Rapport de Projet Technologique

Robin Hardy
Paul José-Vedrenne

Introduction

L'objectif principal du projet Autopilote est de concevoir un bus logiciel qui communique avec des capteurs. Ces capteurs sont des GPS, des accéléromètres ou des gyroscopes. Le bus se charge de stocker les informations produites par un capteur, et de les envoyer à un client observateur.

Il s'agit donc de faire de la programmation réseau, avec le bus qui joue le rôle de serveur et les capteurs et les observateurs qui jouent le rôle de clients.

Ce projet est développé dans le langage Java pour sa portabilité et la simplicité de programmer en réseau avec ce langage.

A la fin du projet, le résultat est une archive sous forme de .jar contenant toute les fonctions permettant d'utiliser ce bus logiciel mis en œuvre, soit pour le déployer en tant que serveur soit pour utiliser des fonctions clientes afin d'utiliser les services disponible et implementés par le bus

Une documentation complète de l'API est disponible ici :
DocumentationAPI

Développements réalisés

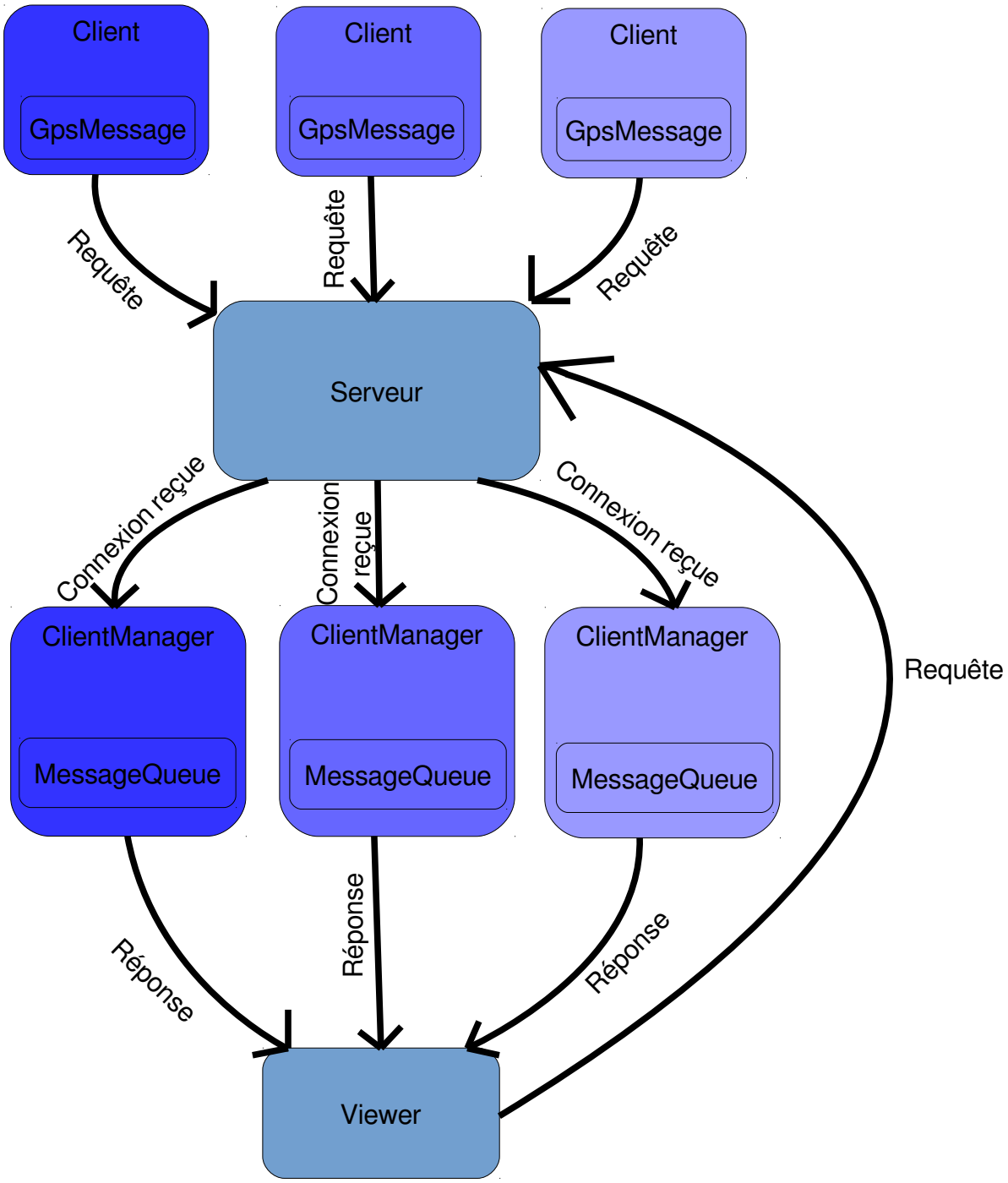
1. Architecture globale

Les classes Java implémentées dans ce projet sont les suivantes :

- Serveur.java
- ClientManager.java
- MessageQueue.java
- Client.java
- Viewer.java
- GPSMessage.java, qui étend BusMessageAbstract.java, qui implémente BusMessage.java

Le serveur correspond donc au bus logiciel et est constitué de 2 classes : **Serveur** et **ClientManager**. L'unique rôle de la classe **Serveur** est d'accepter des connexions, et pour chaque connexion reçue, de créer un **ClientManager** qui va traiter les requêtes spécifiques au client qui vient de se connecter. On a 2 types de clients : la classe **Client**, qui envoie les données du capteur, et la classe **Viewer**, qui est un observateur. Ce dernier lit les données envoyées sur le serveur (le bus). Chaque **ClientManager** possède sa propre **MessageQueue**, une file de **BusMessage** circulaire.
Enfin, le **Client** envoie des coordonnées du capteur en créant un **GPSMessage** qui construit l'objet Json correspondant.

