. DevOps practice of continuous and frequent iterations between development and operation is required, as well as the use of novel DevOps Enablers for risk-based decision support, proper management of actuation conflicts and assurance of trustworthy, security and privacy.

The updated general architecture of the Smart Building use case is shown in Figure 4. A new block called the Building Management System or BMS has been added compared to the use case architecture presented in the deliverable D1.1. The reason for this is that the general control system of the Kubik building has changed in recent months and now the monitored parameters and the actuation orders pass through a centralized system that collects and controls the entire building.

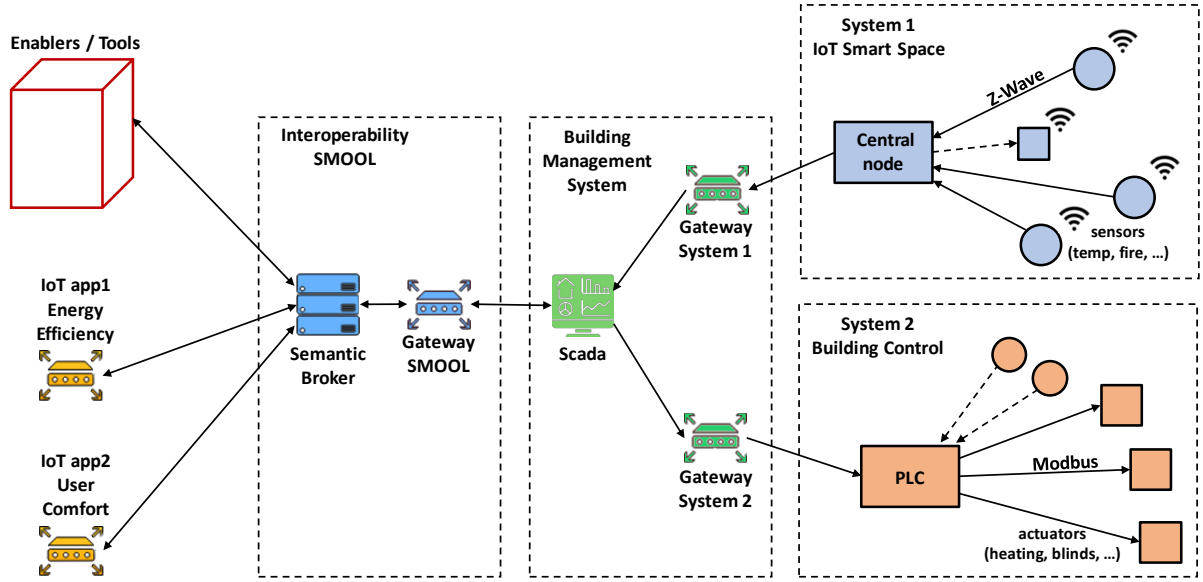


Figure 4: Updated General schema of the Smart Building use case. Source: TECNALIA.

The Smart Building use case is formed by two different systems that interoperate using the Interoperability SMOOL IoT middleware that has a semantic broker for connecting heterogeneous devices or sources of information. The Building Management System also centralize all the information of new wireless IoT devices (system 1) and the legacy building control systems of the building (system 2) using a Scada software. The system 1 is a wireless sensor/actuator network using the Z-Wave protocol. The system 2 is a legacy network consisting of PLCs or industrial PCs that use building automation protocols (KNX, DALI, PROFIBUS, etc.) and direct control over the devices through relays or analog/digital outputs. The updated high-level architecture of the components of the Smart Building use case is shown in Figure 5.

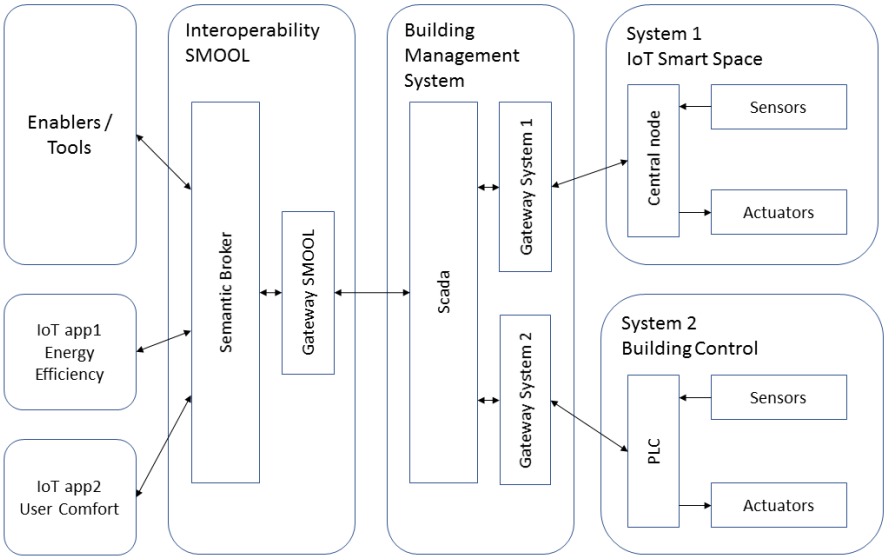


Figure 5: Updated High Level architecture of the Smart Building use case. Source: TECNALIA.

3.3.1 Smart Building Functionality: Energy Efficient Building

The Energy Efficient Building functionality involves concurrent IoT applications related to energy efficiency and user comfort. The functionality comprises different devices that belong either to the IoT Smart Space with Z-Wave wireless communication capabilities or to the PLC-controlled building control system with wired building automation communications.

Smart Energy Efficiency applications perform thermal comfort control of the building using environmental sensors, motorized blinds, electric heaters and HVAC actuators. An energy efficient application that involves temperature values from two to four different sensors that triggered contradictory actuation commands to the same electric heater was programmed. A more complex application involving HVAC actuators, various temperature sensors and different state of blinds which affect the solar radiation that each sensor is receiving is being developed.

Smart User Comfort applications operate as alert IoT systems for elderly care using water flood sensors, smoke sensors, CO2 concentration sensors and power consumption meters. Alert applications were programmed for flood, smoke and unhealthy CO2 concentration that send an alert email when risk values are measured.

The ground floor of the Kubik building is the main location for the Energy Efficient Building functionality. This ground floor has been divided into different rooms which mimics an apartment as shown in Figure 6: bedroom, kitchen, living room and corridor.

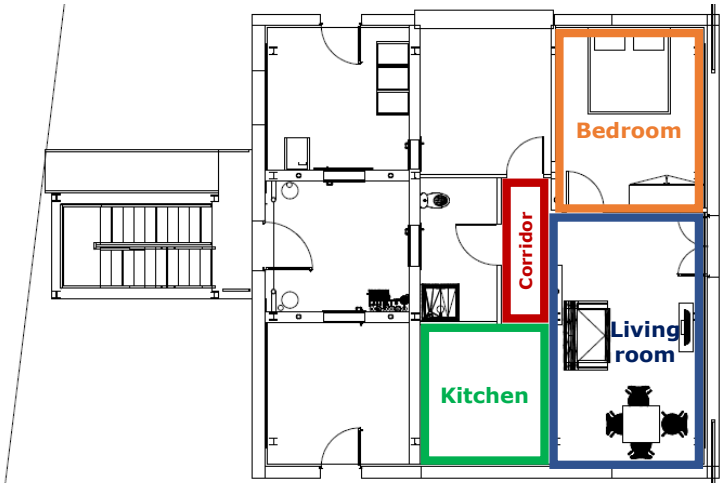


Figure 6: Rooms on the Ground Floor of the Kubik Building. Source: TECNALIA.

The floor plan and the reflected ceiling plan of the ground floor of Kubik with the ENACT sensors and actuators in their approximate location are shown in Figure 7 and Figure 8.

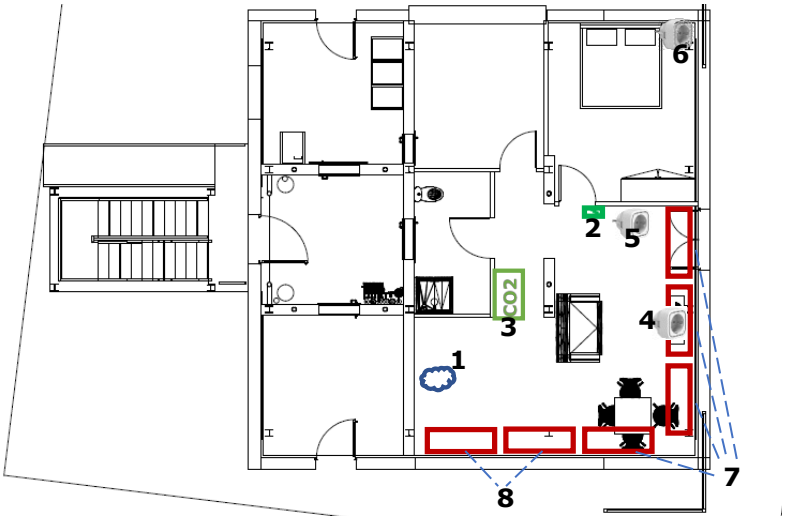


Figure 7: Floor Plan of the Ground Floor of Kubik. Source: TECNALIA.

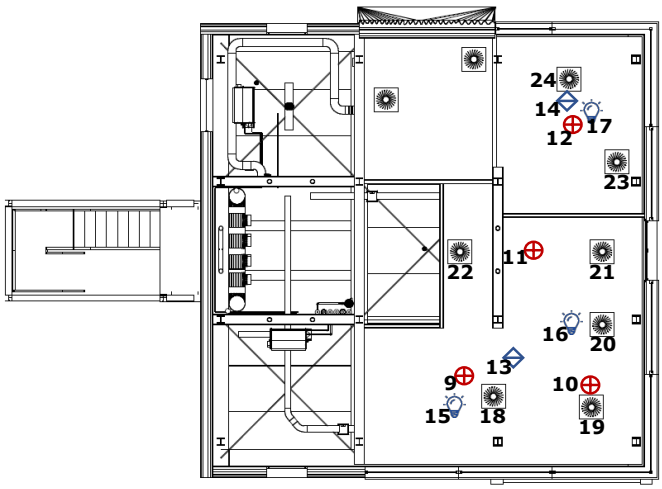


Figure 8: Reflected Ceiling Plan of the Ground Floor of Kubik. Source: TECNALIA.

The table of Figure 9 shows a detailed description of each device represented in the floor plan of the ground floor of Kubik (Figure 7), its location in a specific room, its belonging to the IoT Smart Space or the wired Building Control group and, finally, the measures or commands it provides. The same is done in Figure 10 for the devices shown in reflected ceiling plan of the ground floor of Kubik (Figure 8).

Ground Floor - Floor Plan										
Device				Location				System		Signal
ID	Sensor	Actuator	Device type	Bedroom	Kitchen	Living Room	Corridor	IoT Smart Space Z-Wave wireless	Building Control PLC wired	Measure/Command type and Unit
1	X		Water Flood Sensor		X			X		Flood alarm state (binary): ON/OFF
2	X		Door Multisensor 1	X				X		Position (binary): OPEN/CLOSED
3	X		CO2 Sensor 1		X	X	X	X		CO2 level: 0 ppm to 2000 ppm
4		X	Remote Socket 1			X		X		Switch state (binary): ON/OFF
4	X		Remote Socket 1			X		X		Electric energy consumption: Watts
5		X	Remote Socket 2			X		X		Switch state (binary): ON/OFF
5	X		Remote Socket 2			X		X		Electric energy consumption: Watts
6		X	Remote Socket 3	X				X		State (binary): ON/OFF
6	X		Remote Socket 3	X				X		Electric energy consumption: Watts
7		X	4 Motors for Blinds			X			X	Position (binary): UP/DOWN
8		X	2 Motors for Blinds		X				X	Position (binary): UP/DOWN

Figure 9: Devices/Signals on the Ground Floor and Floor Plan of Kubik. Source: TECNALIA.

Ground Floor - Reflected Ceiling Plan										
Device				Location				System		Signal
ID	Sensor	Actuator	Device type	Bedroom	Kitchen	Living Room	Corridor	IoT Smart Space Z-Wave wireless	Building Control PLC wired	Measure/Command type and Unit
9	X		Ceiling Multisensor 1		X			X		Motion (binary): YES/NO
9	X		Ceiling Multisensor 1		X			X		Temperature: degrees Celsius
9	X		Ceiling Multisensor 1		X			X		Light: 0 lux to 1000 lux
9	X		Ceiling Multisensor 1		X			X		Relative humidity: 20% to 95%
10	X		Ceiling Multisensor 2			X		X		Motion (binary): YES/NO
10	X		Ceiling Multisensor 2			X		X		Temperature: degrees Celsius
10	X		Ceiling Multisensor 2			X		X		Light: 0 lux to 1000 lux
10	X		Ceiling Multisensor 2			X		X		Relative humidity: 20% to 95%
11	X		Ceiling Multisensor 3			X	X	X		Motion (binary): YES/NO
11	X		Ceiling Multisensor 3			X	X	X		Temperature: degrees Celsius
11	X		Ceiling Multisensor 3			X	X	X		Light: 0 lux to 1000 lux
11	X		Ceiling Multisensor 3			X	X	X		Relative humidity: 20% to 95%
12	X		Ceiling Multisensor 4	X				X		Motion (binary): YES/NO
12	X		Ceiling Multisensor 4	X				X		Temperature: degrees Celsius
12	X		Ceiling Multisensor 4	X				X		Light: 0 lux to 1000 lux
12	X		Ceiling Multisensor 4	X				X		Relative humidity: 20% to 95%
13	X		Smoke Detector 1		X	X	X	X		Smoke alarm state (binary): ON/OFF
14	X		Smoke Detector 2	X				X		Smoke alarm state (binary): ON/OFF
15		X	Ceiling Light 1		X				X	Light state (binary): ON/OFF
16		X	Ceiling Light 2			X			X	Light state (binary): ON/OFF
17		X	Ceiling Light 3	X					X	Light state (binary): ON/OFF
18		X	Fan Coil 1		X				X	Operation state (binary): ON/OFF
18		X	Fan Coil 1		X				X	Temperature setpoint: Celsius
19		X	Fan Coil 2			X			X	Operation state (binary): ON/OFF
19		X	Fan Coil 2			X			X	Temperature setpoint: Celsius
20		X	Fan Coil 3			X			X	Operation state (binary): ON/OFF
20		X	Fan Coil 3			X			X	Temperature setpoint: Celsius
21		X	Fan Coil 4			X			X	Operation state (binary): ON/OFF
21		X	Fan Coil 4			X			X	Temperature setpoint: Celsius
22		X	Fan Coil 5				X		X	Operation state (binary): ON/OFF
22		X	Fan Coil 5				X		X	Temperature setpoint: Celsius
23		X	Fan Coil 6	X					X	Operation state (binary): ON/OFF
23		X	Fan Coil 6	X					X	Temperature setpoint: Celsius
24		X	Fan Coil 7	X					X	Operation state (binary): ON/OFF
24		X	Fan Coil 7	X					X	Temperature setpoint: Celsius

Figure 10: Devices/Signals on Ground Floor and Reflected Ceiling of Kubik. Source: TECNALIA.

The first floor of the Kubik building has a diaphanous space that can also be used in the ENACT project. The specific location of this space is shown in Figure 11.

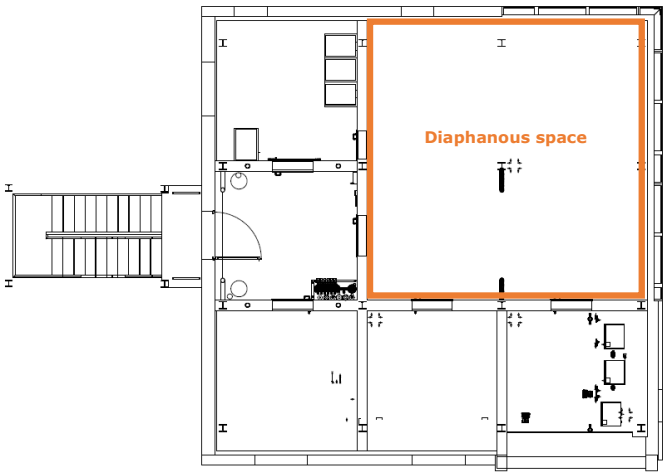


Figure 11: Rooms on the First Floor of the Kubik Building. Source: TECNALIA.

The floor plan and the reflected ceiling plan of the first floor of Kubik with the ENACT sensors and actuators in their approximate location are shown in Figure 12 and Figure 13.

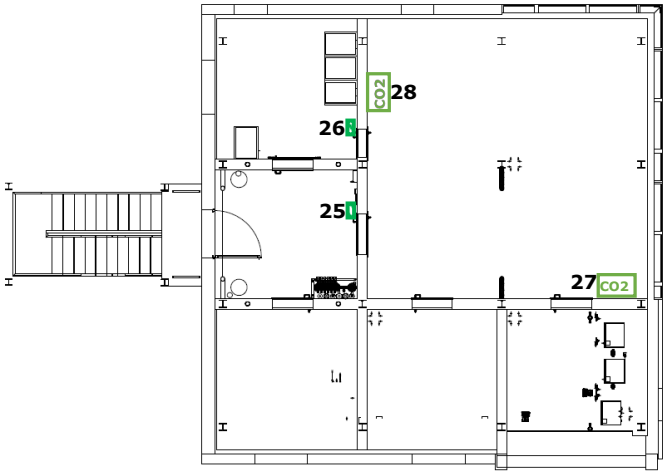


Figure 12: Floor Plan of the First Floor of Kubik. Source: TECNALIA.

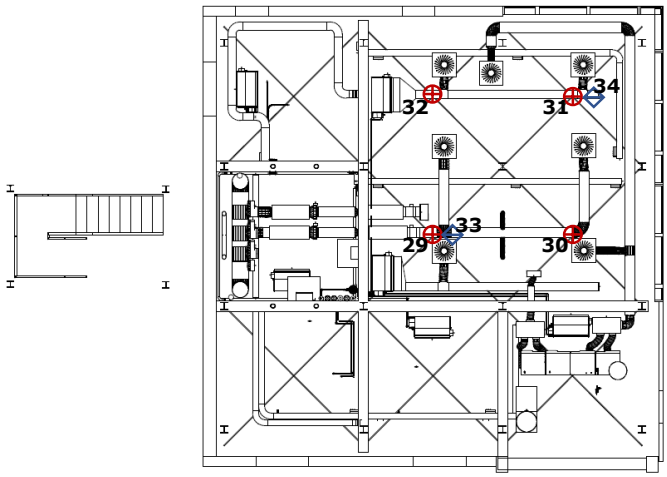


Figure 13: Reflected Ceiling Plan of the First Floor of Kubik. Source: TECNALIA.

The table of Figure 14 shows a detailed description of each device represented in the floor plan of the ground floor of Kubik (Figure 12). The same is done in Figure 15 for the devices shown in reflected ceiling plan of the ground floor of Kubik (Figure 13).

First Floor - Floor Plan							
Device				Location	System		Signal
ID	Sensor	Actuator	Device type	Diaphanous space	IoT Smart Space Z-Wave wireless	Building Control PLC wired	Measure/Command type and Unit
25	X		Door Multisensor 2	X	X		Position (binary): OPEN/CLOSED
26	X		Door Multisensor 3	X	X		Position (binary): OPEN/CLOSED
27	X		CO2 Sensor 2	X	X		CO2 level: 0 ppm to 2000 ppm
28	X		CO2 Sensor 3	X	X		Switch state (binary): ON/OFF

Figure 14: Devices/Signals on the First Floor and Floor Plan and of Kubik. Source: TECNALIA.

Ground Floor - Reflected Ceiling Plan							
Device				Location	System		Signal
ID	Sensor	Actuator	Device type	Diaphanous space	IoT Smart Space Z-Wave wireless	Building Control PLC wired	Measure/Command type and Unit
29	X		Ceiling Multisensor 5	X	X		Motion (binary): YES/NO
29	X		Ceiling Multisensor 5	X	X		Temperature: degrees Celsius
29	X		Ceiling Multisensor 5	X	X		Light: 0 lux to 1000 lux
29	X		Ceiling Multisensor 5	X	X		Relative humidity: 20% to 95%
30	X		Ceiling Multisensor 6	X	X		Motion (binary): YES/NO
30	X		Ceiling Multisensor 6	X	X		Temperature: degrees Celsius
30	X		Ceiling Multisensor 6	X	X		Light: 0 lux to 1000 lux
30	X		Ceiling Multisensor 6	X	X		Relative humidity: 20% to 95%
31	X		Ceiling Multisensor 7	X	X		Motion (binary): YES/NO
31	X		Ceiling Multisensor 7	X	X		Temperature: degrees Celsius
31	X		Ceiling Multisensor 7	X	X		Light: 0 lux to 1000 lux
31	X		Ceiling Multisensor 7	X	X		Relative humidity: 20% to 95%
32	X		Ceiling Multisensor 8	X	X		Motion (binary): YES/NO
32	X		Ceiling Multisensor 8	X	X		Temperature: degrees Celsius
32	X		Ceiling Multisensor 8	X	X		Light: 0 lux to 1000 lux
32	X		Ceiling Multisensor 8	X	X		Relative humidity: 20% to 95%
33	X		Smoke Detector 3	X	X		Smoke alarm state (binary): ON/OFF
34	X		Smoke Detector 4	X	X		Smoke alarm state (binary): ON/OFF

Figure 15: Devices/Signals on the First Floor and Reflected Ceiling of Kubik. Source: TECNALIA.

