POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Département de génie informatique et génie logiciel

INF3995 : Projet de conception d'un système informatique

Exigences techniques - Système mobile de télémétrie en temps réel

Complément au document de demande de proposition no. H2018-INF3995

version 1.0

Adrien Lessard, chargé de laboratoire
Anas Balboul, chargé de laboratoire
Jérôme Collin, ing., M. Sc. A

Février 2018

Table des matières

1 But du document

Le présent document complète l'appel d'offres H2018-INF3995 en précisant les exigences techniques dont les soumissionnaires devront tenir compte pour présenter des propositions conformes.

Les propositions devront détailler et démontrer comment ces exigences pourront être rencontrées. Elles devront également servir de point de départ à l'élaboration du calendrier des diverses tâches à réaliser.

2 Aperçu du produit demandé

Oronos possède un système de télémétrie qui permet de monitorer le statut de ses fusées durant les opérations au sol et durant les vols. Ce système est décrit dans la figure 1. Les données proviennent de la fusée et sont transmises par ondes radio vers l'antenne au sol à un débit de 500kbits/s. L'antenne est connectée à un PCB au sol qui fait le traitement des données reçues. Les données traitées sont transférées à un ordinateur portable agissant comme serveur. Le format du flux de données entrant dans le serveur est décrit à la section 7.1. Le serveur se connecte à un nombre arbitraire de clients (autres ordinateurs portables) et leur transmet le flux de télémétrie dans le même format. Finalement, chaque client analyse le flux de données et l'affiche dans une interface utilisateur.

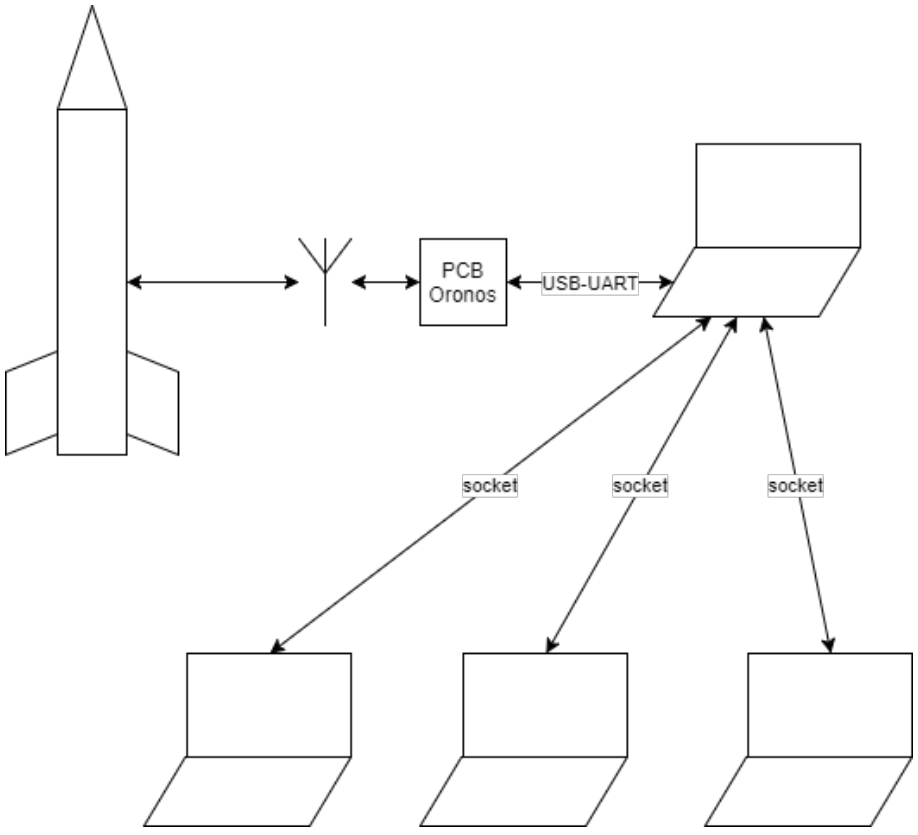

FIGURE 1 – Flot des données - système existant

Le produit à développer apportera du support pour des client mobiles sur des plateformes Android, tel qu'illustré dans la figure 2. Un Zedboard agira en tant que serveur et enverra le flux de données vers les appareils mobiles. Le flux réel de données respecte le format décrit dans la section 7.1. En l'absence d'une vraie fusée pour développer le système, le flux de données et devra être simulé pour en démontrer la fonctionnalité. La simulation devra être faite en émettant des données USB-UART vers un des ports USB du Zedboard, remplaçant ainsi le PCB Oronos, l'antenne et la fusée de la figure 2.

