An Extendable Python Library To Manipulate Sensors Coupled To The Raspberry Pi

Edivaldo M. F. de Jesus Jr*
Instituto Federal da Bahia
Rua Emídio dos Santos, S/N
Barbalho, Salvador Bahia
juniorug@gmail.com

Manoel C. M. Neto†
Instituto Federal da Bahia
Rua Emídio dos Santos, S/N
Barbalho, Salvador Bahia
manoelnetom@ifba.edu.br

ABSTRACT

This article encourages the use of open source technologies, due to the growth of free movement of hardware, inferring the possibility of an alternative for engineers and professionals to develop their projects and provide the dissemination of knowledge. The convergence of radio technologies, microprocessors and personal digital electronic devices is leading to the concept of ubiquitous computing in which intelligent, mobile and stationary devices, coordinate with each other to provide for users immediate and universal access to new services transparently, aimed at increasing human capabilities. This work is intended to be inserted in this context and aims to define, implement and validate the design and implementation of an extensible Python library for manipulating sensors / actuators coupled to the Raspberry Pi using the raspberry-GPIO-python module. The library, called libsensorPy, uses the Abstract Factory pattern to ensure that sensors / actuators and events from the same family being used in conjunction with guaranteed way. On other platforms, such as Arduino, the APIs provide libraries that encapsulate the complexity of implementation and offer only the interface to use. These libraries do not yet exist formally for those who want to use Pyton as development language applied to the Raspberry Pi. The project also presents the results obtained using some of the implemented sensors, system modeling and results described and analyzed.

Keywords

Ubiquitous Computing,Internet of Things, Raspberry Pi, Sensor, Python

*Aluno do curso de Análise e Desenvolvimento de Sistemas(ADS)

†Doutor em Ciência da Computação e Professor do Curso de Análise e Desenvolvimento de Sistemas

Introduction

The UbiComp in its several ramifications and applications, is treated by many as the twenty-first century's new paradigm of computing. It is the area of computing that studies the coupling of the physical world to the world of information and provides an abundance of services and applications, allowing users, machines, data and objects of physical space interact with each other seamlessly. The theme is considered one of the great challenges of research in Computer Science by the National Science Foundation (NSF) [1][5] and is also present in Computing search Grand Challenges report in Brazil from 2006 to 2016 [2][2], published by the Brazilian Computer Society (SBC).

Researches on ubiquitous computing are being held on topics such as: basic access to any wireless device, mobility support within the network transparently, safety, context treatment, efficient use of energy, presentation of multimedia content and so on. This work is focused on building intelligent interactive environments. In these environments, the fundamental idea is to create ways to avoid that the user needs to go to the computer/device, allowing many of these working at a distance. The use of platforms to integrate the devices that make up these environments is one of the key points to create it. Currently there are some options to fill this gap. This text emphasizes one: the project Raspberry Pi [3,4][9, 10].

Raspberry Pi is a widely used platform for professionals who are interested in the field of ubiquitous applications. It provides basic interfaces for creating small projects and / or for those who must be fed by battery. The platform allows you to use high-level programming languages that are quite widespread as C / C ++, Python and Java. The Raspberry allows the development of a range of projects. For example, home automation (turn on and off electrical devices, remote control for TV, Air-conditioners, etc.), eBook and Audiobook readers and so on.

If we seek a way to access the Raspberry Pi bus we shall find a lot of papers in Python. Which incidentally, brings the acronym of the device name (Pi from Python). However, we cannot find any library that abstracts the details of wiring a sensor or actuator to the Raspberry allowing the user takes care just in read the data already processed and/or converted.

The developer must have a degree of experience that can be

considered basic to a computer professional, but advanced for those who are not.
This paper is structured as follow:

- Section 2 present several technologies involved in the preparation of this monograph, aimed to introduce important concepts of Computer area in which the context of the project is inserted
- Section 3 shows correlated works
- Section 4 presents the details of the executed implementation
- Sections 5 and 6 address the testing methodologies used and the results obtained during system validation.
- The last section presents the conclusions.

1 THEORETICAL BACKGROUND

In this section the main concepts studied are presented, which provided subsidies for the development of the proposed project.

1.1 General Context

The term Ubiquitous Computing was first defined by Mark Weiser [33] in the late 80. At this time, Weiser predicted an increasing in functionality and availability of computing services to end users and, on the other hand, he predicted a decreased visibility of these services. For Weiser, the computing would not be exclusive of a computer. He believed that in the future there would be several different devices connected to each other. At a time when users were using PCs (desktops) and that knowledge needed to operate a computer, Weiser bet on a future where the focus of the users would be the task itself, and not the tool used. In this way, they would use the computer without realizing or require specific technical knowledge. [34]

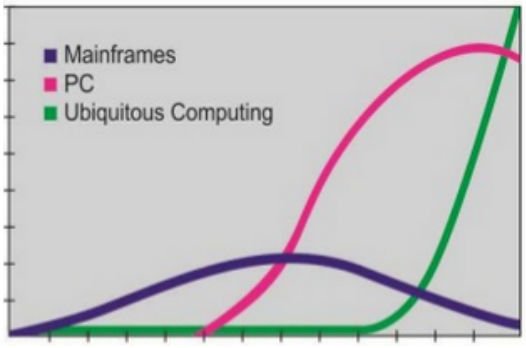

Figure 1: Ages of Computing

The passage of time has shown that betting Weiser was right. For Weiser, [35] evolution of computing has gone through two ages to reach to ubiquitous computing. The first was called the age of mainframe, where many people shared the same computer. The second era was the PC, where each computer was used by one person. Currently, the evolution of distributed information systems, the network connections type options' expansion, mobile computing and the various types of applications on non-conventional computing devices, are just some of the examples that can confirm: the Ubiquitous Computing (the third age) is already a reality. Figure 1 shows a chart with the ages of computing.

Terms such as ubiquitous computing, pervasive computing, nomadic computing, invisible computing, mobile computing and many others, have been used often interchangeably, although they differ conceptually and employing different organization of ideas and management of computer services. Insofar as each area progresses, these concepts will be better understood and its definitions will become clearer. This section presents the key concepts needed to understand the UbiComp besides presenting some project examples in the literature.

1.2 Mobile Computing

Mobile computing is based on the ability of a user to load or move (physically) computer services wherever it moves. In this context, the computer becomes an ever-present device that expands the ability of a user to use the services it offers, regardless of their location. Combined with the ability to access the network, mobile computing has transformed the computing an activity that can be taken almost anywhere.

An important conceptual limitation of mobile computing is that the computational model used in most applications does not change while users are moving. It means that a device is not able to obtain information on the physical context in which computation occurs, and consequently also can not adapt to the new context correctly. A solution to accommodate the changing context would pass to users the responsibility to monitor and manually configure an application / device to the extent that it moves. However, this solution is not well accepted by most users. This limitation was one of the inspirations for pervasive computing .

1.3 Pervasive Computing

The concept of pervasive computing implies that the computer is embedded invisibly in the environment to the user [24]. In this conception, the computer has the ability to: i) obtain environmental information in which it is embedded and ii) use it to build dynamically computational models that allow you to control, configure and tune the application to better suit the needs of a device or user. For this to be possible, the key point is the ability of computers be able to act as "smart" in the environment where users move. This environment is usually populated by computational sensors and services.

1.4 Ubiquitous Computing

As can be seen in figure 2, the UbiComp can be defined as a computer area positioned between the Mobile Computing and Pervasive Computing [14, 24]. Ubiquitous is an adjective originated from Latin (ubiquu) which means "that is at the same time everywhere". Ubiquitous computing benefits from the advances in mobile and pervasive computing and arises from the need to integrate mobility with the functionality of pervasive computing. The term ubiquitous computing will be used here as a junction of pervasive computing

and mobile computing. The justification to perform a distinction of these terms is a device that is embedded in an environment, not necessarily is mobile.

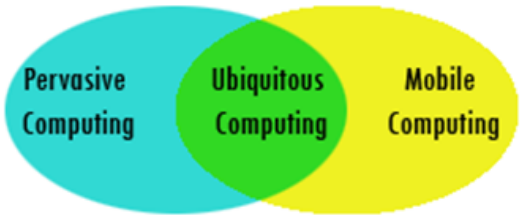

Figure 2: Ubiquitous Computing: intersection between pervasive and mobile computing

Research in Ubiquitous Computing approach about the technologies and infrastructure that enable the deployment of ubiquitous applications through a number of issues including the following:

- how to design hardware and operating systems for sensor platforms?
- how to allow devices to find each other and to use your services?
- how to allow systems involving limited processing resources and energy, to work well?

Generally, ubiquitous applications receive sensor data from other service providers devices, manage user actions, provide support mobility and use context information to perform tasks [6][4]. A ubiquitous system itself has a set of requirements, peculiarities and challenges that influence the design, implementation, deployment and evaluation of its project. [24] These are cornerstones of UbiComp and quite different from those used in the development of systems for PC's. Among these points, we can cite as an example.:

1. Resource-Constrained Devices;
2. Volatile Execution Environments;
3. Heterogeneous Execution Environments;
4. Fluctuating Usage Environments;
5. Invisible Computing;
6. Security and Privacy;

1.5 Internet of Things

The Internet of Things (IoT) is a multidisciplinary field, covering a wide range of subjects, from purely technical issues (eg, routing protocols, semantic queries) to a mixture of technical and social problems (security, privacy, usability) as well as social and business topics. The existing internet of things' applications are potentially diverse. Monitoring of environmental and personal health, monitoring and control of industrial processes, including agriculture, smart spaces and smart cities are just some examples of the IoT applications [5][6].

The connection of physical things to the Internet makes it possible to access remote sensor data and to control the physical world from a distance. The mash-up of captured data with data retrieved from other sources, e.g., with data that is contained in the Web, gives rise to new synergistic services that go beyond the services that can be provided by an isolated embedded system. The Internet of Things is based on this vision. A smart object, which is the building block of the Internet of Things, is just another name for an embedded system that is connected to the Internet[H. Kopetz, Real-Time Systems. Design Principles for Distributed Embedded Applications. Kluwer Academic Publications, 1997].

Everyday physical things that are enhanced by a small electronic device to provide local intelligence and connectivity to the cyberspace established by the internet. The small electronic device, a *computational component* that is attached to a *physical thing*, bridges the gap between the physical world and the information world. A *smart object* is thus a *cyber-physical system* or an *embedded system* consisting of a *thing* (the physical entity) and a *component* (the computer) that processes the sensor data and supports a wireless communication link to the Internet[H. Kopetz, Real-Time Systems. Design Principles for Distributed Embedded Applications. Kluwer Academic Publications, 1997].

