Projet de Fin d'Études

–

Rapport de stage

Développement d'un système de capture du rythme cardiaque dans un environnement M2M

Résumé

Le but du stage était de réaliser une application de démonstration de faisabilité de l'intégration du framework Machine-to-Machine (M2M) dans un système mobile. L'application verticale de démonstration comporte deux utilisateurs principaux :

- le patient, avec son téléphone Android connecté à des capteurs,
- l'infirmier, avec son ordinateur de bureau, monitorant son patient.

L'application globale devait suivre de règles bien précises, données par un standard – la norme Etsi pour des applications eHealth.

Jessica Hornik

Sous la direction de
Frédéric Camps (Laas – Cnrs)

Tuteur pédagogique
Aurélie Hurault (Inpt – Irit)

19 mars, 2014 – 7 septembre, 2014

Table des matières

Remerciements

Je tiens à remercier toutes les personnes qui ont fait que ce stage a pu se dérouler agréablement. En particulier, les personnes que je cite ci-dessous.

Jean Arlat, directeur du Laas, pour m'avoir accueillie au sein de son établissement durant ce stage.

Aurélie Hurault, ma tutrice Enseeiht, pour m'avoir accompagnée tout au long de ce stage.

Frédéric Camps, mon maître de stage au Laas, pour avoir été présent, compréhensif et agréable durant mon stage.

Introduction

Lors de mon stage, j'ai été affectée au service *Informatique : Développement, Exploitation et Assistance (IDEA)* du *Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)*, et ai travaillé en étroite collaboration avec le groupe de recherche *Services et Architectures pour les Réseaux Avancés (SARA)*.

Il m'a été demandé de démontrer la faisabilité de l'intégration d'une plateforme développée par l'équipe *SARA* vers les systèmes mobiles Android par l'exemple. Mon but premier était donc de développer une application utilisant des communications avec la plate-forme existante afin de prouver l'interopérabilité avec un système mobile.

Nous verrons par la suite que cet objectif s'est beaucoup élargi en se transformant en l'objectif suivant :

Objectif :
Permettre le développement d'applications mobiles en embarquant la plate-forme dans un système Android après étude de l'architecture adaptée, puis développer une application tutoriel ainsi qu'une application de démonstration sous Android.

1 Présentation du contexte

1.1 Laas

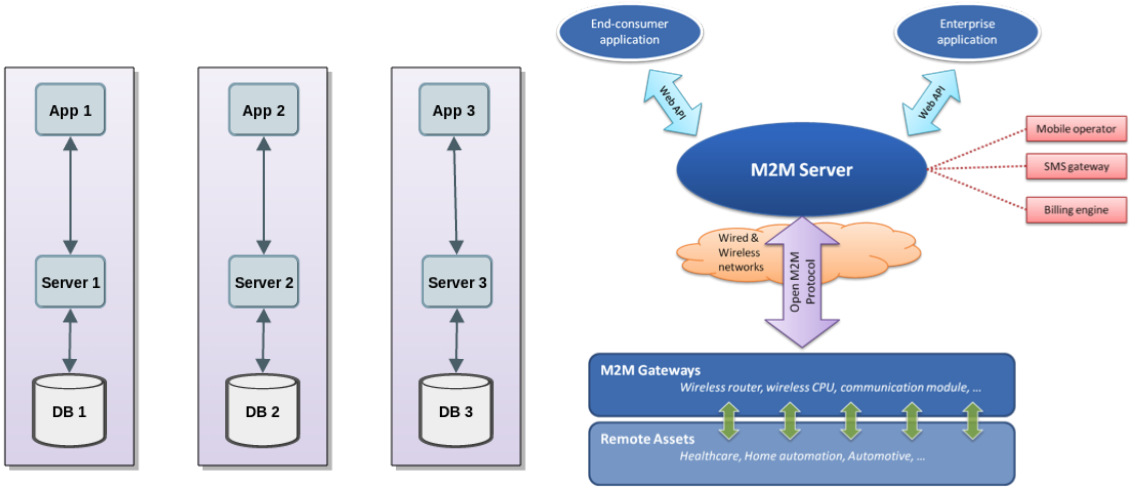
Le *LAAS* est une unité propre du CNRS rattachée à l'Institut des Sciences de l'Ingénierie et des Systèmes (INSIS) et à l'Institut des Sciences de l'Information et de leurs Interactions (INS2I). Situé à Toulouse, il est associé par convention à cinq membres fondateurs de la COMUE "Université de Toulouse" :

— Université Paul Sabatier (UPS),
— Institut National des Sciences Appliquées de Toulouse (INSA),
— Institut National Polytechnique de Toulouse (INP)
— Université du Mirail (UTM)
— Université Toulouse 1 Capitole (UT1).

Le *LAAS* mène des recherches en sciences et technologies de l'information, de la communication et des systèmes dans 8 thèmes scientifiques :

— Informatique critique
— Réseaux et communications
— Robotique
— Décision et optimisation
— HF et optique : de l'EM aux systèmes
— Nano ingénierie et intégration
— Micro nano bio technologies
— Gestion de l'énergie

L'équipe *SARA* est l'une des équipes du thème Réseaux et Communications. Elle s'intéresse notamment à l'analyse, l'évaluation des performance et au prototypage des logiciels et des



(a) Ancienne architecture – verticale (b) Architecture M2M – horizontale

FIGURE 1: Architecture verticale ⟶ horizontale

plate-formes de communication.

1.2 M2M

Machine-to-Machine (M2M) définit un mode de communication entre machines qui vise à être autonomique, c'est-à-dire qui peut fonctionner, se configurer, etc. sans intervention humaine.

Le but est de créer une façon de présenter et de structurer les données de manière à ce que les applications ne soient non plus verticales (cf. figure 1a) mais horizontales (cf. figure 1b). De manière plus explicite, il s'agit non plus que chaque application se retrouve isolée avec son propre serveur et ses propres ressources, mais plutôt de permettre un échange de données qui deviennent communes et accessibles à travers de multiples applications.

C'est cet objectif qui a donné lieu à l'élaboration de normes afin que toutes les applications communiquent de la même manière. L'une d'entre elles est la norme *European Telecommunications Standards Institute (ETSI)*.

1.3 Norme ETSI

Le principe de la norme *ETSI* est de structurer les données de sorte que l'accès soit générique et accessible à tous de la même manière. C'est-à-dire que chacun/chaque application pourra gérer ses données aussi bien que les mettre en commun.

Ce que l'*ETSI* propose, c'est un modèle comme celui de la figure 2. C'est-à-dire que cette norme comprend deux parties :

1. la présentation, sous la forme d'un arbre de ressources
2. la structuration, sous la forme de *Service Capability Layer (SCL)* et d'applications

La figure 3 montre le modèle de communication pensé par l'ETSI entre les différents blocs. Les différents blocs sont :

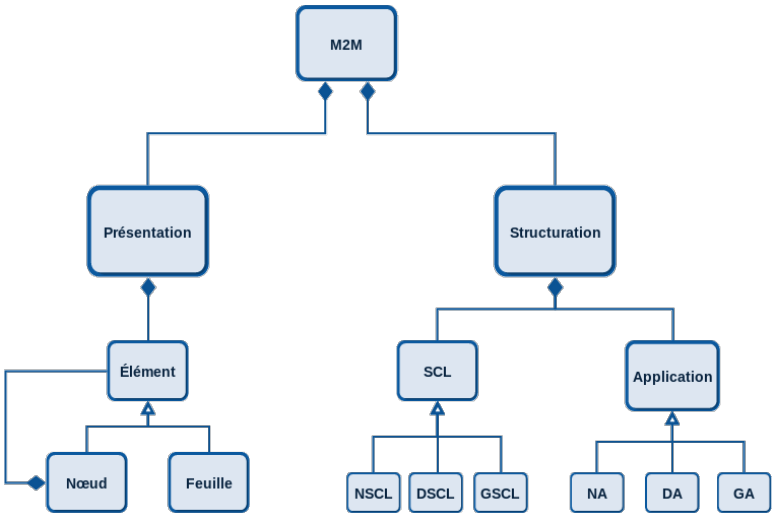


FIGURE 2: M2M ETSI

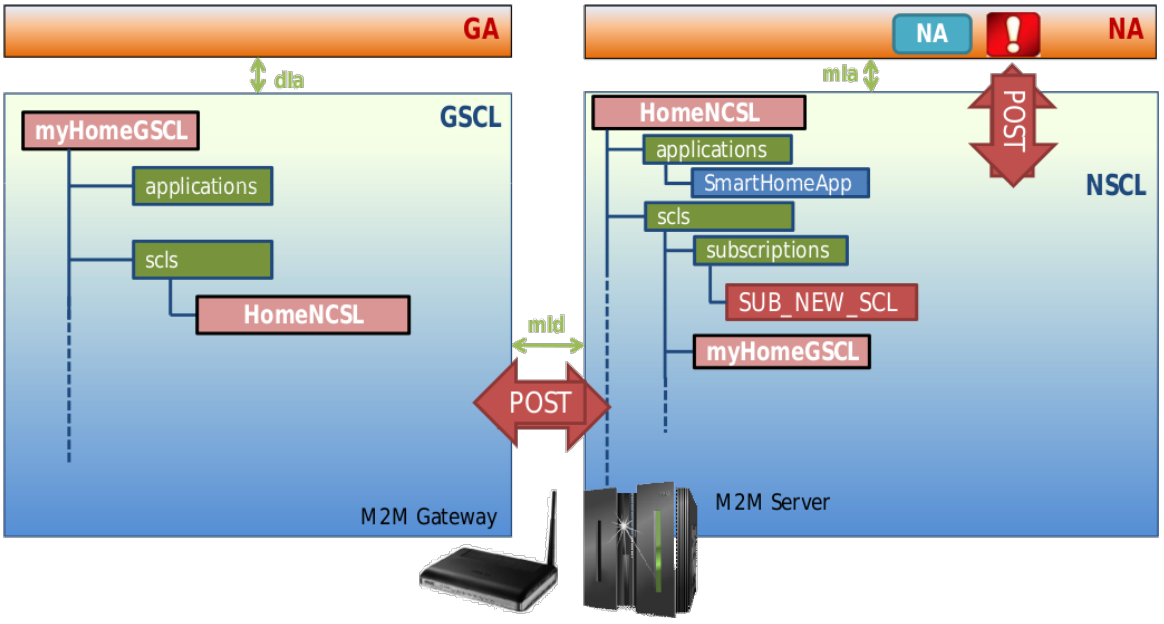


FIGURE 3: Communication ETSI

— *Network Application (NA)* : une application, dans le schéma *SmartHomeApp*, qui se trouve sur un ordinateur par exemple
— *Network Service Capability Layer (NSCL)* : contient l'*arbre de ressources* stocké dans une base de données, ainsi que la gestion des communication (sur un ordinateur)
— *Gateway Service Capability Layer (GSCL)* : pareil que le *NSCL*, mais sur une gateway (pour faire la passerelle entre des capteurs et l'*arbre de ressources*)
— *Gateway Application (GA)* : pareil que *NA*, mais sur la gateway

La communication interblocs s'effectue par des requêtes REST (POST, GET, DELETE, PUT).

Nous pouvons considérer l'exemple suivant :
Il y a une application *NA* sur l'ordinateur, qui s'appelle SmartHomeApp. Son but est de

récupérer les informations que des capteurs vont recueillir. La racine de l'arbre de ressources dans la gateway s'appelle *myHomeGSCL*, et celle du *NSCL* s'appelle *HomeNSCL*. La gateway fait le pont entre les capteurs et la plate-forme.

Le déroulement des événements va être le suivant :

1. Les deux *SCL* s'enregistrent mutuellement (sous une ressource appelée `scls`)
2. L'application bureau s'enregistre auprès de l'*arbre de ressources* du *NSCL*
3. L'application bureau souscrit aux nouvelles données (`ContentInstances`) d'un certain capteur
4. Dès que le capteur enregistre une nouvelle donnée, l'application *SmartHomeApp* reçoit une notification

1.4 Plateforme oM2M

Le projet *One M2M (OM2M)* a vu le jour en été 2013, lorsque la norme *ETSI* est arrivée afin de formaliser la manière dont les échanges de données et le stockage se feraient. C'est un projet *Eclipse* open source.

Au moment de mon arrivée en stage, une première implémentation du projet *OM2M* avait été faite pour fonctionner sur un serveur.

