

---

MASTER ISTR

---

## UE : Conception de Système

# GESTION D'UNE BORNE DE RECHARGE, POUR VEHICULE ELECTRIQUE

---

Année scolaire 2023-2024

Enseignant :	P.BERTHOU
E-mail enseignant :	berthou@laas.fr
Réalisé par :	-SEGHER Aissa et -RAHMOUN Lokmane

# Table des matières

<b>Table des matières</b>	<b>2</b>
<b>Table des figures</b>	<b>4</b>
<b>1 ANALYSE</b>	<b>5</b>
1.1 INTRODUCTION . . . . .	5
1.2 ANALYSE DU MINI PROJET . . . . .	5
1.2.1 Diagramme de contexte . . . . .	6
<b>2 Conception</b>	<b>6</b>
2.0.1 Élément physique du système : . . . . .	6
2.0.2 Définition des besoins : . . . . .	6
2.0.3 Les éléments de l'environnement en relation avec le système : . . . . .	7
2.1 Diagrammes de cas d'utilisation . . . . .	7
2.2 Description des UC : . . . . .	7
2.2.1 Recharger Véhicule . . . . .	7
2.2.2 Charger Batterie . . . . .	7
2.2.3 Reprendre Véhicule . . . . .	8
2.2.4 Configurer Borne . . . . .	8
2.2.5 Gérer Liste Clients . . . . .	8
2.3 Diagramme de séquence : . . . . .	8
2.4 Définition des scénarios : . . . . .	8
2.4.1 Tableau 1 - UC « Recharger Véhicule » . . . . .	9
2.4.2 Tableau 2 - UC « Charger Batterie » . . . . .	10
2.4.3 Tableau 3-UC « Reprendre Véhicule » . . . . .	11
2.4.4 Tableau 3-UC « Gérer Liste Clients » . . . . .	11
2.5 Diagramme de classe . . . . .	12
2.6 Diagrammes de collaboration . . . . .	13
2.6.1 Diagramme de collaboration « Recharger Véhicule » . . . . .	13
2.6.2 Diagramme de collaboration « Charger Batterie » . . . . .	14
2.6.3 Diagramme de collaboration « Gérer liste des clients » . . . . .	14
2.6.4 Diagramme de collaboration « Reprise véhicule » . . . . .	15
2.7 Machine à états « Générateur Save » . . . . .	15
2.8 Contrats Types . . . . .	16
<b>3 Implémentation</b>	<b>19</b>
3.1 Lecteur_carte.h . . . . .	19
3.2 Lecteur_carte.c . . . . .	19
3.3 GenerateurSave.h . . . . .	21
3.4 GenerateurSave.c . . . . .	21
3.5 Prisetrape.h . . . . .	24
3.6 Prisetrape.c . . . . .	24
3.7 timer.h . . . . .	25
3.8 timer.c . . . . .	25
3.9 VoyantBouton.h . . . . .	26
3.10 VoyantBouton.c . . . . .	26
3.11 baseDonnee.h . . . . .	28
3.12 baseDonnee.c . . . . .	29

<b>4 Conclusion</b>	<b>30</b>
<b>5 Annexe</b>	<b>31</b>

## Table des figures

---

1.1	Les principales parties de la phase d'analyse. . . . .	5
1.2	Diagramme de Contexte. . . . .	6
2.1	Diagramme de cas d'utilisation (UC). . . . .	7
2.2	Diagramme de séquence UC « Recharger Véhicule » . . . . .	9
2.3	Diagramme de séquence UC « Charger Batterie » . . . . .	10
2.4	Diagramme de séquence UC « Reprendre Véhicule » . . . . .	11
2.5	Diagramme de séquence UC « Gérer liste clients » . . . . .	12
2.6	Diagramme de classe. . . . .	13
2.7	Diagramme de collaboration "Recharger Véhicule" . . . . .	13
2.8	Diagramme de collaboration "Charger Batterie" . . . . .	14
2.9	Diagramme de collaboration "Gérer liste des clients" . . . . .	14
2.10	Diagramme de collaboration "Reprise véhicule" . . . . .	15
2.11	Machine à états liée au cas d'utilisation "Charger Batterie" . . . . .	15

# 1. ANALYSE

## 1.1. INTRODUCTION

Le rapport vise à concevoir un système simplifiant le fonctionnement d'une borne de recharge de véhicule électrique pour répondre aux exigences du cahier des charges. En utilisant le langage UML, nous pouvons fournir une spécification claire et précise pour la conception. La première étape du processus implique une analyse approfondie du cahier des charges du système à concevoir. Cette analyse permet de saisir les exigences et les spécifications du projet, établissant ainsi une base solide pour la conception. Par la suite, une phase de spécification des différentes fonctions du logiciel à développer est entreprise. Ces fonctions sont explicitées à travers plusieurs "cas d'utilisation" (UC). Chaque UC est accompagné de diagrammes de séquence et de collaboration, offrant une décomposition séquentielle ou non des actions nécessaires pour accomplir chaque fonction du logiciel. Enfin, en vue de la phase ultérieure d'écriture et de codage en langage C, des machines à états peuvent être élaborées. Ces machines à états décrivent le comportement du système dans divers états et définissent les transitions entre eux. Elles offrent une vision claire de la logique du logiciel, facilitant ainsi sa mise en œuvre ultérieure.

## 1.2. ANALYSE DU MINI PROJET

Un projet informatique suit une démarche qui débute par une phase d'analyse, suivie par une étape de conception. Durant la phase d'analyse, l'objectif premier est de comprendre minutieusement et de décrire de manière précise les besoins des utilisateurs ou des clients. Quels sont leurs souhaits en termes d'utilisation du logiciel? Quelles fonctionnalités recherchent-ils? À quel dessein souhaite-t-on utiliser le logiciel? Comment les actions devraient-elles être exécutées? Cette étape est communément appelée "analyse des besoins". Une fois que notre compréhension des besoins est validée, nous procédons à la conceptualisation de la solution, marquant ainsi la partie analyse de la solution.

Dans la phase de conception qui suit, l'objectif est d'apporter des détails supplémentaires à la solution et de clarifier des aspects techniques, tels que l'agencement des différentes composantes logicielles sur le matériel disponible. Pour mener à bien ces deux phases d'un projet informatique, l'utilisation de méthodes, de conventions et de notations spécifiques s'avère essentielle. C'est là que l'UML (Unified Modeling Language) trouve son intérêt, car il fournit un ensemble de conventions visuelles et de notations pour représenter efficacement les concepts et les relations impliqués dans l'analyse et la conception du système informatique. En résumé, lors de la phase d'analyse dans notre cas, l'attention se porte principalement sur trois éléments clés, comme illustré dans le schéma suivant :

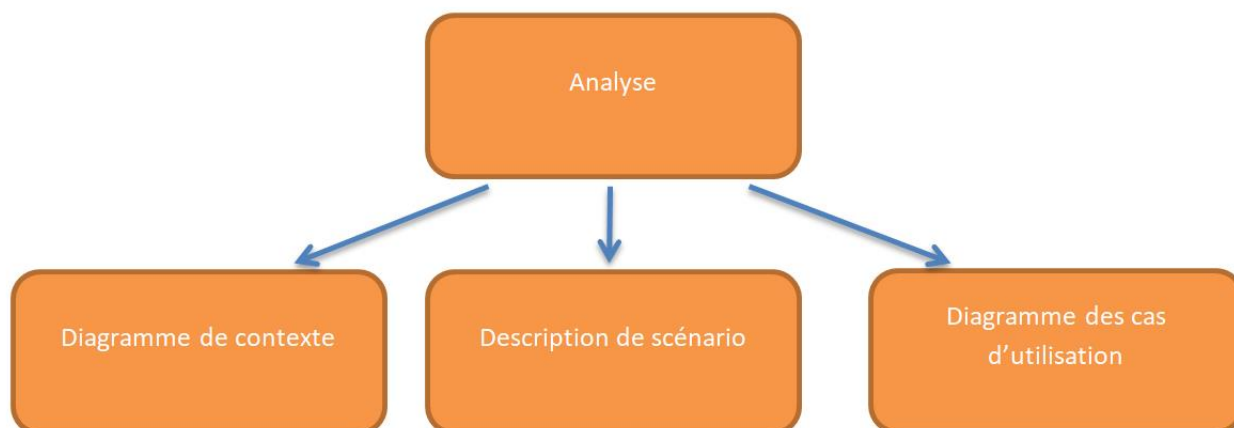


Figure 1.1 – Les principales parties de la phase d'analyse.

### 1.2.1. Diagramme de contexte

Le diagramme de contexte sert à établir les frontières de l'étude en identifiant les éléments extérieurs qui ont un impact sur le système ou qui interagissent avec lui. Il contextualise le diagramme en énumérant les acteurs ou composants qui influent sur le système. Le niveau de détail de ce diagramme varie en fonction des objectifs, permettant de mettre en évidence les interactions essentielles en fonction des besoins spécifiques de la représentation.