1.6 Hardware

One of the fundamental requirements for developing ubiquitous systems is the use of hardware such as sensors, microcontrollers, communication devices (network cards, Bluetooth, etc.) and storage, among others. For example, sensors allow transform use of interactive environments from more transparent interfaces. Currently, there are some platforms that allow insert and control various types of sensors, using of communication interfaces and storage units. This section presents and details the main hardware devices available for the development of ubiquitous systems.

1.6.1 Sensors and Actuators

Sensors are devices that allow us to capture information from the environment in which they are inserted, such as temperature, pressure, presence, humidity, smoke detector, light intensity, among others. In general, the sensors work transforming parts of a physical quantity into an electrical signal, which in turn can be interpreted by electronic devices [7]. In other words, sensors are components that allow an electronic device to interact with the real world.

According to [7], when the sensors operate directly, transforming one form of energy into another are called transducers. The sensors where operations occur indirectly alter their properties, such as resistance, capacitance or inductance, under the action of physical grandeur so that this change is roughly proportional. For example, the light sensor LDR (Light-dependent resistors) vary inversely its resistance the amount of light falling on it. Thus, when there is a large amount of light falling on the sensor, they have a very low resistance and this allows the flow of electric current increases, whereas when there is little light, they have a high resistance and prevent current flow.

An actuator as well as a sensor is a transducer that converts

one form of energy into another, and can also do the opposite [7]. In other words, rather than just transform parts of a physical quantity into an electrical signal, it can transform an electrical signal into a physical quantity such as motion, magnetism, heat, among others. For example, the relays are electromechanical devices that work with small power, but are able to control external circuits that involve high currents. They are basically composed of a coil and a set of contacts. When a current flows through the coil it creates a magnetic field that attracts and closes the contacts, remaining as long as power supply in the coil. As a result, it allows the passage of energy through the relay.

1.6.2 Arduino

Arduino was created in 2005 by Massimo Banzi and David Mellis in Italy with the goal of use as an electronic learning tool and programming for design students, so that they would use in art projects, interactivity and robotics. Electronic learning was expensive: a microcontroller was costing 100 euros. So they decided to make their own board. Sought employees and thus created an efficient technology, accessible and compatible with Windows, Mac and Linux.

Arduino is a platform that popularizes the concept of free hardware. In the book Getting Started with Arduino, Massimo Banzi describes the Arduino as a physical open-source computing platform based on a simple board with input/output pins that implements the Processing language. It is a small but powerful board constituted by a microcontroller that can be easily programmed via a Universal Serial Bus interface (USB) and able to build electronic devices and interesting systems.

Working with the construction of a hardware requires much time and effort, because is necessary to create new circuits, use various components such as resistors and capacitors and many welds. Using Arduino board abstracts much of this construction process, making it simpler, allowing people from different fields of knowledge being able to build their projects. The platform is widely used worldwide for offering advantages such as:

- A multiplatform environment that can run all major operating systems such as Windows, Linux and MacOS.
- Is an open-source hardware, ie, the circuit design is available so that if someone is interested in creating your own card, just buy the necessary components.
- Hardware cost is low.
- Is possible program it via a USB cable instead of a serial port. Remember that today's computers do not have serial ports.
- It has a development environment with intuitive interface for easy use.

The fact that both the hardware and the Arduino software be developed in an open, patent-free, allows its projects to be recreated in different ways. The adoption of open hardware concept motivates those who create projects to contribute with functions and libraries for the Arduino. Is knowledge about knowledge, the same principle of free software.

Note that there is not necessary advanced knowledge in electronics to use the platform. But for those who are interested in deepening the knowledge in this area, there are several available materials that can help, for example, in [19].

1.6.3 BeagleBone

The BeagleBoard is a low-power open-source hardware single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The BeagleBoard was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip.[7] The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence OrCAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used.

BeagleBone is an $89 Manufacturer suggested retail price (MSRP), credit-card-sized Linux computer that connects to the Internet and runs software such as Android 4.0 and Ubuntu. With plenty of I/O and processing power for real-time analysis provided by an AM335x 720MHz ARM® processor, BeagleBone can be complemented with cape plug-in boards to augment functionality.

1.6.4 Raspberry Pi

Raspberry Pi(RPi) is a small computer, approximately with the size of a credit card. It can be used to do many things that are done by a common personal computer [RASPBERY PI FUNDATION, 2014a]. In addition, it can also be used in electronic projects because it has a hardware interface: The general purpose input/output port (GPIO). The Raspberry Pi Foundation, creator of the project, is an educational charity headquartered in the UK and aims to help and encourage the teaching of computer science in schools.

This computer came up with the intention of reconnecting children and youth in computer programming and stimulate the creation of new projects so there is not only the consumption of the technology created by the market. While in the 1990s there was a growth in the number of children and youth who developed programming skills, starting in the 2000s, this number started to decline. [Raspbery PI Foundation, 2014b].

With the realization that, year by year, the students were moving away from programming and reducing the development of skills in computer science. The researchers Eben Upton, Rob Mullins, Jack Lang and Alan Mycrof, from the Computer Laboratory of University of Cambridge had the idea of creating a platform that would allow to reconcile the programming students and handling computers.[3] From 2006 to 2008, early versions of what is now the Raspberry Pi was developed. From 2008, with the expansion and emphasis on mobile devices, the devices are becoming more efficient and cheaper which enabled the launch of the Raspberry Pi model B to the public in February 2012 for $ 35.

The proposal of the Raspberry Pi is therefore be a low-

cost computer, with the ability to interact with the outside world through the sensor coupling. As seen previously, it was thought to be used in the educational environment, in order to assist and encourage the teaching of programming and help understand the operation of computers. However, the Raspberry Pi has been used by people of all ages and interests in various projects, for example, projects involving automation, sensing and robotics, games, multimedia, networks and servers.

Currently there are three Raspberry Pi models: The model A, which costs $ 25 and the B and B+ models, which costs $ 35. As for the price, there is not much difference between the hardware models. The main differences are the amount of USB ports (1, 2 and 4 in the models A, B and B +, respectively), the Ethernet port (model A does not have) and ram (256MB on the model A against 512 for the others).

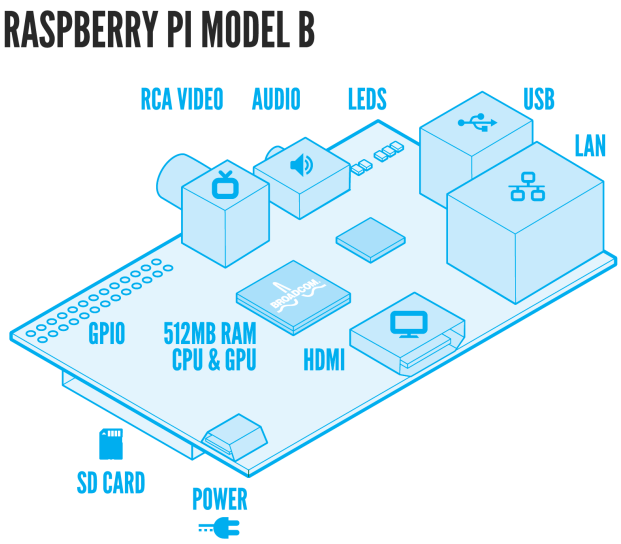

Figure 3: Raspberry Pi Components, model B. Font: (RASPBERY PI FUNDATION, 2014c)

1.6.4.1 Features

The original RPi is based on the Broadcom BCM2835 system on a chip (SoC),[1] which includes an ARM1176JZF-S 700 MHz processor, VideoCore IV GPU,[8] and was originally shipped with 256 megabytes of RAM, later upgraded (models B and B+) to 512 MB.[9] The system has Secure Digital (SD) (models A and B) or MicroSD (models A+ and B+) sockets for boot media and persistent storage.[10]

In 2014, the Raspberry Pi Foundation launched the Compute Module, which packages a BCM2835 with 512 MB RAM and an eMMC flash chip into a module for use as a part of embedded systems.[11] The Foundation provides Debian and Arch Linux ARM distributions for download.[12] Tools are available for Python as the main programming language, with support for BBC BASIC (via the RISC OS image or the Brandy Basic clone for Linux),[14] C, C++, Java, Perl and Ruby.

Some other features presents on RPI:

- Display Serial Interface Connector (DSI): This connector receives a flat-ribbon cable of 15 pins that can be used to communicate with an LCD or organic light-emitting diode (OLED) display screen;
- Camera Serial Interface(CSI) Connector: This port allows a camera module be directly coupled to the card;
- P2 and P3 connectors: These two rows of connectors are JTAG connectors for test to Broadcom chip (P2) and LAN9512 network (P3). Due to the proprietary nature of Broadcom chipset, these connectors are unlikely to be of much used;
- Pin Protection: Most of the pins in the header go directly to the Broadcom chip. It is important to carefully design the components you attach to them as there is a risk you will permanently damage your Pi. Short circuits and wiring mistakes could also ruin your day so double check everything. A multimeter is probably going to help a lot here as you can double check wiring before you connect to the Pi.

1.6.4.2 Power Supply

The device is powered by 5v micro USB. Exactly how much current (mA) the Raspberry Pi requires is dependent on what you hook up to it. We have found that purchasing a 1.2A (1200mA) power supply from a reputable retailer will provide you with ample power to run your Raspberry Pi for most applications, though you may want to get a 2.5A (2500mA) if you want to use all 4 USB ports on the Model B without using an external powered USB hub.

The power requirements of the Raspberry Pi increase as you make use of the various interfaces on the Raspberry Pi. The GPIO pins can draw 50mA safely (that is 50mA distributed across all the pins! An individual GPIO pin can only safely draw 16mA), the HDMI port uses 50mA, the camera module requires 250mA, and keyboards and mice can take as little as 100mA or over 1000mA! Check the power rating of the devices you plan to connect to the Pi and purchase a power supply accordingly. If you're not sure, buy a powered hub.

1.6.4.3 Processor

The System on a chip(SoC) used in the first generation Raspberry Pi is somewhat equivalent to the chip used in older smartphones (such as iPhone / 3G / 3GS). The Raspberry Pi is based on the Broadcom BCM2835 system on a chip (SoC),[1] which includes an 700 MHz ARM1176JZF-S processor, VideoCore IV GPU,[8] and RAM. It has a Level 2 cache of 128 KB, used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible.