La plate-forme venait d'être déployée sur un serveur afin que cette implémentation tourne avec des capteurs installés dans un bâtiment de test, le bâtiment ADREAM. Celui-ci est entièrement autonome en énergie, et a pour vocation d'être entièrement auto-régulé et auto-géré, avec par exemple des capteurs de température et de luminosité.

Pour donner une idée un peu plus concrète de ce que le *M2M* permet, un exemple très simpliste serait :

— Une personne entre dans la pièce,
— un capteur de présence détecte cette personne, et enregistre sa valeur dans les ressources (base de données),
— une notification est levée, comme quoi il faut allumer la lumière,
— la lumière s'allume.

Ou alors, dans le cadre d'un projet de santé :

— La personne agée n'est pas entrée dans la cuisine pour se restaurer,
— un rappel sonore (ou autre) est émis,
— si toujours aucune réaction, une alerte est donnée pour envoyer un médecin sur place.

L'interface web en figure 4 permet de visualiser l'*arbre de ressources*.

2 Stage en pratique

2.1 Sujet et objectifs du stage

Avant le début de mon stage, le sujet a été définit comme suit.

> Développement d'une application de santé dans le cadre d'un projet M2M. Ce développement comprend une phase d'analyse des besoins et de compréhension, celle-ci

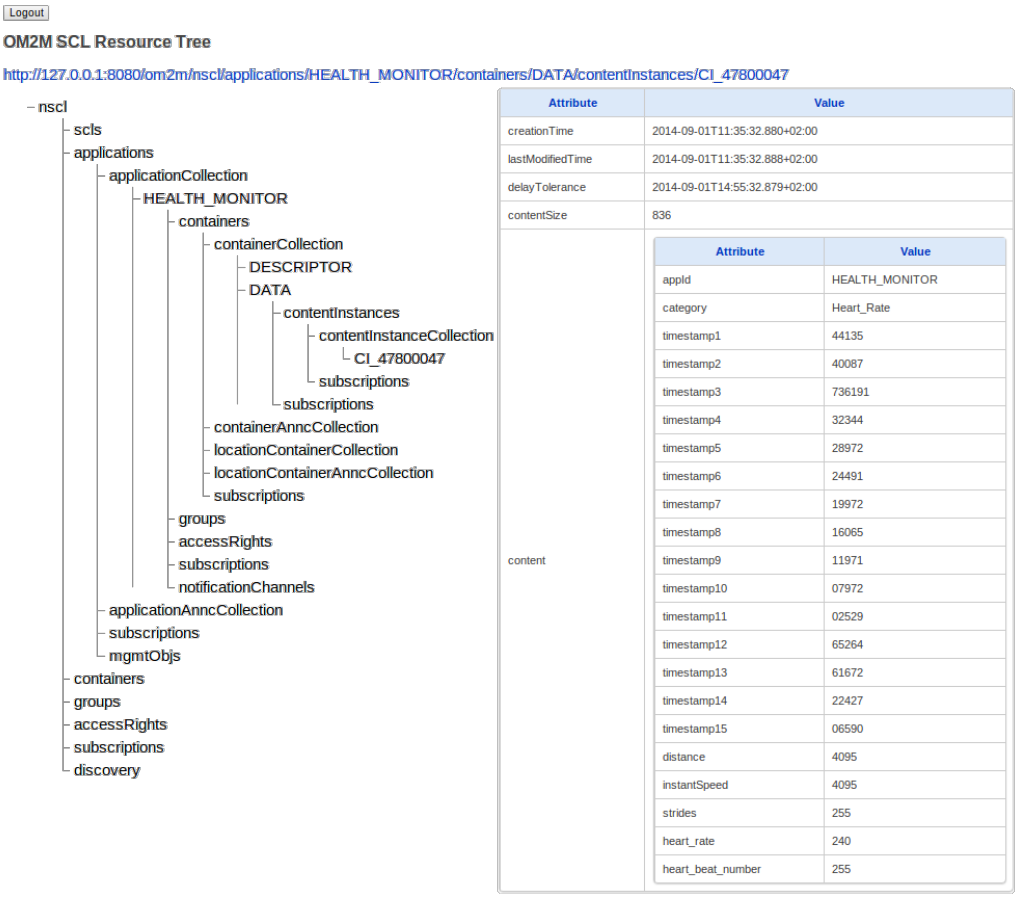


FIGURE 4: Interface web – visualisation de l'arbre de ressources

se fera dans un formalisme UML/SysML. Une partie du développement se fera sur Android, cette application devra dialoguer avec un système M2M développé au LAAS/CNRS. Le système M2M suit le standard M2M de l'ETSI. Le logiciel devra ensuite être intégré à une plate-forme de démonstration du projet ADREAM.

Les technologie mises en œuvre sont les suivantes : J2EE, Android, C/C++, JNI, XML, méthode Agile pour le développement.

Le sujet a pu évoluer selon les besoins et la découverte des différentes priorités du projet. Il s'est transformé en l'objectif suivant :

Objectif :
Permettre le développement d'applications mobiles en portant la plate-forme dans un système Android, puis développer une application tutoriel ainsi qu'une application de démonstration sous Android utilisant cette nouvelle plate-forme embarquée.

La démarche explicant ce changement est expliquée dans la partie suivante.

2.2 Méthodologie utilisée

Au long du projet, chacun des stagiaires du groupe travaillant sur la plateforme oM2M a présenté son travail dans le cadre de réunions de suivi. Ces réunions se sont déroulées pour le lancement du travail toutes les semaines, puis se sont espacées à deux fois par mois.

Au cours de ces réunions, j'ai pu avoir recours à la modélisation UML/SysML afin de me fixer les idées et pour permettre aux participants de mieux appréhender mes propos.

J'ai travaillé de manière incrémentale, c'est-à-dire en procédant à des ajouts de fonctionnalités au fur et à mesure, selon des cycles de conception-développement-intégration.

Le calendrier des tâches a été mis à jour régulièrement par mon superviseur ou moi tout au long du stage.

2.3 Outils de travail

Outils de travail :

- Ubuntu
- Eclipse
- Git
- Redmine (tâches et dépôt)
- Tablette/téléphone Android
- Capteur de fréquence cardiaque (HxM Zéphyr)
- ...

Technologies utilisées :

- Java
- Android
- JavaEE
- REST (client/serveur)
- XML
- XSD
- ...

2.4 Planning du stage

De manière très épurée, nous pouvons considérer que mon stage s'est déroulé selon les étapes suivantes :

1. Découverte du projet (fin mars)
2. Veille technologique (fin mars)
3. Formation Android (mi-Avril)
4. Formation M2M (1 avril)
5. Évaluation de l'intégration dans un système mobile (mi-Avril)

6. Étude du Device Service Capability Layer (DSCL) sur Android (fin Avril)
7. Développement itératif (Avril - Juillet)
   — Découverte d'un problème de compatibilité Android (librairie, architecture non adaptée)
   — Évaluation d'une solution
   — Développement de la solution associée
   — Tests fonctionnels
8. Tests (Juin - Juillet)
9. Intégration, passation du code (Juillet - Août)
10. Rapport de stage (Juillet - Août)
11. Présentation orale (Août - Septembre)

3 Étude comparative DSCL / NSCL

Au début de mon stage, le but était de concevoir et réaliser une application Android afin de démontrer la faisabilité de l'utilisation de la plate-forme à partir d'un terminal mobile.

Cependant, très rapidement, il est apparu une question d'architecture pour l'application verticale de bout-en-bout : serait-il plus judicieux de montrer la possibilité d'utiliser la plate-forme existante à distance à partir d'un terminal mobile ? Ou alors de porter la plate-forme pour être déployable sur un système Android ?

Pour parvenir à une conclusion, il m'a fallu effectuer une étude comparative entre différentes structurations d'application verticale. Cette partie sera consacrée au résultat de l'étude.

3.1 Contexte

Nous voulons développer une application verticale ayant l'architecture globale illustrée en figure 5, avec des communications M2M. Pour aboutir à une architecture plus détaillée, des

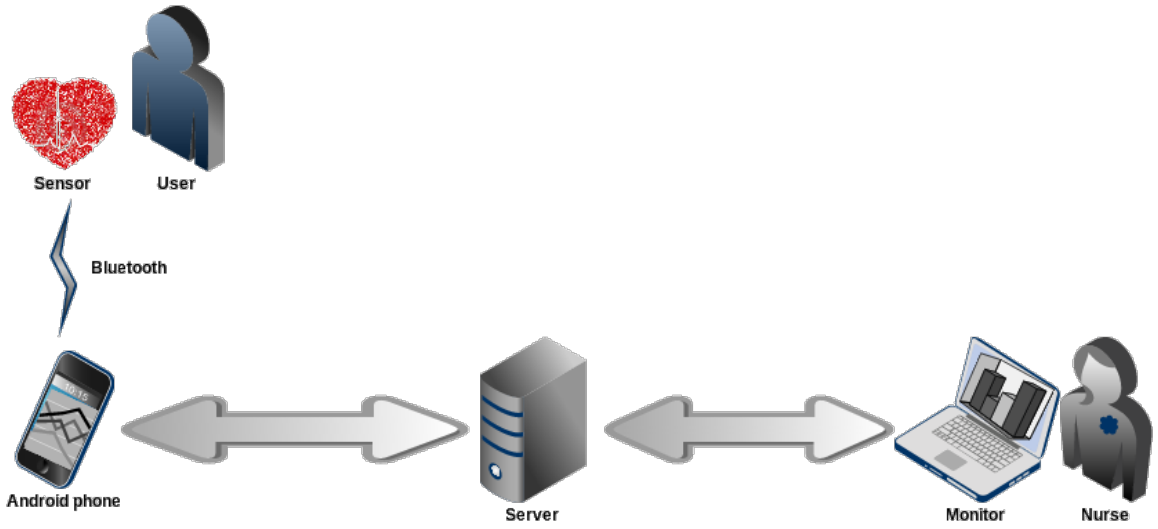


FIGURE 5: Architecture globale

questions se posent, autant pour la performance, que pour l'économie d'énergie, ainsi que pour la généralisation de l'application de manière plus "orientée" *M2M*.
Nous présentons ici trois possibilités d'architecture :

- DSCL développé pour Android et inclus dans l'application mobile, et :
  - soit les données des capteurs sont stockées dans le DSCL,
  - soit les données des capteurs sont stockées dans le *NSCL*,
- *GSCL* déporté vers un serveur distant (voire utilisation d'un *NSCL* seul)

3.2 DSCL inclus dans l'application mobile

La figure 6 montre l'architecture que l'on obtiendrait dans le cas où l'on choisirait de développer un plugin DSCL et de l'intégrer à l'application Android. En orange figurent les parties à développer.

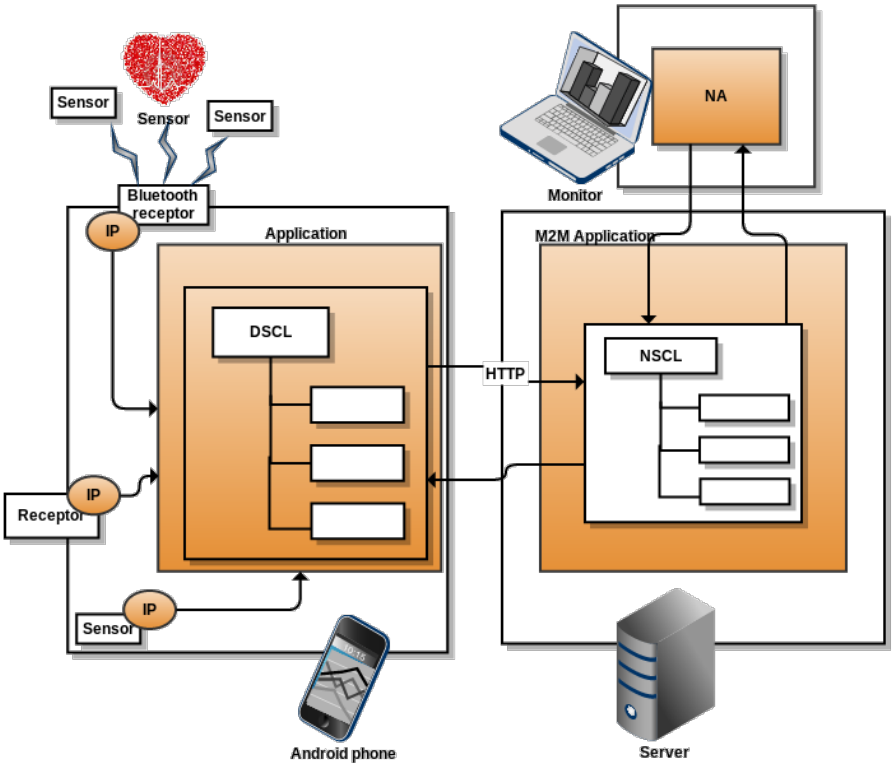


FIGURE 6: Architecture avec DSCL inclus dans l'application mobile

3.2.1 Description et actions

Il faut développer un plugin DSCL pour Android, et ensuite l'utiliser dans l'application Android, et développer les parties IP (Interworking Proxy) pour faire la connexion directement

avec le DSCL.
Avant de développer le plugin DSCL, il faut :

— vérifier la compatibilité SCL et Android sur un processeur ARM32
— vérifier si SCL peut fonctionner sur un process de type machine à états (Android).

