Université de Lille 1

École Polytechnique Universitaire de Lille
Département Informatique-Microélectronique-Automatique

23/02/2016

Pilulier connecté

Rapport de projet de fin d'études

Élèves :
Corentin DUPLOUY & Manouk SIMON

Encadrants :
Alexandre BOE & Thomas VANTROYS

Table de matières

Remerciements

Tout d’abord, nous tenons à remercier, toute l’équipe pédagogique de Polytech’Lille et les responsables de la formation Informatique-Microélectronique-Automatique de nous avoir enseigné les bases pour réaliser ce projet de fin d’études dans de bonnes conditions.

Nous souhaitons également remercier et témoigner toute notre reconnaissance à Monsieur Alexandre BOE, notre responsable de projet, pour son aide précieuse et sa disponibilité tout au long du projet, ainsi que Monsieur Thomas VANTROYS.

Introduction

Dans le cadre de notre dernière année de formation d'ingénieur en Informatique-Microélectronique-Automatique, à Polytech Lille, nous avons eu à réaliser un projet de fin d'études aménagé qui se déroule le long de 4 semaines.

C'est l'occasion pour nous, étudiants, de mettre en application des connaissances acquises depuis le début de notre formation et de les perfectionner autour d'un sujet complet, de l'élaboration du cahier des charges à une éventuelle réalisation finale.

Notre projet s'inscrit dans le domaine de l'e-santé. L'objectif est de concevoir une base qui rendrait un pilulier connecté au smartphone de son utilisateur.

Nous équiperons ce capteur d'un BLE ainsi que de plusieurs capteurs qui permettront d'avertir l'utilisateur dans quel compartiment journalier il doit prendre ses médicaments. De plus, nous réaliseront une application Android qui jouera un grand rôle dans le suivi du patient et les alertes en cas d'oubli.

1. Présentation du projet

1.1. Contexte du projet

D'après des statistiques établis en 2014, un meilleur suivi des traitements médicaux et des médicaments dans le monde permettrait de réaliser 8% d'économies sur le coût total des dépenses de santé.

De plus, on estime en Europe que la mauvaise prise de médicaments est à l'origine de plusieurs dizaines de milliers de décès chaque année.
Selon le rapport IGAS2, en France, un mauvais suivi des heures de prises et de dosage de médicament aurait causé entre 8 000 à 12 000 décès par an et coûté près de 2 milliards € à l'Assurance Maladie. Treize millions de patients résidents en France seraient polymédiqués, et parmi eux de nombreuses personnes sont âgées de plus de 60 ans.

Depuis l'essor des Nouvelles Technologies, la santé connectée est devenue un marché porteur, et est appelée à être la solution aux mauvais suivis et traitements pharmaceutique.
C'est ainsi que de nombreuses solutions se développent chaque jour qui a pour but de venir en aide aux personnes âgées dans les horaires de leurs prises de médicaments et éviter les surdosages ou les erreurs de traitement.

1.2. Description du projet

Ce projet a pour but d'améliorer le suivi de traitement médical d'un patient, en permettant à celui-ci d'ajouter une base à son pilulier hebdomadaire pour le rendre connecté à son smartphone. Il pourra ainsi être alerté, via l'application Android PillBox, s'il doit prendre un médicament à un moment de la journée. Le docteur de l'utilisateur est averti en temps réel, lorsque celui-ci n'a pas pris son traitement au bout d'une durée déterminée.

Le suivi médical permet d'établir une relation de confiance entre tous les intervenants du monde médical, allant du docteur au patient en passant par le pharmacien. Ce suivi doit permettre au patient de se responsabiliser et faciliter la prise de son traitement médical.

1.3. Cahier des charges

Actuellement, il existe sur le marché différentes propositions de pilulier connecté dont chacun avec leurs caractéristiques propres :

- *Imedipac* de *Medissimo* : envoie un SMS et/ou un email à l'utilisateur pour lui rappeler de prendre tel médicament à telle heure.
- *Do-Pill* de *SecuR* : rend accessible les comprimés et gélules à la date et à l'heure indiqué par le médecin pour éviter les surdosages ou les erreurs de traitement.
- *Sivan* de *MedSecure* : prévient l'entourage si erreur du patient dans la prise de ses comprimées et alerte le patient si il doit prendre un comprimé.

Afin de proposer une solution alternative à ses piluliers connectés déjà existant, le patient pourra utiliser son pilulier hebdomadaire standard auquel il intégra une base qu'il le rendra connecté.

De plus, nous devons prendre en compte la donnée suivante : le pilulier doit être transportable. Cela sous-entend d'avoir une base suffisamment compact et qui puisse s'adapter à un pilulier standard.

Cette base permettra au pilulier de remplir les fonctions suivantes :

- Emission d'une alarme lorsque l'usager doit prendre une pilule à une heure et date précise.
- Indication du médicament à ingurgiter dans la boîte du jour et du compartiment de cette journée, via un système de LEDs.
- Détection si l'usager a pris la bonne journée, sinon émission d'une alarme.
- Détection si l'usager a pris la bonne pilule à la bonne heure, sinon émission d'une alarme.

Le suivi du pilulier s'effectue à travers une application Android. L'application permet également d'indiquer le bon compartiment et boîte où récupérer le médicament à ingurgiter.

1.4. Étapes du projet

Après étude du sujet et du cahier des charges nous avons divisé le travail en plusieurs étapes. Nous avons adapté notre planning en fonction des composants que nous avons commandé et reçu. Nous avons aussi privilégié d'abord la conception et l'étude du socle du pilulier et la programmation de chaque module (RTC, BLE, application Android) ; avant de se pencher sur l'interfaçage de tous les modules.

Étape 1 :

- Etude bibliographique
- Schéma du système avec les différents modules à implémenter
- Liste des composants nécessaires pour le projet

Étape 2 :

- Conception du circuit électronique pour l'affichage des LED
- Test non-validé de la détection infra-rouge à partir de composants qui ne sont pas en CMS
- Programmation du RTC et du BLE
- Développement de l'application Android
- Programmation de la communication Bluetooth entre l'application Android et le BLE

Étape 3 :

- Conception des cartes PCB pour réaliser la base connectée du pilulier
- Interfaçage entre la base connectée et l'Arduino
- Interfaçage entre l'Arduino et le BLE

2. Présentation de la réalisation hardware

Cette première partie renseigne sur les différents éléments qui composent le pilulier connecté et présente les différentes fonctionnalités de celui-ci.

2.1. Les composants Arduino

Le prototype, que nous avons tenté de mettre au point, est composé de plusieurs éléments.

