BTS SN

Période	Sem 1	Sem2	Sem3	Sem4
Volume horaire	Cours/TD		TP	
	4		12	

Première approche de la POO

Compteur graphique automobile

Indicateur temporel (hors rédaction du compte-rendu) :

questions	1h	2h	3h	4h	5h	6h	7h	8h	9h	10h	11h	12h
1 .. 8												
9 .. 14												
15 .. 22												
23 .. 26												
27 .. 32												

Documents à rendre :

Compte rendu de TP.

Ressources :

Diagrammes MagicDraw à compléter

Compteur voiture étudiants.mdzip

Utilitaires pour Windows

Logiciel d'envoi de la trame série au terminal d'affichage : SimuEnvoiTrameSerie.exe

Exécutable du terminal d'affichage : CombineGraphique.exe et fichiers Labview.

Terminal d'envoi et de réception série : Terminal.exe

Librairies contenant les classes à utiliser

Dans le répertoire « CodeSource » et à importer dans le projet.

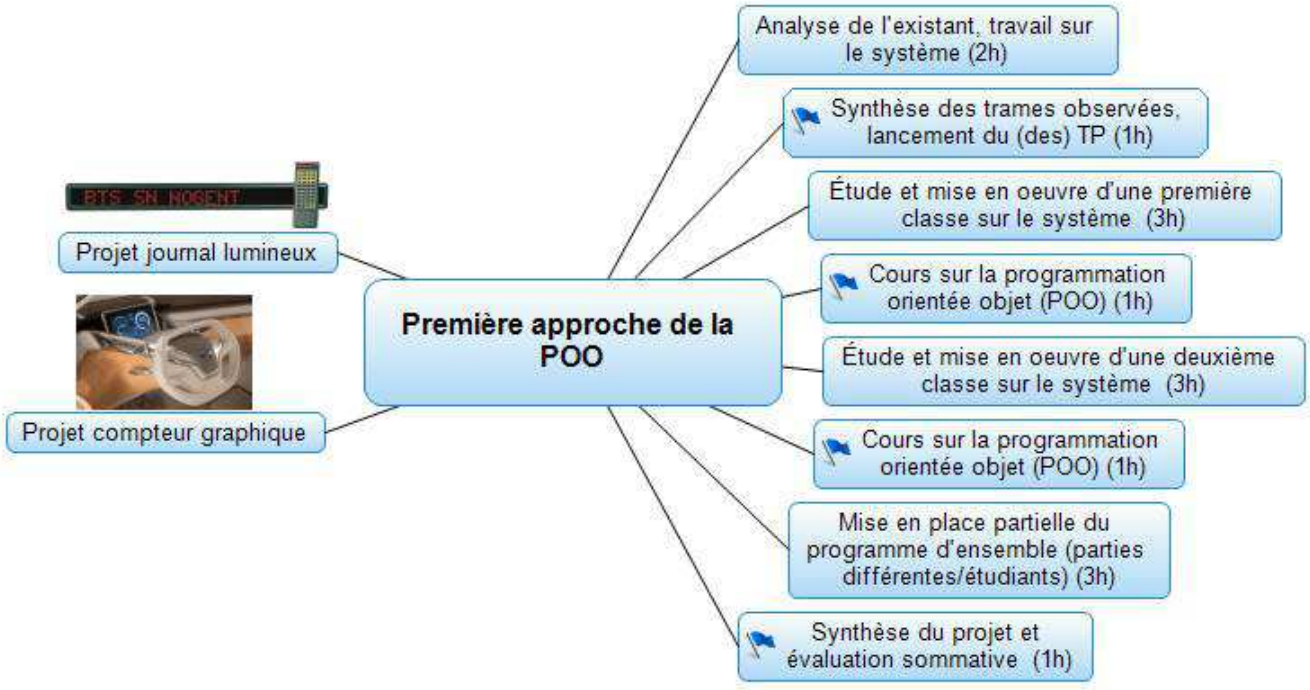
Schéma et fichiers de fabrication de la carte d'extension MBED

ExtensionMBED.zip

Activité 1 – Présentation du cahier des charges et analyse

Présentation du module

Nous allons dans ce module découvrir la programmation orientée objet (POO). La partie pratique pourra être réalisée autour du système « Journal lumineux » ou du système « Afficheur graphique », présenté ici. Les activités auront les mêmes objectifs. Le déroulé du module est le suivant :



Présentation du cahier des charges du projet compteur graphique

Le but de ce projet et de remplacer le compteur à aiguilles de la maquette Exxotest par un affichage graphique. Nous nous concentrerons ici uniquement sur la mise en œuvre de la passerelle, la partie affichage (réalisé ici sur un ordinateur) sera fournie.

On se limitera à l'affichage des informations des feux et des clignotants, du rapport de vitesse enclenché, de la vitesse du véhicule et du régime du moteur.

Support technique utilisé

Nous utiliserons dans ce projet une des maquettes pédagogiques de la marque Exxotest, à savoir les maquettes MT-CAN-LIN-BSI ou DE-1134-F877. Pour pouvoir observer les trames Can qui vont nous intéresser nous utiliserons le logiciel MUX-Trace associé à un boîtier USB de cette même société.

