D'APPLICATIONS MOBILES

TI PART III – IMPLEMENTATION OF THE BLUETOOTH FUNCTIONALITIES

David GREMAUD & Samuel MERTENAT

Télécommunications
T3b & T3g

Fribourg, le 3 mai 2016

DÉVELOPPEMENT D'APPLICATIONS MOBILES

ERREUR ! UTILISEZ L'ONGLET ACCUEIL POUR APPLIQUER SUBTITLE AU TEXTE QUE VOUS SOUHAITEZ FAIRE APPARAITRE ICI.

1 INTRODUCTION

Le but de ce travail pratique, intitulé « TI Part III » est d'implémenter les fonctionnalités liées au Bluetooth afin de pouvoir détecter les balises, cachées au sein de la HEIA.

2 QUESTIONS

2.1 QUESTION 1

Décrivez la réalisation de la classe Beacon, en particulier :

- ***La manière dont les données du scanRecord sont analysées.***
- ***La manière dont le major et minor sont extraites et stockées.***

Pour analyser les données issues du scanRecord, nous débutons par déclarer des constantes, conformément à la spécification fournie dans les consignes, pour récupérer les différentes informations. Nous déclarons également la valeur « minor » de la première balise attendue.

```
private static final int B_INDEX= 2;
private static final int P_INDEX = 4;
private static final int P_LENGTH = 16;
private static final int M_INDEX = 20;
private static final int N_INDEX = 22;
private static final int PO_INDEX = 24;
private static int next_beacon = 246;
```

A l'aide des constantes définies précédemment, nous avons à présent tout loisir de récupérer les données souhaitées du scanRecord. La valeur du « major » se trouve à la position 20 (M_INDEX, 1 octet) et celle du « minor » à la position 22 (N_INDEX, 1 octet).

```
byte[] proxyUuid = new byte[P_LENGTH];
System.arraycopy(scanRecord, start+P_INDEX, proxyUuid, 0, P_LENGTH);
String brutUuid = bytesToHex(proxyUuid);
String uuid = brutUuid.substring(0,8) + "-" +
        brutUuid.substring(8,12) + "-" +
        brutUuid.substring(12,16) + "-" +
        brutUuid.substring(16,20) + "-" +
        brutUuid.substring(20,32);
```

```
int major = (scanRecord[start+M_INDEX] & 0xff) * 0x100 + (scanRecord[start+M_INDEX+1] & 0xff);
int minor = (scanRecord[start+N_INDEX] & 0xff) * 0x100 + (scanRecord[start+N_INDEX+1] & 0xff);
int txLevel = (scanRecord[start+PO_INDEX]);
int rssi = result.getRssi();
```

En accord avec la spécification, nous devons trouver, pour les valeurs « major » et « minor » du premier beacon, les valeurs 1 et 246. Si c'est bien le cas, nous créons une instance de la classe Beacon avec les informations précédemment récoltées.

```
// verification of the beacon data
if(major != 1) return null;
if(minor != next_beacon) return null;

return new Beacon(uuid, major, minor, txLevel, rssi);
```

Dans le cas contraire, si la valeur du « major » n'est pas égale à 1 ou que le beacon détecté n'est point le bon (différent de 246 pour le premier), nous retournons la valeur « null ».

- ***Les méthodes qui permettent de comparer plusieurs instances de Beacon.***

Dans notre implémentation, une seule instance de la classe Beacon existe à la fois. Cette dernière est instanciée dans la méthode de callback « mLeScanCallback » appelée après la la détection d'un périphérique Bluetooth.

```
private ScanCallback mLeScanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, ScanResult result) {
        Beacon data;
        data = Beacon.createFromScanResult(result);
        if (data != null) {
            ...
            int prox = data.calculateProximity();
            if (prox == Beacon.PROXIMITY_IMMEDIATE) {
                ...
            }
        }
    }
}
```

Nous ne disposons donc pas de mécanismes pour comparer les différentes instances de la classe Beacon, mais nous avons implémenté une classe « DiscoverableBeacon » pour stocker les informations relatives à chaque balise à trouver (position, URL des indices, etc).

```
public class DiscoverableBeacon {
    private int mId;
    private String mHintUrl;
    private int mStepsNumber;
    private boolean mIsFound;
    private double mLongitute;
    private double mLatitude;
    ...
}
```

