AMBIENT INTELLIGENCE

SMART IRRIGATION

November 28, 2014

Students:

Samuel Coelho (69350)

João Jerónimo (62521)

Instituto Superior Técnico

DEI

Contents

1 INTRODUCTION

When you have plants, whether you have a huge piece of land or just have a small vase with a flower at home, you will have to solve a problem. You will have to water it based in some environment conditions, such as, temperature and moisture. But, it's very likely that you are not a farmer. You leave home in the morning and come back, tired, to have dinner and spend a while with your family. You don't have time or, you just forget about watering your plants and they just die. If you really like your plants, you will feel really bad about it. Also, most times you just don't know when you have to water it and if you need to move it to another place because, the current place it's too cold or too hot for that particular plant. You just give up about having a nice smelling flower or tomatoes to make a good salad. Or, you can just become a farmer and take care of everything.

Today, Do It Yourself (DIY) movement is becoming very popular. Now, everyone has access to technologies that allow to build systems to automate a lot of different tasks. It would be possible to build a system that would monitor the moisture level and the temperature and, taking into account, those levels, would water the plant or not. To make it even better, we could build some kind of application where the user have access to a User Interface, which could give the user, some information, such as, if the plant is being watered and if the temperature is right for that kind of plant. Despite of the availability of DYI technologies, they still require technical skills, which not everyone has.

We designed a solution, and built a prototype, based on those popular technologies. Our solution aims to be parameterized for each kind of plant. Someone specifies the right environment conditions and the system would water the plant or warn the user to change it to another place if the temperature is not adequate.

This report is about our solution, Smart Irrigation, that monitors the moisture level and temperature based on a given configuration and waters the plant when it is needed. Also giving the user an interface, where he can see the current state of his plant, create one or more configurations, share them and download configurations provided by other users around the world. Someone who knows, very well, the right conditions for a given plant, can share his knowledge about it, creating some configuration, and another user could just get that configuration and use it. The system will actuate according to it. Next section, 2 is about some related work, from which we got some ideas and concepts to build our solution. In section 3 we explain our solution and how it solves the problem. In section 4 we explain

what technologies we used and how our solution was implemented. Section 5 is about the limitations of our solution. Finally, in 6 we conclude the report.

2 RELATED WORK

This section is about related work and for each one, we describe it's benefits, it's limitations and, at the same time, we provide motivation to develop our solution.

DomoBus: DomoBus [1] is a system for applications in domotics. The idea is to have a system with a set of properties, in a given home, defined in a simple XML file. In that file, we define the devices, the properties of each device, and more. Since the main goal of this system is to connect differents kinds of devices, in a particular home and there are a lot of heterogeneous technologies, we need to have a gateway for each technology. Our solution uses a similar approach. We have properties, such as, current moisture level, current temperature, maximum level of moisture, minumum level of moisture, maximum temperature and minimum temperature. It's possible to get and set, at any time, any of these properties.

Code provided by the professor: In the beginning of the project, the professor provided us some C code, that would work in any microcontroller and follow an approach based on state machines. Since, we would need a microcontroller and make it communicate with a computer, it would be a good idea to worry about the blocking communication. Communication between the microcontroller and the computer can take a long time, depending on the number of bytes that are being transmitted. This code included some features, such as a model of tasks. Each task was a state machine, where, in each iteration of the main loop, a small step of the entire computation was executed. In the case of the communication, in each iteration, only a single byte was transmitted and not the entire message at once. This approach have impact on performance because the entire program does not block until the message is completely transmitted. This code aimed to work in any microcontroller we want. The downside is that we need to write bitwise operations to perform simple tasks, such as writing a value in a given output pin or read a value from a given pin. To work with this code, for each operation we need to look at the microcontroller's datasheet to check which bitwise operation we need to perform to do it.

