RAPPORT DE PROJET CONCEPETION SYSTEME

GESTION D'UNE BORNE DE RECHARGE POUR VÉHICULE ÉLECTRIQUE

2 décembre 2019

Encadré par :
BERTHOU Pascal
PASCAL Jean-Claude

CARTAGENA José Sebastian
REGNERE Alexandre

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Remerciements :

Nous tenons tout d'abord à remercier M. PASCAL pour les TD dispensés et ses explications qui nous ont grandement permis de bien démarrer lors des premières séances. Nous remercions également tous nos collègues qui nous ont aidé à avancer dans ce long projet et particulièrement messieurs JOBERT et MOUTOUNAICK.

Table des matières

I. Introduction

Durant le premier semestre de la première année de master SME, il est demandé aux étudiant de réalisé un projet de mise en œuvre avec UML d'une borne de recharge de voiture électrique. Pour ce faire, notre projet se déroule en deux grandes phases :

- Une phase d'analyse expliquant le contexte du projet. Elle nous permet d'identifier les différents acteurs dans le système, ses éléments physiques et les divers scenarii des cas d'utilisation. Dans cette première phase seront présents le diagramme de contexte (cas d'utilisation), les diagrammes de séquences correspondant à chacun des cas d'utilisation et le diagramme de classe.
- Une phase de conception qui traduira l'implémentation en C orienté objet de ce projet. Elle comprendra les diagrammes de collaboration et les contrats type des méthodes.

II. Phase d'analyse

A. Définition du contexte

1. Éléments physiques de l'environnement

Borne :

- Voyants (DISPO, CHARGE, DEFAUT)
- Boutons poussoirs (CHARGE, STOP)
- Trappe de prise
- Lecteur de carte

2. Liste des fonctions principales

- Verrouiller prise
- Identifier client / enregistrer client
- Exploitation / maintenance
- Gérer la recharge du véhicule
- Affichage des défauts (voyants)
- Gérer la liste des clients

3. Contraintes non-fonctionnelles

- Mise à la terre
- Garantir la sécurité du client (pas de déconnexion de la prise quand le contacteur AC est fermé)
- Garantir la sécurité du véhicule

4. Définition des acteurs

- Véhicule
- Opérateur
- Client
- Borne

5. Cas d'utilisation

- Recharger véhicule
- Configurer la borne de recharge
- Gérer la liste des clients

B. Diagramme de contexte

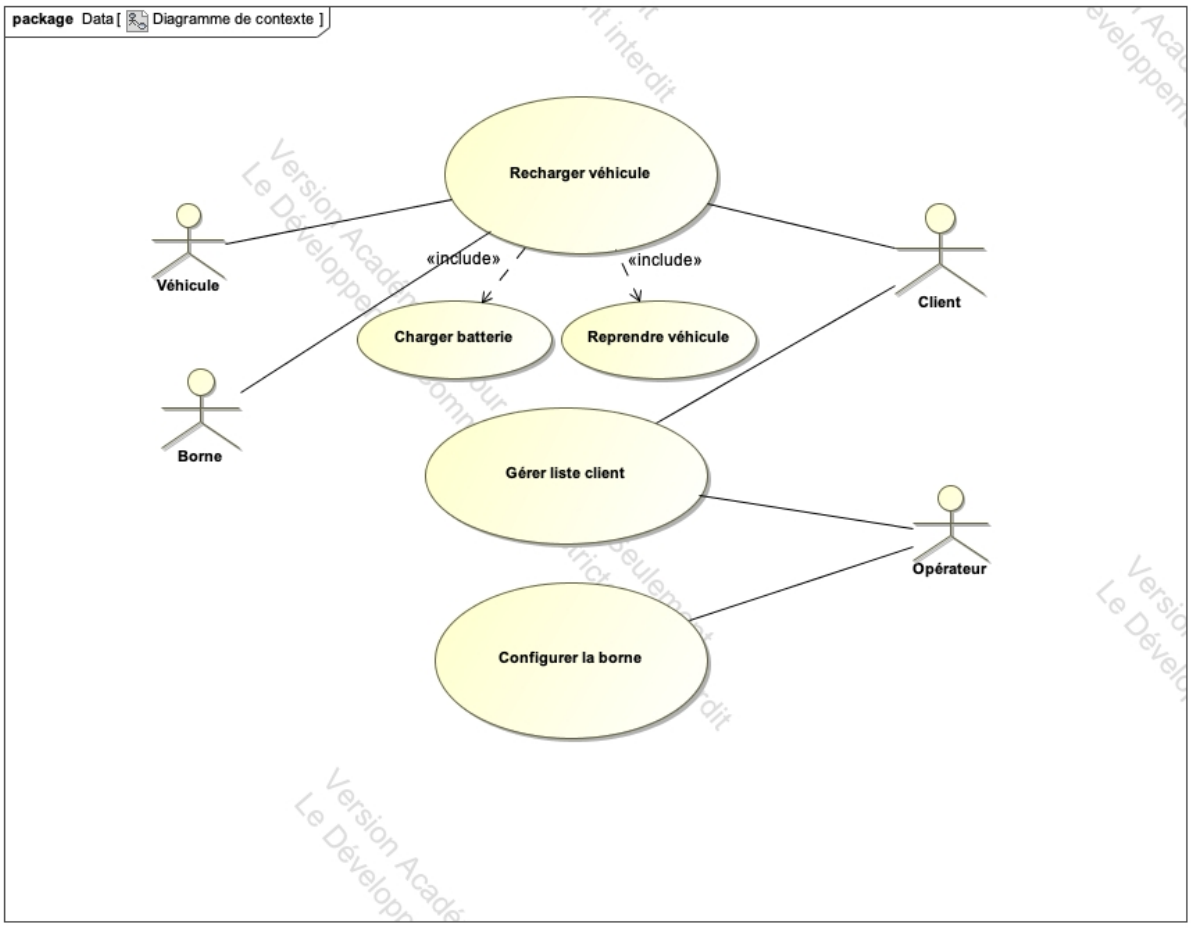

Figure 1 : Diagramme de conexte

Pour plus tard faciliter l'implémentation, le cas d'utilisation « Recharger véhicule » inclura les deux sous cas « Charger batterie » et « Reprendre véhicule ».

C. Description succinte des cas d'utilisation

1. Cas d'utilisation « Recharger véhicule »

Acteurs : véhicule, client et borne
Type : principal
Description : connexion à la borne par le client puis déclanchement du processus de recharge du véhicule

2. Cas d'utilisation « Charger batterie »

Acteurs : borne et véhicule
Type : sous cas
Description : le système effectue la recharge de la batterie en respectant les modes de charges indiqués dans le cahier des charges

3. Cas d'utilisation « Reprendre véhicule »

Acteurs : véhicule, client et borne
Type : sous cas
Description : une fois le processus de recharge terminé, le client doit insérer sa carte pour pouvoir récupérer son véhicule

4. Cas d'utilisation « Gérer liste client »

Acteurs : opérateur et borne
Type : principal
Description : l'opérateur peut ajouter, modifier ou supprimer des clients de la base client

5. Cas d'utilisation « Configurer la borne »

Acteur : borne
Type : principal
Description : L'opérateur peut contrôler à distance l'ensemble de ses bornes par un système de communication GPRS/3G (par exemple connaître l'ensemble des bornes hors service et les statistiques d'utilisation de chacune d'elles).

D. Description des scénarii

1. Cas d'utilisation « Recharger véhicule »

Acteur	Système
1. Le client insère sa carte	
	2. Le système identifie le client
	3. Le système fait clignoter le voyant CHARGE en VERT pendant 8 secondes
4. Le client appuie sur le bouton CHARGE avant une minute écoulée	
	5. Le système éteint le voyant DISPO
6. Le client retire sa carte	
	7. Le système déclenche la charge de la batterie (cas d'utilisation « Charger batterie »)
	8. Une fois la batterie chargée, le système attend que le client insère sa carte (cas d'utilisation « Reprendre véhicule »)
	9. Le système allume le voyant DISPO en VERT

Variante 1 : le client est inconnu

Acteur	Système
	2. Le système n'identifie pas le client
	3. Le système fait clignoter le voyant DEFAUT en ROUGE pendant 8 secondes
	4. Le système allume le voyant DISPO en VERT

Variante 2 : le client n'a pas appuyé sur le bouton CHARGE avant une minute

Acteur	Système
4. Le client n'appuie pas sur le bouton CHARGE avant une minute	
	5. Le système allume le voyant DISPO en VERT

2. Cas d'utilisation « Charger batterie »

Acteur	Système
	1. Le système allume le voyant CHARGE en ROUGE
	2. Le système déverrouille la trappe de la prise
	3. Le système génère un signal continu de 12 V
4. Le client branche la prise	
5. Le véhicule fait chuter la tension à 9V	
	6. Le système allume le voyant PRISE en VERT
	7. Le système verrouille la trappe
	8. Le système génère un signal PWM de fréquence 1 kHz
9. Le véhicule ferme le contacteur S2	
10. La tension chute à 6V	
	11. Le système ferme le contacteur AC
	12. Le système génère un signal PWM variable
13. Le véhicule (du fait de la charge) fait remonter progressivement la tension à 9 V	
14. Une fois le véhicule chargé, le contacteur S2 s'ouvre et la tension est égale à 9 V	
	15. Le système ouvre le contacteur AC
	16. Le système allume le voyant CHARGE en VERT
	17. Le système génère une tension continue

Variante : Le client appuie le sur le bouton STOP

Acteur	Système
	8. Le système génère un signal PWM de fréquence 1 kHz
9. Le client appuie sur le bouton STOP	
	10. Le système ouvre le contacteur AC
	11. Le système allume le voyant CHARGE en VERT
	12. Le système génère une tension continue

Tant que le véhicule est branché, cette variante s'exécute dès que le client appuie sur le bouton STOP, peu importe l'état du système.

