3 Remerciement

Pour commencer j'aimerais remercier toute la team Geneva pour toute la bonne humeur et surtout le boulot que vous avez fourni. Encore plus remercier mon camarade Axel.

Il faut aussi remercier toute la team CHIC qui d'occupe de toute l'organisation des différents groupes.

Je remercie aussi bien mon professeur responsable, René Beuchat, pour le suivi sans faille du projet.

Un grand merci au personnel du LSN pour toute les compétences apprises pendant ces trois années du diplôme.

Bien sur un gros merci à mes camarades de classes, qui étaient toujours disponible pour aider… ou faire des pauses.

Je remercie aussi ma famille pour le soutient à travers cette période… chargée. Et qui se sont occupée de la relecture.

Per aspera ad astra…

Adrien

4 Table des matières

5 Table des figures

Table des figures

6 Introduction

Ce projet de Bachelor a été réalisé en collaboration avec l'institution CHIC dont l'objectif est de favoriser les projets d'innovation. Notre équipe "Team Genève" s'est rencontrée pendant un week-end de novembre 2016 à l'EPFL pour lancer le projet (kick off). Après une première phase de recherche d'idées qui a duré environ 2 mois, notre équipe a poursuivi avec la phase de conception durant 3 mois et lancé la réalisation, phase dans laquelle nous sommes encore actuellement.

6.1 CHIC

CHIC (China Hardware Innovation Camp) est une fondation de la Confédération helvétique qui vise à promouvoir l'ingénierie suisse dans le monde, dans notre cas la Chine. Pour de plus amples informations, vous pouvez consulter le site Internet[i].

6.2 Team Geneva

Afin de développer des solutions répondant à l'objectif fixé par CHIC : créer un objet connecté et innovateur, 8 équipes se sont formées selon la région (Lausanne, Fribourg, Valais, Tessin, Genève). Chaque équipe est composée d'étudiants de différentes orientations (business, design, ingénierie).

Dans le "Team Geneva", nous sommes cinq étudiants de différentes écoles : deux de l'hepia en ingénierie des technologies de l'information en orientation matérielle, une en international business management à l'HEG et deux en design à la HEAD.

Figure 1: Les cinq membres de la team Geneva

Je vais vous présenter rapidement l'équipe :

Tabea, étudiante de l'HEG en International Business Management. C'est elle qui va s'occuper de la partie économique du projet. Elle est en charge du Business Plan, de définir le financement du projet.

Julia, étudiante à la HEAD en User Interactions. Elle va s'occuper de définir les interactions avec l'objet et aussi de l'interface graphique de l'application smartphone.

Loic, étudiant à la HEAD en Product Design. Il va s'occuper de la partie mécanique du projet, du design du boîtier et de l'ergonomie.

Axel, mon collègue de l'hepia, et moi-même nous sommes chargés de la partie ingénierie. Axel s'occupe de la partie RFID et de la majeure partie du PCB (circuit imprimé) et moi du profil Bluetooth pour la connexion avec le Smartphone, l'alimentation du PCB et du backend de l'application Smartphone

6.2.1 Objectifs

Les objectifs principaux du projet ont été fixés pour répondre à la demande de CHIC qui est : développer un objet connecté innovant.

1. L'utilité : l'objet doit être utile au quotidien pour un public lambda
2. L'originalité : l'objet offre des fonctionnalités pas encore disponibles sur le marché
3. La faisabilité : l'objet peut être produit en intégrant les technologies disponibles à ce jour
4. La fiabilité : l'objet consomme peu d'énergie, est autonome et fiable

6.2.2 Réalisation de la partie technique

J'ai réalisé la partie technique du projet ShuQi conjointement avec Axel Collet. Je vous invite à lire son mémoire pour avoir de plus amples informations sur les parties suivantes :

- RFID
- PCB

Le schéma d'alimentation que j'ai développé a été intégré par Axel dans le PCB.

6.3 Blog

On possède aussi un blog que l'on mettra à jour d'ici mi-août 2017. Vous pouvez y accéder depuis :

---

http://chi.camp/projects/team-geneva/

---

7 ShuQi

Après avoir évalué le potentiel des différentes idées, la majorité d'entre nous et des professeurs qui nous ont suivi ont voté pour la solution ShuQi.

Ce petit appareil va permettre de régler un problème de tous les jours : l'oubli.

Il nous arrive bien souvent de laisser un objet qu'on devait prendre avec nous et de l'oublier. Il a fallu trouver une astuce pour qu'un objet nous aide à ne rien oublier.

La solution trouvée est une centrale autonome, que l'on va glisser dans nos sacs, qui va scanner les affaires sur lesquelles un tag sera apposé.

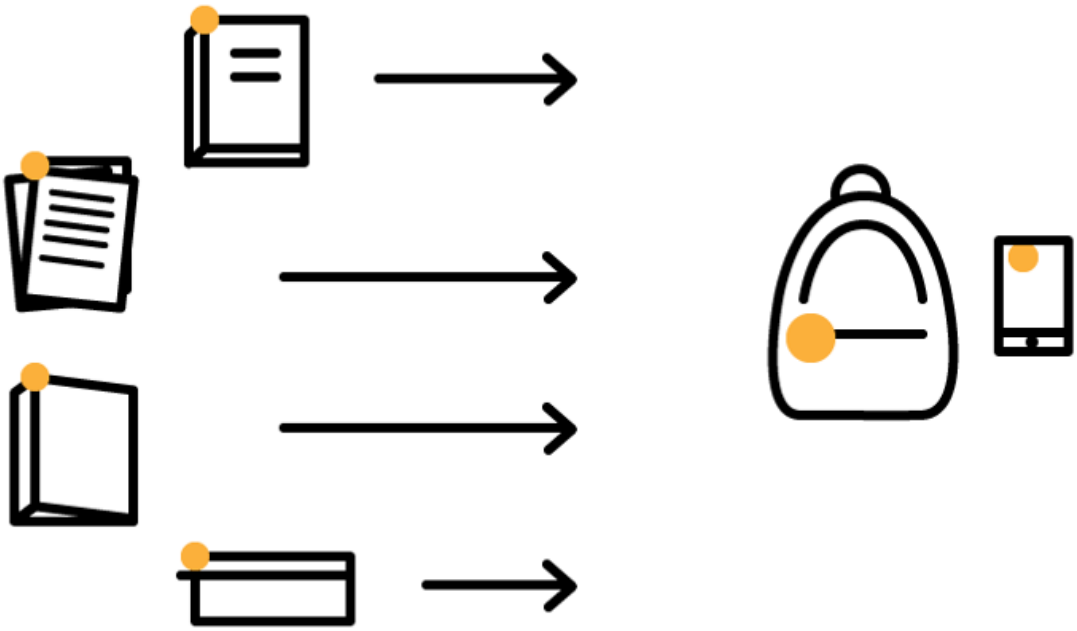


Figure 2: Schéma du concept de ShuQi

Ce qui donne concrètement, un système comportant une centrale et de tags à positionner les affaires. Ces tags seront présentés sous forme d'autocollants à apposer sur une surface plane.

Cette centrale sera connectée à un smartphone à travers une interface Bluetooth qui permet de gérer nos affaires. Les actions définies sont :

- Scan des affaires à proximité (dans un périmètre de env. 50 cm)
- Renommage des tags
- Création de liste d'affaires liées à un agenda
- etc...

Le résultat est une petite boite de 12x6x3 cm contenant tous les composants nécessaires.

Figure 3: Exterieur du ShuQi

A l'intérieur du boitier se trouve le PCB, l'antenne RFID et la batterie.

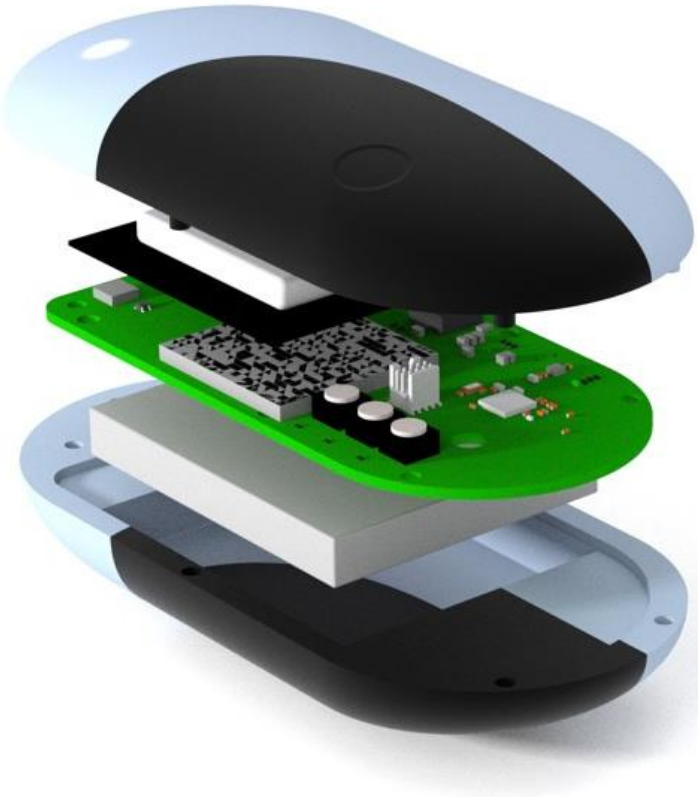


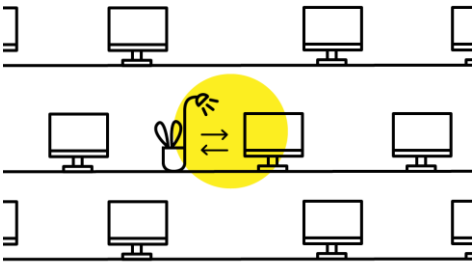
Figure 4: Vue éclaté du ShuQi

7.1 Historique des idées

Avant d'avoir notre idée, nous sommes passés par plusieurs autres idées. Ces trois idées ont été soumises à nos coordinateurs et entre nous tous, les coordinateurs et l'équipe, nous avons tous voté pour le projet qui nous intéressait le plus, notre smartbag. Mais voici les autres projets :

7.1.1 Smart garden

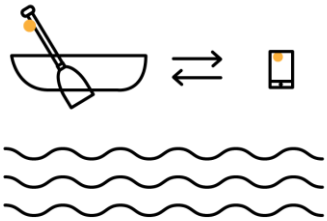
Ceci était notre première idée. Nous avions commencé à penser à faire un jardin connecté, mais ce produit existant déjà, nous l'avons modifié.



