Activité Pratique

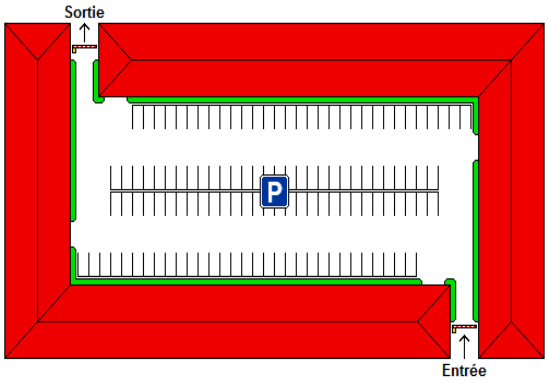
Gestion d'un Parking sous QT5

Table des matières

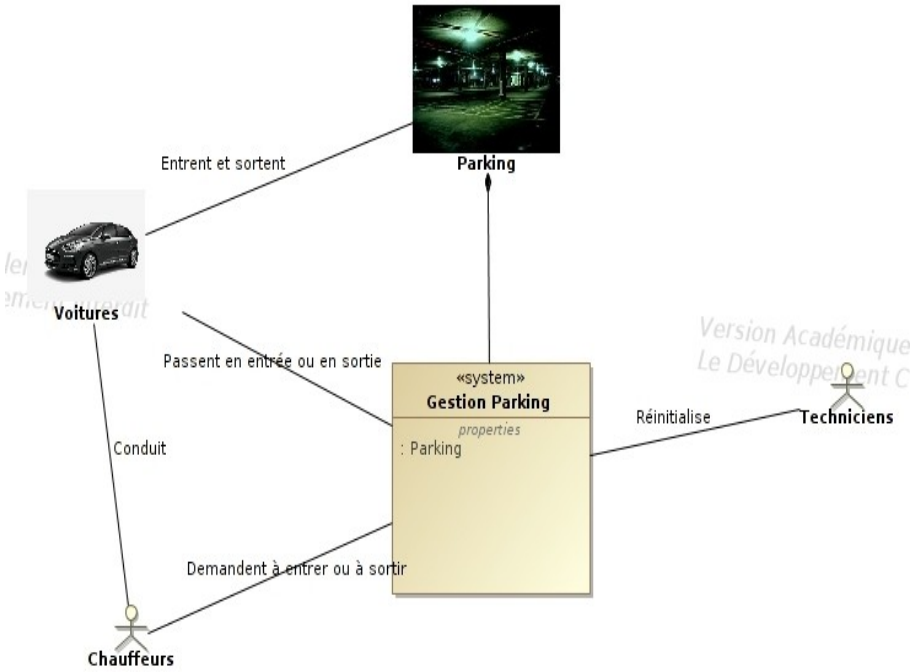
1 - MISE EN SITUATION

Le collège d'une grande commune est implanté au coeur de sa cité historique. Ce centre ville étant très touristique, la municipalité a décidé de rendre le stationnement payant et limité à 2h dans les rues.
Heureusement, le personnel de l'établissement scolaire dispose d'un parking gratuit situé dans une cour intérieure.
Mais en raison du manque de place entre les bâtiments, les voies d'accès sont placées à deux endroits distincts. l'une sert d'entrée et l'autre de sortie. Ces deux voies sont contrôlées par une barrière. Les possibités d'entrer et de sortir se font à l'aide d'une commande par emetteur/récepteur radio pour l'automobiliste. Un système de détection de présence de véhicules, placé après la barrière, par boucle de détection permet de savoir si le véhicule est passé ou non.

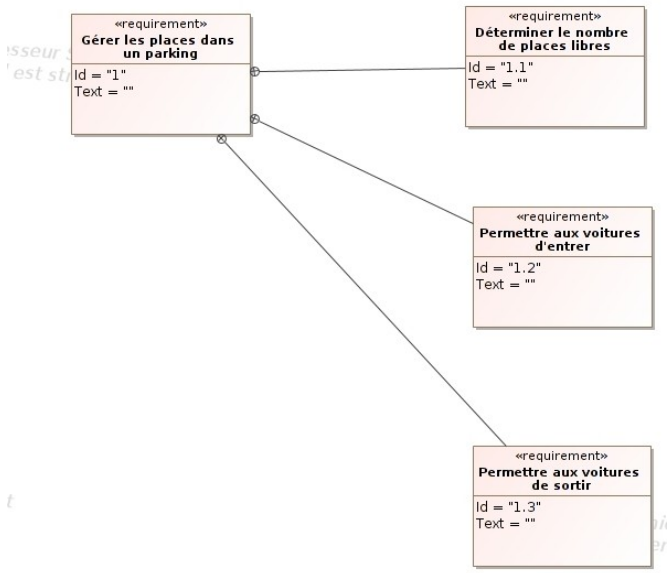
Pour que le parking ne soit pas sursaturé, un système de comptage des véhicules présents a été mis en place. Ce système permet de connaître le nombre de places libres dans le parking.



