# Requirement Analysis and Specification Document for PowerEnJoy

Enrico Migliorini, Alessandro Paglialonga, Simone Perriello

December 30, 2016

# Contents

# 1 Introduction

## 1.1 Purpose

This paper represents the **R**equirement **A**nalysis and **S**pecification **D**ocument of the *System Under Development*, which will implement the ***PowerEnJoy Car-Sharing*** Service. This document aims at explaining the functionalities of the System in terms of Functional Requirements, NonFunctional Requirements and Special Requirements, represented using both diagrams and natural language.

The above is a comprehensive list of functionalities provided by the System, that actually translates to a list of goals that the system should reach.

- G1 The System should allow the registration of the Visitors with their credentials and payment informations.
- G2 The System should allow all Users to use all the functionalities reserved to them.
- G3 The System should be able to give each User the list of all the available cars in a range of 5KM from their GPS position or a specific address.
- G4 The System should allow each of its Users to reserve a Car whose state is Available.
- G5 If an User has reserved a Car and they did not unlock it within 1 hour from the reservation, the System sets the Car state as Available, the reservation expires and the User pays a fixed Fee of 1 EUR.
- G6 The System should allow each User to unlock a previously reserved Car when they are in a distance range of 15 meters from the same Car.
- G7 The System should allow Users to drive a Car which they have previously unlocked.
- G8 The System should be able to know the time usage of the Car, measured in minutes and rounded up.
- G9 The System should allow Users to know where are located the Parking and Charging Areas.

G10 The System should allow each User to end the ride in a Parking or Charging Area.

G11 If the System detects the User took at least two other passengers onto the Car for at least 3 minutes, the system applies a discount of 10% on the last ride.

G12 If a Car is left with no more than 50% of the battery empty, the System applies a discount of 20% on the last ride.

G13 If a Car is left in a Charging Area and the User takes care of plugging the Car into the power grid, the System applies a discount of 30% on the last ride.

G14 If a Car is left at more than 3 KM from the nearest Charging Area or with more than 80% of the Battery empty, the system charges 30% more on the last ride to compensate for the cost required to recharge the car on-site.

G15 If the User enables the money saving option, he/she can input his/her final destination and the System provides the address of the Charging Area where to leave the Car in order to get a Discount on the total Fee. The Charging Area is determined by the System to ensure a uniform distribution of Cars in the city and depends both on the destination of the User and on the availability of Sockets at the selected Charging Area.

## 1.2 Intended Audience

This document is addressed to all the stakeholders involved in the ***PowerEnJoy*** project. This includes, but it is not limited to, the development committee, product designers and engineers, quality assurance, who will decide if the requirements described in this document have met the intended system requirements.

## 1.3 Product Scope

The aim of the ***PowerEnJoy*** project is to provide a *Car-Sharing* Service which implements electric-powered cars only. This system will have to interface the Cars, Charging Areas, allowing Users to reserve, unlock, drive and park Cars, finally charging them the cost of the ride. The System will keep track of Cars' position, battery level, possible damages, plugging state.

An useful approach we have used in this phase is based on the distinction between world and machine requirements, as proposed by M. Jackson and P. Zave.

In this approach, the machine represents the system to be developed, while the world is the environment in which the machine operates. The System under development will define the machine, but has no influence on the world.

There is also a shared set of phenomena that specify, at a high level, the requirements of our System.

The analysis led to the image represented in Figure 1.3.

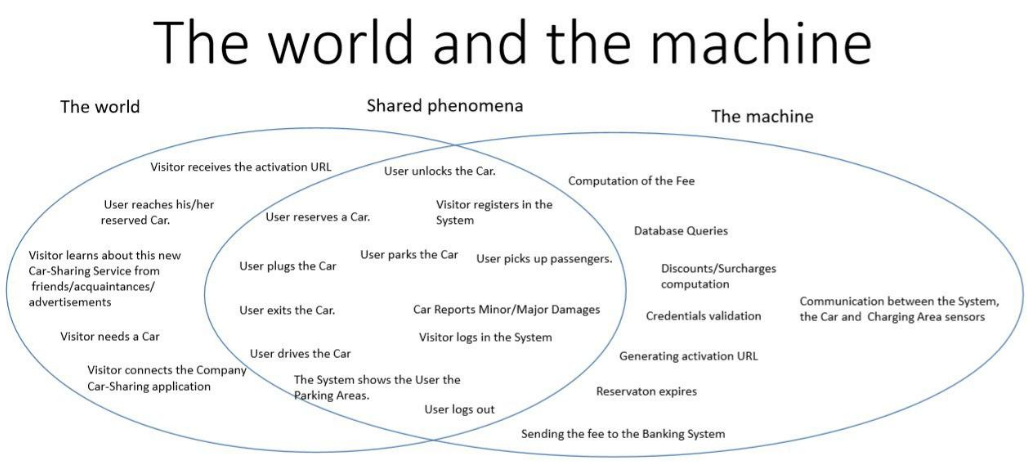


Figure 1: The World And The Machine

## 1.4 Definitions, Acronyms and Abbreviations

### 1.4.1 Business terms glossary

#### 1.4.1.1 User related terms

- ***Account***
  An Account is a virtual representation of a User in the System. The System can read and store information about a User.

- ***Device*** It is the piece of hardware used by the User to run the Application. There is no strict requirements on the kind of device an User can have, except that to use most of the System functionalities it should have a working GPS module.

- ***User***
A person registered on the System, who has access to the System functionalities. In this System there is no distinction between normal and privileged users, i.e. there are no special roles defined for particular users, because there is no need for an interface between the database and the administrators. External operations required by the Company (e.g. an Employee wants to set a machine as unavailable after a Minor Damage) are done directly by the Company, accessing the Database directly.

- ***Visitor***
A person who needs to register in order to use the System functionalities.

#### 1.4.1.2 Car related terms

- ***Battery***
A Battery powers a Vehicle. The charge state of the Battery can be anywhere between 0% and 100%, is reduced when the Vehicle is In Use, and increases when the Vehicle is Plugged to a Charging Area.

- ***Car***
An electric car owned by the Car-sharing service, rented to the User and tracked by the System. A Car can be in one of the following states:

  - *In Use*, if the engine is turned on. In this state, it cannot be Reserved by an User.
  - *Available*, if it can be Reserved by an User.
  - *Reserved*, if an User has reserved it but has still not unlocked it.
  - *Unavailable* if it can't be *Reserved* by any User (for example due to damage, battery exhaustion, maintenance, ...)

  Additionally, the *Plugged* flag indicates if the Car is plugged or not to a socket of a Charging Area. The Car has a set of sensors used to communicate to the System its position, the status of its battery, its damages, the connection to an electrical socket and the number of seats occupied. It also has a display to show various information to the User. In the end, among the actuators, it is worth mentioning those in charge of the remote unlock of the Car.

- ***Car Board***
  It is a piece of hardware and software that has to be installed on each Car to enable the communication between the Car and the Backend. It is responsible to fetch the data of the sensors installed on the Car and to send the Current Fee to the Car display. It is part of the System under development.

- ***Damage***
  The Car are able to detect damages, including their entity. In case of damage, the PowerEnjoy board will notify the System. A Car can detect two kind of damages:

  - *Major*, a serious damage that prevent the normal usage of the Car (e.g. a damage of a mechanical component). Note that in this case the Car will immediately notify the System and the Car will be set as Unavailable.
  - *Minor*, a damage that do not prevent the normal usage of the Car (e.g. a scratch or a dent on the car body). Note that in this case, if the Damage is detected during a Ride, the Car will be set as Unavailable only at the end of the Ride.

  In both cases the System will also communicate the damage to the Company. It is up to the Company the decision on how to proceed, but it is meaningful to note that there is no mechanism that will revert the state of the Car as Available, so the suggestion is to dispatch an Employee to the Car in order to evaluate the damages and the best action to take.

- ***Passenger***
  Any person who travels in a Car, including the driver.

- ***Plug***
  A part of the Car that can be inserted in a Socket of a Charging Area.

- ***Reservation***
  A User performs a Reservation in order to book an Available Car. There are two main constraints on a Reservation: 1) an User can only have one active Reservation at a time and 2) a Reservation expires after a given amount of time (one hour), at the end of which the User will be charged with a Fee of 1E.

- ***Ride***
  Represents the travel done with the Car by the User. It starts from the moment the User ignites the engine of the Car and ends when the Car is parked in a Parking Area,the User and all the other passengers exit the Car.

#### 1.4.1.3 Area related terms

- ***Charging Area***
  A special Parking Area where Cars plugs can be connected to the socket in order to recharge their Battery.

- ***Parking Area***
  A place where the User can leave their Car and exit it to end the Ride. Parking Areas are predefined by the System.

- ***Socket***
  A part of the Charging Area that can be connected with the Plug of a Car.

#### 1.4.1.4 Banking related terms

- ***Current Fee***
  This fee is related to a Ride and is evaluated as

  $$\textit{Time usage of the Car} \cdot \textit{Fee per minute}$$

  where *Time usage of the Car* is the time interval between the start and the end of the ride (rounded up to minutes) and *Fee per minute* indicates the rate decided by the Company for every minute of ride.

- ***Discount***
  A reduction in the Fee because of good behaviour on the part of the

User, e.g. leaving the Cars plugged or bringing it back with a mostly-full battery. The actions that constitute good behaviour are determined ad detailed further in the document.

- ***Fee***
  The final amount of money that the Users will be charged for their usage of the Car-sharing service, or for making a Reservation that is not fulfilled.

- ***Surcharge***
  An increase in the Fee caused by an improper behaviour on the part of the User, e.g. bringing the Cars back with a mostly-empty battery.

#### 1.4.1.5 External systems

- ***Banking System*** An external system that allows the System to charge the users for a Fee.

- ***Mailing System*** An external system that allows to send emails to Visitors and Users.

- ***Mapping System*** An external system that is designed to capture, store, manipulate, analyze, manage, and present spatial or geographical data. It is used in particular to show the GPS position of Cars, Users and Parking Areas on a map, check for existing addresses, and get the exact desired position in a specified address.

#### 1.4.1.6 System terms

- ***Application***
  It is the part of the System which acts as an interface between the User and the Backend and allows the User to access the System functionalities.

- ***Backend***
  It is the part of the System which collects input from users for processing.

- ***Company Database***
  An infrastructure provided by the Company on which the (System) Database will run.

- ***Database***
  A structure that holds all the information used by the System. For instance, a Database could hold records of every User, Car, every time a User rented a Car and so on. The Database infrastructure will be provided by the Company, which in turn will grant us (i.e. the System) the privileges to create our own Database. The Company will be able to access our Database at any time.

- ***Developer*** Every person concerned with facets of the software development process of this System, including the research, design, programming, and testing.

- ***System***
  The software structure this document is about. At a very high level, it is composed by the Application, the Backend, the Database and the Car Board.

#### 1.4.1.7 Other terms

- ***Car-sharing***
  A Car-sharing service allows Users to rent Cars for a limited amount of time, being charged a Fee according to time and possibly applying a Discount or a Surcharge.

- ***Company***
  The enterprise that wants to build the System to provide a *Car-Sharing* Service. It represents the main stakeholder.

- ***Employee***
  A person paid by the Company which is in charge of every kind of Car maintenance (e.g. charging the Car battery on-site, moving a Car to a Charging Area and so on). The employees and their tasks are managed directed by the Company and this System does not offer any functionality used to help their management.

- ***GPS***
  Global Positioning System, it is widely used by the Application to get Users and Cars position. The GPS position of Areas, on the other hand, is predefined and will be retrieved directly from the ***Database***.

