

UNIVERSITE PAUL SABATIER

TOULOUSE III

TP EM7ECEIM  
CONCEPTION DES SYSTEMES

MASTER EEA

# TABLE DES MATIERES

Remarques Pratiques.....	4
Mini Projet: Gestion d'une Borne de Recharge pour Véhicule Electrique .....	5
Annexe 1 : Le débogueur ddd.....	13
Annexe 2 : Umbrello.....	15

## REMARQUES PRATIQUES

Ces TP d'Informatique Industrielle se déroulent sur PC et essentiellement sous le système d'exploitation LINUX. Sont réunis ici des remarques d'ordre pratique concernant l'utilisation des machines de la salle mises à votre disposition.

### 1 Login et mot de passe

Les machines sont reliées en réseau à un serveur qui stocke en particulier tous les fichiers utilisateurs. Chaque utilisateur (chaque binôme en l'occurrence) doit posséder un compte pour pouvoir se connecter à une machine. Un nom de compte vous sera attribué au début de la première séance.

C'est le nom qui devra être rentré à l'affichage de l'invite: `« nom_de_machine login: »`

Il vous sera alors demandé un mot de passe. Chaque compte possède un mot de passe par défaut: `« jsupe2008 »`. Il est vivement conseillé de modifier ce mot de passe dès la première séance en utilisant la commande `« yppasswd »`.

Les règles de sécurité demandent que tout mot de passe comporte au moins 6 caractères et que ces caractères constituent un mélange de lettres majuscules et minuscules ou un mélange de caractères alphabétiques et de chiffres.

Une fois la connexion effectuée, vous vous trouvez en mode « console » et le système attend vos commandes.

Le « prompt » (message d'invite) affiche votre nom de login (nom du compte), le nom de la machine sur laquelle vous êtes et le répertoire dans lequel vous vous trouvez. Exemple:

```
« [mleea_001@alicia mleea_001]$ »
```

### 2 Arrêt des machines

Les accès disques étant tamponnés (bufférisés) un arrêt brutal de la machine risque fortement d'endommager le système de fichiers et par conséquent d'entraîner la perte de tous vos efforts!

Si vous voulez (ou devez) arrêter la machine sur laquelle vous travaillez de manière propre, utilisez la combinaison de touches `« Ctrl+Alt+Del »`.

En cas de blocage complet n'hésitez pas à faire appel à un enseignant.

### 3 Impression des listings

L'impression des listings se fait sur une imprimante connectée au réseau. A partir de la ligne de commande il suffit de taper la commande `« a2ps mon_fichier.c »` pour sortir un listing formaté avec entête. La commande `« a2ps »` est en fait un filtre d'impression qui traduit un texte ascii en postscript en fonction de son type. Un fichier d'extension `« .c »` est ainsi reconnu comme un source C et traité en conséquence (commentaires en italiques, etc...).

L'autre commande (standard) pour imprimer est `« lpr fichier »`.

Le contrôle de l'impression se fait en utilisant la commande `« lpq »` qui permet de visualiser les travaux d'impressions en attente. Chacun des travaux est associé à un numéro qui doit être utilisé pour l'enlever éventuellement de la file d'impression en invoquant la commande `« lprm »` et en lui passant ce numéro en argument.

Il est rappelé que l'imprimante est une ressource collective. Le respect des autres utilisateurs impose de vérifier que l'impression que l'on a lancé s'est bien déroulée et ne bloque pas l'imprimante pour une raison ou pour une autre.

Le papier est par ailleurs une ressource rare qui doit donc être utilisée à bon escient.

En pratique deux imprimantes sont reliées au réseau. L'une d'entre elles est définie comme imprimante par défaut mais on peut accéder à l'autre en précisant son nom grâce à l'option `« -P »`. Par exemple: `« a2ps -Pimp4 prog.c »` enverra le travail d'impression sur l'imprimante 4 (à droite en entrant) et `« a2ps -Pimp3 prog.c »` l'enverra sur l'imprimante 3. L'imprimante définie par défaut dépend de la machine sur laquelle vous travaillez.

# MINI PROJET: GESTION D'UNE BORNE DE RECHARGE POUR VEHICULE ELECTRIQUE

## 1 Cahier des charges

Le Système à concevoir doit permettre le fonctionnement simplifié d'une borne de recharge de véhicule électrique, élément clé du développement dans les villes de véhicules électriques. La borne de recharge standard permet à un utilisateur de raccorder son véhicule électrique pour le recharger en toute sécurité et rapidement. Elle dispose d'un ou deux socles de prises protégés par une trappe verrouillable, de voyants sur la face avant (Disponible, Défaut) et sur le côté (Charge, Prise), de boutons poussoirs (Charge, Stop), de transmission de données pour son exploitation et sa maintenance et d'un lecteur de badge pour identifier le client (voir Annexe 1).

Dans ce système, le client doit s'enregistrer auprès de l'opérateur pour disposer d'un badge d'identification. L'opérateur peut contrôler à distance l'ensemble de ses bornes par un système de communication GPRS/3G (par exemple connaître l'ensemble des bornes hors service et les statistiques d'utilisation de chacune d'elles).