Dans notre activité principale, « BeaconsMapActivity », nous créons ensuite une liste d'instances, sous la forme d'un « ArrayList », que nous populons dans la méthode « onCreate() ».

```
// list of the discoverable beacons
private ArrayList<DiscoverableBeacon> mDiscoverableBeacons;
private int mDiscoverableBeaconsIndex = 0; // 0 = not beacon found

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mDiscoverableBeacons = new ArrayList<DiscoverableBeacon>();
    mDiscoverableBeacons.add(new DiscoverableBeacon(246,
"https://onedrive.live.com/download.aspx?cid=9029D0C756D0A25C&resid=9029d0c756d0a25c%21434&authke
y=%21AOyXHfk56pXiPPQ&canary=", -1, false, -1, -1));
    mDiscoverableBeacons.add(new DiscoverableBeacon(247,
"https://onedrive.live.com/download?cid=9029D0C756D0A25C&resid=9029D0C756D0A25C%21435&authkey=AF_
xhDl4O43BH5c", -1, false, -1, -1));
    mDiscoverableBeacons.add(new DiscoverableBeacon(248,
"https://onedrive.live.com/download?cid=9029D0C756D0A25C&resid=9029D0C756D0A25C%21345&authkey=AFo
m4lFP0egO6Qs", -1, false, -1, -1));
    mDiscoverableBeacons.add(new DiscoverableBeacon(249,
"https://onedrive.live.com/download?cid=9029D0C756D0A25C&resid=9029D0C756D0A25C%21436&authkey=AIW
a3ZShZQ-hJ8Q", -1, false, -1, -1));
}
```

La variable « mDiscoverableBeaconsIndex » nous permet ensuite de pointer vers la bonne instance contenue dans la liste. Cet index est incrémenté d'une unité lorsque le beacon à détecter est trouvé.

- ***La méthode qui permet d'obtenir une estimation de la distance.***

Pour obtenir une estimation de la mesure, nous avons implémenté plusieurs méthodes. La méthode « calculateAccuracy » permet d'évaluer la distance entre le beacon et l'appareil. Pour réaliser cette méthode, nous nous sommes basés sur la formule données qui est :

$$\text{RSSI [dBm]} = -10 * n * log_{10}(distance) + Tx$$

Nous avons transformé cette formule en isolant la variable « distance » et nous obtenons :

$$distance\ [m] = 10^{(\frac{RSSI - Tx}{10 * n})}$$

Ensuite, grâce à la méthode « analyzeLastData », nous examinons un certains nombres de mesures qui permettent d'éviter les grandes fluctuations de RSSI et nous retournons la valeur médiane de notre échantillon. D'autres algorithmes plus poussés pourraient être implémentés pour éviter les valeurs extrêmes.

Pour finir, nous avons implémenté la méthode « calculateProximity » qui indique le niveau de proximité de l'appareil par rapport au beacon, ce qui permet à l'application de colorer le bouton de recherche selon la proximité du beacon.

2.2 QUESTION 2

Affichez un graphe dont l'axe des x est la distance réelle et l'axe des y l'estimée de la distance (moyenne et variance) obtenue par la méthode réalisée dans la classe Beacon, en effectuant des mesures pendant environ 1 minute à 1m, 3m, 5m et 10 m du premier beacon.

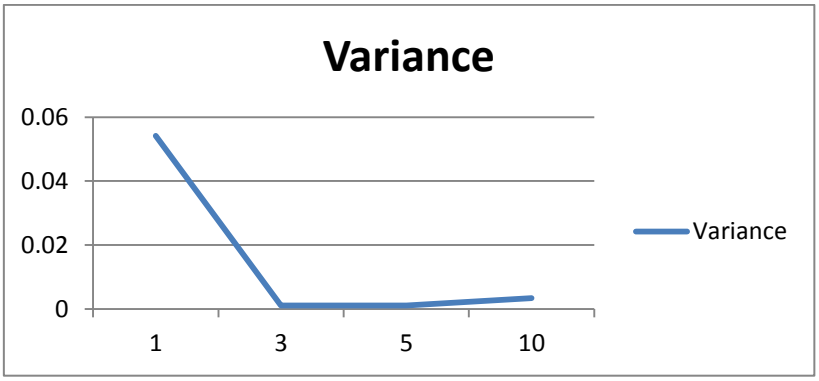

Figure 1 : variance

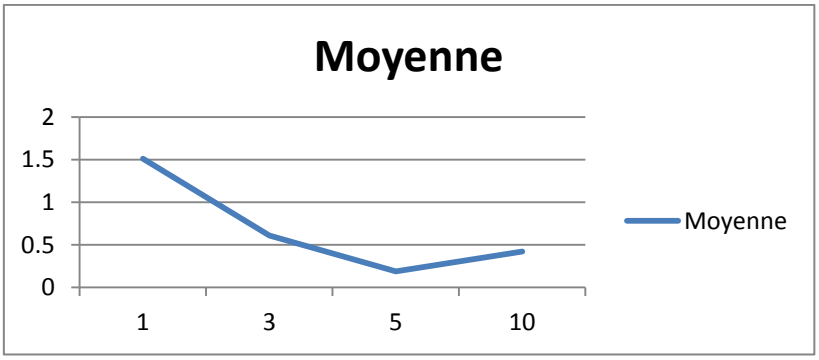

Figure 2 : moyenne

Ce que nous pouvons constater de ces deux graphiques est que notre méthode pour calculer la distance n'est pas fonctionnelle à plus de 1m. Notre méthode « analyzeLastData » nous permet d'avoir une variance assez faible.