### 1.4.2 Document specific terms

- ***Alloy*** A descriptive language that allows to describe a set of structures through constraints.
- ***Internet Protocol Suite*** The set of communications protocols used on the Internet. It is commonly known as **TCP/IP**.
- ***HTTP(S)*** One of the application protocols of the Internet Protocol Suite, widely used by our System.
- ***RASD*** Requirements Analysis and Specification Document. This document, describing the ***System*** to be developed.
- ***UC*** Use Case. A description of interaction between ***User***s and ***System***.
- ***UML*** Unified Modeling Language. A language for modeling Object-Oriented software systems.

# 2 General Description

This section will give a broad overview of the whole System under development. It will explain how the System interacts with external systems and introduce its main functionality.

It will also describe the end users and the functionalities of the System reserved to them, detailing all the informations relevant to clarify their needs.

At last it will present the constraints and assumptions made for the System under development.

## 2.1 Product Perspective

This System will be implemented ex-novo to support all the functionalities required by the *PowerEnJoy Car-Sharing Service*. It will be vastly built on top of the TCP/IP protocol, using a mix of HTTP and HTTP(S) as the main application protocols.

The System, for example, has to communicate with all the required external systems, such as the Banking System, needed to perform the monetary transactions, the Mailing System, which will forward the emails generated by the System, and the Mapping System, which is used in particular to show the GPS position of Cars, Users and Parking Areas on a map, check for existing addresses, and get the exact desired position in a specified address. In order to reach the objective, the System must use all the set of shared protocols and APIs.

Furthermore, the TCP/IP stack will also be used to communicate with the Cars and the different kind of Areas, in order to get all the meaningful information. The System, for example, should be able to have access to all the informations provided by the wide variety of sensors placed inside the Cars, in order to know, in every moment, their GPS position, the status of their battery, their possible damages, their plugged status and the number of seats occupied. On the other hand, the System should also access data coming from a Charging or Parking Area (f.e. the number of free sockets and/or the number of free parking slots).

The interaction among the Users and the System will also be built using the Internet Protocol Suite. Users will access the System functionalities using the Application, which is the interface between the User and the back end of the System. The Application will use the HTTP(S) and TCP/IP protocols to allow the communication.

At last, Users should also be able to communicate their position to the System using their GPS coordinates. For this reason, the Application need access to the User GPS module, which is required in order to unlock previously reserved cars.

### 2.1.1 Class Diagram

The structure explained in the previous section is reported here using a Class Diagram.

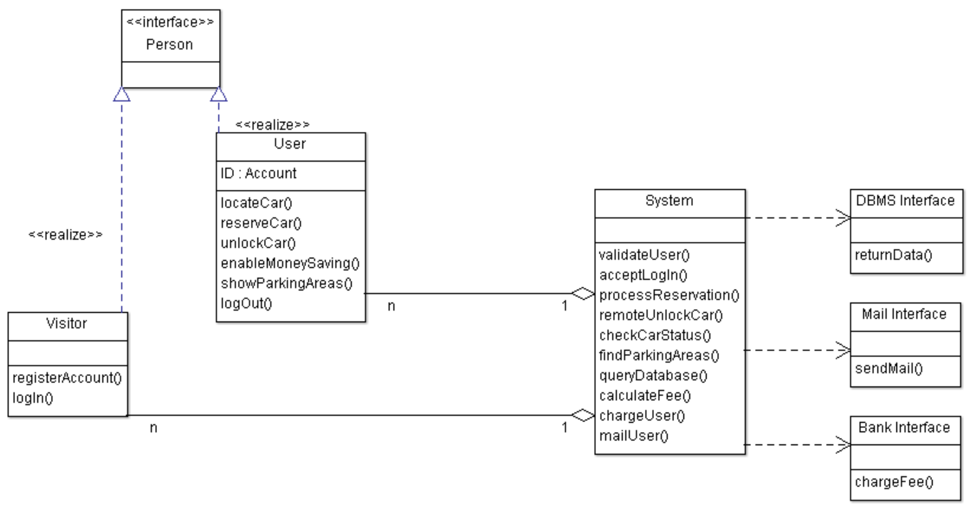


Figure 2: Class Diagram

### 2.1.2 Statechart Diagram

Here we have an UML Statechart diagram explaining the various states in which a Car can be.

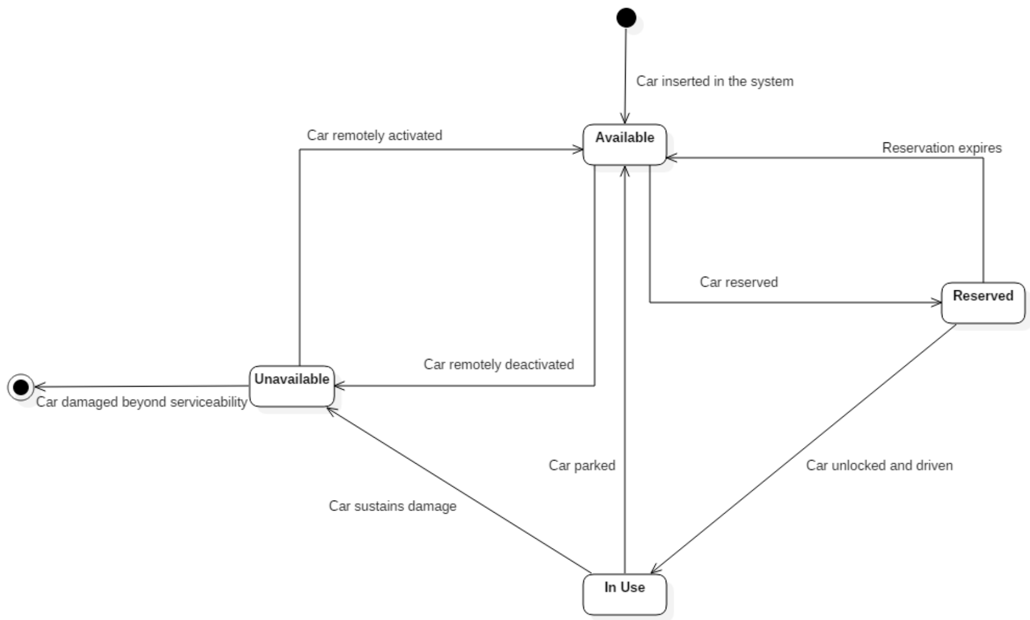


Figure 3: Statechart diagram of Car

## 2.2 Product Functions

Using the Application, the User will be able to register to the System. After a successful registration, the System will create an Account related to the User, which can use it to log in to the System and access the System functionalities.

The User can now use the Application to locate all the Cars, whose state is *Available*, in two different ways: a) specifying a desired address or b) asking the System to locate him/her through the GPS coordinates. In the first case, the Mapping System is called to check for the existence of the specified address and to return a GPS position corresponding to that address. In both cases the GPS position will be sent to the Backend, which will return all the *Available* Cars in a predefined distance range from the previously sent GPS position. The Mapping System will be called again by the Application to show to the User a map with all the Cars.

After locating the *Available* Cars, the User can pick one of them and reserve it; at this moment, a one hour countdown starts during which the User will be able to unlock his/her reserved Car. If the User doesn't unlock the Car during the previous specified time period, the System cancels the reservation, sets the Car status as *Available* and communicates to the Banking System the application of a fee.

The User can unlock the Car asking the System to locate him/her and if the Backend verifies that the User is in a specified distance range from the Car, the Car is unlocked and the User can now enter and drive it. During the drive, the System will display the Current Fee on the screen of the Car.

At the end of the ride, the System, basing on the time usage period of the Car, and on a set of bad or good behaviours, will evaluate the Fee and notify and the Banking System of the total amount to charge to the User's credit card.

## 2.3 User Classes and Characteristics

| **Name** | **Description** | **Actions** |
|---|---|---|
| Visitor | A person who would like to register an account to access the System functionalities. | Can perform the registration and activation of the account. Successively he can log in the System becoming a User. |
| User | Someone who is registered in the System and can access its functionalities | Can locate, reserve and drive Cars. Will be charged for the use of the Cars. |

External access to the Database provided to Employees or any other person is under the responsibility and the regulations of the Company and is not managed by this System.

## 2.4 Constraints

### 2.4.1 Hardware Constraints

The Device used by the User should be able to establish an internet connection to the System using the Internet Protocol Suite. In addition, in order to perform the localization, reservation and unlocking of a Car the User device must have installed a working GPS module.

The set of sensors, actuators and display inside Cars and Areas are already installed by the Company and the System has to communicate with them. Check the assumptions to see all the kind of hardware already shipped with Cars and Areas. On the other hand, the System will have a Car Board installed on each Car which will be used to enable the communication between the Car and the Backend. The aim is to avoid the constant polling of the Car sensors to get the data, using instead a special board to send the data only on specific events.

### 2.4.2 Design and Implementation Constraints

The System will employ the Internet Protocol Suite in order to enable the communication between its various parts, namely Application, Backend, Database and Car Board. The communication must be secured using at least the HTTP(S) application protocol. The communication with the external systems will be built on top of the same set of protocols, accessing the API of this External Systems.

## 2.5 Assumptions and Dependencies

1. The User do not create more than one Account at a time. Every User is accountable for improper use of their Account.
2. Users will never forget their password.
3. The User always provides real correct data in his/her registration form.
4. The Company can decide at any time to revoke an User access to the System, charge the User with a fine or apply any other policy it choose (f.e. due to improper behaviour, unpaid bill, car parked outside a safe area, ...).
5. The User which has reserved and then unlocked the Car will always be the person driving it. It will always be accounted for the usage of the Car and any improper action taken by the Driver.
6. The Database infrastructure in which the Cars, Parking Areas, Charging Areas, Users, etc, are stored, is owned and managed by the Company (and not by this System), which is responsible for its security, reliability and availability. Our System is provided by the Company with read/write access to the Database.
7. The Company is responsible for the Employees and all their actions.

8. The Car has a set of sensors that can detect, in every moment, its position, the status of its battery, the status of the engine, its damages, the connection of its plug to an electrical socket and the number of seats occupied. It also has actuators to remotely unlock its doors and a display to show information to the User. We assume that these devices are always perfectly functioning and their measures are always correct. In the case of damages to this devices, their repair is under the responsibility of the Company.

9. After the doors of the cars are unlocked by the User, he/she always enters the Car, ignites the engine and leave the Parking Area.

10. After a Car is Plugged to the Socket of a Charging Area, it will not be maliciously unplugged by the User himself/herself or by other people.

11. An User can park/stop the Car everywhere and leave the Car at anytime. However, the system will not apply any fine only if he/she ends the ride inside a Parking or Charging Area. In all other cases the Company can decide to charge the User with a fine.

12. If the Car has been left out of a Parking Area there will always be an Employee which immediately reaches it, recharges it and move it to a Charging Area.

13. When a User will park the Car inside a Parking or Charging Area, it will always correctly use one and only one free slot.

14. As soon as the Car battery status gets below 20% of the full capacity AND the Car isn't in a Charging Area AND the Car Status isn't *In Use* OR *Plugged*, there's always an Employee that immediately reaches the Car and recharges it on site; in the meanwhile the Car status is *Unavailable*.

15. When the Car is *In Use* and the battery charge level reaches the 0% of the full capacity the Car stops working.

16. The Car will always be able to communicate a Major Damage to the System.

17. If the Car status is *Unavailable*, the Car will always be reached by an Employee to consider if the Car needs to be repaired, if an User should be charged for the damages or if the Car just needs to be recharged.

18. A car which is *Available* or *Plugged* can be set as *Unavailable* in every moment by an Employee. This is done through another Company's System as it is not provided in this System.

19. A car which is *Unavailable* can be set to *Available* in every moment by an Employee. This is done through another Company's System and it is not provided in this System.

20. Every fine received by the Company for improper use of the Car will be managed by the Company.

21. Parking and Charging Areas have sensors to detect the number of parking and charging slots occupied at any time.

# 3 Specific Requirements

This sections contains all the system interfaces and identifies the functionality of the software to accomplish the system requirements.