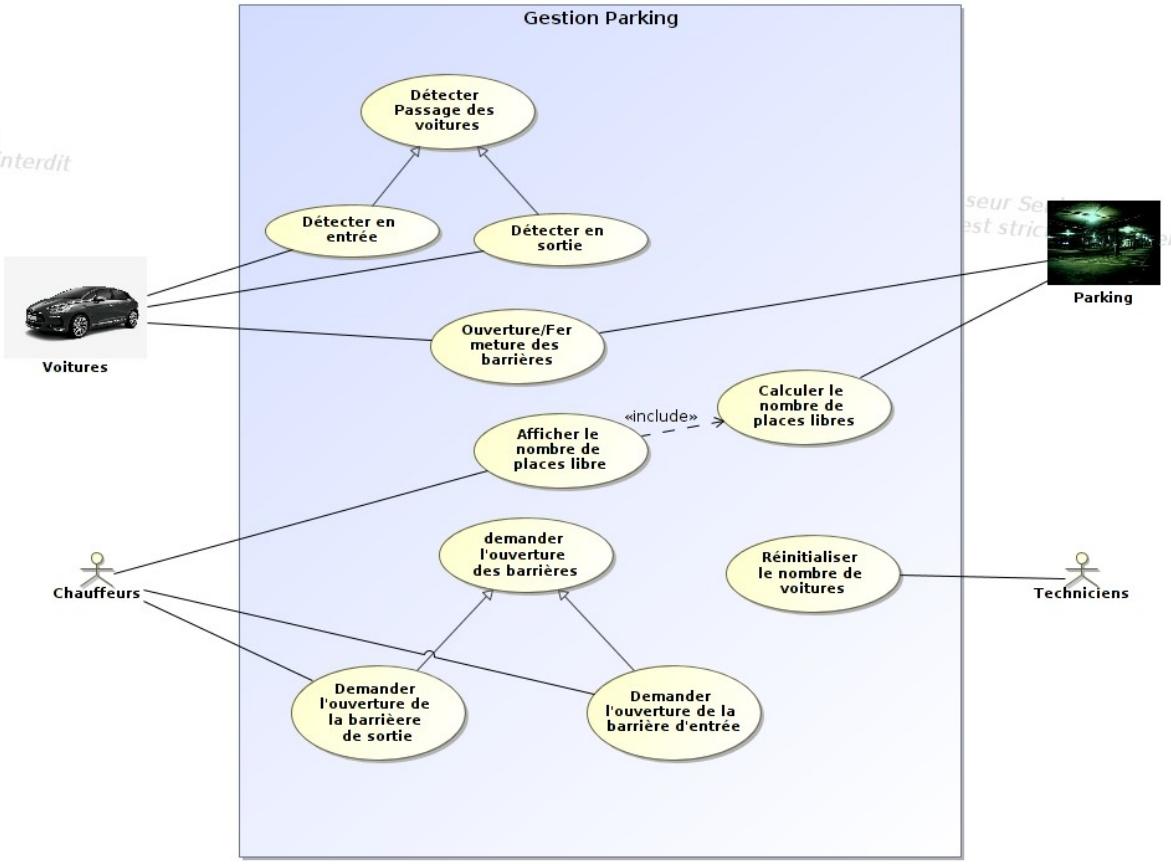
2 - DIAGRAMME DE CONTEXTE



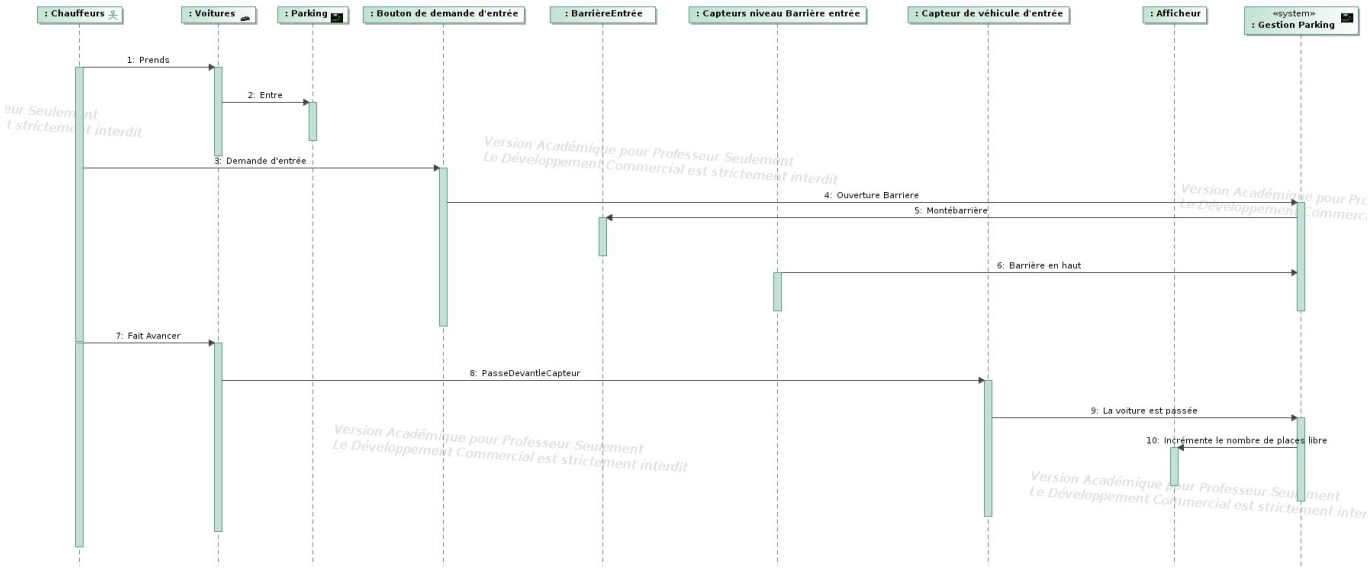
3 - DIAGRAMME D'EXIGENCE



4 - DIAGRAMME DE CAS D'UTILISATION

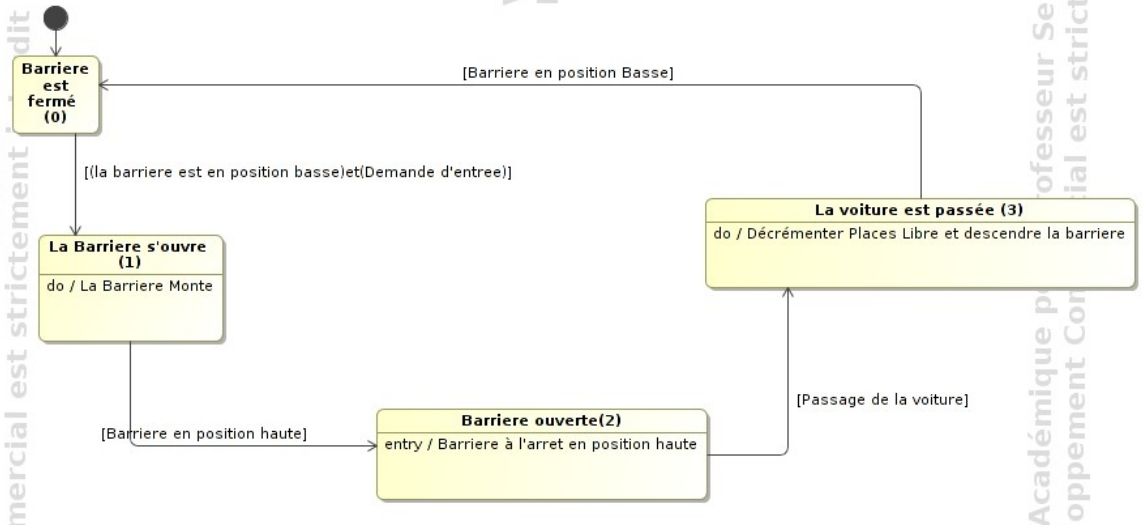


5 - DIAGRAMMES DES SEQUENCES COTÉ UTILISATEUR



6 - DIAGRAMME D'ÉTAT TRANSITIONS POUR UNE BARRIÈRE

6.1 - Diagramme d'état transition pour la barriere d'entree

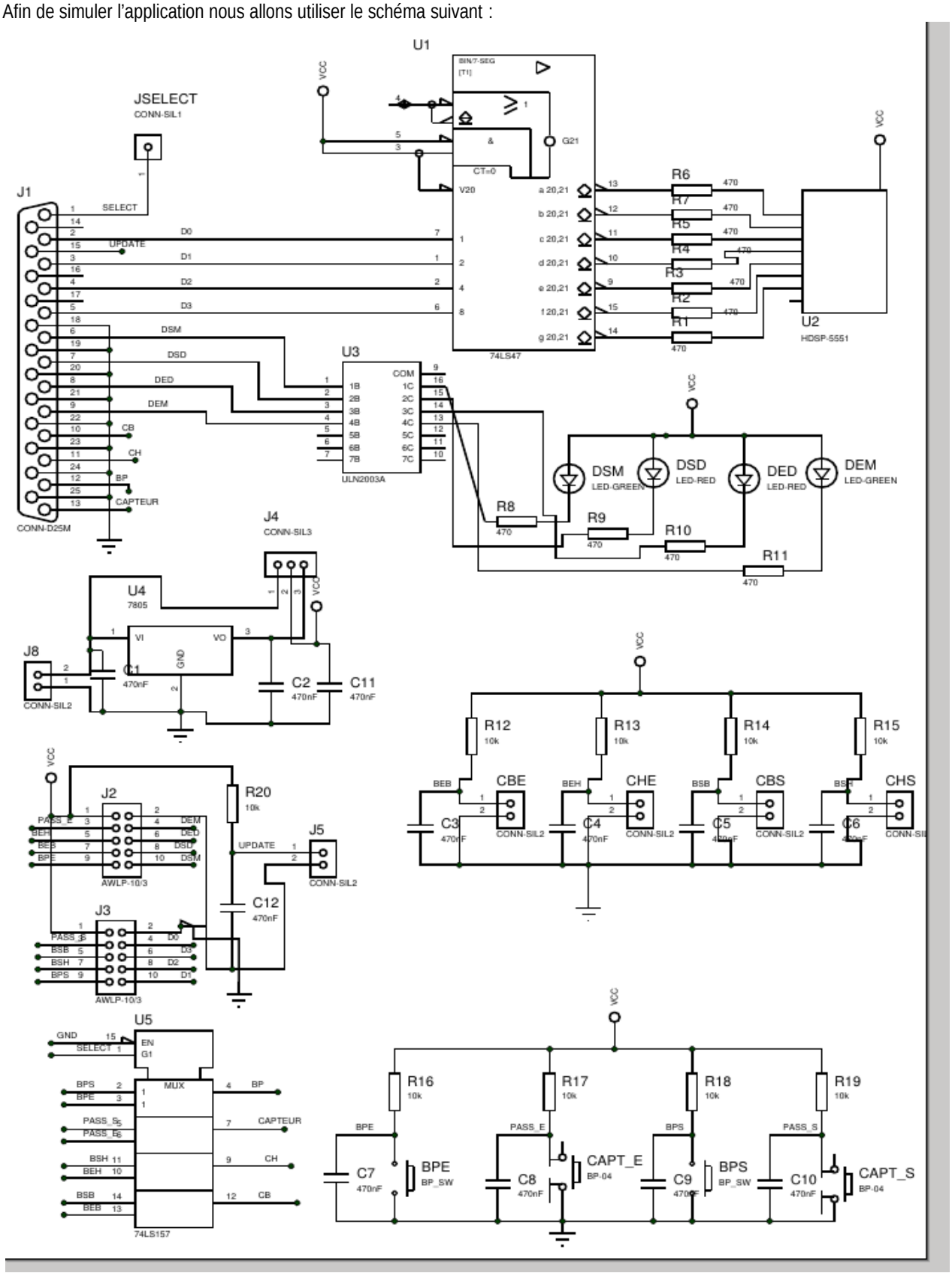


6.2 - Diagramme d'état transition pour la barriere de sortie

Réaliser ce Diagramme d'état transition

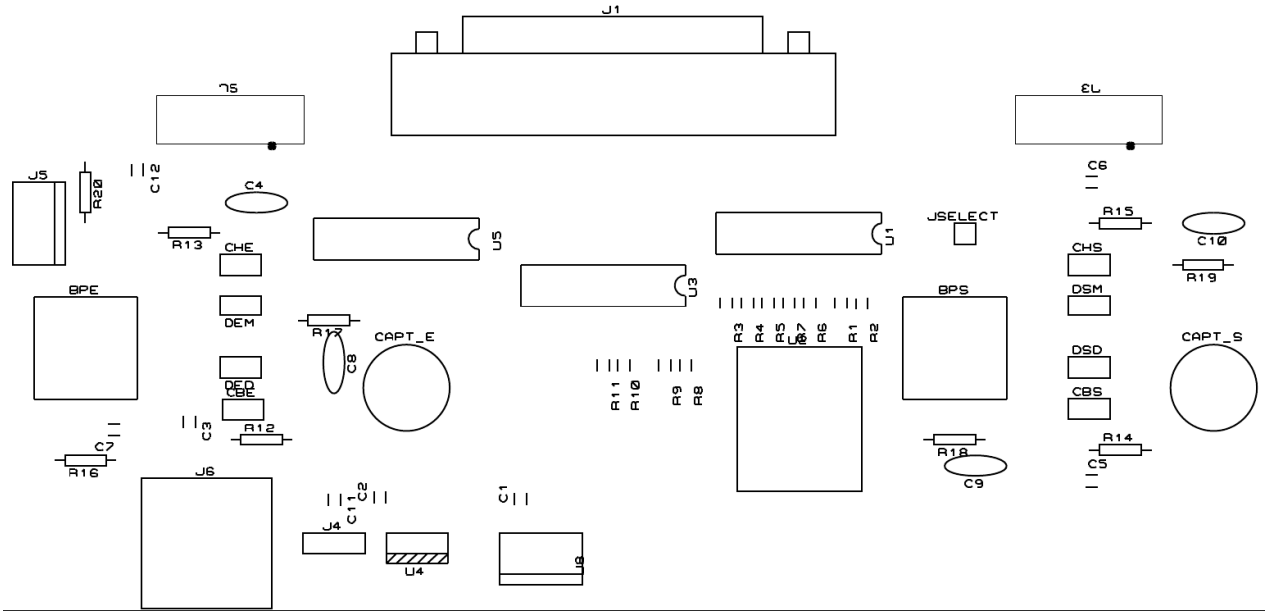
7 - RÉALISATION DE L'APPLICATION

