HEART D4.8 – Final BEMS Hardware Components

January 2020

HEART

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No 768921.

Project title	**Holistic Energy and Architectural Retrofit Toolkit**
Project acronym	**HEART**
Grant Agreement No.	**768921**
Project call	**EEB-05-2017 Development of near zero energy building renovation**
Work Package	**WP4**
Lead Partner	**CTIC**
Contributing Partner(s)	**STILLE, POLIMI**
Security classification	**Public**
Contractual delivery date	**31/01/2020**
Actual delivery date	**31/01/2020**
Version	**1.0**
Reviewers	**Fabrizio Leonforte (POLIMI), Mario Maistrello (ZH)**

HISTORY OF CHANGES

Version	Date	Comments	Main Authors
0	16/12/2019	First Version of Deliverable Development Plan	Martin Álvarez-Espinar (CTIC)
0.1	27/12/2019	First version of the document	Martin Álvarez-Espinar (CTIC)
0.2	09/01/2020	MIMO & HP configurations updated. New section of energy management devices.	Daniel Ibaseta; Martin Álvarez-Espinar (CTIC)
0.3	13/01/2020	Quality review	Fabrizio Leonforte (POLIMI)
0.4	16/01/2020	Changes in the workflow of DHW fan-coils.	Martin Álvarez-Espinar (CTIC)
1.0	28/01/2020	Issue of final versione	Martin Álvarez-Espinar (CTIC)

TABLE OF CONTENTS

Disclaimer

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No 768921.

EXECUTIVE SUMMARY

The Building Energy Management System (BEMS) is one of the key components of the platform. It implements the operation logic to control and distribute the electric energy flux, thermal energy and information, coordinating the main devices —MIMO, HP, thermal storage tanks, PV and fan-coils—, and monitoring their status and configurations. This document describes the hardware components that enable communication and interaction of devices within the BEMS: **Heat Pump Gateway**, **MIMO Gateway**, **Smart fan coil interfaces** and **Building Controller and Gateway**.

All the BEMS elements are linked among them through different networks that protect and guarantee the continuous operation of the system. The system is based on **NarrowBand-Internet of Things (NB-IoT)** as a direct mechanism to access the Internet, and a **WiFi mesh** network. The software of the main interface components of the building (i.e., **MIMO Gateway**, **Heat Pump Gateway**(s), **fan-coils**), and the **Building Controller and Gateway** implements the standards of the **W3C Web of Things**. It will maximise the level of interoperability and scalability of the system.

The Building Controller and Gateway is the core component of the IT system in the building. It acts as a proxy between the devices deployed in the building (i.e., fan-coils, Heat Pump and MIMO) and the external services such as the cloud control platform. This device, implemented on an **UP Squared board**, plays the role of offline BEMS controller.

Both the Heat Pump and the MIMO are managed in a similar way, since they implement an internal **Modbus/TCP server** as principal communications channel. These devices will be connected directly to gateways that will translate the Modbus protocol to the Web of Things model. These gateways are implemented on **UP Board devices** and become the interface for those main appliances.

Smart fan-coils have an internal on-board control system that acts directly on the device sensors and actuators. This control system is connected to the BEMS requests through an interface module that allows the BEMS to controls the basic operations of the appliance (i.e., ON/OFF, critical situation, season change, and fan speed). Apart from the internal parameters of the appliances, smart fan-coils include sensors to monitor the environmental variables such as air temperature and relative humidity of rooms.

This document is an **update of the deliverable D4.7** including the latest adjustments and configurations.

1. HARDWARE COMPONENTS OVERVIEW

The Building Energy Management System (BEMS) is one of the key components of the platform. The BEMS implements the operation logic to control and distribute the electric energy flux, thermal energy and information, coordinating the main devices —MIMO, HP, thermal storage tanks, PV and fan-coils— in the building. The BEMS also enables a direct interaction with the building's users, reporting about the building performance in terms of energy, allowing them to control it.

As shown in the following figure, the main components of the BEMS have interactions among them in terms of electric energy, thermal energy and information. This document describes the communication and control devices involved in the operation of the system but focused on the information flow.

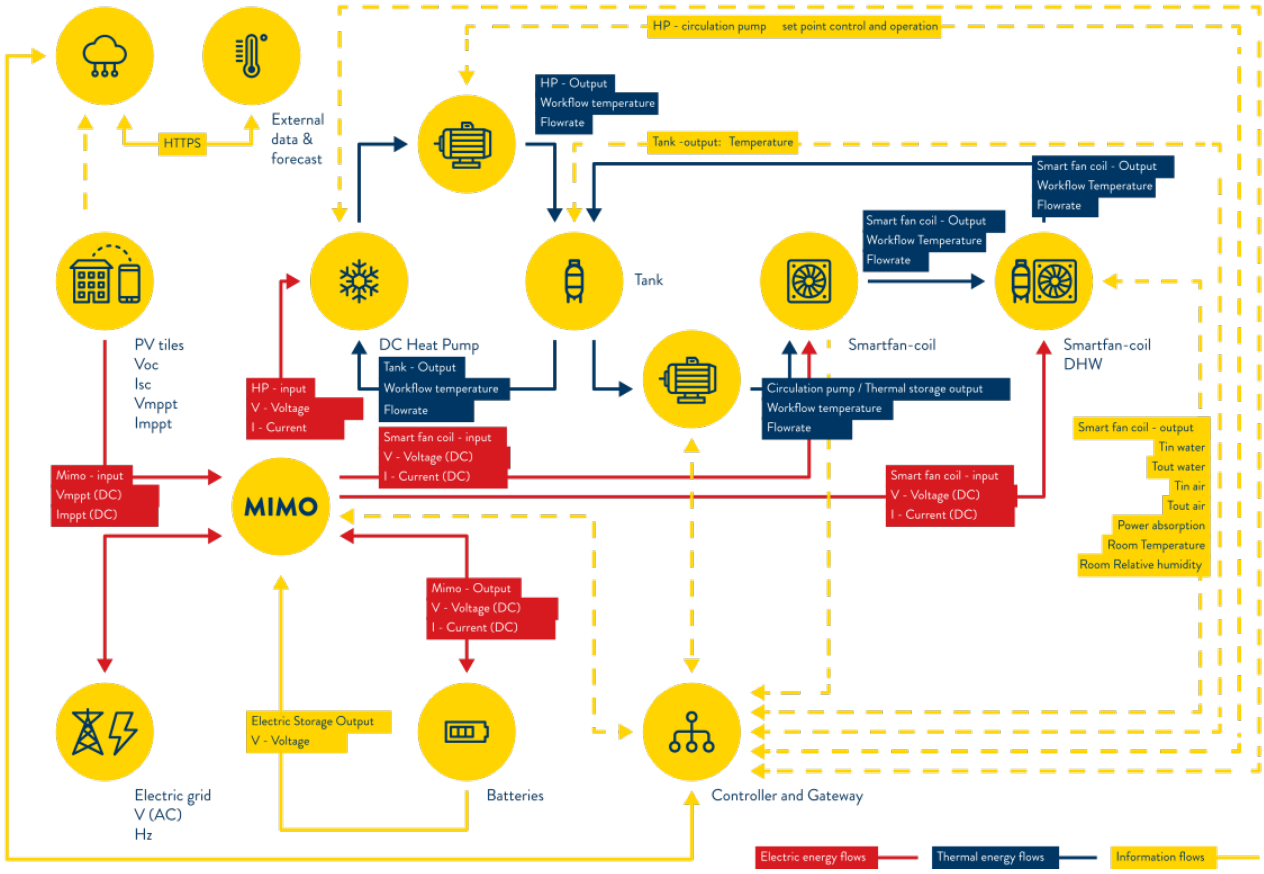


Fig. 1. BEMS components and the interaction among them

Having this in mind, the BEMS is implemented by 3 different levels of controls:

Level 1: a **root control system** (firmware) embedded in each HEART component (e.g. heat pump, MIMO and smart fan coils); at this level the control will be focused to the protection of the operation of the components (e.g. over-temperature, over-current, short circuit, etc.) and on the internal management of each appliance;

Level 2: a basic ***offline*** **control logic**, physically implemented at building level, in a specific hardware component called Building Controller and Gateway; and

Level 3: an adaptive-predictive ***online*** **logic** implemented on the cloud platform.

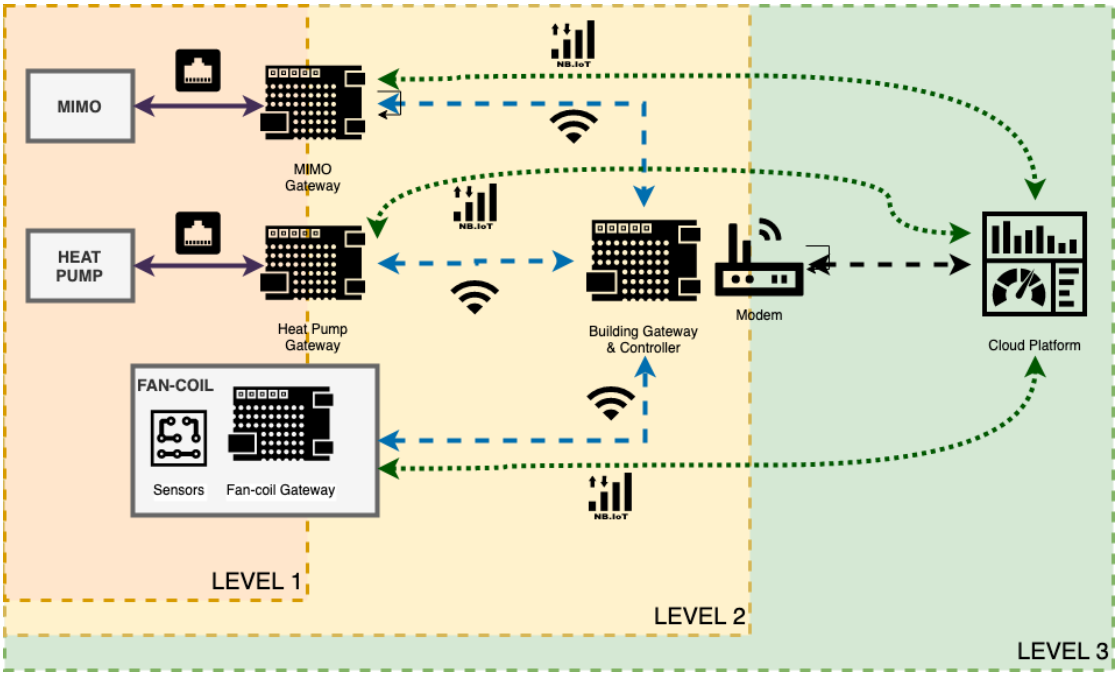


Fig. 2. BEMS logic control levels

The **Building Controller and Gateway** is the cornerstone of the level 2, needed also in the level 3. It provides the basic functions of the BEMS, monitoring and controlling the system in real time. Such controller is also connected to the cloud, to a platform that enhances either the monitoring of the BEMS, as well as its control decisions thanks to its analytic capabilities and access to third-party data —i.e., environmental information, weather forecast, etc. The local **Building Controller and Gateway** is always connected with the building's appliances and devices to guarantee stability of the system in case of disruption of the Internet connection. In order to guarantee availability of the system, the Building Controller and Gateway will be installed on a symmetric configuration on two Up Squared devices.

The selection of the main communication technologies of the BEMS was based on the premise of the uninterrupted operation of the system. Thus, the system will run over two different communication protocols: **NarrowBand-Internet of Things** (NB-IoT) and **WiFi**. This redundant approach will be used within the lifetime of the project. Once the project is finished, the system will rely on NB-IoT communications, keeping the WiFi-based subsystem, as a backup mechanism just in case NB-IoT does not demonstrate full maturity and reliability, or just for eventual low coverage.

In the following sections, all the devices and sub-components required for level 2 and level 3 will be explained in detail.

1.1. NETWORKS AND INTERACTION WITH BUILDING DEVICES

The BEMS is based on different networks that provide devices and the control platform with different levels of communication. As shown in the following figure, both the **MIMO** and the **Heat Pump**(s) have their own local, *ad hoc*, networks to communicate with their respective local interfaces (i.e., **MIMO Gateway** and **Heat Pump Gateway**). The **MIMO Gateway**, **Heat Pump Gateway**(s), **fan-coils** and the **Building Controller and Gateway** will be attached to the main building **wireless LAN**. These devices will be also connected directly to the Internet via **NB-IoT**.

Furthermore, the building has access to the Internet through a wired **Broadband Modem**, connected to the **Building Controller and Gateway** that is configured to enable a reliable connection of BEMS and the cloud-based adaptive-predictive control logic. A network router establishes strong security policies and has available a minimum bandwidth in order to guarantee performance and quality of communications.

