University of Pisa

School of Engineering

Master of Science in Artificial Intelligence and Data Engineering

---

Smart Greenhouse monitoring and management IOT system

Massimo Valentino Caroti
Simone Landi
Academic year 2021-2022

INDICE

INTRODUCTION

The Smart Greenhouse is a prototype remotely controlled and monitored greenhouse. Using IoT-based communication and control technology, the smart greenhouse enables remote control and monitoring of the environment. The goal of this project is to design and implement an IoT solution for autonomous management of a greenhouse.

The system consists of nodes that track soil moisture, ambient light level and temperature with their sensors. The sampled data is sent from the monitors, via MQTT or CoAP, to a collector, which stores it in a database for data collection and analysis. The collector can also detect anomalies in the measurements and manage them.

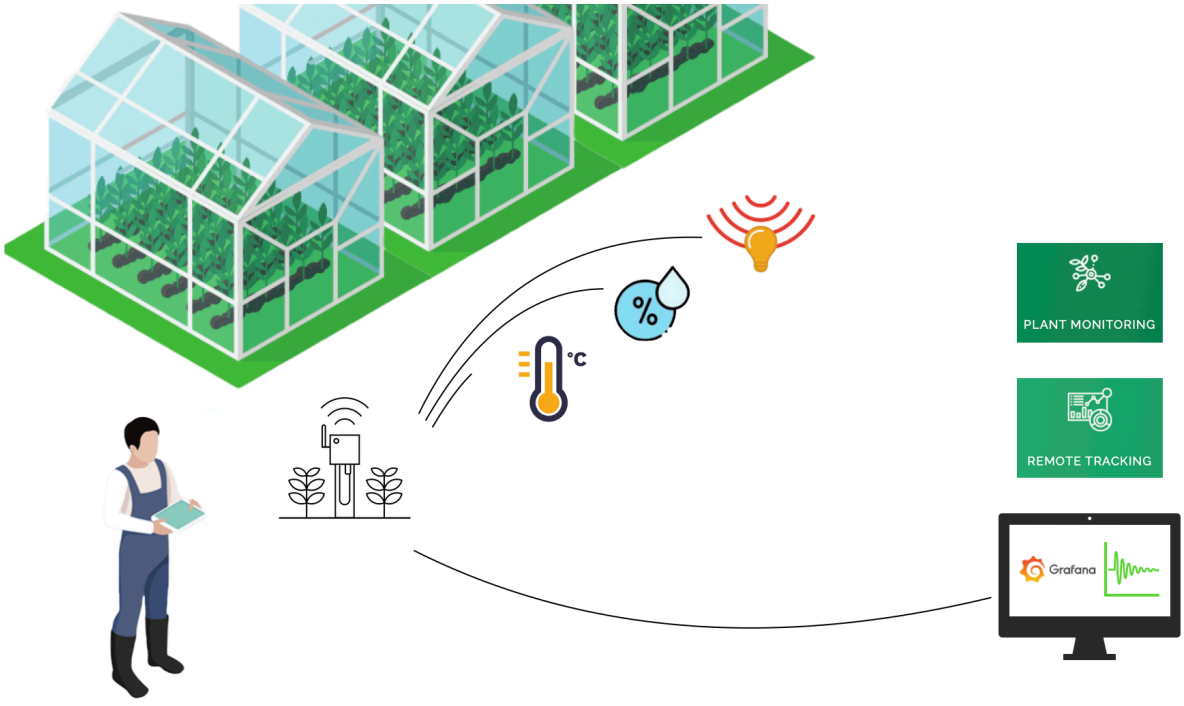
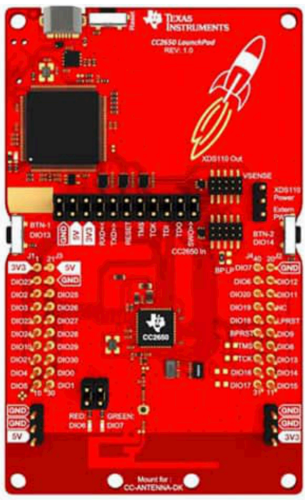


Figure 1: Scheme overview

DESIGN

OVERVIEW



The Smart Greenhouse system consists of a network of monitors that measure environmental parameters, installed directly on a greenhouse. Each monitor is an IoT device (CC2650 Launchpad from Texas Instruments) equipped with the Contiki-NG operating system and exchanges data with a collector, running on an Ubuntu virtual machine.