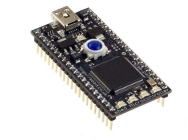
MT-CAN-LIN-BSI

DE-1134-F877

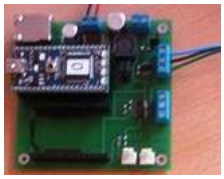
MUX-Trace et boîtier USB

Les informations envoyées au combiné (compteur à aiguille), sont véhiculées sur un bus CAN Low Speed (CAN_LS ou fault tolerant) cadencé à 125 kbits/s. Pour récupérer les informations sur le bus CAN et générer une trame USB (série virtuelle) vers le PC d'affichage, nous utiliserons une carte de prototypage MBED NXP LPC1768 qui intègre de nombreux périphériques et en particulier un contrôleur CAN. Ce microcontrôleur possède un cœur ARM® Cortex™-M3 et se programme grâce à une chaîne de développement gratuite en ligne par programmation C/C++.

Il faut ajouter à cette carte un driver CAN Low Speed (adaptation des niveaux de tension) pour l'interfacer à la maquette Exxotest. La liaison USB utilisera le connecteur mini USB de la carte MBED.



MBED
NXP LPC1768



Carte support
avec driver
CAN

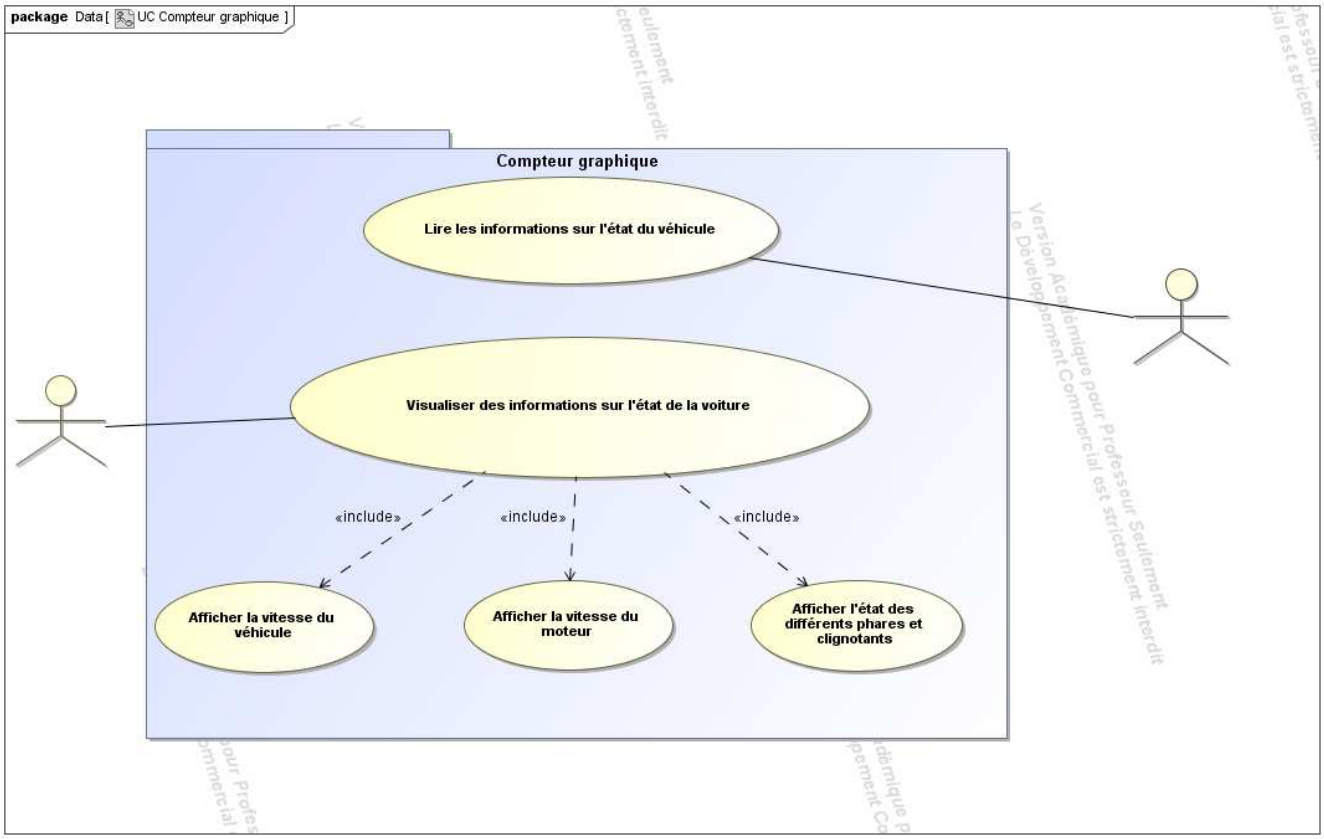
Analyse du cahier des charges

L'objectif de cette partie est, à partir du cahier des charges précédent, de mettre en place les diagrammes UML de cas d'utilisation et de déploiement de notre système.

Le diagramme de cas d'utilisation

Le diagramme de cas d'utilisation présente la liste exhaustive des acteurs interagissant avec notre système. Dans notre cas, notre système est l'ensemble passerelle CAN et son application demandée ainsi que le terminal d'affichage. La seule personne interagissant avec ce système sera l'utilisateur du véhicule. Par contre, il devra communiquer avec le véhicule (BSI : boîtier de servitude et d'intelligence) pour récupérer les informations sur le bus CAN.