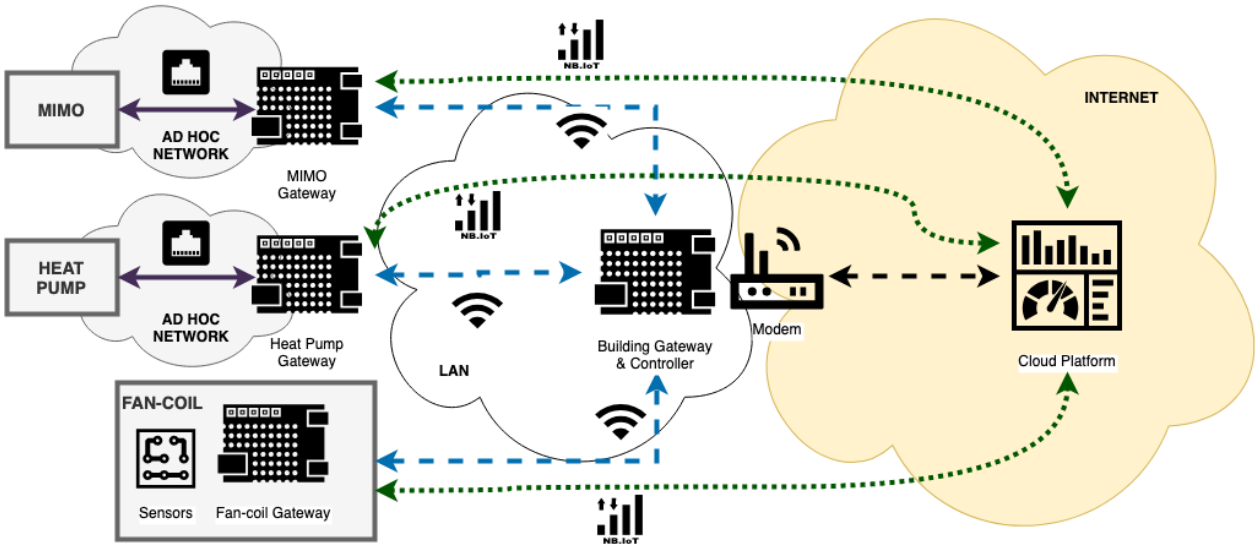


Fig. 3. BEMS information networks

Based on the Web of Things paradigm

The software of the main interface components of the building (i.e., **MIMO Gateway**, **Heat Pump Gateway**(s), **fan-coils**), and the **Building Controller and Gateway** implements Web of Things *servients*. A *servient* is the stack that implements the Web of Things (WoT) building blocks, which can host and expose *things* and/or host clients that consume *things*. A *servient* can host and expose *things* (server role) and/or consume *things* (client role). In this case, WoT defines a series of Application Programming Interfaces (APIs) that enable the access to different devices through the HTTP RESTful protocol implemented in their communication interfaces.

In the WoT paradigm, all the *things* are described as Web resources, identified by Uniform Resource Identifiers (URIs). Third party applications, including the Cloud-based Platform, may interact with them using RESTful services. This offers the system low complexity and loose-coupling stateless interaction as specified in the next sections.

About NB-IoT

NB-IoT is a narrowband radio technology developed by the 3GPP (3rd Generation Partnership Project) released as standard in mid-2016. This technology was designed to address some of the common challenges of Internet-of-Things projects: lowering the cost of communications; extending signal coverage; increasing the battery life; reducing latency; enabling scalability of devices.

Using this technology, there is no need for a direct Wide Area Network (WAN) connectivity for all end devices. The band range (300-3400 Hz) allows a high penetration in buildings. Also, NB-IoT offers a theoretical 250 Kbits/s rate, with a maximum of 85 Kbit/s probed in studies[i] that makes it cover the requirements of the platform.

Another interesting feature is the low consumption, which depends on the connectivity (output power). According to the sample specifications of a NB-IoT module[1] it varies from 65 mA to 250mA.

This technology also provides a high level of security of communications. All NB-IoT-connected devices have unique SIM cards, containing credentials and subscriber data. It also supports encryption using 256-bit keys and possibility of establishing Internet Protocol Security (IPsec) tunnels for end-to-end communications.

Currently, the deployment of NB-IoT in the world is reduced to only a few pilots in major cities in Europe and Asia but it is expected to have a complete deployment in France and Italy by the end of 2019, so the pilots will be covered by, at least, a network carrier providing service.

[1] http://www.quectel.com/UploadFile/Product/Quectel_BC95_NB-IoT_Specification_V1.3.pdf

2. WIRELESS LOCAL AREA NETWORK

This section specifies the hardware required for deploying and running a **local private network for the BEMS**, and the communication with the rest of the HEART's components. This Local Network is based on the IEEE 802.11 (WiFi) standard. It offers signal coverage to all in-building devices, including exhaustive security measures to protect the network and guarantee privacy. The Building Controller and Gateway is in charge of activating and monitoring the type of physical connectivity of devices, either NB.IoT or WiFi.

In order to avoid vulnerability and maximise reliability and uptime of the system, the local network implements the following features:

- The **Service Set Identifier** (SSID) of the network is **fixed and hidden** to avoid discoverability. It is secured and identified with a fixed SSID to protect unauthorized network access by utilizing a pre-shared password.
- The interface components of the core network components (i.e., **Heat Pump Gateway**, **MIMO Gateway**, **Building Controller and Gateway**, and **fan-coils**) support the IEEE 802.11b/g/n wireless networking standard, attached to the high-throughput wireless local area network on the **2.4 GHz band** to guarantees the **maximum range** of the signal.

In order to design the best configuration for the wireless local network, we need to consider four key aspects: coverage, capacity, performance, and installation.

- **Coverage** is the area where devices WiFi be able to connect to the network. Since, the local network intend to cover all rooms in all building's apartments, this variable depends on the physical configuration of the building —floor measures, walls, etc. Devices will have ensured high-availability so the design has the premise of having redundancy of access points. For this reason, the LAN is configured in the **2.4 GHz** band.
- **Capacity** is the ability of each wireless access point to handle a certain number of devices attached to the network. The network design takes into account a minimum number of sensors, actuators, and devices distributed within the building.
- **Performance** of the network implies a minimum available bandwidth enough for devices and applications to perform their tasks in full operation. It is important selecting technologies and systems that offer the highest levels of performance and scalability. Thus, the design has into account the latest high-speed WiFi technologies, such as 802.11g/n, rather than cheaper legacy products with lower performance.
- **Installation** and deployment of the communication wireless network in the building depends on the distribution of rooms and possibility to connect the main communication devices via Ethernet cables to guarantee reliable and fast operation. This is not possible in all cases, such as in the Italian pilot, so a **non-intrusive deployment**. So, a WiFi mesh network is deployed in the building spaces.

The final network configuration depends on the building measurements, so floor plans and blueprints are important to define the ideal infrastructure.

2.1. NETWORK ARCHITECTURE AND IMPLEMENTATION

The Wireless Local Area Network deployed in the building covers all rooms and devices in them. The main networking devices are connected through (Cat 5 or superior) Ethernet. An external provider will supply a **Broadband Modem** to be connected to the Internet Router —the Broadband Modem and Internet Router might be the same device, depending on the product supplied by the Internet provider.

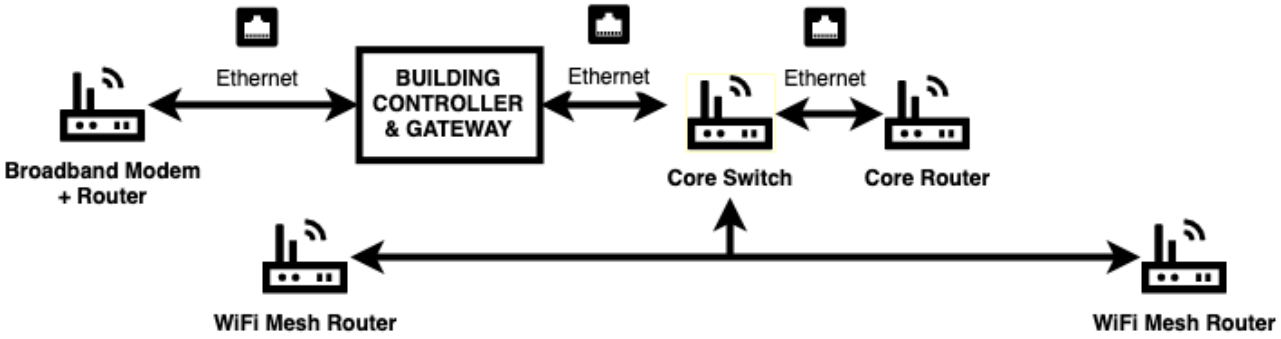


Fig. 4. Physical connection of network devices

The Internet Router will be directly linked to the **Building Controller and Gateway,** acting as a communications proxy for all the devices of the building (i.e., final sensors and actuators). This device is the only one with granted access to the Internet —with the exception of the NB-IoT-enabled devices— to increase the security configuration of the building devices.

In order to manage the LAN and the configuration of the devices in the building, there will be an internal Core Router and a Core Switch, implemented in the same device. The Core Router is a networking router that implements a DHCP server that allocates resources and purveys IPv4 directions to all the devices. The mission of the Core Switch, connected directly to the Building Controller and Gateway, is supporting the deployment of the LAN.

The Core Switch receives, processes, and forwards data between the **Building Controller and Gateway** and the rest of the devices in the building. A local Wi-Fi mesh network will be deployed in the building using **WiFi Mesh Routers** connected to the Core Switch. The WiFi Mesh Routers will be connected to **WiFi Mesh Access Points** distributed in the building to deploy the wireless network. The number of the WiFi Mesh Access Points may vary depending on the configuration of the building.

Mesh WiFi System

The solution is based on an **AmpliFi Mesh Wi-Fi System**[2], a commercial product enabling enterprise-strength network capabilities with simplicity, and with an appealing design. The AmpliFi's system is composed of AmpliFi Routers and MeshPoints, components designed to work in combination to eliminate any dead spots in the building.

[2] https://amplifi.com/docs/AmpliFi_Datasheet.pdf

Fig. 5. AmpliFi HD Mesh Router and MeshPoints

This system uses mesh technology to provide powerful wireless performance in an innovative and simple design. The main objective of the WiFi mesh network is to provide the fan-coils with WiFi connectivity with a **minimum impact for the building tenants**.

This device covers all the requirements of the system, including the following specifications:

Max. Speed	1750 Mbps
Max. Transmission power	26 dBm
Antennas	Dual-Band Antenna, Tri-Polarity
RJ-45 Ethernet ports (router)	4

The approximate cost of the **AmpliFi HD Mesh Router** is €150 (EUR), and every **AmpliFi MeshPoint HD** costs €120 (EUR), approximately.

Core Switch

The configuration of the local network will be completed with a core router/switch that will be connected through Ethernet cable, and linked to the WiFi Mesh Routers. The core switch will be implemented using a **Zyxel GS1900-10HP**[3].

Fig. 6. Zyxel GS1900-10HP (Core Switch)

[3] ftp://ftp.zyxel.com/GS1900-48/user_guide/GS1900-48_V1_Ed2.pdf

This device covers all the requirements of the system, including the following specifications:

Switching capacity	20 Gbit/s
MAC address table	8000 entries
PoE Mode	802.3af PoE; 802.3at PoE+
RJ-45 Ethernet ports quantity	8
Networking Interface	10/100/1000 Ethernet Ports
Maximum Power Consumption	96.2W

The approximate cost of the GS1900-10HP is €220 (EUR).

Italian Case Study

The following example of configuration corresponds to the network design for the Italian pilot, where only the attic supports the installation and deployment of wired connections. **WiFi Mesh access points** (APs) are installed in common areas (i.e. stairs).

The most challenging part of the configuration is the setup of WiFi Mesh access points to provide with full coverage all devices in the building, and enough bandwidth to guarantee performance of the BEMS. Also, it is important to have redundancy of wireless connectivity in case of AP failure. Although physical tests in the field are needed to ensure an adequate configuration, this assessment is based on a realistic approach, and early tests, presuming that the chosen APs provide strong signal in a range of 10 metres with just 3-4 drywalls in between the AP and the device. The installation team performed several in-field tests that confirmed the coverage of the current system design.

Two WiFi Mesh Routers will be installed in the attic, where the main network devices will be located. Also, two APs will be installed in the common areas of each floor (1st, 2nd and 3rd floor), so six WiFi Mesh Access Points in total. The following figure visualizes the wireless network configuration in the building.

