Développement mobile avancé
Laboratoire n°4

Communication avec un périphérique BLE

04.05.2023

Introduction

Ce laboratoire vous propose de réaliser une application mobile pouvant se connecter et communiquer avec un périphérique *Bluetooth Low Energy*. Ce laboratoire couvre la découverte des périphériques compatibles à proximité, la connexion entre le smartphone et un périphérique, ainsi que l'échange de données avec deux services *BLE* (1 standard et 1 custom).

Environnement

Pour ce laboratoire nous vous mettons à disposition :

- Le squelette d'une application que vous compléterez pour les différentes manipulations. Celle-ci utilise directement le *SDK Android Blutetooth*[1] pour la phase de découverte (le scan) et une librairie tierce *Android-BLE-Library*[2] pour la mise en place de la communication avec le périphérique.
- Un écran connecté *Espruino Pixl.js*[3] (Fig. 1) disposant d'un logiciel et de services *BLE* spécifiques pour ce laboratoire

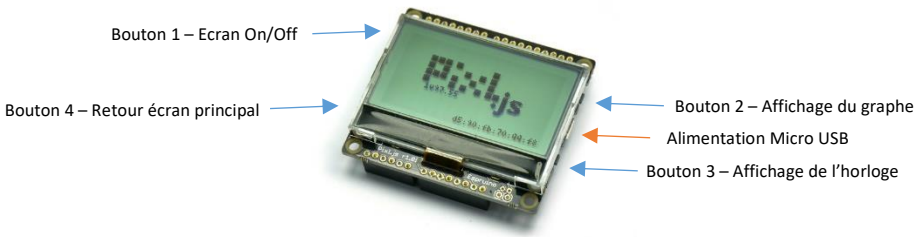

Figure 1 - Un écran Espruino Pixl.js – Configuration des 4 boutons

[1] https://developer.android.com/reference/android/bluetooth/BluetoothAdapter
[2] https://github.com/NordicSemiconductor/Android-BLE-Library
[3] Documentation *Espruino Pixl.js* : https://www.espruino.com/Pixl.js

Manipulations

L'application à développer sera composée d'une seule *Activité* accueillant deux *Fragments* : un *Fragment* destiné au mode « non-connecté » qui permettra alors de lancer un scan *BLE* et d'afficher la liste de périphériques à proximité, et un *Fragment* destiné au mode « connecté » qui permettra d'interagir avec l'écran auquel on s'est connecté. Bien qu'il soit possible d'utiliser directement le *SDK Android* pour réaliser toutes les opérations *BLE*, nous avons décidé d'utiliser une librairie tierce pour simplifier les échanges avec l'écran. La raison principale est que le *BLE* nécessite d'avoir des opérations synchrones, par exemple si on lance la lecture d'une caractéristique, on va devoir attendre de recevoir sa valeur avant de pouvoir lancer une autre opération, la librairie permet de grandement simplifier cela en gérant une file d'opérations.

Les services configurés sur l'écran connecté pour ce laboratoire sont les suivants :

- Current Time Service **0x1805**[4]
  Ce service suit les spécifications officielles[5] du consortium *Bluetooth SIG*. Celui-ci permet de lire et de régler l'heure du périphérique. Les écrans *Pixl.js* n'étant pas alimentés en permanence, leur horloge interne se réinitialise lorsque l'alimentation est coupée, ce service permet également de remettre l'horloge à l'heure courante.

  Seule la caractéristique obligatoire est implémentée :

  - 0x2A2B - Current Time
    Cette caractéristique permet de lire et d'écrire l'heure courante. Elle permet aussi de s'enregistrer au périphérique pour recevoir des notifications régulières contenant l'heure du périphérique. Le format de données échangé suit la spécification officielle, la documentation *Bluetooth* n'est pas toujours très accessible ni simple à prendre en main, il faut suivre plusieurs liens pour obtenir toutes les informations nécessaires. Il existe un dépôt *git* tiers[6] qui regroupe des fichiers *XML* détaillant les différents services et les formats de données échangées. Pour ce laboratoire nous représentons ci-dessous le format des données à utiliser :