L'itération suivante était d'avoir un pot connecté équipé de différents capteurs, dont un qui mesurait la qualité de l'air. Le but était d'avertir l'utilisateur quand son air n'avait plus assez d'oxygène.

Le problème de cette idée était que nous les ingénieurs n'avions vraiment pas de challenge.

7.1.2 Paddle



Une autre idée venant cette fois ci d'Axel était de rendre un paddle connecté. La cible principale étant les clubs de paddle pour pouvoir avoir un suivi continu de leurs clients quand ils sont en sortie. Sachant qu'il y a un dispositif se fixant à la cheville pour éviter de perdre son paddle en tombant, c'était notre cible : un bracelet de jambe connecté.

Ses fonctions étaient de localiser le sportif, lui envoyer une alerte s'il sortait d'une zone prédéfinie, de recevoir des alertes météo en temps réel pour lui dire de rentrer et possiblement de faire une fonction SOS.

Le problème de cette idée étant que ce n'était pas novateur et que les designers ne se retrouvaient pas dedans.

7.1.3 Traqueur de ski

Cette dernière idée étant de moi, cela aurait été de créer un traqueur de ski pour pouvoir enregistrer toute une journée de ski. Ce projet aurait impliqué beaucoup de traitement de signal.

Par contre ce n'était vraiment pas une idée innovante, il y a déjà plusieurs traqueurs pour le ski existant, sur ce constat nous avons abandonné cette idée.

8 Ingénierie

Dans notre partie d'ingénierie, partagée entre Axel et moi, on a du faire un système autonome complet avec la création d'un PCB, le choix des composants, l'application de la centrale et finalement le backend de l'application smartphone.

Ma partie est composée de trois parties principales, dont une partie moins importante que les autres.

- Alimentation
- Bluetooth
- (Application smartphone)

L'application smartphone n'est pas la plus importante car si elle ne fonctionne pas, d'autres alternatives sont envisageables.

L'essentiel est composé de deux technologies principales :

- Bluetooth Low Energy
- RFID UHF

Le Bluetooth qui est utilisé pour la communication entre la centrale et le smartphone et le RFID qui lui est utilisé pour la lectures des tags sur les affaires.

8.1 Outils de développement

- **Altium Designer 16.1 :** Outil de conception assisté par ordinateur pour la création de PCB
- **nRF52 SDK 12.3.0 :** Suite de librairies logicielles permettant d'interagir avec le nRF52.
- **GCC :** Compilateur C avec son débuggeur GDB.
- **Make :** Script de compilation utilisé dans le SDK.
- **Eclipse Mars :** Environnement de développement intégré permettant la compilation, l'exécution, le flashage et le débogage avec le fichier Makefile.
- **Bluetooth Developper Studio :** Utilitaire permettant de créer ou modifier un profil GATT simplement. Couplé au générateur de code nRF.
- **Kinetis avec WireShark :** Sniffeur de paquets Bluetooth.
- **Pencil par Evolus :** Editeur de diagrammes.

8.2 Microcontrôleur

On a utilisé le microcontrôleur nRF52832 de Nordic Semiconductor dans le cadre de ce projet.

L'intégration d'une batterie, nous a contraint à utiliser un microcontrôleur. Pour nos deux périphériques Radio, il a fallu comparer les solutions directement intégrées dans les SOC des microcontrôleurs ou les modules externes.

Pour le RFID, il y en a très peu possédant un périphérique dédié à ça. Et pour le UHF que nous voulions utiliser il n'y en avait aucun, uniquement du RFID HF.

On s’est tourné sur ceux intégrant du Bluetooth Low Energy, et là le choix était plus grand. Voici le tableau comparatif :

UContrôleur	**Processeur**	**Conso. 0dB**	**Conso. veille**	**Commentaires**
nRF51	Cortex-M0	8mA	2.6uA	Prédécesseur nRF52
nRF52	Cortex-M4f	5.3mA	2.7uA	Meilleur en consommation
ti CC2540	8051 (8bits)	27mA	235uA	Consommation trop élevée
NXP QN9020	Cortex-M0	8.8mA	3uA	Mauvais rapport W/mA
NXP KW31Z	Cortex-M0+	6.1mA	N/C	Début de production

Le nRF52 a été choisi pour son excellent rapport consommation/puissance. Il possède un Cortex M4f qui lui permet d’avoir de la puissance en réserve et de multiples périphériques sur le SOC.

8.3 Architecture

L’architecture se compose du microcontrôleur précédemment choisi connecté à un module RFID de la société Invelion.

Le périphérique Bluetooth possède une sortie Antenne, là-dessus on connecte une antenne intégrée au PCB.

Le module RFID fonctionne en 5 V et communique via de l’UART. Il est connecté à sa propre antenne qui est de type Patch.

Les circuits intégrés contrôlant la recharge et l’alimentation sont connecté sur le bus I2C.

Et finalement les seuls composants permettant des interactions avec l’utilisateur sont les boutons et la LED multicolore.

Les boutons sont connectés tout simplement sur une entrée GPIO alors que la LED a besoin d’un protocole spécifique basée sur le temps. Elle sera connectée sur le périphérique SPI car on peut générer des timings précis avec.

Le tout sera alimenté par une batterie Li-Po à une seule cellule. Elle sera rechargée par le port

8.4 Gestion des versions

Ce projet est actuellement sur GitHub sous le nom de ShuQi. Il est possible d’y accéder depuis :

---

https://github.com/taboadaa/ShuQi

Là-dedans se trouve tous les fichiers sources utilisé ainsi que le SDK. Il y aussi les schémas Altium et toute la documentation.

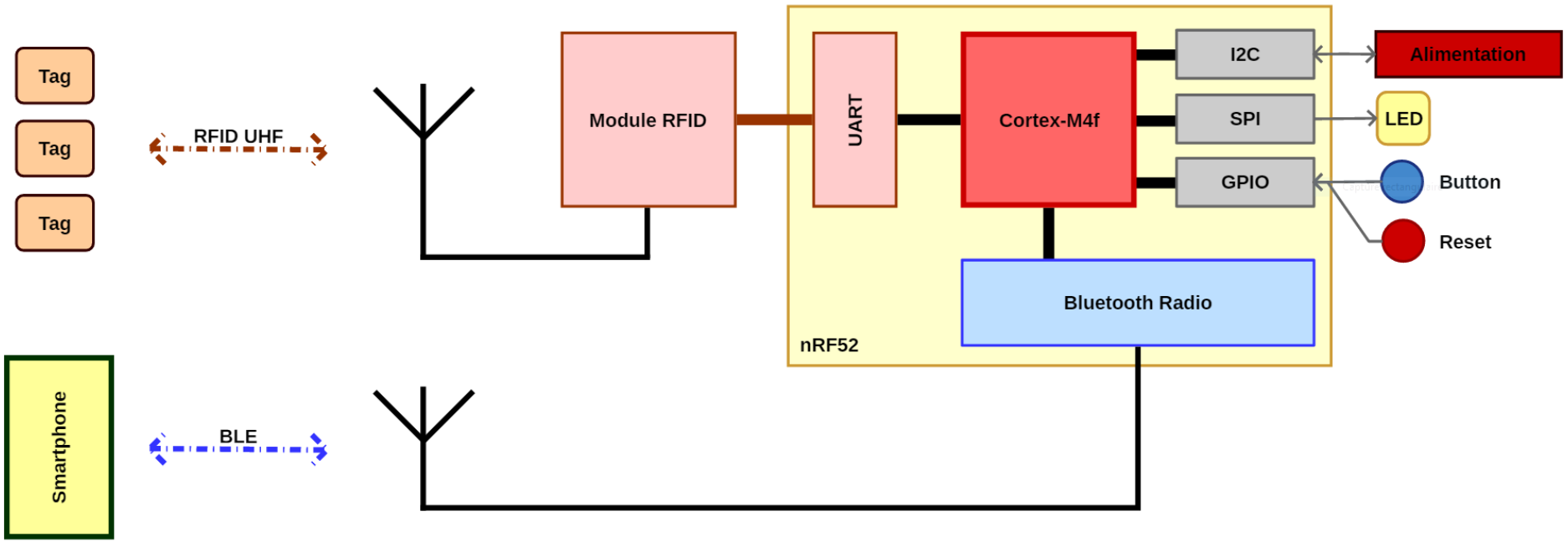


Figure 5: Architecture Hardware de la centrale

9 Alimentation

Une des grosses parties du projet est l'alimentation. Au vu de la nature de ShuQi, ce sera un module qui fonctionnera sur batterie. De ce fait, il y a trois points important :

- La source d'énergie
- La connectique
- Le système de recharge

Sur ces trois points, il faut étudier la question de quel type de source d'énergie on va utiliser, comment on va recharger cette source et finalement de quelle autonomie on aimerait avoir pour pouvoir dimensionner tous ces composants

9.1 Source d'énergie

Pour la source, il est possible d'avoir soit une pile soit une batterie. Les piles étant bien plus polluantes et non rechargeable principalement, on ne va pas les utiliser.

