GLaDOS, a complete smart-home for Beginners

A DIY learning project with real-life applicable results

Odysseas Lamtzidis
Electrical Engineering and Computer Technology
University of Patras
Patras, Greece
hi@odyslam.me

***Abstract*— Internet and open source technologies have enabled anyone to dive into the world of technology. With next to little knowledge about anything, I succeeded in transforming my apartment into a "smart-home" learning various technologies and acquiring a wealth of skills in the process.**

Keywords—smart-home,open-source,DIY,inventing,electronics, Raspberry pi, programming

I. INTRODUCTION

It is known that the Greek University offers little practical knowledge in contrast to the vast theoretical analysis that students go through. This combined with the fact that I found fascinating the world of smart-home automation, I decided to create a system of my own as a learning medium for programming and electronics, the basics of which I already knew. The system grew in size and reliability, ranging in its current form, from controlling various appliances to communicating intelligently with a use of a Facebook messenger bot and a Natural Language Processor (NLP). In this paper I will go through all the stages of the system as I want to demonstrate that it's not only easy to make something of your own, but also the learning process is invaluable for any engineer, no matter the field. The whole project cost around 100 euro, meager compared to it's functionalities.

II. THE SYSTEM

A. *The outline*

The system controls:

- Building's & apartment's door
- Apartment's Boiler
- Various lights
- Hi-Fi
- Media Player windows computer
- A custom Smart power strip

B. *The core system*

The core of this project is a Raspberry pi 2[1] a small single-board computer running a linux distribution (Raspbian). It is developed by the Raspberry Pi Foundation in the UK and it was meant originally as a computer science teaching medium.

Raspberry pi 2 "Fig. 1," is powered by a quad-core ARM Cortex A7 cpu running at 900Mhz and 1GB of RAM. It is also a development platform, as it boasts a 40 GPIO (General Purpose Input Output) pin header it can run various Operational Systems, the most common being Raspbian, which has a huge ecosystem of libraries and platforms written on various languages. For the project I used a platform written in Python 3 and various Python 2.7 & Python 3 libraries as well as C++ platform that can manage Infrared signals (LIRC).

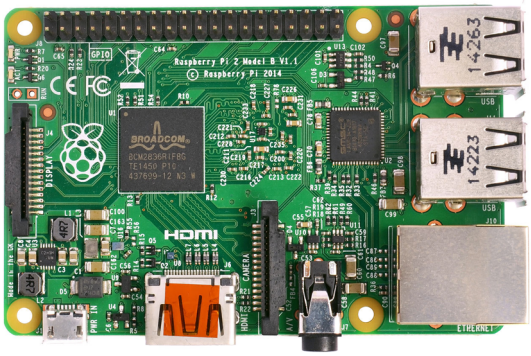

Fig.1. Raspberry pi 2 computer

The smart strip is based on ESP8266-12E [2] "Fig. 2," a low-cost WIFI SoC with full TCP/IP stack and microcontroller capability produced by Espressif. The chip supports various firmwares, I used a community built Arduino firmware as it is easy to use and has substantial performance.

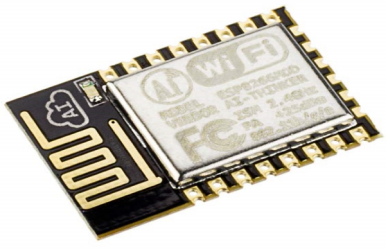

Fig. 2. the Esp8266-2E SoC

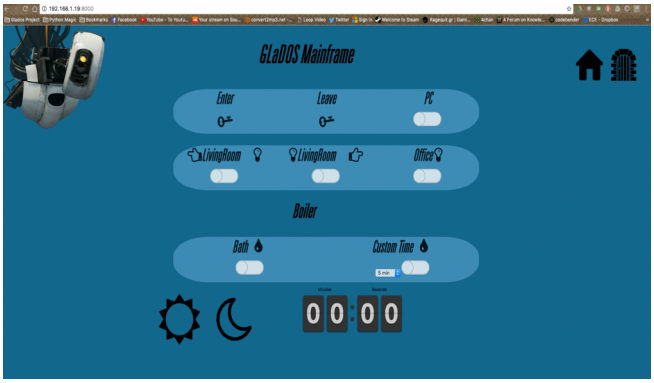

Fig. 3. My custom interface of the Webiopi framework

C. *Hardware used*

The particular interest of this project was that not only it demanded programming knowledge in different languages, but also some basic knowledge in electronics. The hardware used will be categorized by device (Raspberry pi, ESP8266).