4 Testing of DevOps Scenarios

The DevOps scenarios describe the scenarios related to the development and operation of the system. In each specific scenario a set of the ENACT DevOps Enablers are tested. In the ITS use case, the DevOps scenarios are representative of a real implementation of the system components in logistics and maintenance rail activities. In the Digital Health use case, the DevOps scenarios concentrate mainly on the Medical Gateway operation, which is the central component for handling the edge and IoT layers. Finally, in the Smart Building use case, the DevOps scenarios focus on the concurrent operation of multiple energy efficiency and user comfort IoT applications using the same IoT infrastructure which needs a proper orchestration of components, resolving actuation conflicts and mitigating trustworthiness, security and privacy risks.

The table in Figure 16 summarizes all the DevOps scenarios that belong to each of the three use cases, the relevant ENACT enablers that are intended to be tested in the scenario and the current development status of the scenario.

Use case	DevOps scenario	DevOps Enablers	Current status
ITS Domain	Things Data Monitoring	Context Monitoring and Actuation Conflict Management Enabler	To be developed. The first definition is stablished. However, it will be completed in the second part of the project.
	Software Update/Actuation	Orchestration and Continuous Deployment Enabler	First integration completed. For the second iteration, the tool will be integrated into the business layer. At this moment only deployment can be done based on the system status.
	Simulation and Testing	Test, Emulation and Simulation Enabler	To be developed. First definition is completed.
	Root Cause Analysis	Run-time Quality Assurance and Root Cause Analysis Enabler	First integration completed. Exchange data processes have been completed.
	Security Monitoring	Security and Privacy Monitoring and Control Enabler	First definition completed. Current complementary parts for the system are under evaluation process and definition.
Digital Health	Initial Software deployment and GW onboarding	Orchestration and Continuous Deployment Enabler Security and Privacy Monitoring and Control Enabler Test, Emulation and Simulation Enabler	First semi-automatic version of the use case with the initial software deployment and GW on boarding is implemented, partly based on ENACT concepts. ThingML is applied for the GW software stack. Integrations with, and validation of the ENACT enablers will be conducted in the next phase of the project.
	Software deployment and evolution in the Gateway	Orchestration and Continuous Deployment Enabler Security and Privacy Monitoring and Control Enabler Test, Emulation and Simulation Enabler	First semi-automatic version of the use case with the software deployment and evolution in the GW is implemented, partly based on ENACT concepts. ThingML is applied for the GW software stack. Integrations with, and validation of the ENACT enablers will be conducted in the next phase of the project.
	Gateway recovery and factory reset in the field	Robustness & Resilience Enabler	First semi-automatic version of the GW recovery and factory reset in the field is implemented, partly based on ENACT concepts. Integrations with, and validation of the ENACT enabler is planned for the next phase of the project.

	Software testing on the Gateway	Test, Emulation and Simulation Enabler Risk-Driven Decision Support Enabler	Simple software testing on the Gateway is initiated. The plan is to also explore ThingML for generating tests. Furthermore, the plan is to integrate the Risk Driven Decision support enabler in TellU DevOps stack. Integrations with, and validation of the ENACT enablers will be conducted in the next phase of the project.
	Continuous integration of gateway modules and versioning	Test, Emulation and Simulation Enabler Robustness & Resilience Enabler Orchestration and Continuous Deployment Enabler	Simple continuous integration loops of the Gateway modules are initiated in the current use case based on ENACT concepts. Integrations with, and validation of the ENACT enablers will be conducted in the next phase of the project
	Trustworthiness of the medical system	Security and Privacy Monitoring and Control Enabler Risk-Driven Decision Support Enabler Robustness & Resilience Enabler	For the first version of the use case we have spent quite some effort on the security architecture design and implementation. This has included reviews of the security architecture from security experts in ENACT and also by following recent recommendations from OpenID connect HEART WG and ensure compliance with standards and legislations (e.g., ISO 27001 and GDPR). Integrations with, and validation of the ENACT enablers will be conducted in the next phase of the project.
Smart Building	Thermal comfort control – heating design	Risk- Driven decision support Enabler Orchestration and Continuous Deployment Enabler	The testing of the Risk-driven Decision Support Enabler is pending. ThingML models have been integrated with the SMOOL middleware to be automatically deployed by the Orchestration and Continuous Deployment Enabler (GeneSIS).
	Thermal comfort control – conflict in heating actuator use	Context Monitoring and Actuation Conflict Management Enabler Orchestration and Continuous Deployment Enabler	The Actuation Conflict Manager (ACM) Enabler has been tested to resolve the sending of conflicting orders by different IoT flows to the same heating actuator. The integration of the ACM Enabler with GeneSIS is done.
	Thermal comfort control – conflict in temperature actuation	Context Monitoring and Actuation Conflict Management Enabler Orchestration and Continuous Deployment Enabler	The integration of the Behavioural Drift Analysis (BDA) Enabler is pending. The integration of the BDA Enabler with GeneSIS is done.
	Smart building Alerts for User Comfort	Security and Privacy Monitoring and Control Enabler	The Security and Privacy Monitoring Enabler has been tested to check if the messages are sent by authorized nodes, detect abnormal network traffic from/to external devices and ensure the integrity of the sensing and actuation data. The testing of the Security and Privacy Adaptation Enabler is pending.
	Thermal comfort control – self-optimizing controller design	Context-Aware Self-Adaptation Enabler Orchestration and Continuous Deployment Enabler	The testing of the Online Learning Enabler is pending.

Figure 16: Current status of DevOps scenarios. Source: INDRA, TellU and TECNALIA.

4.1 ITS Domain (Rail)

4.1.1 Scenario 1: Things Data Monitoring

The integrations works are initiated but pending to be developed in the next iteration.

4.1.2 Scenario 2: Software Update/Actuation

The Software Update and Actuation scenario is based on the interaction between the GeneSIS ENACT tool (aka. Orchestration and deployment enabler) and the ITS Rail Use Case. The objective of this tool within the ITS system is to make deployments to update the software of the different hardware and applications carried by the gateway section of the system.

The gateway part is considered as the core for the system integration since it connects the Indra Hybrid Cloud Platform and the edge part., At the current project stage this is the only device that is able to be updated with the aim to configure the connection among the different system parts. The software components to be deployed on the gateways are encapsulated into Docker containers. Docker thus serves as bootstrap for GeneSIS to ensure their proper management, control, registration, update, and deployment.

The final implementation of the tool goes from the deployment specification models where the devices to update are chosen based on a specific mode modelling language, the actions to trigger the deployment and the deployment itself. However, at this stage, the tool goes from the installation of the Docker containers with the ITS functionalities to the application that permits the Infrastructure Manager/Rail Operator Maintenance Team (responsible of the execution of the associated software change request) need to select the devices that must be updated.

To perform these tasks, an interface will provide the Infrastructure Manager/Rail Operator Maintenance Team with the different systems views (from a business perspective, including the deployed version, their status, etc.) able to be updated as well as the SW repository with the different releases that could apply to them. It is required to show a distribution of the devices along a geographical area defined by the Infrastructure Manager/Rail Operator Maintenance team and/or by application. Therefore, a geographical and an application criteria are applied. The interface managed by the Infrastructure Manager/Rail Operator Maintenance Team to control the deployments is provided by the Enabler.

The tool itself permits to the Infrastructure Manager/Rail Operator Maintenance Team not only to provide instant manual deployments. It is also possible planning these deployments to be executed at certain time in order to cover several operational situations that may occur with no Infrastructure Manager/Rail Operator Maintenance Team direct supervision.

The data that it is going to be deployed is planned by the Infrastructure Manager/Rail Operator Maintenance Team and serves as an input for the deployment application. The outputs from the Application Behaviour Monitoring and the Actuation Conflicts Control are also expected. Finally, the tool will execute the deployment.

Docker is the underlying mechanism used by GeneSIS to deploy software components (i.e., GeneSIS deploy Docker containers). The instantiation of a container is made from a base image that has all the required Docker software's to be managed. Afterwards, this base image is configured with the required ITS software to enable both the system and the integration among the different system parts. Finally, the image is built to be ready to run at the gateway. Every step at the image generation is tracked and monitored thanks to the Docker tools.

In case the image is deployable, based on the system status information, the personalized imaged is run in the device. Therefore, the new functionalities are ready to be used in the gateway following the ITS business needs.

The current activities performed are summarised into the integration between systems. Several credentials are exchanged in order to both enable the connection to Indra Hybrid Cloud Platform and to the information that permits the deployment. It is expected for further developments at the end of the project to manage these Docker images in a more complex and functional manner following a business logic apart of checking the system status.

The further procedures considered for the deployment are shown in the Figure 17. In this Figure the orders are given by the Rail Operator and the interaction between the tool and the system is done throw the stored data on the database (FIWARE, Indra Hybrid Platform, …)

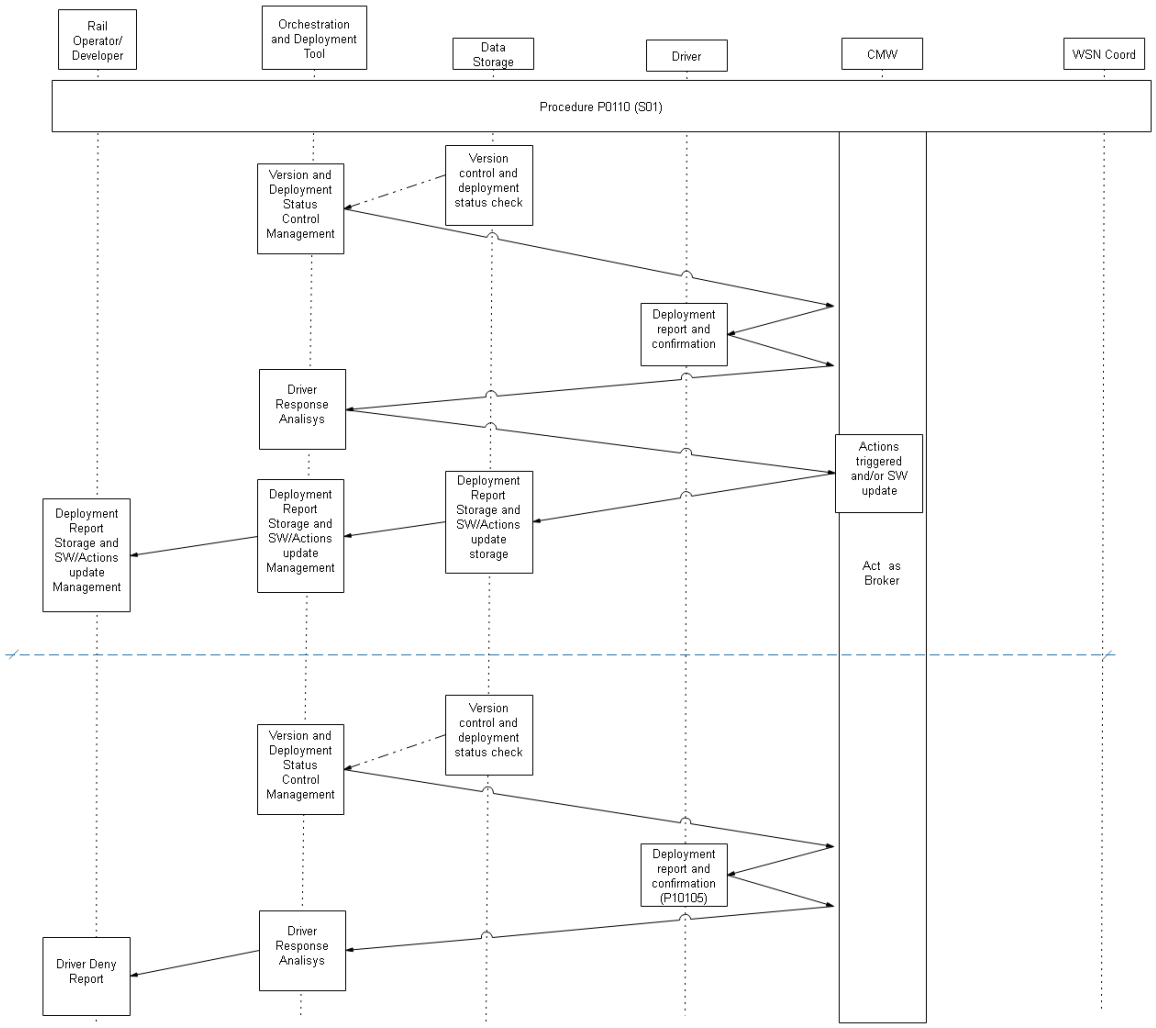


Figure 17: ENACT Deployment Procedures. Source: INDRA.

The plan procedures are explained in the Figure 18:

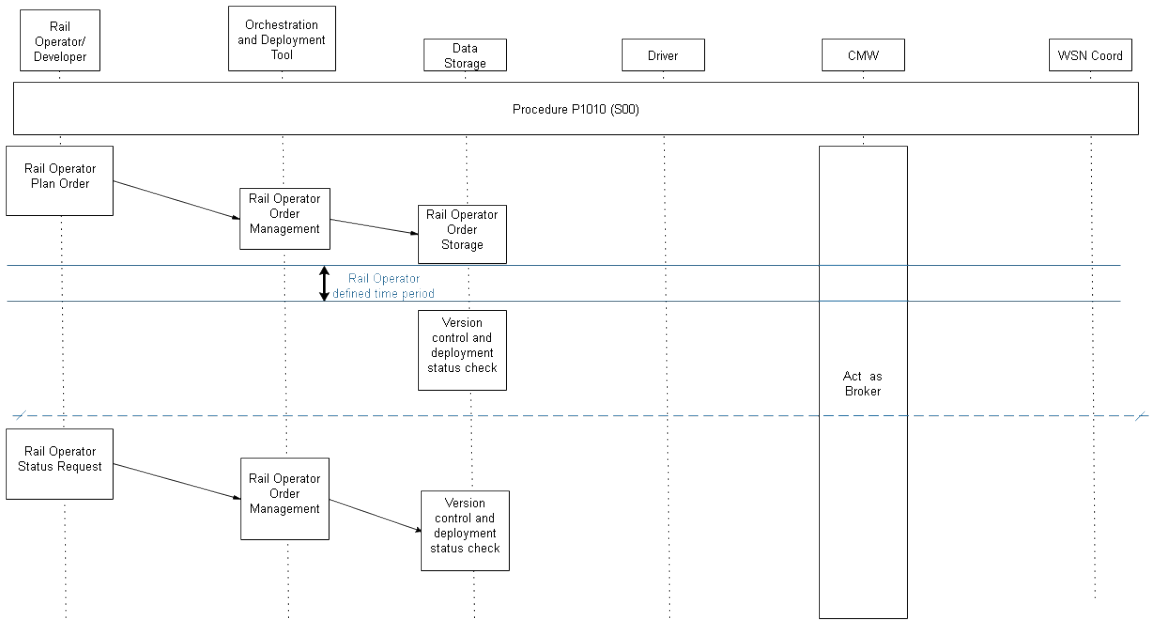


Figure 18: ENACT Programmed Deployment Procedures. Source: INDRA.

Finally, the report of the status data to inform to the Cloud about the availability of the system to receive a deployment in the Figure 19:

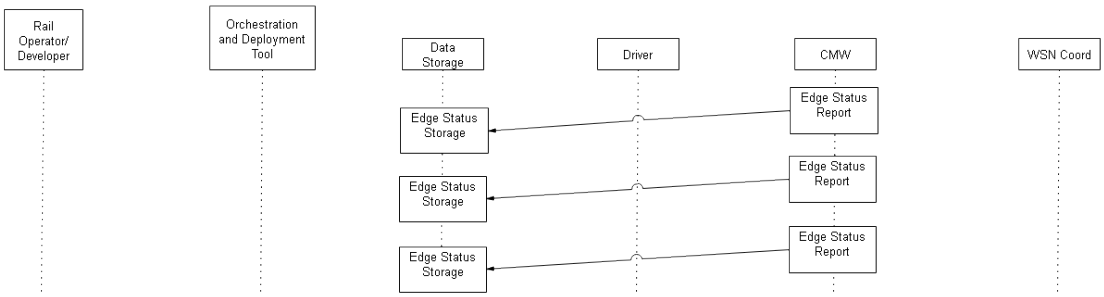


Figure 19: ENACT Deployment Data Report Procedures. Source: INDRA.

4.1.3 Scenario 3: Simulation and Testing

The integrations works are initiated but pending to be developed in the next iteration.

4.1.4 Scenario 4: Root Cause Analysis

The RCA main objective is to detect several events that may occur based on network level parameters, as it is found under development, the progress that was performed in this scenario is the integration at messaging level.

The current activities performed are summarised into the integration between systems. Several credentials are exchanged in order to enable the both connection to Indra Hybrid Cloud Platform and first exchange of information to develop to the tool tests into the ITS Use Case.

4.1.5 Scenario 5: Security Monitoring

The security monitoring introduces a control part which introduce an extra security layer into the use case at different OSI levels. The information reported is taken by the tool in order to be analysed. Several anomalies are detected following a set of rules that permit to detect several attacks predefined.

The tools related with the Rail Use Case are the Security and Privacy Monitoring and Security and Privacy Control. On the one hand, the tool is able to receive traffic network parameters that are compared with predefined traffic model patterns in order to detect possible attacks, on the other hand, the tool applies possible manual enforcements to correct detected attacks.

The Security and Privacy Monitoring tool is in charge into the Rail Use Case operation on detecting possible threats and attacks that may occur. The end devices are authenticated based on the access system implemented in the Rail Business layer. Therefore, the tool must be able to detect disturbances in the mentioned traffic network parameters based on the Rail Operator criteria enhancing the security analysis.

The current activities performed are summarised into the integration between systems. Several credentials are exchanged in order to enable the both connection to Indra Hybrid Cloud Platform and first exchange of information to develop to the tool tests into the ITS Use Case.

Moreover, the Rail infrastructure operates different functionalities that are considered as safe. It is specified not to apply security measurements if any failure is detected due to the strong Rail regulation. The only mitigation action that is indicated is the report of the failures to the Rail Operator. However, other operations are not considered as safe and can be targets for the tool. These operations are related with the Logistics and Maintenance tasks. Once the Security and Privacy Control tool detects the attack and its origin, it can provide the order to the ITS Use Case Hybrid Cloud Platform to revoke the user access to the system.

The revoke access process is performed into the ITS Use Case, when the tool provides the order to revoke a specific user. The ITS Use Case authentication methods are divided into different authority layers that permit to store the access permission per user and per layer. The user is revoked at the most centralized authentication authority and replicated at every layer of the system. At the edge, the revoked user tries to access to the gateways that check the access permissions. In case of a not valid permission, neither the user nor the device can access to the system.

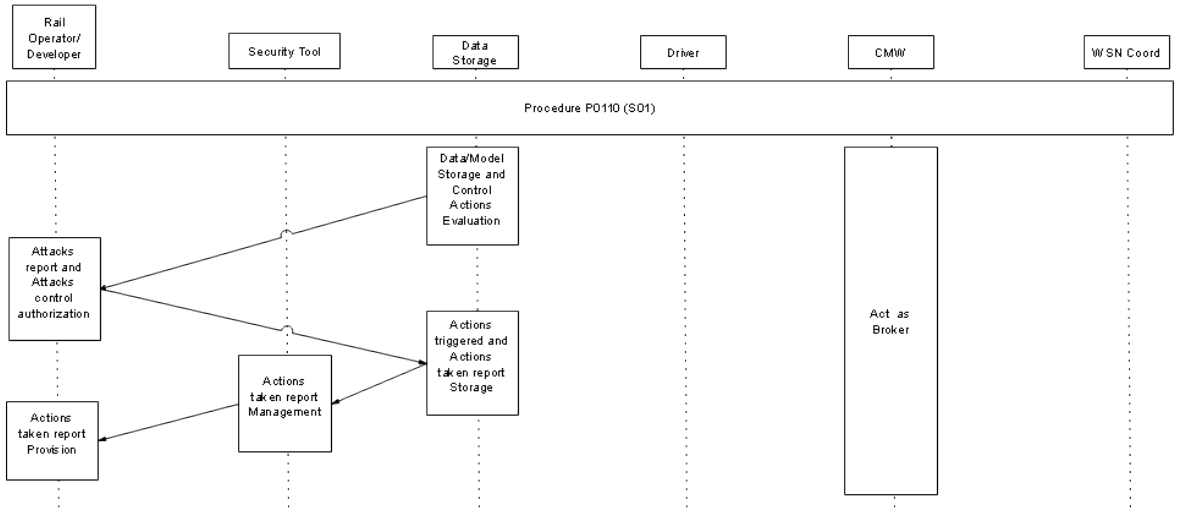


Figure 20: ENACT Security Procedures. Source: INDRA.

The procedures that describe the Security steps besides the data report implemented are shown in the Figure 19 and Figure 20. The Rail Manager control the tool as it is done in the other cases while the tool and the system interact through the data stored.

