Interreg Sudoe
EUROPEAN UNION
TR@NSnet
European Regional Development Fund

Report on THE replication of IoT demonstrators

Partner Responsible
UT3 - Université Toulouse III - Paul Sabatier

Authors:
Guillermo del Campo (UPM)
Rahim Kacimi (UT3)
Malik Iraïn (UT3)
Yacine Ghamri Doudane (ULR)
Kacem Boussekar (ULR)

Project funded by the Interreg Sudoe Programme through the European Regional Development Fund (ERDF)

UNIVERSITÉ TOULOUSE III PAUL SABATIER
POLITÉCNICA INTERNATIONAL CAMPUS OF EXCELLENCE
Ciências ULisboa
cise Electromechatronic Systems Research Centre
UNIVERSIDADE BEIRA INTERIOR
La Rochelle Université
Funseam Fundación para la Sostenibilidad Energética y Ambiental
circe RESEARCH CENTRE FOR ENERGY RESOURCES AND CONSUMPTION
CTA

E1.2.1 – Report on the replication of IoT demonstrators

Activity Description

Objective

The objective of this activity is to replicate the achievements of each university as demonstrators in the other universities. Indeed, there are two options. When it is a hardware platform, this consists of deploying a demonstrator with the nodes and the equipment developed by the university. Whereas for software services and deployment experiences, in this case the interested universities will benefit from the experience feedback of the one who has already developed and deployed them.

The following achievements were proposed:

- By UT3:
  - Noise/Ambient sensor neOSensor (low-cost, Wi-Fi enabled, multi-sensors).
  - A private LoRaWAN network deployment (federated LoRaWAN infrastructure).
  - MQTT Enabled Data Collection for IoT infrastructures (containers deployment).
- By UPM:
  - A private 6loWPAN network deployment (BatLink): 6loWPAN based network coordinator in charge of IPv6-IPv4 tunnelling and network management (routing, neighbours, retransmissions).
  - Energy Metering device (BatMeter): 6loWPAN based power meter (voltage, current, real/apparent power and energy) for electric panel board able to measure up to 18 single electric lines (6 lines per phase).
  - Ambient Sensing (BatSense): 6loWPAN based ambient sensor able to simultaneously monitor temperature, relative humidity, luminosity and presence.
- By ULR
  - Data Management and Analysis Service
  - Wi-Fi enabled metering network.

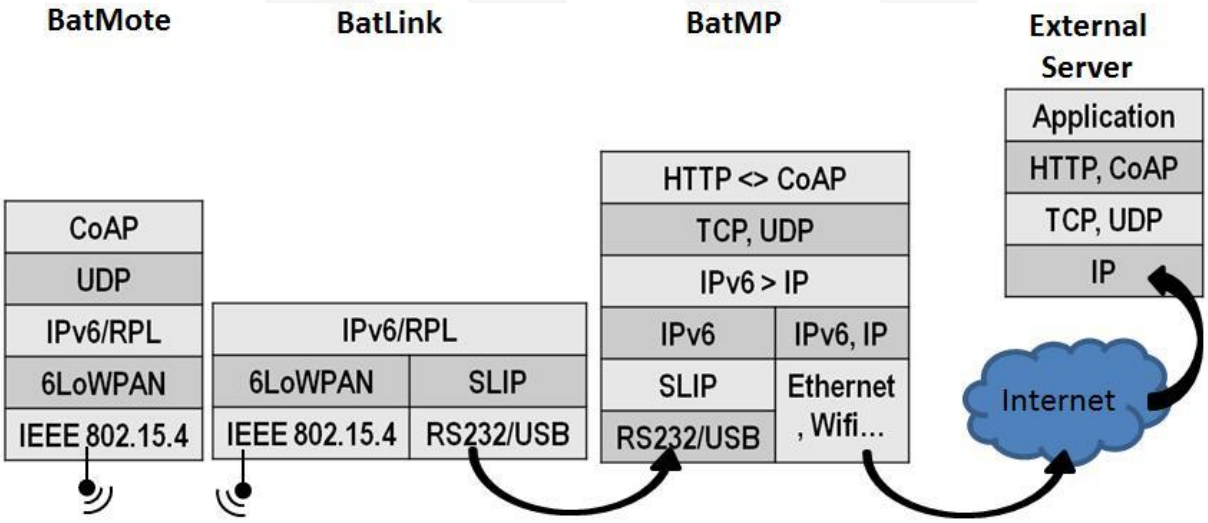
Description of the IoT Demonstrators

BatNet from UPM

BatNet is a wireless devices network that uses IPv6 as the main communication protocol. This allows communication between the devices, creating a mesh network with one way out to the Internet through a concentrator device.
Designed and developed from 2012 in CEDINT-UPM, BatNet family of IoT devices includes three and one phase smart meters, environmental multi sensor, outdoor and indoor lighting controller, smart plug or machine status register.

Communications

A specific protocol model has been chosen in order to avoid interoperability, cost and consumption problems and to make possible further cooperation:



BatNet protocols stack

- IEEE 802.15.4: This standard specifies both physical and Medium Access Control for wireless low data rates personal area networks. It has been chosen the 2,4GHz band due to its many channels and international use.
- 6LoWPAN: The working task group 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) from IETF describes how to adapt IPv6 packets to IEEE802.15.4 based networks. Among others, it adapts the packet size and resolves addresses.
- IPv6/RPL: It is clear that IPv6 is to replace IPv4 soon, because of the lack of IPv4 addresses claim for the IPv6 128-bits ones. RPL (IPv6 Routing Protocol for low

power and lossy networks), defined by the ROLL group from IETF, optimizes the traffic and minimizes the routing states.
- UDP: UDP (User Datagram Protocol) matches the needs of these kinds of networks due to its characteristics (simple, stateless and small headers).
- Communication interface CoAP: CoAP (Constrained Application Protocol), is an adaptation of the application level protocol HTTP for low resources devices and networks. Thanks to this, the RESTful paradigm can be used within this kind of network.

There are two kinds of nodes within the network:

- Standard node: In addition to being part of the network, these nodes manage the different control and monitoring systems for: electric variables (BatMeter, BatPlug, BatSwitch), environmental (BatSense) and for illumination (BatStreetLighting, BatDimmer, BatAmbientLight, BatLEDs).
- Border Router node: It takes charge of connecting BatNet with any other IPv6 network, allowing connections with both devices and external clients. It is meant to be a low process capability and consumption device.

At the same time, the standard nodes may be classified in two types, depending on their behaviour within the network:

- Repeater node: These are network nodes but also work as repeaters for the other nodes, allowing the expansion of the network area as a mesh.
- Leaf node: These uniquely act as network nodes.

The access to network devices (either for collecting data from sensors, or for sending orders to the actuators), may be done through two different ways:

- BatLink: It is a micro-computer (BeagleBone) with a Border Router assembled. This allows tunnelling all the IPv6 traffic through an IPv4 tunnel (if necessary) to ease the access to the BatNet network remotely through the internet and execute directly inside the BatLink monitoring applications, where a large amount of data can be stored.
- BatMP: It is a software platform based on Java which enables the cooperation of users and applications with BatNet devices through a REST interface. It allows total management of BatNet and also the development of new applications and services.

