

**UNIVERSITY OF PISA**

Internet of Things

*Year 2020/2021*

**Implementation of a temperature regulation system**

*Project specification*

*FRANCESCO CAMPILONGO*

*FRANCESCO DEL TURCO*

**1.0 Introduction 3**

**2.0 Architecture 10**

2.1 CoAP Network 11

2.2 MQTT Network

2.3 Collector section 10

2.4 Database

2.5 Data Enconding

2.6 Grafana

**3.0 Deployment and execution 14**

Link GitHub: <https://github.com/FranDT/ProjectIoT>

***1.0 Introduction***

This elaborate is used to present the implementation of a temperature regulation system following the specification of the project, therefore deployed on a MQTT network and a CoAP network.

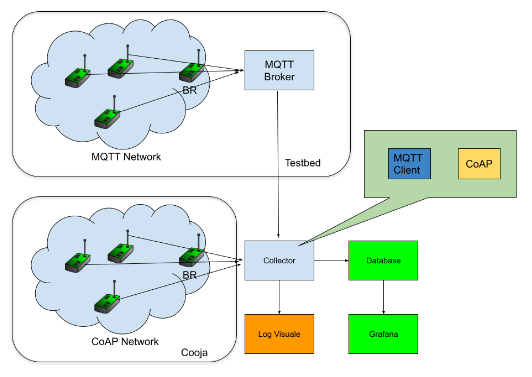
The goal of the system is to recognize, through a thermometer working as sensor, when the temperature of a room exceeds the threshold value, in order to send to the actuator (in this case, we consider it being both able to rise and lower the temperature) a message to regulate the temperature to the standard level. If the temperature of the room gets higher than 25°C or gets lower than 18°C, the corresponding actuator will be activated until the sensed temperature will return to the standard temperature value, equal to 21°C.

This system tries to emulate at best the behavior of a real system, in which the thermometer is usually separated from the actuator, either a conditioner or a radiator: those entities need in any case to be coupled not randomly, but considering the room they’re placed in.

The sensors will send the sensed temperature to the Collector, which is in charge of controlling the temperature and sending a message to the actuator in case a regulation is needed. The information about the sensors, actuators and the measurements are saved inside a MySQL database, that is then connected to a Grafana dashboard in order to give the possibility to the system administrator to check the situation in an easy way.

***2.0 Architecture***

In the following picture we can see a simple scheme of the architecture, as required from the specification of the project:



As we can see, there are three main sections that shape this system: a CoAP Network, a MQTT network and a Collector, connected to a database, that is used as a central hub to collect the information coming from the field devices to be used and controlled in the future.

***2.1 CoAP Network***

A CoAP Network has been created as part of the project and deployed through the Cooja simulator, simulating the presence of 5 nodes: a border router, corresponding to the default border router we can find inside the Contiki-ng/examples/rpl-border-router folder; 2 sensor nodes, corresponding to two thermometers that will have to control two different rooms; 2 actuator nodes, that will be tightly coupled with one of the two sensors (ideally, the one in the same room, done through population of the database before the start of the system).

Sensor nodes will expose a single observable resource define as /temp: thanks to this, after that the node has registered its presence with the Collector, it will sense and send periodically the temperature value of the room, that will be analyzed by the Collector in order for it to decide whether to activate the corresponding actuator or not. Since we work in a simulated environment, the regulation of the temperature is virtually done inside the sensor node itself, with the actuator nodes that will just print a message to show that they received an activation or suspension message from the Collector.

Actuator nodes will expose a single resource named /act: a POST request towards such resource will trigger the execution of the corresponding handler (the only one defined for such resource), in which we will check the option “mode” to understand if we need to turn the actuator on or off; in case the request is to turn on the actuator, the option “value” will be checked, so that if it is equal to “up” the actuator will try to rise the temperature, while if it is equal to “down” it will try to lower the temperature.

As we can see, in this network sensors will only sense the temperature, without doing anything beside communicating the temperature to the Collector; the actuators instead will just wait until they receive a POST request from the Collector, performing at that point the corresponding action. No interaction is present between nodes.

***2.2 MQTT Network***

The MQTT Network has been deployed on the testbed, as shown at chapter 3.0: in particular, we’ve deployed a couple sensor-actuator and a border router; the MQTT broker has been deployed on the testbed as well.