Il existe 4 modes de recharge différents. Le schéma de charge retenu en France est la charge en mode 3 avec un connecteur de type 3 côté borne fournissant un courant alternatif jusqu'à 500V de 3.6kVA (charge normale) jusqu'à 22 kVA (charge accélérée) (voir Figure 2). Dans ce contexte, la charge des batteries du véhicule est contrôlée par la borne de recharge.

Le fonctionnement concernant la recharge d'un véhicule est le suivant : Si la borne est disponible (voyant « Disponible » allumé), le client s'authentifie : en cas de succès le voyant « Charge » clignote pendant 8 s, sinon « Défaut » clignote rouge pendant 8 s. Le client dispose de 1 minute pour appuyer sur le bouton « Charge ». Le voyant « disponible » est alors éteint. La trappe du socle de prises est déverrouillée, et l'utilisateur peut connecter la prise à son véhicule. Une fois connectée, la prise est verrouillée et le voyant « Prise » allumé. Commence ensuite le cycle de dialogue / charge avec le véhicule.

Le circuit de recharge dédié et imposé dans le « Mode 3 » est défini dans la proposition de norme IEC 61851-1 « ELECTRIC VEHICLE CONDUCTIVE CHARGING SYSTEM ». Cela permet de garantir une sécurité maximale des utilisateurs lors de la recharge de leur véhicule électrique. La Figure 3 représente la connectique entre une borne et un véhicule.

Un contrôleur de recharge, nommé générateur SAVE et situé côté infrastructure, vérifie les éléments suivants avant d'enclencher la recharge :

Vérification que le véhicule est bien connecté au système (Etat B); Vérification que la masse du véhicule est bien reliée au circuit de protection de l'installation (Etat C); Vérification de la cohérence des puissances entre le câble, le véhicule et le circuit de recharge (Etat D); Détermination de la puissance maximale de recharge qui sera allouée au véhicule.

L'ensemble de ces vérifications et de la communication se font au travers d'une communication sur fil spécifique, dit « fil PILOTE ». Voir la représentation d'une prise Type 3 sur la Figure 2 et la connectique entre la borne et le véhicule en Figure 3 (on notera la présence du connecteur S2 sur le véhicule). La figure 4 représente la valeur de la tension durant le processus.

Ainsi, la charge se fait en fonction du dialogue suivant entre le véhicule et la borne (Figure 4):

- ☐ Etat A : véhicule électrique non connecté, le générateur SAVE fournit une tension de + 12V. Le voyant « charge » s'allume en rouge.
- ☐ Etat B : véhicule électrique connecté et système d'alimentation non disponible, la tension chute à +9V. S2 est ouvert.
- ☐ Etat C : véhicule électrique connecté et système d'alimentation disponible, le générateur SAVE fournit un signal carré +9V / -12V de fréquence 1 kHz qui aura pour effet de fermer le contacteur S2 sur le véhicule.

- Etat D : S2 est fermé et engendre une nouvelle chute de tension à 6V. Le générateur SAVE fournit un signal (+6V/-12V) de fréquence 1 kHz à rapport cyclique variable. (signal PWM modulation en largeur d'impulsion). Ce rapport cyclique indique la puissance que la borne peut fournir au chargeur. La largeur d'impulsion varie linéairement entre 100 us (6A fourni par la borne) et 800 us (48A). La position fermée de S2 indique que le chargeur du véhicule électrique peut recevoir de l'énergie. La fermeture de S2 entraîne la fermeture d'un contacteur du circuit puissance sur la borne de recharge (AC). *(il n'apparaît pas sur le schéma)*
- Etat E : Quand le véhicule détecte que la batterie est chargée, S2 est ouvert. Le signal remonte à 9V/-12V, indiquant à la borne d'ouvrir le contacteur AC. Le voyant « charge » s'allume en Vert.
- Etat F : retour à 12V quand la prise est déconnectée. Le voyant « charge » s'éteint.

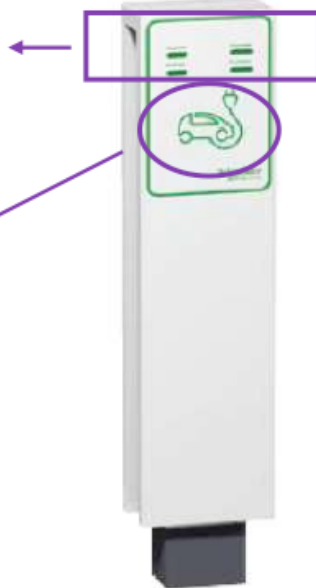
Lors de la reprise du véhicule, le client s'identifie à nouveau sur la borne. Si le véhicule est encore en charge, il appuie sur le bouton « Stop », sinon il peut le récupérer directement après l'avoir débranché. Le contacteur AC doit être ouvert, la prise déverrouillée. Une fois la prise débranchée par le client, la trappe doit être verrouillée, voyant « Prise » éteint et le voyant « disponible » allumé.