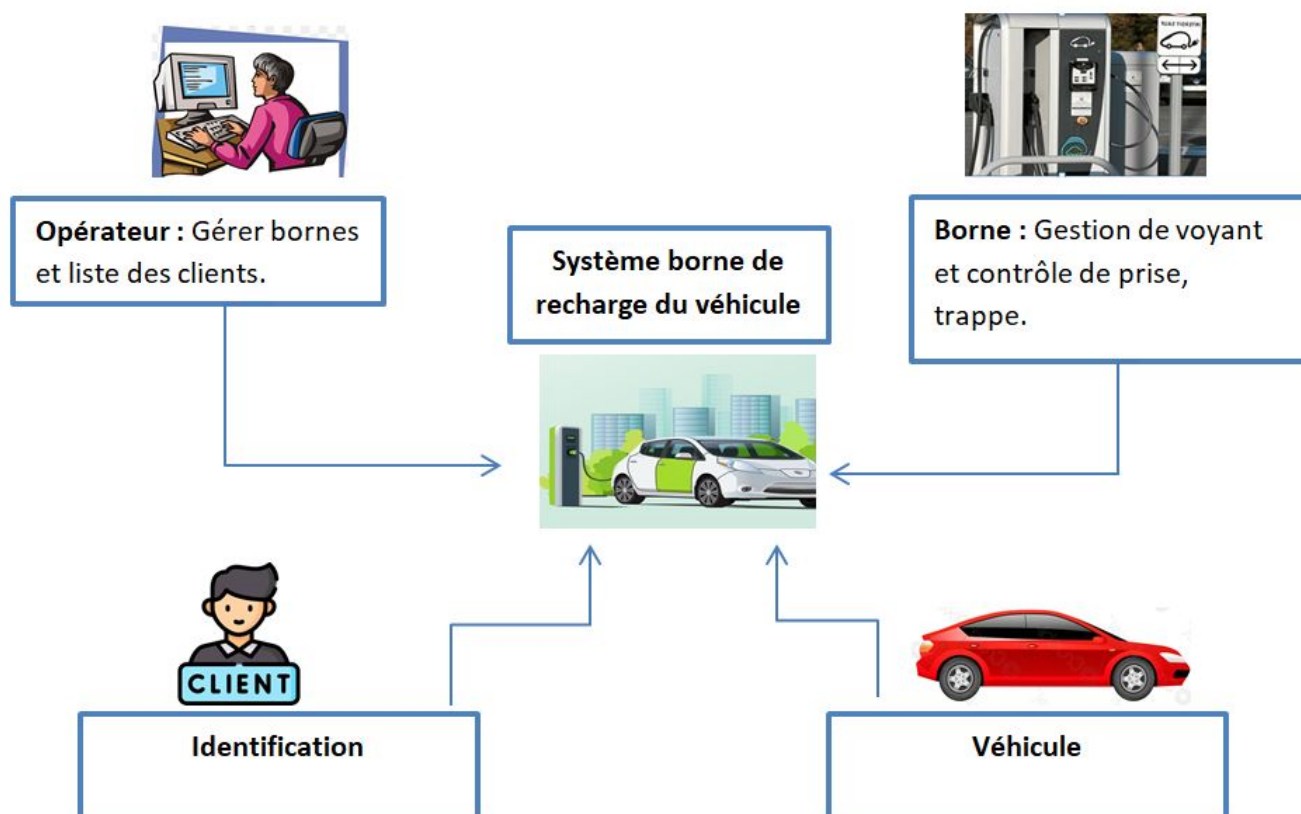


Figure 1.2 – Diagramme de Contexte.

## 2. Conception

Cette étude permet de faire le bilan des besoins d'une application demandée par un client tout en prenant en compte également les contraintes associées.

### 2.0.1. Élément physique du système :

- Voyants.
- Lecteur de badge.
- Bouton poussoir.
- Prise.
- Clavier.
- Écran.
- Système de communication.

### 2.0.2. Définition des besoins :

- Lire carte.
- Communiquer avec opérateur.
- Recharger batterie.

- Gérer des voyants.
- Identifier les clients.
- Déverrouiller socle de prise.

### 2.0.3. Les éléments de l'environnement en relation avec le système :

Les acteurs du système :

- Client.
- Véhicule.
- Opérateur.
- Borne.

## 2.1. Diagrammes de cas d'utilisation

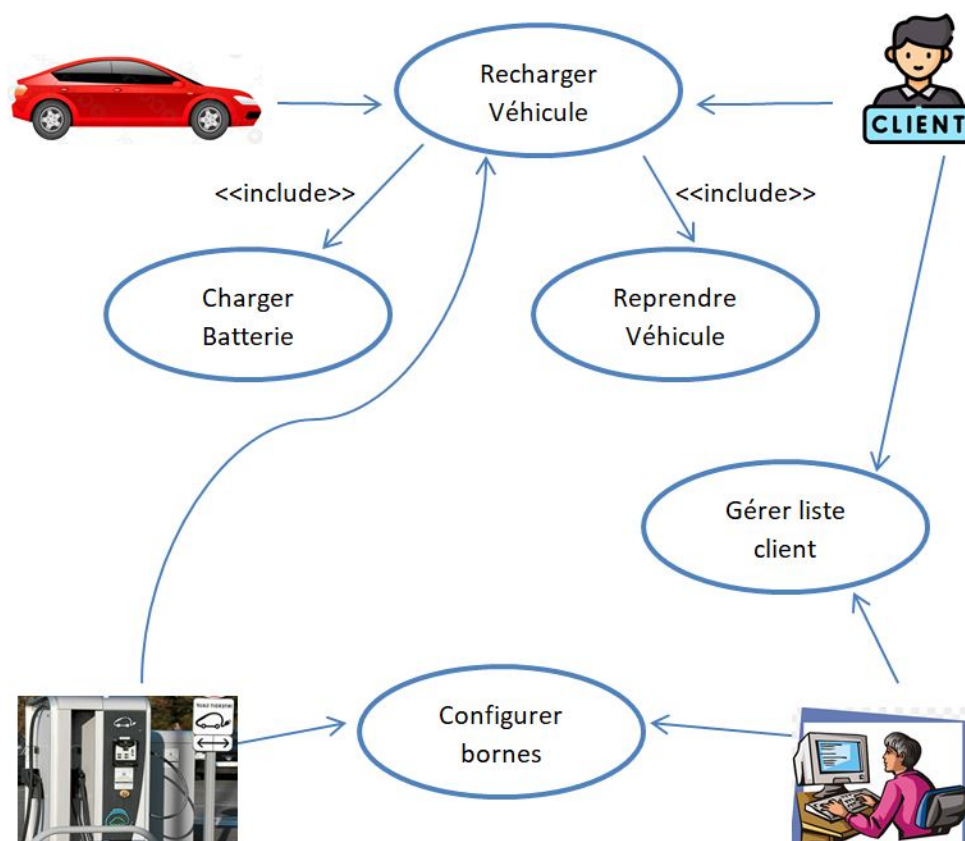


Figure 2.1 – Diagramme de cas d'utilisation (UC).

En se basant sur ce diagramme on peut définir chaque cas d'utilisation.

## 2.2. Description des UC :

### 2.2.1. Recharger Véhicule

**Acteur :** Client, Véhicule, Borne.

**Type :** Principale.

**Description :** Le client se connecte à la borne pour déclencher le processus de recharge de son véhicule.

### 2.2.2. Charger Batterie

**Acteur :** Véhicule, Borne.

**Type :** Sous cas d'UC « Recharger Véhicule ».

**Description :** Le système effectue la recharge de la batterie en respectant le mode de charge.

### 2.2.3. Reprendre Véhicule

**Acteur :** Client, Opérateur, Borne

**Type :** Sous cas

**Description :** L'opérateur s'identifier à nouveau pour reprendre son véhicule.

### 2.2.4. Configurer Borne

**Acteur :** Opérateur, Borne.

**Type :** Principale.

**Description :** L'opérateur configure la borne et peut obtenir l'état des statistiques d'utilisation de la borne.

### 2.2.5. Gérer Liste Clients

**Acteur :** Opérateur, Borne.

**Type :** Principale.

**Description :** La borne vérifie que l'identifiant de la carte du client correspond à un numéro répertorié dans la base de clients.

## 2.3. Diagramme de séquence :

Le diagramme de séquence illustre le comportement dynamique orienté dans le temps de chaque cas d'utilisation.

Dans cette représentation visuelle, chaque entité est symbolisée par une barre verticale, permettant de modéliser de manière détaillée le flux de contrôle.

L'objectif principal est de mettre en évidence les scénarios d'interaction entre les différentes entités du système, fournissant ainsi une perspective temporelle et séquentielle des événements au sein du processus.

## 2.4. Définition des scénarios :

Les cas d'utilisation sont décrits à travers des scénarios qui détaillent la séquence logique des actions constituant ces cas.

### 2.4.1. Tableau 1 - UC « Recharger Véhicule »

Acteurs	Système
1. Le client insère sa carte.	2. Le système lit la carte et identifie le client.
4. Le client appuie sur le bouton charge.	3. Le système fait clignoter le voyant « charge » en vert pendant 8 secondes.
6. Le client retire sa carte.	5. Le système éteint le voyant « disponible ».
	7. Le système déclenche la charge de la batterie : Ref-UC « Charger Batterie ».
	8. Le système allume le voyant « disponible » à vert.
<b>Variante 1 délai de passé</b>	
4. Le système détecte que le bouton charge n'a pas été appuyé.	5. Le système allume le voyant « disponible » à vert.
<b>Variante 2 client inconnu</b>	
2. Le système détecte un numéro de carte inconnu.	3. Le système fait clignoter le voyant « défaut » à « rouge » pendant 8s.
4. Le client retire sa carte.	