3. Cas d'utilisation « Reprendre véhicule »

Acteur	Système
1. Le client insère sa carte pour le retrait du véhicule	
	2. Le système identifie le client comme étant celui qui avait mis son véhicule à charger
	3. Le système déverrouille la trappe
4. Le client retire la prise	
5. Le débranchement du véhicule fait remonter la tension continue à 12 V	
	6. Le système éteint le voyant PRISE
	7. Le système coupe la génération de tension
	8. Le système éteint le voyant CHARGE
	9. Le système verrouille la trappe
10. Le client retire sa carte	

Variante : Le client n'est pas le même

Acteur	Système
	2. Le système n'identifie pas le client comme étant le même
	3. Le système indique que le véhicule branché n'appartient pas à ce client

4. Cas d'utilisation « Générer liste client »

L'opérateur ajoute un client

Acteur	Système
1. L'opérateur insère sa carte	
	2. Le système identifie l'opérateur
3. L'opérateur ajoute un client	
	4. Le système ajoute le client dans la base client

Variante : l'opérateur supprime un client

Acteur	Système
3. L'opérateur supprime un client	
	4. Le système supprime le client de la base client

E. Digramme de séquence

1. Cas d'utilisation « Recharger véhicule »

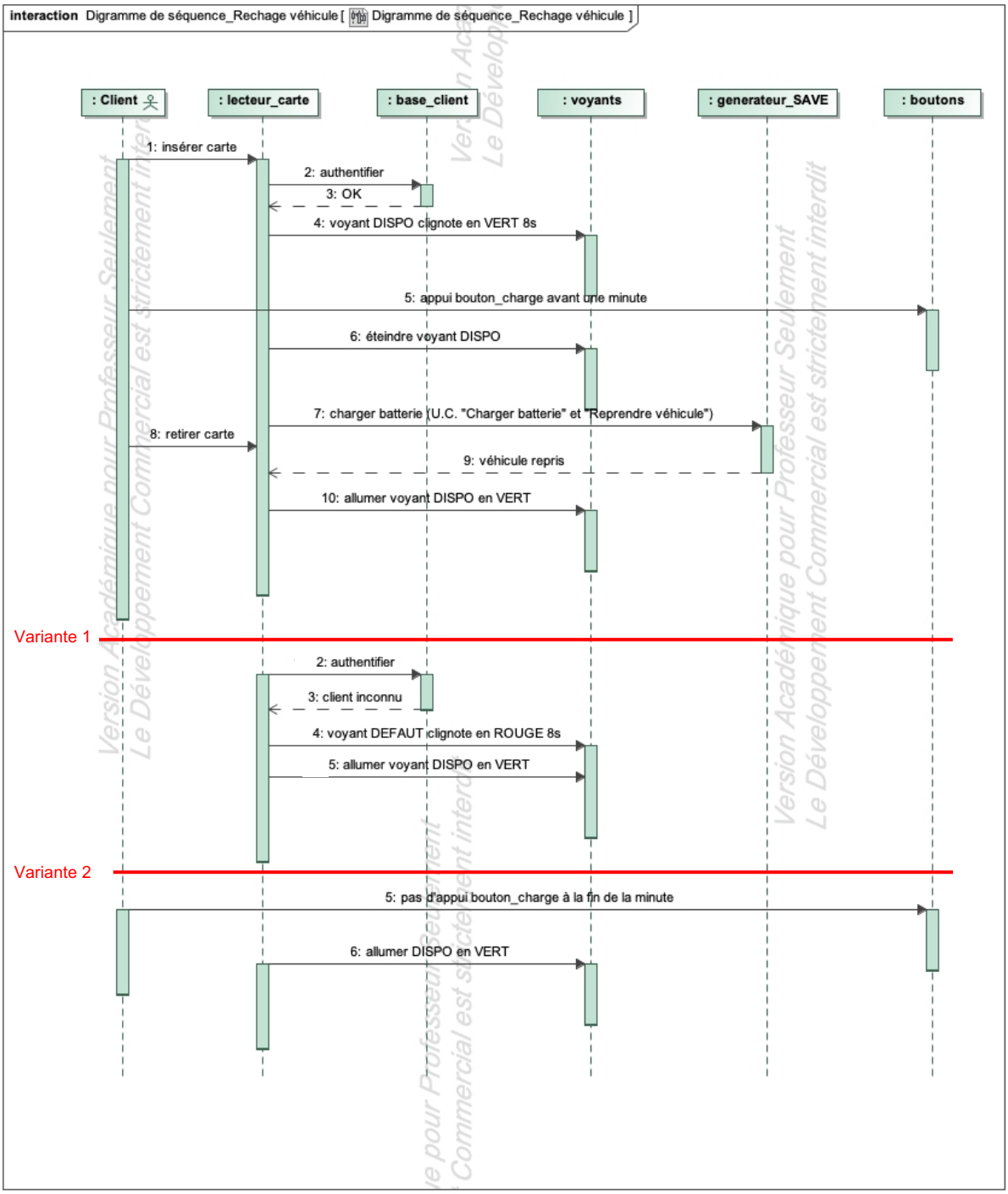

Figure 2 : Diagramme de séquence « Recharger véhicule »

