

TÉLECOM SUDPARIS

Watch'Int
La montre connectée

Auteurs :

Muyao CHEN
Victor FIS
Raymond KUOCH
Nikeethan SELVARATNAM

Tuteur :
Ghalid ABIB

Introduction

Nous sommes actuellement dans un contexte où tous les objets se connectent entre eux. L'image ci-dessous décrit l'évolution rapide de ce secteur.



FIGURE 1 – Schéma descriptif

Alors quoi de plus naturel que de connecter sa montre à son téléphone, afin d'éviter à ce dernier de nombreux aller-retours dans la poche ? Dans une société nécessitant une forte optimisation du temps, cette montre s'avère intéressante du fait de l'économie de temps réalisée pour la lecture de l'heure et de SMS.

Nous nous sommes intéressés à la création d'un prototype de montre connectée : en effet, deux membres de notre équipe maîtrisent le langage C et savent ainsi coder un système Arduino et deux autres membres ont pu pratiquer le langage Java pendant le module “Algorithme et langage de programmation” et ainsi coder une application Android. C'est pourquoi nous avons finalement choisi de combiner le langage C et Java dans notre projet informatique pour coder une montre connectée.

Ce document est le rapport final du projet "Watch'INT", réalisé dans le cadre du module PRO 3600 Développement informatique. Il donne une vision générale de ce qui a été fait au travers de notre progression, des contraintes et difficultés rencontrées, ainsi que de l'évolution du codage. Ce document comporte trois parties à la suite de cette introduction : une présentation du cahier des charges où nous comparerons les objectifs initiaux avec ce qui a finalement été réalisé la description du développement, qui se décompose en 4 parties : une analyse du problème, une conception préliminaire et une conception détaillée, des tests permettant de vérifier la cohérence du code un manuel utilisateur permettant d'utiliser l'Arduino et l'application Androïde Il se termine par une conclusion résumant les points importants. En fin de document, une annexe présente quelques fichiers extraits du code source de l'application, le planning prévisionnel du projet.

Table des matières

1 Cahier des charges	1
1.1 Description de la demande	1
1.1.1 Les fonctions du produit	1
1.1.2 Critères d'acceptabilité et de réception	1
1.1.3 Livrable	1
1.2 Contraintes	2
1.2.1 Contraintes de coût	2
1.2.2 Contraintes de délai	2
1.2.3 Contraintes techniques	2
1.3 Déroulement du projet	3
2 Développement	4
2.1 Analyse du problème et spécification fonctionnelle	4
2.2 Conception préliminaire	6
2.2.1 Android	6
2.2.1.1 Définition d'un modèle de données	6
2.2.1.2 Définition d'un format de fichier associé au modèle de données	6
2.2.1.3 Définition des modules	6
2.2.1.4 Définition des fonctionnalités	6
2.2.2 Arduino	8
2.3 Conception détaillée	9
2.3.1 Android	9
2.3.2 Arduino	13
2.4 Codage	17
2.5 Tests	17
2.5.1 Tests unitaires	17
2.5.2 Tests d'intégration	17
2.5.3 Tests de validation	17
3 Manuel utilisateur	18
4 Conclusion	23
Annexes	25
Annexe 1 : Plan de charge	25
Annexe 2 : Exemple de compte rendu	26

Partie 1

Cahier des charges

Il s'agit de créer une montre connectée à un smartphone Android lors de ce projet informatique. Une application Android sera créée afin de pouvoir se synchroniser et communiquer avec la montre, qui pourra afficher l'heure, la date du téléphone et prévenir l'utilisateur dès qu'un SMS sera reçu. La montre sera créée à base de matériel Arduino (carte microcontrôleur). L'utilisateur pourra synchroniser à tout moment son Smartphone avec l'Arduino.

1.1 Description de la demande

1.1.1 Les fonctions du produit

Deux parties sont à distinguer : développement Android et développement Arduino. Les fonctionnalités qui devront être offertes par chacune des deux parties sont :

1. La partie Android :
 - (a) Etablissement de la connexion bluetooth avec l'Arduino
 - (b) Extraire des SMS du smartphone puis envoyer une notification à l'Arduino en cas de réception
 - (c) Extraire l'heure et la date du smartphone puis les envoyer à l'Arduino
 - (d) Créer une interface graphique permettant d'envoyer des commandes vers la montre
2. La partie Arduino :
 - (a) Etablissement de la connexion bluetooth avec le smartphone
 - (b) Récupérer les informations reçues par bluetooth une fois la connexion établie
 - (c) Afficher ces informations ou les interpréter (par exemple pour une sonnerie)

1.1.2 Critères d'acceptabilité et de réception

Différents critères de performance doivent être respectés par l'Arduino et l'application Android :

1. L'application Android doit en effet être robuste en cas de :
 - (a) Terminaison imprévue
 - (b) Erreurs d'entrée par l'utilisateur
 - (c) Bluetooth non supporté par le téléphone
2. L'application Android doit être performante (interaction entre l'utilisateur et l'application en temps réel sans délai)
3. Optimisation de la consommation énergétique de la montre connectée qui dépend des codes Android et Arduino utilisés

1.1.3 Livrable

Les documents ainsi que les produits suivants seront livrés à la fin du projet :

1. La montre connectée sous la forme d'un Arduino
2. L'application Android utilisée pour synchroniser la montre connectée
3. Le programme de l'Arduino
4. Code source des programmes avec leurs documentations

1.2 Contraintes

1.2.1 Contraintes de coût

Tous les composants de l'Arduino devront être achetés dans le commerce :

1. Carte Arduino Uno R3
2. Module bluetooth HC06
3. Écran Oled



FIGURE 1.1 – Carte Arduino Uno R3



FIGURE 1.2 – Module Bluetooth



FIGURE 1.3 – Ecran Oled

1.2.2 Contraintes de délai

1. Passer la commande de matériels qui dure de 1 semaine à 10 jours
2. Livrable 2 (prototype du logiciel) : 13 mars
3. Livrable 3 (logiciel, tests et rapport) : 22 mai
4. Respecter le planning prévisionnel prévu au début du projet
5. Les parties Android et Arduino doivent finir le code au même moment pour ne pas retarder le projet et ainsi pouvoir effectuer des tests entre les deux systèmes

1.2.3 Contraintes techniques

1. Apprentissage des deux langages de programmation propres à l'Arduino et à Android : C et Java
2. Les programmes devront être le plus clairs et plus structurés possibles (possibilité d'évolution par une tierce personnes)
3. Gestion de la consommation électrique de l'Arduino (économie d'énergie)
4. Mise en place du Bluetooth Low Energy (BLE), présente dès Android 4.3 (API level 18)
5. Utilisation des classes SMSReceiver et BlueoothAdapter sur Android
6. Utilisation de GitHub

1.3 Déroulement du projet

Dans un premier temps, il s'agira d'acheter le matériel nécessaire à la réalisation du projet, c'est à dire la carte Arduino Uno R3, le module bluetooth ainsi que l'écran. Par la suite nous constituerons deux binômes de travail afin de travailler aussi bien sur la partie Android que sur la partie Arduino.

Les différentes parties à traiter pour la programmation de la carte Arduino seront la configuration de la connection Bluetooth, l'affichage de l'heure et la configuration manuelle. Enfin des tests intermédiaires seront réalisés, comme l'affichage de la date et de l'heure ainsi que la connection bluetooth entre deux Arduino.

Pour l'application Android il s'agira de créer une interface graphique qui permettra d'envoyer l'heure du smartphone ainsi que des sms par bluetooth. Cette interface graphique devra figurer une boîte de communication permettant de visualiser les messages s'envoyant entre l'Arduino et l'Android, de plus il y aura également un bouton Send et un List View permettant à l'Android d'envoyer ses propres messages. Enfin, des tests intermédiaires seront également réalisés afin de vérifier la cohérence des différentes méthodes créées, comme par exemple l'utilisation du Bluetooth, la création de Socket, l'extraction des SMS.