L'administrateur du système disposera d'une console de gestion des abonnements. Pour ajouter un nouveau client celui-ci indiquera les informations personnelles du client (nom, etc ...) et associera une carte dont la lecture se fera par le lecteur de carte. Pour supprimer un client, l'administrateur lira la carte et procédera à la suppression du client associé.

#### Voyants face Avant

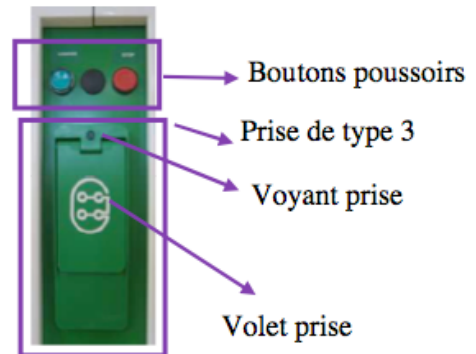


Non utilisé

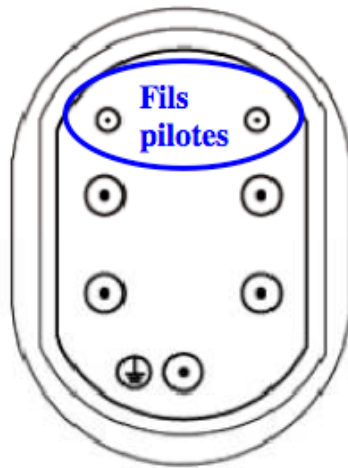


Borne standard sur pied

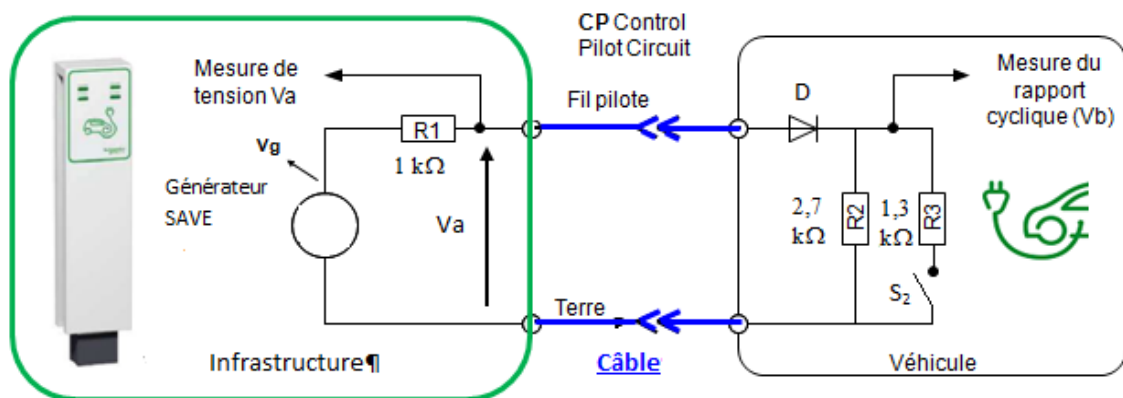
#### Côté droit de la borne



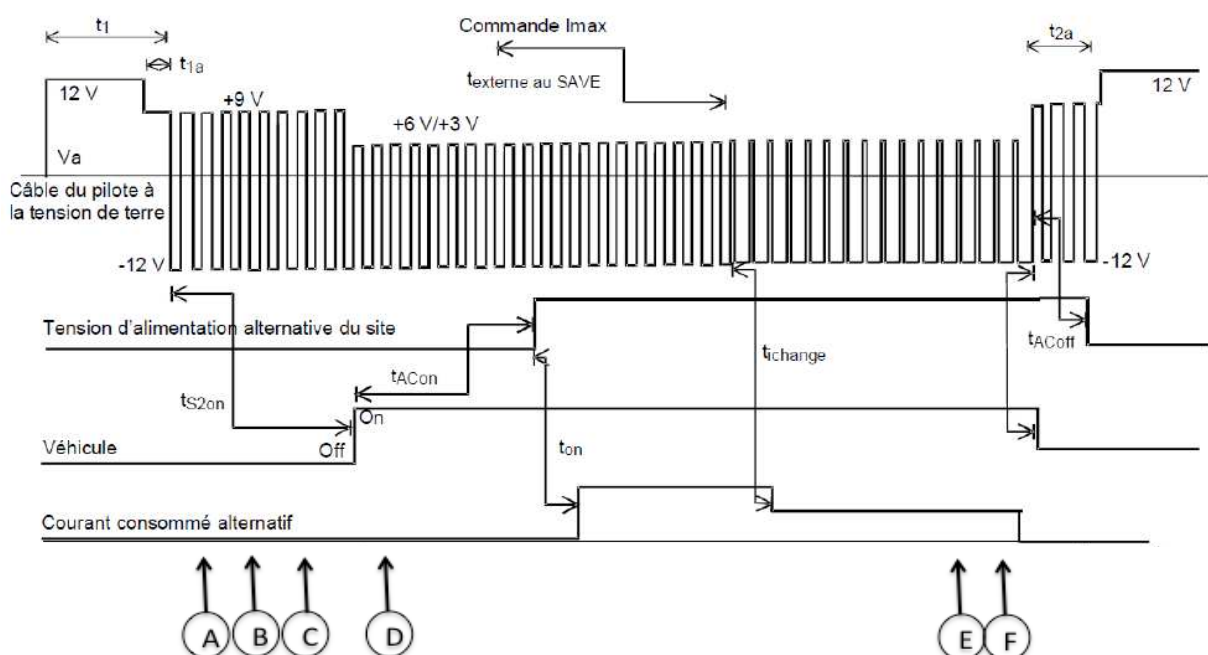
**Figure 1: Présentation de la borne de recharge étudiée**