2. Cas d'utilisation « Charger batterie »

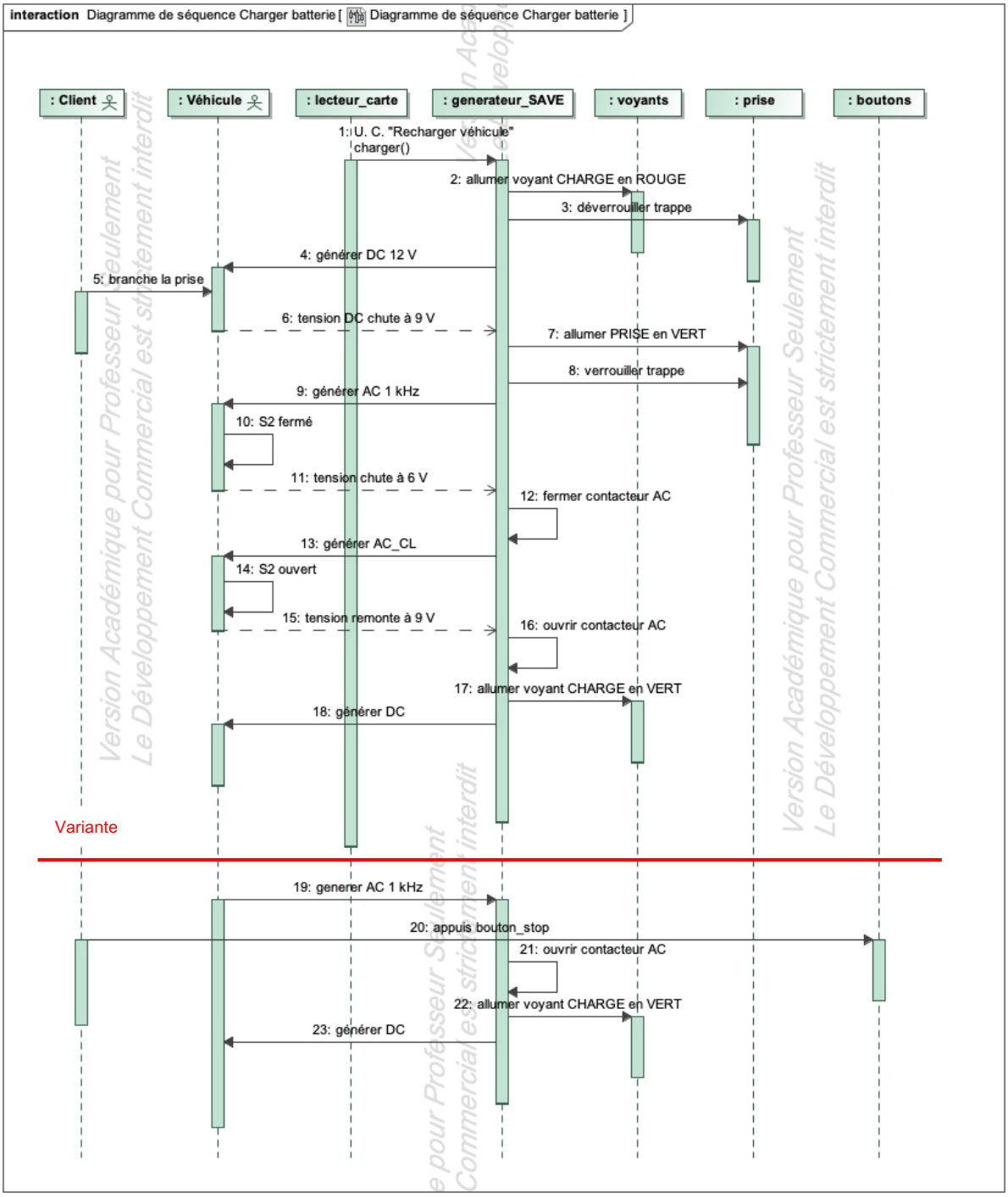

Figure 3 : Diagramme de séquence « Charger batterie »

3. Cas d'utilisation « Reprendre véhicule »

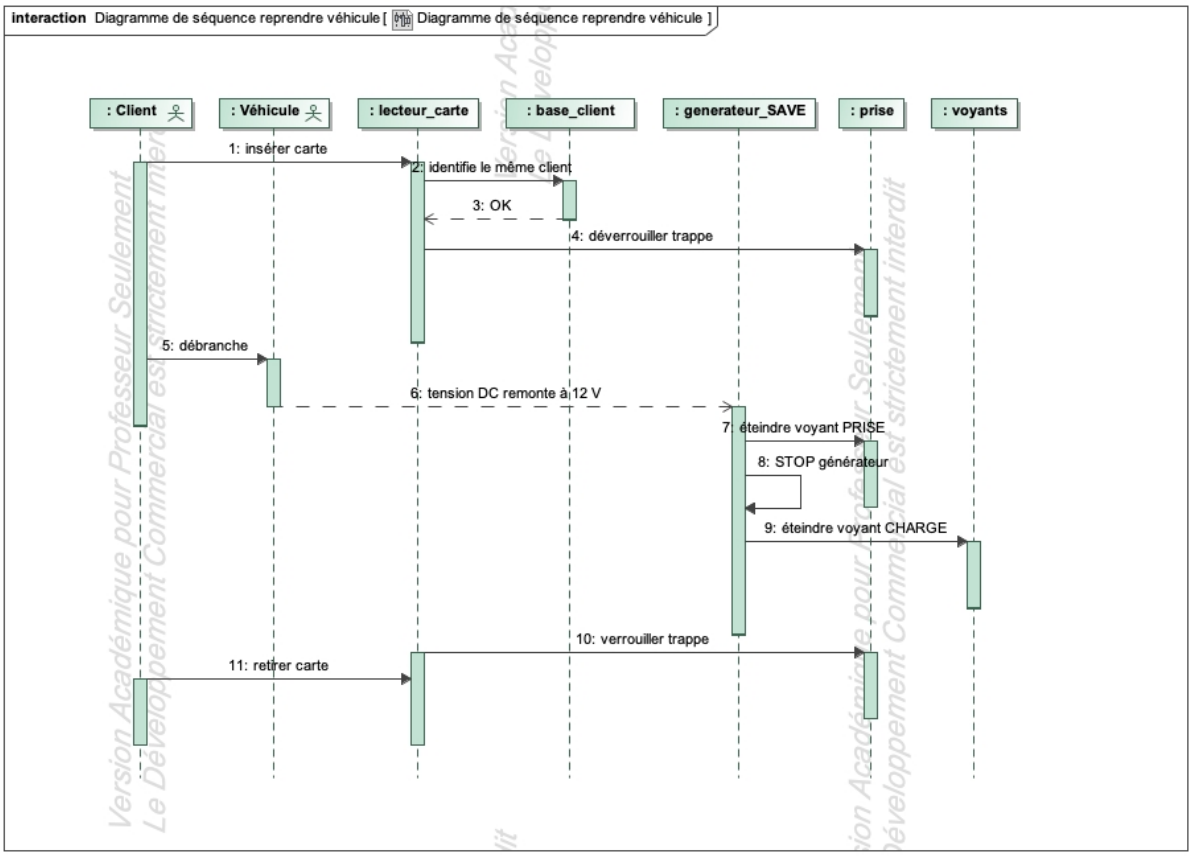

Figure 4 : Diagramme de séquence « Récupérer véhicule »

4. Cas d'utilisation « Générer liste client »

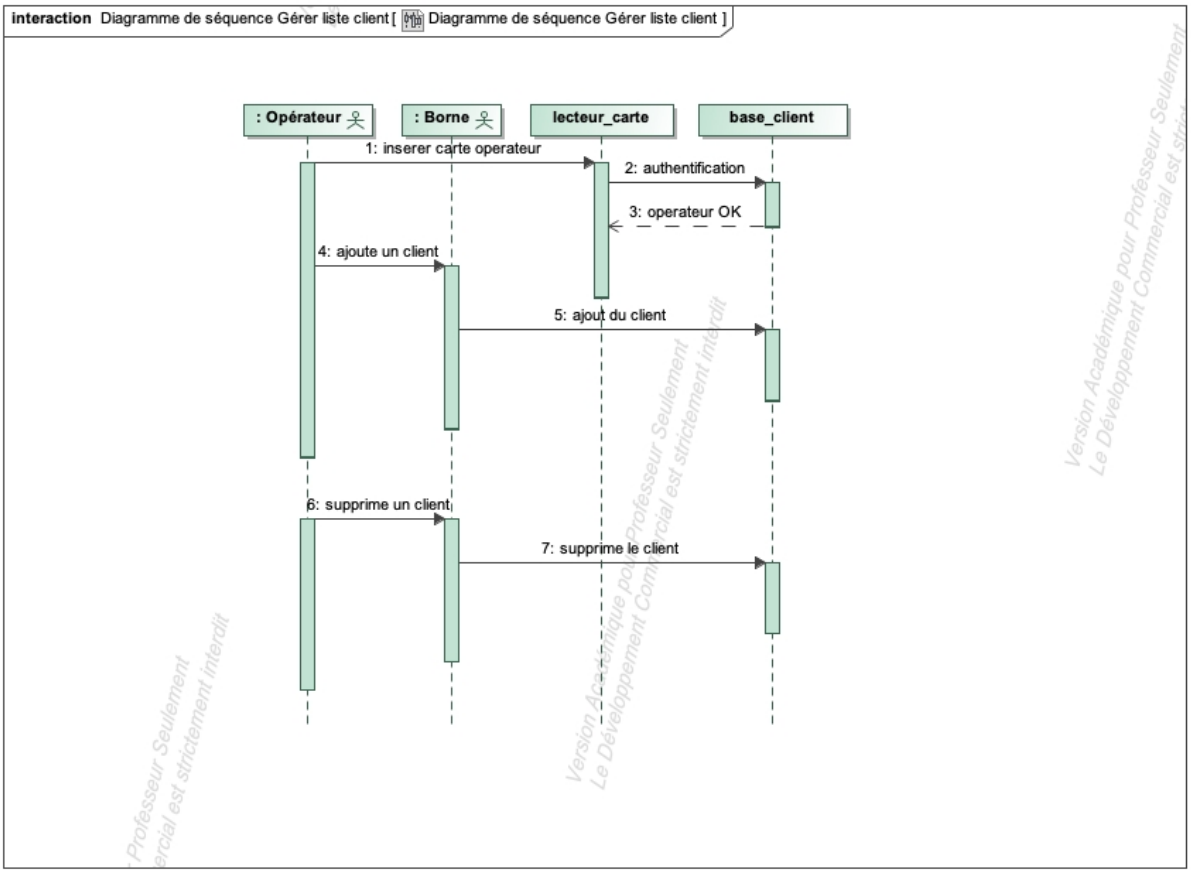

Figure 5 : Diagramme de séquence gérer liste client

F. Digramme de classe

Le diagramme de classe en qui résulte est le suivant :

