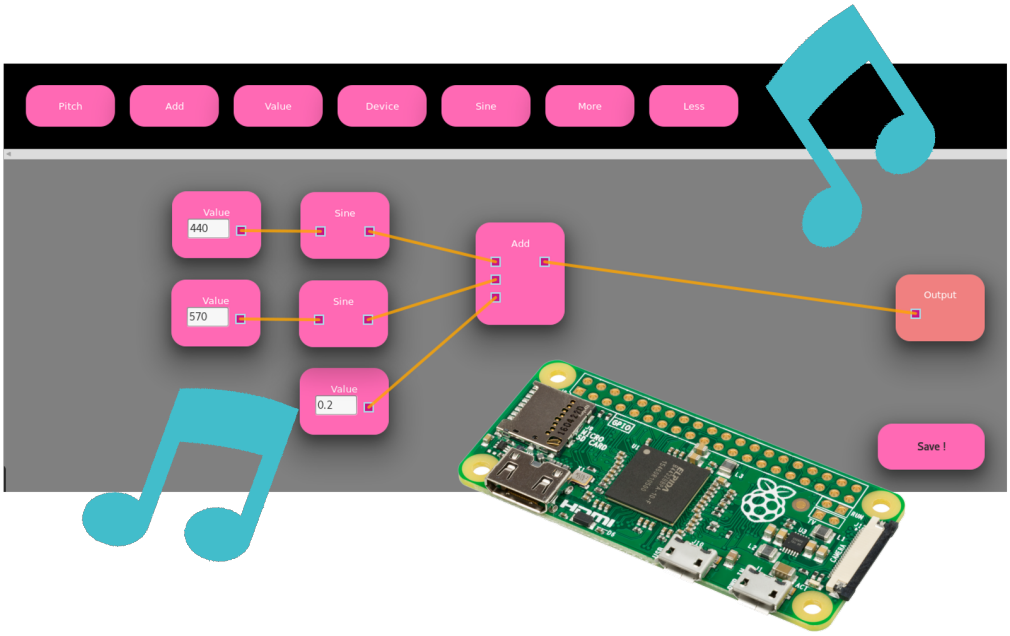
Lycée René Descartes

”The Music Swagger” : De la musique interactive



Estelle FIKET
Thomas POTIER
Nour BOULAHCEN

encadré par
Mr. Bernigole

1er juin 2017

Table des matières

I. Présentation

Passionné par l'informatique, je souhaitais m'attaquer à un projet complexe réalisable en équipe. Avec Estelle et Nour, nous avions d'abord pensé à développer un jeu de type "Rogue" ou le joueur explore plusieurs niveaux. Puis, suite à nos discussions, nous avons décidé de créer un système fonctionnant comme un instrument de musique numérique : nous avons imaginé un système qui permettrait de générer du son en fonction de paramètres physiques tels que l'accélération, la distance ou tout ce qui peut être numérisé.

En effet, le traitement du son est un domaine qui nous plaisait et ce projet fait appel aux différentes notions étudiées cette année en cours d'ISN : représentation de l'information, algorithmique, langage et programmation ainsi que les architectures matérielles. De plus, notre idée ne semblait pas être déjà existante ou tout du moins très peu connue.

C'est ainsi que nous nous sommes lancés dans "*The Music Swagger*". L'idée du projet est de produire un système permettant de créer et faire varier des sons afin de générer une musique à partir de capteurs physiques (tels que gyroscope, mètres ultrasons, ou encore photodiodes). Nous avons souhaité privilégier l'ergonomie qui permet à un utilisateur non expérimenté de pourvoir simplement utiliser le projet grâce à un fonctionnement "Plug&Play" et une interface Web simplifiée.

II. La mise en œuvre

1. Les objectifs

Nous nous sommes donné pour objectif de réaliser un système de génération de son à partir de données physiques. Afin d'y parvenir, nous avons avons défini plusieurs modules.

- une partie "server" qui gère la synthèse du son ainsi que l'organisation des données
- une partie "device" qui s'occupe de rapatrier les données d'un capteur physique (accéléromètre par exemple) au serveur
- une partie "client" qui permet la configuration du système

La partie "server" doit permettre le stockage des données reçues des "devices" tout en gardant leur origine. Elle est elle-même composée de plusieurs sous-parties distinctes : une partie Synthèse, une partie Communication et une partie Configuration.

La partie "device" doit permettre un fonctionnement automatique et abstrait de capteurs : cette partie est composée d'un module de communication ainsi que d'un module de gestion du capteur. Chaque "device" doit être fonctionnel après une simple implémentation d'un capteur et une alimentation.

La partie "client" est basée sur une interface Web permettant une portabilité et une facilité d'utilisation. Elle doit permettre de choisir quel son sera généré à partir de quel "device" et comment. Elle doit être intuitive et elle fonctionne en "boite" "drag&drop".

2. Les logiciels

a) Outils

Afin de nous aider durant la phase de programmation et de reflexion, nous avons utilisé plusieurs logiciels. Le logiciel Web *Slack* (slack.com) a été choisi pour discuter et partager des morceaux de code et des idées au sein de notre groupe de travail. C'est une application permettant le "chat" ainsi que le partage de fichier et l'intégration de tierces parties comme *GitHub*.

Git par le biais de *GitHub* est donc le second logiciel dont nous nous sommes servis afin de partager le code, y avoir un accès constant et pouvoir connaître l'évolution du projet. Enfin, je me suis servi du logiciel *PyCharm Community Edition* qui est gratuit pour un usage personnel et non commercial et qui est très puissant quand à la programmation en python notamment grâce à une correction syntaxique, orthographique (peu utile en Python mais pratique) et une correction logique (utilisation de variables qui n'existent pas, conseils d'optimisation ou d'inutilité d'une partie du code).

Nous nous sommes servis d'un server *L(W)AMP* (Linux (Windows) - Apache - MySQL - PHP) pour le serveur Web. J'ai travaillé tout au long du projet sur un *Linux* (dérivé de *Debian* plus précisément) qui je pense est le logiciel le plus adapté à la programmation. Les ordinateurs *Raspberry* fonctionnent sous *Raspbian Light*. La totalité des logiciels que nous avons utilisé sont gratuits même s'ils ne sont pas tous open-source.