1. Complétez le diagramme de cas d'utilisation suivant en ajoutant le nom des acteurs. Complétez-le avec MagicDraw.

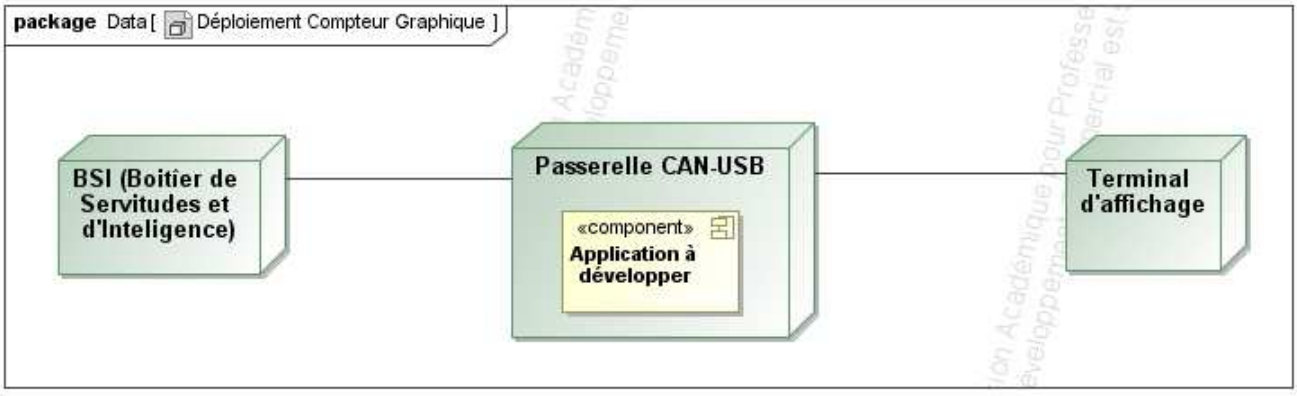


Le diagramme de déploiement

Le logiciel informatique à mettre en place doit :

- communiquer avec le véhicule par l'intermédiaire du bus CAN Low Speed ;
- communiquer avec le terminal de visualisation via une liaison série virtuelle sur bus USB.

2. Complétez maintenant le diagramme de déploiement en précisant les différentes connexions. Saisissez-le avec MagicDraw.



Activité 1A - Découverte de l'existant : les trames CAN du véhicule

Les trames qui nous intéressent sont envoyées via un bus CAN LS par le Boîtier de Servitude Intelligent (BSI) réel (sur la maquette MT) ou simulé (sur la maquette DE).

Détail des informations à récupérer sur le bus CAN

✔ Trame d'état des voyants des feux :

- Identifiant **0x128**
- 8 octets de données (DLC=8)
- 5ème octet : état des feux de signalisation

7	6	5	4	3	2	1	0
Position	Croise.	Route	AB Av.	AB Ar.	Clign. D	Clign. G	

- 6ème octet : mise en marche ou arrêt du combiné

  Marche ⇔ 0x80 ; Arrêt ⇔ 0x00.
- 7ème octet : vitesse engagée

  Point mort (P) ⇔ 0x00 ; Marche arrière (R) ⇔ 0x10 ; 1 ⇔ 0x90 ; 2 ⇔ 0x80 ;

  3 ⇔ 0x70 ; 4 ⇔ 0x60 ; 5 ⇔ 0x50 ; 6 ⇔ 0x40.

✔ Trame de vitesse de déplacement et de rotation du moteur :

- Identifiant **0x0B6**
- 8 octets de données (DLC=8)
- 1er et 2ème octet : régime moteur

  Point fort sur l'octet 1, faible sur l'octet 2. Régime moteur ⇔ nombre transmis/10, exprimé en tr/min.
- 3ème et 4ème octet : vitesse du véhicule

  Point fort sur l'octet 3, faible sur l'octet 4. Vitesse du véhicule ⇔ nombre transmis/100, exprimé en km/h.

Comme vous l'avez compris seulement certains octets de deux trames particulières nous intéresses. Pour observer l'ensemble des trames présentent sur le bus CAN et vérifier les données intéressantes, nous allons utiliser les outils Exxotest, à savoir le logiciel Muxtrace et le boîtier USB associé.

Installation et paramétrage du logiciel Muxtrace et du boîtier USB

3. Connectez le boîtier USB à votre poste et relier le avec les nappes adaptées au bus

CAN LS (maquette DE) ou CAN LS / Confort (maquette MT). Reliez CAN H, CAN L et la masse !

4. Installez le logiciel *Muxtrace* (si ce n'est pas déjà fait). Lancez le logiciel et configurez un nouveau projet (détection automatique de la vitesse du bus, mode espion).

Analyse des trames CAN

Nous pouvons maintenant espionner les données qui sont transmises sur le bus CAN LS du véhicule.

5. Démarrez l'analyse de trame, combien de trames différentes sont présentes ? Vérifiez la présence des trames 0x128 et 0x0B6 et la cohérence des informations données à la page précédente.

Une fois cette première partie du mini-projet terminée, vous connaissez les données à lire et à récupérer sur le bus CAN du véhicule.

Activité 1B - Découverte de l'existant : analyse de la trame série envoyée au terminal d'affichage

Objectifs

Dans cette activité de découverte du matériel, nous utiliserons deux logiciels fournis, l'un pour gérer l'affichage du compteur graphique, l'autre pour envoyer les trames série. Il s'agit ici d'observer le contenu de la trame série qui est envoyée au terminal d'affichage.

Détail de la trame série

La trame série envoyée sur le PC (ou la tablette) pour affichage a le format suivant :

- ✔ 1 octet de départ (STX ⇔ 0x02)
- ✔ 7 octets de données
- ✔ 1 octet de fin (ETX ⇔ 0x03)
- ✔ Périodicité d'envoi 0,2 seconde.
- ✔ Le détail des 9 octets est le suivant :

	Vitesse véhicule		Régime moteur		Rapport de boite	Cde. feux	Etat véhicule	
0x02	P. fort	P. faible	P. fort	P. faible				**0x03**

Installation de l'ensemble des éléments

Il nous faut ici deux ordinateurs avec port série (ou adaptateur USB/série) et un câble série croisé. Pour observer les données reçus sur le second ordinateur on pourra utiliser le petit logiciel « Terminal ».

