

LibsensorPy: 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

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 Abstract Factory pattern to ensure sensors/actuators and events from the same family being used in conjunction with guaranteed way. On other platforms, such as Arduino, 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 development language applied to the Raspberry Pi. This 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.

Introduction

The Ubiquitous Computing (UbiComp) in its several ramifications and applications, is treated by many as the twenty-first century's new paradigm of computing. UbiComp is the area of computing that studies the coupling of 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)[2] and is also present in the report *Grandes Desafios da Pesquisa em Computação no Brasil 2006-2016*[7], published by the SBC (Brazilian Computer Society - *Sociedade Brasileira de Computação*).

Researches on ubiquitous computing are being held on topics such as: basic access to any wireless device, mobility support on the network in a transparent manner, safety, context treatment, efficient use of energy, presentation of multimedia content, among others. This work is focused on building intelligent interactive environments. In these environments, the fundamental idea is to create ways to avoid the user need to go to the computer/device, allowing many of these working at a distance. The use of platforms to integrate 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 [44, 49].

Raspberry Pi(RPi) 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 the 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.

A brief search about how to access the Raspberry Pi bus will bring a lot of papers in Python. Which incidentally, introduce the acronym of the device name (Pi from Python). However, 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 will not be found.

The developer must have a degree of experience that can be considered basic to a professional, but advanced for those who are not.

This paper is structured as follow:

Section 1 present several technologies involved in the preparation of this article, aimed to introduce important concepts

^{*}Student from the course of Analysis and Systems Development (ADS).

[†]Ph.D. in Computer Science, Professor and Researcher from the course of Analysis and Systems Development.

of Computer area in which the context of this project is inserted. Section 2 shows correlated works. Section 3 presents details of the executed implementation. 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 [46] 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 PC. He believed 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 computers without realizing or require specific technical knowledge. [47]

The passage of time has shown Weiser's bet was right. For Weiser, [48] 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 various types of applications on non-conventional computing devices, are just some of the examples that can confirm: Ubiquitous Computing (the third age) is already a reality. Figure 1 shows a chart with the ages of computing.

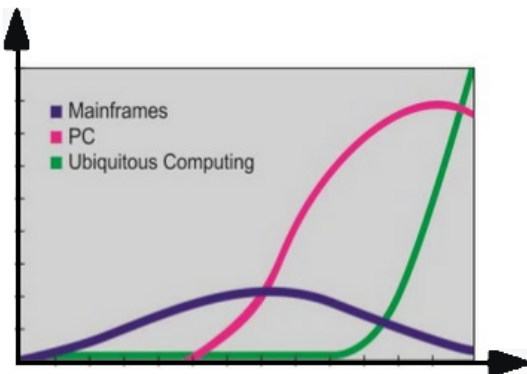


Figure 1: Ages of Computing

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 network, mobile computing has transformed the computing an activity that can be taken almost anywhere [27].

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 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 the computer is embedded invisibly in the environment to the user [26]. 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 controlling, configuring and tuning 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

Ubiquitous is an adjective originated from Latin (*ubiquu*) which means "that is at the same time everywhere". As can be seen in figure 2, the UbiComp can be defined as a computer area positioned between Mobile Computing and Pervasive Computing [12, 26]. 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.

Research in ubiquitous computing approach about technologies and infrastructure that enable 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 their services?

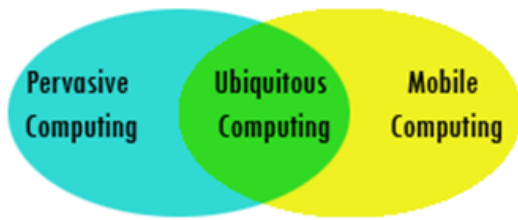


Figure 2: Ubiquitous Computing: intersection between pervasive and mobile computing

- 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 [15]. A ubiquitous system itself has a set of requirements, peculiarities and challenges that influence design, implementation, deployment and evaluation of its project [26]. These are cornerstones of UbiComp and quite different from those used in development of systems for PC's. Among these points, can be cited as an example [26]:

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 (e.g., 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. Environmental and personal health monitoring, monitoring and control of industrial processes including agriculture, smart spaces, and smart cities are just some of the examples of IoT applications [25].

The connection of physical things to the Internet makes it possible to access remote sensor data and to control 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 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 [24].

Everyday physical things 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 sensor data and supports a wireless communication link to the Internet [24].

1.6 Hardware

One fundamental requirement 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 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, sensors work transforming parts of a physical quantity into an electrical signal, which in turn can be interpreted by electronic devices [4]. In other words, sensors are components that allow an electronic device to interact with real world.

According to [4], when sensors operate directly, transforming one form of energy into another are called transducers. 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 [4]. 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, 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 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 [27].

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. Constituted by a microcontroller, this small but powerful board can be easily programmed via a Universal Serial Bus interface (USB) and able to build electronic devices and interesting systems.

Working with 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 multi-platform environment that can run all major operating systems such as Windows, Linux and MacOS.
- An open-source hardware, i.e., the circuit design is available so if someone is interested in creating their own card, just buy the necessary components.
- Hardware cost is low.
- Can be programmable 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 [21].

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 BeagleBone Black 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[8]. The board was developed by a small team of engineers as an educational board and could be used in colleges around the world to teach open source hardware and software capabilities. 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 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[18]. 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 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 [38].

With the realization that, year by year, 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 Mycroft, 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[14]. 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 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).

RASPBERRY PI MODEL B

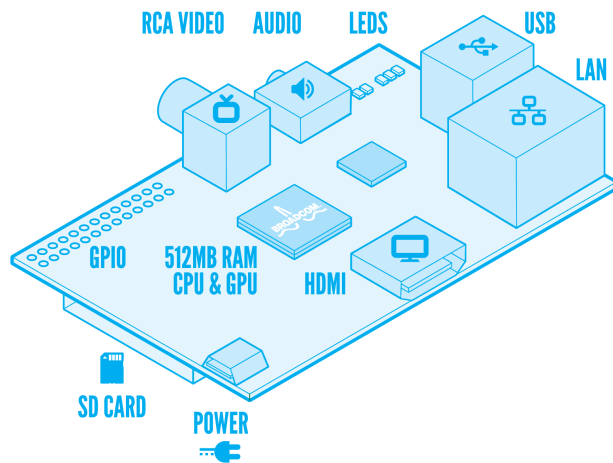


Figure 3: Raspberry Pi Components, model B[39]

1.6.4.1 Features

The original RPi is based on the Broadcom BCM2835 system on a chip (SoC)[5], which includes an ARM1176JZF-S 700 MHz processor, VideoCore IV GPU[6], and was originally shipped with 256 megabytes of RAM, later upgraded (models B and B+) to 512 MB[34]. The system has Secure Digital (SD) (models A and B) or MicroSD (models A+ and B+) sockets for boot media and persistent storage[17].

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 [35]. The Foundation provides Debian and Arch Linux ARM distributions for download. Tools are available for Python as the main programming language, with support for BBC BASIC (via RISC OS image or the Brandy Basic clone for Linux)[23], 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 and 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. Carefully design the components what can be attached to the pins is very important as there is a risk of permanently damage

the RPi. Short circuits and wiring mistakes could also shatter the board, so double check everything. A multimeter is probably going to help a lot here as the developer can double check wiring before connect to the RPi.