The main benefit of this code is that it could work in any kind of microcontroller. Unfortunately, to take advantage of this generic approach, without being attached to a specific board, we would have a slow development process since we would spend a lot of time looking to the board's datasheet instead of writing the code for the application

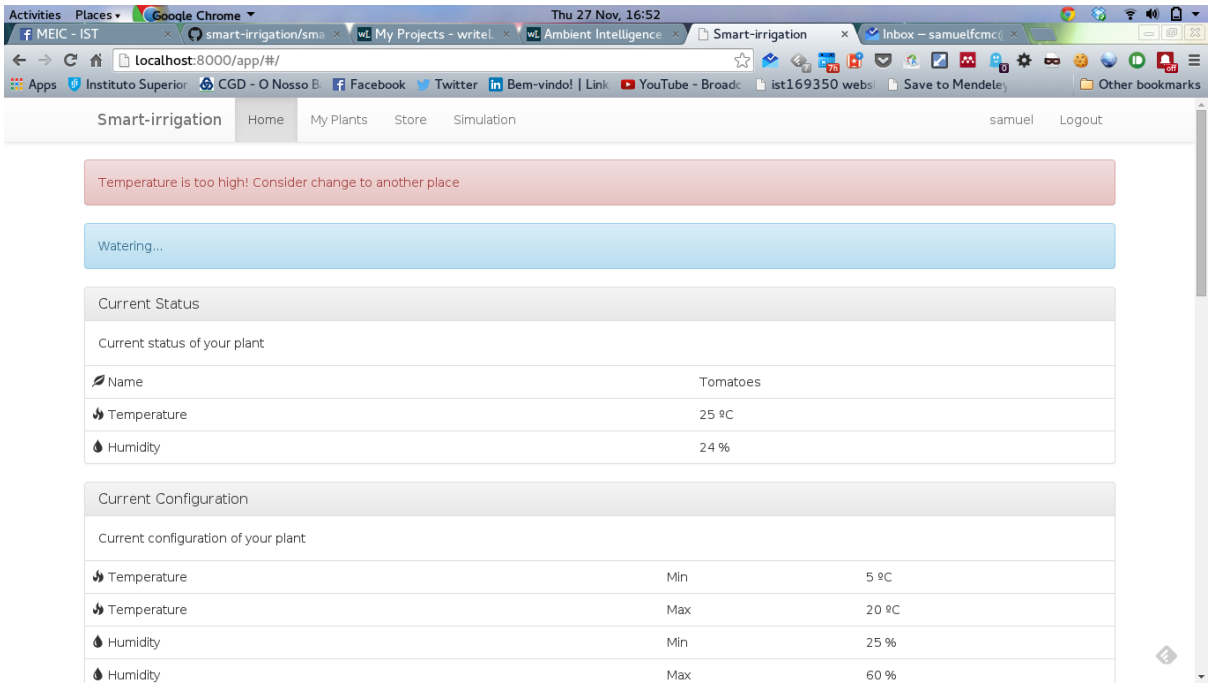
logic. We choose to use an Arduino board, which allows to write the code, that will run there, in C++. Also, that board provides a really simple API. We decided to apply the approach, followed in the code provided by the professor, using the Arduino API. We used a model of tasks and we implemented non-blocking serial port communication. In each iteration of the main loop, only a single byte is transmitted. The downside is that our code will run on Arduino or Arduino-compatible boards. Since there are a lot of Arduino-compatible boards and they are becoming popular, we decided that this trade-off, between reducing the range of boards where it could run and writing a more simple, more maintenable, easier to read and object-oriented code, would be acceptable.

Arduino: Arduino [3] is a programmable microcontroller that provides a simple development environment where it is easy to write code, compile it and upload it to the board through a USB connection. This USB connection can be used, also, to send and receive data from a computer. It has a huge community and there are a lot of Arduino-compatible boards. Also, it is easy to find support for it. We don't need to write the code in the development environment it provides. We can simply write C++ code in any text editor and use a make file to compile it and upload it to the board. In our solution we used a package, for Linux, that includes a makefile and only needs overriding two variables, which are, the serial port where the Arduino is connected and the names of libraries we used.