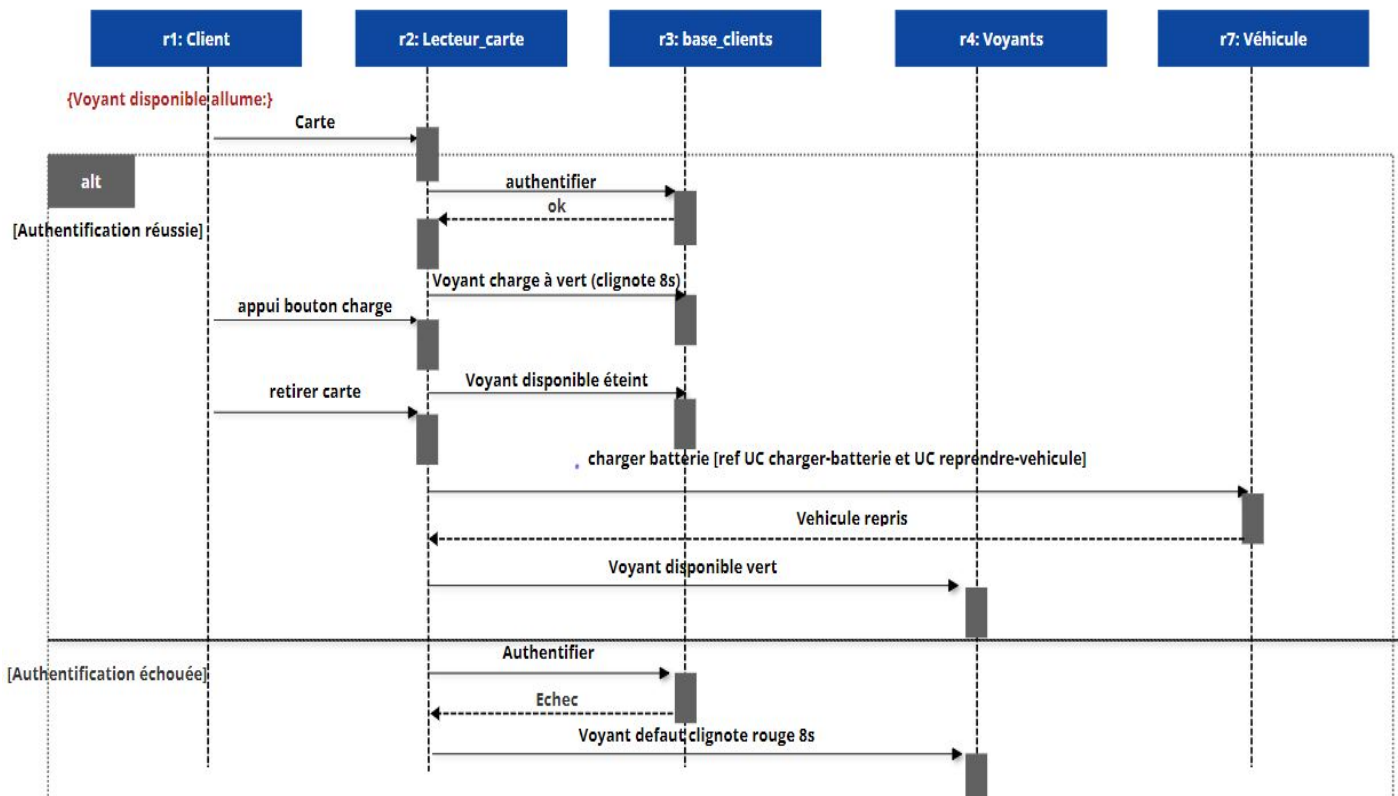


Figure 2.2 – Diagramme de séquence UC « Recharger Véhicule »

## 2.4.2. Tableau 2 - UC « Charger Batterie »

Acteurs	Système
4. Le client branche la prise du véhicule.	1. Le système allume le voyant charge en « ROUGE ».
5. Le véhicule fait chuter la tension à 9V.	2. Le système déverrouille la trappe de la prise.
9. Le véhicule ferme le contacteur S2.	3. Le système génère un signal continu 12V.
10. La tension chute à 6V.	6. Le système allume le voyant prises-en « VERT ».
13. Le véhicule (du fait de la charge de la batterie) fait remonter progressivement la tension à 9V.	7. Le système verrouille la trappe de prise.
16. Le véhicule ouvre le contacteur S2.	8. Le système génère un signal PWM de fréquence 1KHz.
	11. Le système ferme le contacteur AC.
	12. Le système génère un signal PWM Variable.
	14. Le système ouvre le contacteur AC.
	15. Le système génère une tension continue.
	17. Le système allume le voyant charge à « VERT ».
	18. Le système autorise la reprise du véhicule.

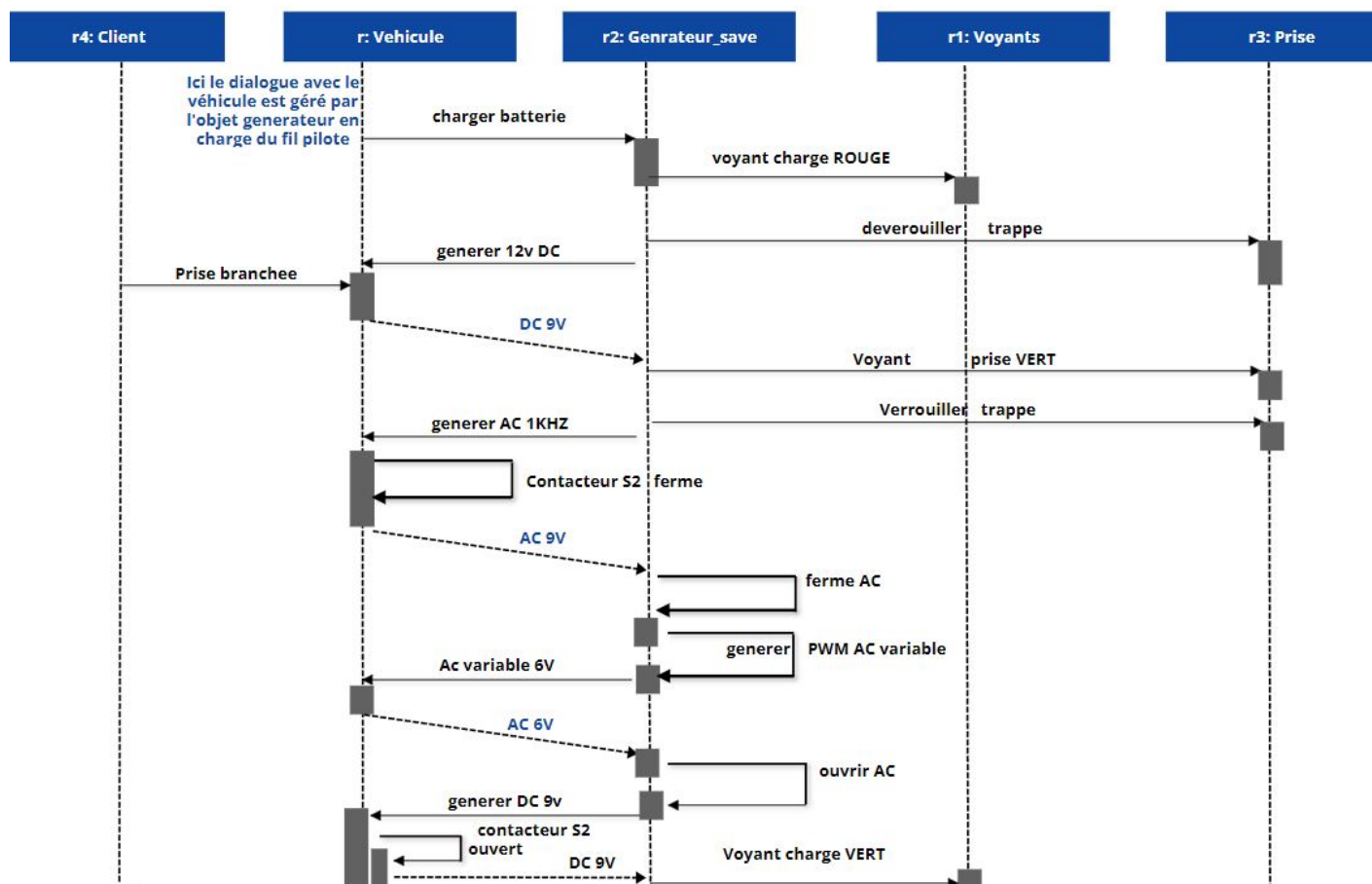


Figure 2.3 – Diagramme de séquence UC « Charger Batterie »

### 2.4.3. Tableau 3-UC « Reprendre Véhicule »

Acteurs	Système
1. Le client s'identifie à nouveau.	2. Le véhicule est en charge.
6. Le client débranche la prise.	3. Appuyer sur STOP.
	4. AC ouvert.
	5. Prise déverrouillée.
	7. Trappe verrouillée.
	8. Voyant prise éteint.
	9. Voyant disponible allumé.

