DUBREUIL Raphaël
VERNET Antoine

RAPPORT PROJET DE FIN D'ETUDES
BORNE INTELLIGENTE HETEROGENE

Tuteur de projet : Hossam Afifi
Télécom SudParis – VAP RSM 3e Année – 2011-2012

Sommaire

I. Introduction

Ce projet, proposé en P.F.E. à la voie d'approfondissement R.S.M. fait partie intégrante d'un projet beaucoup plus vaste commandé par la mairie de Montreuil : Wonderville. Le but avoué du projet est de proposer une connexion sans fil et disponible partout dans la ville de Montreuil pour ses riverains. Ces derniers auraient alors accès par l'intermédiaire d'un portail captif à toutes les informations locales (niveaux de pollution en temps réel, places de parking disponibles...) et à des fonctionnalités de paiement spécifiques à certains services locaux (réservation et achat de sandwich à la cafétéria, de billets de cinéma...). Mais ce qui rend ce projet unique est l'objectif écologique qui lui est imposé. En effet, parmi les impératifs figurent un réseau de bornes qui doit à terme consommer aussi peu d'énergie que possible. Tout doit donc être pensé pour limiter et surtout optimiser les échanges entre les différentes entités du réseau, et notre projet se situe spécifiquement à ce niveau. Nous avons pour objectif de chercher à optimiser les échanges responsables de la mise à jour des données locales, notamment en faisant appel aux connexions haut débit des utilisateurs, qui ne font pas partie du réseau de bornes.

Voici l'intitulé du projet, tel qu'il nous a été proposé au départ :

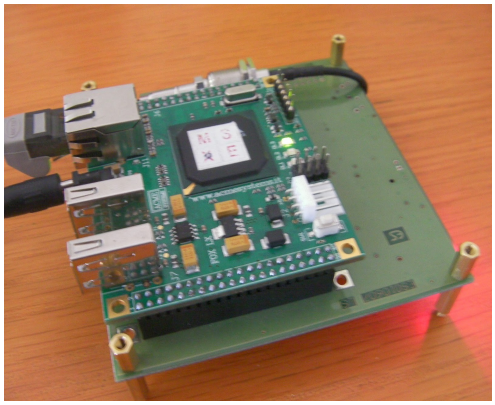
Le but du projet est de continuer le développement sur la borne miniature du département RS2M pour étendre ses capacités à des scénarii où le client s'approche de la borne sans fil, récupère des informations locales comme les horaires de bus, l'état du trafic, mais aussi il contribue à alimenter la borne par des informations provenant de l'internet ; ce qui veut dire que la borne n'est pas connectée à l'internet mais qu'elle obtient sa connaissance soit localement, soit en provenance des clients qui passent à côté.

II. Première approche

La première phase du projet a été pour nous de nous familiariser avec le sujet, à savoir chercher à comprendre derrière les objectifs demandés quelles technologies et quels matériels ou équipements étaient engagés. Nous avons également dû nous renseigner sur l'avancement du projet, c'est à dire ce qui avait déjà été réalisé avant notre arrivée.

Comme le projet commandé par la ville de Montreuil est partagé en un grand nombre de plus petits projets, nous nous sommes réunis dès la première semaine avec d'autres étudiants concernés afin de mieux comprendre notre rôle à chacun, de pouvoir fixer les limites de notre sujet de P.F.E. et surtout de pouvoir prendre connaissance le plus vite possible des logiciels et langages dont nous allons avoir besoin pour réaliser nos objectifs. La première réunion nous a permis de voir à quoi ressemblait typiquement une des bornes sur lesquelles nous devrons travailler par la suite. Nous avons également appris diverses informations sur la manière dont ces bornes fonctionneraient :

- Les bornes sont compatibles avec de nombreuses technologies sans fil, ce qui permet potentiellement à presque tous les utilisateurs munis de téléphone portable de pouvoir s'y connecter. Les bornes seront en effet munies de cartes Wi-Fi, de cartes 3G , de cartes NFC et de cartes Bluetooth. Moyennant une compatibilité des applications présentes sur la borne, l'utilisateur pourra choisir lui-même la manière dont il souhaite s'y connecter.
- Les bornes seront indépendantes, c'est à dire que 2 bornes ne dialogueront jamais directement l'une avec l'autre. Les échanges se feront à travers un intermédiaire, qui sera autant que possible un riverain muni d'une connexion 3G.
- La connexion d'un utilisateur à une borne sera entièrement gérée par cette dernière, et une adresse IP sera fournie au smartphone demandeur directement par la borne. Cette adresse IP sera localisée comme appartenant au sous-réseau de la borne, chaque borne ayant un sous-réseau qui lui est propre.



Architecture matérielle d'une borne

- Toutes les informations locales, qu'elles soient récupérées par des capteurs (niveau de pollution par exemple) ou fournies par des utilisateurs seront à terme stockées dans un serveur central situé à la mairie et connu de toutes les bornes. Régulièrement, les bornes auront à envoyer et récupérer des informations sur ce serveur central

Au cours de cette réunion, nous avons également été en mesure de déterminer avec plus de précision quels étaient nos objectifs et quels moyens nous avions à disposition. Alors que les autres étudiants présents devaient travailler sur le NFC, nous devions travailler sur une application Smartphone destinée à créer un transfert de données entre deux entités extérieures. Il a très vite été décidé que cette application sera codée à l'aide du langage Android, puisque c'est le plus répandu sur les terminaux de type Smartphone, et donc celui qui sera compatible avec la majorité des

utilisateurs.

Avant de se lancer dans la réalisation du projet, nous préférons effectuer par nous même quelques tests avec une borne et nous renseigner un minimum à propos du langage Android. La borne nous est fournie mais, ne possédant que des ordinateurs fixes sans carte Wi-Fi, nous effectuons les tests en filaire.

- Nous branchons donc un câble Ethernet entre notre poste et la borne et configurons manuellement une adresse IP appartenant au sous-réseau de la borne (typiquement en 192.168.1.xxx, les bornes de test ayant une adresse comprise entre 192.168.1.1 et 192.168.1.5 – et un masque de sous-réseau de 255.255.255.0)
- Nous ouvrons le terminal et lançons le protocole ssh entre les deux entités (muni du mot de passe associé), en se connectant en tant que 'root'.
- En tant qu'administrateur, nous avons tous les droits et voyons que les commandes UNIX de base marchent correctement (cd, touch, cp, rm...).
- La borne est munie de quelques fichiers de démonstration, notamment une base de données et quelques pages rattachées à un portail captif. Nous parvenons à les ouvrir et à naviguer facilement dedans.

Finalement ce premier contact avec la borne est positif, la connexion est très simple à réaliser – du moins en filaire, et les fichiers de démonstration sont bien réalisés et fonctionnent correctement. Il nous a seulement fallu quelque temps pour retrouver le bon répertoire car l'organisation des fichiers dans la borne est un peu brouillonne.

Concernant le langage Android, il s'agit comme chacun sait d'un langage de programmation développé par Google pour les applications sur Smartphone. Il est assez proche du langage Java, sur lequel il est basé, ce qui est un point positif pour nous car nous avons un certain nombre de connaissances sur ce langage. Cependant, la philosophie d'Android est quand même bien différente de celle de Java et nous devons absolument prévoir une période d'apprentissage du langage. A ce sujet, nous avons demandé à Mr.Afifi s'il pouvait nous montrer un exemple de projet codé en Android. C'est dans ces circonstances que nous avons eu accès à un projet de 2010-2011 en Android sur l'IP TV. Ça sera certainement un bon point de départ pour notre projet.

III. Description de l'application

Nous avons pris quelques jours pour fixer les choses et créer notre 'cahier des charges', celui qui va décrire quelles devront être les fonctionnalités de notre application ainsi que les contraintes à respecter. Ce cahier des charges ne détaillera pas les aspects techniques mais décrira le déroulement des opérations depuis le lancement de l'application jusqu'à sa clôture. Bien entendu, des modifications pourront être effectuées par la suite par d'autres contributeurs du projet, en fonction de l'évolution des besoins.

L'application que nous allons programmer devra être compatible pour tous les terminaux Android. Elle sera activable par l'utilisateur depuis sa liste d'applications. À l'ouverture de l'application, un message automatique apparaîtra et demandera à l'utilisateur s'il accepte de fournir du débit au programme Wonderville via sa connexion 3G afin de créer un échange de données entre la borne à laquelle il est connecté, et le serveur central. Dans cette première version, comme la connexion est à l'initiative de l'utilisateur, l'adresse IP de la borne sera déjà intégrée au programme. L'utilisateur doit pouvoir avoir la possibilité de refuser de fournir du débit, et dans ce cas l'application va se fermer automatiquement sans avoir effectué le transfert de données.

Dans le cas où l'utilisateur a donné son accord pour prêter sa connexion 3G, un échange de données doit alors se lancer.

Dans un premier temps, l'application doit récupérer un certain nombre de données présentes sur la borne à laquelle elle se connecte. Il est à noter que les critères de sélection de ces données seront décidés ultérieurement. Pour cette version, nous considérons également l'identifiant de la borne, c'est à dire son adresse IP comme connue et donc comme étant un champ pré-rempli dans le code source. Les données seront ensuite récupérées sur le terminal de l'utilisateur et stockées temporairement.

Dans un second temps, l'application doit transférer ces nouvelles données au serveur central situé à la mairie. L'application doit donc créer une requête afin de pouvoir mettre à jour la base de données (il peut s'agir d'ajout de nouvelles données ou alors de mise à jour de données préexistantes). L'adresse du serveur central étant unique, elle pourra aisément être intégrée au code source.

Pendant cette période de transfert de messages, il est également prévu que l'utilisateur ne puisse pas interrompre l'application, et qu'un message indiquant que l'échange de données a été effectué avec succès apparaisse une fois le transfert terminé.

L'application devra être aussi modulaire que possible, afin de permettre à des personnes extérieures au projet de pouvoir contribuer facilement à son amélioration. Elle sera également suffisamment commentée pour que sa compréhension soit rendue plus facile.

