UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Internet of Things

IoT Smart Irrigation System

Project Documentation

TEAM MEMBERS:
Edoardo Fazzari
Mirco Ramo

Academic Year: 2020/2021

Contents

1 — Introduction

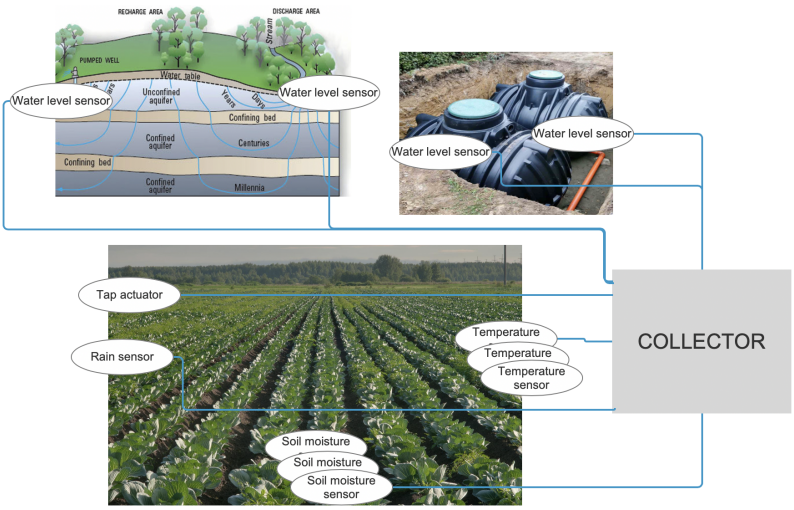
Agriculture is one of the most fundamental resource of food production and also plays a vital role in keeping the economy running of every nation by contributing to the Gross Domestic Production. But there are several issues related to traditional methods of agriculture such as excessive wastage of water during field irrigation, dependency on non-renewable power source, time, money, human resource etc. Since every activity now a days is becoming smart, it needs to smartly develop agriculture sector for growth of country. Out project aims at developing a Smart Irrigation System using IoT Technology with the objective of automating the total job, providing adequate water required by crop by monitoring the moisture of soil and climate condition in order to prevent the wastage of water resources, resulting in many advantages for farmers. The irrigation at remote location from home will become easy and more comfortable. In addition, it will not only protect farmers from scorching heat and severe cold but also save their time otherwise required by the to-and-from journey to the field.

1.1 Deployment Structure

The objective of this project is to adopt a smart irrigation system to water cultivated fields making use of the local water resources, such as aquifers and reservoirs, in the way of using the water resources as more eco-friendly as possible (e.g., without disrupting aquifers). Thus, we can consider two different locations to takes care: the **field** and the **water provisioning site**,

For what concern the *water provisioning site*, it is composed of sensors which have the mean of monitoring the water level both of the aquifers and reservoirs. In this way, the system and the user can know where to take water (by default the aquifer). Although, a single device for source may be enough, we decided to deploy multiple water level sensors in the same source in order to avoid errors in the monitoring (e.g., in the case of a single device if it is detected that the reservoir is empty, but it is not, the irrigation system will use the water from the aquifer pointlessly). All water level sensors will make use of *MQTT* as explained in the "MQTT Network" chapter.

The *field site* is more articulated and will exploit multiple types of sensors and a single type of actuator. The actuator needed is the one capable of providing the water to the plants, which we called "tap actuator". This will be used in conjunction with the other sensors presents in the fields that will monitor the environment, specifically there will be temperature sensors, soil moisture sensors and a rain sensor. All these sensors and the tap actuator will exploit *CoAP* as explained in the "CoAP Network" chapter.



2 — CoAP Network

AAa

2.1 Temperature Sensor

The temperature sensor measures the local temperature in Celsius (at the Collector level will be given the possibility to display the results in Fahrenheit, see the Collector chapter for further details). The goal of this sensors is to quantify and schedule the water provisioning.

2.1.1 Resources

The temperature sensor exposes two resources: the *temperature_sensor* and the *temperature_switch* resources.

The **temperature_sensor** resource is an observable resource that provides to the clients the temperature acquired by the sensor. The resource not only provides the mere temperature to the clients, but it informs if the temperature is lower or greater than a certain threshold. Hence, the sensor exposes a *PUT* method, in order to set up the lower or the upper bound for the temperature.

The change of the bounds is done at step, the user will specify the threshold that he/she wants to change through the CLI: upper or lower. At the server side the request will be processed checking if the value arrived is consistent (e.g., the new value for the lower bound is not greater than the upper bound actual value), after those controls the parameter is updated.

The **temperature_switch** resource is connected to the *isActive* boolean variable, which indicates if the sensor is operating or not. This is done for turning off the temperature sensor when it is raining in order to save energy, since we do not perform any analysis for irrigating when the weather does that for us. For the reason that we want to change the status of the resources based on the rain sensor, it is implemented a *PUT* method for changing the value of the *isActive* variable.

2.1.2 Data Generation

Data is generated every *CLOCK_SECOND* in order to have a rapid simulation. The value for the temperature is updated using the following algorithm:

```
static void temperature_event_handler(void)
{
    if (!isActive) {
        return; // DOES NOTHING SINCE IT IS TURNED OFF
    }

    // extimate new temperature
    srand(time(NULL));
    int new_temp;
    int random = rand() % 4; // generate 0, 1, 2, 3

    if (random == 0) // 25% of changing the value
        if (random < 2) // decrease
            new_temp -= VARIATION;
        else // increase
            new_temp += VARIATION;

    // if not equal
    if (new_temp != temperature)
    {
        temperature = new_temp;
```

```
        coap_notify_observers(&temperature_sensor);
    }
}
```

2.2 Soil Moisture Sensor

Soil moisture sensors measure the water content in the soil and can be used to estimate the amount of stored water in the soil horizon. Soil moisture sensors do not measure water in the soil directly. Instead, they measure changes in some other soil property that is related to water content in a predictable way. Checking the different technologies used for measure soil moisture content, we decide to exploit the *soil water potential*[1].