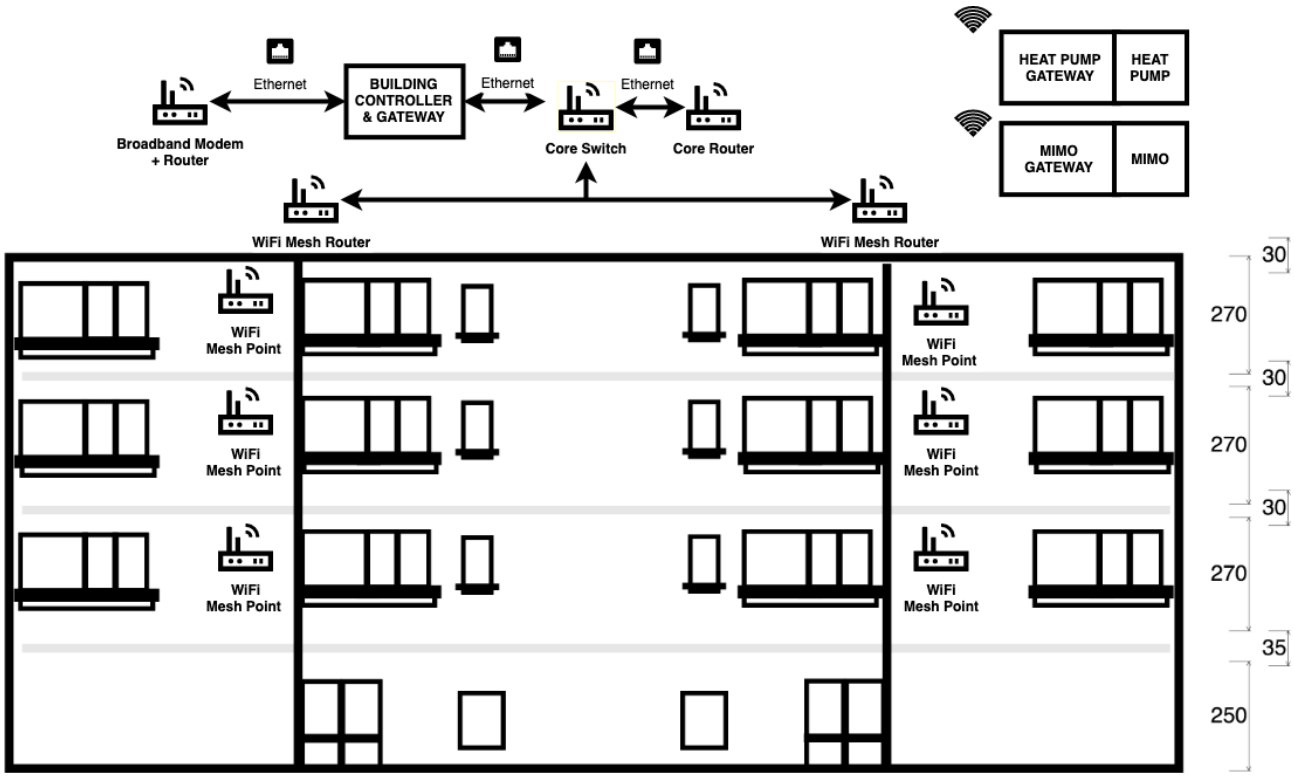


Fig. 7. Configuration of the Italian pilot local network

The complete list of components to deploy the local network for the Italian pilot is summarised in the following table:

Device	Location
(x6) AmpliFi MeshPoint HD	(x2) 1st Floor; (x2) 2nd Floor; (x2) 3rd Floor
(x1) Zyxel GS1900-10HP	Attic
(x2) AmpliFi Mesh Router	(x2) Attic
(x1) Broadband modem[4]	Attic

2.2. SETUP AND OPERATION

The Local Area Network is the basic support of the Web of Thing devices ecosystem in the building. This implies that all the *things* —i.e., sensors, actuators, and appliances— are identified by URIs, described using Semantic Web formats and vocabularies, and accessible through Web Services implemented by in the *servient*.

[4] Provided by a local supplier

Since all the things are distributed in different nodes, the wireless local network is configured to allocate specific devices in concrete IP address ranges. Allocation of device's IP in specific range of addresses enhances performance and enables discoverability of new components in the system.

The local network devices will be identified in the **192.168.1.0/255** range (network mask 255.255.255.0).

Device	**Reserved IP Address**	**Fixed?**
Core Router	192.168.1.1	Yes
Building Controller and Gateway	192.168.1.100	Yes
MIMO Gateway(s)	192.168.1.101 — 192.168.1.105	Yes
Heat Pump Gateway(s)	192.168.1.106 — 192.168.1.110	Yes
Fan Coil(s)	192.168.1.111 — 192.168.1.211	DHCP

This configuration must be implemented in the Core Router to guarantee a correct IP assignation and the right access to the devices.

The expected parameters for the WiFi Local Network is defined be as following:

SSID	Heart_WiFi_AP
Password	WiFiHEART2020
Security	WPA2-PSK
Discoverable	Hidden

3. BUILDING CONTROLLER AND GATEWAY

The Building Controller and Gateway is the core component of the IT system in the building. It is connected to the rest of the devices of the building and it is in charge of implementing the level 2 of the control logic operations. Its configuration is dynamic, since it receives setup information from external services. It acts as a proxy between the devices deployed in the building (i.e., fan-coils, Heat Pump and MIMO) and the external services such as the cloud control platform.

This device plays the role of *offline* BEMS controller, allowing the basic monitoring and control logic of the system when the BEMS components have no direct connection to the cloud platform via NB-IoT. It also acts as gateway between the building and the Internet, enabling a direct and reliable broadband access to the cloud platform, converting protocols, adapting packet formats, and translating messages between networks. It also provides monitoring and reporting capabilities for system management purposes.

The gateway will be connected to the Internet through a **Broadband Modem**, dedicating a minimum bandwidth to guarantee performance and efficient interoperability with the cloud-based BEMS.

The software stack implementing the logic will be regularly updated, installing patches to increase reliability, efficiency and security compliance. The software is implemented in Python 3, and deployed on the device using virtualisation mechanisms using **Docker**[5].

3.1. IMPLEMENTATION

The **Building Controller and Gateway** is developed on an **UP Squared**[6] board, a high performance and low power consumption board that covers all the performance, security and connectivity requirements of the system. In order to guarantee availability of this device, the Building Controller and Gateway will be implemented on a symmetric configuration for redundancy purposes.

Fig. 8. Up Squared Board

[5] https://www.docker.com
[6] http://www.up-board.org/upsquared/specifications-up2/

This Intel®-based board is designed specifically for the Internet of Things paradigm, supporting any operating system —the implementation will be based on Ubuntu Server—, and providing multiple interface connections. The main controller is implemented on this board. This piece of hardware —along with a storage hard drive— has the required process capacity to manage all the BEMS data, as well in terms of communications. This device is connected via Gigabit Ethernet to the main network router.

The Up Squared is integrated with the M.2 2230, a kit to provide this board with WiFi connectivity, also with an NB.IoT chipset integrated in a **Pycom GPy** module.

The main features of the UP Squared Board are shown the following table:

Processor	Intel Pentium N4200 2.5 GHz
RAM	8GB LPDDR4
Storage	128 GB
Connections	2x Gigabit Ethernet
Consumption	~1A

In concrete, the system uses the best configuration available (i.e., Apollo Lake M Intel® Pentium™ 4C 2.5GHz, 8GB RAM and 128 GB of storage) to guarantee the performance and scalability during the lifetime of the project. The approximate cost of this board is €300.00 (EUR).

This board is complemented with other components needed to fulfil the requirements and operation:

(×1) UP Squared board	Main board, communications and processor.
(×1) M.2 2230	WiFi kit for UP Squared.
(x1) Pycom WiFi Antenna	Antenna.
(x1) Pycom Cellular Antenna	Antenna.
(×1) GPy	NB-IoT, WiFi module.
(×1) GCBC100-2A	Power consumption sensor.
(x1) mSATA 128GB	Storage.
(x1) 1500VA UPS	Uninterruptible Power Supply.
(x1) Power supply and case	Case, power adapter, chassis

In order to guarantee the continuous operation of the system, the system may include redundancy features, duplicating the devices. Operation will be balanced depending on the availability of the devices. This includes: (×2) UP Squared boards, (×2) M.2 2230 WiFi kits, (x1) Pycom WiFi Antenna, (x2) Pycom Cellular Antenna, (x2) Power supply and case.

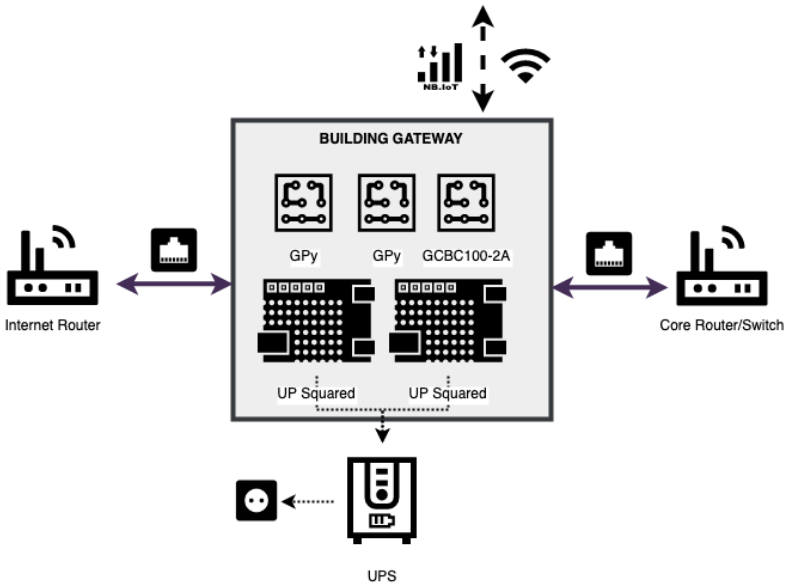


Fig. 9. Components of the Building Controller, including redundancy

The approximate cost of the simple **Building Controller and Gateway** (without redundancy) is **€1120 (EUR)**.

3.2. OFFLINE CONTROL LOGIC

The Building Controller implements the *offline* BEMS controller. It continuously gathers information of the status of the devices in the building and visualise data through intuitive dashboards. It and enables the communication with the external tools such as the Cloud Platform, allowing them to configure some parameters of the system (e.g., set point temperatures, operation limits, etc.). The system may also configure alerts depending on the specific needs during the lifetime of the project.

This network node has the IP address **192.168.1.100** allocated.

The *offline* control is implemented in different virtualised Docker containers that perform decoupled functions. The main modules that implement the *offline* controller, shown in the following illustration, are: the WoT Proxy; Building Device Manager; WoT Catalogue Updater; Building Device Monitor; BEMS Engine; Storage.

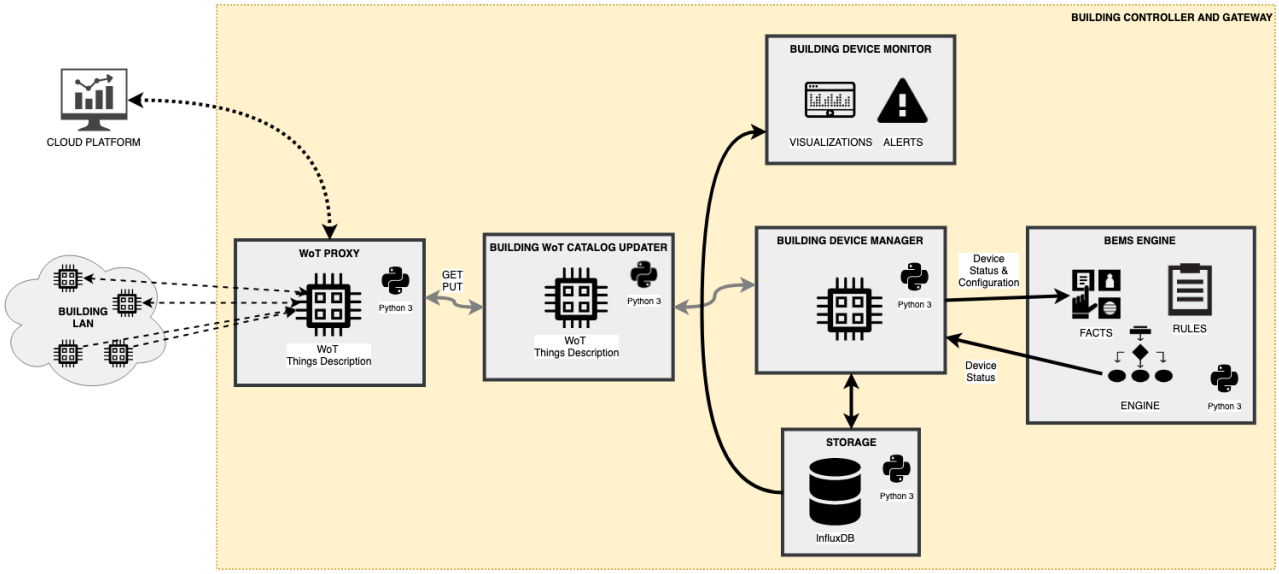


Fig. 10. Software components of the Building Controller and Gateway

WoT Proxy

This module implements the WoT specification in Python 3. It acts as a proxy for all the *things* of the building (i.e., smart fan-coils, Heat Pump, MIMO, and auxiliary devices). This component deploys several services to enable interaction between the devices attached to the private local network and other clients.

The proxy exposed the catalogue of the *WoT Thing Descriptions* of the building on the port **9191**. The document corresponding to the catalogue shows a list of pairs with a unique identification and the corresponding URL path to access the corresponding *WoT Thing Description.* For instance,

```
{
"urn:org:fundacionctic:thing:fancoil:201:proxy": "/thing-proxy-fancoil-201-f23af5b0-ac0f",
"urn:org:fundacionctic:thing:fancoil:202:proxy": "/thing-proxy-fancoil-202-f3ae202e-214d",
…
}
```

So, for instance, dereferencing the URL `http://192.168.1.100:9191/thing-proxy-fancoil-201-f23af5b0-ac0f` clients may retrieve the complete *Thing Description* of a specific fan-coil, including descriptive metadata with all its properties, security measures and methods to access the information.

Building Device Manager

This set of Python 3 scripts gets information from the building devices through the WoT Catalogue and the Things Descriptions provided by the proxy. It also persists and retrieves information to/from the local database.

The BEMS engine uses this component to get and update values related to the statuses of devices, and the configuration.

Building WoT Catalogue Updater

This module, written in Python 3, updates the information from the WoT Catalogue periodically.

BEMS Engine

This component, implemented in Python 3 and PyKnow[7], is the rule engine that manages the different actuations on BEMS devices. This module includes a table of static rules about the operation of the BEMS. The engine uses the static table of rules in combination with instantaneous facts (i.e., status of devices, configurations). These *facts* are retrieved from the database, through the Device Manager.

