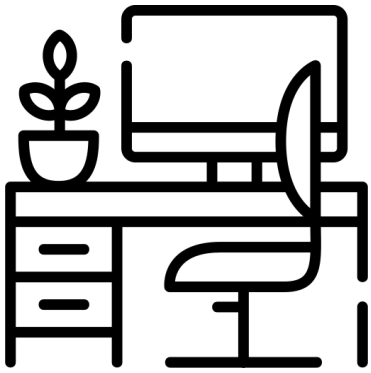
UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila



Autonomous Coworking Space

2022 / 2023

Software Engineering for Autonomous Systems

Professor: *Prof. Davide Di Ruscio*

Project GitHub Repository: https://github.com/GianlucaRea/autonomous-cwspace

Realized by:

Gianluca Rea - gianluca.rea@student.univaq.it - m. 278722

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Requirements

Functional Requirements

1 - Scaling Energy Consumption

Priority: **High**

The system will be able to scale the energy send to each room according to their request.

2 - Room Dynamic Management

Priority: **Medium**

The system will be capable of the identification of new rooms and ignoring existing rooms.

3 - Energy Loss Avoidance

Priority: **Medium**

The system will be able to identify a problem in the energy consumption by the electrical socket avoiding energy loss.

4 - Proactive Scaling Energy Consumption

Priority: **Low**

The system will be able to use a proactive approach instead of a reactive approach by learning the energy consumption of each room

Non-Functional Requirements

- **Scalability**
- **High-Performance DB**
    - The system is supposed to contain a large amount of data, and it should be able to perform aggregation functions and real-time queries on them, so we need a database which is designed for these purposes.

- **Reusability**
- **Cross-platform system**

Component Description

Managed System

Our system will manage a collection of rooms, each room has an advanced energy storage system that will refer to as “battery” for simplicity. The “battery” send to the system this information.

- Level of charging of the battery (batteryLevel)

  Percentage value of the battery level of the room.

- Energy flow entering the battery (batteyInput)

  Quantity of energy entering the battery coming from the grid

- Energy flow exiting the battery (batteryOutput)

  Quantity of energy exiting the battery going out to the room sockets

- Indication flag for the status of the room (status)

  Boolean value which indicates the status of the room

- Room energy demand (energyDemand)

  Generated value which simulate the energy asked by the room by summarizing the request of each socket.

- Number of electrical sockets (sockets)

  Number of sockets in the room.

Besides the classical room, we going to have a room0 that represents the energy grid.