Raspberry Pi:

- Servo & external power supply for apartment's door
- A reed switch for the apartment's door
- 2-Relay module for building's door & Servo
- Infrared Led & transistor for HIFI
- 433Mhz radio transmitter for sockets & boiler
- 433Mhz RF (Radio Frequency) sockets

ESP8266:

- 4-Relay module for smart strip
- AC/DC converter & step down buck module
- Resistors for stability
- Opto-isolator as a switch for the media player

The use of each hardware and its peculiarity will be explained later on as I go through each different module.

D. *Software used*

The core platform is called **Webiopi** [3], a free-to-use framework written in Python and enables one to control Raspberry from a simple web interface "Fig. 3". It was first published in 2012 when there were no platforms that focused on the IoT aspect of Raspberry pi. The back-end of the framework is somewhat badly written and now it should not be the basis of any project, given the plethora of other platforms and frameworks that exist. The framework is written in Python and in its core we have a REST [4] server and a HTTP server "Fig. 4". A REST server is a server that is controlled by making HTTP requests (usually GET, POST) to certain links. This links are structured in a human friendly way. For example, let a REST server with the following syntax: http://ip:port/digital/gpio/40/1. By making a simple GET request to that link, we would be able to turn on GPIO number 40. REST nowadays is widely used and is structure most web APIs use. The HTTP server serves a web-page that acts as an interface that makes calls to the REST server. The HTML page is entirely up to the user to tailor it to his needs, using a custom Javascript library that handles the REST calls. The REST server offers a wealth of functionalities, from reading the state of a GPIO to controlling a range of devices that are pre-configured in the framework (temperature sensors, servos, etc.).

Moreover, we can run custom Python code with the use of a central Python script (hereafter script.py) that is handled by the framework. In this script we can define functions that will be exposed by the REST server. Exposing in this context is the process of making something accessible from the internet, in our case Python functions that can be run remotely. To expose them, we decorate each function as @macro. The REST server gives the possibility to pass arguments from the web interface to these functions as well as return data to the web interface. This intercommunication between the back end and front end uses JSON , a key-value data structure popular in web programming.

Script.py uses a syntax that resembles Arduino, meaning that we can define a function called setup() that is run as soon as the service starts, a function destroy() that run when it stops and a function loop() that is run indefinitely. This file will call any external library and load any Python module that we make.

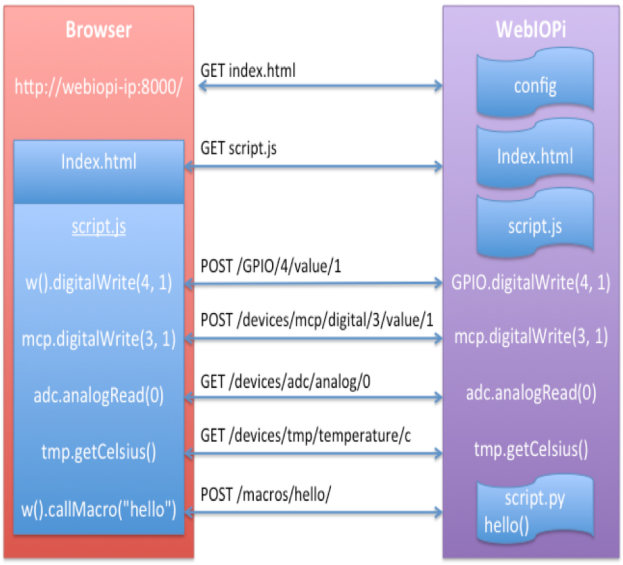

Fig. 4. The outline of the framework

E. *Configuration*

To configure the framework, we simply follow a tutorial to install it and then we have to modify 4 important files.