Pour finir, différents tests seront réalisés avec les deux systèmes pour vérifier la cohérence des deux codes ensemble. Seront testées la portée maximale, la robustesse et la synchronisation entre les deux appareils.

Partie 2

Développement

2.1 Analyse du problème et spécification fonctionnelle

Le diagramme ci-dessous nous donne une vue d'ensemble du fonctionnement de notre montre. Deux applications permettent son fonctionnement. La première qui est sur le téléphone devra pouvoir récupérer des informations dans ce téléphone pour les envoyer ensuite. La seconde, celle de la carte, devra pouvoir envoyer des notifications et permettre la réinitialisation de la montre.

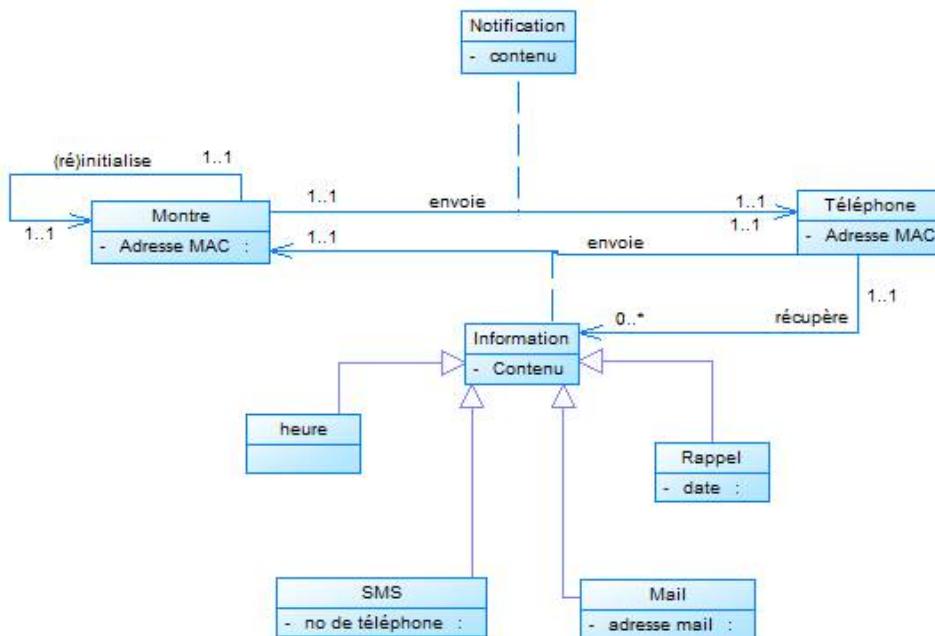


FIGURE 2.1 – Diagramme décrivant le fonctionnement de la montre connectée

Remarque : Le mail a été remplacé par l'appel téléphonique.

Décrivons par la suite les fonctionnalités attendues par chacune des parties Android et Arduino.

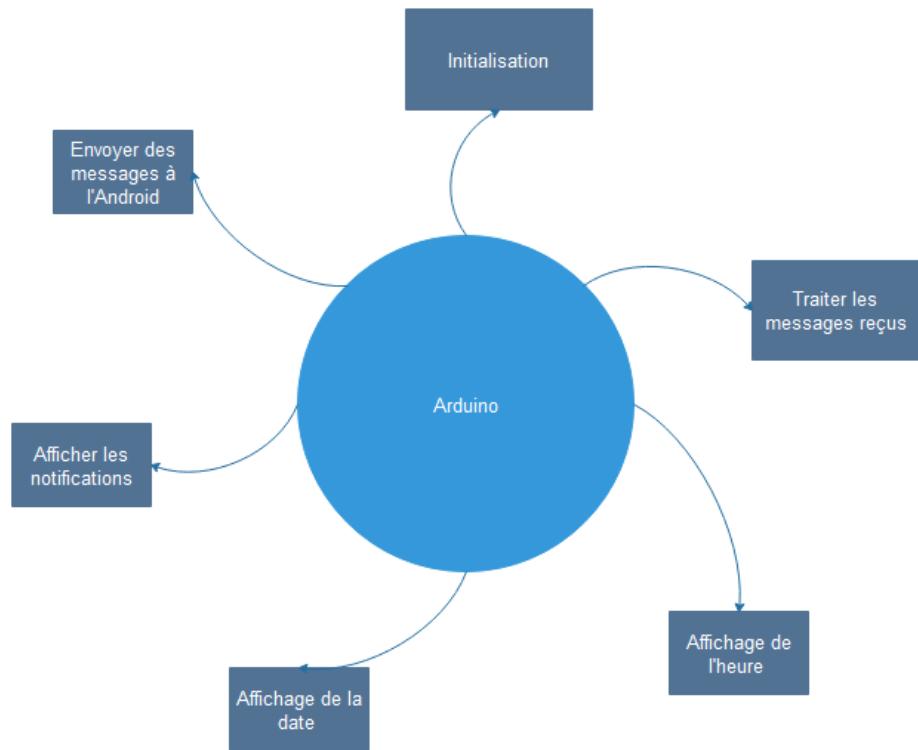


FIGURE 2.2 – Fonctionnalités de l’Arduino

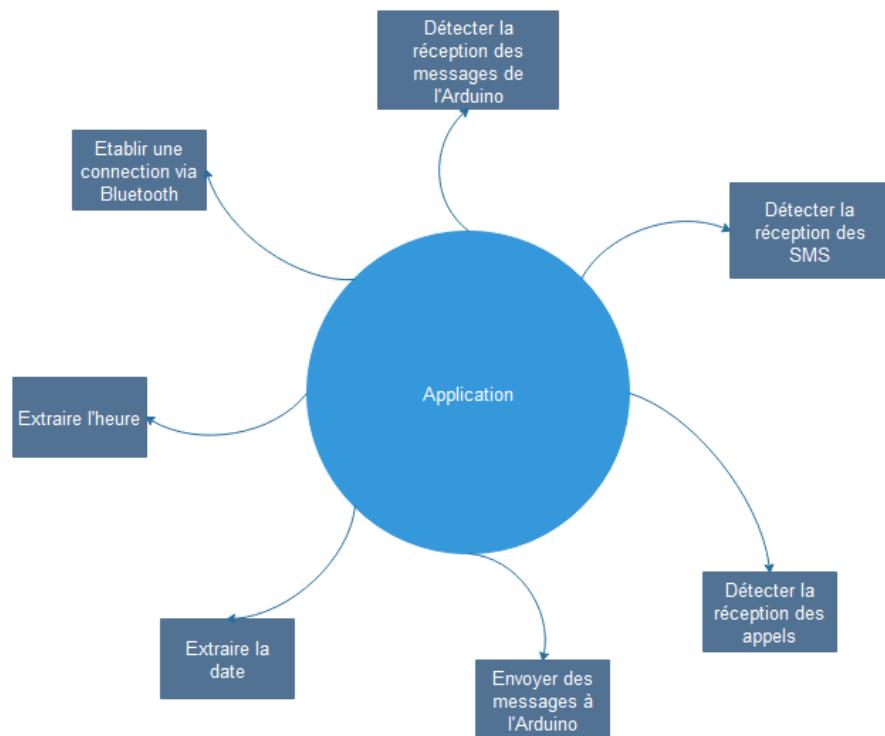


FIGURE 2.3 – Fonctionnalités de l’application Android

2.2 Conception préliminaire

2.2.1 Android

2.2.1.1 Définition d'un modèle de données

Les structures de données qui seront utilisées dans l'application : la liste (chaînée ou double-chaînée), la file (ou queue), le tableau (ou vecteur) : Les données sont organisées de façon orienté objet. Elles sont créées, manipulées par le langage Java. Le stockage de l'information se fait dans la mémoire interne de l'appareil.

