POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Département de génie informatique et de génie logiciel

INF3995

Projet de conception d'un système informatique

Rapport final de projet

Conception d'un système mobile de télémétrie en temps réel

Équipe No. 09

Anthony Abboud (#1681547)
Gabriel Demers (#1710718)
Khalil Bennani (#1566707)
Luc Courbariaux (#1670433)
Mounia Nordine (#1749881)
Reph Dauphin Mombrun (#1662298)

18 Avril 2018

Table des matières

1. Objectifs du projet

La société étudiante Oronos est une équipe multidisciplinaire affiliée à Polytechnique Montréal. Pour la préparation de son entrée à l'Intercollegiate Rocket Engineering Competition (IREC), la compétition la plus exigeante en fuséologie au niveau international, elle a besoin d'un système capable de récupérer et traiter les données que les capteurs de ses fusées collectent en vol. Ce système est basé sur une relation serveur/client : le serveur reçoit une transmission de la fusée et en relaie les informations aux clients.

Le système devra principalement :

- être en mesure de recevoir et traiter une grande quantité de données,
- les présenter de manière configurable selon la fusée,
- être compatible avec l'ancien système utilisé par Oronos,
- permettre l'exécution du client sur tablette android.

Nous verrons dans les pages qui suivent les moyens utilisés (dont l'architecture logicielle) pour répondre à ces exigences, en quoi ceux-ci diffèrent de ceux exposés lors de notre réponse à l'appel d'offre d'Oronos : un exposé de nos résultats après 3 mois de développement.

2. Description du système

2.1 Le serveur (Q4.5)

2.1.1. Configuration

En premier lieu, nous avons effectué la conception d'un serveur permettant l'acheminement des données brutes de la fusée vers les clients, ainsi que la récupération de données diverses à partir d'une interface REST. Malgré le fait que le développement ait été fait spécifiquement sur une plateforme Zedboard avec le système d'exploitation Linux (distribution Ubuntu Linaro), le code que nous avons produit est hautement portable. En effet, celui-ci est entièrement écrit en Python et utilise des librairies pouvant fonctionner tout aussi bien sur Windows que sur Linux. S'il est nécessaire de faire fonctionner le serveur sur la plateforme spécifique que nous avons utilisé pour le développement, alors il suffit de mentionner que nous avons premièrement suivi à la lettre les instructions détaillés dans les excellents documents "TP2 : Sortie HDMI et Linux sur Zedboard" et "TP3 : Serveur Web sur Zedboard" disponibles dans le cadre du cours INF3995. Nous pouvons tout-de-même mentionner que la version de Python utilisée est Python 3 et que nous avons utilisé l'utilitaire Python pip (peut être installé avec "sudo apt-get install python3-pip" ou le get-pip.py de https://pip.pypa.io/en/stable/installing/ s'il ne l'est pas déjà) pour installer les librairies suivantes : cherrypy, simplejson et pyserial. Un fichier "deps.sh" automatisant l'installation de ces trois dernières librairies est inclus dans le répertoire du serveur. De plus, notre projet dépend de tkinter, qui peut être obtenu (si nécessaire) via "sudo apt-get install python3-tk" ou alternativement les instructions données sur la page web http://www.tkdocs.com/tutorial/install.html.

2.1.2. Comparaison par rapport à l'architecture initiale

Une vue d'ensemble des différents modules composant notre architecture est illustrée à la figure suivante:

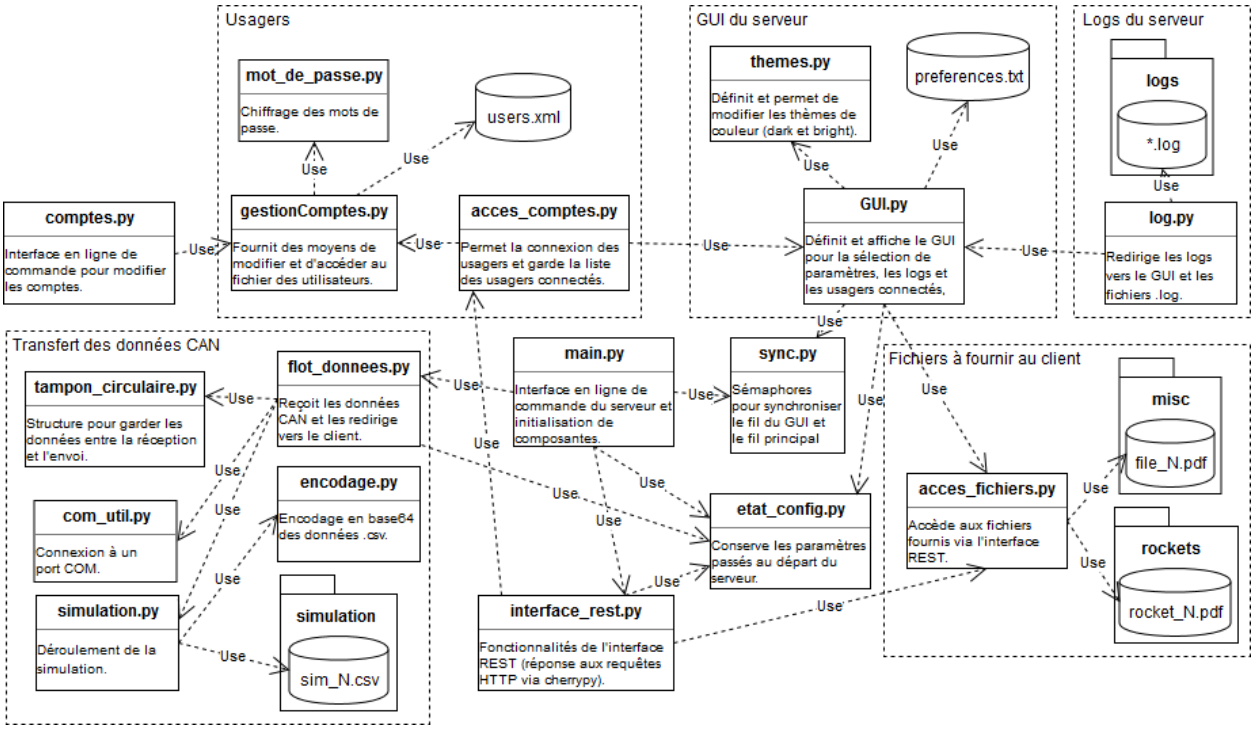

Figure 1. Vue d'ensemble de l'architecture du serveur.

Immédiatement, cette vue d'ensemble nous permet d'effectuer une comparaison avec le diagramme d'architecture fourni dans la réponse à l'appel d'offre. En premier lieu, on remarque que la partie concernant le transfert des données CAN s'est étendue sur plusieurs modules (il n'y avait que flot_donnees.py au départ). Cela est dû au fait que la complexité de cette fonctionnalité n'avait initialement pas été complètement appréhendée. Puis, au fur et à mesure que flot_donnees.py s'est complexifié, il a été jugé une bonne idée de prendre les sous-fonctionnalités nécessaires pour le flot des données, mais indépendantes de celui-ci (comme l'encodage des données de simulation, par exemple) et de les isoler dans des modules séparés. En outre, nous avons ajouté par convénience un répertoire "simulation" regroupant les fichiers .csv utilisés par la simulation.

Ensuite, on remarque aussi que la partie du GUI a pris de l'importance. Notamment, on voit que celui-ci est maintenant affecté par acces_comptes.py et log.py. Cela vient du fait que nous avons décidé de tirer avantage des composantes de tkinter pour non seulement permettre la sélection de paramètres, mais aussi afficher les usagers (donc plus besoin du module affichage_usagers.py) et créer la fenêtre d'affichage des logs, tel que demandé. Il est à noter que main.py n'agit plus directement sur GUI.py; le GUI part maintenant sur son propre fil d'exécution, d'où la nécessité du module sync.py qui encapsule des sémaphores pour s'assurer que le départ du serveur se fasse de manière ordonnée, sans condition de course sur les données de etat_config.py (qui peuvent provenir de la ligne de commande ou du GUI). Un autre détail ajouté est le fichier themes.py. Celui-ci définit les thèmes de couleurs demandés (sombre et clair) et permet leur application récursive sur les composantes de tkinter. Ajoutons aussi que la possibilité de sauvegarder les préférences des options du GUI a été considérée (génération d'un fichier preferences.txt).

D'autres changements plus mineurs peuvent aussi être mentionnés. Par exemple, nous avons jugé pertinent d'éviter la duplication des responsabilités en faisant en sorte que seul gestionComptes.py ait la tâche d'accéder au fichier users.xml et donc que acces_comptes.py doive maintentenant passer par gestionComptes.py pour avoir accès à la liste des usagers enregistrés et de leurs mots de passe. En outre, on voit qu'un répertoire "logs" a été ajouté pour faciliter la sauvegarde bien ordonnée des logs générés.

2.1.3. Fonctionnement interne des composantes individuelles

En plus des changements de grande échelle mentionnés ci-haut, il est clair que notre vision du fonctionnement interne des composantes individuelles s'est grandement éclaircie. Par exemple, considérons le fonctionnement du transfert des données CAN (dans flot_donnees.py et modules associés). Celui-ci peut-être résumé par la figure suivante :

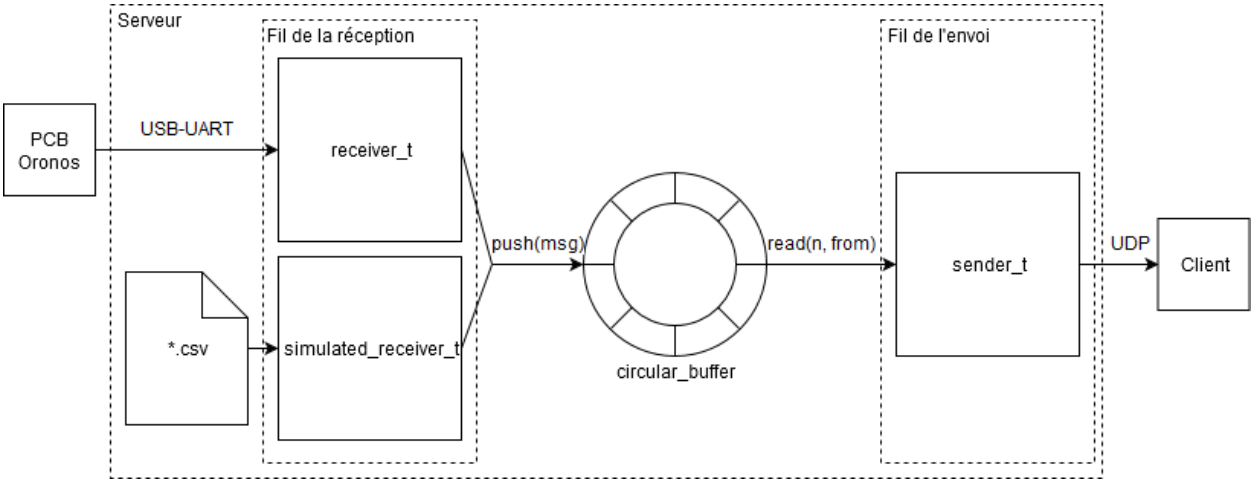

Figure 2. Chemin des données CAN.