6. Installez le câble série entre les deux ordinateurs. Sur le premier ordinateur exécuter le logiciel « EnvoiTrameExxotest», envoi sur le port COM branché ! Sur le second exécuter le logiciel « Terminal », choisir le bon port COM également.

Analyse des trames CAN

Nous pouvons maintenant espionner les données qui sont transmises sur la liaison série vers le terminal d'affichage.

7. Démarrez l'analyse de trame, vérifiez la cohérence des informations envoyées.
8. Sur le second PC, installez le logiciel d'affichage du compteur graphique « Tableau de bord 307 ». Configurez le port COM utilisé et lancez l'application. Vérifiez le bon fonctionnement du terminal d'affichage.

Activité 2 – Envoyer des données série

Dans cette partie, vous allez développer une application sur le micro contrôleur MBED qui devra envoyer des données via USB en série virtuelle. Attention pour que la liaison USB du Mbed soient vue comme un port série virtuel, il faut, sous Windows, installer le driver. Un lien de téléchargement est disponible (Handbook → serial → Windows serial configuration → Download latest driver).

Deux étapes seront nécessaires :

- prise en main des outils, analyse de la classe « Serie » fournie ;
- écrire le programme principal.

Prise en main des outils

La prise en main des outils est décrite dans l'annexe 1 de ce document.

9. Veuillez suivre les instructions et mettre en œuvre ce programme de test.

Analyse de la classe Serie

La classe Serie vous est fournie grâce aux fichiers Serie.h et Serie.cpp.

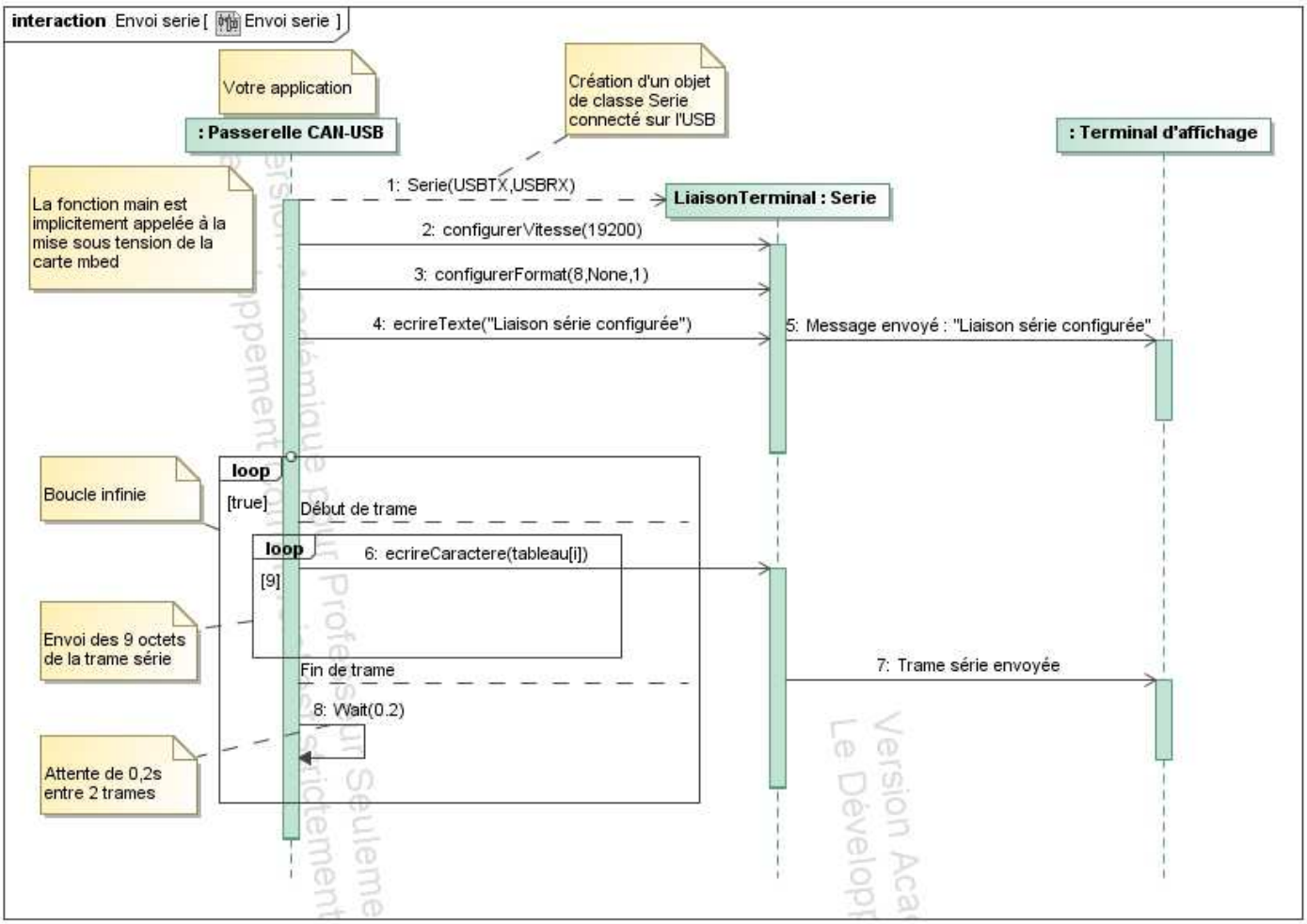
10. Listez les différentes méthodes de cette classe et donnez leurs attributs.
11. Complétez le diagramme de la classe Serie.