## 3.1 External Interface Requirements

This system provides a detailed description of all inputs into and outputs from the system. It also gives a general description of the hardware, software and communication interfaces.

### 3.1.1 Hardware Interface

Reliable software device drivers shall be provided for the Car Boards used by the System. They should be completely tested to prove the full access to the Car prebuilt devices and the correct communication with the Backend. The Areas information coming from the Areas sensors should also be reachable from the System in order to get the number of free parking and/or charging slots. The communication with this sensors should be completely tested to prove the full access of the Backend to this data. The Application should be able to connect to the GPS module of the Device. It should be completely tested to prove the full access to this module.

### 3.1.2 Software Interface

The Application has to communicate with the Mapping System in order to display maps and to get a GPS position given a street address.

The Backend has to communicate with the following systems:

- Mailing System. It has to provide an API to send send mails to the Users.
- Banking System. It has to provide an API to charge credit cards of a specific amount of money.
- Company. It has to provide an API to send notifications and messages.
- Cars. They have to provide full access to the set of sensors needed by the System.

- Company Database. It has to provide a way to reach the System Database hosted on the Company Database infrastructure.

### 3.1.3 User Interface

The Application is the only interface between a Visitor or an User and the System.

A generic Visitor of the Application should only see the registration and login forms. If the Visitor has not registered yet, he/she should be able to do it through the registration form. On the other hand, if the Visitor is already registered, he/she should be able to log in through the login form.

Whenever a Visitor logs in into the System through the Application, he/she becomes a User and can access all the functionalities of the System reserved to him/her.

Every User use the Application to :

- communicate to the System his/her GPS position and visualize it on a map
- locate all the Cars Available in a range of 5 km from his/her GPS position or from a given address and visualize them on a map
- reserve a Car among the one previously displayed
- unlock a Car when he/she is nearby the previously reserved Car
- know if a Car is plugged to a socket of a Charging Area
- know the Battery status of a Car
- know the GPS position of Charging Area and Parking Area and visualize it on a map
- select the *Money saving option*

The Application also communicate to the User short error messages.

### 3.1.4 Communication Interface

As already mentioned, the System heavily uses Internet Protocol Suite protocols, mainly the HTTP(S) protocol, in order to communicate with all its components.

## 3.2 Functional Requirements

The functionality for the various users.

### 3.2.1 Requirements List

**R1** The System shall provide Users with the ability to access all the System functionalities reserved to them.

**R2** The System shall support Users in locating Available Cars within a range of 5 Km from a specific position.

**R3** The System shall support Users in locating Parking Areas and their free parking slots.

**R4** The System shall support Users in locating Charging Areas and their free parking slots and free charging sockets.

**R5** The System shall support Users in reserve Available Cars.

**R6** The System shall apply a fixed Surcharge of 1 EUR if he/she has reserved a Car and not unlocked it within a time range of 60 minutes.

**R7** The System shall support a User in unlock a Car he/she has previously reserved when he/she is in a range of 15 meters from the same Car.

**R8** The System shall charge the User of a fixed fee per minutes, communicating to him/her the Fee he will get charged at the end of the ride basing only on the driving time and the fee per minutes.

**R9** The System shall be able to know if a User has took in the Car he/she is driving at least two other passengers for at least 3 minutes. If so, The System should apply a percentage Discount of 10% on the final Fee of the last ride.

**R10** The System shall allow the User to end the ride in a Parking or Charging Area.

**R11** The System shall allow any User who has ended a ride to plug the Car he/she has driven to a Socket in a time rage of 2 minutes since he/she has ended the ride, in order to get a percentage Discount of 30% on the final Fee of the last ride.

**R12** The System shall apply a percentage Discount of 20% on the final Fee of the last ride if the User will end the ride leaving the Car with more than 50% battery charge status.

**R13** The System shall apply a percentage Surcharge of 30% on the final Fee of the last ride if a User leaves the Car at more than 3 km from the nearest Charging Area or with a battery charge status less than 20%.

**R14** The System shall provide a User the ability to use the "Money Saving Option", telling him/her the position of a Charging Area where he/she has to park the Car he/she is driving in order to get a Discount on the total Fee. The Charging Area is determined by the System to ensure a uniform distribution of Cars in the city of that address and depends both on the destination of the User and on the availability of Sockets at the selected Charging Area.

**R15** When the Car status is *In Use* and a *Minor damage* is detected, the Car status will be set to *Unavailable* by the System at the end of the ride.

**R16** When a *Major damage* is detected the Car status is immediately set to *Unavailable* by the System.

**R17** When the Car is *In Use* and the battery charge level reaches the 0% of the full capacity the Car is immediately set as *Unavailable* by the System.

**R18** The System shall interface with an external Mailing System to send emails to Users.

**R19** The System shall interface with an external Banking System to charge Fee to Users.

**R20** The System shall interface with an external GPS System to know the positions of Users and Cars.

**R21** The System shall interface with an external Mapping System to show the positions of Users, Cars and Areas on a map and to retrieve the GPS position corresponding to a given address.

**R22** The System shall interface with the existing Car to get their GPS position, damages, connection to an electrical socket, the number of seats occupied. In order to fullfil this requirement, a Car Board will be installed on every Car.

#### 3.2.2 Use cases specification

**Register Account**

| | |
|---|---|
| **ID** | UC1 |
| **Description** | The ***Visitor*** wants to create an ***Account*** for the ***Car-Sharing*** Service. |
| **Actors** | ***Visitor***. |
| **Pre-Conditions** | The ***Visitor*** opens the ***Application***. |
| **Flow of events** | 1. The ***Visitor*** selects the function *"Sign Up"*.<br>2. The ***System*** returns a form to enter all the required data: Name, Surname, Birth date, ID Card Number, Driving License number and Credit Card number. It also asks for an email address and a password which will be used for the future logins.<br>3. The ***Visitor*** fills the form with all the required information.<br>4. The ***System*** stores the request together with all the data provided with it, generates a random activation URL and asks the ***Mailing System*** to forward his/her URL to the email address of the ***Visitor***. |
| **Post Conditions** | The ***Mailing System*** sends the activation URL to the ***Visitor***'s email provided in the registration form. |

| | |
|---|---|
| **Exceptions** | • The ***System*** recognizes invalid or missing data in the form compiled by the emphVisitorand informs him/her of the error. The flow of events restarts from point 1.<br>• The Visitor inserts in the form an ID Card Number, or Driving License number, or Email Address, which is already present in the System. The System shows an error message explaining the reason of the error. The flow of events restarts from point 1. |

**Activate Account**

| | |
|---|---|
| **ID** | UC2 |
| **Description** | The ***Visitor*** wants to activate his/her ***Account***. |
| **Actors** | ***Visitor***. |
| **Pre-Conditions** | The ***Visitor*** has received the activation URL on his/her mail box. |
| **Flow of events** | 1. The ***Visitor*** opens the received activation URL.<br>2. The ***System*** activates the account and updates the data of the Visitor to reflect this change. |
| **Post Conditions** | The ***Visitor*** is now become an ***User*** which can access the ***System*** using the pair (email, password) provided during the registration phase. |
| **Exceptions** | • The Activation URL expires after 1 day it has been generated. The Visitor's data are cancelled from the System and the Visitor will have to perform the Registration (UC1) again. |

### Log In

| | |
|---|---|
| **ID** | UC3 |
| **Description** | The ***Visitor*** wants to log in the ***System***. |
| **Actors** | ***Visitor***. |
| **Pre-Conditions** | The ***Visitor*** opens the ***Application***. The ***Visitor*** has already activated his/her account (UC2) |
| **Flow of events** | 1. The ***Visitor*** selects the function *"Login"*.<br>2. The ***System*** shows the ***Visitor*** a login form, asking him to insert the email and password provided in the registration form.<br>3. The ***Visitor*** inserts the pair (email, password) used during the registration phase and selects the function *"Log me in"* |
| **Post Conditions** | The ***System*** verifies the existence of an account associated with the pair (email, password) and logs the ***Visitor*** in. The ***Visitor*** has now become ***User*** |
| **Exceptions** | • The System does not find a valid account associated with the pair (email, password) and shows an error message. The flow of events restarts from point 1. |

### Log Out

| | |
|---|---|
| **ID** | UC4 |
| **Description** | The User wants to log out from the System. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** is logged in the ***System*** |

| | |
|---|---|
| **Flow of events** | 1. The ***User*** selects the function *"Log out"* .<br>2. The ***System*** performs the ***User***'s logout. |
| **Post Conditions** | The ***System*** shows the confirmation of the logout to the ***User***.<br>The ***User*** is now not able to use the ***System*** functionalities dedicated to Users anymore (until he logs in again). |
| **Exceptions** | |

**Show Parking Areas**

| | |
|---|---|
| **ID** | UC5 |
| **Description** | The ***User*** wants to see the ***Parking Areas*** where he can possibly leave the ***Car***. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** is logged in the ***System*** |
| **Flow of events** | 1. The ***User*** selects the function *"Show Parking Areas"*.<br>2. The ***System*** shows the ***User*** a map with all the ***Parking Areas*** distributed around the city. |
| **Post Conditions** | The User is now able to see the distribution of the Areas and the information associated to each Area. |
| **Exceptions** | |

**Locate Available Cars**

| | |
|---|---|
| **ID** | UC6 |
| **Description** | The ***User*** wants to locate the available ***Cars***. |
| **Actors** | ***User***. |

| | |
|---|---|
| **Pre-Conditions** | The ***User*** is logged in the ***System*** |
| **Flow of events** | 1. The ***User*** selects the function *"Locate Cars"*.<br>2. The ***System*** shows a text box asking the ***User*** to provide an address.<br>3. The ***User*** inserts the desired address and selects the *"Locate"* function.<br>4. The ***System*** send the address to the ***Mapping System***.<br>5. The ***Mapping System*** returns the GPS position corresponding to that address.<br>6. The ***System*** shows the ***User*** a map containing all the ***Cars*** whose state is *Available* and which are within a 5 km range from the provided address. |
| **Post Conditions** | The User is now able to see the distribution of the Cars and the information associated to each Car. At this point, he/she can also decide to reserve one of the previously retrieved Cars (see UC7). |
| **Alternative Flow of Events** | The ***User*** selects the function *"Near Me"* instead of Step 1 and sends their ***GPS Coordinates*** to the ***System***. The flow continues at Step 6. |
| **Exceptions** | • The System does not find the inserted address and informs the User. The Flow of Events starts from point 1.<br>• There are no available Cars in the specified address/User's Position. The System informs the User. The Flow of Events start from point 1. |

**Reserve Available Car**

| | |
|---|---|
| **ID** | UC7 |
| **Description** | The ***User*** wants to reserve a ***Car***. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** is logged in the ***System***, the ***User*** does not have another active reservation, the ***User*** is not driving another ***Car***, and the System has found available ***Cars*** when the ***User*** activated the *"Locate Available Cars"* function. |
| **Flow of events** | 1. The ***User*** chooses a specific ***Car*** among those showed on the map.<br>2. The ***User*** selects the function *"Reserve this Car"*.<br>3. The ***System*** stores the ***Reservation*** of the ***Car***, changing its status to ***Reserved***. |
| **Post Conditions** | The ***System*** activates a countdown of 1 hour during which the ***User*** will have the possibility to unlock the ***Reserved Car***. |
| **Exceptions** | If one hour has passed since the reservation has been made and the ***User*** has not unlocked the Car then:<br>• The reservation expires, so that the ***User*** cannot unlock the ***Car*** anymore (unless he/she reserve it again).<br>• The ***System*** changes the ***Car***'s status to ***Available***.<br>• The ***System*** communicates to the ***Banking System*** the ***Fee*** to charge the ***User*** (this sum amounts to 1 EUR).<br>• The ***System*** asks the ***Mail System*** to forward the reservation details to the ***User*** email address.<br>• The ***System*** allows the ***User*** to perform another reservation. |