Ici, on voit que nous avons parti deux fils d'exécutions : un pour la réception des données et un pour leur envoi au client. La réception des données peut soit être faite par un objet de type receiver_t ou par un objet de type simulated_receiver_t. Dans le cas de receiver_t, cela nous donne accès aux vraies données en provenance de la fusée via un port COM ouvert à l'aide de la librairie pyserial. L'entrée des données se fait une donnée à la fois et inclut un mécanisme pour rejeter les données incomplètes ou corrompues. En ce qui concerne simulated_receiver_t, celui-ci appelle le générateur simulation.play qui va puiser dans un fichier .csv et utilise les fonctions de encodage.py pour créer un flux de données binaires encodées en base64. Quelle que soit la classe du récepteur, les données obtenues sont ajoutées une-à-une à une file circulaire de type circular_buffer grâce à une méthode push(msg). Le circular_buffer est une structure de données *thread-safe* (opérations protégées par un mutex) qui garde un nombre de données égal à une puissance de 2. Lorsque la capacité de la file est atteinte, les plus anciennes données sont écrasées par les nouvelles, ce qui évite une croissance sans borne de la mémoire utilisée. En outre, la file associe implicitement un

"numéro de séquence" à chaque donnée, et donc son autre méthode read(n, from) permet au fil d'envoi de récupérer plusieurs données d'un seul coup.

Afin que l'envoi des données s'effectue correctement, un protocole simple chevauchant sur UDP a été développé. Premièrement, le client envoie un message avec, sur des lignes séparées, le numéro de séquence de la première donnée qu'il aimerait avoir et le nombre maximum de données qu'il aimerait recevoir. Ensuite, le serveur (via la classe sender_t) répond avec un message contenant des données. Plus spécifiquement, le message contient le numéro de la prochaine donnée à envoyer, le vrai nombre de données envoyées dans le message et les données elles-mêmes (une par ligne). Le client pourra ensuite garder le numéro de séquence reçu et l'utiliser dans son prochain message. Cela permet de n'envoyer que le nombre strictement nécessaire de données à chaque fois.

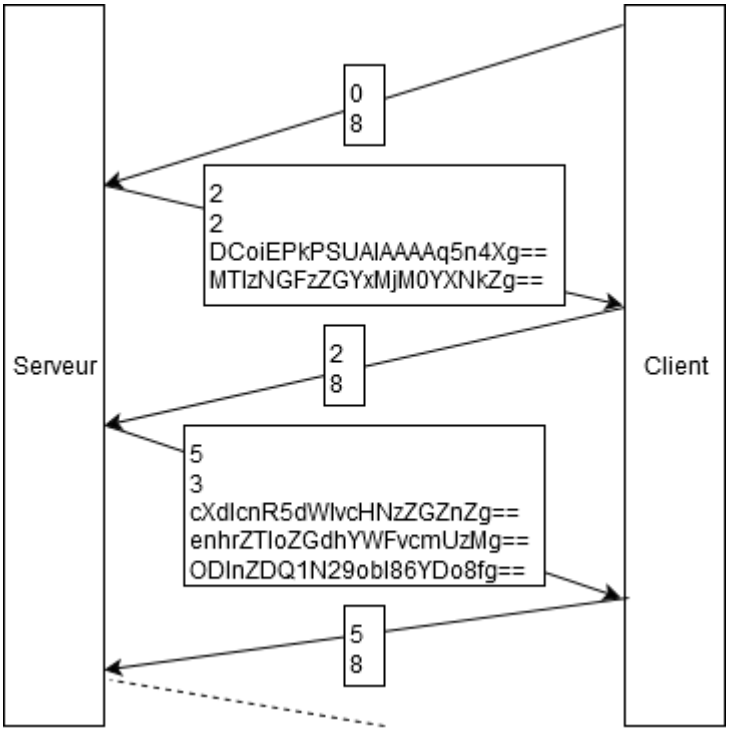

Figure 3. Exemple de transfert de données CAN serveur-client.

En ce qui concerne l'interface REST, nous avons utilisé la librairie cherrypy pour sa simplicité. Dans le but que notre code soit "auto-documenté", nous avons placé chaque méthode REST dans des classes imbriquées reproduisant fidèlement la hiérarchie des routes HTTP (par exemple, pour le GET sur "/config/rockets", on a une méthode GET dans la classe rockets_t imbriquée dans la classe config_t). Bien sûr, chaque méthode s'assure que l'usager est connecté ou est enregistré (pour le login) et retourne un code 401 sinon. L'identité du client est confirmée à chaque fois avec son adresse IP et son type d'appareil mobile (extrait du user-agent).

Pour ce qui est du GUI du serveur, celui-ci est démarré différemment si des arguments en ligne de commande sont passés ou non. Si au moins un argument est passé, les arguments par défaut sont utilisés pour les arguments non-spécifiés et une fenêtre qui ne fait qu'afficher les arguments est créée (fonction option_display_window).

