UNIVERSITÀ DI PISA

Master's degree in Artificial Intelligence and Data Engineering

Internet of Things project documentation

HomeAutomation

Denny Meini

Leonardo Bargiotti

Anno accademico 2023/2024

Introduction

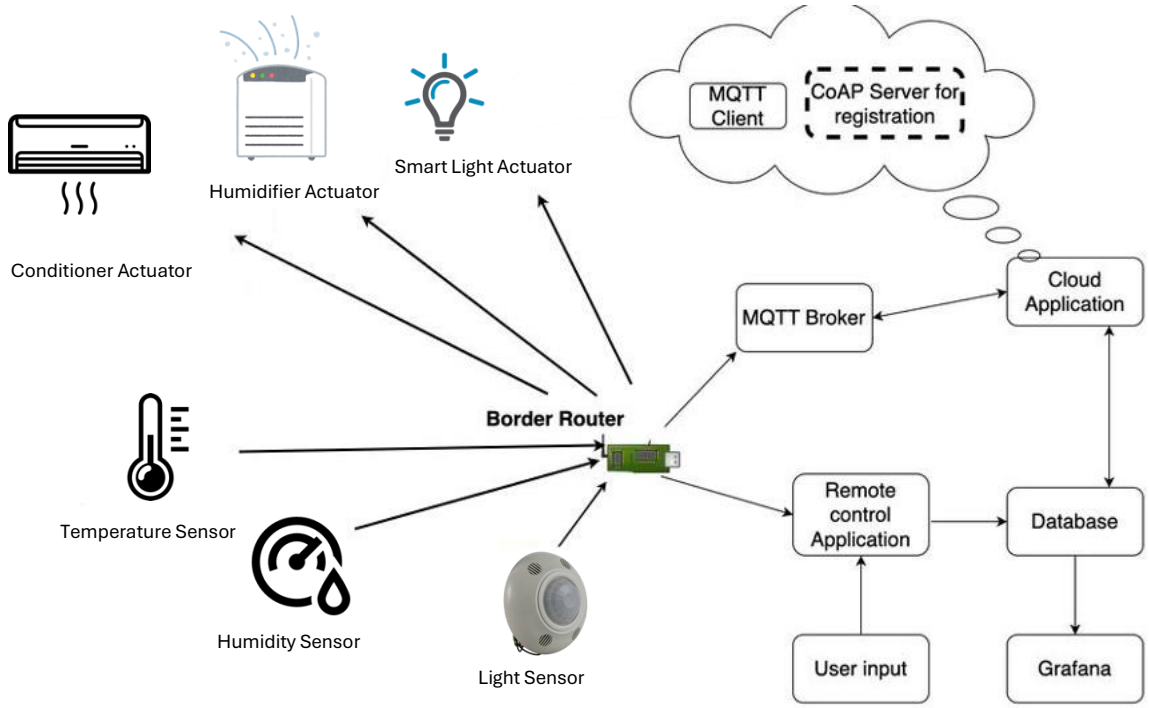
HomeAutomation is an IoT system designed to monitor and optimize environmental conditions within a home, using temperature, humidity, and light sensors. This system is intended to improve household comfort and facilitate the automated management of the home environment.

Thanks to the integration of sensors, the system collects real-time data on the internal conditions of the house, allowing the user to monitor and manage the environment by automating the system's responses based on predefined thresholds. The temperature and humidity sensors help maintain an ideal climate, while the light sensor optimizes lighting.

The main goal of HomeAutomation is to provide an easy-to-use system that leverages IoT connectivity to create smarter home environments.

Architecture

The overall architecture of the application is defined below.



Regarding IoT devices, we have two types: sensors and actuators.

The sensor network consists of three different types of devices: a temperature sensor, a humidity sensor, and a light sensor. Each of these is part of the MQTT data-producing network, as their role is to retrieve external values (synthetically generated in this case) and then publish them on the MQTT topics, where they will be processed by the application.

The actuator network, on the other hand, receives CoAP calls to exposed resources to perform some operations in response to the sensed values. We have one actuator for each type of sensor.

On the other hand, there are two distinct components for processing the retrieved data. The first is the **Cloud application**, which subscribes to MQTT topics to gather the sensed values and store them in a database. The second key component is the **Remote Control application**, responsible for retrieving data from the database, processing it, and making CoAP calls to actuators in response to detected hazardous values. Additionally, it handles user requests submitted via the Command Line Interface.