Encore ici, deux possibilités peuvent être implémentées.

— soit les données sont stockées dans l'abre de ressources du DSCL,
— soit les données sont stockées dans l'arbre de ressources du NSCL.

3.2.2 DSCL contient les données des capteurs

1. **Avantages**
   Les données sont stockées en local, et le téléphone peut s'en servir pour tracer des graphes, par exemple, et informer l'utilisateur (le patient) sans accéder à internet.
2. **Inconvénients**
   — Le mobile est considéré comme une sorte de serveur de données, et donc la consommation d'énergie est fortement accrue, surtout lorsque la communication est de type TCP/IP
   — La souscription d'une machine à la donnée écrite sur le téléphone est soumise aux restrictions d'usage d'internet par l'utilisateur Android (sans connexion de l'utilisateur, pas de données).

3.2.3 NSCL contient les données des capteurs

1. **Avantages**
   Les envois de données sont faits par des POST au NSCL, qui les fait remonter aux applications abonnées. C'est donc le serveur distant qui sert de stockage de données (pas de sur-consommation quand les applications desktop veulent avoir des données).
2. **Inconvénients**
   — Si on choisit de ne rien garder en local, alors chaque accès aux données du patient correspond à un accès internet (avec le temps de latence correspondant)
   — Si on choisit de garder des éléments aussi en local, alors les données sont dupliquées (avec un risque de non-cohérence entre les deux – BDD Android et NSCL)

3.2.4 Synthèse

Données dans	DSCL	NSCL
Sur-consommation	Accès par les applications distantes	Accès par le téléphone
Cohérence	Risque de non-cohérence	Assurée
Implémentation	Vérifier la compatibilité Android	
Avantages	Généralisation et intégration à d'autres applications Android possible	

3.3 GSCL déporté vers un serveur distant

La figure 7 montre l'architecture que l'on obtiendrait dans le cas où l'on choisirait de déporter le GSCL sur le serveur distant, et développer à part l'application Android. En orange figurent

les parties à développer.

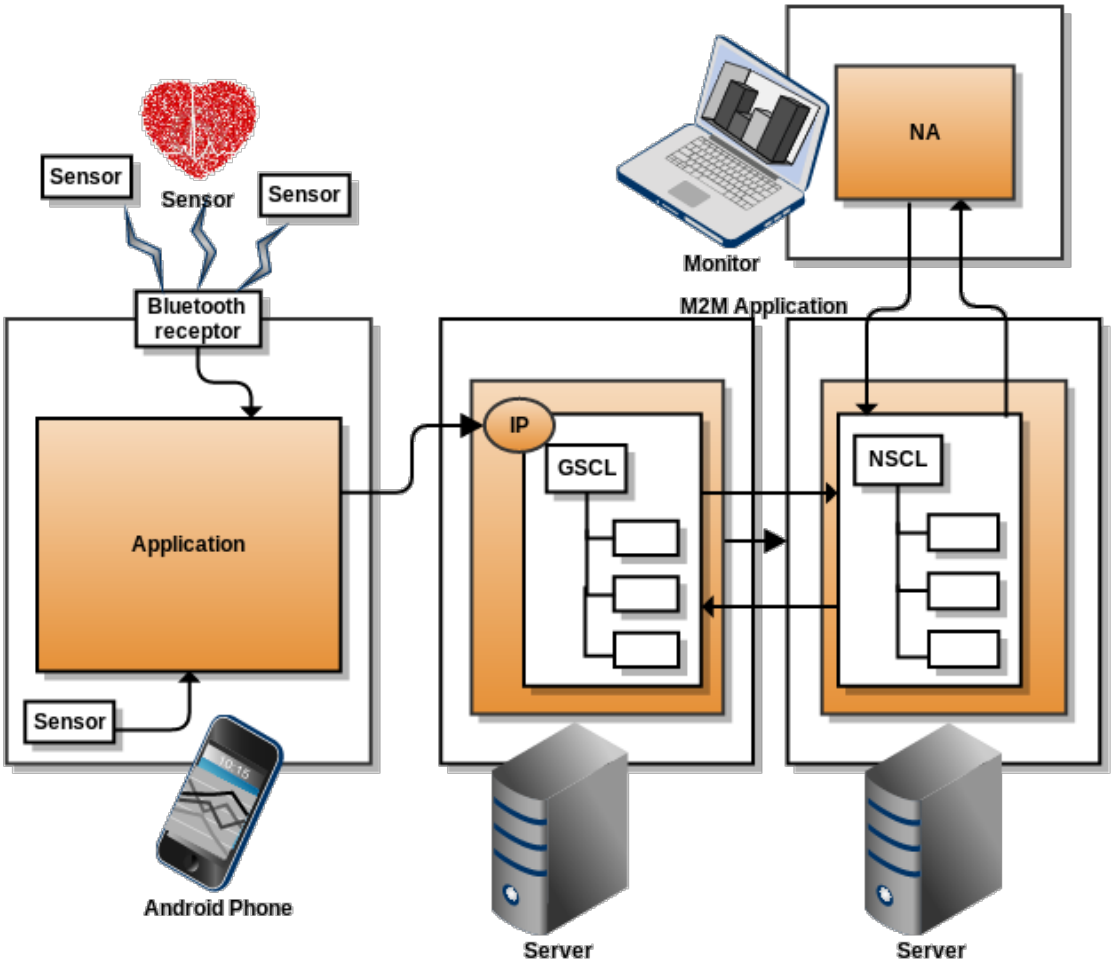


FIGURE 7: Architecture avec GSCL déporté vers un serveur distant

3.3.1 Avantages

— La consommation d'énergie est moindre pour le système d'abonnement
— Les données sont accessibles tout le temps

3.3.2 Inconvénients

— L'application mobile a deux choix :
  — elle envoie au gateway chacune de ses données (elle ne joue qu'un rôle de transmission)

— elle filtre les résultats à transmettre (et l'application tend à redevenir "à l'ancienne" : privée)
— Certaines données sont dupliquées (en base de données locale et dans le gateway), ou alors consommation d'énergie quand le patient veut accéder à ses données.

3.4 Synthèse

Voici un tableau récapitulatif des avantages et inconvénients de chacune des architecture. Il ne reste plus qu'à attribuer un poids d'importance à chacun des points.

Données dans	DSCL	NSCL	GSCL
Sur-consommation	Accès par les applications distantes	Accès par le téléphone (ou duplicité données)	
Cohérence	Risque de non-cohérence	Assurée	
Implémentation	Vérifier la compatibilité Android		Aucune
Généralisation	Possible et facile		Non-assurée

Il est donc apparu que la plate-forme aurait pu aisément être utilisée de manière distante à partir d'un mobile, mais que pour l'intérêt du projet ainsi que de mon stage, le sujet devrait dériver vers un portage de la plate-forme *M2M* entière sur le système Android. Je m'y suis alors attelée.

4 Portage de la plateforme M2M sur Android

Le portage de la plate-forme *M2M* a dû s'effectuer en tenant compte des contraintes liées à Android. C'est pour cela qu'il y a eu différentes sortes de changements à appliquer : des changements liés aux bibliothèques utilisées, et des changements structurels.

4.1 Structuration logicielle M2M

La modélisation au niveau logiciel de l'implémentation déjà existante est présente en figure 8.

Des packages comportent chacun des utilités différentes, comme par exemple un package contenant les représentations des ressources, un s'occupant de l'accès en base de données, d'autres s'occupant de router les requêtes ou d'en formuler (création, déletion, récupération, etc. de ressources), et ainsi de suite.

4.2 Contraintes liées à Android

Android étant un système d'exploitation fonctionnant différemment d'un ordinateur de bureau, il a fallu composer avec, et donc effectuer des changements plus ou moins grands. En particulier, voici plusieurs exemples de ce qu'il faut prendre en compte lorsque l'on traîte avec des systèmes mobiles Android :

— la consommation de batterie, qui est liée à la consommation de données,

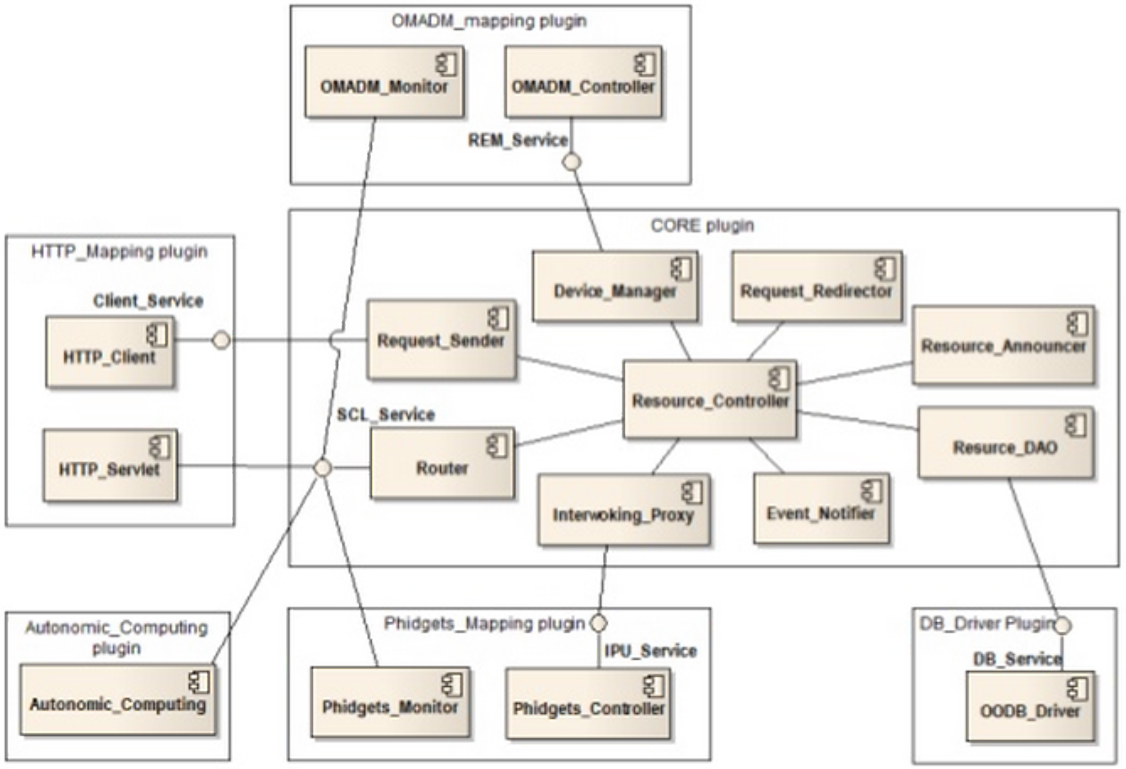


FIGURE 8: Diagramme de composants – architecture existante de la plate-forme M2M

— c'est une machine à états avec IHM, certaines opérations ne peuvent pas s'effectuer sur un thread UI (opération longues, comme un accès réseau par exemple),
— certaines bibliothèques Java ne sont disponibles sur Android que sous forme de stubs vides,
— la mémoire est limitée (à l'installation – sur le google play – et à l'exécution – par machine virtuelle)
— ...

4.2.1 Étude de consommation

La consommation de données se fait suivant la machine à états en figure 9.