Operating System

BatNet devices run Contiki, an open operating system developed by the SICS (Swedish Institute of Computer Science) and designed especially for devices with constrained

resources (limited memory and low process capability). Among other functionalities, Contiki provides an implementation of TCP/IP stack and an adaptation of the IEEE802.15.4 with IPv6/RPL.

Devices may be programmed either physically (ISP or serial interface) or remotely using OTAP (Over the Air Programming).

BatMeter

This device needs to be installed in the electrical panel and is able to measure the consumption of up to 6 electrical lines simultaneously. Consumption is calculated by measuring the voltage and the current of every line in real time. Using this, it can calculate real power, apparent power and power factor.

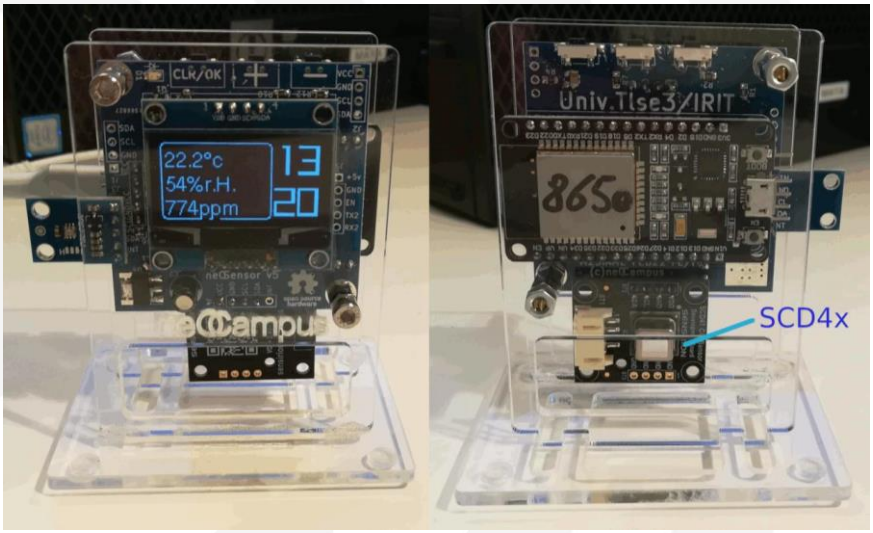
BatMeter device (PCB board and DIN encapsulation)

BatSense

It is a multi-sensor device able to monitor temperature, humidity, luminosity, presence and noise indoors. It may be powered by both power supply and batteries, enabling programming of the measuring interval. The battery life may last much longer thanks to the low power and the standby modes.

BatSense device including temperature, humidity, luminosity and presence sensors

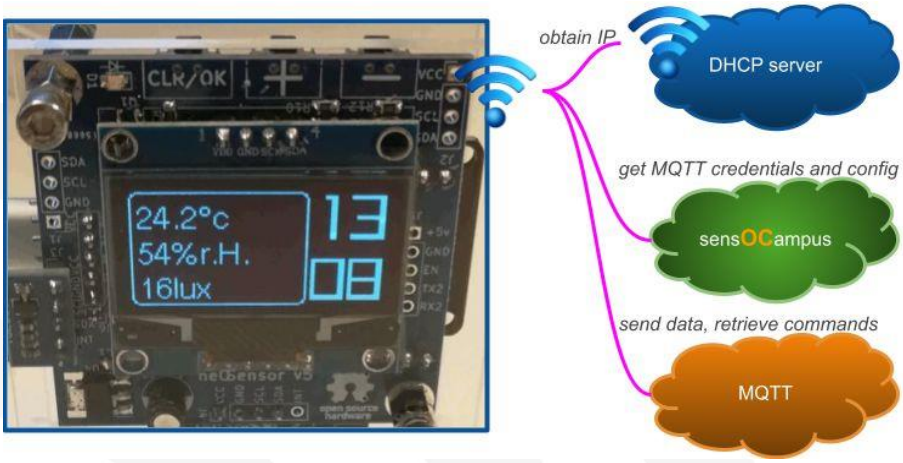
neOSensors from UT3



A neOSensor with a Sensirion SCD4x temperature, humidity and CO2 sensor

In 2015, the need to monitor ambient noise in the main library of Toulouse 3 campus, in a simple and cheap way, arose. That was the first version of a neOSensor. Quickly after, it has been asked to expand neOSensors capabilities: temperature, humidity and luminosity digital sensors were added.

Nowadays, those cheap sensors spread across and above UT3 campus while exhibiting support for a broader range of sensors. Support is now provided for air quality sensors (COV, PM and CO2), especially the latest Sensirion SCD4x CO2 whose measurement is based on ultrasonic waves and PIR sensors (for motion/presence detection).



The different steps to establish a connexion from a neOSensor

Several versions exist, however they all operate more or less in the same way. They connect to a Wi-Fi access point, a dynamic IP is obtained on boot time from a DHCP server. Then a configuration and credentials are retrieved through the sensOCampus API. Finally, data is transmitted to the configured MQTT broker.

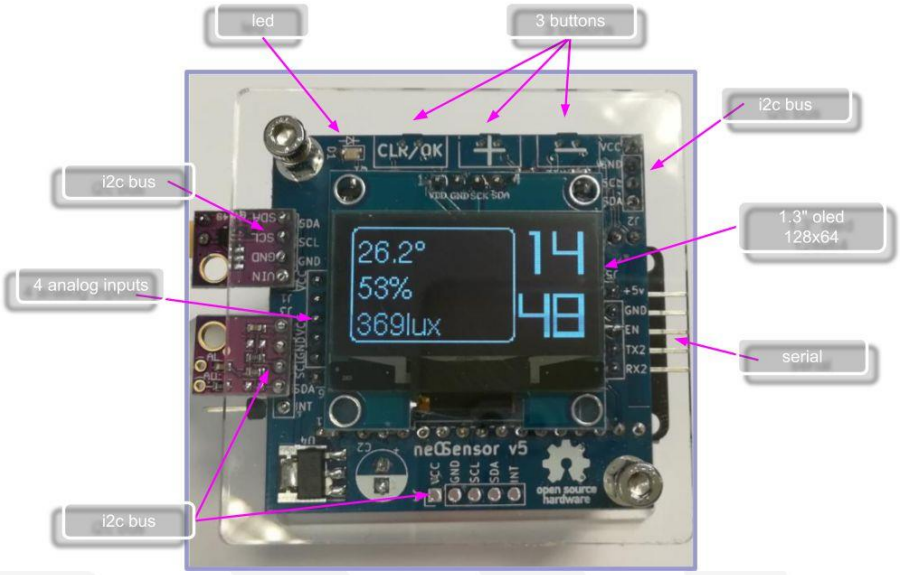
Each data of a sensor is acquired multiple times until it becomes stable according to a per class specific setup. When a new data has been marked as stable, a check is done to assess if it differs enough from the last one that has been sent to the MQTT broker:

data(new) - data(old) > x% ⇒ send

After having sent a new value, the sensor sleeps for a per class specified cooldown value (60s most of the time). Anyway, whatever happens, (e.g. luminosity sensors in our datacenter are most of the time returning a 0 value!) data is sent every 30mn at least.