**Figure 2: Prise Type 3 nécessaire au mode de recharge adopté en France.**



**Figure 3: Le schéma électrique du circuit Fil Pilote est donné ci-dessous**



**Figure 4: Dialogue en modulation de largeur d'impulsion entre la borne et le véhicule**

## **2 Travail à effectuer**

Le cahier des charges précédemment donné représente les besoins d'un client (le constructeur des cornes de recharge électrique). Il est demandé de réaliser un logiciel satisfaisant ce cahier des charges suivant une approche méthodologique rigoureuse en vue de la maintenance future du produit.

### **2.1 Analyse**

L'analyse fine du problème, s'appuyant sur le langage UML, sera faite en Travaux Dirigés en vue de préciser les besoins du client (vision client) pour fournir une spécification claire et précise à l'équipe de conception.

### **2.2 Analyse et conception**

A partir de la spécification fournie faire l'analyse et la conception du logiciel en vue de la réalisation du produit (vision équipe de conception) en utilisant le langage UML. Prenez aussi en compte les cas d'utilisation liés à l'ajout et suppression de clients dans le système.

### **2.3 Implémentation**

L'implémentation devra s'appuyer sur la conception élaborée précédemment.

Elle s'étalera sur deux séances en commençant par l'implémentation des cas d'utilisation « recharger véhicule » et « recharger batterie ». La partie « reprise du véhicule » sera implémentée après validation des UC précédents, puis les variantes liés à l'interruption par le bouton STOP et la gestion des contraintes non fonctionnelles devront ensuite être implémentées. Il ne sera pas nécessaire d'utiliser une structure de données dynamique pour stocker la liste des clients, un simple tableau sera suffisant.

## **3 Mise en oeuvre**

La mise en oeuvre du projet de gestion d'une borne de recharge de véhicule électrique fait appel d'une part à un simulateur logiciel comportant la partie gauche représentant la borne de recharge à piloter et la partie droite affichant l'état simulé d'un véhicule en charge sur la borne et d'autre part à un lecteur de carte relié au boîtier d'extension du PC .

Ces éléments, ainsi que les moyens de les utiliser sont décrits ci-dessous.