3 SOLUTION

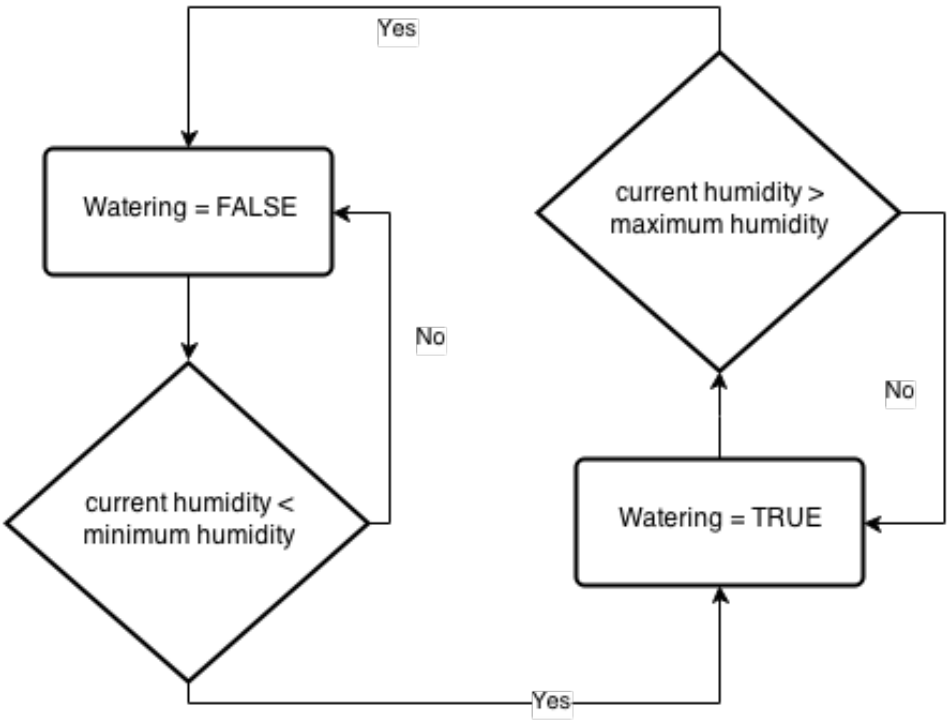
Our solution to automatically irrigate a plant has software and hardware. In terms of hardware, we have potentiometers to simulate moisture and temperature sensors. The actuator, which would be the similar to a tap, in our prototype, is simulated using a LED light. When it is time to water the plant, the LED is on. When it should stop watering, the LED simply turns off. This sensors and the actuator are connected to a microcontroller, where some code is running in an infinite loop, reading values from the simulated sensors and, based on those values, turning on or off the actuator, as is shown in figure 2. We also have an application, running on a PC, where we can monitor the current temperature and moisture levels. Besides that, the application shows a warning if the current temperature is below the minimum temperature or is above the maximum temperature, required for that plant. Figure 1 shows the user interface of the application when the plant is being watered and the temperature is not between the values required.

Figure 1: Screenshot of the application when the plant is being watered and temperature is not right



We followed an approach similar to DomoBus [1]. Our system has a set of properties, current moisture level, current temperature, minimum and maximum values of temperature and moisture levels. It is possible to get the values or set the values of this properties. The application in the PC can communicate with the microcontroller, through a USB connection.

Figure 2: Main loop that runs in the microcontroller

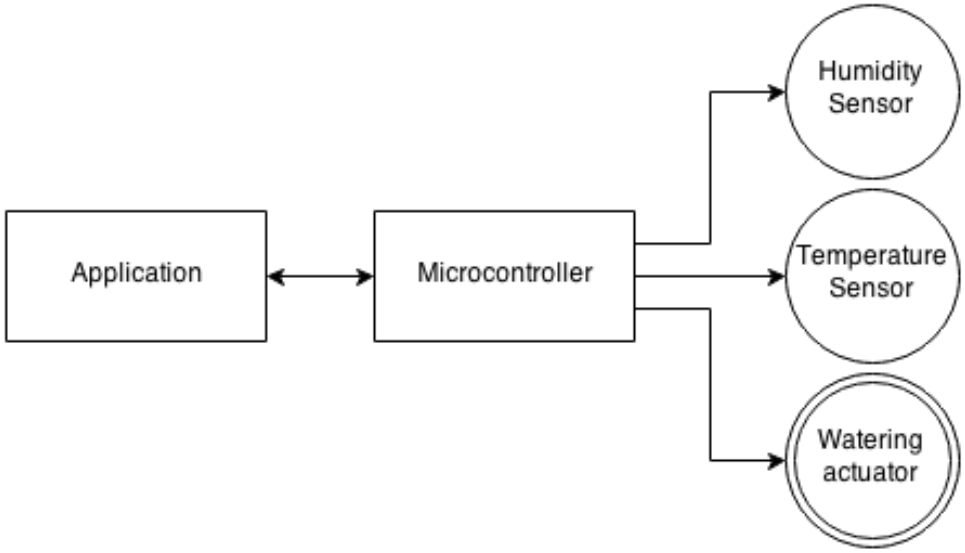


The application can get and set the values of the properties, stored in the microcontroller. To do that, there is a communication protocol, explained in 4, that we created, between the PC and the microcontroller. The current temperature and moisture can be set through the potenciometers or using the application itself. It's possible to simulate every kind of environment. The communication flow between all components in the system is ilustrated in figure 3.

