UNIVERSITÀ DI PISA

ARTIFICIAL INTELLIGENCE AND DATA ENGINEERING

Internet of Things

PROJECT DOCUMENTATION

Design and development of SmartAgriculture: An IoT Telemetry and Control System

Student

LIU CHANG

Academic Year 2022/2023

INDEX

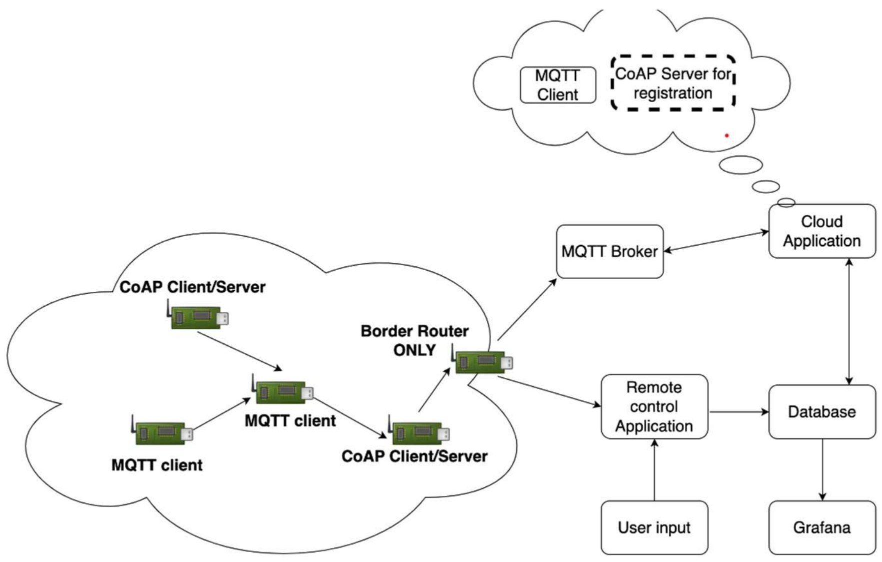
1 INTRODUCTION

With the increasing of global population and shortage of labor forces willing to engage in agricultural work, the need for an automation of management of control of agriculture camp is increasing, because we still need to be fed.

The goal of this project is to offer a smart solution for the management of an agriculture field, automating all the procedures through IoT technology.
The application must expose an CLI to manage the soil humidity and the soil temperature through the activation or deactivation of sprinklers.

2 ARCHITECTURES

The system architecture is shown in the following image.



The system is composed by a network of IoT devices that uses different protocols, where MQTT devices are used to report data, CoAP is used as the application protocol, and they are connected to the only one border router that allows to external access.
There are two types of sensors (humidity and temperature) deployed in the MQTT sensors, and one type of actuator (sprinkler) deployed as CoAP actuator.
The collector collect data from MQTT sensors and stores them in a MySQL database.
The collected data can then be visualized through a web-based interface developed by using Grafana.
A simple control logic is executed on the collector in order to enable or disable the actuators and modify the external environment based on the data that has been collected.

2.1 MQTT Devices

The two main aspect of an agriculture field are the soil humidity and the soil temperature, used to maintain the agricultural field in an optimal state.

2.1.1 Humidity

To manage the soil humidity, at least one humidity sensor and one actuator must be deployed on the field. If the sprinkler is turned on, the humidity will tend to rise, while if the sprinkler is turned off, the humidity will tend to drop. To simulate the real sensor's behavior, when the actuator is off there is still a certain probability that the humidity level will change.
The device that acts as a sensor communicates with the MQTT broker, sending the humidity samples and waiting for commands from the collector in order to maintain the consistence of data. The topic in which the humidity samples will be written is named "humidity", while the one to receive the commands that the actuator is actuating is "waterspurt".
The commands that can be received are the following:

- on : increasing the humidity level because of actuator is actuating.
- off : decreases the humidity level because of actuator is not actuating.

2.1.2 Temperature

To manage the soil temperature, at least one temperature sensor and one actuator must be deployed on the field. If the sprinkler is turned on, the temperature will tend to drop, while if the sprinkler is turned off, the temperature will tend to rise. To simulate the real sensor's behavior, when the actuator is off there is still a certain probability that the temperature level will change.
The device that acts as a sensor communicates with the MQTT broker, sending the temperature samples and waiting for commands from the collector in order to maintain the consistence of data. The topic in which the temperature samples will be written is named "temperature", while the one to receive the commands that the actuator is actuating is "waterspurt".
The commands that can be received are the following:

- on : decreases the humidity level because of actuator is actuating.
- off : increases the humidity level because of actuator is not actuating.

2.2 CoAP Devices

The main aspect of an agriculture field is to maintain it in an optimal state by using sprinklers.