**Unlock Car**

| | |
|---|---|
| **ID** | UC8 |
| **Description** | The ***User*** wants the ***System*** to open the doors of the ***Car*** in order to enter it. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** is logged in the ***System*** and has reserved a ***Car***. |
| **Flow of events** | 1. The ***User*** activates the function *"Unlock Car"*.<br>2. The ***User*** sends his/her GPS coordinates to the ***System***.<br>3. The ***System*** checks that the GPS coordinates of the ***User*** are within a 15 metres range from those of the ***Car*** itself and, if so, unlock the doors of the Car and changes the ***Car*** status to ***In Use***. |
| **Post Conditions** | The ***User*** is now able to enter the ***Car***. |
| **Exceptions** | If the GPS coordinates of the ***User*** are not within a 15 meters range from the ***Car***, the ***System*** shows an error message. |

**Drive Car**

| | |
|---|---|
| **ID** | UC9 |
| **Description** | The ***User*** starts driving the ***Reserved Car***. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** has unlocked the doors of the ***Car*** and entered it. |

| | |
|---|---|
| **Flow of events** | 1. The ***User*** starts the engine of the ***Car***.<br>2. The ***System*** starts a Ride Timer which indicates the time usage of the ***Car***.<br>3. [Extension Point UC 10]<br>4. [Extension Point UC 13] |
| **Post Conditions** | The ***User*** drives the ***Car*** and the ***System*** shows the ***Current Fee*** on the Car display. |
| **Exceptions** | |

**Drive With Passengers <<extends UC 9>>**

| | |
|---|---|
| **ID** | UC10 |
| **Description** | The ***User*** picks up ***Passengers*** to share the ride with. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** is driving the ***Car***. |
| **Flow of events** | 1. The ***User*** picks up other ***Passengers***.<br>2. The ***Car*** sensors detect the presence and number of the ***Passengers***.<br>3. The ***System*** stores the number of ***Passengers*** who were picked up and whether they stayed in the ***Car*** for at least 3 minutes. |
| **Post Conditions** | |
| **Exceptions** | |

### End Ride

| | |
|---|---|
| **ID** | UC11 |
| **Description** | The ***User*** ends the ride and the ***System*** processes the ***Fee***. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** parks the ***Car*** in a ***Parking Area***. |
| **Flow of events** | 1. The ***User*** exits the ***Car***.<br>2. The ***System*** verifies that no one is in the ***Car*** and that the engine is off. If both are true, the ***System*** locks the doors of the ***Car***.<br>3. The ***System*** stores all the information relevant to the computation of the ***Fee***, including the ***Battery*** status, the GPS coordinates of the Car, whether the ***User*** has left the ***Car*** within a 3 km distance range from the nearest ***Charging Area***, whether the ***User*** drove with ***Passengers*** (UC10).<br>4. [Extension Point UC12]. |
| **Post Conditions** | The ***System*** will wait to check if the ***User*** will plug the ***Car*** into a ***Socket*** of the ***Charging Area***. |
| **Exceptions** | If the ***Battery*** status reaches 0% of capacity or the ***Car*** detects a major damage, the ***Car*** stops and an assistance team is deployed. |

### Plug the Car <<extends UC 11>>

| | |
|---|---|
| **ID** | UC12 |
| **Description** | The ***User*** plugs the ***Car*** for recharging. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** has parked the ***Car*** in one of the ***Charging Areas*** designated by the ***System*** within 2 minutes from the end of the ride. |

| | |
|---|---|
| **Flow of events** | 1. The ***User*** plugs the ***Car*** into a ***Socket*** of the ***Charging Area***.<br>2. The ***System*** detects that the ***Car*** has been plugged within 2 minutes since the ride end and sets the ***Plugged*** flag of the ***Car*** to True. |
| **Post Conditions** | The ***Battery*** of the ***Car*** is charging and the ***System*** remembers the ***User***'s action for possible discounts.<br>The ***System*** communicates to the ***Banking System*** the final ***Fee*** to charge the ***User***.<br>The ***System*** asks the ***Mailing System*** to forward the drive details to the ***User*** email address. |

| | |
|---|---|
| **Alternative Flow of Events** | • The ***Battery*** status is higher than 50%, the ***User*** didn't or did take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), didn't leave the ***Car*** further than 3 km from the nearest ***Charging Area***, didn't plug the ***Car*** (UC12), hence the System applies a 20% ***Discount*** on the ***Fee*** of the last ride and communicates it to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.<br><br>• The ***User*** did plug the ***Car*** (UC12), the ***Battery*** status is higher than or equal to 20%, they didn't or did take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), hence the System applies a 30% ***Discount*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.<br><br>• The ***User*** did plug the ***Car*** (UC12), the ***Battery*** status is lower than 20%, they didn't or did take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), hence the System doesn't apply any ***Discount*** or ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.<br><br>• The ***User*** didn't plug the ***Car*** (UC12), the ***Battery*** status is higher than 50%, they either did or didn't take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), did leave the ***Car*** further than 3 km from the nearest ***Charging Area***, hence the System applies a 10% ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***. |

- The ***Battery*** status is between 20% and 50% (included), the ***User*** did take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), didn't leave the ***Car*** further than 3 km from the nearest ***Charging Area***, didn't plug the ***Car*** (UC12), hence the System applies a 10% ***Discount*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.

- The ***Battery*** status is lower than 20%, the ***User*** did take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), either did or didn't leave the ***Car*** further than 3 km from the nearest ***Charging Area***, didn't plug the ***Car*** (UC12), hence the System applies a 20% ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.

- The ***Battery*** status is between 20% and 50% (included), the ***User*** did take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), did leave the ***Car*** further than 3 km from the nearest ***Charging Area***, hence the System applies a 20% ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.

- The ***Battery*** status is lower than 20%, the ***User*** didn't take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), either did or didn't leave the ***Car*** further than 3 km from the nearest ***Charging Area***, didn't plug the ***Car*** (UC12), hence the System applies a 30% ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.

| | |
|---|---|
| | • The ***Battery*** status is higher than 50%, the ***User*** either did or didn't take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), did leave the ***Car*** further than 3 km from the nearest ***Charging Area***, didn't plug the ***Car*** (UC12), hence the System applies a 10% ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***.<br>• The ***Battery*** status is between 20% and 50% (included), the ***User*** didn't take at least 2 ***Passengers*** with him for at least 3 minutes (UC10), did leave the ***Car*** further than 3 km from the nearest ***Charging Area***, hence the System applies a 30% ***Surcharge*** on the ***Fee*** of the last ride and communicates to the ***Banking System*** the ***Fee*** which will be charged to the ***User***. |
| **Exceptions** | |

**Enable Money Saving Option <<extends UC 9>>**

| | |
|---|---|
| **ID** | UC13 |
| **Description** | The ***User*** asks the ***System*** to suggest them a ***Charging Area*** where to leave the ***Car***. |
| **Actors** | ***User***. |
| **Pre-Conditions** | The ***User*** enables the *"Money Saving"* option. |

| | |
|---|---|
| **Flow of events** | 1. The ***System*** asks the ***User*** the destination address, providing them with a text box where to insert it.<br>2. The ***User*** provides the address to the ***System***.<br>3. The ***System*** runs an algorithm which takes in consideration the distribution of the Cars in the city, the final destination of the ***User*** and the availability of power plugs in the ***Charging Areas***.<br>4. The ***System*** will present the result to the ***User***, displaying the address of the ***Charging Area***. |
| **Post Conditions** | The ***User*** can now decide if the proposed ***Charging Area*** is suitable for their needs. Note that, even if the ***User*** chooses this ***Charging Area***, he/she will still have to plug the ***Car*** into a Socket of this area in order to get a discount. |
| **Exceptions** | If the ***Socket*** of the ***Charging Area*** has no more available plugs while the ***User*** is driving to reach it, the ***System*** informs the ***User*** and the Flow of Events starts from point 1. |