The network is an LLN leveraging 802.15.4 and IPv6 protocols, where multi-hop communication is enabled by RPL and traffic with the collector is enabled by a border router.

The monitors are equipped with three different sensors (humidity, light, and temperature) and an alarm system. The latter is capable of turning on a series of LEDs when triggered. The devices support MQTT and CoAP as application protocols for exchanging measurements and actuator commands. The MQTT broker is deployed on the Ubuntu virtual machine. The collector, written in Java, is responsible for receiving data from the dongles, saving it to a database, and detecting anomalies in humidity, temperature, and light level measurements.

Once registered, the monitor is fully operational. In this state it periodically acquires data on the assigned portion of the greenhouse using its sensors. Once a new sample is ready, the monitor sends it to the collector. At the same time, the collector analyzes the samples. If the measurements are above or below a certain threshold, the collector sends an alarm message to the node. In the fully operational state, if the values of the three parameters are below the predefined thresholds, the green LED is lit. If at least one of the parameters does not meet the threshold, the red led is lit. If the soil moisture falls below the minimum threshold, the node turns on the red led until an operator manually

activates the irrigation system. The operator then presses the device button, which triggers the alarm to turn off.

At each stage, the data exchanged is encoded in plain JSON, using a proprietary format. This choice allows for lighter communication, faster processing times, and simpler representation of information.

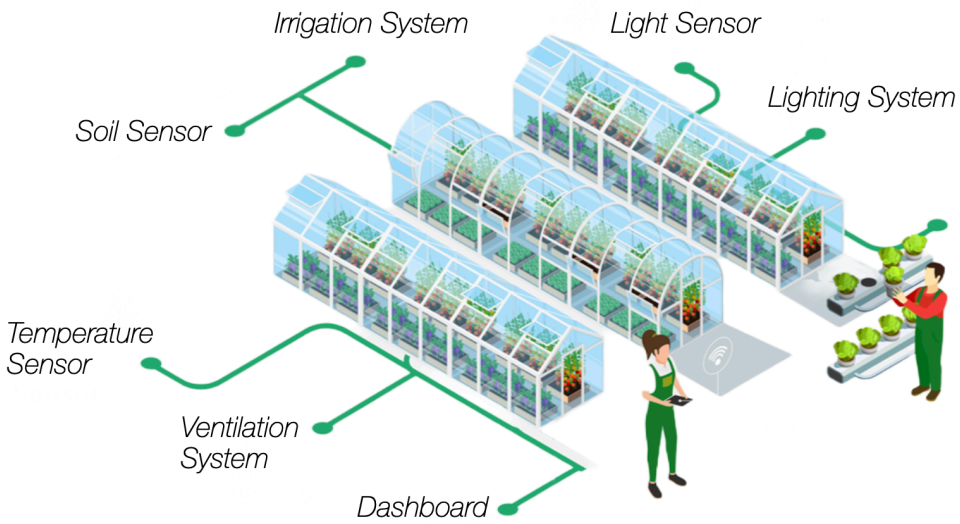


Figure 2: sensors and actuators

MQTT NODE

An MQTT node connects as a client to an MQTT broker, publishing and subscribing to dedicated topics to communicate with the collector. Each node is characterized by an ID.

The nodes publish the data to the following topics: temperature, light and humidity, while the monitor publishes on alarm_ID topic of each nodes to notify the alarm message.

- topic_name holding the last value sampled by all the sensors in the format specified above. For example, the resource temperature will make available the latest measurement as {"n": "temperature", "v": "45", "u": "C", "id": “2”}.

To receive the alerts from the collector the sensor subscribes to the following topic:

- alarm_NODE_ID which is used to receive alarm messages from the collector.

  The messages send by the collector are the following:
  - HUMIDITY_OK
  - TEMPERATURE_OK
  - LIGHT_OK
  - HUMIDITY_ERROR
  - TEMPERATURE_ERROR
  - LIGHT_ERROR

COAP NODE

A CoAP node acts both as a client and server. It registers to the collector issuing a GET request to the server. In addition to this it exposes a sets of resources: humidity, temperature and light in order to notify the collector about state changes and to receive actuator commands. The telemetry related resources are observable.