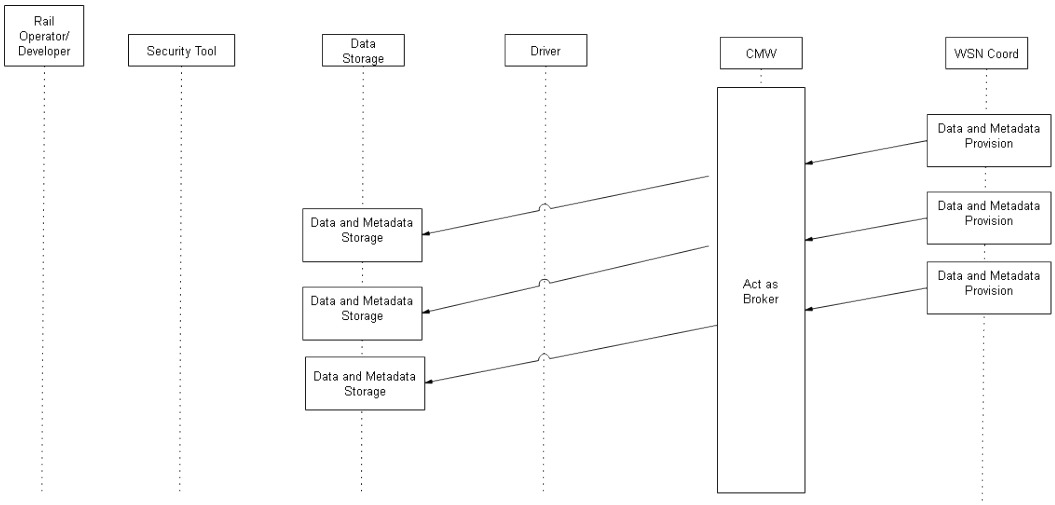
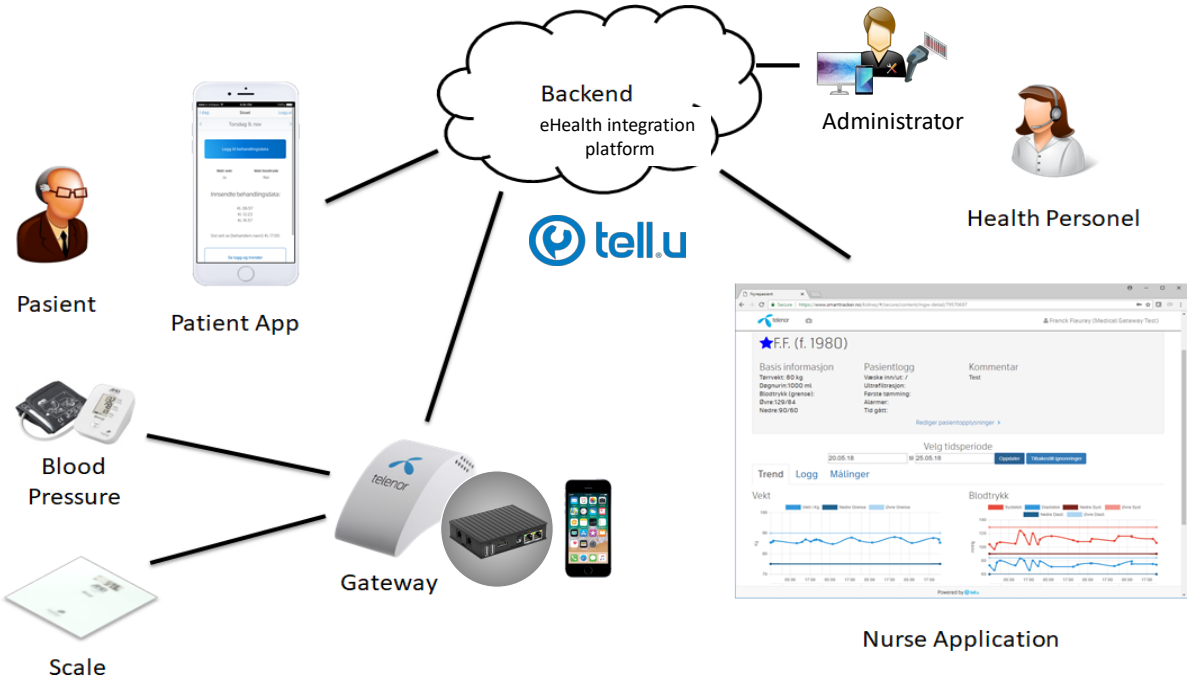


Figure 21: ENACT Security Data Report Procedures. Source: INDRA.

4.2 Digital Health

4.2.1 Architectural design

The system provides a solution for the remote monitoring of patients. Its purpose is to improve and facilitate the communication between the patients and the care providers following them.



The main components of the system are:

- A backend system to manage and store health providers and patient profiles as well as patient medical measurements (e.g., weight, blood pressure, glucose and dialyse measures) and questionnaires.
- A Health Personnel application to create and update patient profiles and view patient medical measurements (in tables and graphs over time).
- A patient mobile application to view their medical data and manually input response to questionnaires and possibly measurements (that are not automatically measured) from medical devices.
- A Personal Health Gateway installed in the patient home to automatically collect measurements taken with Bluetooth connected medical devices.

The types of users of the system and what they do are:

- **Health Personnel.** Hospital nurses, doctors etc. They can create, update and delete care plans and follow up their patients by process tasks associated with patients and their health status. Health status of patients are reviewed for example by browsing their medical data using tables and graphs.
- **Administrators.** Health provider personnel that have administrator rights for the system. They can create, update and delete patients and devices in the system and associate packages of devices with patients.

- **Patients.** The patients apply the system to collect and log their daily medical measurements and to communicate them to the Health Personnel. The patient application allows logging the data provided by the medical devices. The measurement collected by the gateway are not stored locally but pushed to the backend whenever the measurement is collected. The patient app also allows patients to browse their historical data.

The following figure provides a more detailed view of the different components, networks and communication links used by the application. In the development of the system a special attention has been put to protect the patient data both in the normal operation of the system and in cases where parts of the system can be compromised.

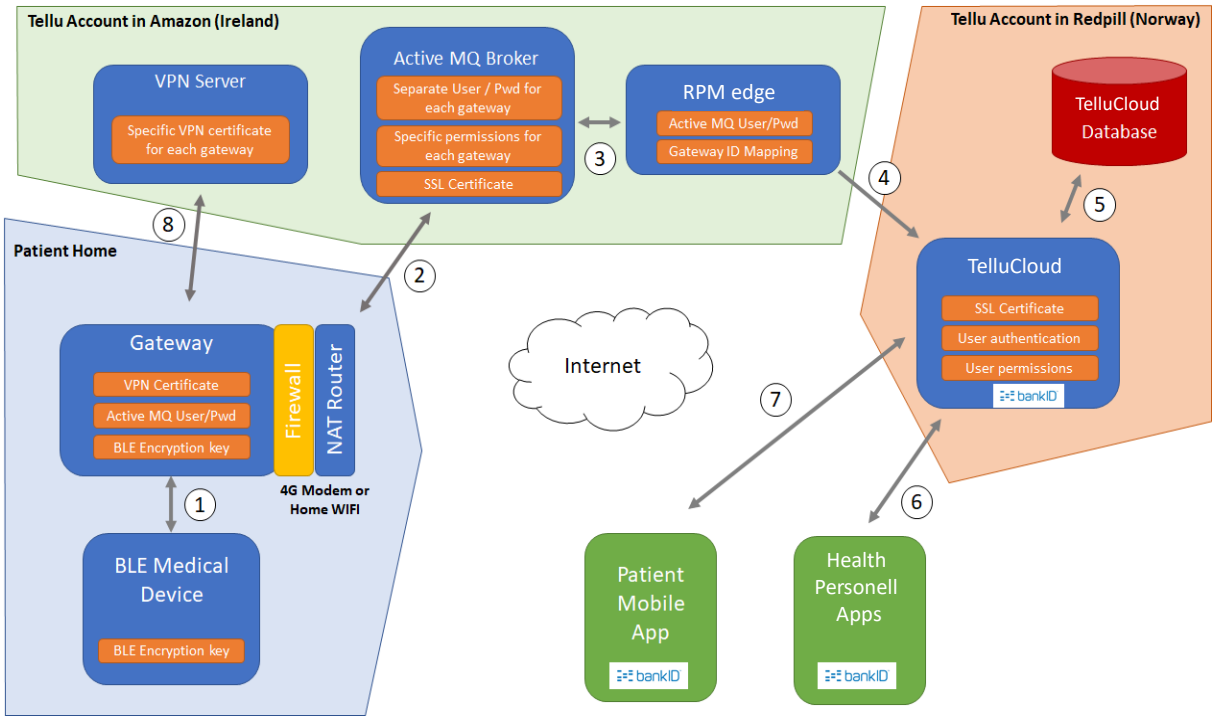


Figure 22 Overall architecture of the Remote Patient Monitoring Use Case. Source: TellU.

4.2.1.1 Description of the components

- **BLE Medical Devices:** These are the blood pressure sensor and scale provided to the patients. They are commercially available medical devices with a BLE (Bluetooth Low Energy) interface. The devices provided to the patients are pre-paired with a Gateway.

- **Gateway:** The gateway is in charge of collecting the measurement from the devices and transmitting them to the backend. The gateway is designed to be powered and connected to the backend. It is listening for measurements coming from the devices it is paired with. It is not scanning for other Bluetooth devices and will not accept connections from other devices than the ones specified in the system by the administrator. It is connected to the internet using an internal 4G modem and a SIM card. If the patient home is not covered with a compatible 4G signal, the gateway can be connected via WIFI to the home network. In any cases, the gateway itself is behind a NAT and firewall to ensure that it is not accepting any incoming connection from the internet. It communicates to the backend (ActiveMQ) and connects to an administration VPN to be remotely administered. Over the VPN, the gateways expose an SSH server which allows to login using an administration private key. Logging in with user and password is disabled.

- **ActiveMQ Broker:** The ActiveMQ broker is the message broker which links the gateways with the backend. It is running in Tellu Amazon private cloud. A gateway name and a randomly generated password are created for each gateway and permissions are set so that each gateway can only access its own message queues (MQTT topics).

- **RPM Edge**: The edge component subscribes for the incoming messages from all gateway through the ActiveMQ broker. Its role is to collect the measurements, forward them to the TelluCloud backend and acknowledge the measurements. In addition to the measurements, the edge also collects and acknowledge gateway heartbeat messages.

- **TelluCloud**: TelluCloud is the backend which links all components together, it is the only component of the system in which data is stored. The TelluCloud account includes the specifications of the users (patients and health personnel), the devices, gateways, interoperations, expected measurements, etc. From the RPM edge TelluCloud is accessed through a "post only" service: data can be sent in, but no data can be retrieved. From the patient and Health Personnel application TelluCloud is accessed over APIs. The API can only be accessed with a valid authentication token which is created using BankID. The token created for a specific user only provides permissions to access the data relevant to that user through the API. The patients can only access their own data and the health personnel can only access the data of his/her own patients.

- **VPN Server**: The VPN server is used for the administration of the gateways. Whenever they are online the gateways connect to the VPN server. Each gateway connects with its own certificate and exposes an SSH server on the VPN. This allows logging into the gateway remotely in order to perform maintenance tasks: security updates, software updates, pairing of devices, etc. The SSH server only allows logging in using a Tellu administration certificate. User/password authentication is disabled. The VPN server is running in Tellu Amazon private cloud, it is dedicated to the patient gateways and only administrators from Tellu can log to this VPN and have access to the private key which allows the remote login to the gateways.

4.2.1.2 Description of the communication links

1. **BLE Medical devices - Gateway**: The gateway uses BLE to collect measurement from the medical devices. During the paring of the device with the gateway (which is done before the gateway is given to a patient), the device and gateway generate and exchange an encryption key. This key is used to encrypt all the communications from the gateway and devices in order to protect the data which is exchanged.

2. **Gateway - Backend (ActiveMQ Broker)**: The communication is encrypted using SSL (similar to HTTPS) and each gateway is authenticated with a username and password which gives it permission to post data only in its specific topic, i.e. one gateway cannot post in another gateway topic and cannot retrieve an event from other gateways. This is important to make sure that even if one gateway is compromised, its credentials do not give access to any other parts of the system. The gateway user name is the gateway host name and each password is randomly generated.

3. **ActiveMQ Broker – RPM Edge:** The communication is encrypted with SSL and is internal to the Tellu virtual private cloud network in Amazon. The Edge is authenticated with a username and password and it has access to data coming from all gateways.

4. **RPM Edge – TelluCloud:** This links the gateway part of the backend with the TelluCloud backend which stores the data. This is a one-way link: the edge posts the data to TelluCloud but cannot retrieve any data. The data is sent over a rest API and encrypted with HTTPS.

Pseudonymization is used to map the gateway ID (which is known by the gateways and printed on the gateways) to IDs which are only known in the RPM Edge component and in TelluCloud. This prevents any external source to post data to TelluCloud.

5. **TelluCloud - Database:** This link is internal to the Tellu private cloud in Redpill. The database is configured to only accept connections from the machines running TelluCloud and which are on the same private network.

6. **TelluCloud – Health Personnel App:** The Health Personnel application is a web application. To login, the Health Personnel needs to be registered in TelluCloud. The authentication is done through bankID in order to guarantee high level of security. Once logged in the Health Personnel app uses a token over HTTPS to access the TelluCloud APIs. Each Health Personnel is only given access to its own set of patients (through the APIs).
7. **TelluCloud- Patient App:** The patient app is a mobile app. To login the patient user has to be created by Health Personnel with administrator rights. The patient profile is created using the patient personal security number. Login to the app is done with bankID. When authenticated the patient can access its data over the Shepherds REST APIs, communicated over HTTPS. Patients permissions are setup so that they can only access their own data over the API.

8. **Gateway - VPN Server**: The communication between the gateway and the administration VPN is encrypted and each gateway uses its own certificate to connect to the VPN. A Certificate can be used only by one device at the time and can easily be revoked if a gateway is compromised.

4.2.1.3 Storage of patient data

There are only two places where measurements are stored:

- **In the medical device**: In case a measurement is made but not collected by the gateway, it is buffered/stored on the device and collected by the gateway the next time the device connects (e.g., after another measurement has be taken). Because the devices are paired with the gateway, only the gateway can retrieve those buffered measurements. Once the data are retrieved and pushed forward by the gateway the measurements are deleted from the device.

- **In the TelluCloud database**: This is the only place where the measurements are permanently stored. The database is running in the Tellu private cloud provided by Redpill Linpro, it is physically located in Norway and only accessible from the TelluCloud servers which are running on the same network.

All other components of the system (gateway, message broker, edge) only do transporting of data and they do not buffer or store data. This ensures that even if compromised, these parts of the system will not contain any data.

4.2.1.4 System conformity

- The system architecture and software are developed in accordance to the GDPR and includes state of the art security features to ensures the protection of the patient personal and medical data.
- All medical devices used in the system (dialysis machine, blood pressure sensor and scale) are commercially available medical devices.
- The system software is not classified as medical device software (ISO 13485 / IEC 62304). Its deployment has to be limited to usages which do not require a medical device certification.

Note: In the context of ENACT and validation of ENACT results, the use case will not involve real patients and real patient data, only test data will be used.

4.2.2 Scenario 1: Initial Software deployment and GW onboarding

In the following subsections a set of DevOps scenarios for the eHealth use case are described. The current status implementing and applying these scenarios are briefly reported in the table listed in the beginning of Section 4 (in figure 16). The scenarios will be further evolved in the next phase of the project based on the new knowledge attained from the ENACT research and from applying the ENACT enablers in the context of these scenarios.

Actors	**Operator**
Description	Initial Software deployment and GW onboarding
Trigger	A new Gateway is provided to a Patient. A Factory reset is performed on the Gateway.
Normal flow	- GW delivered with generic image containing DevOps agent. - GW identifies to a server with its Gateway ID. - If the Gateway ID is OK/Confirmed the GW retrieves credentials and certificates, to connect to VPN backend. - GW identifies to a server in the VPN and retrieves configuration and SW components. - DevOps agent monitors the execution of the components and reports to DevOps backend.
Alternate / Exception flow	A: - GW is not registered. - No VPN key is provided, the GW needs to be registered in the backend before it can be allowed to connect to the VPN and get to the configuration server. B: - SW components do not start up properly/SW components stop working. - Operator is notified. C: - Unreliable connection between the GW and the backend. - Operator is notified, and process is put on hold until reliable connection is in place.
Post condition	GW is running or operator has been notified on exceptions.
ENACT DevOps Framework	- Need to secure the whole process. - Need continuous deployment support. - Framework should provide meaningful information to the operator to diagnose the status (not push detailed log of all components to the operator, but extract meaningful information)
Enabler	**Orchestration and Continuous Deployment Enabler** Security and Privacy Monitoring and Control Enabler Test, Emulation and Simulation Enabler

4.2.3 Scenario 2: Software deployment and evolution in the Gateway

Actors	**Operator**
Description	Software deployment and evolution in the Gateway
Trigger	A change of any of the services running on the GW or an upgrade of an existing component
Normal flow	- GW downloads new versions of the service components from the Binary Repository (continuous process). - Operator triggers the change or update of a component (which is locally available) (Coming from the Developer providing the software updates/change) - GW stops components to be updated and deploys the new configuration (only redeploy the components that are changed). - GW starts the new configuration. - Backend checks that the new configuration is running as specified by the operator.
Alternate / Exception flow	- GW fails to download new version. - New version is not provided as an option to run (for the Operator). - Deployment of the new configuration fails. - GW rolls back to the previous configuration and notifies the Operator (Roll back is not dependent on downloading something as the previous versions are part of the Binary Repository that is already synchronized with the GW) - Backend detects an abnormal behaviour when validating the new configuration. Backend triggers a rollback to the previous version and Operator is notified.
Post condition	GW is running the new configuration or Operator has been notified of any exceptions.
ENACT DevOps Framework	- Need to secure the whole process. - Need continuous deployment support. - Framework should provide meaningful information to the operator to diagnose the status.
Enabler	Orchestration and Continuous Deployment Enabler Security and Privacy Monitoring and Control Enabler Test, Emulation and Simulation Enabler

4.2.4 Scenario 3: Gateway recovery and factory reset in the field

Actors	**Operator**
Description	Gateway recovery and factory reset in the field. The GW is reassigned to another user or is malfunctioning and cannot be fixed by normal redeployment of the continuous deployment framework.

Trigger	The Operator requests a Factory reset remotely or tells the user to execute the Factory reset procedure (e.g., long press on the GW button)
Normal flow	- Trigger - The image of the GW is restored to its Factory image. - On-Boarding procedure is followed as if it was a new GW.
Alternate / Exception flow	- GW recovery and factory reset are not working. - Retry the procedure a second time. - GW is shipped for service.
Post condition	GW is recovered and operating as new (either "directly" or after service is performed)
ENACT DevOps Framework	Ability to trigger remotely low level reset function (without relying on the upper level software stack and DevOps framework to be operational)
Enabler	Robustness & Resilience Enabler

4.2.5 Scenario 4: Software testing on the Gateway

Actors	**Developer**
Description	Software testing
Trigger	Developer has made a new version of the Software that needs to be tested before it is made available in the Production Repository.
Normal flow	- Code is deployed on a test infrastructure that consists of both actual devices/sensors and simulated devices/sensors (either by mocks or by using simulator) - Application specific test suite is executed. - Not all tests can be automated and manual tests have to be run. For example, for Blood pressure measurement one would like to pair the device and measure sending/transport (hard to have a "patient" in the loop in automatic and continuous testing). - Generic Monitoring probes are used to verify that the components are running as expected. - Results of the tests are provided to the Developer.
Alternate / Exception flow	N/A
Post condition	Software is validated
ENACT DevOps Framework	- Risk driven development process and software validation. - Test results are provided in a way that is easily exploitable by the developer (e.g., highlighting regressions and giving easy access to trace of the failing tests) - Need to be able to run and rerun an arbitrary selection of tests. - How to in general create mocks for physical/manual processes (like device pairing)
Enabler	Test, Emulation and Simulation Enabler Risk-Driven Decision Support Enabler

4.2.6 Scenario 5: Continuous integration of gateway modules and versioning

Actors	**Developer**
Description	Continuous integration of gateway modules and versioning
Trigger	Developer commits new code or tag a new release
Normal flow	- The code is pulled by the continuous integration server. - Modules are distributed to be built on appropriate executors (e.g. ARM gateways need code to be built on ARM servers). - Binary modules are pushed to distribution server (similar to a Maven repository for Java). This should support versioning and distinguish between snapshots and releases. For example, using Git repository. - Binaries are synchronized by test gateways. - A test suite is executed. - Validated binaries are made available for production gateways within the Distribution Repository.
Alternate / Exception flow	A: - Code does not compile. - Notification to developer. - Binaries are not made available within the distribution repository. B: - Test fails. - Notification to developer. - Binaries are not made available within the distribution repository.
Post condition	Compilation is triggered automatically for every commit in the source repository. Binaries are only made available if they pass the integration test.
ENACT DevOps Framework	- Need to manage code modules that are built for different architectures from microcontrollers to ARM based devices with specific software libraries and hardware.
Enabler	Test, Emulation and Simulation Enabler Robustness & Resilience Enabler Orchestration and Continuous Deployment Enabler

4.2.7 Scenario 6: Trustworthiness of the medical system

Actors	**Developer and Operator**
Description	Security and Privacy
Trigger	N/A
Normal flow	- Application specific Risk analysis regarding threats in general related to trustworthiness and management of sensitive data in the digital health domain. - All communications with back end are encrypted

	- Encryption keys not shared between GWs and tied to specific GW ID. - Network set up preventing connections between GWs. - GW is not storing or retrieving any patient data. GW is not storing log of measures. Buffered data is encrypted using a backend public key (can only be decrypted by the backend).
Alternate / Exception flow	- No connection: Encryption key is from the server (and resides only there). - No Change of data (checksums) - Key management
Post condition	Trustworthy System
ENACT DevOps Framework	Automatic generation and secure distribution of encryption keys. Possibility to revoke certificate and encryption keys to isolate GWs. Developing and maintaining a Risk Analysis for the trustworthiness of an IoT based system in a system lifecycle perspective.
Enabler	Security and Privacy Monitoring and Control Enabler Risk-Driven Decision Support Enabler Robustness & Resilience Enabler

4.3 Smart Building

The DevOps scenarios for the Smart Building use case show frequent interactions of development and operation of applications for energy efficiency and user comfort that involve multiple IoT sensors and actuators. Those development and operation cycles jeopardize the trustworthiness of the whole IoT environment. The main DevOps scenarios identified for this use case are the proper orchestration and continuous deployment of new IoT components, the resolution of conflicts between different applications due to contradictory orders sent to the actuators or to the actuation on the same physical variable, the assurance of the integrity and confidentiality of the network data and the self-optimization of the thermal comfort control.

