Università di Pisa
Department of Information Engineering

Master's Degree in Artificial Intelligence and Data Engineering
Internet of Things Project

# IoT-based Smart Room

**Authors:**
M. Meazzini, S. Micheloni

Academic Year 2024–2025

# Contents

# 1 Introduction

Nowadays, managing the comfort of a room, whether it is a workspace or part of a house, is a key use case for modern sensing and actuating technologies, as it can enhance users' quality of life and make their work more enjoyable.

To this end, our proposed application integrates IoT devices with Machine Learning technologies to manage the comfort of a Smart Room, while also optimizing energy efficiency through a dedicated Energy Saving mode.

The Smart Room is capable of self-regulating temperature and lighting levels by using dedicated sensors that feed data to the corresponding actuators. A humidity sensor also collaborates with the others to provide a comprehensive overview of the room's status. This information is then used by the Machine Learning module, which runs into the actuators, to determine whether the room is currently occupied.

# 2 System Architecture

The Smart Room application is built using various technologies and components. Below is a complete list of the architectural elements:

- CoAP Network with Border Router, Sensors and Actuators;
- Machine Learning classifier model, located in the actuator nodes;
- Java Application;
- MySQL Database;
- Grafana integration;

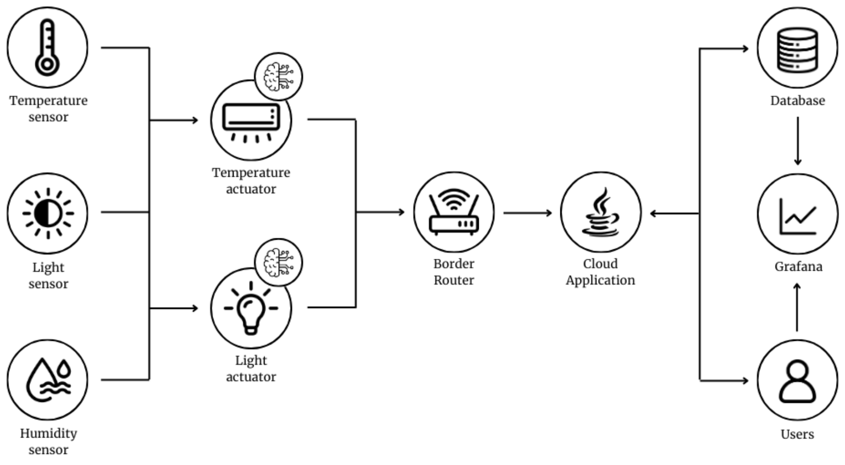

Figure 1: Overview of the system's architecture

In the following chapter, each of these components will be discussed in detail to explain their functioning and justify the choices made during the design process.

## 2.1 CoAP Network

The network of sensors and actuators is implemented using the **CoAP protocol**, which fits particularly well in situations where data don't change with high frequency, such as the parameters that are considered in this application (temperature, light and humidity). In this application, the Contiki built-in RPL Border Router is used to build the network and to let the others node to perform registration and resource discovery and observing.

Other nodes are:

- Temperature actuator, which includes cooler and heater;
- Light actuator;
- Temperature sensor;
- Light sensor;
- Humidity sensor;

Thanks to the **CoAP Restful protocol**, sensors can expose resources about their measured values (Temperature, Light and Humidity), while actuators expose resources about their internal thresholds and their working periods.

We chose to use **CoAP observing** instead of manual polling because it is better suited for applications of this kind, where energy efficiency is a key concern. Moreover, the variation of the monitored parameters does not follow predictable time intervals, making it difficult to define an appropriate polling frequency.

## 2.2 Machine Learning classifier

We trained our ML classification model in order to detect if the room is occupied by people or not just by looking at common sensors measurements. This way we can efficiently determine whether temperature and lighting adjustments are necessary.