Afin de simuler l'application nous allons utiliser le schéma suivant :



- J1 Correspond au port parallèle du PC.
- BPE et BPS simule les capteurs de demande d'entrée et de sortie
- CAPT_E et CAPT_S simule les capteurs de détection de véhicules en entrée et en sortie.
- CBE, CHE, CBS, CHS simulent les capteurs de position haute ou basse des barrières.
- DSM, DSD, DEM, DED sont des indicateurs qui précisent si les barrières de sortie ou d'entrée sont en train de monter ou de descendre.

L'implantation des composants est la suivante :



Organisation du port parallèle :

Registre DATA
Adresse 378 (LPT1) et 278 (LPT2)

Bits	b7	b6	b5	b4	b3	b2	b1	b0
Broches	9	8	7	6	5	4	3	2

Remarques :

- Les sorties sont latchées (le dernier mot écrit peut être lu à la même adresse)
- Les sorties ne sont pas inversées
- Les sorties ne doivent pas être forcée par le périphérique qui y est connecté (sortie totem-pôle)

Registre STATUS
Adresse 379 (LPT1) et 279 (LPT2)

Bits	b7	b6	b5	b4	b3	b2	b1	b0
Broches	11	10	12	13	15	-	-	-

Remarques :

- Seule la broche 11 (bit b7) est inversée.
- Sur certaines cartes compatibles la broche 15 n'est pas connectée.

Registre STATUS
Adresse 379 (LPT1) et 279 (LPT2)

Bits	b7	b6	b5	b4	b3	b2	b1	b0
Broches	11	10	12	13	15	-	-	-

Remarques :