neOSensors features the following interfaces:

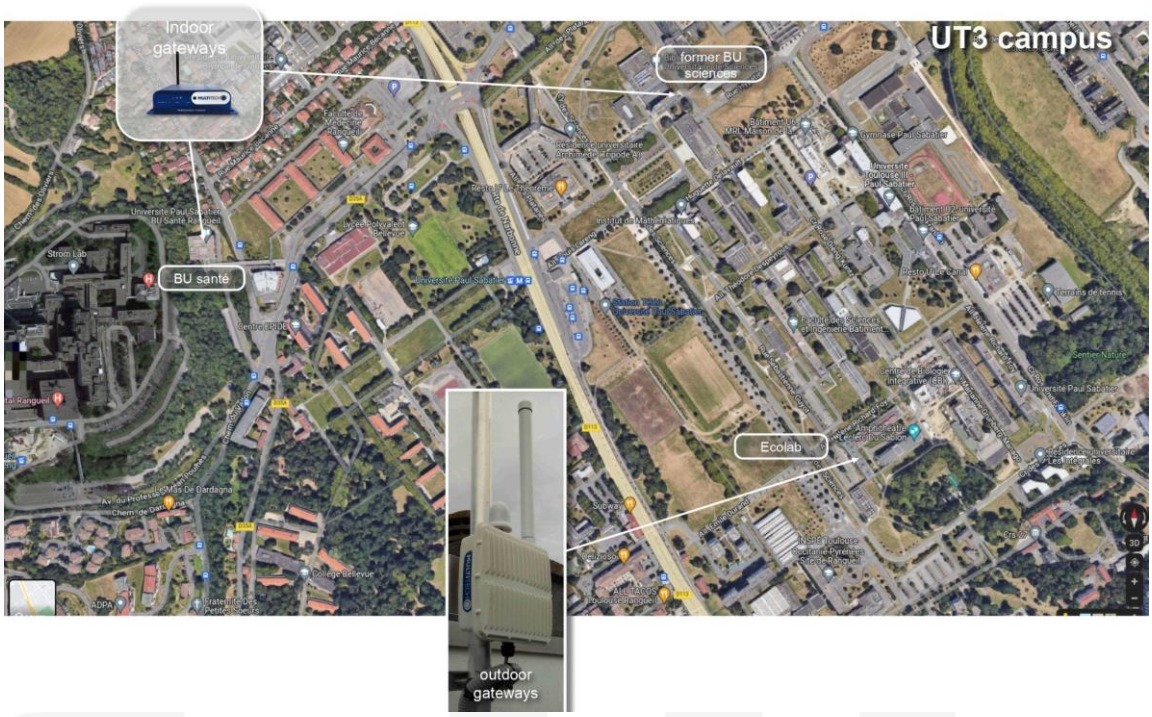
- One I2C bus to connect all i2c sensors together
- One serial link featuring an EN (i.e enable) command
- 3 digital inputs (buttons) on top
- One led on top
- one digital input for a PIR sensor
- 4 analog inputs



Input/output lines of a neOSensor

LoRaWAN Infrastructure of UT3

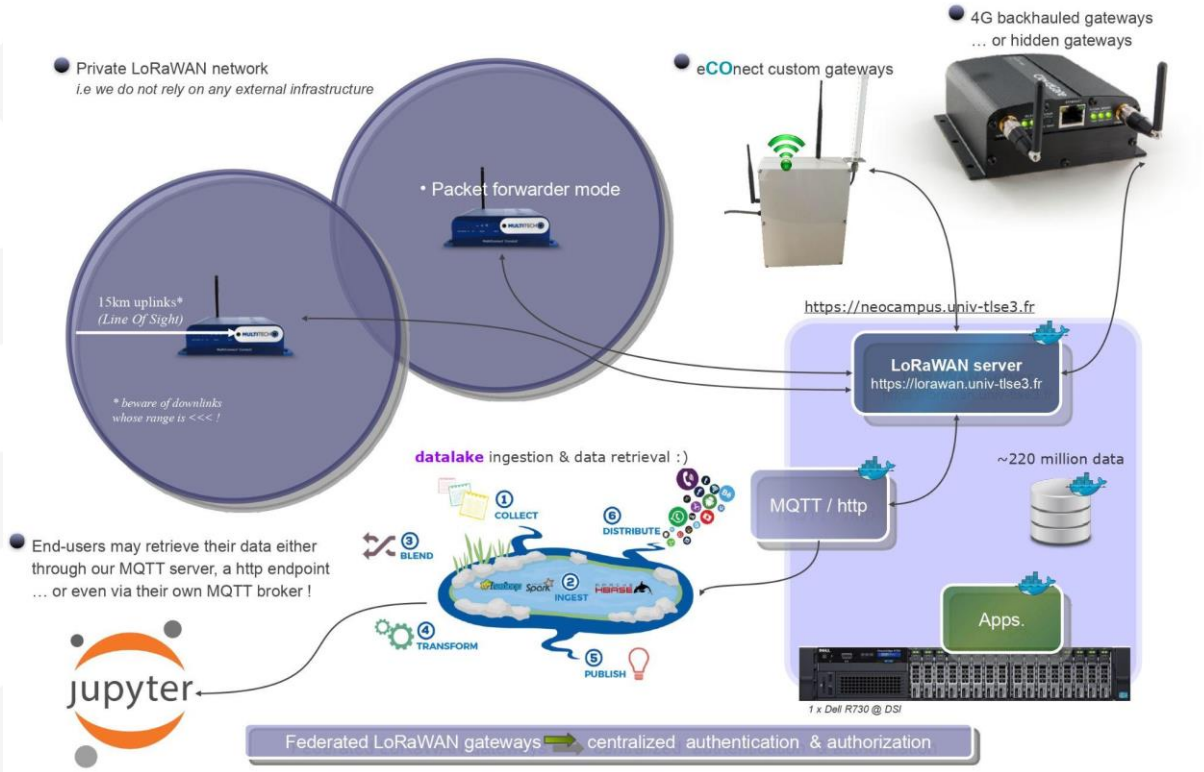
In the context of neOCampus, UT3 started to deploy its own private federated LoRaWAN network.



LoRaWAN gateways located on UT3 campus

The coverage is mainly provided by three different gateways, each configured as "Packet Forwarder". Those gateways are directly connected through a wired network to the university intranet. A few other gateways, deployed for other related projects and connected through mobile networks also help to improve coverage. With the right configuration and credentials, they directly transmit received LoRa packets to the LoRaWAN server.

UT3's LoRaWAN server is handling the data coming from the gateways, by deduplicating messages and validating the integrity of devices. When data is received and accepted, this server publishes it to an MQTT broker using a specific topic, depending on the source device. Users wanting to access the data may either subscribe to the relevant topics or build an application that uses them.



UT3 LoRaWAN network architecture

AI-Based Data Analysis from ULR

We propose an energy-aware, AI-assisted, recommendation system for the building tenants' behaviours. After keeping track and continuous processing of data on tenants' energy consumption and comfort preferences, our system learns a relation between the two -often contradicting parameters and helps users (building tenants in our case) achieve that interactively by continuously recommending actions to them.