### 3.2.3 Use Case Diagram

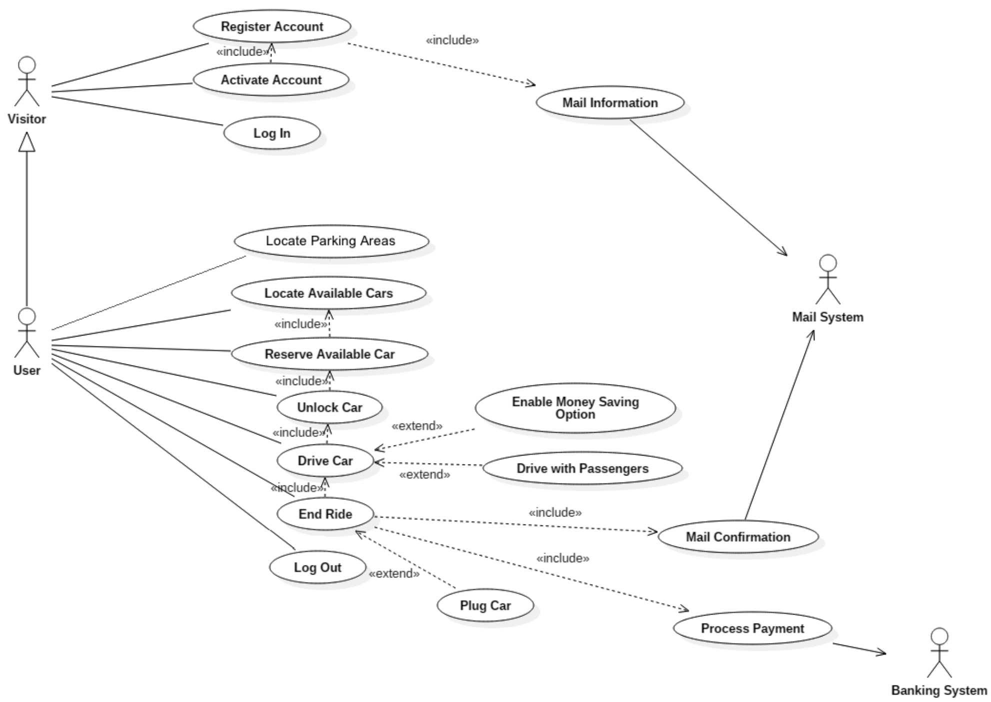


Figure 4: Use Case

### 3.2.4 Activity Diagrams

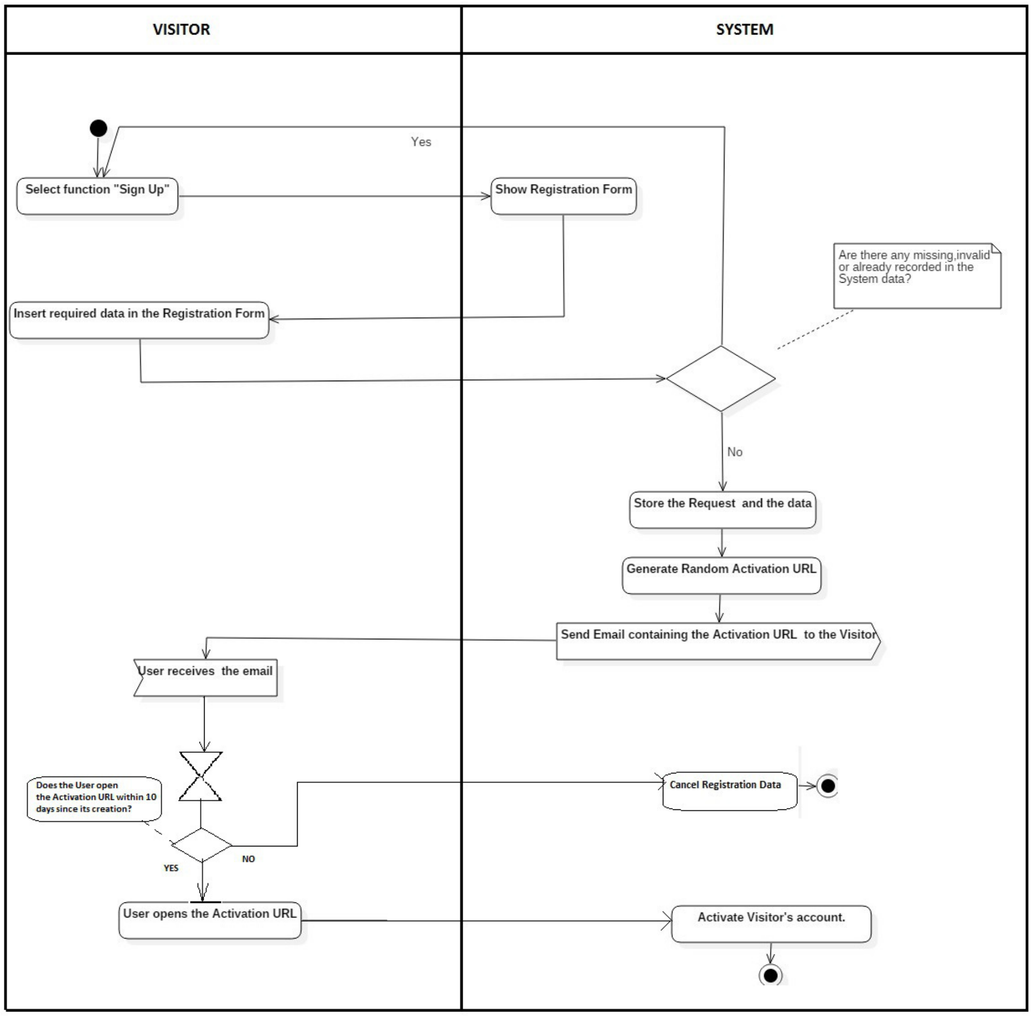


Figure 5: Registration flowchart

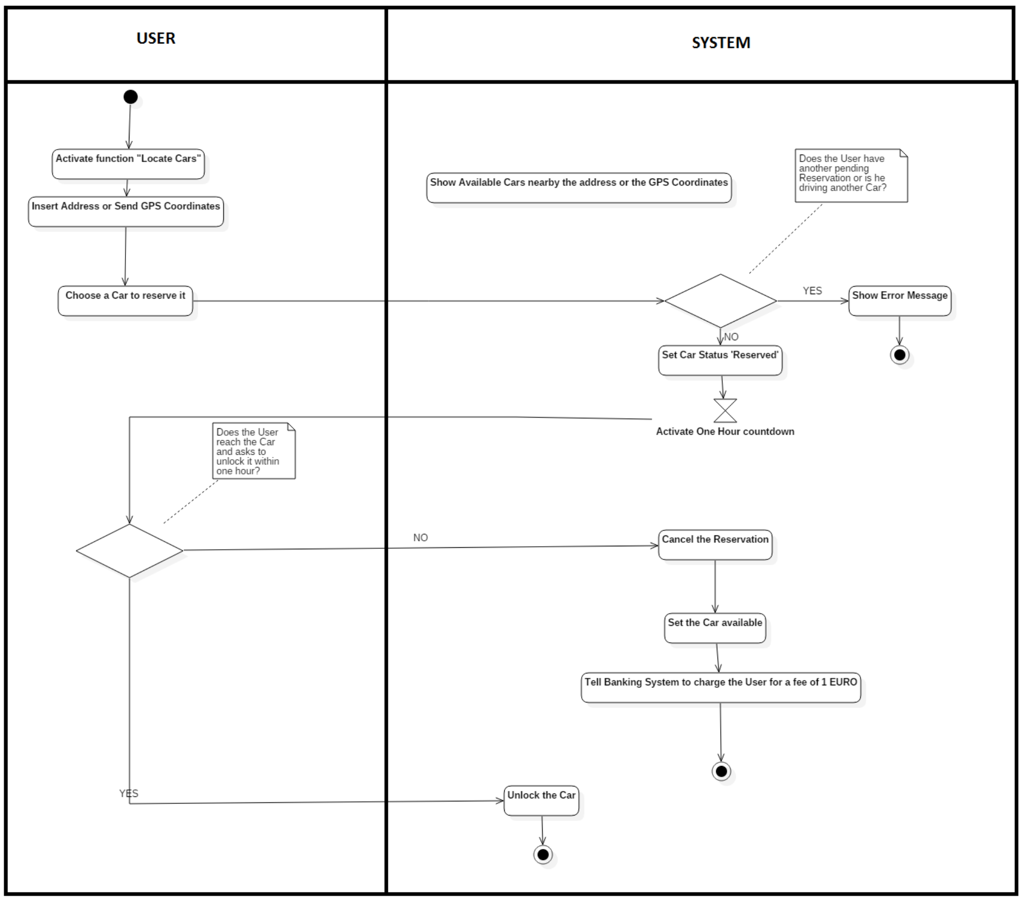


Figure 6: Reservation flowchart

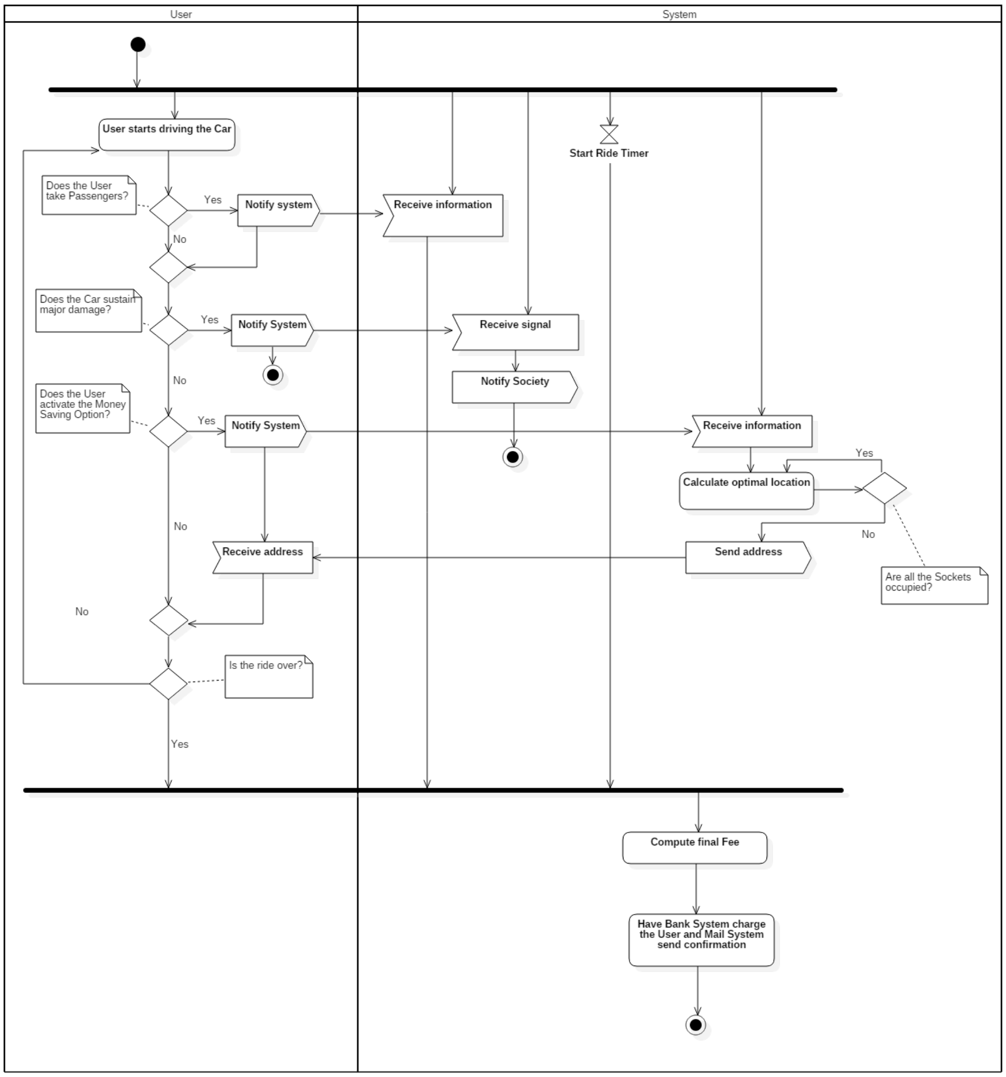


Figure 7: Ride flowchart

### 3.2.5 Sequence Diagrams

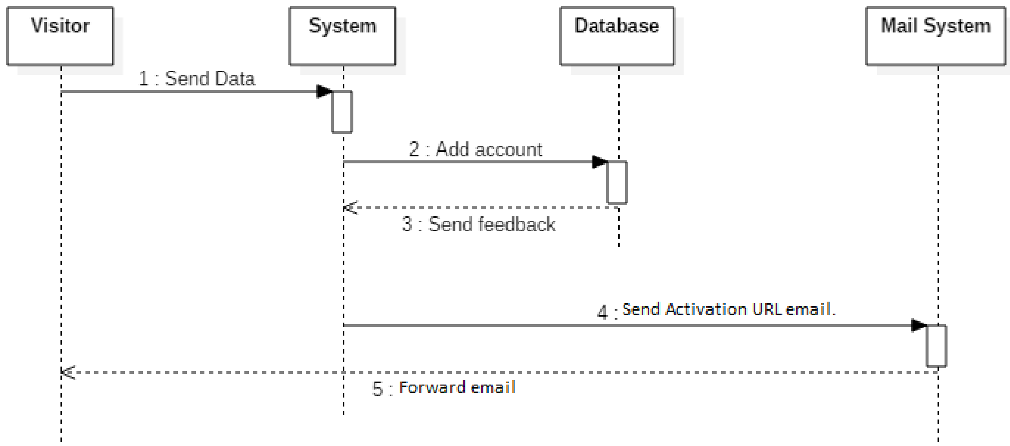


Figure 8: Use Case 1

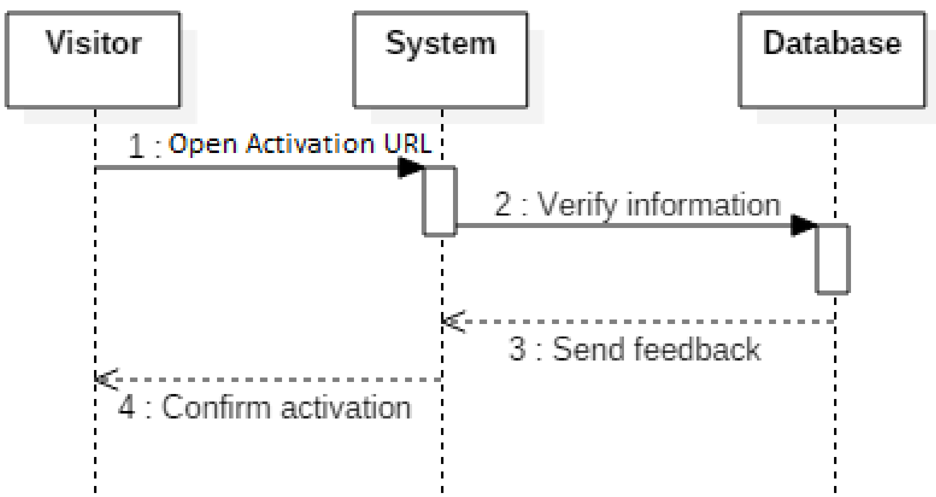


Figure 9: Use Case 2

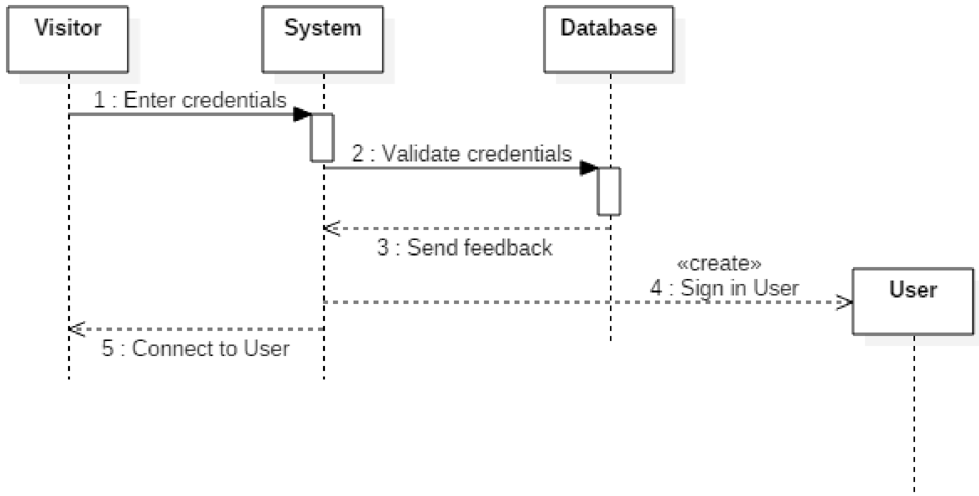


Figure 10: Use Case 3

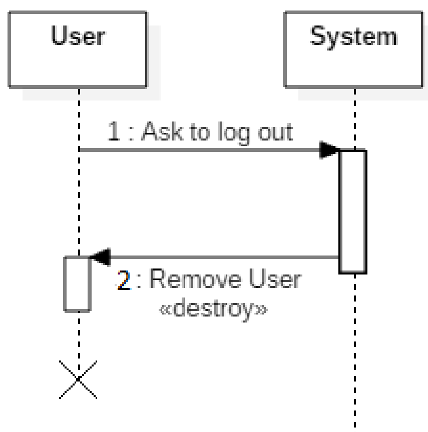


Figure 11: Use Case 4

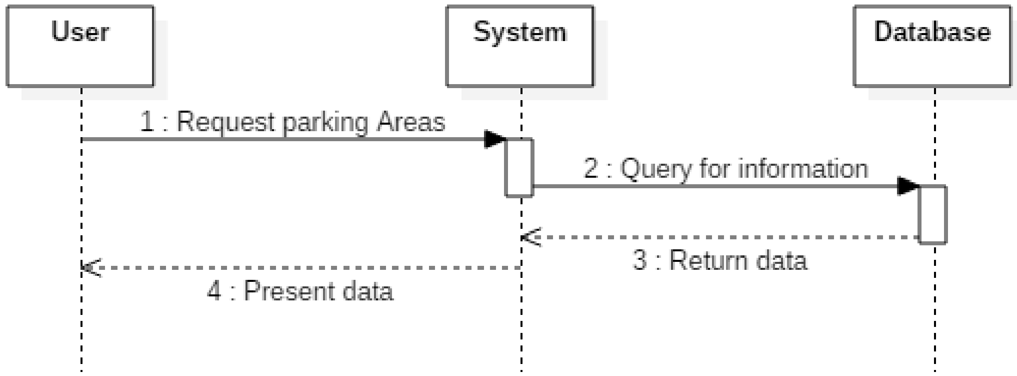


Figure 12: Use Case 5

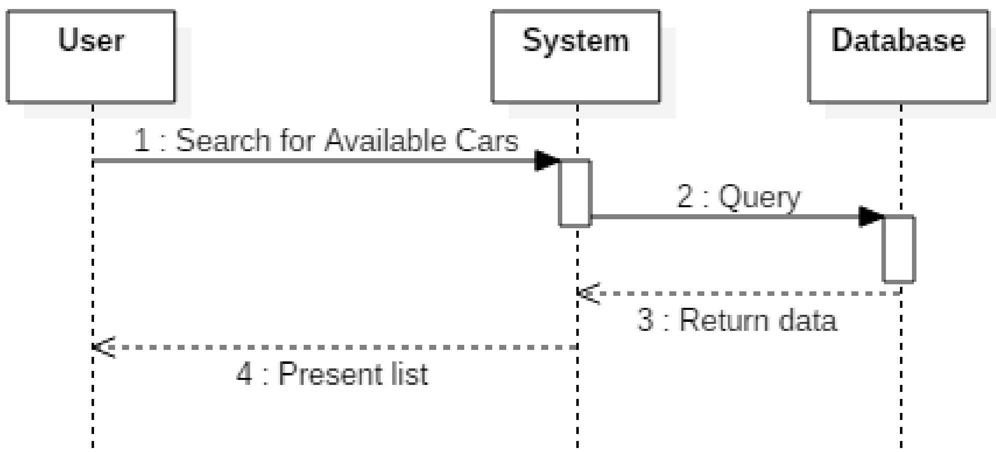


Figure 13: Use Case 6

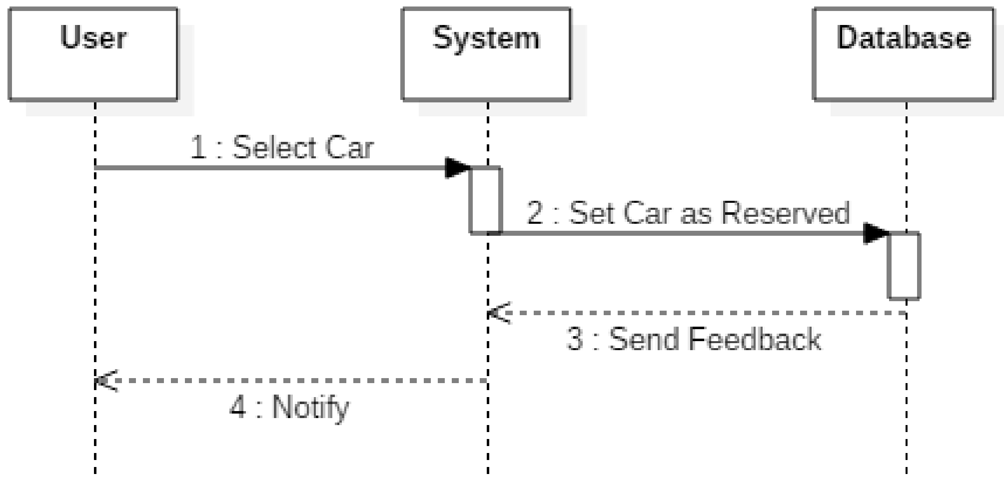


Figure 14: Use Case 7

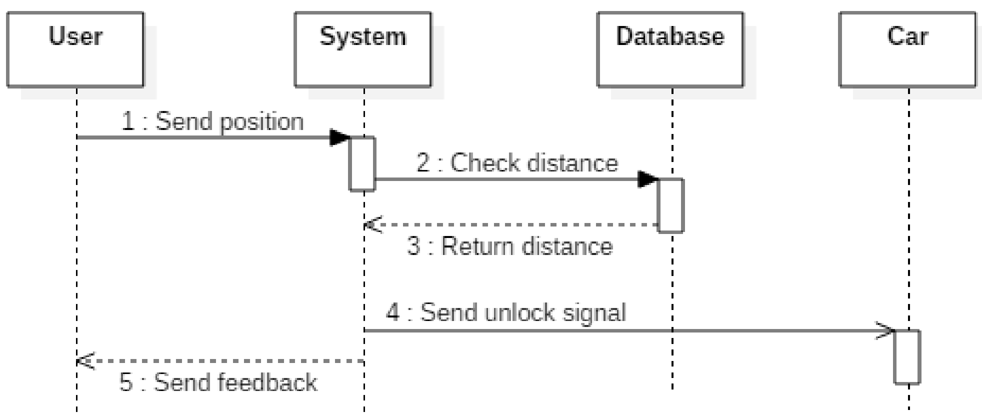


Figure 15: Use Case 8

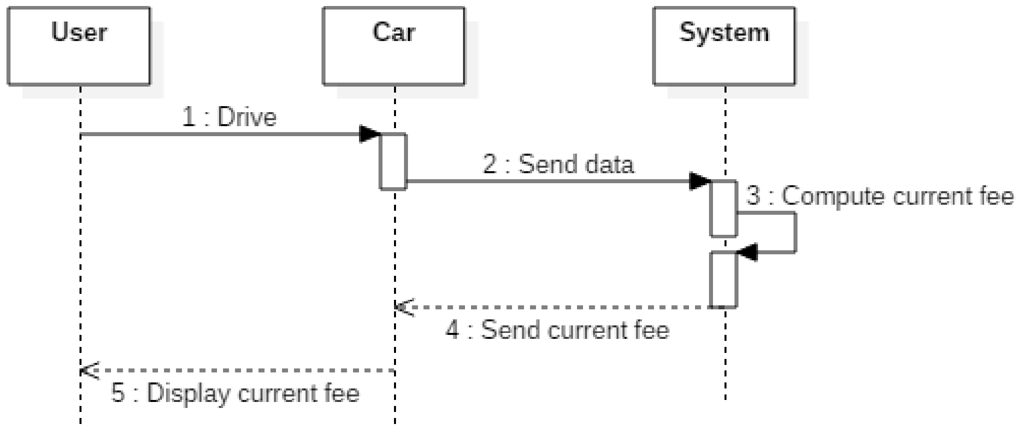


Figure 16: Use Case 9

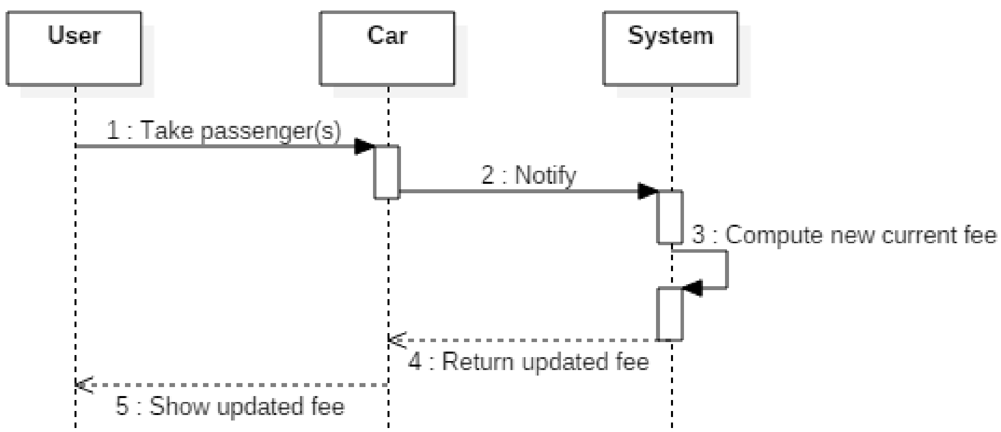


Figure 17: Use Case 10

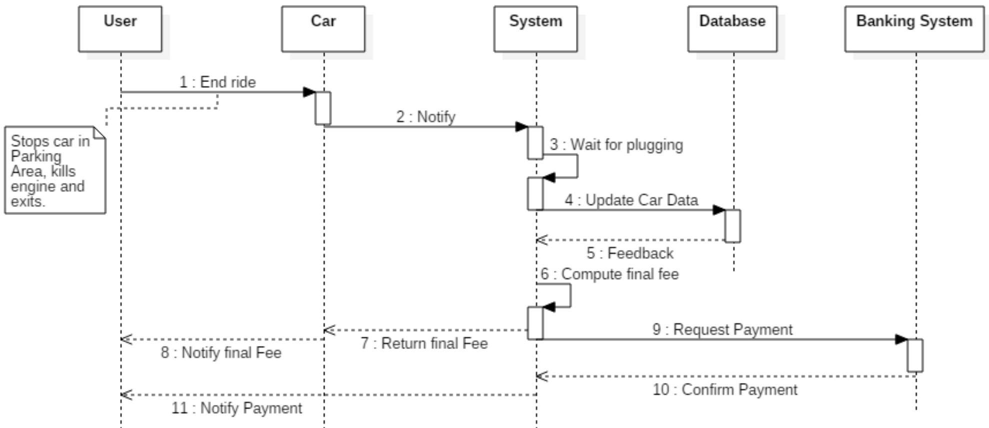


Figure 18: Use Case 11

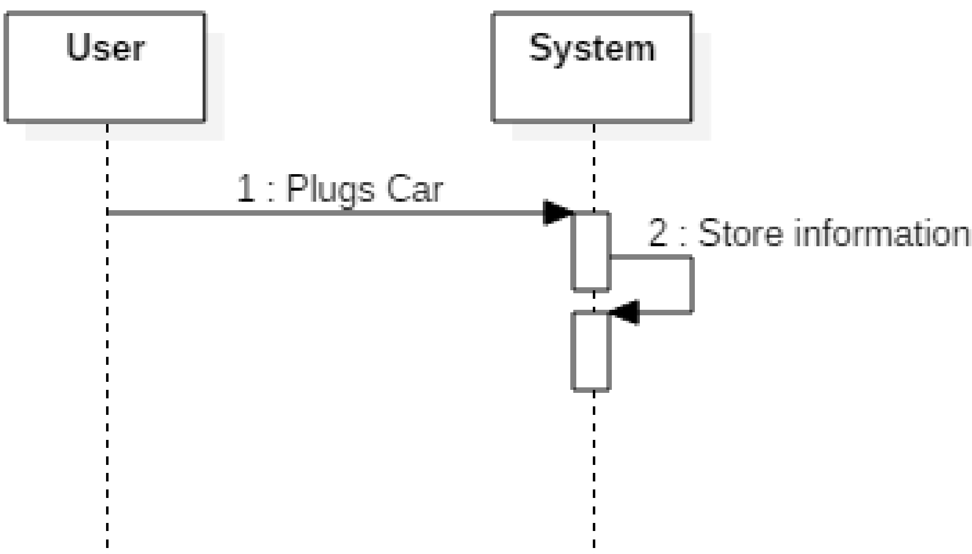


Figure 19: Use Case 12

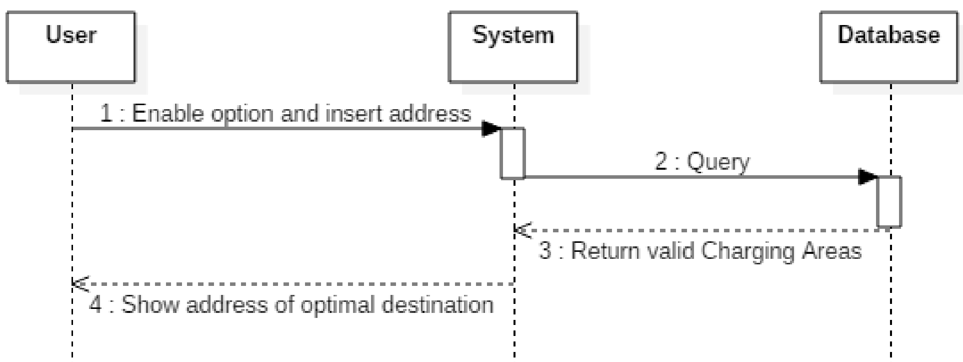


Figure 20: Use Case 13

## 3.3 Non-Functional Requirements

### 3.3.1 Performance Requirements

The System will be able to fulfill clients' requests within 10 seconds from their arrival.

### 3.3.2 Privacy Requirements

All information collected from User and Visitor will not be sold, shared, or rented to others in any way. However, all the information sent to external Systems such as the email text sent to the Mailing System or the Fee sent to the Banking System are not under the responsibility of this System.

### 3.3.3 Safety and Security Requirements

To access all the functionalities of the System, every User should be correctly identified with username and password.

All password should be salted and hashed before the insertion into the Database. For the hashing, it will be used a widely adopted function like SHA-1 or MD5. It is strongly recommended to use prebuilt, already tested libraries provided by all the major programming languages.

The Database should be encrypted in order to prevent the disclosure of relevant sensitive Users data. To further secure Users data, it is important to establish a rigid access control policy on the Database, giving each Developer only a limited access to specific part of the Database. Each Developer (or group of them) should be given the smallest privileges possible in order to fullfil their assignment.