Storage

All the information will be periodically stored in a local InfluxDB[8] database. InfluxDB is a time series database designed to handle high write and query loads, so the system will store the devices data along with specific timestamps. This enables historical traceability of all the devices.

Building Device Monitor

This component implements a Grafana-based[9] Web application to monitor the relevant values of the BEMS. The configuration of this component is variable, and it depends on the stakeholders' demand.

[7] https://github.com/buguroo/pyknow
[8] https://docs.influxdata.com/influxdb
[9] https://grafana.com

4. FAN-COILS

Fan-Coils are appliances based on STILLE's commercial solution, it enables smart functions though either a NB-IoT or the WiFi module. Although section does not include the specifications of the hardware of Fan-Coils (it is defined in the WP6. Components for heat generation, emission and storage), this section includes information about the communications modules that enables interoperability between fan-coils and the BEMS.

There are three types of fan coils: **Smart Fan-Coils** for heating/cooling/air handling in the main rooms of the building; **DHW Fan-Coils** for providing Domestic Hot Water (DHW); and **Smart Radiators** for heating in the small rooms of the building. DHW Fan-Coils are only installed in the apartment bathrooms. The Smart fan-coils provide the BEMS with information about environmental and internal variables, including: Air temperature (inlet/outlet); Air Relative Humidity; Water flow rate; electric power consumption; and water temperature (inlet/outlet). Additionally, DHW fan-coils measure the domestic hot water flow rate.

As shown in the following figure, fan-coils interoperate with the BEMS by:

- Sharing information: **smart fan-coils** send information (i.e., internal parameters, sensors, status) to the **Building Controller and Gateway,** and receive information and commands (i.e., switch on/off, set-point temperature) from it.
- Getting power: **smart fan-coils** receive constant input electric energy from the **MIMO**.
- Sharing thermal energy: **smart fan-coils** receive a water flow from the boiler room.

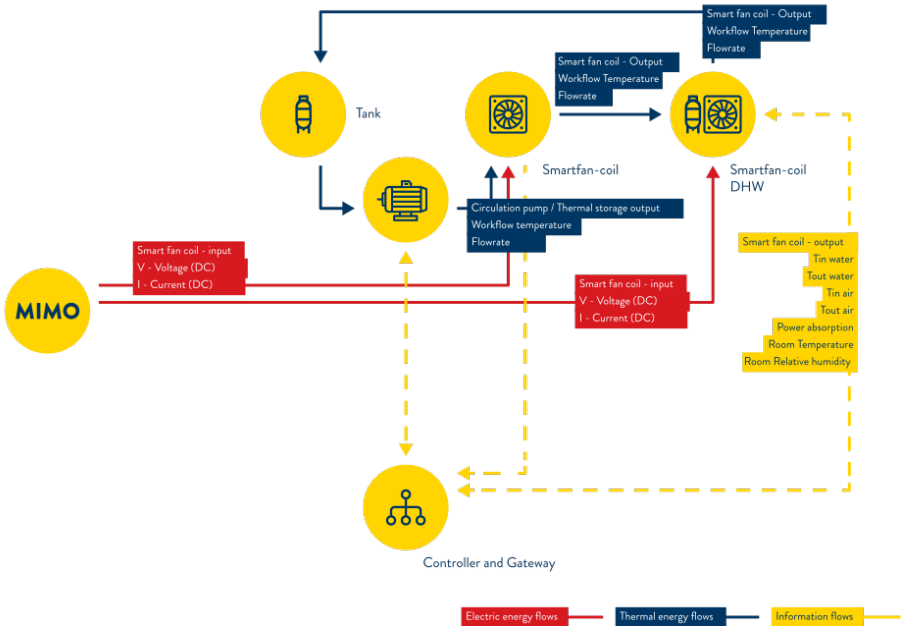


Fig. 11. Interaction of the BEMS with Smart Fan-Coils

Fan-coils receive control information from the BEMS through their fan-coil gateways, implementing basic control logic to adjust their operation.

4.1. ON-BOARD CONTROL SYSTEM

Smart fan-coils, Smart radiators and DHW Smart fan-coils will have an on-board control system that will measure environmental and internal variables useful to achieve BEMS requests.

Fan-coils receive control information from the BEMS to adjust their operations using an interface module (see Communication Interface and Sensors). The On-Board control system is based on an **ATmega328P**[10] processor that controls the basic operations of the appliance (i.e., ON/OFF, critical situation, season change, and fan speed).

Although fan-coil gateways implement three different configurations for each type of fan-coil, including different sets of sensors and control logic, the interface with the fan-coil control PCB will be the same. The interface with the on-board system is composed of five in/out digital pins:

In/Out	Operation
in	turns system 'on' (1) or 'off' (0)
in	season change 'Summer' (1) or 'Winter' (0)
out	compressor 'on' (1) or 'off' (0)
out	Generic alarm 'on' (1) or 'off' (0)

ATmega328P (Fan-Coil Control)

The fan-coils controller is implemented using an Atmel® picoPower® ATmega328/P, a low-power CMOS 8-bit microcontroller based on the RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328/P achieves throughputs close to 1MIPS per MHz. This empowers systems designed to optimize the device for power consumption versus processing speed.

Fig. 12. ATmega328P

[10] http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

The main features of this component are:

Throughput	20 MIPS @ 20MHz
Programmable Memory	32KBytes Flash
I/O	23 Programmable I/O Lines
Operating voltage	1.8 to 5.5V
Interfaces	(2) SPI; Serial USART; I2C;
Consumption	~0.2mA

4.2. COMMUNICATION INTERFACE AND SENSORS

The on-board control system is connected to an interface component, controlled by a **Pycom GPy**, to support both WiFi and NB-IoT communications.

Smart Fan-Coil implements a running Web of Things *servient* implemented in MicroPython that allows to operate with the system. Through these web services, fan-coils receive commands from the BEMS (i.e., switch on, temperature set, hourly set point, desired fan speed, and season set up).

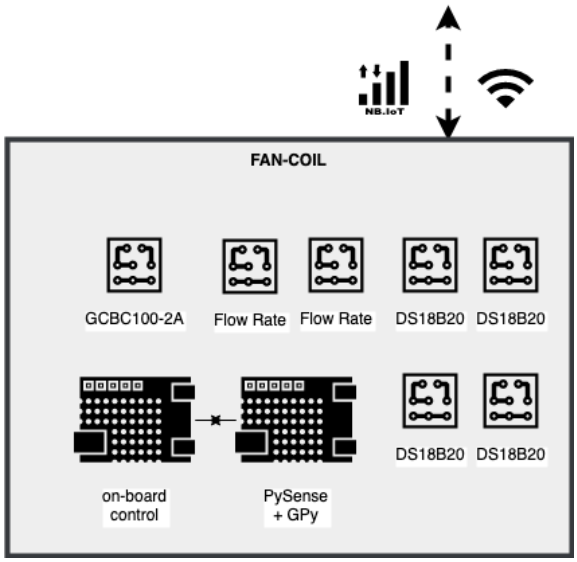


Fig. 13. Fan-coil sensors and interface components

Each type of fan-coil is based on the same control and communications module (Pycom GPy), but with specific configuration of components.

Smart Fan-Coil Configuration

Smart fan-coils will include the following components:

(×1) Pycom GPy	Wi-Fi and NB-IoT module.
(×1) Pyboard Pysense	Shield with sensors (including accelerometer and air relative humidity)

(×1) GCBC100-2A	Power consumption.
(×2) DS18B20	Inlet/outlet water temperatures.
(×2) DS18B20	Inlet/outlet air temperatures.
(×1) G1/*	Water flow sensor.

Smart Radiator Configuration

Smart radiators will include the following components:

(×1) Pycom GPy	Wi-Fi and NB-IoT module.
(×1) Pyboard Pysense	Shield with sensors (including accelerometer and air relative humidity)
(×1) GCBC100-2A	Power consumption.
(×2) DS18B20	Inlet/outlet water temperatures.
(×1) DS18B20	Inlet air temperature.
(×1) G1/*	Water flow sensor.

DHW Fan-Coil Configuration

Smart radiators will include the following components:

(×1) Pycom GPy	Wi-Fi and NB-IoT module.
(×1) Pyboard Pysense	Shield with sensors (including accelerometer and air relative humidity)
(×1) GCBC100-2A	Power consumption.
(×2) DS18B20	Inlet/outlet water temperatures.
(×1) G1/*	Water flow sensor.

Pycom GPy (WiFi and NB-IoT Support)

The Pycom GPy[11] is a microcontroller programmed in MicroPython language, so this implies a rapid development and deployment. It has a variety of versions with multiple communication protocols as WiFi, Bluetooth, and cellular LTE CAT M1/NB1.

Another strong point of this microcontroller is one of its shields. This implementation uses Pysense, as it integrates an accelerometer, temperature, humidity and luminosity sensors, making it a good option to get rid of the cables of separated sensors.

[11] https://pycom.io/product/gpy/

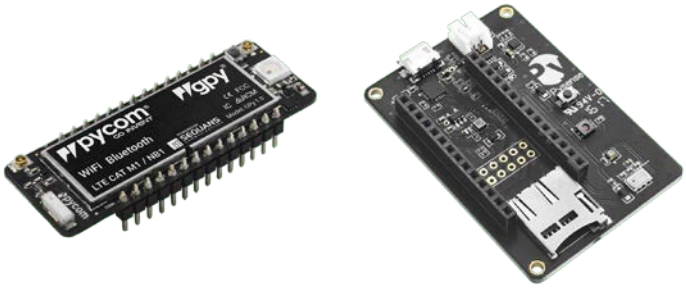


Fig. 14. Pycom GPy and Pysense

Specifications of Pycom GPy:

Processor	ESP32 dual core
RAM	4MB + 520KBytes
Connectivity	WiFi / Bluetooth / Serial
Internal storage	8Mbytes
GPIO ports	28
Consumption	~100mah

The unitary cost of the GPy is €45.00 EUR and the Pysense, €24.00 EUR, approximately.

Air/water temperature sensor: DS18B20

Every Smart Fan-Coil needs to measure water temperature. The DS18B20[12] is a digital thermometer that provides 9-12 bit Celsius temperature measurements. The DS18B20 communicates over a 1-Wire bus that requires only one data line for communication with the central microprocessor. It includes alarm functions with non-volatile user-programmable upper and lower trigger points.

An interesting feature of the DS18B20 is that it has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, the same microprocessor may control several DS18B20s deployed on the same appliance.

The DS18B20 has the following specifications:

Temperature precision	±0.5º Celsius
Range of measurement	-55°to +125°Celsius
Consumption	~1mah
Voltage	3.0V to 5.5V
Communication	Digital

[12] https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

Fig. 15. DS18B20 temperature sensor

The approximate unitary cost of this sensor is €4.00 (EUR).

Water Flow Sensor: G1/*

Fan-Coils need to measure the water flow rate, circulating within them. Water flow sensors consist of a plastic valve body, a water rotor, and a hall-effect sensor. Rotor speed is determined by the flow rate and the hall-effect sensor (transducer that varies its output voltage in response to a magnetic field) outputs the corresponding pulse signal with the flux rate measurement. The G1/* (the concrete model depends on the pipe diameter) will be included in fan-coils.

The G1/* has the following specifications:

Temperature precision	±0.5º Celsius
Range of measurement	1~30 litre/min
Liquid temperature	<120°C
Operating pressure	<1.75MPa
Load Capacity	<15mA
Voltage	5V to 24V
Communication	Digital

Fig. 16. G1/2 Water Flow Sensor

The approximate unitary cost of this sensor is €15.00 (EUR).

DC Power Consumption: GCBC100-2A

The system collects and monitor information related to the continuous power consumption of every controller and board. Thus the BEMS may monitor and adjust the setup with the feedback. The sensor used to measure the power consumption is the GCBC100-2A[13]. It is a PCB mount current sensor for use on the electronic measurement of DC, AC, or pulsed currents with no contact. High-precision detection is achieved with low resistance and low heat generation, supporting the operating voltage and current of the appliances to be measured.

The GCBC100-2A has the following specifications:

Current measuring range	±110A
Operating supply voltage	5V
Sensitivity	20 mV/A

Fig. 17. GCBC100-2A

The approximate cost of the GCBC100-2A is €10.00 (EUR) per unit.

4.3. INSTALLATION AND PINOUT

Interactions with fan-coils are performed and controlled by the local interface implemented on the GPy. This component is linked to the on-board control board through serial port.

After the prototyping phase, the device is mounted on a specific printed circuit board (PCB), created *ad hoc* with the selected configuration of sensors (see Fig. 18).

Fig. 18. Fan-coil interface PCB

[13] https://www4.alps.co.jp/densokunou/productLineDetails/?productNo=GCBC100-2A

The GPy microcontroller is mounted on a PCB that enables external connections to all their pins through screw connectors. The complete pin-out configuration is shown in Fig. 19.