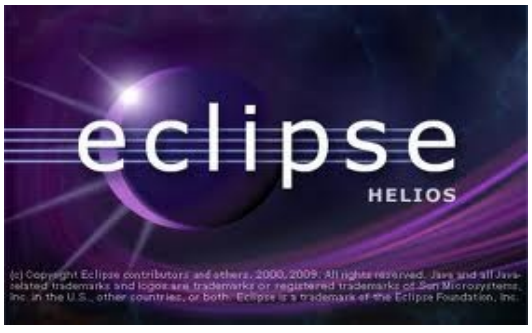
Nous verrons également par la suite que notre projet ne fait pas seulement appel à des notions de type Android ; car en effet au code Android s'ajouteront des pages PHP, ainsi que de la gestion de base de données en SQL.

IV. Installation – Premiers pas

La phase d'organisation en vue de démarrer le codage de l'application débute. Afin de pouvoir se répartir le travail sans aucune restriction, il faut harmoniser le choix des systèmes et logiciels que l'on prévoit d'utiliser.

Le premier choix à effectuer concerne le système d'exploitation que nous allons utiliser. Il parait de premier abord évident que nous devons privilégier une distribution Linux, puisqu'elle nous facilitera largement la tâche lorsque nous aurons à tester des échanges avec la borne. En effet, ce type d'échange n'est pas du tout standardisé ni adapté pour une distribution Windows. Le pôle RS2M nous fournit alors un CD d'installation d'une ancienne version de Debian. Nous devons faire face à un certain nombre de problèmes. En effet nous avons à gérer une interface d'installation très austère et ne sommes pas très sûrs de tous les paramètres entrés. De plus, après avoir réussi l'installation, nous devons faire face à un problème d'affichage qui est apparemment lié à la résolution de l'écran. Face au manque d'information et d'aide concernant ce problème sur Internet, nous décidons finalement d'utiliser Ubuntu. L'installation se passe sans problèmes et nous pouvons ainsi passer à l'étape suivante.

Nous avons ensuite besoin d'un environnement de développement pour pouvoir coder en Android. Après quelques recherches sur Internet, nous remarquons que la plupart des développeurs Android utilisent Eclipse, que nous connaissons déjà puisque nous avons déjà réalisé des projets Java à l'aide de cet IDE. Cet IDE est donc le choix idéal pour nous. L'installation de l'IDE comprend donc l'installation du logiciel Eclipse lui-même et l'installation du JDK - ou Java Development Toolkit. Le JDK contient d'une part le JRE (c'est à dire les composants nécessaires au lancement d'applications Java ainsi qu'une machine virtuelle Java), et d'autre part un ensemble d'outils pour compiler et débugger le code. A cela s'ajoutent un SDK spécifique à Android qui nous fournira les outils pour coder en Android, et l'installation d'un émulateur Android, qui réagira à nos programmes de la même manière qu'un Smartphone, ce qui nous permettra de faire nos tests.



Le langage Android étant finalement relativement proche du langage Java, nous décidons de créer une application Java ayant les mêmes fonctions que celles que nous devons rendre à l'issue du projet. Puis nous n'aurons plus qu'à adapter cette application aux spécificités du langage Android.

Pour créer cette application, nous devons simuler les 3 entités qui interagiront, à savoir l'utilisateur muni de son Smartphone, la borne, le serveur central.

- Après quelques tests, nous comprenons que l'émulateur Android se comporte comme une entité indépendante de l'ordinateur sur lequel il tourne, et qu'il est muni de sa propre adresse IP. L'émulateur pourra donc tenir le rôle de l'utilisateur avec le plus grand réalisme (et en respectant les contraintes du cas réel).
- La borne et le serveur central devront impérativement être représentés par 2 ordinateurs distincts, et ils devront être munis de base de données de structures communes afin de rendre l'échange possible.

Pour cela, plusieurs solutions sont possibles, et l'une d'elles extrêmement simple consiste à installer le pack Xamp qui contient un serveur web php (qui facilitera la gestion des droits d'accès), et une base de données MySQL. Nous disposons alors de tous les outils nécessaires à la réalisation de cette première version de l'application.

Une autre solution est assez simple sous Linux : elle consiste à installer le serveur Web Apache, le pack PHP puis une base de données MySQL (ex : PHPMyAdmin). Dans la pratique cela se fait dans le terminal avec la commande « $ sudo apt-get install --nom-de-l-application-- ».

V. Un Programme en Java

Remarque :

- le code présenté dans cette partie n'a pas été retenu pour le projet mais il nous a permis de nous faire prendre conscience de certains points à considérer par la suite.
- le code dont il est question dans cette partie peut être trouvé dans les annexes (chapitre XI, paragraphe 5)

Le programme que nous avons tout d'abord codé en Java est une version simplifiée de notre projet. En effet, le principal objectif que nous visons avec ce programme est de constater un échange effectif de données entre 2 ordinateurs. En pratique, nous allons lancer un programme Java sur l'une des deux machines qui exécutera 3 types de requêtes SQL (des requêtes SELECT, INSERT, et UPDATE) sur une base de données installée sur la machine distante. Nous aurons ainsi l'occasion d'identifier les problèmes susceptibles d'être rencontrés tout au long du processus.

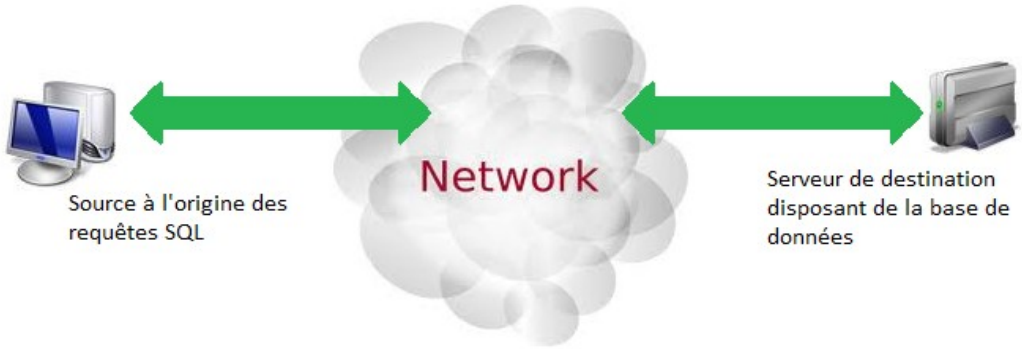


Schéma architectural du programme Java

Cependant afin de pouvoir manipuler des bases de données depuis un programme Java, nous avons besoin d'installer une interface entre Java et la base de données. Cette interface s'appelle JDBC - ou Java Database Connectivity. JDBC permet théoriquement aux applications Java d'accéder à des sources de données pour lesquelles il existe des pilotes relatifs à cette interface. JDBC est muni de toute une batterie de fonctions que nous devons également apprendre à utiliser.

Le programme doit intégrer le pilote JDBC afin de pouvoir importer les librairies et donc fonctions nécessaires. Ce pilote peut se trouver à l'adresse suivante : developpers.sun.com/product/jdbc/drivers . On trouve dans le package téléchargé un fichier .jar qui est le driver JDBC.

Remarque pratique : Pour l'installer sous Eclipse, clic droit sur le projet >> properties >> Java Build Path (à gauche) >> Add External JARs.

Par la suite, la grande majorité du code repose sur l'utilisation de deux classes spécifiques à la manipulation des bases de donnée en Java :

- La Classe 'Connection', dont l'une des fonctions prend en argument l'adresse URL de la base de données sur laquelle on souhaite agir, ainsi que les coordonnées (pseudonyme et mot de passe) d'un compte autorisé à y accéder. Cette classe est celle qui se charge de créer le lien entre les 2 ordinateurs avant l'échange de données.
- La Classe 'Statement', dont l'une des fonctions prend en argument une requête SQL et transforme cette chaîne de caractères en requête effective dans la base de données cible.

Nous devons également avoir une base de données dont la structure est cohérente avec les requêtes lancées par le programme Java. A ce propos, le logiciel gratuit phpMyAdmin fourni avec Xamp est une très bonne interface de gestion de base de données SQL, et bien évidemment beaucoup moins austère que le terminal UNIX.

Pour vérifier le bon fonctionnement du programme, nous utilisons un objet de type 'ResultSet' pour récupérer sous forme de chaîne de caractères le résultat d'une requête SELECT. Dans le cas d'une requête INSERT ou UPDATE, nous n'avons qu'à vérifier les nouvelles valeurs dans les tables de la base de données.

Une fois ce premier code terminé, nous pouvons enfin le tester pour vérifier que l'échange entre les entités se passe sans aucun obstacle, si ce n'est pas le cas nous devons l'identifier précisément. Le code est compilé mais pourtant lorsqu'on le lance, aucun échange n'a lieu. Petit à petit, nous identifions les différents problèmes qui s'opposent à son fonctionnement et pouvons les résoudre. Cela nous permettra d'être plus vigilants encore sur ces points lorsque nous aurons à coder la version Android.

Voici la liste des problèmes identifiés et, le cas échéant, la manière dont nous les avons contournés :

- Nous avons au départ souhaité faire nos tests dans un réseau local afin d'éviter toute intrusion malveillante puisque pendant cette période l'accès à nos fichiers est activé. Malheureusement, si Windows propose des guides en temps réel pour tout paramétrer, c'est plus délicat sur Ubuntu et pour ne pas perdre de temps nous avons utilisé un adressage automatique fourni par TSP.
- L'utilisation de JDBC nous a posé quelques problèmes et il nous a fallu quelque temps pour comprendre que l'on devait utiliser 2 types différents suivant le sens de la requête envoyé : prise d'information ou transfert d'information.
- Nous nous sommes rendus compte après coup que l'accès pour un ordinateur distant est par défaut refusé à tout utilisateur. Il faut donc penser à le changer soit manuellement (droits d'accès RWX) soit par l'intermédiaire d'une commande gérée par Xamp.
- L'adresse URL fournie en paramètre à l'objet de type Connection pour atteindre la base de données n'est pas seulement l'adresse IP de la machine, mais bien l'arborescence exacte menant à l’emplacement de la base de données. La connaissance de l'identité de la machine distante ne suffit pas, il faudra donc penser à créer une arborescence commune à toutes les bornes pour que le programme soit compatible partout.