2.2.1 Resources

The soil moisture sensor exposes two resources: the *soil_moisture_sensor* and the *soil_moisture_switch* resources.

The **soil_moisture_sensor** resource is an observable resource that provides to the clients the soil moisture tension acquired by the sensor. The resource not only provides the mere tension to the clients, but it informs if the value is lower or greater than a certain threshold. Hence, the sensor exposes a *PUT* method, in order to set up the lower or the upper bound for the tension[2].

The change of the bounds is done at step, the user will specify the threshold that he/she wants to change through the CLI: upper or lower. At the server side the request will be processed checking if the value arrived is consistent (e.g., the new value for the lower bound is not greater than the upper bound actual value), after those controls the parameter is updated.

The **soil_moisture_switch** resource is connected to the *isActive* boolean variable, which indicates if the sensor is operating or not. This is done for turning off the temperature sensor when it is raining in order to save energy, since we do not perform any analysis for irrigating when the weather does that for us. For the reason that we want to change the status of the resources based on the rain sensor, it is implemented a *PUT* method for changing the value of the *isActive* variable.

2.2.2 Data Generation

Data is generated every *CLOCK_SECOND* in order to have a rapid simulation. The value for the tension is updated using the following algorithm (the same of to the one used for the temperature):

```
static void soil_moisture_event_handler(void)
{
    if (!isActive) {
        return; // DOES NOTHING SINCE IT IS TURNED OFF
    }

    // extimate new tension
    srand(time(NULL));
    int new_soilTension = soilTension;
    int random = rand() % 4; // generate 0, 1, 2, 3

    if (random == 0) // 25% of changing the value
        if (random < 2) // decrease
```

[1] *Soil water potential* or *soil moisture tension* is a measurement of how tightly water clings to the soil and is expressed in units of pressure called bars. Generally, the drier the soil, the greater the soil water potential and the harder a plant must work to draw water from the soil.

[2] For the default range value we used the ones indicated here: https://www.metergroup.com/environment/articles/defining-water-potential/

```
            new_soilTension -= VARIATION;
        else // increase
            new_soilTension += VARIATION;

    // if not equal
    if (new_soilTension != soilTension)
    {
        soilTension = new_soilTension;
        coap_notify_observers(&soil_moisture_sensor);
    }
}
```

2.3 Rain Sensor

Rain sensor or rain switch is a switching device activated by rainfall. It is used for water conservation since it is connected to the automatic irrigation system, which will cause the system to shut down in the event of rainfall in order to do not waste water and to reduce energy consumption.

2.3.1 Resource

The only resource provided by the rain sensor is a value indicating if it is raining or not, named **isRaining** and stored as a boolean. Since we are only interested when the status of the variable change, we opt to use the observable pattern provided by CoAP in order to minimize the number of interactions with the sensor.

The only possible action is the **GET** method, which will respond with a text saying *"raining"* or *"not raining"* based on the status of *isRaining*.

2.3.2 Data Generation

Data is generated every *CLOCK_SECOND* in order to have a rapid simulation. The value of **isRaining** flips (i.e., if it was indicating raining it turns to not raining, and vice versa) with a probability of 10%. This is done in the *rain_event_handler* function in the following way:

```
static void rain_event_handler(void)
{
    // check if raining
    srand(time(NULL));
    int random = rand() % 10; // generate 0, 1, ..., 9

    bool new_isRaining = isRaining;
    if (random == 0) // 10% of changing the value
        new_isRaining = !isRaining;

    // if not equal, notify
    if (new_isRaining != isRaining) {
        isRaining = new_isRaining;
        coap_notify_observers(&rain_sensor);
    }
}
```

In case the value changes, this is notified to all the subscribers.

2.4 Tap Actuator

The tap actuator is the device aim at irrigating the fields.

2.4.1 Resources

The tap actuator exposes four resources: the *tap_intensity*, the *tap_interval*, *tap_where_water* and the *tap_switch*.

The **tap_intensity** resource is related to the *intensity* variable, which indicates the volume of water to provide at each simulation. Since we want to pull water from the aquifer or the reservoir, we made this resource observable in the way of better controlling the *water level sensors* in the simulation. The method implemented for this resource are the *GET* and *PUT* methods. The *GET* method respond to the user with a string containing the value of the intensity and, separated by a space, a character value indicating if the water is taken from the aquifer ('A') or the reservoir('R'). On the other hand, the *PUT* method gives to the user the possibility to set up the intensity value updating the current one.

The **tap_interval** resource indicates the period of time (expressed in *CLOCK_SECOND*) between two activations of the tap. This resource can be retrieved using the *GET* method associated to it, and it can be updated using the *PUT* method. The *PUT* method does not only update the value, but it updates also the intensity based on the following formula.

$$\text{intensity} = \text{intensity} * \frac{\text{new\_interval\_value}}{\text{old\_interval\_value}} \tag{1}$$

The **tap_where_water** resource is used by the device in order to decide where to take the water for every irrigation activity. Thus, the resource provide a *PUT* method for changing this value and, for sake of control, we implemented the possibility to retrieve the value through a *GET* method.

The **tap_switch** resource works in the same way of the **temperature_switch** and **soil_moisture_switch** resources. The meaning of use this type of resource is for not irrigating when raining since it would be a waste.

3 — MQTT Network

4 — Collector

4.1 MQTT Side

4.2 CoAP Side

4.3 Database And Data Visualization