L'Arduino Mega, basée sur le microcontroleur ATmega1280 de chez Atmel, est le coeur du prototype. L'ensemble du frimware du Dreamer que nous avons développé, est flashé sur cet Arduino.

La shield MOD-RTC de chez Olimex est interfacée avec l'Arduino via la communication Wire. Cette shield, utilisant une horloge temps réel PCF8563, permet de connaître l'heure et la date en temps réel afin d'alerter l'utilisateur le médicament à prendre dans le compartiment approprié au bon moment de la journée. Ainsi, l'exactitude des données est fidélisée au maximum.
La date et l'heure du RTC s'initialisent automatiquement à la date et à l'heure de la compilation du programme, lors de la fonction setup de l'Arduino.

Enfin, la dernière shield qui compose le prototype est un RFduino RFD22301. C'est l'élément central pour la communication avec l'utilisateur. Le pilulier envoi les informations sur la présence ou non de médicament dans les compartiments à l'application Android, installé sur le smartphone de l'utilisateur. A partir de ce dernier, une alarme retenti à chacune des quatre périodes de la journée (matin, midi, soir et nuit) si l'utilisateur doit prendre un médicament dans le compartiment du pilulier approprié.
Il est à noter que le RFduino est interfacé avec l'Arduino Micro à travers une communication I^2C.

2.2. La boîte journalière

Chaque boîte embarque de l'électronique pour permettre l'éclairage d'une LED indiquant le bon compartiment pour quelques minutes. Après plusieurs solutions testées, nous avons décidé de choisir la suivante.

Tout s'articule autour d'un ATtiny13 préalablement programmé à l'aide du code Annexe 1. Comme on peut le voir ci-contre, nous disposons d'un pin Vcc qui alimente un super condensateur pour faire fonctionner le microprocesseur un certain temps, le pin de masse GND, les pins pour contrôler les LED et un pin recevant une liaison série de type UART. En effet, le principe est le suivant : Par défaut, le signal reçu par le pin UART est à l'état haut. A la détection d'un front descendant, le microprocesseur échantillonne à chaque moitié de cycle d'horloge pour obtenir un code de 8 bits.

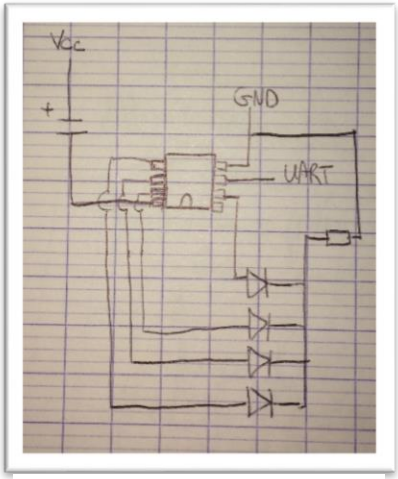


Figure 1. Schematic de l'électronique de la boîte

Le code envoyé est le suivant :

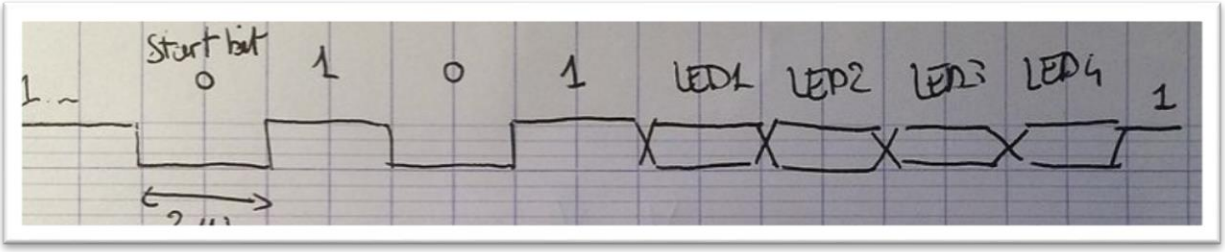


Figure 2. Code UART envoyé à l'ATtiny13

Après 4 bits de vérification, on enverra l'un des bits LED1, LED2, LED3 ou LED4 à l'état haut pour indiquer la LED qu'on souhaite allumer.

Pour réaliser tout cela, nous avons gravé 3 cartes PCB qui sont les suivantes :

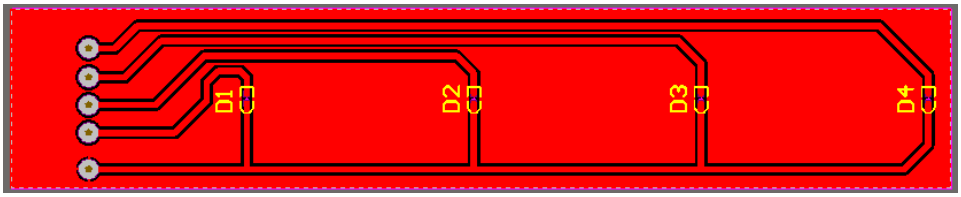


Figure 3. PCB de la carte des LED

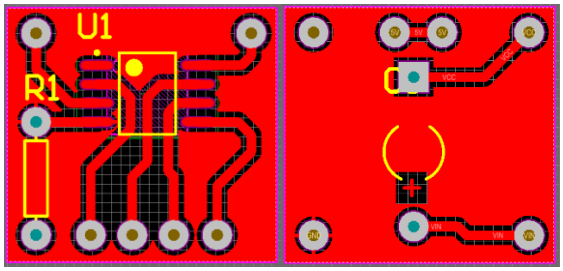


Figure 4. PCB de l'ATtiny et du super condensateur

Une fois tous les composants soudés, voici à quoi ressemble notre boîte journalière avec son électronique embarquée. Les deux dernières cartes verticales ne comportent pas leurs composants car ils ne sont toujours pas arrivés à l'heure actuelle.
On peut cependant avoir un aperçu de la boîte sur la photo ci-dessous.

Figure 5. Allure de la boîte

Voici la carte des LED sous la boîte :

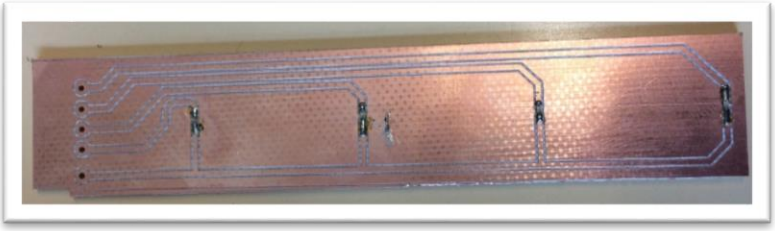


Figure 6. Carte des LED sous la boîte

Elle est donc composée de 4 LED verte CMS, 4 trous pour les pins de contrôle de LED et 1 pour le passage de la masse.