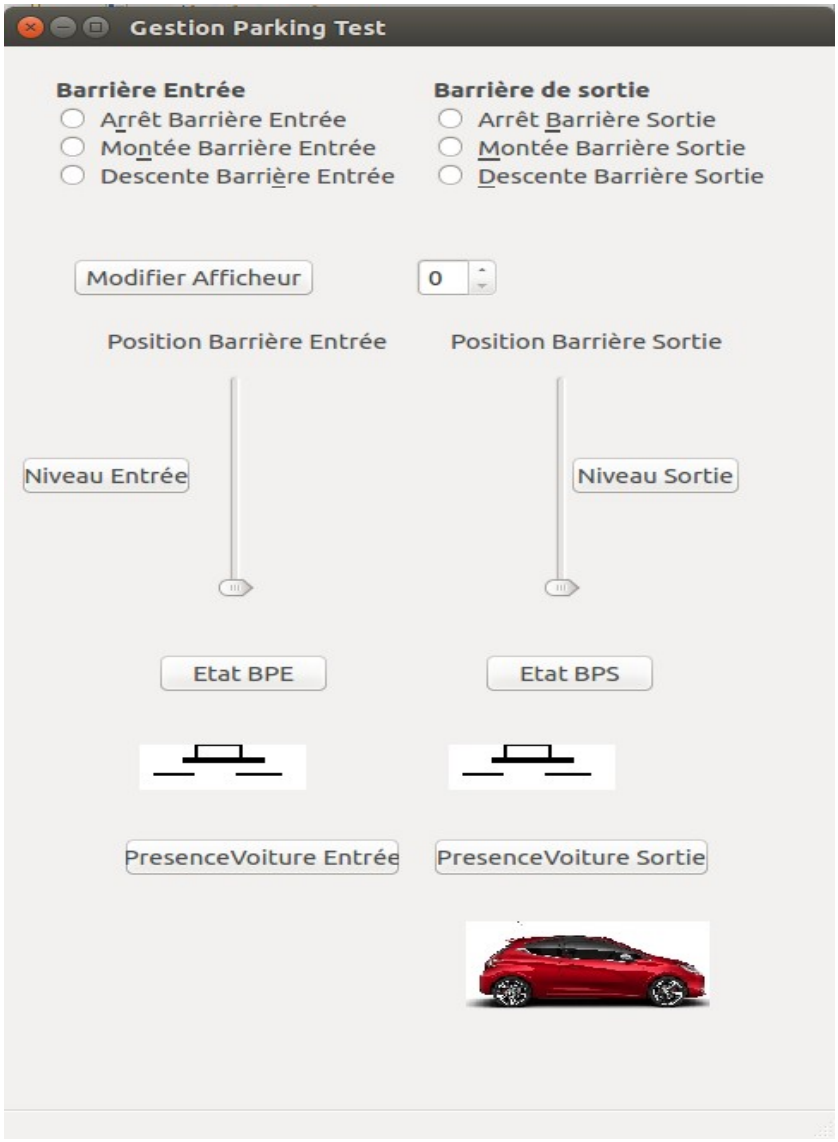
- Seule la broche 11 (bit b7) est inversée.
- Sur certaines cartes compatibles la broche 15 n'est pas connectée.

Description	N° Broche PC	Direction
-Strobe	1	<>
+Data-Bit 0	2	<>
+Data-Bit 1	3	<>
+Data-Bit 2	4	<>
+Data-Bit 3	5	<>
+Data-Bit 4	6	<>
+Data-Bit 5	7	<>
+Data-Bit 6	8	<>
+Data-Bit 7	9	<>
-Acknoledge	10	<
+Busy	11	<
+Paper End	12	<
+Select	13	<
-Auto Feed	14	>
-Error/Fault	15	<
-Printer Init	16	>
-Select	17	>
Ground	18-25	

8 - PREMIERS PROGRAMMES :

8.1 - Tests des entrées/sorties

Écrire un programme TestESDlg permettant de tester vos entées Sortie à en utilisant l'IHM ci dessous :



Pour ceci vous aurez à votre disposition la classe PortLpt.

Réalisation et test des boutons Radio :

dans un premier temps vous ferez les boutons radio et la commande des afficheurs et les commandes associées.

Par exemple pour la commande de la Descente de la barriere en entrée (Led Rouge DED) correspondant au bouton radio « Descente Barrière Entrée » vous utiliserez l'instruction suivante :

```
mlpt->setBit(0x40,true);
```

la Led Rouge DED est commandé par la broche 8 du connecteur DB25 ce qui correspond à la Data 6 de la commande setBit du port parrallele

Data 7 (pin 9)	Data 6 (pin 8) Led DED	Data 5 (pin 7)	Data 4 (pin 6)	Data 3 (pin 5)	Data 2 (pin 4)	Data 1 (pin 3)	Data 0 (pin 2)
				Afficheur			

vous pourrez aussi utiliser le #define
#define DESCENTEBARRIEREENTREE 0x40
et ainsi vous pourrez utiliser l'instruction mlpt→setBit(DESCENTEBARRIEREENTREE,true);
l'instruction mlpt→setBit(DESCENTEBARRIEREENTREE,false); arrêtera la barrière la descente de la barrière en entrée

Compléter le Tableau suivant :

MONTEEBARRIEREENTREE	DEM	
DESCENTEBARRIEREENTREE	DED	0x40
MONTEEBARRIERESORTIE	DSM	
DESCENTEBARRIERESORTIE	DSD	

A Partir de là vous pourrez réaliser toutes les actions associées aux boutons radio.

Lecture des différents capteurs :

Technique pour intégrer une image dans une boite de dialog :
1 : Choisir un gridLayout
2 : Transformer le gridLayout en gridWidget
3 : Ajouter un label dans le gridWidget
4 : Modifier l' « ObjectName » du gridLayout, du gridWidget et du label.
5 : Ajouter le code suivant :

```
QPixmap *pixmap_img;
pixmap_img = new QPixmap("ouvert.png");
// mon_logo se trouve dans le repertoire qui contient mon exe

ui->labelBPE->setPixmap(*pixmap_img);
ui->gridWidgetBPE->setLayout(gridLayoutBPE);
ui->gridWidgetBPE->show();
delete pixmap_img;
```

Pour lire le capteur en position basse de la barrière vous utiliserez l'instruction suivante :

```
#define CAPTEURBASENTREE 0x40
    bool positionbassebarriereentree=mlpt→getBit(CAPTEURBAS,false)
```

