BTS I.R.I.S

Informatique et Réseaux pour l'industrie et les Services techniques

Lycée de la Tourelle

M.BORDATO,M.LEGALL,M.POL,.M.TOMCZAK,M.VILLERS

Système de gestion de parking – Decma Rep

Nom de l'étudiant1:TEP
Nom de l'étudiant2:JEANS-LOUIS
Nom de l'étudiant3:MENDES

Sommaire

1. Présentation

Nom de l'étudiant1:TEP
Nom de l'étudiant2:JEANS-LOUIS
Nom de l'étudiant3:MENDES

1.1_Situation du projet dans l'entreprise

De nos jour les systemes de gestion de parking est omniprésent, on les trouve un peu partout. Notre système sera utilisé pour gérer le parking d'une entreprise afin de faciliter le quotidien des salariés.

1.2_But du projet

Le but de ce projet est d'automatisser le syteme de gestion des accès à des parking automobiles.Il doit permettre d'autoriser l'entrée et la sortie sous certaines conditions par commande de la barriere.

Mais aussi d'afficher l'état du système sur un poste dit de supervision. Ce poste distant sera prevu afin de pouvoir consulter les même données via un réseau ethernet et aussi d'agir sur le systeme(ouverture et fermeture de la barriere).

Nous pourrons de ce fait consulter des informations telles que le nombres de places libres dans le parking.

1.3Fonctionnement du syteme de gestion de parking

-Nous avons deux boucle permetant de détecter la presense de vehicule

-Une carte "micro-controleur" TINI (Tiny Interface de chez Dallas semi Conductor) possédant un environnement temps réels qui se programme en java.

Elle communique avec un serveur qui a pour fonction la supervision du systeme(via Ethernet) et sa commande.

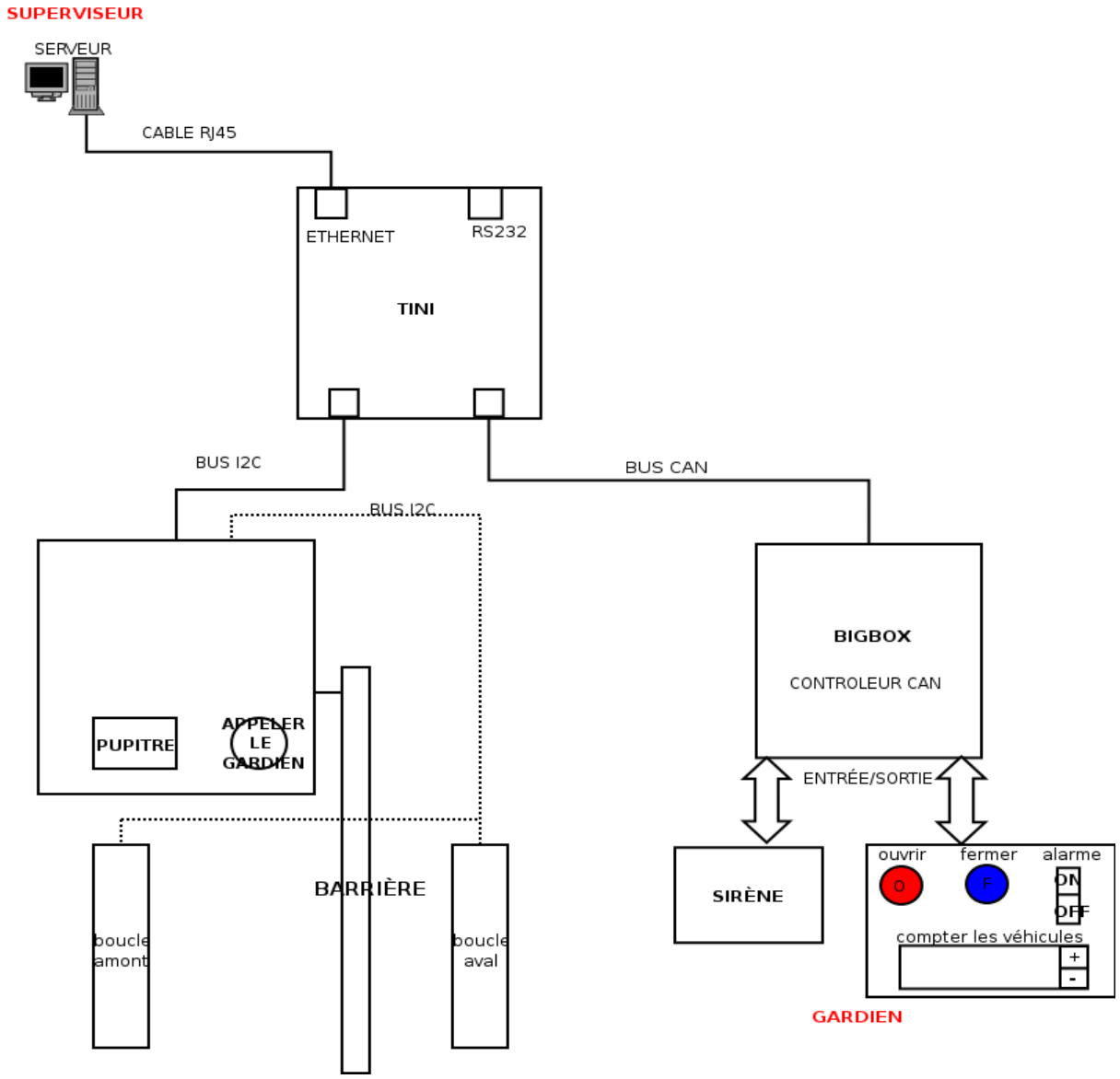
-Les cartes électroniques dédiés à l' application via le bus I2C:PCF8574(tampon 8bits I2C), afficheur,clavier

-Un boitier Big Box équipé d'un controleur CAN 32 Entrée/Sortie.Ces Entrée/Sortie sont composées de boutons poussoire et d'une sirène.Ce boitier est destiné à controler directemen,t la barriere par le gardien.L boitier est relié à la carte TINI à travers un bus CAN.

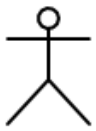
2. Etude et modélisation UML

2.1_Schemat structurel

SCHEMA STRUCTURELLE

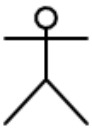


2.2_Use case général



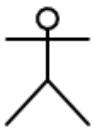
Conducteur de l'automobile

Personne désirant entrer ou sortir du parking



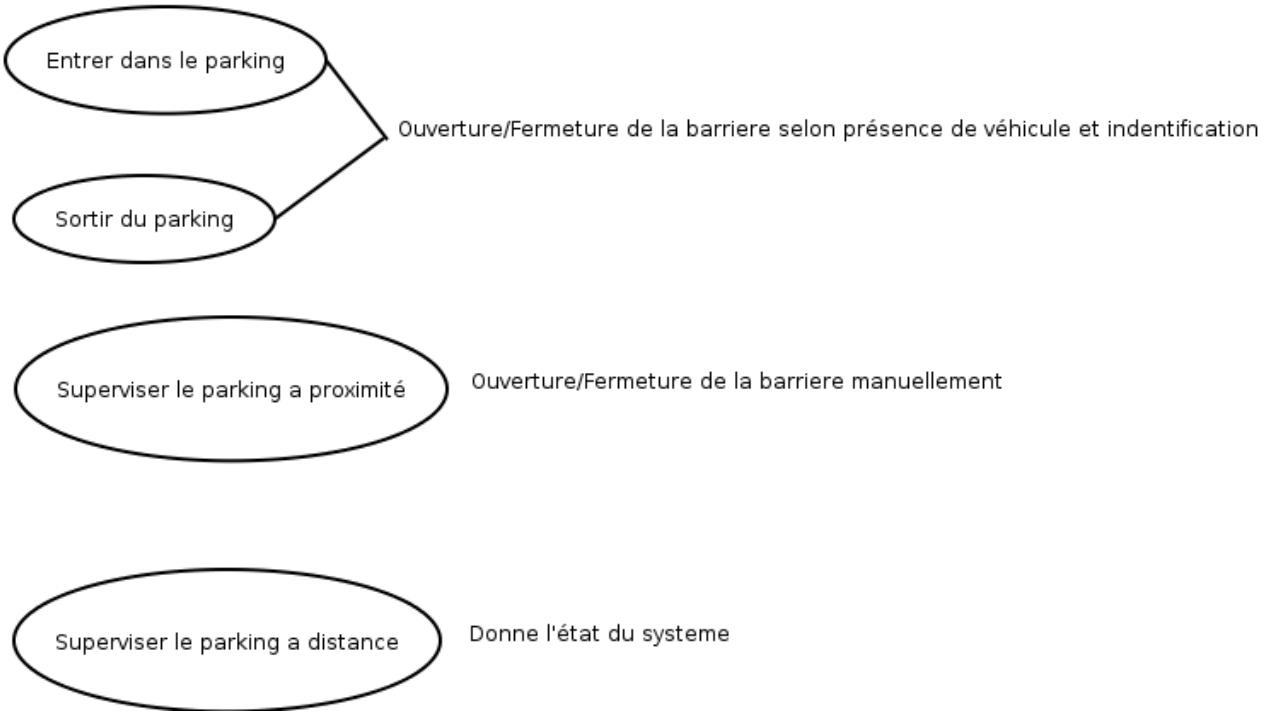
Superviseur

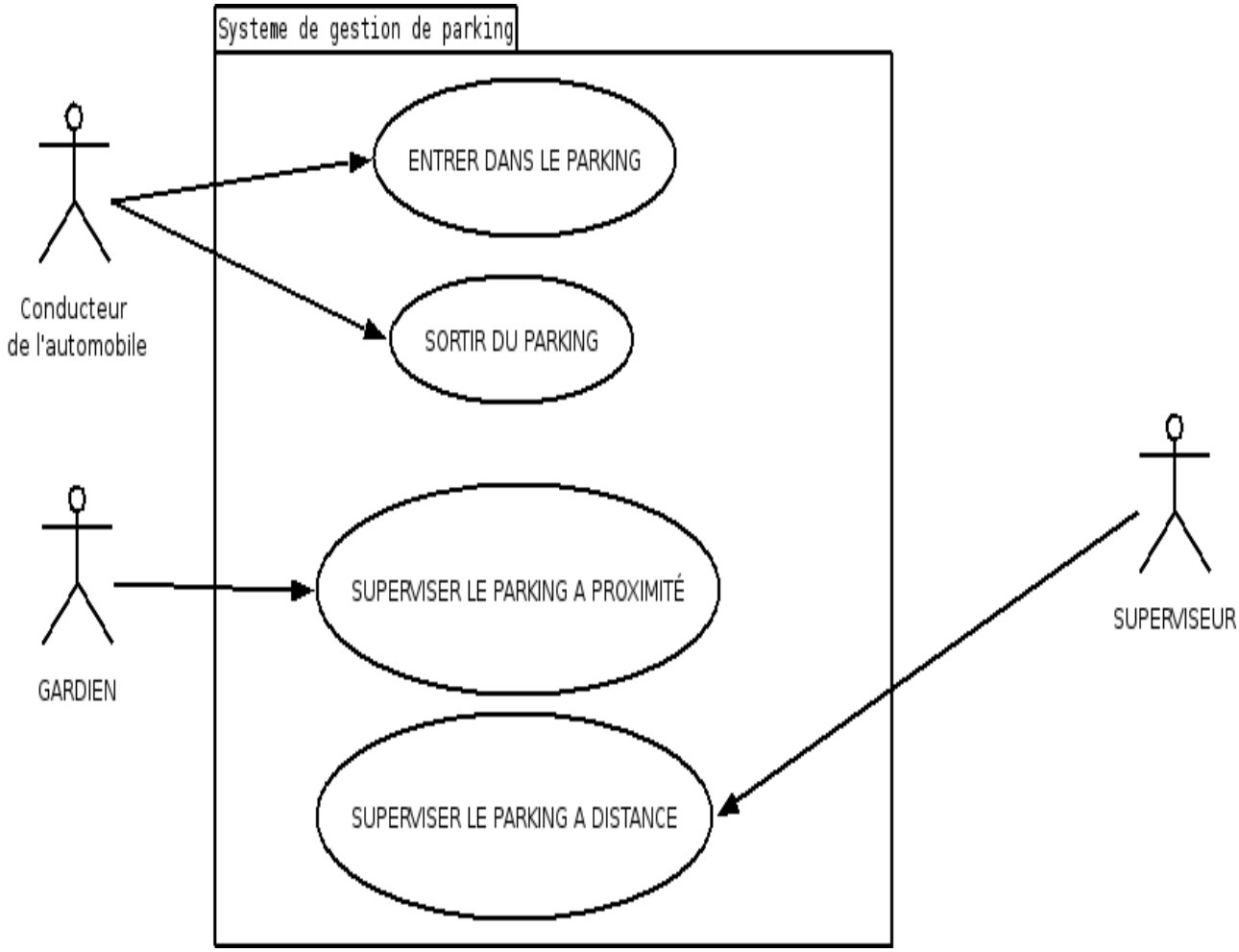
Supervise l'état du systeme via un reseau ethernet



Gardien

Active manuellement la montée ou la descente de la barriere

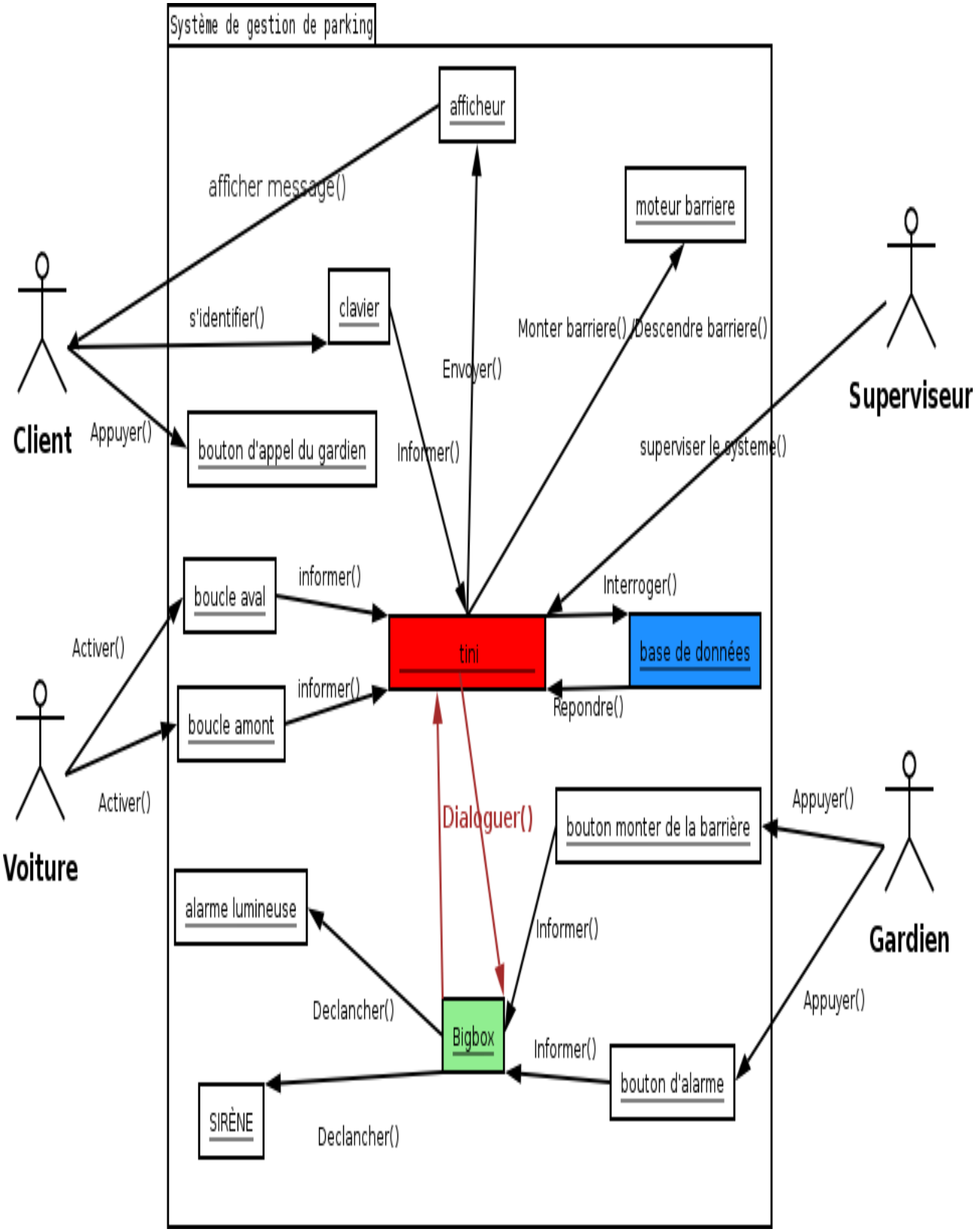




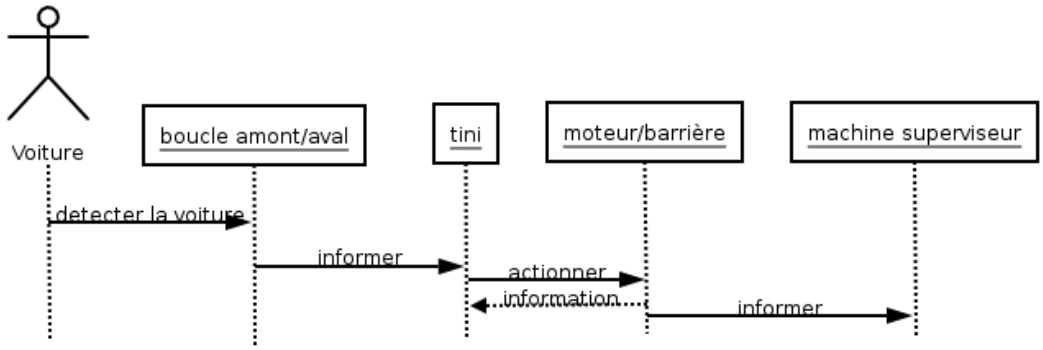
2.3_Diagramme de contexte élaboré

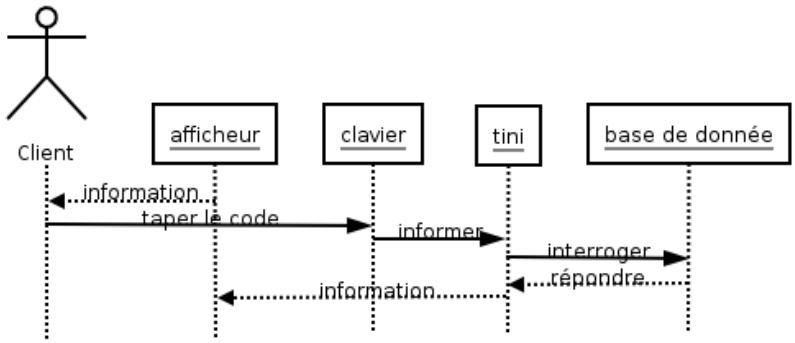
Le diagramme de contexte est une représentation schématique faisant le lien entre le diagramme de cas d'utilisation et le diagramme de class.