Architecture Diagram

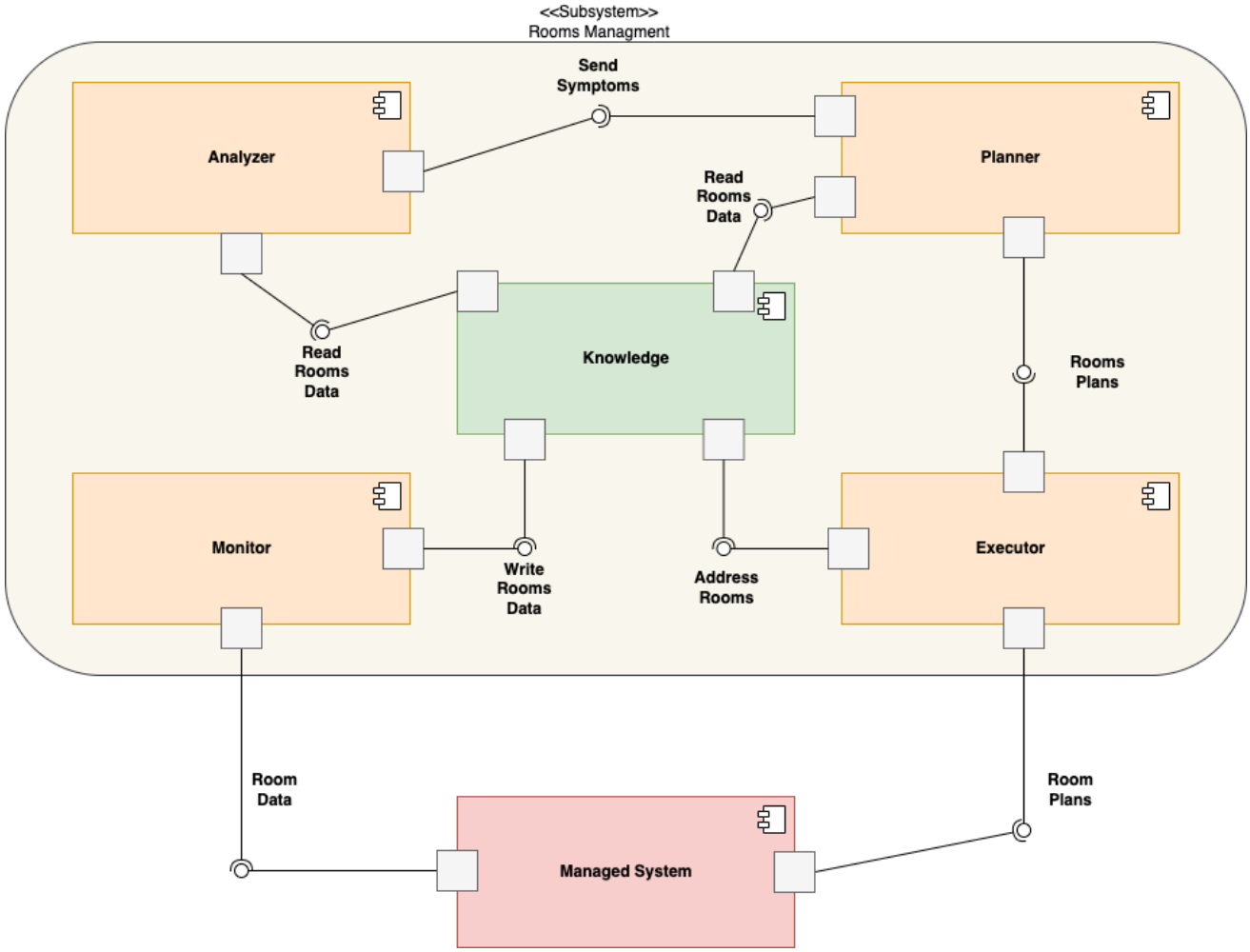


Figure 1: Autonomous Coworking Space - Mape -k Diagram

Monitor

The monitoring consists of the collection of data generated by the sensors inside the office and saving them in the knowledge component

Analyzer

Sensor data collected from the Knowledge component are analyzed on a specific time interval to actively identify and detect environmental symptoms

Planner

Based on the data from the sensors collected from the knowledge component after receiving symptoms from the analysis component the planner has to adapt the values of the variable and sed the plan to the executor component

Knowledge

Through the actuators, this component will actuate the plan elaborated by the planning component

Executor

Through the actuators, this component will actuate the plan elaborated by the planning component