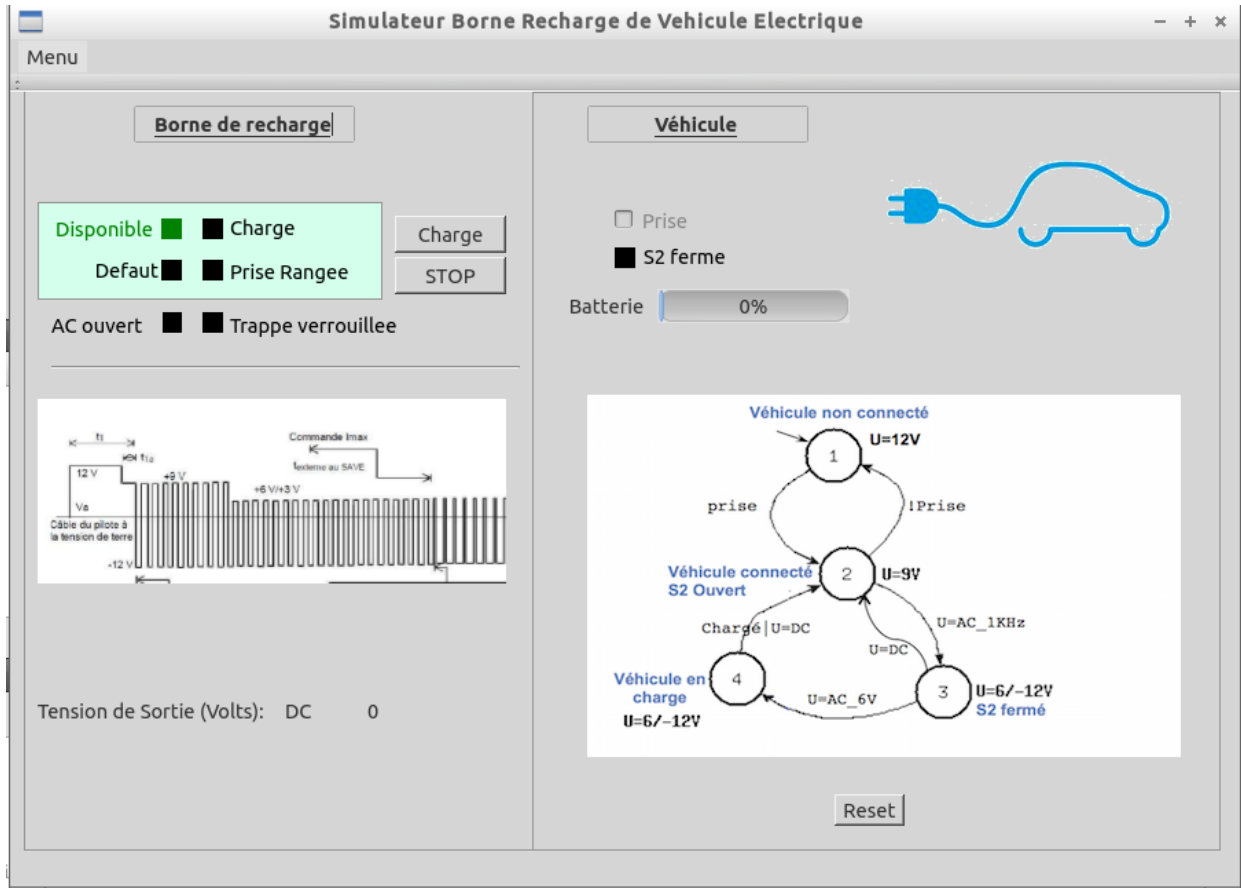
### **3.1 Simulateur**

Le simulateur, dont une vue est représentée ci-dessous, comporte les deux parties suivantes.

Dans sa partie gauche se trouve la borne de recharge à piloter avec :

- un bloc de 4 voyants (Disponible, Charge, Défaut, Prise)
- 2 boutons (Charge et Stop)
- 2 actionneurs à piloter (contacteur AC et verrouillage de trappe)
- un indicateur qui affiche la valeur de la tension commandée sur la prise (DC pour signal continu, AC\_1K pour signal PWM de fréquence 1KHz, AC\_CL pour un signal PWM de 1KHz à rapport cyclique variable).





IHM : sim\_borne

Pour communiquer avec ce simulateur (récupérer des informations ou en fournir) une structure de mémoire partagée est mise à disposition. Celle-ci est définie dans le fichier à inclure: `donnees_borne.h`

```
typedef enum _led {OFF, ROUGE, VERT} led;
typedef enum _pwm {STOP, DC, AC_1K, AC_CL} pwm;

typedef struct
{
    // Borne de recharge
    // OUT
    led led_dispo;
    led led_charge;
    led led_default;
    led led_prise;
    led led_trappe;

    int contacteur_AC; // 0 ouvert; 1 ferme

    // IN
    int bouton_charge;
    int bouton_stop;

    // Vehicule
    // OUT
    pwm gene_pwm;
    int gene_u;

    int timer_sec; // timer incrementé toutes les secondes
} entrees;
```

Une donnée de ce type sera accessible à partir d'un pointeur entrees\* valeurs; (voir l'exemple qui suit).

Les différents champs de la structure entrees sont les suivants:

- led\_xxx correspond aux 4 voyants (dispo, charge, défaut, prise) de la face avant. Ils peuvent prendre comme valeur (OFF, ROUGE, ou VERT).
- led\_trappe, contacteur\_AC sont les deux actionneurs à piloter. led\_trappe doit prendre la valeur VERT pour être déverrouillée, OFF sinon. Le contacteur\_AC doit être positionné à 0 pour être ouvert ou 1 pour être fermé.
- bouton\_charge et bouton\_stop mis à 1 par le simulateur si le bouton est activé. Il doit être remis à 0 après consultation.
- gene\_pwm qui doit prendre successivement les valeurs STOP, DC, AC\_1K, AC\_CL selon le mode de commande de charge explicitée plus haut.
- gene\_u qui retourne la tension sur le fil pilote. Ne pas modifier cette valeur, elle le sera par le simulateur de véhicule en fonction des évolutions des contacts électriques.
- timer\_sec qui renvoi l'horloge système en seconde. Il est recommandé de définir une classe permettant la manipulation plus aisée de ce timer avec: une méthode timer\_raz() pour mémoriser l'heure du déclenchement du timer dans un attribut et une méthode timer\_valeur() renvoyant la différence entre l'horloge actuelle et l'heure du déclenchement.

Pour accéder à ces différents champs, il est nécessaire d'inclure les fichiers mem\_sh.h et donnees.h, de déclarer une variable de type entrees\* et d'appeler la fonction acces\_memoire. Ainsi toute classe qui devra manipuler un des ces champs devra, comme l'exemple du timer ci dessous comporter les attributs et les méthodes suivantes :

```
#include <donnees_borne.h>
#include <memoire_borne.h>

entrees *io;
int shmid;
int depart_timer;

void timer_initialiser()
{
    io=acces_memoire(&shmid);
    /* associe la zone de memoire partagee au pointeur */
    if (io==NULL) printf("Erreur pas de mem sh\n");
    depart_timer=io->timer_sec;
}
```

L'accès aux champs se fait de manière classique à partir du pointeur, par exemple:

io->led\_dispo=VERT permet d'allumer le voyant charge à vert.