2.3. Le support

Le support, lui, est composé de l'Arduino micro, du RTC, du RFduino ainsi que le système de détection Infrarouge.
Selon le schematic suivant, on crée un faisceau Infrarouge à l'aide d'un diode IR et d'une photorésistance de même longueur d'onde pour éviter les interferances avec la lumière ambiante.

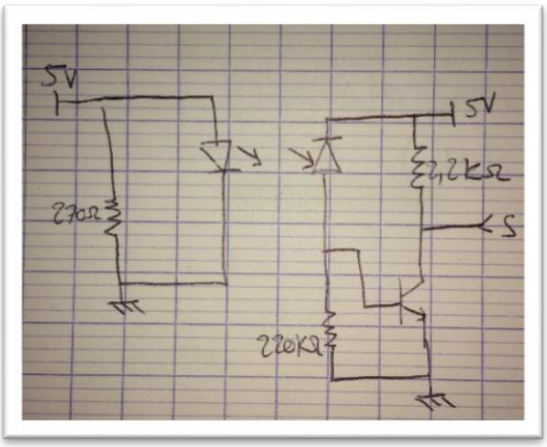


Figure 7. Schematic faisceau Infrarouge

Lorsque la photorésistance capte l'Infrarouge de la diode, un micro courant circule et permet de rendre le transistor NPN passant. Le potentiel au nœud « S » est donc 0. Au contraire, s'il y a présence d'un obstacle empêchant la photorésistance de capter le rayon Infrarouge, le transistor de peut être activé et est donc bloquant. Le potentiel au nœud « S » est 5V ('1' logique).
On peut ainsi détecter la présence d'objet entre les deux composants, dans le cas présent, un quelconque médicament.

Nous avons réalisé le test avec le circuit suivant :

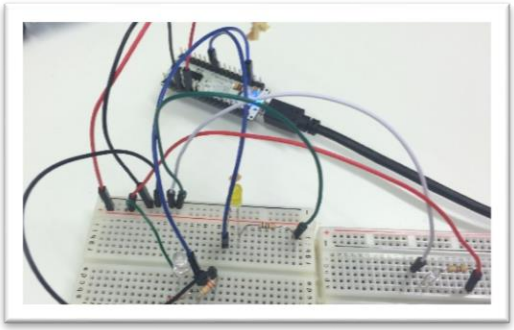


Figure 8. Circuit de Test du faisceau IR

Avec le code disponible en Annexe 2, on programme l'Arduino micro pour allumer la LED en présence d'un obstacle.
Voici deux photos qui confirment le fonctionnement de cette méthode :

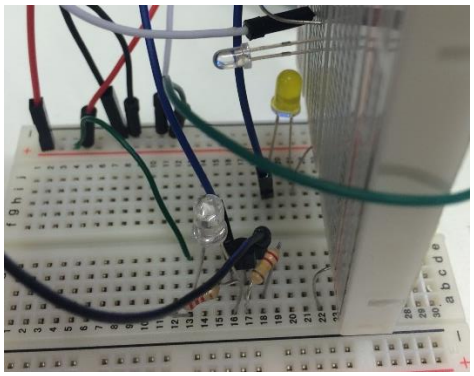


Figure 9. Faisceau sans obstacle - LED éteinte

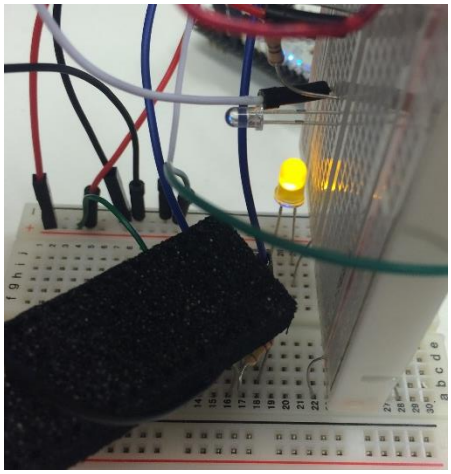


Figure 10. Faisceau avec obstacle - LED allumée

Les diodes IR et photorésistances devant être de part et d'autre de la boîte, il était nécessaire de réaliser deux cartes PCB :

Figure 11. Carte PCB des diodes IR

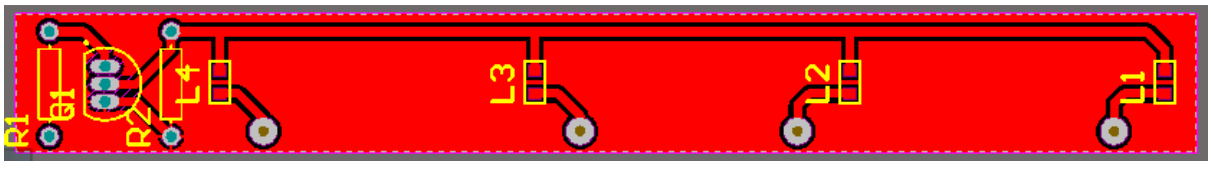


Figure 12. Carte PCB des photorésistances

Ci-dessous, nous pouvons voir les deux cartes soudées avec leurs composants.

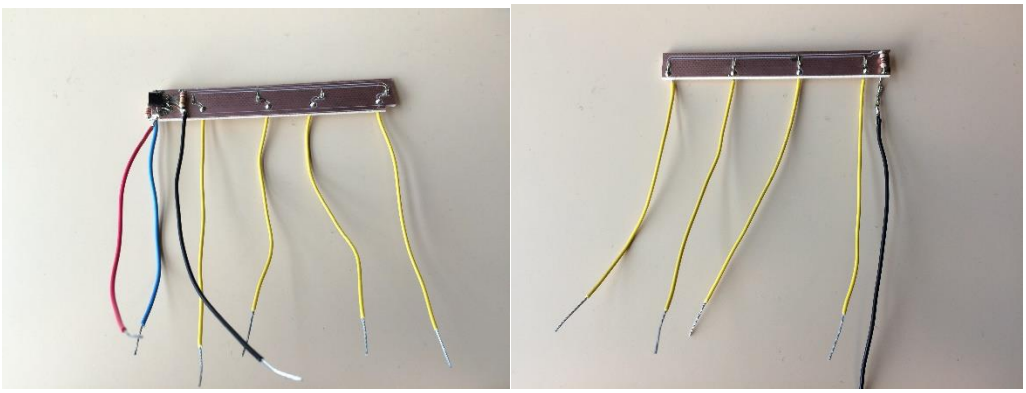


Figure 13. A gauche la carte des photorésistances - A droite la carte des diodes IR

Le code couleur est le suivant :
Le rouge, l'alimentation de la résistance à 5V pour avoir un état logique haut en cas de blocage du transistor.
Le jaune, l'alimentation de chaque diode IR couplé à sa photorésistance.
Le noir, la masse.