System Architecture

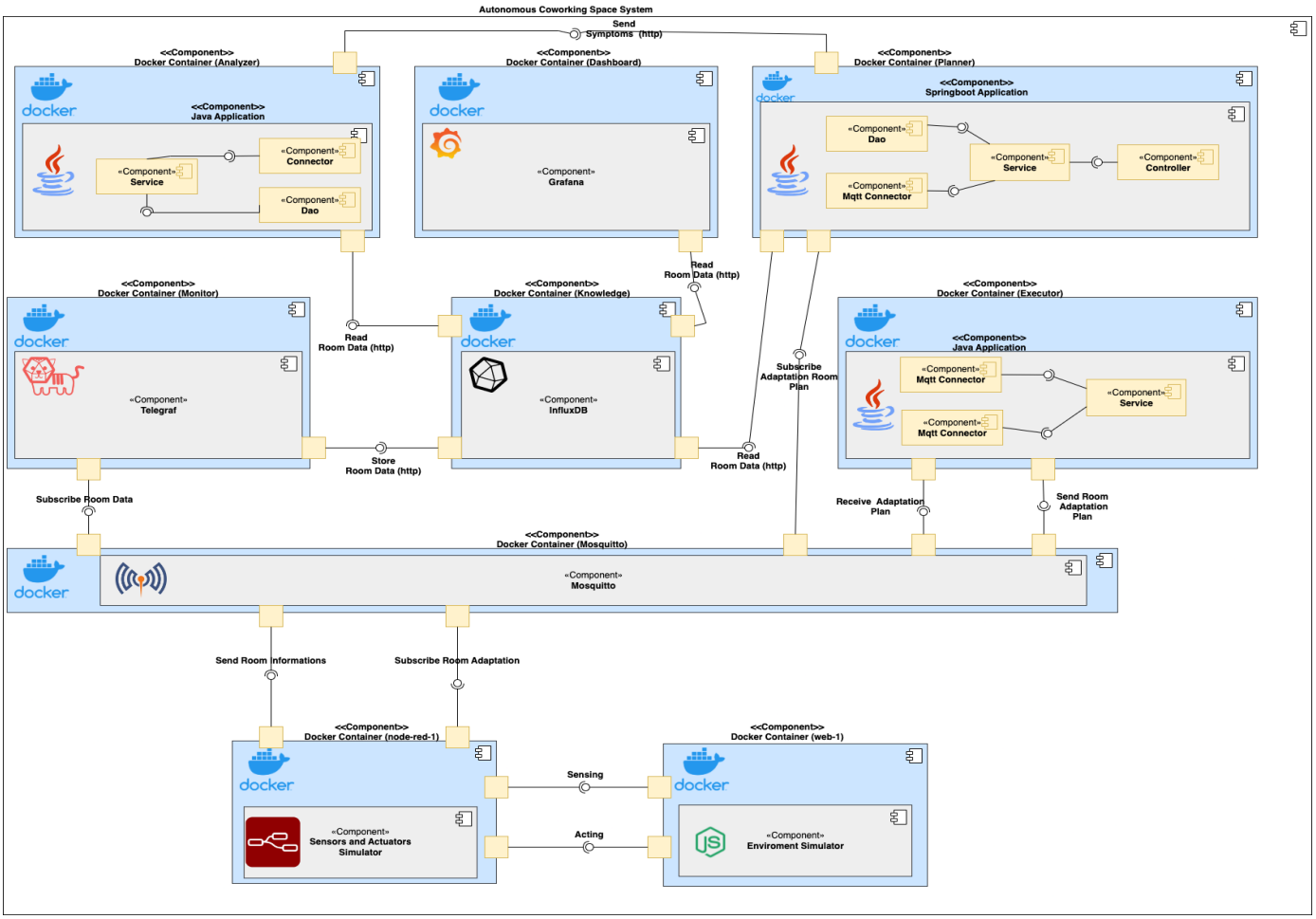


Figure 2: Autonomous Coworking Space - Component Diagram

To form the components of our maps-k loop, we opted for a container system to have the maximum distinction between each component of the loop. The components are deployed on different containers that communicate through the exposure of the involved ports using Docker.

Monitor

The monitor is represented by the docker container which has the Telegraf plugin that captures the mqtt communication sent by the various rooms. It has the duty to save this information inside the knowledge. We have chosen this technology because the tool is developed with the inluxDB database in mind.

Analyzer

The Analyzer component was developed using Java with the implementation of Threads. Using a query on the Knowledge it performs its duty to communicate symptoms to the planner. Based on the symptoms a specific HTTP with a unique endpoint.

Planner

The Planner component was developed using Springboot. Using data symptoms received by the analyzer it makes an adaptation by questioning before the knowledge and then sending the plan to the executor through mqtt.