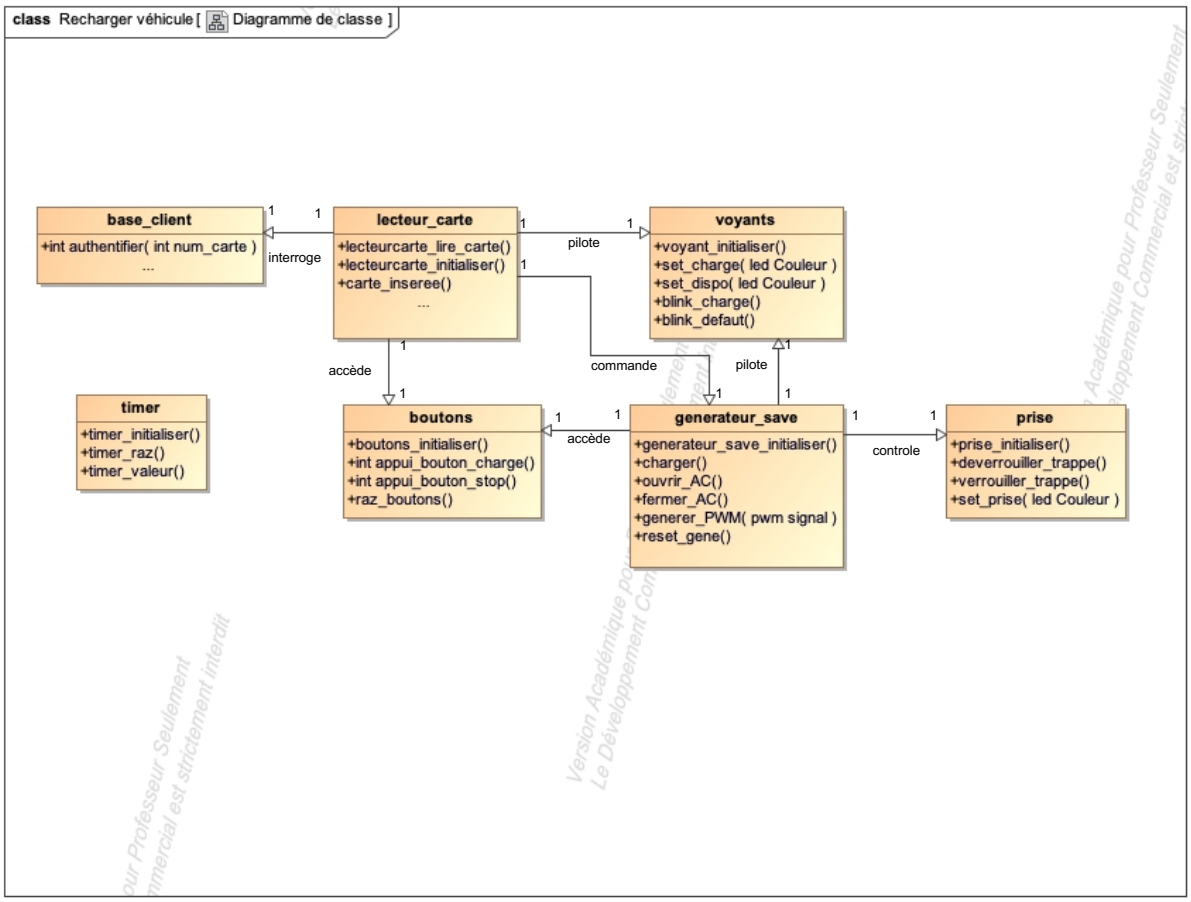

Figure 6 : Diagramme de classe

III. Phase de conception

La phase d'analyse étant terminée, nous pouvons maintenant passer à la phase de conception. Pour commencer, nous élaborons les diagrammes de collaboration de chaque cas d'utilisation.

A. Digramme de collaboration

1. Cas d'utilisation « Recharger véhicule »

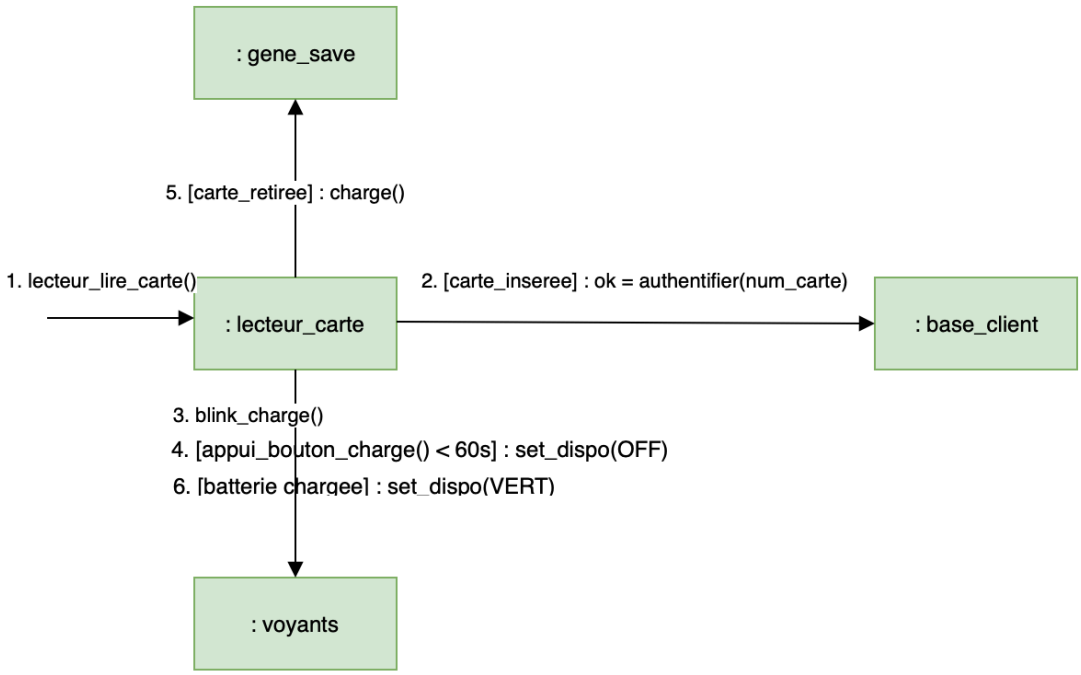

Figure 7 : Diagramme de collaboration « Recharger véhicule »