- L'accès à la base de données est indépendant de l'accès à la machine distante. Nous avons donc dû également éditer des permissions spécifiques aux requêtes effectuées par le programme Java. PhpMyAdmin permet de créer des classes d'utilisateur (couple pseudonyme, mot de passe) auxquels sont associés des droits (requêtes SELECT, DELETE, etc...). Une classe 'device' a donc été créée spécialement pour notre programme avec les droits en SELECT, INSERT et UPDATE.

A l'issue de ces réglages, les tests se sont révélés positifs et les 3 types d'échange ont fonctionné. Ce programme est donc un bon point de départ pour notre projet que nous devrons adapter en Android. Une phase d'apprentissage du langage Android est prévue avant de coder, et pour nous aider, nous avions aussi le projet sur l'IP TV.

Néanmoins il s'est avéré que Android (à l'heure actuelle) ne supporte pas l'utilisation de JDBC nous avons donc dû opter pour une nouvelle architecture pour notre application, celle-ci est détaillée au chapitre VII.

VI. Apprentissage de Android

Nous avons appris à coder en Android grâce au site web d'aide à la programmation 'http://www.siteduzero.com'' que nous recommandons par ailleurs à toute personne souhaitant apprendre les bases de Android. A ce sujet, certains passages de cette partie (Apprentissage de Android) seront directement tirés du site du Zéro.

Android est donc un système d'exploitation à destination des terminaux mobiles. Il équipe principalement des téléphones portables (de type smartphone) mais aussi des baladeurs ou autres tablettes. Android est édité par Google. Contrairement aux différents langages que nous avons étudiés jusqu'à présent, le langage Android ne centralise pas tout le code d'une application sur une méthode 'main' mais fonctionne sur le principe des activités, qui ont un cycle de vie bien défini, comprenant une naissance et une mort. On peut considérer une activité comme un support sur lequel se grefferont les éléments de l'interface graphique.

Cependant, ce n'est pas son rôle que de créer et de disposer les éléments graphiques, elle n'est que l’échafaudage. En revanche, elle s'occupera de définir les comportements que doivent adopter les éléments de l'interface graphique. C'est donc l'activité qui détermine le comportement d'une page, mais pas son esthétique. En Android, le code doit comprendre les actions spécifiques à chaque état possible de l'application ; de plus une des spécificités est que l'interface graphique et le code contenant les événements de l'application sont pratiquement indépendants. Le seul lien entre ces deux interfaces est la liste des identités des « greffons » contenus dans le fichier XML renvoyant sur les différents boutons présents dans l'application.

Lorsqu'on crée un projet Android, on est souvent un peu perdu face à la grande quantité de ressources qui apparaissent. En réalité, chacune d'entre elles possède un rôle bien précis :

- La partie 'drawable' regroupe les fichiers de type image qui devront s'afficher pendant l'application.
- La partie 'layout' qui regroupe des fichiers XML décrivant l'organisation spatiale de l'application (position des boutons et des images, couleurs...)
- La partie 'values' qui regroupe les fichiers XML ayant des variables non nulles avant de lancer une application (chaîne de caractères par exemple)
- le Manifeste Android dont l'un des rôles est de décrire les droits de l'application sur le smartphone hôte.

Chacun de ces dossiers peut ensuite être personnalisé suivant certains critères tels que la langue du Smartphone, sa taille... Cela permet d'adapter le contenu de l'application aux caractéristiques des smartphones. Le compilateur crée ensuite un fichier pré-compilé R.java à partir de tous les fichiers de ressource. Nous n'avons pas à le modifier nous même car il est géré automatiquement.

Une fois l'environnement apprivoisé, il est plus facile de s'y retrouver pour coder. Il est indispensable de comprendre que tout ce qui compose le corps de l'application (images, boutons, valeurs de variables) sont stockées en mémoire dans la partie ressources et sont récupérables grâce à une identité qui leur est fournie dans la description d'un fichier XML. Ainsi on peut lier dans le cœur de l'application des portions de codes à des événements : par exemple la fonction onClick appliquée à un bouton reconnaît si l'utilisateur clique sur ce bouton.

Nous avons donc mis à profit ce tutoriel pour tester les déclencheurs et les événements, les différentes possibilités pour établir une interface graphique... Nous avons aussi pu nous familiariser avec l'émulateur de terminal Android qui dispose de toutes les fonctionnalités d'un vrai terminal. En effet, la navigation sur Internet est possible, de même qu'un paramétrage total du téléphone (langue, heure, mode économie d'énergie...). Nous pouvons désormais commencer à travailler sur l'application que nous devons produire pour ce projet de fin d'études.

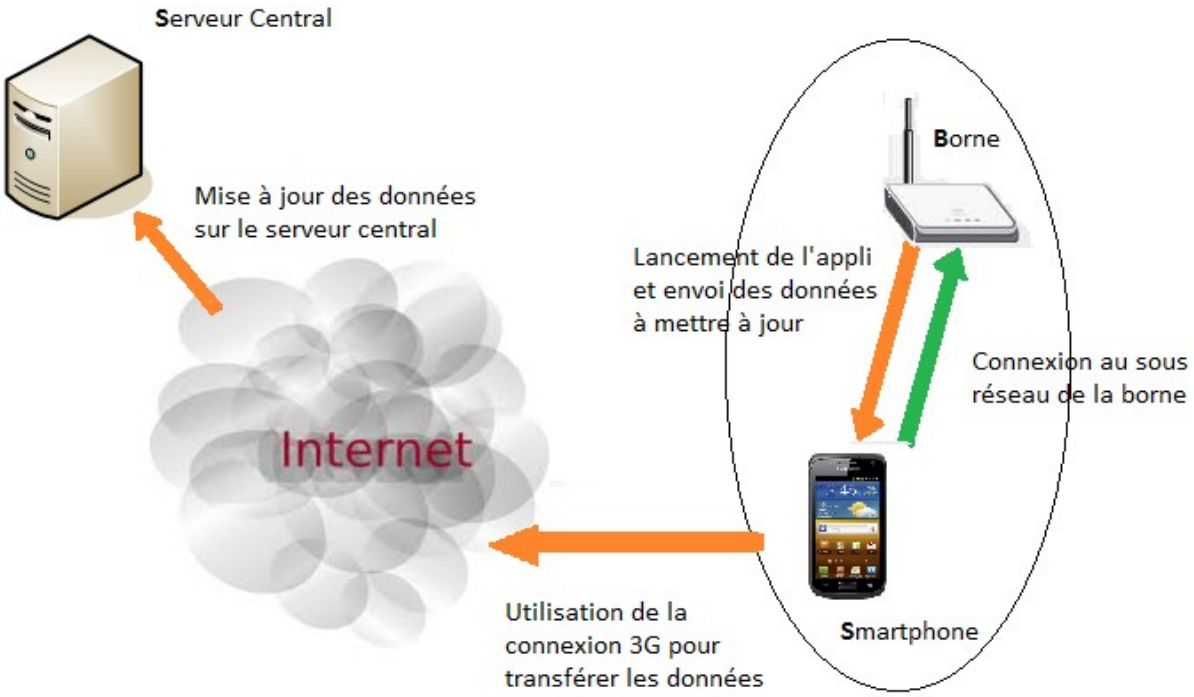
Pour plus de détails concernant notre activité voir la partie VIII. Description du code.

VII. Architecture du programme

Suite aux difficultés rencontrées (voir fin chapitre VI) pour établir une connexion directe entre l'OS Android et une base de données, nous avons opté pour une nouvelle architecture où cette fois le smartphone n'exécute plus lui-même les requêtes MySQL mais va passer par l'exécution de programmes PHP. Ces derniers sont localisés sur les serveurs des bases de données et se chargeront des opérations après lesquels ils renverront les résultats au smartphone. Celui-ci joue donc le rôle d'intermédiaire :

- il demande à la première base de données les informations mises à jour à travers le programme PHP qui les lui envoie
- il renvoie les données à la base de données à mettre à jour en faisant s'exécuter un second programme PHP

Cette architecture a le bénéfice de rendre la connexion moins lourde à gérer pour le smartphone qui ne gère plus directement les opérations MySQL.



Description spatiale des interactions lors d'un transfert de données

VIII. Description du code

Les parties encadrées en bleu sont issues du code en Java, celles en rouge du code en PHP et celles en vert du code en XML. L'explication du code Java suit le code écrit dans le fichier UpdateWonderville.java

Références des documentations utiles :

- http://developer.android.com
- http://fr.php.net/manual/fr/index.php

1. Introduction

Le programme que nous avons codé est donc une application Android sous Eclipse.

L'idée du programme n'est pas de présenter une interface graphique très travaillée mais simplement permettre à l'utilisateur de lancer l'échange de données.
La procédure est la suivante : l'utilisateur si il est d'accord appuie sur un bouton qui va déclencher l'échange entre les bases de données MySQL. Celui-ci est effectué de la façon suivante : ce sont des programmes PHP qui assurent l'exécution des opérations SQL et qui renvoient le résultat (les données à transférer) au smartphone.

Le smartphone n'est donc qu'un relais pour les données, ce qui permet d'ailleurs de le délester de la gestion coûteuse en ressource des opérations SQL.

2. Une Activité Android

Android gère les applications à travers une classe abstraite appelée une activité : elle est définie par la documentation comme étant « une chose qui à la préemption (le focus) que l'utilisateur peut faire. Presque toutes interagissent avec l'utilisateur, elles prennent donc en charge la création d'une fenêtre où placer l'interface utilisateur (UI). »

Notre programme Android « UpdateWonderville » se présente donc sous la forme d'une classe implémentant la classe abstraite « Activity ».

```
public class UpdateWondervileActivity extends Activity implements
          View.OnClickListener {

    // our program

}
```

De manière générale une activité implémente souvent :

- onCreate(Bundle) : l'endroit où l'activité est initialisée, on y utilise souvent la méthode setContentView(int) avec une ressource de contenu qui définit l'UI
- onPause() : ici est traitée le cas où l'utilisateur quitte l'activité

Dans la mesure où il n'y a pas de données à sauvegarder ou de changement à mémoriser nous n'avons pas utiliser onPause().