1.6.4.4 General Purpose Input/Output

Another important aspect of hardware from Raspberry is the group of GPIO pins. These pins are programmable ports to input and output data used to provide an interface between the board and peripherals, microcontrollers/microprocessors, sensors, actuators, etc. The GPIO inter-

face is fundamental for building intelligent interactive environments. It is the interface between the Raspberry and the real world. In simple terms, you can consider the GPIO pins as switches that can be turned on/off.

Similarly to the Arduino, besides the GPIO, the Raspberry also supports PWM (Pulse Width Modulation), UART (Universal asynchronous receiver/transmitter) and SPI (Serial Peripheral Interface). Of the 26 pins available, 17 are reserved for GPIO and 8 are used for power and ground. Figure 2.6 shows the GPIO interface. The pins can be used via code written in a programming compatible language like Python, Scratch, Java, among others [8]. The GPIO pins are available on the PCB via a header and allow you to interface the Pi to the real world.

1.6.4.5 Inputs and Outputs

The number and options of input and output from a Raspberry Pi depend on the model used. Models RPi A+, B+ and 2B GPIO J8 have 40-pin as pinout. Models A and B have only the first 26 pins.In this paper we will present only the pins availlable in the model B.The pins are divided into:

- Digital pin to input or output (programmable) - 17 pins;
- Analog input pins or digital input/output - 6 pins;
- Power pins (gnd, 5V, 3.3V) - 9 pins;

The first item on the list are the useful pins. They are the ones that are available for a programmer use. Is through these pins that the Raspberry Pi is coupled to the sensors to capture environmental information. Among the 17 digital input / output pins there are 2 pins that match the serial communication module UART (Universal asynchronous receiver/transmitter). This module allows communication between a computer (for example) and the Raspberry Pi (see Special Pins section). All pins have more than one function, pins can be input or output, and the roles of each are defined in the code of possible programs that can be run on the RPi.

1.6.4.6 Digital Inputs

All 17 programable pins can be used as digital inputs. When a pin is programmed to function as digital input, you use a command that, when executed, performs the "reading" of the voltage applied to it. Then, after running this command, you can know if the pin is in a "high" or "low" state (on or off).

From the electrical point of view, the program can tell if a pin is fed with 0 (zero) or 5 Volts. This function is usually used to identify whether a button is pressed or a sensor is capturing some environmental information. Note that the digital input function delivers only the values 0 or 1 (no voltage or with voltage). You can not know how much voltage is being applied to the pin.

1.6.4.7 Analog Inputs

The Raspberry Pi does not have analog pins as Arduino, but is possible to use a traditional analog-to-digital converter(ADC) chip connecting to the R-Pi via SPI or I2C. For example the Microchip MCP3424 (I2C interface) which has 4 differential inputs. All ADC should work on the R-Pi, except that the timing is not as good as a microcontroller due to the available Linux versions not being true real-time.

1.6.4.8 Digital outputs

With a digital output is possible only two types of values (0 or 5 volts, 1 or 0, etc.). From a pin programmed as digital output is possible for example light up an LED, connecting a relay, trigger a motor, etc. You can program the RPi for all 17 digital outputs.

1.6.4.9 Special Pins

RPi pins have some special characteristics which can be used from the software functions encoded by a software programmer. Are they:

- PWM - Pulse Width Modulation [1]: Treated as analog output, it is actually a digital output that generates an alternating signal (between 0 and 1) where the time that the pin is in level 1 (on) is controlled. It is used, for example, for engine speed control, or generate voltages with values controlled by the program. For the PWM communication may use pin 12;
- UART - Universal asynchronous receiver/transmitter: One pin(TxD) is used to transmit and another(RxD) to receive data in serial asynchronous format. For example, you can connect a data transmission module via bluetooth and allow communication with the Arduino remotely. Are reserved for UART pins 15 (RXD receives data) and 14 (TXD sends data);
- SPI Port - Serial Peripheral Interface [8]: These are pins that allow synchronous serial communication faster than UART. They allow for example to connect memory cards (SD) and many other things. Are used for this purpose the pins 19 as Master Output, Slave Input (MOSI), 21 as Master Input, Slave Output(MISO), 23 as Serial Clock(SCLK), 24 as Chip Select0(CE0) e 26 as Chip Select1(CE1).
- I2C Bus - Inter-Integrated Circuit: The I2C bus allows multiple devices to be connected to the Raspberry Pi, each with a unique address, that can often be set by changing jumper settings on the module. It is very useful to be able to see which devices are connected to the RPi as a way of making sure everything is working.Are used for this purpose the pins 3 for Serial Data Line(SDA) and pin 5 for Serial Clock Line(SCL)

1.7 Software

This section presents all software solutions which are involved with the proposed model.

1.7.1 Operating systems

Raspberry Pi primarily uses Linux-kernel-based operating systems.
The ARM11 chip at the heart of the Pi (pre-Pi 2) is based on version 6 of the ARM. The current releases of several popular versions of Linux, including Ubuntu,[65] will not run on the ARM11. It is not possible to run Windows on the original Raspberry Pi, though the new Raspberry Pi 2 will be

able to run Windows 10.[67] The Raspberry Pi 2 currently only supports Ubuntu Snappy Core, Raspbian, OpenELEC and RISC OS.

The install manager for the Raspberry Pi is NOOBS. The operating systems included with NOOBS are:

- Archlinux ARM;
- OpenELEC;
- Pidora (Fedora Remix);
- Puppy Linux;
- Raspbmc and the XBMC open source digital media center;
- RISC OS – The operating system of the first ARM-based computer;
- Raspbian (recommended for Raspberry Pi and used in this project)[74] – Maintained independently of the Foundation; based on the ARM hard-float (armhf) Debian 7 'Wheezy' architecture port originally designed for ARMv7 and later processors (with Jazelle RCT/ ThumbEE, VFPv3, and NEON SIMD extensions), compiled for the more limited ARMv6 instruction set of the Raspberry Pi. A minimum size of 4 GB SD card is required. There is a Pi Store for exchanging programs.[77]
  - The Raspbian Server Edition is a stripped version with fewer software packages bundled as compared to the usual desktop computer oriented Raspbian.[78][79]
  - The Wayland display server protocol enable the efficient use of the GPU for hardware accelerated GUI drawing functions.[80] on 16 April 2014 a GUI shell for Weston called Maynard was released.
  - PiBang Linux is derived from Raspbian.[81]
  - Raspbian for Robots[82] - A fork of Raspbian for robotics projects with LEGO, Grove, and Arduino.

A list of other operating systems that can be installed on Raspberry Pi but are not included with NOOBS can be found in the appendix.

1.7.2 WiringPi

WiringPi is a library for access to GPIO interface written in C. Its use can be performed with C, C ++, or other programming languages through wrappers [7]. Wrapper is an outer layer that extends WiringPi and can be implemented in different programming languages. This will allow the programmer to carry out projects not only in C or C ++, but also in language implemented by the wrapper. There are wrappers being developed in various languages such as Java, Ruby and Pyton. This last will be used in this paper from the wrapper raspberry-gpio-python.

1.7.3 Python

Python is considered a very high level language because its syntax is simple and its dynamic typingin addition to being interpreted which makes it great for scripting and robust for various paradigms including object orientation. With all these benefits of language, Python still surprises to be an open source software being available for all major operating systems. []

1.7.4 GPIO in Python

The easiest way to control the GPIO pins is using the module RPi.GPIO Python library. The RPi.GPIO module is installed by default in Raspbian, but if needed, installing the library is easy if followed the RPi.GPIO Installation Guide. Once installed using the pins is as easy as below:

```
1  import RPi.GPIO as GPIO
2
3  # Use GPIO numbers, not pin numbers
4  GPIO.setmode(GPIO.BCM)
5
6  # set up GPIO channels - one input and one output
7  GPIO.setup(7, GPIO.IN)
8  GPIO.setup(8, GPIO.OUT)
9
10 # input from GPIO7
11 input_value = GPIO.input(7)
12
13 # output to GPIO8
14 GPIO.output(8, True)
```

Any RPi.GPIO script must be run as root because this library needs to acces /dev/mem and for safety reasons, it is not recommended to provide access permission to this directory.

1.7.4.1 GPIO.BOARD and GPIO.BCM

The GPIO.BOARD option specifies that you are referring to the pins by the number of the pin the the plug - i.e the numbers printed on the board (e.g. P1) and in the middle of the diagram on figure 4.

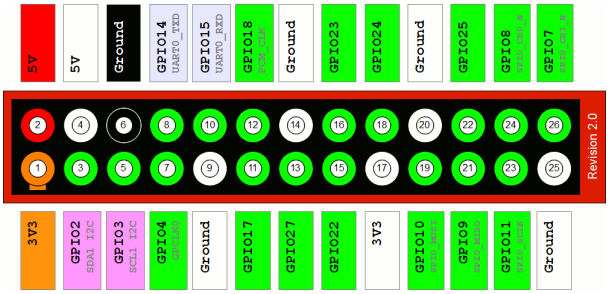

Figure 4: GPIO pins. Font: simple-guide-to-the-rpi-gpio-header-and-pins

The GPIO.BCM option means that you are referring to the pins by the "Broadcom SOC channel" number, these are the numbers after "GPIO" in the green rectangles around the

outside of the diagram above.

Unfortunately the BCM numbers changed between versions of the Model B, and you'll need to work out which one you have guide here. So it may be safer to use the BOARD numbers if you are going to use more than one pi in a project.

1.7.4.2 Pigpio

Pigpio is a Python module for the Raspberry which talks to the pigpio daemon to allow control of the general purpose input outputs (GPIOs). Pigpio Python scripts may be run on Windows, Macs, and Linux machines. Only the pigpio daemon needs to be running on the Pi.

Pigpio provides all the standard gpio features and in addition it provides hardware timed PWM suitable for servos, LEDs, and motors and samples/timestamps gpios 0-31 up to 1 million times per second (default 200 thousand).

2 JUSTIFICATION

As with other platforms, Raspberry Pi allows coupling several sensors whose handling can be made from raspberry-gpio-python or any other API available. On other platforms, such as the Arduino, the APIs provide libraries that encapsulate the complexity of implementation and offer only the interface to use. These libraries do not yet exist formally for those who want to use Python as a development language for Raspberry Pi.