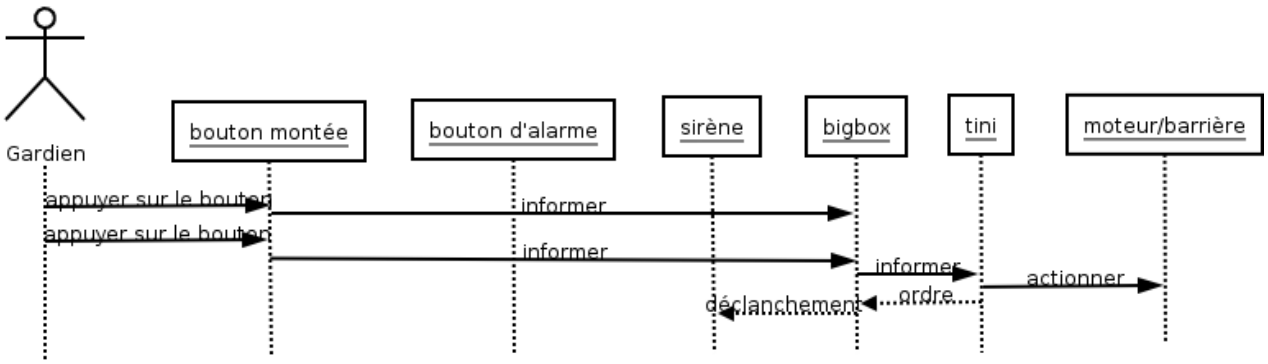
On y represente l'intéraction entre les acteurs et les composant du systeme ainsi que les message envoié entre eux.

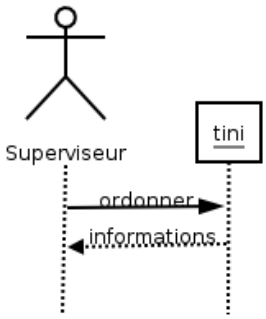

Système de gestion de parking
afficheur
afficher message()
moteur barriere
Client
s'identifier()
clavier
Monter barriere()
Descendre barriere()
Envoyer()
Superviseur
Appuyer()
bouton d'appel du gardien
Informer()
superviser le systeme()
boucle aval
informer()
Interroger()
Activer()
tini
base de données
informer()
boucle amont
Repondre()
Activer()
Dialoguer()
Appuyer()
bouton monter de la barrière
Voiture
alarme lumineuse
Informer()
Gardien
Declancher()
Bigbox
Informer()
Appuyer()
bouton d'alarme
SIRÈNE
Declancher()

2.4 Diagramme de sequence









2.5_Scénarios

Description textuelle du CU Entrée dans le parking

TITRE: Entrée dans le parking.
ACTEUR: Conducteur de l'automobile.
RESUME: Ce CU permet au conducteur de l'automobile d'entrer dans le parking

Description du scénario:

Scénario nominal.

1. Une voiture se présente devant la boucle amont
2. Le système vérifie le nombre de places dans le parking
3. Le système demande son code
4. Le conducteur de l'automobile tape son code d'accès
5. Le système compare le code avec celui qui se trouve sur sa base de donnée
6. Le système ouvre la barrière
7. La voiture se présente devant la boucle aval
8. Le système referme la barrière
9. Le système incrémente le nombre de voiture dans le parking

Enchaînements alternatifs

A1:code d'accès faux
L'enchaînement commence au N°6 du scénario nominal (noté SN)

6. Le système indique au conducteur de l'automobile que le code est faux, pour la première ou la deuxième fois
7. Le système enregistre l échec

Le scénario est reprise au N°3

Enchaînement d'erreurs
E1: plus de places dans le parking

3. Le système affiche que le parking est plein
   Le CU est terminé.

E2: code d'accès définitivement faux

6. Le système affiche que le code d'accès est faux pour la 3ème fois
7. Le système affiche « appeler le gardien »
8. Le conducteur de l'automobile appuie sur le bouton pour appeler le gardien

Description textuelle du CU Sortie du parking

TITRE: Sortie du parking.
ACTEUR: Conducteur de l'automobile.
RESUME: Ce CU permet au conducteur de l'automobile de sortie du parking

Description du scénario:

Scénario nominal.

1. La voiture se présente sur la boucle aval
2. Le système ouvre la barrière
3. la voiture se présente sur la boucle amont
4. Le système referme la barrière
5. Le système décrémente le nombre de voiture dans le parking

Description textuelle du CU Superviser le parking à proximité

TITRE: Superviser le parking à proximité
ACTEUR: Gardien
RESUME: Ce CU permet au gardien d'ouvrir/de fermer la barrière et de déclenché/arrêter la sirène

Description du scénario:

Scénario nominal.

Utilisation en cas d'urgence

1. Le gardien appui sur le bouton de l'alarme
2. La sirène d'alarme se déclenche
3. La barrière s'ouvre

Utilisation manuelle du système

1. Appuyer sur le bouton de montée de la barrière
2. Appuyer sur le bouton de descente de la barrière

Description textuelle du CU Superviser le parking à distance

TITRE: Superviser le parking à distance
ACTEUR: Superviseur
RESUME:

Description du scénario:

Scénario nominal.

Utilisation en cas d'urgence

1. Le superviseur active l'alarme
2. La sirène d'alarme se déclenche
3. La barrière s'ouvre

Utilisation manuelle du système

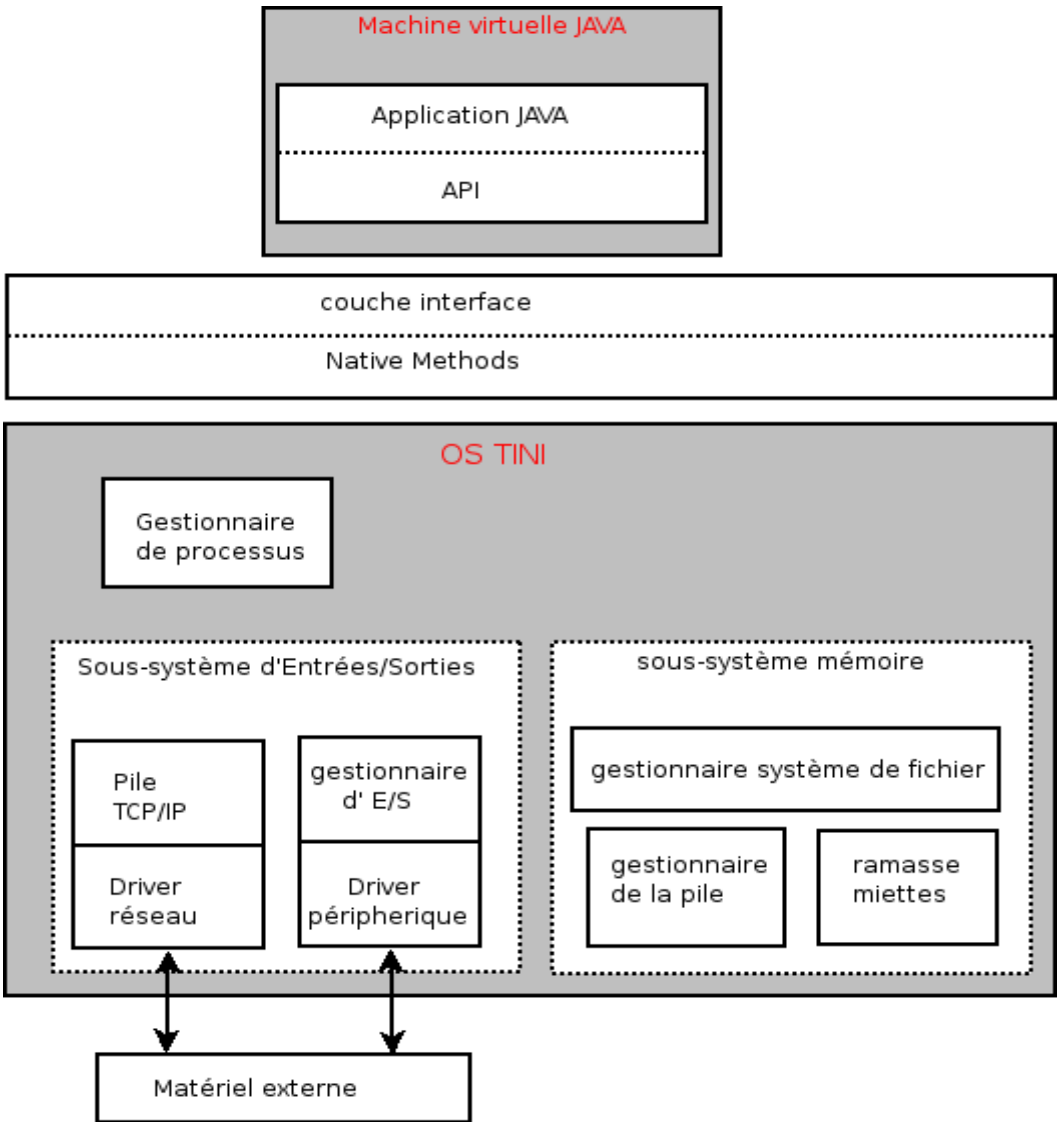
3. Appuyer sur le bouton de montée de la barrière
4. Appuyer sur le bouton de descente de la barrière

3.Environnement matériel et logiciel

3.1 Micro-controleur TINI

La carte TINI (Tiny InterNet Interface) est un module « intelligent » fonctionnant sous JAVA et permettant de dialoguer avec le monde extérieur.
Elle se branche sur une carte de support comportant toute la connectique nécessaire à la communication et à son fonctionnement (alimentation, connecteur ethernet, série ...).

Environnement logiciel



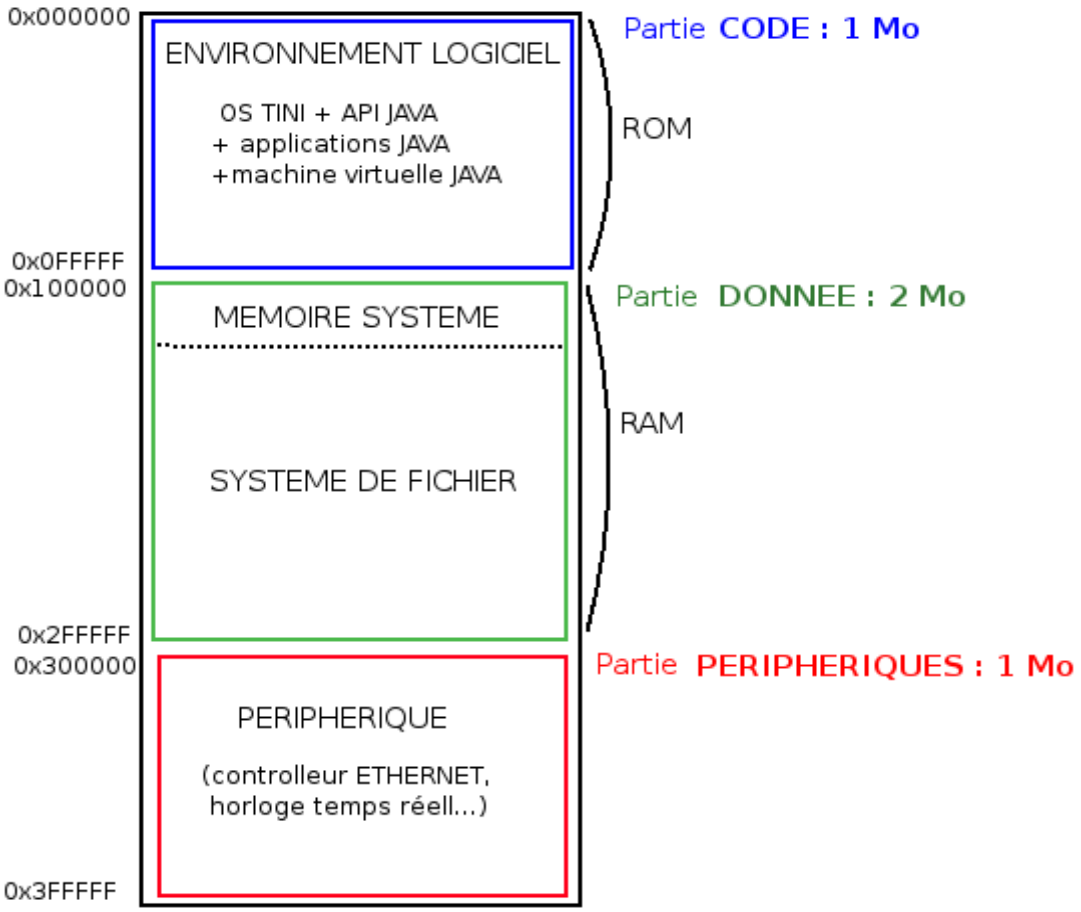
l'OS TINI est le système d'exploitation de la TINI, il est principalement constitué de :

- le sous système mémoire : qui regroupe le gestionnaire du système de fichier, le gestionnaire de la pile ainsi que le ramasse-miettes (garbage collecteur en anglais).

- le sous système d'Entrées/Sorties : il est divisé en deux parties:
  - une pour le réseau Ethernet qui contient la pile TCP/IP
  - une pour les autres périphériques qui contiennent le gestionnaire d'Entrées/Sorties.
- le gestionnaire de processus.

Mais les applications JAVA ne sont pas directement compréhensible par l'OS TINI.
IL existe des « Native Méthodes » pour permettre au API d'exploiter le matériel. En effet le code généré par la machine virtuelle JAVA n'est pas compatible avec la partie Hardware, nous avons donc besoin de ces « Native Méthodes ». Elles permettent aussi d'accéder et de configurer les ressources systèmes telle que le watchdog ou l'horloge temps-réelle.
Le lien entre les « Native Méthodes » et la machine virtuelle JAVA est fait au niveau de la couche interface.

Espace adressable :



3.2_Module de comunication CAN/BigBox

PRESENTATION :

Avec son contrôleur 16bits du MC 68332 et plus de 512kb de mémoire EPROM et 1MB de RAM le CAN-BIGBOX peut gérer plusieurs applications. L'emploie de la mémoire flash EPROM permet à l'utilisateur de télécharger ses propres programmes grâce à la liaison série RS232.

Le CAN-Bigbox fournit:

*8 entrées digitales:0-7

*8 sorties digitales : 8-15

Bus série RS 232 :

Utilisé pour télécharger, debugger ou changer les paramètres système : utilisation du serial connexion interface (sci) du MC 68332.

EPROM (NM93CS46): pour sauvegarder les informations spécifiques de bord

Capacité 1kbits.Le Chip Select de l'EEPROM est contrôlé par le bit 7 du registre de control.

3.3_Serveur superviseur

Materiel

Serveur

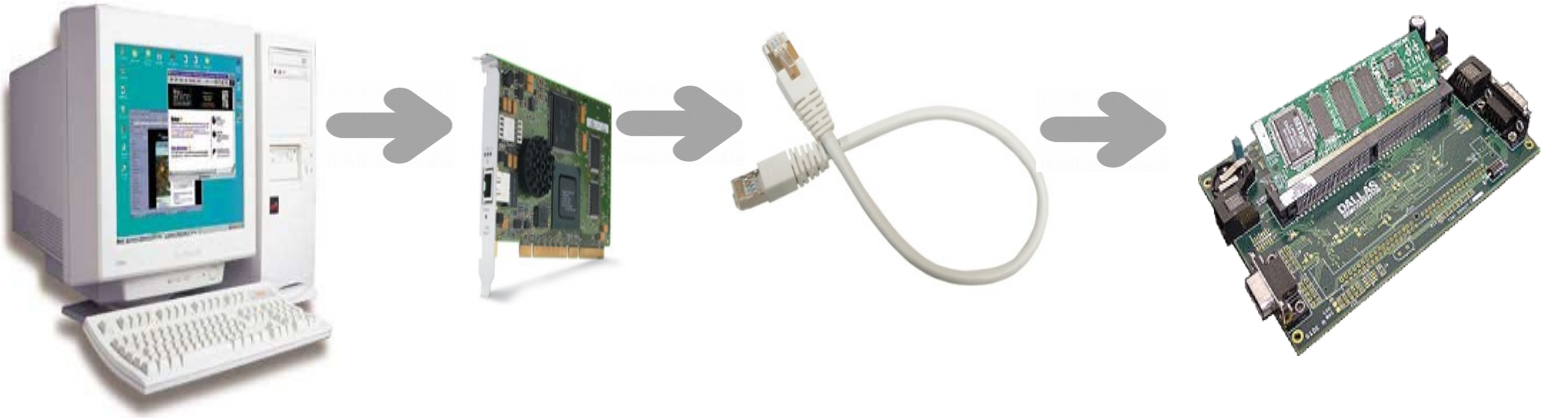
Le serveur superviseur est une ordinateur informatique, qui possède un environnement Windows XP. On l'appel serveur car il partage des ressources avec le micro-controleur.

Ces ressources seront donc partagée a l'aide d'un réseau Ethernet qui est un réseau local. Le serveur superviseur, équipé d'une carte réseau, sera relié au support du micro-controleur à l'aide d'un cable réseau de type RJ45.

Carte réseau

Cable RJ45

Le schéma ci dessous, pour résumer, est une représentation du materiel mis a notre disposition, ainsi que leur liaisons.



Serveur superviseur	Carte réseau	Cable réseau RJ45	Micro-controleur TINI et support

La carte réseau se situe dans l'unité centrale du serveur superviseur. Le micro-controleur est connecté à un support qui lui permet d'avoir une connection Ethernet. Grâce a ces deux materiaux, le serveur et le micro-controleur disposent tous deux d'une connexion Ethernet. On utilise donc un câble réseau afin de les reliés.

.