2.2.1.2 Définition d'un format de fichier associé au modèle de données

L'application est développée en Java mais un appareil sur Android ne comprend pas le Java. Le fichier APK (Android PacKage) est un paquet contenant les fichiers d'une application (le codage du programme, les ressources, fichiers manifest, certificats etc).

2.2.1.3 Définition des modules

L'application est structurée en 3 classes :

- MainActivity
- SMSReceiver
- CallReceiver

2.2.1.4 Définition des fonctionnalités

Cette partie présente les pistes que nous avons suivi au moment de la réalisation du livrable 1. A ce moment-là nous avions une idée assez précise de ce que nous voulions utiliser. L'idée est de connecter un Smartphone avec un Arduino grâce aux deux modules Bluetooth présents sur les deux appareils. La première étape dans l'intégration du Bluetooth dans l'application est de vérifier si le device en question possède cette technologie. Cela est très facile en utilisant l'API Bluetooth et plus particulièrement le BluetoothAdapter. Cette classe permet d'exécuter les fonctionnalités basiques, comme la vérification de la présence du bluetooth, lancer un scan des devices, etc.

La première fonction que nous chercherons à concevoir concerne le module Bluetooth du Smarphone. Nous pourrons ainsi utiliser les fonctions suivantes :

1. `getdefaultAdapter()`

Elle permet de vérifier la présence du Bluetooth sur le terminal ;

2. `isEnabled() / enable()`

Elles permettent d'activer le Bluetooth si celui - ci n'est pas encore activé ;

3. `getBondedDevices()`

Elle permet de regarder parmi les périphériques que nous connaissons, avant de chercher les périphériques à proximité pour une éventuelle connexion et renvoie ainsi une liste de "BluetoothDevice", chacun étant un périphérique connu ;

4. `startDiscovery() / cancelDiscovery()`

Elles permettent de faire / stopper une recherche des terminaux proches visibles.

La deuxième fonction que nous chercherons à concevoir concerne la création d'une interface de connexion entre l'Arduino et le Smarphone. En effet, pour établir une connexion et échanger des données, il faut un côté serveur (device qui reçoit la connexion) et un côté client (device qui envoie la connexion). Ceci est facile en utilisant la bibliothèque BluetoothSocket :

1. `createBluetoothSocket()`

Elle permet de créer un Socket, établissant l'interface de connexion avec l'Arduino ;

2. `connect()`

Elle permet de commencer la connexion ;

3. `close()`

Elle permet de bloquer tous les demandes de connexion avec d'autre appareils, une fois que la connexion avec l'Arduino s'est faite ;

4. `cancel()`

Elle permet de fermer le socket lorsque l'on quitte l'application.

La troisième fonction que nous allons concevoir est la création du flux de données entre l'Arduino et l'Android. En effet, pour pouvoir communiquer avec l'Arduino, le Smartphone doit créer un flux entrant pour recevoir les messages provenant de l'Arduino et un flux sortant pour transmettre des messages à l'Arduino. Nous pouvons ainsi utiliser la bibliothèque Thread qui permet de réaliser cela :

1. `run()`

Elle permet de lancer le processus de lecture et d'écriture de message ;

2. `write()`

Elle permet d'envoyer des données vers le module Arduino.

Nous voulions extraire la date et l'heure du Smartphone puis l'envoyer à l'Arduino. Pour cela, il existe de nombreuses fonctions permettant de réaliser cela et nous avons opté pour la fonction qui renvoie la date et l'heure actuelle en Millisecondes :

1. `System.currentTimeMillis()`.

Nous voulons également extraire les SMS reçus par le Smartphone puis envoyer des notifications à l'Arduino en cas de réception. Pour cela, il est aisément d'utiliser l'interface BroadcastReceiver. Cette interface ne comporte qu'une seule méthode `onReceive()` que l'on devra définir :

1. `Smsmessage()`

Elle permet d'avoir accès aux derniers SMS reçus par le smartphone ;

2. `getMessageBody()`

Elle permet d'avoir accès au message du dernier SMS reçu ;

3. `getDisplayOriginatingAddress()`

Elle permet d'avoir accès aux informations de l'expéditeur du dernier SMS reçu

Nous voulons également, dès que l'on reçoit un appel, envoyer une notification à l'Arduino. Pour cela, nous allons également la classe BroadcastReceiver et définir différentes méthodes :

1. `onCallStateChanged()`

Elle permet d'envoyer un Toast du type : "CALL FROM Number" dès qu'une personne nous appelle Pour cela également, on aura besoin d'importer les classes PHONE STATE LISTENER et TELEPHONY MANAGER qui nous permettent d'extraire les informations du dernier appel reçu.

La quatrième fonction que nous voulons mettre en place est la création d'un ListView qui permet d'afficher les messages envoyé à l'Arduino / ou depuis l'Arduino. Pour cela, il suffit de coder sur l'activitymain.xml :

- création d'un ListView permettant d'afficher les messages
- création d'un EditText permettant à l'utilisateur du Smartphone d'écrire le message qu'il souhaiterait envoyer à l'Arduino, et ainsi par la même occasion, création d'un bouton Send permettant de réaliser cette action
- Utilisation d'un Handler qui permet de poster des messages dans le ListView qui seront ainsi exécutés par le Thread.

Concernant l'interface graphique que nous voulons incorporer à l'application, nous avons une idée assez précise de ce que nous voulons afficher :

- une ListView permettant d'afficher les messages qui s'envoient entre l'Arduino et l'Android
- Un EditText qui permet à l'utilisateur d'écrire un message qu'il désire envoyer à l'Arduino
- Un bouton Send permettant à l'utilisateur d'envoyer le message
- Un bouton Cancel qui permet à l'utilisateur d'effacer et de remettre à zéro la liste de dialogue

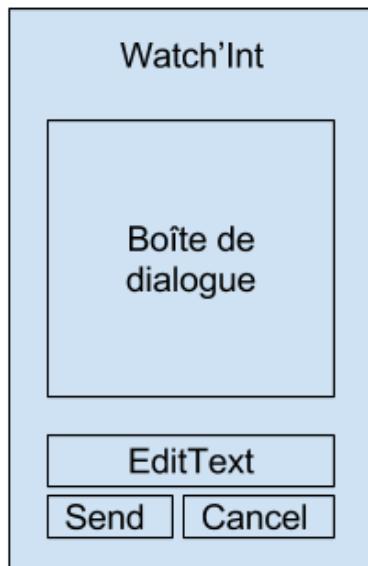


FIGURE 2.4 – Interface graphique

2.2.2 Arduino

Le programme de la partie Arduino comportera deux fonctions principales qui sont initialisation() et une boucle().

La fonction initialisation() fait ce que son nom indique, c'est à dire qu'après avoir initialisé l'état des différentes broches de la carte, il faudra établir la liaison avec le téléphone et récupérer l'heure, sans l'afficher car ceci est fait dans la fonction boucle() pour optimiser la mémoire prise par le programme. Afin de clarifier le code, une fonction sera appelée pour la récupération de l'heure, traitementheure().

La fonction boucle() sera en fait la fonction principale qui décrit le cycle des actions à faire par la montre. Ainsi on y générera la réinitialisation en y testant un bouton, puis on appellera ensuite une fonction, checkinfo(), afin de vérifier la réception ou non d'informations qui peuvent être des sms, des mails ou des rappels (pas l'heure cette fois ci). Enfin sera appelée une fonction mettant à jour l'affichage de l'heure, affichageheure().

2.3 Conception détaillée

Seront détaillées dans cette partie les différents modules dans le but d'avoir une vision détaillée de l'implémentation de notre programme. Nous allons diviser ces explications en 2 parties, la première portant sur l'application Android et la seconde sur le programme Arduino. Dans chaque partie, les différents modules seront présentés par ordre d'apparition.