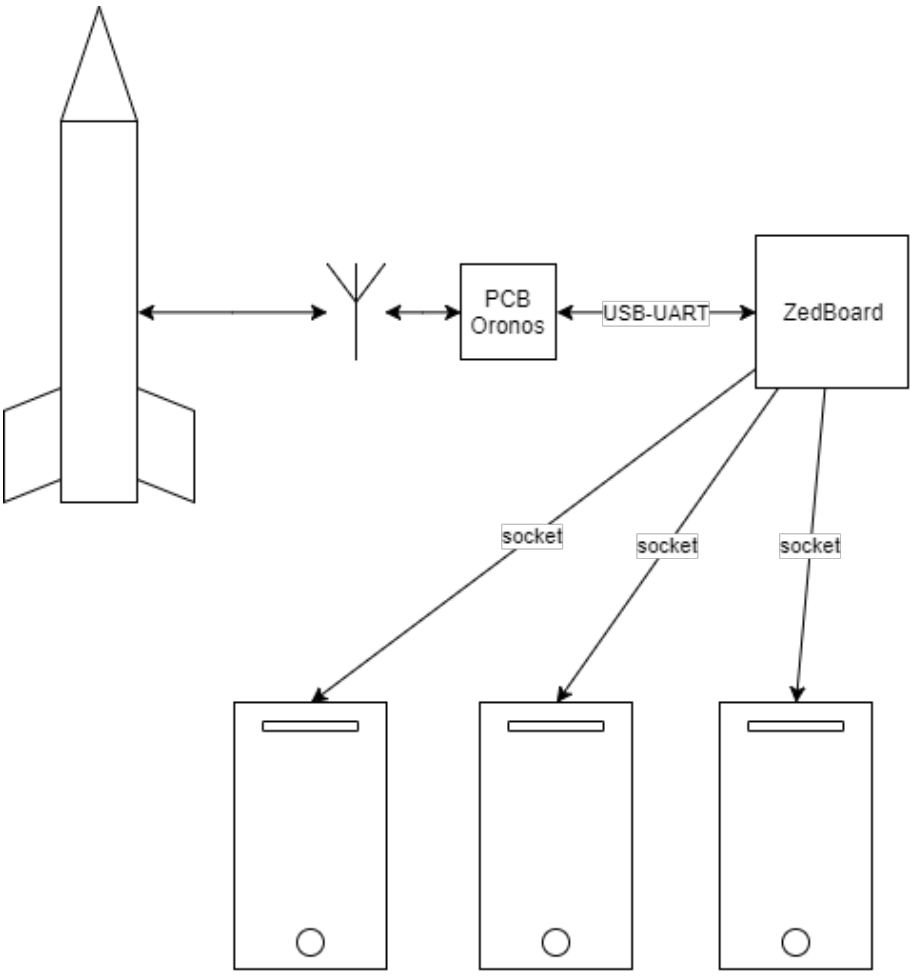

FIGURE 2 – Flot des données - système à développer

De plus, le serveur devra avoir la capacité de jouer en "playback" des logs existants, dont le format est spécifié dans la section 7.2 et une représentation du flot de données présentée à la figure 3.

Pour le plaisir de tester un vrai système, une vraie fusée sera mise à la disposition des équipes durant les périodes de laboratoire. Après le projet réalisé, le code livré sera portable du Zedboard à un ordinateur conventionnel et permettra à Oronos de réaliser le système final, comme décrit à la figure 4.

3 Système

Cette section décrit le système envisagé. Il s'agit ici d'un logiciel s'exécutant sur un système embarqué muni d'un FPGA.

3.1 Matériel

Tel qu'énoncé précédemment, les périphériques disponibles sur le système sont ceux de la carte de développement Zedboard. Celle-ci est munie d'un FPGA/SoC Zynq XC7Z020 de Xilinx, lequel contiendra la logique nécessaire à l'exécution de l'appareil (bus, processeurs, pilotes, intermédiaires logiciel/matériel, etc.) On suppose que l'équipe de développement est déjà familière avec l'environnement de développement de Xilinx et de Linux. L'environnement de développement recommandé est la suite Xilinx Vivado/SDK 2017.2 et une distribution Ubuntu avec interface graphique Xfce suivant la version Analog Device Inc. 2016_R2.

Le système doit être muni, au minimum, des ressources suivantes :

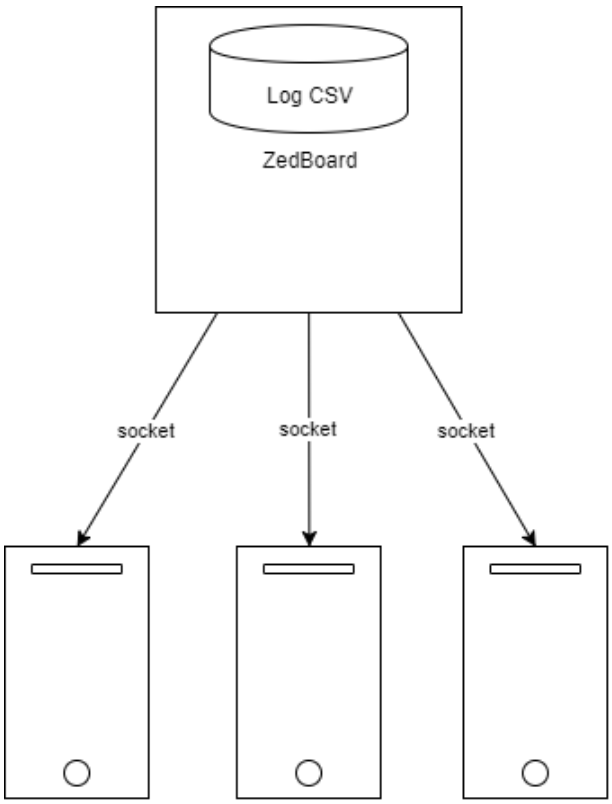

FIGURE 3 – Flot des données - lecture d'un log csv

— un processeur ARM Cortex-A9 proprement configuré pour le contexte ;
— un accès à une mémoire vive externe de 512 MiB ;
— un accès à une mémoire SD externe ;
— un accès à l'interface réseau ;
— un pilote pour contrôler l'interface HDMI (vidéo) ;
— un pilote pour contrôler un UART (pour une console en mode administrateur) ;
— un port USB générique.

3.2 Logiciel

Les contraintes logicielles :

— L'application serveur devra être du python3 seulement ;
— Les modifications apportées aux fichiers fournis doivent être rendues sous forme de patch (fichier .patch) ;
— Mentionner les versions utilisées de ses logiciels ;
— Fournir les fichiers de configuration ;
— Prévoir qu'il faudra fournir les commandes d'installation de ces logiciels ;
— Faire valider auprès du promoteur tout logiciel supplémentaire utilisé dans le produit final livré.

3.3 Android

Tablette mobile avec Android 6 ou plus avec ;