Nous avons effectué un test avec la boîte.

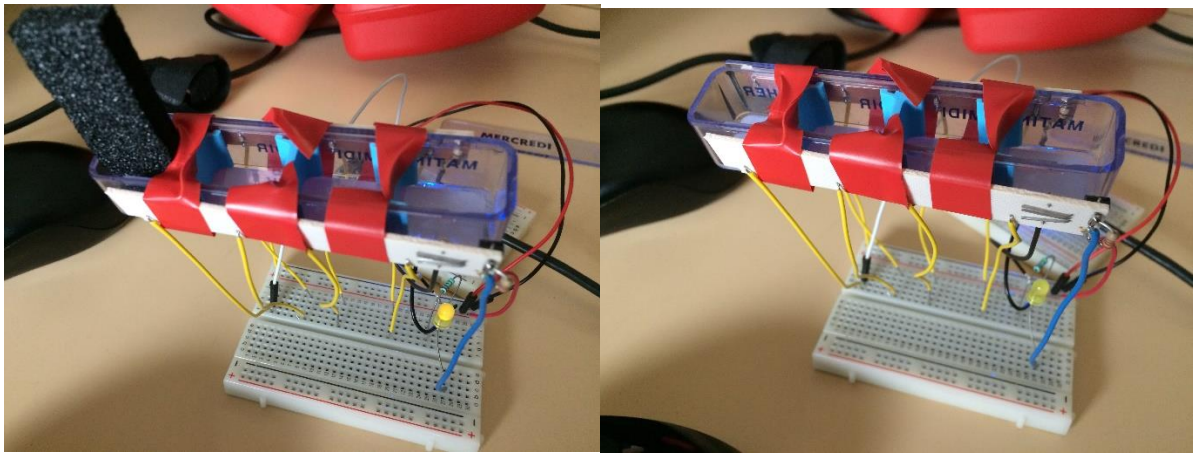


Figure 14. Test du faisceau IR avec un boîter

3. Présentation de la partie Software

Ce chapitre a pour but de donner un aperçu du code développé et d'expliquer les choix techniques pour réaliser notre pilulier connecté.
Il se décompose en deux parties. D'une part, nous argumenterons la partie logicielle qui compose la base à ajouter au pilulier. D'autre part, nous présenterons le code développé qui a servi à mettre au point l'application Android.

3.1. Le socle du pilulier

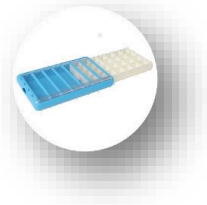
L'algorithme du programme principal se déroule de manière itérative. Selon l'heure et la date obtenu en temps réel, le programme va exécuter des opérations sur le système embarqué du pilulier correspondant à la période de ma journée (compartiment du matin, du midi, du soir ou de la nuit). L'algorithme s'effectue itérativement de la manière suivante :

1. *Lecture du jour de la semaine*
2. *Si le jour de la semaine a changé, alors :*
    1. *Lecture des états de chaque compartiment*
3. *Si le jour de la semaine n'a pas changé, alors :*
    a. *Lecture de l'heure de la journée*
    b. *Si l'heure de la journée est comprise entre 8 et 10 heures, alors on est au Matin*
        i. *Lecture de l'état du compartiment Matin*
        ii. *Mise en marche de la LED du compartiment Matin*
        iii. *Activation de l'alarme de l'application Android*
    c. *Si l'heure de la journée est comprise entre 12 et 14 heures, alors on est au Midi*
        i. *Lecture de l'état du compartiment Midi*
        ii. *Vérification si le contenu dans le compartiment Matin est vide*
        iii. *Mise en marche de la LED du compartiment Midi*
        iv. *Activation de l'alarme de l'application Android*
    d. *Si l'heure de la journée est comprise entre 18 et 20 heures, alors on est au Soir*
        i. *Lecture de l'état du compartiment Soir*
        ii. *Vérification si le contenu dans le compartiment Matin et Midi est vide*
        iii. *Mise en marche de la LED du compartiment Soir*
        iv. *Activation de l'alarme de l'application Android*
    e. *Si l'heure de la journée est comprise entre 21 et 23 heures, alors on est au Nuit*
        i. *Lecture de l'état du compartiment Nuit*
        ii. *Vérification si le contenu dans le compartiment Matin, Midi et Soir est vide*
        iii. *Mise en marche de la LED du compartiment Nuit*
        iv. *Activation de l'alarme de l'application Android*
4. *Si un des états de l'un des compartiments du pilulier hebdomadaire a changé, alors :*
    a. *Envoi des nouveaux états à l'application Android*

3.2. Application Android

Description de l'application Android

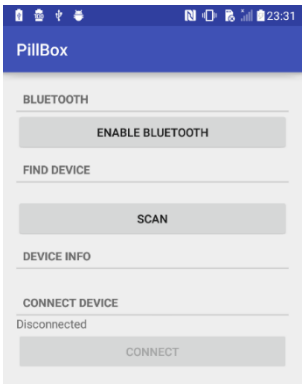
L'application Android est complémentaire de la partie Arduino. Il est possible à l'aide de cette application d'établir un suivi permanent du pilulier connecté. La communication entre l'application et le capteur s'effectue à travers le protocole Bluetooth Low Energy.

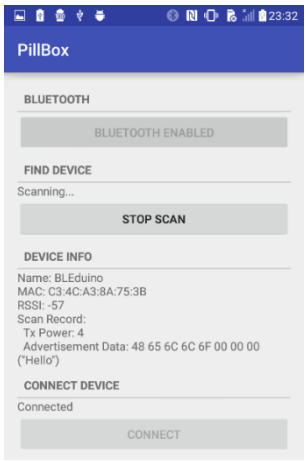


L'application se compose d'une seule interface qui peut exécuter plusieurs activités en arrière-plan.
L'interface se décompose en 4 parties, chacune ayant une fonction précise.

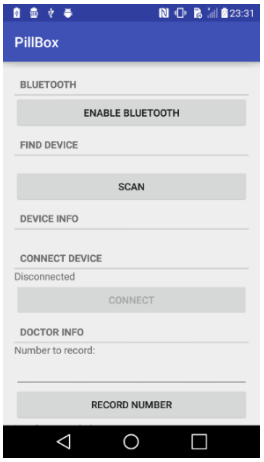
La première partie correspond à la communication Bluetooth. On y aperçu trois boutons qui se succèdent dont chacun correspond à une étape de la mise en communication avec le RFduino :

- *Enable Bluetooth* permet de mettre en marche le BLE du téléphone si celui-ci est éteint.
- *Scan* permet de détecter le RFduino.
- *Connect* met en connexion et récupère les informations de la balise BLE.