1. The HTML page that will serve as the web interface
2. The Javascript file to make a dynamic web interface and CSS to style this page to our preference.
3. Script.py that will be the core of our project.
4. A configuration file for the web page directory, basic HTTP authentication password, etc.

In my project, I mainly used the Python aspect of the library because I needed persistency of the state of the modules. This

means that I only invoked @macros from the web interface, and in these macros I performed whatever action I wanted. The Javascript library was entirely scraped and used Jquery to make the REST calls in an effort to improve performance.

F. Python libraries

One of the most fascinating aspects of the Python programming language is its huge ecosystem of libraries. We can find libraries for any task, regarding many different industries, from data science to Internet of Things. In this part I will introduce the libraries I used without getting into detail.

Pi-switch [5] a library used to control 315/433 MHz remote controlled power sockets. It is used in conjunction with the 433Mhz transmitter, we only need to point the pin from which the transmitter reads data and the data (string, hex, etc.) we want transmit. In my case, some 12bit binary codes that were preconfigured in the sockets as ON/OFF signals.

Vncdotool [6], a command line vnc client that boasts a Python library as well. It was used to control the media player windows computer by emulating the mouse and keyboard and clicking/writing in certain areas of the screen. Although it worked, it needs to be tailor made for each screen and configured after trial and error in order to find the right coordinates for each click and/or type. It is not usable in environments were windows sizes can change, new windows pop etc. In the next version of the project will be largely replaced by a windows service written in Python and controlled via REST.

Selenium [7], a portable software-testing framework for web applications in the Python library version. Although it is meant to automate web application testing, I used it as a fail-safe mechanism for internet outage. When an internet outage was sensed, selenium web-drive would log in into my router's configuration page and restart it. To achieve this I used a Mozilla Firefox plug-in that allows to record an internet session and then convert it to Python commands. Even though it is tailor made for my router's web-page in can be replicated for any router with relative ease. It is worth noting that because my raspberry pi was running without a screen attached to it, Selenium needed an extra library that emulates a screen in order to function properly.

Pywin32 [8], a project that exposes the innards of Windows to Python. With this framework we can build windows services using Python. I created a windows service that controls Spotify on the local machine. In the future this service will be able to shut down and monitor the media player so as to automate even more tasks. For example, as soon as a video starts, by detecting the runtime of certain programs, it could give a signal to the system to shut down or dim the lights appropriately.

Flask [9], a Python micro-framework based on Werkzeug and Jinja 2. It is used to create web applications and was used in the windows service to create a simple REST server, an API that will take commands from the central system.

Fbchat [10], this little library emulates the browser and uses http requests to control the facebook profile of any given account. It's usage is simple, and you can use it to parse each message that is sent to an account or even reply. I used it as a proof-of-concept facebook bot messenger. This library handled the messenger part while another, pyWIT handled the logic and actions of the bot. It can be used for a variety of purposes but it is not official and as such it needs constant updating, as facebook frequently changes the structure of its services. I do not recommend to create a facebook bot using this library as you don't create a valid facebook bot, rather an automated facebook profile and it's reliability is not sufficient.

Pywit[11], is a wrapper library around the HTTP API of the Wit.ai NLP engine. Wit is a service that can extract meaningful data from sentences by training a bot through its web portal. Training means that we enter a phrases and we point what certain words should mean for our bot, after training it can find not only produce data from certain *keywords*, but also take into consideration the context or its place in the sentence. Wit it is not only one of the most accurate of the services that exist, but also free and available even in Greek. For our project we also used a functionality called *conversation*. We can define a *conversation*, meaning the exchange of phrases between a human and the bot, by writing what the bot should say and what answer it should expect. Based on the answer, it can take decisions as to how the conversation should continue but also if it should fire certain *actions*. An action is a function that will be fired in our script, whether it's Python-wit or wit for Node.js. In essence, we send phrases to the service and it return's data, we then take the data and we decide what to do, the decision making of the engine helps to reduce the decision making that we do locally.