Sensing/Actuating network

Sensors MQTT network

The main components of the MQTT network are the sensors. In our application there are 3 types of sensors:

- **Temperature sensor**: is responsible for detecting the temperature in the environment. The code flashed on the sensor simulates this behavior by generating values within a plausible range, which we set between **18** and **28** degrees Celsius. Values that fall below or above this interval are considered inappropriate.
- **Humidity sensor**: is responsible for detecting the humidity in the environment. The code flashed on the sensor simulates this behavior by generating values within a plausible range, which we set between **40%** and **60**%. Values that fall below or above this interval are considered inappropriate.
- **Light sensor**: is responsible for detecting the level of light in the environment. The code flashed on the sensor simulates this behavior by generating values within a plausible range, which we set between **100** and **300 lux.** Values that fall below or above this interval are considered inappropriate.

To ensure that the application handles all possible scenarios, we decided to have the values increase by a certain value every 10 seconds (2 °C for temperature, 3% for humidity, 30 lux for light). Once the maximum limit is reached, the values decrease by the same values every 10 seconds until they hit the minimum limit within the defined range, at which point they start increasing again. The opposite cycle also occurs, where the values begin decreasing and then increase once the minimum limit is reached, creating a continuous oscillating pattern between the two limits.

Each type of sensor was deployed on a single dongle. To enhance detection reliability, it may be beneficial to increase the number of sensors for each type of measurement.

The three sensors publish data every 10 seconds on the topics “temperature,” “humidity,” and “light,” respectively.

Actuators CoAP network

The first action the actuator must take is to register itself with a CoAP Registration server, allowing it to be accessed by the remote control application, which initially does not know its IPv6 address. During registration, each actuator sends a JSON payload containing a key-value pair that identifies its type (e.g., {“type” : “temperature”}).

There are three different actuators:

- **Conditioner actuator**: is used to control the air conditioner, either to cool or heat the air when a temperature value above or below the normal range is detected. A led is activated to signal a value out of the range, with its color set to **RED**.
- **Humidifier actuator:** is used to control the humidifier, either to increase or decrease the humidity in the air when a humidity value above or below the normal range is detected. A LED is activated to signal a value out of the range, with its color set to **BLUE**.
- **Smart Light actuator:** is used to control the lamp, adjusting its intensity either up or down when a value above or below the normal range is detected. A led is activated to signal a value out of the range, with its color set to **YELLOW**.

After the registration, the actuator simply waits for a call or for its button to be pressed.

When a value is sensed out of the range, an actuator call makes the actuator turn on and the LED is switched up (the colors are the ones described before), when the value goes back into the range the LED turn greed to indicate that now the value is into the optimal range.

Once the green LED is on, the user can now confirm he understood by pressing the button and turning off the LED. If the button is not pressed, the green LED will stay on as long as the value is into the range, then will change color to the one that is associated to it and so on.

It’s also possible to turn the actuators on and off manually, simply using the command that the system gives to the user in the CLI. When an actuator is manually turned on the LED becomes green, when it’s turned off manually, the LED turns off.

Application

The application side is responsible for storing, processing, and visualizing all retrieved data. It also takes appropriate action if thresholds are detected by activating the actuators to perform specific functions. The following paragraphs define each component of this side in detail. Each of these classes is implemented as a singleton and extends the Runnable interface to be utilized with a ScheduledExecutor at the application's entry point

Application

This is the core part of application, which is made up of three different components:

- **Periodic data retrieval:** implements a mechanism for the periodic retrieval of data from the sensors, constantly monitoring environmental values against the predefined minimum and maximum thresholds for temperature, humidity, and light. During execution, the class compares the detected values with the preset thresholds and decides whether to activate or deactivate the corresponding actuators to prevent dangerous situations. The class's behavior is governed by a loop that retrieves data from the database and checks if the values are outside the allowed limits. In this case, a command is issued to activate the actuator associated with the relevant sensor. Conversely, if the values return to normal, a command is sent to deactivate the actuator, avoiding unnecessary use.
- **Command line interface**: this class processes the user inputs. The action that can be performed are mainly 4: print the help list, select new thresholds for the sensed values, change the status of the actuators (only in case the specified one is different from the state the actuator is already in) and check their status. Those actions are performed by simply inserting the corresponding number specified by the help command, in the way specified below:
    1. Help list
    2. Select a new value for the temperature threshold
    3. Select a new value for the humidity threshold
    4. Select a new value for the light threshold
    5. Change the status of the temperature system
    6. Change the status of the humidity system
    7. Change the status of the light system
    8. Check the actuators status
- **Main:** is responsible for initializing and managing the execution of various processes using a scheduled thread pool. This approach ensures that the main components of the application are started and monitored automatically without interruption. Specifically, the class sets up a scheduled executor that:
    - Starts the Command Line Interface (CLI).
    - Initiates communication handlers for the COAP and MQTT protocols.
    - Schedules a periodic data retrieval routine.

