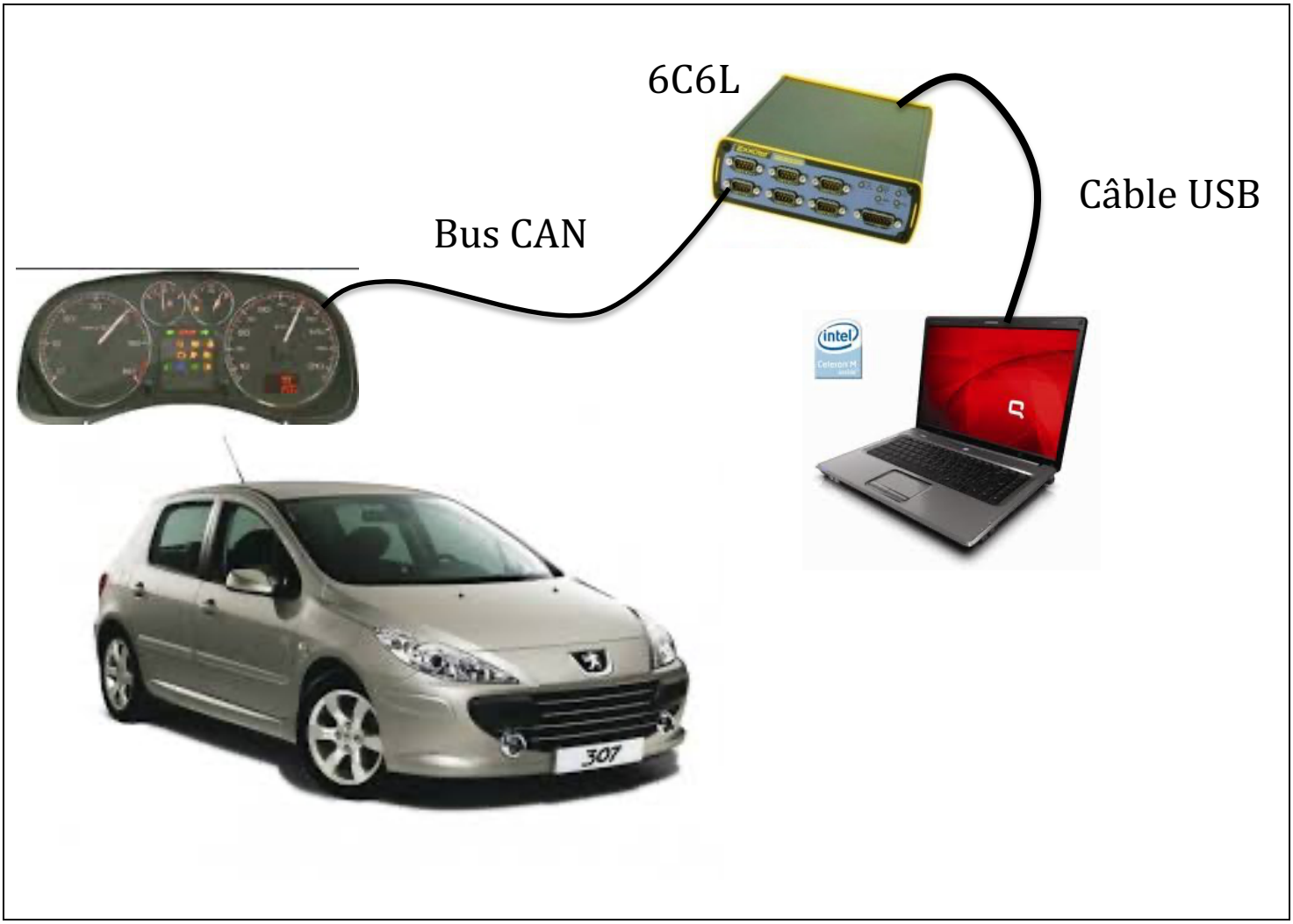
EdouardBRANLY Lycée des Métiers, du Design et des Technologies	**Système de diagnostic de combiné de 307** Matière : Informatique Industrielle Remis par : D. MASSICOT / F. CARON	Année : 2TS SN-IR Date : 29/09/2022 Temps : 40 h Feuille : 1/14