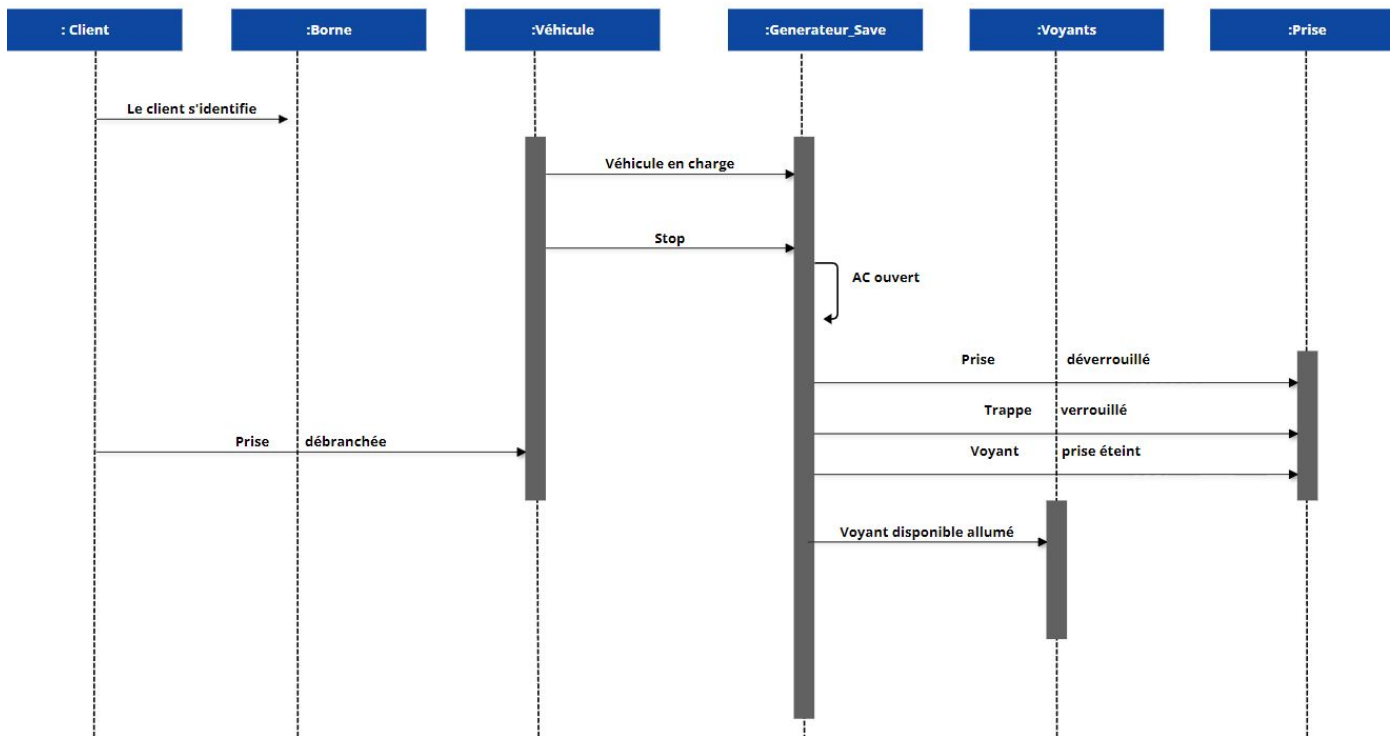


Figure 2.4 – Diagramme de séquence UC « Reprendre Véhicule »

### 2.4.4. Tableau 3-UC « Gérer Liste Clients »

Acteurs	Système
1. Le client insère sa carte	2. Le système identifie le client.
	3. Le système vérifie que le numéro de la carte du client est bien dans la base de données.

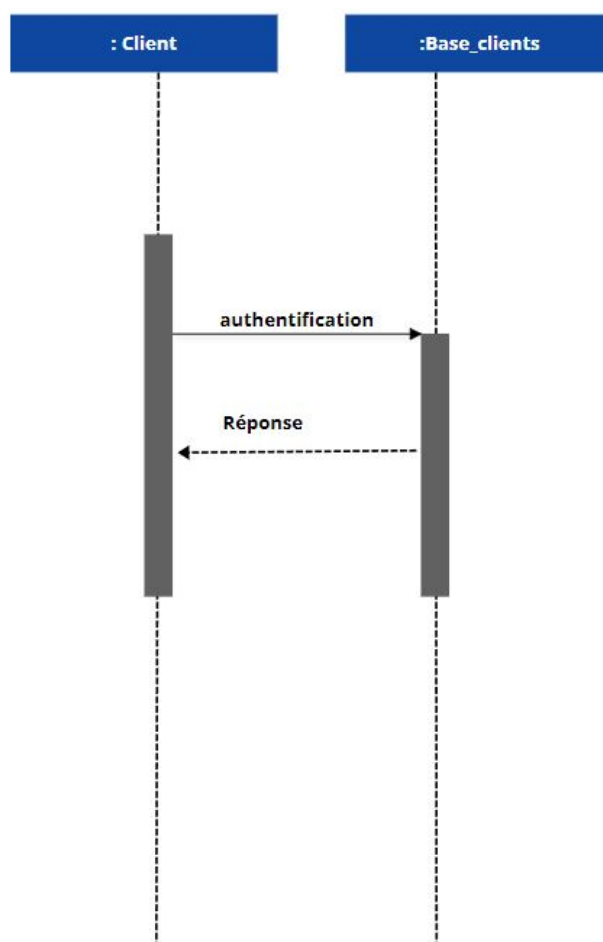


Figure 2.5 – Diagramme de séquence UC « Gérer liste clients »

## 2.5. Diagramme de classe

Les diagrammes de classes constituent l'un des types de diagrammes UML les plus essentiels, offrant une représentation claire de la structure d'un système donné en détaillant ses classes, attributs, opérations, ainsi que les relations inter-objets. Ces schémas visuels présentent de manière exhaustive l'interaction entre les différentes classes d'un système, mettant l'accent sur la structure statique plutôt que sur la dynamique temporelle du système. En d'autres termes, ils offrent une vue statique, mais détaillée, des composants et de leurs relations au sein du système, facilitant ainsi la compréhension de son architecture.

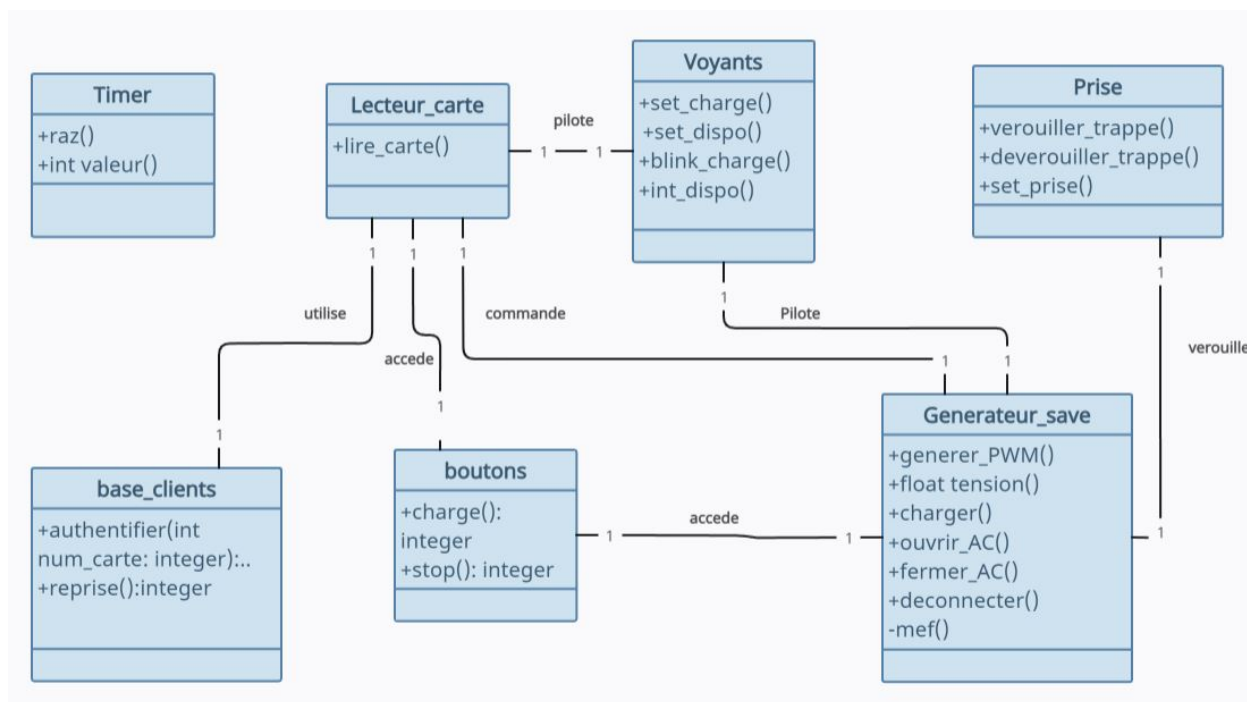


Figure 2.6 – Diagramme de classe.

