UNIVERSITE PAUL SABATIER

TOULOUSE III

TP EM7ECEIM

CONCEPTION DES SYSTEMES

MASTER EEA

TABLE DES MATIERES

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:
« `[m1eea_001@alicia m1eea_001]$` »

2 Arrêt des machines

Les accès disques étant tamponnés (bufférisé) 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 a 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 a l'autre en précisant son nom grace 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'apparait 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. Attention le numéro de carte doit correspondre à celui qui a été utilisé pour déposer le véhicule. 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 procèdera à la suppression du client associé. Voir le cas d'utilisation « administrer borne ».

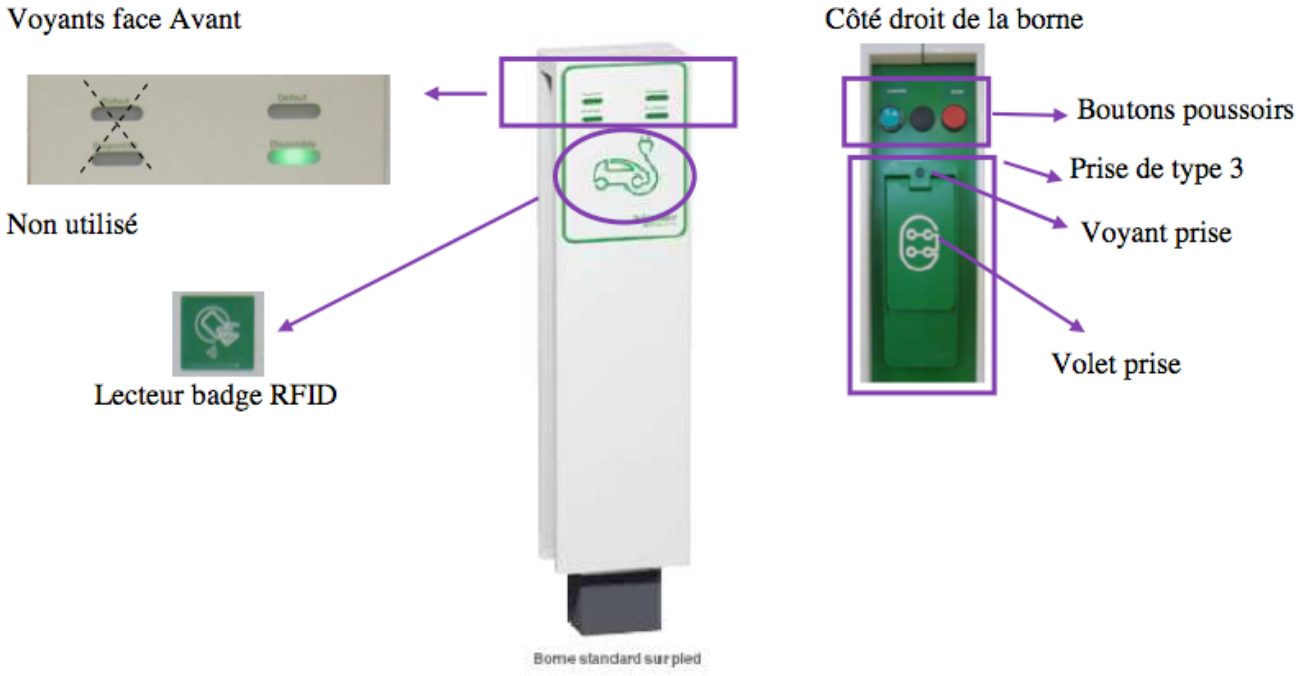


Figure 1: Présentation de la borne de recharge étudiée

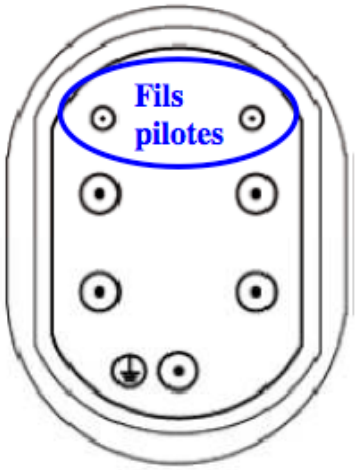


Figure 2: Prise Type 3 nécessaire au mode de recharge adopté en France.