2.1. onCreate(Bundle)

```
public void onCreate(Bundle appli) {

     // code

}
```

2.1.1. Layout et XML

Notre programme est donc constitué d'un appel à onCreate() qui va alors lancer l'affichage de l'interface, simplement constituée d'un texte et d'un bouton. Ceux-ci sont, comme le veut Android stockés dans un fichier XML nommé demande.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bonjour," />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Voulez-vous participer à un projet écologique?"
     />

</LinearLayout>
```

Ce fichier XML est la base de stockage des éléments qui vont se greffer à l'interface graphique, on y reconnaît un texte (TextView) et un bouton (Button). A propos des attributs XML rencontrés on en remarquera quelques uns :

- layout : tous les attributs incluant le mot layout font référence à la place en hauteur et en largeur que doit prendre l'application
- id : l'attribut id sert à référencer les éléments d'une façon plus simple et permet de les récupérer facilement grâce à la fonction findViewById(int) pour ainsi les manipuler aisément
- text : cet attribut permet de définir le texte tel que l'utilisateur le verra. Pour le bouton c'est celui qui apparaîtra sur celui-ci

Cette interface est appelée grâce à la fonction setContentView(R.layout.*demande*). :

```
setContentView(R.layout.demande);
```

Notez également que ce texte et ce bouton sont simplement initialisés juste avant l'appel à onCreate() :

```
Button b = null;
TextView myText = null;
```

2.1.2. Le Bouton et son linker

Nous avons donc un texte suivi d'un bouton qui sont affichés dans l'UI, il est maintenant nécessaire d'associer au bouton l'action qui se produira lorsque l'utilisateur le « cliquera » : il faut le linker. Pour cela on se sert de la méthode setOnClickListener, on demande alors à l'activité d'« écouter » ce bouton et donc le message qui indiquera qu'il a été appelé.

```
b = (Button) findViewById(R.id.button);
b.setOnClickListener(this);
```

Le code décrivant l'action qui s'ensuit est décrit dans une fonction extérieur à onCreate(Bundle) est logiquement appelé onClick(View v).

2.2. onClick(View v)

```
public void onClick(View v) {

     // code

}
```

Le but du bouton est de lancer l'échange de donnée entre les bases de données, avant cela nous allons éditer un nouveau contenu sur l'UI. Cela est assuré par le TextView « myText » initialisé au début du code :

```
TextView myText = null;
```

Cette vue est ensuite « construite » dans le onClick(View v), on lui associe un texte grâce à la méthode de TextView setText(CharSequence text) puis on l'affiche finalement sur l'UI à l'aide de la méthode setContentView(View view) appliqué à notre TextView « myText ».

```
myText = new TextView(this);
myText.setText("Bonjour, vous avez lancé onClick.");
setContentView(myText);
```

Cette partie du code était donc pour informer l'utilisateur du déroulement des opérations, il faut maintenant s'occuper de l'échange de données. C'est à travers la classe ConnectionToServer que nous allons faire cela, nous allons donc en créer une instance et lancer la méthode qui va se charger du transfert (ici nommée execute()) :

```
ConnectionToServer cts = new ConnectionToServer();
cts.execute((Void) null);
```

Le détail des opérations est donné dans la prochaine partie.

2.3. Class ConnectionToServer

C'est donc la classe qui va effectuer le transfert des données mises à jour d'une base de données (appartenant soit à la borne soit au serveur central) à une autre base de données (appartenant inversement au serveur central ou à la base de données).

Cette tâche implique un certain nombre de connexions or nous avons tout d'abord constaté que pour que l'application puisse utiliser le Wi-Fi il lui faut la permission nécessaire. Elle lui est donnée en ajoutant la ligne suivante au fichier automatiquement créé AndroidManifest.xml :

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

Nous avons également appris qu'une connexion ne pouvait être gérée dans ce qui est appelé le "UI thread", le fil principal (celui de l'interface utilisateur). C'est à dire que Android refuse de prendre en charge de telles opérations qu'il considère plus risquées et nécessitant donc d'être exécutées sur un fil séparé (auquel cas s'il survient un problème toute l'application ne plantera pas) et demande à les traiter en tant que "AsyncTask", tâche asynchrone.

2.3.1. AsyncTask

Tout le travail de connexion aux bases de données doit donc être géré comme tel.
Ainsi Android crée un nouveau thread sur lequel il effectuera les opérations de connexions. Cela se traduit par l'utilisation d'une classe spécifique, AsyncTask :

```
private class ConnectionToServer extends AsyncTask<Void, Void, Void> {

     // AsyncTask

}
```

Elle est définie par trois types génériques :

1. Params : le type des paramètres envoyés à la tâche qui va s'exécuter
2. Progress : le type des unités de progression (voir étape 2)
3. Result : le type du résultat de la tâche

Ces types peuvent ne pas être utilisés, ils sont alors initialisés à « Void ».
Notre AsyncTask ne prend aucun de ces paramètres.

Une telle classe déroule ses actions en quatre étapes :

1. onPreExecute () : cette méthode est appelée au tout début de l'exécution de la tâche et est généralement utilisée pour la mettre en place, en montrant par exemple une barre de progression dans l'interface utilisateur
2. doInBackground (Params...) : celle-ci est appelée juste après l'exécution de la précédente en « background »/arrière-plan (en opposition à ce qui est « visible »). On l'utilise pour des tâches relativement longues, elle prend les paramètres passés à AsyncTask et renvoie un résultat à la dernière étape. Cette étape peut se servir de la méthode publishProgress (Progress...) pour envoyer des unités de progression à l'étape suivante.
3. OnProgressUpdate (Progress...) : cette étape n'est appelée que si la précédente a elle-même appelée la méthode publishProgress(Progress...), elle se charge d'afficher dans l'UI thread les unités de progression de la méthode doInBackground(Params...).
4. OnPostExecute (Result) : finalement, cette étape s'exécute à la fin de doInBackground(Params...) (son résultat est passé en paramètre) et permet d'afficher des informations sur l'UI thread.

La seule étape obligatoire est la seconde.

Nous avons décidé de n'utiliser que celle-ci et la dernière afin d'informer l'utilisateur du succès de sa participation au travers d'un « toast » (un court message qui s'affiche un temps limité).

```
@Override
protected Void doInBackground(Void... params) {

     // connexions process

}
```

```
public void  onPostExecute(Void param) {
     Toast.makeText(UpdateWondervileActivity.this,
           "Connexion terminée, merci d'avoir participé!",
           Toast.LENGTH_SHORT).show();
```

```
}
```

Les autres pourraient tout à fait être utilisés pour améliorer la visibilité de l'action effectuée pour l'utilisateur.

Source : http://developer.android.com/reference/android/os/AsyncTask.html

2.3.2. La Connexion - void doInBackground()

Nous sommes donc maintenant dans une tâche effectuée en arrière-plan par Android et qui est autorisée à effectuer des connexions. L'idée est donc de se connecter au serveur et de lancer une application PHP située sur le serveur. Celle-ci se connectera alors à la base de données qui va se charger de récupérer (ou d'insérer) les données et de les transférer (si besoin) au smartphone.

De plus la tâche va se dérouler principalement en deux phases impliquant chacune le lancement d'une application PHP. Une première phase qui va permettre de récupérer les informations de la première base de données et la seconde qui va les envoyer à le deuxième base de données.

Notre code permet également de passer un argument à ces applications PHP pour, par exemple, donner une information supplémentaire sur les données à récupérer. Nous ne l'utilisons à aucune fin particulière dans la partie récupération des informations pour le moment d'autant plus que comme nous le suggérons dans la partie « Améliorations » du présent document (voir VIII. Améliorations), l'application PHP pourrait prendre elle-même les décisions sur quelles informations transférer ou non. En revanche elle est assez utile pour l'envoi des données comme nous le verrons.

Nous allons donc démarrer l'étude du code en commençant par quelques déclarations.

Première phase : Récupération des données

Déclarations initiales

Nous déclarons donc au début un String result qui accueillera le résultat de la récupération des données (voir ci-après).

Nous initialisons également une paire attribut/valeur ; c'est elle qui contient l'argument passé à l'application PHP comme susmentionné. Ainsi « year » est le nom de la variable PHP et 1950 est la valeur que nous voulons lui donner. Nous l'insérons dans une ArrayList pour pouvoir la traiter par la suite (c'est l'argument de la méthode « UrlEncodedFormEntity() » - voir ci-après). Cette paire est un objet NameValuePair construite grâce au constructeur BasicNameValuePair(« attribut », « valeur »).

Enfin nous initialisons un InputStream, dans notre cas pour récupérer le flux venant du réseau (la réponse envoyé par l'application PHP).