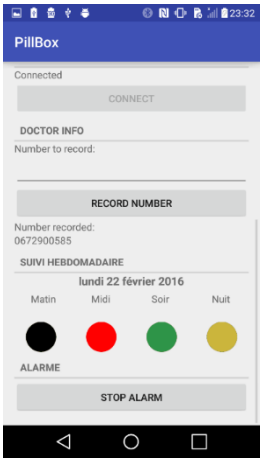
La seconde partie a pour rôle d'enregistrer le numéro du médecin ou du responsable de l'utilisateur. L'enregistrement du numéro est nécessaire à l'envoi d'un SMS si le patient n'a pas pris son médicament dans un temps spécifié.



Nous arrivons ensuite à la troisième partie qui renseigne le suivi hebdomadaire du patient. On retrouve les 4 zones de journées comme l'est indiqué sur les piluliers standards, à savoir le matin, le midi, le soir et la nuit ; ainsi que la date en temps réel.
Pour chaque période de la journée, on indique l'état du compartiment du pilulier correspondant à la période. Cet état est représenté par un point pouvant avoir 4 couleurs différentes selon le contexte :

- Si un point jaune apparait, le médicament à ingurgiter actuellement est dans le compartiment correspondant à ce moment de la journée.
- Si un point vert apparait, le médicament présent dans ce compartiment devra être ingurgité plus tard dans la journée.
- Si un point rouge apparait, le médicament dans ce compartiment a oublié d'être ingurgité plus tôt dans la journée.
- Si un point noir apparaît, il n'y a pas de médicament à prendre dans ce compartiment ou il a déjà été pris.

Enfin dans la dernière partie, on retrouve un bouton *Stop Alarm* qui désactive l'alarme. Celui-ci se déclenche lorsqu'on passe à une nouvelle période de la journée. Plus précisément, lorsqu'on passe du matin au midi.
Si l'utilisateur n'a pas désactiver lui-même l'alarme, celui-ci s'arrête automatiquement au bout d'une période spécifié et entraîne l'envoi d'un SMS.



A savoir que l'interface intègre un conteneur *ScrollView* qui permet le défilement du contenu et donc l'accès à l'ensemble de différentes parties de l'application.

Explication du firmware de l'application Android

Le développement de l'application Android a été long mais instructif. En effet, nous n'avons jamais développé d'application Android et codé en Java auparavant. Nous nous sommes basé alors sur les nombreux tutoriels accessibles en ligne pour réaliser cette application et acquérir quelques bases en Java et développement Android.

L'application a été développée en Java sous le logiciel Android Studio. Son développement s'est déroulé en plusieurs étapes :

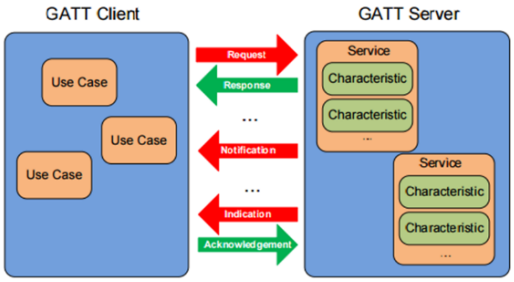
1. Mise en place de la communication Bluetooth Low Energy entre le smartphone et le RFduino (module BLE du socle).
2. Développement de l'interface de suivi hebdomadaire
3. Programmation de l'alarme et de l'envoi du SMS d'alerte

Communication Bluetooth Low Energy

Afin de mettre en place de la communication entre le smartphone et le pilulier, nous nous sommes basé sur le code opensource disponible sur le site officiel *GitHub*.

Le Bluetooth Low Energy est un protocole de communication rapide qui demande peu d'énergie pour son fonctionnement. C'est le système de communication recommandé et primordiale pour les objets connectés du fait de la nécessité d'avoir des composants à faible consommation énergétique.
La communication est construite selon un profil définit la manière dont les périphériques transmettent leurs données aux appareils (*Generic Attribute Profile)* en utilisant des concepts appelés *Services* et *Characteristics*.

Les *Services* peuvent contenir une collection de *Characteristics*. Les *Characteristics* sont des flags de propriété ou les valeurs de capteurs. Chaque *Service* et *Characteristic* a son identifiant (*UUID*) propre.
Le profil GATT propose seulement des connections exclusives ; c'est-à-dire que les périphériques BLE ne peuvent être connecté qu'à un seul smartphone à la fois.



Ici, nous nous retrouvons avec :

- *Generic Attribute Profile Client* pour l'application.
- *Generic Attribute Profile Server* pour le pilulier connecté (RFduino) avec un service propre et plusieurs caractéristiques (communication I2C, UART...).

Suivi Hebdomadaire

L'application Android reçoit les données à travers l'objet *BroadcastReceiver* qui par sa fonction *onReceive* va recevoir les *events* que reçoit l'application dans son *Activity* principal. Lorsqu'un *event* spécifique, par exemple dans notre cas lorsque une donnée est disponible par Bluetooth, le code correspondant du *BroadcastReceiver* s'exécute dès que possible.

Pour obtenir un suivi hebdomadaire en temps réel, nous avons utilisé la fonction *runOnUiThread* qui exécute l'action spécifiée sur le *Thread* d'interface utilisateur de manière immédiate étant donné que dans notre situation le *Thread* courant correspond au *Thread* d'interface utilisateur.

L'appel de cette fonction, une fois les données du RFduino reçu et récupéré dans le *BroadcastReceiver*, permet d'identifier les quatre périodes de la journée (matin, midi, soir et nuit) et d'adapter automatiquement de manière visuel (jaune, vert, rouge, noir) selon les données reçus (respectivement : 1, 2, 3, 4).

Alarme et envoi du SMS d'alerte

L'alarme est déclenchée suite à la réception, via la communication Bluetooth, de la commande « alarme ».
Afin de mettre en place l'alarme, nous avons exploité les services d'alarme du système Android, *AlarmManager class*, qui permet de planifier une activité à être exécuté à un moment donné dans l'avenir. Pour cela, nous avons planifié deux alarmes :

1. Une première alarme qui se répète toutes les 3 secondes.
2. Une deuxième alarme qui permet d'arrêter la première alarme au bout d'une heure.

Pour que l'alarme soit toujours exécutée en *background* de l'application et même à sa fermeture, nous avons utilisé le système d'évènement, *BroadcastReceiver object*, combiné à la création d'un nouveau *Thread* ou *Tâche* dans le service sur lequel dirige le *BroadcastReceiver*. Ce *Thread* utilise les fonctionnalités de *MediaPlayer*, qui permet le contrôle de la lecture de flux audio.