## 3.2 Lecteur de cartes

L'interface avec le lecteur de carte se fait en utilisant les fonctions contenues dans le fichier à inclure lcarte.h:

```

/* Interface lecteur de cartes */
/*-----*/
void initialisations_ports();
/*-----*/
void liberation_ports();
/*-----*/
void attente_insertion_carte();
/*-----*/
void attente_retrait_carte();
/*-----*/
int carte_inseree();
/*-----*/
unsigned short int lecture_numero_carte();
/*-----*/

```

Le lecteur de carte de marque GEMPLUS et de modèle GEMPC USB-SL est relié au PC par l'intermédiaire d'un câble USB. Il respecte la norme PC/SC (Personal Computer / Smart Card) qui offre une bibliothèque logicielle pour l'accès à des cartes à puces sous Windows et Linux. Les cartes à puces utilisées sont de type carte à mémoire de 1 Kb et disposent de mécanismes d'authentification embarqués. Dans le cadre de ce projet, nous avons développé une API (Application Programming Interface) simplifiant l'utilisation de ces cartes.

En pratique, les procédures fournies permettent de réaliser les différentes opérations nécessaires:

On doit d'abord:

- initialiser les ports (`initialisations_ports()`)
- Attendre l'insertion de la carte (`attente_insertion_carte()`)
- enfin, on peut récupérer son numéro (`numero=lecture_numero_carte()`).
- il ne reste qu'à attendre son retrait (`attente_retrait_carte()`).
- Et libérer les ports : (`liberation_ports()`)

### 3.3 Remarques pratiques

Pour avoir accès aux fonctions définies dans les divers fichiers inclus vous devez l'indiquer au compilateur. La compilation devrait donc ressembler à:

```
gcc -Wall -o mon_programme mon_programme.c -ltpborne
```

Toutefois, afin de faciliter la compilation, un fichier de compilation « **Makefile** » vous sera donné. Pour l'utiliser, vous devez le modifier afin de l'adapter à vos besoins, puis taper les commandes « `make depend` » et « `make` » qui auront pour effet de recompiler votre programme.

Le lancement du simulateur se fait (en arrière-plan pour garder la main dans la fenêtre) en tapant: `sim_borne &`.

La communication entre l'interface graphique et votre programme se fait par l'intermédiaire d'un segment de mémoire partagé. En cas d'arrêt brutal du simulateur ou de l'existence d'instances multiples, il se peut qu'un message d'erreur apparaisse au lancement de la commande « `sim_carb` ». Dans ce cas, quittez le simulateur et supprimez les segments de mémoire partagés. Pour cela vous pouvez utiliser la commande *ipcclean* ou les commandes associées *ipcs -m* et *ipcrm -m segid* :

- *ipcclean* supprime la mémoire partagée et les sémaphores d'un serveur qui a dû s'arrêter brutalement.
- *ipcs* fournit des informations sur l'usage des ressources IPC.
- *ipcrm* détruit la ressource IPC spécifiée.

Vous pouvez consulter les fichiers d'entête (.h) dans le répertoire `/usr/local/include`.

### 3.4 Travail à rendre et évaluation

Un rapport de conception, un listing commenté et une mini-recette sont requis pour l'évaluation du projet.

#### Le rapport de conception :

Le rapport de conception couvrira l'**intégralité** de la conception, c'est-à-dire depuis la définition des exigences, jusqu'à la conception détaillée. Vous complétez les cas d'utilisation étudiés en TD et traiterez l'intégralité des cas d'utilisation identifiés en TD, c'est à dire « recharger véhicule », « charger batterie » et « reprendre véhicule ».

- La première partie **d'analyse** précisera les exigences (lister les exigences dans le tableau disponible sur moodle sous le nom « Documents Template d'Ingénierie Système »), les cas d'utilisation avec au moins la définition du contexte, la description des scénarios, la définition des objets, les diagrammes de séquence, une ébauche du diagramme de classes.
- La deuxième partie de **conception**, précisera les diagrammes de collaboration, complétera le diagramme de classes, décrira toutes les méthodes définies par leur contrat type (les contrats type pourront être générés à partir des commentaires du code avec doxygene) et donnera les principes de conception du programme principal.
- La troisième partie sera le **code source** de votre logiciel. Il devra être mis en forme et commenté correctement. Une attention toute particulière sera portée sur la cohérence de votre logiciel avec la spécification.