1 Présentation du système :

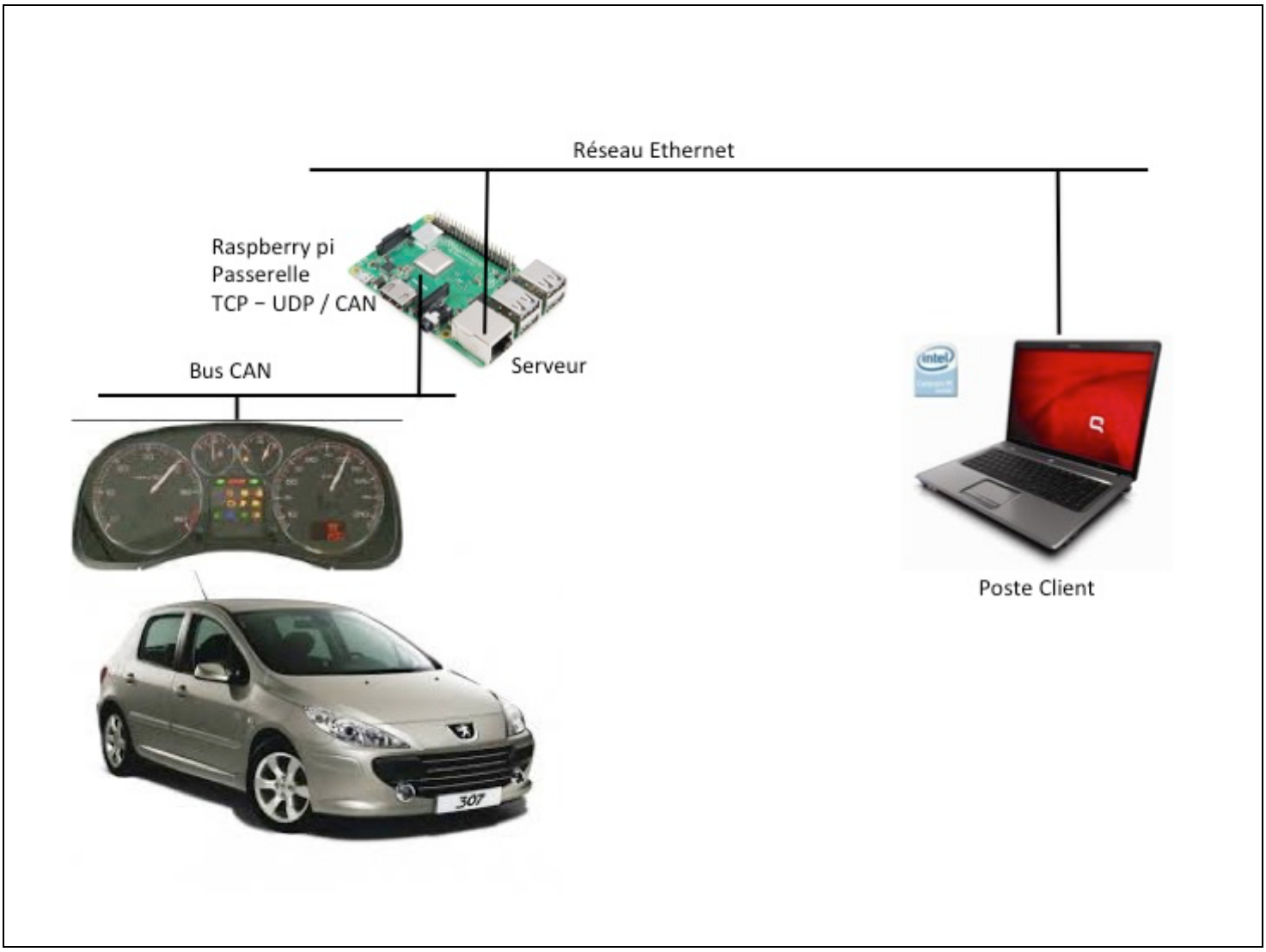
Avant d'être installé dans un véhicule, les combinés de tableau de bord sont préalablement testés. La procédure de test permet de vérifier le bon fonctionnement de tous les voyants ainsi que des micromoteurs qui pilotent le compteur de vitesse, le compte tour, l'indicateur de niveau d'essence, et de température de l'eau.

Au niveau du SAV deux procédures de test peuvent être misent en œuvre :
- La procédure locale : on va venir connecter un outil de diagnostic sur le tableau de bord et effectuer les tests comme vu précédemment.



Synoptique du système pour la procédure locale

- La procédure distante : si le dysfonctionnent n'est pas résolue en local (chez le concessionnaire) on connecte le tableau de bord au réseau interne, de façon à ce que les ingénieurs du centre technique puissent prendre la main et effectuer eux même les tests. Dans ces conditions il faut intégrer au système le concept de Client/Serveur.