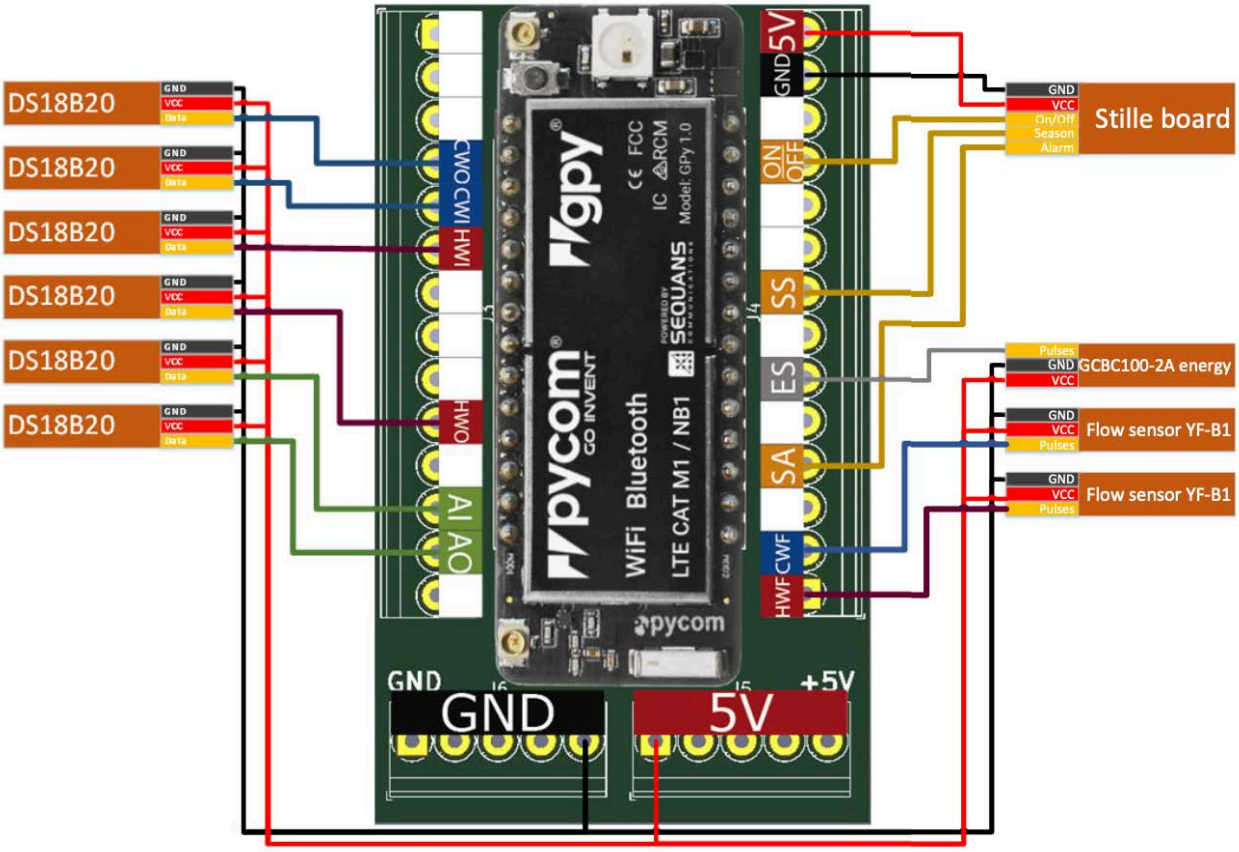


Fig. 19. Pinout of the fan-coil interface board

The microcontroller will be connected to the fan-coil on-board control using the following pins:

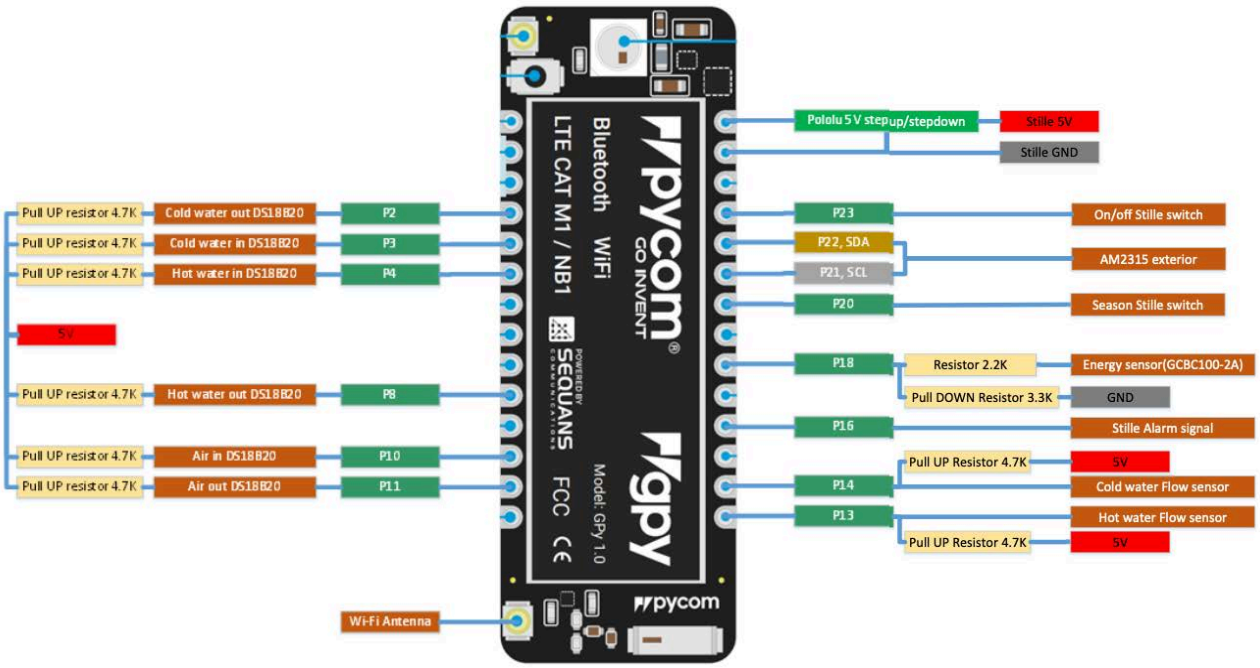


Fig. 20. Detailed pinout of the fan-coil interface board

The final implementation includes stickers for the easy identification of pins and facilitate the subsequent integration in the fan-coil units.

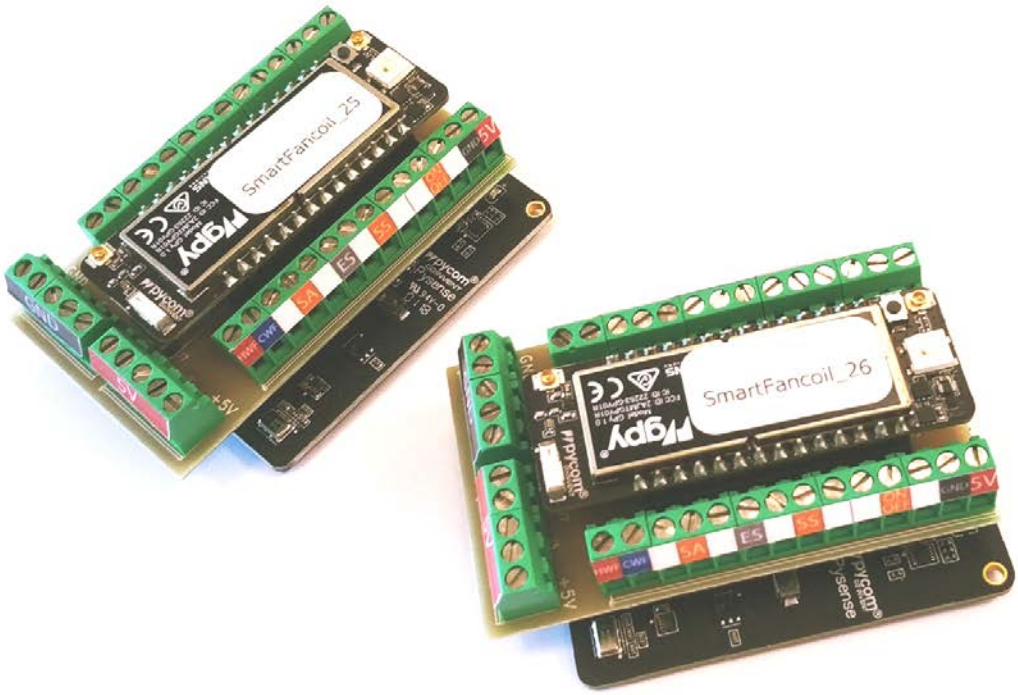


Fig. 21. Final fan-coil interface boards

As shown in Fig. 21, the fan-coil interfaces are labelled with their internal identifier. This reference, along with the place of installation (i.e., building, floor, room) will enable the subsequent monitoring and visualization of all devices and their setup.

4.4. DEVICE CONTROL

Fan-coils include an interface that implements a reduced version of the W3C WoT standard. This interface is implemented on the GPy controller in *Micro Python*, exposing the fan-coil resources using the HTTP protocol and JSON format. This makes the interface compatible with the rest of the BEMS components.

This software manages all the sensors and actuators installed in the fan-coils, sending and gathering values, processing data, making conversions and exposing them to the devices connected to the internal LAN. The interface identifies and publishes the descriptions of the fan-coil *things* on a private URL.

Fan-coils have a fixed range of IP addresses reserved in the wireless LAN:

192.168.1.111/192.168.1.211

For instance, a fan-coil may have allocated the IP 192.168.1.111. After deferring the URL (port 80), the user (i.e, either a machine or a person through a Web browser) retrieves a description of the *thing*, in JSON format, exposing the list of list of sensors and actuators within that fan-coil:

```
{
  "id": "46a1680c-a668-444d-8b6f-6c7f615d312b",
  "actions": {},
  "name": "Fan_coil_01_B_01",
  "description": "Fan coil 1 in the floor 1, apartment B",
  "properties": {
    "DHW_Normal_temp": {
      "forms": [
        {
          "op": "readproperty",
          "contentType": "application/json",
          "href": "https:// 192.168.1.111/Fan_coil_01_B_01/DHW_Normal_temp"
        },
        {
          "op": "readproperty",
          "contentType": "application/json",
          "href": "https:// 192.168.1.111/Fan_coil_01_B_01/Air_RH"
        }
      ],
      "writable": true,
      "description": ""
    },
…
```

Thus, the a fan-coil identified as *Fan_coil_01_B_01* (e.g., fan-coil located in floor 1, apartment B, with ID=01) is identified with a URI that follows the following pattern:

http://HOST/Fan_Coil_[Floor]_[Appartment]_[FanCoil_ID]

As seen in the previous code, the fan-coil exposes the internal properties (sensor measurements, statuses, etc.), as well as its potential actions (switch on/off, set point temperature changes, etc.).

For instance, the measurement of the relative humidity of this fan-coil can be retrieved using the HTTP GET call to the resource: *https://192.168.1.111/Fan_coil_01_B_01/Air_RH*. It would return a JSON document with a decimal number, such as `{ "value": 45.5 }`.

4.5. IMPLEMENTATION OF CONTROL LOGIC

Fan-coil gateways implement a basic logic to control the fan-coils through the on-board controller. Depending on the type of fan-coil (i.e., smart fan-coil, DHW fan-coil and smart radiator) the GPy controller will implement different versions of the firmware to control the fan-coils accordingly to their expected operations.

Basically, each type of fan-coil gateway will implement basic rules to activate/deactivate the fan-coil, depending on parameters such as set-point temperature, current room temperature, water temperature, and summer/winter mode.

Control of Smart Fan-Coils

The control of the Smart Fan-Coils (switch on/off) depends on the set-point temperature specified by the system or the user, the room temperature (provided by the inlet temperature sensor) and the mode of operation (cooling in summer; warming in winter). The complete set of rules of the system is shown in the following workflow.