Variante 1 : Le client n'est pas reconnu

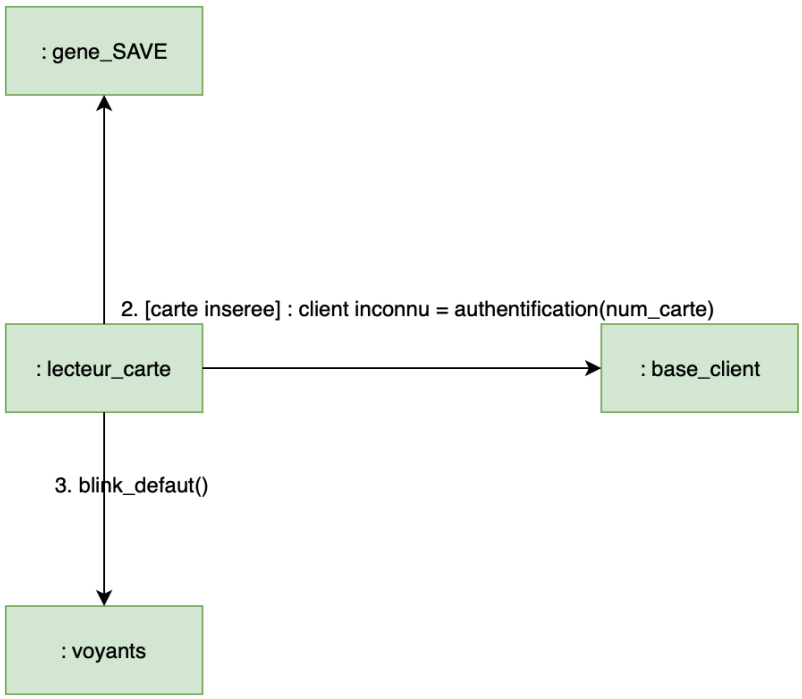

Figure 8 : Diagramme de collaboration « Recharger véhicule », variante 1

Variante 2 : Le client n’appuie pas sur le bouton charge

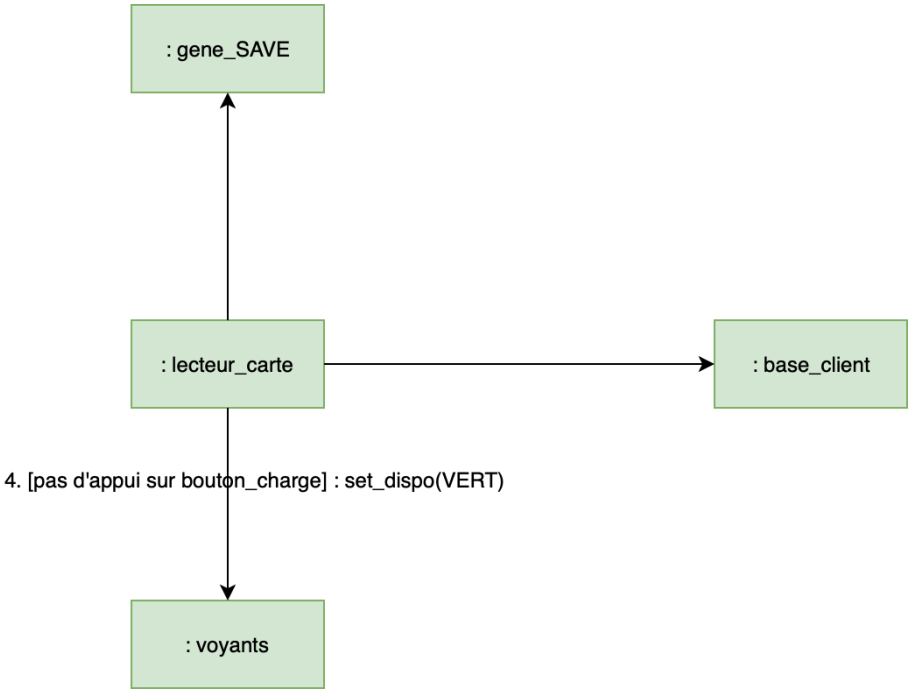

Figure 9 : Diagramme de collaboration « Recharger véhicule », variante 2

2. Cas d'utilisation « Charger batterie »

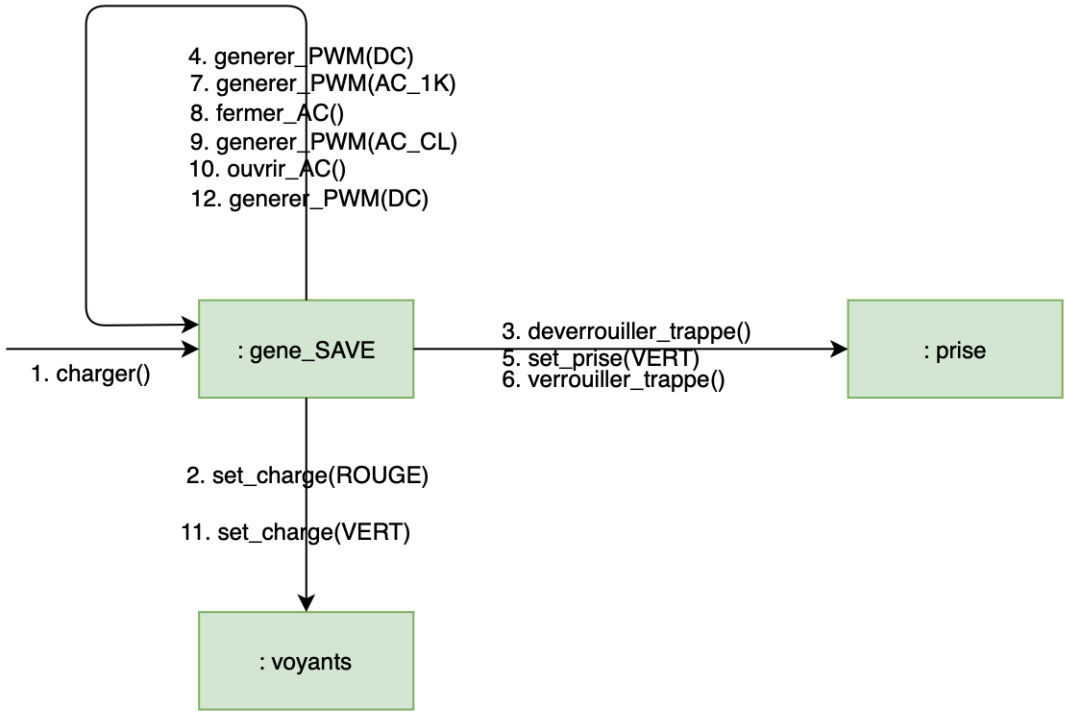

Figure 10 : Diagramme de collaboration « Charger batterie »

Variante : Le client appuie sur le bouton STOP

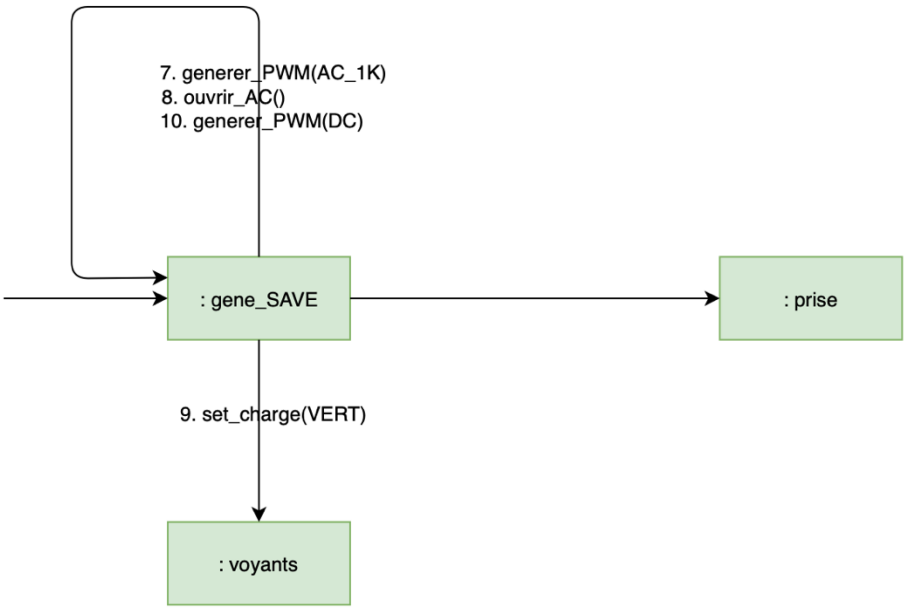

Figure 11 : Diagramme de collaboration « Charger batterie », variante bouton STOP

3. Cas d'utilisation « Reprendre véhicule »

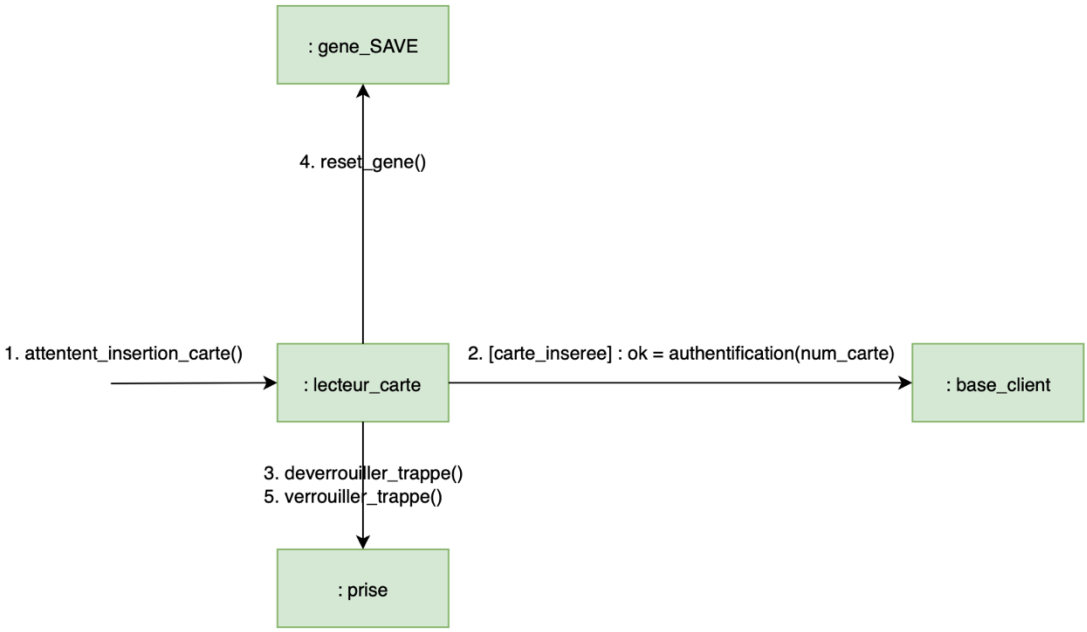

Figure 12 : Diagramme de collaboration « Reprendre véhicule »