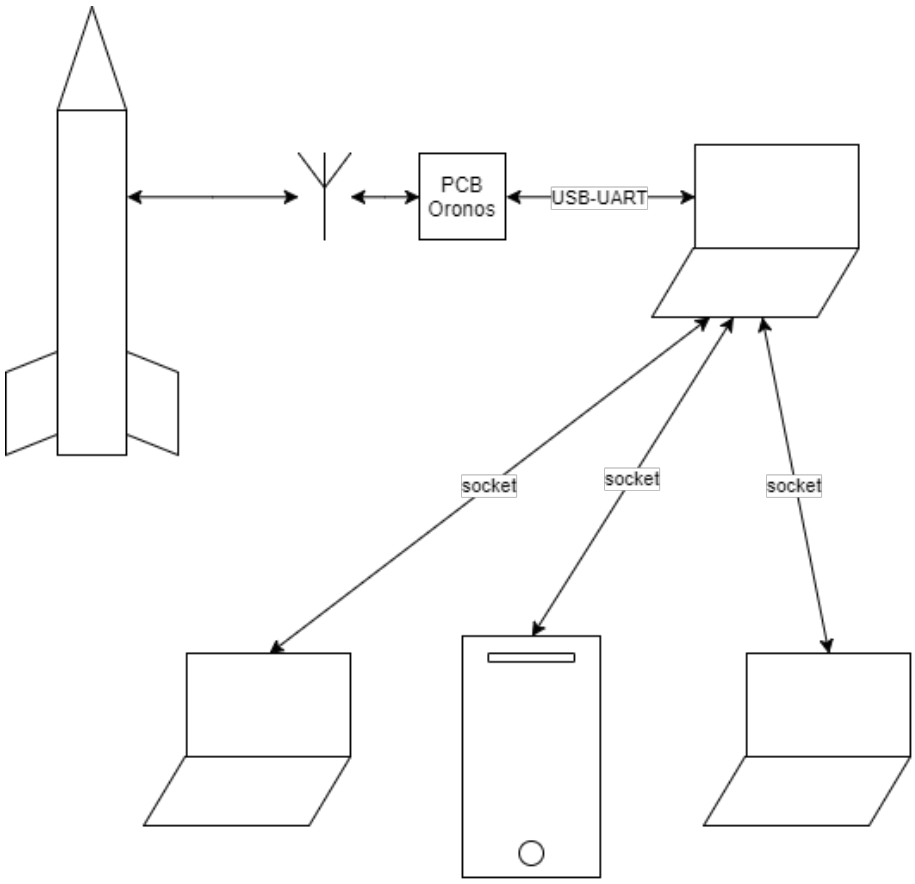

FIGURE 4 – Flot des données - intégration par Oronos après le projet

— Utilisation d'Android Studio pour le développement ;
— Développement en Java obligatoire ;
— Bien identifier les librairies Java utilisées
— Faire valider auprès du promoteur tout logiciel supplémentaire utilisé dans le produit final livré.

4 Fonctionnement du serveur

Le serveur devra être démarré en ligne de commande et devra accepter certains arguments sur la ligne de commande pour favoriser un démarrage rapide. Les arguments sur la ligne de commande ont leur équivalent dans une interface graphique. Ces arguments sont le numéro du port série utilisé, sa vitesse de transmission, le type d'affichage pour la fusée considérée, le lieu de lancement et la saveur du thème de l'interface usager. Invoqué sans argument, le serveur affiche le GUI et permet à l'utilisateur d'entrer ses choix et d'appuyer sur "Start". Invoqué avec des arguements, le GUI s'affiche également mais ne fait qu'afficher les options déjà choisies sur la ligne de commande en plus d'afficher d'autres informations.

La figure 5 montre l'interface usager utilisé présentement par Oronos. Il faudra ajouter des portions d'interface pour les "logs" et des informations sur les clients (voir sous-section 7.4).

Le programme invoqué avec l'option "-h" montre les autres arguments possibles du script Python tel que montré ici :

```
% main.py -h
usage: main.py [-h] [-b BAUDRATE] [-c {serial,simulation}]
               [-f CONNECTOR_FILE] [-s [PORT]] [-r ROCKET] [-m MAP]
```

```
Description:
  Programme serveur permettant d'avoir la communication avec
  une fusée d'Oronos et des PC/tablettes client.

Arguments optionnels:
 -h, --help
      Message d'aide et sortie immédiate.
 -b BAUDRATE, --baudrate BAUDRATE
      Baudrate du port série (connecteur sériel seulement).
 -c {serial,simulation}, --connector_type {serial,simulation}
      Source des données à traiter. (défaut: simulation)
 -f CONNECTOR_FILE, --connector_file CONNECTOR_FILE
      Argument pour le type de connecteur c'est-à-dire le
      port série COM si en mode serial ou le fichier de
      données CSV pour le mode simulation)
 -s [PORT], --server [PORT]
      Active un server pour avoir plusieurs stations au sol.
      Si besoin, spécifiez un port (3000 par défaut)
 -r ROCKET, --rocket ROCKET
      Le nom du fichier XML. Ex:
      10_polaris.xml. (Default: 11_valkyrieM2.xml)
 -m MAP, --map MAP
      Le nom de la carte qui est contenu dans
      /Configs/Other/Maps.xml. Ex: motel_6.
      (Default: spaceport_america)

Copyright 2018. Oronos Polytechnique. Tous droits réservés
```

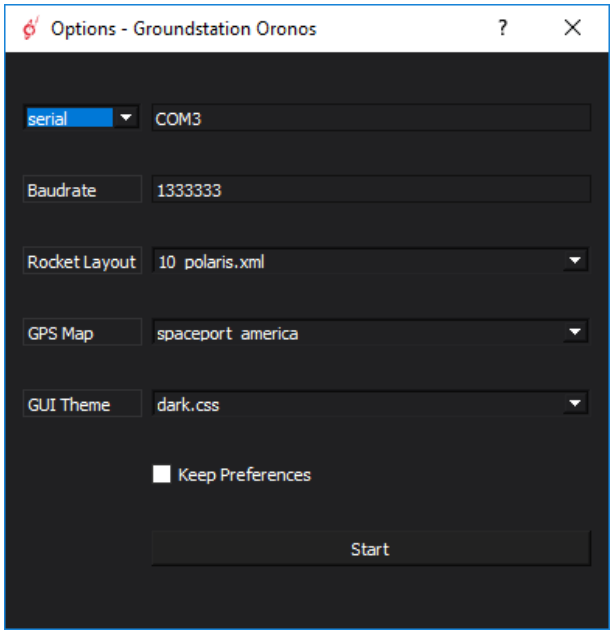

FIGURE 5 – Interface graphique du serveur

Le serveur a comme but principal d'envoyer les données reçues de la fusée sur un socket de communication différent du port 80 qui, lui, sera réservé pour l'interface REST web ordinaire pour toutes les autres formes d'interaction entre la tablette et le serveur. Une certaine liberté vous est laissée quant au transfert du flux

de données vers l'appareil mobile. Les données peuvent être transférées en format brut (base64) ou à travers une autre solution de votre choix (bien documentée !). Les données peuvent avoir subi du formatage du côté du serveur si vous en trouvez le besoin.

Le port 80 présentera une interface REST et peut être vue comme un serveur web conventionnel. Il devra offrir des comptes pour permettre la connexion des tablettes. Les clients devront s'authentifier auprès du serveur en utilisant un compte. Puisque la sécurité n'est pas très importante dans le désert du Nouveau Mexique où tout accès à Internet est impossible, il suffira de stocker les comptes et les mots de passe dans un fichier xml sur le serveur. Les mots de passe devront être stockés en sha512, salé, calculé 0xCAFE fois. Un petit utilitaire python en ligne de commande devra être fourni pour ajouter des utilisateurs, supprimer des utilisateurs et modifier le mot de passe d'un utilisateur.