There will not exist cases in which sensible data belonging to the User, Visitor, Company, will be passed on an insecure channel. All the communications between the different machines of this System will be built on top of an encrypted channel such as HTTPS. It is also required to enforce security mechanism in the communication with all the external systems, avoiding the disclosure of all the System IP addresses and encrypting the communication using HTTP(S) or any other kind of secure protocol.

It is also important to avoid the possibility of all the most relevant attack to the System, including but not limited to:

- SQL-injection: it is important to use most known guidelines to avoid this kind of attack

- bruteforce password attempts: it is important to limit the number of login request per second
- URL parameter tampering: encrypt the URL parameters in the HTTP requests
- XSS: it is important to use well known guidelines to avoid this kind of attack

### 3.3.4 Availability and Reliability

The System (including all its components) provide a reliability and availability of 99.9%. The System will be able to fullfil requests coming from Users in at most 10 seconds 99% of the time.

# 4 Appendix

## 4.1 Alloy

We have used the functionalities provided by the Alloy tool in order to represent the domain assumptions of our System. The model, as we will see, represents a snapshot of the System at a given time. All the interesting part of the code are commented in order to better explain their meaning.

We have also added some interesting predicates to show some possible world which is not in contrast with our assumptions.

### 4.1.1 Gps Utilities

```
module GeoUtilities

sig GpsPoint {}

sig GpsVolume {
  gpsPoints: some GpsPoint
}

fact differentGpsVolumeShouldDifferForAtLeastOnePoint {
  all disj gv1, gv2: GpsVolume |
    (gv1.gpsPoints + gv2.gpsPoints) -
    (gv1.gpsPoints & gv2.gpsPoints) ≠ none
}

pred show() {
  #GpsVolume > 1
}
run show for 5
```

In this file, we have modelled the GPS positions that our System has to cope with.

Given our domain assumptions, positions are exact for CompanyArea because they are predefined. In the reality, it does mean that each Parking Area or Charging Area has a given and exact set of GPS points denoting the volume it occupies.