J'ai d'ailleurs proposé à Estelle et Nour de leur installer un système *Linux* sur une clé USB afin qu'ils puissent utiliser leur ordinateur personnel et pour leur expliquer tout ce dont ils avaient besoin sur *Git* et autres. C'était peut-être un peu compliqué pour eux, car après plusieurs propositions, je n'ai pas eu de clé USB de leur part. J'ai donc été le seul à publier le code sur *GitHub*.

b) Langages

Notre projet possède une complexité qui réside dans le nombre important de langages que nous utilisons. En effet, nous avons eu besoin des languages suivant : Python3, PHP (PHP : Hypertext Preprocessor), HTML5 (HyperText Markup Language), JS (JavaScript), CSS3 (Cascading Style Sheets) et le SQL (Structured Query Language).

Python3 Le Python3 a été utilisé pour une grande partie du projet notamment dans la communication entre le ”server” et les ”devices” ainsi que dans la génération du son.

PHP Le PHP nous à servi au niveau du ”server” pour permettre une communication avec la base de données.

HTML5, JS, CSS3 Ces languages nous ont servi dans la partie configuration du ”server” pour l'interface Web. Le JS a permis l'interaction avec l'utilisateur notamment pour le ”drag&drop”. Le CSS3 a servi pour l'esthétique des pages et l'HTML5 à la forme et au contenu de ces dernières.

SQL Le SQL nous a permis d'interagir entre le PHP et la base de donnée MySQL.

LaTeX J'ai utilisé ce langage pour écrire mon dossier.

c) Communication

Afin de faire communiquer les parties entre elles, nous avons utiliser plusieurs moyens.

Wi-Fi/UDP/IP Ce sont les protocoles utilisés pour la communication entre les ”devices” et le ”server”.

JSON C'est la façon dont nous envoyons la configuration à la page permettant de l'enregistrer via une requête ”POST”.

MusicSwaggerProtocol C'est le protocole que nous avons inventé afin de pouvoir faire communiquer les ”devices” et le ”server” basé sur UDP/IP.

d) Source d'information

Nous avons souvent utilisé des sites internet suivants :

`http://stackoverflow.com` pour des erreurs connues et des astuces

`http://developer.mozilla.org` pour tout ce qui touche au développement Web

Nous utiliserons ensuite les termes suivants : "device" pour qualifier l'ensemble Raspberry Pi et capteur, "server" pour l'ensemble serveur Web et serveur Python3. Pour permettre la communication entre les "devices" et le "server", nous avons opter pour un système de liaison par Wi-Fi, et des "devices" à base de Raspberry Pi Zero W pour la portabilité et le prix. Une interface Web sera mise en place afin de pouvoir configurer le comportement de façon simple et sans besoin d'application particulière. Nous avons choisis d'utiliser Python3 pour sa simplicité, sa portabilité ainsi que la richesse de ses modules. L'interface Web est composée de PHP pour l'interaction avec la base de donnée et l'HTML, JS et CSS pour la mise en forme et les interactions avec l'utilisateur (notamment avec le système de "Nodes" en "drag&drop").

3. Le rôle de chacun

Afin d'utiliser au mieux les compétences de chacun, il a fallu discuter de ce que chacun souhaitait faire et pouvait faire. Estelle débutait en informatique, Nour avait quelques compétences en programmation et moi qui est passionné par l'informatique je connaissais l'ensemble des langages et outils à utiliser pour réaliser notre projet.

J'ai pris à ma charge les développements les plus complexes (et les plus longs à développer) du projet, notamment une partie

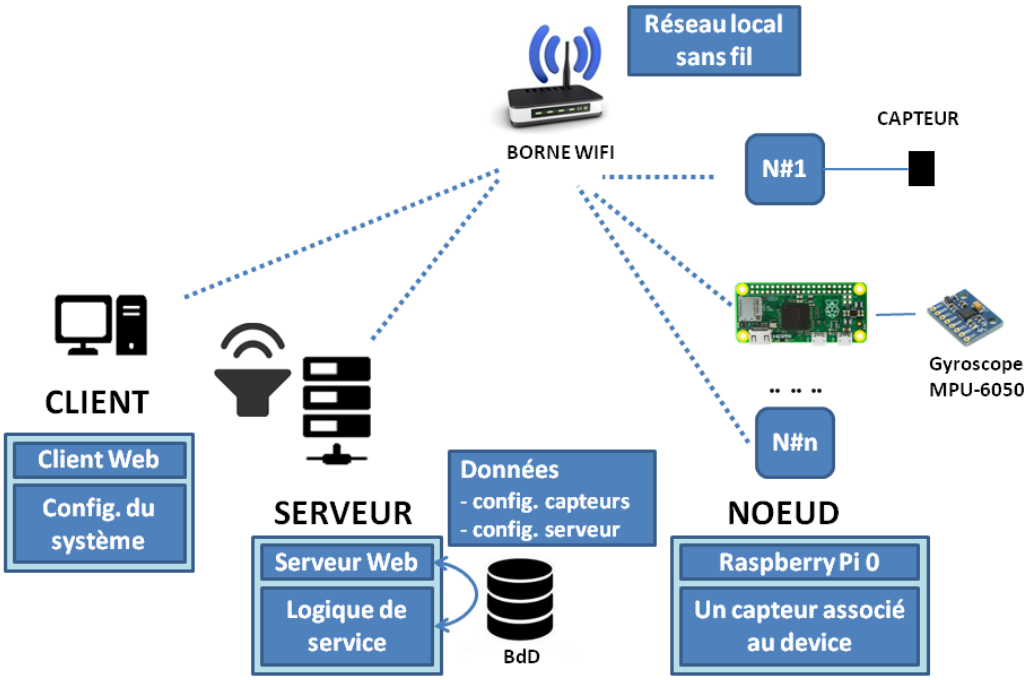
du ”server” qui permet de générer du son ainsi que la partie ”device” et la communication entre les deux. En effet, j’avais déjà quelques notions de communication réseau.

Taches	Planning	
Interface Web	≈20h	Nour, Estelle, Thomas
Communication	≈10h	Thomas
Device	≈10h	Thomas
Server	≈10h	Thomas, Nour, Estelle
Intégration	≈2h	Thomas

III. Réalisation

1. Mon travail

Voici une représentation du projet dans sa globalité.