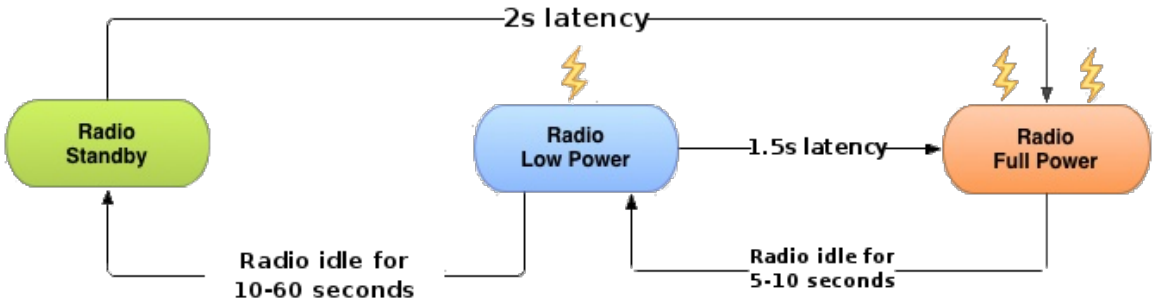


FIGURE 9: Radio cell state diagram

Or, le tableau 1 reflète la consommation de batterie suivant l'état et le type de connexion.

	Establish	Maintain	Transmit	Tail
Wifi	High	Low	Low	Low
3G	Low	Low	High	High

TABLE 1: Consommation de batterie

Nous pouvons alors comparer deux applications ayant la même quantité de données à envoyer (cf. tableau 2). Nous nous rendons compte que la politique d'envoi de données la plus intéressant est de rassembler au maximum toutes les données à envoyer dans un laps de temps court, et d'envoyer les données d'un coup.

	Nb of data operations	Duration between operations	Size of each operation	Time in high/-low state
App 1	3	18 s	1mb/s	18 + 42 s
App 2	3	0 (Bundled)	1mb/s	8 + 12 s

TABLE 2: Comparaison de 2 applications – même quantité de données à envoyer

En résumé, pour économiser de la batterie, il faut :

- optimiser les transferts de données en
  1. minimisant la fréquence des transferts
  2. minimisant la taille du payload des données (bandwidth)
  3. optimisant le timing des transferts
- utiliser les moyens suivants :
  1. regrouper les envois/récupérations de données
  2. éliminer les transferts ou polling périodiques non-critiques
  3. évaluer la criticité des données

4.3 Changements liés aux bibliothèques

Par absence de compatibilité, j'ai été obligée de changer certaines de bibliothèques utilisées par le projet.

À chaque fois, le projet étant open-source et sous licence eclipse, j'ai dû trouver des substituts ayant des licenses compatibles.

4.3.1 Traitements de XML

La plate-forme *M2M* utilise pour moyen de communication des ressources formattées en `XML`. Pour échanger des données, il faut donc avoir un moyen de transformer du `XML` en objet `Java`, et vice-versa. Ce moyen, dans la plate-forme existante, c'est `JAXB`. Il s'agit d'une librairie

qui annote des fichiers Java afin que le *serializer* ou *dé-serializer* sache comment transformer les objets en `XML` et réciproquement.

Ici, `JAXB` a été utilisé pour deux opérations distinctes :

- pour transformer des fichiers `XSD` (modèles à la norme *ETSI* pour les `XML`) en fichiers `Java` annotés (cf. figure 10),
- pour sérialiser/désérialiser des objets en `XML` et vice-versa (cf. figure 11)

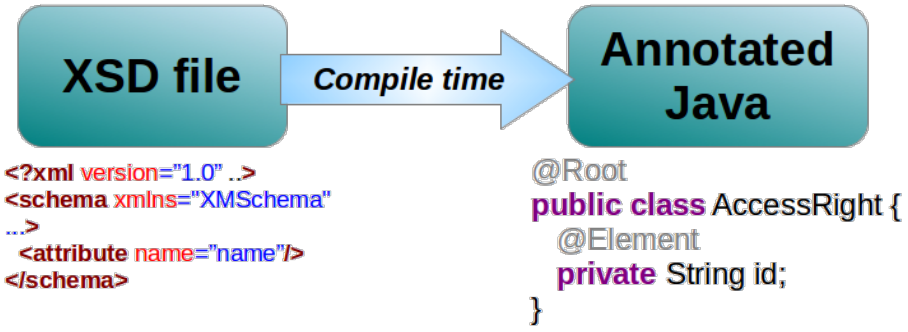


FIGURE 10: Transformation `XSD` ⟶ `Java` annoté avant la compilation

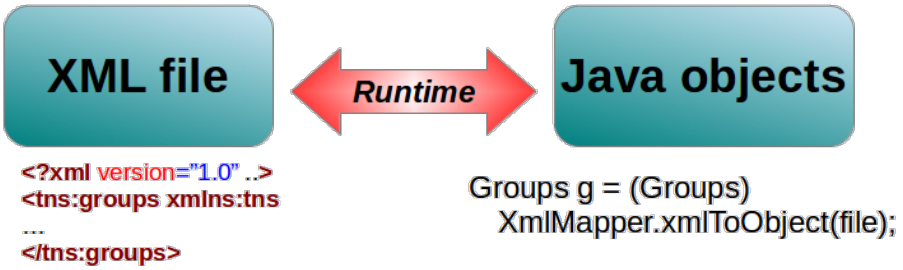


FIGURE 11: Transformation à l'exécution Java ⟷ XML

Cependant, `JAXB` n'a pas pû être réutilisé sur Android pour les raisons suivantes :

- JAXB utilise des classes Java qu'Android ne fournit pas
- les classes Java non fournies par Android peuvent être importées, mais Android ne permet pas d'importer des classes dont le package commence par `java.*` ou `javax.*`, donc un refactor est nécessaire
- Android ne supporte pas les annotations de package
- les bibliothèques manquantes sont trop lourdes à importer (31 M)

J'ai donc trouvé une alternative avec `Simple XML`, solution de plus compatible avec `Android`. Il a donc fallu que j'effectue les modifications suivantes :

- Transformation des ressources existantes (> 90 fichiers de ressources tous inter-dépendants)
- Transformer et ajouter des annotations
- Être en accord avec la bibliothèque

4.3.2 Validation de XML

La plate-forme utilise aussi les fichiers `XSD` afin de valider si le `XML` reçu est bien conforme à celui défini par la norme *ETSI*. Pour cela, elle utilise la validation `XML` avec le package de validation `javax.xml.*`. Cependant, ce package n'existe pas sous Android dans les versions inférieures à 20 (version qui a été mise en circulation ultérieurement à l'apparition du problème).

Nous avons donc importé la librairie `Xerces` pour Android, qui comble les manques d'Android pour le côté validation.

Plus tard, nous pourrons nous demander s'il n'y a pas une manière plus efficace de traiter le problème, en question de performance, et de manière native.

Deux choix pourrons se poser alors :

- utiiser une bibliothèque en `C/C++` pour une optimisation utilisant le Native Development Kit (NDK),
- utiliser l'Application Programming Interface (API) 20 (utilisant nativement la couche `JNI`)

4.3.3 Client HTTP

J'ai dû changer le client `HTTP` qui était utilisé, et le remplacer par un client compatible avec Android : Android `HTTP` client.

4.3.4 Serveur HTTP

La plate-forme embarque un serveur (pour répondre aux requêtes extérieures). Je l'ai remplacée par l'implémentation `Jetty`, qui peut être embarquée sur Android.

4.3.5 Base de données

La plateforme utilisait `DB4O` (database for objects), et le portage sur Android a montré que cette bibliothèque n'était pas appropriée pour Android.

- Problèmes récurrents et connus avec `DB4O` sur Android : accès concurrents mal gérés,
- `DB4O` convient pour des accès simples, mais n'est pas adapté aux bases de données complexes sur Android

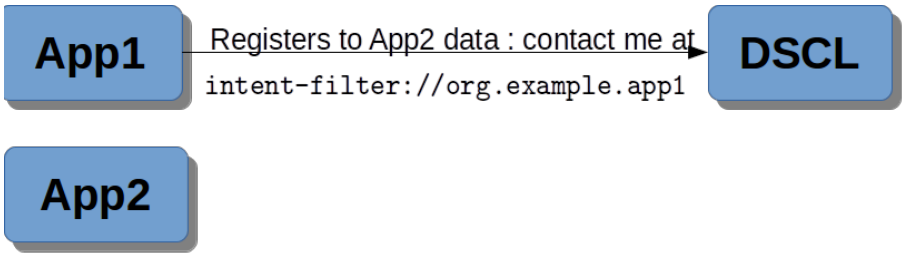
J'ai donc porté mon choix sur `ORMLite`, une bibliothèque plus adaptée.

- Bibliothèque développée en se préoccupant des contraintes liées à Android,
- utilise la base de données native Android SQLite,
- utilise des annotations

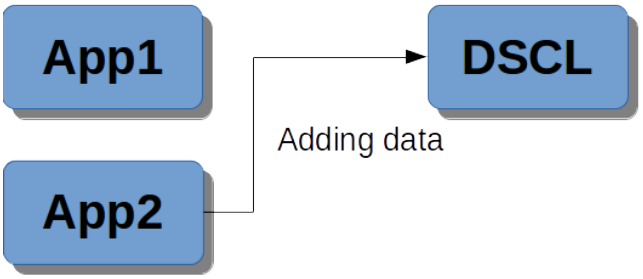
En tant que base de données objet, `DB4O` effectue par lui-même les stockages et récupération des objets en base de données.

Pour une base de données, il existe plusieurs manières différentes de stocker les objets interdépendants :

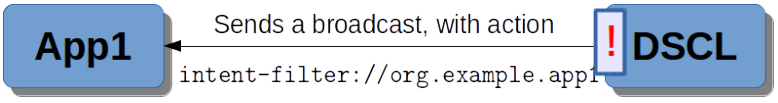
1. soit au moment du stockage, utiliser la notion de clé étrangère, et les objets sont intrinsèquement liés,



(a) Souscription aux données de l'application 2



(b) App2 ajoute des données



(c) Notification pour App1

FIGURE 12: Souscription/notification avec la plate-forme sous Android

2. soit donner un lien explicite entre les objets.

Ici, la solution 2 a été adoptée, en utilisant une `URI` dont le début serait commun. Par exemple : `dscl/applications/LAMP_0/containers/DATA/contentInstances/CI_13051152464` représenterait une donnée dont le container aurait l'URI `dscl/applications/LAMP_0/containers/DATA`.

4.4 Changements structurels

La machine dalvik et Android fonctionnant d'une certaine manière (machine à état, *Java Virtual Machine (JVM)* séparées, etc.), il a été impossible de garder certaines des structurations présentes dans le projet.

4.4.1 Principe de notifications

La plate-forme existante a mis en place un système de notifications (ou d'abonnement) afin que certaines applications puissent recevoir des notifications à chaque fois qu'une nouvelle donnée qui l'intéresse arrive sur un *arbre de ressources* (voir figure 12).

Afin d'implémenter ce principe de notification, la plate-forme de base utilise des `URI` où tournent des programmes qui "écoutent" ces notifications (ports ouverts sur écoute).

Un problème s'est posé pour le portage sous Android : les différentes applications Android ne communiquent pas entre elles (par nature des `JVM`s complètement séparées). De plus, implémenter un principe de serveur restant à l'écoute sur le téléphone serait vraiment trop coûteux

en énergie et donc en batterie.

J'ai donc décidé d'implémenter cette fonctionnalité d'une manière différente : en utilisant une fonctionnalité native sur Android, le `BroadcastReceiver`.

Il s'agit de communiquer une information – qui s'applique à un moment donné – à toutes les applications qui cherchent à obtenir cette information. Autrement dit, il s'agit du principe de publish-subscribe, implémenté de manière native sur Android. Il suffit de souscrire à une certaine action. Par exemple, quand le téléphone finit de booter, le système Android envoie un broadcast donc l'action est `android.intent.action.BOOT_COMPLETED` ; les applications qui y ont souscrites seront donc notifiées quand le téléphone a fini de booter.

> **Note :**
> La méthode décrite ici s'applique pour les notifications intra-Android. La méthode précédente sera toujours appliquée pour les applications externes souscrivant au DSCL Android

4.4.2 Services

Android fonctionne comme une machine à état, et les opérations longues ne peuvent pas s'effectuer dans le thread UI. Il existe une classe dédiées aux opérations *long-running* : un `Service` Android.