On the other hand, GPS positions for Persons and Cars are derived from devices and they are not always accurate. For this reason, we introduced the concept of GpsVolume, consisting of various GpsPoints, and that should be read as ”the volume that a Person/Car can occupy at a given moment basing on their GPS coordinates”. It basically means that, knowing the GPS coordinates of a person at a given moment, we built a probabilistic assumption of the volume in space he/she is occupying. Obviously the same concept applies for the cars.

For the reasons explained above, in our model we have can have different Persons and/or Cars in the same GpsVolume.

To model the fact that persons or cars are nearby we say that they have to share at least one GpsPoint. So if a Person is inside a Car he/she should have some GpsPoint in common with it; the same concept applies for Cars inside CompanyAreas.

We will clarify these aspects in the following pages.

As a last note, we assume that two different GpsVolumes have at least one different GpsPoint.

A simple world is shown in Figure 21

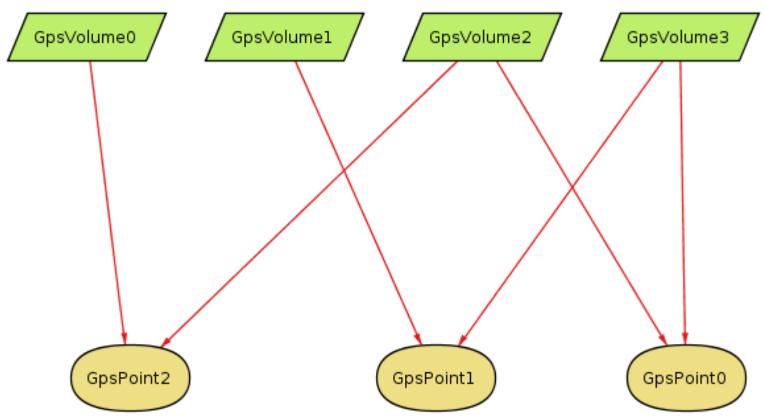


Figure 21: A Gps World

#### 4.1.2 Persons

```
module Persons
open GeoUtilities

/**
```

```
  SIGNATURES
*/
sig Person {
  // We assume that each Person is identified by only one point
  personGpsVolume: one GpsVolume
}
sig User extends Person {}

/**
  PREDICATES/FUNCTIONS
*/
pred show() {
  #Person > 3
}
run show for 6

pred showCouldExistOverlappingPersons() {
  #Person > 1
  #User = 0
  some disj p1, p2: Person |
    p1.personGpsVolume = p2.personGpsVolume
  GpsVolume in Person.personGpsVolume
}
run showCouldExistOverlappingPersons for 2

pred showCouldExistNearbyPersons() {
  #Person > 1
  some disj p1, p2: Person |
    p1.personGpsVolume.gpsPoints & p2.personGpsVolume.gpsPoints ≠ none
}
run showCouldExistNearbyPersons for 4
```

In this file, we have modelled the different kind of people that our System should cope with. In our model, we are not interested in Visitor, so we model simply Persons (general people) and Users (Persons registered to our System).

Figure 22 shows a possible world generated by our Alloy code. We note, for example, that *User0* and *Person1* are both linked to *GpsVolume3*: as we have said before, this is not in contrast with our model.

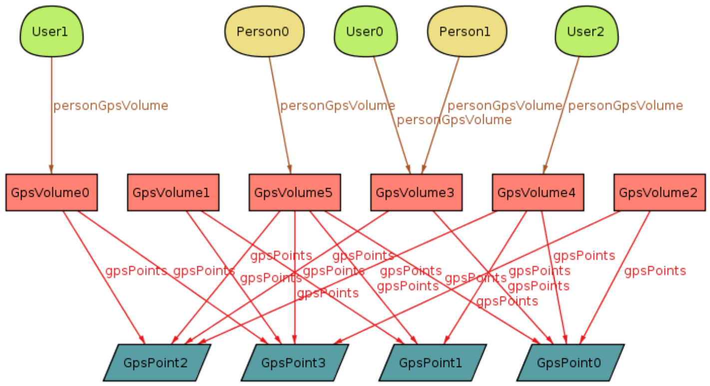


Figure 22: A Persons World

The *showCouldExistNearbyPersons()* predicate is used to show what we have defined as nearby people: two Persons sharing at least one GpsPoint. This is shown in figure 23, where we can see that *User1* and *User2* are nearby.