In the following sections, we present the general architecture and main elements that make up our proposed system. We later dive into the details of each layer and explain the inner workings of its components.

System overview

Our proposed recommendation system is made up off three different modules of software and hardware implementations, each having a necessary role for the other one:

1. A first layer consists of an infrastructure; this serves for physical data collection and transmission to upper levels.
2. A recommendation engine on the server-side, powered by a reinforcement learning agent, which learns from the data collected by IoT devices, and provides recommendations to the user through the application layer.
3. Data presentation will be assured by a set of user applications, each with precise specifications targeted for well-defined user personas.

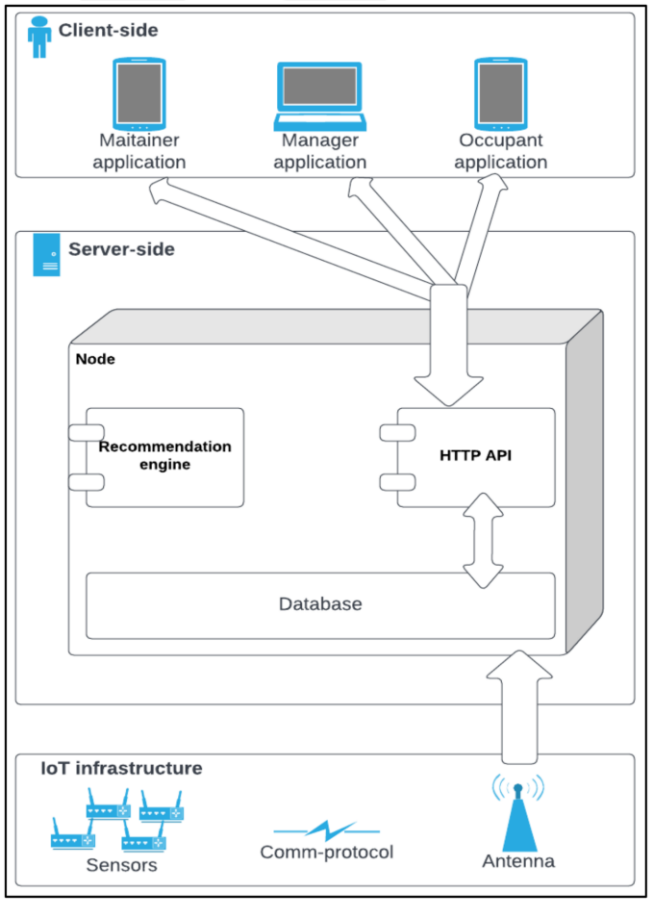


Figure 1. System architecture

In the following sections of this report,
we will go into the details of the two first layers, which constitute the backend of our system.

L3iTRANSNET server hosting the AI-Based Data Analysis from ULR

Infrastructure: Data collection and transfer

Here, we will present the infrastructure needed in the system. Mainly, the fundamental part that collects real-world raw data that is needed for our system to learn from later.

The IoT infrastructure is composed of three main components:

- **Sensors:** We will rely on sensor devices to track the physical state of building rooms and offices, this includes changes in temperature, humidity, and luminosity levels, as well as detection of door and window openings. This set of information will be useful later for our recommender system to make adequate recommendations given the room state (combination of this information).

For the physical sensors, we opted for two types of devices to serve our use case:

- HVAC sensor: sensors to measure temperature, humidity, light and motion that support long range communication.

- Magnetic sensor: to detect the opening/closing of doors and windows.

Moreover, we can heuristically detect if room lightning is turned on or off by checking the difference of brightness levels in and out of the room, taking into consideration the window status which is assumed to be collected thanks to the magnetic sensors.

We model the energy consumption for each room symbolically given the status of the heaters and lights. This is because the energy consumption for current heating systems in most buildings is centralized, which doesn't allow knowing specific consumption levels of each office's heater. Should there be a modern heating system deployed in the future, which allows connection and data retrieval, then this part of energy measurement would be significantly easier and more accurate.

- **Antenna:** The sensor devices will transmit their collected data to a central antenna shared between distant buildings. The role of the antenna is to receive that data and feed it to the remote server for further processing and storing. The antenna will need to behave as a robust IoT gateway; it should resist diverse physical factors such as humidity, dust, wind, rain, and extreme heat. Given the sparsity of deployed sensors and the long distance between buildings, the antenna should also support long-range communication protocols to establish a low-powered WAN connectivity between our IoT devices.

- **Communication protocol:** As mentioned in the previous sections, we will need a long-range communication protocol that allows us to carry the collected data from the sensors to the antenna over possible large distances. For that we chose the LoRaWAN protocol. Commonly used in the context of smart cities, industrial monitoring, and even smart agriculture, this protocol uses LoRa technology to connect multiple sensors or IoT devices that are generally characterized by very low energy consumption levels for longer functional autonomy. This description that fits our set of sensory devices, besides the factor of the long distance between them, makes this protocol a logical choice for us to use in our infrastructure.

Infrastructure: data storage and processing

The data gathered from the infrastructure is fed to an upper layer of our system, where it is stored in a central database, can be retrieved through an HTTP server, and processed with our recommendation engine during the learning phase.

Data storage

The data received from the antenna will be normalized and saved in a PostgreSQL database; a popular relational DBMS that is known for its reliability and ease of scalability.

The same database will also store data entities that are needed by the application layer of the system. An entity–relationship model of this database is illustrated in figure 2:

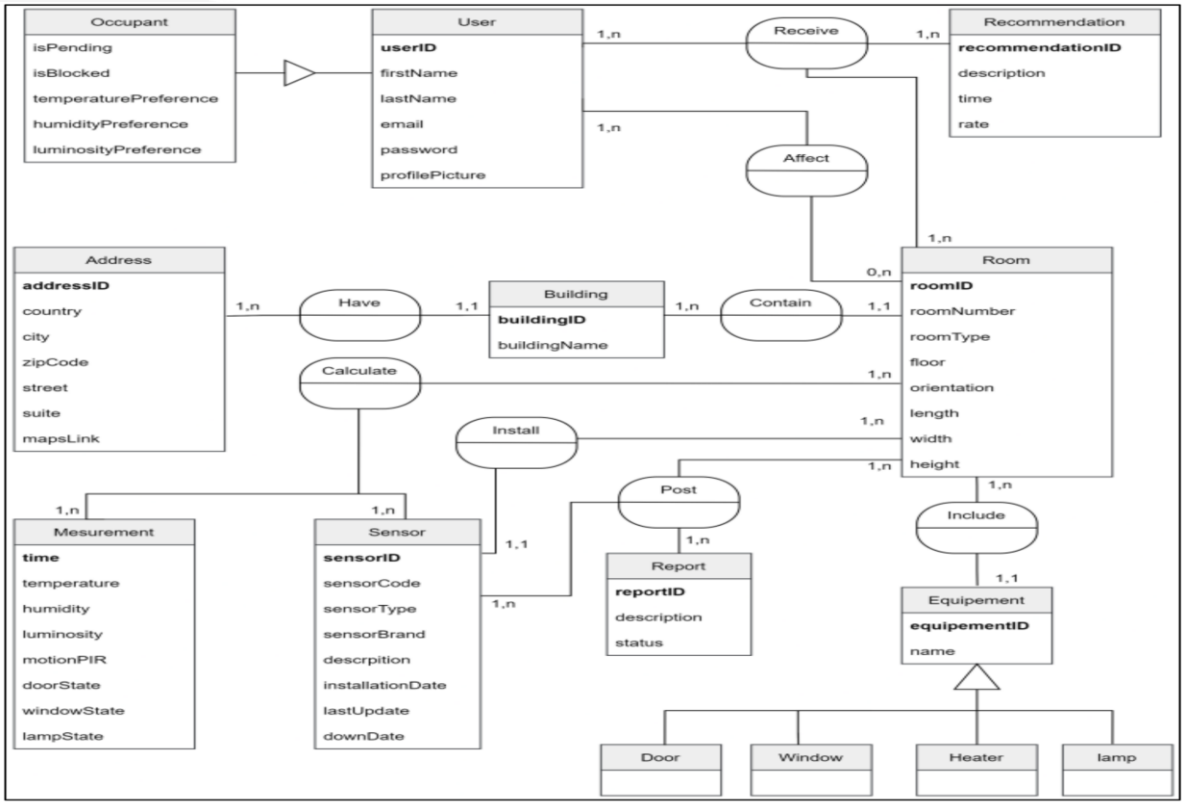


Figure 2. Entity-relationship model

API Gateway

As previously explained, the heterogeneous data in the database will be needed by different services of the system, i.e., the recommendation engine that will use parts of it in its learning, and multiple user applications that use other parts of it in their operation. To ensure this data delivery in an efficient and straightforward way, we add an HTTP API gateway that will serve as a logical interface to our database. This will allow other services (either from frontend or backend) to retrieve their needed data through HTTP requests to the API instead of accessing directly to the database itself. Moreover, the gateway will facilitate access to other server-side services (e.g., the recommendation engine) from external user applications.

The core features of the API can be categorized into controller classes as shown in figure 3, each having a single responsibility according to a specific entity. This generally translates to its typical CRUD tasks (short for create, read, update, delete), that user apps often need to perform on its corresponding database tables.

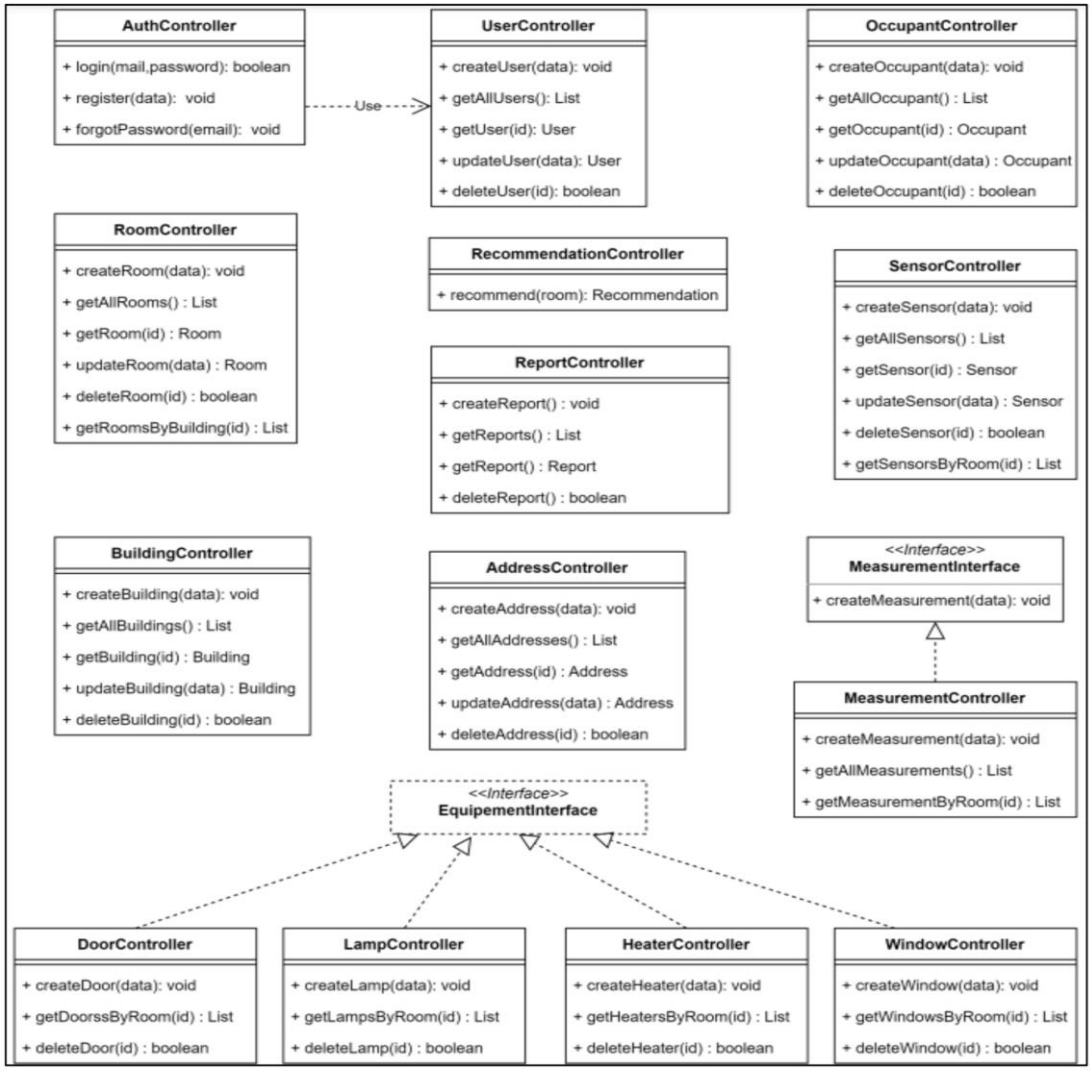


Figure 3 Controller/Interface class diagram

Recommendation engine

Contrary to the other controllers in the previous figure, the "Recommendation Controller" doesn't interact directly with the database but rather with a recommendation engine, which is a separate service in the server-side layer of our system.

The role of the recommendation engine is to help users obtain a comfortable environment in their offices, in a way that consumes minimal energy. A basic example is when a user forgets the heater then starts feeling too hot, in this case they can get a friendly notification that recommends adjusting the heater temperature in order to reach their desired level of comfort (less heat in this case).

In practice, user preference may include more factors than just temperature (e.g., humidity and indoor brightness level), and the room energy might depend on many other factors (e.g., status of window or door). Moreover, the optimal state that a user may wish

will often require a series of actions done over a time horizon to achieve it (e.g., open the window for a while to air the room and reduce humidity, then close it and turn heater on for warming up). This makes it difficult for traditional methods to handle as the search space quickly becomes large.

Having these constraints, we opted for a reinforcement learning -based approach, in which an agent learns to interact with its environment in order to maximize a reward signal. The agent learns through a process of trial and error, continuously taking actions and receiving feedback in the form of rewards or penalties.