”device” C’est la partie englobant le capteur, le ”DeviceBrain” et une partie de communication. Elle communique avec le ”server” grâce à un ”communicator” (interface de transfert de donnée par réseau Wi-Fi sur UDP/IP).

”server” C’est la partie qui permet de générer le son, elle met en forme et concrétise le projet. Elle contient une partie de génération de son, une partie de structuration des modi-

fications du son et d'une partie de communication avec les "devices". Elle a un accès à la base de données du "configurator" pour générer du son en fonction de ce que l'utilisateur désire.

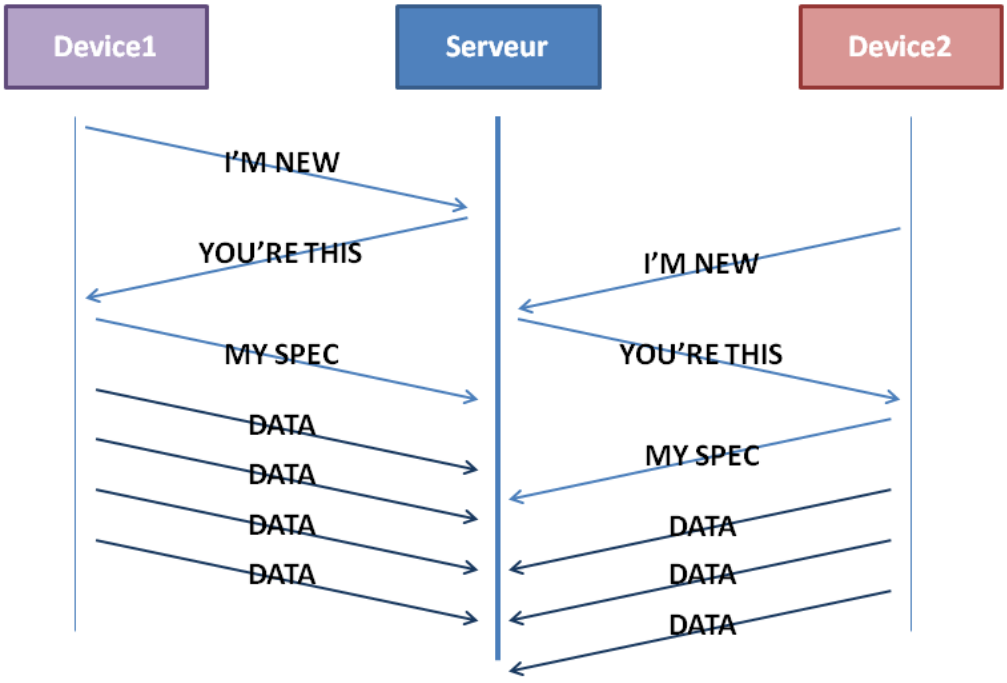
"configurator" C'est l'interface Web basé sur un serveur *LAMP*, indépendante du reste du projet par ses langages (différent de Python3). Elle communique ses données à travers une base de données. Elle est intuitive et fonctionne par "drag&drop" de boites et liens.

Je me suis dès le début chargé de structurer proprement le projet et de définir des interfaces simples et abstraites afin de faciliter le travail de chacun qui n'aurait alors qu'à s'occuper de sa partie en utilisant des fonctions simples et explicites. Exemple :

```
WaveGenerator().sinusoid(time=1000, sample_rate=22000, freq=440)
```

Chaque partie est délimitée par une classe Python.

Plus précisément, j'ai développé la partie de communication qui est présente du coté "device" et "server". Celle-ci s'occupe de toute la communication, le "server" et le "device" ne s'occupent de rien au niveau du réseau. Le "server" reçoit simplement une notification et les données envoyées par un "device" par le biais d'un "callback", c'est-à-dire une fonction qui est appelée lorsqu'un événement se passe. Le "communicator" possède son propre "Thread" ce qui lui permet de ne pas interférer avec la génération du son par exemple. (Voir annexes pour plus d'information sur la communication.)

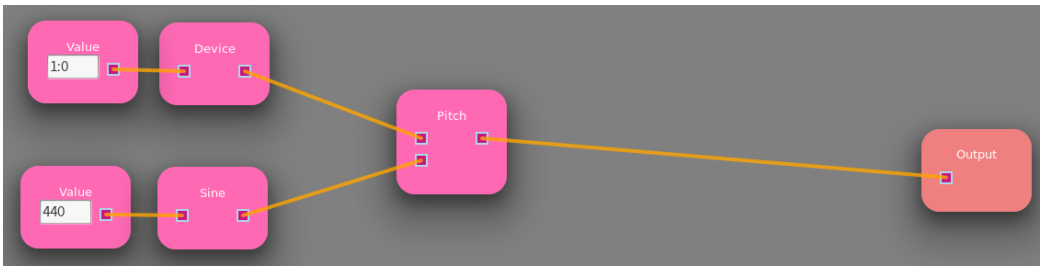


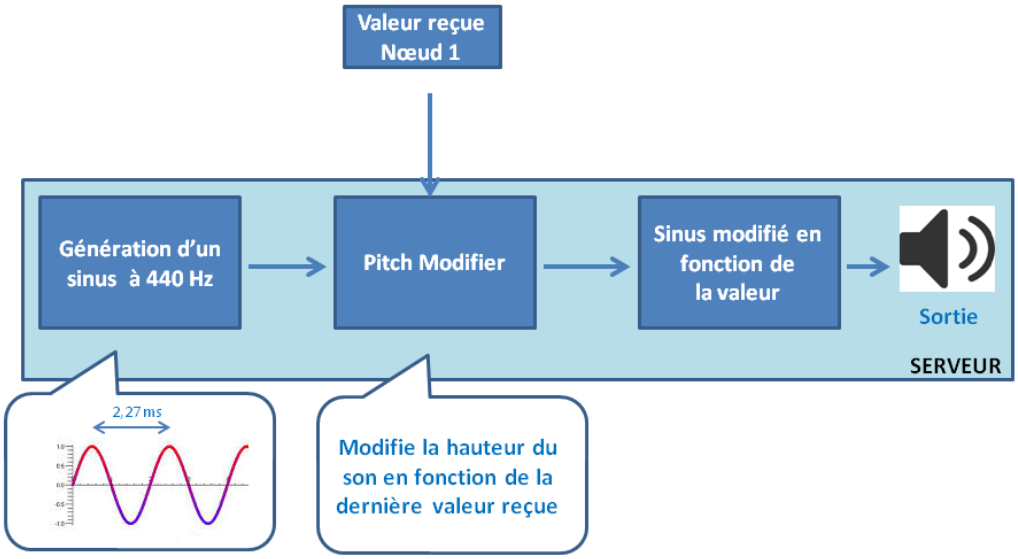
J'ai aussi développé la partie "device" qui fait le lien entre la partie capteur et "communicator". Cette partie est lancée au démarrage des Raspberry Pi et permet au "device" de commencer son fonctionnement.