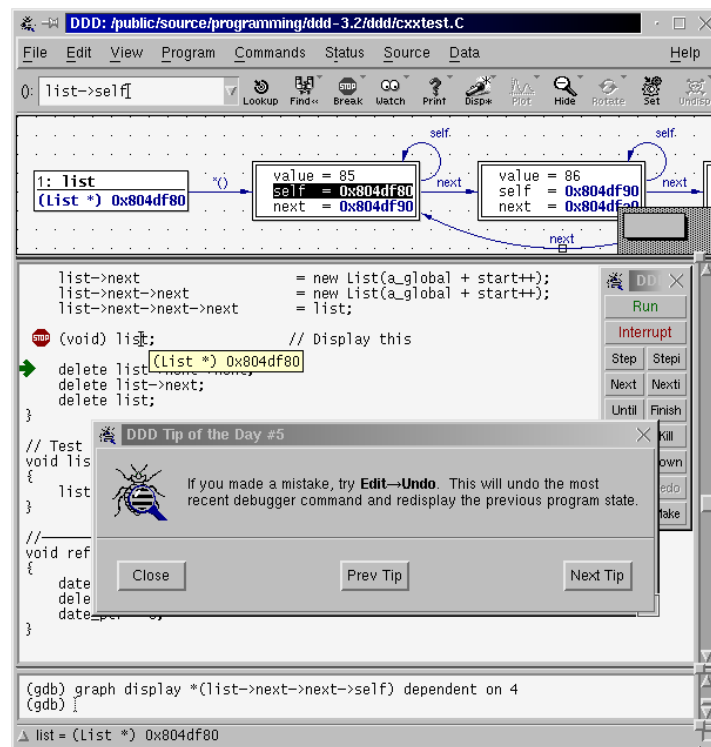
#### La recette :

La recette de votre projet consiste en une **démonstration** de votre logiciel lors de la dernière séance. La couverture des cas d'utilisation par votre logiciel, sa fiabilité, son ergonomie seront des éléments importants de l'évaluation.

### 3.5 Conseils Pour débiter la première séance

- Commencer par le Use Case « recharger véhicule ». Suivre scrupuleusement le diagramme de collaboration vu en TD.
- Copiez l'environnement de développement préparé par l'équipe pédagogique.  
`cp -r /home/MEEA/projet_uml .`
- Editez le premier objet (lecteur\_carte) et testez la compilation. Pour cela ajoutez un affichage (printf) dans la méthode `lecteur_carte_lire_carte()`. Compilez avec la commande `make`. Exécutez par la commande `./borne`.
- Commencez à implémenter le premier UC en suivant le diagramme de collaboration. Faites la lecture du numéro de carte avec l'interface de programmation décrite dans le cahier de TP. Affichez ce numéro dans le terminal (debug). Pour cela complétez `lecteur_carte_lire_carte()`. Compilez et testez.
- Continuez à implanter le premier UC en ajoutant l'objet `base_clients` et sa méthode `authentifier` (elle doit retourner pour l'instant toujours vrai). Faites l'appel depuis `lecteur_carte_lire_carte()`. Compilez et testez.
- Implantez l'objet `Voyants` en utilisant le segment de mémoire partagé de l'IHM décrit dans le cahier de TP. Compilez et testez.
- Maintenant, vous pouvez continuer à implanter vos UC sereinement ...

# ANNEXE 1 : LE DEBOGUEUR DDD



**ddd** est un outil GNU d'aide à la mise au point de programmes. ddd permet de lancer une application, de la stopper où l'utilisateur le souhaite, d'examiner et de modifier des variables, d'exécuter des fonctions pas à pas ...

Pour pouvoir déboguer un programme sous ddd il faut l'avoir compilé préalablement avec l'option `-g` afin de générer une table des symboles utilisée par le débogueur.

```
gcc -g -o monprogramme monprogramme.c
```

## 1. Lancer ddd

A partir d'une fenêtre `xterm` taper la ligne de commande:

```
ddd [<monprogramme> <fichier image memoire>] &
```

Les deux paramètres sont facultatifs (généralement n'utiliser que le premier):

- `monprogramme` : nom de l'exécutable obtenu avec l'option de compilation `-g`
- `fichier image mémoire` : core dump obtenu après un problème lors de l'exécution.

Il est donc possible :

- de faire la mise au point d'un programme après une exécution qui a engendré un fichier core; le débogueur précise alors le type de problème rencontré et situe le problème dans le code source.
- de faire la mise au point sous le contrôle du débogueur (sans fichier core) à l'aide de commandes `xxgdb`.

## 1 La fenêtre ddd

Au lancement on obtient une fenêtre ddd composée de 3 parties<sup>1</sup> :

- fenêtre source : affichage du source du programme en cours de mise au point,
- fenêtre de commande (à droite): présente un certain nombre de commandes ddd sous forme de boutons (**run**, **cont**, **next** ...)

<sup>1</sup> Voir figure

- fenêtre de travail : fenêtre d'entrée des commandes (en ligne, si les boutons du bandeau ne sont pas utilisés), fenêtre d'affichage des résultats des commandes ddd et d'affichage des sorties du programme.

## 2 Commandes de base

Parmi les commandes ddd nous présentons sous forme de tableau quelques commandes élémentaires présentées sous forme de boutons. L'une des commandes ddd permet d'avoir des informations en ligne: **help** suivi du nom d'une commande.

Toutes ces commandes sont soit des commandes en lignes, soit utilisables depuis l'interface graphique.