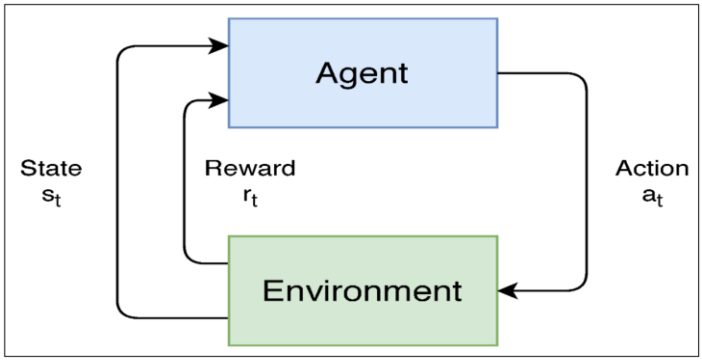


Figure 4. Reinforcement Learning workflow

In our case, the goal is to maintain a comfortable temperature for the occupants while minimizing energy consumption. We can frame this problem as a reinforcement learning problem, with the following components:

- **The agent:** a control system that adjusts the heating and cooling in the building.
- **The environment:** the office itself, including the temperature, humidity, and other factors that influence the comfort of the occupants.
- **The state:** the current conditions in the building, including the temperature and humidity.
- **The action:** the recommendation made by the control system for users (e.g., turning up the heater or opening a window).
- **The reward:** a mix of positive reward for maintaining a comfortable temperature and a negative reward for using too much energy

The control system (the agent) learns through trial and error which actions lead to the highest reward, by adjusting its recommendations about heating and cooling in response to the state of the environment and receiving feedback in the form of the reward signal. Over time, the agent learns a policy that maximizes the expected long-term reward, in this case maintaining a comfortable temperature while minimizing energy consumption.

This reinforcement learning process can thus be summarized in an algorithm as follows:

1. Initialize the policy and other necessary data structures.
2. Initialize the current state and the time step.
3. While the learning process is not complete:
    1. Observe the current state of the environment
    2. Based on the current state and the policy, select an action to take
    3. Execute the action and observe the reward and the next state
    4. Update the policy function based on the reward and the next state
    5. Set the current state to the next state
    6. Increment the time step
4. End the learning process.

From an architectural point of view, the recommender engine will be deployed as a standalone service within the control layer. It can be accessed through an API interface to get recommendations, and it itself will be communicating with a database through the API to retrieve data for learning (used to constitute state and calculate reward).

The RL system can be tweaked to scale efficiently to multiple buildings and university campuses. A straightforward way to do this is to deploy instances of the agent into different edge nodes (e.g., campus building) while leaving the database and learner within the central controller. This means the learning process will happen in one place using data from the infrastructure and feedback collected from all different agents, then with every improvement in the performance, it broadcasts the new model to override those deployed agents on the edge. This is especially efficient in terms of compute power, as it dedicates only a single centralized resource for training while others remain lightweight and just execute the learned model. A similar distributed approach would be the usage of federated learning, where each node will contain both an agent and a learner, constituting a standalone environment for each building that learns only from its own data. Next, these agents may exchange their models' parameters to assess their performances, and then perform updates towards the better model at each timestep. This can be advantageous in terms of data privacy as the agents here learn from their local databases and only share their learned parameters with other agents of other buildings (and not the data itself). To sum up, these two approaches can both be adequate in one way or another, given the future directions of the system and its functional specifications.

Replication of the demonstrators

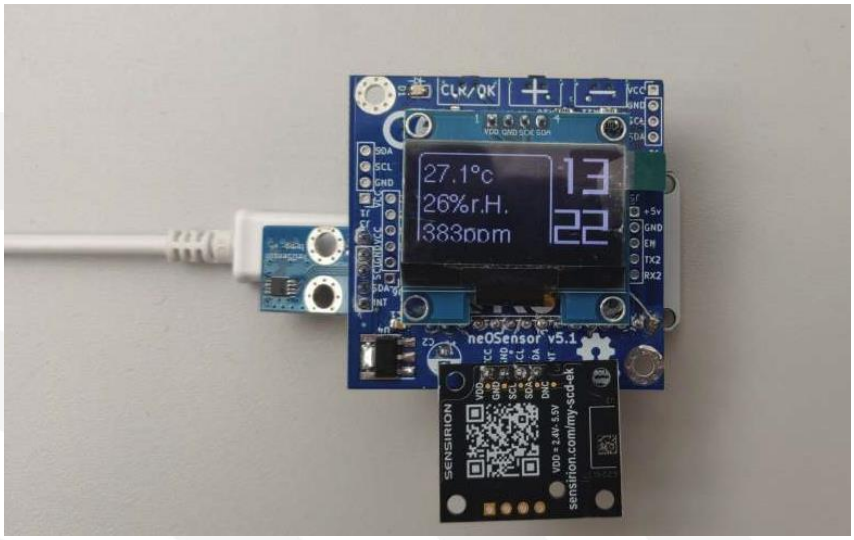
TO DO: Explain what you did with the received demonstrator/algo from other partners. Where did you deploy them? Why are they deployed in this location? What do they monitor?

The replication of IoT demonstrators refers to the process of duplicating an IoT solution or demonstration that has already been developed and tested by a partner. The purpose of replication is to extend the impact and reach of the original solution by making it available to a wider audience. In Tr@nsnet, this is done by replicating the solution in other locations by making it available to other institutions. The replication process typically involves duplicating the hardware and software components of the original solution, as well as any supporting processes and procedures. The goal is to ensure that the replicated solution is identical to the original in terms of functionality, performance, and reliability.

At UPM

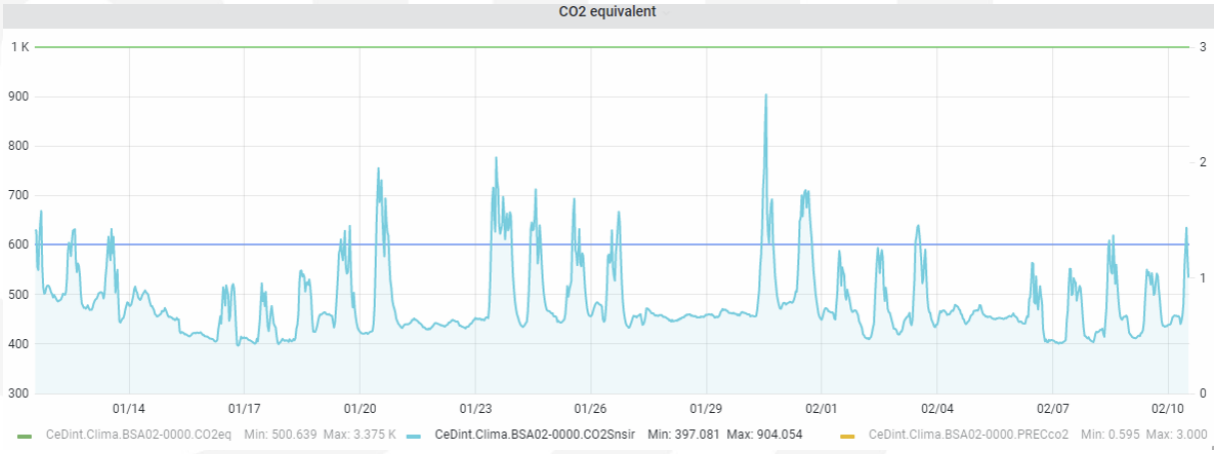
UPM has acquired the necessary components to mount the neOSensor devices and sent them to UT3. After receiving the 5 mounted and configured devices, we have deployed them in our HW laboratory, comparing their sensing performance with BatSense devices and commercial sensors.