2.3.1 Android

Cette partie décrira précisément toutes les variables et méthodes utilisées dans le code.

1. Main Activity

> Voici les variables utiles tout au long du programme et initialisées au début.

Type	Nom	Description
Textview	text	
BluetoothAdapter	myBluetoothAdapter	Module Bluetooth du smartphone
BluetoothSocket	btSocket	Socket ie l'interface de connexion entre le Smartphone et l'Arduino
Handler	handler	Intermédiaire
SMSReceiver	smsReceiver	Réception des SMS
String	address	Vaut "20 :16 :11 :29 :58 :02",adresse Mac de l'Arduino
SMSReceiver	smsReceiver	Réception des SMS
ConnectedThread	connectThread	Nécessaire pour la communication via bluetooth
Int	S	S'incrémente lors de la réception d'un SMS
Int	A	S'incrémente lors de la réception d'un appel

FIGURE 2.5 – Tableau récapitulatif des paramètres

> `UUID.fromString`

Permet de récupérer la valeur de l'UUID

> Il faut aussi des permissions afin de définir ces variables, notamment pour accéder au Bluetooth, la réception des SMS et appels :

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

> `void onCreate (Bundle savedInstanceState)`

Cette méthode est appelé au démarrage de l'application (ou si elle a été tuée) d'où le paramètre Bundle qui vaut null s'il s'agit du premier lancement ou si l'application a quitté normalement, et dont la valeur est différente de null si des données ont été sauvegardées. Dans cette méthode, on définit ce qui doit être créé à chaque démarrage, en particulier l'interface graphique grâce à :

```
setContentView(View)).
```

Nous y définissons certaines variables utilisées dans le codage de deux boutons.

Le bouton est un listener, et on souhaite intercepter l'évènement "clic sur le bouton" : on applique l'interface

```
View.OnClickListener.
```

Il faut redéfinir la méthode :

```
void onClick (View view).
```

De plus, il faut associer le listener à une vue, ceci est effectué avec :

```
setOn [evenement]Listener(on[Evenement]Listener listener).
```

En effet, ce listener va se mettre à l'écoute des boutons et déclencher une méthode lorsque l'un d'eux est cliqué.

```
Button clearBtn = (Button) findViewById(R.id.clearBtn);

    clearBtn.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            text.setText("");
        }
    });
});
```

Lors du clic sur le bouton clearBtn, on met dans le input la chaîne vide.

Pour la mise en oeuvre d'un Listener, on utilisera souvent des classes anonymes : elles permettent de passer en paramètre un ensemble de traitements.

Concernant le bouton Send, celui - ci devra mettre en variable le texte présent dans le input (texte que voudrait envoyer manuellement l'utilisateur) puis l'envoyer à l'Arduino en utilisant la méthode remindReceived et enfin l'afficher dans le service de "Chat".

> `public void onResume()`

On définit dans cette méthode ce qui se passe lorsque l'utilisateur est sur l'application.

`DEBUGshowMessage` :

cela est utilisé pour le debug

```
BluetoothDevice arduinoBt = myBluetoothAdapter.getRemoteDevice(address) :
```

Permet d'obtenir un pointeur vers le module Bluetooth de l'arduino

On définit le canal de communication afin de communiquer avec l'arduino, c'est-à-dire l'ouverture d'un socket, la connection puis la création d'un flux de données pour parler avec l'arduino.

— Ouverture du socket :

```
btSocket = createBluetoothSocket(arduinoBt) :
```

le socket BluetoothSocket permet au téléphone de se connecter au module bluetooth de l'Arduino.
Pour s'assurer de sa création, on renvoie sur l'application un message à l'aide de :

```
Toast.makeText(getApplicationContext(), "Succes socket", Toast.LENGTHLONG).show().
```

On n'oublie pas d'appeler cancelDiscovery() car si le téléphone est toujours à la recherche d'un appareil, l'établissement de la connection est beaucoup plus lente, et risque de rater.

— Commencer la connection

```
btSocket.connect() : connect()
```

Permet d'effectuer une tentative de connection. On affiche un toast pour valider cette étape et on affiche "succès connection" pour le debug si la connection a eu lieu.

```
btSocket.close() :
```

Permet de fermer le flux et libère les ressources qui y sont associées.

— Création d'un flux de données pour parler avec le module

```
connectThread = new ConnectedThread(btSocket) :
```

Création du flux

```
connectThread.start() :
```

ouverture du flux

> `onPause()`

Cette méthode définit ce qui se passe lorsque l'application n'est pas utilisée, c'est à dire en arrière plan. Il faut fermer le socket et libérer les ressources utilisées d'où `btSocket.close()` pour économiser de l'énergie.

> Ouverture d'un socket vers un module spécifié

Cela est fait avec

```
createBluetoothSocket(BluetoothDevice device).
```

On renvoie un Bluetooth socket de protocole RFCOMM prêt à commencer une connection non sécurisé vers le module ayant l'uuid donné dans le cas où la version du SDK est supérieure à 10 :

```
if(Build.VERSION.SDKINT >= 10)
    return device.createInsecureRfcommSocketToServiceRecord(MY_UUID)
```

Sinon, on effectue la même action mais la connection est sécurisée cette fois :

```
device.createRfcommSocketToServiceRecord(MY_UUID)
```

On définit aussi la classe TextHandler et la méthode void `handleMessage(Message msg)` qui est nécessaire pour la réception des SMS. Dans cette méthode, on va convertir l'objet Message en String et ainsi l'ajouter au TextView, nécessaire pour visualiser le message reçu.

```
String textStr = (String) msg.obj;
    text.append(textStr+"\n")
```

> Processus de lecture et écriture

Maintenant que l'on a défini les sockets, il faut pouvoir échanger des données. Pour cela on doit savoir ce qui "rentre" et ce qui "sort". On définit alors deux variables contenant ces données :

```
InputStream tmpIn = null
OutputStream tmpOut = null
```

```
socket.getInputStream()
socket.getOutputStream()
```

permettent d'obtenir le flux entrant et sortant.

> Lancement du processus

Il s'agit de la méthode

```
void run()
```

On définit d'abord le buffer pour le flux entrant byte[] buffer = new byte[10] (10 correspond au nombre de caractères maximal que l'on voudrait recevoir, ce nombre étant fixé par l'Arduino). Le nombre d'octets reçu est :

```
int bytes
```

On envoie, dès le démarrage de l'application, l'heure à l'Arduino qui est stockée dans une variable data :

```
String data = "T" + ((int) (System.currentTimeMillis() / 1000L)).
```

(Le "T" en début de message est ajouté afin que celui-ci soit compréhensible par l'Arduino, puis la méthode currentTimeMillis donne un résultat en millisecondes).

On cherche à envoyer un message à l'Arduino du type "I + S + A + R" : S, A et R seront des chiffres compris entre 0 et 9 ; S correspondant au nombre de SMS non lus, A correspondant au nombre d'appels reçus et R correspondant de rappels. I est seulement une lettre qui provient du protocole de communication que l'on s'est fixé avec le groupe s'occupant du codage de l'Arduino. Pour cela, on va initialiser dès le début toutes ces variables et dès que l'on reçoit l'une des trois, on incrémentera la variable associée de 1.

Comme dit précédemment, ces variables ne peuvent aller jusqu'à 9 (on l'a décidé ainsi), et c'est pourquoi on a permis à l'Arduino de pouvoir réinitialiser toutes les variables à 0 quand il le souhaite : c'est le cas lorsque l'Arduino nous envoie le message "C" (pour Cancel).