Variante : Le client n'est pas le même

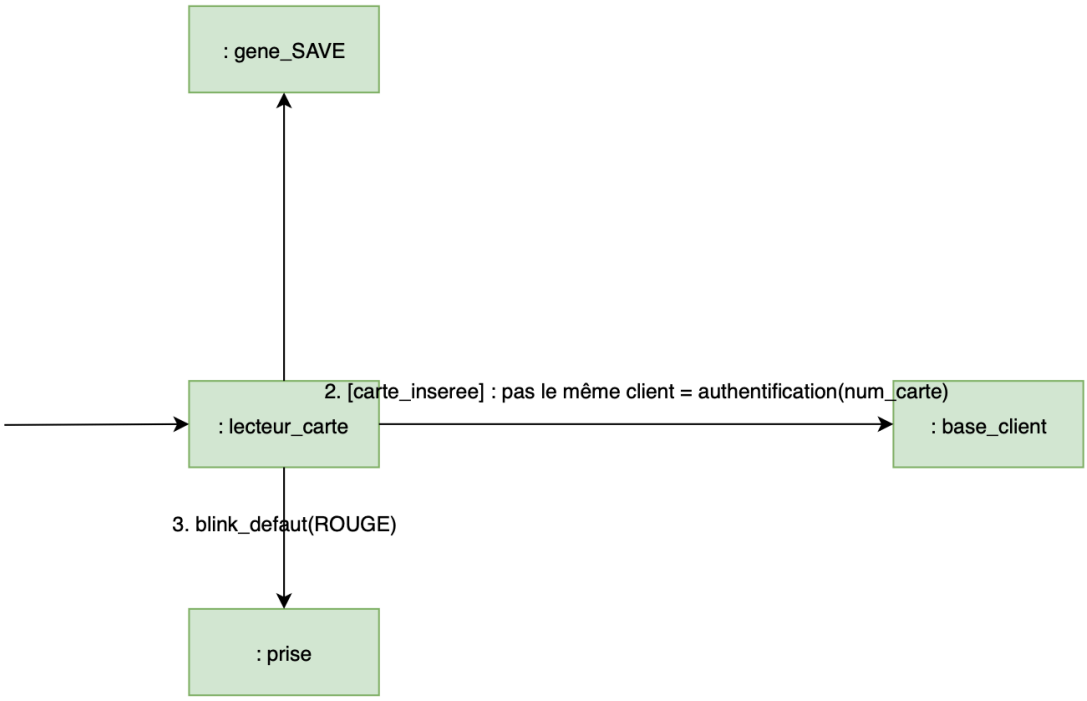

Figure 13 : Diagramme de collaboration « Reprendre véhicule », variante

B. Contrat type

Pour cette dernière partie nous regrouperons les contrats types par leur classe pour faciliter la lecture.

1. Classe : timer

Nom : timer_initialiser() **Responsabilité :** initialise la mémoire partagée pour le timer **Type :** logiciel **Références croisées :** – Classe : timer **Algorithme :** Accéder à la mémoire Si (pas d'accès) { Afficher « erreur pas de mémoire »}	**Nom :** timer_raz() **Responsabilité :** remet à zéro la valeur du timer **Type :** logiciel **Références croisées :** – Cas d'utilisation « Recharger véhicule » et « Reprendre véhicule » – Classe : timer et : lecteur_carte **Algorithme :** depart = valeur du timer **Précondition :** initialisation du timer
Nom : timer_valeur() **Responsabilité :** utiliser la valeur de timer_raz() pour calculer la valeur du temps écoulé **Type :** logiciel **Références croisées :** – Cas d'utilisation « Recharger véhicule » – Classe : timer et : lecteur_carte **Algorithme :** Retourner (valeur du timer) – (depart) **Paramètres Entrées/Sorties :** Sortie : valeur du timer depuis la remise à zéro **Précondition :** initialisation du timer et remise à zéro du timer	

2. Classe : base_client

Nom : authentifier()
Responsabilité : identifie la carte insérée
Type : logiciel et matériel
Références croisées :
- Cas d'utilisation « Recharger véhicule », « Reprendre véhicule » et « Gérer liste client »
- Classe : lecteur_carte

Algorithme :
Comparer numéro carte à base_client
Si (numéro carte dans base_client){
Retourner (1)}
Sinon {retourner (0)}
Paramètres Entrées/Sorties :
Entrée : numéro carte
Sortie : 0 ou 1 en fonction de la carte
Précondition : carte insérée et récupération du numéro de la carte

3. Classe : boutons

Nom : boutons_initialiser()
Responsabilité : initialiser la mémoire partagée pour les boutons
Type : logiciel
Références croisées :
- Classe : boutons et : lecteur_carte

Algorithme :
Accéder à la mémoire
Si (pas d'accès) {
Afficher « erreur pas de mémoire »}

Nom : appui_bouton_charge()
Responsabilité : renvoie l'état du bouton CHARGE
Type : logiciel et matériel
Références croisées :
- Cas d'utilisation « Recharger véhicule »
- Classe : boutons et : lecteur_carte

Algorithme :
Si (appui sur bouton CHARGE) {
Retouner état bouton_charge (qui est à 1)}
Paramètres Entrées/Sorties :
Sortie : 1 si appui
Précondition : initialisation des boutons
Postcondition : remise à zéro des boutons

Nom : appui_bouton_stop()
Responsabilité : renvoie l'état du bouton STOP
Type : logiciel et matériel
Références croisées :
- Cas d'utilisation « Charger batterie »
- Classe : boutons et : gene_SAVE

Algorithme :
Si (appui sur bouton STOP) {
Retourner état bouton_stop (qui est à 1)}
Paramètres Entrées/Sorties :
Sortie : 1 si appui
Précondition : initialisation des boutons
Postcondition : remise à zéro des boutons

Nom : raz_boutons()
Responsabilité : remet à zéro l'état des boutons
Type : logiciel et matériel
Références croisées :
- Cas d'utilisation « Recharger véhicule » et « Charger batterie »
- Classe : boutons, : lecteur_carte et : gene_SAVE

Algorithme :
Imposer l'état des deux boutons à zéro
Précondition : initialisation des boutons

4. Classe : voyants

Nom : voyants_initialiser() **Responsabilité :** initialise la mémoire partagée des boutons **Type :** logiciel et matériel **Références croisées :** – Classe : voyants et : lecteur_carte **Algorithme :** Accéder à la mémoire Si (pas d'accès) { Afficher « erreur pas de mémoire »}	**Nom :** set_charge() **Responsabilité :** gère l'allumage de la led CHARGE **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Recharger véhicule », « Charger batterie » et « Reprendre véhicule » – Classe : voyants, : lecteur_carte et : gene_SAVE **Algorithme :** Allumer led_charge de la couleur souhaitée **Paramètres Entrées/Sorties :** Entrée : couleur **Précondition :** initialisation des voyants
Nom : set_dispo() **Responsabilité :** gère l'allumage de la led DISPO **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Recharger véhicule » – Classe : boutons et : lecteur_carte **Algorithme :** Allumer led_dispo de la couleur souhaitée **Paramètres Entrées/Sorties :** Entrée : couleur **Précondition :** initialisation des voyants	**Nom :** set_defaut() **Responsabilité :** gère l'allumage de la led DEFAUT **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Recharger véhicule » et « Reprendre véhicule » – Classe : voyants et : lecteur_carte **Algorithme :** Allumer led_defaut de la couleur souhaitée **Paramètres Entrées/Sorties :** Entrée : couleur **Précondition :** initialisation des voyants
Nom : blink_charge() **Responsabilité :** fait clignoter la led CHARGE **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Recharger véhicule » – Classe : voyants et : lecteur_carte – En relation avec : timer **Algorithme :** Tant que (valeur timer < 8) { Allumer 1 seconde CHARGE en VERT Éteindre 1 seconde CHARGE } **Précondition :** initialisation du timer et des voyant, remise à zéro du timer **Postcondition :** allumer CHARGE en VERT	**Nom :** blink_defaut() **Responsabilité :** fait clignoter la led DEFAUT **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Recharger véhicule » et « Reprendre véhicule » – Classe : voyants et : lectuer_carte – En relation avec : timer **Algorithme :** Tant que (valeur timer < 8) { Allumer 1 seconde DEFAUT en ROUGE Éteindre 1 seconde DEFAUT } **Précondition :** initialisation du timer et des voyants, remise à zéro du timer **Postcondition :** éteindre DEFAUT