4.Realisation technique

4.1 Répartition des taches entre les candidats

Candidat 1

T1.1 S'approprier le cahier des charges du système

T2.1 Finaliser la modélisation du système

T2.2 Définir un protocole d'échange

T3.1 Coder le module qui gère les évènements du système

T3.2 Coder le module qui gère les états de la borne

T4.1 Tester et valider le module module qui gère les évènements du système

T4.2 Tester et valider le module module qui gère les états de la borne

T5.1 Recette l'installation finale

T6.1 Installer et configurer le S.E et s'approprier les outils de développement

T7.1 Gérer la planification

T7.2 Assurer la traçabilité

T8.1 Rédiger les documents relatifs au projet

Candidat 2

T1.1 S'approprier le cahier des charges du système

T2.1 Finaliser la modélisation du système

T2.2 Définir un protocole d'échange

T3.3 Coder le module de communication CAN avec la BIGBOX

T4.3 Tester et valider le module de communication CAN avec la BigBox

T5.1 Recetter l'installation finale

T6.1 Installer et configurer le S.E. et s'approprier les outils de développement

T7.1 Gérer la planification

T7.2 Assurer la traçabilité

Candidat 3

T1.1 S'approprier le cahier des charges du système

T2.1 Finaliser la modélistion du système

T2.2 Définir un protocole d'échange

T2.3 Définir l'interface Homme/Machine de supervision

T3.4 Coder le module de l'IHM de supervision

T3.5 Coder le module de communication entre l'IHM et la TINI

T4.4 Tester et valider le module de l'IHM de supervision

T4.5 Tester et valider le module de communication entre l'IHM et la TINI

T5.1 Recetter l'installation finale

T6.1 Installer et configurer le S.E et s'approprier les outils de développement

T7.1 Gérer la planification

T7.2 Assurer la traçabilité

T7.3 Rédiger les documents relatifs au projet

4.2 Protocole d'échange

Un protocole d'échange est une façon de s'envoyer des informations d'un système à un autre pour que cette information soit compréhensible par le receveur et donc exploitable.
Entre le Candidat 1 et le candidat 3 . l'échange des informations se fera à travers un serveur-client RPC.

CANDIDAT 1

4.3_Candidat 1 : TEP Samnang

Nom de l'étudiant1:TEP
Nom de l'étudiant2:JEANS-LOUIS
Nom de l'étudiant3:MENDES

1.Introduction

Pour faciliter la mise en œuvre du projet, différentes taches ont été déclinée en fonction des candidats. Ma participation au projet se résume en deux parties distinctes :

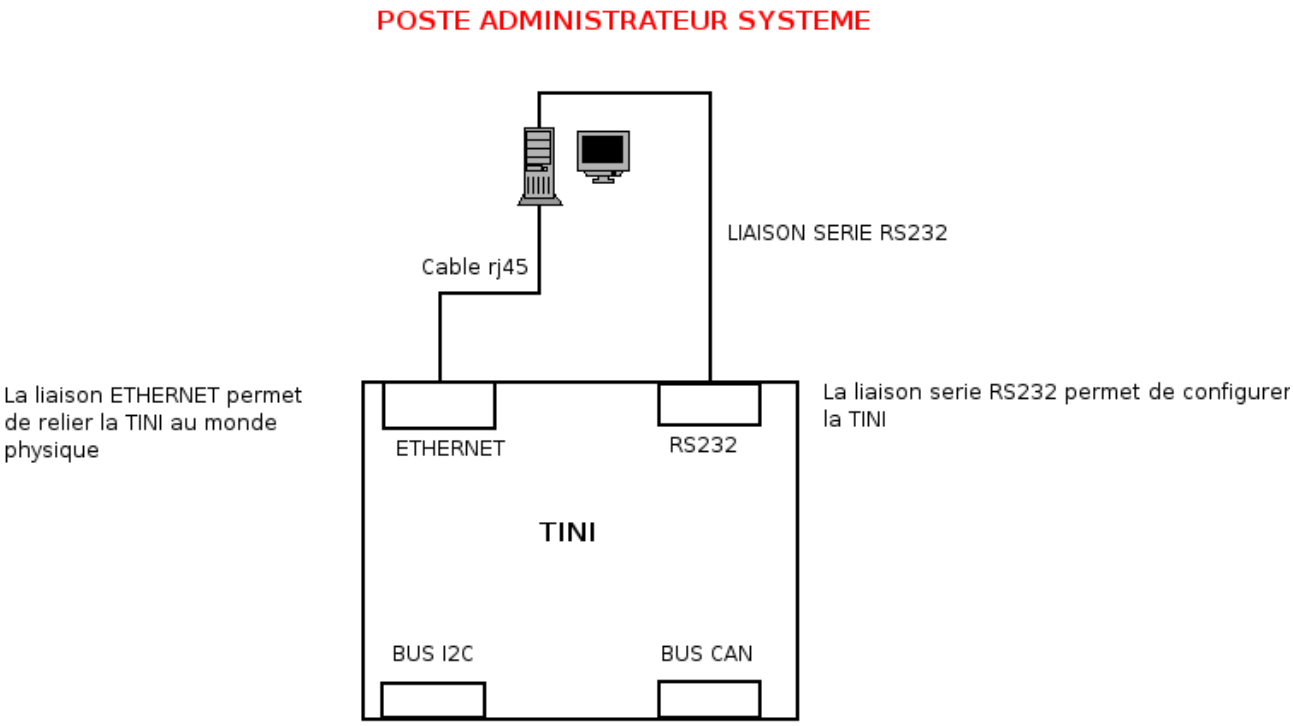
- **Coder le module qui gère les évènements du système**

Le but de ce module est de gérer les événement du système, c'est-à-dire de lire le les états des capteurs présents. Une foi cette étape effectuée, l'information est traitée puis transmis au candidat 3

- **Coder le module qui gère les états de la borne**

Le but de ce module est d'établir une communication entre le bus I2C et le micro-contrôleur TINI. Cette communication permettra ainsi de positionner les composants du système (monter et descendre la barrière , éclairage de l'afficheur etc.) en fonction des besoins du candidat 2 et 3. C'est donc grâce à ce module que les actions nécessaires seront exécutées et qu'il pourra connaître à tous moments l'état du système.

2. Schéma structurel de ma partie



3. Analyse UML

diagramme de séquence enrichie du cas d'utilisation entrer dans le parking

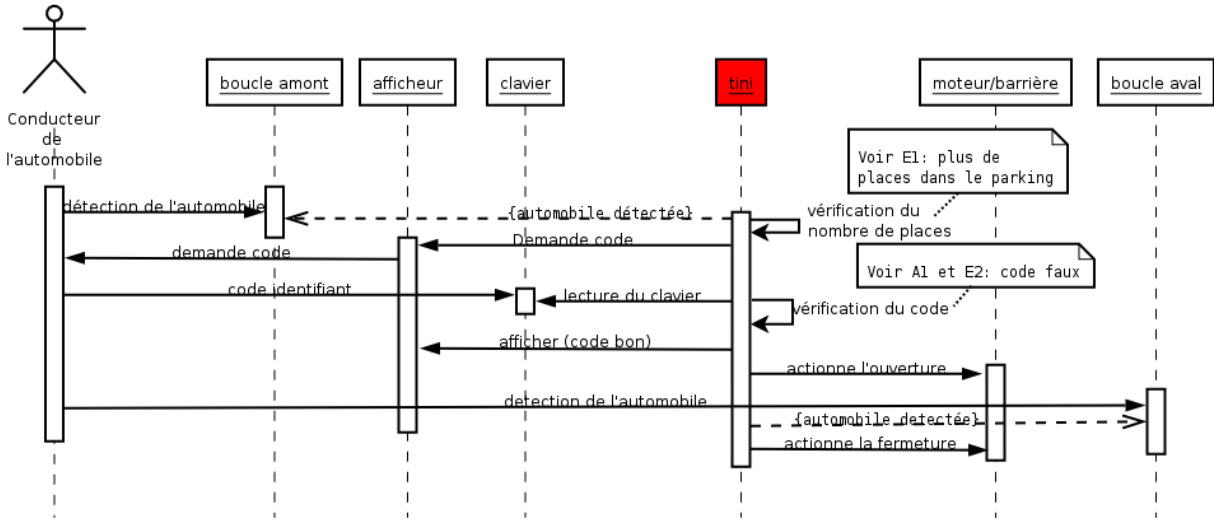
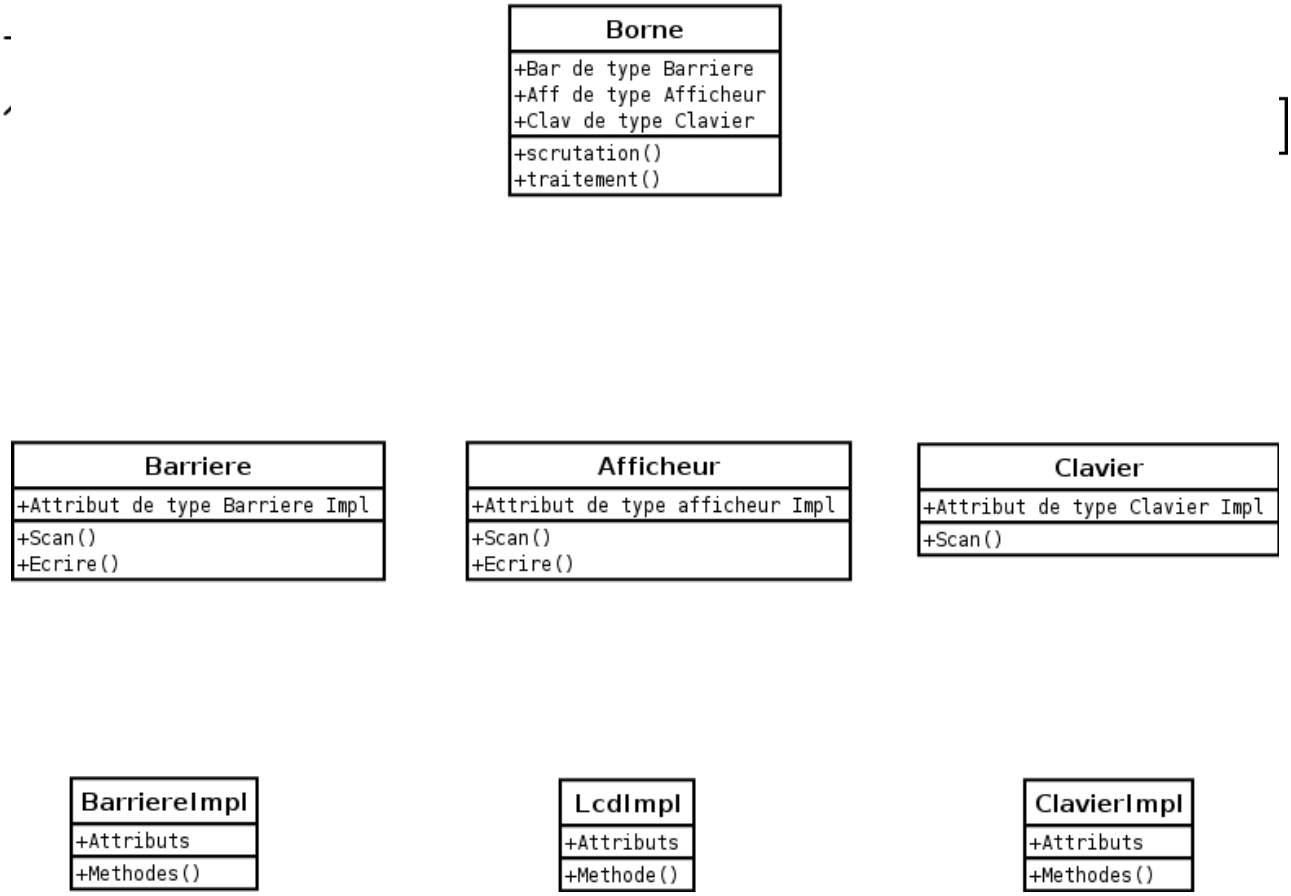


Diagramme de class



Le cahier des charges nous a fournir trois classes :

-La classe **BarriereImpl**

-La classe **LcdImpl**

-La classe **ClavierImpl**

A partir de ces trois classes j'ai créé trois autre classes :

-Une classe *Barriere* avec un attribut de type **BarriereImpl** afin de pouvoir utiliser les méthodes de celui-ci et une méthode *scan()* qui va lire les Entrées/Sorties de la barriere de la borne et une méthode *ecrire()* qui va commander les Entrée/Sorties.

-Une classe *Afficheur* avec un attribut de type **LcdImpl** afin de pouvoir utiliser les méthodes

de celui-ci et une méthode *scan()* qui va lire les Entrées/Sorties de l'afficheur de la borne et une méthode *ecrire()* qui va envoyer un message sur l'afficheur.

-Une classe *Clavier* avec un attribut de type **ClavierImpl** afin de pouvoir utiliser les méthodes de celui-ci et une méthode *scan()* qui va lire les Entrées/Sorties du clavier de la borne.

Puis une classe *Borne* c 'est la class principale qui possède des attributs de type *Barriere*,*Afficheur* et *Clavier* mais aussi deux threads :

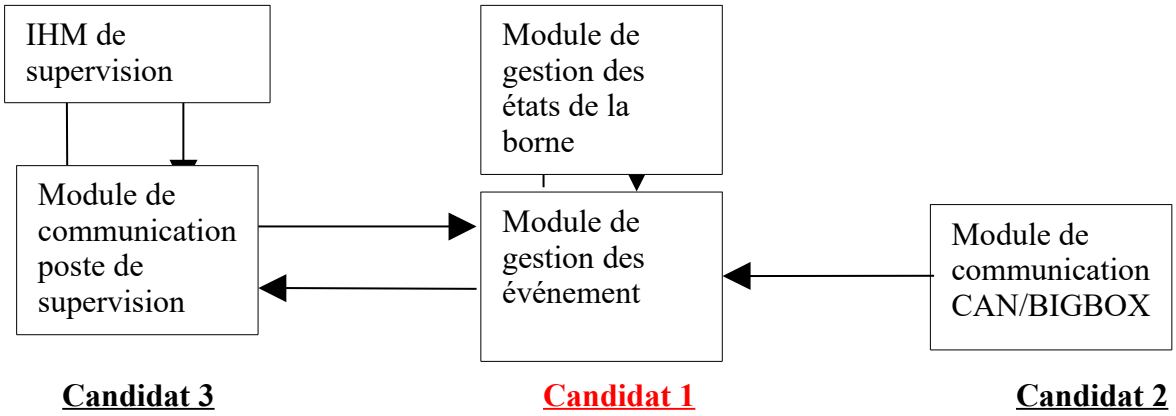
-***Scrutation*** qui uliser la méthode *scan()* de chacune de ces classes afin lire en permanence l'état des Entrées/Sortie de la borne et de mettre ces évènements dans une queue de liste.

-***Traitement*** qui va récupérer ces évènements dans la liste et les traiter un par un.

4. Situation dans le groupe de travail

Etant chargé de l'automatisation du système, je serai le technicien qui agira directement sur le micro-contrôleur TINI . Je devrais donc travailler en étroite collaboration avec le candidat numéro2 et 3 qui devront être au courant de l'avancer de mes travaux tout au long du projet. Par ailleurs, un protocole d'échange devra être crée au préalable en vue d'une intégration future avec le candidat 3.

5.Le protocole d'échange



6.Installation de l'environnement matériel et logiciel

Mise en route du micro-contrôleur TINI (pour Windows)

Pour pouvoir utiliser la TINI, il ne suffit pas d'avoir le matériel il faut aussi avoir les composants logiciel nécessaires à sont fonctionnement.

Il y en a trois :

-**Un environnement de développement JAVA** : il doit contenir la machine virtuelle JAVA , le compilateur, ainsi que les API. On peut utiliser le **JDK** (Java Development Kit) de Sun Micro-systems, téléchargeable sur le site http://java.sun.com .

Remarque : *avec JDK il est nécessaire d'ajouter le répertoire c:\<répertoire du jdk>\bin au chemin d'exécution, c'est-à-dire à la variable PATH.*

- Les API communication JAVA (Comm API): ils permettent d'utiliser le port RS232 de la TINI,nécessaire au chargement de l'environnement logiciel.

On peut les télécharger sur le site : http://java.sun.com/products/javacomm/index.html.

- Le SDK (Software Development Kit) : il contient les programmes nécessaires au fonctionnement et à la configuration de la TINI tel que le Slush qui est le shell (interpréteur de commande) de la TINI.
On peut le télé charger sur le site : http://ibutton.com/TINI/software/index.html.

Un fois ces composants installés il faut faire une série de copie de fichier :
- copier le fichier **c:\commapi\comm.jar** dans le répertoire **c:\< JDK >\jre\lib\ext** , le fichier **c:\commapi\javax.comm.properties** dans le répertoire **c:\< JDK >\jre\lib** et le fichier **c:\commapi\win32comm.dll** dans le répertoire **c:\< JDK >\jre\bin** .

On doit maintenant charger l'environnement logiciel dans la TINI.Cela consiste en 2 étapes :
- charger les fichiers **tini.tbin** et **slush.tbin** qui se trouvent dans c:\< *SDK* >\bin.
- Initialiser la pile.