2.2.1 Sprinklers

To manage the soil temperature and humidity at least one sprinkler needs to be deployed. The sprinklers are installed on the field in a strategic way, and they are smart, so it can be switched on and off remotely and it turns on the LEDs if it’s on and turns off the LEDs if it’s off.
Using data received by the temperature sensor and humidity sensor, the collector can assess whether or not it is necessary to turn on or off the sprinklers accordingly.
Through the command line interface, user users can switch on and off the sprinklers.
The commands that can be received are the following:

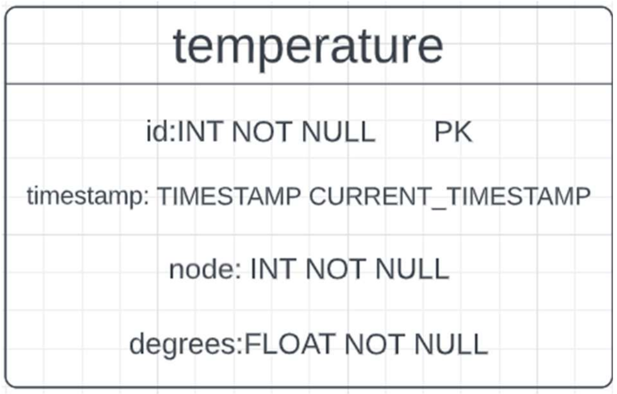
- on : switch on the actuator and LEDs.
- off : switch off the actuator and LEDs.

2.3 Data Encoding

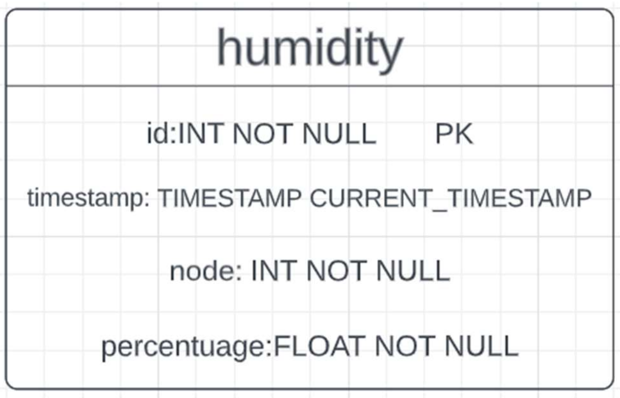
As data encoding, I decided to use JSON, all the data generated by the sensors are transmitted to the collector as JSON objects. This choice is due to the fact that sensors have limited resources, and XML has too complex structures for the application’s need. JSON is more flexible and less verbose, resulting in less overhead. JSON is a text-based format.

2.4 Database

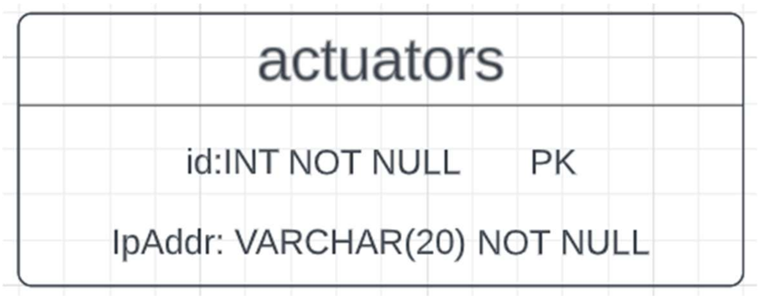
It is essential to store the data collected with sensor, also able to analyze them through Grafana. The database (SmartAgricultureDB) is particular simple, it’s composed by table for each type of measurement (temperature, humidity), there are no dependencies between tables.
For both humidity and temperature, we can have multiple devices, so in the tables it’s necessary to have a field that identifies the node from which we have received the sample.



temperature
id:INT NOT NULL PK
timestamp: TIMESTAMP CURRENT_TIMESTAMP
node: INT NOT NULL
degrees:FLOAT NOT NULL



humidity
id:INT NOT NULL PK
timestamp: TIMESTAMP CURRENT_TIMESTAMP
node: INT NOT NULL
percentuage:FLOAT NOT NULL



actuators
id:INT NOT NULL PK
IpAddr: VARCHAR(20) NOT NULL

2.5 Collector

The collector is the core of the architecture, it takes care of receiving the data and communicates with database in order to save the samples received from sensors and implements a CLI to allow the user to manage the System. I decided to implement the collector in Java by taking advantage of the Paho and Californium libraries.

2.5.1 CLI

The functions made available to the user through the CLI are the following:

- !help <command>: show the details of specific command.
- !get_humidity: gets the humidity level of agricultural field. If there are multiple humidity sensors, the global humidity level is calculated by the mean.
- !get_temperature: gets the temperature level of agricultural field. If there are multiple temperature sensors, the global temperature level is calculated by the mean.
- !spurt_the_water_on: activates the actuators on the agricultural field by let both sensor(because of data consistency) and actuator to know the actuator need to be turned on.
- !spurt_the_water_off: deactivates the actuators on the agricultural field by let both sensor(because of data consistency) and actuator to know the actuator need to be turned off.
- !exit: terminates the program.

2.5.2 Packages and Classes

For a better management of code, I have defined several packages, in this section there are the meaning of these packages and the classes they contain.

2.5.2.1 iot.unipi.it.SmartAgriculture.App
This package contains the main class, which starts the application.
Classes:

- SmartAgricultureCollector: This class implements all the functions of the CLI made available to the user.