4.3.1 Scenario 1: Thermal comfort control – heating design

The frequent changing in the thermal control strategy of the building needs the identification of threats and the selection of security controls to minimize risks by the Risk-Driven Decision Support Enabler, and the safe deployment of IoT application software pieces by the Orchestration and Continuous Deployment Enabler.

The integration works for the Risk-Driven Decision Support Enabler are initiated but pending to be developed in the next iteration.

The Orchestration and Continuous Deployment Enabler is represented by GeneSIS tool. ThingML models have been integrated with the SMOOL middleware (Tecnalia's extension of SOFIA) to be automatically deployed by GeneSIS. This has already been documented in D2.2. SMOOL Knowledge Processors (KP) are the IoT end points of the smart applications, e.g. a temperature sensor or a heating actuator. A SMOOL KP can be deployed by GeneSIS as any other software component. In addition, we integrated the SMOOL KP wizard with ThingML. As a result, a single Eclipse IDE can be used to generate the code of a KP, which can then be directly used as part of a ThingML program. The proper Maven manifests are automatically created facilitating the building and release of the desired application. Details about this integration can be seen in the following video: https://www.youtube.com/watch?v=mfT_AwfkXNc .

The Orchestration and Continuous Deployment Enabler has also been used with the Actuation Conflict Manager Enabler in Scenario 2.

4.3.2 Scenario 2: Thermal comfort control – conflict in heating actuator use

Two or more applications that send different temperature preferences to the same heating actuator will cause a fluctuating operation of the thermal control, this is an issue that must be solved in the development of the applications by the Context Monitoring and Actuation Conflict Management Enabler. The proper orchestration of devices affected by these applications is done through the Orchestration and Continuous Deployment Enabler.

The Actuation Conflict Manager (ACM) Enabler is an application created to help the software developer to solve actuation conflicts, including when two or more different concurrent IoT apps try to send simultaneously conflicting orders. The ACM Enabler import the model of the IoT system created using the GeneSIS software. GeneSIS is a tool developed by SINTEF for the software deployment of Smart IoT Systems. With GeneSIS, you can create your deployment model by adding components and links. For the moment, our preferred way to work with GeneSIS is by deploying a docker component that contains the Node- RED workflows of the IoT system. Node-RED is a visual flow-based programming environment designed by IBM for the Internet of Things. Although several model formats can be

imported into the ACM Enabler, Node-RED and GeneSIS are the main tools designed to be used with the ACM solution.

Once everything is imported and set up in the ACM Enabler, a click on "find conflicts" button automatically detects actuation conflicts and will add the ACM component in the model where conflicts might happen. However, they still need manual configuration. By clicking on the ACM component, several solutions are proposed to help resolving the detected actuation conflict. The user must then choose the right solution and the ACM Enabler is automatically configured according to the developer choice. Once each component has been properly configured, a new conflict free model in the form of a Node-RED or GeneSIS file is created, and the solution can be deployed again.

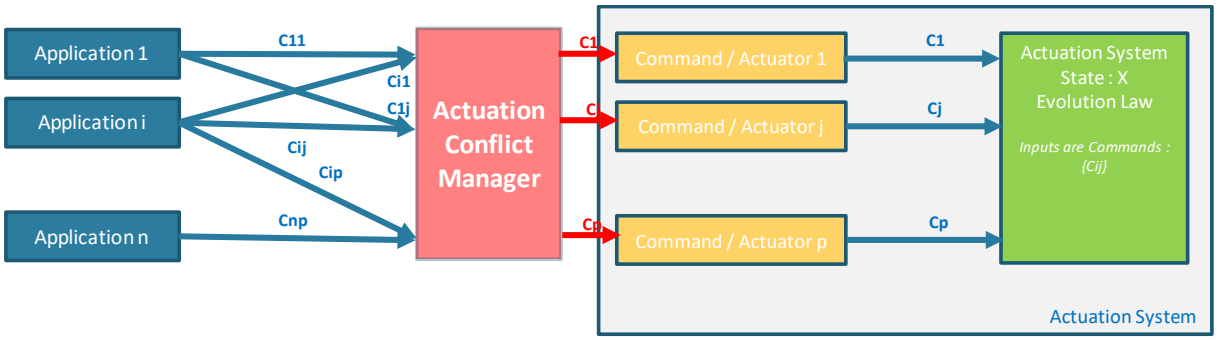


Figure 23: Actuation conflict handling of the ACM Enabler. Source: CNRS.

A direct conflict occurs when two different applications try to access the same node. Basically, two applications accessing the same actuator might send contradictory commands resulting in an indeterministic behaviour.

An example of direct actuation conflict has been programmed and tested in the ACM Enabler. This example is represented by Figure 24 and consists in two IoT applications that are fed with temperature values from two different sensors and both applications try to change the state of the actuator depending on the moment when those temperature values reach a threshold mimicking the operation of a thermostat. The temperature sensors are located at different spots in the room and therefore different temperature values are potentially obtained. In the current version, the actuator behaviour is similar to a relay that can switch ON and OFF thermal heater. The Actuation Conflict Manager Enabler provide different solutions by giving priority to one app or by including an AND / OR logic element to the combination of actuation signals.

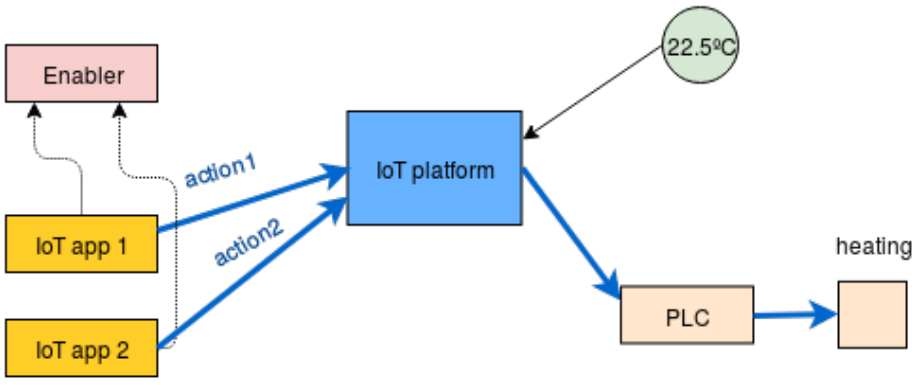


Figure 24: Thermal comfort control – conflict in heating actuator use. Source: TECNALIA.

The Node-RED workflow of Figure 25 is used to model the system. In this figure, two identical subflows representing sensors process can be found. Both subflows are accessing the "command" node to finally send an actuation command to the same heating actuator.

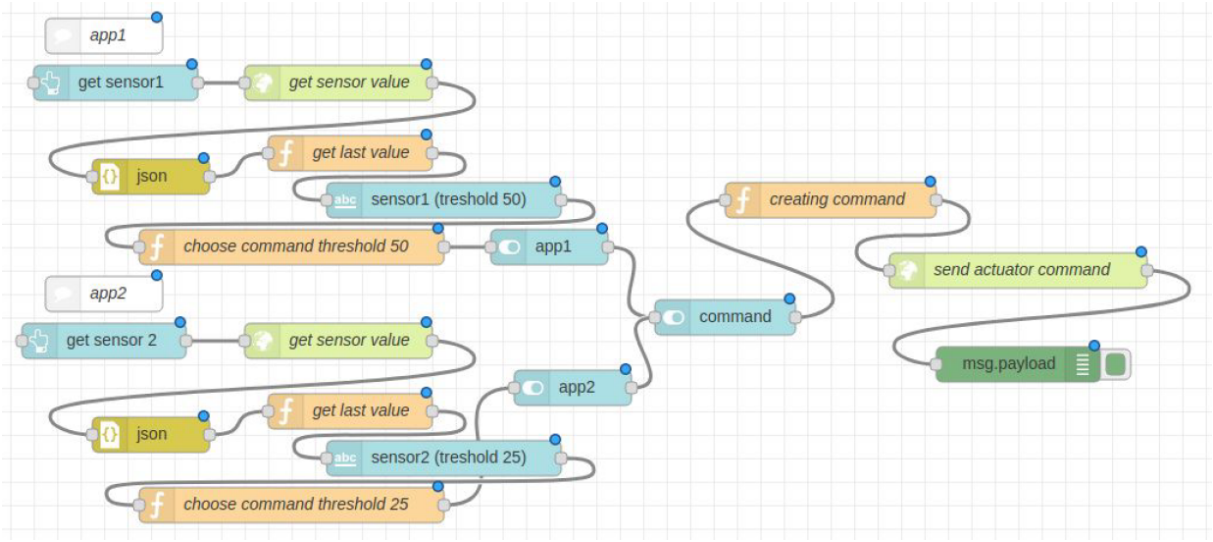


Figure 25: Node-RED example of using the ACM Enabler. Source: TECNALIA.

4.3.3 Scenario 3: Thermal comfort control – conflict in temperature actuation

The running of two or more applications that send actuation orders to different devices can collide when the thermal control of the building is affected, e.g. setting a high temperature setpoint to the HVAC and opening a window. For this reason, a model of the physical system must be incorporated into the thermal control, but this model is subject to differences or uncertainties between the physical model and the real world. This problem must be solved by the Context Monitoring and Actuation Conflict Management Enabler both in design and operation time. The proper orchestration of devices affected by these applications is done through the Orchestration and Continuous Deployment Enabler.

An indirect actuation conflict occurs when two different IoT apps managing two different actuators would like to simultaneous act over the same physical variable (e.g. room temperature), e.g. one by opening the window (lower temperature) and the other by rising the setpoint temperature in the HVAC control system (higher temperature).

The Behavioural Drift Analysis (BDA) Enabler deals with this type of conflict by correcting the operation of a real physical system subject to uncertainties. The BDA Enabler also imports the model of the IoT system created using GeneSIS and Node-RED tools. At the end of the importation phase into the BDA Enabler, a physical process configuration must be specified. This will allow to establish the interaction between a logical node and the environment. For instance, if the last node of a flow represents the activation of a light, it might be linked to a physical process representing the luminosity. The main utility of this process is the possibility to find indirect conflict through the model. If two different actions are linked to the same physical process, it may be an unplanned conflict that should be solved.

The integration works of this scenario are initiated but pending to be developed in the next iteration.

4.3.4 Scenario 4: Smart building Alerts for User Comfort

The communication of building alerts must be done assuring the integrity and confidentiality of the data and avoiding alarm tampering by external people. The Security and Privacy Monitoring and Control Enabler is used to assure the integrity and privacy of the communications through two services: the

Security and Privacy Monitoring does the surveillance of data security and the Security and Privacy Adaptation performs data security enforcement.

This scenario tackles the problem of Smart Building alerts using IoT apps that monitor different aspects of the building, such as an abnormally high or low temperature, the presence of smoke or flood. The Security and Privacy Monitoring and Control (S&P Mon&Control) Enabler is used for assuring that the alarm system is not tampered by external people and safeguard people's privacy. These enablers have been applied to the Z-Wave wireless communication architecture described in annex A.1.3.1 and illustrated in Figure 31.

The S&P Mon&Control Enabler uses port mirroring to send a copy of the network packets that contain sensitive information in the Smart Building to a server where the Enabler is running. Three different nodes in the Smart Building communication architecture have been mirrored: the SMOOL communication broker, the Raspberry Pi that is used as a gateway by SMOOL and the Scada node of the Building Management System of the Kubik building. The three mirrored ports in the Smart Building communication architecture are shown in Figure 26.

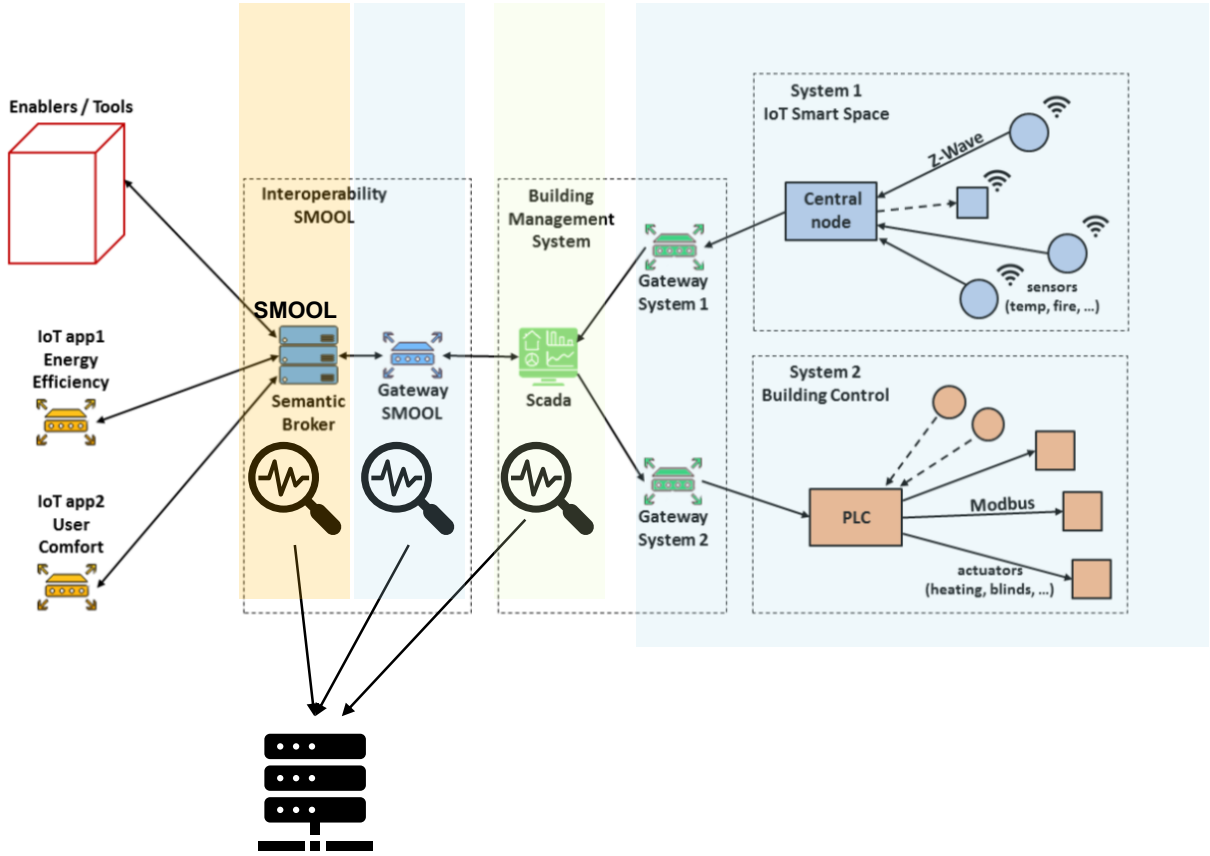


Figure 26: Smart Building nodes monitored by S&P Mon&Control Enabler. Source: TECNALIA.

In the sensing scenario of Figure 27, the S&P Mon&Control Enabler checks if the messages are sent by authorized nodes that belong to the Kubik architecture. If an IP of an external device tries to send or receive a message from the monitored nodes, the S&P Mon&Control detects it and notifies the system administrator blocking network traffic if necessary. The S&P Mon&Control Enabler also ensures the integrity of the sensing data blocking any attempt of tampering the message.

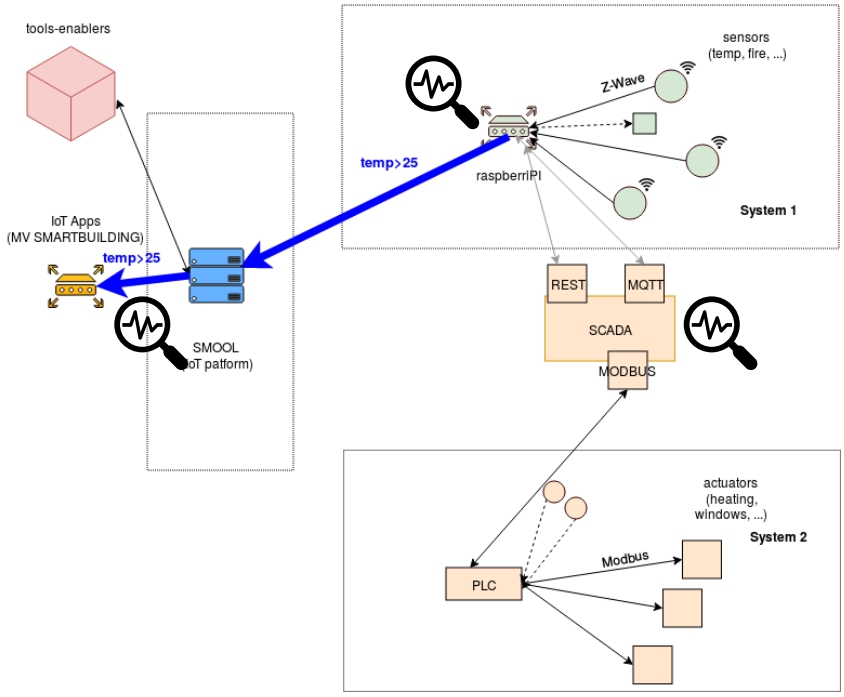


Figure 27: Sensing scenario monitored by S&P Mon&Control Enabler. Source: TECNALIA.

Although the Scenario 4 is only intended for sensor devices that trigger alarm messages or alerts, this alarm can also trigger a security action or siren. In the simplest case, this security action will consist in a ON / OFF command to a siren as illustrated in Figure 28.

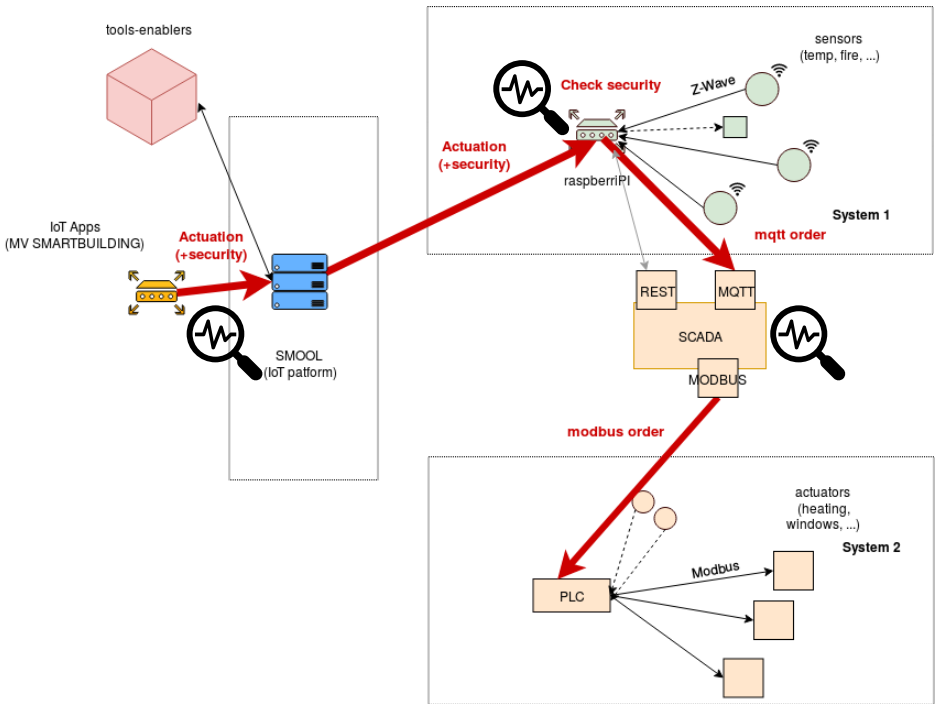


Figure 28: Actuation scenario monitored by S&P Mon&Control Enabler. Source: TECNALIA.