Il reste à définir la méthode qui permet d'écrire et d'envoyer les données :

```
void write(String message).
```

Elle prend en argument une chaîne de caractère qui est convertie en octets

```
byte[] msgBuffer=message.getBytes()
```

puis envoyé vers l'Arduino grâce à

```
outStream.write(msgBuffer).
```

> Fonctions basiques concernant le Bluetooth

```
void checkBluetooth()
```

Détermine si le Bluetooth est supporté sur l'appareil puis l'activation.

On sait si le bluetooth est activée avec

```
btAdapter.isEnabled();
```

S'il n'est pas activé, on demande son activation grâce à un intent :

```
Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBtIntent, 1)
```

Une fenêtre s'affiche alors sur l'écran pour demander l'activation à l'utilisateur.

```
void visible()
```

Permet de rendre visible l'appareil. Elle utilise aussi un intent :

```
Intent getVisible = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
startActivityForResult(getVisible, 0)
```

2. SMSReceiver

On définit la classe SMSReceiver qui hérite de BroadcastReceiver qui n'a qu'une méthode à définir onReceive qui prend en variable un Context et un Intent : Dès que l'on reçoit un SMS, le bundle permet

de récupérer les informations du SMS. Ainsi si on reçoit un intent nous informant de la réception d'un SMS et si le bundle est non nul (on a pu récupérer les informations), on récupère le message grâce aux extras de l'intent. Puis on récupère dans le bundle , l'extra correspondant à l'identifiant "pdus"
Voici la description de la classe SMS Receiver et son utilisation dans le Main Activity. Elle est nécessaire pour la réception des SMS.

On prend l'activité du MainActivity qui est une sous-classe de Context pour faire le lien grâce à :

```
private MainActivity activity;

public SMSReceiver(MainActivity activity)
{
    this.activity = activity;
}
```

Pour pouvoir utiliser les fonctions liées à SMSReceiver, il faut avoir la permission suivante :

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
```

Concernant l'envoi des informations liés au SMS, nous ne pouvons utiliser la fonction write(data) car elle n'est pas valable dans la classe SMSReceiver : nous allons pour cela ajouter dans une Queue (file d'attente) les informations liés au SMS à l'aide la méthode addMessageToSend() et dès que cette file d'attente est non vide, nous envoyons les informations à l'Arduino et les affichons dans le service "Chat" : message du type "SMS FROM +33....."

```
private volatile Queue<String> messagesToSend;
void smsReceived(String from)
{
    ++S;
    connectThread.addMessageToSend("I" + S + A + "SMS FROM " + from);
```

3. Reception d'un appel

De la même façon, on crée une classe CallReceiver qui hérite de BroadcastReceiver. Dès lors qu'on reçoit un appel, on envoie les informations liées à cet appel à la file d'attente qui envoie ensuite un message à l'Arduino du type : "CALL FROM +33....."

2.3.2 Arduino

- Le premier module à prendre en compte dans cette partie est l'initialisation. Ce module sert principalement à connecter le téléphone à la montre et à recueillir l'heure du téléphone afin que la montre soit synchronisée avec. La procédure est la suivante :

```
procedure initialisation()
    Initialisation des broches
    Initialisation du module bluetooth
    Initialisation du module OLED
    Mise a jour de laffichage sur lecran
    tq(!bluetoothactive)
        attendre
    ftq
    traitementmessagessynchronisation()
    Confirmation de fin dinitialisation par le buzzer
fproc
```

- Dans cette initialisation est appelée une autre procédure permettant de traiter le message de synchronisation entre le téléphone et la montre, ainsi le message attendu est l'heure sous la forme TXXXXXXXXXXX, les X étant des chiffres.

```
procedure traitementmessagessynchronisation()
```

```

tq ( T nest pas detecte)
    attendre
ftq
pctime <-- entier recu
si (pctime>1er janvier 2013)
    synchronise lheure de larduino
fsi
fproc

```

3. Vient ensuite le programme principal qui sera répété en boucle. Il assure la réinitialisation de la montre ainsi que le traitement des informations reçues. Ces informations concernent les sms reçus sur le téléphone.

```

Procedure boucle()
    test du bouton
    si(tempsmaintienbouton>1s)
        reset()
    sinon si(0<tempsmaintienbouton<1s)
        envoie notification
    fsi
    checkinfo()
    delai(0,8s)
    affichagedate()
fproc

```

4. Deux procédures sont appelées dans cette boucle, qui sont checkinfo() et affichagedate(). La première sert à traiter les informations reçues, c'est à dire le nombre de nouveaux mails, sms et les rappels, tandis que la seconde procédure est utilisée pour l'affichage de l'heure. Précisons que checkinfo() contient deux autres procédures qui sont affichagecontenu(), et affichagenotifications(), tandis que affichagedate() contient la procédure affichageheure().

```

procedure checkinfo()
    //recuperer les informations
    //separer les informations
    affichagenotifications()
    si(infosrecues!=infosrecuesprec)
        si(nouveaumessage)
            contenu <-- nouveaumessage
            buzzer de reception de message
    sinon
        contenu <-- rien
    fsi
    fsi
    affichagecontenu()
fproc

procedure affichagedate()
    si(heure synchronisee)
        //eclaircir lecran
        affichageheure()
        //allumer led
    sinon
        //eteindre led
    fsi
fproc

```

5. Détailons maintenant les procédures affichagenotifications(), affichagecontenu() et affichageheure(). Les deux premières procédures servent, comme leurs noms l'indiquent à afficher les notifications, c'est à dire le nombre de sms, mail et rappels, et le contenu, c'est à dire des messages de 20 caractères maximum. La procédure affichageheure() servira à isoler l'affichage de l'heure de la gestion de la led indiquant la synchronisation ou non de la montre avec le téléphone.

```

procedure affichagenotifications(Nb_sms, Nb_mail, Nb_rappel)
    //definir taille du texte
    //definir couleur du texte
    //definir la position du 1er caractere
    //afficher "S : Nb_sms M : Nb_mail R : Nb_rappel"
fproc

procedure affichagecontenu(contenu[])
    //definir taille du texte
    //definir couleur du texte
    //definir la position du 1er caractere
    pour i allant de 1 a 20
        //afficher caractere i
    fpour
fproc

procedure affichageheure()
    //definir taille du texte
    //definir couleur du texte
    //definir la position du 1er caractere
    //afficher heure
    affichage(minute)
    affichage(heure)
    //definir la position du 1er caractere de la 2nde ligne
    //afficher "jour / mois / annee"
fproc

```

La procédure affichage() présente dans affichageheure() sert à placer un 0 devant les minutes ou les secondes lorsque celles-ci sont inférieures à 10. Cela permet par exemple d'afficher 12 :04 :05 au lieu de 12 :4 :5 lorsqu'il est midi, 4 minutes et 5 secondes.