Schéma de l'architecture globale du projet



Les requêtes envoyées par les clients sont les suivantes :

- register : connexion au serveur
- deregister : déconnexion
- list : obtenir la liste des capteurs connectés
- send : envoyer un message sous forme de Json
- get : obtenir un message à partir de son id
- get_last : récupérer le dernier message envoyé par un capteur

2. Intégration d'un capteur simulé

La classe SimGPS.java est un client qui simule un capteur GPS. Lorsqu'elle est instanciée, elle modifie ses attributs (les coordonnées) aléatoirement toutes les secondes. Pour cela, on importe la classe Random et à l'aide d'une graine, on génère des nombres aléatoires. Lors de l'envoi du message au serveur, les coordonnées sont celles du GPS simulé à un instant t.

3. Intégration d'un capteur Arduino

Au cours de l'année, nous avons fabriqué un capteur avec une boîte en carton, dont les parois intérieures sont recouvertes de papier aluminium, et dans laquelle on place une bille en plomb.

Chacune des parois est reliée à un bouclier Arduino par des fils. On peut donc récupérer l'orientation de la boîte grâce au bouclier Arduino. En effet, lorsque la boîte est inclinée, la bille touche 2 ou 3 parois en même temps, ce qui nous permet de connaître son orientation.

On peut l'intégrer au bus logiciel en important la bibliothèque RXTXComm.jar que nous avions utilisée pour communiquer avec une puce RFID.

Afin de réaliser un driver compatible avec notre bus il faut réaliser une interface USB pour récupérer les données générées par notre capteur Arduino. Deux parties sont donc à prévoir :

1. Coté Arduino : une partie émettrice permettant d'envoyer des données sous un certains format (par exemple en JSON) sur le port USB
2. Coté Driver : une partie réceptrice (codé en Java) permettant de lire les données transitant par le port USB et des les envoyer sur le bus par le biais de la classe Client permettant d'envoyer des requêtes haut-niveau