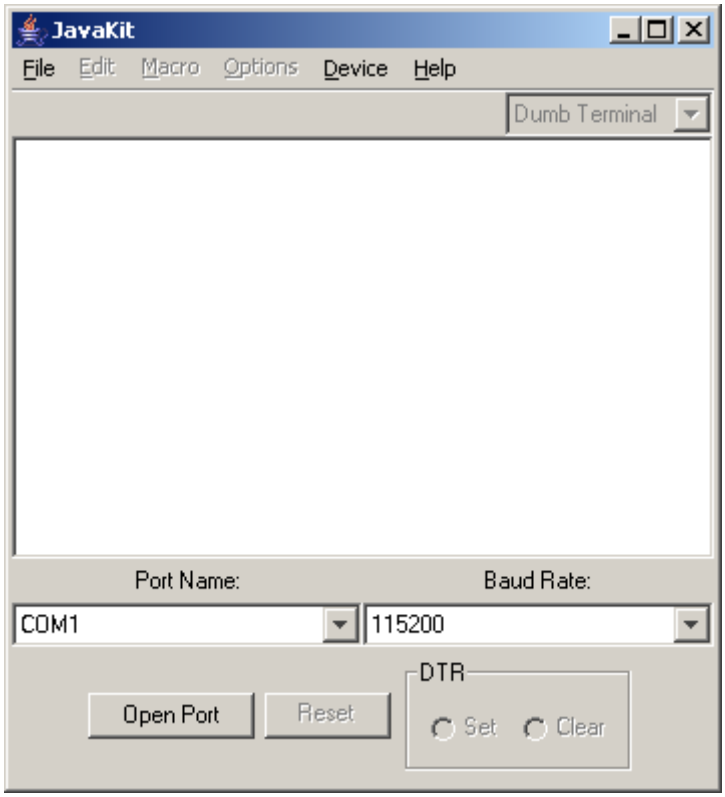
Pour cela on doit utiliser un programme, JavaKit, qui se trouve dans le SDK. Mais pour utiliser JavaKit on doit ajouter le fichier **c:\< *SDK* >\bin\tini.jar** au chemin d'exécution, c'est-à-dire à la variable PATH. On peut le faire en ouvrant un invite de commande DOS et taper la commande suivante :

SET CLASSPATH=c:\< *SDK* >\bin\tini.jar;%CLASSPATH%

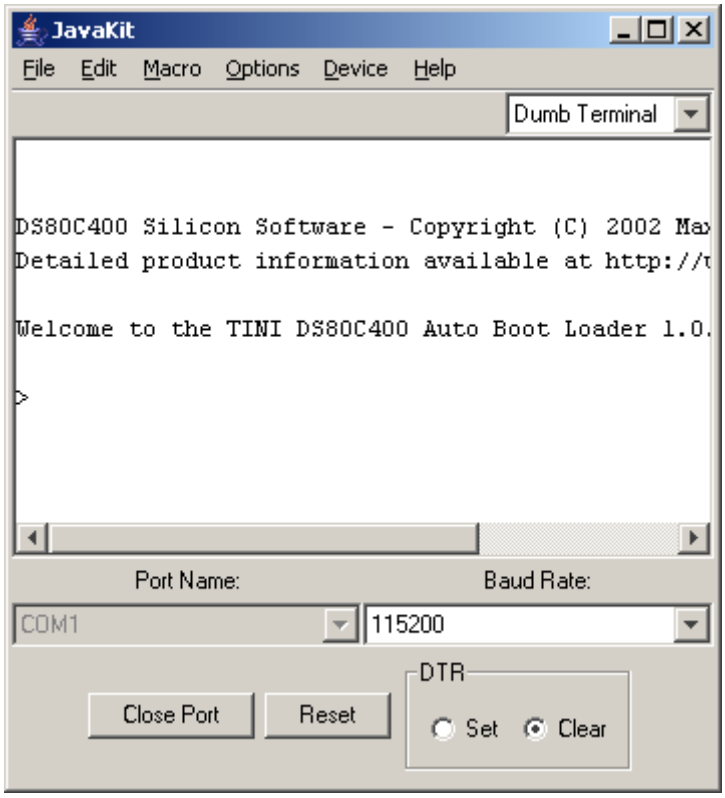
Ensuite on branche la TINI au PC grâce à un câble RS232 et on lance JavaKit à l'aide de la commande :

c:\< JDK >\bin\javaw -classpath c:\< SDK >\bin\tini.jar JavaKit

La fenêtre suivante s'affiche :



On ouvre le port sur lequel est branché la TINI, on vérifie que la vitesse de transmission est bien à 115200 et on clique sur *Reset.* Puis on voit apparaître un prompt :

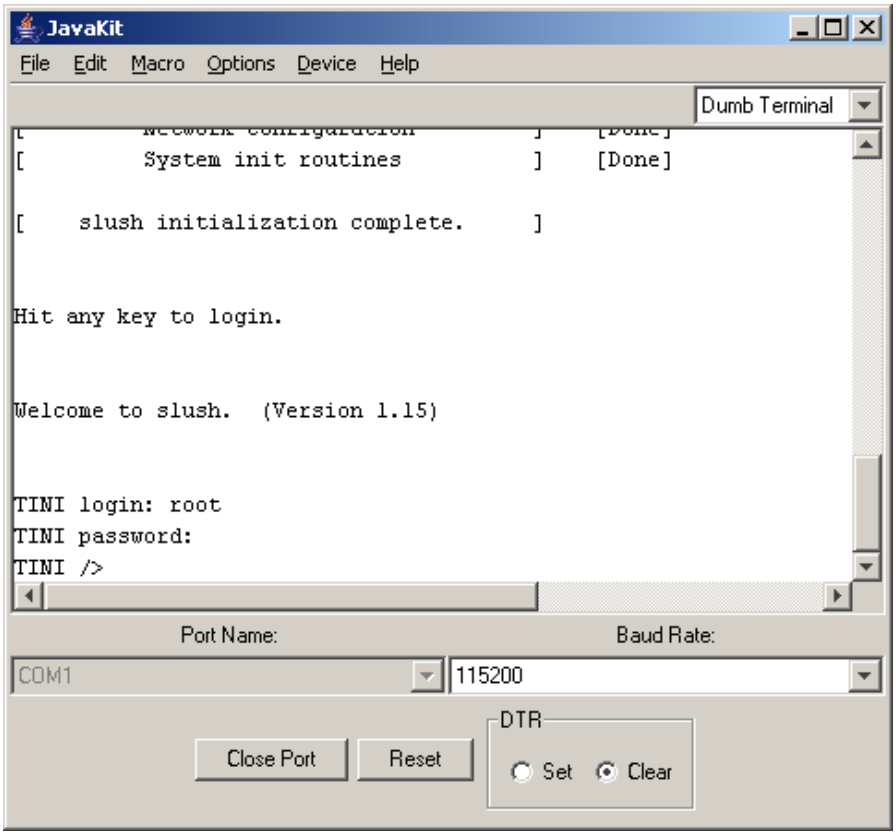


Une fois JavaKit lancé et connecté à la TINI, on peut charger l'environnement logiciel. On clique sur *File* et *Load File,* et on sélectionne les fichiers tini.tbin et Slush.tbin. Cela prend quelques minutes.

Il faut donc maintenant initialiser la pile grâce aux deux commande suivantes, taper successivement :

- BANK 18
- FILL 0

Maintenant il faut lancer le système : on doit taper la commande EXIT, Slush se lance donc et nous demande de nous loger, comme on peut le voir dans l'image suivante :



On se log en root (mot de passe *tini*), pour configurer la TINI.

La commande **help** affiche toutes commande du Slush et si elle est suivie d'une commande elle affiche la description de celle-ci.

Il existe la commande **ipconfig** pour voir et configurer les connections réseau. Ainsi les options **-a** et **-m** permettent respectivement d'attribuer à la TINI une adresse IP et un masque de sous-réseau.

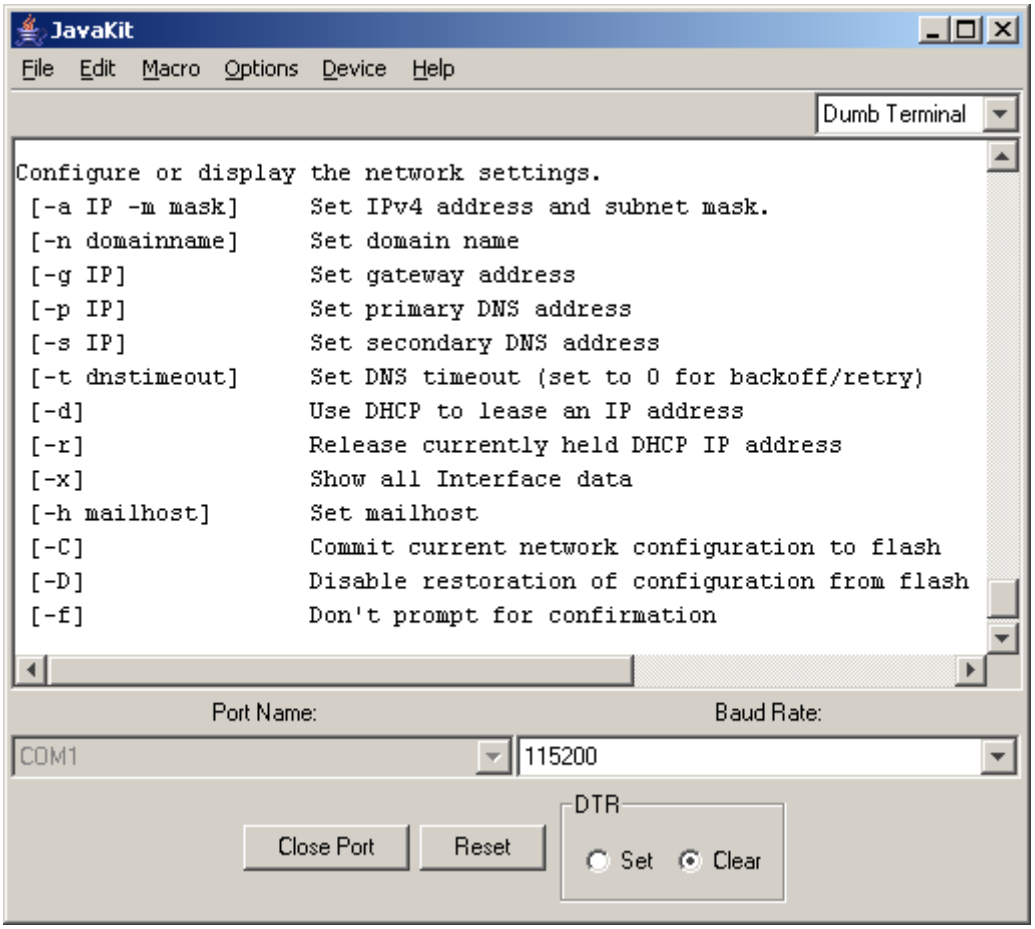
Exemple :

TINI />ipconfig -a 192.168.0.15 -m 255.255.255.0

On lui attribut l'adresse IP 192.168.0.15 avec le masque de sous-réseau 255.255.255.0

Remarque *: Il faut vérifié au niveau des adresses IP que le PC et la TINI sont sur le même réseau.*

Il est aussi possible de lui donné d'autres attributs tel que le nom de domaine ou l'IP de la passerelle, on peut pour ça consulter l'aide en tapant la commande **help ipconfig** :



La TINI a maintenant une adresse IP, on peut donc enlever le câble RS232 et la connecté au PC avec un câble RJ45 croisé pour ouvrir une session telnet depuis ce PC.
Maintenant que la TINI est correctement configuré il ne reste plus qu'à tester un programme.

Principe de programmation

7.Icrémentation n°1: Édition d'un programme de test

BUT:

Concevoir un programme qui ne fait rien d'autre qu'afficher une chaîne de caractère à l'écran, et l'exécuter sur la carte micro-contrôleur TINI en vue de la tester.

PRINCIPE:

La conception d'un programme pour être exécuté sur la carte TINI consiste en quatre phases :

- Édition **du programme** : cela peut être fait avec un simple éditeur de texte en sauvegardant le programme en *.java.*

- **Compilation du programme** : Il faut compilé le programme avec les classes de l'API JAVA de la carte TINI et non celles du JDK. Elles se trouvent dans :

c:\<SDK>\bin\tiniclasses.jar

- **Conversion du programme** : en effet le (ou les) fichier *.class* d'un programme ne sont pas interpréter par la machine virtuelle JAVA de la carte TINI. Il faut pour ça le (ou les) convertir en un fichier **.tini.**

Cela se fait grâce à un programme du JDK : **TINIConvertor.**

Il se trouve dans ***c:\< SDK >\bin\tini.jar***, tout comme JavaKit.

- **Transfer du programme dans la carte TINI :** une fois le programme prêt il faut le transférer à l'aide d'une session FTP sur la carte TINI.

Ensuite il faut ouvrir une session telnet sur la carte TINI et exécuter le programme à l'aide de la commande **java** suivie du nom du programme **.tini.**

APPLICATION:

-Édition du programme :

On édite le programme suivant à l'aide de l'éditeur de texte **JEDIT** :

```
class helloworld{
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

Ce programme ne fait rien d'autre que d'afficher la célèbre phrase « HelloWorld ». On le sauvegarde en helloworld.java.

-Compilation du programme :

Pour compiler le programme avec L'API de la carte TINI on utilise la commande bootclasspath suivie du dossier contenant les classes, c'est-à-dire c:\<SDK>\bin\tiniclasses.jar

On compile le programme à l'aide de la commande suivante :

javac -bootclasspath c:\<SDK>\bin\tiniclasses.jar helloworld.java

Une fois la compilation terminée on obtient un fichier **helloworld.class**

Conversion du programme :

Pour convertir le programme on utilise **TINIConvertor.**

Pour obtenir de l'aide dessus il suffit de le lancer sans paramètres à l'aide de la commande suivante :

java -classpath c:\< SDK >\bin\tini.jar TINIConvertor

Sinon il se lance avec trois paramètres :

-f *<fichier ou répertoire d'entrée (.class)>* ***-d*** *<base de donnée des API>* ***-o*** *<fichier de sortie (.tini)>*

On convertit donc **helloworld.class** avec la commande suivante :

java -classpath c:\< SDK >\bin\tini.jar TINIConvertor -f c:\HelloWorld.class -d c:\< SDK > \bin\tini.db -o HelloWorld.tini

Une fois la conversion terminé on obtient un fichier **helloworld.tini**

- Transfert du programme :

On ouvre une session FTP sur la carte TINI à l'aide de la commande

ftp 192.168.109.236

on se log en root puis on tape la commande ***bin*** pour être sur qu'on est bien en mode binaire et non en mode ASCII.

Ensuite on transfert le programme sur la carte TINI avec la commande :

put helloworld.tini

Pour vérifier que le transfert à bien eu lieu on ouvre une session telnet grâce à la commande

telnet 192.168.109.236

puis on tape la commande ***dir*** qui permet de lister le contenue de la carte TINI.

Maintenant que le programme est bien dans la carte TINI il ne reste plus qu'à l'exécuter :

on le lance depuis la session telnet avec à l'aide de la commande

java HelloWorld.tini

On peut aussi utilisé l'option « & » (après le nom du fichier) pour lancer le programme en tache de fond.

8. Réalisation d'un thread en Java

Les threads

1. Introduction

Dans notre projet la gestion système peut se décomposer en deux taches :

Il y a d'abord la tache de scrutation des états de la borne bien-sur , puis le traitement de ces informations .
Toute la difficulté réside dans le fait que ces taches doivent se dérouler en parallèle et non l'une à la suite de l'autre.

Nous avons donc 2 taches qui se déroulent en parallèle :

- **scrutation des Entrés/Sorties**
- **Traitement des informations**

Pour coder en JAVA plusieurs taches qui s'exécutent en parallèle nous allons utiliser des **THREADS**

2. Qu'est ce qu'un thread ?

Un thread est une portion de code qui est capable de s'exécuter en parallèle à d'autres traitements (ou d'autres threads). On appelle aussi ça un processus léger, mais un thread et un processus sont deux choses à ne pas confondre :

- un processus a une zone mémoire propre et il ne peut pas empiéter sur la zone mémoire d'un autre processus
- les threads eux se partagent une même zone mémoire : en effet si une application dispose de plusieurs threads alors ces threads se partageront la zone mémoire de l'application qui

les a lancés.

Notre application devra donc comporter 2 threads :

- un pour la tache de scrutation des E/S
- un pour la tache de traitement des informations

3. Définir un thread en JAVA

En JAVA un thread est une classe qui hérite de la classe **Thread (**qui fait partie des classes de base du langage). Cette classe possède plusieurs méthodes dont les méthode *run( )* et *start( )* :

run( ) : c'est une méthode que l'on doit redéfinir. Elle doit comporter la fonction du thread c'est-à-dire la tache à accomplir en parallèle à d'autre traitements. Par exemple pour le thread de scrutation, la méthode run( ) devra contenir les instructions qui vont lire les états des actionneurs et des capteurs .

Start( ) : cette méthode n'est pas à redéfinir. C'est par elle que l'on va lancer le thread. En fait elle lance juste le méthode run( ).

Si on lançait directement la méthode run( ) sans utiliser start( ), on ne pourrait pas faire plusieurs traitements à la fois : en effet lorsqu'on lance start( ) elle va elle même lancer run( ) puis on vas ensuite exécuter les instructions suivant start( ), et la méthode run( ) vas s'exécuter en parallèle à ces instructions (qui peuvent éventuellement lancer d'autres threads).

4. L'exclusion mutuelle

Comme on doit coder plusieurs threads, il faut veiller à ce que les différent traitements ne s'entremêle pas.
Il existe pour ça un procédé qu'on appelle l'exclusion mutuelle ou synchronisation.
On vas pour ça utiliser le mot-clés *synchronized* et les méthodes *wait( )* et *notify( )* :

- synchronized :

si on veut qu'un bloc d'instructions (ou une méthode) ne puisse pas s'exécuter en même temps qu'un autre bloc d'instructions alors il faut **synchroniser** ces deux traitements sur un même objet :

```
synchronized(objet) {
        instructions ...
                    ...
}
```

On dit que le thread en cours va poser un verrou sur l'objet avant d'exécuter les instructions du bloc, puis il va enlever ce verrou. Ainsi, si pendant l'exécution de ces instructions, un autre bloc synchronisé sur le même objet veut s'exécuter, il va alors se mettre en attente jusqu'à retrait du verrou. Et de même, si un verrou avais déjà été placé sur objet on aurait aussi attendue le retrait du verrou avant d'exécuter les instructions.

- wait( ) :

Cette méthode va permettre de mettre en attente le thread en cours c'est-à-dire le thread qui vas lancer la méthode :
Exemple :

```
class thread1 extends Thread {
        run() {
                instruction.....
                thread2.wait( );
                instruction.....
        }
```

Dans cette exemple thread1 est le thread en cours, c'est donc lui qui est mis en attente et non thread2. On dit que thread1 est mis en attente par thread2.

- notify( ) :

Cette méthode va permettre de réveiller un thread qui a été mis en attente par le thread en cours. Dans l'exemple ci-dessus, pour réveiller thread1 , il faut que dans le run( ) de thread2 il y est une instruction **notify( ).** Elle vas réveiller thread1 qui a était mis en attente par thread2.

9.Incrémentation n°2: *Codage du thread de scrutation des E/S*

BUT :

Coder un thread pour scruter en permanence l'état des E/S du système, en utilisant le module de communication par bus I2C qui va fournir la méthode de lecture sur le bus,mais aussi de mettre ces évènements dans une queue de liste.

PRINCIPE :

Dans cette partie nous allons nous intéresser à la méthode de lecture sur le bus I2C de la commande barriere.

APPLICATION:

```
public void scrutation extends Thread {

public scrutation(){} //Constructeur par default

public void tempo(long times)
        {
                try { sleep(times);}
```

```
                catch ( InterruptedException e) {};
        }

    public void run() {
        while (true){
            bar.scan();
            tempo(20); // La scrutation de la barriere se fera toute les 20 ms
            }

public final void putEvent(final byte code) // Méthode qui va mettre les évènements dans la
                                            queue de liste
    {
        liste.write(code);
        liste.notify();// pour interrompre l'attente du thread
    }
```

10.Incémentation n°3: Codage du thread de traitement des E/S

BUT :

Coder un thread qui va récupérer dans la queue de liste les événement un par un et de les traiter .

PRINCIPE :

Dans cette partie nous allons nous intéresser à la méthode de récupération des évènements dans la queue de liste

APPLICATION:

```
public byte getEvent() //Méthode de récupération des événement dans la queue de liste.
    {
        byte result;
        result = ((byte)liste.read());
        if (result == -1) try{
                            liste.wait(); // mise en attente du thread a fin de le
                                          coordonner avec l'autre thread
                          }
                        catch (InterruptedException e) {}
        return result;
    }
```

11.Référence croisée

Afin que la Classe ***Borne*** puisse correctement faire référence à la classe ***Barriere*** et que cette classe ***Barriere*** puisse mettre ces événement dans la queue de liste (qui appartient à la classe

Borne) ;nous devons utiliser la *« référence croisée ».*

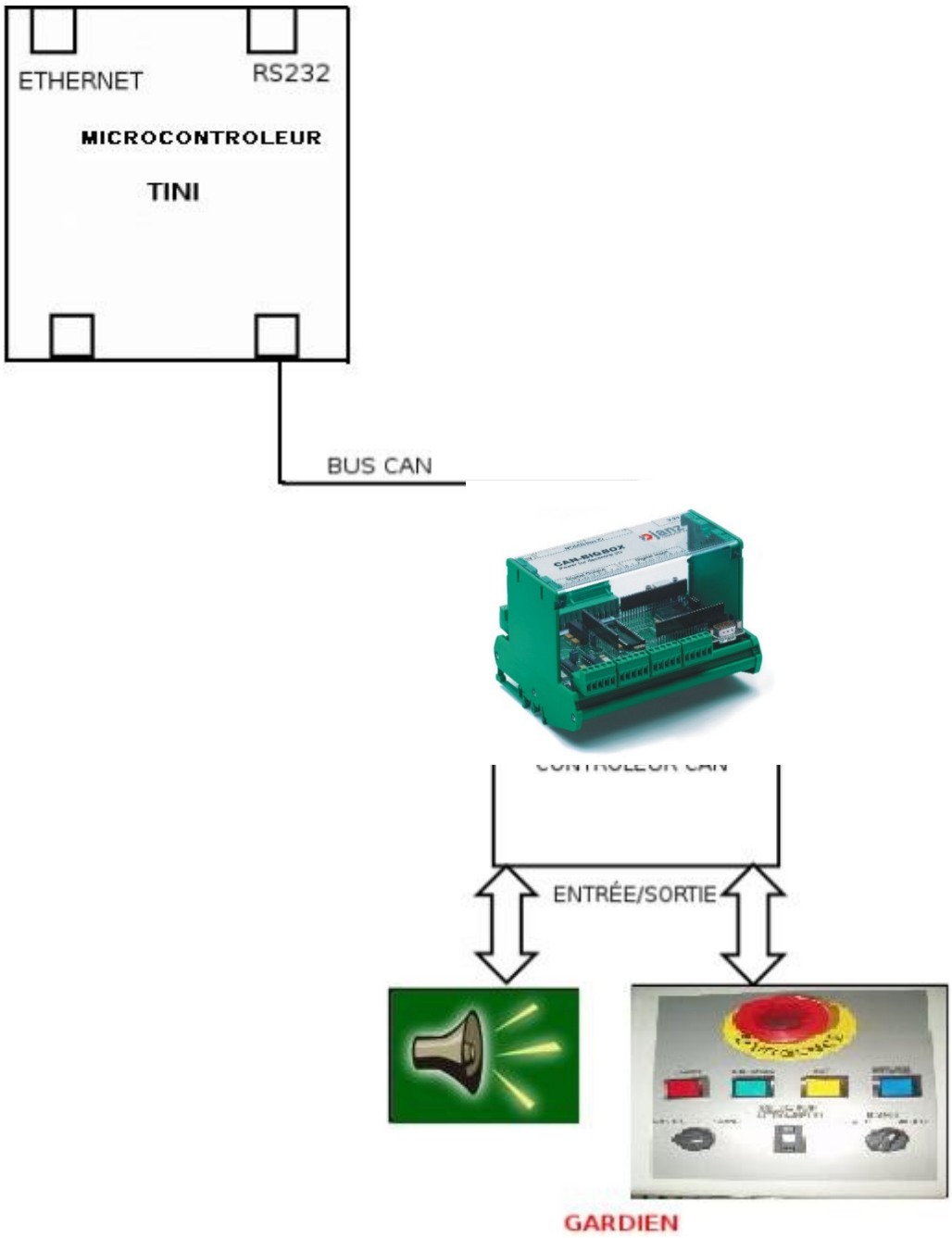
```
public class Borne {

Barriere bar;
TinyEventQueue liste;
gespark gespark;

public Borne(){}//constructeur par défault

public Borne(gespark gespark)// constructeur par passage de reference{
        bar = new Barriere(this);
        liste = new TinyEventQueue();
        this.gespark = new gespark();
                }

public class Barriere {
BarriereImpl BarriereImpl;
Borne borne;
Barriere Barr;
public Barriere(){}// Constructeur par default
public Barriere(Borne borne) throws PortException
{
                BarriereImpl = new BarriereImpl();
                Barr.init();
                this.borne=borne;
}
```

CANDIDAT 2

4.4_Candidat 2 : JEANS-LOUIS Patrice

1.Presentation de ma partie

Diagramme structurel de ma partie:



Dans ce projet j'interviens essentiellement sur la communication CAN du système, ainsi je dois :

1) M'approprier du cahier des charges du système.
2) Coder et tester le module de communication du Bus CAN avec la BigBox.

Ce module me permmetra entre autre d'écrire sur les sorties du module de communication Bigbox, et de lire l'état des entrées du module de communication Bigbox.

3) Cabler le bus de communication CAN Bigbox.
4) Cabler le boitier muni de boutons poussoirs sur le module de communication Bigbox.

5) Insérer ma partie au système de gestion de Parking afin de finaliser le projet.

MON CAS D'UTILISATION GENERAL

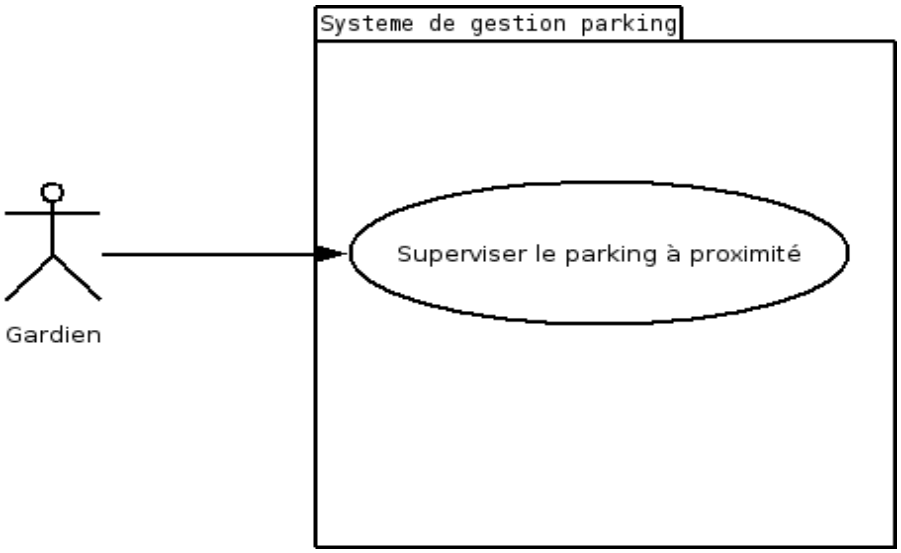
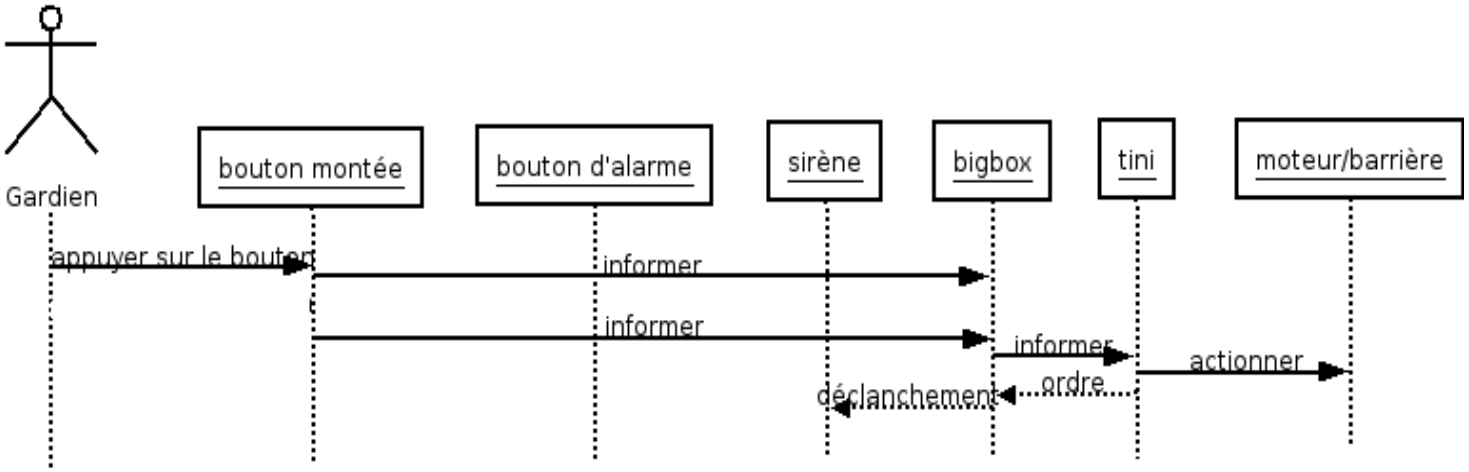


DIAGRAMME DE SEQUENCE



3.Codage

Très important :
Dans le but de faire communiquer les deux modules CAN(de la bigbox et du microcontrôleur) correctement,la vitesse de transmission calculée doit être égale à plus ou moins 0.5% entre les des modules.

Avant d'exécuter ce code, ne pas oublier de régler la Bigbox à une vitesse de 125kbit/s en réglant la Bigbox (SW2*3).

Il n'existe aucune méthode pour directement calculer la vitesse de transmission au lieu de ça vous devez entrer la combinaison appropriée de paramètres de l'horloge

Le calcul de la vitesse de transmission se fait de la sorte:

Vitesse de transmission (bits/s) = 18.432MHz * 10^6/BRP/ (1 + TSEG1 + TSEG2)
18.432MHz : représente la fréquence d'horloge du microcontrôleur.

Pour écrire sur les sorties du module de communication Bigbox

```
import com.dalsemi.comm.*;
import com.dalsemi.system.*;
import com.dalsemi.system.*;

public class cantransmit
{
   /* 125Kbit/s with crystal of 18.432MHz */
   static final int CAN_DIVISOR = 7;
   static final int CAN_TSEG1 = 9;
   static final int CAN_TSEG2 = 7;
   static final int CAN_SJW = 1;
   static final byte CANBUSNUM = CanBus.CANBUS0;
//    static final byte CANBUSNUM = CanBus.CANBUS1;

  static void doTest() throws Exception
    {
       CanBus a = new CanBus(CANBUSNUM);

/* Attribution de la vitesse de transfert de données dans le bus*/

       a.setBaudRatePrescaler(CAN_DIVISOR); //change le BRP à une valeur entre 1 et 256
       a.setTSEG1(CAN_TSEG1);  //change le segment de temps1 à une valeur entre 2 et 16
       a.setTSEG2(CAN_TSEG2);  //change le segment de temps1 à une valeur entre 2 et 8
       a.setSynchronizationJumpWidth(CAN_SJW);

       // On demande au contrôleur CAN Controller de sauter sur le Bus
       a.enableController();

       // Validation du centre de message I pour qu'il transmette ceci permet à tous messages sortants
       //d'utiliser ce registre
       a.setMessageCenterTXMode(1);
       int i;
       byte[] temp = new byte[8];
       System.out.println("Continous send");
       temp[0] = (byte)0x00;  //Octet permettant d'assigner les 8premiers bits de la bigbox

       temp[1] = 0x00;
       int count = 0;
       while (true)
       {
          temp[2] = (byte)count; //ici temp[2] prend la valeur d'un INT
          temp[3] = (byte)(count >>> 8);  //count est mit 8bits vers la droite ds temp[3]
// Si on veut utiliser un ID étendu(29bits) on bloque jusqu'à arrivée de l'accusé
        //          a.sendDataFrame(0x55F6575C, true, temp);
```

```
// Envoie d'une trame en utilisant l'ID standard (11 bit), bloque jusqu'à l'arrivée de l'accusée de réception
          CanFrame Trame = new CanFrame();
            Trame.ID = 2;
          Trame.extendedID = false;
          Trame.messageCenter = 0;
          Trame.remoteFrameRequest = false;
          Trame.data = new byte[8];
          Trame.length = 1;
               a.sendDataFrame(0xdd, false, temp);
          if ((count % 10) == 0)
          {
             Debug.hexDump(count);
          }
          count++;
       }

   }

   static void main(String args[])
   {
      try
      {
         if (TINIOS.isCurrentTaskInit())
            Debug.setDefaultStreams();

         System.out.println("CAN Transmit tester");

         doTest();

         System.out.println("Normal Exit");
      }
      catch (Throwable e)
      {
         System.out.println("Exception");
         System.out.println(e);
      }
   }
}
```

Exécution du programme

Pour être exécuté dans le système,le programme doit être introduit dans le microcontrôleur(TINI),car c'est à partir de lui que nous contrôlons tout le système du parking y compris à proximité qu'à distance.

Méthode à suivre:

1. on crée un fichier *.BAT* exemple un fichier *« compil.bat »*
   Le fichier « compil.bat » à pour intérêt de nous éviter d'avoir retaper tout le chemin nécessaire à l'exécution d'une requête de compilation.

Fichier compil.bat:

```
cls
javac-bootclasspath      T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\tiniclasses.jar;T:\
PROJET\PARKING\testCAn\TINI\tini1.15\bin\modules.jar          T:\PROJET\PARKING\testCAn\
cantransmit.java
```

T:\PROJET\PARKING\testCAn **:** représente le fichier dans lequel se trouvent toutes les classes utilisées dans le programme cantransmit.java .

Après la compilation nous devons convertir le programme ***cantransmit.class*** obtenu lors de la compilation en un programme appelé cantransmit.tini car le microcontrôleur Tini ne comprend pas le type de fichiers sous forme ****.class***

2. on crée un fichier *« convert.bat »* qui comme compil.bat nous évitera d'avoir à retaper tout le chemin nécessaire à l'exécution du programme de conversion.

Fichier Convert.bat :

```
java     -classpath     T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\tini.jar;%classpath%
BuildDependency  -f  T:\PROJET\PARKING\testCAn\cantransmit.class  -o  T:\PROJET\PARKING\
testCAn\cantransmit.tini  -d  T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\tini.db  -add
CANAll -x T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\owapi_dep.txt -p T:\PROJET\PARKING\
testCAn\TINI\tini1.15\bin\modules.jar
```

T:\PROJET\PARKING\testCAn\cantransmit.class : est le fichier d'entré avant la conversion.
T:\PROJET\PARKING\testCAn\cantransmit.tini **:** est le fichier qu'on obtiendra à l'issu de la conversion.

REMARQUE : les classes utilisées à la fois dans compil.bat et convert.bat sont :

2. modules.jar
1. tini.jar

1. tiniclasses.jar.

Transfert du fichier cantransmit.tini dans le microcontrôleur TINI

Pour cela plusieurs étapes sont nécessaires:

1. se connecter au microcontrôleur via son serveur ftp
   taper en ligne de commande :
   T:\>ftp 192.168.109.236
   Connecté à 192.168.109.236
   220 Welcome to slush. (Version 1.15) Ready for user login.
   Utilisateur (192.168.109.236:(none)) : **root**
   mot de passe : **tini**

2. **Transférer le fichier** *cantransmit.tini* **dans la TINI**

on doit spécifier qu'on veut faire un transfert de fichier de type *Binaire*
ftp> **bin**
200 Type set to Binary
ftp> **put** *T:\PROJET\PARKING\testCAn\cantransmit.tini*

Nb :Avec 192.168.109.236 @IP attribuée au microcontrôleur grâce à la commande ***ipconfig.***

Exécution du fichier cantransmit.tini

Cette action se fera par **telnet**

T:\>telnet 192.168.109.235
tini00d598 login: **root**
*tini00d598 password:***tini**
*tini00d598 />***java** *cantransmit.tini*

Pour faire une lecture des entrées du can Bigbox

```
import com.dalsemi.comm.*;
import com.dalsemi.system.*;

public class canreceive
{
   /* 125Kbit/s with crystal of 18.432MHz */
   static final int CAN_DIVISOR = 7;
   static final int CAN_TSEG1 = 9;
   static final int CAN_TSEG2 = 7;
   static final int CAN_SJW = 1;
   static final byte CANBUSNUM = CanBus.CANBUS0;

 /*
   IDENTIFICATION DE LA TRAME RECUE
 */

   static void dumpFrame(CanFrame frame)
   {
      System.out.println("Done receiving frame");
      System.out.println("ID: "+Integer.toHexString(frame.ID));
      if (frame.extendedID)
         System.out.println("Extended ID");
      else
         System.out.println("Standard ID");
      if (frame.remoteFrameRequest)
         System.out.println("Remote Frame");
      else
         System.out.println("Data Frame");
      System.out.println("Length: "+frame.length);
      for (int i = 0;i < frame.length;i++)
         System.out.print(Integer.toHexString(frame.data[i] & 0xFF)+" ");
      System.out.println();
   }

   static final int MAXCOUNT = 100;

   static final boolean dopassive = false;

   static void doTest() throws Exception
   {
      final int DUMPCOUNT = 1;
      long start,stop;
```

```
/*
  CONFIGURATION DE LA VITESSE DE COMMUNICATION ENTRE LE DEUX CONTROLEURS CAN
*/

        CanBus a = new CanBus(CANBUSNUM);
        a.setBaudRatePrescaler(CAN_DIVISOR);
        a.setTSEG1(CAN_TSEG1);
        a.setTSEG2(CAN_TSEG2);
        a.setSynchronizationJumpWidth(CAN_SJW);

        System.out.println("Trames disponibles a lire: "+a.receiveFramesAvailable());

/*
   Maintenant disons au controleur CAN de sauter su le Bus Can
*/
           if (dopassive)
        {
           System.out.println("reception passive acceptee");
           a.enableControllerPassive();
        }
        else
        {
           System.out.println("reception reguliere acceptee");
           a.enableController();
        }
        byte[] temp = new byte[8];
        // utilisation du centre de message 1 pour recevoir
        a.setMessageCenterRXMode(1);
        // demandons au centre de message 1 de ne pas utiliser de masque filtrant
        a.setMessageCenterMessageIDMaskEnable(1,false);
        // demandons au centre de message 1 de comparer son ID de 11 bits avec celui qu'il a reçu
        a.set11BitMessageCenterArbitrationID(1,0x55F6575C); // pour lire les entres de la bigbox
        // Set configurons le centre de message d'accpeter la  reception des messages.
        a.enableMessageCenter(1);
           a.sendDataFrame(0xde, false, temp);
        // creation d'une Trame pour reception
       CanFrame Trame = new CanFrame();

            Trame.ID = 2;
           Trame.extendedID = false;
           Trame.messageCenter = 0;
           Trame.remoteFrameRequest = false;
           Trame.data = new byte[8];
           Trame.length = 1;
```