5. Classe : prise

Nom : prise_initialiser() **Responsabilité :** initialise la mémoire partagée de la prise **Type :** logiciel et matériel **Références croisées :** – Classe : prise **Algorithme :** Accéder à la mémoire Si (pas d'accès) { Afficher « erreur pas de mémoire » }	**Nom :** deverouiller_trappe() **Responsabilité :** déverrouille la trappe et allume la led de la trappe en VERT **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Charger batterie » et « Reprendre véhicule » – Classe : prise, : gene_SAVE et : lecteur_carte **Algorithme :** Allumer led_trappe en VERT **Précondition :** initialisation de la prise
Nom : verrouiller_trappe() **Responsabilité :** verrouille la trappe et éteint la led de la trappe **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Charger batterie » et « Reprendre véhicule » – Classe : prise, :gene_SAVE et : lecteur_carte **Algorithme :** Éteindre led_trappe **Précondition :** initialisation de la prise	**Nom :** set_prise() **Responsabilité :** gère l'allumage de la led PRISE **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Charger batterie » – Classe : prise et : gene_SAVE **Algorithme :** Allumer led_prise de la couleur souhaitée **Paramètres Entrées/Sorties :** Entrée : couleur **Précondition :** initialisation de la prise

6. Classe : lecteur_carte

Nom : lecteurcarte_initialiser() **Responsabilité :** initlialiser les ports de la borne ainsi que les boutons et les voyants **Type :** logiciel et matériel **Références croisées :** – Classe : lecteur_carte – En relation avec : voyants et : boutons **Algorithme** Initialiser ports Initialiser voyans Initialiser boutons	**Nom :** carte_inseree() **Responsabilité :** vérifie qu'une carte est inseree **Type :** logiciel et materiel **Références croisées :** – Cas d'utilisation « Recharger véhicule » et « Reprendre véhicule » – Classe : lecteur_carte **Algorithme** Connu par le système **Précondition :** initialisation des ports

Nom : lecteurcarte_lire_carte()
Responsabilité : gère la lecture des cartes et est à l'origine de tous les cas d'utilisation
Type : logiciel et matériel
Références croisées :

- Cas d'utilisation « Recharger véhicule », « Charger batterie » et « Reprendre véhicule »
- Classe : lecteur_carte
- En relation avec : gene_SAVE, : boutons, : voyants et : prise

Algorithme :

```
Attente insertion carte
Tant que (1) {
  Si (carte insérée) {
  Récupérer le numéro carte
    Si (authentifier carte == 1) {
     Sortir du tant que }
     Sinon {clignoter DEFAUT en ROUGE }
}
Clignoter CHARGE en VERT

Tant que (timer < 60 seconde) {
  Si (appui sur bouton CHARGE) {
   Sortir de tant que }

Attente retrait carte

Charger -> cas d'utilisation « Charger batterie »

Attente insertion carte -> cas d'utilisation « Reprendre véhicule »
  Si (carte insérée) {
  Récupérer le numéro carte
    Si (authentifier carte = 1) {
      Déverrouiller trappe
      Tant que (1) {
       Si (reset gene est terminé){
       Verouiller trappe
       Sortir de tant que }
   Attente retrait carte
  }
Allumer voyant DISPO en VERT
```

Précondition : initialisation des ports, des voyants, des boutons et de la prise

7. Classe : gene_SAVE

Nom : gene_save_initialiser() **Responsabilité :** initialise la mémoire partagée du générateur **Type :** logiciel et matériel **Références croisées :** – Classe : gene_SAVE **Algorithme** Accéder à la mémoire Si (pas d'accès) { Afficher « erreur pas de méoire » }	**Nom :** ouvrir_AC() **Responsabilité :** contrôle le contacteur AC **Type :** logiciel et matériel **Références croisées :** – Cas d'utilisation « Charger batterie » – Classe : gene_SAVE **Algorithme :** Contacteur_AC = 1 **Précondition :** initialisation du générateur SAVE

Nom : fermer_AC
Responsabilité : contrôle le contacteur AC
Type : logiciel et matériel
Références croisées :
– Cas d'utilisation « Charger batterie »
– Classe : gene_SAVE
Algorithme :
Contacteur_AC = 0
Précondition : initialisation du générateur SAVE
	Nom : generer_PWM()
	Responsabilité : génère les différentes tensions demandées
	Références croisées :
	– Cas d'utilisation « Charger batterie »
	– Classe : gene_SAVE
	Algorithme :
	Switch (signal reçu) {
	Cas DC : gene_pwm = DC
	Cas AC_1K : gene_pwm = AC_1K
	Cas AC_CL : gene_pwm = AC_CL
	Cas STOP : gene_pwm = STOP
	}
	Paramètres Entrées/Sorties :
	Entrée : signal
	Précondition : initialisation du générateur SAVE

Nom : reset_gene() **Responsabilité :** réinitialise le générateur SAVE pour préparer la réception d'un nouveau véhicule **Références croisées :** – Cas d'utilisation « Reprendre véhicule » – Classe : gene_SAVE et : lecteur_carte **Algorithme :** Si (tension DC remonte à 12 V) { Éteindre voyant PRISE Arrêter générateur Éteindre voyant CHARGE Retourner (1) **Paramètres Entrées/Sorties :** Sortie : 1 si le reset est terminé **Précondition :** initialisation du générateur SAVE, des voyants et de la prise	

Nom : charger()
Responsabilité : gère la recharge de la batterie et est à l'origine du cas d'utilisation « Charger batterie »
Références croisées :
- Cas d'utilisation « Charger batterie »
- Classe : gene_SAVE et : lecteur_carte

Algorithme :

```
Tant que (pas à la fin) {
 État 1 : allumer voyant CHARGE en ROUGE
          Deverouiller trappe
          Générer tension DC
          Si (tension = 9 V) {État suivant = État 2 }

 État 2 : allumer voyant PRISE en VERT
          Verrouiller trappe
          Générer tension AC 1K
          Si (tension = 6 V) { État suivant = État 3 }
          Si (appui sur bouton STOP) { État suivant = État 5}

 État 3 : fermer contacteur AC
          Générer tension AC CL
          Si (tension = 6 V) { État suivant = État 4 }
          Si (appui sur bouton STOP) { État suivant = État 5 }

État 4 : si (tension = 9 V) { État suivant = État 5 }
         Si (appui sur bouton STOP) { État suivant = État 5 }

État 5 : ouvrir contacteur AC
         Aller voyant CHARGE en VERT
         Générer tension DC
         Fin

État = État suivant
}
```

Précondition : initialisation du générateu SAVE, des voyants et de la prise

IV. Conclusion

En conclusion de ce projet, nous pouvons tout d'abord dire qu'il nous a été très profitable tant en approfondissement de compétence en C qu'en connaissance UML. Aborder un projet grâce à l'UML, nous a montré qu'il était important de savoir analyser une situation ou un contexte, aussi simples soient-ils. En effet, tous les diagrammes nous ont permis d'avoir un langage unifié et explicit de ce que nous devions réaliser. De cette façon, toutes les personnes travaillant sur un projet peuvent en comprendre les aspects sans pour autant en connaître tout le fonctionnement.

Cependant, malgré un très gros travail fournit en dehors des TPs, nous n'avons pas été à même de pouvoir le terminer entièrement. Le cas d'utilisation « Gérer liste client » a beaucoup été discuté mais n'a pas abouti. Notre système fonctionne de manière simple en boucle et en intégrant les quelques variantes.