Sur la plate-forme existante, le démarrage et l'arrêt se faisaient via `OSGi`. J'ai modifié en conséquence afin que sur Android, la plate-forme utilise le mode natif des `Service`.

De même que pour le problème précédemment vu des notifications, une autre forme de communication inter-application doit pouvoir s'effectuer, cette fois-ci dans l'autre sens : d'une application Android standard à l'application contenant la plate-forme *M2M*. Nous voudrions permettre aux applications "standard" d'utiliser certaines méthodes de la plate-forme. Pour cela, la communication se fait avec l'*Android Interface Definition Language (AIDL)*. Pour plus d'explications sur l'*AIDL*, voir le glossaire.

La figure 13 illustre ce fonctionnement.

4.5 Modélisation objet du projet Etsi sur mobile

4.5.1 Diagramme de composant

De manière globale, la structure n'a pas changé, et se trouve donc en figure 8.

4.5.2 Diagramme de classes

Le diagramme de classes de la partie métier est représentée à la figure 14 (ou pour une version plus grande, cf. annexe figure 25).

Les diagrammes de classes du modèle d'implémentation sont représentés dans les figures 15, 16, et 17.

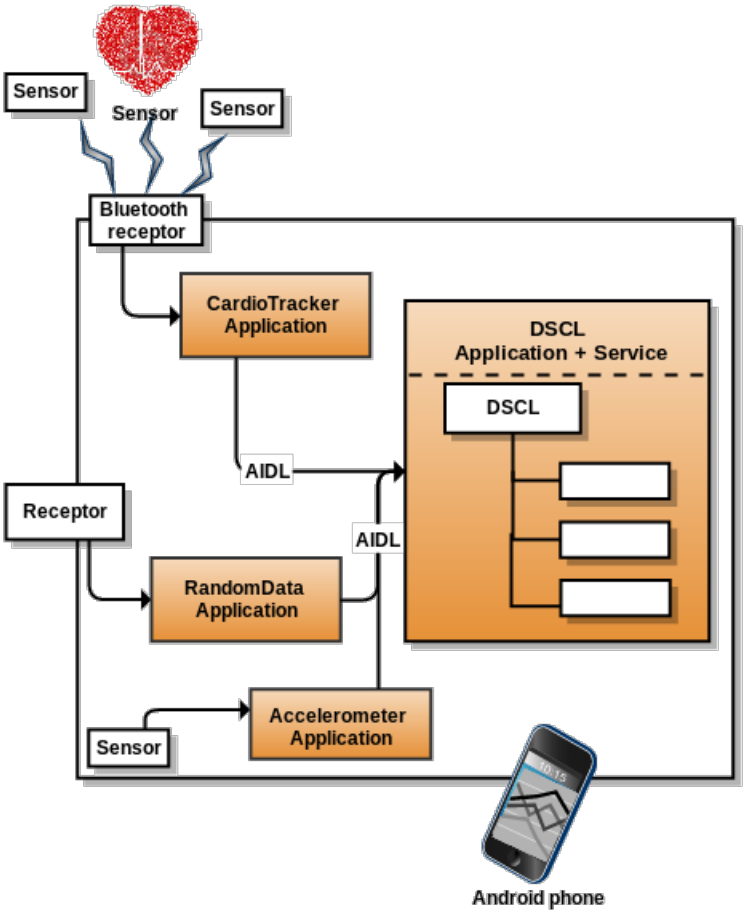


FIGURE 13: Architecture Android avec AIDL

5 Applications Android de démonstration

5.1 Application tutoriel

Afin d'aider les développeurs à démarrer avec la plate-forme sur Android, j'ai mis en place une application Android à valeur de tutoriel. Elle montre comment effectuer les opérations les plus courantes avec la plateforme. Dans ce sens, j'ai en plus écrit une page de documentation.

5.2 Application de capture – `CardioTracker`

L'application de `CardioTracker` a été faite dans le but de démontrer la faisabilité de l'intégration de la plate-forme *M2M* dans un système Android, et son interaction avec des capteurs (ici, un capteur de rythme cardiaque communicant par bluetooth).

La figure 18 montre un screenshot de l'application de capture de fréquence cardiaque.

5.2.1 Use case général

La figure 19 représente le diagramme de cas d'utilisation global de l'application CardioTracker.

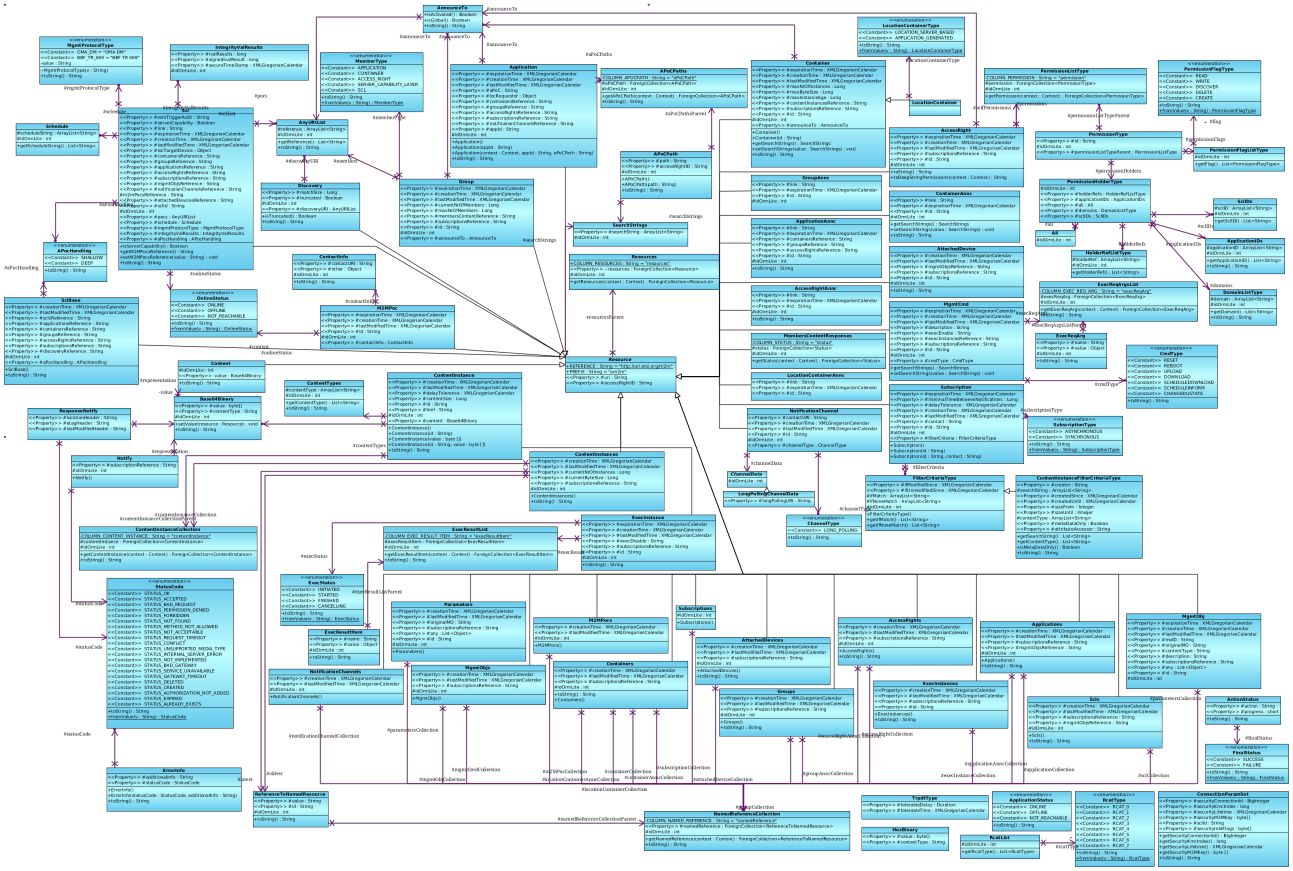


FIGURE 14: Diagramme de classes des classes métier

5.2.2 Use case – Cas détaillé

Le cas d'utilisation 3 représente le fil d'événements à suivre pour enregistrer un "effort".

Le cas d'utilisation 4 réfère à la création ou la modification des données personnelles du patient.

5.2.3 Diagramme de classes

La figure 20 montre les classes ayant un rapport avec le traitement du bluetooth.

La figure 21 montre la classe qui permet de représenter graphiquement (en utilisant Java 2D) le niveau de batterie du capteur ou du téléphone.

La figure 22 montre les classes utilitaires.

5.2.4 Technologies utilisées

Lors du développement de cette application, j'ai eu entre autres l'occasion de manier la communication en bluetooth, l'API Google Maps v2 pour Android et la géolocalisation, ainsi que ma propre plate-forme.

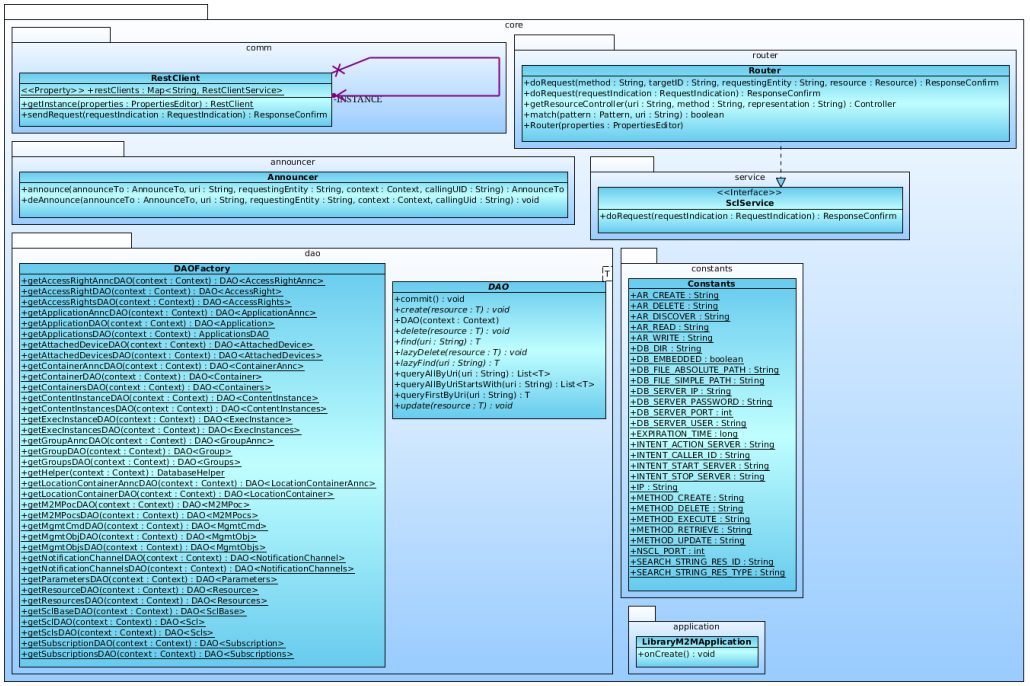


FIGURE 15: Diagramme de classe simplifié (absence des classes spécialisées) du projet bibliothèque M2M pour Android – core package

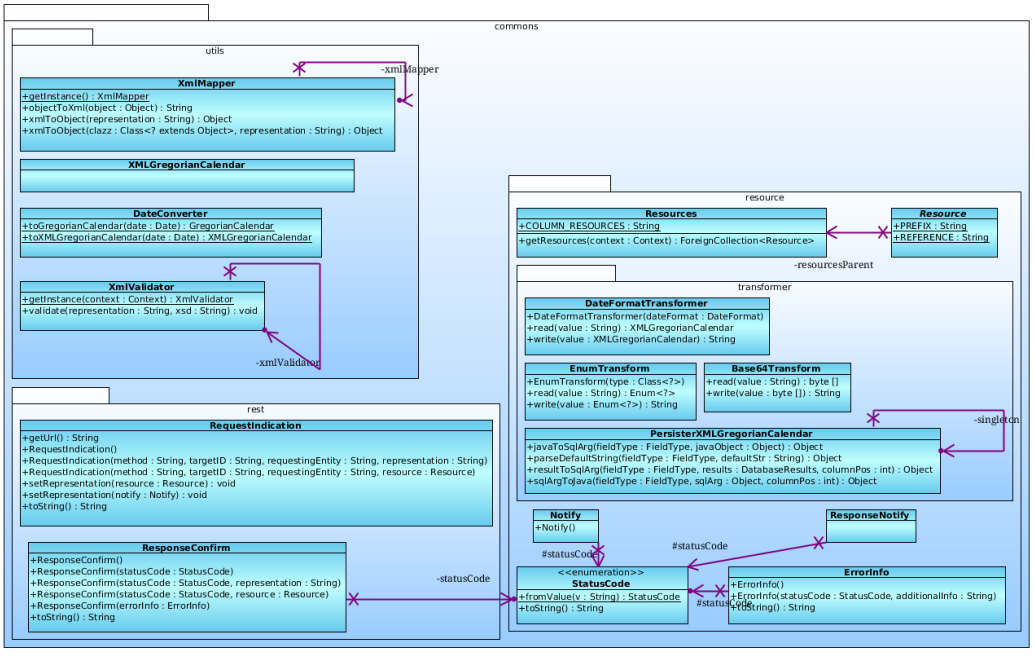


FIGURE 16: Diagramme de classe simplifié (absence des classes spécialisées) du projet bibliothèque M2M pour Android – commons package

6 Réflexions sur des aspects à améliorer

Il reste des points sur lesquels nous pourrions améliorer la plate-forme, ou bien son implémentation.