This may be a consequence of run under the Linux kernel which is not suitable for real time applications - it is multitasking O/S and another process may be given priority over the CPU, causing jitter in the program[??].

The library proposed reconciles the use of an integrated circuit platform with microcontroller , thereby strengthening the use of free hardware, aiming to solve problems of a physical nature, with several components used to the extent that platform, allowing it to have a greater range of utility, potentiating their functions. Among the components which have greater relevance are the sensors that capture and process variations in the environment into electrical signals that are identified by the circuit platform. With the use of a high-level programming language make the communication interface between the computer and the microcontroller, capturing the data sent to the computer, allowing the user to make the necessary analysis.

3 RELATED WORK

As mentioned in section 2, there is no specific library to work with python applied to Raspberry IP, which provides a wide variety of sensors but there are some projects of such libraries for Arduino, and libraries for a particular set of sensors, which will be shown below.

PrivateEyePi[] is a project developed for security/automation that uses binds programming and eletronics.This is an open source project that is free of charge and can be copied, shared and modified without restriction. The user can use the Raspberry Pi or an tiny wireless Arduino to connect sensors and send data over the Internet.

PrivateEyePi Provides tutorials for users explaining how to build, wire, convert the sensor to a wireless battery operated IOT device and how to connect it to the Internet. The project also provides a cloud based alarm system where the client can group sensors using zones. Zones can be activated and alarms triggered based on rules that can be defined.

Some sensors, like relay switches, can be controlled through The Internet. users can also control the alarm system through the PrivateEyePi web based dashboard, which allows monitor the status of sensors and view temperature and humidity readings in real time from The Internet. This dashboard can be seen in figure 5.

Historical information is provided by the analytics module. The library provides also a sophisticated rules engine that permit create rules that are processed in real time to create alerts. Those methods require parameters like sensor values, time of day, days of week, alarm activated/deactivated to define rules specific to individual sensors. Sensors that PrivateEyePi provides PIR motion sensor, DS18B20 digital thermometer, DHT22 for temperature and humidity readings and a generic water sensor.

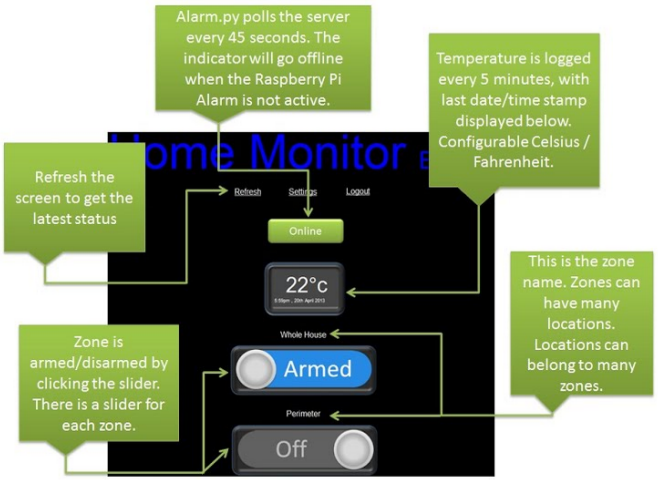

Figure 5: PrivateEyePi web based dashboard. Font: projects.privateeyepi

Pingo is an uniform python API to program devices like the Raspberry Pi, Arduino, pcDuino, Intel Galileo etc. It's an object-oriented API where each board is an instance of a Board subclass. Every board has a dictionary called pins which lists all GPIO pins on the board. Each pin is an instance of a Pin subclass with attributes that users can inspect to learn about its capabilities.

To use pingo, the first step is to instantiate a Board. Each Pingo driver is a concrete board subclass.Two such classes are *pingo.rpi.RaspberryPi* and *pingo.arduino.ArduinoFirmata*. Pingo can automatically detect the board in most common cases. *pingo.detect.MyBoard()* will return an suitable board instance if the script is running on a supported board. If Pingo is running on an unsupported machine (eg. a notebook), it will try to find a connected Arduino using the

Firmata protocol via USB and – if successful – will return a *pingo.arduino.ArduinoFirmata* instance.

Once having a board instance, it's possible to access its pins through the board.pins dictionary:

```
import pingo
from time import sleep

board = pingo.detect.MyBoard()
led_pin = board.pins[13]
led_pin.mode = pingo.OUT

while True:
  led_pin.hi()
  sleep(1)
  led_pin.lo()
  sleep(1)
```

[] describes a library that uses python called pySerial using serial port to communicate with Arduino mainly, but also with Python running on Windows, Linux, BSD (possibly any POSIX compliant system), Jython and IronPython (.NET and Mono). It encapsulates the access for the serial port and provides backends for Python. The module named "serial" automatically selects the appropriate backend.

Features:

- Same class based interface on all supported platforms.
- Access to the port settings through Python properties.
- Support for different byte sizes, stop bits, parity and flow control with RTS/CTS(Request to Send / Clear to Send) and/or Xon/Xoff.
- Working with or without receive timeout.
- File like API with "read" and "write" ("readline" etc. also supported).
- The files in this package are 100% pure Python.
- The port is set up for binary transmission. No NULL byte stripping, carriage return-linefeed translation(CR-LF) etc. (which are many times enabled for POSIX.) This makes this module universally useful.
- Compatible with io library (Python 2.6+)
- RFC 2217 client (experimental), server provided in the examples

This work [??] describes the use of LM35 (temperature sensor) and a Light Dependent Resistor.

Arduino DHTLib [??] is a library for reading temperature and humidity from the sensors of HDT11ś famly, such as DHT11, DHT21, DHT22 DHT33 e DHT44, applied to Arduino.

The DHT11/21/22 has three lines, GND, +5V and a single data line. By means of a handshake the values are clocked out over the single digital line. Handshake for DHT21/22 is identical, but data format are different. The library is rewritten from scratch and is not compatible with earlier DHT libraries to be able to support both DHT's and stay as simple as possible and to minimize footprint.

The interface supports only one function for reading the humidity and temperature from the sensors and store it in two members of the class. The read() function verifies the checksum of the data transmission and it has a time out function. If there is a checksum error the values of temperature and/or humidity might still be valid.

The class has 6 read functions read11(PIN), read(PIN) and readxx(PIN) which have essentially the same interface. They read the DHT connected to PIN, and fill the two class members temperature and humidity. Multiple reads from these class members (Humidity and Temperature) will return the same (previous) values until a new read is done.

The *readXX()* functions return:

- DHTLIB_OK (0) : if the sensor sample and its checksum is OK.
- DHTLIB_ERROR_CHECKSUM (-1) : if the checksum test failed. This means that data was received but may be incorrect.
- DHTLIB_ERROR_TIMEOUT (-2) : if a timeout occured, communication failed.

In case of a DHTLIB_ERROR_TIMEOUT, humidity and temperature will get the value DHTLIB_INVALID_VALUE. In case of DHTLIB_ERROR_CHECKSUM the values of humidity and temperature are left unchanged as it is impossible to determine which byte failed in the checksum. It is up to the programmer to decide what to do. One can compare with previous value, but better reread the sensor.

NewPing Library for Arduino[] is an ultrasonic sensor library for arduino that was developed to work with the sensors SR04, SRF05, SRF06, DYP-ME007 and Parallax PING)))™. It is written with c++ and intended to use with Sketches (Software written using Arduino).

Features:

- Works with many different ultrasonic sensor models: SRF05, SRF06, DYP-ME007, Parallax PING)))™ and SR04.
- Option to interface with all but the SRF06 sensor using only one Arduino pin.
- Doesn't lag for a full second if no ping echo is received like all other ultrasonic libraries.
- Ping sensors consistently and reliably at up to 30 times per second.
- Timer interrupt method for event-driven sketches.
- Built-in digital filter method ping_median() for easy error correction.
- Uses port registers when accessing pins for faster execution and smaller code size.

- Allows setting of a maximum distance where pings beyond that distance are read as no ping "clear".
- Ease of using multiple sensors (example sketch that pings 15 sensors).
- More accurate distance calculation (cm, inches and microseconds).
- Doesn't use pulseIn, which is slow and gives incorrect results with some ultrasonic sensor models.
- Actively developed with features being added and bugs/ issues addressed.

Adafruit's Raspberry-Pi Python Code Library[] is the most closed work with this paper. It is a growing collection of libraries and example python scripts written by Limor Fried, Kevin Townsend and Mikey Sklar under BSD license for controlling a variety of Adafruit electronics with a Raspberry Pi.

This library provides a collection of python scripts to work with sensors, providing classes that can be used to measure values from the sensor that the class implements.

Despite being a library with a considerable amount of sensors, if the user wants to change the type of sensor in a project, it is necessary review the documentation of the new sensor to be used because the library does not use a standard for equivalent sensors.

4 PROPOSED MODEL

Many small embedded systems exist to collect data from sensors, analyse the data, and either take an appropriate action or send that sensor data to another system for processing. One of the many challenges of embedded systems design is the fact that parts that are used today may be out of production tomorrow, or system requirements may change and may will be needed to choose a different sensor down the road.

Creating new drivers is a relatively easy task, but integrating them into existing systems is both error prone and time consuming since sensors rarely use the exact same units of measurement. By reducing all data to a single sensors/event family type and settling on specific, standardised SI units for each sensor family the same sensor types return values that are comparable with any other similar sensor. This enables users to switch sensor models with very little impact on the rest of the system, which can help mitigate some of the risks and problems of sensor availability and code reuse.

LibsensorPy is an extensible library, which allows the user to interact with environment through sensors and actuators coupled to the Raspberry Pi as well as add new sensors / actuators and easy way to practice. Often programmers, who have no knowledge in electronics, and engineers who do not have programming experience need to create ubiquitous systems, requiring a lot of work to implement system's abstractions, configure logic connections between sensors and the microcontroller, capture, understand and process the data in order to make them readable.

With the proposed system, the user only need to worry about using the data that the solution delivery already processed since the system provides the abstractions of the main necessary resources for the processing and, if desired, inform the library in which pins the sensors are connected, if the user does not want to use the default configuration of each sensor. Conversions between units of measure are also provided, facilitating and universalizing the use of the library.

LibsensorPy provides a simple abstraction layer between user's application and the actual sensor Hardware, allowing to drop in any comparable sensor with only one or two lines of code to change in the project that uses the library. This change is essentially in the constructor since the functions to read data and get information about the sensor are defined in the family sensor class, e.g. UltrasonicSensor class.