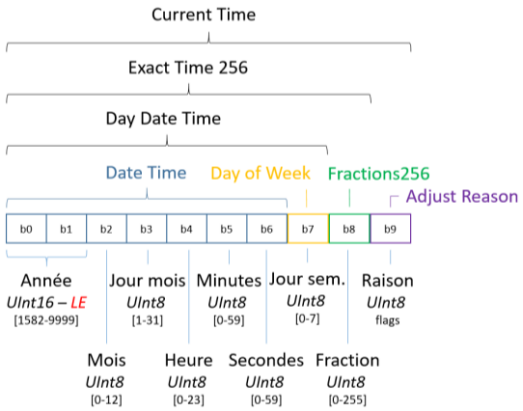

Figure 2 - Format d'échange de données "Current Time"

[4] 0x1805 ⟶ 0000**1805**-0000-1000-8000-00805f9b34fb – Conversion des UUID courts en UUID longs
[5] *Current Time Service* : https://www.bluetooth.com/specifications/specs/current-time-service-1-1/
[6] Spécification format de données *Current Time* : https://github.com/oesmith/gatt-xml/blob/master/org.bluetooth.characteristic.current_time.xml

- Service Custom SYM 3c0a**1000**-281d-4b48-b2a7-f15579a1c38f
  Il s'agit d'un service custom mis en place spécialement pour ce laboratoire. Celui-ci met à disposition 3 caractéristiques différentes :
  - 3c0a**1001**-281d-4b48-b2a7-f15579a1c38f - int (Uint32LE)
    Cette caractéristique permet au smartphone d'envoyer un entier codé sur 4 bytes, au format *UInt32LE*. Le périphérique stocke les 10 derniers entiers reçus et les affiche à l'écran sous la forme d'un graphique
  - 3c0a**1002**-281d-4b48-b2a7-f15579a1c38f - Temperature (Uint16LE)
    Cette caractéristique permet au smartphone de lire la température du thermomètre intégré au périphérique. La température est retournée sous la forme d'un nombre entier codé sur 2 bytes, au format *UInt16LE*. Il faudra diviser ce nombre par 10 pour obtenir des degrés *Celsius*. Le périphérique rafraichit cette mesure toutes les 30 secondes
  - 3c0a**1003**-281d-4b48-b2a7-f15579a1c38f – BTN (Boutons)
    Le smartphone peut s'inscrire (notification) à cette caractéristique. Celle-ci retourne un compteur codé sur un byte (UInt8) qui est mis à jour à chaque fois qu'un des quatre boutons de l'écran est cliqué, celle valeur est notifiée au smartphone, s'il est inscrit

1. Scan BLE - Ajout d'un filtre

Pour ce laboratoire nous vous fournissons un squelette d'application dans lequel l'architecture et certaines fonctionnalités sont déjà mises en place. En particulier il est déjà possible de lancer un scan des périphériques *BLE* à proximité qui seront affichés dans une *RecyclerView*. Votre première tâche consiste à ajouter un filtre au scan de périphériques pour n'afficher que ceux disposant du *Service Custom SYM* (3c0a1000-281d-4b48-b2a7-f15579a1c38f) avec lequel notre app sera compatible.

La taille des messages d'annonces *BLE* étant limitée à quelques bytes, un périphérique qui offrirait plusieurs services ne pourra pas forcément tous les annoncer. L'idée étant ici de faire un premier tri pour se limiter aux périphériques semblant être compatibles, dans la seconde manipulation, nous vérifierons, une fois la communication établie avec le périphérique, que les *services*, les *caractéristiques* et les opérations réellement implémentés correspondent bien à ce qui est attendu.

2. Exploration des services et caractéristiques sur le périphérique connecté

A la suite de la première manipulation, lorsque l'utilisateur clique sur un des périphériques à proximité, l'application se connectera à celui-ci à l'aide de la librairie *Android-BLE-Library*. En particulier, la librairie va gérer par elle-même la connexion au périphérique, elle nous redonnera la main via l'appel à deux fonctions, surchargées dans le *DMABleManager* : *isRequiredServiceSupported()* et *initialize()*.