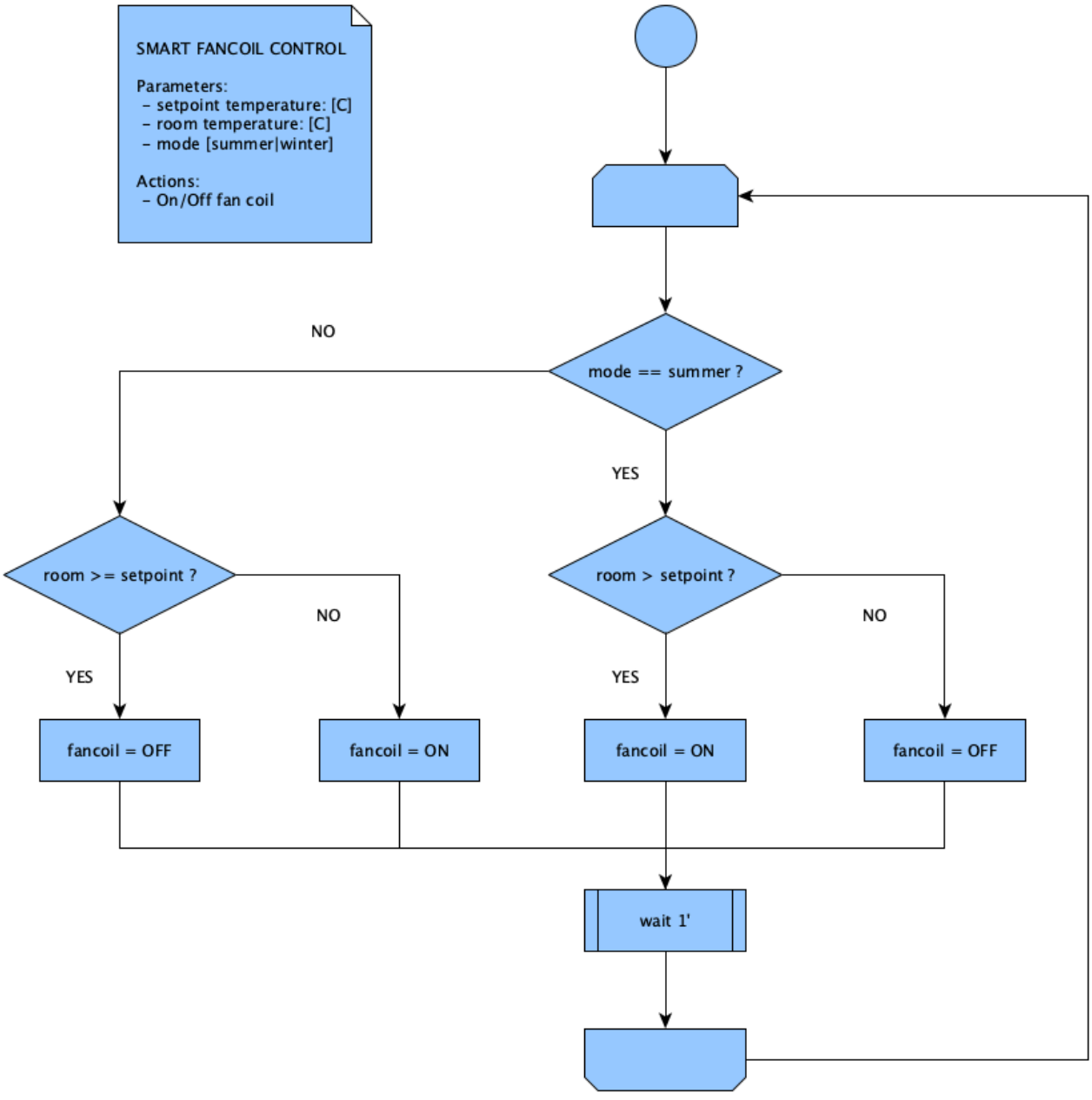


Fig. 22. Smart fan-coil control workflow

Control of Smart Radiators

The control of the Smart Radiators (switch on/off) depends on the set-point temperature specified by the system or the user and the room temperature (provided by the inlet temperature sensor). This type of devices does not support cooling operation as the smart fan-coils. The complete set of rules of the system is shown in the following workflow.

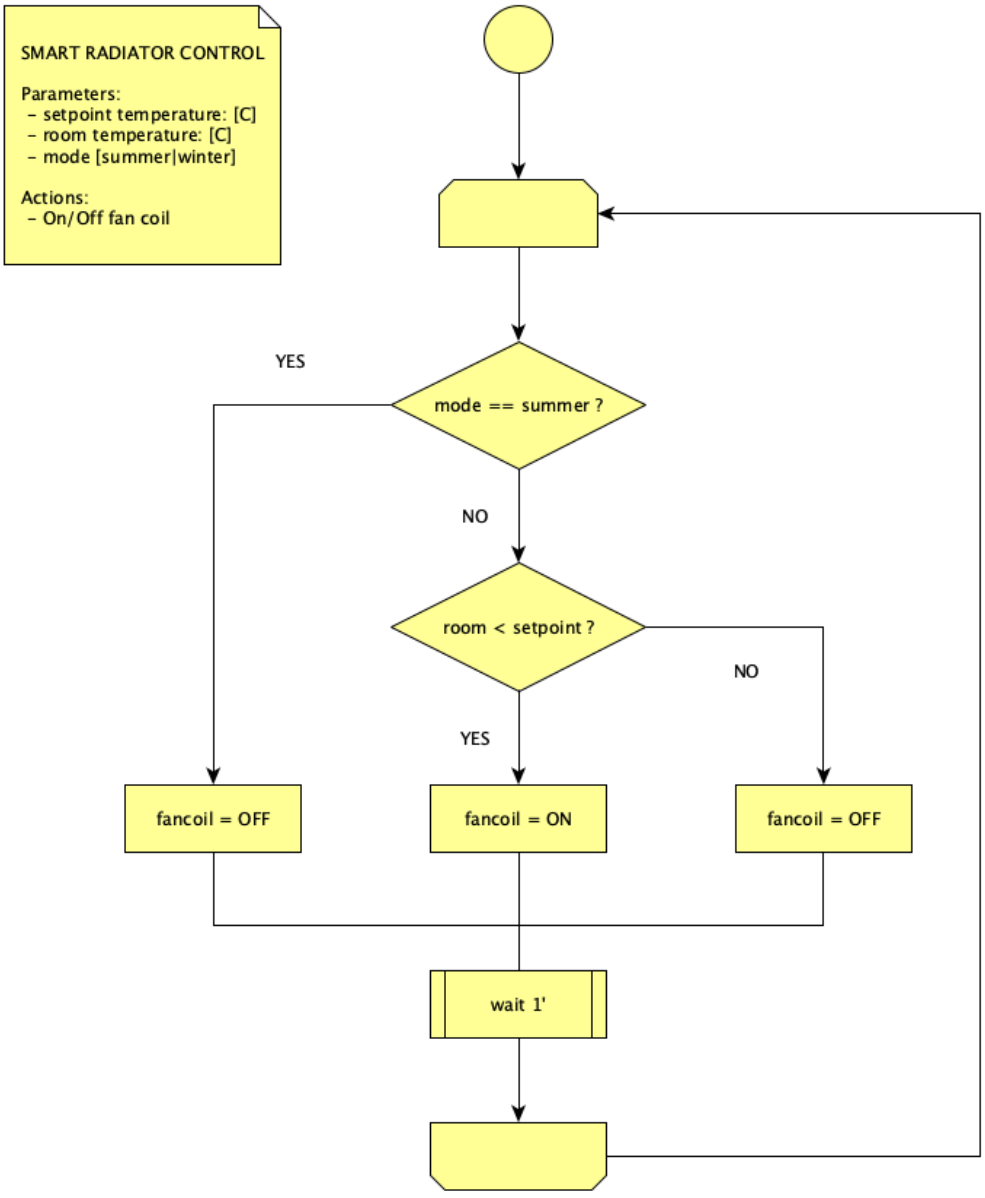


Fig. 23. Smart fan-coil control workflow

Control of DHW Fan-Coils

The control of the DHW Fan-Coils (switch on/off) depends on the set-point water temperature specified by the system, the current water temperature (provided by the inlet temperature sensor). There is a special operation mode to prevent legionella in the water tanks. When this mode is activated (boost mode on), the fan-coil will be switched on and the normal workflow will be bypassed. The complete set of rules of the system is shown in the following workflow.

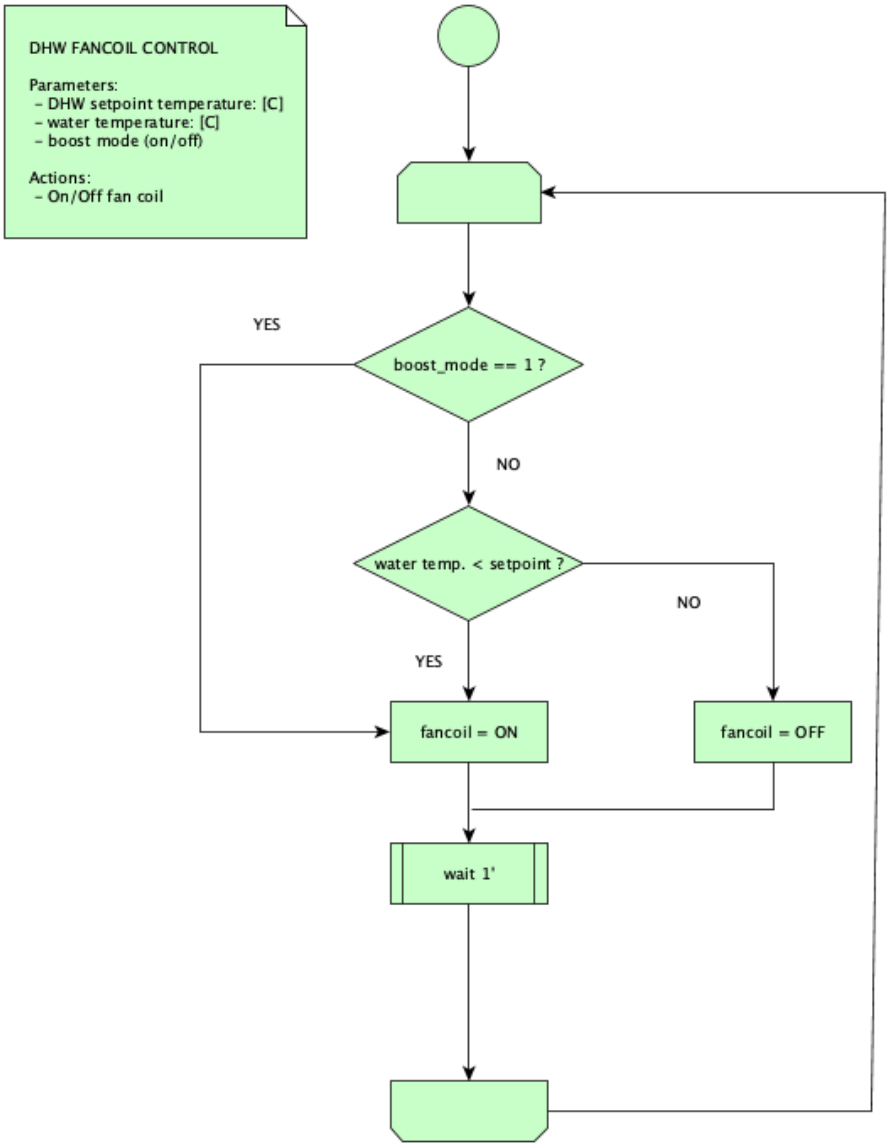


Fig. 24. DHW fan-coil control workflow

5. HEAT PUMP GATEWAY

The Heat Pump interoperates with the BEMS by:

- Sharing information: the Heat Pump implements internally a Modbus/TCP server that enables the operation of the device (i.e., operating modes, set point temperatures, etc.), as well as getting the internal values.
- Getting electric energy: the Heat Pump is powered by the MIMO.
- Sharing thermal energy: the Heat Pump is linked to a water storage. Thermal energy flows from the **PCM Storage** to the **DHW Tank is** managed by the **HP Circulation Pump**.

The Heat Pump is physically connected to the **Heat Pump Gateway**. This gateway performs the role of communication interface with the rest of the BEMS. The Heat Pump implements a Modbus/TCP server that enables the operation from an external device. The Heat Pump and its Gateway are directly connected through Ethernet, creating an ***ad hoc* local network** exclusively for the heat pump operation.

The **Heat Pump Gateway** is in charge of exposing the commands and variables of the heat pump in form of WoT *thing descriptions*.

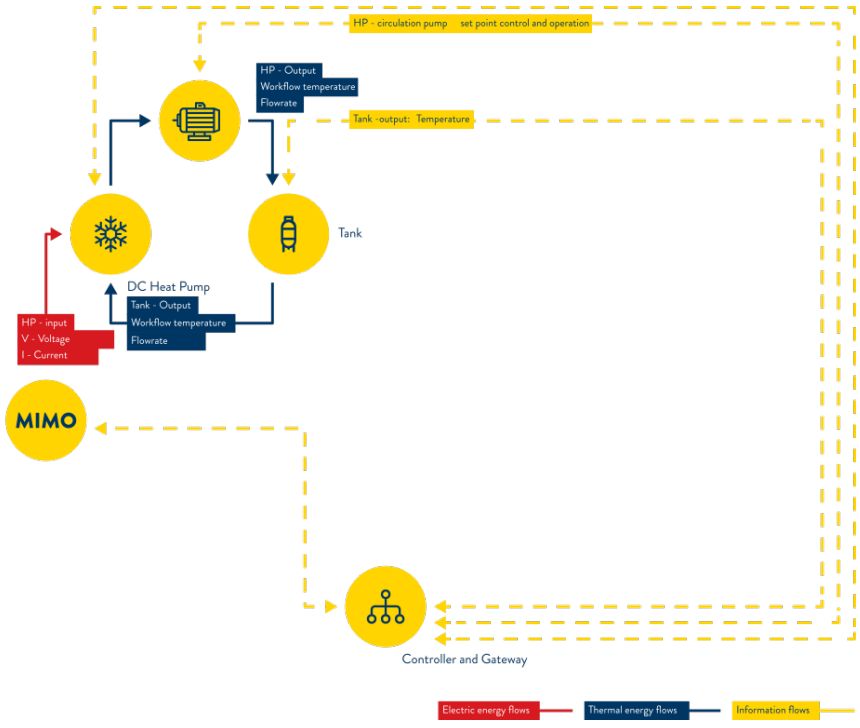


Fig. 25. Interaction of the BEMS with the Heat Pump

5.1. IMPLEMENTATION AND CONFIGURATION

The **Heat Pump Gateway** is implemented on an UP Board, acting as a gateway between the heat pump and the external clients (i.e., cloud platform and building control logic), as shown in the following diagram. The gateway deploys an implementation of the WoT (Web of Things) W3C standard, in Python 3. The device runs on Ubuntu Server as operating system.