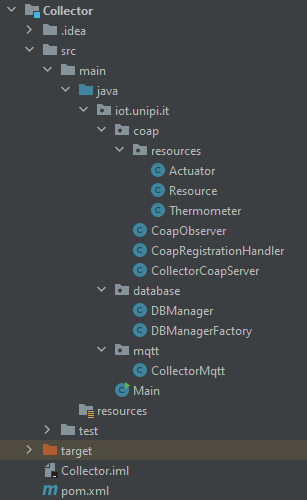
Sensor nodes are not subscribed to any topic, but are in charge to periodically publish a message containing the sensed temperature on the “temperature” topic: these messages will arrive eventually to the Collector, that is subscribed to the same topic and will receive all the messages published by the sensors (to understand which sensor has published each message, the MQTT sensor ID is included in the message).

Actuators nodes instead are not in charge of publishing anything, but each of them is subscribed to a different topic called “actuator\_*actuatorID*“: once the Collector has received a message about the “temperature” topic, it will check the ID of the sensor who published it in the first place; if the temperature needs to be regulated, the Collector will search for the corresponding actuator and will publish a message on the corresponding topic, so that the right actuator can be activate/suspended. In this case, the options “mode” and “value” are passed to the actuator inside the message itself.

The broker is implemented using Mosquitto and, thanks to port forwarding, is able to receive messages both from the Collector deployed locally and from the sensors/actuators deployed on the testbed.

***2.3 Collector section***

The collector section includes multiple parts inside itself, since it has to implement a MQTT part, a CoAP part and a DB handling part. This section is implemented as a Maven project and is written in Java: the structure of the section is the following:



Inside the POM file we’ve declared the dependencies that we’re going to use for our project: in particular, we have Californium and Paho to handle the two networks, the MySQL connector to handle the database and the JSON dependency for data encoding.

Since we need to handle two different networks from the same Java application, we decided to have two different threads, each one dedicated to one network: this division is performed inside the Main class, in which we also initialize the DBManager class, which is handled as a singleton.

The CoAP part of the Collector starts with the CollectorCoapServer, that is a class implementing the CoapServer class and to which a single “registration” resource is added, corresponding to the CoapRegistrationHandler class: the sensors and actuators from the CoAP Network will issue a POST request towards this resource in order to register themselves to the application. Inside the CoapRegistrationHandler class we will have a handlePOST function that will be triggered every time a registration request from a node arrives: the Collector will save the address of the source node and will issue a GET request toward the “/.well-known/core” resource of the node, that will reply with the list of resources exposed by the node. Since each node has one single resource, the Collector will check if the node is a thermometer or an actuator and will register it to the database consequently; if the node is a thermometer, we will also start observing that resource, waiting for the incoming message with the new measurements. The CoapObserver implements the observation itself: each time a message arrives from a sensor, it is analyzed to understand whether it’s necessary or not to send a message to the corresponding actuator, which information needs to be retrieved from the database. We can see that two resources are defined, Thermometer and Actuator: these classes both extend the Resource class, that defines a nodeAddress field (for the IP address) and a resourceName field; the Actuator class also defines an isActive field, that is checked before trying to send information to the corresponding actuator to make sure it has already registered.

The CollectorMqtt class is very similar to the CoapObserver class in its functioning, but initially it obviously has to create an MqttClient instance to connect to the broker and subscribe to the “temperature” topic.

Finally, all the queries to be issued to the database are declared inside the DBManager class, in which the actual connection to the database is performed as well. Since we need just one DBManager instance, we used the DBManagerFactory class to create a singleton instance to be retrieve by the different classes when they need it.

***2.4 Database***

***2.5 Data Encoding***

As requested in the project specification, we studied the best solution in terms of data encoding for our project and we came to the following conclusion:

* For the communication of the measurements in both CoAP and MQTT, we decided to use a JSON messages since we have to insert multiple fields inside the message itself, which is very easy with JSON;
* If the Collector has to send a message to an actuator in the CoAP Network, it will use a POST request where the payload will contain the parameter as plain text, since they’re easier to be parsed inside the actuator when using CoAP;
* If the Collector has to send a message to an actuator in the MQTT Network, it will use JSON for the same reason of point 1.