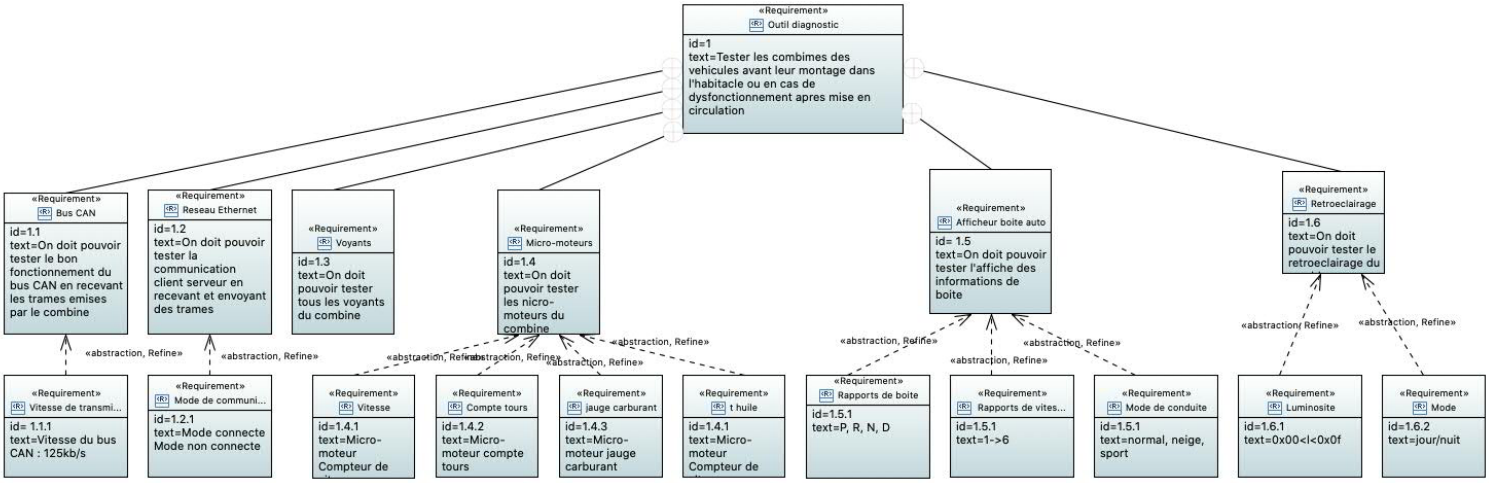


Synoptique du système pour la procédure distante

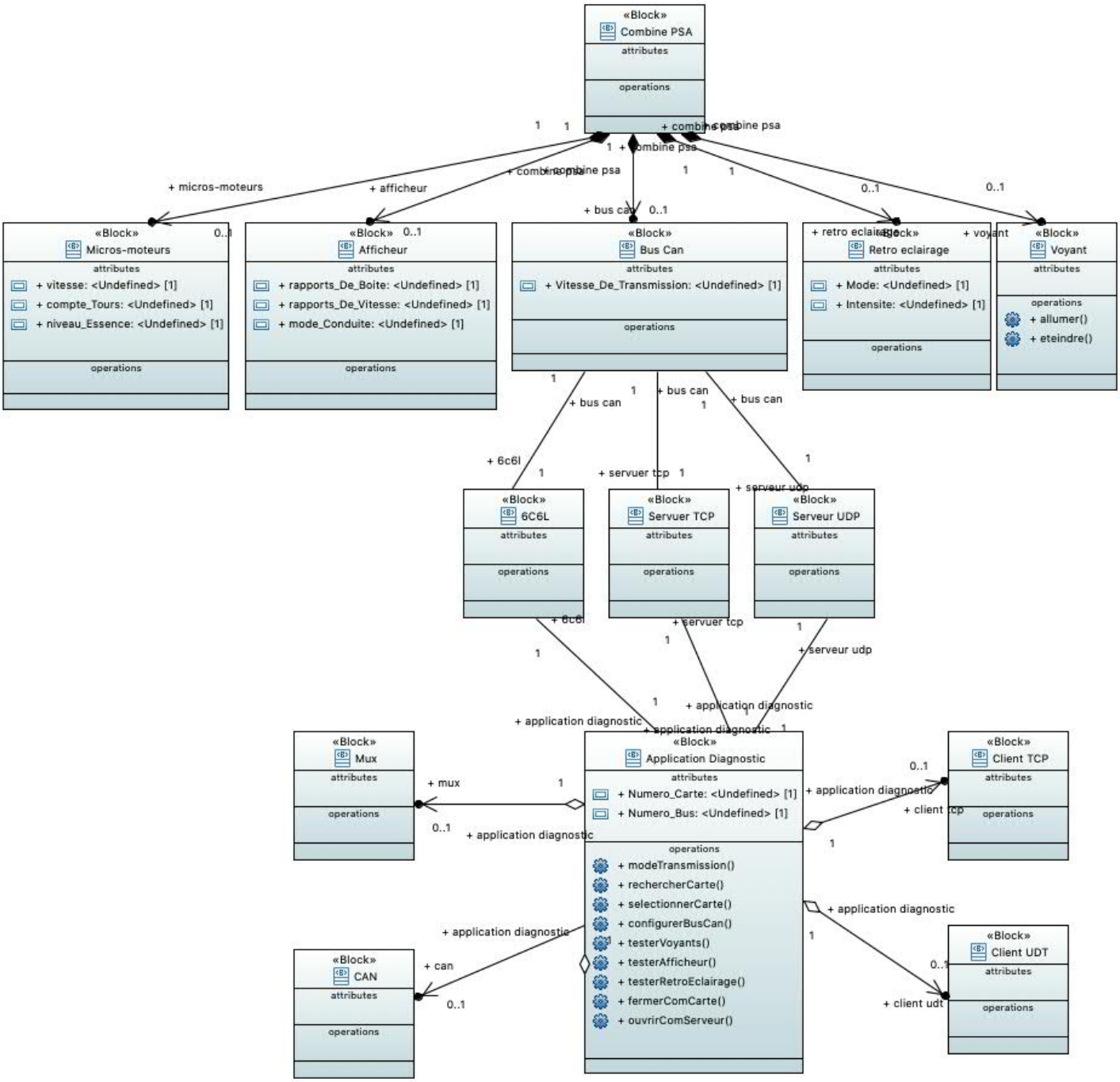
La remonté des données se fait au travers d'une application unique qui donne le choix aux utilisateur d'effectuer un diagnostic en mode local ou en mode distant.