Par contre dans la famille des batteries, il y a pléthore de choix ! Les batteries sont le plus souvent rechargeables et peuvent quasiment avoir la forme que l'on veut ! Les technologies disponibles dans les batteries sont les suivantes :

- NiCd
- NiMH
- Lithium

Les deux premières technologies, le NiCd et le NiMH ont un très gros problème : l'effet mémoire. Qu'est-ce qu'est cet effet ? C'est le fait que si on recharge la batterie avant qu'elle ne soit déchargée, toute la capacité qui n'a pas été utilisé n'est plus disponible. Et au vu de l'usage de ShuQi, il sera rare de la recharger quand la batterie sera vide.

C'est pour ça que l'on va utiliser une batterie basée sur la technologie du Lithium. Rien que pour le fait que ces batteries n'ont pas d'effet mémoire, c'est celle-là qu'on utilise pour la centrale.

Et il faut encore choisir quel type de batterie Lithium on va utiliser, car il y a encore différents types de technologies qui ont chacune quelques différences d'ordre

Les deux types que l'on peut intégrer dans notre produit qui ont un faible cout sont les batteries Li-Ion[1] et Li-Po[2]. La plus grande différence entre les deux c'est le format. Les Lithium-Ion, à cause de leur nature, doivent avoir une coque rigide, ce qui a pour conséquence que, à capacité égale, les Lithium-Ion sont un peu plus grande et un peu plus lourdes que les Li-Po.

Les tensions utilisées dans la centrale sont 3.3 V pour le microcontrôleur et les petits circuits intégré, et 5 V pour la LED multicolore et le module RFID. De ce fait, on n'a pas besoin d'une batterie supérieure à une cellule (3.7 V).

Le choix final se porte sur une batterie Li-Po à une seule cellule qui possède une capacité de 3000 mAh.

[1] Lithium-Ion

[2] Lithium-Polymère

Mais pour des raisons de sécurité, on va lui adjoindre un circuit de protection de la batterie. C'est un petit circuit tout simple qui va monitorer la charge, la santé et la température de la batterie, et le cas échéant, coupe la batterie du système pour la protéger.

9.1 Connectique

Maintenant que la batterie a été choisie, il faut s'occuper du système de recharge. Pour recharger la batterie, on va utiliser la méthode la plus rependue : Le chargement par port USB.

Quasiment tout le monde possède un chargeur USB chez lui, que ce soit pour recharger son smartphone ou autres périphériques. Ce type de rechargement est devenu une norme à tel point que plus personne n'a de soucis en cas d'oubli de chargeur (pour tous les smartphones existant, excepté ceux de Apple).

Le rechargement s'effectue par le connecteur Micro-USB ou dans un avenir proche par son successeur : le connecteur USB Type-C.

Il faut maintenant choisir entre ces deux connecteurs :

Il y a le Micro-USB qui est implémenté dans le marché depuis longtemps. De ce fait il y a un nombre incalculable de références et de designs différents pour la recharge, ce qui facilite le développement et donne un cout de production bas.

Et d'un autre côté, il y a l'USB Type-C qui est l'évolution du précèdent. La plus grande amélioration se situe sur la réversibilité du connecteur. Mais, sachant que c'est une nouvelle norme, il y a encore peu de documentations, de références et de designs disponible. Ce qui implique que pour le moment le cout de l'USB Type-C est bien plus élevé que le Micro-USB.

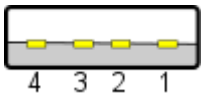
Pour mieux voir les différences entre les deux connecteurs, voici un tableau comparatif :

Micro-USB	**USB type-C**
+ Démocratisé	+ Réversible
+ Bon marché	+ Solidité
- Fragilité	+ Transition vers le type-C
- Transition vers le type-C	- Non démocratisé

Sachant que, s'il y a une mise sur le marché d'un nouveau produit, prends au moins une année. L'USB Type-C est un choix par défaut pour tous les nouveaux produits. Pour la recharge on va donc utiliser le Type-C.

Maintenant on va se pencher sur les détails de cette nouvelle connectique :

Pour commencer, ce nouveau connecteur a été pensé pour rassembler et améliorer les qualités des connecteurs USB et a pour but final de tous les remplacer. Pour le moment il y a deux types de connectique différentes utilisé pour la norme USB :



- Le Type-A qui est la prise standard que l'on trouve sur partout et qui sert le plus souvent comme hôte. Les périphériques de type clavier, souris, imprimantes, etc… se connectent avec ce connecteur à une machine hôte, le plus souvent un ordinateur. Le plus grand avantage de cette prise est sa solidité.

- Et le Micro-USB qui se trouve sur les petits appareils ou le Type-A est trop gros. C'est une évolution du Mini-USB qui a été créé pour le même but mais qui était encore trop gros. Ses principaux défauts sont qu'il n'est pas compatible USB 3.0 et de sa relative fragilité.

De ce fait, le cahier des charges du nouveau connecteur Type-C était simple :

- Compact
- Solide
- Réversible

Et il existe maintenant sous cette forme suivante :

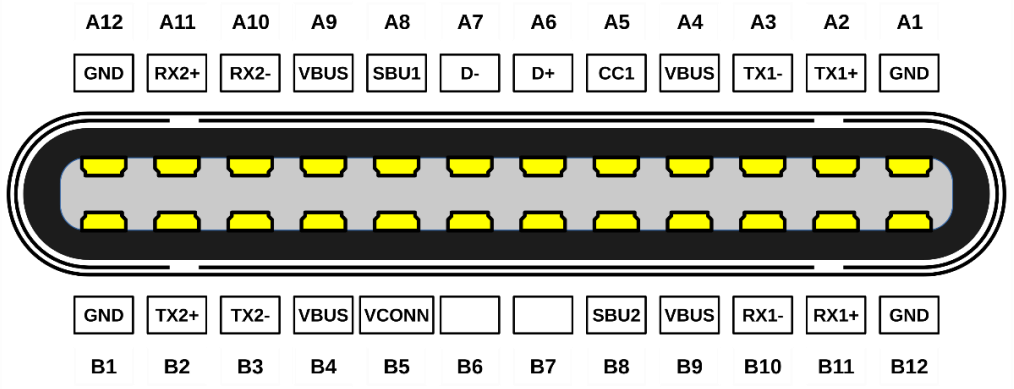


Figure 6: Connecteur USB-C

Le résultat c'est un connecteur réversible de taille comparable au Micro-USB possédant 24 pins.

On va uniquement l'utiliser pour le rechargement de la batterie. De ce fait pas besoin de gérer une connexion USB.

Par contre, ce connecteur demande des nouveaux contrôleurs qui lui permettent de savoir le sens de branchement de la prise et de communiquer avec le périphérique à l'autre bout du câble pour savoir dans quel mode fonctionner.

Petite particularité, vu que cette connectique peut supporter le USB Power Delivery, une norme permettant d'avoir 100 W (20 V, 5 A). Il est obligatoire d'avoir un régulateur de tension qui puisse supporter des pics de 20 V en entrée.

9.2 Système de recharge

Avec la batterie et le connecteur choisi, on peut s'occuper du système de recharge. Il faut pour concevoir ce système rechercher tous les composants pour pouvoir les intégrer sur un PCB.

L'USB Type-C a posé certains problèmes : C'est une connectique récente, de ce fait il y a peu de documentations permettant d'aider à la conception. A part le document contenant toute les spécifications, il y en a très peu d'autre comprenant des explications claires et concises.

Après quelques recherches, on a trouvé un exemple de Texas Instrument qui est un système complet permettant de gérer une batterie via de l'USB Type-C. C'est un ensemble de composants permettant de recharger une batterie Lithium une cellule via l'USB Type-C. C'est un design de référence de Texas, le PMP4496[ii].