Sinon, l'administrateur a la possibilité d'entrer les arguments manuellement dans une fenêtre différente (fonction option_select_window) avant d'appuyer sur le bouton Start et de changer la fenêtre pour celle de option_display_window.

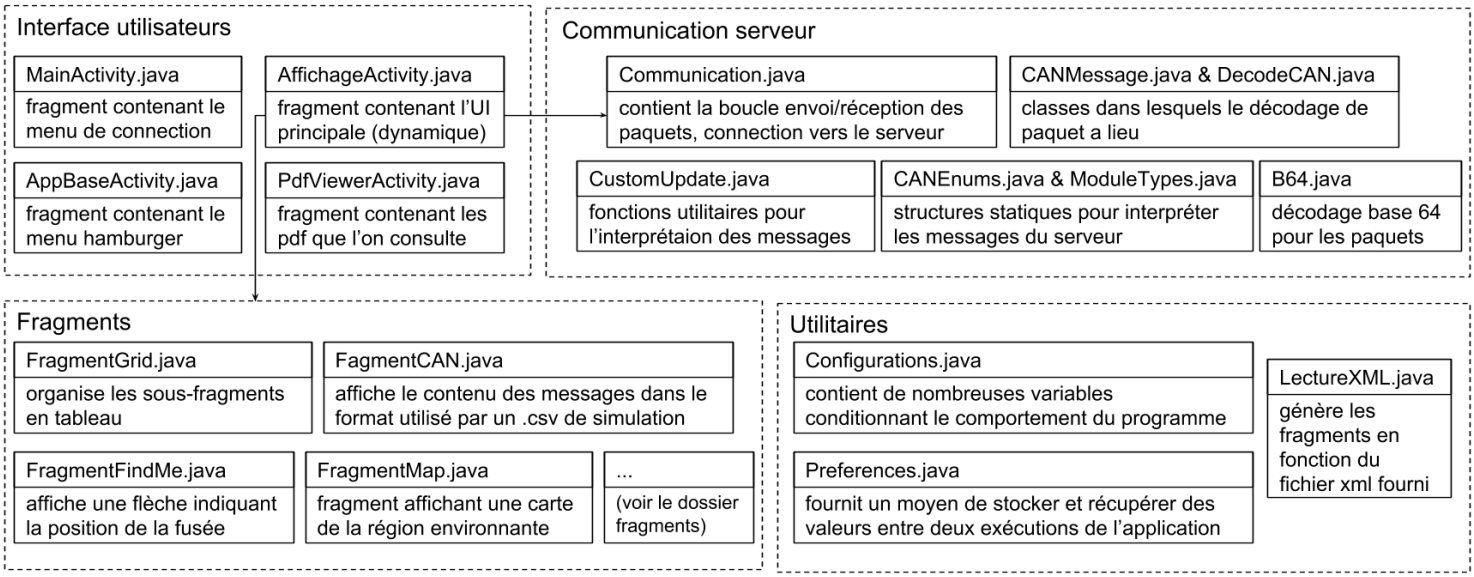

Figure 4 : architecture de la tablette

2.2 Tablette (Q4.6)

Au démarrage de l'application du côté de la tablette, cette dernière affiche l'interface graphique, qui est composée de plusieurs champs de saisie de texte, ces champs sont nécessaires pour accueillir les informations à envoyer par la méthode post de l'interface REST vers le serveur.

Une fois que les informations d'authentification sont envoyées, elles sont traitées par le serveur que lui à son tour renvoie un code de type numérique (200, 401, 404). Chaque code envoyé a une signification universelle, par exemple le code 200 signifie que l'information reçue par le serveur est correcte et que le traitement a réussi.

Parallèlement le fragment Interface IP est invoqué, celui-ci se charge de tout ce qui est en lien avec l'initialisation de l'adresse ip, il récupère la dernière adresse ip avec laquelle l'usager du support (tablette) a pu se connecter, après que l'usager eut fait son choix en sélectionnant la case convenable pour sauvegarder ces informations dans la mémoire interne de la tablette pour les prochaines connexions.

Après avoir reçu le code 200, le fragment RAM prend le relais, il télécharge toutes les données XML nécessaires du serveur et les charge en mémoire vive (temporaire). Alors le fragment Authentification est invoqué. Celui-ci provoque aussi l'affichage du GUI. À partir de ce fragment, le contrôle des champs à remplir est assuré par le module de vérification. Ce dernier contient toutes les règles relatives aux

données saisies par l'utilisateur du système. Après une authentification réussie, le fragment données en temps réel prend le relais et il peut afficher toutes les données qu'il reçoit du serveur.

Enfin le module contrôle d'affichage s'exécute en permanence en arrière-plan pour détecter tout demande de changement d'environnement (mode jour/nuit).

Du côté du code en tant que tel, comme tout projet, il est nécessaire d'importer les librairies nécessaires et cela se fait parfaitement dans le fichier gradle du projet ainsi que les permissions. Voici donc les principaux modules et librairies originaux utilisés lors du développement.

2.2.1 L'interface REST et la librairie OKHTTP

Pour que l'interface REST fonctionne et pour qu'une communication TCP se fasse entre la tablette et le serveur, on a fait appel à la librairie *okhttp*. Cette dernière demande la permission pour avoir une connexion à l'internet ainsi que l'espace de stockage interne. Cela s'est réalisé par l'ajout des propriétés convenables au *gradle*.