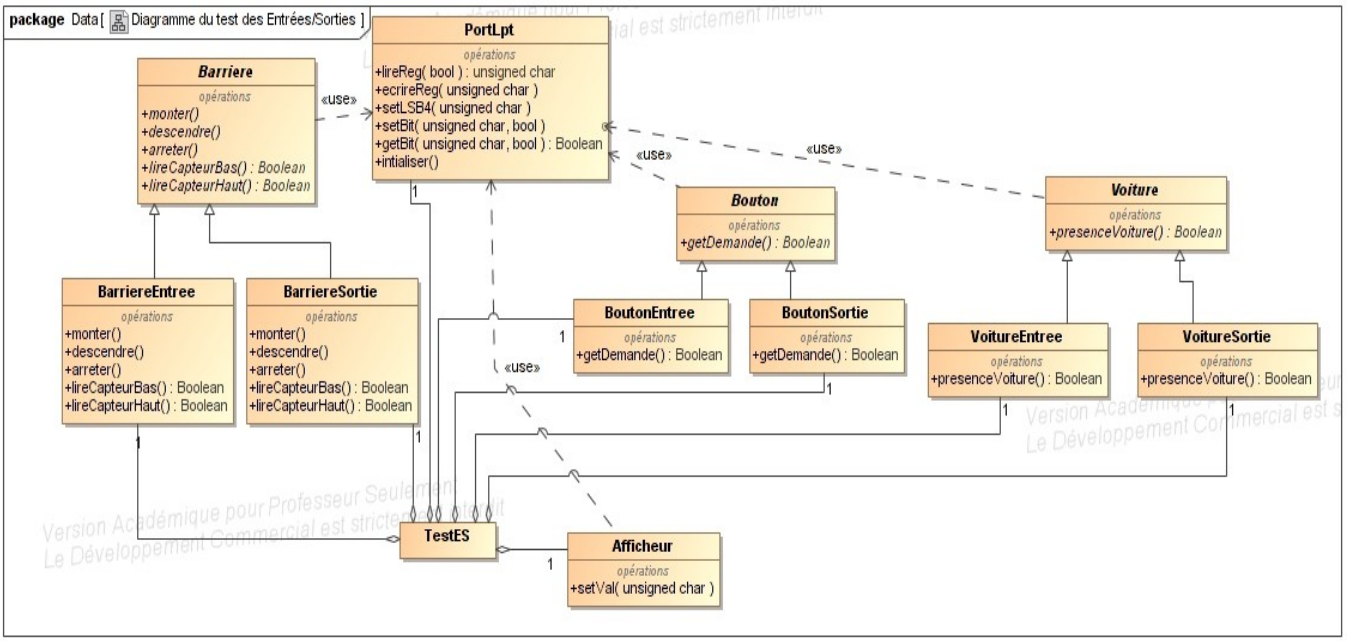
en vous servant du fonctionnement du registre status compléter les valeurs correspondant aux capteurs ci dessous :

CAPTEURBASENTREE	CBE	0x40
CAPTEURHAUTENTREE	CHE	
CAPTEURBASENTREE	CBS	
CAPTEURHAUTENTREE	CHS	

Compléter votre programme de test des différents éléments du système.

8.2 - Réalisation des Méthodes associées aux boutons et aux barrières, aux Afficheurs et aux Leds correspondants à la montée et à la descente des barrières

On vous propose de réaliser les tests en utilisant le diagramme de classes ci dessous :
Pour ceci vous réaliserez un nouveau projet :



Les classes barrière et bouton sont composées de méthodes abstraites (méthodes écrites en italique dans les opérations du diagramme de classe), Justifier ce choix.

- Écrire pour chaque classe le corps associé vous n'oublierez d'intégrer l'association correspondant à la classe PortLpt cette association se fera lors de l'instanciation de la classe.

- Intégrer la classe Afficheur dans le programme de test.

```
/** Afficheur.h*/
#ifndef CLASSES_AFFICHEUR_H_
#define CLASSES_AFFICHEUR_H_
#include "PortLpt.h"

class Afficheur {
        PortLpt *mlpt;
public:
        Afficheur(PortLpt* lpt);
        virtual ~Afficheur();
        void setVal(unsigned char val);
};

#endif /* CLASSES_AFFICHEUR_H_ */
```

```
/*Afficheur.cpp*/

#include "Afficheur.h"

Afficheur::Afficheur(PortLpt* lpt) {
        // TODO Auto-generated constructor stub
        mlpt=lpt;

}

Afficheur::~Afficheur() {
        // TODO Auto-generated destructor stub
}

void Afficheur::setVal(unsigned char val){
        mlpt->setLSB4(val);
}
```

- Intégrer la classe Bouton et ses deux classes dérivées BoutonEntree et BoutonSortie dans le programme de test.