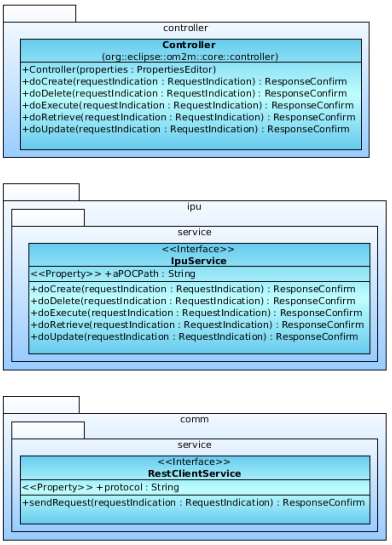


FIGURE 17: Diagramme de classe simplifié (absence des classes spécialisées) du projet bibliothèque M2M pour Android – controller et autres

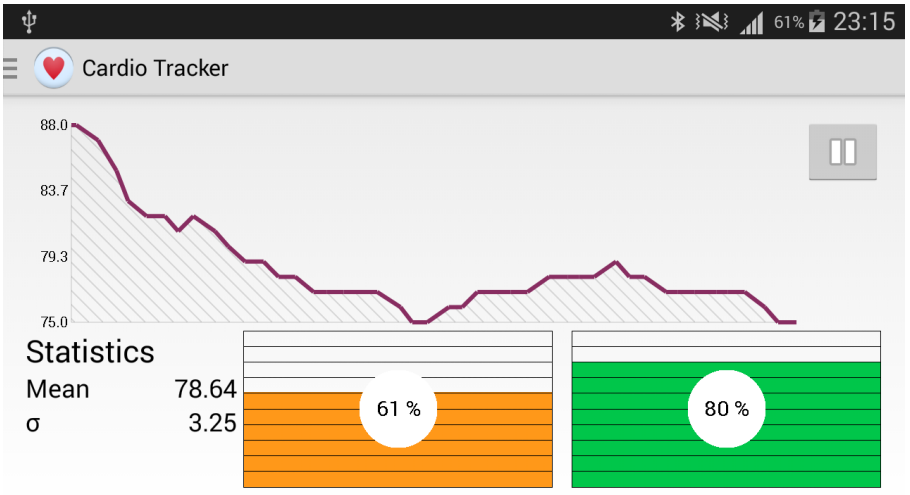


FIGURE 18: Screenshot de l'application CardioTracker

6.1 Plateforme OM2M

Pour être complète, on pourrait par exemple lui ajouter des aspects temps réel, mais cela demanderait que la plate-forme soit temps réel de bout en bout (un élément temps réel contraignant alors tout le système).

On pourrait y ajouter plusieurs autres aspects, comme :

— la sécurité des données (cryptées, ...)
— le traitement des flux continus (pour l'instant, traitement discret)
— la gestion des erreurs à un plus grand niveau (déjà implémentée à un certain stade)

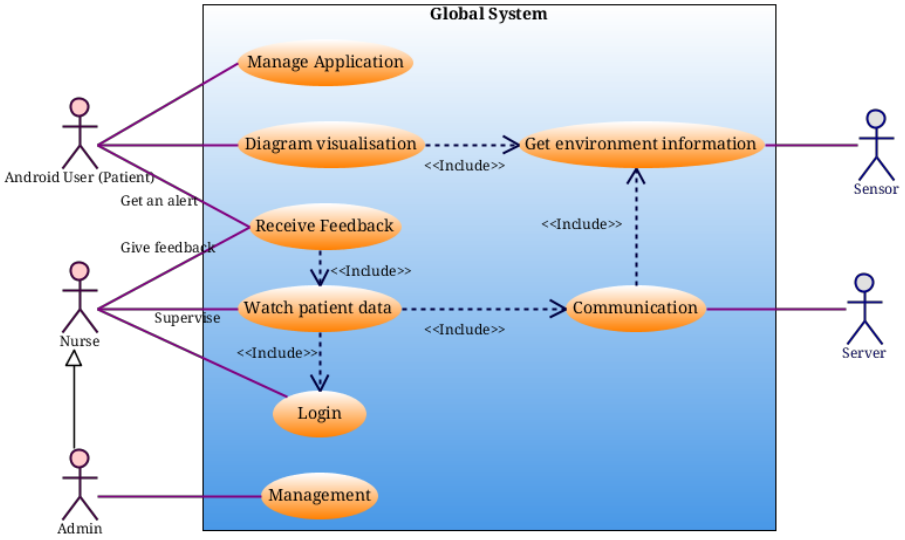


FIGURE 19: Use case général

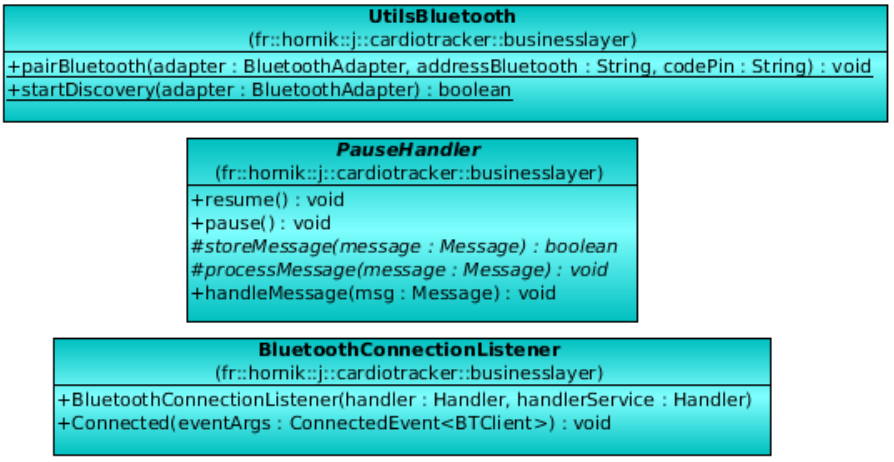


FIGURE 20: Diagramme de classes – bluetooth

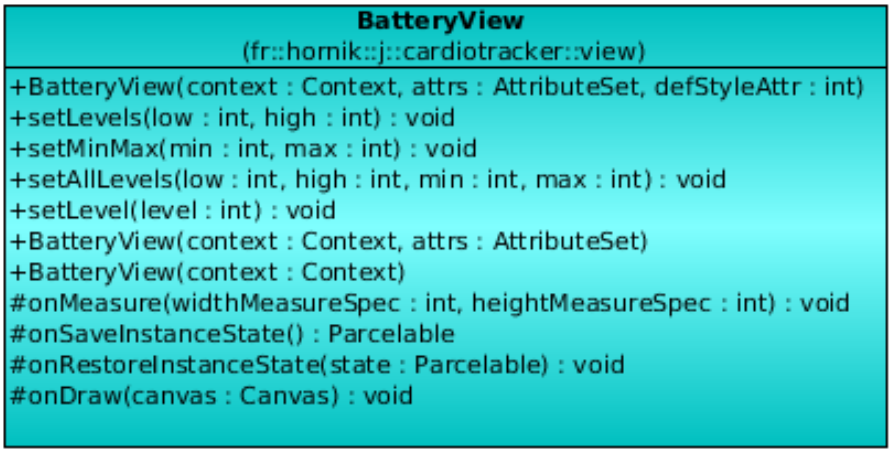


FIGURE 21: Diagramme de classe – BatteryView

Use case name	Record a track
Goal	Record a track, with heart rate, distance, speed, route
Actor	Patient
Basic flow of events	
Step 1	Click on *Start new track* button
Step 2	Select an activity (walking, running, everyday life)
Step 3	Click on *Start*
Step 4	The system tries to connect to the bluetooth device, and lets the user know about a possible failure
Step 5	The system keeps track of the duration, calculates the distance, the speed, and the heart rate
Step 6	The user stops the record
Step 7	The user can add a description
Alternative flows	
Case 1	If the GPS is not enabled, the distance and speed is not calculated, and stays equal to 0.
Case 2	If in step 5, the application brutally shutdown, then 1. The record is closed 2. The record is saved
Conditions	
Preconditions	The application is opened and the user has a HxM Bluetooth connected device, as well as a bluetooth compliant phone or tablet. He also has GPS enabled.
Post-condition	The record is saved and dated
Special requirements	The record shall be handled on a background thread
Extension points	1. Record a workout 2. Record everyday life

TABLE 3: Cas d'utilisation workout

6.2 Implémentation

La transformation des sources Java annotées façon `JAXB` en Java annoté façon `SimpleXML` + `ORMLite` a dû se faire "à la main", car le but premier du stage n'était pas de se pencher sur les modèles et leurs transformations, mais de terminer le projet afin de prouver la faisabilité.

Afin de gagner du temps à l'avenir, suite aux futures modifications possibles des `XSD`, il serait de bon ton de modéliser les fichiers `XSD` pour utiliser l'ingénierie des modèles afin de les transformer en fichiers `Java` annotés directement.

De la même manière, si la modélisation avait déjà été faite, cela m'aurait épargné des semaines de travail.

Use case name	Create or modify personal data
Goal	Input the patient personal data : age, weight, height
Actor	Patient
Basic flow of events	
Step 1	The user fills every field
Step 2	The user validates the form
Step 3	The system validates the fields and save them in the SharedPreferences
Alternative flows	
Case 1	If the patient does not complete one of the fields, then 1. The system displays a warning message and goes back to step 1
Conditions	
Preconditions	The application is opened.
Post-condition	The personal data are stored
Special requirements	The age shall be comprised between 10 to 120, the weight between 10 to 200, the height between 0.40 to 2.50 m.
Extension points	

TABLE 4: Cas d'utilisation créer/modifier les données personnelles

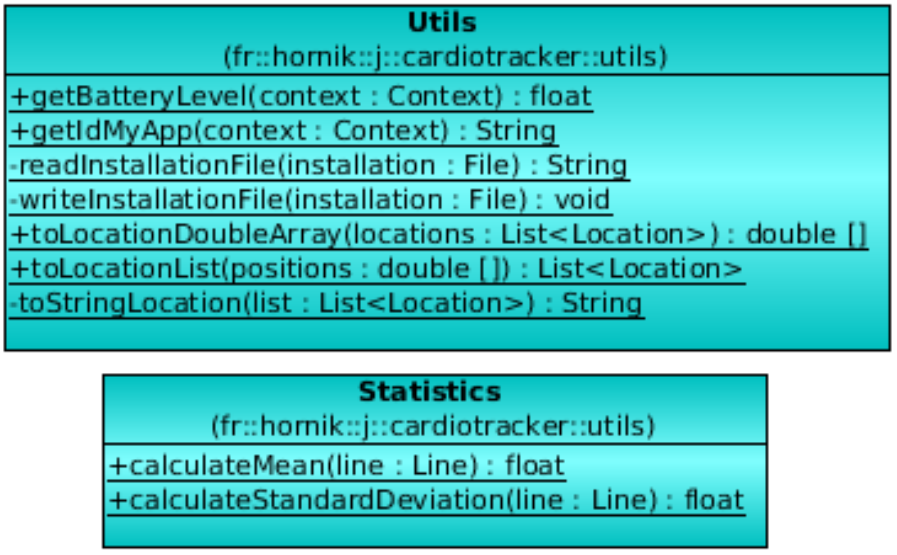


FIGURE 22: Diagramme de classes – utilitaires

7 Éléments subsidiaires

En parallèle au sujet exact de mon stage, j'ai eu l'occasion d'effectuer plusieurs "tâches".