```

procedure affichage(nombre)
    //afficher ":" 
    si(nombre<10)
        //afficher "0"
    fsi
    //afficher nombre
fproc

```

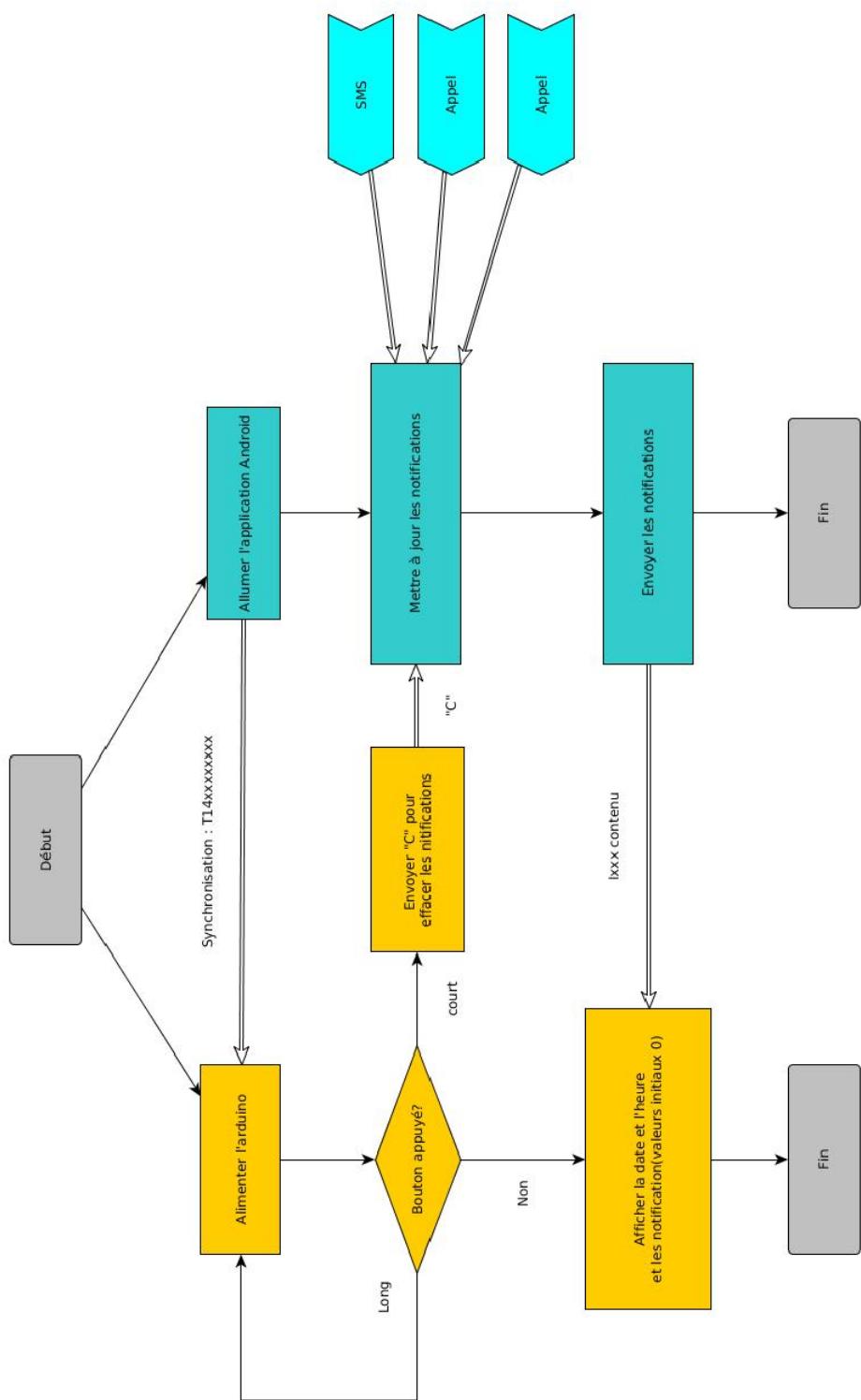


FIGURE 2.6 – Schéma de synthèse

2.4 Codage

La montre sera réalisé à l'aide d'une carte Arduino uno, ainsi le langage utilisé pour mettre en oeuvre notre application sera le C, et nous utiliserons Arduino IDE pour coder. L'application est codée en Java et le code est en annexe. Voici une description du fonctionnement de la montre connectée :

2.5 Tests

2.5.1 Tests unitaires

Tout d'abord c'est le fonctionnement du module bluetooth que nous avons testé. Pour ce faire nous avons utilisé bluetooth terminal, une application android permettant d'envoyer du texte par bluetooth. Le principe de ce test est d'envoyer des caractères et chaînes de caractères à l'Arduino qui ensuite enverra ce qu'il reçoit à un ordinateur via une liaison série. Le test est validé si ce qui est reçu sur l'ordinateur est identique à ce qu'envoie le téléphone.

Nous avons aussi testé le traitement de la réception d'informations, notamment la réception des notifications. Ce test se fait en simulant la réception d'informations, que nous envoyons dans la fonction testée pour vérifier la validité du résultat.

Pour la partie Android, JUnit est utilisé pour réaliser ces tests notamment la fonction write (String data) qui permet d'envoyer les messages vers l'Arduino et la fonction StringOf. Pour le test de write(String data), la OutputStream est modélisé par un ByteArrayOutputStream ; pour la méthode StringOf on met en entrée un message converti en bytes et on vérifie que la sortie est correcte.

2.5.2 Tests d'intégration

La synchronisation de la montre a été testée en utilisant la liaison série de l'arduino et en se passant de module bluetooth. Un programme bash permet d'envoyer l'heure via la liaison série, cette dernière nous permettant ensuite de lire l'heure de la montre en boucle afin de vérifier qu'elle est correcte.

2.5.3 Tests de validation

Les tests de validation ont consisté à tester l'application Arduino avec un terminal bluetooth sur le téléphone puis avec l'application Android créée. L'application Android a été avec la montre.

Par exemple, nous avons testé la portée maximale qui est de 8-10m. La robustesse du système provient du fait qu'on ne se connecte uniquement qu'à l'Arduino et à aucun autre appareil.

Partie 3

Manuel utilisateur

L'application permet à l'utilisateur de se connecter à sa montre connectée (ici représentée par l'Arduino). Nous allons expliquer les différentes étapes à suivre afin de faire fonctionner la montre connectée :

1. Alimenter l'Arduino en le connectant le port USB à l'ordinateur.

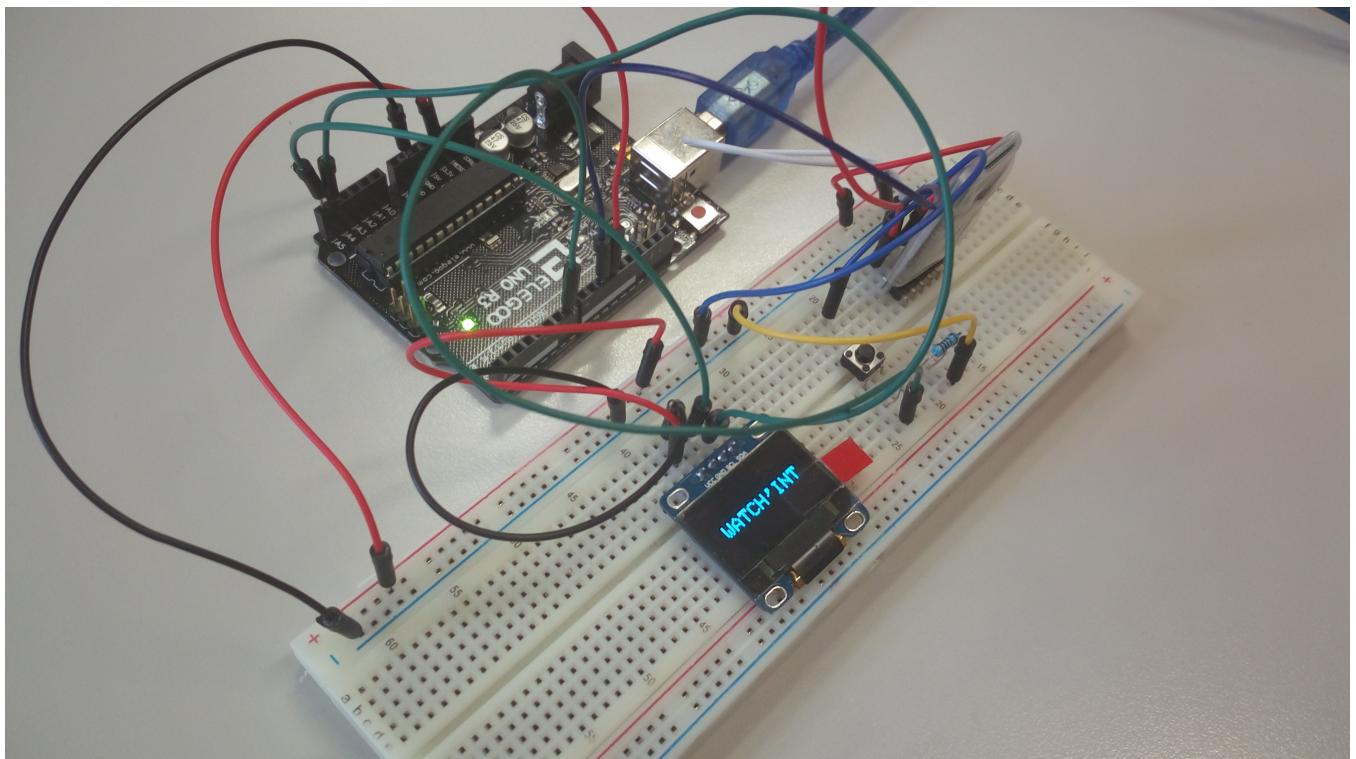


FIGURE 3.1 – Connection du module Arduino à l'ordinateur

2. Allumer l'application Android : Dès le démarrage et dès qu'il sera connecté à l'Arduino (instantanément), il envoie une série de chiffre correspond à la date et l'heure du système.

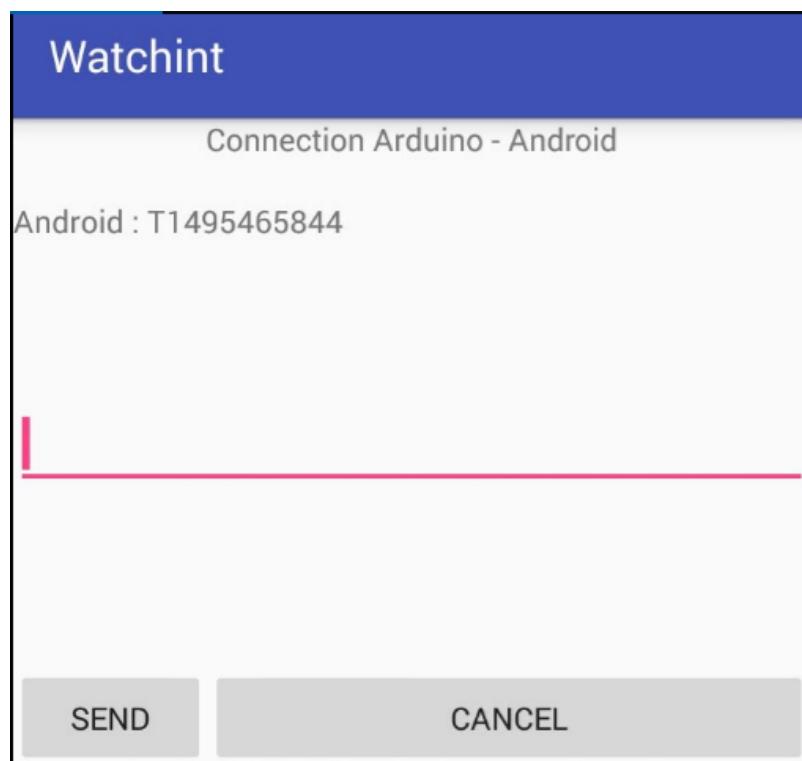


FIGURE 3.2 – Envoi de la date et de l'heure