### 2.2.1 Dataset Description

The used Room Occupancy Detection Dataset contains sensor data collected from a room environment and its relative occupancy status. The dataset includes measurements of various environmental variables and the associated status of the room:

| Feature | Type | UoM |
|---|---|---|
| Temperature | float | C |
| Light | float | Lux |
| Humidity | float | % |
| CO2 | float | ppm |
| Occupancy | bool | - |

Table 1: Dataset features

We chose this dataset because it lets us develop the Energy Saving mode, which is based on the occupancy of a room, while using commonly measured values. This way, the implementation of this application in a real room will be easier and cheaper than using other less common and more complex sensors.

### 2.2.2 Model Training

We trained and tested our classification model using the Room Occupancy dataset by leveraging popular Python libraries such as **pandas** for data handling, **NumPy** for numerical operations, **scikit-learn** for splitting the dataset into training and testing subsets, and **TensorFlow/Keras** for building and training the neural network. Initially, the dataset was preprocessed by selecting relevant features (Temperature, Light, and Humidity), rounding the values, and removing any incomplete or invalid records. The occupancy labels were then converted into categorical format suitable for classification.

The model architecture consists of a **densely connected neural network** with multiple hidden layers, each containing 30 neurons activated by the ReLU function, and a final softmax output layer for binary classification. A dropout layer was included to reduce overfitting. The model was compiled using the Adam optimizer and categorical cross-entropy loss, and trained for 20 epochs with validation on a held-out test set (15% of the data).

After training, the model's accuracy was evaluated on the test data. Finally, the trained model was converted into a lightweight format using the **emlearn** library for deployment in embedded systems.

#### 2.2.3 Performances and Considerations

Given that we are working with three sensors and intend to retain Temperature and Light measurements, we needed to choose between including Humidity or CO2 as the third feature. Initially, we trained and evaluated the classification model using Humidity data, followed by an evaluation using CO2 data. In both scenarios, the model demonstrated excellent performance (both times aroud 98% accuracy).

To make an informed decision on which feature to retain and which to exclude, we assessed the practical implementation costs associated with each option. Through market research, we compiled indicative price ranges for Humidity and CO2 sensors:

| Type | Sensor | Accuracy | Price |
|---|---|---|---|
| Humidity | DHT11 | Low | 3€ |
| Humidity | SHT31 | High | 10–15€ |
| CO2 | MH-Z19B | Low | 30–40€ |
| CO2 | SCD30 | High | 50–90€ |

Table 2: Comparison of different Humidity and CO2 sensors

This comparison clearly demonstrates that opting for a humidity sensor over a CO2 sensor is far more cost effective. Such a choice enables significant cost savings in large-scale deployments, such as hotels with many rooms to regulate.

### 2.3 Java Application

The developed Cloud Application was written in Java, exploiting the Eclipse Californium framework, and consists of different packages that perform different actions with the main functionalities being:

- Registration of CoAP nodes;
- Saving devices info to enable nodes Discovery;
- Saving sensors and actuators data into the MySQL Database;
- Expose commands to the user;

The application also provides logs on the status of its functionalities with a Logger module.

## 2.4 MySQL Database

The employed relational MySQL database is used to save data about the CoAP network and its nodes. In particular it is composed of 6 tables, with the following attributes:

- device_registry:

| Attribute | Type | Notes |
|---|---|---|
| id | int | PRIMARY KEY |
| name | string | NOT NULL |
| ip | string | NOT NULL |
| port | int | NOT NULL |
| registered_at | timestamp | NOT NULL |

- temperature_data, light_data, humidity_data, temperature_actuator, light_actuator:

| Attribute | Type | Notes |
|---|---|---|
| id | int | PRIMARY KEY |
| name | string | NOT NULL |
| value | float/int | NOT NULL |
| timestamp | timestamp | NOT NULL |

  The type of the "value" attribute is float for the sensors tables, and it's int for the tables related to the actuators usage.