V. Annexes

```c
//BORNE.C
#include <stdio.h>
#include <memoire_borne.h>
#include <donnees_borne.h>

#include "lecteurcarte.h"
#include "base_clients.h"

int main()
{

    lecteurcarte_initialiser();

    while (1)
    {
        lecteurcarte_lire_carte();
    }

}
```

```c
//TIMER.C
#include "timer.h"
#include <stdio.h>

entrees *io ;
int shmid ;
int depart_timer ;


void timer_initialiser()
{
 io=acces_memoire(&shmid) ; //associe l zone de memoire partagee au pointeur
 if(io == NULL)
 printf("Erreur pas de mem sh\n") ;
}

void timer_raz()
{
 depart_timer = io->timer_sec ;
}

int timer_valeur()
{
int t ;
 t = (io->timer_sec) - depart_timer ;
 //printf("%d\n", t) ;
 return (io->timer_sec) - depart_timer ;
}
```

```c
//TIMER.H
#ifndef TIMER_H
#define TIMER_H
#include <donnees_borne.h>
#include <memoire_borne.h>

 void timer_initialiser() ;
 void timer_raz() ;
 int timer_valeur() ;

#endif
a
```

```c
//BASE_CLIENT.C
#include "base_clients.h"

unsigned int id_client ;

int authentifier(unsigned int num_carte)
{
 int i;
 unsigned int carte_client[20] = {10, 15, 16, 19, 20, 25, 26, 29, 30, 37} ;


    for(i=0;i<10;i++)
    {
     if(carte_client[i] == num_carte){
     id_client = num_carte ;
     return 1 ;
     }

    }
return 0 ;
}
```

```c
//BASE_CLIENT.H
#ifndef BASE_CLIENT_H
#define BASE_CLIENT_H
#include <lcarte.h>

int authentifier(unsigned int num_carte) ;

#endif
```

```c
//BOUTON.C
#include "boutons.h"


entrees *io ;
int shmid ;

void boutons_initialiser()
{
 io=acces_memoire(&shmid) ; //associe l zone de memoire partagee au pointeur
 if(io == NULL)
 printf("Erreur pas de mem sh\n") ;
}

int appui_bouton_charge()
{
 if( io->bouton_charge == 1) {
        printf("*appui sur bouton_charge* \n");
 }
return io->bouton_charge ;
}

int appui_bouton_stop()
{
 if( io->bouton_stop == 1) {
 printf("*appui sur bouton_stop* \n");}
 return io->bouton_stop ;
}

void raz_boutons()
{
 io->bouton_stop = 0 ;
 io->bouton_charge = 0 ;
}
```

```c
//BOUTON.H
#ifndef BOUTONS_H
#define BOUTONS_H
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <lcarte.h>
#include <mem_sh.h>

void boutons_initialiser() ;
int appui_bouton_charge() ;
int appui_bouton_stop() ;
void raz_boutons() ;


#endif
```

```c
//VOYANT.C
#include <unistd.h>
#include "voyants.h"


entrees *io ;
int shmid ;

void voyants_initialiser()
{
 io = acces_memoire(&shmid) ;
 if (io==NULL) printf("Erreur pas de mem sh\n");
}

void set_charge(led Couleur)
{
 io->led_charge = Couleur ;
}

void set_dispo(led Couleur)
{
 io->led_dispo = Couleur ;
}

void set_defaut(led Couleur)
{
 io->led_defaut = Couleur ;
}


void blink_charge()
{
 while(timer_valeur() < 8)
  {
   if(timer_valeur()%2 == 0)
   set_charge(VERT) ;
   if(timer_valeur()%2 == 1)
   set_charge(OFF) ;
  }
}

void blink_defaut()
{
 while(timer_valeur() < 8)
  {
   if(timer_valeur()%2 == 0)
   set_defaut(ROUGE) ;
   if(timer_valeur()%2 == 1)
   set_defaut(OFF) ;
  }
}
```

```
//VOYANT.H
#ifndef VOYANTS_H
#define VOYANTS_H
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <lcarte.h>
#include <mem_sh.h>
#include "timer.h"

void voyants_initialiser();
void set_charge(led Couleur);
void set_dispo(led Couleur) ;
void set_defaut(led Couleur) ;
void blink_charge() ;
void blink_defaut() ;

#endif
```

```
//LECTEUR_CARTE.C
#include <unistd.h>
#include <stdio.h>
#include "lecteurcarte.h"


void lecteurcarte_initialiser()
{
 initialisations_ports() ;
 voyants_initialiser() ;
 boutons_initialiser() ;
}

void lecteurcarte_lire_carte()
{
unsigned int num_carte = 0 ;

 while(1){

 printf("Presentez carte \n") ;
 attente_insertion_carte() ;

    if(carte_inseree()){
      num_carte = lecture_numero_carte() ;
      printf("Carte : %d \n", num_carte) ;

      if(authentifier(num_carte) == 1 ){
        printf("Authentification Client : OK\n") ;
        break ;
      }
      else{
        printf("Authentification non reconnue\n") ;
        printf("Retirez votre carte\n");
        timer_raz() ;
        blink_defaut() ; // clignotement de DEFAUT
        set_defaut(OFF) ;
        timer_raz() ;
      //sleep(3) ;
      }
    }
 }

 timer_raz() ;
 blink_charge() ; // clignotement de CHARGE
 set_charge(OFF) ;
 timer_raz() ;

 while(timer_valeur() < 60){ // Une minute pour appuyer

    if(appui_bouton_charge() == 1){
          raz_boutons() ;
```

```
      set_dispo(OFF) ;
      break ; // si il y a un appui, on sort du while
    }
 }

 printf("Retirez carte \n") ;
 attente_retrait_carte() ;

 charger() ; // lancement du processus de charge
 printf("Presentez carte pour Retrait \n") ;
 attente_insertion_carte() ;

 if(carte_inseree()){
    if(authentifier(num_carte) == 1 ){
    printf("Authentification Retrait Vehicule : OK\n") ;
    }
    else{
    printf("Mauvaise voiture\n") ;
    }
    deverouiller_trappe() ;
    while(1){  //oblige de faire le while(1) ici et non dans le gene_save (pourquoi
??)
         if(reset_gene()==1){
        verouiller_trappe() ;
        break ;
        }
    }


    printf("Vous pouvez recuperer votre vehicule\n") ;
    printf("N'oubliez pas votre carte\n") ;
    attente_retrait_carte() ;

 }
//liberation_ports() ;
set_dispo(VERT) ;
}
```

```
//LECTEUR_CARTE.H
#ifndef LECTEURCARTE_H
#define LECTEURCARTE_H
#include <lcarte.h>
#include "timer.h"
#include "base_clients.h"
#include "voyants.h"
#include "boutons.h"
#include "generateur_save.h"
#include "prise.h"

void lecteurcarte_initialiser();
void lecteurcarte_lire_carte();

#endif // LECTEURCARTE_H
```

```c
//GENE_SAVE.C
#include "generateur_save.h"

entrees *io ;
int shmid ;


void generateur_save_initialiser()
{
 io=acces_memoire(&shmid) ; //associe l zone de memoire partagee au pointeur
 if(io == NULL)
 printf("Erreur pas de mem sh\n") ;
}

void charger()
{

int Fin = 0 ;
int EtatSuivant = 1, EtatPresent = 1 ;

 while(Fin != 1)
 {
    switch(EtatPresent)
     {
    case 1: //Attente prise
        set_charge(ROUGE) ;
        deverouiller_trappe() ; // led_trappe = VERT
        generer_PWM(DC) ;
        if(io->gene_u == 9){  // valeur de tension retournee
        EtatSuivant = 2 ;}

        break ;

    case 2: //Prise branchee
        set_prise(VERT) ;
        verouiller_trappe() ; // led_trappe = OFF
        generer_PWM(AC_1K) ;
        if(io->gene_u == 6){
        EtatSuivant = 3 ;}
        if(appui_bouton_stop() == 1){
        raz_boutons() ;
        EtatSuivant = 5 ;}

        break ;

    case 3: //Attente_s2_vehicule
        fermer_AC() ;
        generer_PWM(AC_CL) ;
        if(io->gene_u == 6){
        EtatSuivant = 4 ;}
        if( (appui_bouton_stop() == 1) ){ //|| (io->gene_u == 9) )
```

```
        raz_boutons() ;
        EtatSuivant = 5 ;}

        break ;

    case 4: //Charge
        if(io->gene_u == 9){
        EtatSuivant = 5 ;}
        if(appui_bouton_stop() == 1){
        raz_boutons() ;
        EtatSuivant = 5 ;
        }

        break ;

    case 5: //Batterie chargee
        ouvrir_AC() ;
        set_charge(VERT) ;
        generer_PWM(DC) ;
        Fin = 1 ;
        printf("Processus de recharge termine \n") ;

        break ;
    }

    EtatPresent = EtatSuivant ;
        //printf("%d\n", EtatPresent);
 }

}

void generer_PWM(pwm signal)
{
    switch(signal)
    {
     case DC:
     io->gene_pwm = DC ;
     break ;

     case AC_1K:
     io->gene_pwm = AC_1K ;
     break ;

     case AC_CL:
     io->gene_pwm = AC_CL ;
     break ;

     case STOP:
     io->gene_pwm = STOP ;
     break ;
    }
```

```
}

void fermer_AC()
{
 io->contacteur_AC = 1 ;
}

void ouvrir_AC()
{
 io->contacteur_AC = 0 ;
}

int reset_gene()
{
 int reset = 0 ;

 //printf("reset gene \n") ;


    if(io->gene_u == 12)
    {
     printf("reset gene \n") ;
     set_prise(OFF) ;
     generer_PWM(STOP) ;
     set_charge(OFF) ;
     reset = 1 ;
    printf("%d\n", io->gene_u) ;
    }
return reset ;
}
```

```
//GENE_SAVE.H
#ifndef GENERATEUR_SAVE_H
#define GENERATEUR_SAVE_H
#include <donnees_borne.h>
#include <memoire_borne.h>
#include <lcarte.h>
#include <mem_sh.h>
#include "voyants.h"
#include "prise.h"
#include "boutons.h"

void generateur_save_initialiser() ;
void charger();
void generer_PWM(pwm signal) ;
void fermer_AC() ;
void ouvrir_AC() ;
int reset_gene() ;

#endif
```