aWareHouse

Environment Control System for Warehouses

MSc in Informatics and Computer Engineering
Programming Paradigms
EIC0065-2S

Duarte Nuno Pereira Duarte - 201109179 (ei11101@fe.up.pt)
Hugo José Freixo Rodrigues - 201108059 (ei11086@fe.up.pt)
João Pedro Matos Teixeira Dias - 201106781 (ei11137@fe.up.pt)

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

May 17, 2015

Abstract

We are in the rise of IoT (Internet of things). A world were everything is connected and we can, with simple tools, monitory and control everything. In this context, there is a lot of space for a more recurrent use of different programming paradigms because of the need of interaction with different layers of system architecture for a single application, since hardware to web.

Our application, *aWareHouse*, was designed with the objective of, with a simple interface, we can monitory a house or a warehouse in terms of environment conditions (temperature, humidity, sound and luminosity).

For accomplishing this was used a combination of hardware/software and some different programming languages, in a way that gave us a stable application that can be used for setting a alarm system when occurs changes in our environment, for taking decisions analysing the past conditions and the relations with external (meteorological) conditions or simple look at the current conditions inside our warehouse.

Contents

1. Introduction

The *aWareHouse* project was developed for the Programming Paradigms course unit of the Master in Informatics and Computing Engineering at FEUP. The project was developed with the objective of combine, at least, three programming paradigms in the same application.

The base motivation for this project was the Internet of Things, as defined by *Gartner*:

"The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment."

So, we designed a low cost system capable of giving the user the possibility of, using a relative small hardware box, monitory the environment of a given place like a home or warehouse. This, associated with a system capable of maintain records of the past conditions of the environment (plus external weather conditions), results in one application that gives the user the capacity of taking decision, be aware of the environment status using alerts and see the current conditions.

As explained before the capabilities of this application make this a system useful in a lot of situations, for example, the monitoring of a warehouse and verify the relation between weather and internal conditions to make decision on what is the best settings for a refrigeration system and assure that the, for example, temperature is always below the maximum recommended. Other example is simple use it as a house control system and, for example, verify if someone forgot to turn any light off.

2. System Description

2.1 Conceptual Description

2.1.1 Functionalities

The *aWarehouse* application consist in a system that scans the environment of the room where the sensors are placed and stores all the information about it on a database.

The system can read temperature, humidity, sound and luminosity values as described in 2.1. Additionally user can define maximum and minimum values so the system will alert whenever an environment factor exceeds this range. This way the user can have absolute control of the room where *aWarehouse* is installed. This application can be very useful in a server room, with the temperature sensor the user can control the room temperature and prevent an overheating. With the sound sensor the user knows if someone is in the room and with the luminosity sensor the user can check if,for example, a door is open.

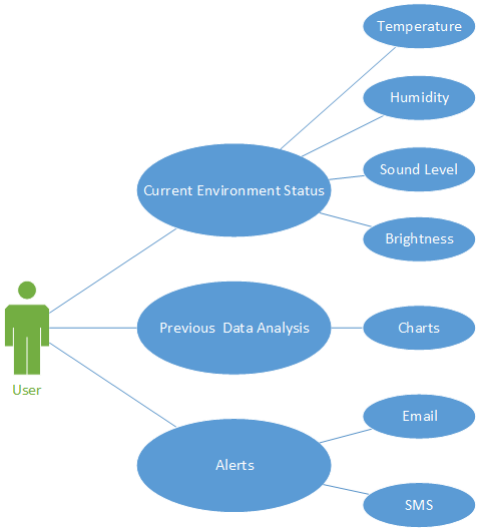

Figure 2.1: Use Case diagram.

2.1.2 Architecture

2.1.2.1 Physical Architecture

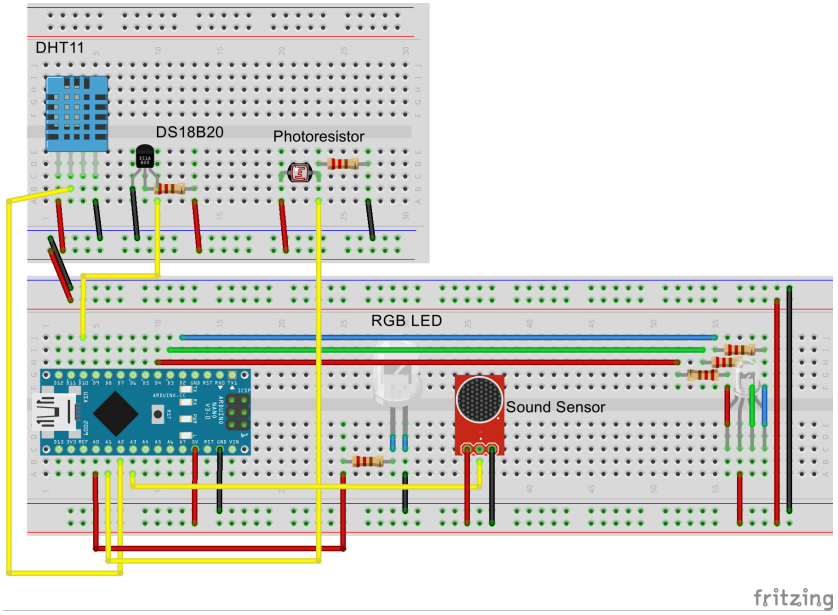

Figure 2.2: Circuit diagram.

Sensor	Description	Input/Output
DHT11	Basic and low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin.	Temperature and Humidity values.
DS18B20	1-wire digital temperature sensor fairly precise (+- 0.5°C over much of the range) and can give up to 12 bits of precision from the onboard digital-to-analog converter.	Temperature values.
LM393	One single channel output sound level. Low level output signal and when there is sound output low, lights lit.	Sound level values.
Diffused LED	A diffuse LED with with separate red, green and blue LED chips inside, capable of emitting a color-mix resulted of the values passed to each chip.	RGB Color or Color-Mix.
Photo cell	Photo cell or CdS photoresistor is a little light sensor. As the squiggly face is exposed to more light, the resistance goes down and the voltage goes up.	Voltage value.

Table 2.1: Sensors description.

Arduino

Figure 2.3: Arduino micro-controller.

The Arduino Nano (fig.2.3) is a small, complete, and breadboard-friendly

micro-controller based on the ATmega328 processor chip and works with a Mini-B USB cable for energy and data transfer. It has 32 KB of flash memory space of which 2 KB used by bootloader.

Each of the 14 digital pins on the Nano can be used as an input or output and 8 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values).

The Arduino provides an UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An FTDI FT232RL on the board channels this serial communication over USB and the FTDI drivers (included with the Arduino software) provide a virtual COM port to software on the computer.

Raspberry Pi

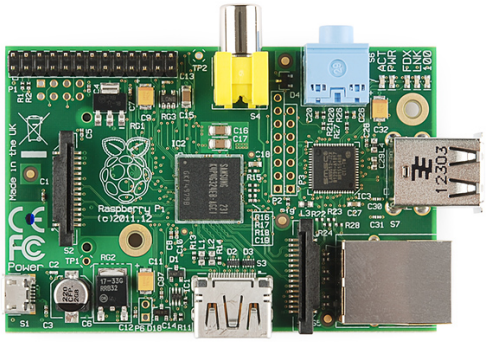

Figure 2.4: Raspberry Pi 1 Model B.

The *Raspberry Pi 1 Model B* (fig. 2.4) is a single-board computer which can be used for many of the things that a desktop is used to.

The design is based around a Broadcom BCM2835 SoC, which includes an ARM1176JZF-S 700 MHz processor, VideoCore IV GPU, and 512 Megabytes of RAM. The memory used is a SD card for booting and long-term storage. This board is intended to run Linux kernel based operating systems. Additionally this has two USB ports and a 10/100 Ethernet controller.

2.1.2.2 Logic Architecture

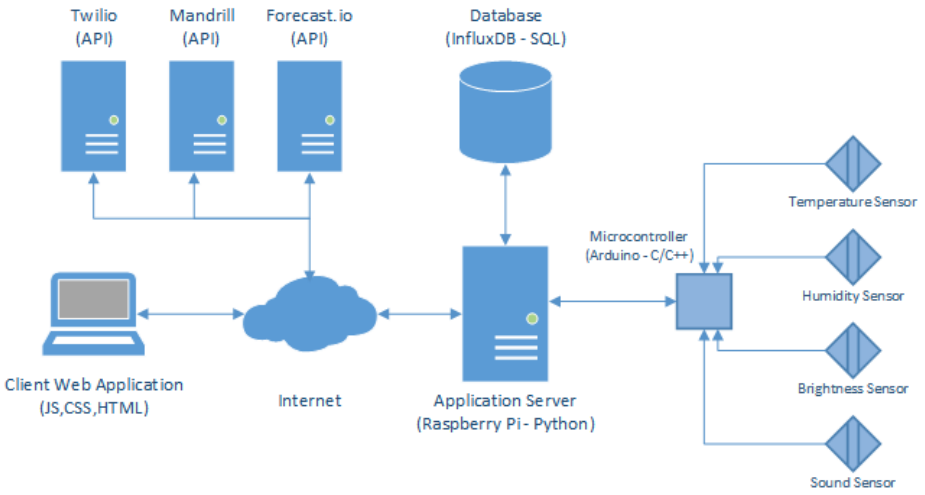

Figure 2.5: Architecture diagram.

2.1.3 Programming Languages and Technologies

In the context of realizing this project it was conciliated a diversity of technologies and programming languages that, working together in the architecture presented at fig. 2.5, made the application possible. Starting by enumerating the programming languages used and the reasons for choosing those languages are the following.

- Arduino Language[13] is a low level language based on C/C++ and integrated with AVR Libc[14] and allows the use of any of its functions for interacting with hardware. Since its a derivative language of C/C++ it has a multi-paradigm design being imperative and object-oriented. The version used is v1.6.4.
  - DHT Sensor Library[9] is an Arduino library for the DHT series of low cost temperature/humidity sensors that facilitates the reading of values from this sensors.
  - OneWire Library[10] is used to access 1-wire devices made by Maxim/Dallas, such as temperature sensors. For temperature sensors, witch is our case, the DallasTemperature[8] library can be used with this library.

- Python Language[15] is a general-purpose, high-level programming language. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. The version used is v2.7.9.
  - Flask Library[11] is a micro web application framework for creating simple web RESTful applications.
  - Schedule Library[12] make possible to run Python functions (or any other callable) periodically at pre-determined intervals using a simple, human-friendly syntax.
  - pySerial Libray[5] facilitates the use of serial port communication in Pyhton applications.
- Web Languages[17] are a set of languages that are interpreted by a browser in a rendering process to present the content on a browser screen.
  - HTML5 or HyperText Markup Language is the language used to describe and define the content of a Web page in a structured format.
  - JavaScript(ECMAScript 5/6) is the programming language that runs in the browser, which is used to build advanced user interactive Web sites and applications.
  - CSS3 or Cascading Style Sheets are used to describe the appearance of Web content.
- SQL or Structured Query Language is a special-purpose programming language designed for managing data held in databases. In our case, InfluxDB database, has a SQL like query language[16] for find and grouping data.

In addiction to the programming languages we used a set of technologies as a part of our application as presented in table 2.2.

InfluxDB	InfluxDB[6] is a time series, metrics, and analytics database. It's written in Go and its oriented for recording events data, like data from sensors.
Grafana	Grafana[7] is an open-source metrics dashboard with direct integration with various databases like InfluxDB.
Web API's	• Forecast.io: Weather forecast RESTful API that returns the current weather conditions for a specific location (Latitude, Longitude) • Twilio: Simple RESTful API for sending text messages (SMS). • Mandrill: Powerful, scalable, and affordable email infrastructure as a service, using a RESTful API:

Table 2.2: Technologies.

2.2 Implementation Description

2.2.1 Implementation Details

This project had the need of using and establishing a diversity of API's (Application Programming Interface) for communication between the different application layers and external services used.

On one hand, we used various RESTful API for accessing external services (Mandrill[3],Forecast.io[2] and Twilio[1]) as refereed in table 2.2. The RESTful stands for Representational State Transfer, that consists in using the standard HTTP operations for communication data in JSON or XML format. In our case, the external services used can communicate using JSON (JavaScript Object Notation) messages for requesting and pushing data.

Additionally we standardize a REST API for communicating with the Python server from our web application.

On other hand, for establishing communication between the Python server and the Arduino we needed to use Serial Port communication using the Arduino serial port[4], and a Python library called *pySerial*[5]. For this we

standardized a system of request/response were we passed a character that expressed the operation wanted and the Arduino responded using a JSON string with the values requested. The serial port communication is emulated above an USB port.

2.2.2 Development Environment

The development environment used depended on the layer that we where working, because different languages/technologies required different environments.

For developing the low level hardware phase, using the Arduino platform, we needed to use the Arduino Software (IDE)[18], that compiles the Arduino code, checked for errors and upload it to the Arduino Board.

When developing the Python server application we need to use the Python compiler for analysing Python source code and generating Python bytecode, and the Python IDLE that is the Python IDE built with the Tkinter GUI toolkit. Beside this, when deploying the Python server code on the Raspberry Pi we used SSH (Secure Shell) to connect with it and, by this way, initiating text-based shell session on the remote machines in a secure way, allowing us to run commands on the machine prompt without the need of being physically present near to the machine.

Another tool that we needed was the Go language compiler (v1.4) because we used InfluxDB, that is written in Go Lang[19], and we needed to compile it for ARM processor (Raspberry Pi processor architecture).

For developing the client web application and extending the Grafana we used a text editor, more specifically, Visual Studio Code.

Additionally we used Git control version system for sharing/working with code between the members of the working team associated with the GitHub free repository system for students.

3. Conclusion

The *aWareHouse* project was designed to take advantage of different programming paradigms in a way that resulted in a stable product, using various programming languages, frameworks, libraries and technologies.

In a way of summary, we used low-level programming for Arduino and sensors, taking advantages of imperative paradigm. For the server that communicates with the web application and the Arduino we used Python that is a multi-paradigm language (object-oriented, imperative and functional). For the web application itself we used the standard web technologies (JavaScript, HTML and CSS). JavaScript, principally, is know for being another multi-paradigm language (scripting, object-oriented with prototypes, imperative and functional).

The actual result was a solution that can easily adapted for the market with a relative good precision, trustability and stability. Additionally it is easy adaptable and/or extendible with new functionalities (sensors for example).

This project fits in a time that we start to hear everyday more about *Internet Of Things* and *Big Data*, and the possibilities that this brings to us. We conciliated this two big thematics and give the user the power to watch and be aware of a warehouse or any confined space, and, at the same time, be capable of see the past data and the influence of external conditions in inside environment.

4. Improvements

Although we think that *aWareHouse* is almost ready to market at this stage we already found space for improvement and evolution, namely:

- Addition of more and basic sensors like, for example, a motion sensor.
- The storage system has to be improved since that with the accumulation of past data the system can be really slow.
- Possibility of integrate some automatic advice system, using artificial intelligence, that provides the user advices on what to change to make the environment more cold for example.
- Improve the security of application creating user accounts and better data protection.

We see this points as growing space, and possibility of make a better application for the final user.

References

[1] *Twilio REST Web Service Interface*, http://www.twilio.com/docs/api/rest

[2] *The Dark Sky Forecast API*, http://developer.forecast.io/docs/v2

[3] *Mandrill API Documentation*, http://mandrillapp.com/api/docs

[4] Arduino, *Serial*, http://www.arduino.cc/en/reference/serial

[5] *pySerial*, *pySerial's documentation*, http://pyserial.sourceforge.net/

[6] *InfluxDB*, *An open-source, distributed, time series database with no external dependencies*, http://influxdb.com/

[7] *Grafana*, *An open source, feature rich metrics dashboard and graph editor for Graphite, InfluxDB & OpenTSDB*, http://grafana.org/

[8] *Dallas Temperature Control Library*, http://milesburton.com/Main_Page?title=Dallas_Temperature_Control_Library

[9] *Using a DHTxx Sensor*, Adafruit Industries, https://learn.adafruit.com/dht/using-a-dhtxx-sensor

[10] *OneWire Library*, Jim Studt, http://www.pjrc.com/teensy/td_libs_OneWire.html, 2007

[11] *Flaks*, Armin Ronacher, http://flask.pocoo.org/, 2014

[12] *Python job scheduling for humans*, Daniel Bader, https://github.com/dbader/schedule

[13] *Arduino Language Reference*, http://www.arduino.cc/en/Reference/HomePage

[14] *AVR Libc Modules*, http://www.nongnu.org/avr-libc/user-manual/modules.html

[15] *Python*, https://www.python.org/

[16] *InfluxDB Query Language*, http://influxdb.com/docs/v0.8/api/query_language.html

[17] *Web technology for developers*, Mozilla Developer Network, https://developer.mozilla.org/en-US/docs/Web

[18] *Arduino Software*, Arduino, http://www.arduino.cc/en/Main/Software, 2015

[19] *Contributing InfluxDB*, InfluxDB, https://github.com/influxdb/influxdb/blob/master/CONTRIBUTING.md, 2015

Appendices

A. Appendix

A.1 Hardware

Raspberry Pi

Technical Specs:

- Chip: Broadcom BCM2835 SoC full HD multimedia applications processor
- CPU: 700 MHz Low Power ARM1176JZ-F Applications Processor
- GPU: Dual Core VideoCore IV® Multimedia Co-Processor
- Memory: 512MB SDRAM
- Ethernet: onboard 10/100 Ethernet RJ45 jack
- USB: 2.0 Dual USB Connector
- Video Output: HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
- Audio Output: 3.5mm jack, HDMI
- Onboard Storage: SD, MMC, SDIO card slot
- Operating System: Linux (Raspbian OS)
- Dimensions: 8.6cm x 5.4cm x 1.7cm

Arduino

Technical Specs:

- Microcontroller: Atmel ATmega328
- Operating Voltage (logic level): 5 V
- Input Voltage (recommended): 7-12 V
- Input Voltage (limits): 6-20 V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 8
- DC Current per I/O Pin: 40 mA
- Flash Memory: 32 KB of which 2 KB used by bootloader
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz
- Dimensions: 0.73" x 1.70"
- Length: 45 mm
- Width: 18 mm
- Weigth: 5 g

Sensor DHT11

Technical specs:

- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20-80% humidity readings with 5% accuracy

- Good for 0-50°C temperature readings +- 2°C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

Sensor DS18B20

Technical specs:

- Usable temperature range: -55 to 125 °C (-67F to +257F)
- 9 to 12 bit selectable resolution
- Uses 1-Wire interface- requires only one digital pin for communication
- Unique 64 bit ID burned into chip
- Multiple sensors can share one pin
- +-0.5C Accuracy from -10°C to +85°C
- Temperature-limit alarm system
- Query time is less than 750ms
- Usable with 3.0V to 5.5V power/data

Sound Sensor (LM393)

Technical specs:

- LM393 Controller
- Electret microphone
- Working voltage: DC 4 - 6 V
- With a signal output instruction
- One single channel output
- Low level output signal
- When there is sound output low, lights lit

Diffused Led

Technical specs:

- 10mm diameter
- Red: 623 nm wavelength, Green: 523 nm, Blue: 467 nm
- Red: 1.8-2.2V Forward Voltage, at 20mA current, Green: 3.0-3.4V, Blue: 3.0-3.4V
- 50 degree viewing angle.
- Red: 700 mcd typical brightness, Green: 2100 mcd, Blue: 900 mcd

Photo cell (CdS photoresistor)

Technical specs:

- When its light, the resistance is about 5-10KOhm, when dark it goes up to 200KOhm.
- The voltage on the pin will be 2.5V or higher when its light out and near ground when its dark.