The security monitoring of the Kubik network has already been performed using the S&P Mon & Control Enabler, while the Security and Privacy Adaptation Enabler will be tested in the next iteration to enforce communications security.

4.3.5 Scenario 5: Thermal comfort control – self-optimizing controller design

The thermal comfort control of a building is a trade-off between user comfort and energy consumption, e.g. the heating may be turned off when the room or building is not occupied. However, the thermal inertia of the building makes this thermal control a real challenge. The Context-Aware Self-Adaptation Enabler will be used to find and update the optimal control parameters at run time through the Online Learning tool. The Orchestration and Continuous Deployment Enabler will also be used to properly deploy those control parameters and orchestrate the involved IoT devices.

The integration works for the Online Learning Enabler and the Orchestration and Continuous Deployment Enabler are pending to be developed in the next iteration.

5 Conclusion

In the first stage of the case studies implementation, the use case provider partners focused on developing trustworthy and ready-to-use scenarios to test the DevOps Enablers and the final validation of the integrated ENACT DevOps framework. An initial group of DevOps Enablers have been tested and the works for testing and validating the Enablers under development have been initiated.

From the ITS use case perspective, the implementation part is considered enough to start a further implementation with the Enabler tools. This can be probed as first approaches are taken with three ENACT tools with no compatibility issues. It must be highlighted that the system is designed to provide the maximum possible flexibility into a highly safe and secure environment. Further improvements into the functionalities are performed to enable the integration with the rest of the planned tools for the ITS use case.

In the eHealth use case a main effort has been on implementing the new remote patient monitoring system and to investigate and learn how to be able to apply DevOps principles for its operation and evolution based on ENACT concepts and state of the art tools. This has resulted in a reference implementation where trustworthy DevOps is partly supported based on ENACT concepts and tooling (e.g., ThingML, Risk Management concepts, Context aware access control etc) as well as some state of the art DevOps tools such as Ansible, Jenkins, Jira, PagerDuty, ZenDesk, Grafana and Prometheus. This reference implementation will be the baseline when integrating and validating further the ENACT DevOps enablers in the next phase of the project.

In the Smart Building use case, the implementation done in Kubik building comprises two different systems that interoperate using the Interoperability SMOOL IoT middleware and the Building Management System or BMS. These two differentiated systems are the IoT Smart Space (system 1) composed by new wireless IoT devices and the Building Control Space (system 2) composed by the legacy building control systems of the building. This architecture enables the operation of complex Smart Energy Efficiency and Smart User Comfort applications to test the Enablers developed in the project and validate the ENACT DevOps framework.

6 References

[1] ENACT D2.2 - Risk-driven Continuous Delivery of Trustworthy Smart IoT Systems — First Version
[2] GS1 - RFID in RAIL – European Guideline for the Identification of Railway Assets using GS1 Standards

A ANNEX

A.1 DESIGN

This section includes additional explanations to the architecture design of some of the use cases and documentation to justify it.

A.1.1 ITS Use Case

This section is intended to further explain the architecture design at the section 3.1.1. The design is intended to cover the On Track and On Board part in a rail environment. To this project the Train Integrity, Train Inauguration and Logistics and Maintenance are covered from both On Track and On Board parts.

As a first step before describing the part detail decisions, the protocols used to enable the system from a big perspective can be divided into messaging protocols, functionalities and general services.

- **Messaging protocols**: The messaging protocols are divided into MQTTS and AMQPS to cover several layers in the OSI stack. The MQTTS protocol is designed to carry the edge information which is considered light in a flexible manner. It is inferred that this protocol is ideal to the Rail infrastructure to cover the future applications and devices to be used beside the current rail systems. Its security layer covers the required need for the security rail protocols. Therefore, it matches perfectly with the project requirements into the ITS Use Case.

  AMQP is considered similar to MQTT. However, it is designed to carry heavier load of information following the similar MQTT design ideas. It is used from the gateway part to the Indra Hybrid Platform as it transports aggregated traffic from all the infrastructure. The security for this protocol covers the required security level for the specified interfaces.

- **Functionalities:** The Train Inauguration and Train Integrity functionalities are considered applications required for the future trend in the rail environment to automatically register and track the rail compositions using wireless technology. These applications are considered secure and safety. Therefore, using the Indra Sistemas SA expertise in this area, both the infrastructure and the data treatment at this stage are considered to obtain a safe wagons detection, wagons that will form the complete composition, and the train integrity track.

  Regarding the Logistics and Maintenance, the application is not considered safety at this stage. In any case, the transmission of these data is redundant to ensure the track location. The Logistics (location) is done based on the Train Integrity data as it contains global positioning data that permits tracking the train in the On Board Case. In the On Track case, the redundant path, it is sent throw RFID technology following the rail European standard [2] and reported to the Cloud throw the On Track gateway. The information related with the Maintenance is only reported throw the On Board part and consists at this stage in the consumption power from the Nodes that are used for the other applications. This data is used to monitor the status of the devices. However, it is not considered critical for the normal system behaviour.

- **General services:** These services are used for every application and infrastructure involved. This includes the tools that are connected throw the Cloud infrastructure. These services authenticate every client that tries to publish or to be subscribed to any information generated into the system using several credentials against an LDAP server. These services also provide a secure path among the messaging clients and the ITS system encrypting and enabling the communications.

The infrastructure is divided into three main stages: Indra Hybrid Cloud Platform, Gateway and Edge part. These are guided by the following criteria:

- **Edge Part:** The On Board part is the main core in the edge part. This system section has its specific On Board Gateway which is in charge to communicate the On Board information generated to the Cloud throw AMQP.

  The On Track part is intended to cover the logistics (Train location) as a complement to the On Board part, which transmits the data throw an On Board Gateway. The wagons have a RFID tag per wagon that stores the train identification data. This information is transmitted by MQTT to the RFID receiver which is connected to the On Track infrastructure throw an On Track Gateway. Several On Track Gateways can form the On Track infrastructure communicating each other by AMQP aggregating traffic. Both On Board and On Track gateways have the same implementation and are subject to interact with the tools.

- **Gateway:** The gateway main philosophy is transmitting the information throw the system independently of the application or the system status. It is considered application and partner agnostic in order to make it flexible to any entity, information type and application run. The only requirement is that the information must be transmitted using the two mentioned messaging protocols, MQTT and AMQP.

  The Gateway is also in charge of providing Wi-Fi and wired connection, and 3GPP connection to enable the maximum communication capabilities to any external authorized entity ensuring these interfaces from a hardware and software perspective considered in the rail environment. The Hardware protection is enabled following the rail standards and the software is done following the general services explained.

- **Indra Hybrid Platform:** The Platform is designed to be configured in two main entities, a private and a public one. The private part is in charge of managing the internal ITS functionalities to preserve the data security. The public part enables the connection to the resto of the systems using an AWS interface to store the data as a replica. This second part is also used as a connection for the tools and other platforms as FIWARE.

A.1.2 E-Health Use Case

The overall architectural design of the system is already presented in Section 4.2. The figure below shows the different components, networks and communication links used by the application. The local patient infrastructure is set up with a local/edge infrastructure consisting of a set of medical devices and a home gateway. A set of such local infrastructures are then connected and aggregated into a cloud based infrastructure. An Important aspect of the setup is the security architecture to have proper security and privacy across the IoT edge and cloud space. Moreover, the system needs to function as expected and in a timely manner, even when technical problems occur.

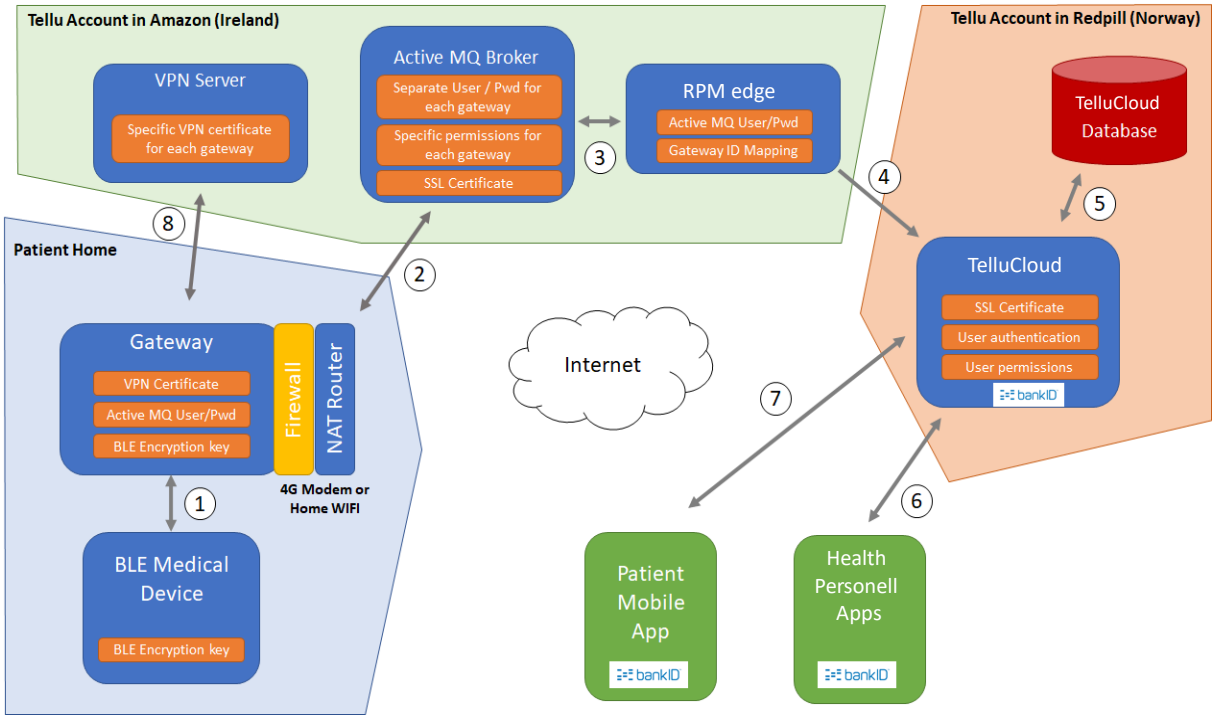


Figure 29 Overall architecture of the Remote Patient Monitoring Use Case. Source: TellU.

The architecture of the edge component with the Gateway as well as the edge controller part of TelluCloud is depicted in the figure below. The Medical gateway is the heart of the architecture, controlling the edge and connecting devices and sensors in the IoT space with the larger ecosystem spanning IoT, edge and cloud., The Gateway itself consists of a set of microservices with a set of gateways for various devices and protocols (e.g., BLE, Glucose meter, blood pressure meter,, scale, etc), the TelluCloud edge for the integration and interaction with the larger ecosystem. The architecture includes independent components for device and GW management. The Gateway includes an MQTT broker and support for standard internet communication protocols. The components run on docker containers

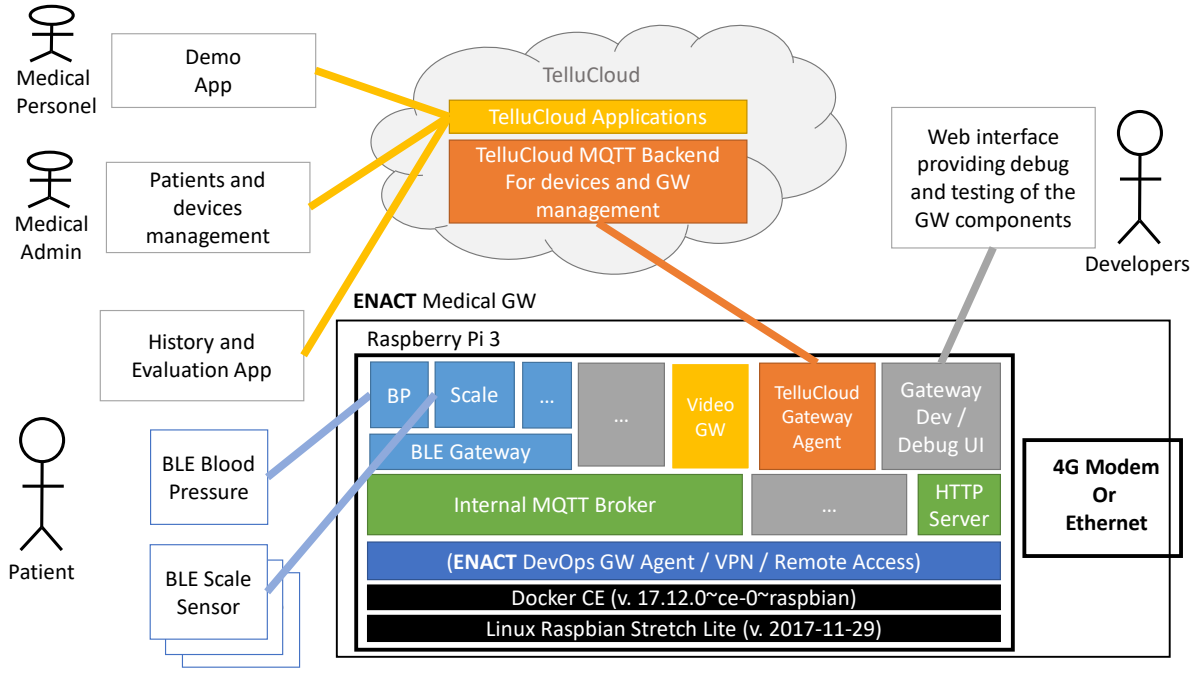


Figure 30 Edge architecture. Source: TellU.

A.1.2.1 Mobile app design

This Section describes the design of the mobile app for patients. The app has a modular architecture. We first describe the overall architecture and platform, and this is followed by the design documentation for each module.

Platform

One of the main design decisions was which technology(s) to base the app development on. It was a clear requirement to support both Android and iOS devices, since both have a significant market share. Web, Android and iOS are all mentioned in the project application. A pure web application is very portable in theory, but it can encounter web browser implementation differences on different systems, and it limits what the app can do. We narrowed the choice down to three options:

Native Android + native iOS: This means a separate app for each platform. An initial Android version would be easy to do, as Tellu has extensive Android experience and has an existing code base to work from for much of the wanted functionality. However, developing and maintaining two separate projects would require a lot of resources, and we don't necessarily want to reuse old code since we need to follow a certified process for this app.

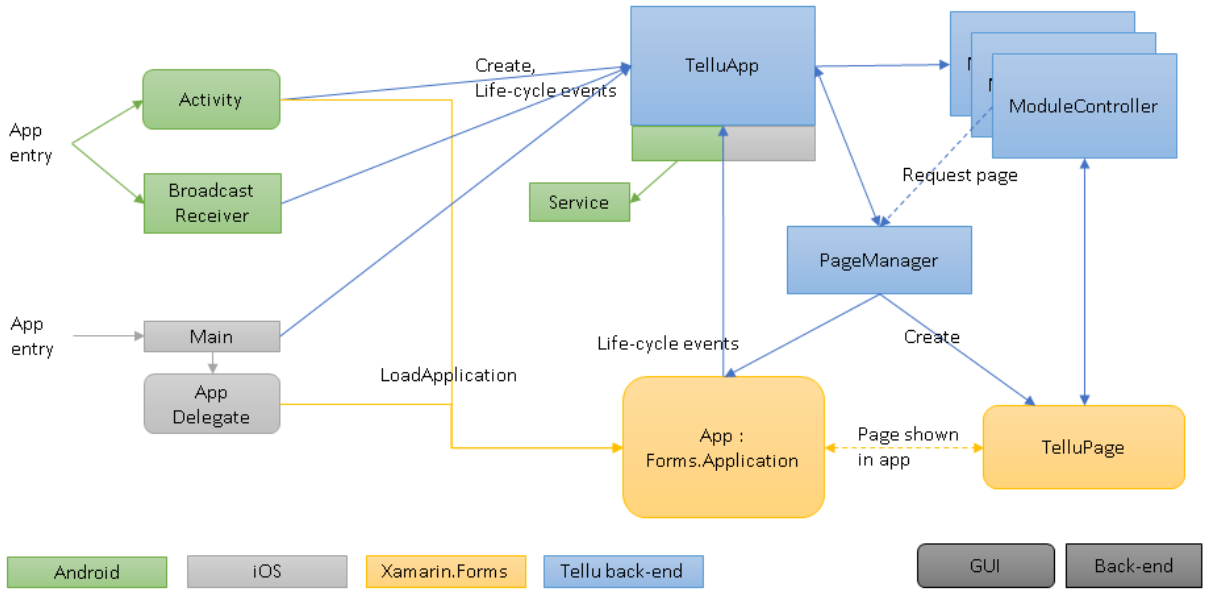
Xamarin for Android + iOS: Xamarin is a framework for writing apps for both platforms in C#. This allows reusing much of the same code for both platforms. Xamarin allows using the native APIs on the respective platforms, and some platform-specific code is needed, but good design can keep this to a minimum. We did not have prior Xamarin experience in the project, but the general impression was that Xamarin is now a mature framework, so we decided to evaluate it further.

HTML frontend + native backends: A common UI can be made with HTML5 technologies. This can be connected to a separate backend component on each platform, for things like Bluetooth communication.

The decision was made to test Xamarin, and it made a good impression. We should distinguish between Xamarin and Xamarin Forms. Xamarin maps all native APIs to C#, and development is done with Microsoft's .Net tools. This allows some shared code, and lets the developer do all work in a single language and toolset. However, all device APIs are still separate, with the GUI typically being the main part of the app to develop in the platform-specific way, so the developer needs a good knowledge of each platform. Xamarin Forms is a common GUI API built on Xamarin. Using Forms, the GUI is built in a platform-independent way, greatly reducing the need for platform-specific code. In our case, we are mostly left with Bluetooth communication, which must be done with platform-specific APIs. It therefore provides the same advantages as the "HTML frontend + native backend" option, but with better performance and device integration, as we get compiled code rather than JavaScript. It is a good middle ground, allowing you to do as much "native" code as you want while providing you with the means to do as much multi-platform code as possible. As our initial tests showed that Xamarin Forms was working well, we decided to use this approach. The only drawback was that it meant having to learn a new language, platform and toolchain, so that it takes longer to get to a first working version of the app.

Overall architecture

We have created a top-level architecture for the app to promote a modular design, a clear distinction between GUI and backend code, and a clear separation of platform-specific and generic code. The figure below shows this architecture. On the left side we have the startup, which is platform-specific (green is Android, grey is iOS). On Android we must implement Android components which are started by the system, while iOS has a main method. Most of the rest of the code is generic, with Xamarin Forms used for the user interface (orange), and Tellu code implementing our architecture patterns (blue).



With these patterns we have an app level and a module level. The app level represents the app as a whole, providing an infrastructure for modules to plug into. TelluApp is the central object, handling the app lifecycle in a generic way. This can have platform-specific extensions, such as using an Android Service component to run in the background. Xamarin Forms provides an Application object which is the top-level component of the user interface. The module level is on the right side of the figure. Each module represents distinct functionality, and the architecture allows plugging in new modules without affecting the rest. Each module has a ModuleController, which can hold the backend logic and state which is independent of a specific GUI page, and it has a set of pages, making up the user interface. Each page inherits from the TelluPage class to plug into the pattern, but it is a regular Xamarin Forms page and can house any Forms elements. As long as it's based on the base classes of the architecture, a module can do what it wants, including usage of platform-specific APIs.

Based on this architecture, the app is split into a set of modules, corresponding with the functionality wanted in the requirements. The user interface of a module is further split into pages, with only one page shown at a time. This gives us a hierarchical way of describing the app as modules and pages. Each module represents distinct functionality and can be fairly well isolated from other modules, but of course a module can have relations to other app modules and to external components. The high-priority requirements have been split into five modules. These are shown in the figure below, with their main relationships. The rest of this chapter describes the modules. In addition to pages and relations, the design of a module includes any data and background processing it needs.

A.1.2.2 TelluCloud Authentication Broker

We have developed TelluCloud Authentication Broker which federates identities provided by other federation services and identity providers. By using the standardized protocols OpenID Connect v1.0[1], TelluCloud Authentication broker can provide identities from a set of federation services and identity providers, including:

- ID-porten: ID-porten is the national identity federation service in Norway that provides identities verified by five different identity providers: MinID, BankID, BankID on mobile, Buypass or Commfides. Identities provided by ID-porten must only be used by public services, or service providers operation on behalf of a public service.

[1] https://openid.net/specs/openid-connect-core-1_0.html

- HelseID: HelseID is an identity federation service provided by Norsk Helsenett for authenticating health personnel in Norwegian e-health solutions. HelseID federates identities from regional health organizations, local identity providers and ID-porten. In addition, HelseID will enrich the identity with information about the user from Helsepersonellregisteret and Personregisteret.

- Social Identity Providers: TelluCloud Authentication Broker can also be configured to federate identity from social identity providers. Currently a number of social identity providers are supported, such as Facebook, Google, LinkedIn, Microsoft, Twitter and PayPal.

Tellu also have sub-contractors to federate identities from ID-providers outside of Norway, including providers in Nordic countries and a set of countries in the EU.