L'utilisation efficace de HTTP accélère le chargement et économise de la bande passante. *OkHttp* est un client HTTP efficace qui supporte HTTP / 2 et permet à toutes les demandes du même hôte de partager un socket.

Aussi, le regroupement de connexions à l'aide de *okhttp* réduit la latence des requêtes (si HTTP / 2 n'est pas disponible), le GZIP de *okhttp* transparent réduit les tailles de téléchargement et la mise en cache des réponses évite complètement les demandes répétées. Lorsque le réseau rencontre des problèmes, le *okhttp* récupère silencieusement ces problèmes de connexion.

Si le service a plusieurs adresses IP, *okHttp* tentera d'autres adresses si la première connexion échoue. Ceci est nécessaire pour IPv4 + IPv6 et pour les services hébergés dans des centres de données redondants. *OkHttp* initie de nouvelles connexions avec des fonctionnalités TLS modernes (SNI, ALPN) et retombe sur TLS 1.0 en cas d'échec de la liaison.

Utiliser *okHttp* est facile. Son API de (requête,réponse) est conçue avec des constructeurs fluides et mutables. Il prend en charge les appels de blocage synchrones et les appels asynchrones avec rappels.*OkHttp* supporte Android 2.3 ainsi que les versions ultérieures.

2.2.2 L'étiquette MAP et la librairie MapBox

Pour avoir une carte pour l'étiquette Map, nous avons utilisé la librairie de MapBox qui est une plateforme de données de localisation pour les applications mobiles et Web. Il fournit des blocs de construction pour ajouter des fonctionnalités d'emplacement telles que des cartes, des éléments de navigation et de recherche. Cette librairie est utile pour notre application parce qu'il permet d'avoir accès à la carte en mode hors ligne.

Il faut premièrement délimiter une région géographique. Ensuite, le SDK Mapbox Maps demande toutes les ressources requises auprès des serveurs de Mapbox et il les stocke dans une base de données sur le périphérique. Les ressources doivent être sous forme de JSON pour être stockées.

2.2.3 L'étiquette PLOT et la librairie AndroidPlot

Pour avoir un graphique dynamique pour l'étiquette Plot, on utilise la librairie open source AndroidPlot. Cette dernière nous a permis de créer des graphiques dynamiques, c'est-à-dire qu'on affiche des données envoyées par le serveur en temps réel. Dans notre projet, il inclut également deux occurrences de TextLabelWidget utilisées pour afficher un titre facultatif pour l'axe de domaine et de plage, et une instance de XYLegendWidget qui, par défaut, affichera automatiquement les éléments de légende pour chaque série ajoutée au tracé.

De plus, nos graphiques sont capables d'ajuster les intervalles de leurs axes automatiquement en fonction des données reçues (en fonction de l'axe Y).

Enfin, dépendamment du nombre de type de données reçues et qu'on désire afficher, un même graphique est capable d'afficher en temps réel jusqu'à 6 séries de données différentes.

2.2.4 L'étiquette FINDME et la librairie OpenGL ES 2.0

Le but de cette étiquette étant d'afficher une flèche 3D suivant la fusée par rapport à la position de la tablette, nous nous sommes tournés vers la librairie OpenGL ES, qui permet de créer des environnements OpenGL sur applications mobiles de façon bien plus légère en termes de mémoire.

Tout d'abord, avant d'afficher quoi que ce soit, il fallait déterminer la position de la tablette. Pour ce faire, la tablette a dû demander et activer les permissions de localisation par l'usager. Une fois faite, on utilise simplement un ''LocationListener'' qui nous déterminera la longitude et latitude de la tablette à chaque intervalle spécifique de temps. Nous avions par contre rencontré quelques difficultés qui on fait en sorte que l'on a du ''hardcoder'' la position de la tablette. Premièrement, les signaux GPS ne se rendaient pas jusqu'à la salle d'évaluation de projet, ce qui a fait en sorte que rien ne s'affiche lorsqu'on est à l'école. En plus, nous n'avions pas réussi à déterminer l'orientation en 3D de la tablette, ce qui a fait en sorte que l'usager doit tenir la tablette parallèlement au sol ainsi que de l'orienter directement vers le Nord afin que la flèche pointe constamment vers la position de la fusée. La partie de ce code a été conservée en commentaires dans le fichier FragmentFindMe.java.

Ensuite, nous avions besoin de l'autre pointe de la flèche: la position de la fusée. Pour ce faire, une fonction de mise à jour des données a été implémentée: dès qu'un message possède une longitude, latitude ou altitude de la fusée, on met à jour les variables correspondantes.

```java
@Override
public void mettreAJour() {
    for (int i=0; i< Communication.data.size(); i++) {
        CANMessage msg = Communication.data.get(i);

        // Longitude
        if(msg.msgID == CANEnums.CANSid.GPS1_LONGITUDE){
            myRenderer.rocketLongitude = -(double) msg.data1;
        }
        // Latitude
        else if(msg.msgID == CANEnums.CANSid.GPS1_LATITUDE){
            myRenderer.rocketLatitude = (double) msg.data1;
        }
        // Altitude
        else if(msg.msgID == CANEnums.CANSid.GPS1_ALT_MSL){
            myRenderer.rocketAltitude = (double) msg.data1;
        }
    }
}
```

Figure 5 : Fonction de mise à jour des données du fragment FindMe.