The monitor exposes the following endpoints:

- /topic_name, holding the last value sampled by the corresponding sensor in the format specified above. For example, at the endpoint temperature the monitor will make available the latest measurement as {"n": "temperature", "v": "45", "u": "C", "id": "3"}. This endpoint only accepts GET requests;

- /alert, which is used to receive alarm messages from the collector. It accepts POST/PUT requests with a plain text payload containing the type of anomaly detected. For example value=3 means that the collector has detected a temperature measurement above the predefined threshold. The POST/PUT handler will then take care of changing the LED colors and notifying the relative processes. If value=2 is received from the collector it means that the temperature alarm can be deactivated because the measurements have gone back to normal. Value=1 means that the collector has detected a low humidity level. This issue can only be resolved manually.

COLLECTOR

The collector is responsible for interacting with both MQTT and CoAP monitors, saving the data in a MySQL database and detecting anomalies in the measurements.
To receive information from the MQTT monitors, the collector subscribes to the topics:

- /resoruce_name in order to receive telemetry data sent by all the monitors.

To send informations to the MQTT monitors, the collector publishes to the topics:

- /alarm_NODE_ID in order to notify the nodes when it detects an anomaly in the measurements.

The CoAP collector is deployed on the virtual machine outside the LLN, so the nodes need to register to the collector which is assigned a well known ip, in our case localhost. After a node registers, the collector sets up the observe relations to receive updates about the measurements made by the sensors belonging to that node. An observe relation is created by instantiating a CoapCollector class for each possible sensor. Each time the Collector receives a new sample, it checks if the values are below/above a certain threshold. If an anomaly is detected the collector sends a POST request to the /alarm endpoint specifying the type of error in the payload of the request.

DATA ENCODING

The exchanged data between sensors and the collector is encoded in JSON, in the form of JSON objects. The choice was lead by the fact that the sensors have constrained resources in terms of memory, battery and computation capability and XML has a too complex structure and interoperability is not required. This choice allows for a more lightweight communication, faster processing times and a simpler representation of the information since JSON is more flexible and less verbose, resulting in less overhead. Since JSON is a text-based format, probably a binary encoding like CBOR would have been better; unfortunately nowadays Contiki does not have any library that implements this type of encoding.

GRAFANA

For data visualization, Grafana was used to display most of the information presented above.

The structure of Grafana is divided into two sections, one for general control of all greenhouses and a section in which the data of the greenhouses can be viewed individually.

To visualize the status of each greenhouse, we created a Grafana dashboard.

One greenhouse per node, three monitoring panels for each node:

- the moisture panel shows the soil moisture trend. The moisture can only decrease, when it reaches the threshold and the operator activates the irrigation system, the moisture returns to normal.
- the sunlight panel shows the most recent measurement made by the sensor placed outside the greenhouse.
- the temperature panel shows the temperature trend. Whenever the temperature exceeds the threshold, you can see that subsequent measurements are reduced by the actuator until the temperature returns to normal.

An example is shown in the figure below:

Figure 3: Greenhouse Grafana dashboard

Regarding the general monitoring section: a graph for each measurement using data from all greenhouses over the last month. These graphs provide a broader view in which we monitor monthly water consumption for irrigation, ventilation and lighting system use.