Knowledge

To archive the not functional requirement of a high-performance DB of TimeSeries type we have chosen InluxDB. It has the performance and capacity to model the data coming from the monitor.

Executor

The Executor component was developed using Java. It receives a unique adaptation message and sends a single adaptation to each room.

Mqtt Broker

To make MQTT connections we used the broker mosquito enclosed in Docker container.

Dashboard

To observe the various adaptations of the system, the Grafana dashboard connected directly to the Knowledge was used.

Environment

The environment is been simulated using a docker component that depends on each other. The first docker container is the nodejs application which simulates the environment while the second container is the node-red tool which simulates the talk of sensors and actuators with the mqtt broker.

Sequence Diagrams

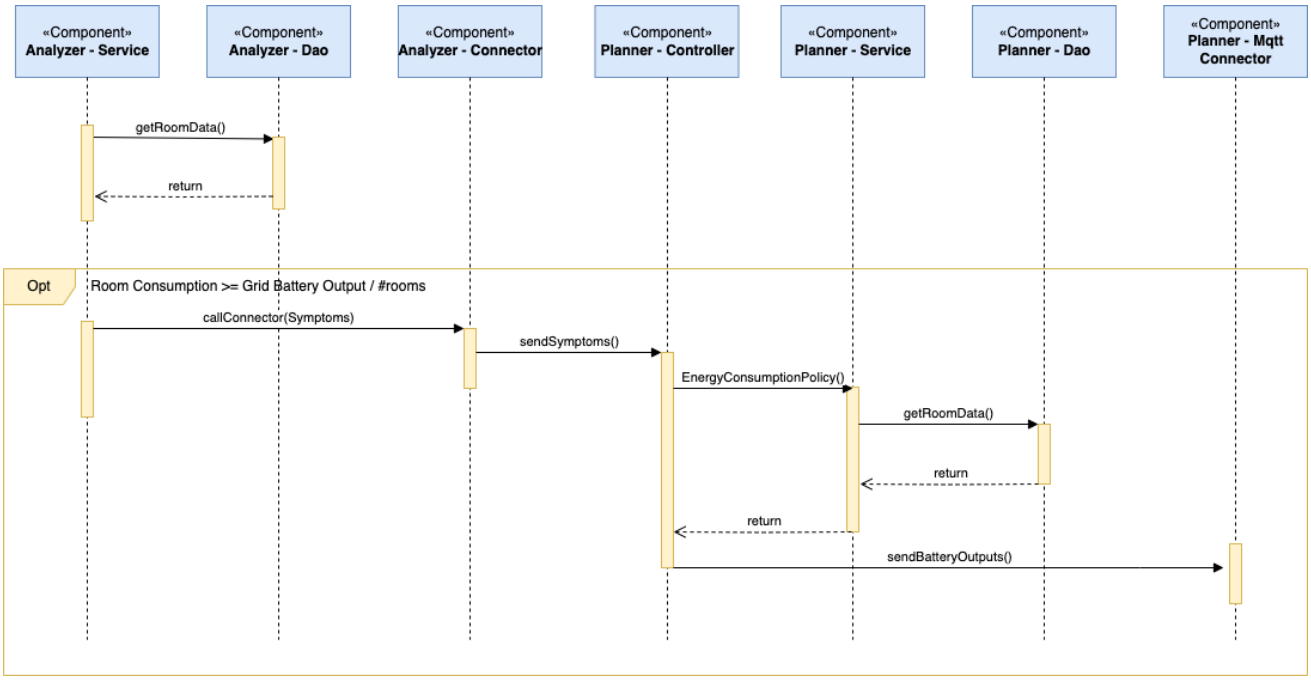


Figure 3: energyConsuption - Analyzer Planner Interaction Flow

To illustrate the data flow of the system, we took functional requirement #1 as an example. The above sequence diagrams describe its dynamics. The first sequence diagram illustrate the interaction between the analyzer and planner while the second sequence diagram shows the executor interaction.