Aussi, j'ai réalisé la partie de génération du son dans le "server". Ainsi, ce dernier peut aller chercher la dernière configuration disponible dans la base de données, retracer la structure en arbre ayant pour racine la sortie du son et générer à partir de là des portions de son de 100ms (intervalle minimal pour pouvoir avoir deux période complète de la plus basse fréquence audible). Il est donc doté d'une partie permettant la lecture de son et d'une partie permettant la génération de son. De plus, une inter-

face simple est disponible pour créer des ”boxes” qui permettent de prendre un(des) son(s) ou une(des) valeur(s) et de les traiter. Chaque ”box” représente un nœud de l’arbre.

Voici un exemple de configuration ainsi que son schéma fonctionnel :





2. Rendu final

Afin de ne pas trop entrer dans le détail, voici une explication simplifiée du fonctionnement du logiciel. Tout d'abord, le serveur Web *LAMP* et le "server" doivent être mis en marche.

Ensuite, l'utilisateur peut allumer les différentes "devices". Il va maintenant pouvoir configurer le *server* via l'interface Web.

Dès que le "server" détecte une configuration fonctionnelle, il va l'appliquer et générer le son. Le "device" dès son allumage envoie des données au "server" qui toute les 50ms va générer du son en fonction de ces dernières et de la configuration.

Enfin, le son est émis et peut être entendu.

3. Évolution

Le projet est fonctionnel mais il peut être amélioré. En effet, tout l'intérêt de ce projet est qu'il est extrêmement modulaire et très facilement améliorable. De plus, de nouveaux "devices" peuvent être implémentés en quelques lignes (détecteur de CO2, ...) et de nouveaux modificateurs de son pourraient être aisément ajoutés. Une fonctionnalité de partage de configuration sur internet pourrait être très inintéressante, ce qui permettrait en plus de pouvoir sauvegarder et restaurer une configuration. Enfin, une fonctionnalité d'import de fichiers son pourrait être un point à ajouter.

4. Bilan

Les premiers enseignements ont été riches. Nous nous sommes lancés dans une projet intéressant mais ambitieux. Nous avons beaucoup discuté du projet en cours d'ISN, mais aussi au CDI, et de nos discussions, je pensais que nous avions tous la même vision de ce qu'il fallait faire même si nous n'avancions pas au même rythme. Je ne me suis rendu compte que tardivement qu'Es-

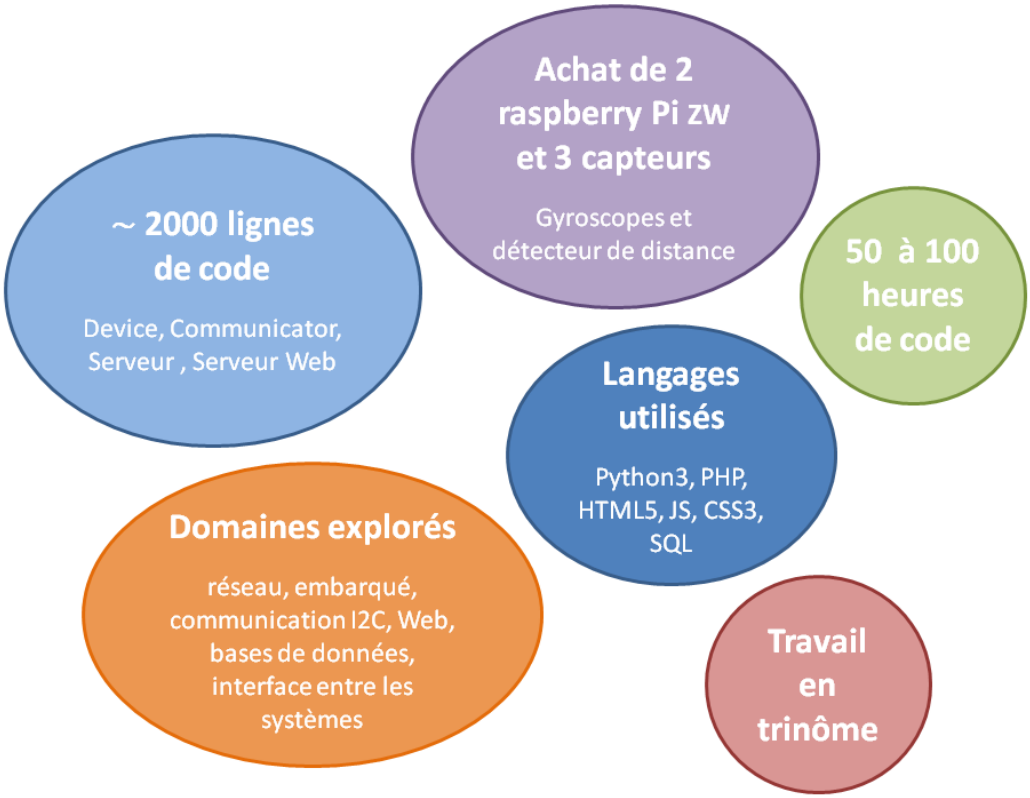
telle et Nour avaient plus de mal à avancer et je leur ai apporté mon support pour les aider à programmer en leur expliquant comment il fallait faire.

De plus, l'arrivée tardive du code de Nour et Estelle n'a permis de faire l'intégration de l'ensemble que tardivement, ce qui a nécessité de reprendre des parties de code déjà rédigées. C'est une partie du projet que je n'avais pas envisagé de cette façon : cela m'a pris beaucoup de temps en plus de ce qui était à faire et je pense que les outils collaboratifs auraient du être utilisés plus tôt pour échanger plus vite sur les problématiques rencontrées. Afin d'améliorer l'avancement du projet, il aurait fallu rédiger un cahier des charges détaillé dès le début du projet et peut-être travailler sur un projet moins complexe : le projet nous a séduit tous les trois, mais je crois que j'étais le seul en début de projet à bien appréhender la complexité des développements.