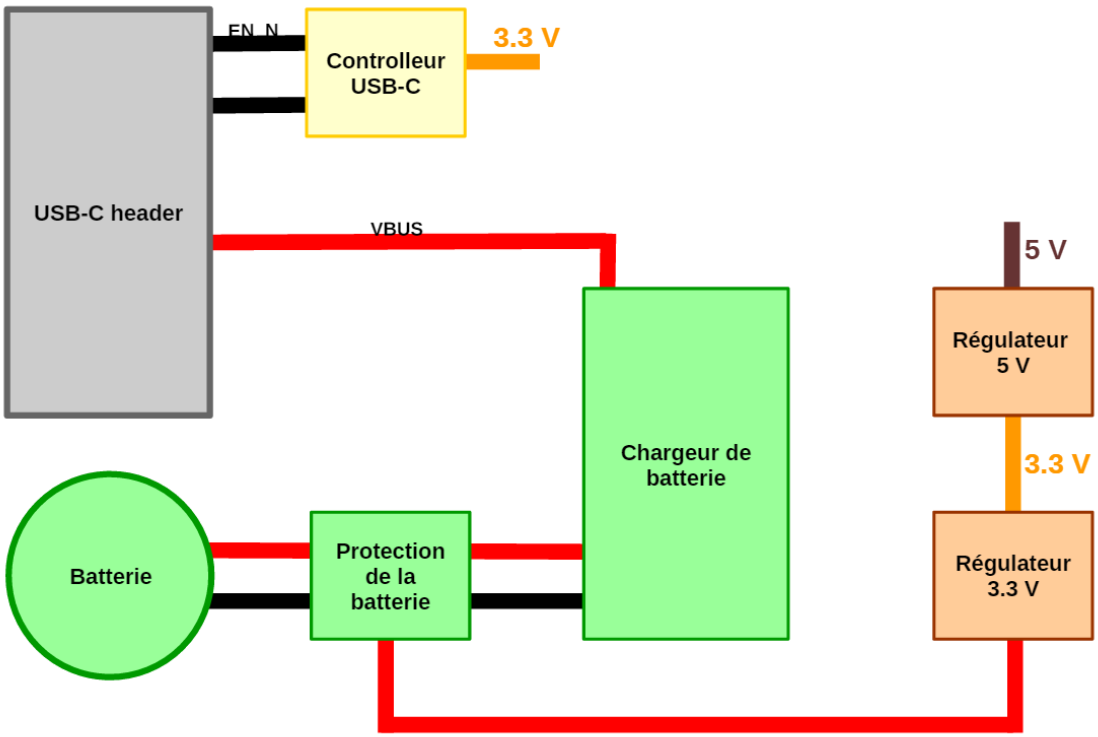


Figure 7: Schéma de l'alimentation

Un petit descriptif des composants se trouve sur la page suivante.

La sécurité d’abord. On commence par le circuit intégré qui s’occupe de protéger la batterie : BQ29705DSER. Il s’occupe de monitorer la batterie en continu pour détecter les cas de surcharge ou de trop haute décharge. En cas de détection d’un de ces cas, il va s’activer pour isoler la batterie.

Ensuite vient la gestion de l’USB Type-C, vu que c’est un connecteur réversible qui a une multitude de fonctions, il a besoin d’un petit circuit intégré permettant d’indiquer de quel sens se trouve le câble et quels sont les modes supportés. Ce circuit c’est le TUSB320LIRWBR. A chaque connexion du câble, il va y avoir un pont entre le shield et le GND du connecteur, ce qui tire son port EN_N à zéro et qui l’active. En s’activant, il va communiquer que l’on est un DRP.

La recharge est gérée par le BQ25895RTWR qui est un contrôleur de charge qui supporte la charge rapide. Lui est connecté directement sur le VBUS du l’USB, et c’est de la d’où il tirera le courant pour la recharge.

Pour finir viens la partie régulant l’alimentation des composants de la carte. Il y a, en entrée, un régulateur de tension qui supporte des pics de 20 V. C’est une obligation de l’USB Type-C car ce connecteur est compatible USB-PD qui peut monter jusqu’à des tensions de 20 V.

La batterie ou le VBUS se connecte sur l’entrée du premier régulateur de tension, un LM1085IS-3.3/NOPB qui est un LDO qui peut supporter une charge maximale de 3 A. Cette tension alimente le microcontrôleur et tous les autres petits circuits intégrés.

Ensuite vient le Step-Up qui est connecté après le régulateur. Son rôle est d’alimenter la LED multicolore et le module RFID.

10 Communications

La centrale doit maintenant communiquer avec le monde extérieur. L'objectif est de communiquer avec un smartphone.

Comment s'occuper de la communication entre la centrale et le smartphone. Il y a différentes méthodes possibles mais la solution retenue se fera sous la forme d'un schéma maitre-esclave :

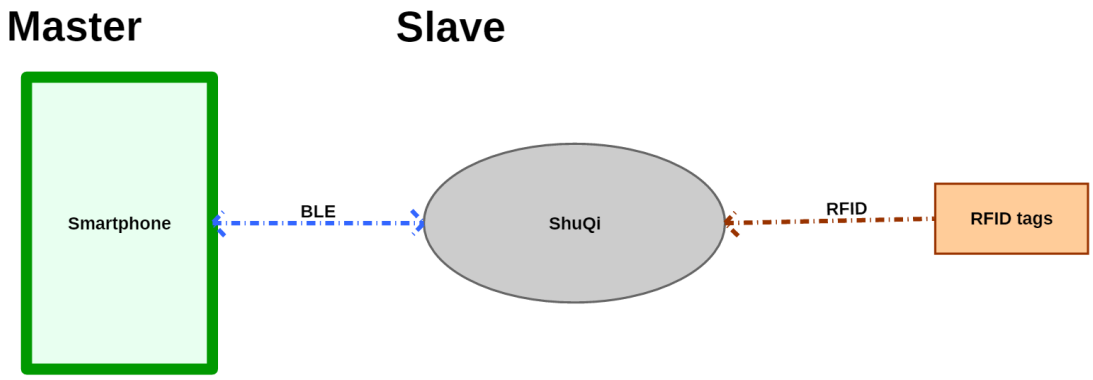


Figure 8: Maitre - Esclave

La base sera en attente de l'arrivée d'une commande de la part du smartphone dans un mode basse consommation. Elle se réveillera à chaque fois que le smartphone va écrire dans une caractéristique.

Au réveil, elle va traiter la nouvelle information récupérée sur la caractéristique changée.

10.1 Connectivités

Ceci est la plus grande partie du projet. C'est elle qui s'occupera de la communication entre la centrale et le smartphone.

Il fallait trouver un moyen de communication entre un smartphone et la centrale. Ce qui nous limite dans les protocoles supportés par la majorité :

- Bluetooth
- Bluetooth Low Energy
- Wifi
- NFC

Le cahier des charges était que l'on puisse communiquer depuis le smartphone vers la base à faible portée de l'ordre d'un à trois mètres environ avec la plus basse consommation possible.

Ce qui élimine déjà le NFC qui possède une portée utile d'environ trois centimètres.

Il nous reste le Wifi et les Bluetooth, trois technologies qui respectent la contrainte de portée. Maintenant, sachant que l'on n'aura pas beaucoup de données à faire transiter, les trois technologies restantes peuvent le supporter sans problème.

Question consommation électrique, le Bluetooth Low Energy est le moins énergivore et le plus simple à mettre en place.

10.2 Bluetooth Low Energy

La technologie choisie est le Bluetooth Low Energy (Bluetooth 4), ou anciennement appelé le Wibree. Contrairement à la dénomination de version de ce protocole, ce n'est pas le remplaçant du Bluetooth Standard.

C'est plutôt que dans la famille du Bluetooth, il y a deux sous catégories complémentaire qui sont :

- Bluetooth Standard
- Bluetooth Low Energy

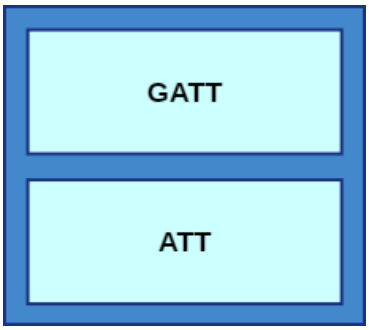
Ce sont deux technologies complémentaires qui partagent un même nom mais n'ont pas la même utilité. Le Bluetooth Standard est utilisé principalement pour des applications qui demandent un transfert de données soutenu (streaming, musique, etc…) alors que le Bluetooth Low Energy est utilisé pour des transferts de données courts et pas en continu.

De par leur nature différente, le Bluetooth Low Energy est optimisé pour une consommation moindre.

Pour notre projet, on a besoin de demander de temps en temps à la centrale de scanner les objets alentours et de les récupérer sur le smartphone. Au vu de ce constat, le Bluetooth Low Energy colle parfaitement à notre usage !

10.3 ATT / GATT

Le BLE amène deux nouveaux protocoles en son sein : le ATT et le GATT[iii]. Le GATT est un niveau au-dessus du ATT.



Le ATTribute protocol est le protocole de base utilisé dans le BLE. Il est composé uniquement d'un attribut qui lui est composé de trois éléments :

- Handle
- UUID
- Value

Le Handle est un nombre codé sur 16 bits qui identifie l'attribut sachant qu'il peut y avoir plusieurs attributs avec le même UUID sur un périphérique.

L'UUID qui est l'identifiant de l'attribut. Il n'y a aucun UUID prédéfini pour le protocole AAT, c'est le GATT qui le défini principalement.

Et la Value qui est d'une taille définie selon l'attribut qui est défini par l'UUID. Ce qui implique qu'un client doit savoir de quelle taille est cette valeur.

Les limitations de ce protocole sont qu'il ne peut pas y avoir une transmission de données utiles plus grande que 20 octets car le MTU maximum est de 22 octets.

Ensuite vient le GATT, appelé Generic ATTribute Profile. Il se trouve sur le protocole ATT et est utilisé comme profil de base pour tous les profils BLE. C'est une définition de plusieurs attributs ATT qui, ensemble, définissent des services.

Le GATT est composé d'une hiérarchie d'attributs organisé sous forme de services. Il peut y en avoir plusieurs différents sur un périphérique. Chacun contient une ou plusieurs caractéristiques qui elles peuvent encore contenir une ou plusieurs propriétés.