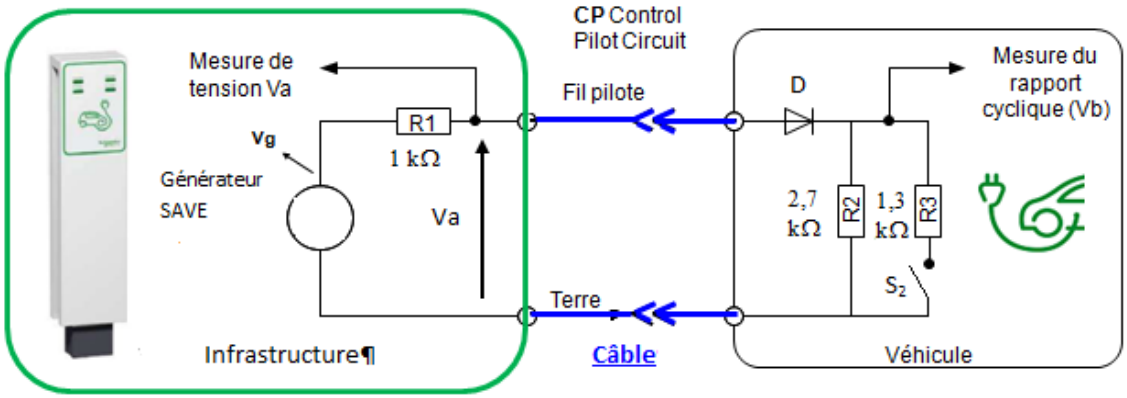


Figure 3: Le schéma électrique du circuit Fil Pilote est donné ci-dessous

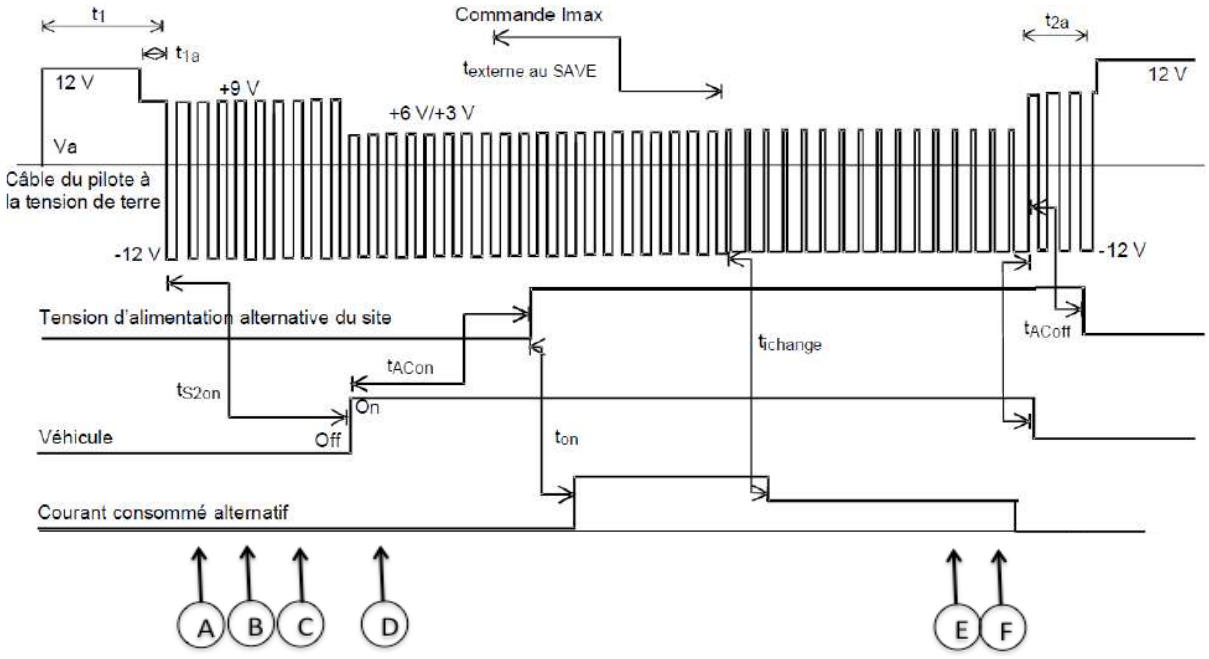


Figure 4: Dialogue en modulation de largeur d'impulsion entre la borne et le véhicule

2 Travail à effectuer

Le cahier des charges précédemment donné représente les besoins d'un client (le constructeur des bornes 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 liées 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