Il aurait été judicieux aussi de faire régulièrement des points de synchronisation avec nos développements respectifs, nos réunions étaient très utiles, mais peut-être pas assez précises et détaillées sur le contenu de nos travaux, les échanges étant principalement oraux.

Le bilan reste très positif, le projet fonctionne bien conformément aux objectifs que nous nous étions fixés et j'ai beaucoup appris sur le travail collaboratif. Plus globalement, j'ai aussi discuté et travaillé ponctuellement avec plusieurs personnes dans différents projets et j'ai tiré une grande satisfaction de pouvoir donner des idées, apporter un support aux différentes équipes qui pour certaines ont quelques lignes de code que j'ai pu rédiger.

Voici un bilan très synthétique du projet.



5. Licence

Ce projet est en open-source, sous la licence GPLv3, cela veut dire que n'importe qui est libre d'utiliser, de copier, de modifier et de distribuer le projet source ou dérivé, même pour de l'argent. Cependant en cas de modification, le résultat doit être placé sous la même licence. Nous avons choisi cette licence d'abord car elle est gratuite, et ensuite car nous voulions que notre projet puisse évoluer au maximum de ses capacités et parce que nous ne sommes pas contre l'idée que quelqu'un puisse commercialiser notre projet. C'est également dans cette optique que nous

avons rendu le code disponible sur GitHub, permettant à n’importe qui de proposer des ajouts.

Annexes

Music Swagger Protocol

```
MusicSwaggerProtocol v1.0 definition :
    - based on UDP/IP
    - simplest possible (shortest)

    - broadcast every packets

    - Server Port : 55666
    - Device Port : 55665

Data representation :
    | DEST CUID | SRC CUID | OP NUM | DATA LEN | DATA | ( CRC
    ↪  |)
    | (8b)      | (8b)     | (8b)   | (8b)     | (8b) | (
    ↪  (8b)|)

    - DESTination Connection Unique Identifier : the CUID of
    ↪  the destination of the packet (0x00 is the server, 0xff
    ↪  is no one particularly)
    - SouRCe CUID : the CUID of the source of the packet
    - OPeration NUMber : number describing which operation is
    ↪  given
    - DATA LENgth : the size of the data following in bytes
    - DATA : the actual data of the packet, formatted as the OP
    ↪  requires to
    (- CRC)

Constants :
    - DATA_VALUE_SIZE = 0x20
    -
```

```
Operations list :
    - OP NUM = OP NAME :
        - Description...
        | DATA DESC    |
        | (SIZE in bit)|
        - PARAMETER :
            - OPTION : DESC

    - 0x00 = INFO :
        - Generally for server response, give some info about
        ↪  the situation
        | ID  | INFO |
        | (8b)| (...)|
        - ID :
            - 0x00 : Nothing to say
            - 0x01 : Packet error
            - 0x02 : Invalid CUID
            - 0x03 : Not supposed to receive that
            # - 0x04 : I'm still alive ! (if not received for
            ↪  1min : forgetting the device)
            # TODO
        - INFO :
            Some text about the situation (ASCII).

    - 0x01 = IAMNEW :
        - When you start a device that needs a CUID to
        ↪  communicate over the network
        | GUID   |
        | (128b) |
        - GUID :
            The GUID of the device


    - 0x02 = YOURETHIS :
```

```
    - Server response to a 0x01 (give CUID)
    | GUID   | CUID |
    | (128b) | (8b) |
    - GUID :
        The GUID of the device
    - CUID :
        The new CUID of the device

-0x03 = MYSPEC :
    - give the device specs to server
    | NCHAN | NLEN | DLEN | NAME   | DESC   |
    | (8b)  | (8b) | (8b) | (NLENb)| (DLENb)|
    - NCHAN :
        The number of channel available
    - NLEN :
        The length of the device name
    - DLEN :
        The length of the description
    - NAME :
        The name of the device (utf8)
    - DESC :
        The description of the device (utf8)


- 0x10 = GIVE DATA
    - When device need to give data to server (can't work
    ↪  before a MYSPEC packet)
    | CHAN1            | ... |
    | (DATA_VALUE_SIZE)| ... |
    - CHAN1, ... :
        The values of each chanel

- 0x20 = GOODBYE
    - Before the device is turned off
```

Base de donnée

(Certaines bases sont vides car se remplissent durant le fonctionnement)

Table 1 : Content of table available_tools

ID	inputs	need_input	name	type
1	2	0	Pitch	PITCH
2	3	0	Add	ADD
3	0	1	Value	VALUE
4	1	0	Device	DEVICE
5	1	0	Sine	SINE
6	4	0	More	MORE
7	4	0	Less	LESS
9	0	0	Random	RAND
10	2	0	Multiply	MULTI
11	2	0	Sum	SUM
12	2	0	Doppler	DOP
13	2	0	Ampli	AMP
14	2	0	DIST	Distortion