1.1 Analyse SysML du système :

1.1.1 Diagramme d'exigences :

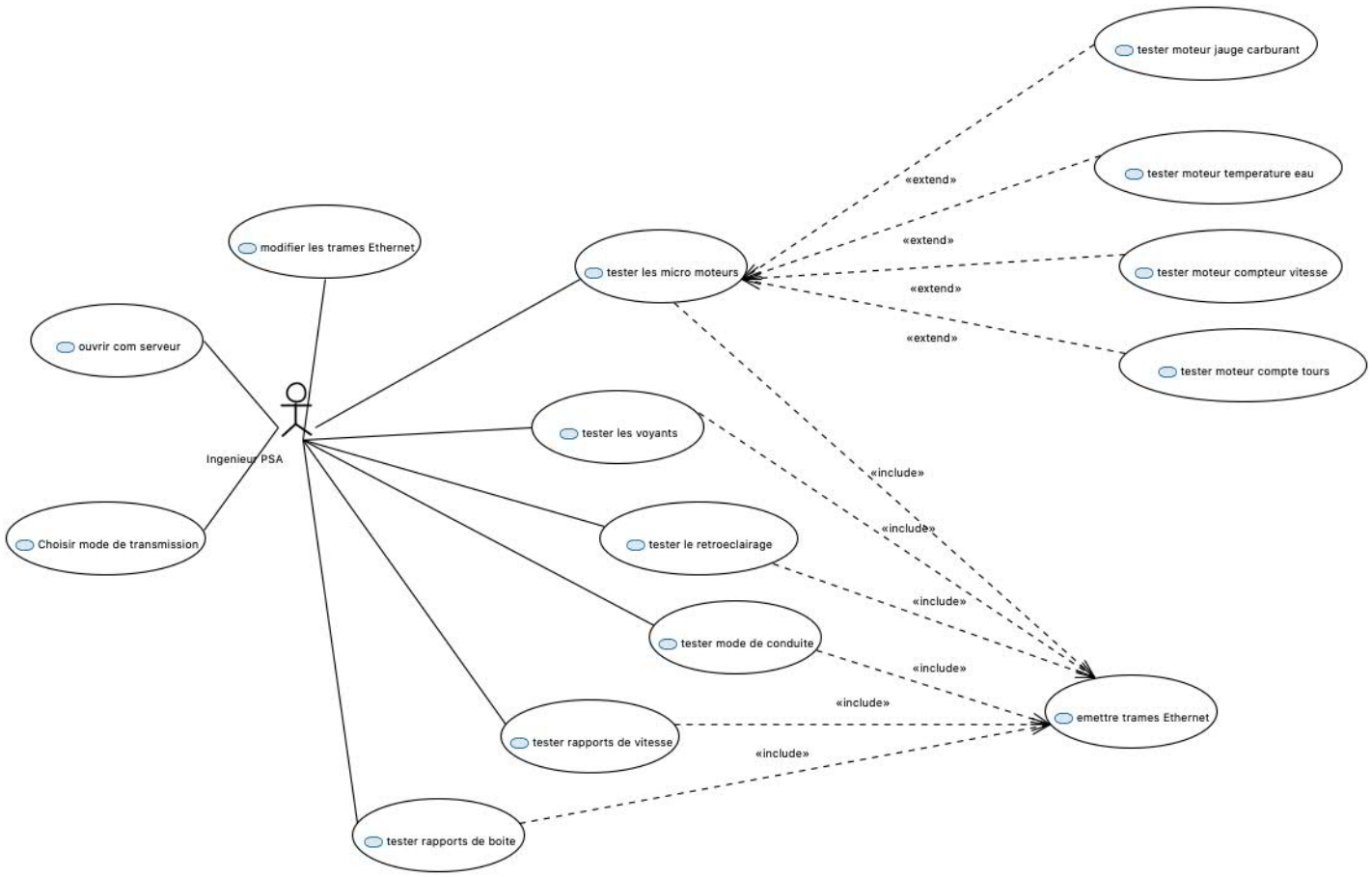


1.1.2 Diagramme de blocks