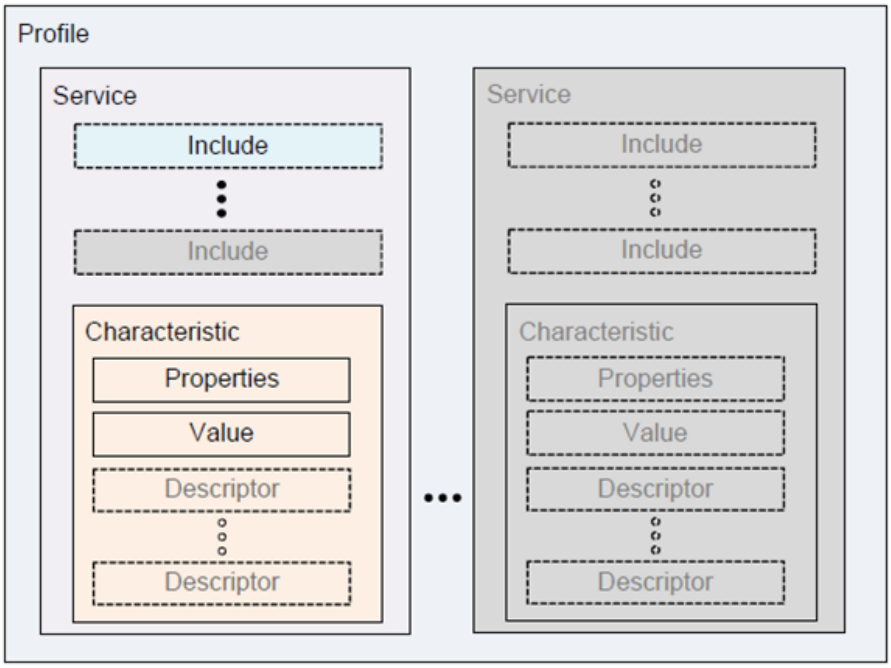


Figure 9: Hiérarchie du GATT

Le fonctionnement des deux protocoles fait qu'il y a un schéma de communication Client -Serveur. Ici la centrale sera le serveur ATT qui s'occupera des lectures et écritures de caractéristiques qui seront définies.

10.4GAP

Le Generic Access Profile est le protocole servant à gerer les accès sur le Bluetooth. C'est lui qui s'occupe de l'appairage des appareils entre eux et qui définit sa visibilité en mode advertising.

10.5Sécurité

Le Bluetooth possède quatre modes de sécurité disponible :

- JUST_WORKS
- NUMERIC_COMPARISON
- PASS_KEY
- OUT_OF_BAND

Le JUST_WORKS est le niveau de sécurité le plus simple. C'est un appairage direct entre deux appareils sans aucune confirmation quelconque. Les deux appareils d'échangent une clé et ensuite, après appairage, les communications sont encryptées. Ce qui ne protège pas des attaques de type MITM[3]. Les besoins de matériel pour utiliser ce mode sont faibles, un bouton suffit.

Le NUMERIC_COMPARISON par contre demande un écran de chaque côté avec un bouton. Avant de s'échanger les clés, il faut confirmer manuellement en regardant le code, qui va s'afficher sur les deux écrans, s'il est identique. Après confirmation, l'appairage s'effectue. Le MITM n'est plus possible.

Ensuite vient le PASS_KEY, quasiment le même niveau de sécurité que le NUMERIC_COMPARISON. C'est l'utilisation qui n'est pas pareil. Ici il n'y a pas besoin d'écran sur le périphérique, mais à la place c'est un clavier sur lequel on doit inscrire un code.

Et pour finir, le mode OUT_OF_BAND qui est le plus sécurisé. Dans ce mode l'échange de clé ne s'effectue pas sur la connexion Bluetooth mais par un autre moyen quelconque. Le plus utilisé pour échanger les clés c'est le NFC. C'est la méthode la plus sécurisé et la plus facile d'accès si on utilise le NFC.

Pour ShuQi on utilise le mode JUST_WORKS car il n'y a pas d'écran ou de clavier dessus. Et pour l'utilisation que l'on en fait, les attaques ne sont pas un très grand problème.

Le nRF52 peut faire du NFC, il est parfaitement possible d'implémenter le mode OUT_OF_BAND mais il ne l'a pas été car les smartphones Apple ne peuvent pas utiliser le NFC. C'est une des évolutions à faire pour les prochaines versions.

10.6Security Mode

Il y a trois modes de sécurité possible : 1, 2 et 3.

Le mode 1 est une connexion sans aucune sécurité, l'information passe en clair. Le mode 2 utilise une petite clé de alors que le mode 3 possède une grande clé.

On utilise le mode 2 pour la connexion pour la centrale.

[3] Man In The Middle

10.7Profil Stuff Manager

Il a fallu créer un profil Bluetooth personnalisé pour ce projet car il n'y en avait aucun prédéfini qui collait à notre cahier des charges.

Pour commencer, on a pris le profil Proximity pour lui rajouter un service : Stuff Manager Service.

Le profil Proximity (PXP) possède trois services :

- Link Loss Service (LLS)
- Immediate Alert Service (IAS)
- (Tx Power Service (TPS))

Au début on voulait rajouter un service en plus, mais il y avait trop de services inutilisés. Du coup la création d'un profil entier depuis zéro a été nécessaire.

Le résultat c'est le profil Stuff Manager (STMP) qui possède deux services pour le moment :

- Link Loss Service (LLS)
- Stuff Manager Service (STMS)

Il reste du PXP le service LLS qui s'occupe de lancer les alertes en cas de perte de connexion entre le client et le serveur. Il est, par exemple, utile pour savoir quand on a oublié notre sac.

Il y a surtout l'ajout du nouveau service crée de toute pièce pour l'occasion, le Stuff Manager Service. Il a fallu créer son propre service car il n'y en avait aucun préexistant qui pouvais faire l'affaire. Il y avait bien les suivants qui se rapprochaient de l'usage voulu, mais aucun n'était parfait.

Proximity Profile (PXP)	Services inutiles, Pas d'accès à un tableau
Object Transfert Profile (OTP)	Trop complexe, Pas les bonnes tailles des caractéristiques

Par contre, pour le moment, il manque un service dans ce profil : Le service Battery Service (BAS). Car il sera utile de savoir qu'elle est la charge restante de la centrale.

Petite anecdote, la centrale ne s'affiche pas sur les smartphone IOS car ils font un filtre sur les profils. Sachant que c'est un profil personnalisé, les appareils IOS ne l'affichent pas du tout.

10.7.1 Stuff Manager Service

Ce service est la base de la communication entre la centrale et le smartphone. C'est ce service qui va s'occuper de lancer le scan et d'énumérer la liste des tags à proximité.

De ce fait, ce service comporte quatre caractéristiques :

- Manager Mode
- Entry Number
- Entry Selection
- Entry Value

C'est sur ce service que l'on peut changer l'état de la centrale avec la caractéristique manager Mode. L'algorithme pour accéder à ce service est le suivant :

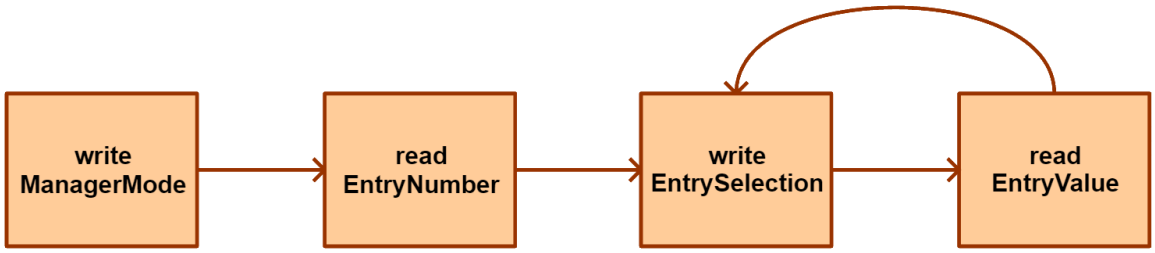


Figure 10: Algorithme pour la lecture de Stuff Manager Service

Pour pouvoir lire les tags, il faut écrire dans ManagerMode le MODE_READ, ensuite la centrale lance le scan RFID, récupère les UUID des tags et change la valeur EntryNumber qui est le nombre total de tags lus pendant ce scan. Ensuite on écrit dans EntrySelection le tag que l'on veut récupérer sur EntryValue.

10.7.1.1 Manager Mode

Octets order	LSO - MSO
Data type	uint8
Size	1 octet
Units	enumeration

Cette caractéristique est accessible uniquement en écriture. Et il n'est possible d'écrire que trois valeurs différentes au format *uint_8* :

MODE_SLEEP	0
MODE_READ	1
MODE_RECOGNIZE	2

Ces trois valeurs correspondent aux états de la machine d'états de la centrale. Au vu du schéma de communication retenu entre la centrale et le smartphone, la lecture de cette caractéristique n'était pas utile.

Une amélioration possible c'est de rajouter un accès en lecture de cette caractéristique pour que l'on puisse savoir dans quel état se trouve la centrale. Mais ce n'est pas une fonctionnalité très utile car normalement c'est le smartphone qui va gérer l'état de la centrale.

10.7.1.2 Entry Number

Octets order	LSO - MSO
Data type	uint16
Size	2 octets
Units	unitlesss

La deuxième caractéristique par ordre d'accès est celle-ci. C'est une valeur au format uint16 accessible en lecture uniquement qui informe du nombre de tags différents qui ont été lu par la centrale.

Sa valeur de base est de 0 et change à la fin du cycle de scanning des tags au nombre de tags qui ont été lus. S'il n'y a aucun tag qui a été lus, la valeur sera de 65535.