Écriture du programme principal

Notre premier programme utilisera uniquement la classe « Serie ». Il faudra :

- Créer un objet de classe « Serie » relié sur la liaison USB-série virtuelle (USBTX, USBRX).
- Appeler les méthodes configurerVitesse() et configurerFormat() pour indiquer les caractéristiques de notre liaison série.
- Envoyer un message de départ grâce à la méthode ecrireTexte().
- Envoyer toutes les 200ms le tableau des 9 octets de la trame en utilisant une boucle « for » et la méthode ecrireCaractere().

Ces 4 objectifs sont représentés dans le diagramme de séquence suivant :



Puisque vous allez utiliser la classe Serie, vous devez inclure le fichier Serie.h qui vous a été

fourni. Cette inclusion ce fait comme suit :

```
#include "Serie.h"
```

Pour déclarer un tableau de 9 caractères et l'initialiser, on utilisera la syntaxe suivante :

```
char tableau[9]= {0x02,0,0,0,0,0,0,0,0x03};
```

12. Ajoutez cette inclusion et cette déclaration dans le fichier *main.cpp*.
13. À partir du diagramme de séquence précédent, modifiez le fichier *main.cpp* de votre projet.
14. Compilez votre projet, implantez le dans la carte Mbed et testez-le.

Activité 3 – Récupérer des données CAN

Dans cette partie, vous allez récupérer les données sur le bus CAN du véhicule.

Trois étapes seront nécessaires :

- analyse des classes « BusCAN » et « MessageCAN » fournies ;
- écrire le programme principal ;
- amélioration du programme principal.

Analyse des classes BusCAN et MessageCAN

La classe MessageCAN vous est fournie grâce aux fichiers MessageCAN.h et MessageCAN.cpp.

15. Listez les différentes méthodes de cette classe et donnez leurs attributs.

La classe BusCAN vous est fournie grâce aux fichiers BusCAN.h et BusCAN.cpp.

16. Listez les différentes méthodes de cette classe et donnez leurs attributs.
17. Complétez le diagramme des classes BusCAN et MessageCAN.

Écriture du programme principal

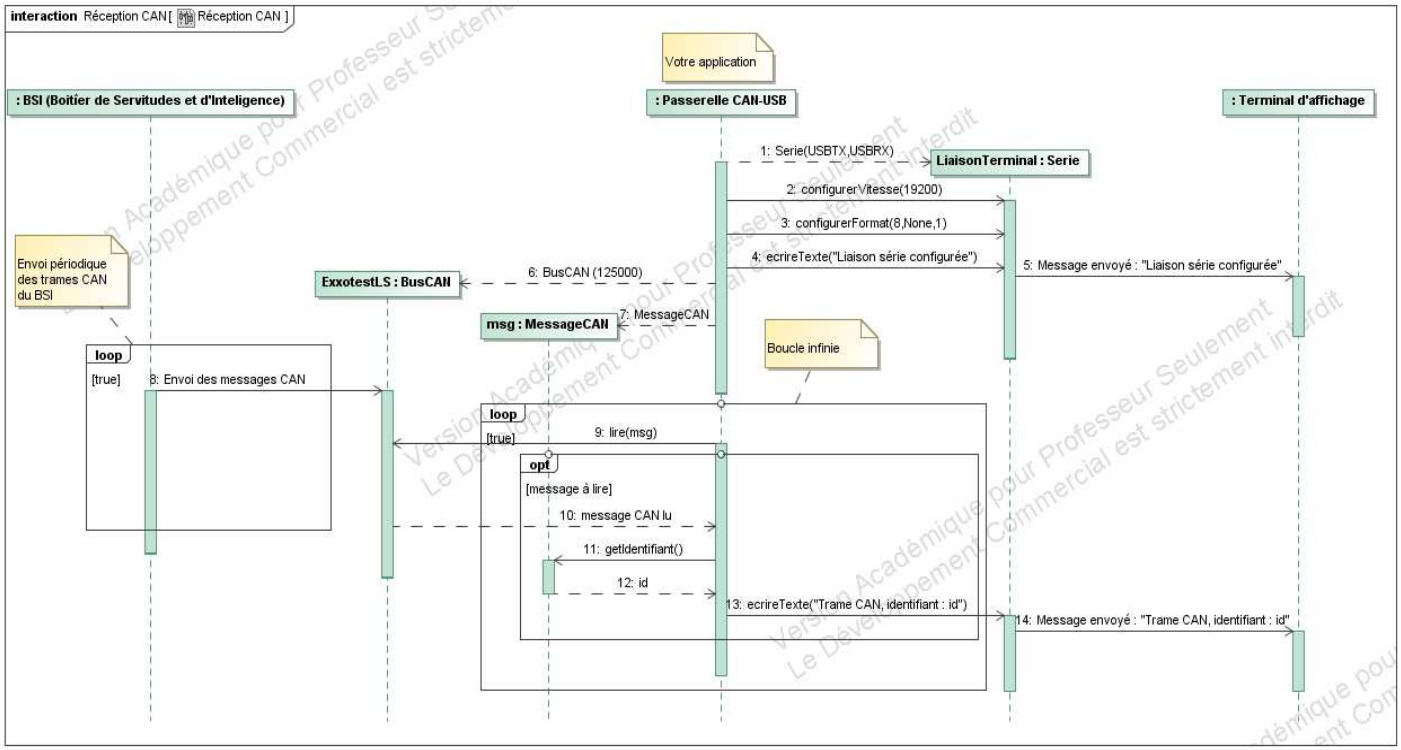
Envoi de l'identifiant du message reçu

Notre premier programme utilisera les classes Serie, BusCAN et MessageCAN. Il faudra :

- Créer un objet de classe BusCAN et configurer la vitesse de communication.
- Créer un objet de classe MessageCAN.
- Créer et configurer un objet de classe « Serie » relié sur la liaison USB-série virtuelle.