The process of authenticating a user is implemented using the OpenID Connect v1.0 protocol. OpenID Connect is an authentication layer on top of the authorization framework OAuth 2.0. The protocol defines a set of different authentication flows, where the choice of the used flow typically depends on the type of client application and type of user that will be authenticated. For the communication between the service provider and TelluCloud Authentication Broker, authorization code flow, implicit flow and hybrid flow are supported. Communication between TelluCloud Authentication Broker and identity providers use authorization code flow.

A.1.2.3 App OpenID Connect implementation

The mobile app user must authenticate in the web interface of the TelluCloud Authentication Broker. Based on the type of ID used, this could be two-factor authentication. For test and demonstration, we have mainly used our own Tellu ID, as this is a simple username and password login. When designing this aspect of the mobile app, it was important to study and follow current best practices[2]. Many apps have used an embedded web view, so that the web page is shown within the app. This is not good practice, as it allows the app to gain access to the user's input in the web page. The current best practice is to use the external browser component in the device, invoking this from the app and having it redirect back to the app with credentials once the authentication is completed. Protocols for this are implemented on all recent versions of both Android and iOS.

[2] https://tools.ietf.org/html/rfc8252

To implement the protocol in the app, we found a C# library supporting Xamarin apps: IdentityModel.OidcClient2 [3] . This OpenID Connect Client Library is open-source and certified by the OpenID Foundation. When the app does not have valid credentials for using the service, the user presses a button to start the authorization process:

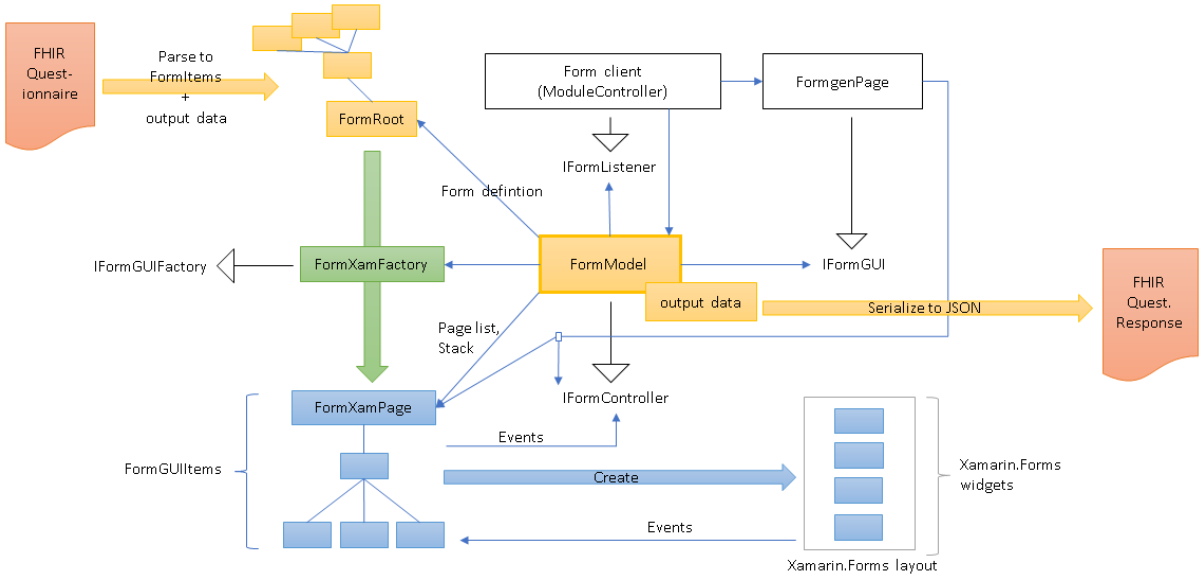
- The browser opens on the TelluCloud Authentication Broker URL set up for this application.
- The user authenticates in the web interface.
- The browser is closed with a redirect to the app, with a TelluCloud authorization token and other session data.

The token is stored in encrypted storage, so that the user does not need to reauthenticate while the token is still valid. Tokens have a limited lifespan, decided by the service which issues them. When no longer valid, a new authentication may be required (the protocol also supports a separate refresh token with a longer lifespan, which can be used to get a new authorization token). Managing the session takes some care, and we need to remember that it is managed separately by the browser and by the app. The browser stores its session, so if the app invokes the browser to do authentication while the browser still has a valid session, it will redirect directly back to the app with the current credentials, not allowing any user interaction. So, we have a logout option in the app, allowing the user to terminate the session. This invokes the browser through the OidcClient library, returning to the app when the logout transaction is completed.

A.1.2.4 Questionnaires

The Questionnaire module implements a generic user interface engine which can render the usual set of UI elements, such as labels, input fields, selection lists and buttons. It has a parser to parse JSON documents according to the FHIR standard for Questionnaire definitions. The idea is that questionnaires are created by medical personnel through the web interface of the system, and these can be tailored to the specific patient if needed. The app retrieves the list of available questionnaires for the logged in patient from the cloud, and the detailed definition is retrieved when the questionnaire is selected. The definition is parsed in the app and transformed to the internal format used by the user interface engine, so that it can be rendered, and the patient can answer the questions. Once done, the results are serialized to a FHIR QuestionnaireResponse JSON document, which is posted to the cloud.

[3] https://github.com/IdentityModel/IdentityModel.OidcClient2



The figure above shows the architecture of the user interface engine. Starting in the top left, we have a FHIR Questionnaire document retrieved from the cloud. Parsing it produces a tree structure of Form nodes, rooted in FormRoot, which is an abstract definition of a user interface suited to forms and questionnaires. This user interface can be organized as one or multiple pages and further divided into groups. The leaf nodes define the input and output controls. The heart of the model is the FormModel object, which in addition to the abstract form tree contains a data structure to hold user input, and meta-data with information about data types, cardinality, etc. Altogether, the FormModel holds everything needed to implement the user interaction, but note that this is still just an abstract model, and is not tied to the Xamarin Forms user interface framework used in this app.

The implementation into Xamarin Forms is shown in the lower part of the figure and has two stages. First, a factory class creates a structure of FormGUIItems, in our case for Xamarin Forms. These objects have two tasks. They create the actual Xamarin Forms UI widgets which make up the screen contents, and they work as controllers for the UI widgets they create, receiving events such as when the user types in an input field or presses a button. They are the link between the model and the UI. Only when the appropriate Xamarin Forms page appears on screen are the FormGUIItems activated, putting content into that page. The rest can exist independently of what is shown on screen.

The data in the model is organized as a FHIR QuestionnaireResponse object. This object has the same hierarchical structure of items as the Questionnaire. When the user presses the submit button in the UI, this part of the model is serialized into a JSON document, which can then be posted to the cloud.

The engine can parse expressions which refer to the data model using an XPath[4] syntax. The full power of this is not used for FHIR Questionnaires, but it is used to implement the *enableWhen* element of FHIR, where parts of the questionnaire are enabled or not based on conditions on answers from other items. This makes it possible to implement dynamic questionnaires which adapt to user input, such as answering one question and then getting different messages or questions based on the answer.

We have so far implemented the most relevant and useful aspects of FHIR. Here is a list of supported features:

- Basic header/meta data elements: id, meta, version, name, title, status, date, description, etc.
- Hierarchical structure of items (items inside other items)
- Item type *group*, for page or group within page

[4] XPath is a standard for addressing nodes in an XML structure.

- Item type *label*, for labels, messages, etc.
- *string*, *decimal* and *integer* item types for questions with input field
- *choice* item type for multiple choice questions, with answer options as integers, strings or coding with separate values and labels
- Item attribute *required* – questions can require answers or be optional
- *enableWhen* element – any item can be shown or not based on a set of Boolean expressions

A.1.3 Smart Building Use Case

A.1.3.1 Z-Wave Devices Architecture Design

One of the objectives of the ENACT project has been to create the tools to integrate in the Kubik building new IoT devices complementary to the legacy ones that communicate using building automation protocols. These devices are based on Z-Wave wireless technology.

The following communications scheme has been developed for the Z-Wave infrastructure.

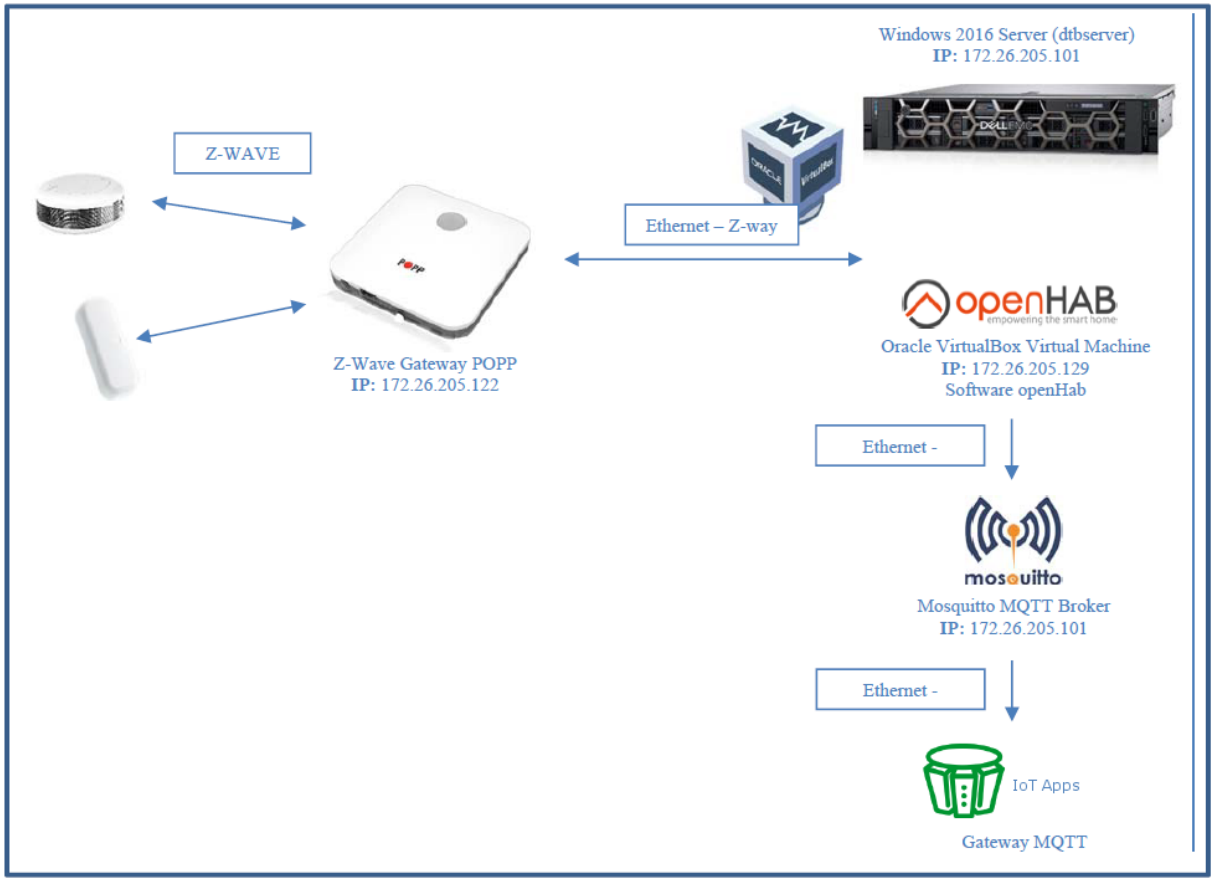


Figure 31: Z-Wave Devices Architecture. Source: TECNALIA.

The POPP HUB Gateway is the communication gateway responsible for establishing the wireless network with Z-Wave wireless sensors and actuators, centralizing communication with all of them.

OpenHAB: It has been necessary to use a software platform to integrate the Z-Wave wireless devices on the PoPP HUB gateway. This automation environment will allow the use of communication standards

for IoT. In our case, we have opted for OpenHAB, an open system independent of the provider, hardware or communication protocol to integrate.

MQTT Broker: The publication of OpenHAB sensor and actuator data to be accessed by the IoT apps is done through an MQTT server, for which we have opted for the installation of the "Mosquitto Broker" software.

The IoT Apps read the MQTT messages for sensor data and write MQTT messages for actuation data through Z-Wave devices.

A.1.3.2 PLC Devices Architecture Design

The Beckhoff PLC communicates directly with the Raspberry Pi SMOOL gateway through a local Ethernet connection.

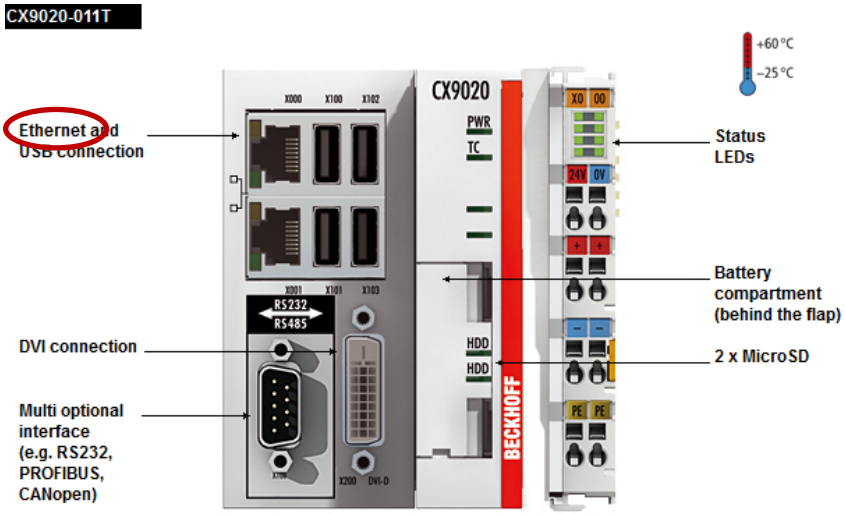


Figure 32: Ethernet Connection of Beckhoff PLC Devices. Source: TECNALIA.

The Automation Device Specification (ADS) describes a device-independent and fieldbus-independent interface governing the type of access to ADS devices such as a Beckhoff PLC. In order to make the ADS communication possible as an ADS client for the Raspberry Pi SMOOL gateway the Java library AdsToJava.dll is used.

In turn, the Beckhoff PLC communicates with wired sensors and actuators through building automation protocols (KNX, DALI, PROFIBUS, etc.) and closed/open contacts in the case of simple relays, e.g. the relays that are used to raise and lower the blinds. The diagram of the electrical circuit between the relay output terminals of the Beckhoff PLC and the motors for raising and lowering blinds is shown in the following figure.

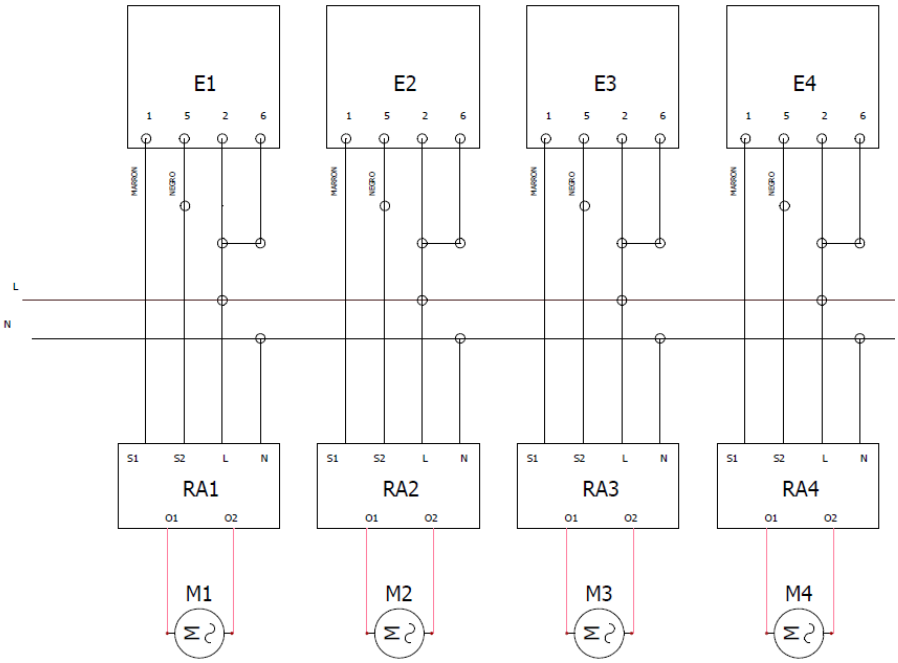


Figure 33: Electrical circuit diagram between PLC device and blind motors. Source: TECNALIA.

A.1.3.3 SMOOL Middleware Design

A.1.3.3.1 Semantic Information Broker (SIB)

SMOOL applications consist of a set of Knowledge Processors (KPs) that exchange data among them. Therefore, the KPs are the most important component of the SMOOL middleware infrastructure and define the end-points of the data in the smart applications. However, to implement the behaviour of the smart space that dispatches the data messages to the subscribers a second component, so called Semantic Information Broker (SIB), is required.

All the interactions between the KPs and the SIB are implemented using the Smart Spaces Access Protocol (SSAP). The SSAP defines the messages involved in each of the services mentioned before, describing the data structure to be included in each of them. The SSAP protocol is XML based, which is very convenient to share ontology-based semantic information as this information is already XML.

As depicted in the following picture, the SSAP module is the one that manages the interactions between the SIB and the KPs. This way, heterogeneous devices can talk the same language (SSAP), improving interoperability among devices from different vendors. The SSAP module connects to the Gateway module to send/receive the messages to/from the communications network. SMOOL is intended to be used with different communication technologies, and the Gateway module is the one that controls whether a message should be transmitted by the TCP/IP stack, Bluetooth, or any other communication technology/network.

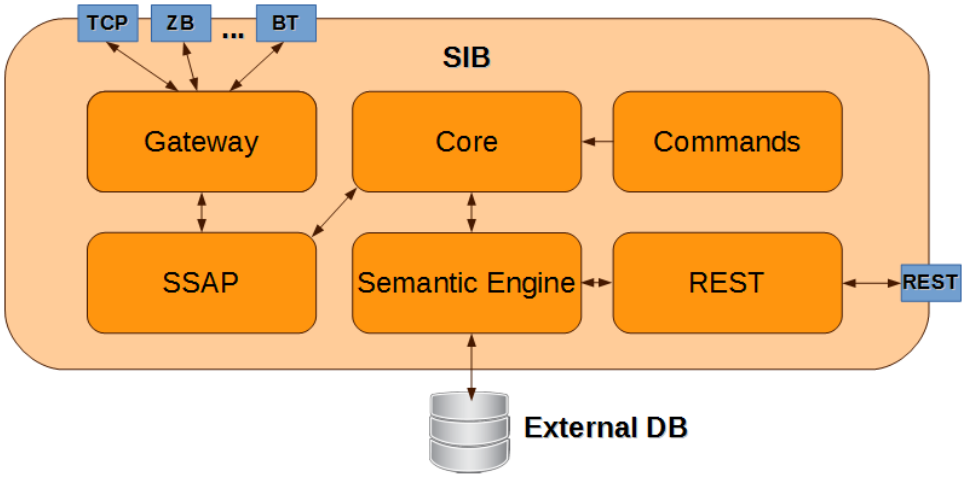


Figure 34: SIB architecture. Source: TECNALIA.

A.1.3.3.2 Knowledge Processor (KP)

Knowledge Processors (KP) are the end points of the smart applications. These components implement the logic of the applications and produce/consume data to fulfil their tasks.

The internal structure of a KP is below. The KPs need to access the smart space services to interact with it by using SSAP messages. However, using SSAP messages directly is not an easy task and it may lead to errors. Thus, the KPs include a model layer, which converts programming objects representing ontology concepts to the required SSAP messages and vice versa.

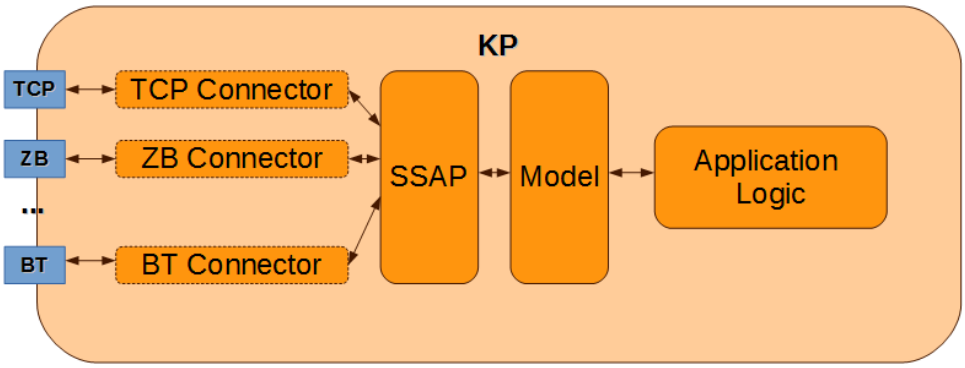


Figure 35: KP architecture. Source: TECNALIA.

Contrarily to the SIB, a KP will typically connect to the smart space using a single connector. Each connector is capable to access the SIB using a concrete gateway kind, for example, a KP should use the Bluetooth connector to access the SIB via the Bluetooth gateway. Therefore, developers must select which gateway module each KP will use to connect to the smart space (e.g. TCP/IP, Bluetooth, Zigbee, etc.).