De plus, au bout de l'activation de la seconde alarme, si la première alarme n'est pas déjà désactivée, on envoie un SMS d'alerte au numéro enregistré au préalable, à partir du service SmsManager d'Android. Ce service permet de faire appel à l'application intégrée à Android « Message », et d'envoyer un SMS.

Enfin, si l'utilisateur appui sur le bouton *Stop Alarm*, la fonction appelé va d'abord arrêter la future planification de la prochaine sonnerie de l'alarme, puis annuler l'appel de l'*event BroadcastReceiver*.

4. Bilan

4.1. Les difficultés rencontrées

Au cours de ce projet, nous avons rencontré un certain nombre de difficultés à la réalisation du socle, notamment au niveau du software. Mais aussi, nous avons dû faire face à notre absence de connaissance et de base concrète au développement d'une application Android et à la programmation Java.

Du côté du socle

Lors de l'établissement de la liste du matériel nécessaire, nous avons sélectionné à la fois un RFduino et un Arduino Micro. En effet, le premier permet la liaison du pilulier connecté à l'application Android. Le second, de par la nécessité d'accès à de nombreux *pins* pour le système embarqué, devait jouer le rôle de microcontrôleur principal.
Or, pour n'importe quel protocole de communication utilisé par le RFduino (SPI, I2C), le *hardware* de celui-ci ne peut pas prendre en charge de fonctionner en tant qu'esclave. L'algorithme réalisé, en vue des tests finaux, ne pouvait donc fonctionner et a dû être remodelé.

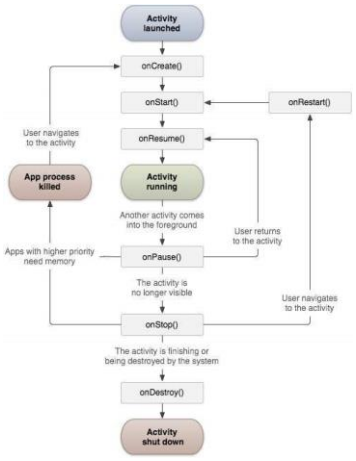
De plus, nous avons remarqué des erreurs de lecture du temps ou de date, générées par le MOD-RTC, sont assez régulières malgré le fait que le MOD-RTC soit bien initialisé.

Du côté de l'application Android

Le développement de l'application Android a été mal planifié et exécuté. En effet, nous avons réparti le développement en deux parties :

1. Programmer et tester jusqu'à validation les différentes fonctionnalités que doit composer l'application (calendrier, suivi hebdomadaire, gestion de la mise en marche de l'alarme et du SMS, communication Bluetooth…).
2. Rassembler ces différentes activités sur une même application et les rendre accessible via un menu latéral.

Comme dit précédemment, certaines de ces fonctionnalités (calendrier, stockage de données, menu latéral…) ont été développées à l'aide de tutoriel et module de code disponible sur le Web puis adaptées à notre utilisation.
Cependant, l'*Activity* principal censé gérer les différentes fonctionnalités, via un menu latéral, pouvait accueillir seulement des *Fragments* et non des *Activity* comme l'oblige les différents modules. Un *Fragment* représente un comportement ou une partie de l'interface dans une *Activity* tandis qu'une *Activity* représente un seul écran avec une interface où le système Android lance son programme avec une activité particulière et créée à partir d'un appel *onCreate*() et autre callback (*onResume*(), *onStart*()…).



Nous avons donc dû revoir notre copie, et développer une application Android plus standard mais fonctionnel et proposant une gestion du cycle de vie et une mise en arrière-plan des activités plus stable et performante.

4.2. Les développements à apporter

Après ce bilan, il est évidant qu'il y a de nombreux points à développer et améliorer. Tout d'abord, il faudrait déployer l'électronique sur toutes les boîtes. Pour gérer l'envoie du message UART, il sera possible d'utiliser un démultiplexeur.

Il serait aussi possible de faire un prototype « fini » c'est-à-dire avec un petit boîtier préalablement réalisé avec la découpe laser du FABLAB.

L'ajout de caractéristiques et fonctionnalités, pas forcément nécessaires, pourrait pousser le projet vers un prototype complet. On peut notamment citer l'ajout d'une alarme sonore, de LED pour indiquer la boîte et prendre mais également la présence dans le support, un écran LCD qui indique la date et l'heure, une batterie pour rendre le prototype.

Conclusion

Ce projet aura permis d'exploiter et d'approfondir nos connaissances dans les domaines de la conception électronique et du développement d'application microcontrôleur. Nous avons eu l'occasion d'utiliser le JAVA que nous n'avons pas appris en SA ce qui a orienté le projet vers un côté SC.

D'autre part, nous nous sommes à nouveau confrontés à la difficulté de la gestion de projet. En effet, revenant d'un séjour à l'étranger, nous avons dû tout mettre en œuvre pour essayer d'aboutir à un résultat dont on ne peut toujours pas juger l'allure.

Les tests des caractéristiques et fonctionnalités ont pu être effectués et validés dès le début du projet mais la suite a été bloqué par la commande des composants que nous n'avons reçu qu'hier dans l'après-midi.

A l'heure d'aujourd'hui, les cartes électroniques sont prêtes, mais nous sommes plus préoccupés à la rédaction du rapport et la résolution du problème de communication entre l'Arduino micro et le RFduino auquel nous nous sommes heurtés, que de donner l'allure que nous désirions au prototype.

Annexes

Annexe 1. Code de l'ATtiny13

```
#define F_CPU 10000000
  #include <avr/io.h>
  #include <util/delay.h>
  int main (void){
     DDRB |= 0x0F;
     int i, code=0;
     while(1){
        while((PINB & 0x10) == 0x10){}
           _delay_us(1000);
           for(i=128 ; i>=1 ; i=i/2){
              code = (((PINB & 0x10)/16) * i) + code;
              _delay_us(2000);
           }
           if((code & 0xF0) == 0x50){
              PORTB |= (code - 0x50);
           }
           else {
              code = 0;}}}
```

Annexe 2. Code de l'Arduino micro pour le faisceau IR

```
1. const int IRDetectorPin = 2;
2. const int ledPin =  13;
3.
4. void setup() {
5.   pinMode(ledPin, OUTPUT);
6.   pinMode(IRDetectorPin, INPUT);
7. }
8.
9. void loop(){
10.          digitalWrite(ledPin, digitalRead(IRDetectorPin));
11.      }
```