```
String result = "";

ArrayList<NameValuePair> nameValuePairs = new
     ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("year", "1950"));

InputStream is = null;
```

La Requête HTTP POST

Vient ensuite la requête HTTP POST qui doit assurer la connexion au serveur et le lancement de l'application PHP.

Pour cela nous avons besoin d'un premier objet de type HttpClient qui va exécuter la requête, d'un second objet de type HttpPost qui va , en quelque sorte, contenir l'objet de la requête, d'un troisième objet de type HttpResponse qui va récupérer le résultat « brut » de la requête et enfin un quatrième et dernier de type HttpEntity qui va extraire de HttpResponse les entités qu'il contient.

Notre HttpClient httpclient est « construit » à l'aide du constructeur DefaultHttpClient(), le HttpPost httppost est, quant à lui, initialisé à l'aide du constructeur HttpPost() auquel est passé en argument l'URL menant au document PHP.

On passe ensuite les arguments à httppost à l'aide de la méthode setEntity qui prend comme argument le résultat de la méthode UrlEncodedFormEntity() : c'est ici qu'intervient la paire initialisée tout à l'heure (notez que si il y a besoin on peut passer plus d'une paire en les mettant dans l'ArrayList). UrlEncodedFormEntity() va encoder au format URL les arguments passés).

Ensuite vient l'exécution de la requête associée à httppost à l'aide de la méthode execute() appliquée à httpclient, le résultat de cette opération est retourné dans response (de type HttpResponse). Finalement on récupère la réponse sous forme d'entités grâce à la méthode getEntity() sur response retournée dans entity (de type HttpEntity).

Remarque pratique :

- il faudra toujours penser, surtout lors des tests, à remettre à jour l'adresse IP et vérifier que le chemin est le bon (nom du fichier compris)
- sous Linux les déplacements dans les dossiers se font à l'aide du slash « / », sous windows ils se font à l'aide de l'antislash « \ »

Enfin l'InputStream is récupère le contenu des entités en vue de la prochaine étape.

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("http://192.168.1.94/mysql.php");
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
HttpResponse response = httpclient.execute(httppost);
HttpEntity entity = response.getEntity();
is = entity.getContent();
```

La Conversion de la réponse

Cette partie n'est pas forcément des plus intéressantes et n'est pas fondamentale à la compréhension du code mais elle est néanmoins nécessaire - en résumé le résultat du HTTP POST n'est pas exploitable tel quel et nécessite d'être travaillé afin d'en obtenir une variable de type String qui sera stocké dans la variable result initialisé au début (voir « Déclarations initiales »).

Tout commence avec la création d'un « BufferedReader », reader, qui va mettre en mémoire tampon l'InputStream is. Ainsi on peut facilement manipuler is (en allégeant le calcul pour la machine).
On passe au constructeur BufferedReader() un argument de type InputStreamReader qui va désigner l'Input Stream à lire (dans notre cas is) et la nature de son contenu c'est à dire des caractères de l'alphabet d'Europe occidental définit par la norme ISO 8859-1, l'int 8 définit la taille de reader en caractère (donc en octet).

Ensuite un StringBuilder, sb, est initialisé cet objet permet de manipuler des chaînes de caractères facilement. Il va nous servir à récupérer et réorganiser les lignes obtenus du BufferedReader reader (qui viennent de la réponse HTTP venant du fichier PHP) pour en faire un fichier intelligible.

Cela est assuré par la boucle while qui à chaque ligne de reader va l'ajouter (méthode append (char[] chars)) dans la chaîne sb en y ajoutant le caractère spécial « \n » pour le saut de ligne dans une chaîne de caractères.

Enfin is a rempli son rôle de récepteur des informations entrantes, il est donc fermé : is.close (). On récupère finalement le contenu de sb dans result (initialisé au début voir « Déclarations initiales »), en tant que String.

```
BufferedReader reader = new BufferedReader(
     new InputStreamReader(is, "iso-8859-1"), 8);
StringBuilder sb = new StringBuilder();
String line = null;
while ((line = reader.readLine()) != null) {
     sb.append(line + "\n");
}
is.close();

result = sb.toString();
```

JSON et fichier Log

En regardant en parallèle de ce document le code présenté en java vous remarquerez que les parties de code présentées aux paragraphes « La Requête HTTP POST » et « La Conversion de la réponse » sont en fait dans des blocs « try/catch » :

```
try {
     // code
} catch (Exception e) {
     // what to do in case of problem during the try
}
```

Dans le « try » se trouve le code à exécuter et dans le « catch » le code si jamais l'exception citée en argument se produit.

Dans un code java « normal » on peut facilement renvoyer des messages à l'utilisateur/programmeur en se servant de la fonction System.out.println(String s). En programmation sous Android cela n'est pas possible car il n'y a pas de console dans le programme s'effectuant sous machine virtuelle (ou dans un équipement Android).

C'est pourquoi il existe un outil (uniquement sur l'émulateur Android – pas sur les équipements sous Android) fourni sur Eclipse qui sont les fichiers log. On peut dans ce fichier inscrire des messages au fur et à mesure du code (que cela soit dans un bloc « catch » ou ailleurs).

On l'utilise de la façon suivante : on appelle sur l'objet Log une des cinq méthodes d, e, i, v ou w (respectivement pour debug, error, info, verbose ou warning – le choix appartient au programmeur sachant que Eclipse se sert de toute façon de ces méthodes) – qui prennent toutes en argument deux Strings (que le programmeur peut remplir librement) : un pour le « tag » (par exemple le nom de l'activité) et l'autre pour le message en lui-même.

Ces messages sont donc envoyés si besoin dans le fichier log correspondant et sont consultables sous Eclipse (voir « Dans la pratique » ci-après).

Pour en revenir à la suite de notre code voici un exemple de l'utilisation du log « info ».
Les données récupérées par le PHP et envoyées à notre application sont sous forme de document JSON (voir ci-après le paragraphe sur le fichier PHP), il convient donc de les extraire comme tel pour les interpréter correctement.
La variable String result contient donc une chaîne de caractères résultant des opérations précédentes qui est en fait un fichier JSON : c'est pourquoi on initialise jArray qui est un JSONArray à partir de result. Cela permet donc d'afficher dans le fichier log i le contenu de result simplement en utilisant la structure JSON pour récupérer les différents éléments.

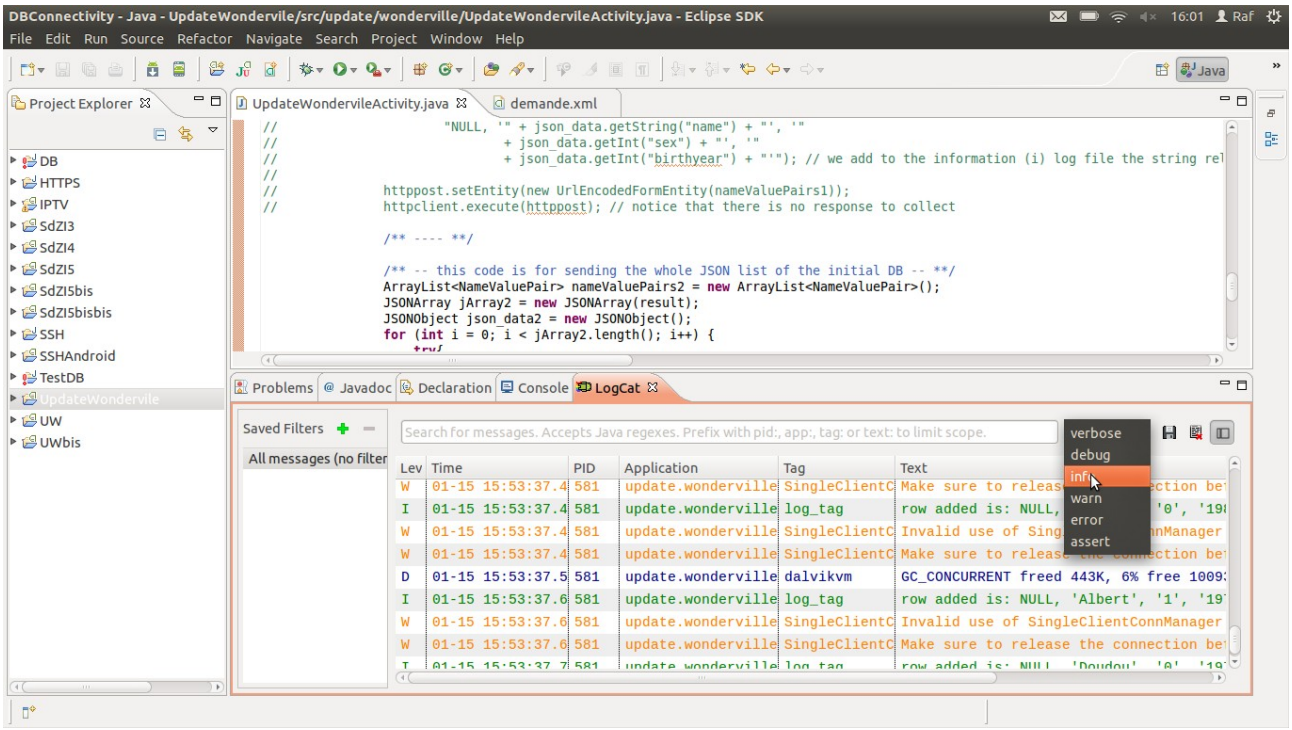
```
JSONArray jArray = new JSONArray(result);
for (int i = 0; i < jArray.length(); i++) {
     JSONObject json_data = jArray.getJSONObject(i);
     Log.i("log_tag", "n°1 id: " + json_data.getInt("id")
           + ", name: " + json_data.getString("name")
           + ", sex: " + json_data.getInt("sex")
           + ", birthyear: " + json_data.getInt("birthyear"));
}
```

Remarque : Cette partie est complètement dispensable elle permet juste en phase de test de vérifier ce qui a été pris de la première base de données.

<u>Attention :</u> toutes les références à ces fichiers log devront être supprimés du code lors de l'envoi de l'application sur un équipement Android.

<u>Dans la pratique :</u>

Sous Eclipse allez dans l'onglet LogCat (si il n'est pas visible : Window >> Show View >> Other... >> Android >> LogCat), on retrouve dans la colonne de gauche « Lev » le type de Log demandé (d, e, i, v ou w), le premier argument tag dans la colonne « Tag » et le second argument dans la colonne « Text » :



<u>Deuxième phase : Renvoi des données</u>

Pour rappel nous avons pris de la première base de données les éléments qui nous intéressaient : nous avons donc un String result qui contient en fait un JSON avec tous les éléments stockés de façon structurée.
Il s'agit donc maintenant de renvoyer ce JSON vers la seconde base de données.

Nous remarquerons tout d'abord qu'ici les tâches sont réduites car il n'y a pas d'information à récupérer. Le programme, en comparaison avec la première phase ne consiste donc qu'en un envoi des données donc une connexion HTTP.

Code identique à la première requête HTTP POST à ceci près que il n' y a pas de HttpResponse et le code qui s'ensuit à initialiser. En revanche il va être nécessaire d'effectuer plusieurs requêtes HTTP : autant que d'éléments présents dans le JSON (autrement dit d'éléments prélevés dans la première base de données ; ce nombre est accessible grâce à la méthode length() appliquée à jArray2), d'où l'utilisation d'une boucle for sur les éléments du JSON (référencés de 0 pour le premier à jArray2.length() – 1 pour le dernier).

Nous allons également réutiliser le système du NameValuePair (voir « Déclarations

initiales ») pour passer les arguments au fichier PHP – à chaque itération de la boucle for nous allons réinitialiser sa valeur pour l'adapter à l'élément en cours.
Nous réinitialisons donc le JSON (localement - d'où la non réutilisation du JSON jArray précédent) et initialisons ce qui va être l'élément en cours : le JSONObject json_data2.

```
ArrayList<NameValuePair> nameValuePairs2 = new
ArrayList<NameValuePair>();
JSONArray jArray2 = new JSONArray(result);
JSONObject json_data2 = new JSONObject();
for (int i = 0; i < jArray2.length(); i++) {

     // code (voir ci-dessous) :
           récupérer l'objet en cours
           entrer la valeur de l'argument pour le PHP (nameValuePairs2)
           (envoyer un message au log i)
           envoyer l'argument au PHP
           exécuter la requête

}
```

Nous nous plaçons donc maintenant dans la boucle for, la première étape est donc de récupérer l'élément en cours dans json_data2 grâce à la méthode getJSONObject(int i) appliquée à jArray2. L'int i est donc celui de l'itération en cours qui référence l'élément du JSON.
On peut ensuite constituer la chaîne de caractères qui contiendra les valeurs à inscrire dans la table lors de la requête INSERT mySQL que nous insérerons dans l'array nameValuePairs2.
Pour le suivi (en phase de test) nous inscrivons cette chaîne dans le log i.
Enfin nous associons nameValuePairs2 à la requête HTTP POST à l'aide de la méthode setEntity() sur httppost et finalement nous exécutons la requête.

```
json_data2 = jArray2.getJSONObject(i);
nameValuePairs2.add(new BasicNameValuePair("year", "NULL, '"
     + json_data2.getString("name") + "', '"
     + json_data2.getInt("sex") + "', '"
     + json_data2.getInt("birthyear") + "'"));
Log.i("log_tag",
     "row added is: NULL, '" + json_data2.getString("name") + "', '"
           + json_data2.getInt("sex") + "', '"
           + json_data2.getInt("birthyear") + "'");
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs2));
httpclient.execute(httppost);
```

2.3.3. Fin de l'AsyncTask - onPostExecute() - et de l'activité

A ce stade les échanges de données sont terminés, le but est atteint, il ne nous reste plus qu'à envoyer un message pour informer (/remercier...) l'utilisateur de la fin des connexions.

Pour cela nous utilisons la méthode onPostExecute(Void Params) de AsyncTask (voir 2.3.1. AsyncTask) qui nous permet d'effectuer une action après l'exécution de la méthode doInBackground().

Nous avons décidé d'informer l'utilisateur de la fin des opérations en affichant un court message grâce à un « toast » c'est à dire un message qui s'affiche par dessus l'UI pendant un instant

donné.

Nous appelons pour cela la méthode makeText sur un objet Toast, cette méthode prend trois arguments :

- le contexte : ici notre activité UpdateWondervilleActivity
- le message : le message à afficher sur l'UI
- le temps d'apparition du message : ici un temps d'apparition « court »

```
public void onPostExecute(Void param) {
                Toast.makeText(UpdateWondervileActivity.this,
                            "Connexion terminée, merci d'avoir
participé!",
                            Toast.LENGTH_SHORT).show();

}
```

Remarque : on pourrait faire un message prenant en compte le succès de l'opération de transfert en faisant remonter une information depuis doInBackground().

3. Les programmes PHP

Comme nous l'avons vu ce n'est pas l'activité Android elle-même qui gère les requêtes sur les bases de données MySQL, elle ne s'occupe que d'activer des programmes PHP qui sont en fait sous la forme d'un fichier PHP attaché au serveur, localisé au même endroit que la base de données (MySQL dans notre cas).

Ces programmes PHP s'occupent donc de récupérer ou d'insérer des lignes dans leurs bases de données en se servant des informations fournies par le smartphone : dans le cas de la récupération il n'y a pas (pour le moment) de « consignes » (sous forme de la condition « WHERE ») en revanche pour l'insertion le smartphone doit passer tous les contenus.

3.1. Récupération des données

Tout commence par la connexion au gestionnaire de base de données (phpMyAdmin), pour cela on utilise la fonction mysql_connect ("gestionnaire", "user_name", "password") avec les trois arguments :

- chemin de la base de données par rapport au serveur (dossier public internet – ex sous Ubuntu : /var/www – la mention localhost entre guillemets suffit)
- Le nom du compte d'utilisateur sous lequel le programme va se connecter
- Le mot de passe associé au compte d'utilisateur

```
mysql_connect("localhost","device","");
```

Remarque : Il faut s'assurer que le compte utilisé a bien les droits auprès de la base de données.

La fonction mysql_select_db ("db_name") permet de sélectionner la base de données dans laquelle se trouve la table.

```
mysql_select_db("Wonderville");
```

Ensuite vient la requête en elle-même exécutée grâce à la commande mysql_query ("requête"), son résultat est stocké dans la variable $q.

```
$q=mysql_query("SELECT * FROM people WHERE birthyear>'".
     $_REQUEST['year']."'");
```

C'est ici qu'est récupérée la variable passée en Java grâce à la méthode setEntity () appliquée à notre instance d'HttpPost httppost (voir « La Requête HTTP POST »).

```
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
```

On reconnaît dans la variable qui apparaît dans la requête PHP, $_REQUEST['year'] l'attribut de la paire que nous avions initialisée :

```
nameValuePairs.add(new BasicNameValuePair("year", "1950"));
```

Elle est ici concaténée (à l'aide de « . ») à la requête. La requête est donc « lue » de la façon suivante par le programme : "SELECT * FROM people WHERE birthyear>'1950'".

La boucle while qui suit sert à retourner $q sous forme d'un tableau associatif dans une autre variable $e grâce à la fonction mysql_fetch_assoc(). $e est finalement retournée dans $output sous forme de tableau à index numérique.

```
while($e=mysql_fetch_assoc($q))
$output[]=$e;
```

Tout ceci afin de finalement retourner $output sous forme de JSON (ainsi les données sont structurées), pour cela nous utilisons deux fonctions :

- json_encode() avec en argument $output
- print() avec en argument le résultat retourné par la fonction appelée ci-dessus – la valeur est envoyée vers la sortie, dans notre cas vers le programme Android ; remarquez que si vous appelez la fonction « à la main » (sans passer par l'application Android) vous obtiendrez le résultat à l'écran

```
print(json_encode($output));
```

Enfin nous fermons la connexion à la base de données grâce à mysql_close() ;

```
mysql_close();
```

3.2. Insertion des données

Le code PHP pour insérer est plus simple que le premier puisqu'on y retrouve rigoureusement les mêmes étapes en enlevant celles relatives au traitement et au renvoi de l'information.

Dans ce qui est repris par rapport au code précédent la seule différence est au niveau de la requête qui n'est plus un SELECT mais un INSERT. On récupère dans la variable year passée en argument (voir paragraphe précédent) la valeur à donner à la ligne à insérer. Voir le code java (« Deuxième phase : Renvoi des données ») pour la constitution de cette partie de la requête.

Suit le code PHP de cette partie insertion.

```
<?php
     mysql_connect("localhost","device","");
     mysql_select_db("Wonderville");

     mysql_query("INSERT INTO `Wonderville`.`people` (`id`, `name`,
           `sex`, `birthyear`) VALUES (".$_REQUEST['year'].")");

     mysql_close();
?>
```

IX. Améliorations

Les délais pour réaliser ce projet étant relativement courts, nous avons parfois dû faire des choix au moment de démarrer une nouvelle tâche. Ainsi, les difficultés rencontrées face à la découverte du langage Android nous ont par moment fait perdre de précieuses heures de travail pour comprendre l'erreur qui nous faisait face. Nous avons donc considéré qu'un rapport de projet détaillant précisément notre code ligne par ligne et expliquant toutes les subtilités du langage serait plus profitable pour un groupe tenu de reprendre notre projet. Mais nous avons tout de même pensé à des suggestions d'amélioration que nous allons détailler autant que possible dans cette partie.

Mise à jour optimisée de la base de données

Comme chaque borne disposée dans Montreuil récupérera des données spécifiques aux échanges qu'elle aura réalisé, et surtout sans informer les autres bornes, il est important de développer un mécanisme qui mettra à jour la base centrale en recoupant intelligemment les informations de chaque borne. Pour cela, nous devons harmoniser les différentes tables de notre base de données. Il nous faut :

- Un attribut de type BOOLEAN renseignant sur la présence ou non de cette entrée dans d'autres bornes. Par exemple : 'Niveau de pollution à proximité de la borne n' -> NON ; 'Nombre de places de cinéma disponibles' -> OUI.
- Un attribut de type DATETIME renseignant sur la dernière mise à jour effectuée sur l'entrée. Cela servira à considérer l'information la plus récente.
- Un attribut de type CHAR renseignant sur le type de requête à effectuer pour cette entrée : 'i' pour un INSERT, soit l'ajout d'une nouvelle entrée dans la base centrale ; 'u' pour un UPDATE.

Puis ensuite, l'application procéderait comme suit :

- Le Smartphone récupérerait l'ensemble des données présentes dans la base de données locale de la borne et les organiserait dans un tableau d'éléments JSON. Puis il supprimerait toutes les entrées ayant l'attribut 'i' pour éviter les doublons lors de la prochaine mise à jour.
- Pour éviter les échanges inutiles et énergivores en 3G, l'application enverrait ensuite le tableau JSON sur une page php gérée par le serveur de la mairie (qui abrite également la base de données centrale).
- Des requêtes INSERT seraient créées pour les éléments disposant de l'attribut 'i'.
- Des requêtes UPDATE seraient créées pour les éléments disposant de l'attribut 'u' ET 'NON'
- Enfin, pour les éléments disposant du couple 'u' ET 'OUI', une requête SELECT des entrées jumelles de la base centrale serait envoyée afin de comparer la date de mise à jour. Puis, seulement les éléments plus récents que ceux de la base centrale seraient envoyés via une requête UPDATE.

Enfin on pourra remarquer que le smartphone effectue une connexion par ligne à insérer. Il pourrait peut-être être fait en sorte que cela soit fait en une seule connexion en utilisant par exemple plusieurs BasicNameValuePair et en laissant le fichier PHP gérer l'intégralité du tableau JSON.

Récupération automatique de l'adresse IP de la borne

Notre code considère l'adresse IP de la borne sur laquelle on se connecte comme étant connue à l'avance. Cependant il serait normal que l'application se charge elle-même de récupérer l'adresse IP de la borne. Pour cela nous avons envisagé deux solutions possibles.

- La première solution consiste à toujours choisir le même type d'adresse IP pour une borne relativement à l'adresse de son sous réseau. Par exemple si l'adresse du sous réseau est x.x.x.n alors l'adresse de la borne sera x.x.x.n+1. Ainsi, le Smartphone n'aura qu'à relever l'adresse IP de son sous-réseau auquel il fera une opération arithmétique simple pour obtenir l'IP de la borne et lancer l'échange.
- La seconde solution est plus pointue du point de vue réseau, et nous n'avons pas eu le temps de faire suffisamment de recherches pour la détailler. Cependant sur le principe, l'idée est de récupérer le SSID puis l'adresse MAC du point d'accès grâce aux fonctions Android liées à la classe WifiConfiguration (sources : http://developer.android.com/reference/android/net/wifi/WifiConfiguration.html). Puis on peut théoriquement récupérer l'adresse IP au moyen d'une requête RARP.

Sécurité

On remarquera que pour le moment nous n'avons traité d'aucun sujet concernant la sécurité. Nous n'avons pas eu le temps de traiter ce point mais nous avons eu quelques idées à ce propos. Premièrement, le lien entre le smartphone et le serveur central étant assuré par la connexion 3G de l'utilisateur, il est donc déjà sécurisé par cette technologie. En revanche la connexion entre l'utilisateur et la borne toute proche est assurée par une connexion WiFi qui n'inclut aucun mécanisme de sécurité. Nous pourrions donc mettre en place un système HTTPS mais cela nécessiterait d'implémenter un mécanisme avec des certificats pas forcément facile à gérer.

Une autre solution consisterait à exécuter la requête grâce au protocole SSH. Il suffit de mettre en place un serveur SSH sur la borne et d'utiliser une bibliothèque Java pour exécuter les requêtes. Par exemple nous avons trouvé la bibliothèque JSCH créée par le groupe Jcraft. Vous trouverez au chapitre XI (en annexe), au paragraphe 6 un exemple utilisant JSCH.

Site : http://www.jcraft.com/

Connectivité

Une dernière idée d'amélioration est de laisser le choix à l'utilisateur d'utiliser le type de connexion qu'il veut (WiFi, Bluetooth...) pour se connecter à la borne en le faisant choisir via un bouton dans l'interface graphique.
Cela impliquerait l'utilisation de fonctions spécifiques (voir doc Android) . Attention, dans ce cas n'oubliez pas de donner les permissions dans AndroidManifest.xml (voir chapitre VIII, paragraphe 2.3).

Voici un exemple d'une fonction qui permet de manipuler la connectivité WiFi :

```
WifiManager wm = (WifiManager) getSystemService(WIFI_SERVICE);
if (wm.getWifiState() == WifiManager.WIFI_STATE_DISABLED) {
     startWifi(); // Démarrer le wifi dans le cas où il est désactivé
}
```

X. Conclusion

Ce projet nous a beaucoup apporté, principalement parce qu'il a touché à différents langages et différentes technologies. En plus du langage Android, nous avons dû gérer des fichiers PHP, des requêtes SQL et comprendre les protocoles réseaux utilisés. Principalement en ce qui concerne Android, il est indispensable de savoir rechercher dans la documentation officielle car ce langage est encore jeune et l'on trouve peu de tutoriels ou d'aide en ligne. C'était d'ailleurs plutôt formateur même si cela nous a fait perdre du temps.

Par ailleurs ce projet a laissé libre cours à notre imagination et en dehors du cahier des charges nous avons eu carte blanche sur la manière d'aborder le problème et sur le type de solution à proposer. Malheureusement nous n'avons pas eu le temps de coder toutes nos idées et nous nous retrouvons obligés de proposer des suggestions d'amélioration. Pour finir, un des éléments qui nous aura peut être manqué tout au long de ce projet est de pouvoir disposer d'un réel équipement Android pour effectuer nos tests, car si ceux-ci se sont avérés concluants sur l'émulateur, nous ne savons pas à quel point ce dernier respecte le comportement d'un terminal physique et s'il est vraiment fiable. Aussi, nous conseillons au département RS2M de fournir s'ils le peuvent aux prochains groupes des Smartphones pour leur permettre de réaliser leurs tests.

XI. Annexes : codes sources

1. UpdateWonderville.java

```
package update.wonderville;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class UpdateWondervileActivity extends Activity implements
            View.OnClickListener {

     Button b = null;
     TextView myText = null;

     @Override
     public void onCreate(Bundle appli) {
           super.onCreate(appli);
           setContentView(R.layout.demande);

           b = (Button) findViewById(R.id.button);
           b.setOnClickListener(this);

     }

     public void onClick(View v) {

           myText = new TextView(this);
           myText.setText("Bonjour, vous avez lancé onClick.");
           setContentView(myText);
```

```
        ConnectionToServer cts = new ConnectionToServer();
        cts.execute((Void) null);

        //finish();
    }

    private class ConnectionToServer extends AsyncTask<Void, Void,
        Void> {


        @Override
        protected Void doInBackground(Void... params) {

            String result = "";


            ArrayList<NameValuePair> nameValuePairs = new
                ArrayList<NameValuePair>();
            nameValuePairs.add(new BasicNameValuePair("year",
                "1950"));

            InputStream is = null;

            try {
                HttpClient httpclient = new DefaultHttpClient();
                HttpPost httppost = new HttpPost(
                    "http://192.168.1.14/mysql.php");
                httppost.setEntity(new
                    UrlEncodedFormEntity(nameValuePairs));
                HttpResponse response =
                    httpclient.execute(httppost);
                HttpEntity entity = response.getEntity();
                is = entity.getContent();
            } catch (Exception e) {
                Log.e("log_tag", "Error in http connection 1",
                    e);
            }

            try {

                BufferedReader reader = new BufferedReader(
                    new InputStreamReader(is, "iso-8859-
                    1"), 8);
                StringBuilder sb = new StringBuilder();
                String line = null;
                while ((line = reader.readLine()) != null) {
                    sb.append(line + "\n");
                }
                is.close();

                result = sb.toString();

            } catch (Exception e) {
                Log.e("log_tag", "Error converting result 1",
                    e);
            }

            try   {
```

```
                        JSONArray jArray = new JSONArray(result);
                        for (int i = 0; i < jArray.length(); i++) {
                              JSONObject json_data =
                                    jArray.getJSONObject(i);
                              Log.i("log_tag", "n°1 id: " +
                                    json_data.getInt("id") + ", name: "
                                    + json_data.getString("name") + ",
                                    sex: " + json_data.getInt("sex") +
                                    ", birthyear: " +
                                    json_data.getInt("birthyear"));
                  }

                  } catch (JSONException e) {
                        Log.e("log_tag", "Error parsing data 1", e);
                  }

                  try {
                        HttpClient httpclient = new DefaultHttpClient();
                        HttpPost httppost = new HttpPost(
                                    "http://192.168.1.14/mysqlInsert.php
                        ");



//                      ArrayList<NameValuePair> nameValuePairs1 = new
//                      ArrayList<NameValuePair>();
//                      JSONArray jArray = new JSONArray(result);
//                      JSONObject json_data =
//                            jArray.getJSONObject(6);
//                      nameValuePairs1.add(new
//                            BasicNameValuePair("year", "NULL, '" +
//                            json_data.getString("name") + "', '" +
//                            json_data.getInt("sex") + "', '" +
//                            json_data.getInt("birthyear") + "'"));
//                      Log.i("log_tag", "NULL, '" +
//                            json_data.getString("name") + "', '" +
//                            json_data.getInt("sex") + "', '" +
//                            json_data.getInt("birthyear") + "'");
//
//                      httppost.setEntity(new
//                            UrlEncodedFormEntity(nameValuePairs1));
//                      httpclient.execute(httppost); // notice that
//                            there is no response to collect


                        ArrayList<NameValuePair> nameValuePairs2 = new
                              ArrayList<NameValuePair>();
                        JSONArray jArray2 = new JSONArray(result);
                        JSONObject json_data2 = new JSONObject();
                        for (int i = 0; i < jArray2.length(); i++) {
                              try{
                                    json_data2 =
                                          jArray2.getJSONObject(i);
                                    nameValuePairs2.add(new
                                          BasicNameValuePair("year",
                                          "NULL, '" +
                                          json_data2.getString("name") +
                                          "', '" +
                                          json_data2.getInt("sex") + "',
```

```
                                    '" +
                                    json_data2.getInt("birthyear")
                                    + "'"));
                              Log.i("log_tag", "row added is:
                                    NULL, '" +
                                    json_data2.getString("name") +
                                    "', '" +
                                    json_data2.getInt("sex") + "',
                                    '" +
                                    json_data2.getInt("birthyear")
                                    + "'");
                              httppost.setEntity(new
                        UrlEncodedFormEntity(nameValuePairs2));
                              httpclient.execute(httppost);
                        } catch (Exception e){
                              Log.i("log_tag", "erreur sur: NULL,
                                    '" +
                                    json_data2.getString("name") +
                                    "', '" +
                                    json_data2.getInt("sex") + "',
                                    '" +
                                    json_data2.getInt("birthyear")
                                    + "'");
                        }
                  }

            } catch (Exception e) {
                  Log.e("log_tag", "Error in http connection 2",
                        e);
            }

            return null;

      }

      public void onPostExecute(Void param) {
            Toast.makeText(UpdateWondervileActivity.this,
                  "Connexion terminée, merci d'avoir participé!",
                  Toast.LENGTH_SHORT).show();
      }

 }

}
```

2. demande.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bonjour," />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Voulez-vous participer à un projet écologique?"
     />
</LinearLayout>
```

3. mysql.php

```
<?php
     mysql_connect("localhost","device","");
     mysql_select_db("Wonderville");

     $q=mysql_query("SELECT * FROM people WHERE birthyear>'".
           $_REQUEST['year']."'");$q=mysql_query("SELECT * FROM people
           WHERE birthyear>'".     $_REQUEST['year']."'");

     while($e=mysql_fetch_assoc($q))
     $output[]=$e;

     print(json_encode($output));

     mysql_close();
?>
```

4. mysqlInsert.php

```
<?php
     mysql_connect("localhost","device","");
     mysql_select_db("Wonderville");

     mysql_query("INSERT INTO `Wonderville`.`people` (`id`, `name`,
`sex`, `birthyear`) VALUES (".$_REQUEST['year'].")");

     mysql_close();
?>
```

5. Code JDBC

Vous trouverez ici le code relatif au chapitre V.

5.1. DatabaseConnectivityMain.java

```
public class DatabaseConnectivityMain {

	public static void main(String[] args) {

		GetFromDatabase dbget = new GetFromDatabase();
		dbget.whatsNew();

		InsertIntoDatabase dbinsert = new InsertIntoDatabase();
		dbinsert.insert();

	}

}
```

5.2. GetFromDatabase.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class GetFromDatabase {

	public void whatsNew() {

		Connection con = null;
		Statement stmt = null;
		ResultSet rs = null;


		/*---DataBase---*/
		String JDBCUrl = "jdbc:mysql://157.159.103.220/LEAD";
		String username = "device";
		String password = "device";

		/********** Connection to the DB **********/
		try {

			System.out.println("----- n°1
				DatabaseConnectivity.java getFromDatabase
				------");
			System.out.println("\nConnecting to the SSD
				Database.....");
			Class.forName("com.mysql.jdbc.Driver");

			con = DriverManager.getConnection(JDBCUrl, username,
				password);
		} catch (Exception f) {
			System.out.println("\nAn error has occurred during the
				connection phase!  This is most likely due to
				your CLASSPATH being set wrong and the" + "
				Oracle classes unable to be found.  Otherwise
```

```
                        the database itself may be down.  Try telneting
                        to port 1521 and see if it is up!");
                f.printStackTrace();

        }
        /********** End of Connection to the DB **********/

        /********** Process of the data **********/
        try {
                System.out.println("\nConnection Successful.....
                        creating statement....");
                stmt = con.createStatement();
                rs = stmt.executeQuery("SELECT lname FROM student");

                while (rs.next()) {
                        System.out.println("\nName=" +
                                rs.getString("lname"));
        }
        } catch (SQLException e) {
                System.out.println("\nAn error has occurred during the
                        Statement/ResultSet phase.  Please check the
                        syntax and study the Exception details!");
                while (e != null) {
                        System.out.println(e.getMessage());
                        e = e.getNextException();
                }
        }
        /********** End of Process of the data **********/

        /********** Disconnection from the DB **********/
        finally {
                try {
                        if (rs != null)
                                rs.close();
                        if (stmt != null)
                                stmt.close();
                        if (con != null)
                                con.close();
                } catch (Exception ex) {
                        System.out.println("An error occurred while
                                closing down connection/statement");
                }
        }
        /********** End of Disconnection from the DB **********/
    }

}
```

5.3. InsertIntoDatabase.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
```

```java
public class InsertIntoDatabase {

	public void insert() {

		Connection con = null;
		Statement stmt = null;
		Integer ru = null;

		/*---DataBase---*/
		String JDBCUrl = "jdbc:mysql://157.159.103.220/LEAD";
		String username = "device";
		String password = "device";

		/********** Connection to the DB **********/
		try {

			System.out.println("----- n°2
				DatabaseConnectivity.java getFromDatabase
				------");
			System.out.println("\nConnecting to the SSD
				Database......");
			Class.forName("com.mysql.jdbc.Driver");

			con = DriverManager.getConnection(JDBCUrl, username,
				password);
		} catch (Exception f) {
			System.out.println("\nAn error has occurred during the
				connection phase!  This is most likely due to
				your CLASSPATH being set wrong and the" + "
				Oracle classes unable to be found.  Otherwise
				the database itself may be down.  Try telneting
				to port 1521 and see if it is up!");
			f.printStackTrace();

		}
		/********** End of Connection to the DB **********/

		/********** Process of the data **********/
		try {
			System.out.println("\nConnection Successful.....
				creating statement....");
			stmt = con.createStatement();

			ru = stmt.executeUpdate("INSERT INTO area(name,
				description) VALUES ('Dupond', 'beau gosse')");


		} catch (SQLException e) {
			System.out.println("\nAn error has occurred during the
				Statement/ResultSet phase.  Please check the
				syntax and study the Exception details!");
			while (e != null) {
				System.out.println(e.getMessage());
				e = e.getNextException();
			}
		}
		/********** End of Process of the data **********/

		/********** Disconnection from the DB **********/
```

```
            finally {
                  try {
                        if (stmt != null)
                              stmt.close();
                        if (con != null)
                              con.close();
                  } catch (Exception ex) {
                        System.out.println("An error occurred while
                              closing down connection/statement");
                  }
            }
            /********** End of Disconnection from the DB **********/
      }

}
```

6. SSHex.java

Ici est montré un exemple d'application Java se servant de JSCH pour réaliser des connexions SSH.

```
/* help from http://www.jcraft.com/jsch/examples/ScpTo.java*/

package ssh.exemple;

import java.awt.Container;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.io.IOException;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

import com.jcraft.jsch.Channel;
import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;
import com.jcraft.jsch.UIKeyboardInteractive;
import com.jcraft.jsch.UserInfo;

public class SSHex {

      public static void main(String[] args) throws JSchException,
            IOException {

            JSch jsch = new JSch();
            Session session = jsch.getSession("root", "192.168.1.3");

            // username and password will be given via UserInfo
                  interface.
            UserInfo ui = new MyUserInfo();
            session.setUserInfo(ui);
```

```
            session.connect();

            // exec remotely
            String command = "touch fichier";
            Channel channel = session.openChannel("exec");
            ((ChannelExec) channel).setCommand(command);

            channel.connect();

            channel.disconnect();
            session.disconnect();

            System.exit(0);

      }

      public static class MyUserInfo implements UserInfo,
            UIKeyboardInteractive {
            public String getPassword() {
                  return passwd;
            }

            public boolean promptYesNo(String str) {
                  Object[] options = { "yes", "no" };
                  int foo = JOptionPane.showOptionDialog(null, str,
                        "Warning", JOptionPane.DEFAULT_OPTION,
                        JoptionPane.WARNING_MESSAGE, null, options,
                        options[0]);
                  return foo == 0;
            }

            String passwd;
            JTextField passwordField = (JTextField) new
                  JPasswordField(20);

            public String getPassphrase() {
                  return null;
            }

            public boolean promptPassphrase(String message) {
                  return true;
            }

            public boolean promptPassword(String message) {
                  Object[] ob = { passwordField };
                  int result = JOptionPane.showConfirmDialog(null, ob,
                        message, JOptionPane.OK_CANCEL_OPTION);
                  if (result == JOptionPane.OK_OPTION) {
                        passwd = passwordField.getText();
                        return true;
                  } else {
                        return false;
                  }
            }

            public void showMessage(String message) {
                  JOptionPane.showMessageDialog(null, message);
            }
```

```
            final GridBagConstraints gbc = new GridBagConstraints(0, 0,
                  1, 1, 1, 1, GridBagConstraints.NORTHWEST,
                  GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0,
                  0);
            private Container panel;

            public String[] promptKeyboardInteractive(String
                  destination, String name, String instruction, String[]
                  prompt, boolean[] echo) {
                  panel = new JPanel();
                  panel.setLayout(new GridBagLayout());

                  gbc.weightx = 1.0;
                  gbc.gridwidth = GridBagConstraints.REMAINDER;
                  gbc.gridx = 0;
                  panel.add(new JLabel(instruction), gbc);
                  gbc.gridy++;

                  gbc.gridwidth = GridBagConstraints.RELATIVE;

                  JTextField[] texts = new JTextField[prompt.length];
                  for (int i = 0; i < prompt.length; i++) {
                        gbc.fill = GridBagConstraints.NONE;
                        gbc.gridx = 0;
                        gbc.weightx = 1;
                        panel.add(new JLabel(prompt[i]), gbc);

                        gbc.gridx = 1;
                        gbc.fill = GridBagConstraints.HORIZONTAL;
                        gbc.weighty = 1;
                        if (echo[i]) {
                              texts[i] = new JTextField(20);
                        } else {
                              texts[i] = new JPasswordField(20);
                        }
                        panel.add(texts[i], gbc);
                        gbc.gridy++;
                  }

                  if (JOptionPane.showConfirmDialog(null, panel,
                        destination + ": " + name,
                        JOptionPane.OK_CANCEL_OPTION,
                        JOptionPane.QUESTION_MESSAGE) ==
                        JOptionPane.OK_OPTION) {
                        String[] response = new String[prompt.length];
                        for (int i = 0; i < prompt.length; i++) {
                              response[i] = texts[i].getText();
                        }
                        return response;
                  } else {
                        return null; // cancel
                  }
            }
      }
}
```