## 2.5 Grafana

In this application, Grafana is used as a monitoring tool to observe the key variables over time that represent the state of the CoAP-based sensor and actuator network.

The charts displayed in Grafana include data from both the sensors and the actuators. In particular: the sensors provide data used to generate three time-series graphs for temperature, light, and humidity, while the actuators provide data regarding their operating periods.

While the sensor graphs are straightforward and serve mainly as real-time monitors, the actuator charts can offer valuable insights into the effect of the Energy Saving mode on actuator usage, and consequently, on overall energy consumption in a practical deployment.

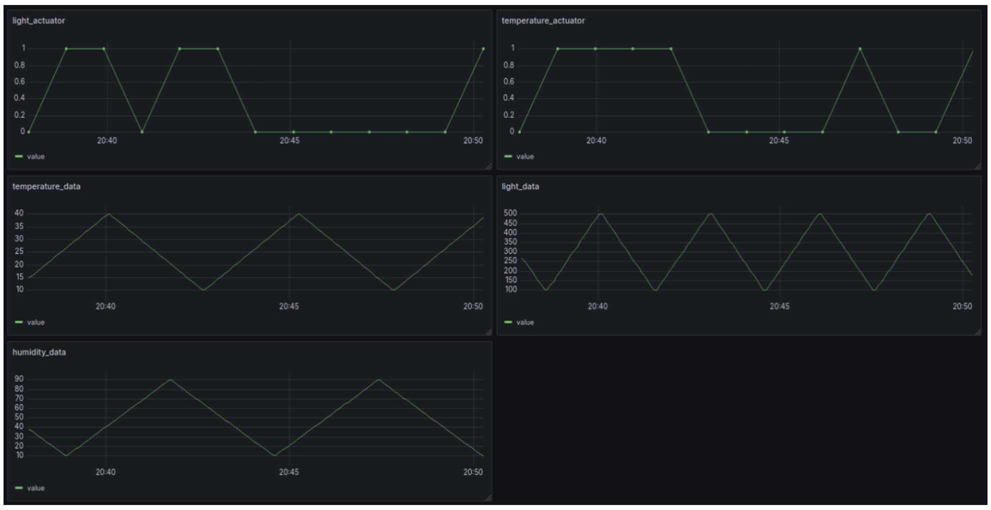

Figure 2: Overview of the system's architecture

## 3 Application Workflow

This section describes all the main phases of the application's operation, starting from the setup process, covering the management of room parameters, and ending with data collection and its visualization in Grafana.

### 3.1 Nodes Registration and Discovery

After the preliminary phase, where the Java application is launched, the database is cleared, and the sensor and actuator nodes connect to the Border Router, the **registration phase** takes place.

During this phase, each node sends a registration request to the Border Router, containing its identifier (e.g., sensor_light). The Border Router intercepts the request and forwards it to the Java application, which stores the device information in the database, specifically in the **device_registry** table.

The correct execution of this phase is essential to ensure the success of the following **discovery phase**. In this stage, each actuator node needs to discover the addresses of the sensor nodes so it can receive their measurements using **CoAP observing**, without accessing the database, avoiding unnecessary overhead.

To achieve this, each actuator sends three separate requests to the application, one for each sensor. Upon receiving these requests, the application responds with the IP address of the requested sensor, which the actuator then stores for use during the subsequent observing phase.

### 3.2 Room Parameters Regulation

Once the actuators have discovered and stored the IP addresses of the sensor nodes, the core phase of the application can begin: the regulation of the room's temperature and lighting.

The main actors in this phase are the actuators, which initially send three observing requests to the resources exposed by the sensors. These resources provide the measured values (simulated in our case) of temperature, light, and humidity.

Once observing is successfully established, each actuator is continuously notified of any changes in the sensor measurements. Upon receiving an update, each actuator performs two main actions: It stores the new value and uses its built-in classification model to determine whether the room is currently occupied. If the received measurement corresponds to the physical quantity managed by the actuator (e.g., the temperature value is received by the temperature actuator), the actuator will decide whether to turn on or off by comparing the value with its internal thresholds.