Table 2 : Content of table boxes

ID	TYPE	BOX_ID	SPEC_PARAM

Table 3 : Content of table connections

ID	GUID	CUID	inited

Table 4 : Content of table links

ID	FROM_B	TO_B	WHERE_L

Table 5 : Content of table specifications

ID	numchan	name	description	CUID

Table 6 : Content of table update_number

ID

device.py

```
1  import time, random, threading, uuid, os, ctypes
2  from dev_config import Config as cfg
3  from communicator.communicator import Communicator
4
5
6  class Device(object):
7      """
8      C'est une classe permettant de definir clairement les
   ↪  caracteristiques d'un device, et de servir d'interface
   ↪  entre chaque device et d'autres parties comme des
   ↪  controlleurs.
9      Elle ne contient pas d'implementation mais uniquement la
   ↪  structure.
10     Il est recommande d'utiliser uniquement ces fonction et
   ↪  non celles ajoutees par une potentielle implementation
   ↪  pour des problemes de compatibilite.
11     """
12     num_of_chanels = None
13     # entier indiquant le nombre de "chanels" disponibles sur
       ↪  le device.
14     chanels = None
15     # liste de 'SensorValue' representant les dernieres
       ↪  valeurs enregistrees
```

```
16      status = None
17      # chaine de caractere definissant le statut actuel du
        ↪  device
18      name = ""
19      description = ""
20
21      # name and description of the device
22
23      def __init__(self):
24          super(Device, self).__init__()
25
26      def get_status(self):
27          """
28          :return: 'self.status'
29          """
30          return self.status
31
32      def get_fresh_values(self):
33          """
34          Cette fonction va simplement rafraichir les donees
    ↪  avant de les retourner.
35          :return: 'self.get_value()'
36          """
37          self.refresh()
38          return self.get_values()
39
40      def get_values(self):
41          """
42          Use the cfg to format values !
43          :return: array of formatted values
44          """
45          new_format = [0, 2 ** cfg.DATA_VALUE_SIZE - 1]
46          return [val.get_formated_values(new_format) for val in
            ↪  self.chanels]
```

```
47
48      def refresh(self):
49          """
50          Cette fonction va tenter de recuperer des donnees
    ↪   "fraiches" et les stocker dans la self.last_value.
51          WARNING : Elle peut donc prendre un certain temps a
    ↪   s'executer !
52          """
53          raise NotImplementedError()
54
55      def get_num_of_chanels(self):
56          """
57          :return: 'self.num_of_chanels'
58          """
59          return self.num_of_chanels
60
61      def get_name(self):
62          return self.name
63
64      def get_description(self):
65          return self.description
66
67
68  class ThreadedDevice(Device, threading.Thread):
69      """
70      Une version sur thread de device (dans le cas ou
    ↪   l'appareil aurait besoin d'un rafraichissement constant.
71      """
72      is_killed = None
73      # booleen qui permet de stopper la boucle principal du
        ↪   thread et donc de l'arreter
74      is_running = None
75      # booleen qui permet de mettre en pause le thread
76      refresh_interval = None
```

```
77      # interval de relancement de 'self.refresh' (must be over
        ↪  50 because of the music sample)
78      callback = None
79
80      # called when refresh is finished
81
82      def __init__(self, refresh_interval, callback=None):
83          super(ThreadedDevice, self).__init__()
84          self.daemon = True
85          # not waiting thread to stop before exiting program
86          self.set_refresh_interval(refresh_interval)
87          self.is_running = False
88          self.is_killed = False
89          self.set_callback(callback)
90          self.init()
91          self.start()
92
93      def init(self):
94          """
95          Implement if the device you need initialisation
96          """
97          return
98
99      def run(self):
100         """
101         Fonction execute dans un different thread.
102         """
103         self.is_running = True
104         while not self.is_killed:
105             time.sleep(self.refresh_interval / 1000)
106             if self.is_running:
107                 self.refresh()
108                 if self.callback is not None:
109                     self.callback(self)
```

```
110
111     def pause(self):
112         """
113         Pause the refreshing.
114         """
115         self.is_running = False
116
117     def play(self):
118         """
119         Allow to refresh.
120         """
121         self.is_running = True
122
123     def kill(self):
124         """
125         Stop the thread.
126         """
127         self.pause()
128         self.is_killed = True
129         self.join()
130
131     def set_refresh_interval(self, refresh_interval):
132         self.refresh_interval = refresh_interval
133
134     def set_callback(self, callback):
135         self.callback = callback
136
137
138 class DeviceChanel(object):
139     """
140     Petite classe permettant de stocker en plus de la
    ↪  derniere valeur, l'interval de valeurs possibles.
141     """
142     value_range = None
```

```
143     # liste de deux elements representant intervalle de
        ↪ valeurs possibles
144     last_value = 0
145
146     # entier representant la valeur actuelle du sensor
147
148     def __init__(self, value_range):
149         self.value_range = value_range
150
151     def set_value(self, value):
152         """
153         :param value: nouvelle valeur
154         """
155         if value < self.value_range[0]:
156             value = self.value_range[0]
157         if value > self.value_range[1]:
158             value = self.value_range[1]
159         self.last_value = value
160
161     def check_range(self, value):
162         return self.value_range[0] <= value <=
            ↪ self.value_range[1]
163
164     def get_value(self):
165         """
166         :return: 'self.last_value'
167         """
168         return self.last_value
169
170     def get_range(self):
171         """
172         :return: 'self.value_range'
173         """
174         return self.value_range
```

```
175
176     def get_formated_values(self, new_format):
177         """
178         :param new_format: liste 2 elements [min,max]
179         :return: 'self.get_value()' formate avec l'interval
    ↪  'new_format'
180         """
181         if self.get_value() != None:
182             return int((((self.get_value() -
                ↪  self.get_range()[0]) * (new_format[1] -
                ↪  new_format[0])) / (