The symptom that is used for the #1 functional requirement is “energy consumption”. The Analyzer Service uses a schedule to periodically analyze the data collected from the Analyzer Dao component. For each room, there is a check for the energyDemand value, and

if the check turns out true the Analyzer Connector will elaborate a JSON message and send it to the planner.

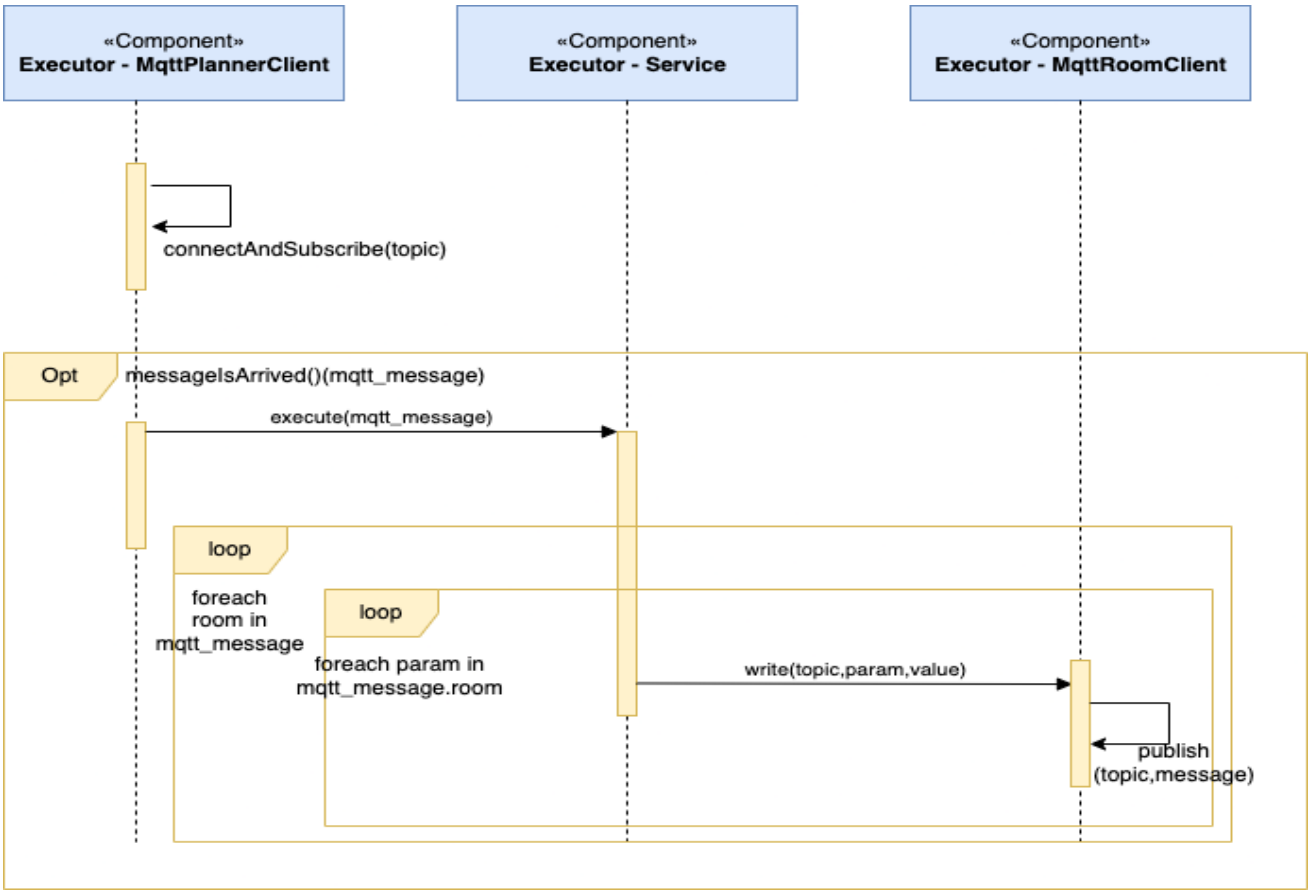


Figure 4: energyConsuption - Executor Flow

The Planner, having received the message from the analyzer, invokes the Energy Consumption Policy declared within it which will scale the outputs of the rooms to adapt them to their request. To do this, he will need to contact the Planner - DAO component to retrieve the data of the various rooms. Once the output values have been calculated, they will be sent to the mqtt broker using the Mqtt Connector component.