## 2.6. Diagrammes de collaboration

En UML, les diagrammes de collaboration permettent une représentation simplifiée des diagrammes de séquence en ignorant la chronologie, mais en se concentrant sur les échanges de messages entre les objets. Les diagrammes de collaboration des UCs « Recharger Véhicule » et « Charger Batterie » sont représentés »

### 2.6.1. Diagramme de collaboration « Recharger Véhicule »

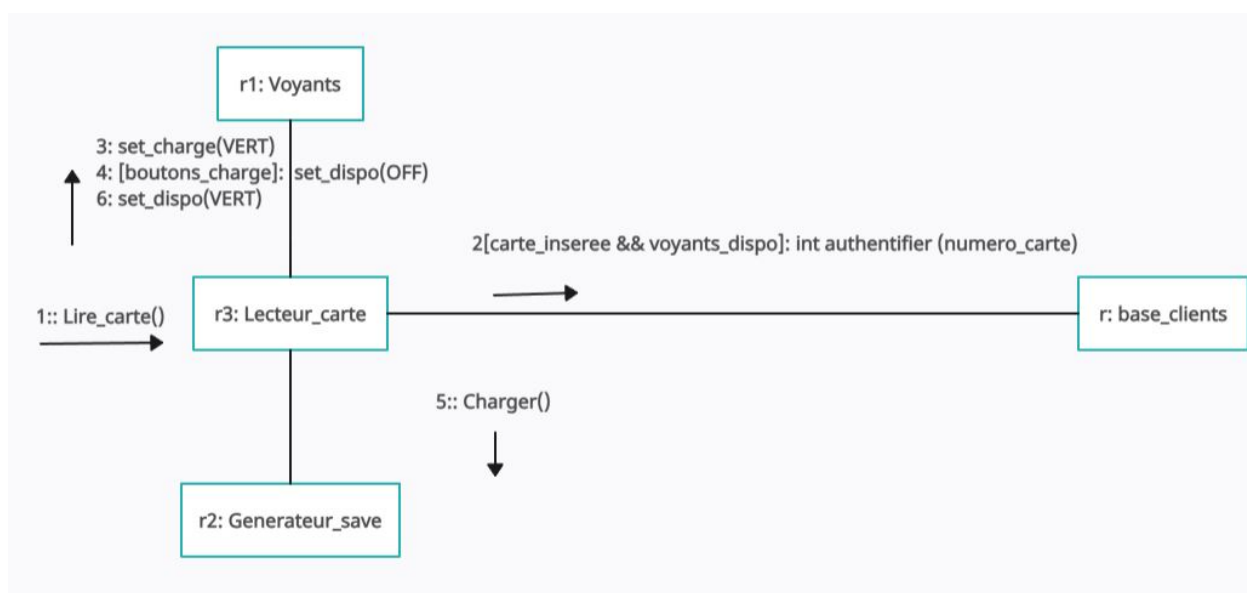


Figure 2.7 – Diagramme de collaboration "Recharger Véhicule"

## 2.6.2. Diagramme de collaboration « Charger Batterie »

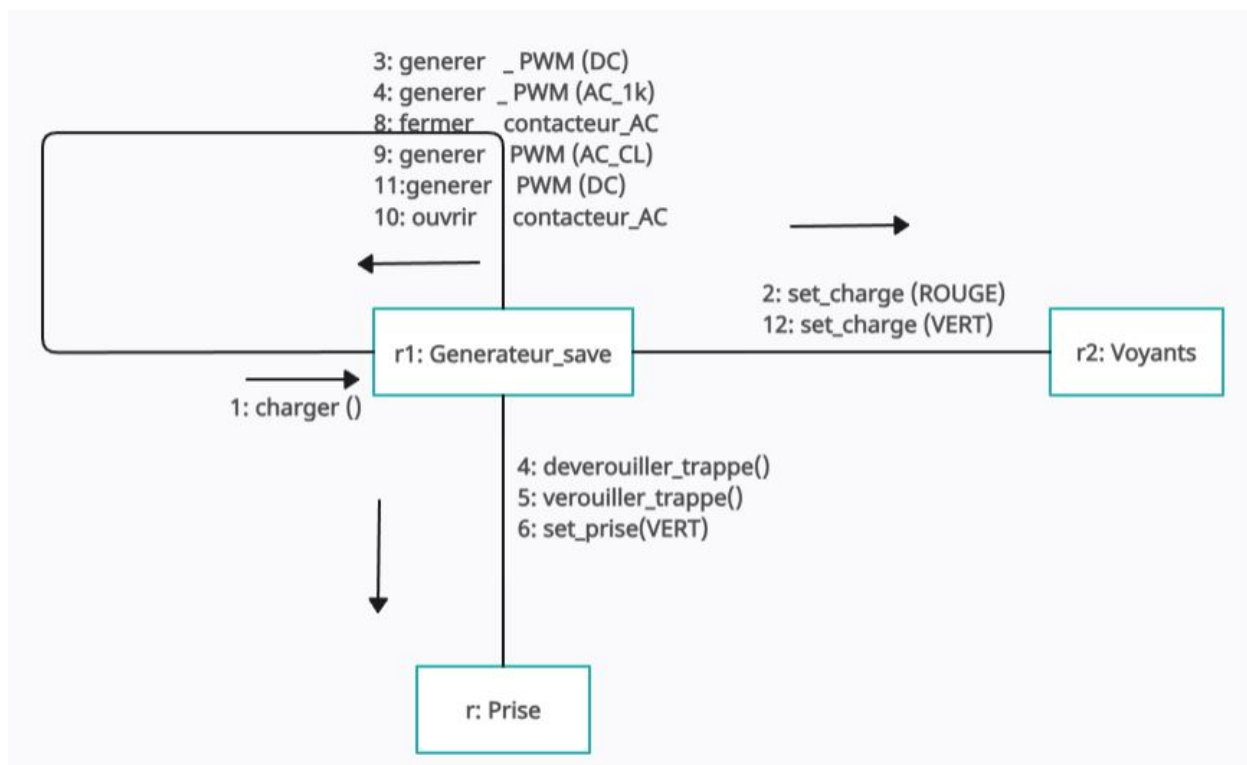


Figure 2.8 – Diagramme de collaboration "Charger Batterie"

## 2.6.3. Diagramme de collaboration « Gérer liste des clients »

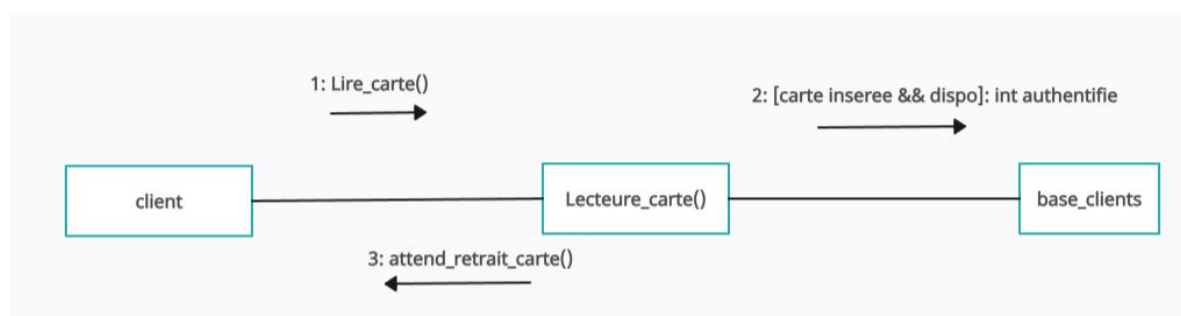


Figure 2.9 – Diagramme de collaboration "Gérer liste des clients"

## 2.6.4. Diagramme de collaboration « Reprise véhicule »

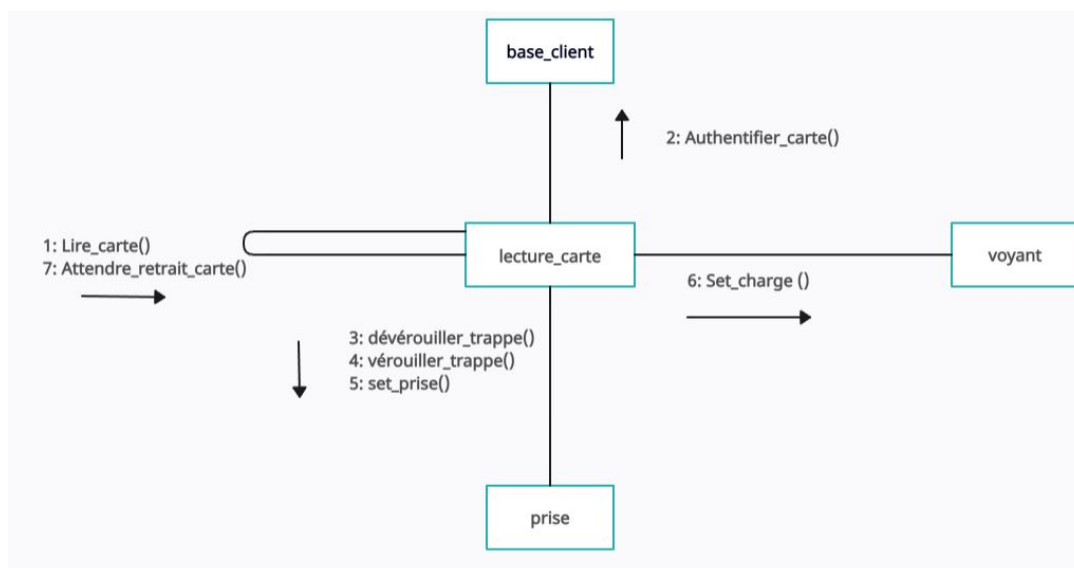


Figure 2.10 – Diagramme de collaboration "Reprise véhicule"

## 2.7. Machine à états « Générateur Save »

Les machines à états représentent des modèles où une transition peut déclencher l'émission d'un message de sortie vers l'environnement avant que la machine n'entre dans le nouvel état défini par cette transition. En UML, ces machines à états prennent en charge des actions qui dépendent de l'état du système et des événements déclencheurs, tout en incluant des actions d'entrée et de sortie dépendant de l'état.

Dans notre cas, nous appliquons la machine à états pour modéliser le cas d'utilisation du chargement de la batterie, simplifiant ainsi la conception globale du système. L'adoption de cette approche facilite également la programmation, offrant une méthode plus claire.