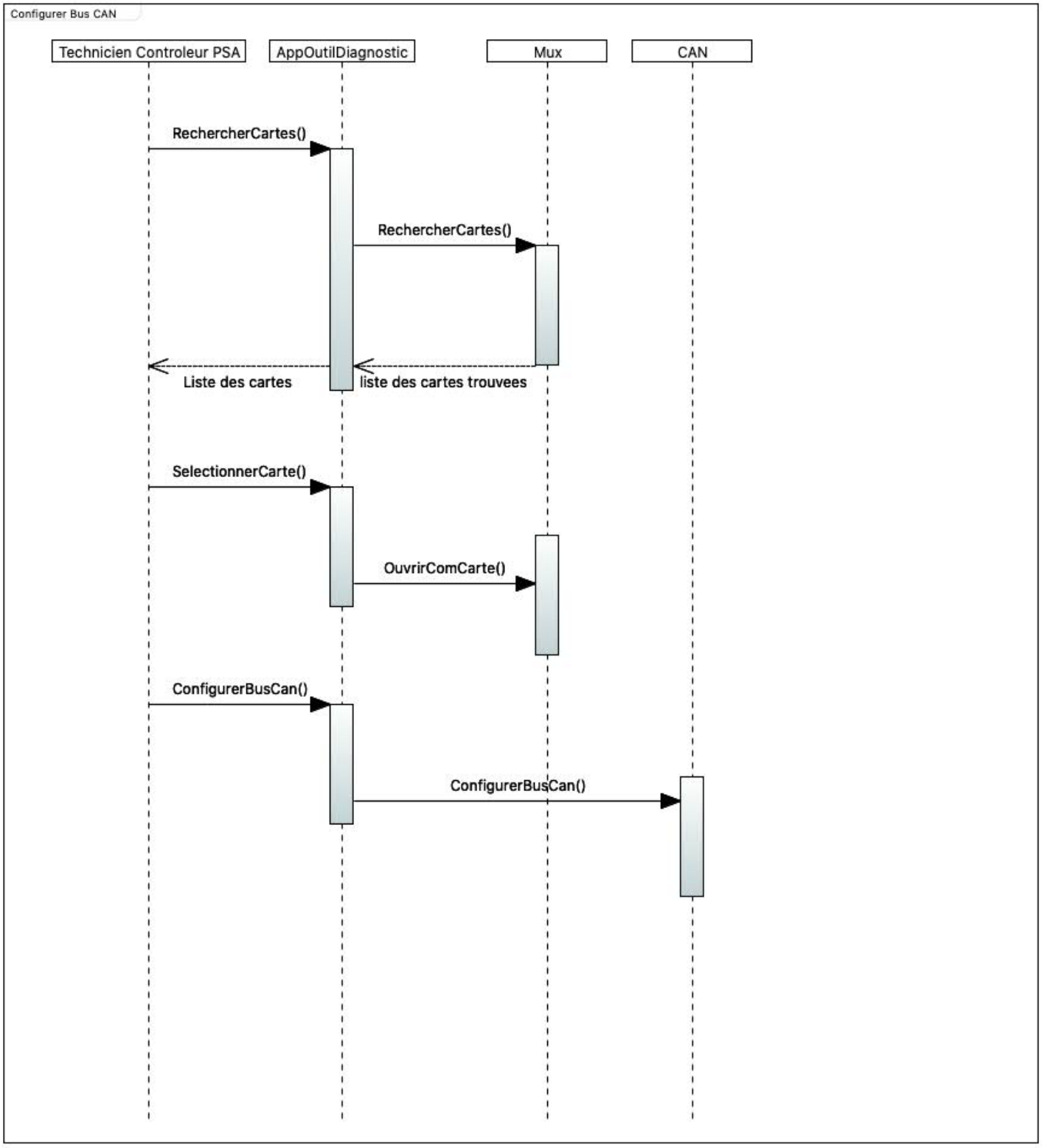


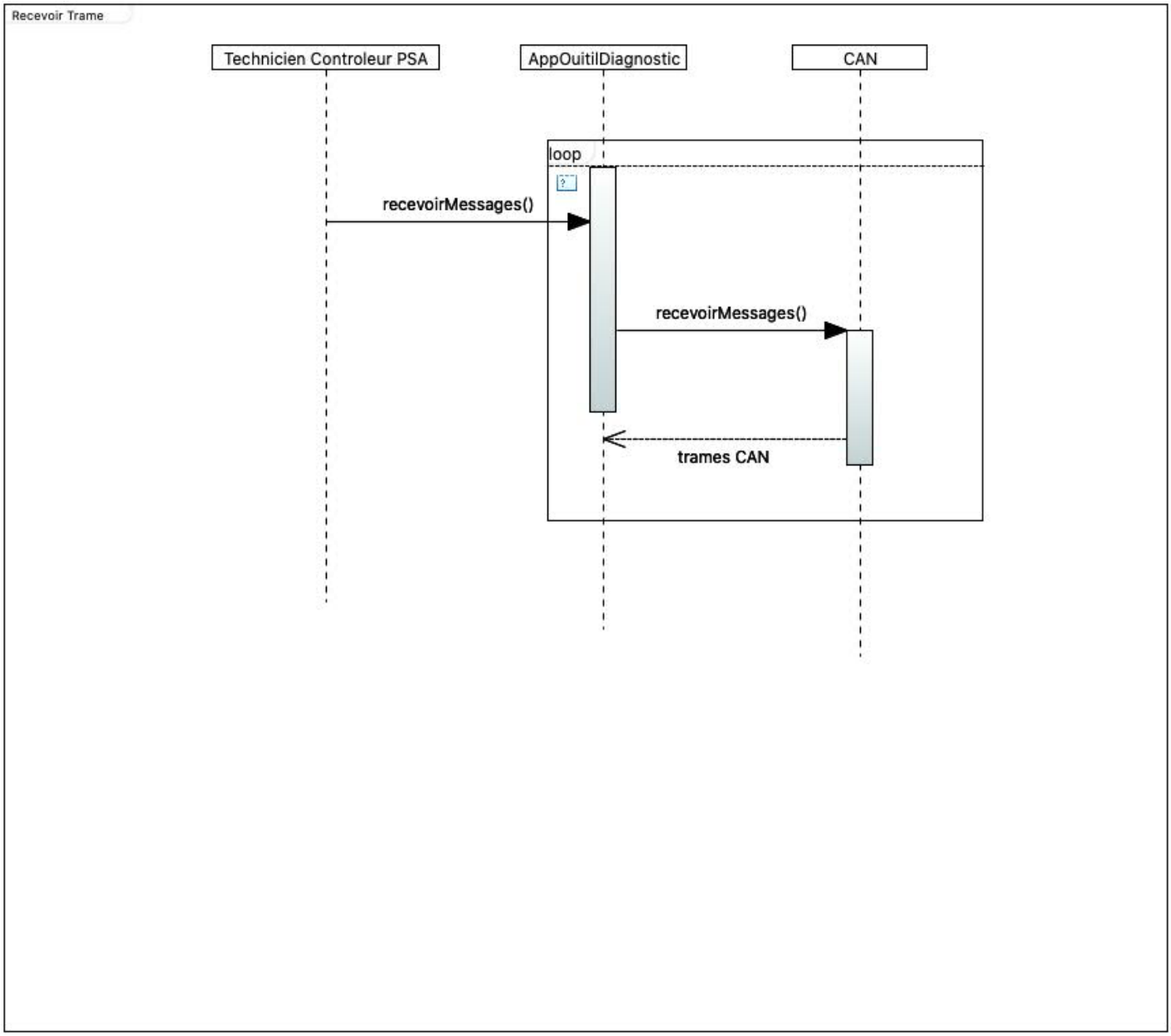
1.2 Analyse UML de l'application :