2.3 QUESTION 3

Décrivez la réalisation de la méthode utilisée pour télécharger les hints et les afficher à l'écran, en particulier :

L'affichage des indices est déclenché par une pression sur le bouton situé en bas, à gauche de l'écran.

- ***Le mécanisme utilisé pour réaliser une tâche asynchrone de téléchargement.***

Pour télécharger les indices, contenus dans des fichiers textes, hébergés sur le cloud de Microsoft, nous utilisons une tâche asynchrone afin d'effectuer le chargement du fichier et la lecture de ses données. En utilisant ainsi une tâche asynchrone, l'acquisition des indices se fait en parallèle et ne bloque donc pas le thread principal. Notre mécanisme est basé grandement sur le code fourni par M. Ayer[1], dont seule la méthode « doInBackground(...) » a subit des modifications importantes.

La méthode « doInBackground(…) » reçoit en paramètre l'URL, à laquelle nous devons effectuer le téléchargement pour récupérer les précieux indices. Après avoir acquis le fichier, nous devons le parcourir à l'aide de la méthode « readLine() » et nous procédons également à une « sélection » de son contenu. En effet, nous souhaitons en extraire que les indices ; nous utilisons alors les méthodes « contains(…) » et « trim() » pour, respectivement vérifier la présence des caractères « minor » et les caractères alphanumériques.

Contenu du premier fichier :

```
Note: minor 247

Vous pouvez bien sûr vous arrêter afin d'approfondir votre savoir dans votre domaine préféré.
Ou partir à la recherche du prochain trésor.
Je suis grand. En fait le plus grand.
Je n'aime pas la lumière et on ne me rend visite pour la trouver... Si ce n'est lorsque mon
invité principal est brillant !
```

Méthode « doInBackground(…) » :

```
@Override
protected String doInBackground(String... urls) {
    ...
    String hints = "";
    FileInputStream fis = null;
    try {
        File input = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS), fileName);
        fis = new FileInputStream(input.getPath());
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader bf = new BufferedReader(isr);
        String line = "";
        while ((line = bf.readLine()) != null) {
            if (line.contains("minor")) {
                // do nothing
                // first line of the text hint, which contains "minor"
                // example: Note: minor 247
            } else if (line.trim().length() == 0) {
                // do nothing
                // second line of the text hint
                // the line contains no letters/numbers
            } else {
                // add the line to the string
                hints += line + "\n";
            }
        }
    } catch (IOException e) {
```

[1] *https://gitlab.forge.hefr.ch/serge.ayer/mobile-applications-15-16*

```
            e.printStackTrace();
        }
        return hints;
    }
```

Lorsque l'exécution de la méthode « doInBackground(…) » est achevée, elle retourne la liste des indices sous la forme d'une chaine de caractères (les indices sont séparés par un saut de ligne : \n). La méthode « onPostExecute(…) » est ensuite appelée.

- ***Le mécanisme utilisé afin de synchroniser l'affichage du hint à l'écran avec le UI thread.***

La méthode « onPostExecute(…) » requiert la valeur retournée par la méthode citée précédemment et crée un pop-up, par le biais d'un fragment, auquel les données sont transmises par la création d'un « Bundle ». Ce dernier est ensuite ajouté en tant qu'argument au fragment.

```
@Override
protected void onPostExecute(String hints) {
    Log.d(TAG, "BeaconsMapActivity.DownloadAsyncTask.onPostExecute() called");

    // create a bundle with the hints and send them to the fragment
    Bundle bundle = new Bundle();
    bundle.putString("hints", hints);

    // create hints dialog
    FragmentManager fm = getFragmentManager();
    HintsDialogFragment dialogFragment = new HintsDialogFragment();
    dialogFragment.setArguments(bundle);
    dialogFragment.show(fm, "Hints list");
}
```

Les données sont ensuite récupérées dans la méthode « onCreateView(…) » du fragment « HintsDialogFragment », avec lesquelles nous créons par la suite une liste. Cette liste est affichée à l'utilisateur, qui a tout loisir de la cacher, en appuyant à un quelconque emplacement à l'écran, en dehors du pop-up.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState){
    ...
    hints = getArguments().getString("hints");
    // get every hint for the current beacon (separate by a line)
    String[] hintsTable = hints.split(System.getProperty("line.separator"));
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_1, hintsTable);
    listHints.setAdapter(adapter);
    return view;
}
```

3 CONCLUSION

Ce travail pratique fut l'occasion de mettre en place l'utilisation du bluetooth et de disposer d'une application quasiment terminée. La méthode de mesure de la distance entre un beacon et l'appareil est probablement erronée étant donnée les mesures obtenues, mais cela facilitera la recherche des beacons.