Le serveur offrira aussi à la tablette la possibilité de télécharger automatiquement des fichiers de base nécessaire aux fonctionnement de la tablette au départ et les réglages de base pour s'accorder avec ce que le serveur lui enverra comme données (carte, type de fusée, etc.) Le serveur rendra aussi disponible à la tablette des fichiers PDF ordinaires pouvant être téléchargés de la tablette et pouvant être consultés par l'usager à tout moment pour des besoins complémentaires à la compétiton (enregistrement, dessins, etc.)

POST /users/login

sera utilé pour se connecter au serveur et être reconnu par le système, il faudra fournir le nom d'usager tout comme le mot de passe. Le JSON à envoyer au serveur sera le suivant :

```
{
  "username": string,
  "password": string
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé

POST /users/logout

Il faut évidemment un complément à la requête précédente. Le fichier JSON à envoyer au serveur aura la structure suivante :

```
{
  "username": string,
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé (déjà déconnecté)

GET /config/basic

Le serveur est déjà configuré pour une situation particulière (fusée, endroit, etc...) Il faut donc retransmettre ces informations aux tablettes. Le serveur retournera :

```
{
  "otherPort": integer, // port pour les données brutes
  "layout": string, // nom du fichier des informations
                       pour le modèle de fusée considéré
  "map": string,  // carte pour le terrain de lancement
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé

GET /config/rockets

Permet à la tablette d'obtenir les noms de tous les fichiers XML décrivant les formats de données émis par les différentes fusées :

```
{
  "file1": string, // exemple: polaris.xml
  "file2": string, // exemple: valkyrie_ii.xml
  "fileN": string  // jusqu'au dernier...
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé

GET /config/rockets/<name>

Pour récupérer le contenu du fichier XML *name*.xml d'une fusée donnée et spécifié dans la liste précédemment reçue. Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé
- 404 : fichier inexistant

GET /config/map

Pour que la tablette puisse savoir quel terrain de lancement a été précisé au lancement du serveur. La tablette a déjà la carte et les informations associées dans ses données de fonctionnement intrinsèque.

```
{
  "map": string, // exemple: motel_6
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé

GET /config/miscFiles

Permet à la tablette d'obtenir des noms de fichiers PDF divers stockés sur le serveur en rapport avec la compétition ou la compétition :

```
{
  nFiles: integer, // peut être zéro.
  "file1": string, // exemple: inscriptions.pdf
  "file2": string, // exemple: attestation.pdf
  "fileN": string  // jusqu'au dernier...
}
```

Le serveur retournera les codes HTTP d'états possibles suivants pour confirmer ou rejeter la transaction :

- 200 : ok
- 401 : non autorisé

GET /config/miscFiles/<name>

Pour récupérer le contenu du fichier PDF *name*.pdf qui peut être utile d'avoir aussi sur la tablette :

- 200 : ok
- 401 : non autorisé
- 404 : fichier inexistant

5 Fonctionnement de la tablette Android

La tablette est relativement passive en ce sens qu'elle envoie très peu d'information du serveur mais en reçoit beaucoup. Au démarrage de l'application Android, la tablette devra présenter un fragment permettant de préciser l'adresse IP du serveur avant d'amorçer la connexion. L'application devra mémoriser cette adresse pour éviter que l'usager doive la préciser chaque fois mais le fragment devra toujours être présenté pour permettre un changement. Avec l'adresse IP, il devra y avoir un moyen d'entrer le nom d'usager et le mot de passe.

Un menu de type "hamburger" au haut de l'application permettra aussi de spécifier le thème à utiliser. Avec ce menu, il sera aussi possible d'accéder aux quelques fichiers PDF sur le serveur et qui peuvent être utiles en compétition.

Évidemment, la dernière opération à préciser sera de lancer le mode de réception des données sur le port prévu à cette fin et de procéder à l'affichage des données en temps réel. Cette partie est la plus critique et exigera une bonne part de créativité de votre part.

6 Données d'Oronos

Les données transmises par télémétrie sont toutes issues du bus CAN de la fusée, qui correspond à une séquence binaire omniprésente dans les communications. Le protocole CAN prévoit 11 bits d'identifiants de messages, 18 bits d'identifiant supplémentaire et 64 bits de données. Le fichier fourni CANSid.csv établit la liste des identifiants de 11 bits utilisés. Le format est le suivant :

`CAN_SID_TXT;DATATYPE1;DATATYPE2;COMMENT`

Le fichier generateEnums.py donne l'attribution des valeurs numériques aux messages CAN. Remarquez que l'ajout d'un message CAN dans le milieu de la liste peut provoquer un décalage des identifiants suivants. Le logiciel développé devra être robuste à ce genre de modification.

Les types `DATATYPE1` et `DATATYPE2` dictent comment la donnée doit être interprétée, en fonction des types possibles, tel que décrit dans le tableau 1

Table 1 – Valeurs possibles pour DATATYPE1 et DATATYPE2

Type	Format
INT	Entier signé 32 bits
FLOAT	IEEE 754 float, 32 bits
UNSIGNED	Entier non signé, 32 bits
TIMESTAMP	Entier non signé, 32 bits. Représente le nombre de millisecondes écoulées depuis le début de l'exécution sur le PCB concerné dans la fusée. Le type TIMESTAMP est souvent utilisé pour représenter exactement le moment où une donnée a été échantillonnée d'un capteur et sert par la suite à tracer des graphes précis.
MAGIC	Valeur hexadécimale, 32 bits. Correspond au crc32 de la valeur textuelle de l'dentifiant CAN. Sert de "mot de passe" pour les messages importants qui ont des conséquences importantes s'ils sont envoyés (p. exemple détonner 8g de poudre noire). Il va de soi que la réception de la mauvaise valeur de MAGIC cause un rejet du message. Le calcul de la valeur MAGIC est présenté dans le listing 1
NONE	32 bits assignés à 0. Possible uniquement dans DATATYPE2. Indique que la donnée sera inutilisée.

Listing 1 – Calcul du CRC32