183             self.get_range()[1] - self.get_range()[0])) +
                ↪  new_format[0])
184         return None
185
186
187 class Random2Device(ThreadedDevice):
188     num_of_chanels = 2
189     chanels = [DeviceChanel([-100, 100]) for i in
        ↪  range(num_of_chanels)]
190     name = "MyRandom2Device"
191     description = "This is a test device giving random values."
192
193     def init(self):
194         self.status = "Started !"
195
196     def refresh(self):
197         for chan in self.chanels:
198             ran = chan.get_range()
199             chan.set_value(random.randint(ran[0], ran[1]))
200         self.status = "Refreshed to : " +
            ↪  str(self.get_values())
201
202
```

```
203 class HCSR04Device(ThreadedDevice):
204     """
205     HCSR04 device : distance ultrason (RPi GPIO seulement)
206 
    ↪   (https://electrosome.com/hc-sr04-ultrasonic-sensor-raspberry-pi/)
207     """
208     num_of_chanels = 1
209     chanels = [DeviceChanel([1, 200])]
210     name = "HCSR04UltrasonicGPIOSensor"
211     description = "Implementation for the HCSR04 ultrasonic
        ↪  sensor giving a distance based on an echo sound."
212     GPIO = None
213     # gpio library pointer
214     timeout = 1000
215 
216     # timeout for sensor in ms
217     def init(self):
218         import RPi.GPIO
219         self.GPIO = RPi.GPIO
220         self.GPIO.setmode(self.GPIO.BCM)
221         self.trigger_pin = 23
222         self.echo_pin = 24
223         self.GPIO.setup(self.trigger_pin, self.GPIO.OUT)
224         self.GPIO.setup(self.echo_pin, self.GPIO.IN)
225 
226     def refresh(self):
227         beginning = time.time()
228         self.GPIO.output(self.trigger_pin, False)
229         time.sleep(0.1)
230         self.GPIO.output(self.trigger_pin, True)
231         time.sleep(0.00001)
232         self.GPIO.output(self.trigger_pin, False)
233         pulse_start, pulse_end = 0, 0
234         while self.GPIO.input(self.echo_pin) == 0:
```

```
235             pulse_start = time.time()
236             if pulse_start - beginning > self.timeout / 1000:
237                 return
238         while self.GPIO.input(self.echo_pin) == 1:
239             pulse_end = time.time()
240             if pulse_end - pulse_start > self.timeout / 1000:
241                 return
242         pulse_duration = pulse_end - pulse_start
243         distance = int(pulse_duration * 17000)
244         self.chanels[0].set_value(distance)
245 
246     def kill(self):
247         super(HCSR04Device, self).kill()
248         self.GPIO.cleanup()
249 
250 
251 class L3GD20Device(ThreadedDevice):
252     # constants
253     L3GD20_ADDRESS = 0x6B
254     L3GD20_POLL_TIMEOUT = 100
255     L3GD20_ID = 0xD4
256     L3GD20H_ID = 0xD7
257 
258     L3GD20_REGISTERS = {
259         "WHO_AM_I": 0x0F,
260         "CTRL_REG1": 0x20,
261         "CTRL_REG2": 0x21,
262         "CTRL_REG3": 0x22,
263         "CTRL_REG4": 0x23,
264         "CTRL_REG5": 0x24,
265         "REFERENCE": 0x25,
266         "OUT_TEMP": 0x26,
267         "STATUS_REG": 0x27,
268         "OUT_X_L": 0x28,
```

```
269             "OUT_X_H": 0x29,
270             "OUT_Y_L": 0x2A,
271             "OUT_Y_H": 0x2B,
272             "OUT_Z_L": 0x2C,
273             "OUT_Z_H": 0x2D,
274             "FIFO_CTRL_REG": 0x2E,
275             "FIFO_SRC_REG": 0x2F,
276             "INT1_CFG": 0x30,
277             "INT1_SRC": 0x31,
278             "TSH_XH": 0x32,
279             "TSH_XL": 0x33,
280             "TSH_YH": 0x34,
281             "TSH_YL": 0x35,
282             "TSH_ZH": 0x36,
283             "TSH_ZL": 0x37,
284             "INT1_DURATION": 0x38
285         }
286
287         L3GD20_SENSIBILITY = {
288             "250DPS": 0.00875,
289             "500DPS": 0.0175,
290             "2000DPS": 0.07
291         }
292
293         L3GD20_RANGE = {
294             "250DPS": 0x00,
295             "500DPS": 0x10,
296             "2000DPS": 0x20
297         }
298
299         # default sensitivity
300         sensibility = "250DPS"
301
302         # used to send and receive with from sensor
```

```
303     bus = None
304
305     num_of_chanels = 4
306     chanels = [DeviceChanel([0, 2 ** 16]) for _ in range(3)] +
        ↪  [DeviceChanel([0, 255])]
307     name = "L3GD20"
308     description = "Implementation for the L3GD20 gyroscope
        ↪  giving the 3D angular speed and the temperature of the
        ↪  sensor."
309
310     smbus = None
311
312     # smbus library pointer
313     def init(self):
314         import smbus
315         self.smbus = smbus
316         self.bus = self.smbus.SMBus(1)
317
318         # checking sensor type
319         id = self.read_byte(self.L3GD20_REGISTERS["WHO_AM_I"])
320         if id == self.L3GD20_ID:
321             print("Sensor is L3GD20 !")
322         elif id == self.L3GD20H_ID:
323             print("Sensor is L3GD20H !")
324         else:
325             raise Exception("Sensor unrecognized !")
326
327         # initializing sensor
328         # reset to normal
329         self.write_byte(self.L3GD20_REGISTERS["CTRL_REG1"],
            ↪  0x00)
330         # turn the 3 channels on
331         self.write_byte(self.L3GD20_REGISTERS["CTRL_REG1"],
            ↪  0x0F)
```

```
332
333             # set resolution to "self.sensibility"
334             self.write_byte(self.L3GD20_REGISTERS["CTRL_REG4"],
                ↪  self.L3GD20_RANGE[self.sensibility])
335
336         def get_orientation(self):
337             x1, x2 =
                ↪  self.read_byte(self.L3GD20_REGISTERS["OUT_X_L"]),
                ↪  self.read_byte(self.L3GD20_REGISTERS["OUT_X_H"])
338             y1, y2 =
                ↪  self.read_byte(self.L3GD20_REGISTERS["OUT_Y_L"]),
                ↪  self.read_byte(self.L3GD20_REGISTERS["OUT_Y_H"])
339             z1, z2 =
                ↪  self.read_byte(self.L3GD20_REGISTERS["OUT_Z_L"]),
                ↪  self.read_byte(self.L3GD20_REGISTERS["OUT_Z_H"])
340             x = int((x1 | (x2 << 8)))  #
                ↪  *self.L3GD20_SENSIBILITY[self.sensibility])
341             y = int((y1 | (y2 << 8)))  #
                ↪  *self.L3GD20_SENSIBILITY[self.sensibility])
342             z = int((z1 | (z2 << 8)))  #
                ↪  *self.L3GD20_SENSIBILITY[self.sensibility])
343             return [x, y, z]
344
345         def get_temperature(self):
346             t = self.read_byte(self.L3GD20_REGISTERS["OUT_TEMP"])
347             if (t & 128) != 0:
348                 t = t - 256
349             return -t + 128
350
351         def read_byte(self, register):
352             return self.bus.read_byte_data(self.L3GD20_ADDRESS,
                ↪  register)
353
354         def write_byte(self, register, value):