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 the project hook up to it. Purchasing a 1.2A (1200mA) power supply from a reputable retailer will provide to the user with ample power to run the Raspberry Pi for most applications, though may want to get a 2.5A (2500mA) if necessary 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 much as are used 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 what are planed to connect to the Pi and a power supply must be purchased accordingly.

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)[5], which includes an 700 MHz ARM1176JZF-S processor, VideoCore IV GPU[6], 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 interface is fundamental for building intelligent interactive environments and it is the interface between the Raspberry and the real world. In simple terms, the GPIO pins can be considered 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. These pins can be used via code written in a programming compatible language like Python, Scratch, Java, among others [6]. The GPIO pins are available on the PCB via a header and allow the Pi to interface with 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 [40]. In this paper will be presented only the pins available 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 programmable pins can be used as digital inputs. When a pin is programmed to function as digital input, can be used a command that, when executed, performs the "reading" of the voltage applied to it. Then, after running this command, the user will 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 values 0 or 1 (no voltage or with voltage). How much voltage is being applied to the pin is not known.

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 RPi via SPI or I2C. For example the Microchip MCP3424 (I2C interface) which has 4 differential inputs. All ADC should work on the RPi, 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. All 17 digital outputs from the RPi can be programmed.

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: Treated as analog output, this pin 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 can be used, for example, for engine speed control, or generate voltages with values controlled by the program. For the PWM communication may use pin 12 [3];
- UART - Universal asynchronous receiver/transmitter: One pin(TxD) is used to transmit and another(RxD) to receive data in serial asynchronous format. For example, to 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) [19];
- SPI Port - Serial Peripheral Interface: 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) [37];
- 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. To be able to see which devices are connected to the RPi is very useful as a way of making sure everything is working. Are used for this purpose pins 3 for Serial Data Line(SDA) and pin 5 for Serial Clock Line(SCL) [1].

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 [20], will not run on the ARM11. Run Windows on the original Raspberry Pi is not possible, though the new Raspberry Pi 2 will be able to run Windows 10[9]. 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;
- Raspbian¹ [42] – 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[41].
 - The Raspbian Server Edition is a stripped version with fewer software packages bundled as compared to the usual desktop computer oriented Raspbian[50, 36].
 - The Wayland display server protocol enable the efficient use of the GPU for hardware accelerated GUI drawing functions[45]. On 16 April 2014 a GUI shell for Weston called Maynard was released.
 - PiBang Linux is derived from Raspbian[30].
 - Raspbian for Robots[22] - 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 A².

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 [29]. 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 Python. 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 typing, besides 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[13].

1.7.4 GPIO in Python

The easiest way to control 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

¹Recommended for Raspberry Pi and used in this project

²http://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-RaspbianServerEdition-78

library is easy if followed the RPi.GPIO Installation Guide³. 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 access `/dev/mem` and for safety reasons, 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 the user are referring to the pins by the number of the pin in the plug - i.e the numbers printed on the board (e.g. P1) and in the middle of the diagram on figure 4.

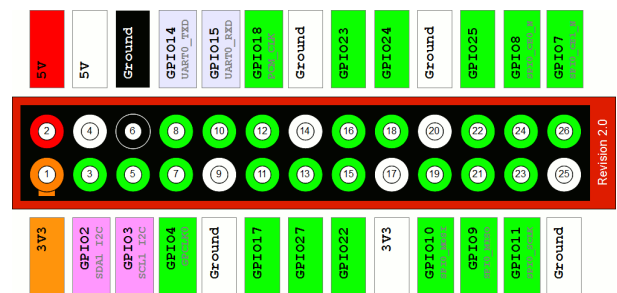


Figure 4: GPIO pins [33]

The GPIO.BCM option means that the program is 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, BCM numbers changed between versions of the Model B, and is necessary to work out which one will be used. So it may be safer to use BOARD numbers if more than one pi will be used in a project.

1.7.4.2 Pigiopio

Pigiopio [31] is a Python module for the Raspberry which talks to the pigpio daemon to allow control of the general purpose input outputs (GPIOs). Pigiopio Python scripts may be run on Windows, Macs, and Linux machines. Only the pigpio

³<http://sourceforge.net/p/raspberry-gpio-python/wiki/install/>

daemon needs to be running on the RPi.

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 RELATED WORK

As mentioned in the introduction, there is no specific library to work with python applied to Raspberry Pi, 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.

2.1 PrivateEyePi

PrivateEyePi[32] is a project developed for security/automation that uses binds programming and electronics. 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 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.

2.2 Pingo

Pingo⁴ 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*.

⁴<http://www.pingo.io/docs/>

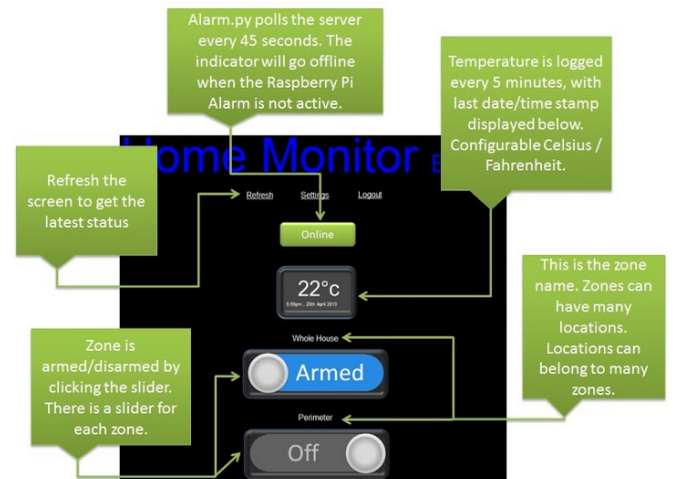


Figure 5: PrivateEyePi web based dashboard[32]

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:

```
1 import pingo
2 from time import sleep
3
4 board = pingo.detect.MyBoard()
5 led_pin = board.pins[13]
6 led_pin.mode = pingo.OUT
7
8 while True:
9     led_pin.hi()
10    sleep(1)
11    led_pin.lo()
12    sleep(1)
```

2.3 PySerial

The work from [13] 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 back-ends for Python. The module named "serial" automatically selects the appropriate back-end.

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).
- 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 I/O library (Python 2.6+)
- RFC 2217 client (experimental), server provided in the examples.

That work [13], describes the use of LM35 (temperature sensor) and a Light Dependent Resistor.

2.4 DHTLib

Arduino DHTLib⁵ is a library for reading temperature and humidity from sensors of HDT11’s family, such as DHT11, DHT21, DHT22, DHT33 e DHT44, applied to Arduino.

The interface supports only one function for reading 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.

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. The programmer will decide what to do. One can compare with previous value, but better reread the sensor.

2.5 NewPing

NewPing Library for Arduino⁶ is an ultrasonic sensor library for Arduino that was developed to work with sensors SR04, SRF05, SRF06, DYP-ME007 and Parallax PING)))TM. Intended to use with Sketches (Software written using Arduino), the library is written with C++.

