Bachelorarbeit

An IoT based Monitoring System using Raspberry Pis and Seattle Testbed

verfasst von

Peter Klosowski

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2018

Studienkennzahl lt. Studienblatt:	A 033 521
Fachrichtung:	Informatik - Medieninformatik
Betreuer:	Univ.-Prof. Dipl.-Math. Peter Reichl, M.A. St.
Gutachter:	Dr. Albert Rafetseder, BSc MSc

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Peter Klosowski, Wien, 28. Februar 2018

Acknowledgements

First and foremost, the author wants to thank his supervisor, Peter Reichl. Not only is his first name a great one, but his support throughout this semester was extremely well received and welcomed. His feedback and advice improved this work immensely.

Special thanks go to the author's bachelor advisor, Albert Rafetseder, for the hours spent together reviewing, discussing, and writing the numerous emails that were exchanged over the past five months. His patience with the author and qualified input was most appreciated.

The author would also like to thank his friends and family. The help was highly valued, for standing by the authors side, proof reading this work and also distracting the author countless of times.

Furthermore, the author wants to thank STMicroelectronics for providing free samples of their temperature, humidity, and pressure sensors to support this bachelor thesis.

Abstract

Single board computers, such as the Raspberry Pi have been popular for years. More and more, Internet of things (IoT) devices are pushed onto the market to satisfy customer needs. This bachelor thesis proposes an approach to build a decentralized environmental monitoring system that is flexible regarding sensor compatibility. The system developed in this thesis uses Seattle Testbed as a code base and as a result authorized users were able to collect data via the Internet. The restricted Python (Repy) application programming interface (API) was extended with compatible drivers for the sensors. To visualize the data, a website was developed using Angular-nvD3, that is based on the modern plotting tool D3.

Contents

Figures

Tables

Acronyms

API application programming interface. II, IV, 11–14
C celsius. IV, 9, 16, 17
CO_2 carbon dioxide. IV, 1, 9, 10, 19
CoSy Cooperative Systems. IV, 4
CPU central processing unit. IV, 2, 6
eCO_2 equivalent carbon dioxide. IV, 7, 10, 15, 16, 19, 20
EEML extended environments markup language. IV, 3
FLOSS free/libre open source software. IV, 2
GCC GNU Compiler Collection. IV, 12
GPIO general purpose input/output. IV, 1
GUI graphical user interface. IV, 3
hPa hectopascal. IV, 9, 16, 17
I^2C inter-integrated circuit. IV, 1, 5, 9–11, 13
IoT Internet of things. II, IV, 1, 3–7, 10, 15, 16, 20
IR infrared. IV, 1, 7, 9, 15, 16, 18
IT information technology. IV, 1
JSON JavaScript object notation. IV, 14
mm millimeter. IV, 1
MQTT Message Queuing Telemetry Transport. IV, 3, 20
nm nanometer. IV, 1
PC personal computer. IV, 1
PCB printed circuit board. IV, 9
ppb parts per billion. IV, 10, 19
ppm parts per million. IV, 10, 19

Repy restricted Python. II, IV, 11–13, 15
SB sandbox. IV, 2, 5, 12, 13, 20
SCL serial clock. IV, 1, 10
SDA serial data. IV, 1, 10
seash Seattle shell. IV, 8, 13, 15, 16, 20
TVOC total volatile organic compounds. IV, 1, 7, 10, 15, 16, 19, 20
UML unified modeling language. IV, 4
UV ultraviolet. IV, 1, 7, 9, 15, 16, 18
V volt. IV, 10
VM virtual machine. IV, 6, 8
VOC volatile organic compound. IV, 1, 19
WSN wireless sensor network. IV, 3

1 Motivation and Theoretical Background

Since the release of the Raspberry Pi [16], a low-cost single-board computer, a groups of hobbyists, academics, companies, hackers and IT criminals, have grabbed onto this trend of Internet of things (IoT) and are experimenting. They have built useful and unique devices with it, such as smart locks and mirrors. One field widely explored is the monitoring of various environments. Using sensors to expand the application of the Raspberry Pi, it is possible to cover a lot of different use cases, e. g. connecting a camera to observe the surroundings or record climate change over time. Flexibility regarding the operating system, programming language, sensors and expanding the Raspberry Pi with shields are just a few of the advantages, when working with the Raspberry Pi [13].

This kind of embedded systems are all part of IoT. IoT is a term for every object that gets hooked up to the Internet through devices such as the Raspberry Pi. The goal of embedded system is to make human lives easier and improve the live quality. This is achieved by offering new functions to older devices and even control them when nobody is near them. On the other hand, they need to be developed with security in mind to prevent attackers getting access to them. [26]

The sensors used for this project were collecting various kinds of data. One of them was a light sensor. It recorded visible and infrared (IR) light. Visible light has a wavelength of 400 to 770 nanometer (nm) and, as the name states, it is visible to humans. Light with a wavelength from 1 to 400 nm is called ultraviolet (UV). This light hurts the human skin and might cause skin cancer in the worst case scenario. The light closest to the visible red light is called IR light. It can go up to 1 millimeter (mm), starting at a wavelength of 760 nm, and the human perceives it as heat. [14]

Another sensor was responsible to collect data regarding air quality. Total volatile organic compounds (TVOC) are polluting the air. The higher the concentration of these compounds is, the more harmful it is to humans. Without airing a room regularly, the health of humans and animals inside there is at risk. Electronic devices such as printers and personal computers (PCs) are emitting the substances ozone and benzene, that are included in VOCs. Carbon dioxide (CO_2) is part of VOCs as well and therefore, humans are contributing to that by breathing. [5]

The sensors can be connected to a Raspberry Pi through the general purpose input/output (GPIO) pins. Some of them are capable of utilizing the inter-integrated circuit (I^2C) protocol that allows multiple connections of peripherals. Newer versions of the Raspberry Pi, starting with Revision B, are using bus 1 and the GPIO pins 3 (serial data (SDA)) and 5 (serial clock (SCL)) to exchange messages with the sensors. The bus is a system that allows communication between two components and the pins are the physical interface. Each connected slave has its own address that is up to 7 bits long. It is worth mentioning that two slaves cannot have the same address because it would conflict with messaging the right receiver. [9, p. 167ff]

In this work, a Raspberry Pi 2B was combined with three different sensors. Knowing that the finished box would be used by other academics and students for their projects it was necessary to meet some requirements. Ideally, the box should not be too expensive so multiple ones can be produced and set up. To do efficient experiments, a decentralized approach was essential to deploy them for different scenarios and use cases. Therefore the execution of specific software code on multiple boxes to collect data was necessary. Depending on what kind of data should be collected, different code snippets would be used and uploaded to the boxes. They should be

in sync even if the units do not know of each other. Another point was to build a free/libre open source software (FLOSS), so it should be accessible in the future, and can be used to enrich their projects. Open Data was an important aspect as well. For example, if desired, it should be possible to upload all collected data and present it to readers who observe the development of it and the project. This way, it is possible to provide more transparency, contribute to the public and provide a platform to reuse collected data [11].

To solve problems with cluster management, software updates or code pushing a software solution was used, called Seattle. Seattle is described as "a free, open-source platform for networking and distributed systems research" [20]. Seattle runs on many devices allowing researchers and students to access machines that are connected to the network through the Internet. They are limited though, and are just allowed to work inside a sandbox (SB) that limits the use of central processing unit (CPU) power, memory and storage space. [12] [7]

1.1 Application of the Box

As previously mentioned, the expected field of use is more academic, but that does not imply the practical use. It was already clear in the beginning, that a more general approach should be taken. Therefore, a quick replacement of one or more sensors would be necessary at some point.

A possible application would be to use moisture, temperature, light and water quality sensors, pair them with the Raspberry Pis. After that, deploy all of it with the software build to measure the soil in a garden or perhaps even a park or golf course. In an apartment it would be possible to use this kind of technology to improve the health of home plants.

Another good use would be to monitor the well being of occupants in an apartment by installing air quality, temperature, humidity, and pressure sensors. If one of those values would indicate a bad influence to the health or comfort of the home owner, then it would be a good idea to take countermeasures. This could also safe lives, if the occupant gets notified in time to leave the premises.

1.2 Overview of this work

This document is structured as followed. The previous introduction gave a small overview over the motivation and theoretical background of this work. Important terms were defined and described. Section 2 compares this work with other related work and points out the difference to previously published papers. An overview of the system design is given in Section 3. The use case and deployment diagram are described in detail. The implementation of drivers, the process of extending the Seattle SB, the website for visualization, and wiring the sensor components to the Raspberry Pi are part of Section 4. In Section 5, the results of a functional and experiment evaluation are described. The last part of this work, Section 6, is the conclusion. It elaborates on positive outcomes, issues, that appeared during development, and future work.

2 Related Work

Several IoT solutions were thought of and developed in the past. Many of these following papers describe use cases of decentralized IoT devices utilizing different technologies.

Reinfurt L. et al [17] gathered five different state-of-the-art design patterns regarding IoT that have been identified by reviewing existing products. They recognized the problems that occurred during the software implementations of IoT devices and also offered a satisfying solution.

Shete R. and Agrawal S. [21] presented a low cost urban climate monitoring system using a Raspberry Pi and various sensors such as air quality, light and temperature. The collected data was uploaded to Adafruit IO, a cloud IoT system, via Wi-Fi. Through the use of an Message Queuing Telemetry Transport (MQTT) broker, the data was distributed to subscribed user.

Roselle B. Anire et al [4] focused on monitoring soil in a greenhouse. A wireless sensor network (WSN) combined with the ZigBee protocol was used as a solution. Four Raspberry Pis were deployed and connected together over the network and data was sent to a PC that aggregated it. A simple graphical user interface (GUI) was created to visualize this data and inform the farmer about the soil conditions.

Ibrahim M. et al [10] collected environmental data using a Raspberry Pi and sensors connected to it, including a sensor registering earthquakes was installed. A protocol, extended environments markup language (EEML), was used to share the sensor data with another remote device such as a laptop, smartphone or a web service in the cloud.

Wixted A. et al [25] presented LoRa and the LoRaWAN protocol and compared it to other state-of-the-art technologies that are used for IoT. The protocol is explained and evaluated on reliability and performance with a positive outcome, such as network coverage in hard-to-reach places with numerous gateways.

Related to this work, [21], [4], and [10] are also building IoT systems. They are deployed in different environments. Taking [21] as a model, the low cost approach of the climate monitoring system affected the sensor components that were bought for this project. Sensors for air quality, light and temperature were used. For future work it would be possible to update the visualization website using MQTT and implement LoRaWAN, that was used in [25]. To build a secure and reliable IoT environment, it is advised by [17] to make use of the IoT patterns discussed in that paper. Seattle uses some of this patterns that improve the IoT device.

3 System Design and Solution Concept

Based on the requirements discussed in the introduction of this work, a software design was developed. The whole documentation, partially described here, can also be retrieved from the personal GitHub repository[1]. Regarding the following content, it is worth mentioning that the existing Seattle Testbed [19] was used as a codebase. This decision affected some of the software design as a result, such as the deployment and the use cases.

All of the following system designs were modeled with the unified modeling language (UML) standard [15]. That standard is widely used in software engineering and allows a quick understanding of high and low level architectures with the provided tools. It is also capable of creating business and similar models, but these were not necessary this time, because this work was a pure software solution.

3.1 System Overview – Use Case

At the beginning of the project, the future features were defined using a Use Case diagram. Figure 1 illustrates six use cases that were actually implemented. There are two actors that will be using them. In this case the "User" will be any person who has access to the system and is allowed to operate with it. The "IoT box" is the combination of a Raspberry Pi, sensors and the Seattle software build. The use cases described in the following Sections 3.1.1 to 3.1.3 are handled by Seattle. Use cases discussed in Sections 3.1.4 to 3.1.6 are contributions of this thesis.

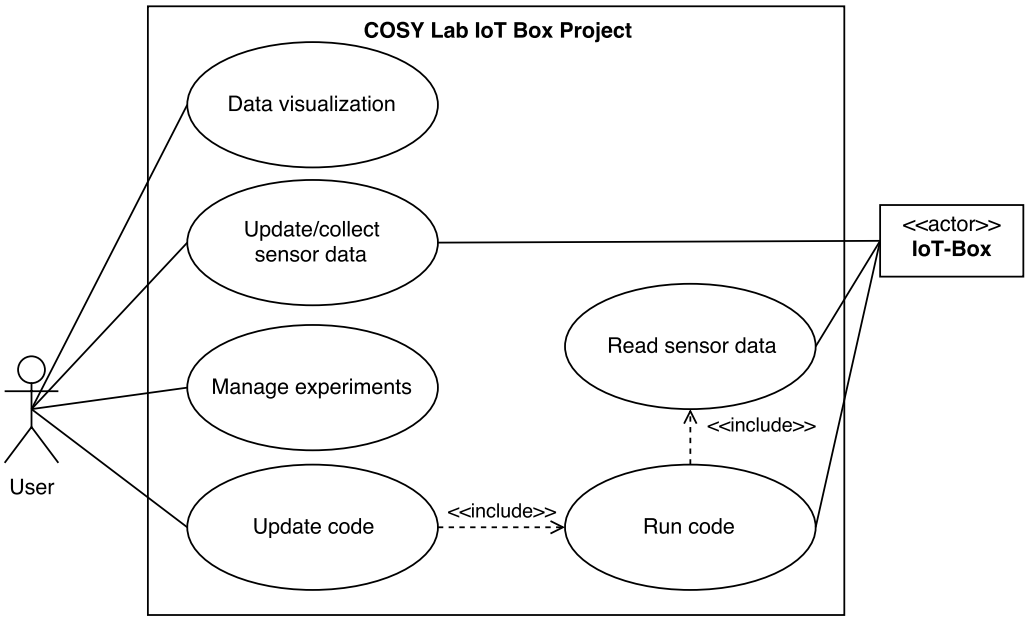


Figure 1: Use case diagram describing the CoSy Lab IoT box Project

[1] https://github.com/Peter0014/COSY-Lab-IoT-Box/tree/master/DesignDoc

3.1.1 Manage Experiments

The use case includes two work flows regarding experiments that are run on one or more IoT boxes. Experiments in this scope are installations of the software on the Raspberry Pis. It is possible to register new experiments and remove ones that are no longer needed. Also, secure access should be guaranteed in case a user who is not authorized tries to access this information.

3.1.2 Update Code

To adapt to different use cases of the boxes, it is necessary to update its purpose and code over the Internet. The user is able to upload a code snippet with the functions that he or she wants to run and decides to push it to one or more boxes. Here the user has the choice which sensor data is collected, the interval, and the file format it is saved in. After a successful upload, the boxes will try to run it and begin its data collection or return an error to the user. Access control should also be implemented in this case to allow only authorized users to update code fragments.

3.1.3 Run Code

All uploaded codes needs to be run by the boxes. It is important to do so the user is able to collect data from the sensors attached to the box. Depending on the code that is pushed, different types of sensor data can be fetched and saved.

3.1.4 Read Sensor Data

First, the data needs to be collected. Over the I^2C bus a connection will be established and sensor data will be fetched with help of the drivers imported into the Seattle SB beforehand. The uploaded code decides which sensors are contacted and the time interval in which new data is appended to a file. Depending on the use case, it should be possible to use different sensors that were originally not thought of at the beginning of this work or are good additions to the existing ones. Because every sensor is different, like newer revisions or different kinds of design and companies, it will be necessary to update the software loaded onto the Raspberry Pis. This process is explained in more detail in Section 4.

3.1.5 Collect Sensor Data

It is possible to download the files over the Internet produced by the user supplied code and after some sensor data was read and saved to the boxes. The data can be used to plot some graphs using the dedicated website that was developed during this work and is described further in Section 4 and the next use case. Regarding the Open Data requirement, it is possible to publish that data for everybody to download, use, share, and work with.

3.1.6 Data Visualization

The data visualization is another crucial part of this work. It would also be possible to visualize the data with applications such as Microsoft Excel[2] and similar tools. The

[2] `https://products.office.com/de-at/excel`

end user would download raw data sets to import them into those tools to plot charts. On the other hand it is useful to offer the user a tool, or web page in this case, where he or she can upload data and it is automatically visualized. Additionally, it would be possible to store data on the web server to demonstrate it using the plotting tool that is specifically customized for the format and output of the boxes.

3.1.7 Alternatives

Searching for solutions, it was decided that Seattle was a satisfying approach in this case. However, there were other thoughts and alternatives before Seattle came up. For example, it was an option to built a decentralized system behind it from scratch. IoT patterns mentioned in [17] would have been used. Device shadowing for a persistent storage of the data allow to reach a device, even if it is offline and unavailable right now. Implementing a rules engine would have supported with dynamic routines for fetching sensor data if the user sent a particular message to the system. If the boxes were used in outdoor places that are not overlooked or unsecured, it would also have been helpful to have a remote lock feature.

Some of these patterns are available in the Seattle software, such as a rules engine that supports actions via a special scripting language. Additionally, it also covers and solves problems such as cluster management and code pushing. It was also decided with the supervisor that building an IoT system from scratch would be too large a scope for a bachelor project.

3.2 System Overview – Software Deployment

The software deployment illustrates the distribution of software artifacts. Looking at Figure 2 one can differentiate between four different devices:

"User Client" is typically the home computer of the end user

"Web Server" is used for the visualization of the data

"Raspberry Pi" is part of the subsystem "IoT box" and executes the user provided code

"Various Sensors" are part of the subsystem "IoT box" and collect data from the environment

The components are described in more detail in the following sections.

3.2.1 Raspberrry Pi

The hardware runs the Seattle software. The Seattle virtual machine (VM) executes the user provided code and prevents malicious content that could potentially break out of the VM to get access to the system it is running on. It also prevents the application from using too many resources from components such as the CPU or memory. [27, p. 38] [12]

The Seattle Node Manager is important for several reasons. First, it manages access control and allows just authorized parties to execute code on the device it is installed on. At the same time, it provides an interface to upload code, manage a VM, and download collected data and logs from it. [27, p. 38]

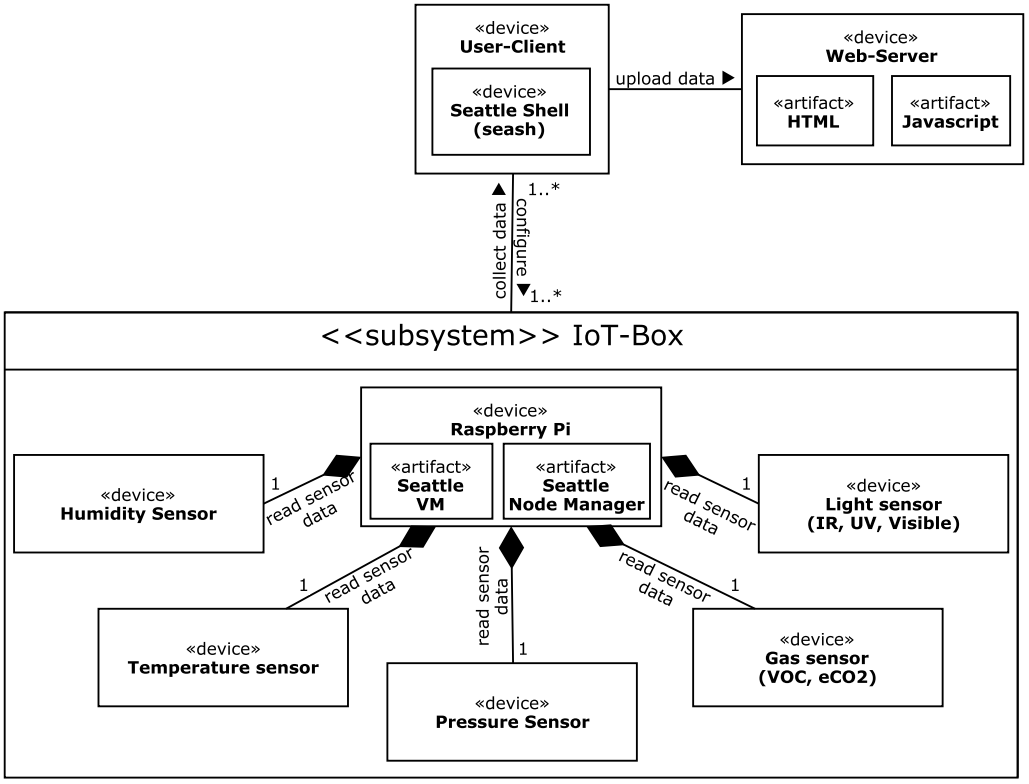


Figure 2: Deployment diagram

Different sensors connected to a Raspberry Pi are components of the subsystem "IoT box". In this case, the sensors are: a humidity, temperature and pressure sensor, a gas sensor collecting TVOC and equivalent carbon dioxide (eCO_2) data, as well as a light sensor for IR, UV and visible light. All of them are described in more detail in Section 4.1. Using Seattle allows to have one or many IoT boxes that are configured by one or many authorized user clients.

3.2.2 Web Server

The visualization of the data takes place on the web server, or to be precise, on the user client by processing the JavaScript files that are provided by the server. It is possible to load a file with the sensor data into the script and let it plot different graphs for every available data entry. The output is presented to the user through the web page in his or her browser. Regarding Open Data, it is also possible to save files on the web server by the administrator to share it with the public.

3.2.3 User-Client

Through Seattle shell (seash), it is possible to connect to Raspberry Pis if the user is authorized to do so. The user is able to install seash onto his or her device and configure the VM, push code to it, as well as to collect the data that is generated through his or her code. Seash operates here as a service manager and allows the interaction with the node manager [27, p. 38]. The user client also connects to the web server via a browser of his or her choice and opens up the web page to process his data, that he downloaded from the Raspberry Pis.

4 Implementation

There are three parts to the implementation. The first part was the decision of buying several sensors, built a hardware interface, connect them to the Raspberry Pi and test them. The second part consisted of developing the software to operate the sensors and extend Seattle in that way, that it can call those software functions. In the last part, a website was built that can be used for visualization and informing other people about the project and progress.

4.1 Exploring Sensors

Looking for different use cases that were not just practical but also useful in the sense of doing research with them, the decision was made to buy three different sensors. The sensors measure 7 different physical quantities. In combination, they were collecting useful data about the environment that could be used in all rooms of a home, in a lab, and outdoors. Furthermore, all sensors are breakout boards by the company Adafruit[3]. It was necessary to solder the header onto the printed circuit board (PCB) and was also required that all sensors support the I^2C protocol.

4.1.1 Temperature, Barometric Pressure and Humidity Sensor

One sensor is a BME280 environmental sensor was manufactured by Bosch[4]. It measures temperature, humidity and barometic pressure, also known as atmospheric pressure. It has an accuracy of ± 3% regarding humidity, ± 1 hectopascal (hPa) with barometic pressure and ± 1.0celsius (C) with temperature. [1]

Additionally, STMicroelectronics[5] provided sample sensors from their product line. For one, the HTS221 for relative humidity and temperature and the LPS25HB for absolute pressure was sent to support the project. They have an accuracy of ± 0.5C regarding temperature between 15C and 40C, ± 3.5% with humidity and ± 1 hPa with barometic pressure. [23] [24]

4.1.2 Light Sensor

The SI1145 light sensor from SiLabs[6] calculates the UV index based on the IR and visible light from the sun. As a side note, this sensor measures light levels and the IR and visible light values are "based on how much light the sensor sees, and there's no 'units' to them". [3]

4.1.3 Air Quality Sensor

There is a lot of air pollution in big cities that is harmful to every person if it is present in high amounts. This is especially prominent in the morning when people drive to work. The CO_2 pollution is increasing immensely [6]. To measure that, it was decided to install an air quality sensor - the CCS811. The chip was developed by ams[7]

[3] https://www.adafruit.com/
[4] https://www.bosch-sensortec.com/
[5] http://www.st.com/
[6] https://www.silabs.com/
[7] http://ams.com/

and collects equivalent carbon dioxide (eCO_2) and total volatile organic compounds (TVOC) data. It has a range from 400 to 8192 parts per million (ppm) regarding eCO_2 and 0 to 1187 parts per billion (ppb) with TVOC. The value eCO_2 is also a bit misleading because it is based on the measured TVOC. An increasing in TVOC indoors also quite often means an increase in CO_2, because a person entered the room. This sensor does not measure the amount of CO_2 in the air.

It is also recommended to run it 48 hours after buying it to burn it in. Before starting to read data, it is always necessary to let the sensor run for 20 minutes. Unfortunately, after starting the data collection for the evaluation it was clear that the air quality sensor does not work properly with the Raspberry Pi because the sensor uses clock stretching that the Pi does not support. The sensor will be replaced in the future. [2]

4.2 Wiring

The Fritzing[8] application was used to visualize the wiring of the sensors with the Raspberry Pi. Figure 3 illustrates all three previously mentioned sensors on a breadboard to the left of a Raspberry Pi 2B. A breadboard is perfect for testing purposes because it does not require any soldering of components.

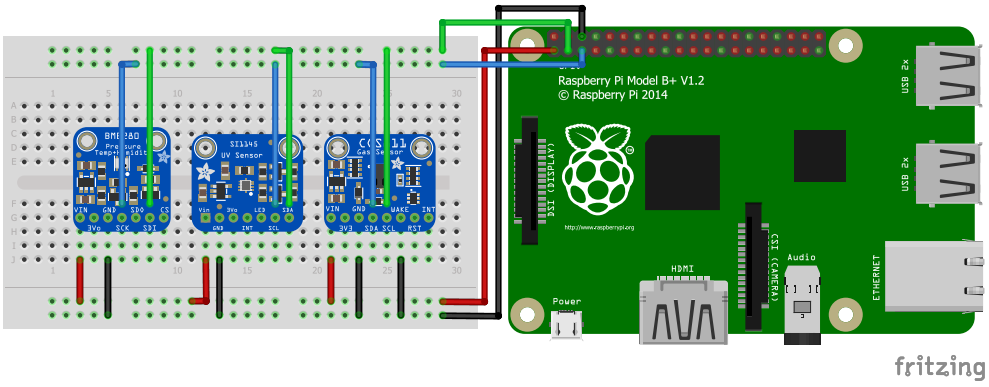


Figure 3: Wiring diagram of an IoT box

Two wires are necessary to connect the sensors to the physical interface of the I^2C bus. The blue wire represents the connection to pin 5, the SDA pin. Pin 3, or SCL, connects to the sensors through the green wire. It is also necessary to power the sensors. Therefore, the red wire is attached to the 3.3 volt (V) pin and the black one to the ground.

The wires were connected, according to the I^2C wiring on the websites from Adafruit. From left to right, the sensors in Figure 3 are the BME280, the SI1445, and the CCS811. Every sensor receives power at the pinout with the label `Vin`. The ground is at `GND`. The SDA and SCL pinouts are labeled `SDA` and `SCL`. The CCS811 needs another ground wire that connects to the `WAKE` pinout. [1] [2] [3]

[8] `http://fritzing.org`

4.3 Software Development

The next step taken after connecting the sensors to the Raspberry Pi over the I^2C dedicated Pins was to develop software classes and extend the Seattle Repy V2 code to call the functions in these classes. Figure 4 shows a small excerpt of these classes that are described in the following sections.

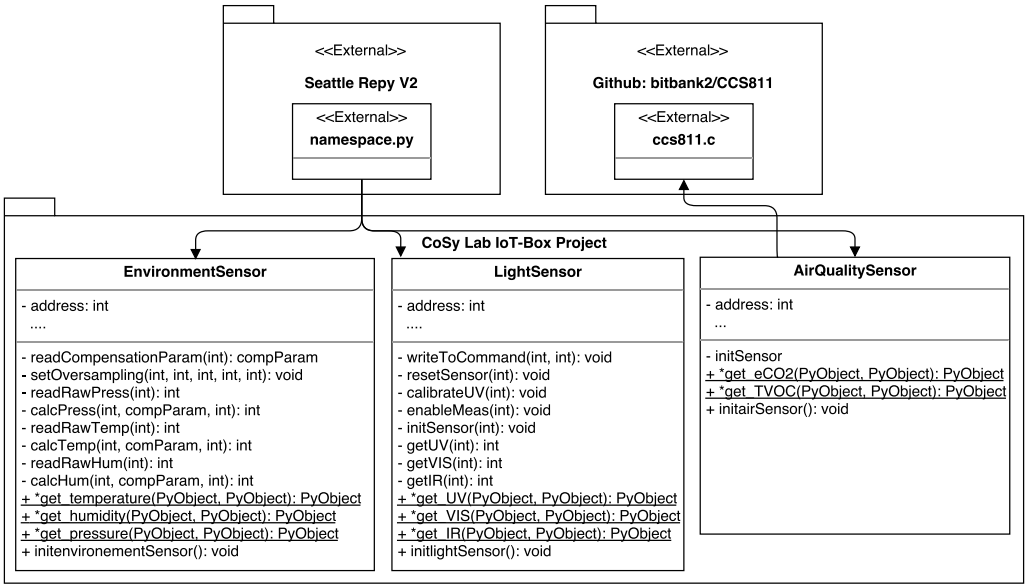


Figure 4: Excerpt of the class diagram

4.3.1 Writing the Classes

Seattle is programmed using the Python language. Therefore, Python would have been a good choice for this situation. Consequently, it was inevitable to use a wrapper if C code would be used to extend the existing Seattle code base.

The decision was made based on a personal preference. Being more in contact C through internships and courses at the university, it was easier to perform this project with the already available know-how. Looking for available resources online, there was an official section in the documentation of Python about extending it with C[9], as well.

With the CPython API, it is possible to use C code in a Python class. It is imported like any other module in Python. However, the C class needs to be prepared before Python is able to call it. A method is necessary for Python to initialize the methods used in the C code. Python also cannot process the data types used in C. Therefore, the data type `PyObject` needs to be created and returned.

The classes `EnvironmentSensor`, `LightSensor` and `AirQualitySensor` are drivers, which read various values from the sensors. They are imported by Seattle Repy V2 in the `namespace.py` class. To compare development time, the class fetching the air

[9] `https://docs.python.org/2/extending/extending.html`

quality values is just a wrapper and uses the implementation from Github by the user bitbank2[10]. Writing just a wrapper for an existing driver took two hours. Depending on the complexity of a sensor, it could take effort and time to implement a driver. The drivers for the BME280 and SI1145 took each three to four days to implement.

4.3.2 Extending Seattle with C

To extend the Seattle SB with sensor functions, drivers were written with the help of the documentations from the sensors. After testing them on the device itself and checking that the right values were returned, a shared library and Python module was created using the GNU Compiler Collection (GCC). For example, a shell command could be written as in Figure 5.

```
gcc -shared -Xlinker -export-dynamic -o environmentSensor.so
-I/usr/include/python2.7/ -lpython2.7 -lwiringPi
BME280_TempSensor.c
```

Figure 5: Build command for a shared library

In this case a shared library is created with the parameters `-shared -Xlinker -export-dynamic`. The output file would be `environmentSensor.so` and the input file is the C class `BME280_TempSensor.c`. The WiringPi and Python libraries and imports are necessary to build the class properly. The `.so` file, that is produced calling that command, can be loaded into Python as a module now. Visible are just special functions though. Looking at Figure 4, the static functions that return the `PyObject` are added in the init function (like `initenvironmentSensor()`) so that Python is able to work with them.

```
...
import environmentSensor
...
RPI_ENVIRONMENT_SENSORDATA_WRAPPER_INFO = {
    'get_temperature' :
        {'func' : environmentSensor.get_temperature,
         'args' : [],
         'return' : Float()},
         ...
}
USERCONTEXT_WRAPPER_INFO.update
        (RPI_ENVIRONMENT_SENSORDATA_WRAPPER_INFO)
```

Figure 6: Import of the shared library environmentSensor.so in namespace.py

Having a working Python module available, it was now possible to extend Seattle with that code. The modules were added to the Repy SB by expanding the API with the specific function calls, such as `get_temperature`. This was done in the

[10] `https://github.com/bitbank2/CCS811`

`namespace.py` file (see Figure 4 and 6). The extended Repy V2 source code is available online at [18].

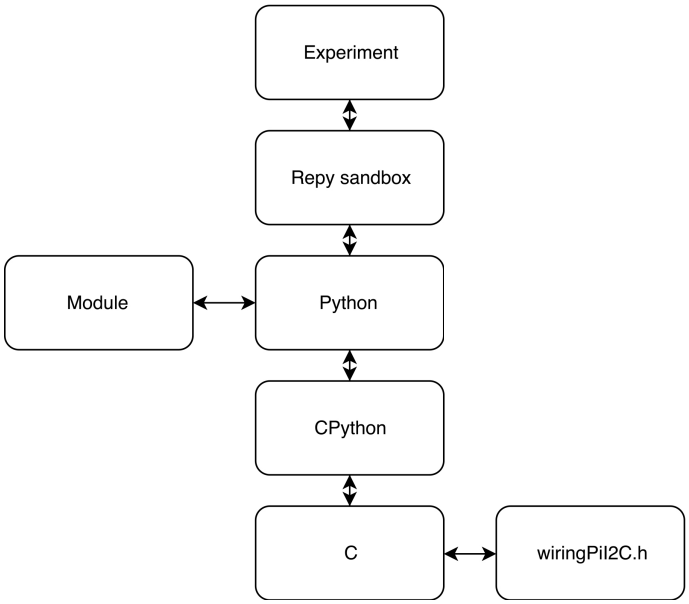


Figure 7: Path from experiment to C code

This work allows the user, in the end, to call the functions through seash over the Internet. The message flow when running the Repy code provided by the user is illustrated in Figure 7. Here, the experiment calls the API in the Repy SB that runs the Python code. At this point, it was possible to call a Python module that accesses the sensors. Through the CPython API the C code is executed and accesses the I^2C messages through the wiringPi library. The values are then returned to the experiment where it is being processed.

4.4 Visualization and Website

A website to visualize the sensor data also is part of this work. The goal of the website was to provide the user with means to visualize his or her data and to have a platform to inform about other information regarding the boxes, as well as developments.

The website, that is visible in Figure 8, uses various libraries and frameworks to offer those features. The front-end was built using Bootstrap[11], JQuery[12], and Font Awesome[13] (for various kinds of icons). Since Bootstrap is able to optimize a web page for both mobile and desktops, one may view this website on most modern devices at multiple screen resolutions. As a side note the website is just there to showcase the visualization and provide a template for others therefore there is just one web page available as of now.

[11] `https://getbootstrap.com/`
[12] `https://jquery.com/`
[13] `https://fontawesome.com/`

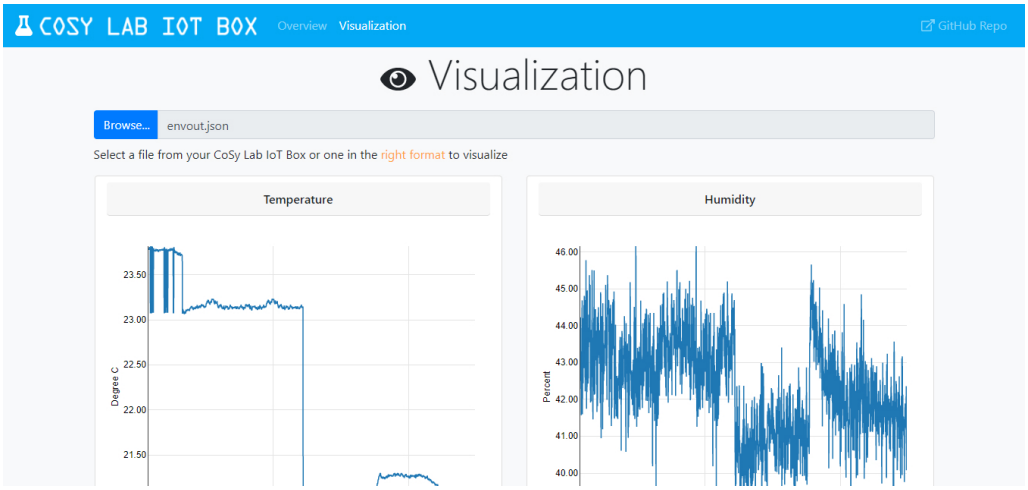


Figure 8: Screenshot of the visualization website

To plot the charts, the JavaScript library Angular-nvD3[14] was used. As the name states Angular[15], NVD3[16] and D3.js[17] are also required to be loaded. They are the back-end of this website and are responsible to build the charts using the data uploaded by the user.

To create a visualization, the user is asked to upload a file in JavaScript object notation (JSON) format[18]. The function that processes the file searches for a specific key and adds the value and date to the corresponding chart. The available keys used in this project are:

- Time
- Temperature
- Humidity
- Pressure
- UV
- IR
- VIS
- eCO2
- TVOC

The collection of key-value pairs is iterated and pushed into the values array of the charts. The charts are refreshed through the API from Angular-nvD3 after the last task is finished. The process, after uploading the file, normally takes just a few seconds and depends on the amount of values being inserted.

[14] http://krispo.github.io/angular-nvd3/
[15] https://angular.io/
[16] http://nvd3.org/
[17] https://d3js.org/
[18] https://www.json.org/

5 Evaluation and Testing

Many use cases are possible with the IoT box and there are a plethora of possibilities to extend it with the available sensors on the market. Therefore, it is not easy to test the box for every eventuality, but some evaluation and testing can be done for the most probable utilization of it.

5.1 Functional Evaluation

Working with the box since the first prototype was built, there were some work flows that reoccurred quite often. Based on that a use case scenario was created:

Use Case Scenario - Collecting Data and Visualizing it

- A researcher successfully connects over seash with his cluster of IoT boxes with his personal key
- He prepared a Repy code that he now uploads to the boxes and starts them
- After enough data is collected, the researcher stops the execution of his code and downloads the files that were created during that time period
- Fortunately, he saved all those collected values in one JSON file and has now temperature, humidity, pressure, UV, IR, visible light, eCO_2 and TVOC data that he uploads to the web page
- The website processes his file and plots graphs for the data that the researcher can analyze

Some tests were conducted on the basis of this use case scenario that are shown in Table 1. Not every function was tested explicitly: some functions are already implemented in Seattle and were not worked on as part of this work, for example up- and downloading files to the boxes.

A majority of the tests worked well. The partial result with the long-term experiment is connected with the other two partial results. As mentioned in Section 4.1.3, the air quality sensor does not fully work with the Raspberry Pi. It was just possible to collect data for a shorter period of time than with the others. The other sensors worked properly as we see in Table 1. Therefore, a long-term experiment is possible if the sensor is not used.

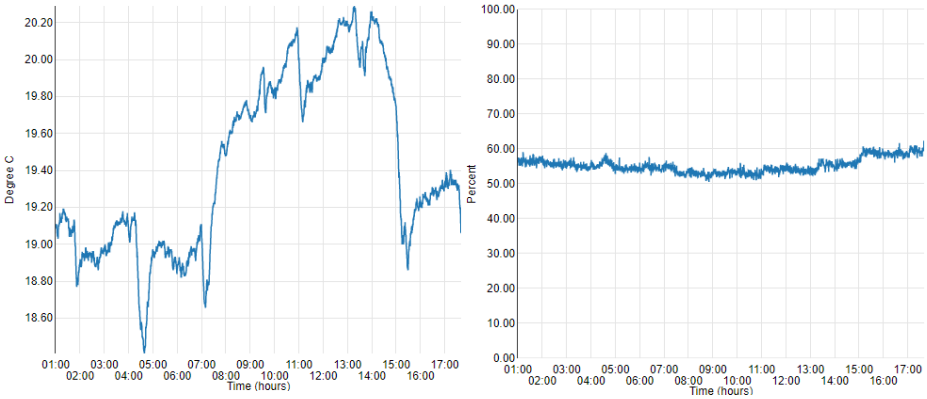
Test	Result
Doing a long-term experiment	Partial
Reading temperature	Worked
Reading humidity values	Worked
Reading pressure values	Worked
Reading UV values	Worked
Reading IR values	Worked
Reading visible light values	Worked
Reading eCO_2 values	Partial
Reading TVOC values	Partial
Visualizing collected data	Worked

Table 1: Tested features of the IoT box implementation

5.2 Experiment Evaluation

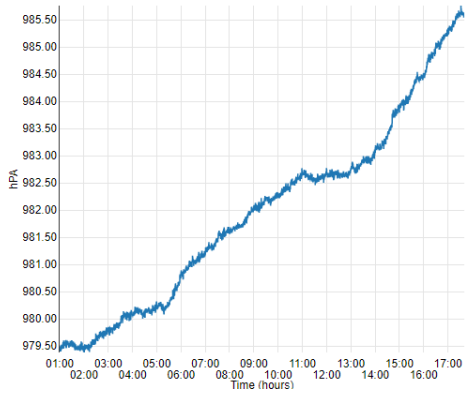
For the experiment the IoT box was put on a windowsill during winter in Vienna, on 09.02.2018. Right beneath the window is a heater and the room dimension is $15m^2$ (around $161ft^2$). According to WetterOnline[19], the outdoor temperature at "Hohe Warte" was between 0C and 3C. humidity was measured at 74% and pressure at 1019 hPa. The IoT box collected data for several hours before the files were downloaded through seash. Having the file at hand, it was loaded into the visualization web page to create charts.

[19] `https://www.wetteronline.at/wetterdaten/wien?period=4&month=02&year=2018`



(a) Temperature data (in C)

(b) Humidity data (in %)



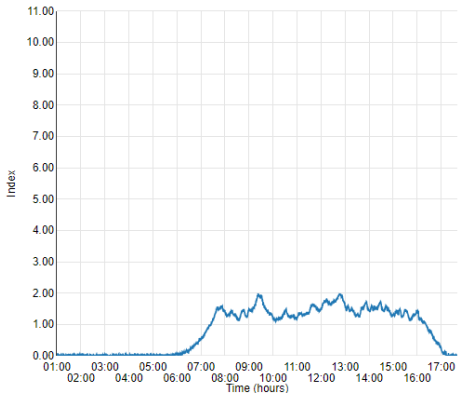
(c) Pressure data (in hPa)

Figure 9: Data collected by the BME280 sensor

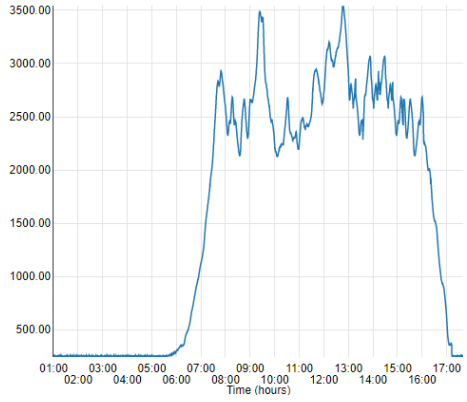
Figure 9a shows the measured temperature. At 7 AM the heating started and the temperature had risen up to its peak at 1:15 PM to 20.30C. At 3 PM, the window was tilted for 30 minutes and the temperature fell 2C before rising again. The lowest point was reached at 4:40 am with a temperature of 18.42C. The mean is 19.43 from all 2000 collected values.

The humidity in Figure 9b did not show significant change. It started at midnight at 56.89% humidity and in the end just raised by less than 6% to 62.34%, its peak. The lowest point was at 7:50 AM with a value of 50.55% and the mean of 55.24%. Tilting the window also increased humidity by a small amount. This can be explained by the fact that the window was just tilted and no real exchange of inside and outdoor air happened here.

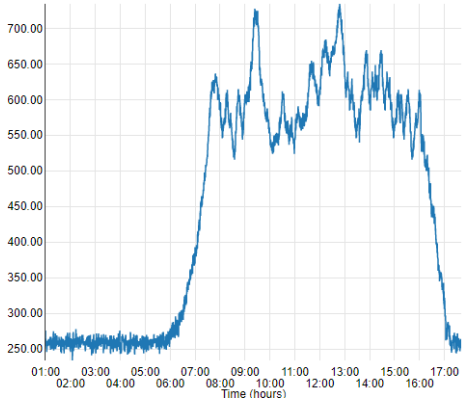
Comparing the pressure with the information from WetterOnline, it is apparent that it was increasing that day. This also reflects with the values recorded by the box. Starting at 979.39 hPa, the numbers grew up to 985.77 hPa.



(a) UV data (index)



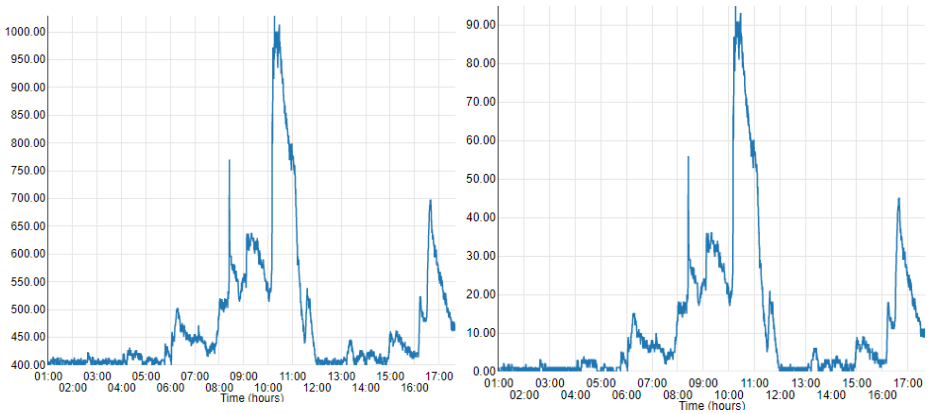
(b) IR light data



(c) Visible light data

Figure 10: Data collected by the SI1145 sensor

On this day, the sunshine was limited. Inspecting Figure 10b and 10c, we can see that the sunrise was at 7 AM and the sunset happened at 5 PM. The graphs indicate, there are no units to the values because the sensor just measures how much light falls into it. Therefore, it is there for information, but using those two values, it is possible to calculate the UV index that is shown in Figure 10a. The World Health Organization categorized the UV index into several categories. It is categorized into low (0-2), moderate (3-5), high (6 and 7), very high (8-10) and extreme (11+) [8]. Hence, the scale of the y-axis in Figure 10a. That day it stayed under two. The value is low enough as to not be harmful to human skin.



(a) eCO_2 data

(b) TVOC data

Figure 11: Data collected by the CCS811 sensor

After the 20 minutes burn in that are necessary for the sensor to calibrate correctly, it started to measure some data. The eCO_2 and TVOC was at its peak at 10:30 AM with values over 1000 ppm and 90 ppb respectively. It is shown in Figure 11a and 11b that the two graphs look almost identical. This is because the eCO_2 value is based on the TVOC detection. Humans also produce CO_2 and VOCs by exhaling it. In this case, analyzing the graphs, every spike means that the room was occupied by one or more persons.

6 Conclusion

The solution approach presented in this work is combining Seattle with the Raspberry Pi to build IoT devices and fetch the sensor data that was collected. After working with the setup for several weeks, it emerged, that the advantages of Seattle made it the suitable candidate as a software base. This advantages were cluster management, code pushing, the SB and extendability of the API. The flexibility of the approach made the solution future proof because new sensors can be attached and used with existing ones.

During the implementation, no larger problems were presented. The only negative development was the TVOC/eCO_2 sensor that did not properly with the Raspberry Pi. As already described in the functional evaluation, it was still possible to operate with it, but not in the way original intended. That was a small drawback regarding the full functionality of the IoT box, but it would be fixable by replacing the sensor with another one.

Additionally, it is possible to configure the IoT boxes over the Internet. The boxes need a working connection so the user can upload and start his or her file through seash. On the other hand, it also simplifies the process of configuring multiple boxes at once.

6.1 Future Work

In the future it would be possible to expand the sensor diversity. There are a lot of different sensors for sale like motion, sound, GPS and plenty more. Even sensors that have not been developed yet and are just future thinking could possibly be paired with a Raspberry Pi which has Seattle installed.

To conduct some outdoor research, it would be helpful to have support for LoRa and LoRaWAN. LoRaWAN is based on LoRa and is able to store data in a database over a long distance by uploading them through a gateway [22]. An application server like the visualization website for example could fetch the data and have an up-to-date collection of charts. Using MQTT, the data could be available the moment it gets uploaded to the database, if the website subscribes to the broker.

References

[1] Adafruit - BME280. https://learn.adafruit.com/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout?view=all, accessed February 10, 2018.

[2] Adafruit - CCS811. https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor?view=all, accessed February 10, 2018.

[3] Adafruit - SI1145. https://learn.adafruit.com/adafruit-si1145-breakout-board-uv-ir-visible-sensor?view=all, accessed February 10, 2018.

[4] Anire, R. B., Cruz, F. R. G., and Agulto, I. C. Environmental wireless sensor network using raspberry pi 3 for greenhouse monitoring system. In *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)* (Dec 2017).

[5] Brown, N. Überwachung der Raumluftqualität (Monitoring indoor air quality). *elektronik industrie*, 4 (2017), 62–65.

[6] Caldeira, K., and Davis, S. J. Accounting for carbon dioxide emissions: A matter of time. *Proceedings of the National Academy of Sciences 108*, 21 (2011), 8533–8534.

[7] Cappos, J., Dadgar, A., Rasley, J., Samuel, J., Beschastnikh, I., Barsan, C., Krishnamurthy, A., and Anderson, T. Retaining sandbox containment despite bugs in privileged memory-safe code. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), ACM, pp. 212–223.

[8] Fioletov, V. E., Kerr, J. B., and Fergusson, A. The uv index: definition, distribution and factors affecting it. *Can J Public Health 101*, 4 (2010), 5–9.

[9] Gay, W. *Mastering the Raspberry Pi*. Apress, 2014.

[10] Ibrahim, M., Elgamri, A., Babiker, S., and Mohamed, A. Internet of things based smart environmental monitoring using the raspberry-pi computer. In *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)* (Oct 2015).

[11] Janssen, M., Charalabidis, Y., and Zuiderwijk, A. Benefits, adoption barriers and myths of open data and open government. *Information systems management 29*, 4 (2012), 258–268.

[12] Li, T., Rafetseder, A., Fonseca, R., and Cappos, J. Fence: Protecting device availability with uniform resource control. In *USENIX Annual Technical Conference* (2015), pp. 177–191.

[13] Maksimović, M., Vujović, V., Davidović, N., Milošević, V., and Perišić, B. Raspberry pi as internet of things hardware: performances and constraints. *design issues 3* (2014), 8.

[14] Mori, K., Nishida, T., and Hashimoto, H. Ultraviolet light measuring chip and ultraviolet light sensor using the same, Apr. 22 2003. US Patent 6,551,493.

[15] Object Management Group. Unified modeling language. https://www.omg.org/spec/UML/2.5.1/PDF, 12 2017.

[16] Raspberry Pi Foundation. https://www.raspberrypi.org, 2012 (accessed February 5, 2018).

[17] Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., and Riegg, A. Internet of things patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs* (2016), ACM, p. 5.

[18] Repy V2 - Github Fork. https://github.com/Peter0014/repy_v2, 2017 (accessed February 9, 2018).

[19] Seattle. https://seattle.poly.edu/html/, 2013 (accessed February 5, 2018).

[20] Seattle - Github. https://github.com/SeattleTestbed, 2014 (accessed February 5, 2018).

[21] Shete, R., and Agrawal, S. Iot based urban climate monitoring using raspberry pi. In *2016 International Conference on Communication and Signal Processing (ICCSP)* (April 2016).

[22] Sornin, N., Luis, M., Eirich, T., Kramp, T., and Hersent, O. Lorawan specification. LoRa Alliance, 2015.

[23] STMicroelectronics - HTS221. http://www.st.com/content/st_com/en/products/mems-and-sensors/humidity-sensors/hts221.html, accessed February 13, 2018.

[24] STMicroelectronics - LPS25HB. http://www.st.com/content/st_com/en/products/mems-and-sensors/pressure-sensors/lps25hb.html, accessed February 13, 2018.

[25] Wixted, A. J., Kinnaird, P., Larijani, H., Tait, A., Ahmadinia, A., and Strachan, N. Evaluation of lora and lorawan for wireless sensor networks. In *2016 IEEE SENSORS* (Oct 2016).

[26] Xia, F., Yang, L. T., Wang, L., and Vinel, A. Internet of things. *International Journal of Communication Systems 25*, 9 (2012), 1101.

[27] Zhuang, Y., Rafetseder, A., and Cappos, J. Experience with seattle: A community platform for research and education. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI* (2013), IEEE, pp. 37–44.