3.1 Application

The application that runs on the PC has some features. One of them is the possibility to send any configuration to the microcontroller. A configuration is the set of minimum and maximum values for moisture and temperature. It is also possible to get the current status of the system. The status is the current temperature and moisture and the configuration that is being used. The application asks for these values, to the microcontroller, and checks if the temperature is below the minimum temperature or above the maximum temperature. If it is below, an alert appears on the screen, warning the user. The same happens when the

Figure 3: Main components of Smart Irrigation and interactions between them



temperature is above the maximum. The user is advised to move the plant to another place. There is also a store, in the application, that allow users to share their configurations and get configurations from the other users and send them to the microcontroller. In our application, a plant can have more than one configuration. Each configuration has the minimum and maximum values for the moisture and temperature and, also, has the number of days after this conditions apply, since we planted that particular plant. Each plant has its own ideal conditions but, those conditions will change over time. However, when we want to tell, the microcontroller, which plant we have, we need to specify how many days have passed. Based on this value, the application will pick the right configuration for that moment and send it to the microcontroller.

4 IMPLEMENTATION

4.1 Overview

Our solution encompasses an Arduino microcontroller and a host. The host serves simultaneously as a communication base with which the Arduino communicates, and as a webserver to control the gadget. The host parses all the text inputs for our system and send them to the Arduino in binary form. The microcontroller does not deal with textual data.

There exists a host⟷microcontroller protocol. This is a command-oriented protocol, clearly inspired by DomoBus data model, in which 2 operations are defined: GET and SET. Each one has it's own *opcode*, which identifies the operation that is to be performed. The commands are sent in very simple chunks of data, in which message size is defined by the *opcode* in the first byte. Table 1 shows the messages' structure.

OPCODE	REGISTER	VALUE
GET (0x01)	reg	—
SET (0x02)	reg	val

Table 1: Message Structure description

As can bee seen, GET message doesn't need a VALUE field. Instead, the response message will send the corresponding VALUE back. REGISTERs are:

CURRENT_TEMPERATURE Stores the last temperature level read from sensor.

CURRENT_HUMIDITY Stores the last moisture level read from sensor.

TEMPERATURE_MIN Minimum required temperature for plant.

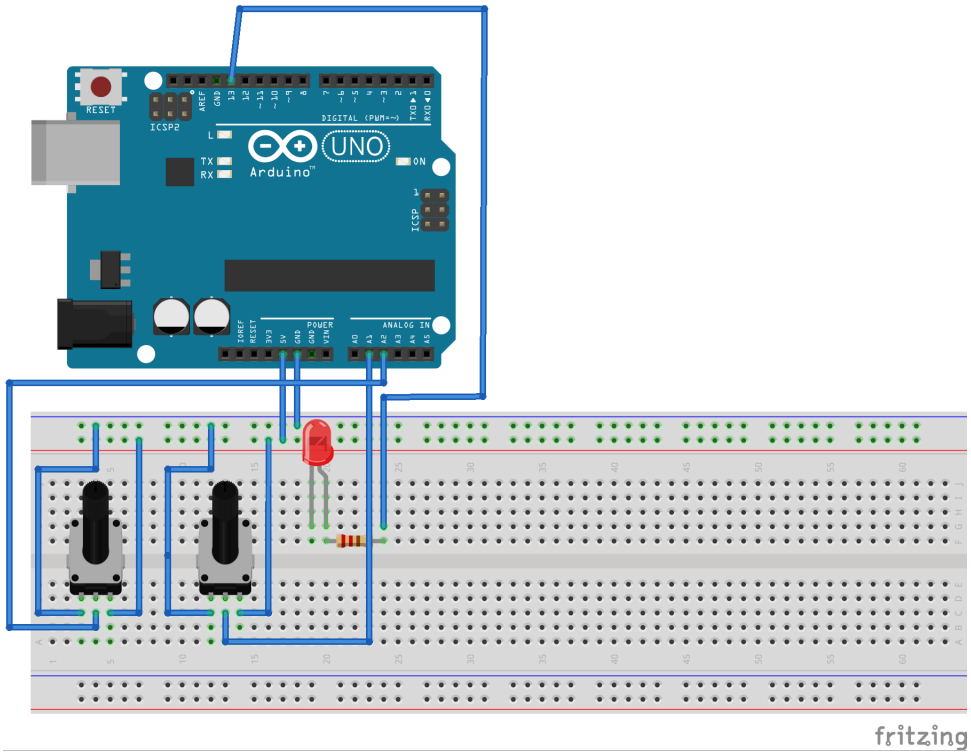
TEMPERATURE_MAX Maximum required temperature for plant.