3. Les SMS sont maintenant instantanément transmis à la montre, ainsi que leur nombre sous forme de notifications.

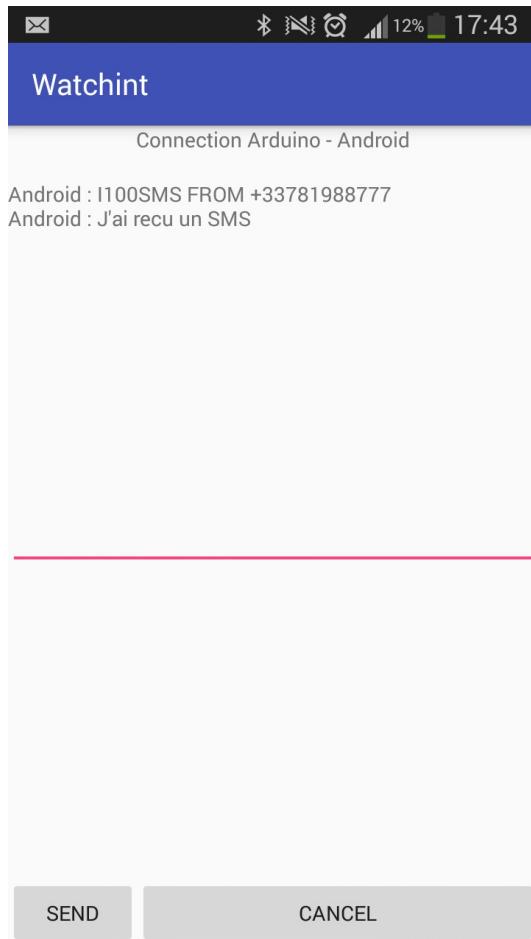


FIGURE 3.3 – Réception d'un SMS

Le message I100 est automatiquement envoyé à l'Arduino suivi du numéro de l'expéditeur. I100 signifie qu'il y a 1 SMS reçu, 0 Appel et 0 Rappels.



FIGURE 3.4 – Ecran LCD lors de la réception d'un SMS

4. Des rappels peuvent être envoyés manuellement : pour cela, l'utilisateur inscrit son message dans l'emplacement puis appuie sur le bouton Send. Par exemple, nous allons envoyer le message "Test pour projet".

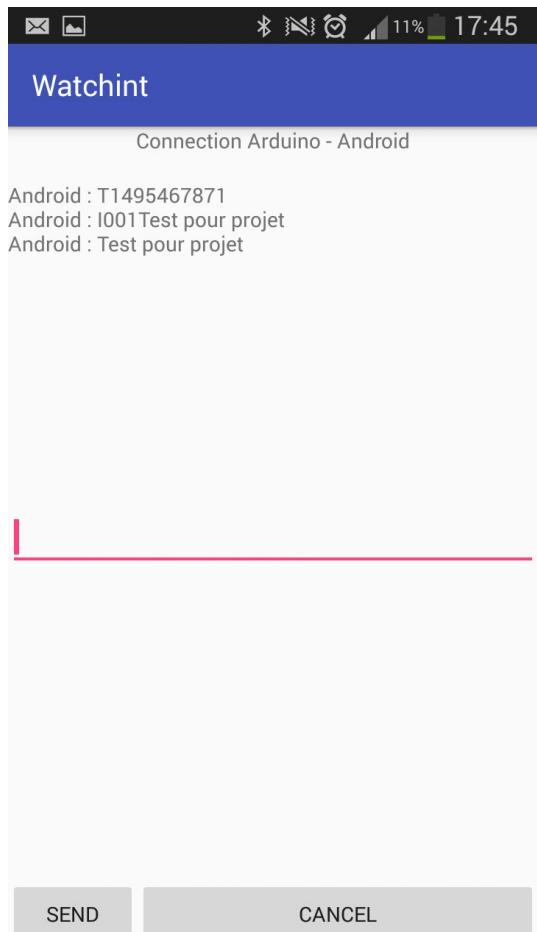


FIGURE 3.5 – Réception d'un rappel manuel

On envoie le message I001 signifiant qu'il y a eu un rappel envoyé.



FIGURE 3.6 – Ecran LCD lors de la réception d'un rappel

5. Un appui court sur le bouton de l'Arduino permet de réinitialiser le nombre de notifications.

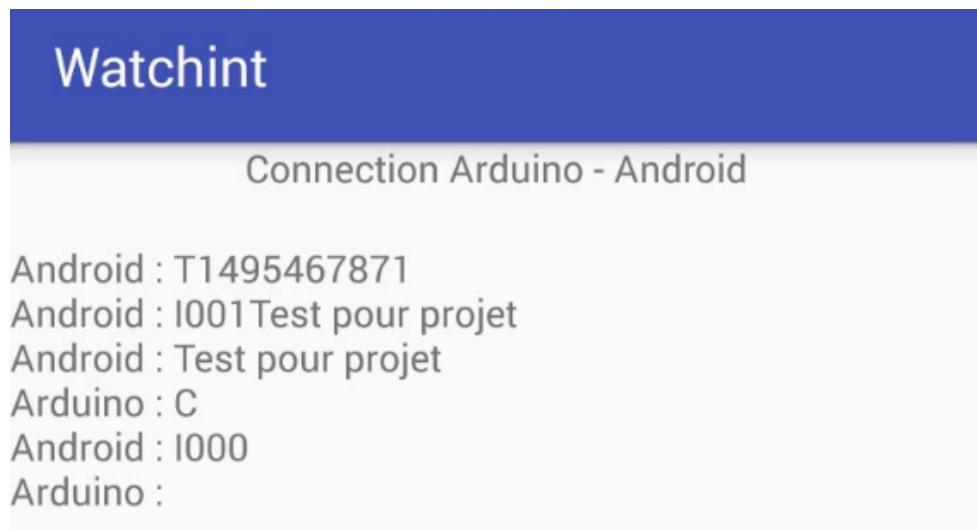


FIGURE 3.7 – Appui court sur le bouton de l'Arduino

Un appui long permet de réinitialiser la montre.

Partie 4

Conclusion

Les objectifs que nous nous étions fixés ont globalement été atteints, bien que seuls les sms et la date aient pu être récupérés. En effet la récupération des mails et des rappels s'est avérée plus complexe et d'autres problèmes ont pris la priorité.

Parmi les problèmes que nous avons rencontrés, nous avons notamment eu des difficultés à établir une connexion fiable entre la montre et le téléphone, ce pour des raisons matérielles. En effet cette connexion pouvait être plus ou moins fiable selon le téléphone utilisé.

Ce projet informatique fut très intéressant tant dans l'apprentissage d'un langage informatique que dans la mise en pratique de la gestion de projet, dont nous n'avions pas utilisé tous les éléments dans le projet GATE. De ce fait nous avons notamment appris à utiliser Github et l'utilisation de cette plateforme a été enrichissante.

D'autres extensions seraient envisageables, par exemple pour améliorer le confort d'utilisation de l'application :

1. Envoyer des notifications lors de la réception d'un mail (configuration et utilisation d'Internet lors du codage Android)
2. Utiliser les commandes vocales pour obtenir les informations recherchées.
3. Recevoir des bulletins météorologiques en temps réel (utilisation de l'appel de Services Web SOAP permettant de connaître la météo du jour)
4. Recevoir des rappels du calendrier et d'autres sources (utilisation de la bibliothèque Calendar)
5. Planifier les déplacements et trouver le chemin le plus court entre un point de départ et un point d'arrivée (nécessite l'utilisation de l'algorithme de Dijkstra et de l'API Google Maps)

Ces extensions n'ont pas été implantées pour l'instant. Mais il serait intéressant de continuer ce projet informatique en essayant de mettre en place ces différentes fonctionnalités.

Annexes

Annexe 1

WATCH'INT		PLAN DE CHARGES PRÉVISIONNEL						SUIVI D'ACTIVITÉS (Charge Consommée)					
Description de l'activité	Charge en %	Charge en H	Charge en H / Participant				Charge en %	Charge en H	Charge en H / Participant				