10.7.1.3 Entry Selection

Octets order	LSO - MSO
Data type	uint16
Size	2 octets
Units	unitless

Cette caractéristique fonctionne par paire avec Entry Value. Ici elle n'accepte qu'un accès en écriture dans un format uint16.

A chaque écriture, la centrale va commencer par comparer la valeur reçue avec le nombre de tags lus. Si la valeur est plus grande que le nombre de tags lus, elle va changer la valeur de Entry Value par une valeur vide (que des zéros). Sinon elle va aller chercher le bon tag dans sa mémoire et l'écrire sur Entry Value.

10.7.1.4 Entry Value

Octets order	LSO - MSO
Data type	uint128
Size	16 octets
Units	unitless

Ceci est celle on l'on va récupérer l'id du tag RFID demandée sur Entry Selection. C'est une caractéristique accessible uniquement en lecture sur un format de uint128.

Le format utilisé a été choisi sur la base de la taille de l'EPC des tags RFID qui est de 96 bits. Sachant que les formats disponibles sont des multiples de deux, le uint128 a été choisi.

10.7.2 Link Loss Service

Ce service possède qu'une seule caractéristique accessible en lecture et écriture qui se nomme Alert Level.

C'est tout simplement une énumération de plusieurs niveaux d'alertes différents :

NO_ALERT	0
MILD_ALERT	1
HIGH_ALERT	2

Dans notre implémentation, le smartphone va pooler cette caractéristique à un temps défini, et si ce n'est pas possible d'y accéder, va lancer une alerte et une notification à l'utilisateur.

10.7.3 Battery Service

Ce service est prédéfini, c'est un service permettant de récupérer la charge de la batterie. C'est le smartphone qui va récupérer cette valeur et va ensuite lancer des alertes à l'utilisateur en cas de batterie faible.

Ce service n'est pas encore implémenté ni ajouté dans le profil pour le moment.

11 Application

La centrale est gérée par une logique assez simple. Au vu de son schéma esclave, elle sera la plupart du temps en mode basse consommation. Son réveil ne se fera que de la part du smartphone qui a été appairer.

Pour commencer, la centrale comporte trois états :

- STATE_SLEEP
- STATE_READ
- STATE_RECOGNIZE

L'état de base est STATE_SLEEP. Comme son nom l'indique, la centrale ne fait aucune action et se mettra en veille. Elle sera uniquement réveillée par des interruptions qui viennent soit du bouton, soit de la part du Bluetooth en cas d'écriture sur une des caractéristiques.

A l'écriture de la caractéristique *Manager Mode*, on va récupérer ce qui a été écrit et on va comparer la valeur pour savoir dans quel mode la centrale doit passer. Cette caractéristique est celle qui a la plus grande priorité, ce qui fait que l'on peut interrompre les opérations pour passer dans le mode que l'on veut.

Mais il y a une limitation, on ne peut pas passer du mode STATE_READ au STATE_RECOGNIZE sans passer par STATE_SLEEP comme marqué sur le schéma suivant.

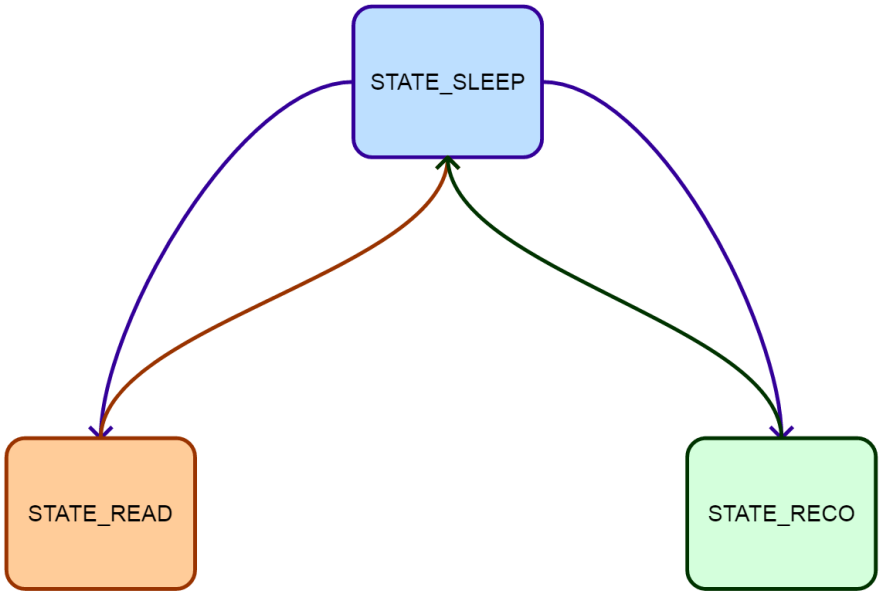


Figure 11: Etats de la centrale

Une explication plus détaillée de chaque état suivra.

Ensuite, vu que l'application sera assez complexe, on va utiliser GCC pour la compilation car Keil est limité à un programme de 32ko avec la licence gratuite. Ce qui n'est pas un grand problème vu que le SDK de Nordic est compatible Keil, IAR et GCC. Par contre, la différence entre Keil et GCC c'est qu'il y a une configuration à effectuer pour avoir un environnement de travail fonctionnel[iv].

Comme tout le projet sera généré par un makefile, il est possible d'utiliser plusieurs IDE différents, dans notre cas on va utiliser Eclipse.

11.1 Architecture

L'application va être basée sur le SDK de Nordic qui supporte les microcontrôleurs nRF51 et nRF52. La version utilisée de ce SDK est la stable dernière en date : *V12.3.0*.

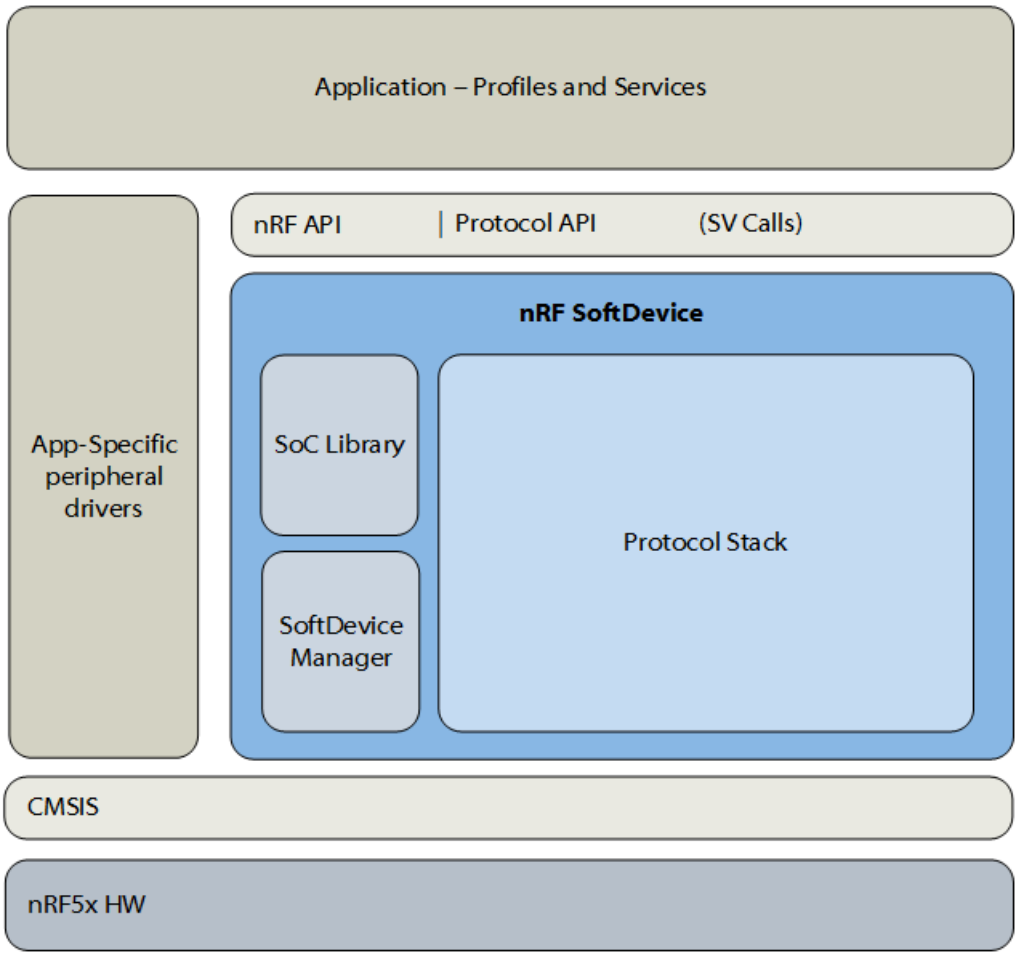


Figure 12: Architecture logicielle de la centrale

La partie des services Bluetooth est générée par un plugin spécifique aux *nRF* sur *Bluetooth Developper Studio*.