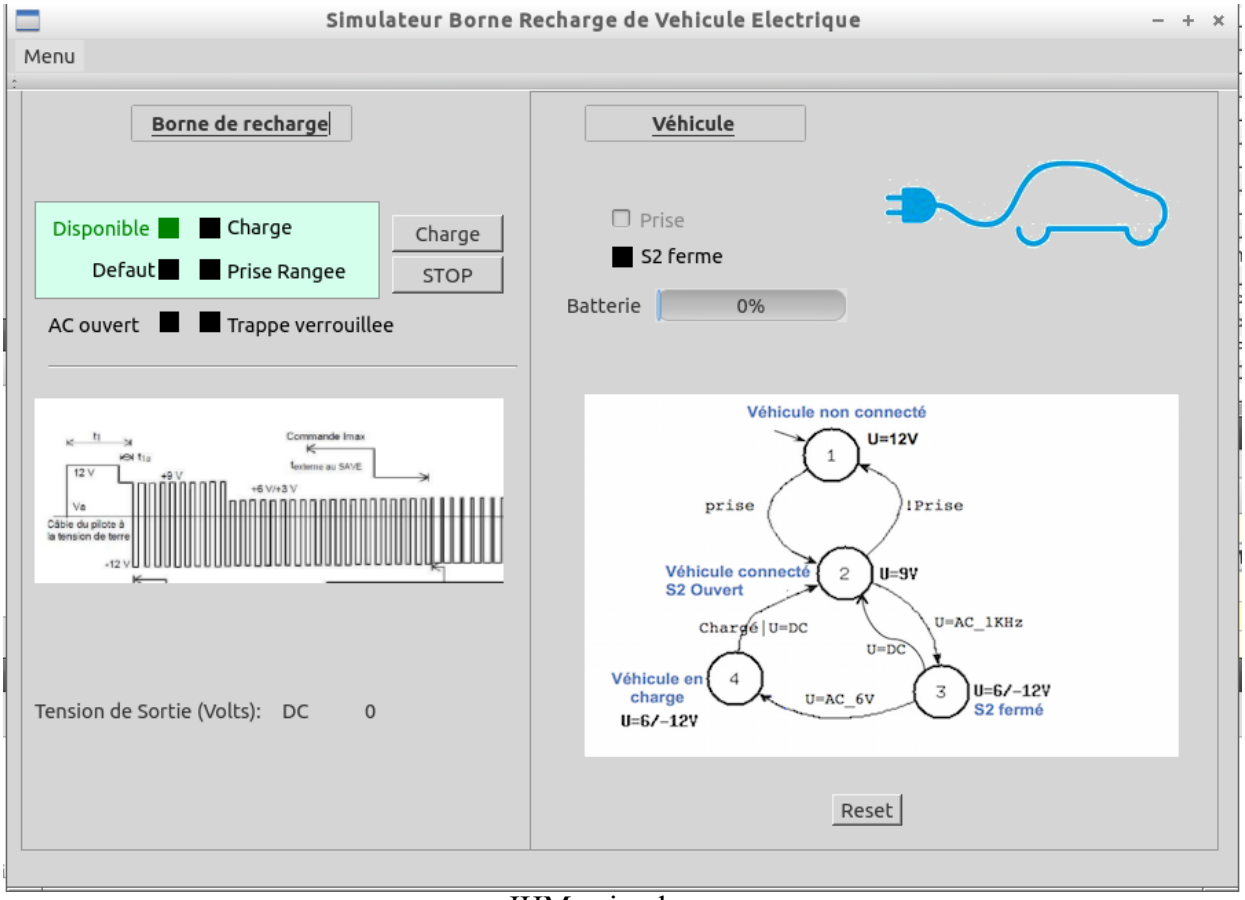
La mise en oeuvre du projet de gestion d'une borne de recharge de véhicule électrique fait appel d'une part à un simulateur logiciel comportant la partie gauche représentant la borne de recharge à piloter et la partie droite affichant l'état simulé d'un véhicule en charge sur la borne et d'autre part 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_defaut;
  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 déclanchement du timer dans un attribut et une méthode `timer_valeur()` renvoyant la différence entre l'horloge actuelle et l'heure du déclanchement.

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, elle est déjà développée et l'interface disponible dans `lcarte.h` (ne pas la saisir, le include suffit).

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 compléterez 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 », « reprendre véhicule » et « administrer borne ».