At this point, the Executor who owns an mqtt client that previously executed the subscribe() on the same topic, will receive the message sent by the Planner. The message contains the updated data of each room that the Executor-Service component is able to forward on the respective acting topics of the rooms. To do this, a first iteration of the rooms received from the planner is needed, and a second iteration nested over all the parameters to be updated.

Architecture Redesign

Component Substitution

We have chosen to redesign the planner to implement a proactive approach and to test the decoupling of the system in terms of containers. The new planner will be a new Java project integrated in terms of connectors with the others, which will adopt a proactive approach, so the policies will make use of all the data present in the knowledge rather than using only the current observations (reactive approach). In this way, the system will use all the data present in the knowledge, but any immediate short-term changes would not be satisfied.

The Idea

Using the aggregation functions of our knowledge component, we decided to use a linear regressor that projects the next datum on the basis of the last n analyzed. In particular, an average of the energyDemand values of the rooms is extrapolated every 10 minutes and the regressor is created. Extracting the series of values from the query described in the previous paragraph. The system makes a prediction on the next value of the series using a simple linear regression in which the independent variable X refers to the time (interval t - n / t0) and the dependent variable Y refers to the demand. Using this simple method we can possibly predict an average value in a time interval following the current series by exploiting the current demand trend and therefore understand whether it is constant, ascending, or descending.

$$a = \frac{\left(\sum y\right)\left(\sum x^2\right) - \left(\sum x\right)\left(\sum xy\right)}{n\left(\sum x^2\right) - \left(\sum x\right)^2}$$

$$b = \frac{n\left(\sum xy\right) - \left(\sum x\right)\left(\sum y\right)}{n\left(\sum x^2\right) - \left(\sum x\right)^2}$$