Concrètement, lorsque la génératrice ( $gene_u$ ) est connectée, la tension passe à 9V. Afin d'effectuer une transition d'un état à un autre, il est impératif que la valeur de  $gene_u$  change. De plus, le processus de chargement peut être interrompu en utilisant la commande **STOP**, ajoutant ainsi une flexibilité opérationnelle au système.

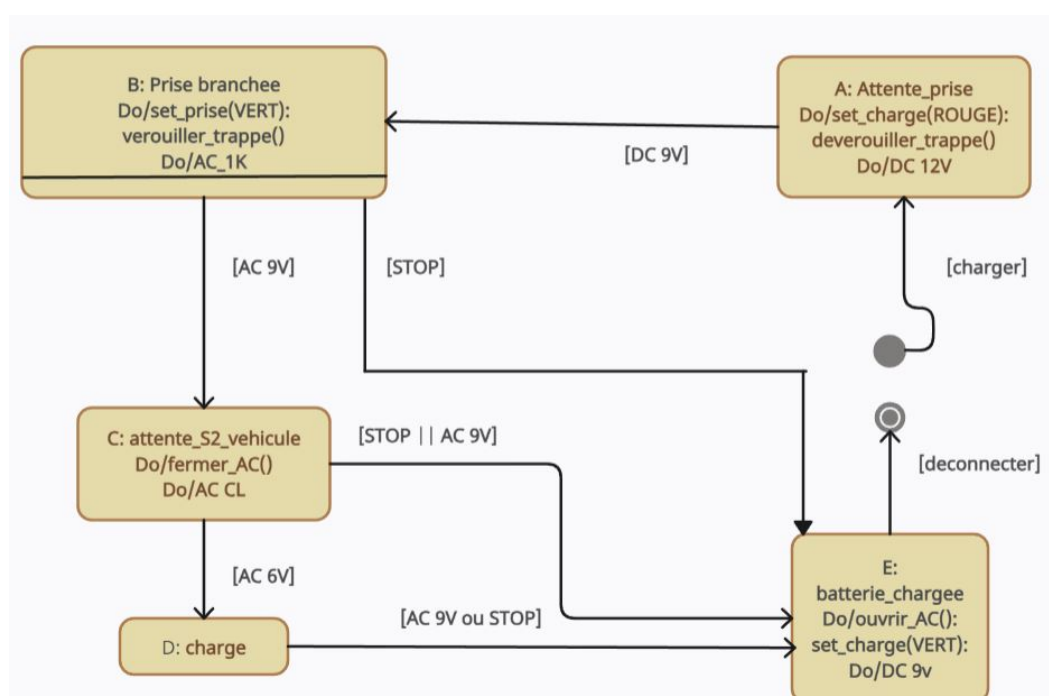


Figure 2.11 – Machine à états liée au cas d'utilisation "Charger Batterie"

## 2.8. Contrats Types

L'objectif est de définir les éléments à vérifier lors de l'exécution d'un programme. Pour chaque programme, nous fournirons son nom, son type, ses entrées, ses conditions préalables et postérieures, ainsi que sa référence croisée.

### **void lecteurcarte\_Lire\_carte ()**

- **Responsabilité** : Lecteur de la carte.
- **Type** : logiciel-public.
- **Références croisées** : U.C « Recharge véhicule », U.C « Reprendre véhicule », en relation avec les classes `base client`, `générateur Save`, `voyant`.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Lecture carte initialisation.
- **Postcondition** : Aucune.

### **void lecteurcarte\_Lire\_numcarte ()**

- **Responsabilité** : Affichage numéro de la carte.
- **Type** : logiciel-public.
- **Références croisées** : U.C « Recharge véhicule », U.C « Reprendre véhicule », en relation avec les classes `base client`, `générateur Save`.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Aucune.
- **Postcondition** : Numéro de la carte.

### **set\_charge ()**

- **Responsabilité** : Allumé quand il y a une voiture en train de se charger.
- **Type** : interface-public.
- **Références croisées** : « Recharge véhicule ».
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Bouton charge.
- **Postcondition** : Le voyant charge s'allume après avoir appuyé sur le bouton charge.

### **int GetBoutonCharge () :**

- **Responsabilité** : Retourne l'état du bouton de charge.
- **Type** : logiciel-public.
- **Références croisées** : « Recharge véhicule », en relation avec les classes `voyant`, `SaveGénérateur`.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Remis à 0.
- **Postcondition** : Retourne 1 ou bien 0, dépendant de l'état du bouton charge.

### **int GetBoutonStop () :**

- **Responsabilité** : Retourne l'état.
- **Type** : logiciel-public.
- **Références croisées** : U.C « Recharge véhicule », U.C « Reprendre véhicule », en relation avec les classes `voyant`, `SaveGénérateur`.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Remis à 0.
- **Postcondition** : Retourne 1 ou bien 0, dépendant de l'état du bouton charge.

### **int authentifier(unsigned short int code) :**

- **Responsabilité** : Vérifier l'authentification d'un client et son existence dans la base de données.
- **Type** : logiciel-public.
- **Références croisées** : User case « Recharge véhicule », User case « Reprendre véhicule ».
- **Paramètre d'entrée** : `code`.
- **Précondition** : La carte insérée.
- **Postcondition** : Retourne 1 si l'authentification est réussie, sinon retourne 0.

### **void trappe\_verouiller () :**

- **Responsabilité** : Verrouiller la trappe.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie », U.C. « Reprendre véhicule ».
- **Algorithme** : Déverrouiller la trappe soit avant le branchement de la prise ou bien son débranchement.

- **Paramètre d'entrée** : Aucun.
- **Précondition** : Prise branchée ou débranchée.
- **Postcondition** : Aucune.

**void trappe\_deverrouiller () :**

- **Responsabilité** : Déverrouiller la trappe.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie », U.C. « Reprendre véhicule ».
- **Algorithme** : Déverrouiller la trappe soit avant le branchement de la prise ou bien son débranchement.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Authentification et (la batterie est chargée ou bien stop).
- **Postcondition** : Aucune.

**void fermerAC() :**

- **Responsabilité** : Fermeture du contacteur.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie », U.C. « Reprendre véhicule ».
- **Algorithme** : Fermeture du contacteur.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : S2 fermé et  $gene\_u=6v$ .
- **Postcondition** : Aucune.

**void ouvrirAC() :**

- **Responsabilité** : Ouverture du contacteur.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie », U.C. « Reprendre véhicule ».
- **Algorithme** : Fermeture du contacteur.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : S2 ouvert et  $gene\_u=9v$ .
- **Postcondition** : Aucune.

**void set\_prise(led color) :**

- **Responsabilité** : Allumer la LED prise.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie », U.C. « Reprendre véhicule ».
- **Paramètre d'entrée** : Couleur de la LED.
- **Précondition** : Véhicule connecté.
- **Postcondition** : Aucune.

**void generer\_PWM(pwm u) :**

- **Responsabilité** : Génération d'un signal (ça dépend du signal  $u$ ).
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie ».
- **Paramètre d'entrée** :  $u$  qui retourne la tension sur le fil pilote.
- **Précondition** : AC fermé.
- **Algorithme** : La génération d'une PWM doit prendre successivement les valeurs STOP, DC, AC\_1K, AC\_CL selon le mode de commande de charge.
- **Postcondition** : Aucune.

**int timer\_raz() :**

- **Responsabilité** : Mémoriser l'heure du déclenchement du timer dans un attribut et une méthode.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie ».
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Bouton charge.
- **Postcondition** : L'heure est mémorisée.

**void reprendre() :**

- **Responsabilité** : La reprise du véhicule.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie » et U.C. « Reprendre véhicule ».

- **Algorithme** : Le client peut reprendre sa véhicule une fois la batterie est chargée ou bien s'il a appuyé sur stop.
- **Paramètre d'entrée** : Aucun.
- **Précondition** : Prise débranchée ( $gene\_u=12v$ ).
- **Postcondition** : Aucune.

**int timer\_valeur() :**

- **Responsabilité** : Renvoyer la différence entre l'horloge actuelle et l'heure du déclenchement.
- **Type** : logiciel-public.
- **Références croisées** : U.C. « Charge batterie ».
- **Paramètre d'entrée** : Aucun.
- **Précondition** : `timer_raz`.
- **Postcondition** : Aucune.

### 3. Implémentation

Après avoir mené l'analyse et la conception, nous avons procédé à la programmation des classes ainsi que de leurs méthodes en utilisant le langage C. Nous avons structuré notre travail à l'aide de fichiers composés, où les actions réalisées sont enregistrées dans les fichiers ".c", tandis que les prototypes de toutes les méthodes sont regroupées dans les fichiers ".h".