			Raymond	Muyao	Victor	Nikeethan			Raymond	Muyao	Victor	Nikeethan	
Total	100%	282	73	68	68	73	118%	333	89	83	76	85	
Gestion de projets	30%	84	21	21	21	21	32%	105	26	28	24	27	
Réunion de lancement	1%	4	1	1	1	1	1%	3	1	1	1	0	
Planning prévisionnel et Suivi d'activités	11%	32	8	8	8	8	11%	38	10	10	9	9	
Réunions de suivi	9%	24	6	6	6	6	9%	30	8	8	7	7	
Rédaction	4%	12	3	3	3	3	7%	23	5	5	5	8	
Outils collaboratifs (svn, etc.)	4%	12	3	3	3	3	3%	11	2	4	2	3	
Spécification	3%	8	2	2	2	2	4%	13	4	3	3	3	
Définition des fonctionnalités	3%	8	2	2	2	2	4%	13	4	3	3	3	
Conception préliminaire	13%	36	11	11	11	11	9%	30	8	9	9	11	
Définition d'un modèle de données	3%	8	2	2	2	2	1%	4	2	0	0	2	
Définition d'un format de fichiers associé au modèle de données	4%	10	3	2	2	3	3%	9	2	2	2	3	
Définition des fonctionnalités	5%	14	3	4	4	3	4%	13	2	4	4	3	
Définition des modules	1%	4	3	3	3	3	1%	4	2	3	3	3	
Conception détaillée	25%	70	17	18	18	17	29%	96	29	22	17	28	
Définition des classes	2%	6	3	0	0	3	2%	6	3	0	0	3	
Définition des méthodes	7%	20	5	5	5	5	8%	25	10	5	5	5	
Définition des tests unitaires	4%	12	3	3	3	3	5%	15	3	5	4	3	
Auto-formation	9%	24	4	8	8	4	10%	33	5	10	6	12	
Maquettage des interfaces	3%	8	2	2	2	2	5%	17	8	2	2	5	
Codage	14%	39	12	9	9	12	14%	47	12	12	15	8	
Codage des classes	1%	4	2	0	0	2	1%	4	2	0	0	2	
Codage des méthodes	7%	21	6	6	6	6	7%	22	6	7	5	4	
Codage des tests unitaires	5%	14	4	3	3	4	6%	21	4	5	10	2	
Intégration	6%	16	5	3	3	5	5%	15	5	4	3	3	
Intégration des modules	3%	8	2	2	2	2	2%	6	2	2	1	1	
Tests d'intégration	3%	8	3	1	1	3	3%	9	3	2	2	2	
Soutenance	6%	18	5	4	4	5	6%	20	5	5	5	5	
Préparation de la soutenance	5%	14	4	3	3	4	5%	16	4	4	4	4	
Soutenance	1%	4	1	1	1	1	1%	4	1	1	1	1	

FIGURE 4.1 – Plan de charge

Annexe 2

Réunion d'avancement

Lieu : Laboratoire bâtiment A 3ème étage

Date : 18/05/2017 à 14h

Durée : 1h20

Présents : M. Abib, Raymond, Muyao, Victor, Nikeethan

Ordre du jour :

1. Présenter l'avancement du projet
2. Démonstration des derniers tests

Informations échangées :

1. Discussion sur les test unitaires
 - > Partie Android : M.Abib nous a expliqué ce qui était attendu des tests unitaires
 - > Partie Arduino : Vérification des tests unitaires
2. Démonstration des derniers tests
 - > Test envoi de la date et l'heure - validé
 - > Test envoi d'un rappel manuel - validé
 - > Test de la réception d'un SMS - non validé Le problème est que lors de la réception d'un SMS, I100 est envoyé vers l'Arduino mais il ne s'affiche rien sur l'écran. Il faut appuyer sur le bouton de l'Arduino afin de recevoir les informations du SMS. Il faut revoir l'algorithme de l'Android afin de corriger cela (une erreur possible dans la boucle while du run).
3. Discussion sur le livrable M.Abib nous a demandé d'envoyer le livrable dès que possible pour une vérification.

Todo List :

- > Corriger l'erreur au niveau des SMS
- > Tests unitaires à compléter
- > Rédaction du livrable final

Bibliographie

- [1] Google API. <<https://developer.android.com/guide/topics/connectivity/bluetooth.html>>.
- [2] Arduino. <<https://www.arduino.cc>>.