```
uint32_t MAGIC = crc32(const char* CAN_SID_TXT);
```

7 Données à transférer par le serveur

Le serveur lit les données de télémétrie et les transfère par réseau vers les appareils mobiles pour affichage. Il y a plusieurs formats de données de télémétrie :

7.1 Format 1 - Stream USB

Comme montré dans la figure 1, un PCB d'Oronos émet des données en UART, qui sont converties en USB-UART directement par la carte d'Oronos. Il suffira de brancher le PCB d'Oronos par un cable Mini-USB, et de brancher l'autre bout sur un port USB du Zedboard, qui sera perçu comme port série COM (probablement /dev/ttyUSB0) par le système d'exploitation Linux.

Le flux de données se fait dans un format encodé en base 64 qui, lorsque décodé, respecte le format de la structure de données suivante :

```
typedef struct
{
        unsigned SID:11;
        unsigned destSerial:4;
        unsigned destType:5;
        unsigned srcSerial:4;
        unsigned srcType:5;
        unsigned :3;
        union{
        uint8_t datab[8]
        uint32_t datai[2];
        uint32_t datau[2];
        float dataf[2];
    };
        uint32_t crc32;
} UARTRxStruct;
```

Les champs destSerial, destType, srcSerial et srcType constituent ensemble les 18 bits d'identifiants supplémentaires prévus par le protocole CAN. Un CRC de 32 bits est inclut par Oronos à la fin du message. Tout message ayant un CRC32 invalide sera rejeté. Le CRC est calculé sur tous les éléments, sauf les 32 bits de CRC, c'est-à-dire de SID à data{b,i,u,f}.

Un PCB d'Oronos sera mis à la disposition des équipes durant les périodes de laboratoire. Le fonctionnement avec un PCB d'Oronos est nécessaire.

Pour effectuer des tests hors des séances de laboratoire, chaque équipe devra simuler un flux de données en temps réel. Le flux de données devra être fourni au système Linux par une méthode de votre choix. Vous devrez construire le stream de données en fonction d'un des fichiers de logs csv fourni. Il serait évidemment souhaitable que ces tests exercent tous les cas possibles de données.

7.2 Format 2 - log en csv

Les systèmes d'Oronos enregistrent les données de vol dans des logs en format csv. Il est ainsi possible de rejouer les vols passés ou de tester des nouvelles fonctionnalités sur des données réelles. Oronos stocke les logs en format textuel parce que les valeurs numériques des identifiants peuvent changer. Les logs en csv respectent le format suivant :

`TIMESTAMP;DIRECTION;SRC_TYPE;SRC_SERIAL;DST_TYPE;DST_SERIAL;CAN_MSGID;DATA1;DATA2`

Les champs `SRC_TYPE` et `DST_TYPE` correspondent à un membre de **enum** ModuleType en annexe. Les champs `SRC_SERIAL` et `DST_SERIAL` correspondent au numéro de série du PCB sur 4 bits et peuvent prendre des valeurs de 0 à 14 inclusivement. La valeur 0xF est réservée pour des communications de type broadcast.

Le champ `DIRECTION` détermine dans quel sens les données circulent (vers le sol ou vers la fusée). Dans le cadre du travail à rendre, seules les données vers le sol (`IN`) seront considérées.

Le champ `TIMESTAMP` représente le temps où le message a été reçu par le serveur lors du vol. Le temps commence à 0s à la première ligne du log. Le format correspond à des éléments de la structure `UARTRxStruct` :

- `SRC_TYPE = UARTRxStruct.srcType`
- `SRC_SERIAL = UARTRxStruct.srcSerial`
- `DST_TYPE = UARTRxStruct.destType`
- `DST_SERIAL = UARTRxStruct.destSerial`
- `DATA1 = 32 premiers bits de UARTRxStruct.data{b, i, u, f}`
- `DATA2 = 32 derniers bits de UARTRxStruct.data{b, i, u, f}`

Quelques fichiers de log "réels" seront fournis et devront être placés dans le système de fichiers Linux du Zedboard. Ils ont été obtenus à partir de lancements passés.

7.3 Log d'exécution

Pour chaque exécution, le serveur devra générer un fichier de log pour enregistrer les événements survenus. Tout événement anormal ou pertinent doit être enregistré. Le fichier doit contenir la date et l'heure dans son nom. Chaque nouvelle ligne commence par une date et une heure, suivie d'un tag INFO, DEBUG, WARNING ou ERROR. Remarque : il n'est pas nécessaire d'enregistrer tous les messages de télémétrie entrants pour des raisons de mémoire.

7.4 Affichage de débug

Pour chaque exécution, le serveur devra afficher sur son écran, en temps réel, dans une console, le même contenu que le log d'exécution.

Une seconde console devra afficher en temps réel les usagers connectés, le temps écoulé depuis l'atablissement de chaque connexion et le type d'appareil mobile utilisé ainsi que son adresse IP.

8 Layout

Le UI doit être construit dynamiquement par des fichiers XML qui vous sont fournis. Il y a un fichier XML par fusée. Le layout en XML a été conçu pour un affichage sur un écran d'ordinateur portable de 1366x768 pixels. Ainsi, la hiérarchie des éléments n'est pas nécessairement adaptée à un affichage mobile. Les tags `GridContainer` et `Grid` pourront être ignorés si jugé nécessaire. La hiérarchie des autres éléments doit être respectée. Une certaine liberté vous est donnée quant à l'action à prendre pour remplacer `GridContainer` et `Grid`. Cependant, **deux actions (swipe, tap, etc.) seulement devront être nécessaires pour accéder à toute information**.

L'orientation de la tablette doit être bloquée en mode portrait.

8.0.1 Tag Rcoket