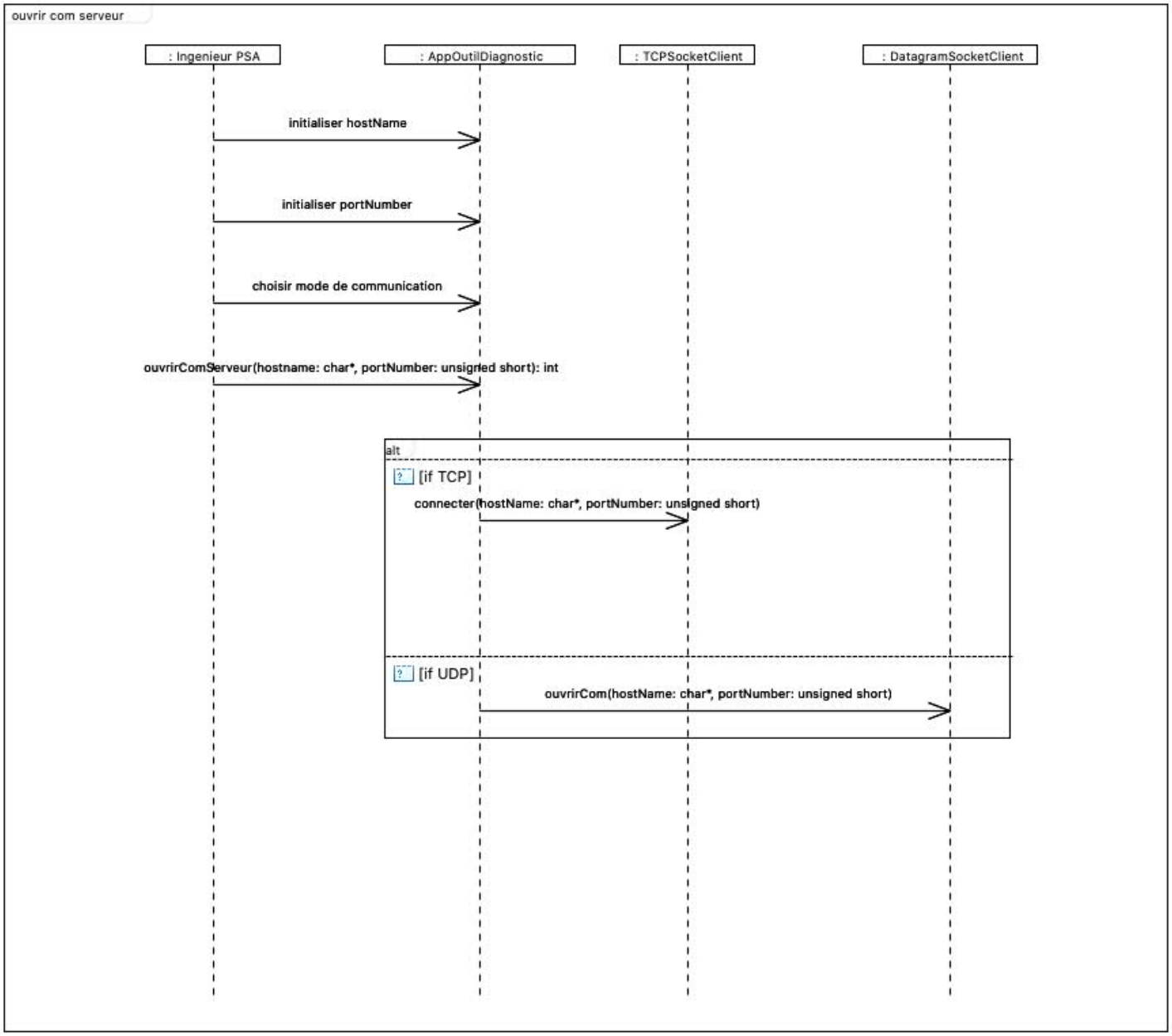
1.2.1 Diagramme des cas d'utilisation :



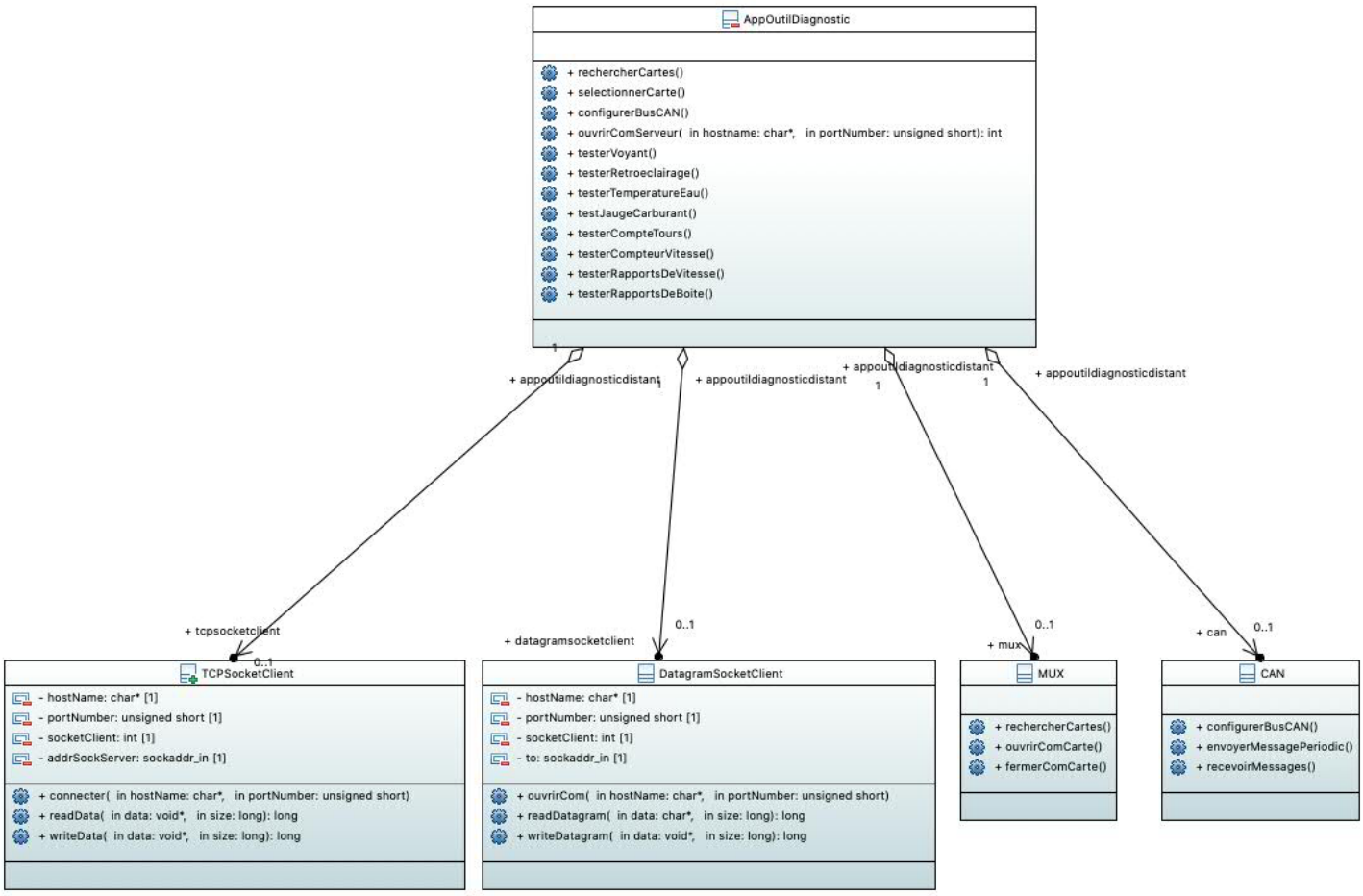
1.2.2 Diagramme de séquences :