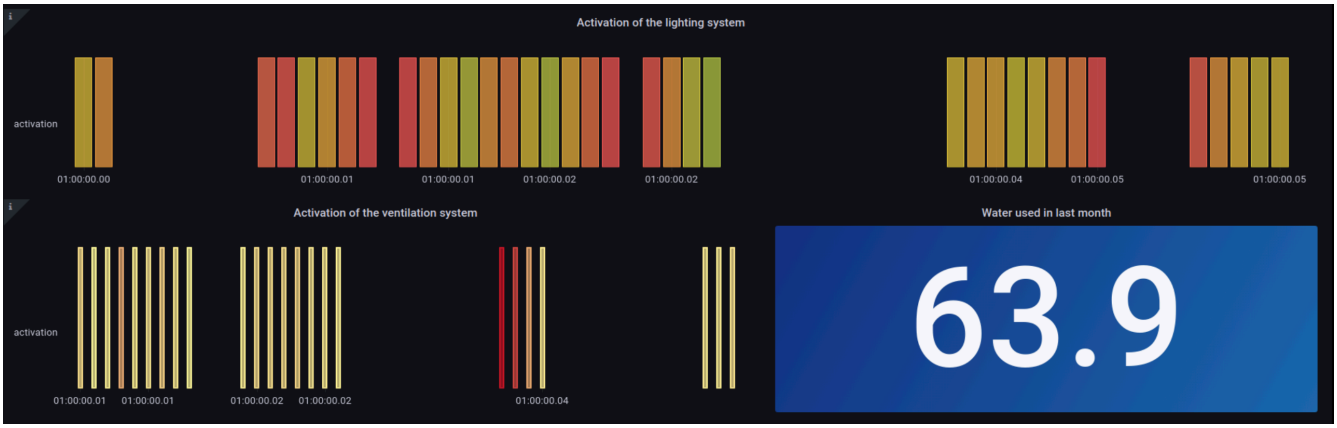


Figure 4: Monitoring center Grafana dashboard

TESTING

The dongles used are IoT devices without sensing capabilities, so we need to simulate the sensors using the C language. To simulate measurements, we start a process for each type of sensor: sunlight intensity, soil moisture, and temperature inside the greenhouse.

- For sunlight intensity, we created a function that randomly simulates the light outside the greenhouse, so the sensor makes measurements by generating a random value within a dynamic range.
- For soil moisture, initially the sensor generates a random value that is decremented with each new measurement.
- For the temperature inside the greenhouse, we created a function that simulates, with randomness, the temperature inside the greenhouse, then the sensor performs measurements by generating a random value within a dynamic range.

In addition, the alarm system is also simulated. For all measurements we define thresholds, when at least one of the measurements made by the sensor are below the threshold value, the red LED on the sensor is turned on and the corresponding actuator is activated.
Specifically:

- For soil moisture the MinThresholds is 40, when the measurement is below this threshold, the red led is turned on and remains so until an operator pushes the irrigation button. Once pressed the led is turned off (if there aren't other alert) and at the next measurement the soil moisture will have a value greater than the threshold.
- For the temperature inside the greenhouse the MaxThresholds is 30, the alarm is simulated by changing the color of the led to red. In addition,

actuator commands are simulated by changing the way temperature samples are generated. In fact, at each measurement, the temperature is reduced by 1 degree Celsius to simulate the return of the greenhouse to a normal temperature by a ventilation system.

- For sunlight intensity, the MinThresholds is equal to 10, the alarm is simulated by changing the color of the LED to red but unlike other measurements, the activation of the actuator, in this case the UV lights, does not affect the values of the next measurements. This is to be more efficient and use artificial light only when sunlight is not enough for rapid growth.

The collector is implemented in Java and contains both MQTT and CoAP collectors.

COOJA

The test is performed by deploying all the components:MQTT broker, Java collector, database; on the same Ubuntu virtual machine. The IoT system is simulated by creating a single LLN in which a border router (node 1), three CoAP nodes (nodes 2, 3 and 4) and two MQTT nodes (nodes 5 and 6) are instantiated as Cooja motes.

The default configuration is used:

- the broker listens on port 1883
- the CoAP collector is reachable at [fd00::1]:5683
- the database is reachable at localhost:3306
- new samples are generated every 10 seconds