- 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 débuter 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 ou`
  Ou clonez le dépôt git avec la commande suivante :
  `https://github.com/pascalberthou/projetuml-borne.git`
- 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`. Executez par la commande `./borne`.
- Commencez à implémenter le premier UC en suivant le diagramme de collaboration. Faite 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). Faite 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 : OUTILS POUR LA MODELISATION UML

MagicDraw : Outil de la societe no magic, racheté par Dassault. Version démonstration gratuite suffisante. https://www.magicdraw.com

Modelio : Outil avec une partie opensource suffisante pour le cours UML. Excellent outil aussi. https://www.modeliosoft.com/fr/
Tutoriel modelio : https://github.com/ModelioOpenSource/Modelio/wiki/Quick-Start-Guide

Umbrello : Outil un peu ancien mais libre sous Linux. Pas forcément le meilleur choix. https://apps.kde.org/fr/umbrello/

Draw io : Outil de dessin uniquement, ne permet pas de profiter complètement des abstractions et des vérifications de la cohérence des diagrammes.
https://www.drawio.com/blog/uml-class-diagrams

Principes d'utilisation des outils de modélisation UML :

Ces outils sont généralement basés sur le même principe de projet. La première étape étant de créer le projet en choisissant le langage de modélisation (UML). Ensuite apparaît un écran découpé en plusieurs parties dont au moins : un explorateur de modèle (à gauche (1)), une vue du diagramme en cours (la plus grande (2)), les propriétés de l'élément sélectionné (en général en dessous (5)). L'objectif est de commencer à créer un diagramme en choisissant son type et il apparait dans l'explorateur et la fenêtre centrale. Les éléments de modélisation apparaissent et sont à faire glisser dans le diagramme. Ils sont affichés aussi dans l'explorateur. Quand désiré, démarrer le diagramme suivant de la même manière, mais cette fois, essayer d'utiliser au maximum les artefacts déjà crées (par exemple une classe) en la faisant glisser dans le nouveau diagramme (par exemple un diagramme de séquence). Bien évidemment il faudra rajouter aussi de nouveaux éléments, mais la réutilisation des artefacts permet de créer une cohérence entre les diagrammes.

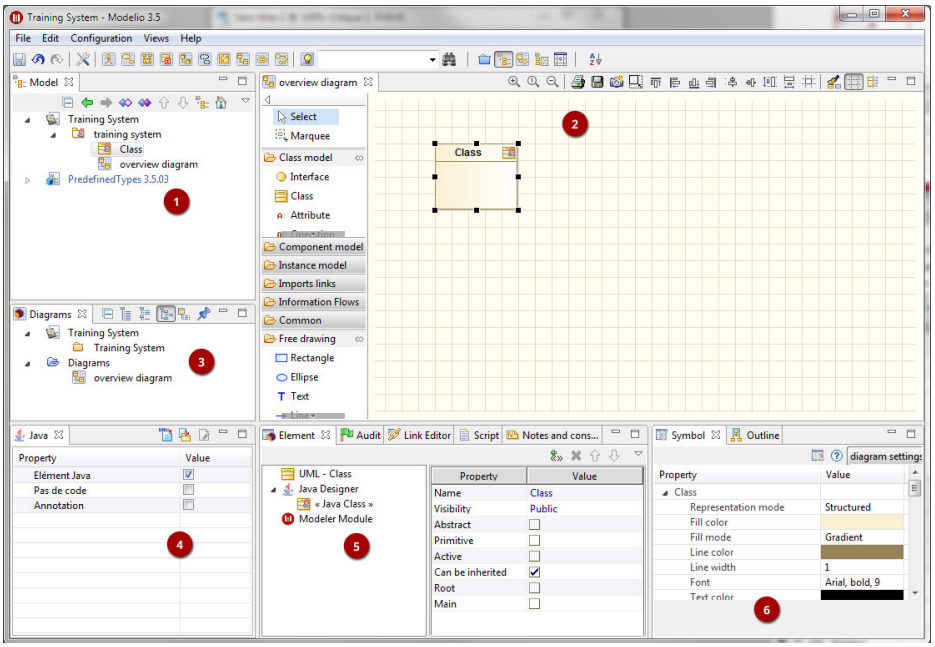


Figure 1 -Vue générale de l'outil Modelio

ANNEXE 2 : INTRODUCTION A GITHUB

Git est l'outil moderne de gestion de version et de configuration le plus utilisé aujourd'hui. C'est une bonne idée de se familiariser avec en commençant à développer des petits projets. GitHub est une plateforme qui met à disposition un serveur Git et de nombreux outils facilitant la gestion de projet. Voir les vidéos disponibles sur moodle présentant l'outil.