#### 3.1. Lecteur\_carte.h

```
#include <lcarte.h>
void lecteurcarte_initialiser();
void lecteurcarte_lire_carte();
```

#### 3.2. Lecteur\_carte.c

```
#include <unistd.h>
#include "lecteurcarte.h"
#include "lcarte.h"
#include <stdio.h>
#include "donnees_borne.h"
#include "donnees.h"
#include "mem_sh.h"
#include "VoyantBouton.h"
#include "timer.h"
#include "Prisetrape.h"
#include "baseDonnee.h"
#include "GenerateurSave.h"

% Fonction pour initialiser le lecteur de carte
\begin{lstlisting}[language=C]
void lecteurcarte_initialiser()
{
    % Appel de la fonction d'initialisation des ports
    initialisations_ports();
}
% Fonction pour lire la carte inseree
void lecteurcarte_lire_carte()
{
    % Declaration des variables
    int num;
    int i;
    int time;
    int ver;

    % Affichage d'un message pour indiquer a l'utilisateur d'insérer la carte
    printf("Insérer la carte svp\n");

    % Attente de l'insertion de la carte
    attente_insertion_carte();

    % Verification si une carte est inseree
    if (carte_inseree() == 1)
    {
        % Affichage d'un message indiquant que la carte est inseree
```

```

printf("Carte inseree\n");

% Lecture du numero de la carte
num = lecture_numero_carte();

% Affichage du numero de la carte
printf("%i\n", num);

% Verification si le numero de la carte est dans la base de donnees
for (i = 0; i < 10; i++)
{
    printf("tab1[%d] = %d\n", i, tab1[i]);
    if (num == tab1[i])
    {
        ver = 1; % Numero de carte trouve dans la base de donnees
        break;
    }
    else
        ver = 0; % Numero de carte non trouve dans la base de donnees
}

% Affichage du resultat de la verification
printf("ver = %d\n", ver);

% Traitement en fonction du resultat de la verification
if (ver == 1)
{
    printf("Le code est bon\n");

    % Affichage d'un message demandant d'appuyer plusieurs fois sur
    le bouton de charge
    printf("Veuillez appuyer plusieurs fois sur le bouton charge\n");

    i = 6; % Si le code est valide, on sort
           de la boucle apres la boucle for
    timer_raz(); % Memorisation du temps de declenchement du timer
    clignoteCharge(); % Reprise apres clignotement de 8 secondes

    do
    {
        GetBoutonCharge();
        time = timer_valeur();
        printf("time=%d\n", time);
        usleep(5000);
    } while (time <= 60 && GetBoutonCharge() != 1);

    % Verification du temps ecoule et de l'etat du bouton de charge
    if (time < 60)
    {
        resetDispo();
        Set_Charge();
        trappe_deverouiller();
        generer_PWM(DC);
        charger();
        reprendre();
        printf("Reprendre\n");
    }
    else
    {
        set_dispo();
    }
}

```

```

    }
}
else
{
    printf("Le code est faux\n");
    clignoteDefaut();
}

% Affichage d'un message demandant de retirer la carte
printf("Retirer votre carte svp et merci pour votre visite :)\n");

% Attente du retrait de la carte
attente_retrait_carte();

% Affichage d'un message indiquant que la carte a ete retiree
printf("Carte retiree\n");
}
else
{
    % Affichage d'un message indiquant que la carte n'est pas inseree
    printf("Carte n'est pas inseree\n");
}
}

```

### 3.3. GenerateurSave.h

```

#include <unistd.h>
#include "lecteurcarte.h"
#include "lcarte.h"
#include "donnees_borne.h"
#include "donnees.h"
#include "mem_sh.h"
# include "VoyantBouton.h"
#include <stdio.h>
#include <stdlib.h>

void generateuresave_initialiser();
void generer_PWM(pwm u);
void charger();
void fermerAC();
void ouvrirAC();
void reprendre();

```

### 3.4. GenerateurSave.c

```

#include "GenerateurSave.h"
#include "Prisetrape.h"
#include "baseDonnee.h"

% Declaration des variables globales
entrees *io;
int shmid;

% Initialisation du generateur de sauvegarde
\begin{lstlisting}[language=C]
void generateuresave_initialiser()
{

```

```

    io = acces_memoire(&shmid);
}

% Fonction pour generer une PWM

void generer_PWM(pwm u)
{
    io->gene_pwm = u;
}

% Fonction pour charger le vehicule

void charger()
{
    int i = 0;
    int charge = 0;
    int k = 0;

    while (charge == 0)
    {
        i = k;

        switch (i)
        {
            case 0:
            {
                if (io->gene_u == 9)
                    k = 1;

                if (GetBoutonStop() == 1)
                    k = 4;
            }break;

            case 1:
            {
                trappe_verouiller();
                generer_PWM(AC_1K);
                if (GetBoutonStop() == 1)
                    k = 4;
                if (io->gene_u == 6)
                    k = 2;
            }break;

            case 2:
            {
                fermerAC();
                generer_PWM(AC_CL);
                if (GetBoutonStop() == 1)
                    k = 4;
                else
                    k = 3;
            }break;

            case 3:
            {
                if (io->gene_u == 9 || GetBoutonStop() == 1)
                    k = 4;
            }break;
        }
    }
}

```

```

        case 4:
        {
            ouvrirAC();
            generer_PWM(DC);
            Set_ChargeVert();
            charge = 1;
            k = 0;
            printf("ok\n");
        }break;
    }
}

% Fonction pour fermer le contacteur AC

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

% Fonction pour ouvrir le contacteur AC

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

% Fonction pour reprendre la charge
void reprendre()
{
    int num;
    int i;

    % Verification si une carte est inseree
    if (1)
    {
        printf("Carte inseree\n");
        num = lecture_numero_carte();

        printf("%i\n", num);

        % Authentification de la carte
        for (i = 0; i < 3; i++)
        {
            if (authentifier(num) == 1)
            {
                i = 6;

                if (GetBoutonStop())
                {
                    printf("Le code est bon\n");
                    ouvrirAC();
                    printf("Veuillez debrancher la prise svp\n");
                    trappe_deverouiller();

                    while (io->gene_u != 12)
                    {

```

```

        usleep(100);
    }

    trappe_verouiller();
    set_dispo();
    reset_prise();
    reset_charge();
}
else
{
    printf("Veuillez debrancher la prise svp\n");
    trappe_deverouiller();

    while (io->gene_u != 12)
    {
        usleep(100);
    }

    trappe_verouiller();
    set_dispo();
    reset_prise();
    reset_charge();
}
}
else
{
    printf("Le code est faux\n");
    clignoteDefaut();
}
}
}
}
}

```

### 3.5. Prisetrape.h

```

#include <unistd.h>
#include "lecteurcarte.h"
#include "lcarte.h"
#include <stdio.h>
#include "donnees_borne.h"
#include "donnees.h"
#include "mem_sh.h"

void prise_initialiser();
void trappe_deverouiller();
void trappe_verouiller();
void set_prise(led color);
void reset_prise();

```

### 3.6. Prisetrape.c

```

#include <unistd.h>
#include <lcarte.h>
#include <donnees_borne.h>
#include <memoire_borne.h>
#include "Prisetrape.h"

```

```
entrees *io;
int shmid;

void prise_initialiser()
{
    io = acces_memoire(&shmid);
}

void trappe_deverouiller()
{
    io->led_trappe = VERT;
}

void trappe_verouiller()
{
    io->led_trappe = OFF;
}

void set_prise(led color)
{
    if (color == VERT)
    {
        io->led_prise = VERT;
    }

    if (color == OFF)
    {
        io->led_prise = OFF;
    }
}

void reset_prise()
{
    io->led_prise = OFF;
}
```

### 3.7. timer.h

```
#include <unistd.h>
#include "lecteurcarte.h"
#include "lcarte.h"
#include <stdio.h>
#include "donnees_borne.h"
#include "donnees.h"
#include "mem_sh.h"

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

### 3.8. timer.c

```
% ===== timer.c =====%
#include "timer.h"

% Declaration des variables globales
entrees *io;
int shmid;
```

```
int depart_date;

% Fonction pour initialiser le timer
\begin{lstlisting}[language=C]
void timer_initialiser()
{
    io = acces_memoire(&shmid);
    if (io == NULL)
        printf("Erreur : pas de memoire partagee\n");
}
% Fonction pour remettre a zero le timer
void timer_raz()
{
    depart_date = io->timer_sec;
}
% Fonction pour obtenir la valeur actuelle du timer
int timer_valeur()
{
    int val;
    val = io->timer_sec - depart_date;
    return (val);
}
```

### 3.9. VoyantBouton.h

```
#include <unistd.h>
#include "lecteurcarte.h"
#include "lcarte.h"
#include <stdio.h>
#include "donnees_borne.h"
#include "donnees.h"
#include "mem_sh.h"

void voyant_initialiser();
void clignoteDefault();
void clignoteCharge();
int GetBoutonCharge();
void resetDispo();
void Set_Charge();
void Set_ChargeVert();
int GetBoutonStop();
void set_dispo();
void reset_charge();
```

### 3.10. VoyantBouton.c

```
#include <unistd.h>
#include "lecteurcarte.h"
#include "lcarte.h"
#include <stdio.h>
#include <donnees_borne.h>
#include "donnees.h"
#include <memoire_borne.h>
#include "mem_sh.h"
#include "VoyantBouton.h"
#include "baseDonnee.h"
#include "Prisetrape.h"
```

```
#include "GénérateurSave.h"

% Declaration des variables globales
entrees *io;
int shmid;

% Fonction pour initialiser les voyants et les boutons
\begin{lstlisting}[language=C]
void voyant_initialiser()
{
    io = acces_memoire(&shmid);
    if (io == NULL)
        printf("Erreur : pas de memoire partagee\n");
}

% Fonction pour faire clignoter le voyant default
void clignoteDefaut()
{
    int i;
    for (i = 0; i <= 4; i++)
    {
        io->led_default = OFF;
        sleep(1);
        io->led_default = ROUGE;
        sleep(1);
    }
}

% Fonction pour faire clignoter le voyant charge
void clignoteCharge()
{
    int i;
    for (i = 0; i <= 4; i++)
    {
        io->led_charge = VERT;
        sleep(1);
        io->led_charge = OFF;
        sleep(1);
    }
}