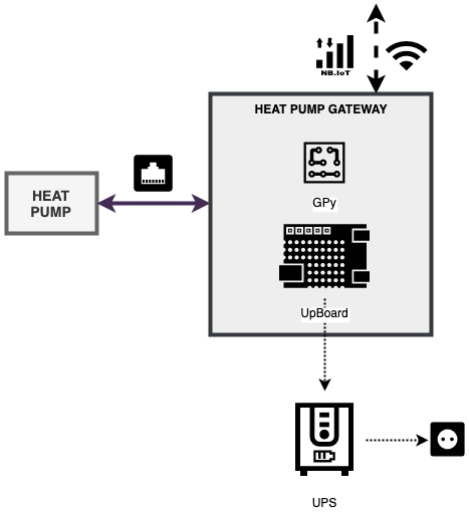


Fig. 26. Heat Pump and Heat Pump Gateway components

The Ethernet port of the **Heat Pump Gateway** must be directly connected to the Heat Pump using a Cat-5 (or superior) cross over Ethernet cable.

Fig. 27. Heat Pump Gateway installation details

The communication with the **Building Controller and Gateway,** performed through the WiFi LAN, needs to be pre-configured. Using this configuration, the Heat Pump Gateway implements a proxy between the local devices and their sensors and actuators, and the internal network.

The *ad hoc* internal network allowing the connection between the **Heat Pump** and its respective gateway has the following set up:

Device	Reserved IP Address	Fixed?
Heat Pump	192.168.2.1	Yes
Heat Pump Gateway	192.168.2.2	Yes

5.2. DEVICE CONTROL

The local gateway for the Heat Pump identifies and exposes all the parameters and operations of the Heat Pump. It also translates the Modbus operations to the WoT approach using the HTTP protocol and JSON formats. The *servient* that contains and serves the resources may be accessed on:

http://192.168.1.106:9090[14]

Dereferencing this URL, the inventory of the things exposed by this gateway is shown. In this case, it only contains the heat pump. This catalogue contains specific URIs to identify that thing, only accessible from the local network. They follow the following pattern:

http://192.168.1.106:9090/HEART_HP_gateway-xxxx-xxxx-xxxx-xxxx-xxxx

Where xxxx-xxxx-xxxx-xxxx-xxxx is an auto generated UUID code to uniquely identify each *thing*. Dereferencing those URIs, *thing* descriptions (i.e., descriptive metadata about its features and characteristics) are found. These descriptions also show all the available interactions with the *things*.

The following code shows a partial example of description of a *thing* that exposes one of the Heat pump parameters:

```
{
  "id": "urn:org:fundacionctic:thing:HEART_Heat_Pump",
  "name": "HEART Heat pump monitor Thing",
  "properties": {
    "Outdoor_temperature": {
      "forms": [
        {
          "href": "http://192.168.1.106:9292/heart-heat-pump-monitor-thing-5c2e5e22-039c-
f002-56c7-a9f6d63485db/property/outdoor-temperature",
          "op": "readproperty",
          "contentType": "application/json"
        },
        {
          "href": "http://192.168.1.106:9292/heart-heat-pump-monitor-thing-5c2e5e22-039c-
f002-56c7-a9f6d63485db/property/outdoor-temperature/subscription",
          "op": "observeproperty",
```

[14] If there are several, they have the 192.168.1.106 — 192.168.1.110 range reserved for them

```
          "contentType": "application/json"
        }
      ],
      "observable": true,
      "type": "string"
    },
…
```

Automatic scripts may get the specific URLs to gather data from the different resources (read operations), or configure the device (write parameters). For example, in the previous example, there is a property *Outdoor_temperature* that could be read fetching the resource using the GET method:
http://192.168.1.106:9292/heart-heat-pump-monitor-thing-5c2e5e22-039c-f002-56c7-a9f6d63485db/property/outdoor-temperature

The petition would trigger a process in the "thing" and return a 200 HTTP CODE (OK) with the following body in JSON format: `{ "value": 23 }`

It also can be written using the PUT method to the same URL. This petition should contain in the body a JSON with the desired value. For instance: `{ "value": 120 }`

It would instantly update the value in the Modbus slave, but it might take a while for it to be updated on the Heat pump gateway. Also, the internal HTTP server manages the standard error codes in case of inexistent resources (i.e., 404) or internal failures (i.e., 500).

Table of WoT properties and Modbus Registries:

Mode	WoT Property Name	Modbus Register	Format
read	Outdoor_temperature	10	INT16
read	DHW_temperature	11	INT16
read	Heat_outlet_temp	12	INT16
read	Heat_inlet_temp	13	INT16
read	Buffer_storage_temp	14	INT16
read	ES_inlet_temp	15	INT16
read	ES_outlet_temp	16	INT16
read	Heating_circuit_pump	23	INT16
read	Buffer_charging_pump	24	INT16
read	Compressor	25	INT16
read	Error	26	INT16
read	four_way_valve_Air	27	INT16
read	COP	30	INT16
read	Operating_hours_in_DHW_mode	42-43	UINT32
read	Operating_hours_in_Heat_mode	44-45	UINT32
read	Calorimeter_Heating	60-61	UINT32
read	Electric_meter_Heating	62-63	UINT32
read	Calorimeter_DHW	64-65	UINT32
read	Electric_meter_DHW	66-67	UINT32
read	Electric_meter_total	68-69	UINT32
read	Electric_meter_capacity	70-71	UINT32
read	Calorimeter_total	72-73	UINT32
read	Calorimeter_capacity	74-75	UINT32
write	Operating_mode	100	UINT16
write	HC_Setpoint_Room_setpoint_temp	102	INT16
write	HC_Setpoint_Heat_inlet_Setpoint_tem	103	UINT16

	p		
write	HI_T_min_Cool	104	INT16
write	DHW_Normal_temp	105	INT16
write	DHW_Minimum_temp	106	INT16
write	PV_request	117	UINT16
write	Power_input_specification	125	UINT16
write	Clear_error_reset	128	UINT16
write	Outdoor_temperature_value	129	INT16
write	Outdoor_temperature_Active	130	UINT16
write	Buffer_temperature_value	131	INT16
write	Buffer_temperature_Active	132	UINT16
write	DHW_temp_value	133	INT16
write	DHW_temp_Active	134	UINT16

6. MIMO GATEWAY

The MIMO interoperates with the BEMS basically distributing the electric energy from/to the building devices.

Likewise the Heat Pump, the MIMO is physically connected to the **MIMO Gateway**. This gateway performs the role of communication interface with the rest of the BEMS. The MIMO implements a Modbus/TCP server that enables the operation from an external device. The MIMO and its Gateway are directly connected through Ethernet, creating an ***ad hoc* local network** exclusively for the heat pump operation.

The **MIMO Gateway** is in charge of exposing the commands and variables of the MIMO in form of WoT *thing descriptions*.

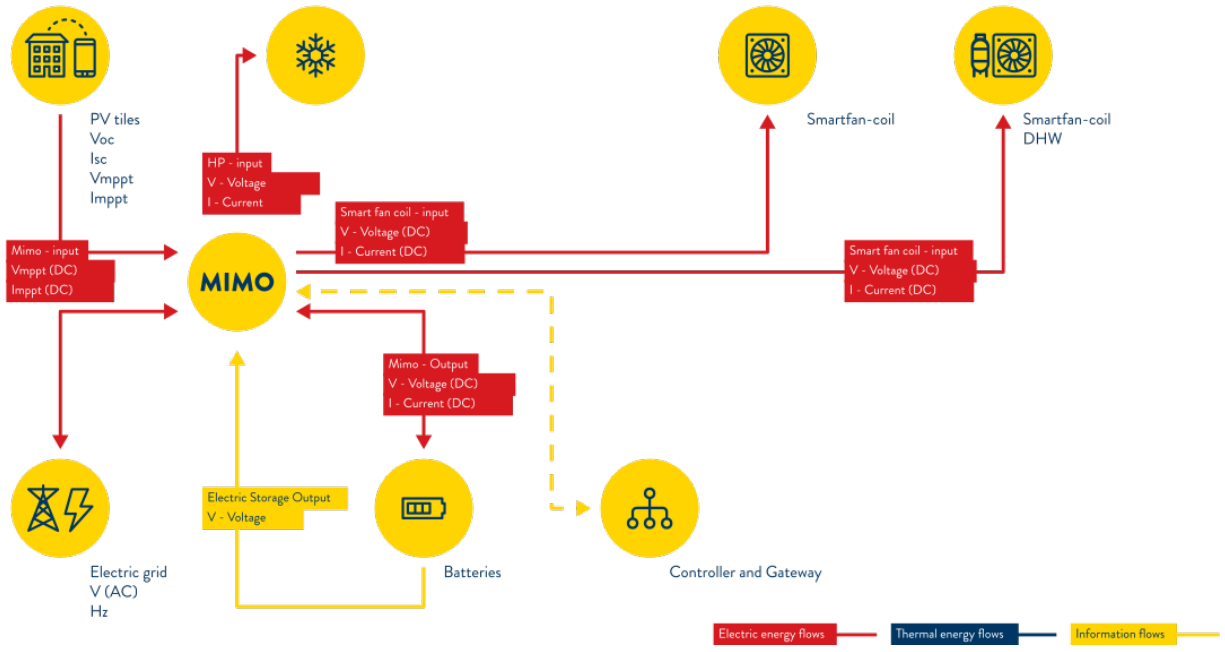


Fig. 28. Interaction of the BEMS with the MIMO

6.1. IMPLEMENTATION AND CONFIGURATION

The **MIMO Gateway** is implemented on an UP Board, acting as a gateway between the MIMO and the external clients (i.e., cloud platform and building control logic). This local gateway deploys an implementation of the WoT (Web of Things) W3C standard, in Python 3. The device runs on Ubuntu Server as operating system.

The following diagram shows the components of the MIMO Gateway.

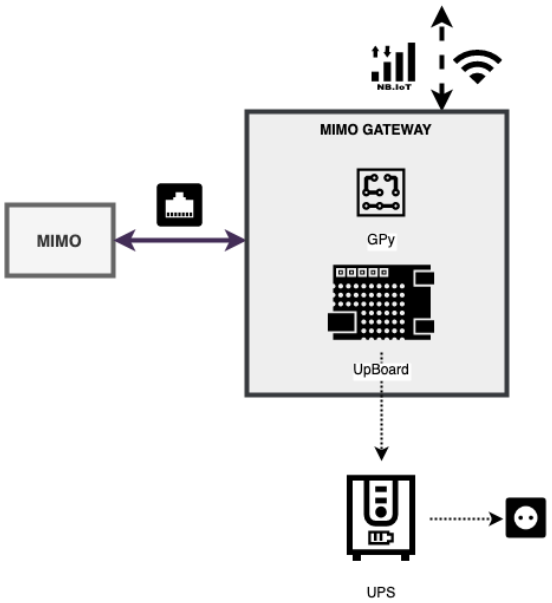


Fig. 29. MIMO and MIMO Gateway components

The Ethernet port of the **MIMO Gateway** must be directly connected to the Heat Pump using a Cat-5 (or superior) cross over Ethernet cable.

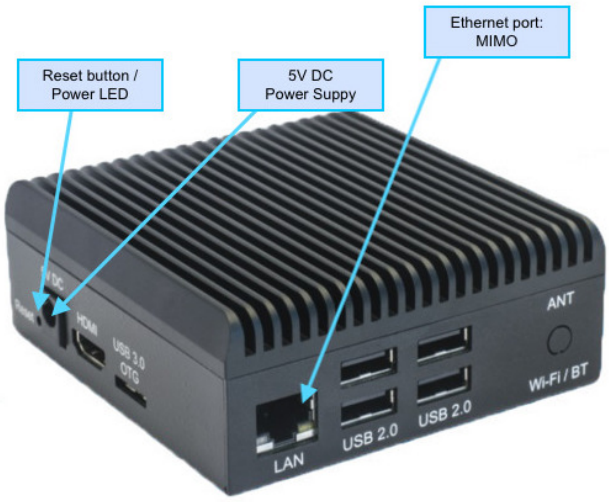


Fig. 30. MIMO Gateway installation details

The communication with the **Building Controller and Gateway**, performed through the WiFi LAN, needs to be pre-configured. Using the following configuration, the MIMO Gateway implements a proxy between the local devices and their sensors and actuators, and the internal network.

The *ad hoc* internal network allowing the connection between the **MIMO** and its respective gateway has the following set up:

Device	Reserved IP Address	Fixed?
MIMO	192.168.2.1	Yes
MIMO Gateway	192.168.2.2	Yes

6.2. DEVICE CONTROL

The local gateway for the MIMO identifies and exposes all the parameters and operations on the MIMO. It also translates the Modbus operations to the WoT approach using the HTTP protocol and JSON format. The *servient* that contains and serves the resources may be accessed on:

http://192.168.1.101:9090[15]

Dereferencing this URL an inventory of the things exposed by this *servient* is shown. In this case, it only contains one, the MIMO. This catalogue of one thing contains specific URIs to identify them, only accessible from the local network. They follow the following pattern:

http://192.168.1.101:9090/HEART_MIMO_gateway-xxxx-xxxx-xxxx-xxxx-xxxx

Where xxxx-xxxx-xxxx-xxxx-xxxx is an auto generated UUID code to uniquely identify each *thing*. Dereferencing those URIs, *thing* descriptions (i.e., descriptive metadata about its features and characteristics) are found. These descriptions also show all the available interactions with the *things*.