This is imporant useful for two reasons:

1. Users can use the data right away because it's already converted to SI units that is understandable and can compare, rather than meaningless values like 0..1023.
2. Because SI units are standardised in the sensor library, users can also do quick sanity checks working with new sensors, or drop in any comparable sensor if needed better sensitivity or if a lower cost unit becomes available, etc.

Light sensors will always report units in lux, pressure sensors will always report units in hPa and so forth, freeing user up to focus on the data, rather than digging through the datasheet to understand what the sensor's raw numbers really mean. Also the library offer methods to convert the standard SI to other measurement unit, thus who are using the library can abstracts these conversions. A sheet of Standardised SI values and measurable units and conversions for each sensor can be seen in appendix XXXX.

By reducing all data to a single sensors 'type' and settling on specific, for each sensor family, the same sensor types return values that are comparable with any other similar sensor. This enables the user to switch sensor models with very little impact on the rest of the system, which can help mitigate some of the risks and problems of sensor availability and code reuse.

The purpose of this work is the focus on library's usability and the abstraction from the way that this library interacts with the sensors / actuators to capture the data, besides contributing to the open source community and the growing community of developers for the Raspberry Pi.

The technologies used to develop the tool were basically the Python language and RPI.GPIO and PigPio modules that provide connection abstractions to basic pins and the special pins I2C, UART and SPI PWM.

4.1 Method/Methodology

To develop this project we adopted the object-oriented analysis and design (OOAD). OOAD is a popular technical approach to analyzing, designing an application, system, or business by applying the object-oriented (OO) paradigm and visual modeling throughout the development life cycles to

foster better stakeholder communication and product quality.

According to the popular guide Unified Process, OOAD in modern software engineering is best conducted in an iterative and incremental way. Iteration by iteration, the outputs of OOAD activities, analysis models for Object-oriented analysis (OOA) and design models for Object- oriented design (OOD) respectively, were refined and evolved continuously driven by key factors like usability and efficiency.

The object-oriented approach is ambitious: it encompasses the entire software lifecycle. When examining object-oriented solutions, we should check that the method and language, as well as the supporting tools, apply to analysis and design as well as implementation and maintenance. The language, in particular, should be a vehicle for thought which will help through all stages of your work. In this context, Python was essentially fundamental since it provides fully object oriented functionalities that attend the project's requirements.

Using Object-oriented modeling, we divided the work in two aspects: The modeling of dynamic behaviors like events and use cases, and the modeling of static structures like classes and components. OOA and OOD were the two distinct abstract levels (i.e. the analysis level and the design level) during OOM. The Unified Modeling Language (UML), a popular international standard languages used for object-oriented modeling was used.[7]
The benefits of OOM are:

4.1.1 Efficient and effective communication

Users typically have difficulties in understanding comprehensive documents and programming language codes well. Visual model diagrams can be more understandable and can allow users and stakeholders to give developers feedback on the appropriate requirements and structure of the system. A key goal of the object-oriented approach is to decrease the "semantic gap" between the system and the real world, and to have the system be constructed using terminology that is almost the same as the stakeholders use in everyday business. Object-oriented modeling is an essential tool to facilitate this.

4.1.2 Useful and stable abstraction

Modeling helps coding. A goal of most modern software methodologies is to first address "what" questions and then address "how" questions, i.e. first determine the functionality the system is to provide without consideration of implementation constraints, and then consider how to make specific solutions to these abstract requirements, and refine them into detailed designs and codes by constraints such as technology and budget. Object-oriented modeling enables this by producing abstract and accessible descriptions of both system requirements and designs, i.e. models that define their essential structures and behaviors like processes and objects, which are important and valuable development assets with higher abstraction levels above concrete and complex source code.

Patterns used:

- Abstract Factory;
- Singleton;
- Observer;
- Composite;

4.2 Functional Requirements

In the table 1 are defined the main functional requirements order to be able to meet all the objectives proposed by libsensorPy library:

Table 1: Functional requirements of libsensorPy

Id	Functional Requirements	Actor
RF01	Collect data sent by the sensors via the GPIO pins;	Library
RF02	Manipulate data sent by the sensors, turning them into readable data to the user;	User
RF03	Allow additions of new sensors / actuators and factories via inheritance;	User
RF04	Allow the user to set the pins to be used for connection between sensor and Raspberry;	User
RF05	Allow the creation of composite sensors, enabling a single sensor to measure more than one physical greatness;	User
RF06	Allow configuration of a set of conditions (events) que, When met, trigger an action.	User

4.3 Non-functional Requirements

Are characterized as non-functional requirements for the proper functioning of libsensorPy, the items in table 2.

Table 2: Non-functional requirements of libsensorPy

Id	Non-Functional Requirements	Category
RNF01	The library must allow adition of new sensors/families;	Extensibility
RNF02	The data reported by the application must be faithful to that presented in the environment;	Confidence
RNF03	The library should be usable in any Raspberry Pi model ;	Portability
RNF04	The library will be developed using the Python language;	Software

4.4 Why not to use Java

At the beginning of this project we decide to develop the library using Java as the programming language, Since These libraries do not yet exist formally for those who want to use Java as a development language. We thought that it could be a consequence of the recent adoption of the use of Java for Raspberry (when compared with other APIs and platforms). But after implement the first sensors,the DHT11 and HC-SR04, when tried to test, we got no data from the sensors.

After some research about what could be the problem, we found that we cannot assume that invoking sleep will suspend the thread for precisely the time period specified, since Thread.sleep() and the Thread.Join() methods are dependent on the operating system and the version of the JVM[????]

These sensors need to wait microsseconds between send a signal to the sensor and get back the data readed from it, And because of this not real time property from both java and the raspberry pi, we couldn't read data from those sensors. After that we moved on to change the programming language and decided to use python, especially because it's the recommended programming language by the Raspberry Pi developers.

4.5 Architecture

The system consists of a module that implements the abstract factory pattern, guaranteeing the independence of how products are created, composed and represented. For this, the system must be configured with one of multiple product families (family sensors). It provides one product class library: sensors, actuators and events related to the family of each sensor, but are only revealed their interfaces, not their implementations. Figure 6 shows the main class diagram.
Main components:

- AbstractSensorFactory: Declares an interface for operations that create abstract sensors and events;
- ConcreteFactories: Implement operations to create concrete sensors and events;
- AbstractSensor: Declares an interface for sensors;
- AbstractActuator: Declares an interface to the actuators;
- AbstractEvent: declares an interface for the events;
- ConcreteSensors, ConcreteActuators, ConcreteEvents: Define the concrete objects to be created by the corresponding concrete factory and implement AbstractSensor, AbstractActuator and AbstractEvent interfaces respectively.

A single instance of the concrete factory is created at runtime, using the Singleton pattern. This factory creates products with a particular implementation. To create other products from a different family, one should use a different factory. The concrete factory class being used appears only once in the application, facilitating changes. The product family changes all at once, promoting consistency across products, ensuring that used objects are all of the same family, represented by the concrete factory being used.

Due to some sensors belong to more than one family (measure more than one physical quantity), the library allows the use of composite-type sensors composed of basic sensors. This ability is facilitated by Python feature that allow multiple inheritance. For example, the DHT11 sensor is capable of measuring temperature and humidity so , were then created three sensors: Two basic sensor from distinct families (DHT11Temperature to measure temperature and DHT11Humidity to the family of humidity sensors) and a composite sensor (DHT11Composite) that aggregates the two basic sensors.
The idea of the separation of basic and composite sensors is to allow the creation of lighter objects, if the user just want a sensor that measures only a physical quantity. The composite pattern structure allows basic and composite sensors being viewed by the user in the same way. We can see this structure in figure 9.

The interactions between the user and the library are described in the use case diagram in figure 8. The sequence diagram for the creation and capture of data by the sensor can be seen in figure **??**.

4.5.1 Architectural Style

The libsensorPy presents characteristics of three architectural styles:

- Traditional, influenced by programming language: OO;
- Based on Implicit Invocation: Event Based;
- Layered Style: Virtual Machine.

The traditional style adopted reflects the basic relationships of organization and control flow between components provided by the Python language. The only provided structure is a set of objects whose lifetime varies according to its uses. This library uses connectors type procedures call and event . The Procedure Call connectors model the control flow by invocation techniques and perform data transfer between the components involved through the use of parameters. Examples of Procedure Call connectors include functions, procedures, object oriented methods, callback and system calls

The Virtual Machine style is applied when we imagine the layers between the hardware and the Raspberry py libsensorPy library using procedure calls as connectors between the layers. We have thus the layers:

- Physical: GPIO
- Operating System
- RPi.GPIO
- LibsensorPy

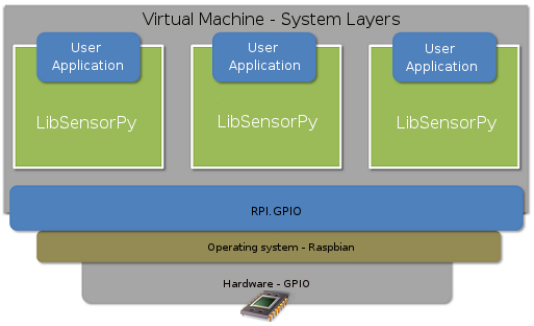

Figure 10: LibsensorPy's layers

The envent-based style is an architectural style based on implicit invocation that provides an indirect interaction between loosely coupled components facilitating the adaptation and improving system scalability. The components of