% Fonction pour reinitialiser le voyant disponibilite
void resetDispo()
{
    io->led_dispo = OFF;
}

% Fonction pour activer le voyant disponibilite
void set_dispo()
{
    io->led_dispo = VERT;
}

% Fonction pour obtenir l'etat du bouton de charge
int GetBoutonCharge()
```

```
{
    io->bouton_charge = 0;
    usleep(5000);
    if (io->bouton_charge == 1)
    {
        printf("Bouton charge appuye\n");
        return 1;
    }
    else
        return 0;
}

% Fonction pour activer le voyant de charge

void Set_Charge()
{
    io->led_charge = ROUGE;
}

% Fonction pour reinitialiser le voyant de charge

void reset_charge()
{
    io->led_charge = OFF;
}

% Fonction pour activer le voyant de charge en vert
void Set_ChargeVert()
{
    io->led_charge = VERT;
}

% Fonction pour obtenir l'etat du bouton d'arret
int GetBoutonStop()
{
    io->bouton_stop = 0;
    usleep(5000);
    if (io->bouton_stop == 1)
        return 1;
    else
        return 0;
}
```

### 3.11. baseDonnee.h

```
#include <lcarte.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int authentifier(unsigned short int code);
void gerer_client();
```

### 3.12. baseDonnee.c

```
#include "baseDonnee.h"

% Declaration de la base de donnees avec 10 codes
unsigned short int tab1[10] = {451, 965, 777, 758,
123, 4495, 610, 456, 258, 149};

% Fonction pour authentifier un code dans la base de donnees
\begin{lstlisting}[language=C]
int authentifier(unsigned short int code)
{
    int i = 0, j = 0;
    for (i = 0; i < 10; i++)
    {
        if (code == tab1[i])
        {
            j = 1;
        }
    }
    return j;
}

% Fonction pour gerer un nouveau client et mettre a jour la base de donnees
void gerer_client()
{
    int card_num;
    FILE *fic;
    int code;

    fic = fopen("base_client.dat", "w"); % Ouverture du fichier pour ecriture

    if (fic == NULL)
    {
        fprintf(stderr, "L'ouverture de %s est impossible\n", "base_client.dat");
    }

    printf("Insérer la nouvelle carte svp\n");
    attente_insertion_carte();

    if (carte_inseree() == 1)
    {
        card_num = lecture_numero_carte();
        fprintf(fic, "%i\n", card_num); % Ecriture du numero
        de carte dans le fichier
    }

    fclose(fic); % Fermeture du fichier
}
\section{borne.c}
#include <stdio.h>
#include <memoire_borne.h>
#include <donnees_borne.h>
#include "lecteurcarte.h"
#include "VoyantBouton.h"
#include "timer.h"

int main()
{
    lecteurcarte_initialiser();
}
```

```
voyant_initialiser();
timer_initialiser();
prise_initialiser();

while (1)
{
    lecteurcarte_lire_carte();
}

return 0;
}
```

## 4. Conclusion

Ce mini projet a constitué une opportunité inestimable pour mettre en pratique l'ensemble des connaissances acquises au cours de la conception des systèmes.

L'utilisation de l'UML, en particulier pour les diagrammes de séquence et de collaboration, en tant que langage de modélisation dédié à la conception de divers types de systèmes, a été un pilier essentiel de notre démarche. La phase d'implémentation a représenté le moment clé où nous avons consolidé nos compétences, en expérimentant activement lors des séances pratiques pour maîtriser les aspects techniques du projet, notamment ceux liés aux diagrammes de séquence et de collaboration.

L'étude de conception d'une borne de recharge pour véhicules électriques a souligné l'importance cruciale des diagrammes de séquence et de collaboration dans l'approche UML. Ces outils n'ont pas seulement contribué à une gestion rigoureuse de ce mini projet, mais ont également grandement facilité la compréhension du système, évitant ainsi une immersion excessive dans des détails techniques trop spécialisés.

Cette expérience enrichissante a renforcé notre compréhension de la réalisation de projets informatiques complexes, mettant particulièrement en lumière l'efficacité des diagrammes de séquence et de collaboration de l'approche UML dans la communication entre les différentes parties prenantes du projet.

## 5. Annexe

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

#### 1 Cahier des charges

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

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

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

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

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

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

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

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

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

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

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

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

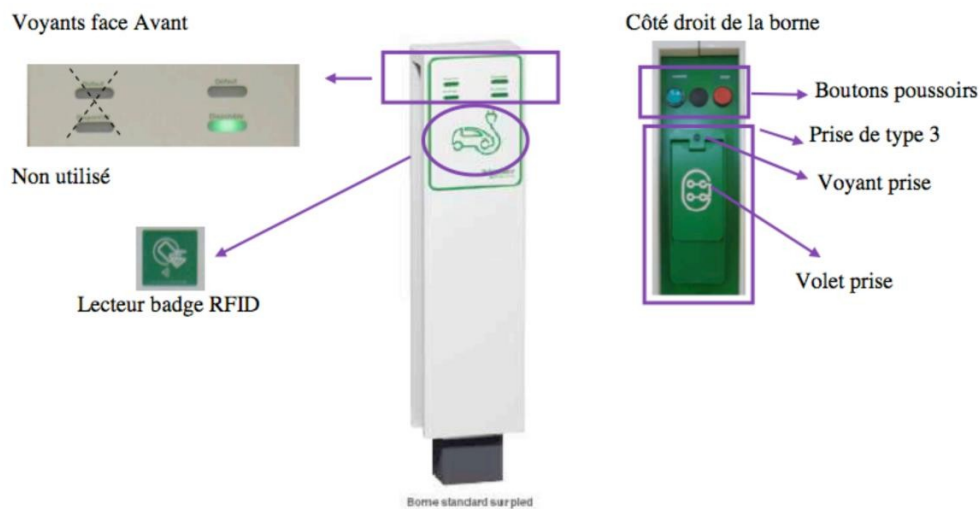
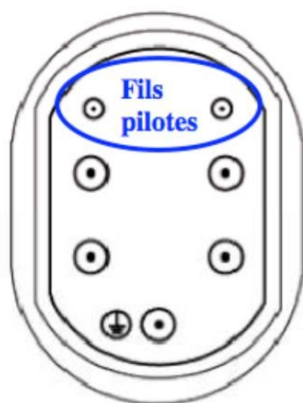
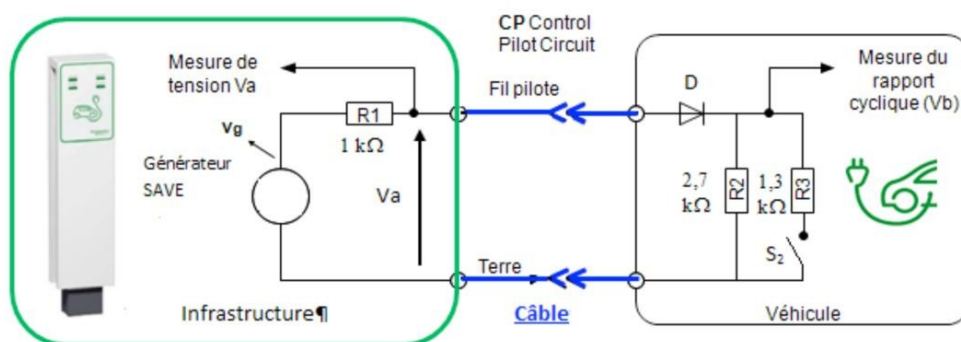


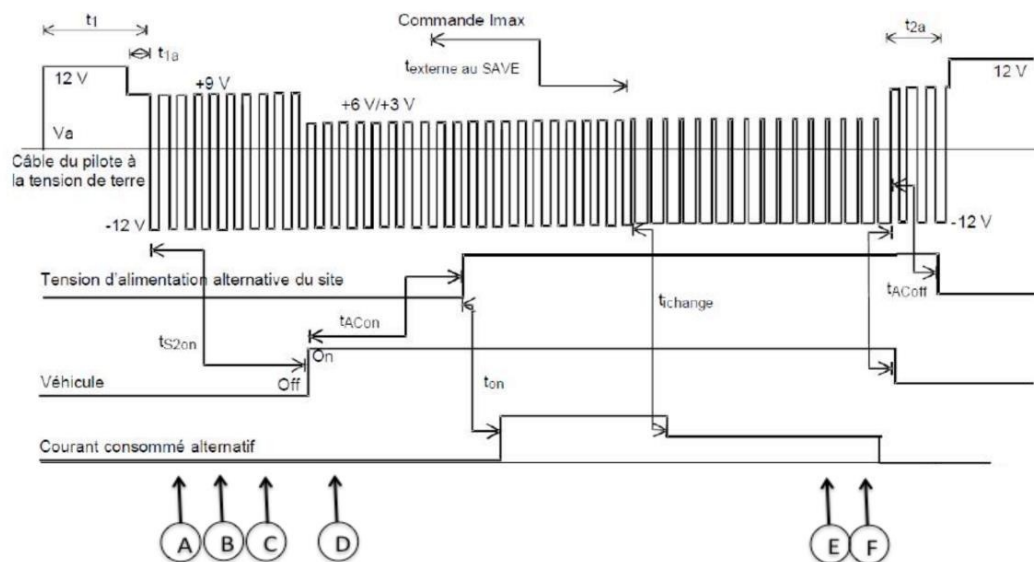
Figure 1: Présentation de la borne de recharge étudiée



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



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



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