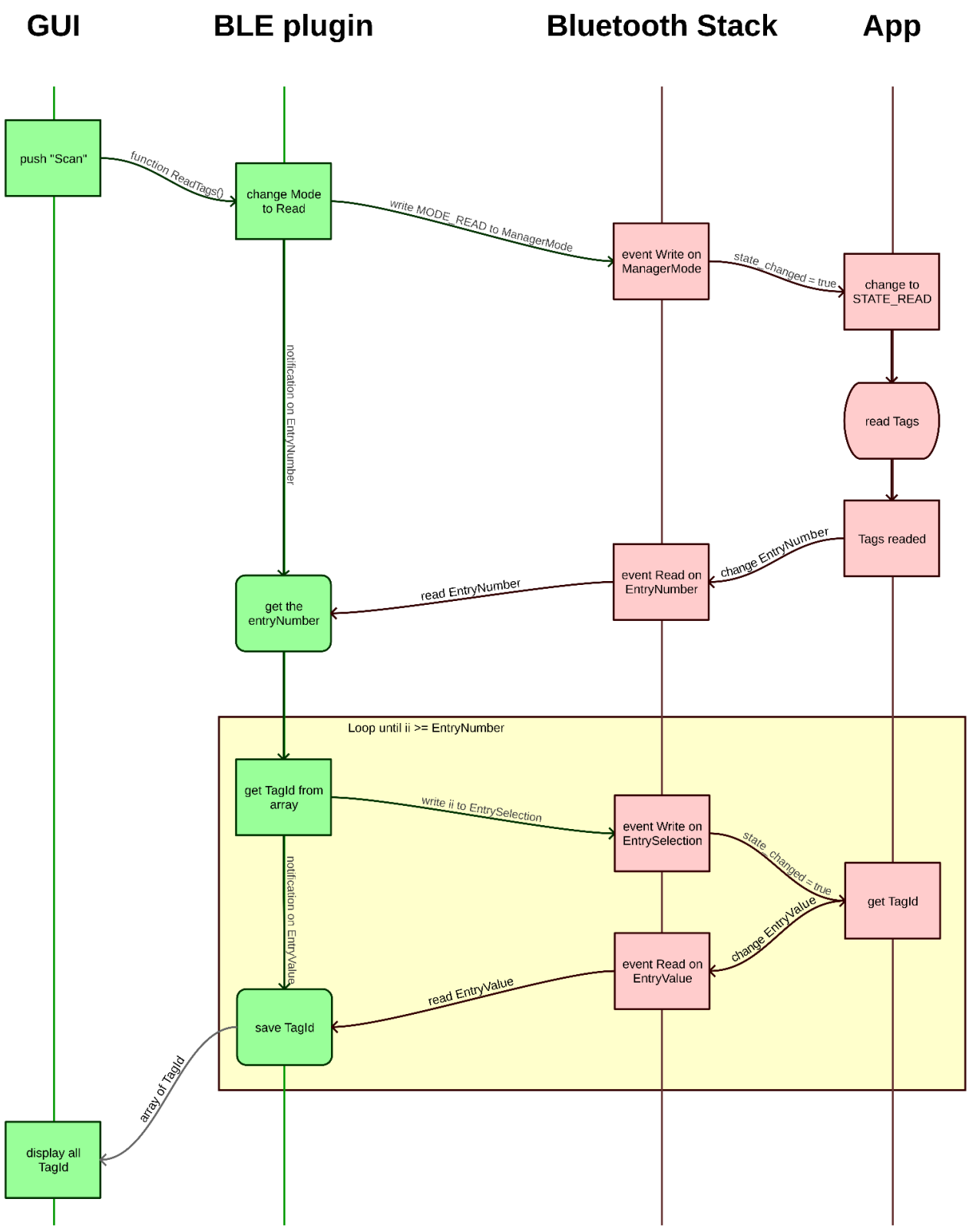


Figure 13: Suite d'appels à la suite d'une demande de Scan

11.2 Etats

11.2.1 STATE_SLEEP

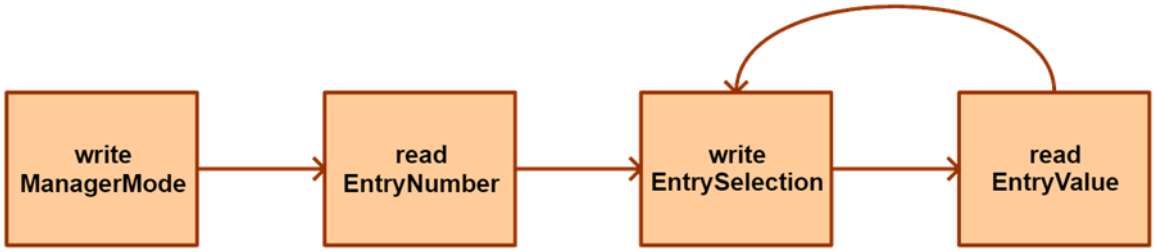
Cet état est celui ou la centrale va passer le plus de temps, il s'occupe de mettre le processeur en veille. Tous les périphériques inutilisés seront éteints sauf le module Radio avec le SoftDevice.

La connexion BLE sera en mode Advertising en attendant que le smartphone se reconnecte pour effectuer des actions.

11.2.2 STATE_READ

Celui-ci est l'état le plus important : Il s'occupe de lire les tags à proximité via le RFID, les gardez en mémoire pour les mètres à disposition à travers le service *Stuff Manager*. De ce fait, dès que l'état change, il va allumer le module et lui lancer la commande de scan. Ensuite tous les tags scannés à proximité vont être récupérer dans un buffer et ensuite ils seront mis en RAM jusqu'au prochain changement d'état.

C’est de loin le mode le plus consommateur d’énergie vue que le module RFID fonctionne.



Bien sûr s’il y a une coupure de connexion entre le smartphone et la centrale pendant cet état, elle va repasser automatiquement dans l’état STATE_SLEEP.

Par contre il serait possible de rajouter un timeout qui repasse la centrale sur l’état STATE_SLEEP au cas où l’application du smartphone plante et ne lui redonne pas l’ordre de changer d’état.

11.2.3 STATE_RECOGNIZE

Cet état est spécifique et utilisé pour la configuration des tags. Il n’est pas implémenté pour le moment mais l’idée est que l’on réduit au maximum la portée de détection des tags et que l’on reçoit le tag le plus proche. Comme ça l’utilisateur approche le tag qu’il veut nommer de la centrale et il s’affiche sur le smartphone.

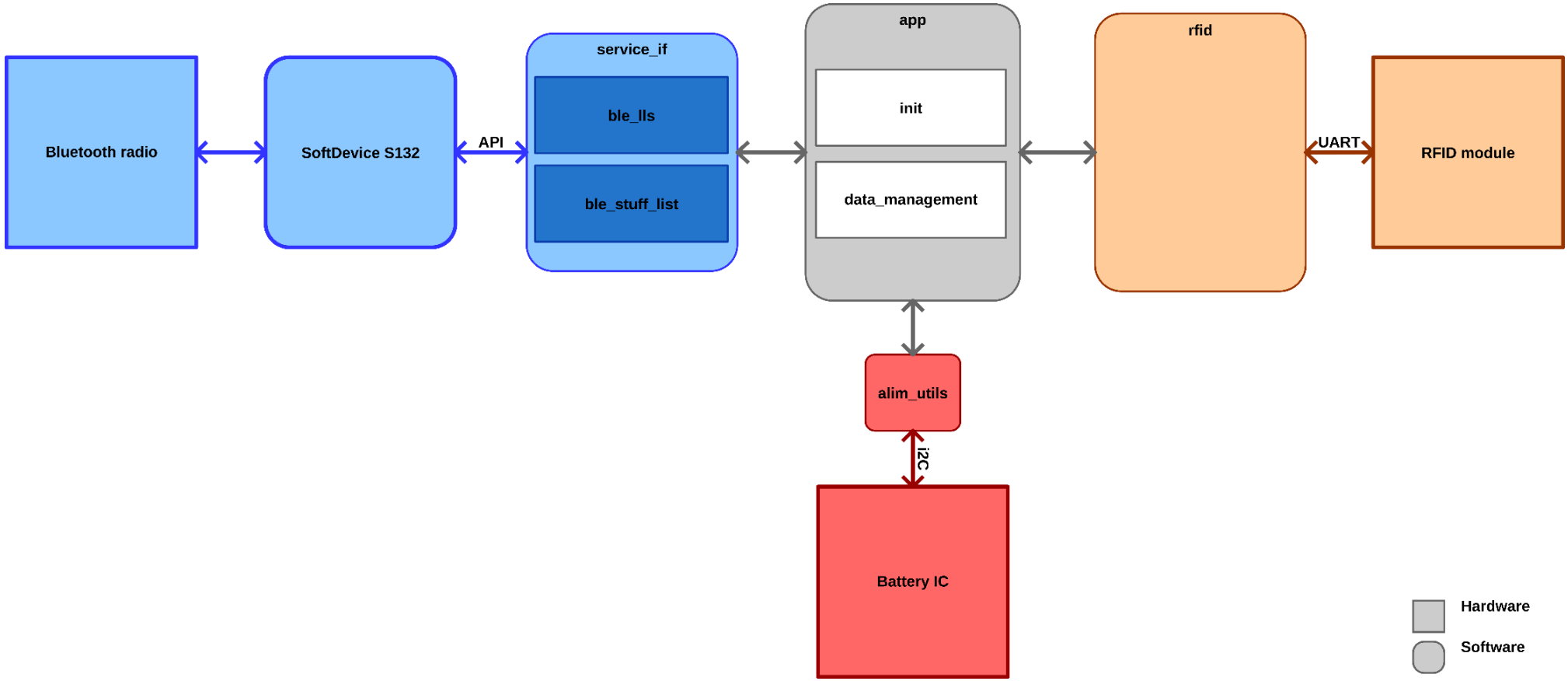


Figure 14: Architecture logicielle de la centrale

12 Application Smartphone

La troisième partie, dont je me suis occupé, de ce projet est l'application pour Smartphone.