Il faut aussi noter que, afin d'afficher la flèche dans un environnement OpenGL, ces coordonnées ont dû être converties en coordonnées cartésiennes. Les fonctions illustrées dans l'image suivante ont été créées afin de bien convertir pour avoir une direction en 3D.

```
public float lookUp(){
    float heightAngle = (float) (180 *
            Math.tan(((rocketAltitude-clientAltitude)/1000)/
                    (Math.sqrt(Math.pow(subX, 2) + Math.pow(subY, 2)))) / Math.PI);
    return heightAngle;
}

public float findRocketDirection(){
    // Latitude et longitude en radians
    double radClientLatitude = clientLatitude * Math.PI / 180;
    double radRocketLatitude = rocketLatitude * Math.PI / 180;

    // Coordonnees cartesiennes du client
    double xClient = radiusEarth * Math.cos(radClientLatitude);
    double yClient = radiusEarth * radClientLatitude;

    // Coordonnees cartesiennes de la fusee
    double xRocket = radiusEarth * Math.cos(radRocketLatitude);
    double yRocket = radiusEarth * radRocketLatitude;

    // Difference
    subY = yRocket - yClient;
    subX = xRocket - xClient;

    // Calcul de l'angle en 2D
    float angle = (float)(180 * Math.atan(Math.abs((subY)/(subX))) / Math.PI);

    // Ajustement de l'angle dependamment de la direction desiree de la fleche
    if(rocketLatitude > clientLatitude && rocketLongitude > clientLongitude){
        angle -= 90.0f;
    }
    else if(rocketLatitude < clientLatitude && rocketLongitude < clientLongitude){
        angle += 90.0f;
    }
    else if(rocketLatitude < clientLatitude && rocketLongitude > clientLongitude){
        angle += 180.0f;
    }

    return angle;
}
```

Figure 6 : Fonctions de conversion d'unités pour les coordonnées du client et de la fusée mise à jour des données du fragment FindMe.

Maintenant que l'on possède une direction, on peut finalement utiliser les classes et méthodes de la librairie OpenGL ES afin de créer un environnement et d'afficher une flèche 3D tournant en temps réel en fonction de la position actuelle de la fusée. Les techniques apprises en INF2705 ont été appliquées.

3. Fonctionnement général (Q5.4)

3.1 Administration du serveur

Afin de faire fonctionner le serveur, assumant que la plateforme est déjà configurée tel que décrit dans la section 2.1.1, l'administrateur doit premièrement placer le répertoire du serveur sur la carte. Pour ce faire, l'administrateur peut cloner le dossier git ou encore utiliser un client FTP (e.g. transfert avec FileZilla). Après que le serveur ait été placé sur la carte, il sera probablement nécessaire d'enregistrer des usagers. Cela peut se faire en appelant le script comptes.py (utiliser l'option -h pour savoir les options possibles). Ensuite, le serveur peut être démarré avec le script main.py (sans options pour avoir le GUI initial, avec l'option -h pour voir les options).

Il est à noter que, comme les choses sont maintenant, le serveur doit être redémarré pour que les changements aux utilisateurs prennent effet. Aussi, pour fonctionner dans les labos de l'école, le port des données doit être entre 5000 et 5050. En outre, on note que le serveur doit être redémarré pour passer d'un mode de réception ("serial" ou "simulation") à l'autre. Pour le mode "serial", la connexion série doit être ouverte avant le démarrage du serveur et doit être ensuite maintenue. Pour le mode "simulation", il est préférable de mettre les fichiers .csv dans le dossier "simulation". En général, les fichiers .pdf devraient aller dans le dossier "misc" et les fichiers .xml de fusées devraient aller dans le dossier "rockets".

3.2 Utilisation du client

Pour compiler, c'est-à-dire pour l'installation de l'application sur la tablette, il suffit de cliquer sur le bouton "Run" (petit triangle vert) à partir d'Android Studio. Une fois l'application installée sur la tablette, elle peut être lancée en touchant son icône comme toute autre application. À l'ouverture de l'application, une première page s'affiche et demande d'entrer un nom d'utilisateur, un mot de passe et une adresse IP. Au besoin, on peut cocher les cases pour faire en sorte que l'application sauvegarde les données de l'usager et du mot de passe pour que le client n'ait pas à entrer ces informations à chaque fois, une démarche répétitive qui peut devenir rapidement fastidieuse sinon. Une fois authentifié, l'usager peut accéder à la page des données qui s'affiche en temps réel, selon le fichier de fusée utilisé. Il faut aussi noter que du côté de certains points du livrable 2, tels que l'esthétisme, l'ergonomie et la stabilité, l'application n'est pas totalement achevée. Par ailleurs, il est possible d'accéder au menu en touchant la frontière gauche de l'écran puis en glissant son doigt vers la droite. Ainsi, il est possible de changer le thème, d'accéder au fichier PDF qui provient du serveur et finalement de se déconnecter.

4. Résultats des tests de fonctionnement du système complet (Q2.4)

Le serveur marche à peu près intégralement :

Il reçoit les signaux UART et les retransmet, expose une interface REST permettant de s'authentifier et de télécharger les documents, peut être lancé en ligne de commande et avec un GUI comprenant un affichage des usagers; un bug restant a trait avec l'option donnée par le GUI d'enregistrement des préférences qui ne permet de lancer le serveur une fois sélectionnée.