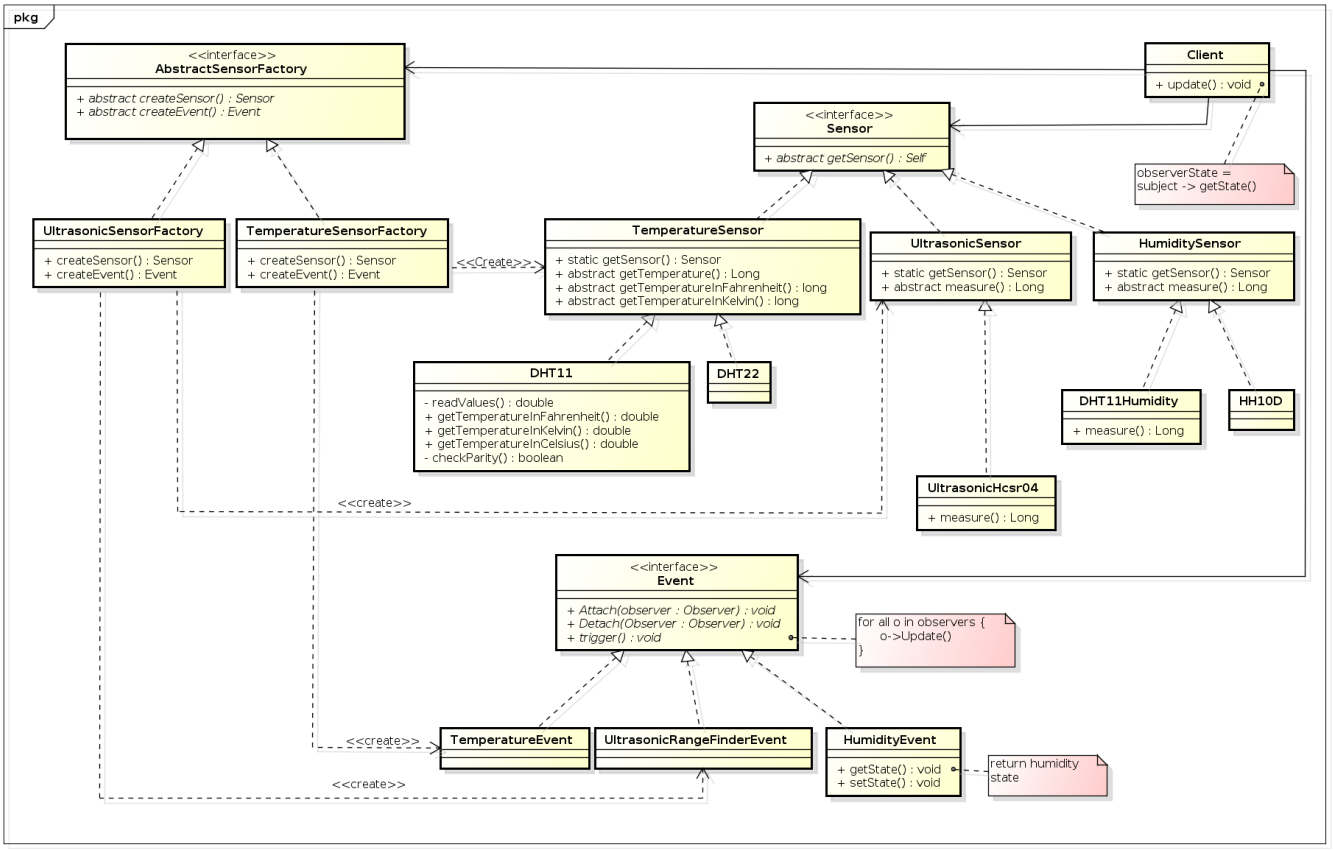

Figure 6: Class diagram

the Event type (TemperatureEvent, SmokeEvent, etc.) communicate only via events transmitted by an event connector. This connector then relays the events for all components of the Observer type showing interest in the event in question (Observer pattern), thereby improving the efficiency of distribution of events.

4.5.2 Architectural Pattern

This poojject was developed under the architectural pattern Sense-Compute-Control (SCC). This pattern is typically used in the structuring of embedded control applications.Second[20] ,a Sense/Compute/Control (SCC) application is one that interacts with the physical environment. Such applications are pervasive in domains such as building automation, assisted living, and autonomic computing. SCC applications can be defined according to an architectural pattern involving four kinds of components, organized into layers [5]:

1. Sensors at the bottom, which obtain information about the environment;
2. Then context operators, which process this information;
3. Then control operators, which use this refined information to control;
4. Actuators at the top, which finally impact the environment

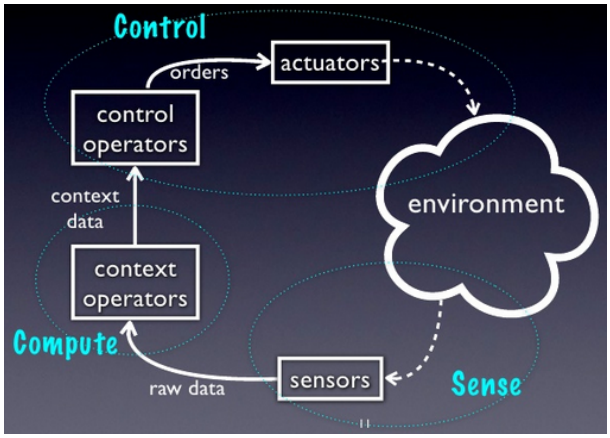

Figure 11: Sense-Compute-Control environment. Font: Damien Cassou.

Each layer corresponds to a separate class of components:

- Sensors send information sensed from the environment to the context operator layer through data sources. Sensors can both push data to context operators and respond to context operator requests. We use the term ”sensor” both for entities that actively retrieve information from the environment, such as system probes, and entities that store information previously collected

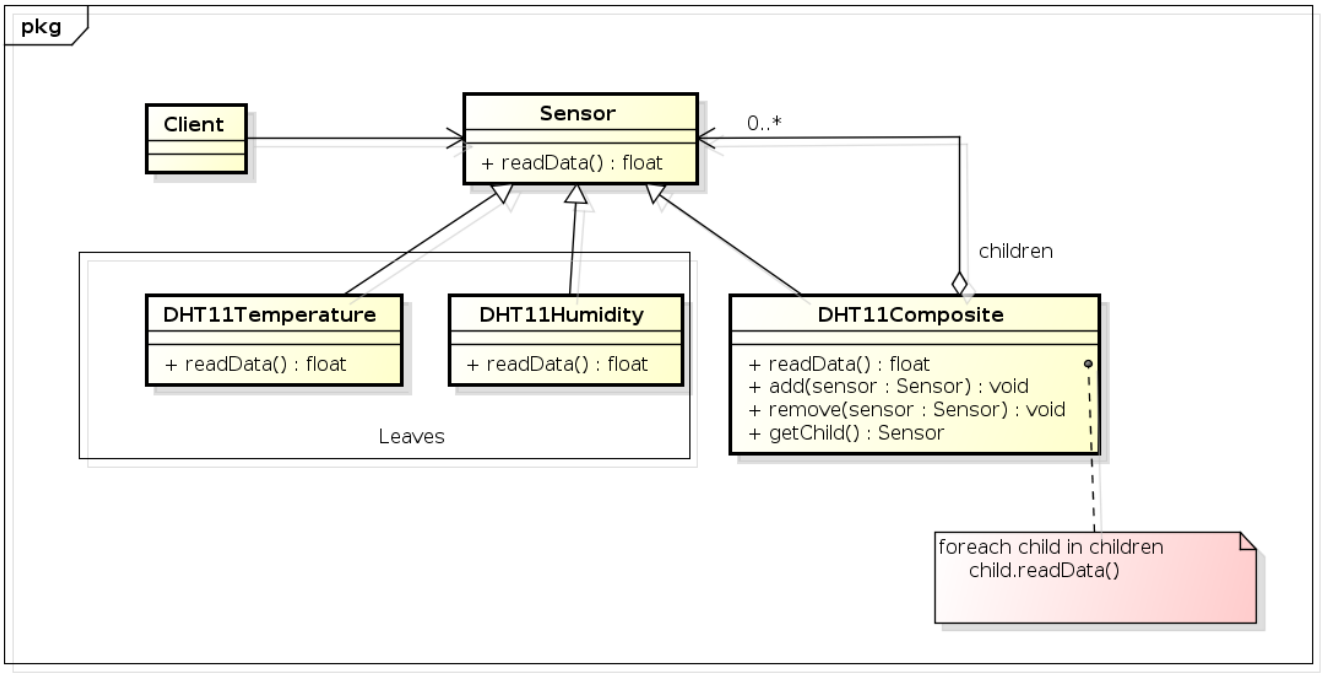

Figure 7: Composite class diagram

from the environment, such as databases.

- Context operators refine (aggregate and interpret) the information given by the sensors. Context operators can push data to other context operators and to control operators. Context operators can also respond to requests from parent context operators.
- Control operators transform the information given by the context operators into orders for the actuators.
- Actuators trigger actions on the environment.

Sensors are proactive or reactive components whereas context operators, control operators and actuators are always reactive. These properties ensure that SCC applications are reactive to the environment state. That is, all computation is initiated by an observer interaction with a sensor.

As the underlying architecture is component-based, the application can be fully distributed. To prevent concurrent handling of events in a component, all interactions of a component are queued and executed one at a time, sequentially.

The application follows the five basic principles of design oriented to objects, called "SOLID". Addressed initially by Robert Martin, in an article called Principles Of Ood, the author elaborates five-oriented programming techniques to objects where each technique is one of SOLID letters of the word. These five principles are:

- Single Responsibility Principle;
- Open Closed Principle;
- Liskov Substitution Principle;
- Interface Segregation Principle;
- Dependency Inversion Principle;

Figure 12 shows the library's structural view.

4.6 What is needed to install libsensorPy

Before use libsensorPy is necessary the user configure and enable SPI and I2C ports on raspperry:

```
1 sudo apt-get install python-smbus
2 sudo apt-get install i2c-tools
```

For Raspbian users, check /etc/modprobe.d/raspi-blacklist.conf and comment "blacklist i2c-bcm2708" and "blacklist spi-bcm2708" by running:

```
1 sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

And adding a # (if its not there), on these informations: *blacklist i2c-bcm2708* and *blacklist spi-bcm2708*.

For Wheezy or something-other-than-Occidentalis, add the following lines to /etc/modules:

```
1 i2c-dev
2 i2c-bcm2708
3 spi-bcm2708
```

Install pigpio:

```
1 wget abyz.co.uk/rpi/pigpio/pigpio.zip
2 unzip pigpio.zip
```

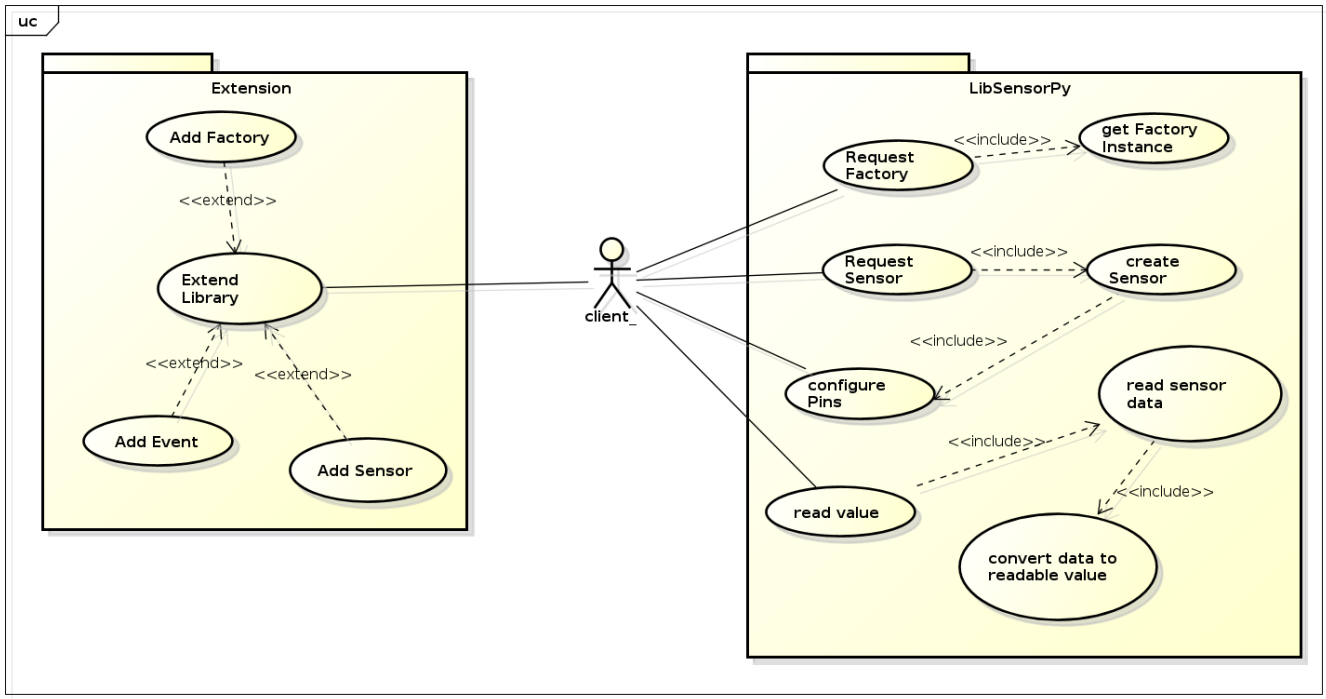

Figure 8: Use Case diagram

```
cd PIGPIO
sudo make
sudo make install
```

install pip:

```
sudo apt-get install python-pip
```

and finally install libsensorPy:

```
sudo pip install libsensorPy
```

4.7 How to extend the library

The Abstract Factory pattern, how it was implemented in this solution allows the extension of the library easily, following the open/closed principle, becoming it easy to modify and avoiding the user application does not suffer from the impact of these changes. This requires user programming skills. The user can add new sensors, events and new factories. To add new sensors, simply, besides implement the sensor class, create a concrete factory that inherits from the factory refering to the sensor family to be added and overwrite the createSensor () method, making this method to create the new sensor, if desired, or call the create method of the superclass sensor, ensuring system compatibility.

To add a new sensor family just create a new factory that implements the abstract methods of AbstractSensorFactory interface. If desired, is possible also to create an abstract family class for this , if there is not in the library, creating thus a "contract", where the attributes are specified, methods and functions that the concrete sensors classes of this family are required to implement .

Follows an example of how to extend the library: the HCSR04 class was implemented to create objects of the ultrasonic sensor HC-SR04. Being the family of ultrasonic sensors, this class was created as a subclass of UltrasonicSensor abstract class, which has the abstract methods *distance_in_cm()* and *setup()* to be implemented. It has also created a concrete factory ExtendedUltrasonicSensorFactory, which inherits the UltrasonicSensorFactory class and overrides the *createSensor()* method, as seen following:

```
'''
Created on 29/03/2015
@author: Junior Mascarenhas
File: hcsr04.py
'''
import RPi.GPIO as GPIO
import time
from abstractclass.ultrasonicSensor import \
                UltrasonicSensor

class HCSR04(UltrasonicSensor):
    '''
    classdocs
    '''

    def __init__(self,trigger=18,echo=27):
        '''
        Constructor
        '''
```

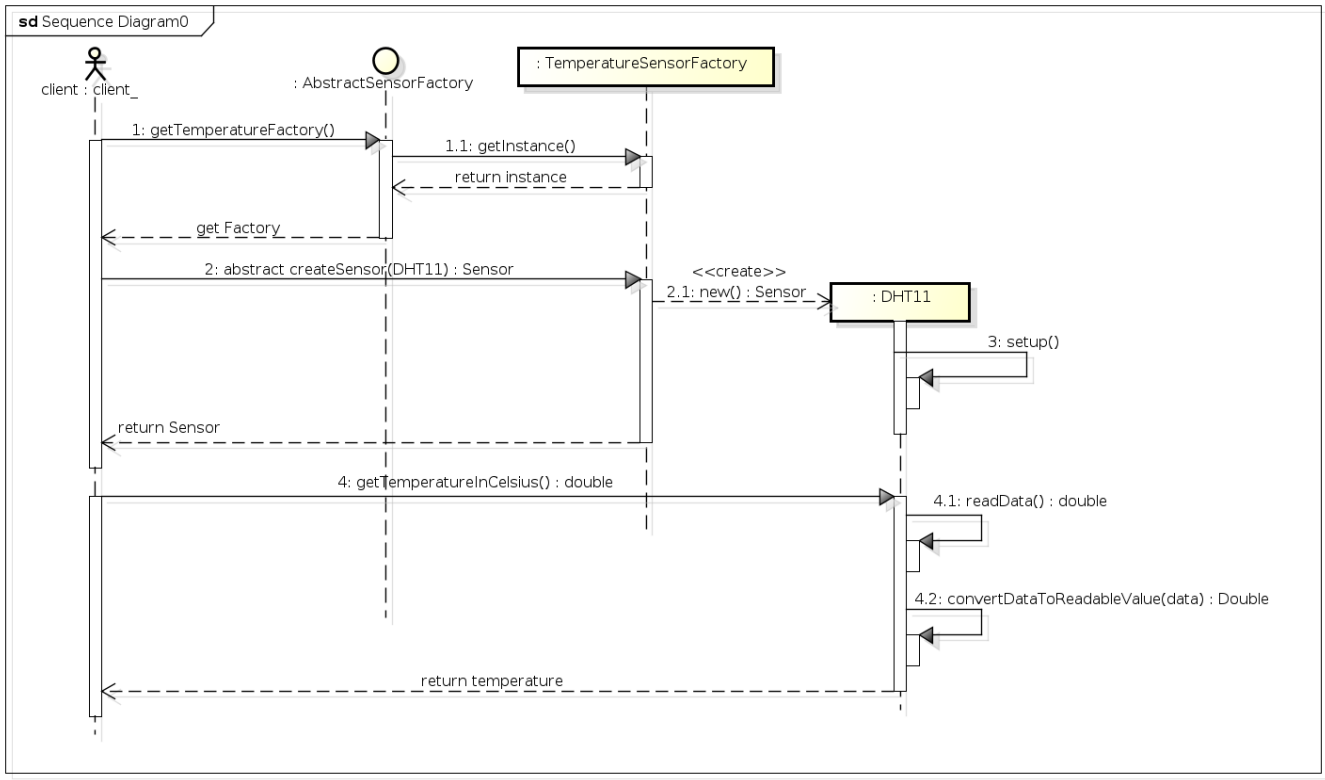

Figure 9: Sequence diagram

```
        UltrasonicSensor.__init__(self)
        self.__distance = ""
        self.__trigger = trigger
        self.__echo = echo
        self.setup()

    def setup(self):
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)

    def changeSetup(self, trigger, echo):
        self.__trigger = trigger
        self.__echo = echo

    def distance_in_cm(self):

        GPIO.setup(self.__trigger,GPIO.OUT)
        GPIO.setup(self.__echo,GPIO.IN)
        GPIO.output(self.__trigger, GPIO.LOW)
        time.sleep(0.3)
        GPIO.output(self.__trigger, True)
        time.sleep(0.00001)
        GPIO.output(self.__trigger, False)

        while (GPIO.input(self.__echo) == 0):
            signaloff = time.time()

        while GPIO.input(self.__echo) == 1:
            signalon = time.time()

        timepassed = signalon - signaloff
        self.__distance = timepassed * 17000
        return self.__distance
```

```
'''
Created on 29/03/2015
@author: Junior Mascarenhas
File: extendedUltrasonicSensorFactory.py
'''

from abstractclass.abstractSensorFactory \
    import AbstractSensorFactory
from concretefactory.ultrasonicSensorFactory \
    import UltrasonicSensorFactory
from hcsr04 import HCSR04