A.2 SYSTEM DESCRIPTION

This section is divided into hardware and software sections intending to indicate the hardware and software used per partner. The usage of this HW/SW is done in the main document scenarios description.

A.2.1 ITS Use Case

This section is intended to briefly show the current hardware and software developments carried in the ITS Use Case.

A.2.1.1 INDRA

A.2.1.1.1 Software

As it is stated in the previous design section, the main two parts carried by Indra Sistemas SA at this stage of the project are the gateway software development and the Indra Hybrid Platform.

The **Gateway** is served by several subroutines that are summarized as: messaging server, authentication authority, communications system, application modulation, and data integrity check.

- The messaging server tasks is based on a RabbitMQ server which is intended to stablish the queues for the MQTT messages, retransmit it to the required edge entities and transform it to the AMQP protocol in order to enable traffic aggregation procedures communicating the Cloud and the edge.

The messaging mechanisms are implemented based on a publish/subscription method. The topic and payloads used are following a rail ontology specified and shared among the partners that are involved into the system integration.

- The authentication is served as a general service and spread along the entire Cloud and Gateway system. The LDAP authentication system is based on several authentication authorities that provide accesses to the clients in case of the LDAP server and all the authentication authorities validate it.

- The communication systems part enables several software developments are performed in order to deal with wired and wireless systems.

- The application module is oriented to the deployment tool. The applications are located at containers and the software manager is in charge of it. It is implemented into the gateway in order to run several gateway versions and tackle further updates.

- The data train integrity is designed based on CRC protocols with 32 bits. These procedures ensure with a despicable error probability that the data is not changed besides all the encryption mechanisms implemented.

The **Indra Hybrid Platform** software development is intended to enable all the Cloud functionalities required to treat, to store, and to represent the system data. The platform is prepared to enable the connection to every external entity, including the tools. The only requirement is an AMQP connector. All the contacted tools to be implemented at this stage of the project and FIWARE are able to implement an AMQP connector. Therefore, the only requirement is the provision of certificate among system to connect them.

A.2.1.1.2 Hardware

The hardware developments are reduced to the gateway part as the Indra Hybrid Platform is developed at the Cloud and the internal CPD in Indra Sistemas SA.

The hardware part is based on rail certified devices which have the following specifications:

- EN50155 Class TX
- 40°C to +85°C
- 2x serial ports (RS232-RS422-RS485)
- 2x Ethernet 10/100/1000 Mb/s
- M12 X-coded 8-poles Female
- M12 A-coded 4 Pin Male connector

A.2.1.2 EDI

EDI is responsible for implementing the On-board and On-track Wireless sensor network and WSN adapters, the functionality required are reporting of Integrity, Logistics and Maintenance and Inauguration data as well as calculation of train composition integrity.

A.2.1.2.1 Software

As it is stated in the previous design section, the main two parts carried out by EDI at this stage of the project are the On-Board and On-Track WSN gateway software development.

On-Board WSN adapter consists of several subroutines: raw data gathering from on-board sensor network, raw data forwarding to Gateway, integrity data processing and reporting to Gateway.

- On-board Wireless Sensor Network consisting of “things” provides data about train integrity (an accelerometer, a Received Signal Strength Indicator (RSSI) sensor, and a Global Navigation Satellite System (GNSS) receiver) and Radio Frequency Identification (RFID) tag located on the wagon and connected energy harvester data.
- Data forwarding to On-board Gateway is accomplished through Message Queue Telemetry Transport (MQTT) client which is implemented using paho library.
- Integrity data processing uses raw integrity data to calculate train integrity value, which is true or false. Raw integrity data are acquired through MQTT client from Gateway using paho library and calculated data are published to the Gateway in the same way, but in different topic.

On-Track WSN adapter consists of two subroutines: raw data gathering about passing train and raw data forwarding to Gateway.

- On-track Wireless Sensor Network provides raw data about Radio Frequency Identification (RFID) tags of the passing train wagons.
- Data forwarding to On-Track Gateway is accomplished through MQTT client which is implemented using paho library.

A.2.1.2.2 Hardware

The hardware developments are focused on the On-track and On-board Wireless Sensor Network devices called “things”.

On-board “thing“ consist of:

- L152RE Nucleo board
- 868 MHz ZigBee radio
- 3 axis accelerometers

- Global Navigation Satellite System (GNSS) receiver
- UHF RFID EPCglobal Gen 2 reader
- Powered from energy harvesting device

On-track "thing" consists of Arduino Uno 3 board powered from On-track WSN adapter.
On-board and On-track WSN adapters consist of RaspberryPi 3B powered by Power over Ethernet.

A.2.1.3 BOSC

BOSC is responsible for development and implementation of Energy Harvesting (EHS) subsystem for On-board and On-track Wireless sensor network and WSN adapters. EHS developed will conform to requirements set by partners, e.g. EDI, to provide full functionality required for reporting of Integrity, Logistics and Maintenance and Inauguration data. EHS hardware will provide output information according to EHS current battery charging state as input for ADC of the thing and necessary calculations in the on-board thing.
Main energy requirements for EHS are:

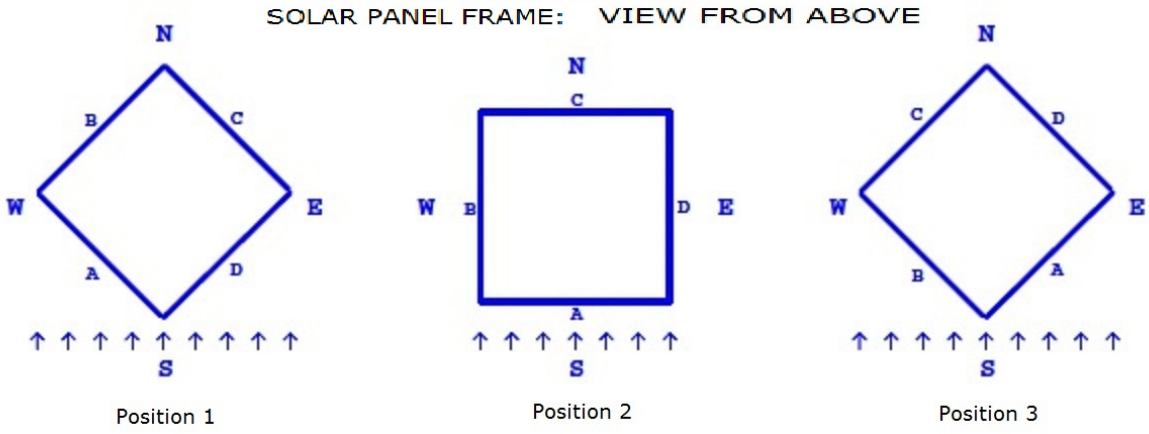
- daily electricity consumption for one node is 0.17Ah => 0.51Ah for 3 nodes;
- each wagon with one EHS module;
- daily energy consumption for the module will be 1.89Wh at average voltage across a battery 3.7V;
- train can move in any direction;
- EHS modules have to withstand outdoor environment;
- EHS module have to deliver all the necessary electrical energy for the WSN nodes all through year.

A.2.1.3.1 Software

Software necessary for EHS monitoring is developed by EDI in close collaboration with BOSC.

A.2.1.3.2 Hardware

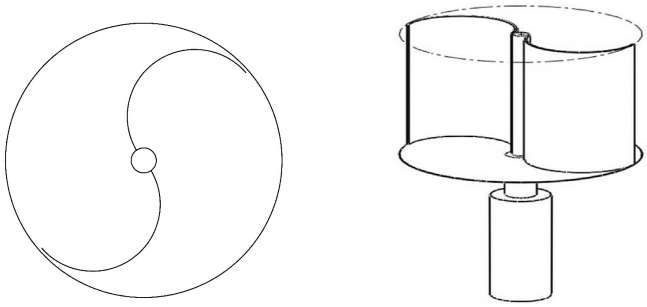
Figure below shows panel layout for designed solar panel frame for different frame positions (azimuth) in the space.



A, B, C, D - panels of the solar panel frame. Used solar panel – mono crystalline, 4.5W installed power, 4 panels for one frame. Installed power - electrical power produced by solar panel at maximum power point at solar irradiance 1000W/m2 and panel temperature +25°C.

As an addition to provide full time Power necessary for mission critical train safety solution like train integrity control system besides to solar module EHS will be paired with *Savonius Wind Turbine* with Electric Generator. Two energy source balancing hardwares will be built in.

Figure: Top and Front View of a Savonius Wind Turbine

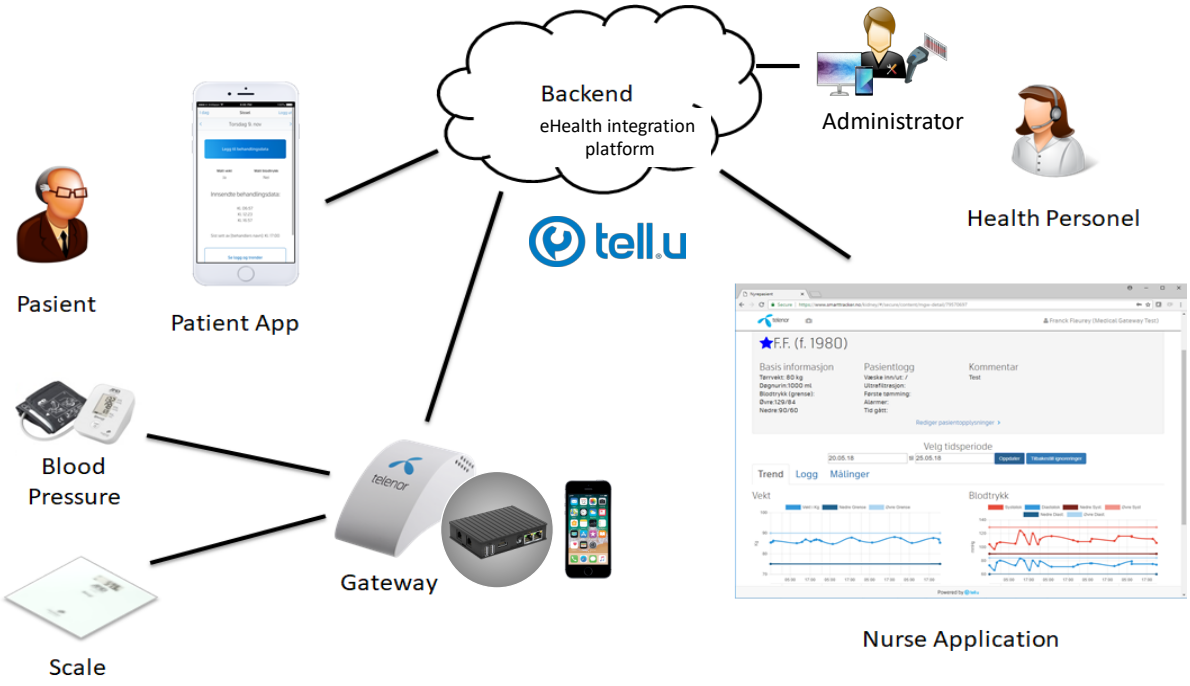


Design of particular Savonius Wind turbine is being developed, designed, and optimized using Matlab/Simulink to get the most appropriate design conforming with particular on track equipment used as ENACT demonstrator facility.
Primary harvester hardware output compatibility

- ADC readable input voltage range 0-1500mV (ADC reference voltage is 1500mV;
- Hybrid harvester output BINDER 09 0412 80 04, 4 pins (socket):
- Power supply for the thing: +5.0V (VCC) stabilized (LTC3130, short circuitry current 450mA);
- ADC first input 0-1500mV (correspond to battery instantaneous load current 0-750mA);
- ADC second input 0-1500mV (correspond to battery voltage 0-15V). In reality battery voltage could fall in range from 6.0V till 9.0V;
- Ground output (GND);
- Connector BINDER 99 0409 00 04.

A.2.2 E-Health Use Case

The overall solution for the remote monitoring of patients is already described in Section 4.2. Its purpose is to improve and facilitate the communication between the patients and the care providers following them.



The main software components of the system are:

- A backend system to manage and store health providers and patient profiles as well as patient medical measurements (e.g., weight, blood pressure, glucose and dialyse measures) and questionnaires.
- A Health Personnel application to create and update patient profiles and view patient medical measurements (in tables and graphs over time).
- A patient mobile application to view their medical data and manually input response to questionnaires and possibly measurements (that are not automatically measured) from medical devices.
- A Personal Health Gateway installed in the patient home to automatically collect measurements taken with Bluetooth connected medical devices.

Hardware Devices
The system includes a library of Bluetooth devices, a PHG and it support both Android and iOS mobile devices.

The Tellu Personal Health Gateway is essentially a software stack that can run on several physical GWs. Currently we apply a standard GW from Compulab that are CE marked and we have made our own prototype GW based on RaspberryPi

Some devices that are currently supported is:

- Axis videocameras (several hundred models)
- Welfare technology sensors and devices that support the Social Care Alarm Internet Protocol (SCAIP), large library of devices
- A set of Bluetooth medical devices including
    - Contour Next One blood glucose meter
    - A&D Medical UC-352BLE scale

- A&D Medical UA-651BLE blood pressure meter
- Oximeter, Nonin, model Nonin 3230 BLE
- Thermometer, A&D UT-201BLE

For the devices there is some particular issues in terms of required physical interventions to operate them and which set some particular challenges in order to have complete DevOps operation of the system. This includes:

- Pair device: pair a supported Bluetooth device with the phone/tablet, so that it can be used to make measurements. The app scans for Bluetooth devices. If a supported, unpaired device is detected, we try to connect to it. Pairing is handled by the underlying system, with acknowledgement from the user needed. Our app detects when the status of the device changes to *paired*. A status screen is shown during this process, keeping the user informed.
- Make measurement: The user first selects the type of measurement to make. The app scans for supported devices with the relevant capabilities. We then show a status screen similar to that used for pairing, keeping the user informed without getting too technical. Once we detect an appropriate device in the scan, we go through a number of states and Bluetooth transactions: connecting, discovering the full set of services supported, for some devices writing the current date/time to the device is necessary, and then requesting the data we are interested in. Once we have a measurement, it is shown in a result page. We now also support entering values manually for the selected type of measurement. This is useful to support devices without Bluetooth connectivity, and as a backup if the Bluetooth protocol fails. Finally, the user can submit or discard the measurement.

Behind the user interface, the Bluetooth interaction is managed by a component we call BLEManager.

A.2.3 Smart Building Use Case

A.2.3.1 Z-Wave Communication Description

The Z-Wave is a wireless communications protocol that has been standardized by the Z-Wave Alliance, an international consortium of manufacturers that oversees interoperability between Z-Wave products and enabled devices. The Z-Wave wireless protocol operates at 868.42 MHz in Europe. This band avoids interference with Wi-Fi and other protocols that operate on the crowded 2.4 GHz frequency. Z-Wave is also designed to provide reliable, low-latency transmission which enables real-time operation.

The Z-Wave protocol is optimized for battery-powered devices, and most of the time the device remains in a power saving mode to consume less energy, waking up only to transmit or receive data. Only EnOcean protocol uses less energy than Z-Wave, but the unbeatable Z-Wave interoperability and its numerous benefits makes Z-Wave the most widely used protocol in home automation with the largest number of compatible devices in the market.

Figure 36: POPP HUB Gateway for communication with Z-Wave devices. Source: TECNALIA.

A.2.3.2 PLC Communication Description

The communication between the Beckhoff PLC and the wired sensors and actuators is carried out using building automation protocols. In this section, the communication protocols for controlling lighting, fan coils and blinds are described.

The lighting is controlled using DALI (Digital Addressable Lighting Interface) communication protocol. DALI is a manufacturer-independent protocol defined in the IEC 62386 standard that is commonly used to communicate between digitally controllable lighting systems, such as electronic ballasts, transformers and power dimmers. The DALI protocol is used in building automation to control individual lights and lighting groups.

The fan coil units are ultimately controlled by 0-10V analog control protocol where zero voltage is considered off and ten control voltage is considered full on. The 0 to 10V control is intended to be linear. The output of a receiver should be half when it receives 5V control voltage. Fan coil controllers translate analog control protocols to TwinCAT Modbus automation protocols. In the Kubik building, the fan coils units are integrated into a SCADA (Supervisory Control and Data Acquisition) software that facilitates fan coil control through the setting of start / stop commands and temperature setpoints.

The blinds are raised and lowered by using directly controlled relays that close/open the mains current of each of the motor blinds through volt free contacts. The up/down direction of the motors is changed by reversing the voltage and limit switches indicates when the blind is fully raised or lowered.

A.2.3.3 SMOOL Middleware Description

SMOOL is an IoT platform, implemented under the publish/subscribe paradigm, that aims at tackling the latter issues through the definition of (1) an open API and middleware services based on existing standards that provides a communication back-bone for smart applications, (2) a common and extensible data model for smart spaces that enables interoperability among vendors at application (semantic) level and (3) a set of design and development support tools that drastically reduces the development time of smart value added applications.

The whole SMOOL solution takes into account legacy devices, implementing gateways for existing smart components and developing guidelines for other vendors to develop their own. This way the SMOOL middleware can be split into two main components: the SMOOL Infrastructure, which enables the technical interoperability of the devices at runtime; and the SMOOL Superstructure, which includes all the design and development tools associated with the Infrastructure, providing application (semantic) level interoperability.

SMOOL addresses the economic sustainability challenge of mass- markets with an open business model approach, where cost-to- return becomes part of the solution. The SMOOL middleware builds upon the Smart-M3 information sharing platform, result of the SOFIA (EU funding) research project. SMOOL infrastructure reuses the Smart Space Access Protocol (SSAP) developed in SOFIA; however, SMOOL extends the capabilities of the application development tools and greatly simplifies the API for new developers.

SMOOL open source code is currently hosted in Bitbucket (https://bitbucket.org/jasonjxm/smool/wiki/Home) and the project is actively maintained by TECNALIA.

A.2.3.3.1 The publish-subscribe model

The SMOOL infrastructure is a data distribution middleware based on the publish/subscribe paradigm. Compared to the classic client/server applications, which are typically used for applications where interactions are established on a one-to-one basis, publish/subscribe architectures perform better when messages are to be distributed to multiple clients. In other words, in publish/subscribe applications data is more important than the interactions themselves.

This is often the case of smart applications and pervasive computing systems. For example, when a new device enters a smart space subscribing to the data that is relevant for the device is much simpler than locating all the services provided by all the server devices in the area and selecting the ones that are useful for it. Moreover, the latter approach might lead to compatibility issues (e.g. similar services might define different APIs).

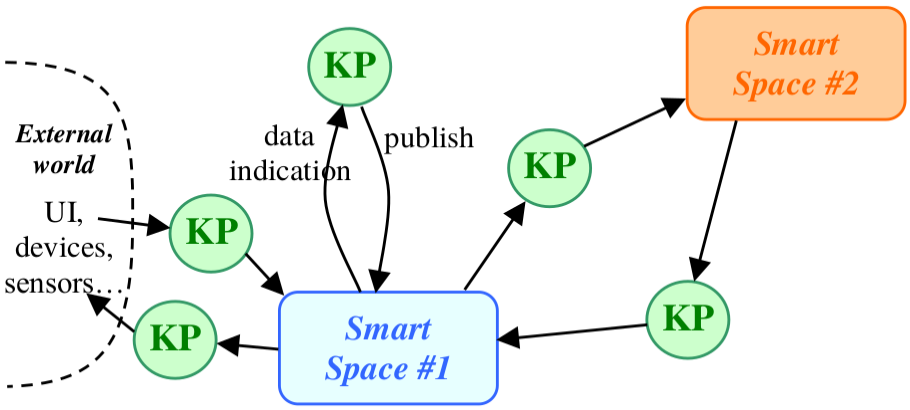


Figure 37: SMOOL infrastructure schema. Source: TECNALIA.

The figure above describes the SMOOL publish/subscribe schema, in which the clients (so called Knowledge Processors or KPs) provide data to the smart space, often coming from sensor measures taken from the real world or decisions made by smart application users through smart devices, and consume this data to create added value data, interact with the real-world or even to share it across several smart spaces.

A.2.3.3.2 Semantic oriented

SMOOL introduces a semantic model that is used to mark each of the message types shared in a smart space. The semantic model is basically an ontology which defines the different message kinds, establishes semantic relations between the different message types and ensures application level compatibility. In addition, due to the extension characteristics of the ontologies, the semantic engine in SMOOL ensures backward compatibility of the applications even when new message types are added to the ontology.

Ontologies in SMOOL are used in two ways. On the one hand, the ontology is used to allow abstract subscriptions to data of many different types. For example, a KP may subscribe to the concept Sensor and will receive notifications whenever a message of the Sensor type or any subtype is published (e.g. TemperatureSensor). On the other hand, the SMOOL ontology is used by the design and development tools to automatically generate the data model code required by the KPs to produce/consume data of each message type.

A.3 INSTALLATION AND SUPPORT

This section is used to explain the installation necessary for the systems described in the deliverable to be operative.