Figure 5: Linear Regression Equation

Results

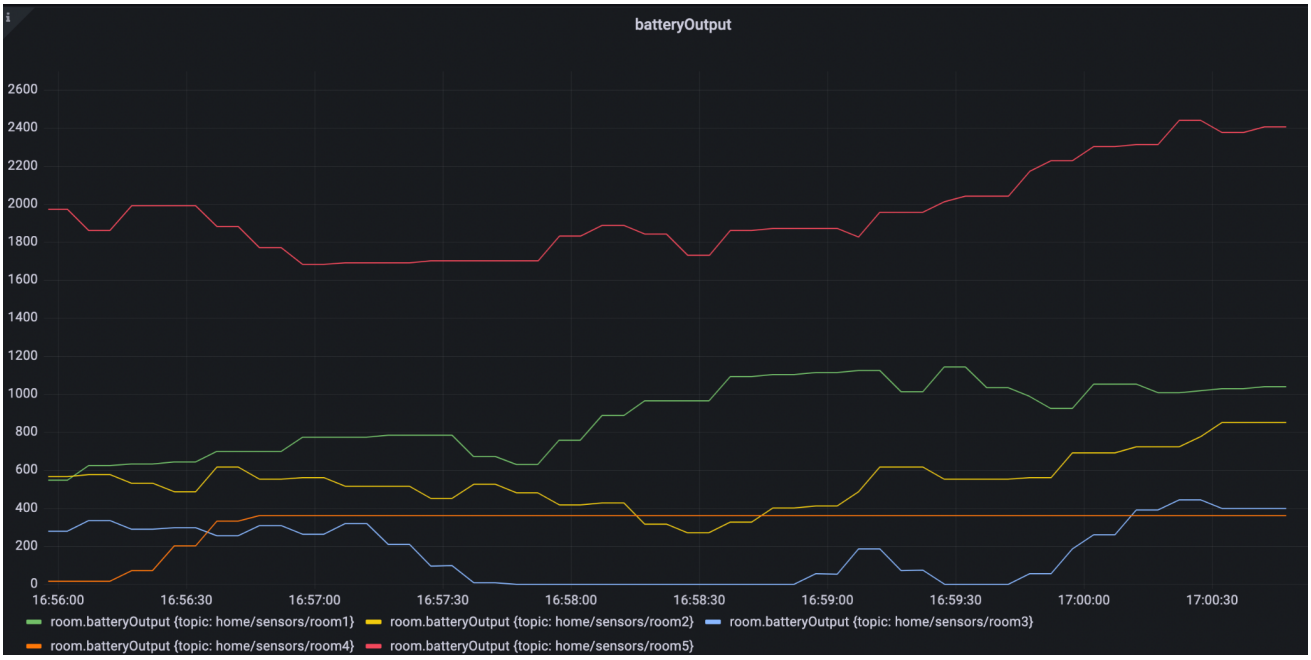


Figure 6: Reactive Approach

Figure 6 represents the use of a reactive approach in the system. We can see how the batteryOutput is adapted discontinuously having important deviations during the life of the process. Below, however, we observe figure 7 in which a proactive approach is implemented. As can be seen, the changes to the batteryOutput range are more linear and continuous.