Recevoir Trame
Technicien Controleur PSA
AppOuitilDiagnostic
CAN
loop
recevoirMessages()
recevoirMessages()
trames CAN


ouvrir com serveur
: Ingenieur PSA
: AppOutilDiagnostic
: TCPSocketClient
: DatagramSocketClient
initialiser hostName
initialiser portNumber
choisir mode de communication
ouvrirComServeur(hostname: char*, portNumber: unsigned short): int
alt
[if TCP]
connecter(hostName: char*, portNumber: unsigned short)
[if UDP]
ouvrirCom(hostName: char*, portNumber: unsigned short)

1.2.3 Diagramme de classes :



2 Expression du besoin mode de communication local :

L'acquisition et la transmission des trames se fera au travers des modules EXXOTest :

- Modules 6C6L : (modules présentés en TP)
- Mise en œuvre de la librairie associée : **`Muxdll`**.

Le logiciel Muxtrace permettra de tester les programmes réalisés.

2.1 Travail préparatoire :

Classe MUX :

- En respectant le diagramme de classes présenté ci-dessus, déclarez la classe Mux.
- En respectant le diagramme des classes présenté ci-dessus, définissez les méthodes de la classe Mux. Les méthodes **rechercherCartes()** et i**nitDriverCarte()** vont permettre d'initialiser les variables de la carte, et d'ouvrir le driver qui permettra d'accéder aux fonctions de la bibliothèque CAN.

Classe CAN :

- En respectant le diagramme des classes présenté ci-dessus, déclarez la classe CAN.
- En respectant le diagramme des classes présenté ci-dessus, définissez la méthode **configurerBus()** qui va permettre de configurer le protocole CAN, ainsi que la vitesse de transfert du bus. Dans un premier temps la vitesse de transmission sera : High Speed à 500 kb/s.
- Définissez la méthode la méthode **envoieMsgPeriodique()**, qui doit permettre de saisir, filtrer et paramétrer la trame à envoyer au format CAN.
- Définissez la méthode **recevoirMsg()** qui permet d'afficher les données des trames CAN reçues en hexadécimale.
- Définissez la méthode **fermerComCarte()** qui permet de libérer le driver de la carte choisie.

2.2 Etapes du développement :

Le développement de votre application se fera au travers de test unitaires.

Test unitaire de la classe Mux :

Ce test unitaire permet de tester le nombre de carte présent sur votre PC et d'initialiser la carte que vous aurez choisi parmi celles connectées a votre machine..
La fonction membre **rechercherCartes()** utilise plusieurs méthodes de la librairie fournie avec votre carte, voir description en fin de document ou dans les fichiers **DLL_MUX_CAN.pdf**

Le résultat attendu est une bonne configuration de la carte présente visualisable par une LED clignotante sur la carte, ainsi que par les messages STATUS_OK renvoyés par les fonctions appelées.

Test unitaire de la classe CAN :

La configuration du bus CAN reste classique, avec une vitesse de transmission de 500 Kbits.
Pour des raisons de confidentialité, les identifiants, et les données à transmettre vous seront données lors du développement du test unitaire.

Le résultat attendu est une bonne configuration du protocole et de la vitesse du bus, l'émission d'une trame sur le bus CAN visualisable sur un autre ordinateur présent sur le bus CAN, disposant du logiciel Muxtrace, une

bonne réception d'une trame sur le bus CAN, visualisable dans un memo, dans lequel tous les champs de la trame doivent apparaître.

2.3 Test d'intégration : mise en œuvre du système de test du combiné :

Le test de bon fonctionnement du combiné de tableau de bord se fait avant son installation dans l'habitacle du véhicule.
Modifiez votre programme de façon à configurer le bus CAN en Low Speed à 125 kb/s.
Simuler le démarrage du véhicule au travers de boutons ou de « slider » à état.
Tous les voyants du combiné seront testés par l'appui sur deux boutons différents.
- un bouton pour les allumer
- un bouton pour les éteindre

Les tests des micros moteurs du compte tour, du compteur de vitesse, de la jauge d'essence et de la température de l'eau se feront au travers de « slider »
Le test des rapports de vitesses de boite automatique et des modes de conduite (sport, hivers, normal) se fera au travers de « sliders » à état.
Le test du rétro-éclairage du combiné se fera au travers d'un « slider »
Envoyez les messages qui permettent de tester le combiné.

Les trames appropriées vous seront communiquées en temps voulu.

2.4 Pour aller plus loin :

Pour enrichir votre interface graphique, vous pouvez utiliser du code QML. (compteur, jauges…).

3 Expression du besoin mode de communication distant :

3.1 Travail préparatoire :

3.1.1 Client / serveur en mode connecté :

Client TCP :

- En respectant le diagramme de classes présenté ci-dessus déclarez la classe TCPSocketClient.
- Définissez les différentes méthodes de la classe. Le constructeur devra effectuer toutes les initialisations nécessaires.

Serveur TCP :

- En respectant le diagramme de classes présenté ci-dessus déclarez la classe TCPSocketServer.
- Définissez les différentes méthodes de la classe. Le constructeur devra effectuer les différentes initialisations nécessaires, et finira par se mettre en attente d'une connexion.

3.2 Etapes du développement :

Le développement de votre application se fera au travers de test unitaires.