- Lire en boucle le bus CAN grâce à la méthode lire() et renvoyer sur la liaison série l'identifiant du message lu grâce à la méthode getIdentifiant().

Ces objectifs sont représentés dans le diagramme de séquence suivant :

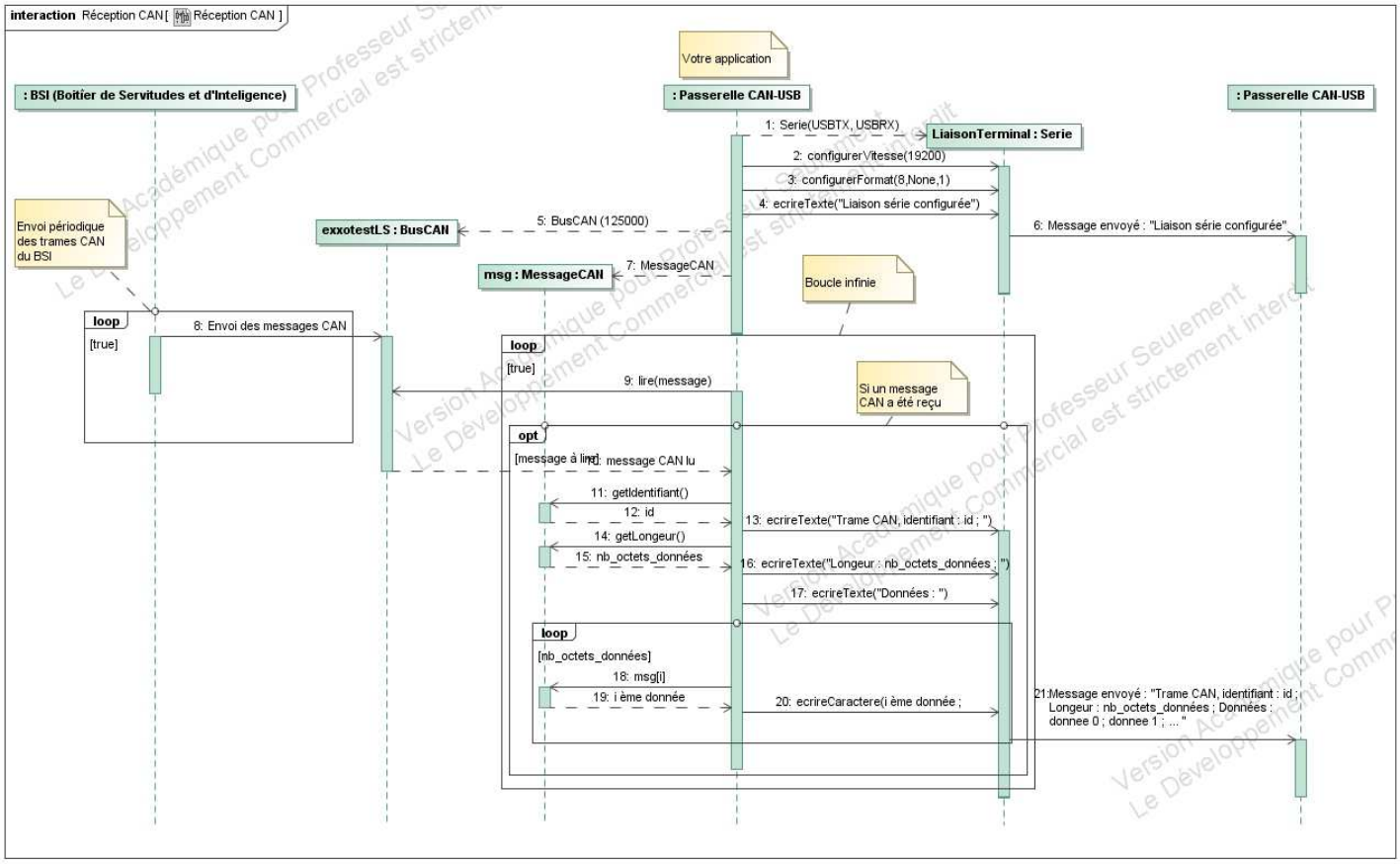


Puisque vous allez utiliser les classes Serie, BusCAN et MessageCAN vous devez inclure les fichiers .h qui vous ont été fournis. Pour formater un nombre en chaîne de caractères (tableaux de caractères), vous utiliserez la fonction « sprintf » détaillée à l'annexe 2.

18. Ajoutez cette inclusion dans le fichier *main.cpp*.
19. À partir du diagramme de séquence précédent, modifiez le fichier *main.cpp* de votre projet.
20. Compilez votre projet, implantez le dans la carte Mbed et testez-le en reliant le bus CAN de la carte Mbed a celui de la maquette Exxotest.

Envoi des données du message reçu

Nous compléterons le programme précédent en renvoyant au PC en plus de l'identifiant du message CAN reçu, sa longueur et tous ses octets de donnée. Le diagramme de séquence modifié est donc le suivant :



21. À partir du diagramme de séquence précédent, modifiez le fichier *main.cpp* de votre projet.
22. Compilez votre projet, implantez le dans la carte Mbed et testez-le en reliant le bus CAN de la carte Mbed a celui de la maquette Exxotest.