The following code shows a partial example of description of a *thing* that exposes the MIMO parameters:

```
{
  "id": "urn:org:fundacionctic:thing:HEART_MIMO",
  "name": "HEART MIMO monitor Thing",
  "properties": {
    "PV_Current": {
      "forms": [
        {
          "href": "http://192.168.1.101:9292/heart-mimo-monitor-thing-5c2e5e22-039c-f002-
56c7-a9f6d63485db/property/pv-current",
          "op": "readproperty",
          "contentType": "application/json"
        },
        {
          "href": "http://192.168.1.101:9292/heart-mimo-monitor-thing-5c2e5e22-039c-f002-
56c7-a9f6d63485db/property/pv-current/subscription",
          "op": "observeproperty",
          "contentType": "application/json"
        }
      ],
      "observable": true,
      "type": "string"
    },
…
```

[15] If there are several, they have the 192.168.1.101 — 192.168.1.105 range reserved for them

Automatic scripts may get the specific URLs to gather data from the different resources (read operations), or configure the device (write parameters). For example, in the previous example, there is a property *PV_Current* that could be read fetching the resource using the GET method: `http://192.168.1.101:9292/heart-mimo-monitor-thing-5c2e5e22-039c-f002-56c7-a9f6d63485db/property/pv-current`

The petition would trigger a process in the "thing" and return a 200 HTTP CODE (OK) with the following body in JSON format: `{ "value": 5 }`

In the case of writable properties, the client may use the PUT method on the URL. This petition should contain in the body a JSON with the desired value. For instance: `{ "value": 0 }`

It would instantly update the value in the Modbus slave, but it might take a while for it to be updated on the Heat pump gateway. Also, the internal HTTP server manages the standard error codes in case of inexistent resources (i.e., 404) or internal failures (i.e., 500).

Table of WoT properties and Modbus Registries of the MIMO:

Mode	WoT Property Name	Modbus Register	Format
write	Connect_Grid	40001	UINT16
write	Disconnect_Grid	40002	UINT16
write	Connect_Central_Heat_Pump	40003	UINT16
write	Connect_Local_Heat_Pump	40004	UINT16
write	Connect_PV	40005	UINT16
write	Connect_Battery	40006	UINT16
read	Battery_Power_Demand	40007	INT16
read	Grid_Contactor_Status	40008	UINT16
read	PV_Contactor_Status	40009	UINT16
read	Battery_Contactor_Status	40010	UINT16
read	Grid_Protection_Relay_Status	40011	UINT16
read	DC_Link_Status	40012	UINT16
read	Grid_Voltage_L1	40013	UINT16
read	Grid_Voltage_L2	40014	UINT16
read	Grid_Voltage_L3	40015	UINT16
read	Grid_Current_L1	40016	UINT16
read	Grid_Current_L2	40017	UINT16
read	Grid_Current_L3	40018	UINT16
read	Grid_Power	40019	INT16
read	PV_Voltage	40020	UINT16
read	PV_Current	40021	UINT16
read	PV_Input_Power	40022	UINT16
read	CHP_Voltage	40023	UINT16
read	CHP_Current	40024	UINT16
read	CHP_Output_Power	40025	INT16
read	Battery_Port_Voltage	40026	UINT16
read	Battery_Port_Current	40027	INT16
read	Battery_Port_Power	40028	INT16
read	Local_Heat_Pumpo_Volts	40029	UINT16
read	Local_Heat_Pump_Output_Power	40030	INT16
read	DC_Link_Voltage	40031	UINT16
read	Temperature_1	40032	INT16
read	Temperature_2	40033	INT16

read	Temperature_3	40034	INT16
read	Temperature_4	40035	INT16
read	Temperature_5	40036	INT16
read	Temperature_6	40037	INT16
read	Temperature_7	40038	INT16
read	Temperature_8	40039	INT16
read	Battery_Chargue_Demand	40040	UINT16
read	Battery_Master_Status	40041	UINT16
read	Battery_Master_Fault_Bits	40042	UINT16
read	Battery_Highest_Cell_Voltage	40043	UINT16
read	Battery_Lowest_Cell_Voltage	40044	UINT16
read	Battery_Array_Voltage	40045	UINT16
read	Battery_State_Charge	40046	UINT16
read	Battery_Charger_Connected	40047	UINT16
read	Battery_Array_Current	40048	INT16
read	Battery_Minutes_To_Full	40049	UINT16
read	Battery_System_Temp_High	40050	INT16
read	Battery_System_Temp_Low	40051	INT16
read	Battery_Minutes_To_Empty	40052	UINT16
read	Battery_Nodes_Missing	40053	UINT16
read	Grid_Connection_Status	40054	UINT16
read	PV_Connection_Status	40055	UINT16
read	Battery_Connection_Status	40056	UINT16
read	CHP_Connection_Status	40057	UINT16
read	LHP_Connection_Status	40058	UINT16
write	Disconnect_Central_Heat_Pumpdis	40059	UINT16
write	Disconnect_Local_Heat_Pump	40060	UINT16
write	Disconnect_PV	40061	UINT16
write	Disconnect_Battery	40062	UINT16

7. THERMAL ENERGY MANAGEMENT

The control of the thermal energy distribution among the building water storage devices will be controlled by commercial Programmable Logic Controllers (PLCs) that will implement the specific rules of the BEMS. As shown in the following figure, the devices involved in this management of thermal energy are: the PCM Storage (Tank), Circulation Pumps, Three-way-valves, DHW Tank, and Smart fan-coils.

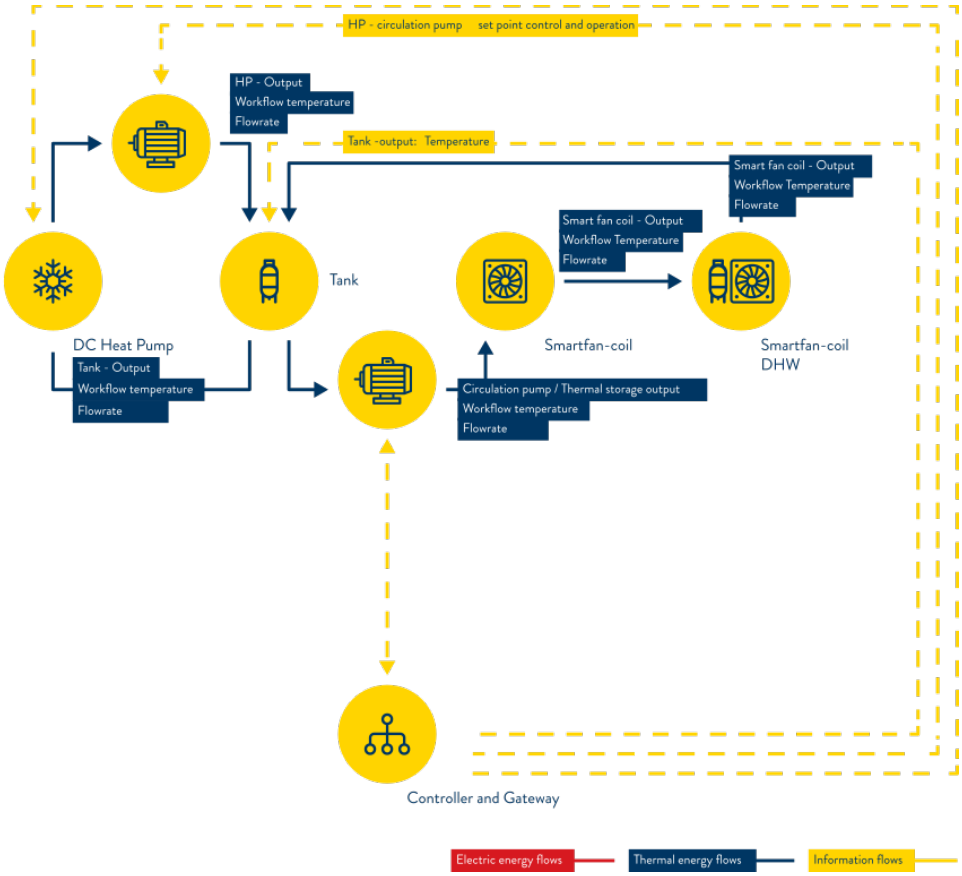
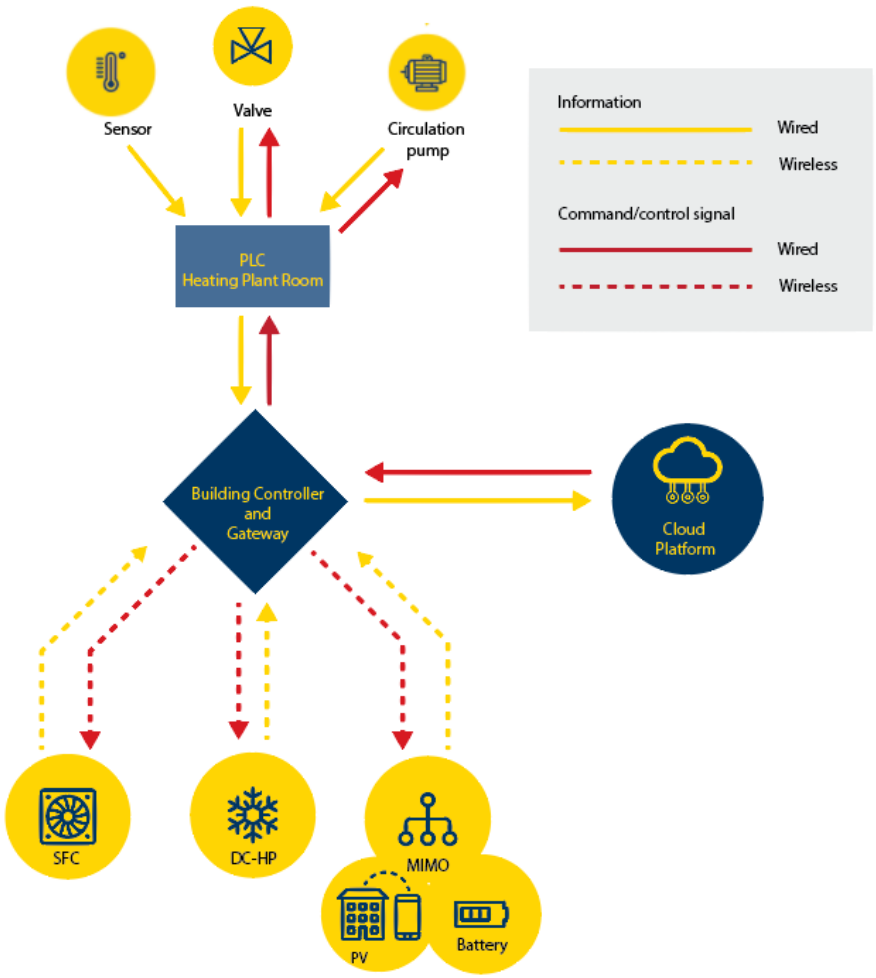


Fig. 31. Thermal energy flow, controlled by the BEMS

The Building Controller and Gateway will interact with the flow valves and circulation pumps in order to monitor the distribution of fluid among the devices. The activation/deactivation commands will depend on predefined rules, configuration and status of the system. Automatic PLCs will guarantee the robust operation of the auxiliary systems (sensors, valves and circulation pumps) placed in the technical room as illustrated in the following figure



More in detail, the selected solution will be based on the Siemens **SIMATIC S7-1200 (CPU 1215C)**[16], a compact automation solution with integrated communication and technology functions. This PLC provides the system with a reliable and flexible solution with extended communication capabilities such as:

- **OPC UA Data Access** as a server enables standardized horizontal and vertical communication as well as compliance with industry-specific standards.
- **Cloud connectivity** that enables data storage and analysis, to guarantee efficiency and predictive maintenance.
- **Secure data transmission** to preserve privacy of data.

[16] https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7215-1AG40-0XB0

Fig. 32. SIMATIC S7-1200 PLC

Main features of the selected PLC:

CPU	CPU 1215 DC/DC/DC
Memory	125 KB work memory / 4 MB Load memory
Networking	2 Industrial Ethernet ports with integrated switch
Input/Output	14 DI /10 DQ and 2 AI/2 AQ integrated
Modularity	Expandable by 1 signal board (SB), 8 signal modules (SM), 3 communication modules (CM)

The control system will include the Communication Board CB 1241[17], supporting RS485 Modbus communications, and the SM 1231 RTD signal module[18] to support analogue input.

[17] https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7241-1CH30-1XB0
[18] https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7231-5PD32-0XB0

Fig. 33. Communication Board CB 1241

Main features of the CB 1241:

Interfaces	Freeport; ASCII; Modbus; Modbus RTU master/slave; USS
Protocols	Freeport; 3964; Modbus RTU master; Modbus RTU slave

Fig. 34. SM 1231 RTD signal module

Main features of the SM 1231 RTD:

Temperature error	(+/-) 25 °C ±0.1%, to 55 °C ±0.2% total measurement range
Number of analogue inputs	4; Resistance thermometer
Max input voltage	±35 V

Temperatures will be measured using seven (7) PT100 2-wire temperature sensors.

Fig. 35. PT100 2-wire temperature sensor

The heat energy in the system will be measured through three heat counter with nominal flow of 6 m^3/h and one of 7-8 m^3/h. The selected device is a Conteca Caleffi 7554. These devices provides the system with direct local reading using a LCD screen, and a centralized bus transmission towards the main PCL through RS-485.

Fig. 36. Conteca Caleffi 7554

Main features of the Caleffi 7554:

Operation voltage	24 V (+10% / -5%) AC
Interface	Bus RS-485
Measurement sensitivity	<0.05 ºC

[i] Puschmann A et al., "Implementing NB-IoT in Software - Experiences Using the srsLTE Library", 2007