7.1 Journée de présentation M2M à l'Insa – Poster

Le 15 mai, des industriels et l'équipe d'*OM2M* ont présenté différents aspects du *M2M*. Durant cette journée, il m'a été demandé (en plus d'assister aux conférences) de réaliser un poster sur le sujet de mon stage, c'est-à-dire *M2M* sur système mobile Android. Le poster que j'ai présenté se trouve en annexe, figure 24.

7.2 Formation Android – Présentation des widgets

Début Avril, Frédéric Camps a donné une formation Android de 4 jours à laquelle j'ai assisté. Il m'a été demandé de présenter aux participants le fonctionnement des widgets à placer sur l'écran d'accueil du téléphone.

J'ai donc expliqué ce fonctionnement via une présentation, et fourni le code source d'une application tutoriel spécialement développée pour illustrer mes propos (cf. annexe sur les widgets).

8 Bilan du stage

Tout ce qui se trouve sur la figure 23 représente de manière schématique les résultats que j'ai obtenus :

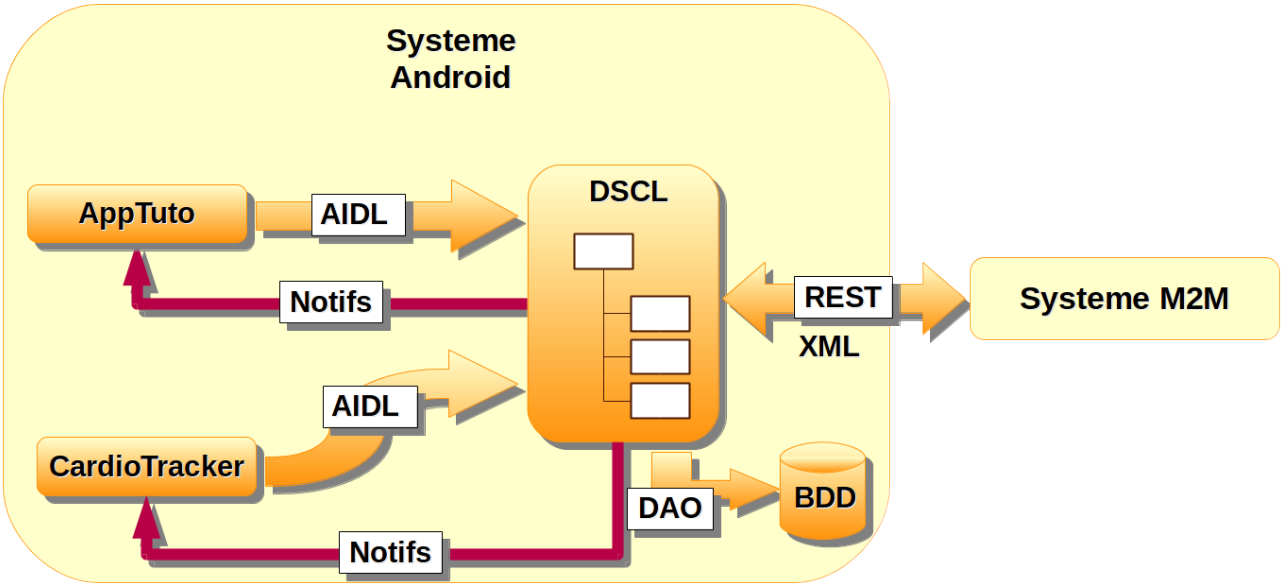


Figure 23: Résumé des résultats obtenus

- une application `DSCL` contenant la plate-forme, les moyens de communication HTTP (et XML), avec la base de données locale, avec les autres applications du téléphone, ...
- une application tutoriel montrant les capacités de la plate-forme
- une application CardioTracker démontrant l'interaction avec un capteur de fréquence cardiaque

Les objectifs ont été atteints, car le but premier était de prouver la faisabilité de l'incorporation d'un système mobile dans la plate-forme. Ici, la réussite a dépassé les objectifs dans le sens où j'ai même incorporé la plate-forme dans les systèmes mobiles.

Conclusion

J'aurais pu développer une solution plus facile en utilisant l'équipement Android en tant que Gateway. Cependant, cette solution n'aurait rien apporté de nouveau à la plateforme OM2M. C'est pour cela que mon sujet a pu évoluer dans le sens d'une plus grande utilité au projet. Nous avons donc décidé que je porterais la plateforme pour être déployable sur Android. Après ce portage, il faudrait alors faire en sorte que des applications Android puissent utiliser correctement la plateforme, d'où le développement de l'application de démonstration exemple et l'application `CardioTracker`.

Acronymes

AIDL Android Interface Definition Language. 20, 30, *Glossaire :* AIDL

API Application Programming Interface. 18, 22, 30, *Glossaire :* API

DSCL Device Service Capability Layer. 10, 11, 30, *Glossaire :* DSCL

ETSI European Telecommunications Standards Institute. 5, 7, 17, 18, 30, 31, *Glossaire :* ETSI

GA Gateway Application. 6, 30

GSCL Gateway Service Capability Layer. 6, 11, 30

IDEA Informatique : Développement, Exploitation et Assistance. 4, 30

JVM Java Virtual Machine. 19, 30

LAAS Laboratoire d'Analyse et d'Architecture des Systèmes. 4, 30

M2M Machine-to-Machine. 5, 7, 11, 14, 16, 20, 21, 28, 30, *Glossaire :* M2M

NA Network Application. 6, 30

NDK Native Development Kit. 18, 30

NSCL Network Service Capability Layer. 6, 7, 11, 30

OM2M One M2M. 7, 28, 30, *Glossaire :* OM2M

SARA Services et Architectures pour les Réseaux Avancés. 4, 30

SCL Service Capability Layer. 5, 7, 30, *Glossaire :* SCL

Glossaire

AIDL Android Interface Definition Language (AIDL) est similaires aux autres IDL. Il nous permet de définir une interface que le client et le service acceptent afin de communiquer entre eux en utilisant l'IPC (interprocess communication). Sur Android, un process ne peut normalement pas accéder à la mémoire d'un autre process. L'AIDL est là pour pouvoir faire communiquer ces deux process (qui ont donc besoin de décomposer leurs objets en primitives que l'OS comprend). – traduit de la documentation officielle. 20, 30

API Une Application Programming Interface (API) est un ensemble particulier de règles et de spécifications qu'un logiciel peut suivre pour accéder et utiliser des services et ressources fournies par un autre logiciel en particulier qui implémente cette API. 18, 30

ARBRE DE RESSOURCES C'est un arbre hiérarchisé (resource tree) où les données des capteurs (par exemple) sont stockées. Dans le cadre de OM2M, cet arbre est matérialisé par une base de données, persistante ou non. 5–7, 19, 30

DSCL C'est le framework qui va permettre à la plateforme de fonctionner dans un device. Dans notre contexte, le device correspond à un téléphone ou une tablette Android. 10, 30

ETSI C'est un organisme qui vise à établir des normes qui soient globalement applicables dans le domaine de l'information et des communications (ICT). Voir le site de l'ETSI. 5, 30

M2M M2M définit un mode de communication qui vise à être autonomique, c'est-à-dire qui peut fonctionner, se configurer, etc. sans intervention humaine. Ce document illustre l'architecture M2M avec la norme *ETSI*. 5, 30

OM2M Le projet OM2M est un projet eclipse open source (M2M). Voir le site. 7, 30

SCL C'est le framework qui gère le stockage des ressources et la communication (entre lui-même et les applications, et entre plusieurs SCLs). 5, 30

Appendices

A Documents du stage

La figure 24 montre le poster que j'ai présenté lors de la journée de formation M2M à l'Insa le 15 mai 2014.

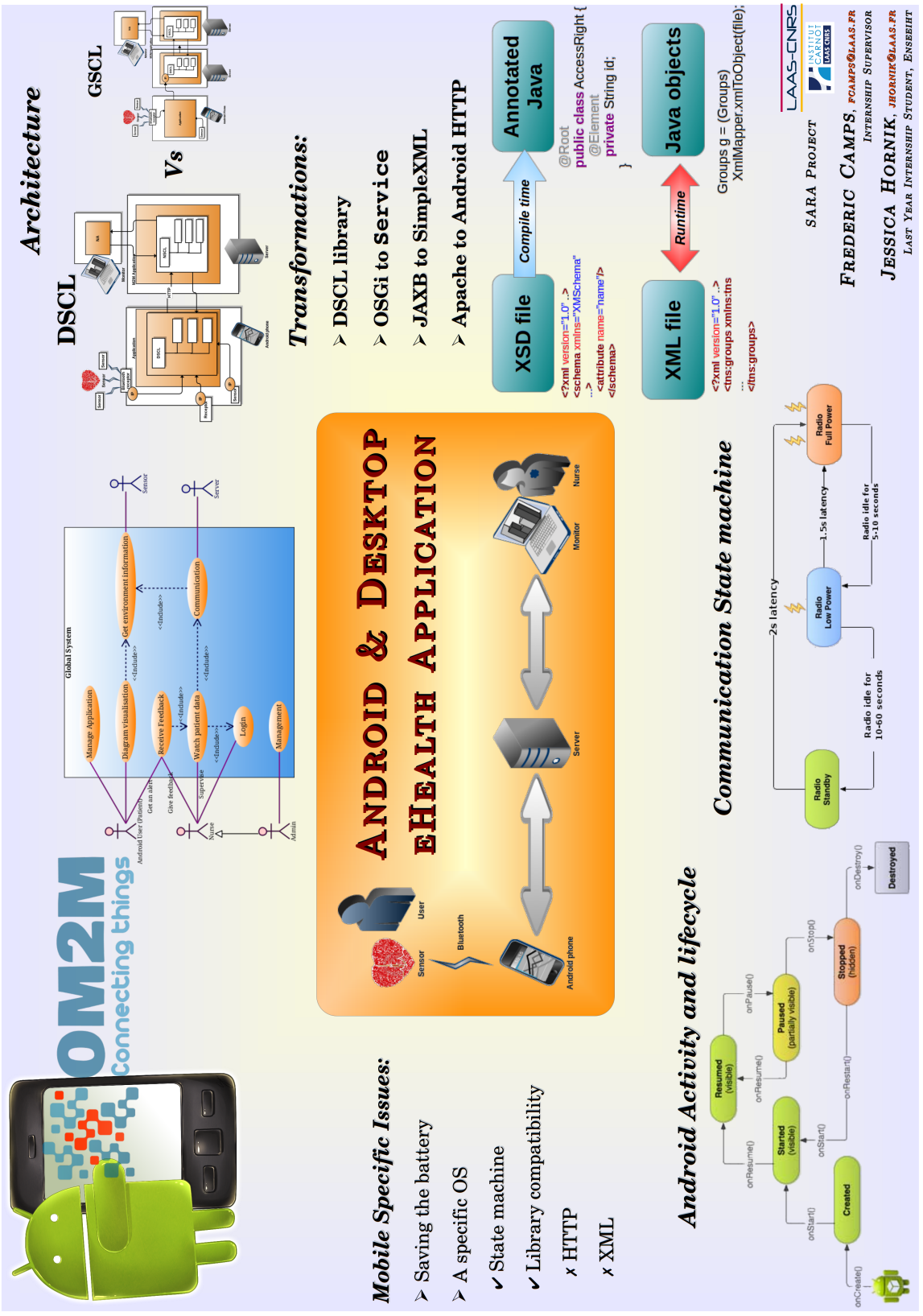


FIGURE 24: Poster présenté lors de la journée de formation M2M à l'INSA le 15 mai

B Documentation pour les Intents

From your M2M application, you can let your user customize your settings. Here is how you should proceed.

Quick Setup

Add this code to your *Activity*

```
Intent intent = new Intent("org.eclipse.om2m.site.dscl.PREFS");
intent.putExtra("idCallingApp", MY_ID);
startActivity(intent);
```

With somewhere safe in your app :

```
private static final String MY_ID = "idThatIWillRemember";
```

ExternalRedirectActivity

This is the activity that another Android application can launch, thanks to these parameters :

```
<intent-filter>
  <action android:name="org.eclipse.om2m.site.dscl.PREFS" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

You should add some extra to the intent with your application ID (as the *DSCL* will remember you that way afterwards, so please **make your application remember** it also). It then redirects to the internal redirect activity in order to know if the call comes from the DSCL application (internal) or from another application (external).

Extras to add

When calling this intent, you must add the *extra* String called "idCallingApp". This is the way your application will be recognized afterwards. If you do not set this attribute, your id will be set to "guest".

InternalRedirectActivity

Knows whether the call was made internally or not, and sets the Preferences to set accordingly. It then evaluates your phone's version and either make you go towards a *fragment* based activity or an ancient (API < 11) activity.

Preferences settings

Thanks to your application ID, lets your user customize the preferences of your application.

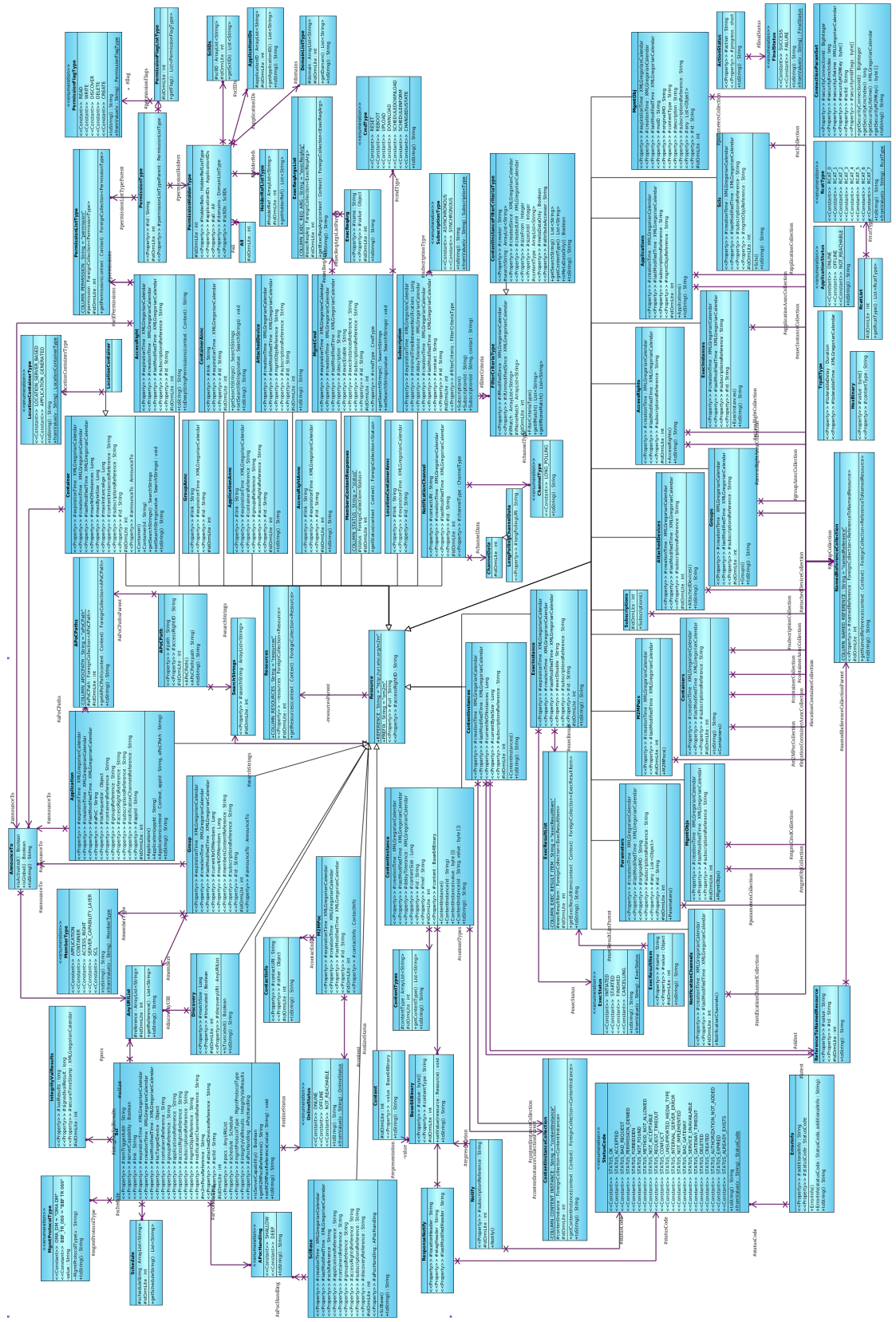


FIGURE 25: Diagramme de classes des classes métier

Caution !
You should have called this with your application ID (see ExternalRedirectActivity). Otherwise, you will be "invited" here as a guest.

PreferencesActivity (API $\geq$ 11)

This uses the fragment-based architecture in order to load/find the preferences.

PrefsLegacyActivity (API $<$ 11)

This is for devices with API pre-Honeycomb. It uses the deprecated methods for loading/-finding the preferences in the application (not in fragments).

C Présentation sur les widgets Android

Creating home widgets – on Android

Jessica HORNIK

9th April, 2014

Contents

Contents

Présentation d'un widget

Figure: Screenshot d'un widget

Mini-application

- Présentation d'un élément de l'application
- Moins d'informations que l'application en elle-même
- Peut permettre à l'utilisateur de lancer l'application

≥ Android 3.1

Adaptable en taille

Éléments graphiques

Layout
- FrameLayout
- LinearLayout
- RelativeLayout

Gestes limités
- onClickListener
- Geste haut/bas

View
- AnalogClock
- Button
- Chronometer
- ImageButton
- ImageView
- ProgressBar
- TextView

≥ Android 3.0
- GridView
- ListView
- StackView
- ViewFlipper
- AdapterViewFlipper

Contents

Étapes

- Créer un fichier layout
- Créer un fichier XML (`AppWidgetProviderInfo`) pour les propriétés du widget
- Créer un `BroadcastReceiver` qui fera le lien avec l'UI du Widget
- Enregistrer le receiver du widget dans `AndroidManifest.xml`
- Éventuellement ajouter une activité de configuration du Widget

Fichier layout

widget_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com
      /apk/res/android"
    android:id="@+id/rl_widget"
    android:background="@drawable/shape_widget" >
    <TextView
        android:id="@+id/tv_widget_station_number"
        android:layout_centerHorizontal="true" />
    <TextView
        android:id="@+id/tv_widget_available"
        android:layout_centerVertical="true" />
    <TextView
        android:id="@+id/tv_widget_total"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true" />
    <ImageView
        android:id="@+id/iv_widget_reload"
        android:src="@android:drawable/ic_menu_rotate"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true" />
    <TextView
        android:id="@+id/tv_widget_last_update"
        android:layout_alignParentBottom="true"
        android:layout_toLeftOf="@id/iv_widget_reload" />
</RelativeLayout>
```

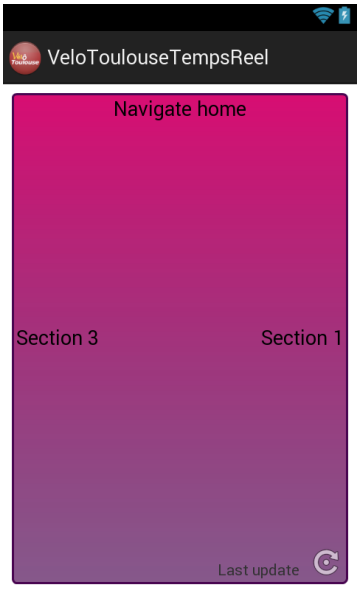


Figure: Widget layout

XML file

Fichier pour décrire le widget

widget_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialLayout="@layout/widget_layout"
    android:minHeight="72dp"
    android:minWidth="146dp"
    android:resizeMode="horizontal|vertical"
    android:updatePeriodMillis="180000"
    android:widgetCategory="keyguard|home_screen" >
</appwidget-provider>
```

Caractéristiques

- Update toutes les 30 min
- Resizable

Architecture Receiver – Service

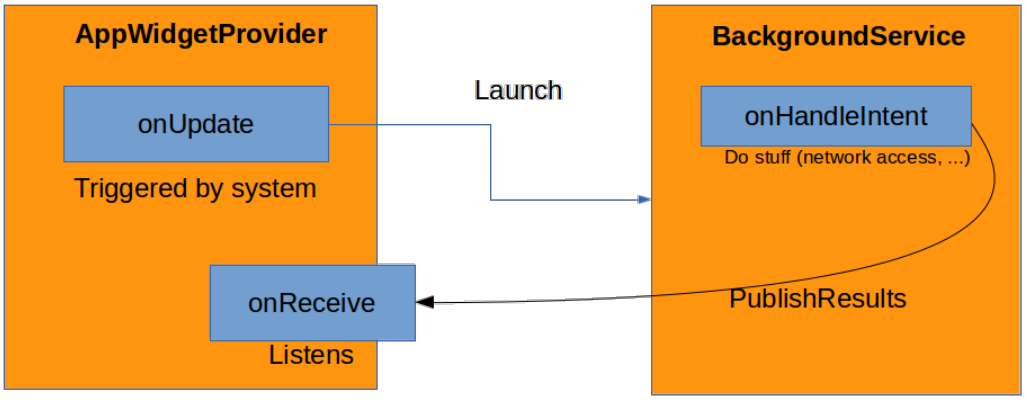


Figure: Architecture

Receiver

AppWidgetProvider

- Recevoir des mises à jour système
- Un `Receiver` et plus (Réception d'`Intents`)
- Opérations sous 5 secondes !

```
public class WidgetProvider extends
    AppWidgetProvider {
     public void onEnabled(Context context) {}
     public void onUpdate(Context context,
        AppWidgetManager appWidgetManager,int[]
        appWidgetIds) {}
     public void onDisabled(Context context) {}
     public void onReceive(Context context, Intent
        intent) {}
     public void onDeleted(Context context, int[]
        appWidgetIds) {}
}
```


Receiver

Accès aux vues depuis le `Receiver`

On peut avoir accès aux différents éléments UI grâce à un objet `RemoteViews`.
On prend la référence avec :

```
RemoteViews v = new RemoteViews(context.
    getPackageName(), R.layout.widget_layout);
```

Exemples :

- `v.setTextViewText(R.id.tv_name, "exemple");`
- `v.setTextViewColor(R.id.tv_name, Color.RED);`

`Service` pour opérations métier

```
public class BackgroundService extends IntentService {
    public static String NOTIFICATION = "notif";
    @Override
    protected void onHandleIntent(Intent intent) {
        HistoryEntry result = null;
        if (HttpUtils.isConnected(getApplicationContext())) {
           result = HttpUtils.sendHttpFillDBGetHistoryEntry("Toulouse",
                222, getApplicationContext());
        }
        publishResults(result, widgetId);
    }

    private void publishResults(HistoryEntry result, int widgetId) {
        Intent intent = new Intent(NOTIFICATION);
        intent.putExtra(Constants.INTENT_RESULT, result);
        sendBroadcast(intent);
    }
}
```

- Pour effectuer des opérations longues (network access, ...)
- Envoie des notifications pour mettre à jour la vue

Déclarer dans le AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.hornik.j.velotoulousetempsreel"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
    <uses-permission android:name="android.permission.WAKE_LOCK" />

    <application android:theme="@style/AppTheme" >
        <activity .../>

        <receiver android:name="fr.hornik.j.velotoulousetempsreel.receiver.WidgetProvider" >
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
                <action android:name="fr.hornik.j.velotoulousetempsreel.receiver" />
            </intent-filter>

            <meta-data
                android:name="android.appwidget.provider"
                android:resource="@xml/widget_info" />
        </receiver>
        <receiver android:name="fr.hornik.j.velotoulousetempsreel.alarmmanager.
              AlarmManagerBroadcastReceiver" >
        </receiver>
        <service
            android:name="fr.hornik.j.velotoulousetempsreel.receiver.BackgroundService"
            android:process=":my_process" >
        </service>
    </application>
</manifest>
```

Résultat

Figure: Résultat du widget resizable