MQTT

This component manages the communications handled by the sensors.

- **MQTT handler**: It manages the connection and interaction with an MQTT broker for receiving data from the sensors. The class establishes a connection to the local MQTT broker and subscribes to the sensor-related topics (temperature, humidity, and light). When a message arrives on one of the topics, the content is parsed and inserted into the database, updating the sensor values in the system

CoAP

This component manages the communications handled by the actuators.

- **CoAP client**: manages communication between the system and the actuators through the CoAP protocol. Its main role is to send commands to the actuators based on the conditions detected by the sensors. The class determines if the environmental conditions exceed a threshold and, if so, sends a command to activate or deactivate the corresponding actuator. The device's response is handled to ensure that the operation was successfully executed, updating the system's status accordingly. This approach allows for dynamic reactions to changes in the monitored environment.

- **CoAP registration**: manages the registration of actuators in the system through a CoAP resource. When an actuator sends a POST request to register, the class extracts and parses the data from the request. The IP address of the actuator is recorded, and it is assumed that actuators start in the OFF state. After obtaining the information from the request payload, the class attempts to update the database to reflect the registration of the actuator. This mechanism allows the system to maintain an updated list of registered actuators, facilitating the management of communications and monitoring the status of the actuators within the CoAP architecture.

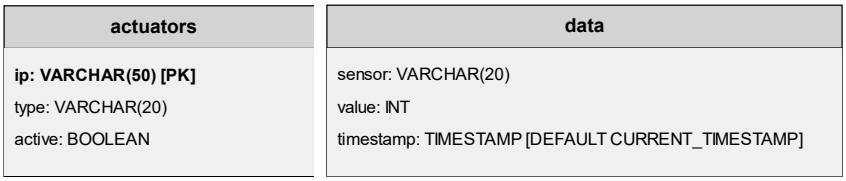
Data encoding

When choosing a data encoding language, we primarily considered two options: XML and JSON. After evaluating the pros and cons, we decided that JSON was more suitable for our application for several reasons. JSON is less verbose and generates simpler messages, which reduces bandwidth usage during data exchange. Additionally, managing payloads became easier, as there are numerous libraries available to efficiently build and parse JSON messages with minimal effort.

Database

The database component is used to store two different pieces of information in two specific tables inside of an SQL Database, in order to be retrieved later. Those two aspects are the data retrieved by the sensors and the actuators registered to the application and stored by using the **DBDriver** class.

The two tables are composed in the following way:



Regarding the actuators, we identify them using their IP addresses, which will be unique to each actuator. Additionally, we include the type of actuator and its status, which can be either “true” or “false,” indicating whether it is active or inactive.

The data table stores information about the sensors using their type and the timestamp of data insertion. Additionally, the sensed data is recorded in INT format.

The class that manages insertions and retrievals from the tables is, as previously said, DBDriver: it provides a DBMS connection towards our SQL instance, and 5 different methods/queries

- **resetActuators**: used to reset the actuators tables before the start of the CoAP server.
  Query: DELETE from actuators
- **updateActuators**: used to insert the registered actuators and to change their status when they get turned on/off.
  Query: *REPLACE INTO actuators (ip, actuator_type, status) VALUES(?,?,?)*
- **retrieveActuator**: used to retrieve the status and the ip address of a given actuator
  Query: *SELECT ip, status FROM actuators WHERE actuator_type = ?*
- **insertData**: used to insert the data retrieved from the MQTT queues, sensed by the IoT devices.
  Query: INSERT INTO data (value, sensor) VALUES(?,?)

- **retrieveData**: used to retrieve the most recent value for each distinct type of sensor. Query: *SELECT sensor, value FROM data WHERE (sensor, timestamp) IN (SELECT sensor, MAX(timestamp) FROM data GROUP BY sensor)*

Grafana

To monitor the real-time status of the sensed data, we connected a Grafana instance to the corresponding database table. This enabled both us and potential users to observe the external environment's behavior and how it changes over time.