4. Intégration d'un capteur sur Smartphone

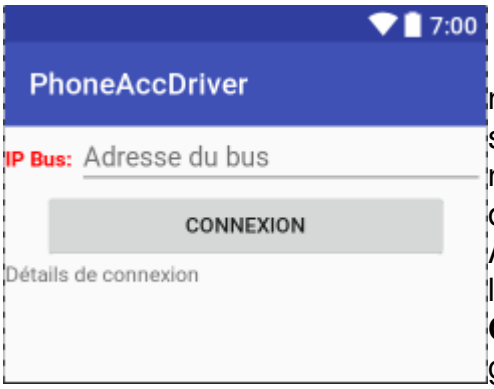
Avec l'utilisation du système d'exploitation Android, il est possible de récupérer les données de capteurs embarqués dans un smart phone. Cependant il faut respecter le principe d'application Android. Une application Android est composé d'une activité, permettant d'effectuer des traitements interne et invisible pour l'utilisateur comme récupérer les données d'un capteur, et d'une interface graphique, permettant de faire le lien avec l'utilisateur.

Les seuls données que l'utilisateur doit saisir dans l'interface graphique sont :

- L'adresse IP du bus
- Le nom du capteur

En utilisant la classe **Client** de notre API il est possible d'effectuer des requêtes haut niveau sur le bus et donc d'envoyer des messages contenant des données de capteurs.

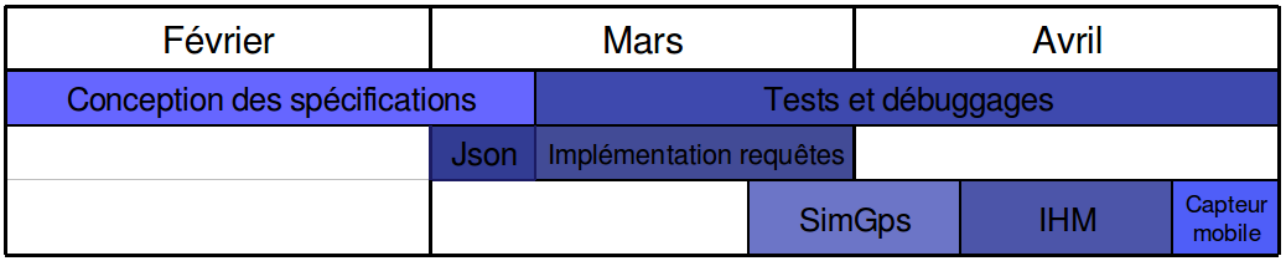
L'interface graphique ce présente comme telle :



Un appuie sur connexion permet de se créer une nouvelle instance de **Client** avec comme adress IP celle saisie dans le champ Adresse du bus. Ensuite une requête de type **register** avec un nom de driver codé en dur (saisie du nom non implémenté) et d'une classe (ici Accelerometer) est envoyé au bus. Dès que l'enregistrement est validé (dès qu'une réponse de type **OK** est reçu) l'activité récupère les info du capteur, génère un messages et l'envoie sur le bus à l'aide de la requête **send**, implémenté dans le classe **Client**

Organisation du travail

1. Diagramme de Gantt



2. Points durs rencontrés

La première difficulté rencontrée a été d'apprendre à programmer en réseau, au début de l'année. En effet, le réseau était totalement nouveau et a été enseigné au premier semestre de la licence 3. Cela a donc demandé un temps d'adaptation.

Une autre difficulté a été de concevoir les spécifications relatives aux différentes requêtes envoyées par le bus et les clients. Cela a demandé une réflexion globale du groupe et a pris plus de temps au final que leur implémentation.

Enfin, la dernière difficulté a été d'apprendre à utiliser le Json, un nouveau format de donnée textuelle. Cependant, cette difficulté s'est avérée rentable car elle a simplifié les choses par la suite. En effet, il est bien plus aisé de communiquer en réseau par du Json plutôt que par de simples chaînes de caractères.

Conclusion

En conclusion, conduire ce projet a nécessité d'apprendre beaucoup de nouvelles choses en programmation et en gestion de projet. Nous n'avons pas seulement développé, mais aussi réfléchi en groupe sur les spécifications.

Ce projet, constitué d'une partie conception et d'une partie développement, a mis en avant la partie conception, qui n'était pas vraiment présente dans les projets que nous avons pu faire jusqu'ici.