class ExtendedUltrasonicSensorFactory\
            (UltrasonicSensorFactory):
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
    @staticmethod
    def createSensor(sensorType):
        if (sensorType == "HCSR04"):
                return HCSR04()
        else:
```

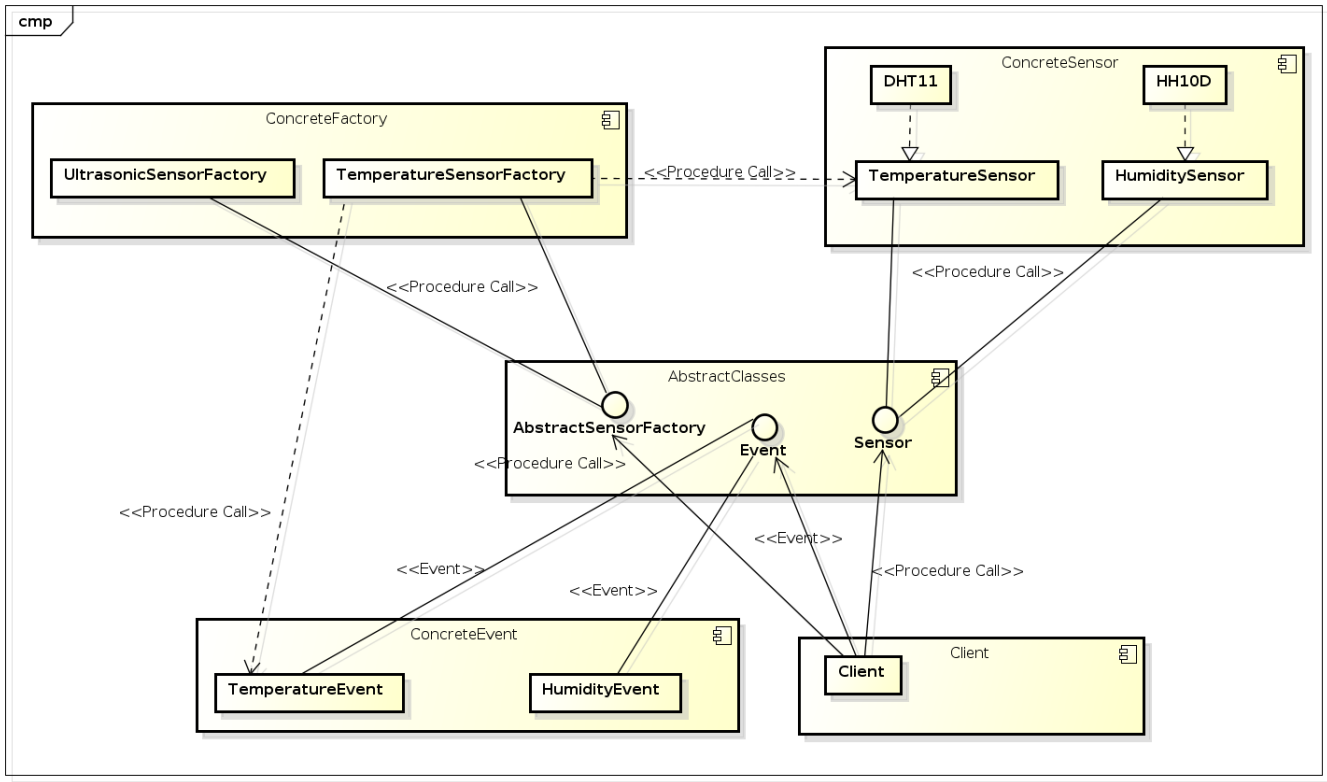

Figure 12: libsensorPy's structural view

```
28                return super(sensorType)
```

4.8 Case Study

In this work the case study began with software verification tests which consisted of functional tests of libsensorPy. These tests were performed as the system was being implemented.

Verification tests are related to the system specification, the purpose of these tests is to verify the compliance with the library's functional requirements. The tests were performed based on test cases.

Below is presented *dht11CompositeExample.py*, an example of how to use the library and we used these example classes to test the library.

```
1  '''
2  Created on 29/03/2015
3  @author: Junior Mascarenhas
4  File: dht11CompositeExample.py
5  '''
6
7  from concretefactory.compositeSensorFactory \
8          import CompositeSensorFactory
9
10 if __name__ == '__main__':
11
12   dht11 = CompositeSensorFactory.createSensor\
13             ("DHT11Composite")
14   print ("Temperature in Celsius: " \
15           + dht11.getTemperature() + u"\u00b0" + "C")
16   print ("Temperature in Fahrenheit: " \
17           + dht11.getTemperatureInFahrenheit() +"F")
18   print ("Temperature in Kelvin: " \
19           + dht11.getTemperatureInKelvin() +"K")
20   print ("Humidity: " + dht11.getHumidity() + "%")
```

Was then given to continue the study validation tests. These tests were performed after development's completion of LibsensorPy, and was based on the library's availability to the people who are involved in ubiquitous systems development activities.

This library was presented to a post graduation student in distributed and ubiquitous computing that are developing work in this area. The main objective was to determine whether libsensorPy meets the user needs.

At the end of the presentation this student were invited to complete the satisfaction of libsensorPy's survey. Figure 14 shows satisfaction survey screen using Google Forms.

4.9 Case Study results

The result of the case study was satisfactory. The tested sensors worked as expected, and the data reported were consistent with environment. The interviewed, during the case study, confirmed that libsensorPy meets your needs. However, this students made some notifications about show more

Figure 13: Testing the sensors

use examples.

We tested the sensors DHT11Temperature, DHT11Humidity, DHT11Composite and HCSR04. Figure 13 shows the results obtained. We would like to test more sensors but we just had these exemplares on hand. From HCSR04 we tested first with a distant object, and on the second reading we put the object at fifteen centimeters from the sensor. Just as a reminder, all sensors implemented but not tested were developed based on their Datasheet and specifications, as a guarantee that they will probably work as expected.

Analyzing the results obtained, it is concluded that the libsensorPy help the development of ubiquitous systems using the Raspberry Pi platform and hence can minimize the current problems faced by developers who have little knowledge of electronics discussed above. However, the suggestions received during the case study can be harnessed to the library improvement in future work.

5 CONCLUSION AND FUTURE WORK

This work presents an extensible open source library, available at Python Package Index (PyPI), the official third-party repository for the Python programming language [] and on Github [https://github.com/juniorug/libsensorPy], the Git web-based repository hosting service, Which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

Its goal is to facilitate the creation of ubiquitous systems using the Raspberry Pi. The tool is implemented in Python and is designed to facilitate the inclusion of new sensors, families and factories of sensors, as well as ease of use by the user, abstracting technical and behaviors specific to that system's type, using design patterns and following the SOLID principles.

The library's differential is the use of Abstract Factory pattern, allowing sensors and events from the same family work together, to be easily exchanged if necessary and the same type of sensors may be replaced without being necessary changes in existing code. The library also provides basic and composite sensors according to the client's needs, as well as a set of events related to specific types of sensors.

Some suggestions can be seen below:

- Test the sensors that have been implemented but have not been tested;
- Add new sensors to the library;

6 REFERENCES

[1] M. Barr. Pulse width modulation. *Embedded Systems Programming*, 14(10):103–104, 2001.

[2] A. d. L. Carvalho, F. d. Ponce de Leon, et al. Grandes desafios da pesquisa em computação no brasil–2006–2016. *São Paulo: Sociedade Brasileira de Computação*, 2006.

[3] A. K. Dennis. *Raspberry Pi Home Automation with Arduino*. Packt Publishing Ltd, 2013.

[4] O. S. DOESn't FIt All. "innovations in ubicomp products.". 2013.

[5] M. K. G. M. G. S. V. J. A. K. J. G. M. L. J. L. E. L. M. J. A. M. J. e. a. J. A. Armstrong, P. A. Ferguson. *National science foundation*, 2001.

[6] K. Krishnakumar, J. Gubbi, and R. Buyya. A framework for iot sensor data analytics and visualisation in cloud computing environments.

[7] W. PI. Wiring pi - gpio interface library for the raspberry pi – about. `http://wiringpi.com/`, October 2014. [Online; accessed 02-October-2014].

[8] A. M. Rooke. System for serial peripheral interface with embedded addressing circuit for providing portion of an address for peripheral devices, Oct. 16 2001. US Patent 6,304,921.

[9] B. Trapp. Raspberry pi: The perfect home server. linux j. *São Paulo: Sociedade Brasileira de Computação*, May 2013.

[10] M. Wirth and J. McCuaig. Making programs with the raspberry pi. In *Proceedings of the Western Canadian Conference on Computing Education*, page 17. ACM, 2014.

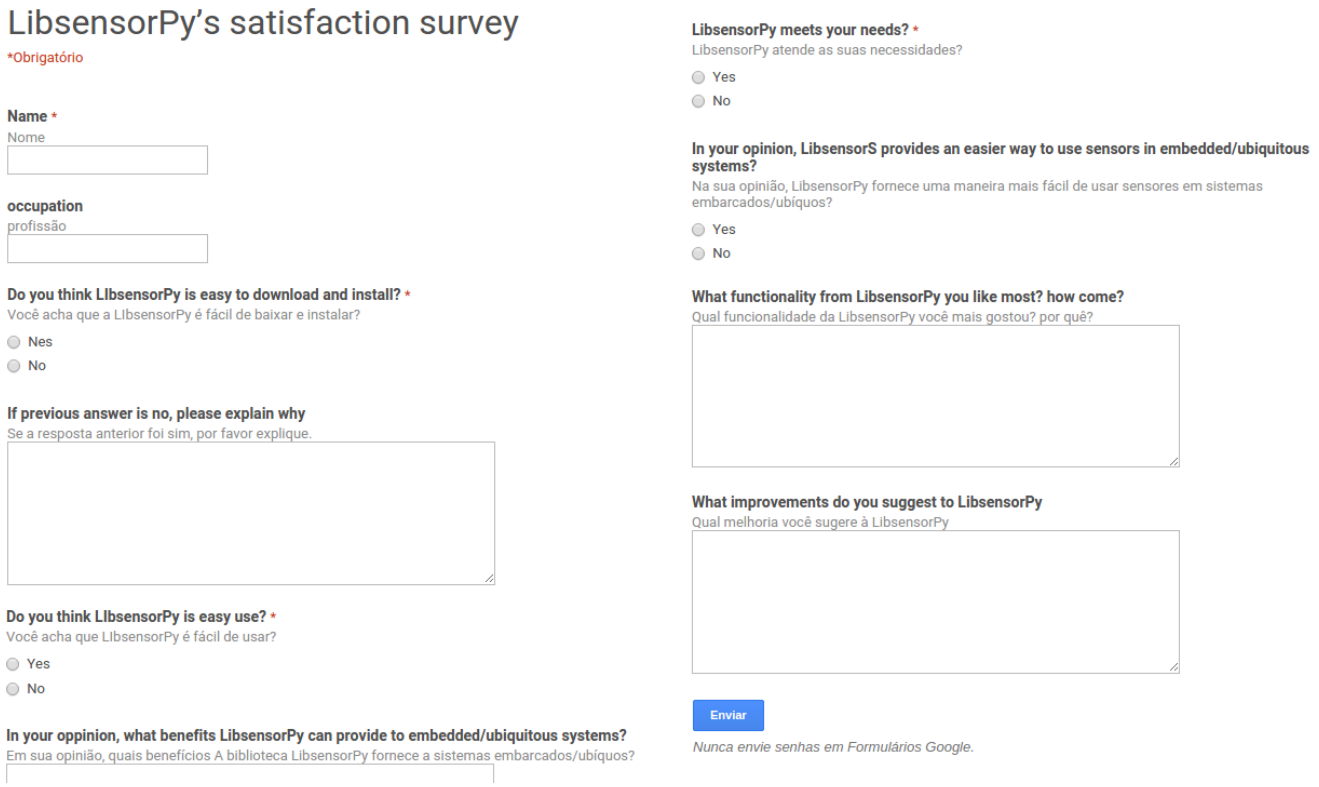

Figure 14: LibsensorPy's satisfaction survey

APPENDIX

A Headings in Appendices

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form: