

## LO52 Projet Automne 2013

# Android dans le monde de la domotique



# Sommaire

Introduction .....	3
I Fonctionnement général.....	4
II Le réseau de capteurs.....	6
1 Les composants.....	6
1.1 Généralités .....	6
1.2 La carte Arduino .....	6
1.3 Les capteurs .....	6
1.4 Le module Xbee .....	7
1.5 Schéma de montage .....	8
2 La configuration.....	9
2.1 Configuration du module XBee .....	9
2.2 Protocole de communication .....	10
2.3 La programme embarqué par la carte Arduino .....	11
III La station domotique centrale.....	12
1 Généralités.....	12
2 Le programme qui lit les données.....	12
3 Le Serveur Web.....	12
4 La base de données.....	14
5 Automatisation.....	14
IV Le périphérique mobile.....	15
1 Généralités.....	15
2 Création du nouveau produit.....	15
3 Intégration de la bootanimation personnalisée.....	16
4 Intégration du composant libbmp.....	16
5 Modification du fond d'écran par défaut.....	17
6 Intégration de l'application de domotique.....	17
V L'application domotique.....	18
Conclusion.....	21
Annexes : outils à installer sur le serveur.....	22

# **Introduction**

Ce projet a pour objectif la réalisation d'un système de domotique au complet, afin de mettre en application une bonne partie des compétences acquises tout au long de l'UV LO52. En effet, étant donné sa nature et les consignes apportées, le projet a couvert plusieurs domaines, et a nécessité la réalisation de plusieurs composants indépendants et variés, qui une fois assemblés, ont constitué un système complet et autonome. Ainsi, on retiendra parmi ces nombreux développements la création d'un système Android adapté, la mise en place d'un réseau de capteurs sans fil, le développement de programmes afin de stocker, de traiter et de rendre disponible les données de façon pertinente, et enfin la création d'une application Android permettant d'afficher ces données depuis un appareil mobile.

Etant donné le nombre conséquent de composants intervenant dans la réalisation du projet, le fonctionnement du système dans son ensemble sera d'abord exposé d'une manière assez générale, puis chacun de ces éléments seront à leur tour détaillés dans une partie qui leur est propre. Cette manière de procéder permettra au lecteur de comprendre les raisons de la présence des différents composants et la position qu'ils occupent dans le système, avant d'aborder les points plus spécifiques et plus techniques relatifs à chacun de ces différents éléments.

Les consignes qui devaient être respectées étaient nombreuses, et ont permis de répondre au mieux aux besoins et aux contraintes inhérentes au projet, et d'orienter nos travaux dans la bonne direction. Toutefois, plusieurs parties ont été laissées à l'appréciation des étudiants, dans le but de développer des qualités nécessaires à la réalisation d'un projet, tel que la recherche d'une solution la plus adaptée à une problématique donnée. Tous les choix qui ont été faits seront argumentés et justifiés dans la suite de ce document.

## I Fonctionnement général

Avant de commencer l'explication détaillée sur chacun des éléments constituant le système de domotique, il convient d'apporter quelques indications sur le fonctionnement général de l'ensemble.

Plusieurs stations sont installées dans différentes pièces de la maison, et communiquent à un élément central la valeur de leurs capteurs à travers un réseau sans fil. Chaque station dispose d'un certain nombre de capteurs qui récupèrent différentes mesures sur leur environnement, comme par exemple la température ou la luminosité.

L'élément central du réseau sans fil, qui récupère toutes les données transmises par les stations, est branché à la station domotique centrale (le serveur). Ce dernier se charge de traiter et de stocker les données au sein d'une base de données. Sur ce même serveur, un service web est installé et rend les données accessibles à tout élément connecté au réseau de l'ordinateur, voir à l'ensemble de l'Internet.

Dans notre cas d'utilisation, un périphérique mobile (la tour de contrôle) est utilisée pour afficher les informations relatives aux capteurs. Une application a été développée et a pour rôle de proposer un affichage ordonné et ergonomique des différentes mesure effectuées.

Afin d'aider le lecteur à visualiser le fonctionnement général du système domotique, le schéma suivant résume de façon grossière la position des composants et leurs interactions au sein de l'architecture. Tous les composants intervenant dans le système seront détaillés dans la suite du rapport.

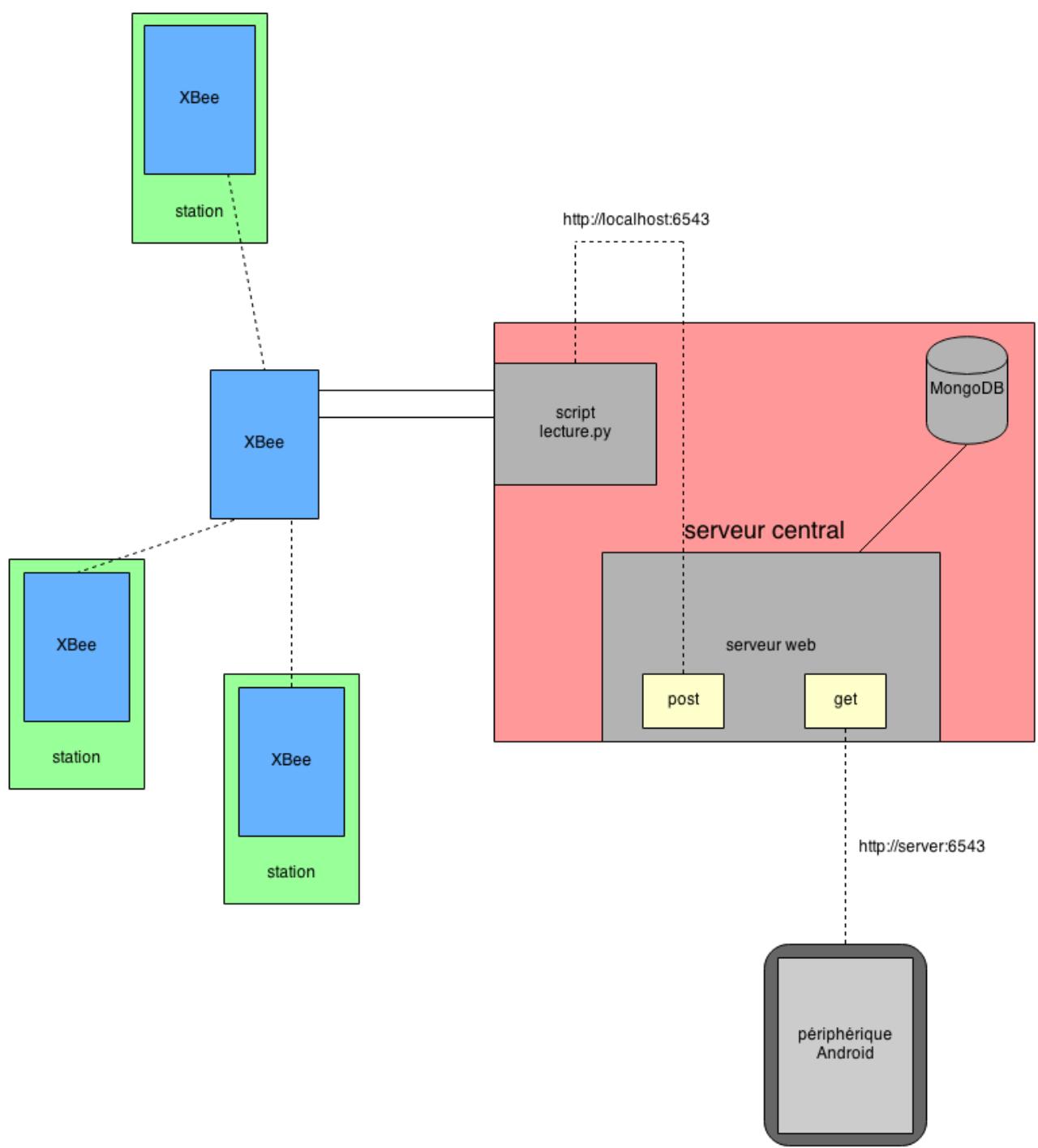


Illustration 1: Schéma général

## II Le réseau de capteurs

### 1 Les composants

#### 1.1 Généralités

Dans notre système, un nombre défini de stations est utilisé pour mesurer certains paramètres de leur environnement direct, tel que la température ou la luminosité. L'équipement doit être à la fois capable de prendre les mesures souhaitées, mais aussi de transmettre ces mesures pour qu'elles soient traitées.

C'est pour cette raison que toutes les stations sont composées d'une carte Arduino, d'un ou plusieurs capteurs et d'un module Xbee pour la transmission sans-fil.

#### 1.2 La carte Arduino

Arduino est un circuit imprimé en matériel libre sur lequel se trouve un micro-contrôleur programmable ainsi que plusieurs entrées/sorties analogiques et numériques pour un interfaçage avec d'autres circuits. Ce composant est utilisés pour des tâches très diverses comme la domotique ou le pilotage du robot. Il correspond parfaitement aux besoins et aux contraintes associées au projet.

Le modèle utilisé sera l'Arduino Uno. Si ce n'est pas la version la plus récente du circuit imprimé, elle sera amplement suffisant pour l'utilisation qui en sera faite.

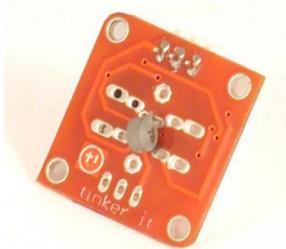


Illustration 2: Arduino Uno

#### 1.3 Les capteurs

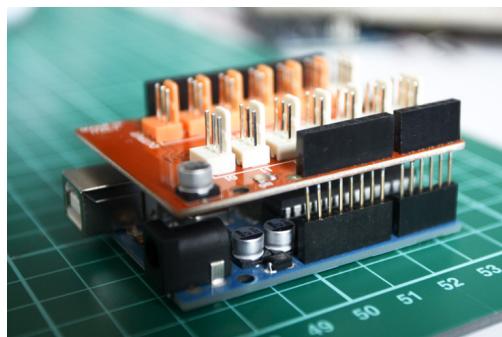
Les capteurs sont utilisés pour faire des mesures et interagir avec leur environnement, au

sens large du terme, puisqu'ils peuvent correspondre à des photorésistances, des thermistances, des capteurs de chocs, mais aussi un simple bouton poussoir. Les capteurs peuvent aussi bien émettre un signal analogique qu'un signal numérique, du moment que ceux-ci soient branchés sur les entrées/sorties qui conviennent.



*Illustration 3:  
Exemple de capteur -  
Thermistor*

Pour des questions de simplicité de montage, on utilisera un composant permettant de faire l'interfaçage des capteurs avec la carte Arduino : le sensor shield. Cet élément permet de réaliser les branchements facilement à l'aide de simples clips.



*Illustration 4: un Arduino et son  
Sensor shield*

## 1.4 Le module Xbee

C'est ce composant qui se chargera de transmettre les données relevées à un élément central, au travers d'une liaison sans-fil. Pour ce faire, c'est la technologie ZigBee qui sera utilisée.

ZigBee est un protocole de haut niveau basé sur la norme IEEE.802.15.4 permettant la communication courte distance, au même titre que le Bluetooth ou le WIFI. Il a la particularité de consommer très peu d'énergie et d'être extrêmement peu coûteux, ce qui lui confère un avantage non négligeable dans des applications de domotique où les éléments doivent souvent durer plusieurs années et ne sont pas facilement remplaçables.

Dans notre système domotique, chaque station est équipée d'un module Xbee pour pouvoir communiquer. De plus, un module central, chargé de réceptionner toutes les données en provenance des stations, est branché directement au serveur chargé de traiter les informations et de les rendre

disponibles.

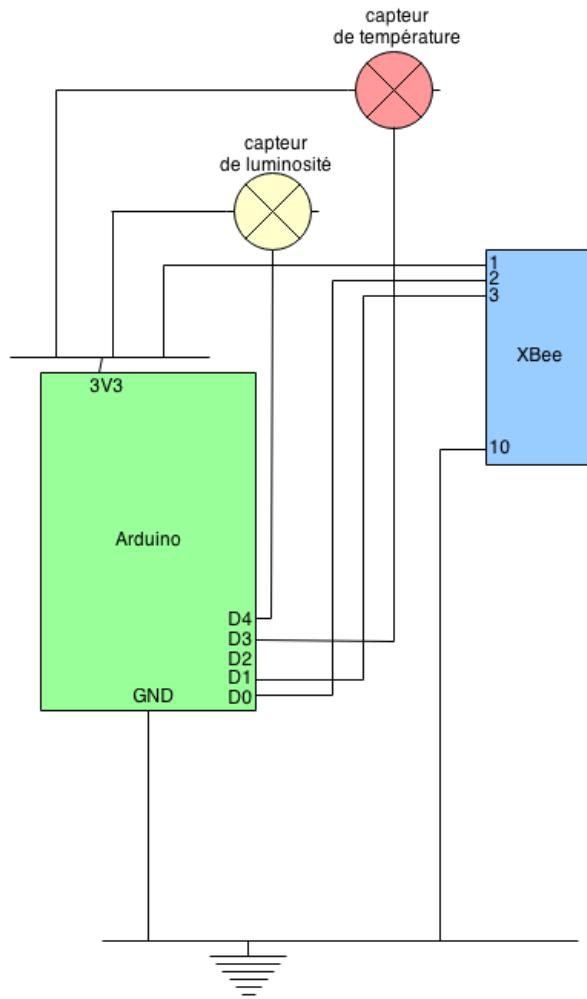


*Illustration 5: un composant XBee*

## 1.5 Schéma de montage

Le schéma de montage proposé explique comment les différents composants d'une station sont branchés entre eux. Etant donné le rôle transparent du Sensor Shield qui n'est là que pour faciliter les branchements, celui-ci n'apparaît pas.

On notera que ce qui est présenté ici n'est qu'un exemple de branchement, étant donné que chaque station assemblée peut être différente pour s'adapter aux besoins spécifiques liés à son environnement.



*Illustration 6: Schéma d'une station*

## 2 La configuration

### 2.1 Configuration du module XBee

Un certain nombre de paramètres doivent être configurés pour mettre en place la transmission des données. Pour ce faire, il suffit de brancher le module Xbee sur un ordinateur, soit directement par USB, soit par le biais de la carte Arduino. Le programme picocom est utilisé pour configurer le composant

On notera qu'il existe plusieurs façons de configurer ces composants. Il a été décidé d'aller au plus simple. C'est notamment pour cela qu'on utilisera un adressage local, amplement suffisant pour les communications qui devront être établies.

Les différents paramètres à configurer sur les modules XBee sont les suivants :

- **ATID** : *l'identificateur du réseau, qui sera le même pour tous les modules Xbee*
- **ATCE** : *indique qui sera le coordinateur dans le réseau. Seul un module aura ce paramètre configuré à 1, de préférence le module central qui récupère toutes les informations*
- **ATMY** : *définit l'adresse du module sur 2 octets ; la valeur de ce paramètre sera donc différente pour chacun des modules*
- **ATDH** : *utilisé seulement pour le mode d'adressage global, il devra être mis à 0 sur tous les modules.*
- **ATDL** : *indique l'adresse du destinataire des communication. Ce paramètre référencera l'adresse (ATMY) de l'élément central.*
- **ATWR** : *pour enregistrer la configuration*

Ci-dessous un exemple d'une procédure de configuration pour un module Xbee d'une station Endpoint du système de domotique :

```
> ATID 1988
OK
> ATCE 0
OK
> ATMY B1
OK
> ATDH 0
OK
> ATDL A1      //correspond au module Xbee qui réceptionne les données
OK
> ATWR
OK
```

Pour des raisons de gains de temps et d'automatisation de la procédure, un script bash a été créé et permet de configurer les modules très rapidement. Il suffit, après avoir branché le module à l'ordinateur, de lancer le script de la manière suivante :

```
./configureXBee </path/to/serial> <A1|B1|B2|B3>
```

Il a été décidé arbitrairement que A1 constituait l'adresse du module central qui récupère les données en provenance des stations et que B1, B2 et B3 correspondaient aux adresses des modules installés sur les stations de mesures.

## 2.2 Protocole de communication

Après avoir configuré le réseau de capteurs, il faut s'entendre sur un protocole de communication. Là aussi, ZigBee offre un grand nombre de possibilités, et là aussi le parti pris a été d'aller au plus simple en définissant manuellement la structure de la trame des paquets de données échangés entre les différentes nœuds du réseau, plutôt que d'utiliser le support applicatif fourni par la ZygBee Alliance et certainement adapté à des réseaux plus complexes.

Il a été décidé que chaque trame envoyée correspondrait à une et une seule valeur pour un capteur donné, et ceci afin que chacun jouisse d'une indépendance dans leur fonctionnement. Cela a l'avantage de ne pas contraindre des capteurs à attendre que d'autres capteurs aient effectué leurs mesures, mais cela a pour inconvénient de multiplier les communications.

En ce qui concerne le protocole pour la communication entre deux modules Xbee, chaque trame est structurée de la façon suivante :

07XE	ID STATION	ID CAPTEUR	VALUE
------	------------	------------	-------

Les deux premiers octets sont utilisés pour indiquer le début de la trame. 07XE correspond à une norme standard.

Les deux octets suivants correspondent à un identifiant unique de la station et permettant à l'application chargé du traitement des mesures de connaître la provenance des données réceptionnées.

Ensuite, les deux octets suivants correspondent à l'identifiant du capteur. Chaque type de capteur possède un identifiant qui lui est propre, et ce sur toutes les stations du système de domotique. Par exemple, tous les thermistors installés auront le même identifiant. Ceci permet à l'application qui traite les données des capteurs de savoir facilement à quoi exactement chacune des mesures correspond. Par contre, cela implique également qu'une station ne peut pas accueillir deux capteurs de même type ; cela s'avèrerait de toute manière inutile.

Enfin les deux derniers octets correspondent à la valeur transmise par le capteur. En pratique, toutes les mesures se situent entre 0 et 256, car il n'est pas nécessaire d'avoir plus de précision.

## 2.3 La programme embarqué par la carte Arduino

Pour insérer le code au sein du micro-contrôleur, il suffit de connecter la carte à un ordinateur par le biais d'un câble USB. Le constructeur fournit sur son site un IDE permettant d'écrire, de compiler et de téléverser le programme développé.

Le programme écrit au sein du micro-contrôleur de la carte Arduino met en œuvre tout ce qui a été expliqué plus haut. Il est conçu pour récupérer les données communiquées par les différents capteurs et de les transmettre au module Xbee via son port série. Cette procédure est effectuée environ une fois par seconde.

Un exemple de ce programme est fourni ci-dessous. On appellera qu'étant donné que les montages peuvent être différents d'une station à l'autre, il va de soi que le programme en est modifié lui aussi. Toutefois, le fonctionnement général restera identique.

Ci dessous, un exemple de code embarqué par une carte Arduino au sein d'une station Endpoint :

```
int id = 1;           //id de la station

int pinLight = A1;    //id du capteur température et pin correspondant
int pinTemp = A0;     //id du capteur température et pin correspondant

void setup() {
  // Le module XBee est branché sur le port série
  Serial.begin(9600);
  pinMode(INPUT, pinLight);
  pinMode(INPUT, pinTemp);
}

void loop() {
  int valueLight = map(0, 1023, 0, 255, analogRead(pinLight)); //pour convertir la
  sendFrame(id, pinLight, valueLight); //mesure entre 0 et 255

  int valueTemp = map(0, 1023, 0, 255, analogRead(pinTemp));
  sendFrame(id, pinTemp, valueTemp);

  delay(1000); //on recommence toutes les secondes
}

//fonction utilisée pour envoyer une trame
int sendFrame(int id, int pin, int value) {
  if(Serial.available()) {
    Serial.write(0X7E);
    Serial.write(pin);
    Serial.write(value);
  }
}
```

## **III La station domotique centrale**

### **1 Généralités**

Maintenant que le réseaux de capteurs a été mis en place, la prochaine étape va consister à récupérer et traiter les données émises par les stations. La station domotique centrale est en réalité un vulgaire ordinateur, sur lequel est branché un module Xbee. Nous avons déjà explicité dans la partie précédente le rôle, le fonctionnement et la configuration de cet élément central du réseau de capteurs, chargé de recevoir les mesures émises par les autres modules.

Il faut maintenant que l'ordinateur qui nous sert de station centrale puis traiter ces données. Un certain nombre de composants ont donc été mis en place pour remplir le rôle qui lui a été attribué.

### **2 Le programme qui lit les données**

L'objectif de ce programme est simplement de lire les informations reçues par le module Xbee et de les transmettre à un programme tiers chargé de les traiter. Ce programme tiers n'est autre qu'un serveur Web et sera détaillé dans la prochaine partie

Le programme développé à cet effet est un script python. En effet Python est un langage assez simple et intuitif, et dispose d'un certain nombre d'outils prêt à l'emploi. Il est utilisé et exécuté de la façon suivante :

```
python lireSerial.py </path/to/serial>
```

Bien évidemment, le programme doit être exécuté tant qu'il est nécessaire de lire des informations sur le module Xbee, c'est à dire pendant toute la durée où le système domotique est démarré. Le script pourra ainsi être lancé en tâche de fond, ou encore être installé comme un service.

### **3 Le Serveur Web**

Le serveur web doit remplir deux rôles : d'une part il doit enregistrer les données transmises par le programme qui lit les données dans une base de données, et d'autre part il doit rendre disponible ces mêmes données lorsque la demande en a été faite

Tout d'abord, rappelons que le choix de la solution à mettre en place a été laissé à l'appréciation des étudiants, bien qu'une liste des choix possibles et de conseils aient été apporté. Notre choix s'est porté sur un petit server web, écrit en python et peu connu : *Cornice*. Ce logiciel a la particularité d'être léger, de permettre le déploiement d'un serveur web très rapidement, et de fournir le squelette des scripts utilisés pour implémenter les différents services. En effet, plus qu'un serveur web classique tel qu'*Apache*, *Cornice* dispose nativement d'un framework facilitant et accélérant le développement, tout en assurant un certain niveau de sécurité. Il a été choisi de le lancer sur le port 6543 afin qu'il n'interfère pas avec d'autres programmes ayant pu être installés sur

le serveur et indépendants de ce projet.

Le format utilisé pour la transmission des données est le format JSON. Il permet de représenter de l'information structurée tout comme le format XML, et est souvent sélectionné pour le genre d'utilisation qui en est faite ici.

Notre serveur dispose d'un service « sensors » accessible depuis l'url « <http://<serveur>/<port>/sensors> » et avec deux méthodes :

- la méthode post, appelé par le programme qui lit les données et qui les transmet au serveur web. Les informations relatives à une trame, donc à un capteur, sont envoyées au format JSON via une requête de type post avec la structure suivante (les valeurs des attributs sont des exemples) :

```
{  
    nArduino: 1                                //identifiant de l'arduino  
    nSensor: B7                                 //identifiant du capteur  
    value: 141                                  //valeur de la mesure  
    measureTime: 1387028836                     //timestamp de la mesure  
}
```

- La méthode get a pour but de fournir les valeurs les plus récentes de chaque mesure de chaque station, au format JSON et selon la structure suivante (les valeurs des attributs sont des exemples) :

```
{  
    "arduinoss": [  
        {  
            "arduinoid": 1,  
            "sensors": [  
                {"sensorID": 1, "value": 16, "measureTime": 1387037097},  
                {"sensorID": 2, "value": 5, "measureTime": 1387037113}  
            ]  
        },  
        {  
            "arduinoid": 2,  
            "sensors": [  
                {"sensorID": 1, "value": 18, "measureTime": 1387037102},  
                {"sensorID": 2, "value": 4, "measureTime": 1387037117}  
            ]  
        },  
        {  
            "arduinoid": 3  
            "sensors": [  
                {"sensorID": 1, "value": 12, "measureTime": 1387028836}  
            ]  
        }  
    ]  
}
```

## 4 La base de données

Même si dans l'immédiat, une base de données n'est pas forcément utile, étant donné qu'on ne permet l'accès qu'au dernières valeurs enregistrées. Toutefois, la mise en place de cet outil permettrait d'implémenter plus de fonctionnalités, en proposant par exemple un historique ou des moyennes sur une période définie par l'utilisateur, voir même des courbes. Les améliorations proposées sont détaillées dans la fin de ce document.

Notre choix s'est porté sur MongoDB, un système de bases de données assez particulier puisque faisant partie des nouvelles bases de données dites NoSQL. Ce type de base de données orienté est très peu structuré et limité en termes de fonctionnalités SQL, mais propose une scalabilité très importante et se révèle très adapté lorsqu'on a affaire à un très grande quantité d'enregistrements. Même si on en est encore loin d'arriver à de telles problématiques dans notre projet, il faut rappeler que chacun des capteurs de chaque station envoie environ un enregistrement par seconde. On imagine bien alors les conséquences de ce fonctionnement lorsque le système est opérationnel en quasi-permanence, comme il a vocation à l'être.

Pour ce qui est de la structure des données, nous avons uniquement une « table » qui contient tous les enregistrements. Un enregistrement correspond simplement à une mesure pour un capteur sur une station. Il est donc structuré de la manière suivante :

```
{ "_id" : ObjectId("52aac60936044f76185baa99"), "nArduino" : 1, "nSensor" : 1, "value" : 13, "measureTime" : 1345643 }
```

Le champ « `_id` » est automatiquement créé par le SGBD et est utilisé comme clé primaire.

## 5 Automatisation

On l'a remarqué, plusieurs applications autonomes et interagissant entre elles sont nécessaires sur le serveur pour que notre système de domotique fonctionne. Ainsi, il faudrait lancer tous les services en veillant à respecter un certain ordre : la base de données avant le serveur web, le serveur web avant le programme qui lit les données, etc...

Pour des raison de simplification, un petit programme a été créé pour automatiser la procédure. Il s'agit d'un petit script Shell qui est exécuté de la manière suivante :

```
./python startAll.sh </path/to/serial>
```

Le script se charge de démarrer tous les programmes et reste en exécution. Lorsque le script est arrêté, il se charge alors d'ordonner l'arrêt de ces même programmes avant de se terminer.

Lorsqu'on est en train de debugger l'application, il est préférable de lancer les outils indépendamment et chacun leur tour afin d'avoir plus d'informations sur le fonctionnement de chaque outil et de pouvoir analyser plus finement les problèmes.

## IV Le périphérique mobile

### 1 Généralités

Le périphérique mobile a pour rôle de permettre à l'utilisateur de visualiser les informations relatives aux différents capteurs. Plus pour mettre en pratique les connaissances acquises au cours de cette UV que par réelle nécessité, l'objectif était de créer notre propre système Android personnalisé, en y modifiant certains paramètres et en y ajoutant des composants, afin de rendre le système unique. L'objectif était aussi d'intégrer dans l'image du système créé nativement l'application développée pour interagir avec le système, pour que l'utilisateur ait un outil clé en main dès l'installation de celui-ci.

### 2 *Création du nouveau produit*

Un produit correspond à une configuration spécifique pour un matériel donné. Notre but étant de créer un système personnalisé, un nouveau produit a logiquement été créé. Le produit a été nommé complètement arbitrairement « roki\_rakat » et hérite du produit « Pandaboard », configuration déjà écrite par le constructeur de la carte qui nous servira de périphérique mobile.

Notre nouveau produit a été défini au niveau de l'AOSP sous device/linaro/roki\_rakat et comporte l'arborescence suivante :

*AndroidProducts.mk* (makefile de base pour l'AOSP)  
*BoardConfig.mk* (configuration de base du produit)  
*CleanSpec.mk* (pour le nettoyage après une compilation)  
*overlay/* (dossier de ressources surchargées pour notre produit)  
*roki\_rakat.mk* (makefile principal du produit, hérite de la configuration Pandaboard)  
*vendorsetup.sh* (Pour ajouter roki\_rakat dans la liste des produits lors de la compilation)

Ci-dessous le fichier *roki\_rakat.mk* :

```
$(call inherit-product, device/linaro/pandaboard/pandaboard.mk) #on hérite de la  
#Pandaboard  
DEVICE_PACKAGE_OVERLAYS := \ #définition du dossier overlay pour la surcharge  
device/linaro/roki_rakat/overlay  
  
PRODUCT_PACKAGES += libusb-lib \ #intégration des composants créés  
libusb-lib-static \  
libbmp-lib \  
native_hellow \  
libbmp-lib-static \  
native-bmp \  
libbmp_Jni \  
com.android.utbm.lo52.test  
  
PRODUCT_NAME := roki_rakat  
PRODUCT_DEVICE := roki_rakat  
PRODUCT_BRAND := Android  
PRODUCT_MODEL := Full Android on roki_rakat
```

```
PRODUCT_COPY_FILES +=           #pour inclure la bootanimation personnalisée  
system/media/bootanimation.zip:system/media/bootanimation.zip  
  
include $(call all-subdir-makefiles)
```

Le fichier roki\_rakat.mk a été modifié afin d'intégrer les différents éléments qui ont été ajoutés : la bootanimation personnalisée, le composant libbmp, l'interface JNI correspondante, le fond d'écran par défaut et l'application de domotique. La notion d'héritage s'avère très utilisée ici puisque seules les configurations qui diffèrent de la configuration par défaut du constructeur de la carte sont écrites.

### **3    *Intégration de la bootanimation personnalisée***

Une bootanimation est une animation qui est affichée lors du démarrage du système d'exploitation. Il en existe une par défaut pour le système d'exploitation d'Android, celle-ci étant facilement modifiable.

Dans un premier temps il a été nécessaire de créer un fichier « desc.txt » et d'y intégrer les directives suivantes :

```
960 800 30  
p 1 0 part0  
p 1 0 part1  
p 0 0 part2
```

Le premier et le deuxième nombre correspondent respectivement à la largeur et à la longueur (en pixel) de l'animation. Le troisième nombre définit son nombre d'images par secondes (fps).

Chacune des lignes suivantes correspond à la définition de chacun des dossiers contenant les images de l'animation.

### **4    *Intégration du composant libbmp***

Le composant libbmp a été ajouté à notre système. Pour ce faire, il a été ajouté un dossier /external/libbmp qui contiendra les ressources nécessaires. De plus, comme pour tout composant intégré à l'AOSP, un fichier Android.mk se trouve à la racine de ce dossier et permet de donner les directives nécessaires à l'intégration du composant dans l'image du système.

En ce qui concerne le binaire natif, le procédé a été le même. Un nouveau composant libbmp\_natif a été créé au sein du dossier /external. Une interface JNI a été développée, de même qu'un fichier Android.mk/

Enfin, une extension de framework a été implémentée, utilisant l'interface JNI précédente. Un fichier de permission a également été créé pour rendre le fichier .jar correspondant accessible.