Annexe 3. Code de l'arduino master

```
////////////////////////////
// Utilisation des librairies
#include <Arduino.h>                // Librairie Arduino
#include <Wire.h>                   // Librairie I2C (communication avec l'horloge temps réel et BLE)
#include <RTC.h>                    // Librairie RTC (commande de l'horloge temps réel)
////////////////////////////

//////////////////////////////////////////////////////////////////////
// Définition et initialisation des variables des pins des capteurs et leds
const int ir1 = 5;                  // Capteur matin
const int ir2 = 6;                  // Capteur midi
const int ir3 =  9;                 // Capteur soir
const int ir4 =  10;                // Capteur nuit
const int alim =  11;               // Alimentation capteur
const int signalCapteur =  14;      // Signal IR
const int UARTpin =  15;            // Commande led
const int VCCpin =  13;             // Alimentation led
//////////////////////////////////////////////////////////////////////

////////////////////////////////////
// Définition des variables du pilulier
String today = "";
String yesterday = "";
const char* weekdayname[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday","Sunday"};
int day = 0;
int momentDay = 0;
int matin,midi,soir,nuit = 4;
int oldMatin,oldMidi,oldSoir,oldNuit = 4;
boolean AlarmSended=false;
////////////////////////////////////

////////////////////////////
// Programme d'initialisation
void setup() {

  // Initialisation de la libraire Wire pour la communication I2C
  Wire.begin();

  // Initialisation du port série pour la communcation ordinateur-arduino
  Serial.begin(9600);

  // Délai
  delay(100);

  // Initialisation du RTC
  Serial.println("Initialisation du RTC...");
  //initRTC();
  Serial.println("Initialisation du RTC : OK");

  // Délai
  delay(100);

  // Initialisation des E/S des capteurs et led
  pinMode(ir1, OUTPUT);
  pinMode(ir2, OUTPUT);
  pinMode(ir3, OUTPUT);
  pinMode(ir4, OUTPUT);
  pinMode(alim, OUTPUT);
  pinMode(signalCapteur, INPUT);
  pinMode(UARTpin, OUTPUT);
  pinMode(VCCpin, OUTPUT);
}
////////////////////////////
```

```
//////////////////////////////////////////////////////////////////
// Fonction qui retourne l'état du compartiment du moment de ma journée
int timeTakeDrug(int state, int timeDay){
  boolean sDay;

  if(state==2){

    // Récupération de l'état du compartiment selon le capteur
    sDay = stateDay(timeDay);

    // Si state est vrai, alors il n'y a plus de médicament
    if(sDay){
      return 4;
    }
    else {
     return 1;
    }
  }
  return 4;
}
//////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////
// Fonction qui retourne l'état d'un des compartiments si vide
int timeLostDrug(int state){

    // Si state était à prendre, donc on a pas pris le médicament au bont moment
    if(state==1){
    return 3;
  }
  return 4;
}
//////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////
// Fonction qui lit l'état des capteurs pour un compartiment donné
bool stateDay(int timeDay){
  digitalWrite(alim, HIGH);  //alim pin random à dénifir
  if(timeDay==1){
    // Matin
    digitalWrite(ir1, HIGH);
    if(digitalRead(signalCapteur) == 0){
      digitalWrite(alim, LOW);
      digitalWrite(ir1, LOW);
      return true;} // medicament a été pris
    else{
      digitalWrite(alim, LOW);
      digitalWrite(ir1, LOW);
      return false;
    }
  }
  else if(timeDay==2){
    // Midi
    digitalWrite(ir2, HIGH);
    if(digitalRead(signalCapteur) == 0){
      digitalWrite(alim, LOW);
      digitalWrite(ir2, LOW);
      return true;} // medicament a été pris
    else{
      digitalWrite(alim, LOW);
```

```
      digitalWrite(ir2, LOW);
    return false;
    }
   }
   else if(timeDay==3){
    // Soir
    digitalWrite(ir3, HIGH);
    if(digitalRead(signalCapteur) == 0){
      digitalWrite(alim, LOW);
      digitalWrite(ir3, LOW);
      return true;} // medicament a été pris
    else{
      digitalWrite(alim, LOW);
      digitalWrite(ir3, LOW);
    return false;
    }
   }
   else if(timeDay==4){
    // Nuit
    digitalWrite(ir4, HIGH);
    if(digitalRead(signalCapteur) == 0){
      digitalWrite(alim, LOW);
      digitalWrite(ir4, LOW);
      return true;} // medicament a été pris
    else{
      digitalWrite(alim, LOW);
      digitalWrite(ir4, LOW);
    return false;
    }
    return false;
   }
}
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////
// Fonction qui allume la led du compartiment approprié
void onLED (int timeDay){
  digitalWrite(UARTpin, HIGH); // définir la pin qui fera la liaison série
  digitalWrite(VCCpin, HIGH);  // définir la pin qui alimentera le uc attiny
  delayMicroseconds(2000);

  digitalWrite(UARTpin, LOW);
  delayMicroseconds(2000);
  digitalWrite(UARTpin, HIGH);
  delayMicroseconds(2000);
  digitalWrite(UARTpin, LOW);
  delayMicroseconds(2000);
  digitalWrite(UARTpin, HIGH);
  delayMicroseconds(2000);  // init de la liaison UART

  if(timeDay == 1){
    //matin
    delayMicroseconds(2000);
    digitalWrite(UARTpin, LOW);
    delayMicroseconds(6000);
  }
  else if(timeDay == 2){
    //midi
    digitalWrite(UARTpin, LOW);
    delayMicroseconds(2000);
    digitalWrite(UARTpin, HIGH);
    delayMicroseconds(2000);
    digitalWrite(UARTpin, LOW);
```

```
    delayMicroseconds(4000);
  }
  else if(timeDay == 3){
    //soir
    digitalWrite(UARTpin, LOW);
    delayMicroseconds(4000);
    digitalWrite(UARTpin, HIGH);
    delayMicroseconds(2000);
    digitalWrite(UARTpin, LOW);
    delayMicroseconds(2000);
  }
  else if(timeDay == 4){
    //coucher
    digitalWrite(UARTpin, LOW);
    delayMicroseconds(6000);
    digitalWrite(UARTpin, HIGH);
    delayMicroseconds(2000);
  }
  else {}

  digitalWrite(UARTpin, HIGH);

  // après l'ajout du super condensateur, on peut couper Vccpin quand le CS est chargé
}
//////////////////////////////////////////////////////


//////////////////////////////////////////////
// Fonction qui envoie des données au RFduino
void I2CSend (int id){
  Wire.beginTransmission(6);    // Send to I2C slave device 6

  if (id==2){
    Wire.write(id);
    Wire.write(matin);
    Wire.write(midi);
    Wire.write(soir);
    Wire.write(nuit);
  }
  else if(id==3){
    Wire.write(1);
  }
  else {}

  Wire.endTransmission();

  delay(100);
}
//////////////////////////////////////////////

//////////////////////////////////////////////
// Fonction qui recoit des données au RFduino
void I2CRequest (){
  Wire.requestFrom(6,4);              // Request 4 bytes from I2C slave device 6

  byte data1, data2, data3, data4;

  while ( Wire.available( ) )           // Slave may send less than requested
  {
    data1 = Wire.read( );           // Receive each byte
    data2 = Wire.read( );
    data3 = Wire.read( );
    data4 = Wire.read( );
  }
```

```
  Serial.println(data1);
  Serial.println(data2);
  Serial.println(data3);
  Serial.println(data4);
}
////////////////////////////////////////////

//////////////////////
// Programme principal
void loop() {

  // Récupération du jour de la semaine
  //today = getDay();
  today = weekdayname[day];

  // Mise à jour des états du compartiment à chaque jour
  if (today!=yesterday){

    // Lecture des états des compartiment du pilulier pour le nouveau jour
    matin = 2;
    midi = 2;
    soir = 4;
    nuit = 2;

    // Envoi des états du compartiment du pilulier
    I2CSend(2);

    // Mise à jour du jour d'avant
    yesterday = today;
  }
  else{

    // Récupération de l'heure de la journée
    //momentDay = getTime(3);

    // Si l'heure est compris entre 8 et 10 heure, alors on est dans le compartiment matin
    if(8<=momentDay<=10){
      // Récupération de l'état du compartiment par les capteurs
      matin = timeTakeDrug(matin,1);
      // Mise en marche de la LED du compartiment approprié
      onLED(1);
      // Envoi de la commande alarme au RFduino pour transmission Bluetooth
      if (!AlarmSended){
        I2CSend(3);
        AlarmSended = true;
      }
    }

       // Si l'heure est compris entre 12 et 14 heure, alors on est dans le compartiment midi
    else if(12<=momentDay<=14){
      // Mise à jour de l'état si oublie de prendre le médicament
      matin = timeLostDrug(matin);
      // Récupération de l'état du compartiment par les capteurs
      midi = timeTakeDrug(midi,2);
      // Mise en marche de la LED du compartiment approprié
      onLED(2);
      // Envoi de la commande alarme au RFduino pour transmission Bluetooth
      if (!AlarmSended){
        I2CSend(3);
        AlarmSended = true;
      }
    }
    // Si l'heure est compris entre 18 et 20 heure, alors on est dans le compartiment soir
    else if(18<=momentDay<=20){
```

```
      // Mise à jour de l'état si oublie de prendre le médicament
      matin = timeLostDrug(matin);
      midi = timeLostDrug(midi);
      // Récupération de l'état du compartiment par les capteurs
      soir = timeTakeDrug(soir,3);
      // Mise en marche de la LED du compartiment approprié
      onLED(3);
      // Envoi de la commande alarme au RFduino pour transmission Bluetooth
      if (!AlarmSended){
        I2CSend(3);
        AlarmSended = true;
      }
    }
    // Si l'heure est compris entre 21 et 23 heure, alors on est dans le compartiment nuit
    else if(21<=momentDay<=23){
      // Mise à jour de l'état si oublie de prendre le médicament
      matin = timeLostDrug(matin);
      midi = timeLostDrug(midi);
      soir = timeLostDrug(soir);
      // Récupération de l'état du compartiment par les capteurs
      nuit = timeTakeDrug(nuit,4);
      // Mise en marche de la LED du compartiment approprié
      onLED(4);
      // Envoi de la commande alarme au RFduino pour transmission Bluetooth
      if (!AlarmSended){
        I2CSend(3);
        AlarmSended = true;
      }
    }
    else {}
  }

   // Si changement d'état d'un des compartiments
   if ( (matin!=oldMatin)||(midi!=oldMidi)||(soir!=oldSoir)||(nuit!=oldNuit) )
    // Envoi des nouveaux états au RFduino pour transmission Bluetooth
    I2CSend(2);
    // Mise à jour des anciens états
    oldMatin = matin;
    oldMidi = midi;
    oldSoir = soir;
    oldNuit = nuit;
    // Activation de l'alarme
    AlarmSended = false;
  }

  // Incrémentation pour tester pour une semaine
  momentDay = momentDay + 2;
  if(momentDay==24){
    momentDay = 0;
    day = day + 1;
    if(day==6){
      day = 0;
    }
  }

  // Délai
  delay(20000)
}
//////////////////////
```