```
System.out.println("RECEPTION EN COURS");
int count = 0; // count compte le nombre de donnée reçues
int endcount = MAXCOUNT;
int firstpacket = 0;
start = TINIOS.uptimeMillis();
boolean error = false;
while ((count < endcount) && (!error))
{
   // On bloque dans l'attente de la reception d'une trame

      a.receive(Trame);

     while (!a.receivePoll(Trame))
    {
      System.out.println("hey");
    }

    temp = Trame.data;

    if (count == 0)
    {
      count = (temp[2] & 0xFF) | (temp[3] & 0xFF) << 8;
      endcount = count + MAXCOUNT; // MAXCONUT etant le maximum de donnees reçues
      firstpacket = count;
    }

    int reccount = (temp[2] & 0xFF) | (temp[3] & 0xFF) << 8;
    if (dopassive)
    {
      if (reccount != firstpacket)
      {
        System.out.println("reception d'un count egal a "+reccount+" Esperant un count de
"+count);
        error = true;
      }  // fin du if
    }
    else
    {
      if (reccount != count)
      {
        System.out.println("Reception d'un count de"+reccount+" Esperant un count de
"+count);
        error = true;
      }
    } // fin du else
```

```
            if ((count < DUMPCOUNT) || error)
            {
               dumpFrame(Trame);
            }

               count++;
         }      // fin du while

         stop = TINIOS.uptimeMillis();
         System.out.println("TEMPS COMPLEMENTAIRE "+(stop-start));
         System.out.println(start+":"+stop);
         System.out.println("SORTIE APRES "+(count-firstpacket)+" iterations");

         a.close();
      }

      static void main(String args[])
      {
         try
         {
            if (TINIOS.isCurrentTaskInit())
               Debug.setDefaultStreams();

            System.out.println("TEST DE CAN receive");

            doTest();

            System.out.println("Sortie normale");
         }
         catch (Throwable e)
         {
            System.out.println("Exception");
            System.out.println(e);
         }
      }
   }
```

Exécution du programme

Fichier compil.bat:

```
cls
javac-bootclasspath T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\tiniclasses.jar;T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\modules.jar T:\PROJET\PARKING\testCAn\canreceive.java
```

Fichier covert.bat

```
java -classpath T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\tini.jar;%classpath% BuildDependency -f T:\PROJET\PARKING\testCAn\canreceive.class -o T:\PROJET\PARKING\testCAn\canreceive.tini -d T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\tini.db -add CANAll -x T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\owapi_dep.txt -p T:\PROJET\PARKING\testCAn\TINI\tini1.15\bin\modules.jar
```

REMARQUE : les classes utilisées à la fois dans compil.bat et convert.bat sont :

3. modules.jar
2. tini.jar
2. tiniclasses.jar.

Description des differentes commandes utilisées

-add NOMS : permet d'ajouter les noms des dependances à ajouter au projet.(les diffenertes dépendences sont séparées entre elles par un point-virgule.
-p PATH : chemin d'acces à vos dependances(cette commande peut inclure des repertoires et des fichiers séparés entre eux par un point-virgule.
-x DEP_FILE : Nom de fichier du fichier de base de données de texte de dépendance.
-o : specifie le type de fichier en sortie de la conversion ainsi que le repertoire dans lequel il se trouvera.

Transfert du fichier canreceive.tini dans le microcontrôleur TINI

Pour cela plusieurs étapes sont nécessaires:

3. se connecter au microcontrôleur via son serveur ftp
taper en ligne de commande :
T:\>*ftp 192.168.109.236*
Connecté à 192.168.109.236
220 Welcome to slush. (Version 1.15) Ready for user login.
Utilisateur (192.168.109.236:(none)) : **root**
mot de passe : **tini**

4. **Transférer le fichier** *canreceive.tini* **dans la TINI**

On doit spécifier qu'on veut faire un transfert de fichier de type *Binaire*

ftp> **bin**
200 Type set to Binary
ftp> **put** *T:\PROJET\PARKING\testCAn\canreceive.tini*

Nb :Avec 192.168.109.236 @IP attribuée au microcontrôleur grâce à la commande ***ipconfig.***

Exécution du fichier canreceive.tini

Cette action se fera par **telnet**

T:\>telnet 192.168.109.235
tini00d598 login: **root**
*tini00d598 password:***tini**
*tini00d598 />***java** *canreceive.tini*

3.Conclusion

Ce proejt m'a été très instructif,car non seulement il m'a permit de mettre en application mes connaissances acquises tout au long de ces 2 années,et m'a surtout donné un aperçu assez réaliste de la vie en entreprise,la réalisation de projets en groupe,le partage de connaissance...c'est donc une très belle expérience dans le dommaine humain vant tout.
Pour m'approprier du cahier des charges j'ai dù commencer par elaborer plusieurs diagrammes et schémas. Notamment :

Schéma ensemble de système.

Diagramme cas utilisation.

Diagramme de séquence

Diagramme de séquence élaboré.

- **Pour le CAN**
  - ➔ J'ai pu attribuer chaque bouton poussoir du boîtier qui m'a été donné de câbler et de brancher sur la bigbox.
  - ➔ Analyse des fonctionnalités de la bigbox grâce à sa documentation technique
  - ➔ j'ai rédigé la page de présentation de la bigbox,ainsi que celle du bus can.
  - ➔ J'ai Câblé le bus CAN devant relier le can bigbox au can du microcontrôleur (tini)
  - ➔ j'ai réussi à envoyer une trame du microcontrôleur à la bigbox grâce au bus CAN.

Malgré tout j'ai eu quelques contraintes aussi bien matérielles qui ont été par la suite résolus,me causant un serieux manque de temps par la suite:

exemple de contraintes matérielles:

- Attribution de boutons poussoir adéquats pouvant être incorporés au systême.
- Les sucres(petites connectiques permettant de relier les entrées/sorties du module de communication BigBox avec l'extérieur(il permettent de cables differents périfériques tels que les boutons poussoirs).

Je n'ai donc pas eu le temps de finaliser l'intégration de ma partie avec les candidats 1 et 3.

4.annexe

Class CanFrame

`byte[]`	**data** Frame data area.
`boolean`	**extendedID** Standard ID (11 bits) if `false` or Extended ID (29 bits) if `true`.
`int`	**ID** Standard ID (11 bits) or Extended ID (29 bits) based on the value of `extendedID`.
`int`	**length** Length of data area.
`int`	**messageCenter** CAN Controller Message Center.
`boolean`	**remoteFrameRequest** Remote Frame if `true` or Data Frame if `false` .

Detail des Methodes utilisées.

setID

```
public void setID(int ID)
```

Met le champs ID dans la trame CAN.

Parameters:
`ID` - identifier on frame

getID
```
public int getID()
```

recupère le champs ID sur la trame à envoyer ou reçue
Returns:
L'ID de la trame

setMessageCenter
```
public void setMessageCenter(int MC)
```

valide un centre de message d'où envoyer cette trame
Parametres:
`MC` – centre de message à utiliser

getMessageCenter
public int **getMessageCenter**()
Recupere le centre de message où la trame a été reçue ou celle d'où la trame sera enviyée.
Returns:
centre de message

setRemoteFrameRequest
```
public void setRemoteFrameRequest(boolean RTR)
```

valide le bit RTR dans la trame pour changer celui-çi d'une trame de données vers une trame plus etendue.
Parameters:
`RTR` - `true` pour une Remote Frame, `false` pour une Data Frame.

getRemoteFrameRequest
```
public boolean getRemoteFrameRequest()
```

Récupere le bit pour determiner si c'est une Remote Frame.
Returns:
`true` pour une Remote Frame, `false` pour une Data Frame

setData
```
public void setData(byte[] buf)
```

configure la donnée pour une trame de sortie. La taille MAXimum de la trame est 8 octets
Parameters:
`buf` – donnée à mettre dans la trame.

getData

```
pusetIDgetIDsetMessageCentergetMessageCentersetRemoteFrameRequestgetRemoteFrameR
equestsetDatablic byte[] getData()
```

Récupere la donnée pour un envoie ou une reception de trame.

Returns:

la donnée dans la trame.

setLength

```
public void setLength(int length)
```

Attribut une longueur pour la donnée dans une trame de sortie Maximum La taille MAXimum de la trame est 8 octets.

Parameters:

`length` – longueur de la donnée dans la trame.

getLength

```
public int getLength()
```

Récupere la longueur de la donnée dans les trames en entrée ou en sortie.

Returns:

longueur de le trame de donnée.

receiveFramesAvailable

```
public int receiveFramesAvailable()
                           throws CanBusException
```

Récupere le nombre de trames en attente de reception. Si le nombre de trames à recevoir est zero ,le valeur retournée sera zero

.

Returns:

nombre de trames en suspend en attente de reception.

Throws:

CanBusException – si le controleur est ouvert.

receive

```
public void receive(CanFrame frame)
             throws CanBusException
```

Reçoit une trame CAN de données. Cette methode bloque jusqu'à reception de la trame CAN .

Parameters:

`Trame` – la trame dans laquelle placer les données en entrée.

Throws:

CanBusException – si le controleur n'est pas ouvert ou s'il y'a une erreur de BUS.

sendFrame

```
public void sendFrame(CanFrame frame)
               throws CanBusException
```

Envoie une trame CAN du centre de message demandé en objet CanFrame.Si la file d'attente de transmission est un,cette méthode bloquera jusqu'à ce que le message soit bien transmit,ou en cas d'erreur.Si la file d'attente est plus grande que un(1),le mesage sera mit en attente de transmission et la methode retournera immediatement.

Parameters:

`frame` – trame à envoyer.

Throws:

CanBusException – Si le controleur n'est pas ouvert, ou s'il y'a une erreur de bus.

sendDataFrame