<b>run</b>	lance l'exécution
<b>cont</b>	continue l'exécution après arrêt sur un point d'arrêt
<b>next</b>	exécute une fonction en un pas
<b>step</b>	exécution pas à pas (ligne par ligne)
<b>break</b>	mise en place d'un point d'arrêt
<b>delete</b>	efface un point d'arrêt
<b>print</b>	affiche le contenu d'une variable
<b>print *</b>	affiche le contenu d'une variable pointée
<b>file</b>	permet de sélectionner un fichier exécutable
<b>finish</b>	termine l'exécution d'une fonction
<b>interrupt</b>	interrompt la mise au point (devra être relancée par run)
<b>quit</b>	termine la session xxgdb

## 3 Remarques sur la mise au point de programmes

ddd est un outil de mise au point puissant, assez facile à utiliser à minima ... mais c'est un outil de plus à connaître. Dans bien des cas, quelques instructions d'écriture, bien placées dans les programmes, permettent de localiser rapidement les problèmes et d'y remédier.

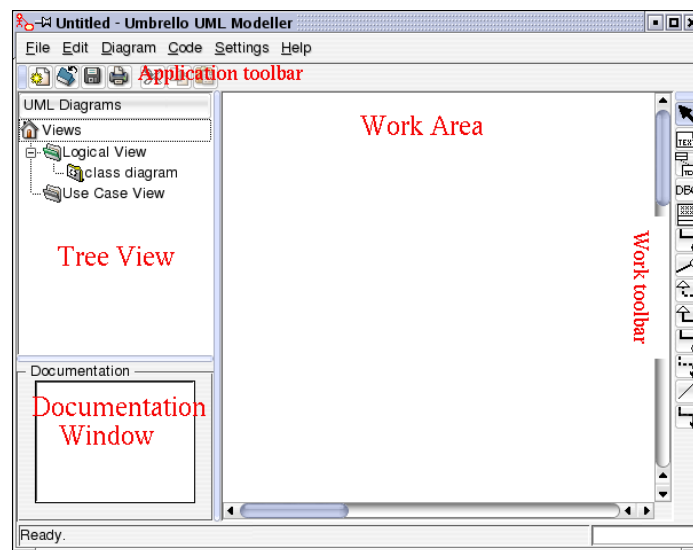
Par contre, dans le cas d'utilisation intensive de pointeurs ddd s'avère un outil indispensable pour détecter des pointeurs non initialisés et autres problèmes causant des erreurs d'exécution.

## ANNEXE 2 : UMBRELLO

**Umbrello** est un outil de modélisation UML utilisable pendant la phase de conception et le développement de logiciels. Les principales fonctionnalités actuelles du logiciel sont : l'édition des diagrammes UML et la génération automatique de code. Umbrello 1.2 supporte les diagrammes suivants : diagrammes de classes, de séquences, de collaborations, de cas d'utilisation, d'états, d'activités, de composants, et de déploiement.

### 1. Interface graphique

Pour exécuter Umbrello tapez simplement la commande « umbrello » ou cherchez l'icône dans le menu Applications.



La fenêtre principale d'umbrello est composée de trois parties :

- en haut à gauche, une partie permettant de naviguer entre les différentes entités du modèle.
- en bas à gauche, une partie documentation donnant un rapide aperçu d'une entité.
- à droite, la zone d'édition des diagrammes UML. A l'extrême droite, se trouve une barre d'icônes contenant les différentes entités relatives au diagramme en cours d'édition.

### 1 Création d'un projet et principe de modélisation

La création d'un nouveau projet et l'ajout de nouveaux diagrammes sont relativement simples. La création d'un nouveau projet se fait par la barre de menu « Fichier / Nouveau ». L'ajout d'un nouveau diagramme se fait en sélectionnant dans l'espace de navigation la vue qui recevra le nouveau diagramme (par exemple la vue logique), puis à l'aide du bouton droit de la souris ou du menu « Diagramme ». Cliquer sur « Nouveau » et sélectionner le type de diagramme à ajouter à la vue. Pour chaque type de diagramme, la barre d'icônes à droite indique les éléments UML utilisables dans ce diagramme. Cliquer sur un de ces icônes pour ajouter l'élément au diagramme.

### 2 Modification des propriétés des éléments UML

Tous les éléments de modélisation UML (classe, état, cas d'utilisation, ...) sont modifiables et paramétrables en utilisant le menu « propriété » accessible depuis l'élément par le bouton droit de la souris. Ce menu permet par exemple d'ajouter des attributs et des méthodes à une classe ou de changer le nom de rôle d'une association.

### **3 Génération automatique du code source**

Un des attraits des ateliers de génie logiciel est leur capacité à générer automatiquement le code source du programme à partir du modèle. Bien sûr, cette génération n'est jamais totale et doit toujours être complétée par le programmeur. Umbrello produit le squelette du programme à partir du diagramme de classe. Cette fonctionnalité n'est disponible dans umbrello que pour certains langages, en particulier C++ et Java. Elle n'est donc pas utilisable pour de la génération de code C.

Pour utiliser cette fonctionnalité, utiliser le menu « Code source / Code Generation Wizzard ... », puis suivre les différentes fenêtres de dialogue et terminer par le bouton « Générer ». Dans le répertoire spécifié se trouvent les fichiers générés relatifs aux classes sélectionnées.