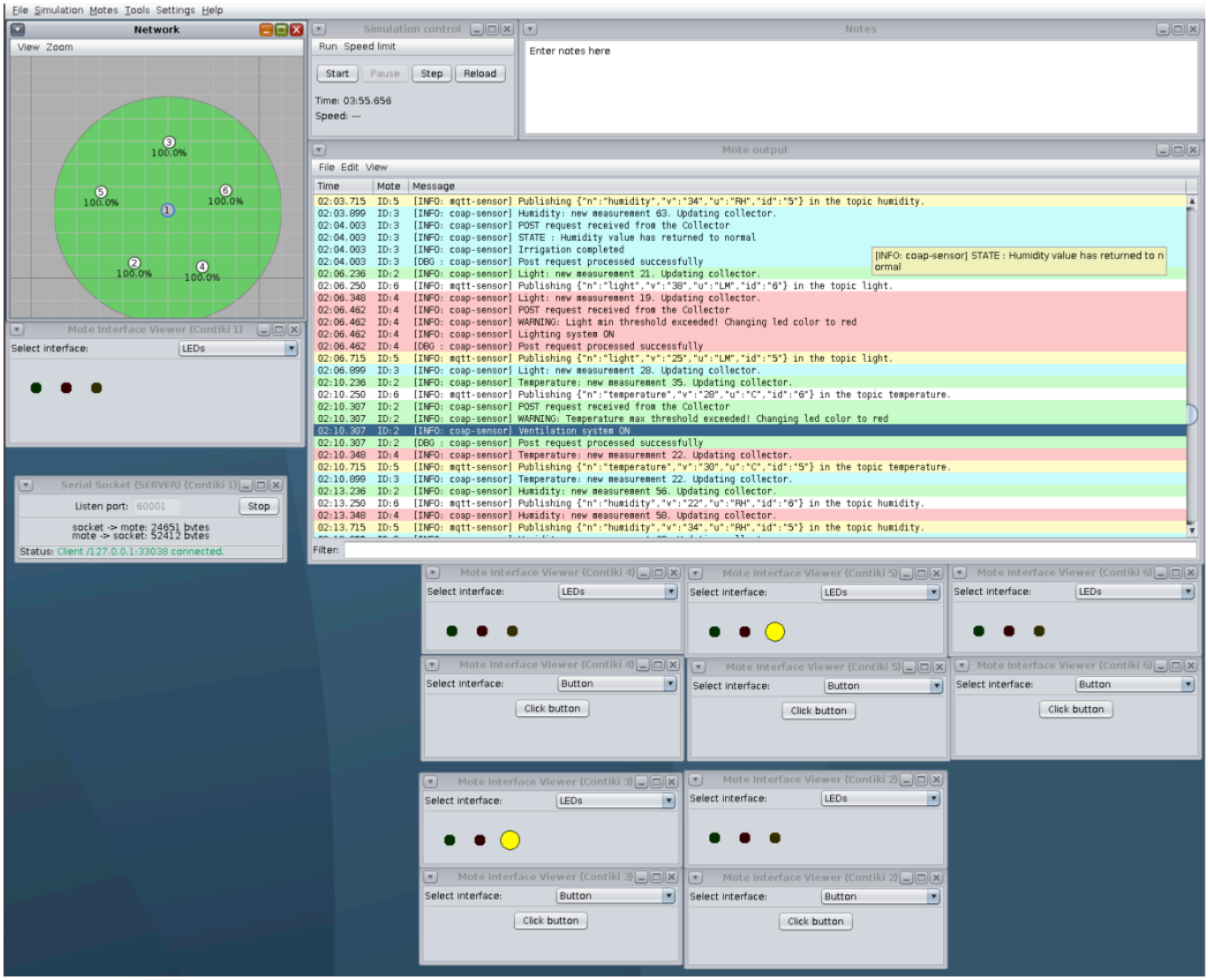


Figure 5: Simulation - Cooja

```
COAP - Machine: 2] value is 21LM measurement on light
MQTT - Machine: 6] value is 38LM measurement on light
COAP - Machine: 4] value is 19LM measurement on light
COAP - WARNING low sunlight detected, mim threshold exceeded!
COAP - State has changed, sending POST request to the node with state= LIGHT_ERROR to coap://[fd00:0:0:0:204:4:4:4]/alert
COAP - Changed status to state= LIGHT_ERROR on node with uri coap://[fd00:0:0:0:204:4:4:4]/alert
MQTT - Machine: 5] value is 25LM measurement on light
COAP - Machine: 3] value is 28LM measurement on light
COAP - Machine: 2] value is 35C measurement on temperature
COAP - WARNING high temperature detected, max threshold exceeded!
COAP - State has changed, sending POST request to the node with state= TEMPERATURE_ERROR to coap://[fd00:0:0:0:202:2:2:2]/alert
COAP - Changed status to state= TEMPERATURE_ERROR  on node with uri coap://[fd00:0:0:0:202:2:2:2]/alert
MQTT - Machine: 6] value is 28C measurement on temperature
COAP - Machine: 4] value is 22C measurement on temperature
MQTT - Machine: 5] value is 30C measurement on temperature
COAP - Machine: 3] value is 22C measurement on temperature
COAP - Machine: 2] value is 56RH measurement on humidity
MQTT - Machine: 6] value is 22RH measurement on humidity
COAP - Machine: 4] value is 58RH measurement on humidity
MQTT - Machine: 5] value is 34RH measurement on humidity
COAP - Machine: 3] value is 62RH measurement on humidity
COAP - Machine: 2] value is 22LM measurement on light
MQTT - Machine: 6] value is 39LM measurement on light
COAP - Machine: 4] value is 20LM measurement on light
COAP - Sunlight has returned to normal above the threshold
COAP - State has changed, sending POST request to the node with state= LIGHT_OK to coap://[fd00:0:0:0:204:4:4:4]/alert
COAP - Changed status to state= LIGHT_OK on node with uri coap://[fd00:0:0:0:204:4:4:4]/alert
MQTT - Machine: 5] value is 26LM measurement on light
COAP - Machine: 3] value is 27LM measurement on light
```