Amélioration du programme principal

Filtrage logiciel des messages reçus

Dans notre projet, seuls les messages CAN d'identifiants 0x128 et 0x0B6 nous intéressent.

Nous allons donc modifier le programme pour ne récupérer que ces messages.

23. Modifiez votre programme pour n'envoyer au PC que les messages CAN qui nous concernent.
24. Compilez votre projet, implantez le dans la carte Mbed et testez-le en reliant le bus CAN de la carte Mbed a celui de la maquette Exxotest.

Filtrage matériel des messages reçus

Le filtrage logiciel prend du temps de traitement au microcontrôleur, surtout si le bus CAN est

« chargé » en messages. Le Mbed peut filtrer ces messages de façon matériel, seuls les messages autorisés seront reçus et pourront donc être lus. La méthode accepterMessage() de la classe BusCAN permet de définir les identifiants des messages acceptés.

25. Modifiez le programme de la question 20 pour filtrer matériellement les messages CAN et ne renvoyer au PC que ceux qui nous concernent.
26. Compilez votre projet, implantez le dans la carte Mbed et testez-le en reliant le bus CAN de la carte Mbed a celui de la maquette Exxotest.

Activité 4 – Mise en place de la passerelle CAN-USB

Dans cette partie, vous allez finaliser l'application qui doit récupérer les données sur le bus CAN du véhicule et les renvoyer au terminal d'affichage. L'envoi de la trame série a été finalisé à la question 12, il suffit de modifier les données du tableau à chaque fois qu'une trame CAN correspondante arrive.

Rangement des données CAN intéressantes dans le tableau

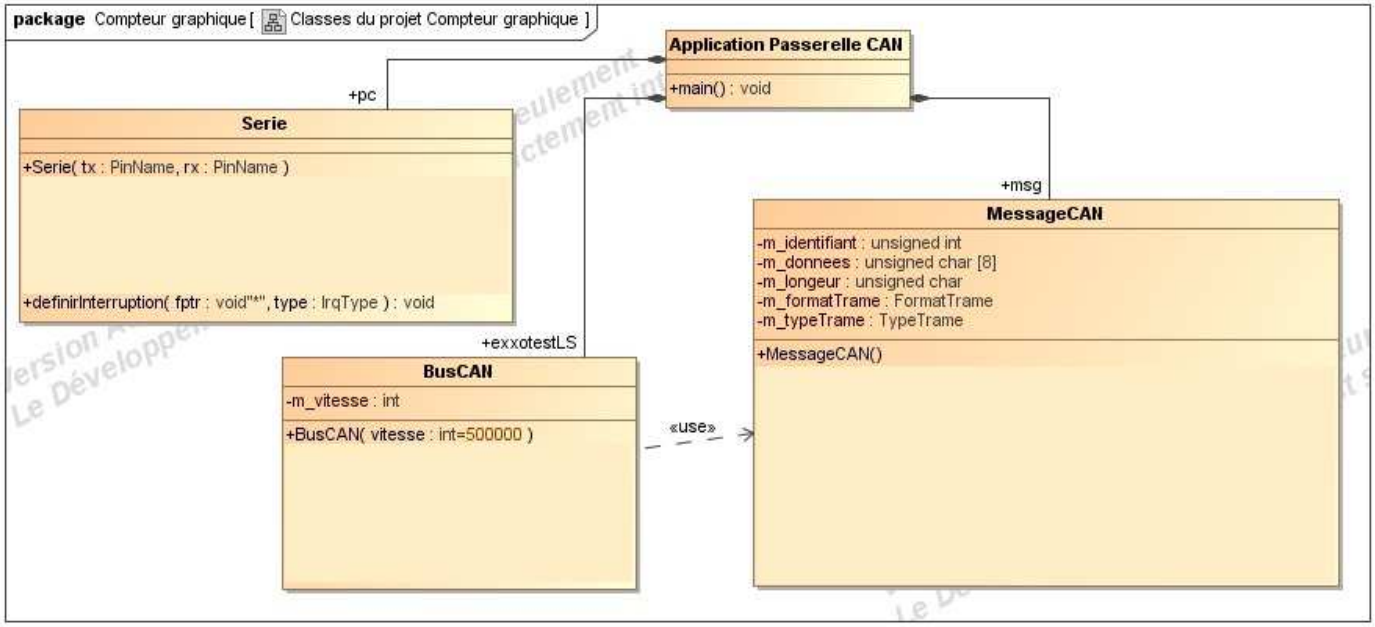
27. Créez un nouveau projet à partir des programmes des questions 12 et 24. Ajouter le code pour mettre a jour les valeurs du tableau de données lors des réceptions CAN.
28. Compilez votre projet, implantez le dans la carte Mbed et testez-le en reliant le bus CAN de la carte Mbed a celui de la maquette Exxotest.
29. Faites un test complet avec le terminal d'affichage.

Plutôt que de scruter continuellement le bus CAN pour voir si une trame est arrivée, on utilise souvent une interruption sur réception CAN. Si on autorise cette interruption, un programme sera exécuté à chaque fois qu'une trame CAN autorisée arrivera sur le microcontrôleur. La méthode definirInterruption() de la classe BusCAN permet d'autoriser cette interruption et de définir le sous programme qui sera exécuté.

Mise en place d'une interruption CAN (optionnel)

30. Créez un nouveau projet à partir du programme de la question 26. Modifiez et réorganisez le code pour mettre en place cette interruption sur réception CAN.
31. Compilez votre projet, implantez le dans la carte Mbed et testez-le en reliant le bus CAN de la carte Mbed à celui de la maquette Exxotest.
32. Faites un test complet avec le terminal d'affichage.