Features:

⁵<http://playground.arduino.cc/Main/DHTLib>

⁶<https://code.google.com/p/arduino-new-ping/>

- Works with many different ultrasonic sensor models: SRF05, SRF06, DYP-ME007, Parallax PING)))TM 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.

2.6 Adafruit Raspberry Pi Python Code

Adafruit’s Raspberry-Pi Python Code Library⁷ is the most closed work with this paper. Written by Limor Fried, Kevin Townsend and Mikey Sklar under BSD license for controlling a variety of Adafruit electronics with a Raspberry Pi, a growing collection of libraries and example python scripts are provided.

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, is necessary review the documentation of the new sensor to be used because the library does not use a standard for equivalent sensors.

2.7 Discussions and Observations

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 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.

⁷<https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code>

This may be a consequence of run under the Linux kernel which is not suitable for real time applications, a multitasking O/S and another process may be given priority over the CPU, causing jitter in the program [28].

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 data sent to the computer, allowing the user to make the necessary analysis.

3 LIBSENSORPY

Many small embedded systems exist to collect data from sensors, analyses the data, and either take an appropriate action or send sensor data to another system for processing. One of the many challenges of embedded systems design is the fact 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, standardized 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 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 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 important 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 standardized 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 Standardized SI values and measurable units and conversions for each sensor can be seen in appendix B.

The purpose of this work is the focus on library's usability and the abstraction from the way this library interacts with sensors/actuators to capture 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, SPI and PWM.

To develop this project was 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 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.

Object-oriented approach is ambitious: it encompasses the entire software lifecycle. When examining object-oriented solutions, should be checked 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 the 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, the work is divided in two aspects: Modeling dynamic behaviors like events and use cases, and the modeling of static structures like classes and components. OOA and OOD were 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. The benefits of OOM are:

- 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.

- 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 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;

In the table 1 are defined the main functional requirements in order to be able to meet all the objectives proposed by libsensorPy library. Are characterized as non-functional requirements for the proper functioning of libsensorPy the items in table 2.

3.1 Architecture of LibsensorPy

The system consists of a module that implements the abstract factory pattern, guaranteeing independence of how products are created, composed and represented. For that, the system must be configured with one of multiple product families (family of 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

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 the user to set the pins to be used for connection between sensor and Raspberry;	User
RF04	Allow the creation of composite sensors, enabling a single sensor to measure more than one physical greatness;	User
RF05	Allow configuration of a set of conditions (events) that, When met, trigger an action.	User

Table 2: Non-functional requirements of libsensorPy

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

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 used objects are all of the same family, represented by the concrete factory being used.

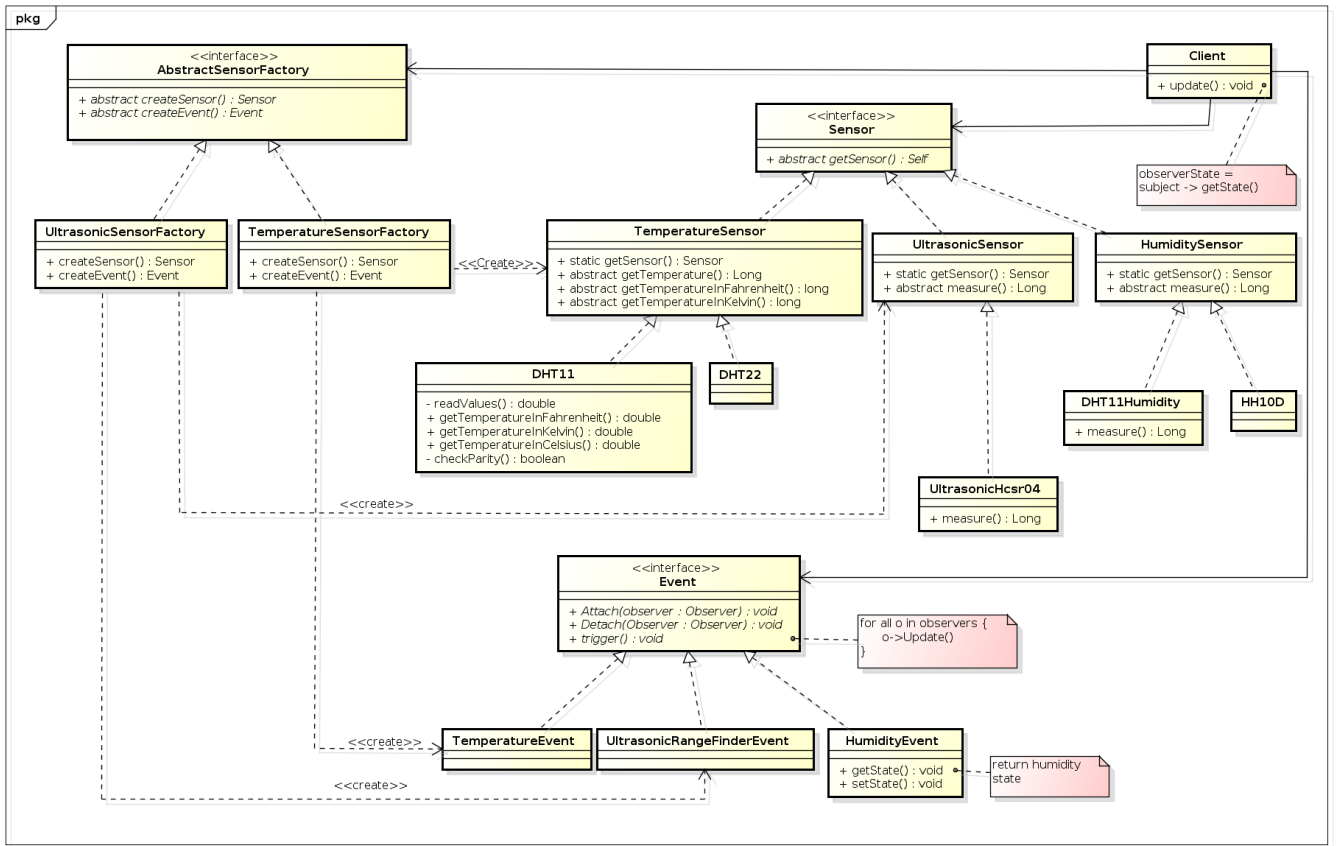


Figure 6: Class diagram

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. This structure can see in figure 7.

The interactions between the user and the library are described in the use case diagram in figure 10. The sequence diagram for the creation and capture of data by the sensor can be seen in figure 12.

3.1.1 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 Python language. The only provided structure is a set of objects whose lifetime varies according to its uses. This library uses connectors type procedure call and event. Procedure call connectors model the control flow by invocation techniques and perform data transfer between 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 between the hardware and the libsensorPy library, using procedure calls as connectors between the layers. The system has thus the layers:

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

The event-based style is an architectural style based on implicit invocation and provides an indirect interaction between loosely coupled components facilitating the adaptation and improving system scalability. The components of the Event type (TemperatureEvent, SmokeEvent, etc.) communicate only via events transmitted by an event connector.