Additionally, if the Energy Saving mode is enabled and the room is classified as unoccupied, the actuator will remain off even if the received values exceed the thresholds.

## 3.3 LEDs and Buttons meaning

While performing room parameter regulations, it's possbile to interact with nodes LEDs and Buttons to perform simple actions, in particular:

### 3.3.1 Temperature actuator:

- Green LED: Simulates the activation of the cooling system;
- Red LED: Simulates the activation of the heating system;
- Yellow/Blue LED: Blinks every 1s if Energy Saving mode is active;
- Button: Toggles Energy Saving mode;

### 3.3.2 Light actuator:

- Red LED: Simulates the activation of the light;
- Yellow/Blue LED: Blinks every 1s if Energy Saving mode is active;
- Button: Toggles Energy Saving mode;

### 3.3.3 Sensors:

- Green LED: Starts blinking every 1s when the sensor is successfully registered to the application

## 3.4 Data Encoding and Transmission

Smart Room features different types of data transmission, each one with its specific encoding format, which is designed to be lightweight and intuitive:

- Sensor discovery made by actuators: in this phase actuators send 3 requests to the Cloud Application to discover the ip of the sensor nodes and the Application sends a message containing only the ip in the payload, with no particular encoding.
- Sensor data sent to actuators: while observing the sensors' resources, the actuators receive measured values in the payload. In this case no further details in the payload are necessary since the actuator can interpret the received value by looking at the resource's URI (i.e. sensors/light).

- Editing thresholds with a command: If an user wants to edit the internal thresholds of an actuator, he can do it using the relative command. In this case, a POST request will be sent to the threshold resource with a specific encoding: "min=value&max=value" (or "min=value" in the light actuator), this way the actuator can easily parse the request.

## 3.5 Data Collection and Remote Application

While the sensors and actuators are busy measuring and regulating the room parameters, the Java application collects data related to the sensor measurements and the operating periods of the actuators through **CoAP observing**, in order to populate the MySQL database.

In addition to this, the Java Application exposes a set of user executable commands, that consists in simple POST and GET requests to nodes and Database queries, such as:

- Set actuators thresholds;
- Show current thresholds;
- Show registered devices;
- Show recent sensor data;
- See help;
- Close the application.

## 4 Use Case

The simplicity of this project's implementation makes it suitable for integration into common, pre-existing manually regulated environments. This is possible because the application relies on widely available sensors that are already present in most workspaces, or can be easily purchased at a low cost for use in typical household rooms.

The main use case we considered is hotels, where user comfort is a key factor for success. With our low-cost, self-regulating system for temperature and lighting, hotels can maintain a pleasant environment while also controlling costs through the proposed Energy Saving mode. It is also important to note that many modern hotels are already equipped with temperature and light sensors in each room, which would further reduce the cost of implementation.

Another relevant use case is the typical household, where energy efficiency and affordability are often major concerns. Thanks to its low power consumption and minimal hardware requirements, the proposed system can be easily integrated into existing home environments to improve comfort while keeping installation and operational costs very low. This makes it a viable solution for families aiming to reduce energy usage without sacrificing living quality.

These are just two of the many possible use cases, one only needs to consider the wide variety of environments regularly occupied by people, such as offices, schools, hospitals, or public buildings, all of which could benefit from automated comfort regulation.

## 5 Conclusions

This project has demonstrated the capability of the CoAP protocol to support a real-time regulation system, even when integrated with Machine Learning components. The system effectively balances responsiveness and energy efficiency, proving that lightweight protocols like CoAP can be a solid foundation for intelligent IoT applications.

Thanks to its simplicity and modular design, this application can be easily extended or adapted to more complex scenarios, including the integration of additional sensors, more advanced learning models, or remote control functionalities. Its flexibility opens the door to a wide range of future developments.