Annexe 1 – Diagramme de classes à compléter



Annexe 2 – Prise en main des outils Mbed

Structure matérielle de la carte microcontrôleur.

Les cartes de développement MBED que nous utiliserons (mbed NXP LPC1768) sont fabriquées autour d'un microcontrôleur puissant (cœur ARM Cortex-M3). Ces cartes disposent de nombreux bus de communication (série, I2C, SPI, USB, CAN, Ethernet, …) ainsi que de nombreux périphériques standards (timers, convertisseurs, E/S, mémoire, …). Vous trouverez tous les détails à l'adresse : http://mbed.org/handbook/mbed-Microcontrollers.

Programmation des MBED.

Les cartes MBED disposent d'un environnement de développement (IDE) en ligne avec un compilateur C/C++ intégré. Vous devez donc créer un compte en ligne ou vous retrouverez vos différents projets. De nombreux exemples y sont également disponibles ainsi que toute la documentation (Handbook). Une fois votre nouveau projet créé, le programme écrit et la compilation réussie, le code machine est automatiquement téléchargé sur votre machine. Il ne reste plus qu'à l'envoyer à votre carte MBED et à faire un « reset » pour exécuter ce nouveau programme.

Première mise en œuvre.

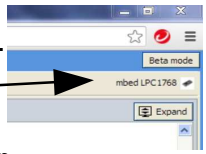
a. Utilisez le câble USB pour connecter votre microcontrôleur mbed à un PC. Le voyant d'état s'allume, indiquant qu'il est sous tension. Après quelques secondes d'activité, le PC reconnaît le microcontrôleur mbed comme un disque USB standard.

b. Aller sur le nouveau lecteur USB, puis cliquez sur MBED.HTM pour ouvrir la page d'acceuil dans un navigateur Web. Choisissez en haut à droite "Login or signup", en utilisant le compte mbed déjà créé. Cela vous donnera accès à des outils, des bibliothèques et de la documentation.

c. Vous pouvez créer maintenant un nouveau projet et commencer à travailler !

d. Créez un premier projet « essai_led » en sélectionnant « nouveau ». Un petit programme est automatiquement créé. Éditez le programme principal (main.c).

e. Choisissez une cible pour votre projet en cliquant sur l'icône en haut à droite. Vous devez choisir « mbed LPC1768 » comme sur la figure.



f. Compilez le programme, placez le fichier .bin dans votre microcontrôleur faite un reset (bouton bleu) et observez le résultat. Que se passe-t-il ? Est-ce attendu ? Commentez chacune des lignes du programme en vous aidant de la documentation en ligne (Handbook).

Annexe 3 – Écriture formatée dans une chaîne : sprintf

Source de la documentation

Site developpez.com :
http://c.developpez.com/cours/bernard-cassagne/node74.php#SECTION00753000000000000000

Utilisation

La fonction sprintf admet un nombre variable de paramètres. Son utilisation est la suivante :

sprintf (*chaîne* , *format* , $param_1$, $param_2$, ... , $param_n$)

Description

La fonction sprintf réalise le traitement de la chaîne *format* et écrits le résultat dans le tableau de caractères *chaîne*. Un *null* est écrit dans *chaîne* en fin de traitement.

La chaîne *format* contient des caractères ordinaires (c'est à dire différents du caractère %) qui doivent être copiés tels quels, et des séquences d'échappement (introduites par le caractère %), décrivant la manière dont doivent être écrits les paramètres $param_1$, $param_2$, ... $param_n$.

Si il y a moins de $param_i$ que n'en réclame le *format*, le comportement n'est pas défini. Si il y a davantage de $param_i$ que n'en nécessite le *format*, les $param_i$ en excès sont évalués, mais leur valeur est ignorée.

Valeur rendue

La fonction *sprintf* retourne le nombre de caractères écrits, ou une valeur négative si il y a eu une erreur d'entrée-sortie.

Séquences d'échappement

Les séquences d'échappement sont introduites par le caractère %, elles comportent un nombre variable de caractères indiquant le format désiré pour « l'affichage » de $param_i$.

Par exemple l'écriture entière ou flottante du nombre, l'affichage décimal ou hexadécimale, le nombre de caractères minimal souhaité, la précision d'affichage, … (détails en suivant le lien).

Exemples

Suite des caractères du tableau Temp formaté avec la fonction *sprintf* :

sprintf(Temp,"Decimal = %d",16); → Decimal = 16

sprintf(Temp,"Decimal = %+d",16); → Decimal = +16

sprintf(Temp,"Decimal = %+d",-16); → Decimal = -16

sprintf(Temp,"Decimal = %10d",16); → Decimal = 16

(10 caractères complétés par des espaces)

sprintf(Temp,"Decimal = %10.6d",16); → Decimal = 000016

(10 caractères dont 6 complétés par des zéros, les autres par des espaces)

sprintf(Temp,"Hexadecimal = %x",16); → Hexadecimal = 10

sprintf(Temp,"Hexadecimal = %#x",16); → Hexadecimal = 0x10

sprintf(Temp,"Flottant = %.4f",1.234567890123456789e5); → Flottant = 123456.7890

Si i vaut 15 que j vaut 16 et que v vaut 3.25, la commande

sprintf(Temp,"Nb1 = %#x, Nb2 =%d, tension = %4.2fV",i ,j ,v) ;

renvoie la suite de caractères → Nb1 = 0xF, Nb2 = 16, tension = 03.25V