## **5 Modification du fond d'écran par défaut**

Afin de personnaliser notre système Android, nous avons été amené à personnaliser le fond d'écran. Le dossier « overlay » est utilisé pour surcharger toutes les ressources du systèmes, tel que les icônes et autres fond d'écran. L'arborescence suivante a donc été créée :

```
overlay/linaro-ics-tp/frameworks/base/res/res
```

- Drawable-xlarge-nodpi
- Drawable-large-nodpi
- Drawable-nodpi

Chacun des dossier « Drawable » possède les ressources utilisées pour pouvoir s'adapter au périphérique utilisé. Les dossiers « Drawable-xlarge-nodpi » et « Drawable-large-nodpi » regroupent les ressources adaptées à une résolution 1920 x 1280 pixels, et le dossier « Drawable-nodpi » pour une résolution de 960 x 800 pixels

## **6 Intégration de l'application de domotique**

Pour que notre application de domotique soit finalement intégrée à l'image de notre système Android personnalisé produite lors de la compilation, il est nécessaire de modifier le makefile contenant les instructions nécessaires pour la compilation de notre produit (ici roki\_rakat.mk), de manière à ce qu'apparaisse les lignes suivantes :

```
PRODUCT_PACKAGE += \  
RokiRakat
```

Le nom du package utilisé (RokiRakat) correspond au nom de l'application au sens de ce qui est indiqué dans son propre makefile.

## V L'application domotique

L'application de domotique développée est utilisée pour permettre à l'utilisateur d'interagir avec le système de domotique. Les données sont hiérarchisées et organisées par station.

L'application a été conçue pour s'adapter en fonction du format de l'écran. Des « Fragment » ont été utilisés à cet effet. Au final, même si l'application est limitée en terme de design, elle n'en reste pas moins ergonomique et intuitive.

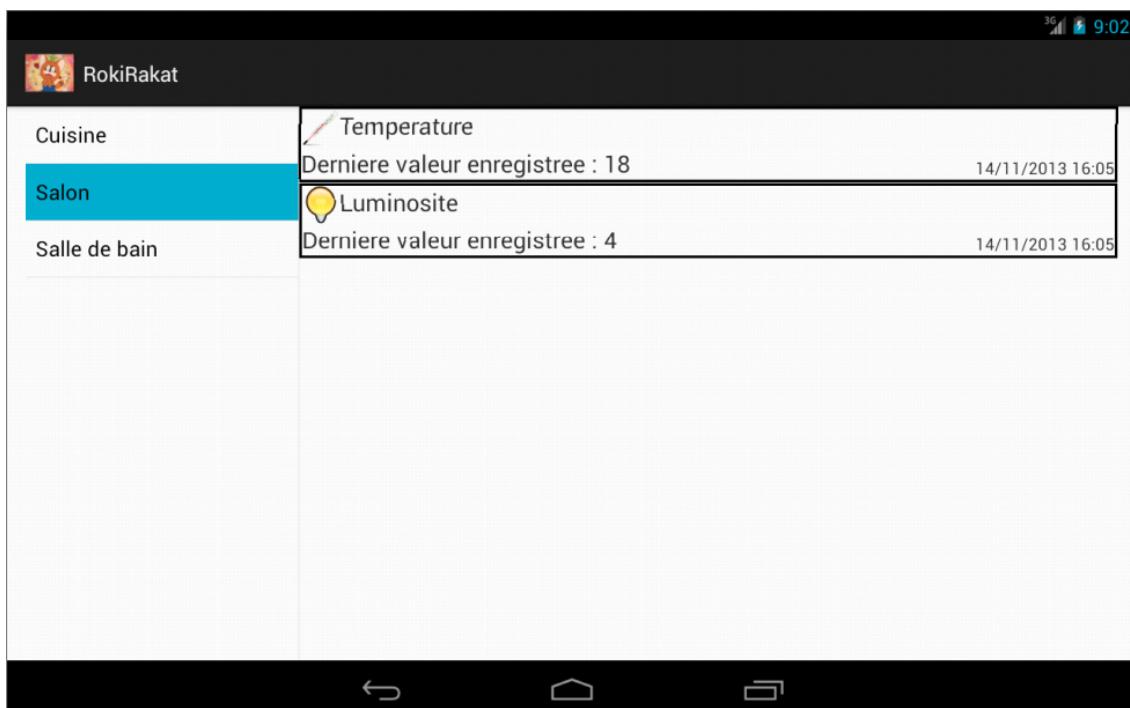
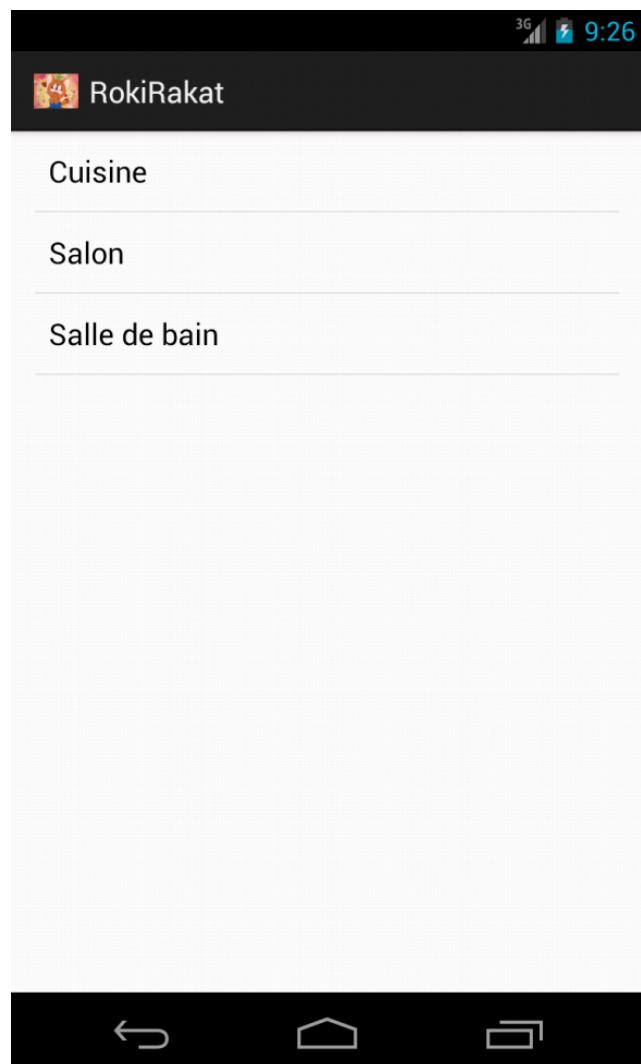


Illustration 7: RokiRakat sur Tablette

Lorsqu'on utilise une tablette, le format de l'écran est généralement plus grand que sur un téléphone. L'application exploite cet espace supplémentaire en affichant toutes les informations dans la même activité. Une sélection d'une station sur la liste de gauche aura pour effet de rafraîchir la liste de droite et de donner des informations sur ses capteurs. L'application s'en retrouve dynamique et réactive.

Le problème se pose lorsqu'on dispose d'un smartphone avec un écran de taille limitée. Il est alors déconseillé d'afficher un trop grand nombre d'informations, au risque de réduire la lisibilité. L'application s'en retrouve modifiée sur les écrans de ce type.

Tout d'abord, une première activité affiche la liste des stations disponibles :



*Illustration 8: RokiRakat sur Smartphone (1)*

Lorsqu'une station est sélectionnée, une deuxième activité s'affiche avec l'ensemble des capteurs disponibles :



Illustration 9: RokiRakat sur Smartphone (2)

## Conclusion

Notre système, bien que minimal, est fonctionnel et prêt à être utilisé. Il est ainsi possible, si ce n'est de contrôler activement un élément de la maison comme un volet électrique ou un thermostat, de relever certains variables qui auront été mesurées par nos capteurs placés à des endroits stratégiques de notre maison. Nous avons bien conscience de l'imperfection du projet, au sens où il s'agit d'un système minimal. Néanmoins, l'objectif n'aura pas été de produire un système très complexe, comme il en existe sur le marché aujourd'hui, mais de montrer qu'avec peu de moyens, et des compétences minimales, il est tout à fait possible de créer son propre système domotique de A à Z, qui plus est en n'utilisant quasiment que des produits dits « Open Source ».

Rien dans le projet n'était en soi très complexe ou insurmontable ; la synchronisation entre toute une série de composants du système aura été probablement la partie la plus fastidieuse. Il a été indispensable de commencer par bien comprendre les processus du système avant d'implémenter quoi que ce soit.

Comme améliorations possibles de notre projet, nous avons pensé à un certain nombre de fonctionnalités qui apporteraient de la valeur ajoutée à notre système. D'abord, il est tout à fait possible, en conservant les composants de base, de créer un système domotique actif, où il serait possible de réellement « commander » des éléments de la maison à distance. En effet, actuellement le système est unidirectionnel : des informations sont uniquement envoyées depuis les stations domotiques vers l'utilisateur. On pourrait maintenant imaginer pouvoir allumer une lampe, voir ouvrir la porte du garage directement depuis l'application.

Comme nous l'avons vu dans la partie traitant du serveur central du système, nous stockons toutes les données émises par les sondes dans une base de données. Or, l'application n'exploite pas réellement les fonctionnalités d'historique qu'apporte cette base de données, dans le sens où seulement les valeurs les plus récentes sont affichées. On pourrait proposer au sein de l'application d'afficher des moyennes sur un intervalle de temps défini, voir même d'afficher des courbes et autres diagrammes.

Ces deux améliorations ne sont que des exemples et ne représentent en aucun cas une liste exhaustive. Il est même fort probable qu'un système domotique puisse constamment être amélioré et n'est donc jamais réellement terminé.

## **Annexes : outils à installer sur le serveur**

Plusieurs composants interviennent dans le fonctionnement de notre serveur. Aussi, si le lecteur souhaite tester par lui même le système après avoir récupérer les sources associées, il est impératif d'installer les applications suivantes :

- Une environnement Python fonctionnel
- VirtualEnv pour séparer les environnements Python
- La bibliothèque Python SERIAL
- La bibliothèque Python JSON
- La bibliothèque Python pymongo
- Le serveur Python Cornice
- La base de données MongoDB