Exemple : `<Rocket name="Polaris" id="10">`. Conteneur principal du XML. L'attribut `name` contient le nom textuel de la fusée. L'attribut `id` contient l'identifiant unique de la fusée. Le titre de l'applicatoion doit contenir ces informations.

8.0.2 Tag GridContainer

Exemple : `<GridContainer>`. Conteneur principal d'une grille de widgets placés dans une grille. Ce tag doit contenir uniquement des tags `Grid`. Tel que décrit plus haut, ce tag peut être ignoré.

8.0.3 Tag Grid

Exemple : `Grid col="1" row="2"`. Conteneur de widgets. Les attributs `row` et `col` donnent la position de `Grid` dans le `GridContainer`. Tel que décrit plus haut, ce tag peut être ignoré.

8.0.4 Tag TabContainer

Exemple : `<TabContainer>`. Ce conteneur regroupe plusieurs tags `Tab` et place chacun d'entre eux dans un tab. Aucun attribut n'est nécessaire.

8.0.5 Tag Tab

Exemple : `<Tab name="GPS and Temperature">`. Ce conteneur décrit le nom que prendra le tab avec l'attribut `name`. Il doit contenir seulement un widget.

8.0.6 Tag DualVWidget

Exemple : `<DualVWidget>`. Ce widget contient exactement deux widgets en layout vertical. La démarcation entre les deux widgets est claire.

8.0.7 Tag DualHWidget

Exemple : `<DualHWidget>`. Ce widget contient exactement deux widgets en layout horizontal. La démarcation entre les deux widgets est claire.

8.0.8 Tag DebugOptions

Exemple : `<DebugOptions/>`. Ce widget est un mode de débug pour la version PC seulement. Ne pas utiliser.

8.0.9 Tag ButtonArray

Exemple : `<ButtonArray nbColumns="1">`. Ce widget sert à envoyer des commandes vers la fusée. Seule la version PC est autorisée à envoyer des commandes. Ce widget peut être ignoré.

8.0.10 Tag Modulestatus

Exemple : `<Modulestatus nGrid="15" nColumns="5" />`. Ce widget sert à indiquer le statut de chaque PCB dans les systèmes avionique. Il permet de déterminer en un regard si tous les systèmes sont actifs. Le widget contient une grille déterminée par les attributs `nGrid` et `nColumns`, qui ajoute dynamiquement un élément à chaque nouveau PCB détecté. Le nom du PCB (enum ModuleType) et son numéro de série doivent clairement être indiqués. Une couleur de fond verte et le statut ONLINE doit être affichée lorsqu'au moins un message a été reçu de ce PCB dans les dernières deux secondes. Si le dernier message date entre 2 et 4 secondes, la couleur de fond doit être orange et le statut doit être DELAY. Si le dernier message date de 4 secondes et plus, la couleur de fond doit être rouge et le statut doit être OFFLINE.

8.0.11 Tag DisplayLogWidget

Exemple : `<DisplayLogWidget/>`. Ce widget affiche textuellement des messages entrants reçus, convertis dans le format CSV. Une limite du nombre de messages peut être imposée pour éviter de saturer la mémoire des appareils (5000 ?).

Si des tags enfants CAN sont présents, l'affichage des messages doit se restreindre à ceux spécifiés. Si non, tous les messages sont affichés.

8.0.12 Tag Plot

Exemple : `<Plot name="" unit="g" axis="Acceleration">`. Les widgets Plot permettent de créer un graphique qui se met à jour au fur et à mesure que les données arrivent. Un ensemble de tags enfants CAN décrivent les séries de données à afficher sur le graphique. Un maximum de 6 séries de données seront affichées sur un même graphe. Le graphe a un historique de données (en secondes) configurable dans le UI. Le graphe ajuste l'échelle de ses axes automatiquement pour couvrir les données affichées.

8.0.13 Tag Map

Exemple : `<Map />`. Le tag MAP contient une carte sur laquelle on affiche deux points : la position géographique du serveur et la position géographique de la fusée. L'échelle de la carte s'ajuste automatiquement. N'ayant pas accès à internet sur le site de la compétition, la carte doit être connue à l'avance. Les positions géographiques utilisées sont les suivantes. Une carte carrée d'une arrête de 5km doit être contenue en mémoire dans l'application.

1. Spareport America @32.9401475,-106.9193209,152m/data=!3m1!1e3?hl=en
2. Motel 6 @32.3417429,-106.7628682,53m/data=!3m1!1e3?hl=en
3. Convention Center @32.2799304,-106.7468314,67m/data=!3m1!1e3?hl=en
4. St-Pie de Guire @46.0035479,-72.7311097,422m/data=!3m1!1e3?hl=en

La position du serveur correspond aux coordonnées fournies. La position de la fusée est donnée par les messages CAN GPS{1, 2}_{LATITUDE, LONGITUDE, ALT_MSL}. La carte doit ajuster son échelle automatiquement pour englober la position du serveur et celle de la fusée.

8.0.14 Tag FindMe

Exemple : `<FindMe />`. Affiche une flèche en 3D qui pointe vers la fusée pour savoir où pointer les antennes et où aller chercher la fusée après le vol. L'orientation de l'appareil mobile est connue par le compas interne. Sa position est connue par le GPS interne. La position de la fusée provient des messages CAN GPS{1, 2}_{LATITUDE, LONGITUDE}.

8.0.15 Tag RadioStatus

Ignorer.

8.0.16 Tag CustomCANSender

Ignorer.

8.0.17 Tag DataDisplayer

Exemple : `<DataDisplayer>`. Ce widget affiche des données correspondant à un groupe de messages CAN. Il peut contenir un nombre arbitrairement grand de tags `CAN`, mais les applications habituelles ne dépassent pas 16. La disposition des données n'est pas imposée, mais doit être intuitive et claire.

8.0.18 Tag CAN

Le tag CAN est un des plus utilisés. Il sert à afficher une donnée de télémétrie, de l'étiqueter, de la formatter et de définir des seuls d'acceptabilité des valeurs, dont voici quelques exemples. La tabe 2 décrit les attributs et leurs fonctions.