Figure 6: Simulation - Collector

Figure 7: Simulation - Humidity ERROR

Figure 8: Simulation - Humidity OK

MYSQL DATABASE

All the data produced by the sensors are stored in a MySql Database by the Java Collector. The database runs locally on the same VM where the collector runs. The database is named SmartGreenHouse and contains 3 tables, one table for each measurement where measurement ID, IDdevice, measurement, Unit , Timestamp are contained

```
-- Table structure for table `temperature`
DROP TABLE IF EXISTS `temperature`;
CREATE TABLE `temperature` (
                             `sampleid` int(11) NOT NULL AUTO_INCREMENT,
                             `sample` float NOT NULL,
                             `unit` varchar(45) NOT NULL,
                             `machineid` varchar(45) NOT NULL,
                             `timestamp` datetime DEFAULT CURRENT_TIMESTAMP,
                            PRIMARY KEY (`sampleid`)
) ENGINE=InnoDB AUTO_INCREMENT=4920 DEFAULT CHARSET=latin1;

-- Table structure for table `light`
DROP TABLE IF EXISTS `light`;
CREATE TABLE `light` (
                             `sampleid` int(11) NOT NULL AUTO_INCREMENT,
                             `sample` float NOT NULL,
                             `unit` varchar(45) NOT NULL,
                             `machineid` varchar(45) NOT NULL,
                             `timestamp` datetime DEFAULT CURRENT_TIMESTAMP,
                            PRIMARY KEY (`sampleid`)
) ENGINE=InnoDB AUTO_INCREMENT=4632 DEFAULT CHARSET=latin1;

-- Table structure for table `humidity
DROP TABLE IF EXISTS `humidity`;
CREATE TABLE `humidity` (
                             `sampleid` int(11) NOT NULL AUTO_INCREMENT,
                             `sample` float NOT NULL,
                             `unit` varchar(45) NOT NULL,
                             `machineid` varchar(45) NOT NULL,
                             `timestamp` datetime DEFAULT CURRENT_TIMESTAMP,
                            PRIMARY KEY (`sampleid`)
) ENGINE=InnoDB AUTO_INCREMENT=4724 DEFAULT CHARSET=latin1;
```

1) create database : CREATE DATABASE smartgreenhouse;
2) select database : USE smartgreenhouse;
3) import dumb : SOURCE smartgreenhouse_dump.sql;

COLLECTOR

This module of the monitoring and control system collects data from IoT devices. These are its main functionalities:

- Process the received data and apply some modifications to one or more actuators, executing a control logic
- Store received data in a MySql database
- Output data to the user via a textual log

SIMULATION

Before deploying the collector check that the Mosquitto MQTT broker is running. To deploy the collector move into the IoT_Project/collector folder and run the following commands:

```
mvn clean install
mvn package
java -jar target/collector.iot.unipi.it-0.0.1-SNAPSHOT.jar
```

Open the Cooja simulator and import simulation_SmartGreenHouse.csc

To deploy the border router:

- add the socket on the border router (Tools -> Serial Socket (SERVER) -> Contiki
- start the serial socket
- use Tunslip6

```
make TARGET=cooja connect-router-cooja
```

- Start the simulation

DEPLOYMENT

To deploy the system on the dongles, flash the code on all the nodes:

- Border router:

  make TARGET=nrf52840 BOARD=dongle border-router.dfu-upload PORT= devttyACM0

- MQTT nodes:

  make TARGET=cc26x0-cc13x0 BOARD=/launchpad/cc2650 PORT=/dev/ttyACM0 NODEID=0x000x mqtt-sensor.upload

- COAP nodes:

  make TARGET=cc26x0-cc13x0 BOARD=/launchpad/cc2650 PORT=/dev/ttyACM0 NODEID=0x000x coap-sensor.upload

- After that power on all the nodes and use Tunslip6 to connect with the border router:

  make TARGET=nrf52840 BOARD=dongle connect-router PORT=/dev/ttyACM0