Test unitaire de la classe TCPSocketClient :

Ce test unitaire permet de tester la connexion à un serveur existant, l'envoie d'une requête et la réception de la réponse.

Test unitaire de la classe TCPSocketServer :

Ce test unitaire doit permettre de configurer le serveur et de le mettre en attente de connexion. Votre client devra pouvoir se connecter dessus, envoyer une requête et recevoir une réponse

3.2.1 Client / serveur en mode non connecté :

Client UDP :

- En respectant le diagramme de classes présenté ci-dessus déclarez la classe DatagramSocketClient.
- Définissez les différentes méthodes de la classe. Le constructeur devra effectuer toutes les initialisations nécessaires.

Serveur UDP :

- En respectant le diagramme de classes présenté ci-dessus déclarez la classe DatagramSocketServer.
- Définissez les différentes méthodes de la classe. Le constructeur devra effectuer les différentes initialisations nécessaires..

3.3 Etapes du développement :

Test unitaire de la classe DatagramSocketServer :

Ce test unitaire permet de tester, l'envoie d'une requête et la réception de la réponse entre votre client et un serveur existant.

Test unitaire de la classe DatagramSocketServer :

Ce test unitaire doit permettre de configurer le serveur. Votre client devra pouvoir lui envoyer une requête et recevoir une réponse

3.4 Test d'intégration : mise en œuvre du système de test du combiné :

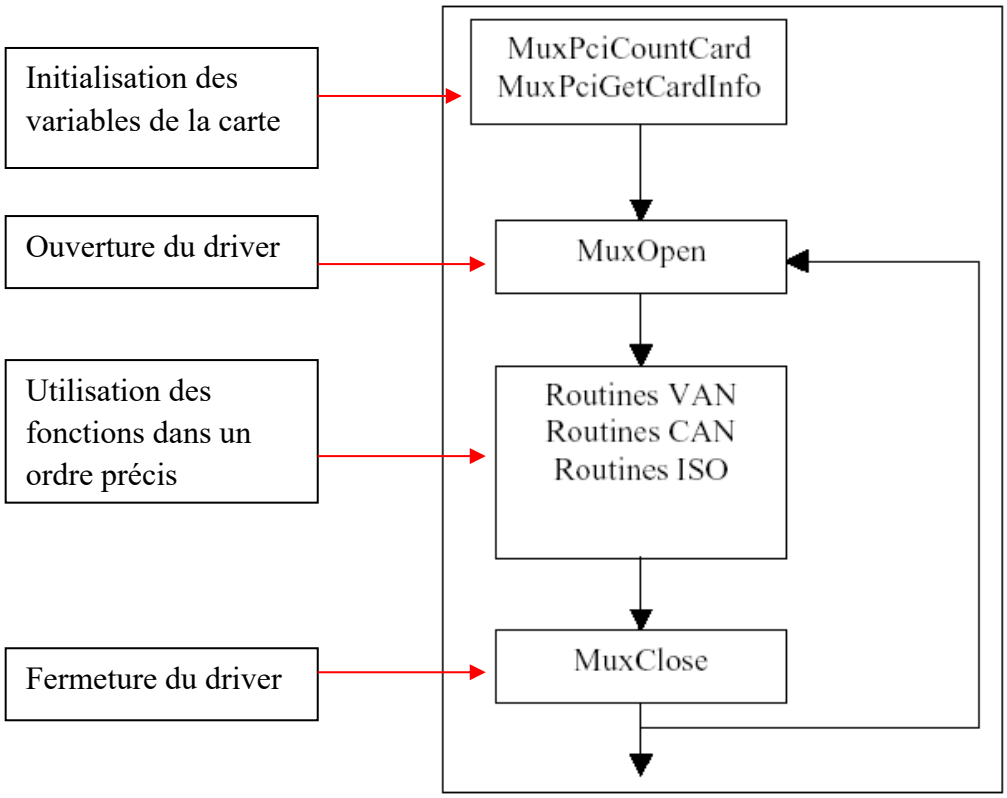
Intégrez les classes TCPSocketClient et DatagramSocketClient à votre application de diagnostic et ainsi permettre à un utilisateur de choisir dans quel mode il veut fonctionner (local ou distant). En distant il doit avoir le choix entre deux modes de communication : soit en mode connecté soit en mode non connecté.

4 Exploitation pédagogique :

Compétences terminales susceptibles d'être abordées et évaluées	
C4.1	Câbler et/ou intégrer un matériel
C4.2	Adapter et/ou configurer un matériel
C4.3	Adapter et/ou configurer une structure logicielle
C4.4	Développer un module logiciel
C4.5	Tester et valider un module logiciel et matériel
C4.6	Intégrer un module logiciel
C2.1	Maintenir les informations
C4.7	Documenter une réalisation matérielle et/ou logicielle
C3.1	Analyser un cahier des charges
C3.5	Contribuer à la définition des éléments de recette au regard des contraintes du cahier des charges

ANNEXE 1
Utilisation de la DLL MUXDLL

Notre partenaire nous ont fourni une librairie, laquelle contient toutes les fonctions qui servent à utiliser la carte PCI et USB ainsi que toute la gamme de carte multiplexage d'EXXOtest. Toute la documentation de ces fonctions se trouve en annexe, ou en ligne sur le serveur. Pour utiliser ces fonctions il faut respecter un ordre d'appel :



Afin de pouvoir utiliser cette DLL (Dynamic Library Link) il y a juste à copier le muxdll.dll dans le répertoire SYTEM du système d'exploitation (si vous laissez la dll dans votre répertoire de projet c'est cette dll qui sera utilisée lors de l'appel de la dll) et d'inclure muxdll.lib au projet dans l'outil de développement. inclure le fichier **`refmux.h`** pour pouvoir utiliser les fonctions de la bibliothèque.