/* Bouton.h*/

```
#ifndef CLASSES_BOUTON_H_
#define CLASSES_BOUTON_H_
#include "PortLpt.h"
#define BOUTONDEMANDE 0x20

class Bouton {
protected:
	PortLpt *mlpt;
	bool e_s;
public:
	Bouton(PortLpt * lpt);
	virtual ~Bouton();
	virtual bool getDemande();
};

#endif /* CLASSES_BOUTON_H_ */
```

/*Bouton.cpp*/

```
#include "Bouton.h"

Bouton::Bouton(PortLpt * lpt) {
	// TODO Auto-generated constructor stub
	mlpt=lpt;
}

Bouton::~Bouton() {
	// TODO Auto-generated destructor stub
}

bool Bouton::getDemande(){

}
```

/*BoutonEntree.h*/

```
#ifndef CLASSES_BOUTONENTREE_H_
#define CLASSES_BOUTONENTREE_H_

#include "Bouton.h"
#include "PortLpt.h"

class BoutonEntree: public Bouton {

public:
	BoutonEntree(PortLpt *lpt);
	virtual ~BoutonEntree();
	bool getDemande();
};

#endif /* CLASSES_BOUTONENTREE_H_ */
```

/*BoutonEntree.cpp*/

```
#include "BoutonEntree.h"
//PointCol::PointCol(int abs,int ord, unsigned char r,unsigned char g, unsigned char b):Point(abs,ord)

BoutonEntree::BoutonEntree(PortLpt *lpt):Bouton(lpt) {
	// TODO Auto-generated constructor stub
}

BoutonEntree::~BoutonEntree() {
	// TODO Auto-generated destructor stub
}

bool BoutonEntree::getDemande(){
	if (mlpt->getBit(BOUTONDEMANDE,false))
		return false;
	return true;
}
```

- Faire la même chose avec la classe Voiture et ses deux classes dérivées .

- Faire la même chose avec la classe Barrière et ses deux classes dérivées .

Peut on supprimer les boutons correspondant à une demande d'information pour obtenir l'état des boutons poussoirs et de la la barrière ? Si oui proposer une solution.

8.3 - Algorithme associé au diagramme d'état transition.

IHM pour la réalisation de l'algorithme :

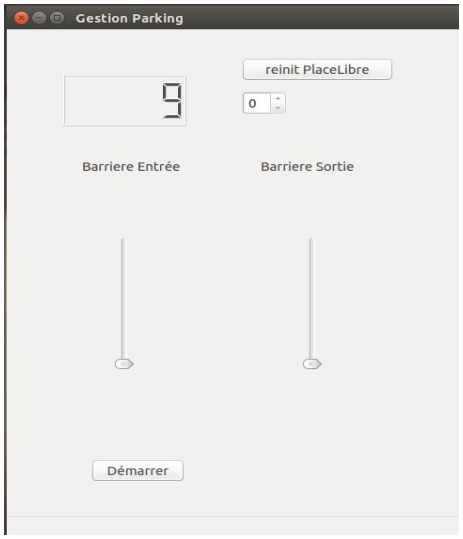
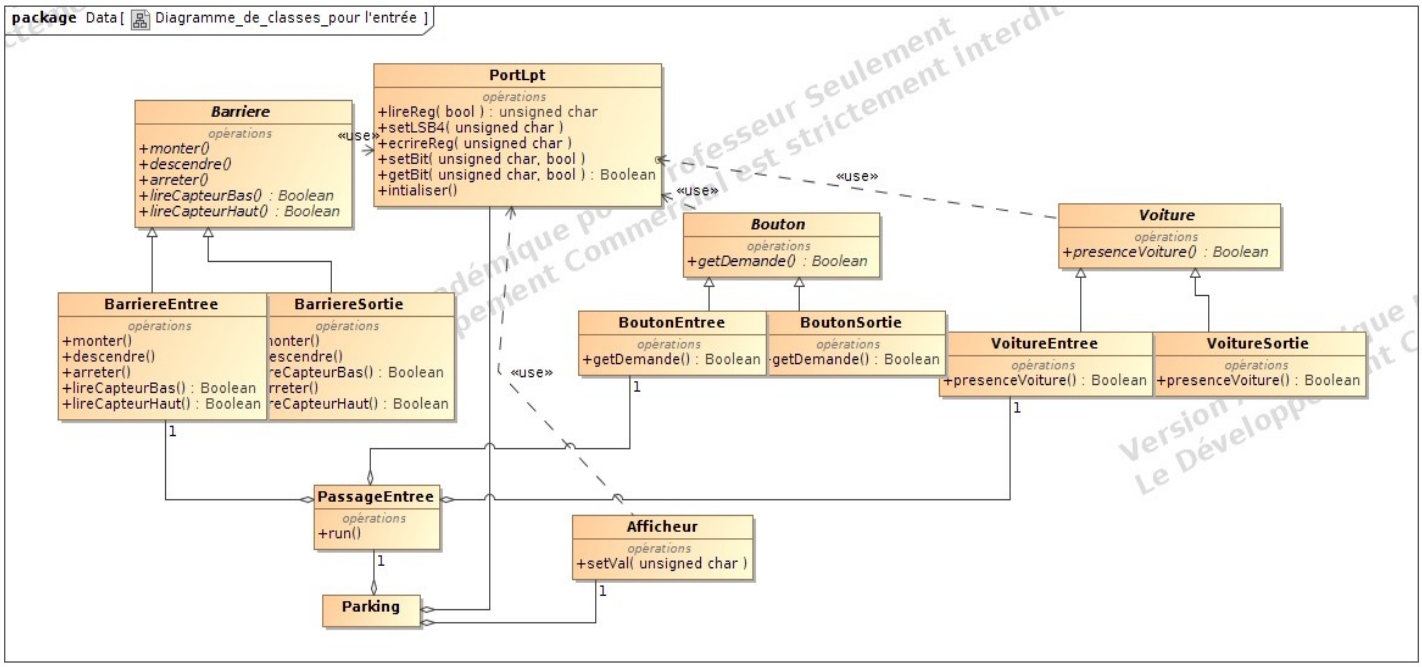


Diagramme de classe pour gérer les véhicules entrants:



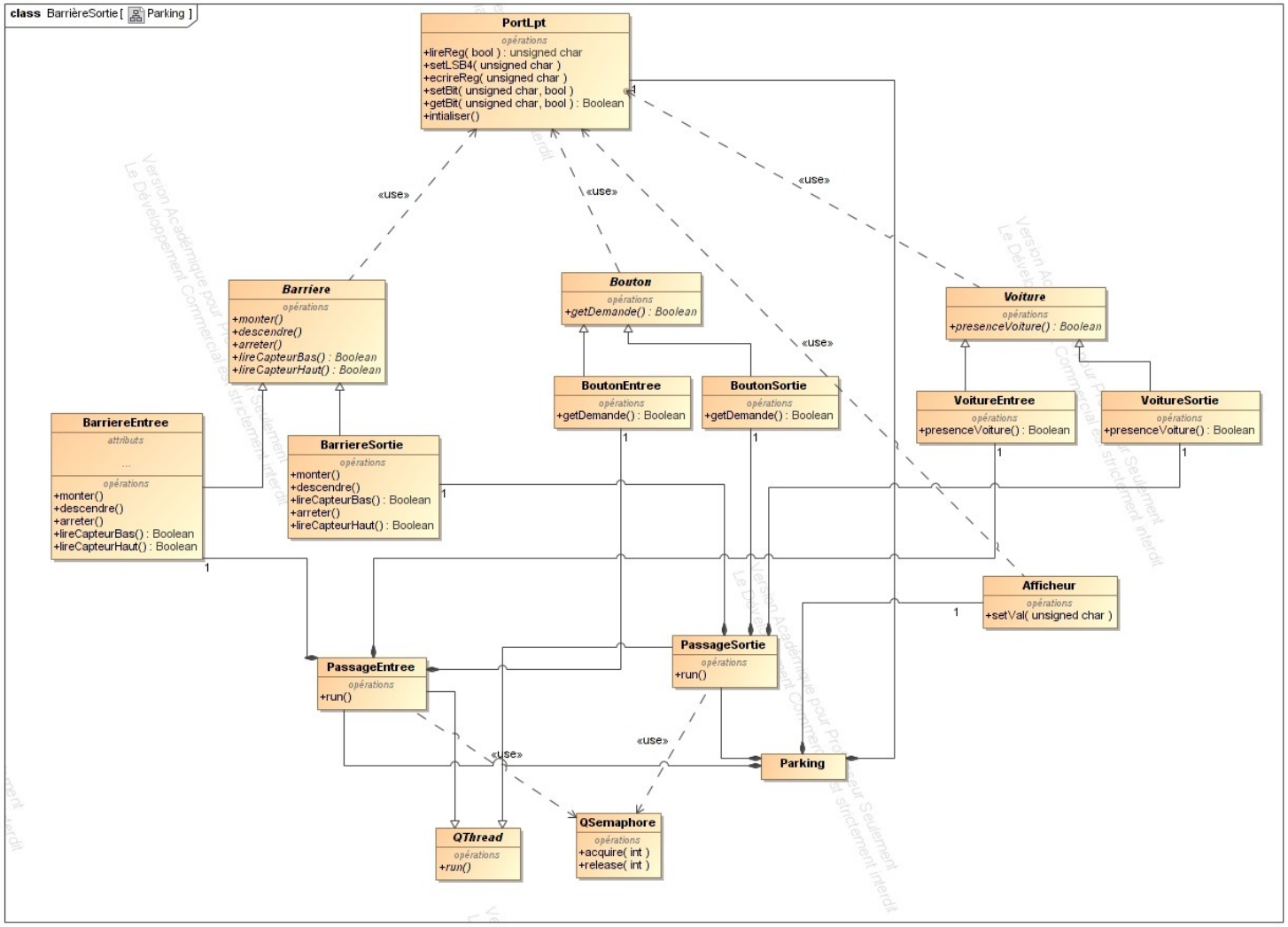
L'appui sur démarrer lancera l'automatisme associé au parking.
Traduction d'un algorithme associé à un diagramme d'état :

```
etat=0 ;
selon le cas où (etat)
debut
        cas 0 : Actions associées à l'état 0
                si (condition de sortie de l'état)
                        etat = ?
        cas 1 :    ……
…….
```

Donner l'algoritme associé au diagramme d'état correspondant à l'entrée d'un véhicule.

Réaliser l'IHM proposé pour la gestion du parking.
Réaliser les Classes Parking et PassageEntrée en intégrant l'intanciation des classes testées définies dans l'IHM.
Réaliser la méthode Automatisme() de la classe PassageEntree lancé par l'appui sur le bouton démarrer.
Dans la méthode Automatisme traduire l'agorithme que vous avez écrit ci dessus.
Tester votre Apllication.
Que pensez-vous du résultat ?
Peut on intégrer l'automatisme assicié au passage d'un véhicule en sortie ?
Si oui essayer de le faire.

9 - PROGRAMMES MULTITACHE



A partir des programmes précédents insérer les classes « PassageEntree » et « PassageSortie »
Compléter la méthode run de ces deux classes en vous servant des diagrammes d'état des passages en entrée et en sortie.

10 - TRAVAIL EN GROUPE

Par groupe de 3 :

1 : stocker, au fur et a mesure que les voitures entrent ou sortent, dans une base de données cette informatormation en la datant.

2 : Réaliser un site web qui affichera le nombre de places libres, et permettra de visualiser l'historique. A distance pourra être mis à jour le nombre de places libres dans le parking.

3 : Faire un IHM permettant de visualiser l'historique de l'utilisation du parking avec le nombre de véhicules présents selon le jour où l'heure.