Ce serait un euphémisme de dire que le client est incomplet :

Le fragment d'authentification et connexion marche, même si, dans certains cas, entrer une adresse IP incorrecte cause un crash. Seule une partie des affichages sont implémentées (fragment FindMe, fragment MAP, fragment CAN, fragment DataDisplayer, fragment Plot). Faire défiler un fragment vers la gauche cause un arrêt du programme (nous avons remarqué que cela se passe que lorsque l'on entre dans certains fragments spécifiques). Le changement de thème fonctionne, mais pas le téléchargement de fichiers (du moins lors du test avec le chargé, on peut par contre voir que la réception est bien faite en étudiant les logs en temps réel sur Android Studio). La réception des données CAN marche en simulation et avec un serveur lancé sur un ordinateur, mais pas avec la carte FPGA (nous pensons cependant que cela a quelque chose à voir avec les restrictions de port du laboratoire).

5. Déroulement du projet (Q2.5)

Ce qui a bien été réussi durant le déroulement du projet:

+ Les communications étaient bien gérées.
+ Le travail en binôme était présent.
+ Les tâches ont été explicitées et réparties au départ
+ Code base conséquente au cours du temps.
+ La reprise des tâches et les réparations étaient presque immédiates.

Ce qui n'a pas bien été réussi:

- Une fois qu'on a réalisé que la répartition des tâches était trop lourde pour les membres travaillant sur le client, cette répartition n'a pas bien été réajustée afin de bien compléter toutes les demandes du client.
- Travail de dernière minute avant la remise des deux livrable, ce qu'on voulait éviter.
- Manque inquiétant de test; une grande partie de l'application contenait plusieurs bogues lors de la remise.
- Certaines tâches n'ont pas été mises en priorité lorsqu'elles auraient dû l'être.
- Manque de prévention et de la gestion des risques.

- Certaines demandes étaient créées sur Redmine, mais assignées à personne.
- Il y avait un manque de vérification du travail des autres membres de l'équipe du côté des demandes à cause du manque de temps. Donc, souvent, on ne lisait pas nécessairement le code des autres qui était en lien direct avec le nôtre, ou essayer de comprendre en détail un document de spécification.

6. Travaux futurs et recommandations (Q3.5)

Du côté du serveur, il resterait à réparer les bogues de l'option des préférences qui ne permet pas son démarrage, ainsi qu'un test exhaustif afin de renforcer n'importe quoi qui causerait un crash potentiel.

La grande majorité de ce qui reste se trouve dans la partie de l'application Client. Côté stabilité, on observe une lacune majeure. En effet, il y a énormément d'actions qui causent des crashs, ce qui fait en sorte que certains aspects ne sont pas utilisables. Il faut donc améliorer la stabilité de l'application, compléter l'affichage des données sur plusieurs fragments, implémenter l'historique configurable dans FragmentPlot, mieux gérer les threads de l'application, implémenter des mesures de sécurité (vérifications, mutex, sémaphores, etc) ainsi que le perfectionnement de l'esthétisme.

7. Apprentissage continu (Q12)

Gabriel Demers:

Lors de ce projet, une lacune dans mon savoir-faire que j'ai remarqué se situe au niveau de ma spécification des interfaces entre mes parties du projet et celles des autres. Par exemple, je savais que la communication entre le serveur et le client ne devait s'effectuer que sur certains ports spécifiques (ex. 80 et 5000-5050) et je n'ai probablement pas assez insisté sur ce point. J'aurais peut-être aussi dû mieux communiquer l'importance du ‘'user-agent'' pour la détection du type d'appareil mobile utilisé.

De manière générale, la méthode principale qui a été prise de manière consistante pour remédier à ces problèmes a été d'assister directement les personnes ayant ces problèmes lorsqu'on m'en informait. Une possiblement meilleure solution serait de plus documenter les interfaces entre parties, mais je dois alors éviter de tomber dans deux pièges dans lesquels je suis tombé lors de ce projet: assumer que la plupart des gens ont un niveau de confort avec certains sujets qui est similaire au mien, et assumer que la plupart des gens sont motivés à faire certaines choses sans assistance (comme lire le code de quelqu'un d'autre en lien direct avec le leur, ou essayer de comprendre en détail un document de spécification).

Ces aspects pourraient être améliorés en prenant une approche plus proactive qui consisterait à réfléchir plus fortement sur les problèmes potentiels auxquels les membres de l'équipe pourraient faire face, puis d'aller voir ceux-ci plutôt que d'assumer qu'ils prendront l'initiative. Un inconvénient de cette approche est que cela me

demanderait plus de temps qui pourrait autrement être investi dans le raffinement de ma partie, mais cela pourrait tout de même en valoir la peine.

Khalil Bennani:

Durant ce projet, j'ai appris beaucoup de chose, la plus importante d'entre elle se résume comme suit:

Pour n'importe quel projet, il faut démarrer par une petite ligne de code et l'améliorer, le gonfler, ajouter d'autre ligne jusqu'à temps d'avoir un projet présentable. Notre grande lacune était d'essayer de construire chacun de nous, individuellement, une tâche ou une sous-tâche et d'essayer de connecter toutes ces pièces d'un énorme problème.

Chacun des membres avait un savoir-faire dans un langage ou dans une technologie spécifique, mais on n'avait pas l'expertise et le savoir de mettre en valeur notre savoir-faire.