Le marché se décompose principalement entre trois OS : Android, iOS et Windows Phone. La part de marché de chaque OS au Q3 2016[v] est de 86.8% pour Android, 12.5% pour iOS et 0.3% pour Windows Phone. De ce fait, on va se concentrer sur la plateforme Android et il y a un autre avantage, c'est que aucune licence n'est demandée pour pouvoir programmer dessus.

Maintenant, il est possible de créer une application Android de multiple façon différente :

Android Studio qui est l'IDE proposé par Google qui permet de créer des applications assez simplement si on utilise les objets de base (ce qui implique une interface graphique assez rudimentaire). Par contre le problème c'est que c'est uniquement pour Android.

L'autre solution est de passer par un framework type Cordova qui permet de créer des applications basées sur du HTML5/Javascript et qui permet de les compiler pour plusieurs plateformes.

12.1 Frameworks

Pour le choix du framework, la plupart sont basée sur Cordova qui est une base saine pour créer une application multiplateforme (Ordinateurs et smartphones). Cordova est Open Source et possède une énorme communauté, de ce fait il est extrêmement facile de trouver de la documentation. Un autre de ses points positif, c'est le système de plugins : Il existe une multitude de plugins pour quasiment tout faire.

Il existe un plugin permettant d'utiliser le BLE via des fonctions Javascript qui est compatible Android et IOS.

Or l'application est plus spécifiquement pour les smartphones, donc on va utiliser Ionic.

12.2 Bluetooth

Comme c'est des solutions basées sur du HTML5/Javascript, il n'y a pas d'accès à des fonctions natives de base. De ce fait il faut utiliser des plugins qui s'interfacent entre les fonctions natives et la partie Javascript.

Pour le Bluetooth il y a le plugin Cordova BLE Plugin[vi] qui permet d'utiliser le Bluetooth sur les appareils Android et IOS. Il possède une API complète pour utiliser le Bluetooth Low Energy.

Le Javascript est un langage asynchrone, de ce fait il faut utiliser les callbacks. De ce fait il a fallu les utiliser.

Or, par manque de temps, l'application smartphone n'est pas fonctionnelle.

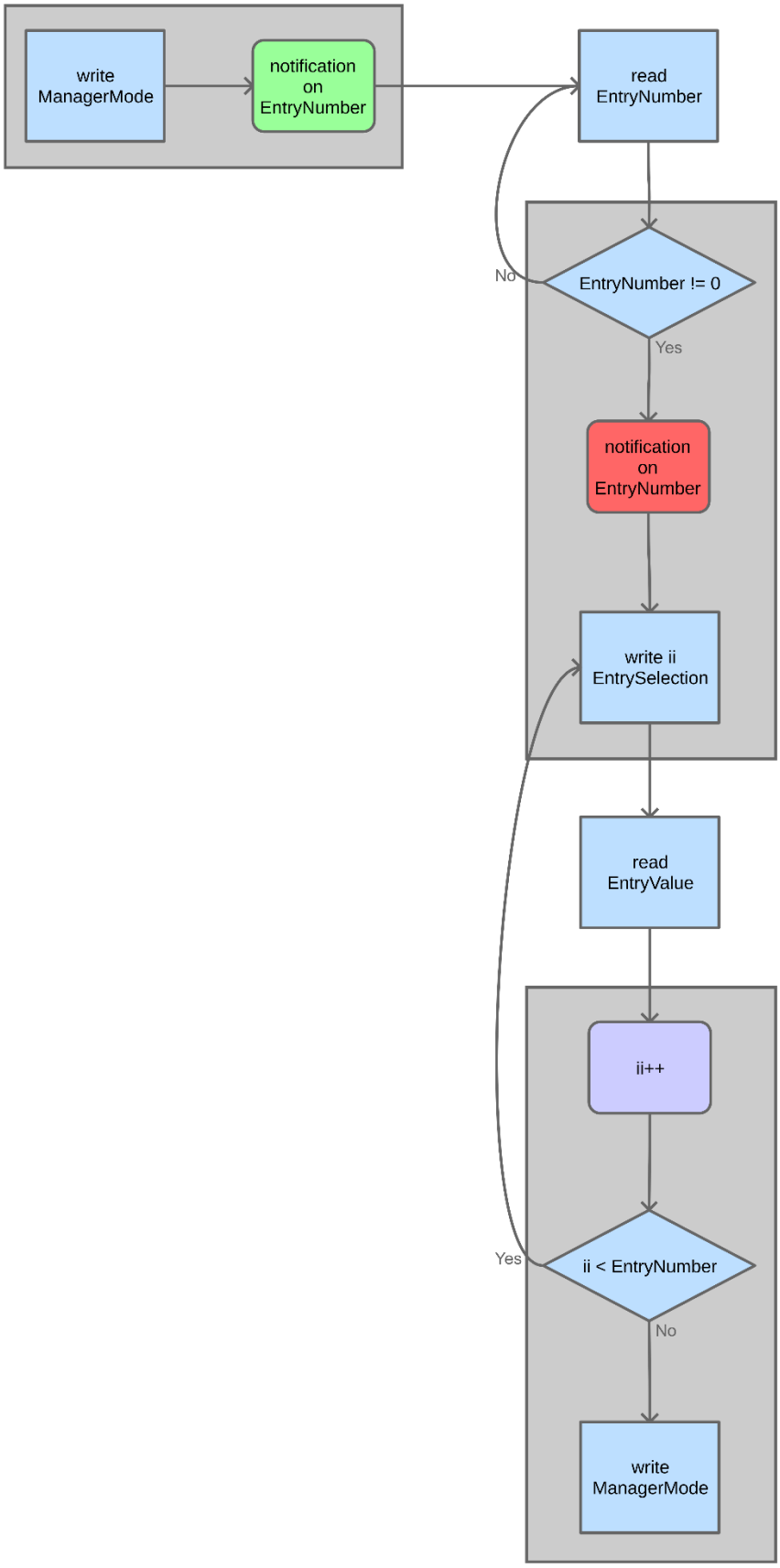


Figure 15: Fonction ReadTags()

13 Conclusion

Ce projet m’a apporté beaucoup de nouvelles compétences dans plusieurs domaines. Par exemple dans le Bluetooth Low Energy et sur l’USB Type-C.

Il a fallu lire beaucoup de documentations différentes, tout en passant par les milliers de pages de spécifications de chaque norme. C’était un travail assez long et fastidieux, ces documentations ne sont pas vraiment rédigées pour être comprises sans avoir des connaissances spécifiques préalables.

Le Bluetooth Low Energy était une nouveauté. Je n'avais jamais développé avec ce protocole avant et je ne pensais pas que c’était aussi différent du Bluetooth normal. J’ai eu beaucoup de documentation à lire pour comprendre comment fonctionne ce protocole.

Il y a également tout ce qui se rapporte à l’USB Type-C qui a exigé aussi de lire beaucoup de documentation. Du fait de sa relative jeunesse, il y a très peu de tutoriels, de résumés ou d’explications claires et concises pour comprendre plus facilement comment il fonctionne.

Dans la collaboration, le fait de travailler avec des personnes de différents horizons m'a beaucoup appris. Au début, on ne savait pas avec qui on allait devoir collaborer, tout ce qu’on savait c’est qu’on serait cinq en tout pour la Team Geneva. Les difficultés principales étaient liées aux centres d'intérêts divergents et au langage métier spécifique à chacun. Nous avons tous trouvé notre place dans l'équipe et avons eu plaisir à travailler sur ce projet.

De manière générale, je suis très satisfait du projet réalisé et du résultat. Si j'avais pu avoir les compétences spécifiques liées au Bluetooth Low Energy et sur l'USB Type-C dès le départ, j'aurais gagné beaucoup de temps. Ce temps-là, j'aurais pu l'investir dans l'optimisation de l'application de la centrale et le développement de fonctionnalités additionnelles.

La collaboration avec les designers est plus intéressante que contraignante.

Axel Collet

14 Tables des sigles et acronymes

BLE	Bluetooth Low Energy
RFID	Radio Frequency IDentification
NFC	Near Field Communication
SDK	Software Developpement Kit
GCC	GNU C Compiler
IDE	Environnement de développement intégré
UUID	Unique Identifier
PXP	Proximity profile
LLS	Link Loss Service
IAS	Imediate Alert Service
BAS	Battery Service
STMS	Stuff Manager Service
STMP	Stuff Manager Profile
ATT	Attribute Protocol
GATT	Generic Attribute Profile
PCB	Printed Circuit Board
SOC	Socket on a Chip
NiCd	Nickel Cadmium
NiMH	Nickel Metal Hydrure
Li-Ion	Lithium-Ion
Li-Po	Lithium-Polymère

15 Références

15.1 Nordic Semiconductor

- Nordic Developer Zone
- https://infocenter.nordicsemi.com/index.jsp
- http://developer.nordicsemi.com/nRF51_bluetooth_development_studio_plugin/

15.2 Bluetooth

- Bluetooth SIG
- https://www.bluetooth.com/specifications/adopted-specifications

15.3 USB

- USB Implementers Forum
- http://www.usb.org/developers/usbtypec/

15.4 Batteries

- http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries
- http://www.ti.com/tool/PMP4496#0

[i] http://chi.camp/projects/
[ii] http://www.ti.com/tool/PMP4496#0
[iii] https://epxx.co/artigos/bluetooth_gatt.html
[iv] https://devzone.nordicsemi.com/tutorials/7/development-with-gcc-and-eclipse/
[v] http://www.idc.com/promo/smartphone-market-share/os
[vi] https://www.npmjs.com/package/cordova-plugin-ble