First interaction with the neOSensor is quite straightforward, it just requires entering the Wi-Fi access point (without credentials). Once we connect to the sensor Wi-Fi, the configuration of the device is enabled, for which we must connect it to our local Wi-Fi network and select the advanced options according to our needs.
In this replication, all neOSensors devices have been configured in neOCampus sandbox mode, which does not allow access to the sensOCampus web server, but leaves the device operating in the default mode defined by UT3. In this mode we have access to the different environmental parameters by visualizing them through the LCD screen of the devices.



neOSensor device working in sandbox mode

In order to check their correct operation, we have deployed the neOSensor devices close to BatSense devices (temperature, humidity and luminosity) and commercial sensors (CO2). The results obtained with both neOSensor and BatSense devices were very similar, validating both IoT devices. Also, the CO2 sensor values from neOSensor device are very aligned with those from the commercial sensor.



CO2 data collected from the neOSensor device

BatSense and neOSensor devices mounted in wall of the HW laboratory at CEDINT-UPM

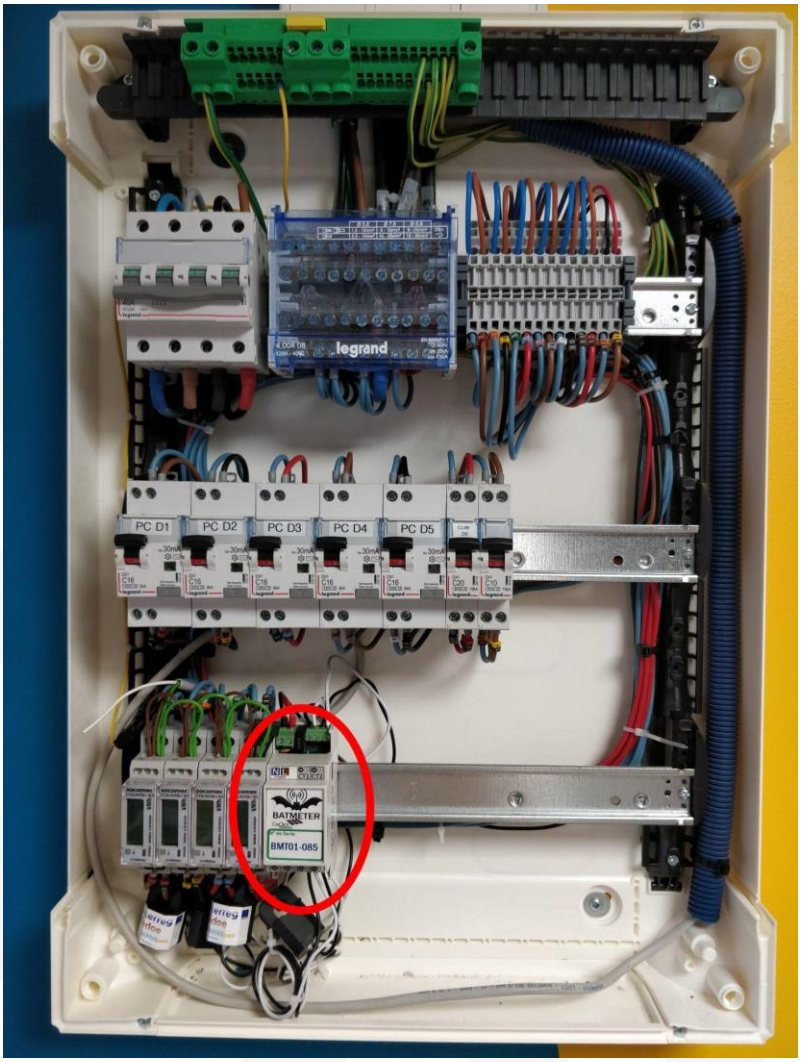
As a possible future work, it would be convenient to obtain the access credentials to the sensOCampus web server in order to change the data storage point and to integrate the neOSensor with UPM data storage, visualization and management platforms.

At UT3

We received one kit of Bat devices from ULM containing:

- One BatLink, the network sink aggregating the sensor devices coming from a 6LoWPAN link and publishing them on a MQTT broker, using a standard wired ethernet connection.
- Two BatSense, sending temperature, humidity, luminosity and presence through a 6lowpan link
- Two BatMeters, sending power usage data also through a 6lowpan link

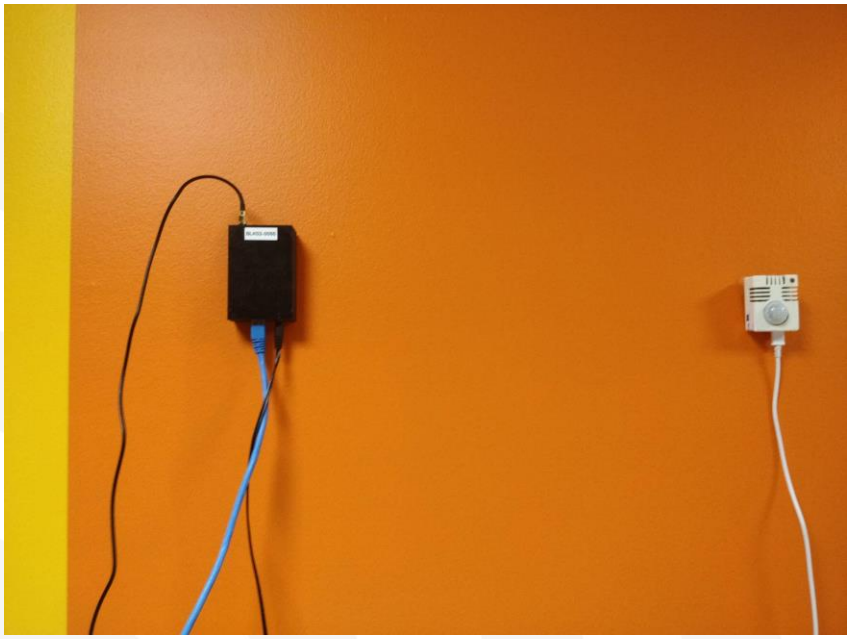
One BatSense and one BatMeter are deployed in the “Supervision Room” located in the IRIT Lab at UT3, monitoring the room. The BatMeter is installed in the electrical panel alongside with an industrial power meter. We expect to compare the values and to find little to no differences.



Electrical panel containing the BatMeter

OBJ

The devices are located in this room because it is one of the few meeting the requirements for deployment, meaning that we have access to the electrical panel, so the BatMeter may be installed. Also, we can easily control network traffic and connect wired devices painlessly, facilitating BatLink set up.



The BatLink and a BatSense deployed in the Supervision Room

As devices remain in their initial configuration, data is sent to the BatLink every 5 minutes. The BatLink is configured to forward data to the neOCampus MQTT broker prefixed with the topic TestTopic/transnet/.