```
<CAN name="1 - LAT" id="GPS1_LATITUDE" display="__DATA1__" minAcceptable="30"
maxAcceptable="35" chiffresSign="6" specificSource="MCD" serialNb="1"/>.
<CAN name="2 - ALT bar 2" id="BAR_PRESS2" minAcceptable="25000" maxAcceptable="95000"
customUpdate="pressToAlt" specificSource="MCD" serialNb="2" updateEach="10"/>.
<CAN name="2 - I_BAT" id="RPM_CURRENT" display="__DATA1__ A" minAcceptable="0.05"
maxAcceptable="7.0" chiffresSign="2" specificSource="MCD" serialNb="2"/>
```

8.1 Thème

L'application a deux choix de thèmes : light et dark. Le thème dark (figure 6) respecte la mode actuelle et permet de visualiser les données dans un environnement fermé intérieur. Le thème bright (figure 7), bien que très peu esthétique, est celui utilisé en compétition dans le désert parce que la lumière intense du soleil diminue la visibilité sur l'écran.

Le choix de thème sera persistant et une option permettra à l'usager de changer de thème sans devoir redémarrer.

8.2 Log

Pour chaque exécution, la tablette devra générer un fichier de log pour enregistrer les événements survenus. Tout événement anormal doit être enregistré. Le fichier doit contenir la date et l'heure dans son nom. Chaque nouvelle ligne commence par une date et une heure, suivie d'un tag INFO, DEBUG, WARNING ou ERROR.

9 Rappel de requis généraux importants

1. Toute information doit être accessible en deux actions ou moins dans le UI
2. Une compilation du projet ne doit pas être nécessaire après la modification des fichiers csv ou xml
3. Le code doit être bien documenté car il sera réutilisé.
4. Aucun crash ne sera toléré.

Table 2 – Attributs du tag CAN

Attribut	Fonction
name	Obligatoire. Représente l'étiquette et sera affiché dans le UI près de la donnée correspondante
display	Facultatif. Détermine si DATA1 ou DATA2 est affiché. Un symbole d'unité de mesure peut être ajouté et sera concaténé à l'affichage de la donnée. Par défaut, DATA1 est affiché
minAcceptable	Facultatif. Détermine le seuil inférieur d'acceptabilité de la donnée. Si le seuil est dépassé, la donnée doit être affichée en rouge. Autrement, la donnée doit être affichée en vert. Si aucune donnée n'a encore été reçue, la couleur du thème doit être adoptée.
maxAcceptable	Facultatif. Détermine le seuil supérieur d'acceptabilité de la donnée. Si le seuil est dépassé, la donnée doit être affichée en rouge. Autrement, la donnée doit être affichée en vert. Si aucune donnée n'a encore été reçue, la couleur du thème doit être adoptée.
chiffresSign	Facultatif. Détermine le nombre de chiffres significatifs à afficher. Par défaut, tous les chiffres significatifs sont affichés.
specificSource	Facultatif. Si la donnée entrante provient de la source spécifiée, le champ est mis à jour. Sinon, aucune action n'est prise. Par défaut, le champ est mis à jour.
serialNb	Facultatif. Si la donnée entrante provient d'une source avec le numéro de série spécifié, le champ est mis à jour. Sinon, aucune action n'est prise. Par défaut, le champ est mis à jour.
customUpdate	Facultatif. Appelle la fonction spécifiée par la valeur du champ. Cette fonction retourne l'affichage sur mesure.
updateEach	Facultatif. Restreint le rafraîchissement des données pour des raisons de performance. Par défaut, toutes les données sont affichées.

5. Le code doit être robuste aux inconsistances
   (a) CANSID inexistant
   (b) Mauvais type de données fourni dans DATATYPE1 et DATATYPE2
6. Ne pas afficher les données corrompues
7. Démontrer une équivalence d'information affichée entre la version PC et la version mobile
8. S'il y a lieu, le partage de nouvelles structures de données entre plusieurs langages (C/Python/Java principalement) devrait se faire de manière automatique avec des fichiers auto-générés.
9. Messages CAN à ignorer dans CANSid.csv :
   (a) Messages sous Emergency Event data
   (b) Messages commençant par MOTOR
   (c) Messages ayant en commentaire (To the rocket)
10. Compiler l'application mobile devrait se résumer à ouvrir Android Studio et d'appuyer sur "Build". Aucune autre action ne devrait être nécessaire.
11. La vitesse du port COM du ZedBoard devra être fixée à 921600 baud, 1 stop bit, none parity.
12. Un log csv de démonstration devra être construit pour démontrer l'entièreté des fonctionnalités du système. Le log ne devrait pas dépasser 5 minutes.

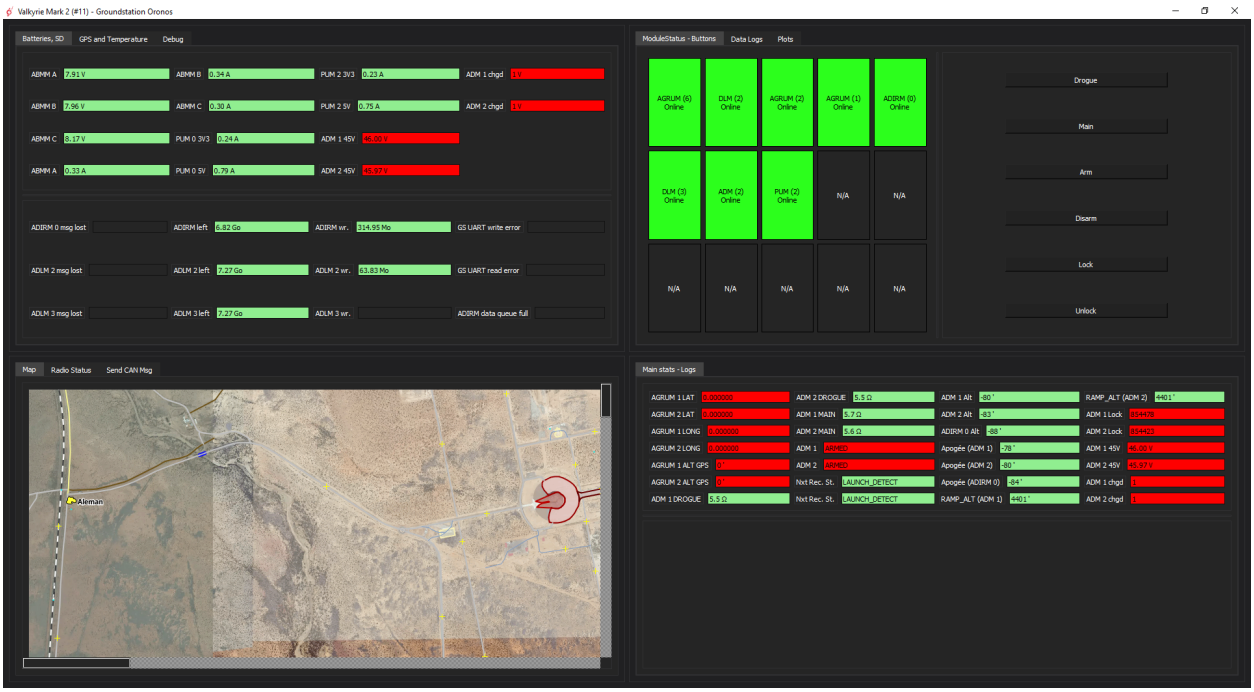

FIGURE 6 – UI PC, thème dark

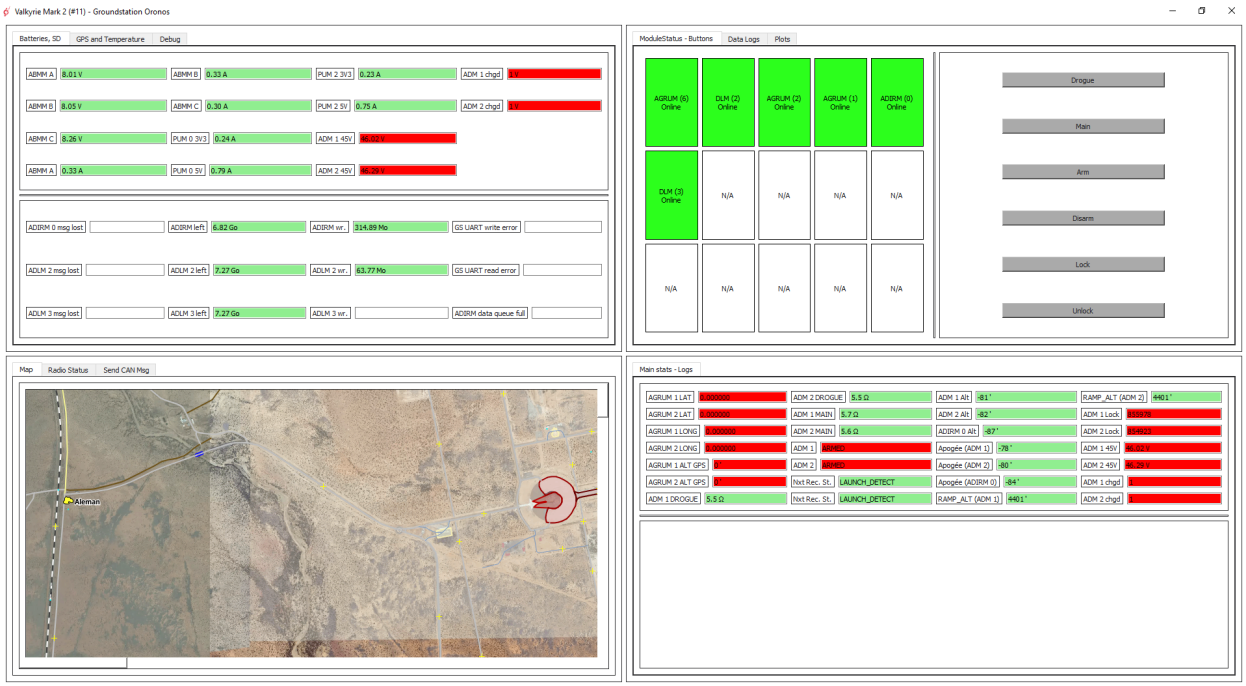

FIGURE 7 – UI PC, thème bright

10 Contenu des livrables

Deux phases de développement sont exigées pour ce projet, chacune aboutissant à un livrable fonctionnel selon les critères explicités dans cette section des exigences techniques.

10.1 Livrable 1

Pour le livrable 1, évalué le jeudi 29 mars 2018, Oronos s'attend aux fonctionnalités suivantes :

- Le transfert de tout le flot de données entre le serveur et la tablette avec tout le fonctionnement de l'affichage correctement disposé ;
- Les comptes, le transfert des fichiers XML ;
- Le serveur en ligne de commande seulement ;

10.2 Livrable 2

Au livrable 2, évalué le jeudi 12 avril 2018, vous devez terminer le projet, ce qui implique, par rapport au livrable 1 :

- Les thèmes sur la tablette et l'esthétisme bien en place ;
- Les fichiers optionnels PDF à télécharger ;
- L'interface graphique du serveur ;

Note : l'évaluation du premier livrable portera beaucoup sur l'atteinte des fonctionnalités prévues alors que le second portera beaucoup plus sur l'esthétisme, l'ergonomie et la facilité d'utilisation. Les grilles d'évaluation utilisées par le promoteur seront disponibles aux entrepreneurs avant les évaluations des livrables.

A Énumérations utiles