HUMIDITY_MIN Minimum moisture threshold to start irrigation.

HUMIDITY_MAX Maximum moisture threshold to stop irrigation.

Messages are always sent by host to the Arduino, with reponse (possibly ignored) from the Arduino, and never backwards.

Figure 4: Diagram with the pratical implementation



4.2 Arduino

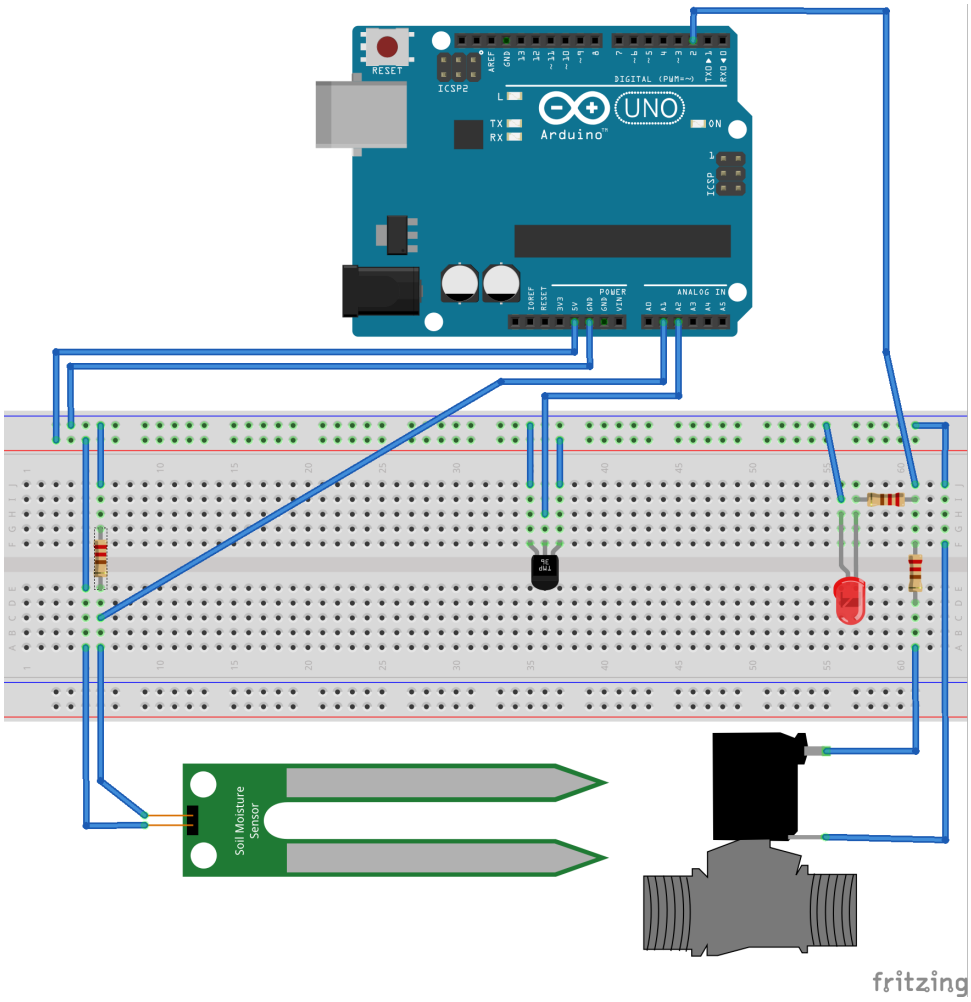
An Arduino [3] microcontroller is used to control some simple resistive sensors and a solenoid valve. In pratical implementation and testing, we didn't use "real" components, but some potentiometers and LEDs to simulate them instead as it is shown in figure 4.