```

```
355             return self.bus.write_byte_data(self.L3GD20_ADDRESS,
                ↪  register, value)
356
357         def refresh(self):
358             res = self.get_orientation() + [self.get_temperature()]
359             for i in range(self.num_of_chanels):
360                 self.chanels[i].set_value(res[i])
361
362
363     class GY521(ThreadedDevice):
364         """
365         A I2C sensor which contains a accelerometer, a gyroscope
        ↪  and a thermometer.
366         """
367         I2C_ADDRESS = 0x68
368         I2C_REGISTERS = {
369             "PWR_MGMT_1": 0x6B,
370             "ACCEL_XOUT_H": 0x3B,
371             "ACCEL_XOUT_L": 0x3C,
372             "ACCEL_YOUT_H": 0x3D,
373             "ACCEL_YOUT_L": 0x3E,
374             "ACCEL_ZOUT_H": 0x3F,
375             "ACCEL_ZOUT_L": 0x40,
376             "TEMP_OUT_H": 0x41,
377             "TEMP_OUT_L": 0x42,
378             "GYRO_XOUT_H": 0x43,
379             "GYRO_XOUT_L": 0x44,
380             "GYRO_YOUT_H": 0x45,
381             "GYRO_YOUT_L": 0x46,
382             "GYRO_ZOUT_H": 0x47,
383             "GYRO_ZOUT_L": 0x48,
384
385         }
386
```

```
387     # used to send and receive with from sensor
388     bus = None
389     # smbus library pointer
390     smbus = None
391
392     num_of_chanels = 8
393     chanels = [DeviceChanel([0, 2 ** 16]) for _ in range(8)]
394     name = "GY521"
395     description = "Implementation for the GY521 gyroscope and
        ↪ accelerometer giving the acceleration, angular speed
        ↪ and the temperature of the sensor and also the computed
        ↪ overall acceleration. (ax,ay,az,t,gx,gy,gz,as)"
396
397     def init(self):
398         import smbus
399         self.smbus = smbus
400         self.bus = self.smbus.SMBus(1)
401
402         # power device up
403         self.write_byte(self.I2C_REGISTERS["PWR_MGMT_1"], 0)
404
405     def read_byte(self, register):
406         return self.bus.read_byte_data(self.I2C_ADDRESS,
            ↪ register)
407
408     def write_byte(self, register, value):
409         return self.bus.write_byte_data(self.I2C_ADDRESS,
            ↪ register, value)
410
411     def get_raw_data(self):
412         ax =
            ↪ ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["ACCEL_XOUT_H"])
            ↪ << 8 | self.read_byte(
413             self.I2C_REGISTERS["ACCEL_XOUT_L"])).value
```

```
414             ay =
                ↪  ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["ACCEL_YOUT_H"])
                ↪  << 8 | self.read_byte(
415                 self.I2C_REGISTERS["ACCEL_YOUT_L"])).value
416             az =
                ↪  ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["ACCEL_ZOUT_H"])
                ↪  << 8 | self.read_byte(
417                 self.I2C_REGISTERS["ACCEL_ZOUT_L"])).value
418             t =
                ↪  ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["TEMP_OUT_H"])
                ↪  << 8 | self.read_byte(
419                 self.I2C_REGISTERS["TEMP_OUT_L"])).value
420             gx =
                ↪  ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["GYRO_XOUT_H"])
                ↪  << 8 | self.read_byte(
421                 self.I2C_REGISTERS["GYRO_XOUT_L"])).value
422             gy =
                ↪  ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["GYRO_YOUT_H"])
                ↪  << 8 | self.read_byte(
423                 self.I2C_REGISTERS["GYRO_YOUT_L"])).value
424             gz =
                ↪  ctypes.c_int16(self.read_byte(self.I2C_REGISTERS["GYRO_ZOUT_H"])
                ↪  << 8 | self.read_byte(
425                 self.I2C_REGISTERS["GYRO_ZOUT_L"])).value
426             return [ax, ay, az, t, gx, gy, gz]
427
428         def refresh(self):
429             raw = self.get_raw_data()
430             for i in range(7):
431                 self.chanels[i].set_value(raw[i])
432             accsum = pow(pow(raw[0], 2) + pow(raw[1], 2) +
                ↪  pow(raw[2], 2), 0.5)
433             self.chanels[7].set_value(accsum)
434
```

```
435
436 implemented_devices = {
437     "L3GD20Device": L3GD20Device,
438     "HCSR04Device": HCSR04Device,
439     "Random2Device": Random2Device,
440     "GY521": GY521
441 }
442
443
444 class DeviceBrain(object):
445     global_uid = None
446     device = None
447     communicator = None
448
449     def __init__(self, device):
450         if os.path.isfile(cfg.GUID_FILENAME):
451             guidfile = open(cfg.GUID_FILENAME, "r")
452             self.global_uid = guidfile.read()
453             guidfile.close()
454         else:
455             guidfile = open(cfg.GUID_FILENAME, "w")
456             self.global_uid = str(uuid.uuid4()).replace("-",
                ↪  "")
457             guidfile.write(self.global_uid)
458             guidfile.close()
459         cfg.log(self.get_guid())
460         self.device = device
461         self.communicator = Communicator(False,
            ↪  self.get_guid())
462         cfg.log("Ok")
463         # device mode of communicator
464
            ↪  self.communicator.give_my_spec(device.get_num_of_chanels(),
            ↪  device.name, device.description)
```

```
465             if isinstance(self.device, ThreadedDevice):
466                 self.device.set_callback(self.send_data_to_serv)
467 
468         def get_guid(self):
469             """
470             :return: 'self.global_uid'
471             """
472             return self.global_uid
473 
474         def send_data_to_serv(self, device):
475             print(device.get_values())
476             if self.communicator.is_ready():
477 
                    ↪  self.communicator.give_data_packet(device.get_values())
478 
479         def stop(self):
480             self.device.kill()
481             self.communicator.stop()
482 
483 
484     if __name__ == "__main__":
485         device = brain = None
486         try:
487             device = implemented_devices[cfg.SENSOR](100)
488             brain = DeviceBrain(device)
489             while True: continue
490         except KeyboardInterrupt as e:
491             print("<Ctrl-c> = user quit")
492         finally:
493             print("Exiting...")
494             if brain:
495                 brain.stop()
```

GitHub

(Tout le code ne peut pas être disponible en annexe dans un souci de gâchis de papier et de lisibilité.)

`http://github.com/TheMusicSwagger/Device` La partie ”device” du projet

`http://github.com/TheMusicSwagger/Server` La partie ”server” du projet

`http://github.com/TheMusicSwagger/Communicator` La partie ”communicator” du projet

`http://github.com/TheMusicSwagger/Documentation` La documentation du projet