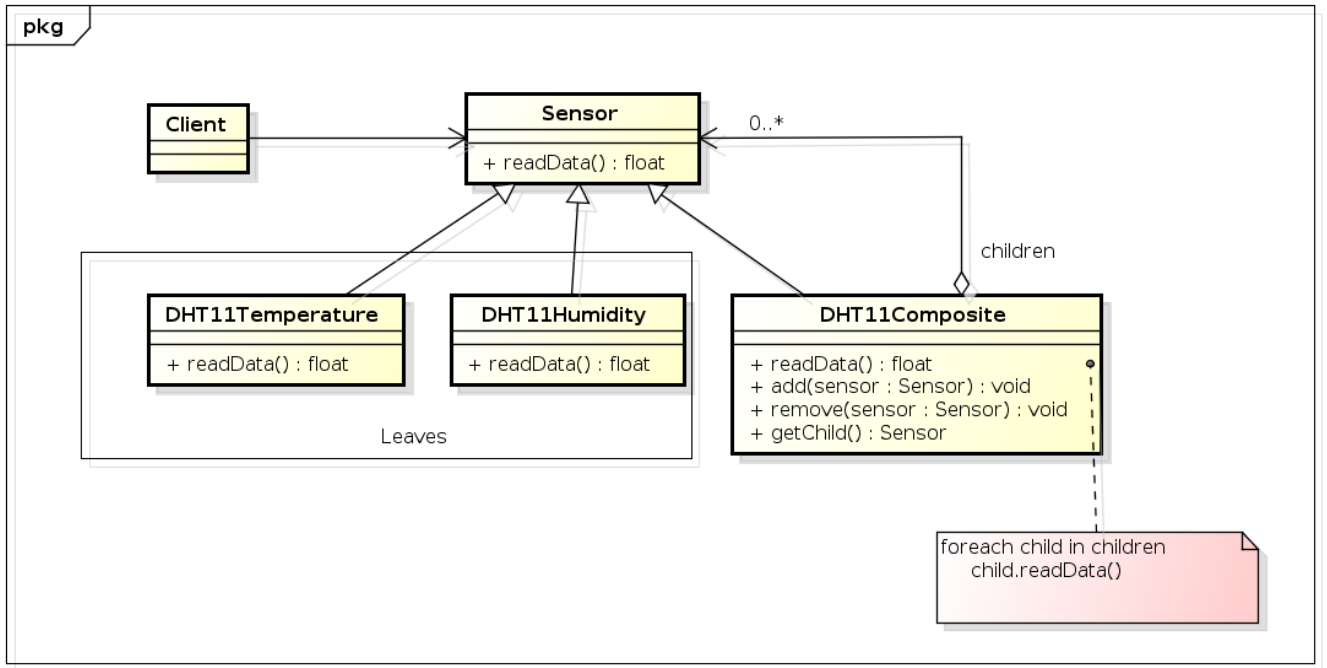


Figure 7: Composite class diagram

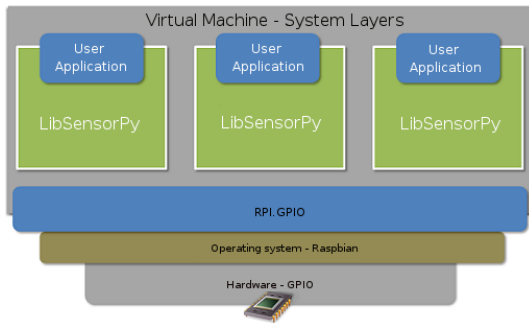


Figure 10: LibsensorPy's layers

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.

3.1.2 Architectural Pattern

This project was developed under the architectural pattern Sense-Compute-Control (SCC). This pattern is typically used in the structuring of embedded control applications. Second [43], 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 [16]:

1. Sensors at the bottom, which obtain information about the environment;
2. Then context operators, which process this informa-

tion;

3. Then control operators, which use this refined information to control;
4. Actuators at the top, which finally impact the environment

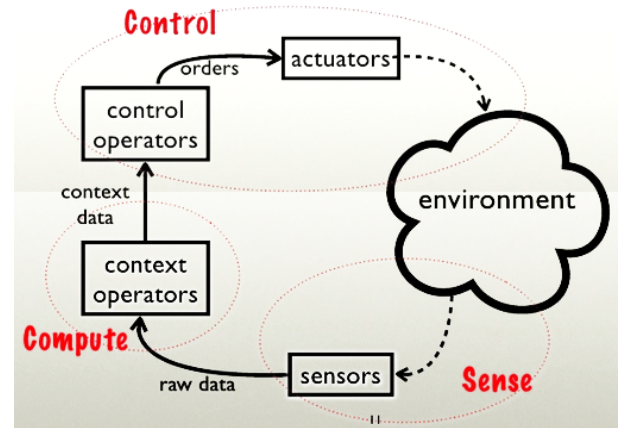


Figure 11: Sense-Compute-Control environment [10].

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. The term "sensor" is used both for entities that actively retrieve information from the environment, such as system probes,

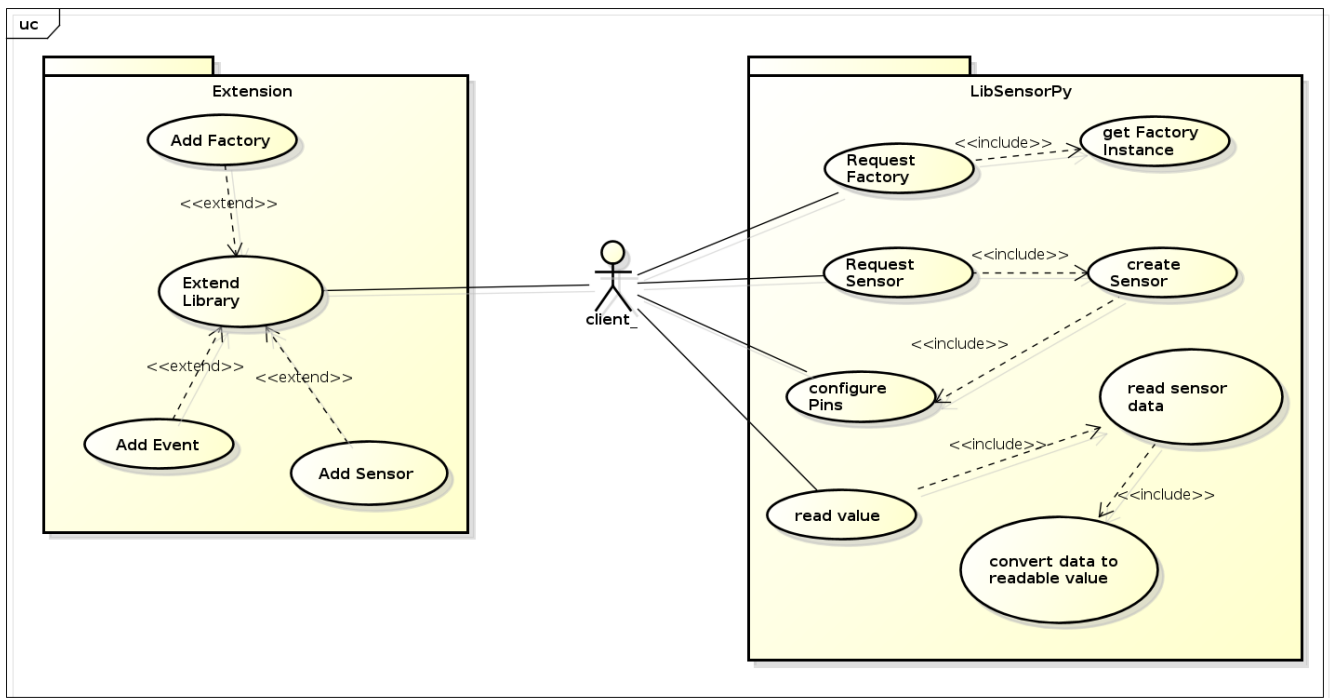


Figure 8: Use Case diagram

and entities that store information previously collected 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 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" [11]. 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 13 shows the library's structural view.