Annexe 4. Code BLE avec RFduino slave

```
////////////////////////////
// Utilisation des librairies
#include <Arduino.h>                // Librairie Arduino
#include <Wire.h>                   // Librairie I2C (communication avec l'arduino)
#include <BLE.h>                    // Librairie BLE (communication bluetooth / reception de commande et traitement)
////////////////////////////

////////////////////////////////////
// Définition des variables du RFduino
int matin,midi,soir,nuit = 4;
////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////////////////
// Fonction qui s'exécute comme un évènement dès qu'il y a des données transmi par l'arduino micro
void receiveEvent(int byteCount)
{
  // Initialisation des paramètres
  byte id = 0;    // Identifiant des données transmises
  int data = 0;   // Données transmises

  // Tant qu'il y a des données sur le bus
  while( 1 < Wire.available( ) )
  {
    // Lecture de l'identifiant
    id = Wire.read( );

    // Si l'identifiant correspond à la mise jour des états des compartiments
    if(id == 2)
    {
      // Lecture des données venant du Arduino (maître)
      matin = Wire.read( );
      midi = Wire.read( );
      soir = Wire.read( );
      nuit = Wire.read( );

      // Envoi des nouveaux états à l'application Android
      sendEtatDaily(matin,midi,soir,nuit);

      Serial.println(data);
    }
    // Si l'identifiant correspond à la mise en marche de l'alarme
    else if (id==3) {
      sendAlarmeBLE();
    }
    // Sinon, on ne fait rien
    else {}
  }
}
//////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////
// Programme d'initialisation
void setup() {

  // Initialisation de la communication I2C
  Wire.beginOnPins(5,6);

  // Délai
  delay(100);

  // Réception d'un évènement
```

```
  Wire.onReceive(receiveEvent);

  // Délai
  delay(100);

  // Initialisation de la communication série
  Serial.begin(9600);

  // Délai
  delay(100);

  // Initialisation du BLE
  Serial.println("Initialisation du BLE...");
  initBLE();
  Serial.println("Initialisation du BLE : OK");

  // Délai
  delay(100);
}
//////////////////////////////

//////////////////////
// Programme principal
void loop() {

  // Délai
  delay(100);
}
//////////////////////
```