La première, *isRequiredServiceSupported()*, devra déterminer si l'appareil auquel on veut se connecter possède bien tous les *services*, toutes les *caractéristiques* et toutes les opérations attendues. Elle devra retourner *true* si c'est bien le cas, sinon la librairie interrompra la connexion. Vous pouvez profiter de cette méthode pour garder une référence vers les *services* et les *caractéristiques* avec lesquels on devra interagir par la suite.

La seconde, *initialize()*, sera appelée par la librairie juste avant que la procédure de connexion soit finalisée. On pourra, par exemple, s'enregistrer ici sur les caractéristiques offrant des *notifications.*

3. Mise en place de l'API de communication avec le périphérique

Il s'agit de mettre en place ici les méthodes « métiers », par exemple pour la lecture de la température, l'utilisateur appuiera sur un bouton de la *GUI*, l'évènement sera transmis au *ViewModel* avec un appel à la méthode *readTemperature()*, s'il existe une connexion celle-ci passera directement l'appel à la méthode *readTemperature()* du *DMABleManager*, qui lancera la lecture sur la *caractéristique* correspondante. La lecture sur le périphérique étant asynchrone, il s'agira d'enregistrer un callback qui utilisera l'interface *DMAServiceListener*, implémentée par le *ViewModel*, pour transmettre la valeur reçue en retour. En plus de la lecture de la température, il faudra mettre en place toutes les méthodes métiers nécessaires à la réalisation de la manipulation 4.

4. Conception de l'interface utilisateur

Votre travail sera ensuite de créer l'interface du *BleConnectedFragment*, depuis celle-ci l'utilisateur devra pouvoir visualiser les données reçues (lecture, notification) depuis le périphérique et déclencher l'envoi de données ou d'évènements vers celui-ci (lecture, écriture de données). Il faut que toutes les fonctionnalités mises à disposition dans les services *Current Time Service* et *Service Custom SYM* soient utilisées, votre solution devra permettre, en particulier, de :

- Afficher la température du périphérique (lecture) ;
- Afficher le nombre de boutons cliqués (notification) ;
- Envoyer un nombre entier au périphérique (écriture) ;
- Afficher l'heure du périphérique (notification) ;
- Mettre à jour l'heure sur le périphérique (écriture).

5. Questions théoriques

Veuillez répondre aux questions suivantes dans votre rapport :

5.1 La caractéristique permettant de lire la température retourne la valeur en degrés *Celsius*, multipliée par 10, sous la forme d'un entier non-signé de 16 bits. Quel est l'intérêt de procéder de la sorte ? Pourquoi ne pas échanger un nombre à virgule flottante de type *float* par exemple ?

5.2 Le niveau de charge de la pile est à présent indiqué uniquement sur l'écran du périphérique, mais nous souhaiterions que celui-ci puisse informer le smartphone sur son niveau de charge restante. Veuillez spécifier la(les) caractéristique(s) qui composerai(en)t un tel service, mis à disposition par le périphérique et permettant de communiquer le niveau de batterie restant via *Bluetooth Low Energy*. Pour chaque caractéristique, vous indiquerez les opérations supportées (lecture, écriture, notification, indication, etc.) ainsi que les données échangées et leur format.

Durée / Evaluation

- Durée de 4 périodes, à rendre le dimanche **21.05.2023** à **23h55** au plus tard.
- Pour rendre votre code, nous vous demandons de bien vouloir zipper votre projet Android Studio en veillant à bien supprimer les dossiers build (à la racine et dans app/) pour limiter la taille du rendu.
- Vous remettrez également un rapport au format **pdf** comportant au minimum vos explications sur la solution que vous proposez pour les différentes manipulations, celles-ci devront en particulier couvrir les différents points et questions mentionnés dans la donnée.
- Merci de rendre votre travail sur *CyberLearn* dans un zip unique. N'oubliez pas d'indiquer vos noms dans le code, sur vos réponses et de commenter vos solutions.