```
public void sendDataFrame(int ID,
                          boolean extendedID,
                          byte[] data)
                   throws CanBusException
```

Envoie une trame de donnée CAN depuis le premier centre de message disponible sans utiliser de CanFrame explicite.si la taille de la file d'attente est un(1) , cette methode va bloquer jusqu'à la transmission du message reussie,ou qu'une erreur se produise. Si la taille de la file d'attente est plus grande que un, les messages seront mis en attente pour transmission et la methode retournera immediatemment.

CAN-BIGBOX

PRESENTATION :

Avec son contrôleur 16bits du MC 68332 et plus de 512kb de mémoire EPROM et 1MB de RAM le CAN-BIGBOX peut gérer plusieurs applications. L'emploie de la mémoire flash EPROM permet à l'utilisateur de télécharger ses propres programmes grâce à la liaison série RS232.

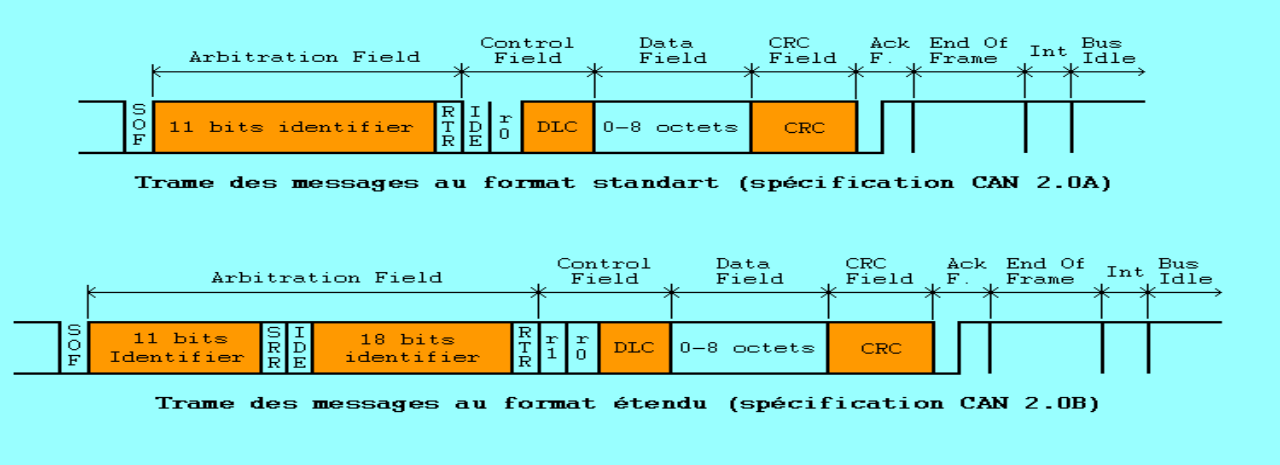
Le CAN-Bigbox fournit:
*8 entrées digitales:0-7
*8 sorties digitales : 8-15

Bus série RS 232 :
Utilisé pour télécharger, debugger ou changer les paramètres système : utilisation du serial connexion interface (sci) du MC 68332.

EPROM (NM93CS46): pour sauvegarder les informations spécifiques de bord
Capacité 1kbits.Le Chip Select de l'EEPROM est contrôlé par le bit 7 du registre de control.

Les broches de donnée série en entrée, celle de sortie et de l'horloge de donnée série sont connectés au port QSPI (voir doc big box 2-14)
La broche TPU doit être programmée en fonction de la configuration des directions.

RTC (real time clock): sellectionable par le bit **D5** du registre control. Dans la BigBox le bit **D5** est inversé ainsi le RTC n'est pas sélectionné après l'allumage ou le reset.
Bit **D6** du registre de control permet de contrôler le signal d'écriture.



CONFIGURATION

Reset : le logiciel reset peut être exécuté avec la commande *reset* (voir la description de la commande)

A l'allumage de la BigBox, le logiciel ICANOS lira par défaut les deux DIF (switch) **SW1** et **SW2.**
DIP switch 1SW1) valide une base pour les identifiants CAN //pour distinguer le noeud CAN et le réseau CAN.

Sélection de signal

Avant que le sélecteur de signaux ne devienne actif, le registre d'assignement des broches doit être initialisé :

CSPAR0 0x3ABE
CSPAR1 0x03FD

CONTROLEUR CAN 82c200

-Longueur de données: 0 à 8bits
-NRZ
-Vitesse de transfert (plus de 1Mbits/s)

☻ :le contrôleur est connecté au bus de données « parent » alors on peut y accéder de n'importe quelle adresse.

La broche 7 CLKOUT est utilisée pour alimenter le modul bus socket.

INTERFACE CAN

Le bus CAN peut être isolé électriquement du contrôleur CAN. Il est aussi possible d'utiliser l'alimentation du bus CAN // dans ce cas le convertisseur DC/DC doit être retiré et le régulateur de tension (IC22) doit être inséré. La tension minimum doit être de 6v.

Unité ENTREE/SORTIE digitale

1. Configuration en entrée

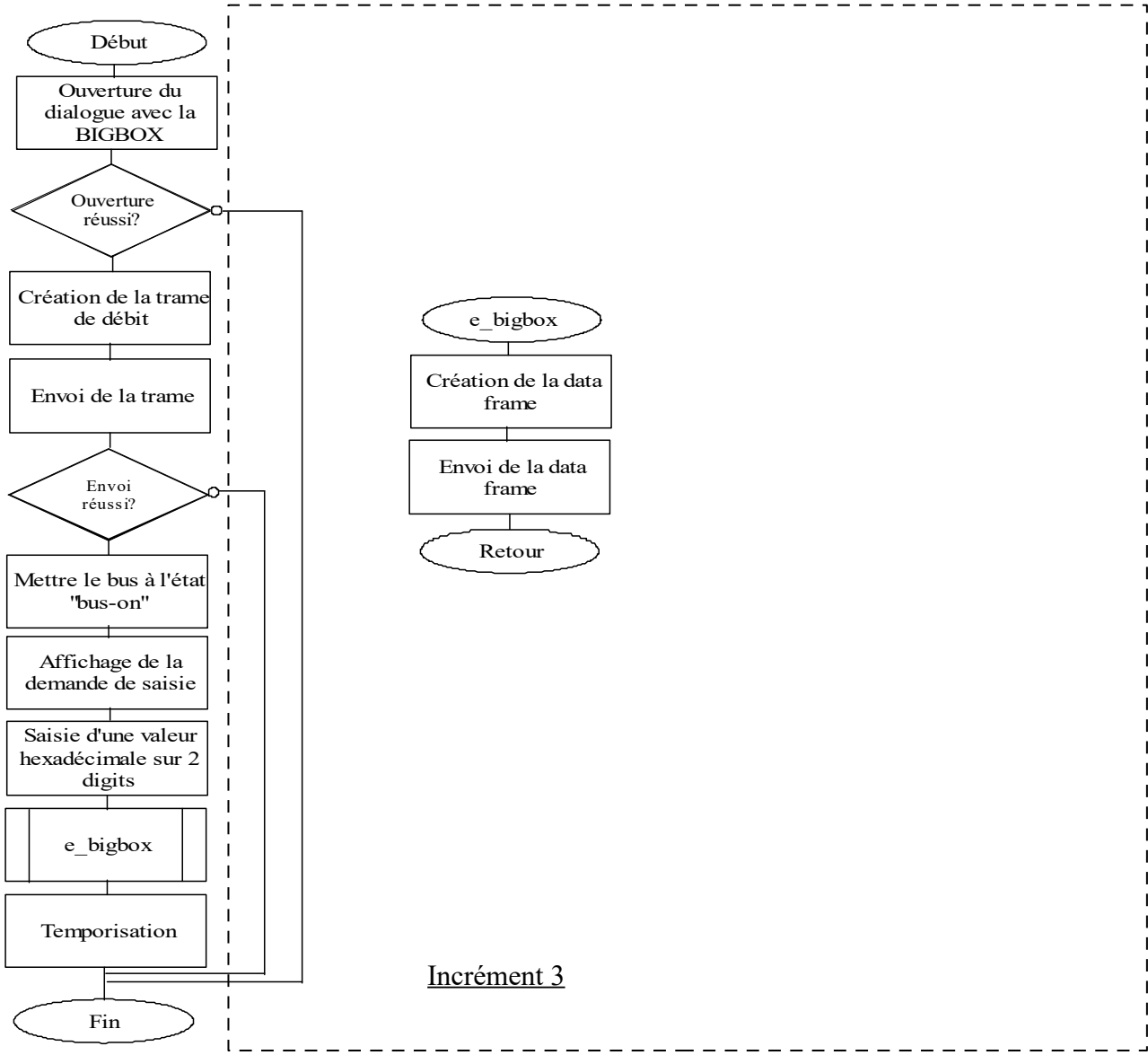
8 entrées électriquement découplées avec 4 cathodes connectées en temps réel.
Donc 2 groupes isolés des autres et de l'alimentation de la BigBox.
Le courant d'entrée de chaque chaîne est inférieur à 10ma

ECRITURE SUR LES SORTIES DIGITALES

Ecrire sur les sorties de la BIGBOX

Le rôle de cet incrément est d'écrire un octet sur les sorties de la BIGBOX. La saisie de cette valeur s'effectuera via un terminal et sera saisie sous forme hexadécimale 2 digits.

Algorithme



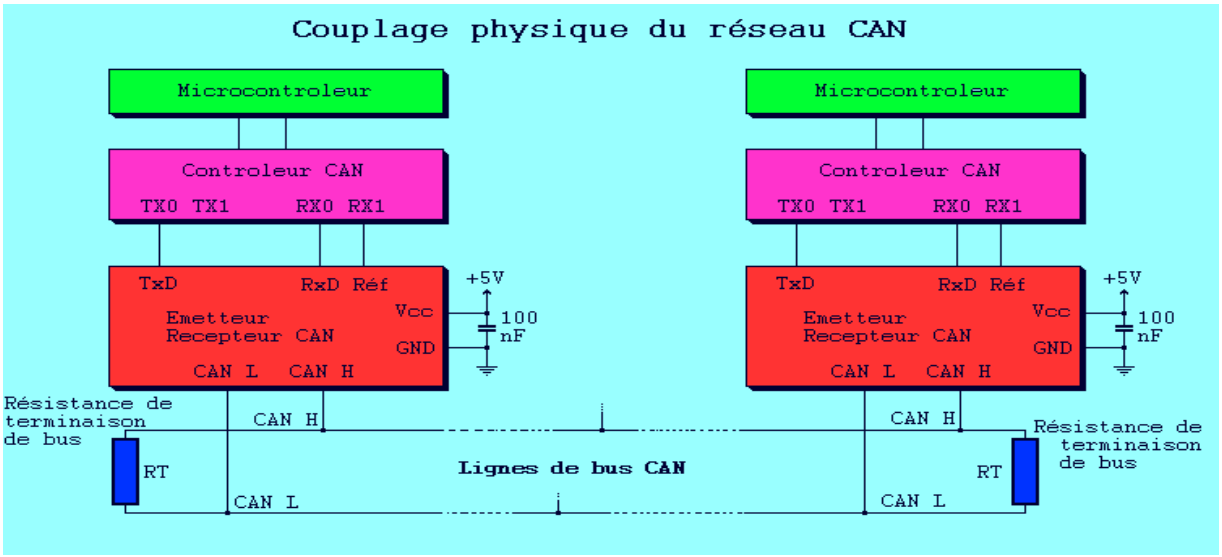
Les sorties digitales du bus CAN-Bigbox sont activées par envoie d'un message CAN en utilisant (SW1*8)+221 de longueur 1.
La sortie **i** doit être validée (ou mise à 0) si le bit **i** avec l'octet de données du message CAN est **1(ou 0).**

Exple : SW1 est validé à 4.Le message CAN avec l'identifieur 253 et l'octet de données 0x04 validera le 3ème bit de sortie. Toutes les autres sorties ne sont pas validées.

LECTURE SUR LES ENTREES DIGITALES

Adresse specifique pour les modules en entrée

Pour lire une entrée digitale depuis le CAN-Bigbox, une requête de données utilisant (**SW1*8**) **+222** doit être envoyée au CAN Bigbox.
Le CAN Bigbox répond par un message de longueur **1**.
Cet octet de données montre les statuts de **8** lignes d'entrée du CAN Bigbox.



Terminaison du bus:

Une résistance de 120 Ohms peut être installée entre les lignes de signal haut (CAN_H) et de signal bas (CAN_L) du bus CAN.

INITIALISATION

Peut se faire avec la commande **INIT**

Les Leds

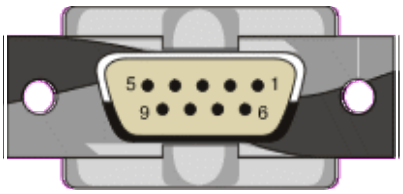
ICANOS utilisent les quatre LED sur CAN BigBox comme indicateurs de statut:

Led 4 : Apres la phase d'initialisation des ICANOS
Led 3 : Apres détection d'une erreur sur le contrôleur CAN
Led 2 : Non utilisée
Led 1 : Si la transmission de données initiée par le CAN Bigbox ne peut être complété (led éteinte par défaut)

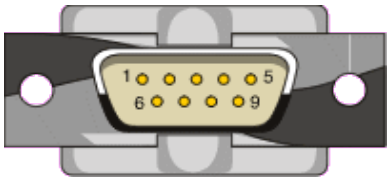
Brochage du CAN

BROCHE	***DESCRIPTION***
1	***NC***
2	CANL
3	EXGND
4	NC
5	NC
6	EXGND
7	CANH
8	NC
9	EXVCC

BROCHAGE DU BUS CAN

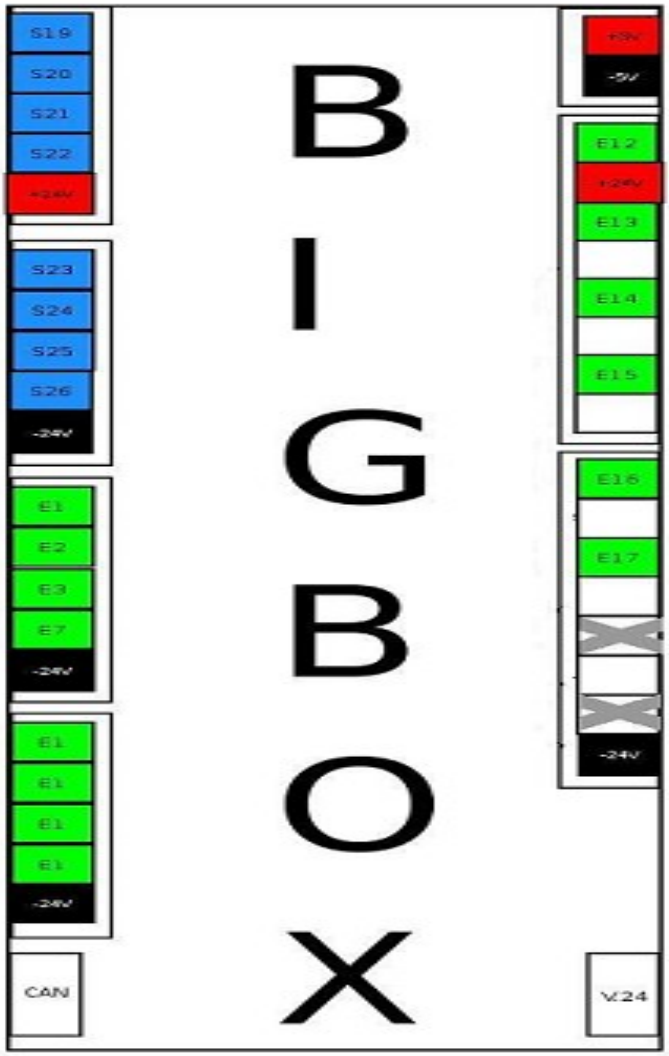


Fiche Femelle



Fiche Mâle

ENTRÉES SORTIES DE LA BIGBOX

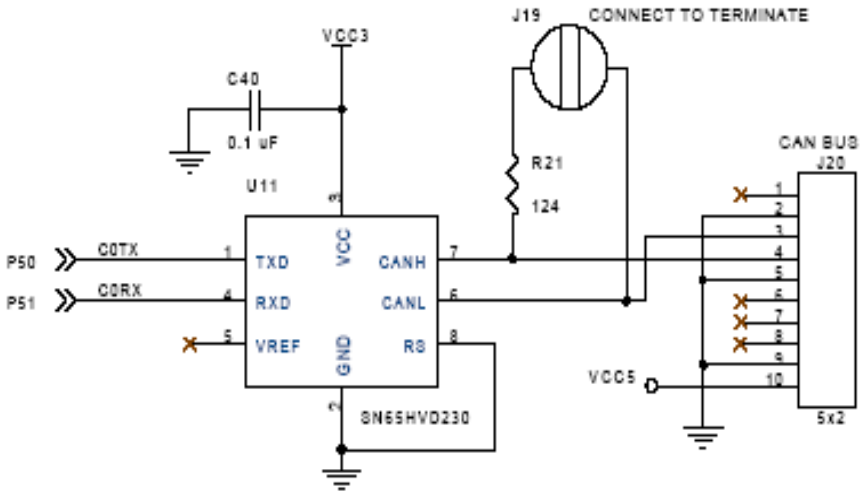


LEGENDE :

: LES SORTIES

: LES ENTRÉES

Câblage d'un bus CAN



En fonction de l'assignation des broches la connexion entre la liaison J20 et la RS282 peut se faire da la sorte:

Les niveaux de transmission du bus sur les lignes physiques sont compris entre 0 et 5 volts, mais afin de garantir la transmission d'un bit dominant (0) ou d'un bit récessif (1), la norme spécifie le codage des niveaux de tensions suivants :

Nature du bit	*Codage de la tension*
1	CAN H < CAN L +0.5V
0	CAN H > CAN L +0.5V

CANDIDAT 3

4.5_Candidat 3 : MENDES Sébastien

1.But

Le but de l'étudiant 3 est de créer une interface graphique qui sera sur une machine appelée le serveur superviseur et utilisée par le superviseur du système, afin qu'il puisse voir les différents états du système à distance.

Pour cela il doit assurer une communication entre le serveur et le micro-contrôleur TINI, via le protocole RPC, pour connaître et afficher ces différents états du système par l'intermédiaire de l'IHM.

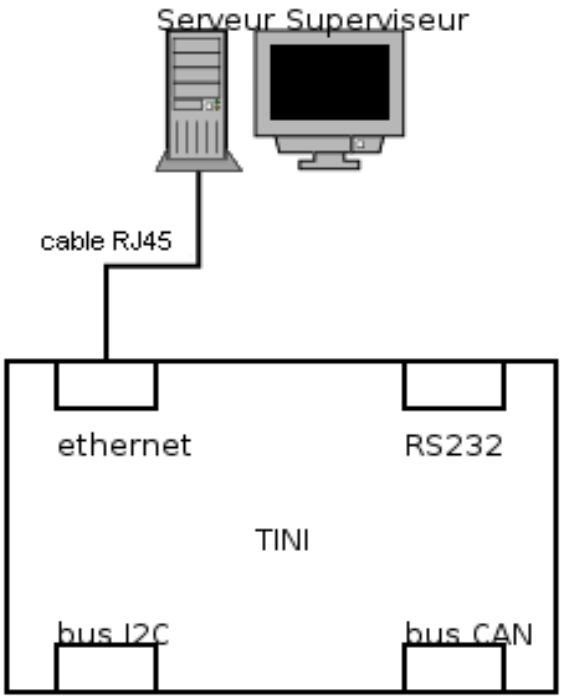
2.Principe

1/ Le système

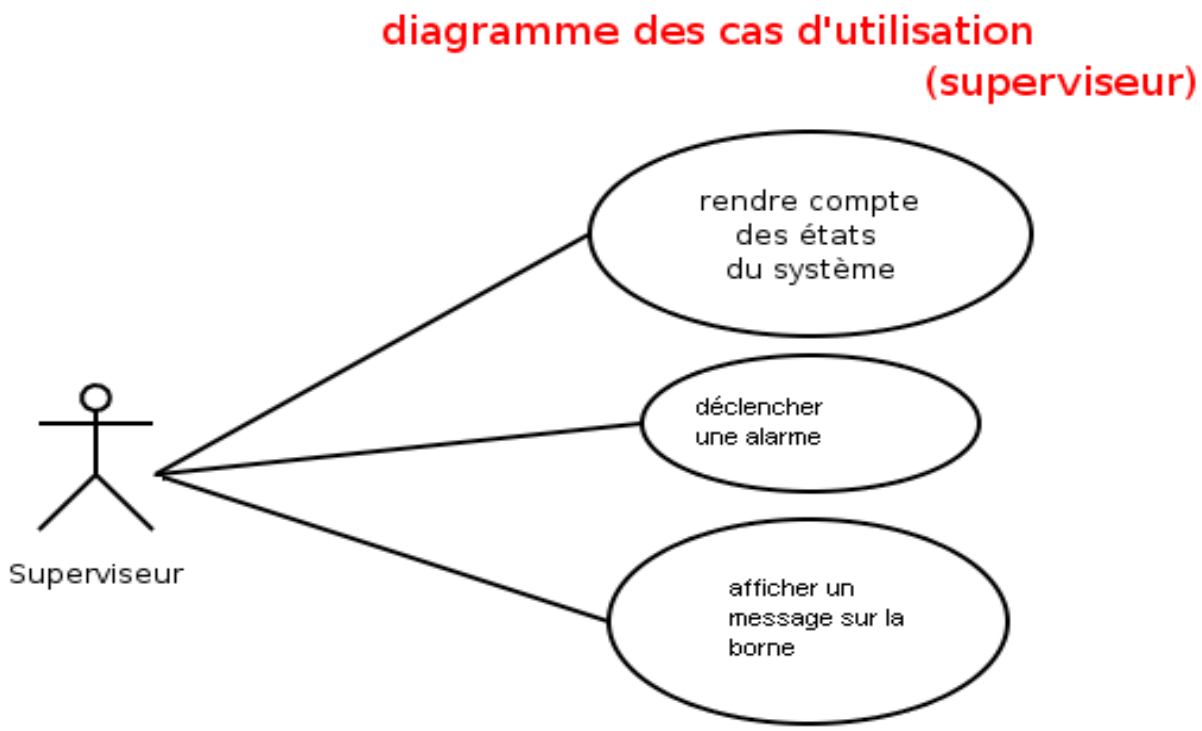
On observe sur ce diagramme que le serveur superviseur communique avec le micro-contrôleur à l'aide d'un câble RJ45 pour une liaison Ethernet. Le serveur se situe à l'extérieur du système. Ce câble est relié de la carte réseau du serveur superviseur, au port Ethernet de la carte de support du micro-contrôleur TINI.

DIAGRAMME STRUCTUREL

(Partie superviseur)



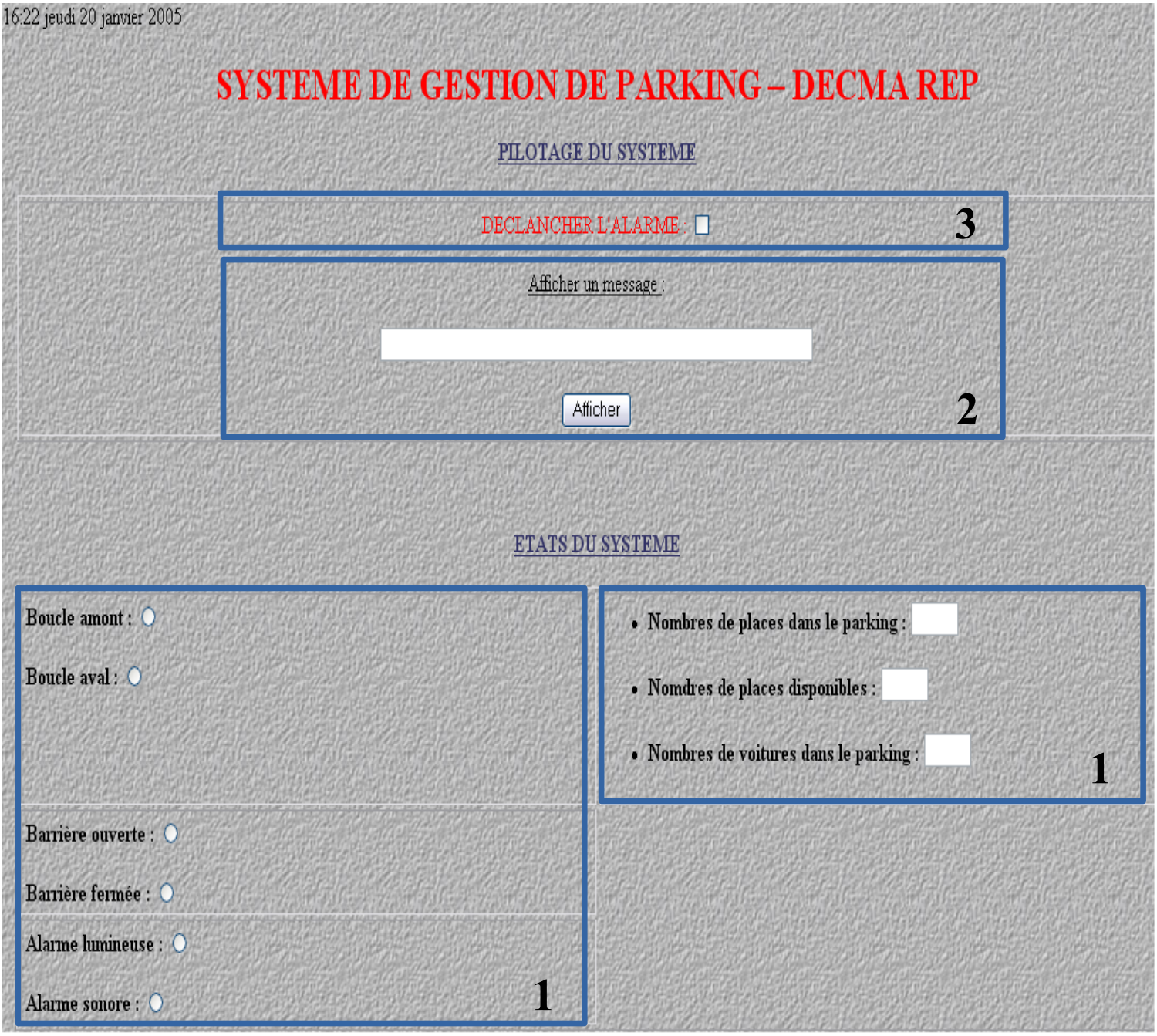
Le serveur récupère et envoie des données à la TINI



Le superviseur disposera d'une interface lui permettant de connaître les états du système que le micro-contrôleur enverra sur le serveur, envoyer un message sur la borne et déclencher l'alarme. Toutes les données envoyées auront pour destination le micro-contrôleur, ces données seront ensuite traitées par celui-ci. Le travail qui se situe au niveau de micro-contrôleur sera traité par l'étudiant n°1, il aura pour tâche de faire fonctionner le système de façon automatique.

Le serveur superviseur, doit pouvoir communiquer avec le micro-contrôleur via le protocole RPC (remote procedure call), qui est un protocole de niveau 5 sur le modèle OSI.

2/ Maquette de l'interface



Cette maquette a été conçu à l'aide de « OpenOffice ». Ce logiciel nous a permis de pouvoir mettre en oeuvre une maquette de notre interface homme machine afin de pouvoir connaître les différents éléments qui vont constituer cette interface.

Selon le cahier des charges, l'interface devra pouvoir:

1/ Afficher les différents états du système:

Les différents états du systèmes seront donc l'état ouvert ou fermé de la barrière, l'état actif ou inactif des boucles amont et aval, l'état active ou inactive de l'alarme lumineuse et sonore, ainsi que le nombre de places libres, le nombre de places occupées et le nombre de places disponibles du parking.

Sur cette maquette, on a mis en place des voyants, qui sont en faite des boutons radio, qui s'allumeront lorsque le capteur correspondant s'allumera. Des champs permettront l'affichage des places libres, occupées et disponibles dans le parking.

2/ Envoyer un message sur la borne:

Possibilité d'écrire un message et a l'aide d'un bouton « envoi », afficher le message sur la borne.

Ceci a été rendu possible grâce à une zone de texte, où l'on entrera le message à envoyer, et un bouton poussoir qui permettra l'envoi du message sur la borne.

3/ déclencher une alarme:

Un clic sur le bouton « Alarme » permettra d'activer l'alarme du système.

Ce bouton est en faite une case que l'on doit cocher. Lorsque cette case sera cocher, l'alarme sera déclencher. Tant que cette croix sera visible, l'alarme sera enclenchée.

Cette maquette a donc été conçue pour répondre au cahier des charges. Les éléments qui compose cette IHM ont été choisi pour simplifier l'utilisation de cette outils.

3/ Interface homme/machine

Contraintes matérielles:

Interface de développement : JBUILDERX

Codage : JAVA

Communication avec le micro-contrôleur : Protocole RPC

3.1/ JBuilderX

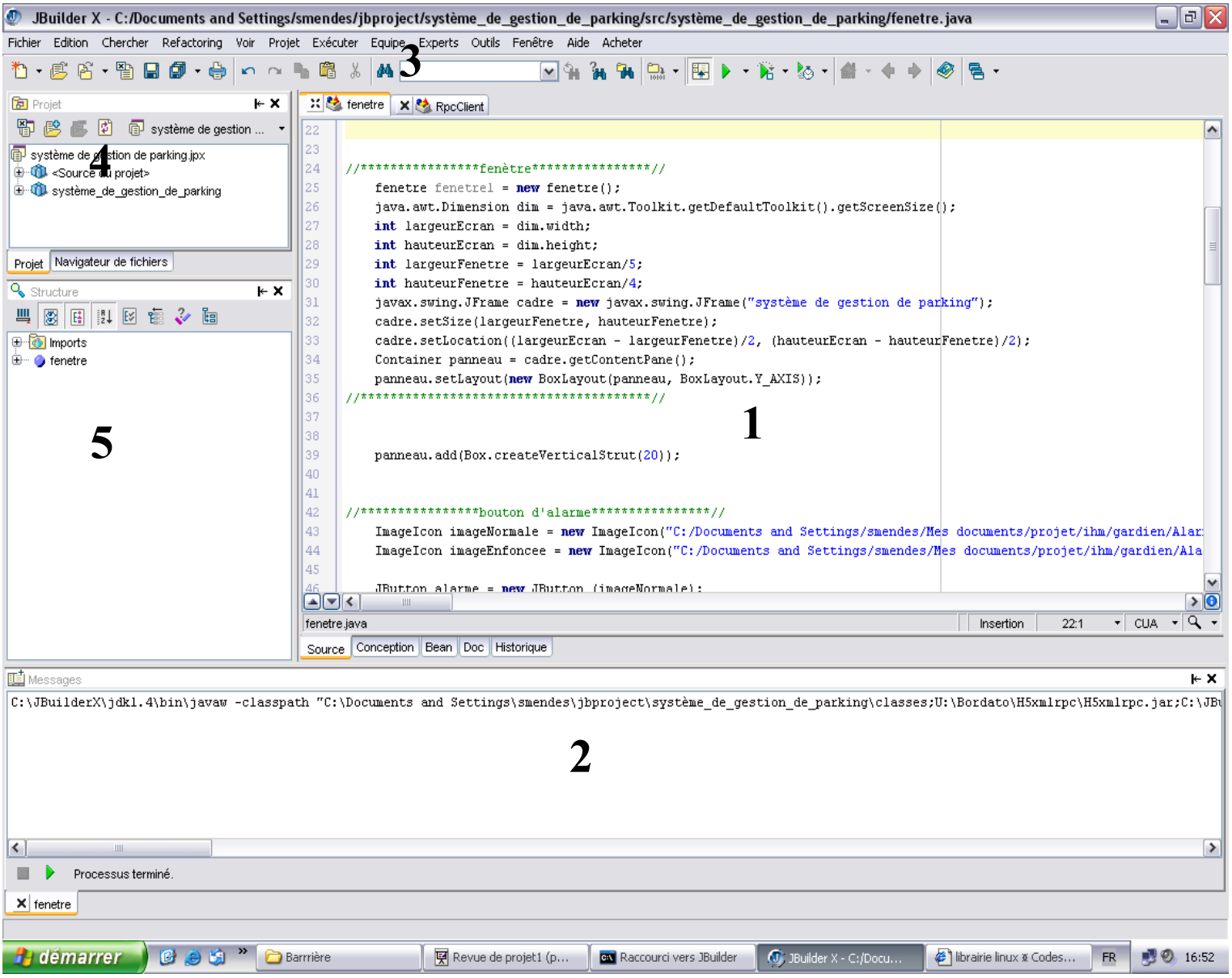
Afin de créer une interface homme machine, il nous a été imposé d'utiliser JbuilderX. Ceci logiciel de développement qui permet de concevoir rapidement des applications en codage JAVA. Il permet d'avoir des indications sur les erreurs et facilite la création, la compilation et l'exécution de ce codage.

Nous avons donc dû télécharger ce logiciel à partir d'Internet sur le site http://www.borland.fr/jbuilder/. Nous avons donc pris JbuilderX, la version gratuite et en Français, qui est plus formellement appelée Jbuilder personnel, qui est une de nos contraintes matériels.

Afin de pouvoir installer ce logiciel sur notre poste de travail, nous avons dû nous enregistrer sur ce même site, pour que l'on puisse recevoir un fichier permettant d'utiliser ce logiciel. Ce fichier est en faite une clé qui permet d'utiliser ce logiciel gratuitement.

Il est nécessaire d'installer une machine virtuelle JAVA pour permettre l'exécution du code JAVA sur notre machine locale. Pour cela, on nous a fournit le J2RE 1.4.2 de SUN Développement, que nous avons dû installer.

Voilà comment ce présente l'interface JbuilderX:



1------- Fenêtre dans laquelle se trouvera le codage que l'on veut effectuer.

2------- Fenêtre dans laquelle se trouveront les messages de compilation.

3------- Barre d'outils

4------- Fenêtre où sont indiquer les différents fichiers enregistrer (nom du projet, fichiers.java).

5------- Fenêtre où les détails, les imports utiliser, des classes, sont affichés.

3.2/ Composants swing

Pour créer notre interface, on nous a imposer d'utiliser les composants swing JAVA. Toute notre interface sera composée de ces éléments. Swing propose de nombreux composants dont certains possèdent des fonctions étendues, une utilisation des mécanismes de gestion d'événements performants (ceux introduits par le JDK 1.1) et une apparence modifiable à la volée (une interface graphique qui emploie le style du système d'exploitation Windows ou Motif ou un nouveau style spécifique à Java nommé Metal).

Tous les éléments de Swing font partie d'un package qui a changé plusieurs fois de nom : le nom du package dépend de la version du J.D.K. utilisée :

- com.sun.java.swing : jusqu'à la version 1.1 beta 2 de Swing, de la version 1.1 des JFC et de la version 1.2 beta 4 du J.D.K.
- java.awt.swing : utilisé par le J.D.K. 1.2 beta 2 et 3
- javax.swing : à partir des versions de Swing 1.1 beta 3 et J.D.K. 1.2 RC1

Les composants Swing forment un nouvelle hériarchie parallèle à celle de l'AWT. L'ancêtre de cette hiérarchie est le composant JComponent. Presque tous ces composants sont écrits en pur Java : ils ne possèdent aucune partie native sauf ceux qui assurent l'interface avec le système d'exploitation : JApplet, JDialog, JFrame, et JWindow. Cela permet aux composants de toujours avoir la même apparence quelque soit le système sur lequel l'application s'exécute.

Tous les composants Swing possèdent les caractéristiques suivantes :

- ce sont des beans
- ce sont des composants légers (pas de partie native) hormis quelques exceptions.
- leurs bords peuvent être changés

La procédure à suivre pour utiliser un composant Swing est identique à celle des composants de la bibliothèque AWT : créer le composant en appelant son constructeur, appeler les méthodes du composant si nécessaire pour le personnaliser et l'ajouter dans un conteneur.

Swing utilise la même infrastructure de classes que AWT, ce qui permet de mélanger des composants Swing et AWT dans la même interface. Sun recommande toutefois d'éviter de les mélanger car certains peuvent ne pas être restitués correctement.

Les composants Swing utilisent des modèles pour contenir leurs états ou leur données. Ces modèles sont des classes particulières qui possèdent toutes un comportement par défaut.

3.3/ Remote Procedure Call (RPC)

Remote Procedure Call traduit en Français donne « appelle de procédures à distances ». Ceci est un protocole d'échange, une façon de s'envoyer des informations qui soit compréhensible par le receveur et donc exploitable.

Ce protocole utilise la couche 5, couche session, du modèle OSI (Oppen Systèmes Interconnection). Ce modèle est une norme qui permet d'interconnecter différents systèmes. Ce modèle comporte 7 couches. Les couches 1, 2, 3 et 4 sont des couches dites basses, elles sont nécessaires à l'acheminement entre les extrémités concernées et dépendent du support physique. Les couches 5, 6 et 7, couches hautes, sont responsable du traitement de l'information relative à la gestion des échanges entre systèmes informatiques. Par ailleurs, les couches 1 à 3 interviennent entre machines voisines, et non entre les machines d'extrémité qui peuvent être séparées par plusieurs routeurs. Les couches 4 à 7 sont au contraire des couches qui n'interviennent qu'entre hôtes distants.

Voici donc les 7 couches du modèle OSI:

7	Application	NFS/NIS
6	Présentation	XDR
5	Session	RPC
4	Transport	TCP/UDP
3	Réseau	IP
1/2	Liaison/ Physique	Ethernet/FDDI

RPC est un protocole décrivant le passage de paramètres et la sélection de la fonction distante à exécuter ainsi que la récupération de la valeur résultat de la fonction.

Ce système s'efforce de maintenir le plus possible la sémantique habituelle des appels de

fonction, autrement dit tout doit être le plus transparent possible pour le programmeur. Pour que cela ressemble à un appel de fonction local, il existe dans le programme client une fonction locale qui a le même nom que la fonction distante et qui, en réalité, appelle d'autres fonctions de la bibliothèque RPC qui prennent en charge les connexions réseaux, le passage des paramètres et le retour des résultats. De même, côté serveur il suffira (à quelques exceptions près) d'écrire une fonction comme on en écrit tous les jours, un processus se chargeant d'attendre les connexions clientes et d'appeler votre fonction avec les bons paramètres. Il se chargera ensuite de renvoyer les résultats. Les fonctions qui prennent en charge les connexions réseaux sont des "stub". Il faut donc écrire un stub client et un stub serveur en plus du programme client et de la fonction distante. Côté serveur, un objet similaire nommé lie permet de réaliser le même type d'opération côté serveur.

3.Application

1/ Codage de l'interface homme/machine

1.1/ Le Cadre

Pour coder cette interface nous avons dû tout d'abord créer un cadre. Ce cadre à donc été dimensionner et équipée d'un conteneur qui va permettre d'insérer les différents éléments que constituera notre interface. Ce cadre sera le contenu de la fenêtre qui sera notre interface.