There is a (resistive) moisture sensor and a temperature sensor, which are connected to analog inputs (A1 and A2 respectively) of the Arduino. There's also a solenoid valve, connected to the digital input number 2.

As for the code running in the microcontroller, we are using the ArduinoThread [2] library, which enables the use of a form of cooperative multitasking. Complex operations can be implemented by applying a State Machine design pattern. We try to apply Object Oriented principles where they are useful, without forgetting the low-level nature of the project.

There is a thread for communication (*CommThread*) with the host. This is somewhat semantically-agnostic, in the sense that messages are sent as byte streams. This thread's class uses a callback function that is executed when a message is received, to interpret it. Messages

Figure 5: Electronic diagram for the SmartIrrigation gadget



are next processed by *Configuration*. *Configuration* class may store the received value or return the existing value to *CommThread*, which sends it back to the host. The *Configuration* class is also storing some values that are not supposed to by written to by the host, as the CURRENT_HUMIDITY and the CURRENT_TEMPERATURE registers.

IrrigationThread contains the core logic for the gadget, repeatedly testing reading the moisture and temperature sensors, and acting on the solenoid-valve based on these levels. If CURRENT_HUMIDITY falls below HUMIDITY_MIN, the valve is opened, and when it grows above HUMIDITY_MAX, the valve is closed. This use of a pair of threshold values is necessary so that we have a definite operation for the valve. As the moisture in theory starts to grow in the moment that the valve is open, and starts to fall in the exact moment the valve is closed, the irrigation time would otherwise be determined not by the defined levels but by the speed of the hardware, which is not what is supposed to occur.

AnalogSensor class is just a generic "driver" for sensors that connect to the analog pins of the Arduino, performing needed scale convertions (but only when the read value changes). It is subclassed by HumiditySensor and TemperatureSensor classes.

4.3 Web Application

As we explained in 3, an application running on a PC, is part of the solution. But, instead of implementing an usual desktop application we decided to create a web application. We thought it was the best option, because, that way, it would be easier to show the store feature and other online features, such as, each user could create his own configurations and would have a list of configurations, created by him or "downloaded" from the store.

4.3.1 Web Framework

To develop it, we used a framework, for python, which is Django [4]. We choose it because, we know python and it is really easy to use the framework. It included everything that is needed for any web application. It also provides user authentication. It uses the Model-View-Controller pattern. We have an abstraction layer for the persistent storage mechanism. We do not need to write SQL queries. We just create and update Models. Those operations are performed in the database but we just have to worry about objects with properties.

4.3.2 Building the user interface

To have a nice looking user interface without spending too much time on it, we used a framework, which is Bootstrap [5], that provides lots of components, such as buttons, some icons, and much more. Also, this framework gives, to the application, a responsive design. This means that, the application could be used in mobile devices, with smaller screens, without the need of re-design.

4.3.3 Client-side framework

In most web applications, the browser makes a request to the server. Then, the server does some computation and returns a response that contains a HTML file and the entire page is refreshed. We decided to not follow that classic approach. Instead, we used a javascript client-side framework, AngularJS [6], created by Google, which allows developers, to build web applications without the need of the server to do everything. As a result, our web application do not need to refresh the entire page each time, some computation needs to be done on the server. We implemented some endpoints, on the server, that just return JSON objects instead of a full HTML page. Using this framework, we just needed to create static html files and, using a special syntax, reference some variables, on them. This variables are updated according to the responses of the endpoints, that are invoked, in the background, and return JSON objects. As a result, the user do not see the entire page refreshing. Also, this framework, allowed us to define that some markup will appear, or not, based on some conditions. For instance, if a user is seeing a given plant configuration and he is not the owner of it, he should not be allowed to edit it. Each time the user requests to see a configuration, in the response, there is information saying that if the user, who is logged in, is the owner or not. If not, the form inputs in the page to see a configuration, should be disabled. Disabling them is done in client-side. However the server checks, always, if the user is the owner if he tries to edit some configuration, because, the user can mess with client-side code.