It is worth mentioning that the BatLink comes with a self-hosted MQTT broker. This local broker receives all messages and forwards them to a given external broker, while the external broker forwards back any message about the BatNet. However, our broker configuration seemed incompatible with that functionality, so a python script to replicate it was made.

At ULR

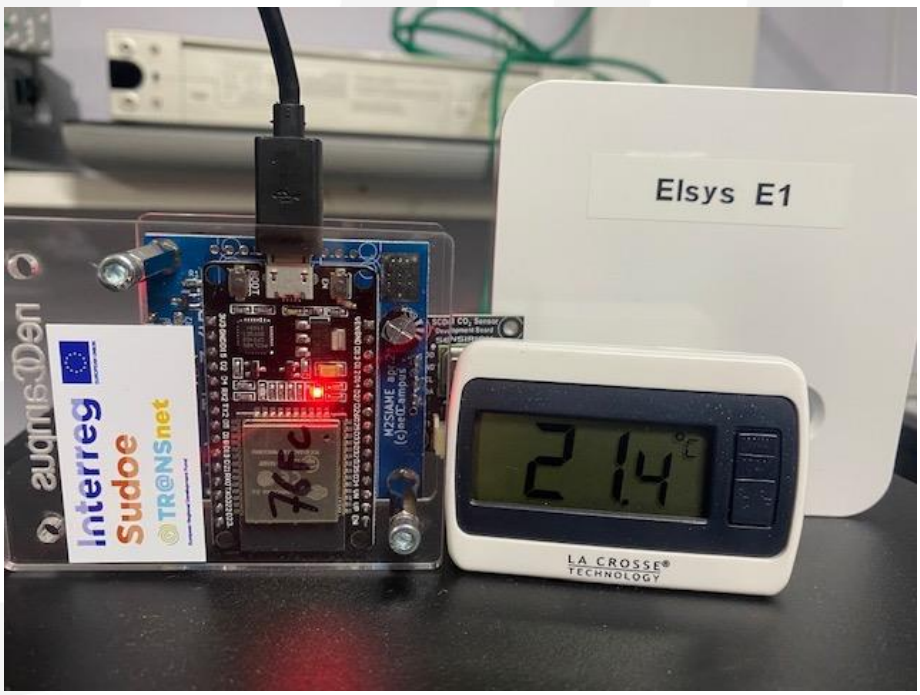
After ordering the basic chipset recommended by UT3 to build neOSensors and having shipped them to UT3 premises for manufacturing the neOSensor devices, UT3 returned us with a kit of five (5) neOSensor devices. These devices allow monitor the humidity, temperature, luminosity and CO2 level.

These devices are aimed to be deployed in the L3i Lab at ULR, in the vicinity of the commercial sensors (Elsys ERS) used by ULR to monitor the humidity, temperature, luminosity and CO2 level within its spaces.

Our objective behind deploying these sensors is twofold:

1. Allowing to validate the replicability of the neOSensors in a different setting;
2. Showing the ability of neOSensors to be a good (and cheap) alternative to commercial sensors as we expect to compare the values obtained by both (neOSensors and Elsys ERS) and to find little to no differences.

To do so, three (3) neOSensors had been deployed in the L3i lab premises at ULR, two (2) in offices and one (1) in a server room, all three in the vicinity of the commercial sensors (Elsys ERS) used by ULR, allowing to monitor the corresponding room. The following two pictures depict one neOSensor deployed close by an Elsys ERS sensor.



Joint deployment of neOSensor devices and Elsys ERS devices in L3i server room

Joint deployment of neOSensor devices and Elsys ERS devices in an L3i office room

Even though, ULR Smart Build platform is not using MQTT, it had been quite easy to integrate the neOSensors into the ULR Smart Build platform, with minimal configurations (after 1 hours of joint work with an UT3 engineer, remotely, through videoconferencing platform). More precisely, the neOSensor devices had been configured so that they interconnect with ULR's Smart Building Data Platform through the deployed LoRa antenna. ULR deployment make use of the same LoRa antennas as those used by UT3 (i.e. MultiTech Conduit LoRa Gateway Outdoors) but then makes use of a slightly different data flow management software platform (i.e. starting from the server that collects the data from the LoRa antenna).

Also, comparing the values returned by the neOSensor devices, these values differ by less than 0.5% the values returned by the Elsys ERS devices during the entire monitoring period. This is a marginal percentage, validating the fact that the custom made neOSensors gives quite accurate results as the Elsys ERS commercial sensors.

These two statements indeed validate the two objectives mentioned before about the ability of the custom made neOSensors to be easily installed, made operating and providing quite accurate results. So, as such, we can confirm that the custom made neOSensors are good candidates for replacing the costly commercial sensors.

User Involvement in the Deployment Process

At UT3

During the replication process, involving users in our university living labs required building trust and establishing a strong relationship with the user community. This was done through open and transparent communication, as well as a commitment to addressing their needs and concerns.

For instance, in the design of neOSensors we organized crowdsourcing to gather ideas from the user community to upgrade the design and co-creation workshops to bring together students and researchers to co-create its latest release and to prototype the LoRaWAN-enabled neOSensor.

At ULR

During the deployment process of both the Elsys ERS and neOSensor within the ULR premises, we could also get a first measure of the acceptability of the campus users about having such devices in their offices. While no issue had been reported by having such devices in technical rooms (storage rooms, server room, cleaning staff room, boiler room, relaxation room …), in offices it had created many discussions with the technical team and even over the lab's forum. Indeed, not clearly informing the occupants about the goal behind such device deployment created immediate rejection and controversy by some users, even before the devices got wired and connected. While most of the users didn't mention anything about having such devices installed in their offices, few users showed a strong argument against, unless having full knowledge about the data being collected and its use, which is in line with the GDPR principles though. Few users were in favour of having their office being part of a Living Lab experimentation. This first trial at ULR premises, comforted us in the necessity of conducting large communication campaigns before starting our future large-scale living lab deployment at ULR, along with users' consultation and clear information about the data use, which is in accordance with the GDPR regulation. More than that, this comforted our idea about following a self-data approach giving to the campus users access to all collected data through a mobile application (cf. activity 2.5) in order to show energy consumption predictions and influence their behaviour within the campus.

Conclusion

Based on the objectives of the activity, the participating universities successfully replicated the achievements of each other. The achievement of UT3 in deploying neOSensors and a private LoRaWAN network with MQTT data collection, were successfully replicated by the other universities. Similarly, UPM's BatNet deployment also replicated, along with ULR's AI-based recommendation system.

The user involvement in the deployment of these demonstrators was discussed in the report. At UT3, user involvement was achieved through open communication, co-creation workshops, and crowdsourcing to upgrade the neOSensor design. Deployment of sensors in offices created controversy at a partner university, with some users rejecting their deployment without full knowledge of the data collected and its use. This experience emphasized the importance of large communication campaigns, user consultation, and clear information about data use in accordance with GDPR regulation for future living lab deployment. The self-data approach consideration was also strengthened to give campus users access to all collected data through a mobile application to influence their behaviour within the campus.

Overall, the activity was successful in achieving its objectives, and the participating universities were able to replicate each other's achievements in a manner that allowed for knowledge exchange and collaboration. The results of this activity can serve as a model for future collaborations between universities to enhance their research capabilities and knowledge base.