```
fenetre fenetre1 = new fenetre();
```

Permet de créer un cadre.

```
java.awt.Dimension dim = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
int largeurEcran = dim.width;
int hauteurEcran = dim.height;
int largeurFenetre = largeurEcran/5;
int hauteurFenetre = hauteurEcran/4;
```

Permet de dimensionner le cadre.

```
javax.swing.JFrame cadre = new javax.swing.JFrame("système de gestion de parking");
```

Permet de donner un nom a la fenêtre.

```
cadre.setSize(largeurFenetre, hauteurFenetre);
cadre.setLocation((largeurEcran - largeurFenetre)/2, (hauteurEcran – hauteurFenetre)/2);
Container panneau = cadre.getContentPane();
panneau.setLayout(new BoxLayout(panneau, BoxLayout.Y_AXIS));
```

Créer le conteneur de ce cadre qui sera représenter par la variable *panneau*.

Les boutons poussoirs, les zones de textes, les labels, seront intégrés au conteneur de la fenêtre.

Ci-dessous nous allons explique comment mettre en place ces composants et comment les intégrés au conteneur.

1.2/ Les boutons poussoirs

```
JButton alarme = new JButton ();
//création d'un bouton qui sera représenter par la variable alarme.
panneau.add (alarme);
//add ajoute au conteneur panneau le bouton poussoir alarme.
```

Il est possible d'afficher une image sur ce bouton à l'aide de cette ligne de code:

```
ImageIcon imageNormale = new ImageIcon("<chemin de l'image a afficher>");
//Pour permettre ce connaître l'emplacement de l'image à afficher
JButton alarme = new JButton (imageNormale);
//Change l'image standard du bouton poussoir alarme.
```

Si l'on veut garder l'image standard d'un bouton et juste lui donner un nom, il suffit de l'indiquer à sa création:

```
JButton envoi = new JButton ("envoyer");
//Affichera envoyer sur le bouton poussoir envoi.
```

1.3/ Les labels

Les labels permettent d'afficher du texte ou une image sur la fenêtre de notre interface.
Pour afficher du texte, cette ligne de code est nécessaire:

```
panneau.add (new JLabel ("NOMBRES DE PLACES DANS LE PARKING"));
//Le texte écrit entre guillemets sera ajouté au panneau.
```

Pour afficher une image, nous devons procédé comme ceci:

```
ImageIcon imageStandard = new ImageIcon("<chemin de l'image a afficher>");
//La variable imageStandard représentera l'image que l'on désire afficher.
JLabel standard = new JLabel(imageStandard);
//Crée le label représenté par la variable standard à l'aquelle on ajoute l'image.
panneau.add(standard);
//Ajoute cette image au conteneur.
```

1.4/ Les champs de saisie de texte

Des composants permettent de saisir du texte, ce sont des champs de saisie de texte. Pour les mettre en place, nous devons faire appel au composant JtexteField:

```
JTextField longueurTexte = new JTextField ();
//Crée le champ de saisie de texte représenter par la variable  longueurTexte.
panneau.add (longueurTexte);
//Ajoute ce composant au panneau.
```

1.5/ L'affichage

L'affichage de ces composants se font à l'aide de deux choses:

1/ Le cadre que l'on a créer doit être insérer dans une fenêtre comme dans cette exemple:

```
cadre.pack();
```

2/ Puis afficher sur l'écran de notre machines comme ceci:

```
cadre.show();
```

Pour que l'application de ce codage s'arrête lorsque la fenêtre ce ferme, nous devons utiliser cette ligne de code:

```
cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

4.Conclusion

La plupart du Matériel et des différents éléments que nous avons utiliser nous a été imposé. Nous aurions pût utiliser un langage différent ainsi qu'un autre environnement de développement pour répondre aux attentes du cahier des charges.

Un autre protocole aurait aussi pût être utiliser pour permettre au serveur superviseur de communiquer avec le micro-controleur.

Pour réaliser cette application, le stricte minimum a été mis en place. L'utilisation d'une machine équiper de logiciels gratuit et d'un câble réseau pour la reliée au micro-controleur.

5. Annexe

```
package système_de_gestion_de_parking;

import java.awt.*;
import javax.swing.*;



public class fenetre
{
 /* public fenetre()
  {
  }*/
   public static void main(String[] args) {


    int nbVoitures = 0;
    int lib = 0;
    String message = new String ();




//****************fenètre****************//
    fenetre fenetre1 = new fenetre();
    java.awt.Dimension dim = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
    int largeurEcran = dim.width;
    int hauteurEcran = dim.height;
    int largeurFenetre = largeurEcran/5;
    int hauteurFenetre = hauteurEcran/4;
```

```
    javax.swing.JFrame cadre = new javax.swing.JFrame("système de gestion de parking");
    cadre.setSize(largeurFenetre, hauteurFenetre);
    cadre.setLocation((largeurEcran - largeurFenetre)/2, (hauteurEcran - hauteurFenetre)/2);
    Container panneau = cadre.getContentPane();
    panneau.setLayout(new BoxLayout(panneau, BoxLayout.Y_AXIS));
//************************************//


    panneau.add(Box.createVerticalStrut(20));


//****************bouton d'alarme***************//
    ImageIcon imageNormale = new ImageIcon("C:/Documents and Settings/smendes/Mes
documents/projet/ihm/gardien/AlarmeOffN1.gif");
    ImageIcon imageEnfoncee = new ImageIcon("C:/Documents and Settings/smendes/Mes
documents/projet/ihm/gardien/AlarmeOnN2.gif");

    JButton alarme = new JButton (imageNormale);
    alarme.setPressedIcon(imageEnfoncee);
    panneau.add (alarme);
//******************************************//


    panneau.add(Box.createVerticalStrut(20));


//****************envoi d'un message sur la borne***************//
    JLabel affiche = new JLabel ("Afficher un message :");
    panneau.add (affiche);


    JTextField longueurTexte = new JTextField (message);
    panneau.add (longueurTexte);


    JButton envoi = new JButton ("envoyer");
   // envoi.setText(message);
    panneau.add (envoi);

//************************************************************//


    panneau.add(Box.createVerticalStrut(20));


//****************places dans le parking***************//
    panneau.add (new JLabel ("NOMBRES DE PLACES DANS LE PARKING"));
```

```
   panneau.add(Box.createVerticalStrut(20));

   panneau.add (new JLabel("parking de 30 places:"));

   panneau.add(Box.createVerticalStrut(7));

   ImageIcon imagePark =  new ImageIcon("C:/Documents and Settings/smendes/Mes
documents/projet/ihm/gardien/complet1.gif");
   ImageIcon imagePark2 =  new ImageIcon("C:/Documents and Settings/smendes/Mes documents/
projet/ihm/gardien/nocomplet.gif");
   //ImageIcon imagePark1 =  new ImageIcon("C:/Documents and Settings/smendes/Mes
documents/projet/ihm/gardien/complet2.gif");
   if (nbVoitures == 30)
   {
       JLabel PComp = new JLabel(imagePark);
       panneau.add(PComp);
   }
   else
   {
    JLabel PnoComp = new JLabel (imagePark2);
    panneau.add (PnoComp);
   }


   panneau.add(Box.createVerticalStrut(20));


   panneau.add(new JLabel("places occupées:"));
   panneau.add(new JLabel (""+nbVoitures));


   panneau.add(Box.createVerticalStrut(20));


   panneau.add(new JLabel ("places libres:"));
   lib = 30 - nbVoitures;
   panneau.add(new JLabel (""+lib));
//*********************************************************//


   panneau.add(Box.createVerticalStrut(20));


//****************Etats du parking*****************//
   panneau.add(new JLabel ("Etat de la barrière :"));
   panneau.add(Box.createVerticalStrut(7));
```

```
    ImageIcon imageStandard = new ImageIcon("C:/Documents and Settings/smendes/Mes
documents/projet/ihm/borne/NoPassage.gif");
    JLabel standard = new JLabel(imageStandard);
    panneau.add(standard);
//***********************************************//


    panneau.add(Box.createVerticalStrut(20));


    cadre.pack();
    cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    cadre.show();
  }
}
```