4.3.4 Client-side dependencies management

As in any application, we did not implement everything from the scratch. In terms of client-side libraries, we used, at least two, Boostrap, explained in 4.3.2, and AngularJS, in 4.3.3. Each of this libraries, have their own javascript, CSS files and others. Also, Bootstrap (4.3.2) depends on another library, which is jQuery [7], that is a library to make html elements manipulation easier. We could download each one and include the files we needed in our application. Instead of managing this dependencies manually, we used a tool, called Bower [8]. Bower is a tool, written in NodeJS, to manage client-side dependencies. We need a file where all dependencies are specified. Then, we run a command and the dependencies are downloaded and installed. After this step, we just need to include the files we need in our HTML markup. We choose to manage this dependencies this way to avoid having this libraries in our repository and they are a lot easier to mantain than doing it manually.

4.3.5 Irrigation Control Module

We created the Irrigation Control Module, to abstract the communication through the USB port between the computer and the Arduino (4.2). Despite of this module being part of the web application, it has its own file. It was done this way to allow a better separation of concerns. This module needs to receive the name of the USB port, and it provides functions to get information about the properties that are stored in the Arduino. For instance, it has a function to get an object that represents the current state of the system, which means, the current temperature and moisture and the current configuration that is being used. That function would take care of sending the right bytes, to the Arduino, that is running code that handles received bytes through the USB port. After all the needed communication, this function returns an object that represents the current state of the system. The function, on the server, that called this function, gets this object and put it in the response to the client. This separation of concerns could allow us to replace the Arduino, by another device with more resources, for instance a Raspberry PI [10], and move this module to that platform, without the need of reimplementing the entire application logic again. Also, this could be used with some kind of communication, for instance, with a smartphone, to allow the user to interact with the system using an app in his smartphone, instead of having to run a web application and connect the microcontroller through the USB port.

5 LIMITATIONS

Our implementation is prone to two kinds of limitations. Hardware errors and software errors.

First, as hardware limitations, there could be a problem with the correct positioning of the sensors (a problem common to every sensor-dependent device). This problem can manifest itself in two ways: (1) the moisture level may stall in the same value in spite of the water being poured in the plant, or (2) the moisture level may grow two fast, unrelatedly to the irrigation really provided to the plant.

In the first case, one has put the moisture sensor too far from the valve's tube. In the second case, the valve's tube and the moisture sensor are simply too close. However, this can be (possibly) turned into a non-problem if the vase is sold or otherwise provided as a complete unit with the gadget already attached to it.

Also interesting to this analysis is the event of the system running out of water. In this case, the valve may open with no effect, and the Arduino senses no growth in the moisture level. This condition coincides with the first case named above. The user should be notified if that happens.

Another possible class of limitations would be the software ones, where the relevant delays that are implemented may cause some lags. For example, the sensors are read with an interval of 5 seconds, and while this delay passes there could be poured too much water in the plant.

Another software limitation is the inability to control several vases with the same Arduino microcontroller. Because of this, the application of the product might become expensive.

The time of the day is not considered when it is to decide whether to start irrigating. This may lead to water wasting too. The temperature sensor may provide an empirical source of data to take this factor into account.

6 CONCLUSION

This is a domotics device. It may be used to irrigate the plant that the user has indoors if the user is lacking the time to care for its plants. There could be some developments intended to integrate the gadget with some kind of "home server", such as those provided by domotics solutions. It could be made compatible with DomoBus, as it shares it's data model.

The goals started to be modest, and then we thought (with an incentive from the professor) about expanding the feature set, namely with the existence of a host, and by programming the microcontroller in an extensible way. The existence of a host motivates the communication work, while the requirement of being extensible motivates using a cooperative thread system, allowing further extension of the software, which may in the future easily allow controlling more than one vase instance by a single Arduino microcontroller.

This project was interesting, in the sense that it represented something "new". From a technological point of view, it involved raw hardware work, serial communications, and also involved some Web programming work to control the gadget with a Web interface. From another point of view, we can also say that has given us another look towards domotics and it's possibilities as an important field of Computer Science, with much of it yet to be invented.

REFERENCES

[1] Homepage of DomoBus.net, <http://domobus.net/>

[2] ArduinoThread, <https://github.com/ivanseidel/ArduinoThread>

[3] Arduino - Home, <http://www.arduino.cc/>

[4] Django Framework, <https://www.djangoproject.com/>

[5] Boostrap, <http://getbootstrap.com/>

[6] Angular JS, <https://angularjs.org/>

[7] jQuery, <http://jquery.com/>

[8] Bower, <http://bower.io/>

[9] Node.js, <http://nodejs.org/>

[10] Raspberry Pi, <http://www.raspberrypi.org/>