A.3.1 ITS Use Case

This section is intended to show the installation requirements and procedures used in the installation based on the Test Plan that are intended to be applied for the project final demonstration. The installation can be divided at On Board and On Track sections:

- The **On Board** section requires the nodes installation at each wagon and loco. Moreover, the gateway is installed at the loco in order to have the best possible coverage position with the WSN Coordinator and the communication infrastructure. The gateway is connected to the Indra Hybrid Platform throw a 3GPP communication link. Therefore, the coverage this type of infrastructure must be deployed or an alternative owner V2X communication infrastructure. The scheme is shown in the Figure 38:

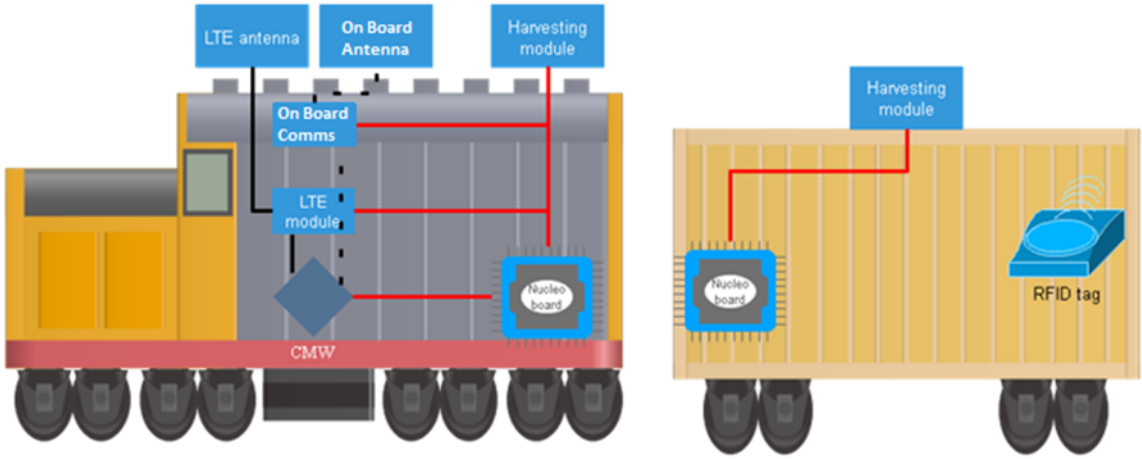


Figure 38: ITS On Board Installation. Source: INDRA.

The WSN power supply is independent. The WSN is power supplied by an energy harvesting system installed On Board. However, the gateway is enabled with its own not harvester power supply system.

- The **On Track** section requires a similar infrastructure to track the train using RFID tags instead of the On Board Train Integrity and Train Inauguration nodes. To transmit this information, it is required On Track gateways to report this information using the defined rail ontology to the Cloud system throw the previous mentioned 3GPP communications.

The **Cloud part** is considered as a distributed and independent system. Therefore, no further needs are considered for this part.

It must be highlighted that the gateways are certified to be installed On Board and On Track following the European rail certifications.

A.3.2 E-Health Use Case

The eHealth use case is provided as a SaaS, the overall architecture of the service is as shown in the figure below (as previously presented)

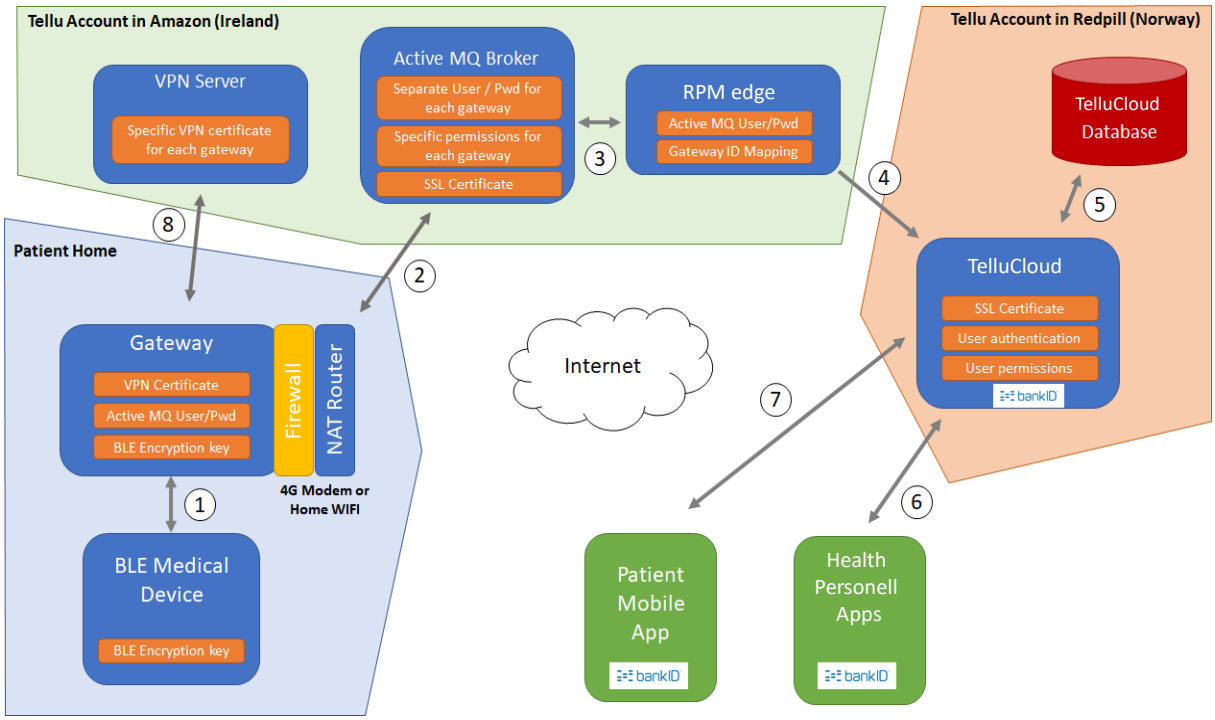


Figure 39 Overall architecture of the Remote Patient Monitoring Use Case. Source: TellU.

The applications are provided as web applications and as apps to be downloaded from App store or Google Play. Access need to be granted before to access the system. For the PHG and devices these need to be registered and authorised in the system and Bluetooth devices need to be paired with the PHG to apply them and provide measurements to the system. The code of the system resides in GitHub in a private repository.

A.3.3 Smart Building Use Case

A.3.3.1 Z-Wave Devices Installation

The POPP HUB portal is the communication portal of the Z-Wave wireless network. The administration panel of the PoPP HUB gateway can be accessed through the URL of the web page: http://<POPP_HUB_IP_address>:8083/smarthome

Once the username and password have been entered, the “Devices" menu can be accessed from the list located in the upper right corner of the screen and the "Add new" option can be selected. At this time, POPP HUB performs a wireless scan in search of the new item that should appear selected to automatically link to the gateway.

Figure 40: POPP HUB administration panel to link new Z-Wave devices. Source: TECNALIA.

In another view, it is possible to see the values published by each sensor in the POPP HUB gateway. The update of the sensor data depends on each device and the type of variables it publishes. For example, the temperature information is updated on each variation of 1ºC, while the digital variables are sent by change.

Figure 41: Values published by each sensor using the POPP HUB gateway. Source: TECNALIA.

The openHAB software platform is used to access the data of Z-Wave wireless devices by the IoT Apps by means of MQTT messages. The administration panel of a running openHAB instance is accessed from the URL: http://<openHAB_IP_address>:8080/paperui/index.html

Figure 42: OpenHAB software administration panel. Source: TECNALIA.

In this environment, the new device must be registered in openHAB. In the "Inbox" menu, the new devices that can be added in the openHAB software from the POPP gateway are displayed. Once added, it will appear in "Settings - Things" and, in turn, the items that the device may have will be added to "Settings - Items". In the "Control" menu, it can be seen the data collected in openHAB by the different sensors.

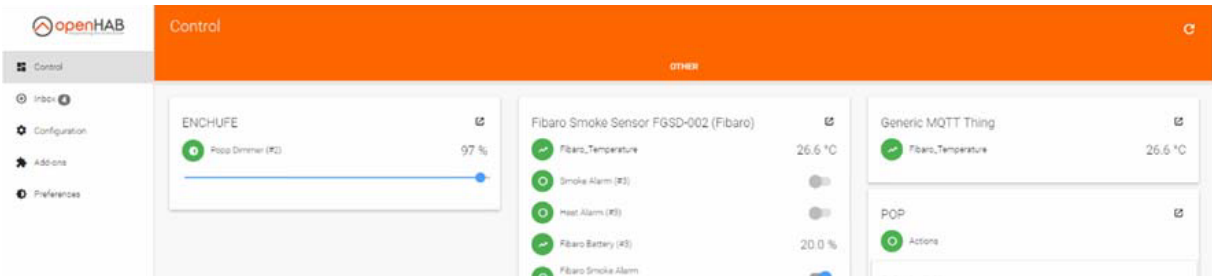


Figure 43: Control menu of the OpenHAB software. Source: TECNALIA.

The publication of openHAB data is done through an MQTT server, specifically the "Mosquitto Broker" software has been installed. Sensors publish sensor data on the Mosquitto server and actuators consume actuation orders from the Mosquitto server using MQTT messages.

A.3.3.2 PLC Devices Installation

The communication with the Beckhoff PLC, which in turn communicates with wired sensors and actuators through building automation protocols, is done using the Automation Device Specification (ADS) protocol. The ADS communication is implemented on the Raspberry Pi SMOOL gateway using the Java library AdsToJava.dll. The sensor and actuation data are stored on the PLC variables or registers. Those data variables are read and written from using the AdsToJava.dll library.

The files "TcAdsSOAP.java" and "Base64.java" of the library must be added to the IoT application and compiled. The following object provide access to the ADS WebService where the IP address is the URL of the Beckhoff PLC:

```
TcAdsSOAP tcSoap = new TcAdsSOAP("http://<IP>/TcAdsWebService/TcAdsWebService.dll")
```

Reading PLC-variables that contain sensor data:
To read PLC-variables use the following method of the TcAdsSoap-object from the AdsToJava.dll:

```
DataInputStream SOAPCall(String netId, int port, long group, long offset, int cblen)
```

- `String netId`: String indicating the AdsAmsNetId of the PLC. The AdsAmsNetId is an extension of the TCP/IP address and identifies a Beckhoff PLC in a network.
- `int port`: Port number of the Runtime-System.
- `long indexGroup`: Index Group of the variables to read.
- `long indexOffset`: First byte to read.
- `int cblen`: Number of bytes to read.
- `DataInputStream`: Return value with the read data, throws exception on failure.

There are 3 methods that read integer, boolean and string variables and convert them to their corresponding type:

```
short ReadInt(String netId, int port, long group, long offset)
boolean ReadBool(String netId, int port, long group, long offset)
String ReadString(String netId, int port, long group, long offset, int cblen)
```

Example of reading a boolean variable:

```
boolean BoolVariable;
TcAdsSOAP tcSoap = new
          TcAdsSOAP("http://172.26.205.163/TcAdsWebService/TcAdsWebService.dll");
BoolVariable = tcSoap.ReadBool("172.26.205.163.1.1", 801, 16416, 0);
```

Writing PLC-variables that trigger an actuation order:
To write PLC-variables use the following method of the TcAdsSoap-object from the AdsToJava.dll:

```
boolean SOAPWrite(String netId, int port, long group, long offset, byte[] data)
```

- `String netId`: String indicating the AdsAmsNetId of the PLC.
- `int port`: Port-number of the Runtime-System.

- `long indexGroup`: IndexGroup of the variable.
- `long indexOffset`: First byte to write to.
- `byte[] pData`: Byte-array containing the data to write.
- `boolean`: return value "true" if successful, "false" if failure.

There are 3 methods that write integer, boolean and string variables:

```
boolean WriteBool(String netId, int port, long group, long offset, boolean data)
boolean WriteInt(String netId, int port, long group, long offset, int data)
boolean WriteString(String netId, int port, long group, long offset, String data)
```

Example of writing a boolean variable:

```
boolean BoolVariable = true;
TcAdsSOAP tcSoap = new
          TcAdsSOAP("http://172.26.205.163/TcAdsWebService/TcAdsWebService.dll");
tcSoap.WriteBool("172.26.205.163.1.1", 801, 16416, BoolVariable);
```

A.3.3.3 SMOOL Middleware Installation

SMOOL clients for ENACT:
Java examples for publishing data and subscribing to events for retrieving data. The data is created in the KUBIK smart building (temperature, humidity, etc...).
See SMOOL documentation and videos on SMOOL source code repository.
The current repo contains 3 JAVA projects:

- Producer: publish data (temp, CO2, smoke...)
- Consumer: subscribe to data and process when arriving
- Security: control clients sending data and actions

Usage (for 90% of users):

*Installation: C*heck or install JAVA 8 or upper version.
No more install or build steps are required (the project contains the .class and libs). Clone the GIT repo and ready!

Running: Start the Java app either by using the startup.sh or by executing the command java -cp bin:lib/* ENACTConsumer.logic.ConsumerMain.
A Producer is already sending values to smool.tecnalia.com server so you should retrieve messages. If using another server, start the Producer project.

Usage (for the rest 10% of users):
These users should add their own ENACT logic to the Java apps. The logic can be implemented in separate projects (and use the SMOOL clients as proxies) or inserted into the example SMOOL clients.

*Installation: C*heck or install JAVA 8 or upper version.
Clone the repo.
Use your favourite JAVA IDE (Intellij, Eclipse, Netbeans, etc...).
OPTIONAL, if using Eclipse, you may want to install the SMOOL updatesite (See SMOOL documentation above) so you can create your custom SMOOL clients by using the wizards. Note: instead of downloading official updatesite, download current alpha release for Enact here, ask for adding view permissions.

*Running: A*dd the ENACT code at the /logic folder. You could use a symbolic link to the real folder, so the ENACT-only code could be tracked by using your own, private project (see next section).
run the app from the IDE, the default server to be connected (tcp: smool.tecnalia.com) is already running.

OPTIONAL, instead of that public server, you can use your own SMOOL server (so other clients will not receive 'test' messages). See SMOOL documentation for starting private servers. Note: instead of downloading official server, download current alpha release for Enact here, ask for adding view permissions.

Updating the code:
IMPORTANT: keep track/copy of your /logic folder (by using Git, Mercurial, Subversion...or backup regularly that folder). The folder contains YOUR code, while the rest of folders are only for SMOOL data-model and connectors, and they can be overridden safely. Lots of options are valid, but symbolic link of the folder to another external folder is the safest and simplest one.

Updating the SMOOL client:

Either: 'track' /logic folder. Pull repo (git pull ...). Replace /logic folder by symbolic link.

Or if you installed the SMOOL updatesite: 'track' /logic folder. Create a new customized SMOOL client from Wizards. Replace /logic folder.

THING-ML Integration:

On projects where some of the autogenerated code is performed by using ThingML files: The current projects contain the generated code as a Maven project. Download the project and on Eclipse select File/import/maven project/select pom. Remember to use the /logic folder in the thingml-gen folder.

For projects generated from scratch: First install SMOOL and ThingML on Eclipse. Then create a new KP and finally execute ThingML on file smool.thingml. See instructions on smool-thingml integration. (Note: current SMOOL version for Enact project is at https://bitbucket.org/ubuntuubuntu/smool_alpha/downloads)

Videos:
https://youtu.be/u-2u2vwxFvY
https://youtu.be/mfT_AwfkXNc

A.3.3.4 SMOOL Middleware Support

Specialized design and development tools contribute to reducing the time-to-market of the applications, augmenting the quality of the final software and reducing the number of bugs. This is especially true when a new technology is involved. To reduce the learning curve of the SMOOL technology, the framework includes a set of helper tools that ease the adoption of the middleware concepts and provide support for the implementation and testing of smart applications. All the tools have been implemented as Eclipse plugins, fully integrated with the IDE, which makes them more usable for developers.

Currently, the SMOOL toolkit includes two main tools:

- KP generation wizard
- SIB control and monitoring user interface

A.3.3.4.1 5 KP Generation Wizard

The KP generation wizard guides the smart application developer during the configuration of the KPI for a particular smart device. As result, the wizard will generate a project inside Eclipse including all the KPI libraries and the top level SMOOL API.

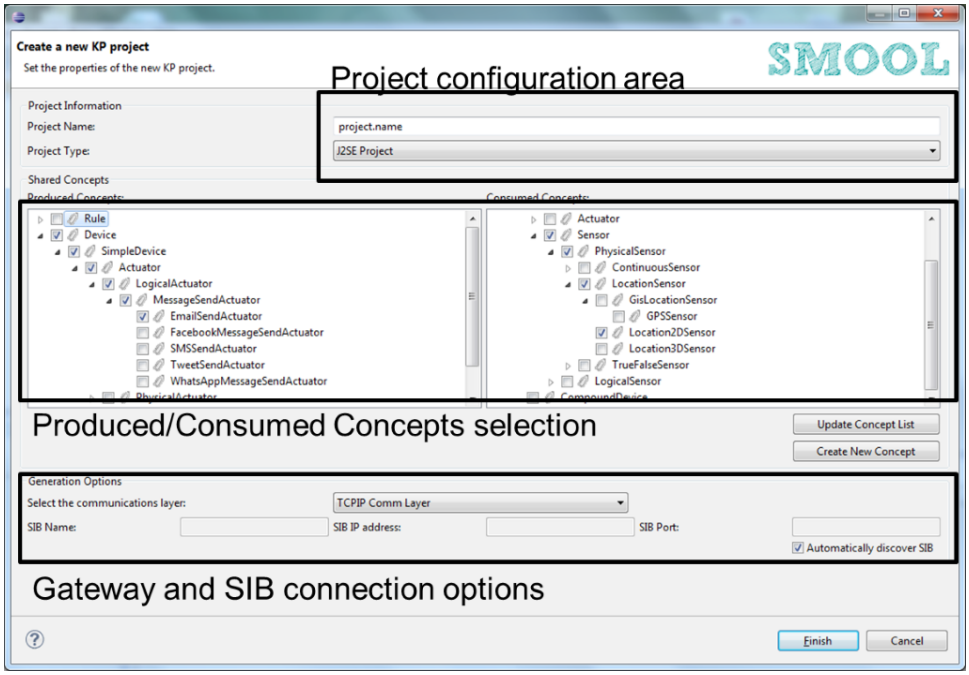


Figure 44: Creating a KP by using the SMOOL wizard. Source: TECNALIA.

A.3.3.4.2 SIB Control and Monitoring UI

The SMOOL toolkit also contains a specific user interface (by activating the SMOOL Perspective in Eclipse) to create smart spaces and control the messages, subscriptions, etc. generated by the smart applications connected to it. The goal is to provide a means to debug the smart applications in a real smart space context.

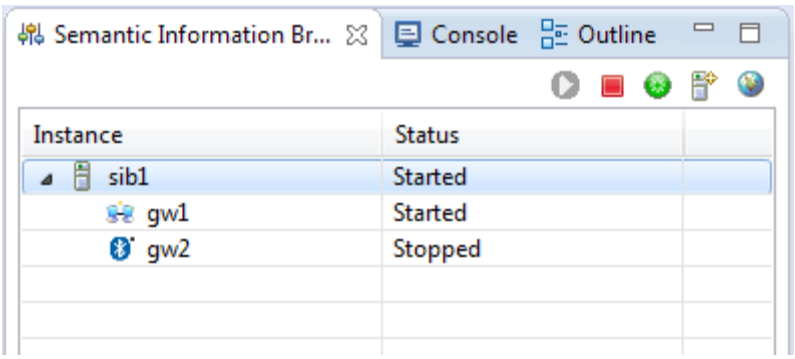


Figure 45: SIB management panel. Source: TECNALIA.

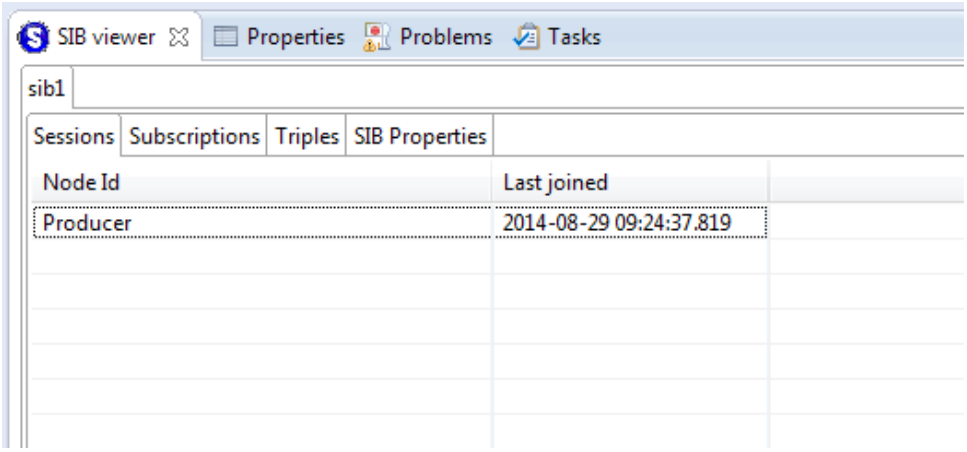


Figure 46: SIB monitoring panel. Source: TECNALIA.