J'ai essayé de communiquer au maximum avec les autres membres de l'équipe, posé des questions et travaillé en pair pour combler cette lacune.

En bref, si on avait codé par pair depuis le début, je ne pense pas qu'on aurait pu faire face à ce type de problème.

Mounia Nordine:

Ce projet est différent des projets que j'ai faits avant, il m'a permis d'apprendre le développement d'une application sur Android Studio et l'intégration des librairies de code source ouvert.

Durant les anciens projets, je ne prenais jamais l'initiative sur quelque chose que je n'ai pas de l'expérience. Donc, j'ai essayé de travailler sur ce point cette session pour me confronter à différents problèmes.

Selon mon point de vue, notre équipe avait une lacune au niveau de la division des tâches au départ ainsi qu'au niveau de la compréhension totale du projet. Pour éviter cette erreur, durant les prochains projets, je vais insister sur une lecture en groupe pour que tous les membres de l'équipe aient le même angle de vue.

Luc Courbariaux:

Je ne pense pas avoir acquis de nouvelles compétences du point de vue technique que ce soit logiciel ou informatique au cours de ce projet.

J'ai découvert le cadriciel Java Android, tentant de remplacer les outils communs des développeurs (par exemple AsyncTask à la place d'un Thread), une nouvelle raison de ne pas toucher à Java parmis les quelques milliers que je connais.

Mes lacunes ont à mon avis été plutôt au niveau de la logistique ou communication au sein de l'équipe. J'ai, en compagnie d'Anthony, essayé de mettre en

place un calendrier des tâches et garder en tout temps une idée de l'avancée du projet. Cependant il semble que cela a été insuffisant. J'ai entre autres trop donné priorité à l'exécution de mes tâches techniques assignées, pour lesquelles j'ai plus de goût.

À l'avenir, j'essaierai d'être encore plus à cheval sur les échéances internes prévues au sein de mon équipe le long du projet, et ne pas hésiter à émettre l'idée de changer de main les tâches qui ont attendu trop longtemps à être accomplies. Bien que nous nous soyons entendus à le faire lors de ce projet, cela n'est pratiquement jamais arrivé.

Anthony Abboud:

D'un point de vue technique, ce projet m'a permis d'apprendre le langage Python ainsi que le développement d'application Android.

Du côté de mon savoir-faire, une lacune identifiable serait mon manque de connaissances profonds sur certains aspects de l'informatique; même si on a vu quelque chose dans un cours passé, ça me prend un peu de temps pour me rappeler sur la procédure d'implémentation. Donc, j'ai fait ce que je fais toujours, je passe un peu de temps à relire des notes ou visionner des vidéos, pour ensuite passer à une implémentation rapide et efficace de mes demandes.

Cela fait par contre en sorte que, quelques fois, certaines petites demandes me prennent beaucoup plus de temps que prévu. Une solution pour remédier à ce problème serait de poser plus de questions à mes coéquipiers qui sont experts dans un domaine spécifique. Ainsi, je démontre plutôt ma force du côté de la gestion d'équipe, puisque j'ai déjà énormément d'expérience et de maîtrise dans cet aspect-là.

Reph D. Mombrun:

Par rapport au travail pour un(e) client(e), je retiens comme priorité, la clarification aussitôt que possible de tous les détails du document de spécification des requis du système (SRS), pour faire une analyse adéquate du problème. Sachant que les requis changent en général, il faut aussi maintenir une communication constante avec le client. De plus, il faut communiquer très régulièrement avec les membres de l'équipe. Enfin et surtout, je retiens qu'il faut faire preuve d'initiative et de proactivité pour essayer de prévenir des situations problématiques. Ces points mentionnés constituent des points à travailler davantage de mon côté, afin d'atteindre d'excellents résultats plus souvent à l'avenir.

Du point de vue technique, j'en ai appris énormément que ce soit seul ou en travaillant en binôme : travailler avec des documents XML en Java, compréhension des fragments sous Android, chiffrement de mot de passe en python, petits serveurs de tests multi usagers Python TCP/UDP, bien définir des problèmes, comment mettre sur pied un environnement 3D avec OpenGL ES 2+ dans un fragment et pas simplement dans une activité, utilisation de socket sous Android, traitement de messages can, localisation sous Android et j'en passe. Donc, l'expérience était extrêmement enrichissante et sera certainement mise à profit dans le futur.

8. Conclusion (Q3.6)

Notre conception du serveur a été très proche du produit tel que réalisé; on utilise le même graphique pour ce rapport que pour la réponse à l'appel d'offre.

Le client, quant à lui, a été modélisé de manière quelque peu abstraite alors, et notre vision actuelle est très différente. Au départ, nous n'avions pas en tête que l'affichage des données devait être fait dynamiquement, et les problèmes qui viennent avec ce requis.

Nous prenons tout de même une certaine fierté à l'idée que bien qu'incomplet, le logiciel que nous avons réalisé ne demanderait en réalité que peu de travail supplémentaire (de l'ordre d'une dizaine d'heures-ingénieur) afin de le rendre totalement fonctionnel.

9. Références

- Documentation de la librairie d'AndroidPlot: https://github.com/halfhp/androidplot/blob/master/docs/index.md

- Documentation d'OpenGL ES: https://developer.android.com/guide/topics/graphics/opengl.html

- Documentation étudiée pour le GPS: https://developer.android.com/guide/topics/location/strategies.html