```
////////////////////////////////
////  Recovery arming states  ////
////////////////////////////////
typedef enum {
        RECOVERY_DISARMED = 0,
        RECOVERY_ARMED,
        N_ARMING_STATES
} ARMING_STATE;

////////////////////////////////
//// Main state machine states ////
////////////////////////////////

typedef enum {
        RECOVERY_INIT = 0,
        RECOVERY_WAIT_FOR_ARMING_SIGNAL,
        RECOVERY_WAIT_FOR_LAUNCH_DETECT,
        RECOVERY_WAIT_FOR_MACH_DELAY,
        RECOVERY_WAIT_FOR_APOGEE,
        RECOVERY_WAIT_FOR_MAIN_CHUTE_ALTITUDE,
        RECOVERY_SAFE_MODE,
        RECOVERY_N_SATES
} RECOVERY_STATE;

////////////////////////////////
//// Diagnostic failure modes ////
////////////////////////////////

typedef enum {
        BARO_INIT_FAIL = 0,
        BW_RES_DROGUE_INCORRECT,
        BW_RES_MAIN_INCORRECT,
        ALTITUDE_INCORRECT,
        VOTING_INVALID,
        RAMP_ALTITUDE_INCORRECT,
        BAD_CALLBACK,
        N_DIAGNOSTICS
} RECOVERY_DIAGNOSTIC_FAILURE;


////////////////////////////////
////      Module types      ////
////////////////////////////////
typedef enum {
        ADM = 0,                // Avionics Deployment Module
        ADIRM = 3,              // Air Data Intertial Reference Module
        ADLM = 4,               // Avionics Data Logging Module
        APUM = 6,               // Avionics Power Unit Module
        NUC = 7,                // Groundstation = serveur = pas un PCB
        GS = 7,                 // Groundstation = serveur = pas un PCB
```

```
        MCD = 15,                   // Moduel for Control and Deployment
        AGRUM = 16,             // Avionics GPS Radio Universal Module
        ADRMSAT = 17,           // Avionics Deployment Radio Module SATellite
        ATM_MASTER = 18,        // Avionics Thrust Module, master
        ATM_SLAVE = 19,         // Avionics Thrust Module, slave
        UNKNOWN_MODULE = 0x1E,
        ALL_MODULES = 0x1F   // Permet un broadcast CAN a tous les modules
} ModuleType;

////////////////////////////////
////      Module types        ////
////////////////////////////////
#define ALL_SERIAL_NBS 0xF   // Permet un broadcast CAN a tous les numéros de série
```