3.2 What is needed to install LibsensorPy?

Before use LibsensorPy is necessary to configure and enable SPI and I2C ports on rasperry:

```

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

```

Check `/etc/modprobe.d/raspi-blacklist.conf` if using Raspbian, 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-Occidentals, add the following lines to `/etc/modules`:

```

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

```

Install pigpio:

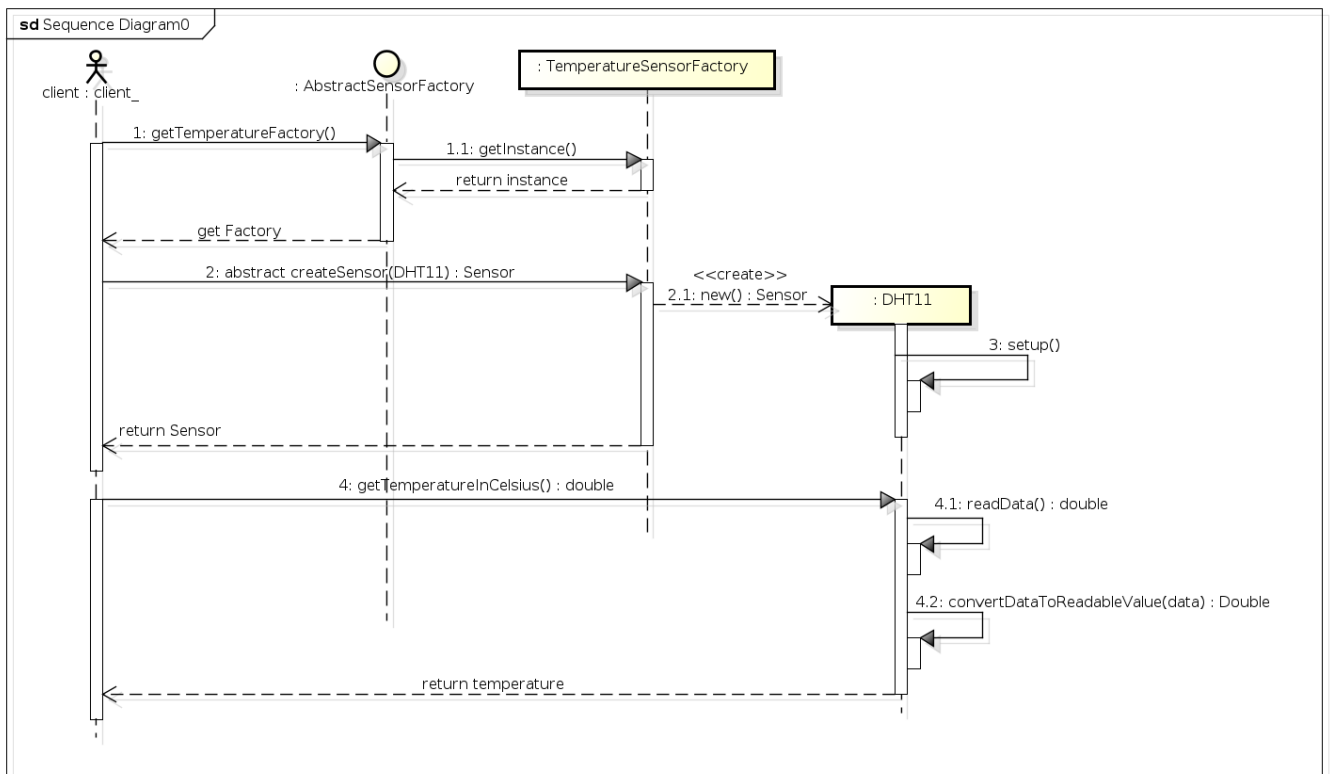


Figure 9: Sequence diagram

```

1 wget abyz.co.uk/rpi/pigpio/pigpio.zip
2 unzip pigpio.zip
3 cd PIGPIO
4 sudo make
5 sudo make install

```

install pip:

```

1 sudo apt-get install python-pip

```

and finally install LibsensorPy:

```

1 sudo pip install LibsensorPy

```

3.3 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 referring 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 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:

```

1 '''
2 Created on 29/03/2015
3 @author: Junior Mascarenhas
4 File: hcsr04.py
5 '''
6 import RPi.GPIO as GPIO
7 import time
8 from abstractclass.ultrasonicSensor import \
9     UltrasonicSensor
10
11 class HCSR04(UltrasonicSensor):

```

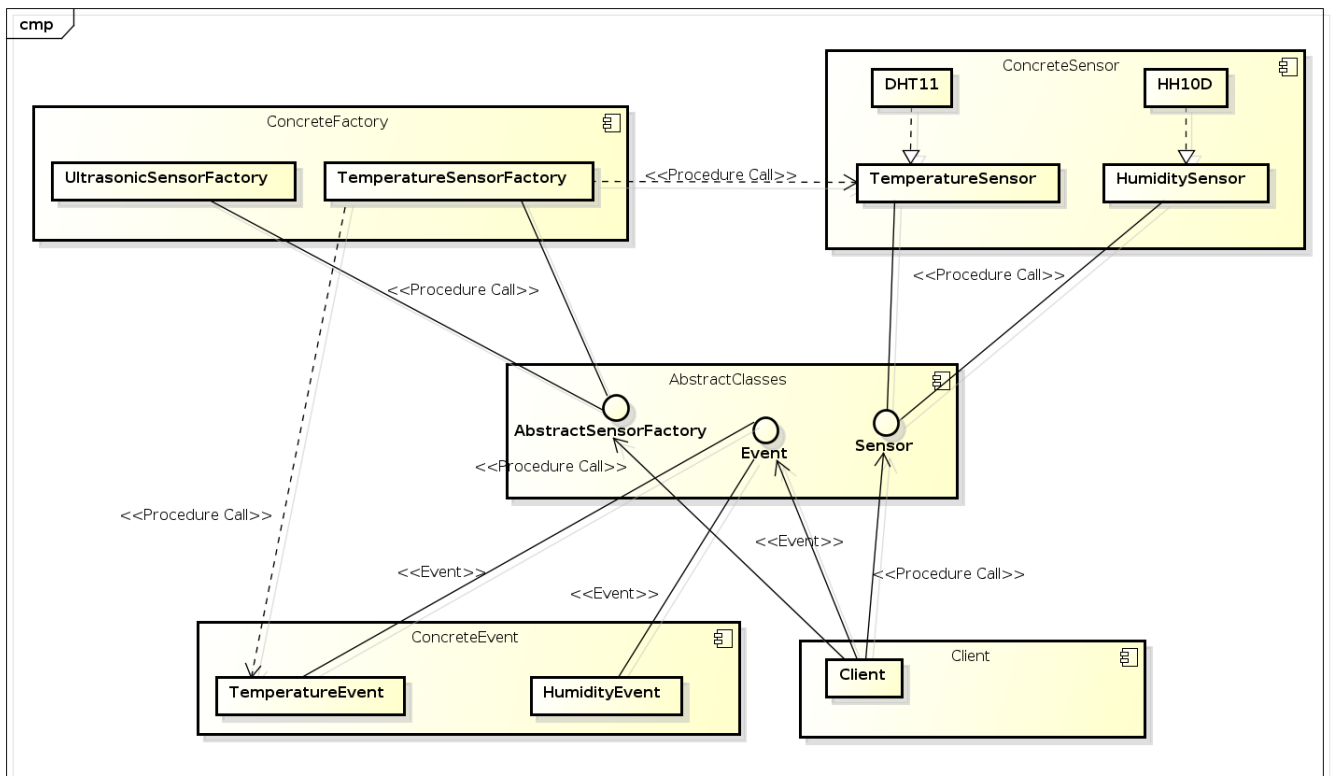


Figure 12: LibsensorPy's structural view

```

12  """
13  classdocs
14  """
15
16  def __init__(self, trigger = 18, echo = 27):
17      """
18      Constructor
19      """
20      UltrasonicSensor.__init__(self)
21      self.__distance = ""
22      self.__trigger = trigger
23      self.__echo = echo
24      self.setup()
25
26  def setup(self):
27      GPIO.setmode(GPIO.BCM)
28      GPIO.setwarnings(False)
29
30  def changeSetup(self, trigger, echo):
31      self.__trigger = trigger
32      self.__echo = echo
33
34  def distance_in_cm(self):
35
36      GPIO.setup(self.__trigger, GPIO.OUT)
37      GPIO.setup(self.__echo, GPIO.IN)
38      GPIO.output(self.__trigger, GPIO.LOW)
39      time.sleep(0.3)
40      GPIO.output(self.__trigger, True)
41      time.sleep(0.00001)
42      GPIO.output(self.__trigger, False)

```

```

43
44      while (GPIO.input(self.__echo) == 0):
45          signaloff = time.time()
46
47      while GPIO.input(self.__echo) == 1:
48          signalon = time.time()
49
50      timepassed = signalon - signaloff
51      self.__distance = timepassed * 17000
52      return self.__distance

```

```

1  """
2  Created on 29/03/2015
3  @author: Junior Mascarenhas
4  File: extendedUltrasonicSensorFactory.py
5  """
6
7  from abstractclass.abstractSensorFactory \
8      import AbstractSensorFactory
9  from concretefactory.ultrasonicSensorFactory \
10     import UltrasonicSensorFactory
11  from hcsr04 import HCSR04
12
13  class ExtendedUltrasonicSensorFactory\
14      (UltrasonicSensorFactory):
15
16      """
17      classdocs
18      """

```

```

pi@raspberrypi ~/Documents/libsensorPy/examples $ pwd
/home/pi/Documents/libsensorPy/examples
pi@raspberrypi ~/Documents/libsensorPy/examples $ sudo python dht11TemperatureExample.py
Temperature in Celsius: 29°C
Temperature in Fahrenheit: 86.0F
Temperature in Kelvin: 302.15K
pi@raspberrypi ~/Documents/libsensorPy/examples $ sudo python dht11HumidityExample.py
Humidity: 46%
pi@raspberrypi ~/Documents/libsensorPy/examples $ sudo python dht11CompositeExample.py
Temperature in Celsius: 30°C
Temperature in Fahrenheit: 86.0F
Temperature in Kelvin: 302.15K
Humidity: 46%
pi@raspberrypi ~/Documents/libsensorPy/examples $ sudo python ../extension/hcsr04ExtensionExample.py
Distance in cm: 213.315486908 cm
Distance in inches: 84.4977220774 in
pi@raspberrypi ~/Documents/libsensorPy/examples $ sudo python ../extension/hcsr04ExtensionExample.py
Distance in cm: 15.0978565216 cm
Distance in inches: 6.1307243824 in
pi@raspberrypi ~/Documents/libsensorPy/examples $ 

```

Figure 13: Testing the sensors

```

19 def __init__(self):
20     '''
21     Constructor
22     '''
23     @staticmethod
24     def createSensor(sensorType):
25         if (sensorType == "HCSR04"):
26             return HCSR04()
27         else:
28             return super(sensorType)

```

```

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 who 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 15 shows satisfaction survey screen using Google Forms.

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

The sensors HCSR04, DHT11Temperature, DHT11Humidity and DHT11Composite were tested. Figure 14 shows the results obtained. It would be tested more sensors but just these exemplars were available. The HCSR04 was tested first with a distant object, and on the second reading was 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 they will probably work as expected.

Analyzing the results obtained, was concluded that the LibsensorPy help the development of ubiquitous systems using the Raspberry Pi platform and hence can minimize 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

3.4 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. Tests were performed based on test cases.

Below is presented *dht11CompositeExample.py*, an example of how to use the library and these example classes were used 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: " \

```

LibsensorPy's satisfaction survey

*Obrigatório

Name *

Nome

occupation

profissão

Do you think LibsensorPy is easy to download and install? *

Você acha que a LibsensorPy é fácil de baixar e instalar?

☐ Yes

☐ No

If previous answer is no, please explain why

Se a resposta anterior foi sim, por favor explique.

Do you think LibsensorPy is easy use? *

Você acha que LibsensorPy é fácil de usar?

☐ Yes

☐ No

In your opinion, what benefits LibsensorPy can provide to embedded/ubiquitous systems?

Em sua opinião, quais benefícios A biblioteca LibsensorPy fornece a sistemas embarcados/ubiquos?

LibsensorPy meets your needs? *

LibsensorPy atende as suas necessidades?

☐ Yes

☐ No

In your opinion, LibsensorS provides an easier way to use sensors in embedded/ubiquitous systems?

Na sua opinião, LibsensorPy fornece uma maneira mais fácil de usar sensores em sistemas embarcados/ubiquos?

☐ Yes

☐ No

What functionality from LibsensorPy you like most? how come?

Qual funcionalidade da LibsensorPy você mais gostou? por quê?

What improvements do you suggest to LibsensorPy

Qual melhoria você sugere à LibsensorPy

Enviar

Nunca envie senhas em Formulários Google.

Figure 14: LibsensorPy's satisfaction survey

improvement in future work.

4 CONCLUSION AND FUTURE WORK

This work presents an extensible open source library, available at Python Package Index (PyPI)⁸, the official third-party repository for the Python programming language. The latest version can be downloaded and installed using:

```
sudo pip install LibsensorPy
```

The source code is available on Github⁹, 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. All documentation can be found at the official website¹⁰.

Its goal is to simplify 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 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

⁸<https://pypi.python.org/packages/source/l/LibsensorPy/LibsensorPy-0.0.7.tar.gz>

⁹<https://github.com/juniorug/LibsensorPy>

¹⁰<http://LibsensorPy.com>

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 and unit conversions.

Some suggestions can be seen below:

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

5 REFERENCES

- [1] Adafruit. Configuring i2c. <https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c>, September 2014. [Online; accessed 15-March-2015].
- [2] J. A. Armstrong, P. A. Ferguson, M. K. Gaillard, M. Greenwood, S. V. Jaskolski, A. K. Jones, G. M. Langford, J. Lubchenco, E. L. Menger, J. A. Miller Jr, et al. National science foundation. 2001.
- [3] M. Barr. Pulse width modulation. *Embedded Systems Programming*, 14(10):103–104, 2001.
- [4] L. P. Borges and R. de Carvalho Dores. Automação predial sem fio utilizando bacnet/zigbee com foco em economia de energia. *Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG*, (06), 2010.
- [5] Broadcom. Bcm2835 media processor. <http://www.broadcom.com/products/BCM2835>,

- September 2012. [Online; accessed 06-May-2014].
- [6] M. Brose. Broadcom bcm2835 soc has the most powerful mobile gpu in the world? <http://www.grandmax.net/2012/01/broadcom-bcm2835-soc-has-powerful.html>, January 2012. [Online; accessed 13-March-2015].
 - [7] 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.
 - [8] G. Coley. Take advantage of open-source hardware. *EDN*, 54(16):20–23, 2009.
 - [9] K. Dallas. Windows 10 coming to raspberry pi 2. building apps for windows. <http://blogs.windows.com/buildingapps/2015/02/02/windows-10-coming-to-raspberry-pi-2/>, February 2014. [Online; accessed 25-March-2015].
 - [10] C. C. J. L. Damien Cassou, Émile Balland. Architecture-driven-programming for sense/compute/control applications. <http://pt.slideshare.net/DamienCassou/architecture-driven-programming-for-sensecompute/control-applications>, May 2010. [Online; accessed 19-March-2015].
 - [11] M. A. P. A. Daniel Pace Schmitz. Utilização dos princípios solid na aplicação de padrões de projeto. *Revista Engenharia de Software Magazine* 50, 2006.
 - [12] R. B. de Araujo. Computação ubíqua: Princípios, tecnologias e desafios. In *XXI Simpósio Brasileiro de Redes de Computadores*, volume 8, pages 11–13, 2003.
 - [13] J. J. I. de Souza, L. B. P. do Nascimento, and P. R. dos Santos Filho. Arduino e python: Do it yourself.
 - [14] A. K. Dennis. *Raspberry Pi Home Automation with Arduino*. Packt Publishing Ltd, 2013.
 - [15] O. S. DOESn't FIIt All. "innovations in ubicomp products.". 2013.
 - [16] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus. Architecture-driven self-adaptation and self-management in robotics systems. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 142–151. IEEE, 2009.
 - [17] Elinux.org. Verified usb peripherals and sdhc cards;. http://elinux.org/RPi_VerifiedPeripherals, May 2012. [Online; accessed 13-March-2015].
 - [18] R. P. FOUNDATION. What is a raspberry pi?., 2014a. <http://www.raspberrypi.org/help/what-is-a-raspberry-pi/>, October 2014. [Online; accessed 04-October-2014].
 - [19] Gadgetoid. Pinout: Uart. <http://pi.gadgetoid.com/pinout/uart>, July 2014. [Online; accessed 15-March-2015].
 - [20] G. Halfacree. Raspberry pi review: Eben upton reveals all. <http://www.linuxuser.co.uk/features/raspberry-pi-interview-eban-upton-reveals-all>, 2014. [Online; accessed 25-February-2015].
 - [21] O. Henrique B. Filho. Componentes eletronicos e unidades de medida, conceitos basicos. <http://www.hardware.com.br/tutoriais/componentes-eletronicos-unidades-medida/>, 2012. [Online; accessed 22-September-2014].
 - [22] D. Industries. A fork of raspbian for robotics projects with lego, grove, and arduino. <http://sourceforge.net/projects/dexterindustriesraspbianflavor/>, June 2013. [Online; accessed 25-March-2015].
 - [23] Jaguar.orpheusweb.co.uk. Brandy basic. <http://jaguar.orpheusweb.co.uk/branpage.html>, July 2005. [Online; accessed 13-March-2015].
 - [24] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.
 - [25] K. Krishnakumar, J. Gubbi, and R. Buyya. A framework for iot sensor data analytics and visualisation in cloud computing environments.
 - [26] J. Krumm. *Ubiquitous computing fundamentals*. CRC Press, 2009.
 - [27] M. C. M. Neto. Desenvolvimento de aplicações ubíquas com arduino e raspberry pi.
 - [28] Oracle. Concurrency in java. <http://docs.oracle.com/javase/tutorial/essential/concurrency/sleep.html>. [Online; accessed 11-January-2015].
 - [29] W. PI. Wiring pi - gpio interface library for the raspberry pi - about. <http://wiringpi.com/>, October 2014. [Online; accessed 02-October-2014].
 - [30] Pibanglinux.org. Inspired by crunchbang linux, and based on raspbian. features the openbox desktop environment. <http://www.pibanglinux.org/>, May 2013. [Online; accessed 25-March-2015].
 - [31] Pigiopio. Pigiopio, a python module for the raspberry which talks to the pigpio daemon to allow control of the general purpose input outputs. <http://abyz.co.uk/rpi/pigpio/python.html>, February 2015. [Online; accessed 25-March-2015].
 - [32] Privateeyepi.com. A home automation and monitoring project for raspberry pi. <http://projects.privateeyepi.com/>, March 2015. [Online; accessed 08-March-2015].
 - [33] Raspberrypi-spy.co.uk. Simple guide to the rpi-gpio header and pins. <http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/>, June 2012. [Online; accessed 25-March-2015].
 - [34] Raspberrypi.org. Model b now ships with 512mb of ram. <http://www.raspberrypi.org/model-b-now-ships-with-512mb-of-ram/>, October 2012. [Online; accessed 13-March-2015].
 - [35] Raspberrypi.org. Raspberry-pi compute module new product. <http://www.raspberrypi.org/raspberry-pi-compute-module-new-product/>, May 2012. [Online; accessed 13-March-2015].
 - [36] Raspberrypi.org. Raspbian wheezy. <http://www.raspberrypi.org/downloads>, January 2013. [Online; accessed 25-March-2015].
 - [37] Raspberrypi.org. Raspberri pi spi. <http://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md>, July 2014. [Online; accessed 15-March-2015].
 - [38] b. RASPBERRY PI FOUNDATION. About us. 2014b. <http://www.raspberrypi.org/about/>, October 2014. [Online; accessed 01-October-2014].

- [39] c. RASPBERRY PI FOUNDATION. Faqs. 2014c. <http://www.raspberrypi.org/help/faqs/>, October 2014. [Online; accessed 01-October-2014].
- [40] Raspberrypi.org. Gpio usage. <https://www.raspberrypi.org/documentation/usage/gpio/README.md>, July 2014. [Online; accessed 15-March-2015].
- [41] Raspbian. Introducing the pi store. <http://www.raspberrypi.org/introducing-the-pi-store/>, September 2014. [Online; accessed 25-March-2015].
- [42] Raspbian. Raspbian – debian optimized for the raspberry pi hardware. <http://www.raspbian.org/>, September 2014. [Online; accessed 25-March-2015].
- [43] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009.
- [44] B. Trapp. Raspberry pi: The perfect home server. linux j. *São Paulo: Sociedade Brasileira de Computação*, May 2013.
- [45] E. Upton. Wayland. <http://www.raspberrypi.org/archives/4053>, May 2013. [Online; accessed 25-March-2015].
- [46] M. Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.
- [47] M. Weiser. The world is not a desktop. *interactions*, 1(1):7–8, 1994.
- [48] M. Weiser and J. S. Brown. The coming age of calm technology. In *Beyond calculation*, pages 75–85. Springer, 1997.
- [49] 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.
- [50] L. Yau. Raspbian server edition version 2.4. the rantings and ravings of a madman. <http://sirlagz.net/2013/06/13/raspbian-server-edition-2-4/>, July 2013. [Online; accessed 25-March-2015].
- Kano OS;
- Nard SDK For industrial embedded systems;
- Sailfish OS with Raspberry Pi 2;
- Tiny Core Linux – a minimal Linux operating system focused on providing a base system using BusyBox and FLTK;
- IPFire – a dedicated firewall/router distribution for the protection of a SOHO LAN;

APPENDIX

A Other operating systems

- Xbian– Using the Kodi (formerly XBMC) open source digital media center;
- openSUSE;
- Raspberry Pi Fedora Remix;
- Slackware ARM – Version 13.37 and later runs on the Raspberry Pi without modification.
- FreeBSD and NetBSD;
- Plan 9 from Bell Labs and Inferno(in beta);
- Moebius – A light ARM HF distribution based on Debian;
- OpenWrt – Primarily used on embedded devices to route network traffic;
- Kali Linux – A Debian-derived distro designed for digital forensics and penetration testing;
- Instant WebKiosk – An operating system for digital signage purposes (web and media views);
- Ark OS – Website and email self-hosting;
- Minepion – Dedicated operating system for mining cryptocurrency;

B Units and scales

Table 3 shows units and scales provided by LibsensorPy.

Table 3: Units and scales

Sensor Family	Default units	Other units provided
Accelerometer	G	m/s ²
Altitude	meter(m)	Centimeter (cm)
Humidity	Percent (%)	
Light	Lux	
Magnetometer	micro-Tesla (uT)	
Motion		
Pressure	hectopascal (hPa)	psi, bar, mmHg, N/m ²
Temperature	Celsius	Fahrenheit, Kelvin
Ultrasonic	Centimeter (cm)	

C Implemented Sensors

Table 4 shows the list of implemented sensors.

Table 4: List of implemented sensors

Family	Sensor	Class names	Example Script
Accelerometer	ADXL345	Adxl345	adxl345Example.py
	LSM303D	LSM303DAccelerometer	lsm303dAccelerometerExample.py
Altitude	BMP085	Bmp085Altitude	bmp085AltitudeExample.py
Humidity	DHT11	DHT11Humidity	dht11HumidityExample.py
	DHT22	DHT22Humidity	dht22HumidityExample.py
Lux	TCS34725	TCS34725	TCS34725Example
Magnetometer	HMC5883L	HMC5883L	hmc5883lExample
	LSM303D	LSM303DMagnetometer	lsm303dMagnetometerExample.py
Motion	PIR	PIR	pirExample
Pressure	BMP085	BMP085Pressure	bmp085PressureExample.py
Temperature	BMP085	Bmp085Temperature	bmp085TemperatureExample.py
	DHT11	DHT11Temperature	dht11TemperatureExample.py
	DHT22	DHT22Temperature	dht22TemperatureExample.py
Ultrasonic	HC-SR04	HCSR04	hcsr04Example.py
	Parallax Ping	ParallaxPing	parallaxPingExample.py
	SRF04	SRF04	srf04Example.py
	SRF05	SRF05	srf05Example.py
	URM37	URM37	urm37Example
	HY-SRF05	HYSRF05	hysrf05Example
Composite	BMP085	BMP085Composite	bmp085CompositeExample.py
	DHT11	DHT11Composite	dht11CompositeExample.py
	DHT22	DHT22Composite	dht22CompositeExample.py
	LSM303D	LSM303DComposite	lsm303dCompositeExample.py

D Raspberry Pi, Models and Specifications

Figure 16 shows all Raspberry Pi's models and specifications.

	Model A	Model B	Model B+
Target price:	US\$25 Ext tax (GBP £16 Exc VAT)	US\$35 Ext tax (GBP £22 Exc VAT)	US\$35 Ext tax (GBP £22 Exc VAT)
System-on-a-chip (SoC):	Broadcom BCM2835 (CPU + GPU. SDRAM is a separate chip stacked on top)		
CPU:	700 MHz ARM11 ARM1176JZF-S core		
GPU:	Broadcom VideoCore IV, OpenGL ES 2.0, OpenVG 1080p30 H.264 high-profile encode/decode		
Memory (SDRAM) iB	256 MiB (planned with 128 MiB, upgraded to 256 MiB on 29 Feb 2012)	256 MiB (until 15 Oct 2012); 512 MiB (since 15 Oct 2012)	512 MiB
USB 2.0 ports:	1 (provided by the BCM2835)	2 (via integrated USB hub)	4 (via intergrated USB hub)
Video outputs:	Composite video Composite RCA, HDMI (not at the same time)		Composite video requires 4 Pole Adapter
Audio outputs:	TRS connector 3.5 mm jack, HDMI		
Audio inputs:	none, but a USB mic or sound-card could be added		
Onboard Storage:	Secure Digital SD / MMC / SDIO card slot		Micro Secure Digital / MicroSD slot
Onboard Network:	None	10/100 wired Ethernet RJ45	
Low-level peripherals:	26 General Purpose Input/Output (GPIO) pins, Serial Peripheral Interface Bus (SPI), I ² C, I ² S , Universal asynchronous receiver/transmitter (UART)		40 General Purpose Input/Output (GPIO) pins, Serial Peripheral Interface Bus (SPI), I ² C, I ² S, I ² C IDC Pins, Universal asynchronous receiver/transmitter (UART)
Real-time clock:	None		
Power ratings:	300 mA, (1.5 W)	700 mA, (3.5 W)	~650 mA, (3.0 W)
Power source:	5 V (DC) via Micro USB type B or GPIO header		
Size:	85.0 x 56.0 mm x 15mm	85.0 x 56.0 mm x 17mm	85.0 x 56.0 mm x 17mm
Weight:	31g	40g	40g

Figure 15: Models and Specifications¹¹

¹¹http://en.wikipedia.org/wiki/Raspberry_Pi