G. Non Python libraries.

Lirc (Linux Infrared Remote Control) [12], is a package that allows you to decode and send infra-red signals. It's the easiest way to control IR devices, easier even than using microntrollers like Arduino. Lirc has two components, one that records the IR signals of any given remote control using a IR receiver and one that replicates them using an IR led. The astonishing feature is that before recording, by pressing buttons on the remote, lirc "learns" all about the decoding of the IR signals. Then by simply recording the IR remote, you can create a digital remote, that will emit the given signal whenever you want. Lirc is used from the command line. I used it to control my hi-fi system (on/off, volume, mute, etc) using Python.

aREST [13], is a wonderful library for numerous microcontrollers, including the esp8266 with Arduino core. By simply including this library, we can access the esp8266 pins from the internet. It enables also to expose user defined functions. Although it is very useful, HTTP is quite a heavy protocol for our work and it enlarges the power consumption considerably. A better approach would be using the MQTT protocol that is substantially more lightweight and created for the needs of IoT.

III. MODULE ANALYSIS

A. The philosophy

The System consists of various different sub-systems with quite different nature and performance. One very interesting factor that I had to take into consideration was time. There are a couple of tasks that need time with no feedback available of whether the task has succeeded. To facilitate my work and the

readability of the code, I broke each part of the system in a different module that had a single class. Each part thus was converted into an object with its own methods and attributes. Each module imports only the libraries it needs and script.py imports all the modules, as well any library it may need. In this chapter I will go through each and every module, explaining both the software aspect and where applicable the hardware as well. In "Fig. 5" there is an outline of the system and it's modules.

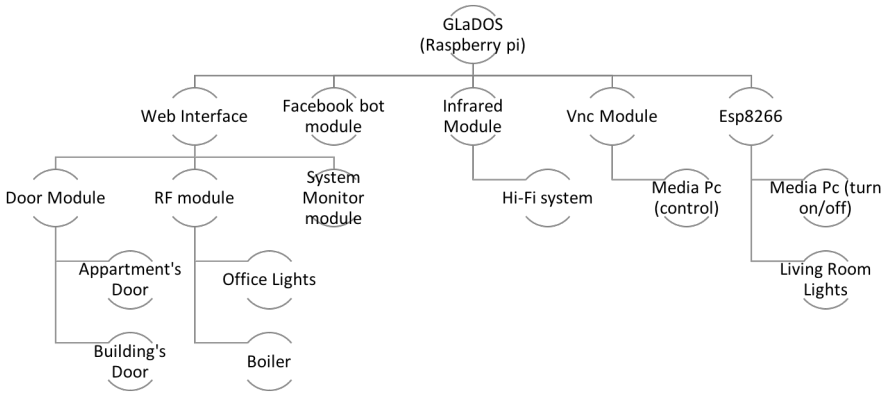

Fig. 5. The outline of the System

B. *Script.py*

As I have said, script.py is the centrepiece of the system. In this script, after the imports, I initialize all the variables (input/output GPIO number, esp8266's ip, etc) and create instances of each module. All these are defined outside of the setup() function as I need them to be global so they are usable from all the @macros. In the setup() I use the Webiopi Python library to configure the GPIO and make some initial configuration, such as locking the door. After this, script.py rests idle and waits for one of its @macros to be called by the web interface. In the @macros, I have defined certain sequences that automate certain parts. An example is that there are 2 "buttons" to enter the house, one that simply opens the doors and another that turns-on various lights as also the computer and finally plays music. The code can be inspected on Github[14].

C. *The Door module*

Every part of this project was created to solve a particular problem of mine, the first being the continuous misplacement of my keys. Thus I needed a system that would enable me to open remotely both the door of the building but the apartment's as well. The solution of the first one was easy, by inspecting the building intercom system I was able to figure which cables I needed to short in order to open the door, using a Relay it was easily solved. It is worth mentioning that intercom hacking of an apartment's intercom system can affect the intercom system of the whole building.

The apartment's door was more intricate "Fig. 6". I could use a smart lock, an electric bolt or an electric strike. The first one was expensive while the others did not provide the needed security, as I needed something that would be able to open in case of black-out and that could not be picked easily. The answer was a lock that has handle in the inside end. I 3d printed 2 gears and screwed a servo on the door. Servo control is not as easy as it may seem since they use PWM (Pulse Width Modulation) [15] to be controlled. It is advised to use specific circuits or controllers to control them as the use of a single GPIO on a computer like Raspberry pi created a constant jitter. This jitter is the result of the precise timing that PWM demands to be smooth. The jitter was nullified by turning off the servo while the door was closed by using a relay. Finally, servos can be demanding regarding to amperage, this is why it is advised to use an external power source and not the micro-controller or computer itself.

The last component of the Door module is a reed switch; it is a device that allows current to pass through when it is a sufficiently powerful magnetic field. By placing the switch on the doorframe and a magnet on the door. This way we can detect whether the door is open. This is crucial so as the Servo can lock automatically. Now, we need to detect a close door – open door – close door sequence so the servo locks the door only after someone uses it. This sequence is programmed in the Door module, which is available on Github.

Fig. 6. The door mechanism

D. *The Boiler module*

To control the Boiler, I used a Din Rail Power Relay as the boiler demands current up to 15-20 Amperes. The power relay's secondary coil is controlled by AC current, as such I used a RF socket that was plugged right under the switchboard. The Power

Relay is fitted after the security switch that exists with every boiler, to use it we need to leave the switch in the ON position.

The socket in turn was controlled from the RF transmitter plugged into the Raspberry. To use any RF device, you need to know the message that you need to transmit and is usually pre-configured in the devices controllers. Since this is rarely documented, we can simply decode the signal of the associating RF remote control that usually is paired with the said device using a microcontroller like Arduino and readily available libraries. In my case, the signal was a sequence of 12 bits. Regarding software part, Pi-switch, which is the library we use to send the RF signals is only available in Python2.7 while webiopi runs Python3. Thus I created a simple CLI (command line interface) script for sending any signal and invoked it from the script that handled the state of the boiler and its timing. The code is available on Github.

E. The Pc module

The pc module is based on Vncdotool. To use it we need to have a vnc server installed on the machine we want to control. After, the difficult part is to find the right screen coordinates to click or write. After testing and using keyboard shortcuts I was able to log in/out and control my Spotify client. Although it worked, it's reliability was poor and even the slightest interface change could render it obsolete. The use of a custom windows service is much easier and offer more options. Lastly, since there was no feedback from the actual pc, I had to use time to ensure that the previous action of any given action was complete, as in Log in *then* play music.

Vncdotool was written in Python2.7 and I needed to create a CLI script that was called from the Python3 module . The code is available on Github .

F. System monitor module

The system needs reliability above all, especially when the entry in the house depends on it. As such I needed a script to monitor not only that the web interface is up and running but also that it is connected to the internet. For this reason , I created a script that was called by linux repeatedly (cronjob) and would either reboot the Raspberry or would reboot the router. The later was trickier and used a library called Selenium, as described above. The code is available on Github.

G. Facebook bot module

Facebook bot was one of the most interesting aspects of this system, enabling human-like communication. The library that was used is pretty straightforward to use, as such I will focus on the WIT.ai aspect of the bot, its NLP. Each time a message was received, after it was verified that it came from my account, the script would send it to WIT and invoke its conversation module. With each message that the bot received, it invoked the same conversation API. To understand whether this conversation was the same or a new one, we also sent a Python dictionary called context and a conversation id. Each time a conversation ended, the bot would erase the context and change the id of the conversation.

As I described above, the bot has been trained not only to understand certain words and phrases (or their relative place) but also how to respond to certain conversations or stories. With the first message of each conversation, the bot was trained to understand the general intent of the conversation (e.g. boiler control, door control, etc.) and commence the right story which would dictate its response and then what to expect.

Each message that we send, it returns as a set of entities, data that has been extracted (e.g. turn on/off, close, open). This data is handled by the action, the function that we have defined that will be run. This function will search for certain data that it needs in order to perform a certain action (i.e. turn on the boiler for 30 minutes) and depending on the entities it will return a used defined context. Now the bot in the next message will not only process the message, but the context as well in order to decide the right route in the pre-defined story. An example is in order to demonstrate the system, albeit in Greek, it serves our purpose:

Let's say we have this story "Fig. 6,"

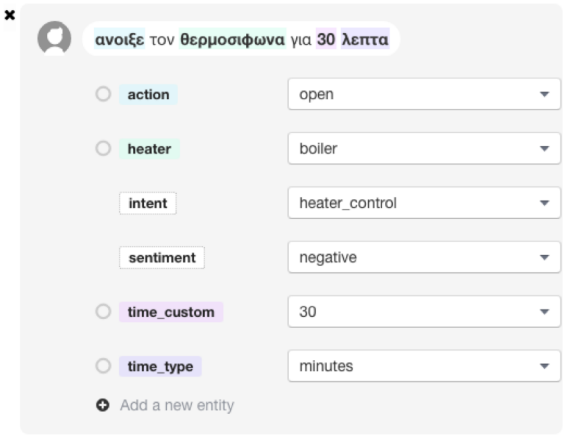

Fig. 6. The start of a story, the first message the bot receives

As you can see, the bot detects certain *entities* that are passed to the *action* function. This function will check that is has whatever it needs in order to turn on the boiler for X minutes. If it has what it needs, it will return a *context* with the keys: *ctime, timetype, open. s_positive* is whether the user was polite or not.

In our example, we say that he was, so given this context the bot continues this particular route. As you can see in¨Fig. 7", the grey boxes indicate routes for other contexts, as for example if the user wasn't polite, or if he didn't indicate for how long he wanted the boiler to be on. To perform each action, it simply makes the right request to Webiopi's REST server, the same that is done using the web interface. The code is available on Github.

It is worth mentioning that the conversation API will be deprecated and WIT.ai will only serve to extract data from messages [16]. The logic and decision making will be done by the user's script.

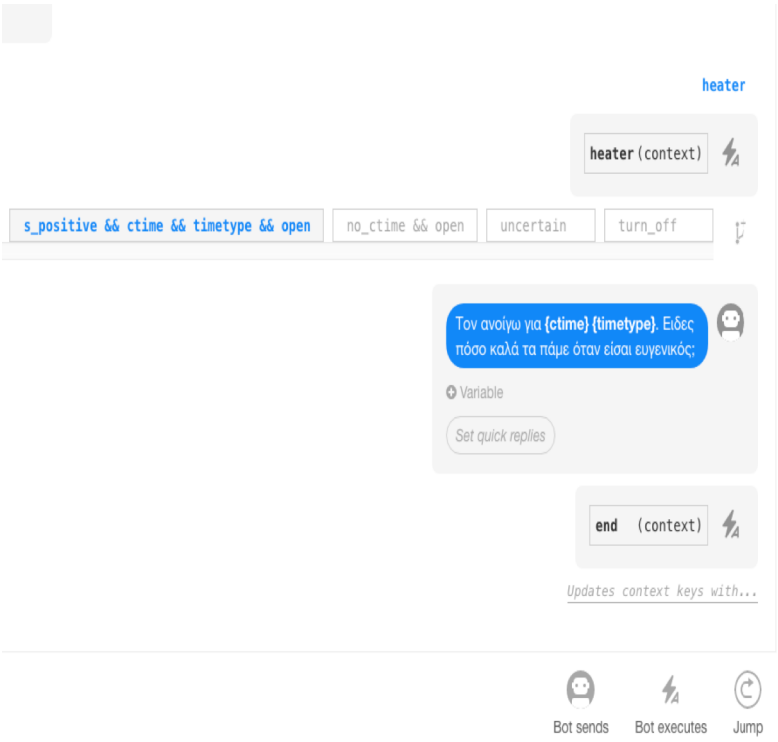

Fig. 7. The answer of the bot of the same story as in "Fig. 6,"

H. *Esp8266 smart strip module*

This is the first module where it is not based on Python. Of course, we use Python from the Raspberry side to make the REST requests, but the interest lies mostly on the setup of ESP8266 "Fig. 8".

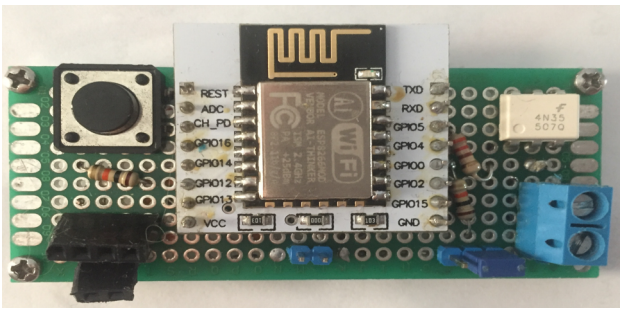

Fig. 8. The Smart-strip's prototype board

The ESP8266 will control 4 different relays as such we have the 4 dupont females on the left. Moreover we need to power the relay module, thus the 2 dupont females. The button resets the chip and the jumper on the right will change it's mode (normal functionality, flashing new firmware). The opto-isolator (the white IC on the top right corner) enables the ESP8266 to act as an electronic switch of the computer's power. The computer will turn on if a certain circuit is shorted, either by pressing a button or by turning on the opto-isolator. Finally, the central male pins power this board. To power it wer use an AC/DC converter that produces 5v and 700mA dc voltage that passes through a DC/DC buck converter which not onlly downscales the voltage to 3.3 but also stabilizes it and counters electrical noise.

The software is very straightfoward, thanks to the use of the aREST library. The code of the esp8266 is available on Github.

I. *The infrared module*

I created a simple CLI Python script that is called by the system and runs the appropriate LIRC shell commands.

The hardware was simple as well, we simply connect a IR led to a GPIO (the use of a transistor as amplifier is imperative) and we point the led to the direction of the device that we want to control.

IV. SUMMARY

It is apparent by now that the system is highly custom-tailored to the needs of apartment and is not scalable. Moreover, the reliability of certain elements (like VNC) was poor and needed constant maintaining. In any case, it worked for one year (in this paper's form) and it never produced a critical problem. Overall the experience was fairly positive, and in fact did decrease the electric bills as the boiler would always stay on for a certain period. The door module also was also extremely usable for opening the door from any room of the house.Regarding security, the platform was only accessible from the local network and was protected with BasicHTTPAuth. Regarding the door access, I argue that it was more secure than the average door lock. The point of this whole project is to demonstrate that with relatively little programming and electronics knowledge, one can start a project that has tangible results. It can serve as a great learning experience to dive into programming, as it demands many different languages and technologies. Regarding the IoT, nowadays there are many complete platforms like HomeAssistant [17] that offers a mature ecosystem and highly customisable frameworks. It is important to underline these systems truly shine when proper automation is implemented and the house reacts by-itself to the actions of the habitants.

V. REFERENCES

[1] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
[2] http://espressif.com/en/products/hardware/esp8266ex/overview
[3] http://webiopi.trouch.com/
[4] https://en.wikipedia.org/wiki/Representational_state_transfer
[5] https://github.com/lexruee/pi-switch-Python
[6] https://github.com/sibson/vncdotool
[7] http://www.seleniumhq.org/
[8] https://sourceforge.net/projects/pywin32/
[9] http://flask.pocoo.org/
[10] https://github.com/carpedm20/fbchat
[11] https://github.com/wit-ai/pywit
[12] http://www.lirc.org/
[13] https://arest.io/
[14] https://github.com/OdysLam/GLaDOS-project/blob/master/Python%20Modules/GLaDOS.py
[15] https://learn.sparkfun.com/tutorials/pulse-width-modulation
[16] https://wit.ai/blog/2017/07/27/sunsetting-stories
[17] https://home-assistant.io/