2.5.2.2 iot.unipi.it.SmartAgriculture.coap
This package is used to manage the communication with the various IoT devices within the network whose the application protocol is CoAP.
Classes:

- CoapRegistrationHandler: This class extends the CoapServer class, an execution environment for CoAP Resources. In particular, this server has a resource of type CoapRegistrationResource. This is also the class that interfaces between the CLI and the classes related to each IoT device in the CoAP network. Specifically, it allows to translate each command received from the CLI into respective method.
    - CoapRegistrationResource is an inner class that extends CoapResource. This resource ("registration") allows the registration of IoT devices to the server via POST method request and the deregistration via a DELETE request.

2.5.2.2.1 iot.unipi.it.SmartAgriculture.coap.actuator
Classes:

- CoapActHandler: This class is used to manage the actuator.

2.5.2.2.2 iot.unipi.it.SmartAgriculture.coap.actuator.water
Classes:

- ActuatorCoap: This class handles the communication with IoT devices that are responsible for maintaining the soils humidity and temperature by using the control logic implemented, used to switch on or off the actuators.

2.5.2.3 iot.unipi.it.SmartAgriculture.log
In this package the logging functions have been developed, necessary both in the debugging phase and for the final application. Thanks to the log, it will be possible to understand the commands that have been given by the collector, and the reason.
Classes:

- Logger: This class defines the Logger object, which will be exploited by a good part of application modules. This class is also Singleton, this is useful to avoid shared access to the log file.

2.5.2.4 iot.unipi.it.SmartAgriculture.model
This package contains the classes required for the model. These classes are the Java bean for the application. The JSON objects that the application receives from the nodes will be transformed into these model objects.

Classes:

- HumidityModel: This class stores all the information about a sample received by the Humidity sensor, like the identifier of the node and the value observed.
- TemperatureModel: This class stores all the information about a sample received by the Temperaturesensor, like the identifier of the node and the value observed.

2.5.2.5 iot.unipi.it.SmartAgriculture.mqtt

This package contains the classes required to implement the MQTT client. Some sensors communicate with the collector through an MQTT broker, so it’s necessary to implement an MQTT client that can collect data and give the information about actuator’s state to ensure data consistency.

Classes:

- MQTTHandler: This class implements the connection with the MQTT broker, receiving the topic “humidity” and “temperature” messages, and sending messages to the topic “waterspurt” to ensure data consistency.

2.5.2.5.1 iot.unipi.it.SmartAgriculture.mqtt.sensors

This package contains one package for each type of device that communicates with the MQTT broker, because is needed some classes to storing the data received and decide what action implement.

2.5.2.5.1.1 iot.unipi.it.SmartAgriculture.mqtt.sensors.humidity

In this package there are the classes needed to store data received from humidity sensors.

Classes:

- HumidityCollector: This class keeps the most recent samples of each humidity sensor, in order to compute an average and decide what action to perform on the actuators.

2.5.2.5.1.2 iot.unipi.it.SmartAgriculture.mqtt.sensors.temperature

In this package there are the classes needed to store data received from temperature sensors.

Classes:

- TemperatureCollector: This class keeps the most recent samples of each temperature sensor, in order to compute an average and decide what action to perform on the actuators.

2.5.2.6 iot.unipi.it.SmartAgriculture.persistence

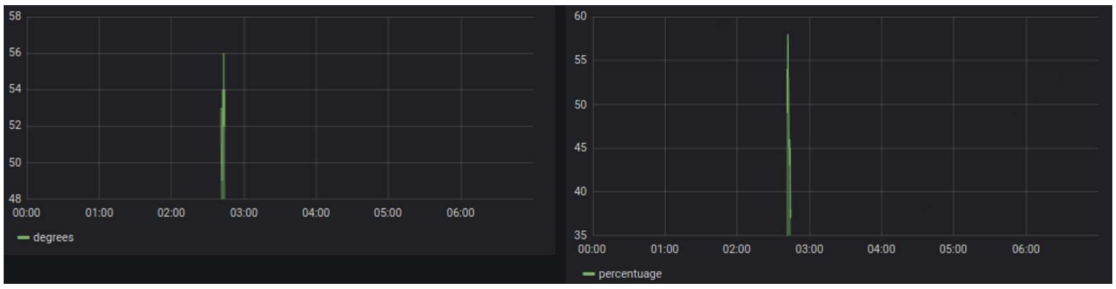
This package deals with the management of the persistency of data, indeed it contains the class used to interface with the database.

Classes:

- DBDriver: This class is responsible for the implementation of all the queries for MySQL. We used the Singleton design pattern, because a single instance of this driver must be shared by all application classes.

2.6 Grafana

The dashboard is implemented on Grafana in order to able to monitor in real time the data that’s stored in the database, and therefore to be able to view the trend of the monitored parameters.
More precisely, it’s created a panel for humidity and for a panel for temperature.
Through this dashboard you can see how the measures values remain within the established ranges.



In this example there are two sensors to monitoring the temperature and the humidity.