Rapide tutorial Github pour le projet

1/ Créer son compte sur Github (https://github.com)

2/ Valider son compte pour github éducation pour profiter des outils tels que copilot dans vscode (https://docs.github.com/fr/education/explore-the-benefits-of-teaching-and-learning-with-github-education/github-education-for-students/apply-to-github-education-as-a-student)

3) Créer son projet Git, dans https://github.com/dashboard, en cliquant sur **new**

4) Importer le dépôt squelette de TP (Import a repository.) et préciser l'url du dépôt à copier (https://github.com/pascalberthou/projetuml-borne.git)

A partir de maintenant tout est fait sous Linux en lignes de commande.

5) Configurer son compte git sur la machine :

```
git config --global --add user.name "Pascal Berthou"
git config --global --add user.email "pascal.berthou@laas.fr"
```

6) Configurer la machine pour accéder en lecture/ecriture au dépôt distant sur github

- générer une clé ssh sur la machine linux : ssh-keygen et regarder le contenu du fichier généré
- ajouter cette clé dans votre profil (https://github.com/settings/keys) dans Authentication keys en copiant.collant le contenu du fichier (commence par SSHA256)
- tester si cela fonctionne : ssh -T git@github.com (https://docs.github.com/fr/authentication/connecting-to-github-with-ssh/testing-your-ssh-connection)

7) Créer son répertoire de travail et récupérer les sources de son projet qui vient d'être cloné

```
git clone https://github.com/VOTRE-USERNAME-GITHUB/VOTRE-PROJET-GITHUB.git
```

8) vérifier la présence des fichiers en allant dans le répertoire et ls

```
$> ls
Docs     README.md    lecteurcarte.c
Makefile borne.c      lecteurcarte.h
```

9) Travailler normalement en ajoutant et modifiant les fichiers

10) Ajouter les nouveaux fichiers au dépot

git add monfichier.c monautrefichier.c

11) Valider localement les modification (ici les sauvegarde ne sont que locales)

git commit -m " un message explicite "

12) Indiquer (la première fois seulement) quel est le dépot distant sur github

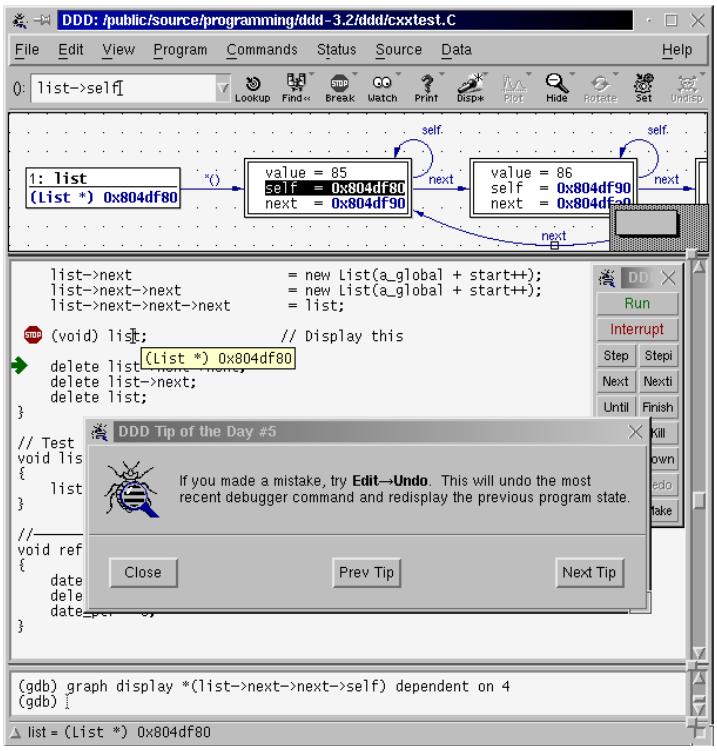
```
git remote set-url origin git@github.com:VOTRE-USERNAME-GITHUB/VOTRE-PROJET-GITHUB.git
```

13) Envoyer les modifications sur le dépôt github

git push

Si tout est bien configuré il ne devrait pas y avoir de message d'erreur !

ANNEXE 3 : 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[1] :

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

[1] 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.

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