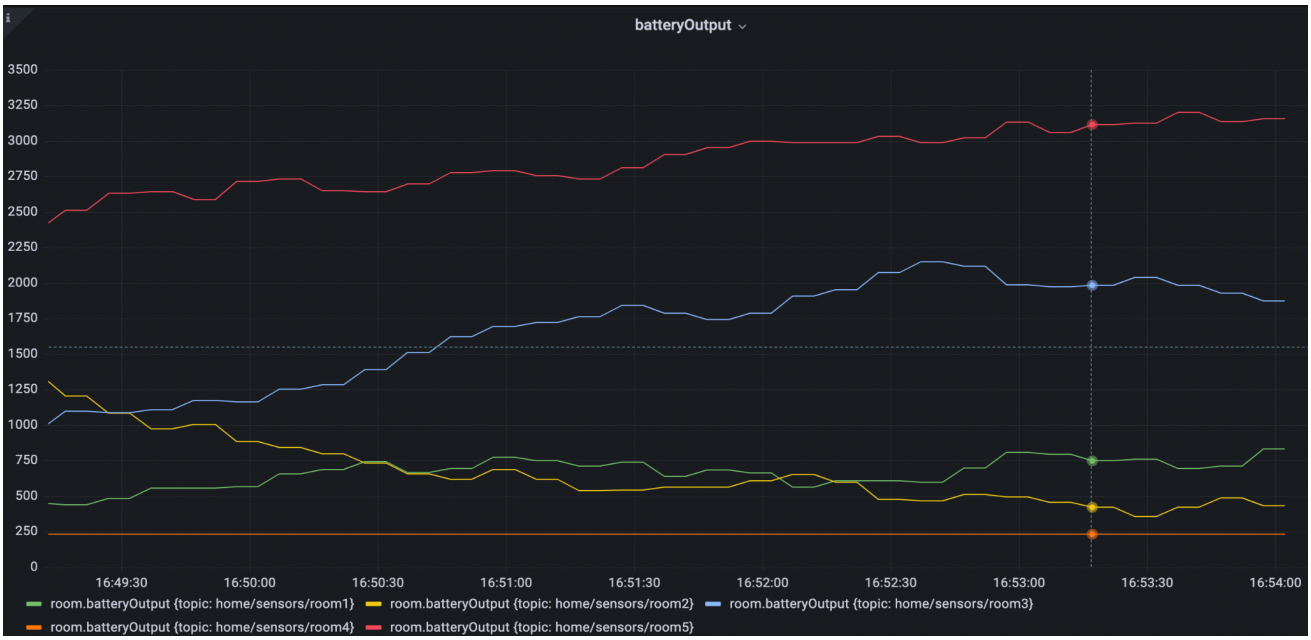


Figure 7: Proactive Approach

How Functional Requirements have been addressed

Functional Requirement #1 - Scaling Energy Consumption

Functional requirement #1 has been achieved by adding a particular symptom called "energyConsumptionAdaptation" thanks to which the system is able to understand when a room is consuming more energy than expected and adapts the output of the batteries based on the demand. In figure 8 we can see the correlation between the energy demand of each room and the battery output of that specific room.

Figure 8: Consumption Adaptation under Proactive Approach

Functional Requirement #2 - Room Dynamic Management

Requirement #2 is not about a symptom but about the scalability of the system in adding or removing rooms on which the autonomous system has to act. For this purpose, it was necessary to manage two fundamental cases:

- Dynamic addition of the room: Given the architecture of the system in which the monitor is subscribed to each topic under a specific path (home/sensors), for each new room it will be enough to send the message on the topic /home/sensors/roomID and the system will go to consider the new room in a completely autonomous way. With the specification, however, the system does not manage the definition of the name and ID of the room but it is done a priori during the configuration of the room to be added.
- Dynamic removal of a room that is no longer monitored: To deal with this problem, it was chosen to be based on a specific time window. In this way, all messages sent before the chosen window are not considered and therefore the zone will no longer be considered (in this way a zone can also return to activity at any time).

Functional Requirement #3 - Energy Loss Avoidance

Functional requirement #2 was achieved through the use of the "status" symptom which is triggered when the parameters are out of standard and from which it can be understood that there are problems on the electrical network of the room. There are two possibilities:

1. The room consumes more than the number of sockets multiplied by their maximum consumption
2. The battery has a charge level of 0 so there is some energy loss problem.

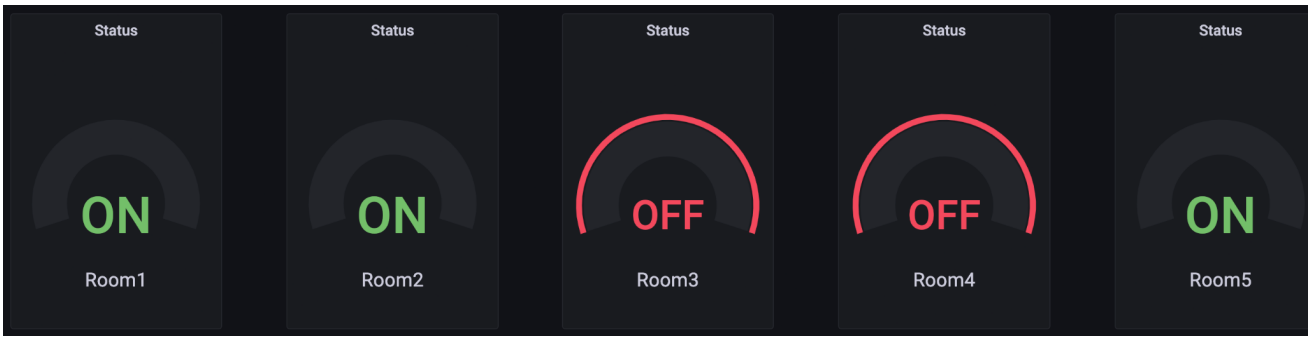


Figure 9: Room Status Panel

Reintroducing the room into the system can only be done manually by resetting the status to true. After resetting the value to true the system will analyze the room again and if it does not find critical conditions again it will consider it normal.

Functional Requirement #4 - Proactive Scaling Energy Consumption

For this specific functional requirement see the Architecture Redesign previously discussed.

TECHNOLOGIES