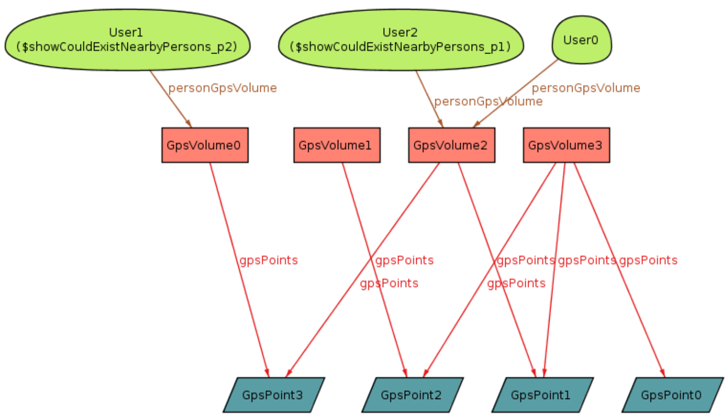


Figure 23: Nearby Persons

### 4.1.3 Cars

```
module Cars
//open util/boolean
```

```
open GeoUtilities
open Persons

/*
  SIGNATURES
*/
sig Car {
  batteryStatus: one BatteryStatus,
  carSeats: some CarSeat,
  usedSeats: Person lone -> lone carSeats,
  damages: set Damage,
  currentState: one CarState,
  pluggedStatus: one PluggedStatus,
  engineStatus: one EngineStatus,
  carGpsVolume: one GpsVolume
}
{
  (usedSeats.carSeats) ≠ none implies currentState = InUse
  currentState ≠ none
  currentState ≠ InUse implies (usedSeats.carSeats) = none
  currentState = InUse implies pluggedStatus = PluggedOff
  (currentState in Reserved + Available) implies
    batteryStatus = HighBattery
  currentState = InUse implies batteryStatus ≠ ZeroBattery
  (batteryStatus = LowBattery and
    currentState ≠ InUse and
    pluggedStatus = PluggedOff) implies
    currentState = Unavailable
  engineStatus = EngineOn implies currentState = InUse
  currentState ≠ InUse implies engineStatus = EngineOff
}

abstract sig BatteryStatus {}
// Battery less than or greater than 20%
lone sig LowBattery, HighBattery extends BatteryStatus {}
```

```
lone sig ZeroBattery extends LowBattery{}

abstract sig EngineStatus {}
lone sig EngineOn, EngineOff extends EngineStatus {}

abstract sig PluggedStatus {}
lone sig PluggedOn, PluggedOff extends PluggedStatus {}

abstract sig CarState {}
lone sig Available, Unavailable, Reserved, InUse
    extends CarState {}

sig CarSeat {}

abstract sig Damage {}
sig MajorDamage, MinorDamage extends Damage {}

/*
  FACTS
*/
// Trivial relations
fact allEngineStatusAreAssociatedToSomeCar {
  all es: EngineStatus | es in Car.engineStatus
}

fact allPluggedStatusAreAssociatedToSomeCar {
  all ps: PluggedStatus |  ps in Car.pluggedStatus

}

fact allBatteryStatusMustBeAssociatedToSomeCar {
  all b: BatteryStatus | b in Car.batteryStatus
}

fact allCarStatesMustBeAssociatedToSomeCars {
  all cs: CarState | cs in Car.currentState
}

fact allCarSeatsMustBeAssociatedToOneCar {
  all cs: CarSeat | one c: Car | cs in c.carSeats
```

```
}

fact damagesMustBeAssociatedToACar {
  all d: Damage | d in Car.damages
}


// Others
fact personsAreNotUbiquituous {
  all disj c1, c2: Car | no p: Person |
    p in (c1.usedSeats).CarSeat and
    p in (c2.usedSeats).CarSeat
}

fact personsInUsedSeatsHaveSamePositionOfCar {
  all c: Car, p: Person | p in (c.usedSeats).CarSeat
   implies
    p.personGpsVolume.gpsPoints & c.carGpsVolume.
   gpsPoints ≠ none
}

fact majorDamagesImpliesUnavailableCars {
  all c: Car, m: MajorDamage | m in c.damages implies
    c.currentState = Unavailable
}


/**
  ASSERTS
*/
assert allPersonsCantBeInDifferentCars {
  all disj c1, c2: Car | no p: Person |
    p in (c1.usedSeats).CarSeat and p in (c2.usedSeats)
   .CarSeat
}
check allPersonsCantBeInDifferentCars for 10

assert allPersonsInACarMustHaveThatCarPosition {
  all p: Person, c: Car | p in (c.usedSeats).CarSeat
   implies
```

```
    p.personGpsVolume.gpsPoints & c.carGpsVolume.gpsPoints ≠ none
}

assert allMajorDamagedCarsAreUnavailable {
  all m: MajorDamage, c: Car | m in c.damages implies
    c.currentState = Unavailable
}
check allMajorDamagedCarsAreUnavailable for 10

assert allReservedOrAvailableCarsHaveHighBatteries {
  all c: Car | c.currentState in (Reserved + Available) implies
    c.batteryStatus = HighBattery
}
check allReservedOrAvailableCarsHaveHighBatteries for 3

assert noCarInUseHaveZeroBattery {
  no c: Car | c.currentState = InUse and c.batteryStatus = ZeroBattery
}
check noCarInUseHaveZeroBattery for 10

assert allCarWithUsedSeatsShouldBeInUse {
  all c: Car | (c.usedSeats).CarSeat ≠ none implies c.currentState = InUse
}
check allCarWithUsedSeatsShouldBeInUse for 10

assert allCarsNotInUseAndNotPluggedAndWithLowBatteryShouldBeUnavailable {
  all c: Car | (c.batteryStatus = LowBattery and
    c.currentState ≠ InUse and
    c.pluggedStatus = PluggedOff) implies
    c.currentState = Unavailable
}
check allCarsNotInUseAndNotPluggedAndWithLowBatteryShouldBeUnavailable for 10
```

```
assert noPluggedCarIsInUse {
  all c: Car | c.currentState = InUse implies c.pluggedStatus = PluggedOff
}
check noPluggedCarIsInUse for 10

assert allEnginesOnAreAssociatedToInUseCars {
  all c: Car | c.engineStatus = EngineOn implies c.currentState = InUse
}
check allEnginesOnAreAssociatedToInUseCars for 3

assert allUsedSeatsHaveSamePositionOfCars {
  all c: Car | (c.usedSeats).CarSeat ≠ none implies
    (c.usedSeats).(c.carSeats).personGpsVolume.gpsPoints &
      c.carGpsVolume.gpsPoints ≠ none
}
check allUsedSeatsHaveSamePositionOfCars for 3


/*
  PREDICATES/FUNCTIONS
*/
// A car may be perfectly functioning but still unavailable (the external
// employee has manually set the status to Unavailable)
pred showCouldExistSomeUnavailableCarWithNoMajorDamageAndHighBattery {
  #Car > 0
  #Unavailable = #Car
  #MajorDamage = 0
  #LowBattery = 0
  #Person = 0
  GpsVolume in (Car.carGpsVolume + Person.personGpsVolume)

}
```

```
run showCouldExistSomeUnavailableCarWithNoMajorDamageAndHighBattery for 3

pred showCouldExistSomeCarWithLoweBattery {
  #Car > 0
  #LowBattery > 0
}
run showCouldExistSomeCarWithLoweBattery for 3

// A car may have minor damages but still available (the external
// employee has manually set the status to Available)
pred showCouldExistSomeAvailableCarWithMinorDamages {
  #MinorDamage = #Car
  #Available = #Car
}
run showCouldExistSomeAvailableCarWithMinorDamages for 3

// It does mean that a User has turned the engine off outside a parking area
pred showCouldExistSomeInUseCarsWithEngineOff {
  #Car > 0
  #InUse = #Car
  #EngineOff = #Car
}
run showCouldExistSomeInUseCarsWithEngineOff for 3

// Same as before, all the people have left the car, even it is still in use
pred showCouldExistSomeInUseCarsWithEngineOnAndAllPersonsOutside {
  #Car > 0
  #InUse = #Car
  #EngineOn = #Car
  #Person > 0
  #Damage = 0
  #CarSeat = #Car
```

```
  #Car.usedSeats = 0
  GpsVolume in (Car.carGpsVolume + Person.personGpsVolume)
}
run showCouldExistSomeInUseCarsWithEngineOnAndAllPersonsOutside for 6

// Not only users have access to the car. We ensure that a User reserve a Car,
// but we don't know if he/she will use it.
pred showCouldExistSomeInUseCarsWithAllSeatsOccupiedByNonUsers {
  #Car > 0
  #Person > 0
  #User = 0
}
run showCouldExistSomeInUseCarsWithAllSeatsOccupiedByNonUsers for 3

// Show that different people can be in the same car
pred showMorePersonsInOneCar {
  #Car.usedSeats > 1
  #Car = 1
}
run showMorePersonsInOneCar for 7

pred show() {
  #Car > 0
  #Person > 0
  #GpsVolume > 1
  #Car.damages < 3
}
run show for 3
```

In this piece of code we show our model for the Cars managed by our System; a possible world is shown in figure 24. We can note that there is a single car, characterized by

- a PluggedOff status: this is consistent since the car is also InUse;
- an EngingOn status: as for the above, this is consistent since the car is also InUse;
- two different MinorDamages: this is consistent since Users can also use Cars that have minor damages;
- a LowBattery: this is consistent since the car is InUse; when the Car will be parked, its status, according to our assumptions, will be set to Unavailable
- two CarSeats: they are occupied by an User and a Person, that are both nearby our Car (i.e. they have at least one GpsPoint in common with our Car).

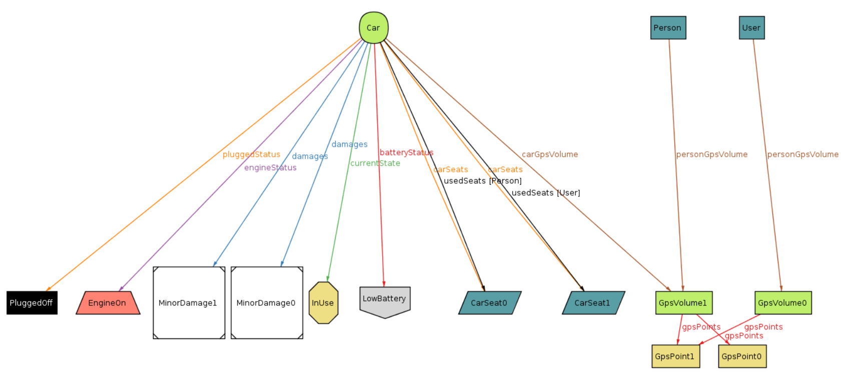


Figure 24: A Cars World

We have also shown in 25 that the execution of all the assertions have not generated counterexamples, so we can assume reasonably assume that our model is consistent.

An important aspect of our System is that a Car can be In Use, but with no person inside it. The world for this scenario is represented in Figure 26. Another interesting aspect shown in this image is that, although no one is inside the Car, its engine is still on.

Another meaningful aspect of our System is the possibility to have perfectly functioning cars whose status is Unavailable. This is surely due to some external Employee who have manually set the status of the Car for whatever reason. This world is shown in Figure 27.

**17 commands were executed. The results are:**
#1: No counterexample found. allPersonsCantBeInDifferentCars may be valid.
#2: No counterexample found. allMajorDamagedCarsAreUnavailable may be valid.
#3: No counterexample found. allReservedOrAvailableCarsHaveHighBatteries may be valid.
#4: No counterexample found. noCarInUseHaveZeroBattery may be valid.
#5: No counterexample found. allCarWithUsedSeatsShouldBeInUse may be valid.
#6: No counterexample found. allCarsNotInUseAndNotPluggedAndWithLowBatteryShouldBeUnavailable may be valid.
#7: No counterexample found. noPluggedCarIsInUse may be valid.
#8: No counterexample found. allEnginesOnAreAssociatedToInUseCars may be valid.
#9: No counterexample found. allUsedSeatsHaveSamePositionOfCars may be valid.
#10: Instance found. showCouldExistSomeUnavailableCarWithNoMajorDamageAndHighBattery is consistent.
#11: Instance found. showCouldExistSomeCarWithLoweBattery is consistent.
#12: Instance found. showCouldExistSomeAvailableCarWithMinorDamages is consistent.
#13: Instance found. showCouldExistSomeInUseCarsWithEngineOff is consistent.
#14: Instance found. showCouldExistSomeInUseCarsWithEngineOnAndAllPersonsOutside is consistent.
#15: Instance found. showCouldExistSomeInUseCarsWithAllSeatsOccupiedByNonUsers is consistent.
#16: Instance found. showMorePersonsInOneCar is consistent.
#17: Instance found. show is consistent.

Figure 25: Executions of checks and predicates for Cars

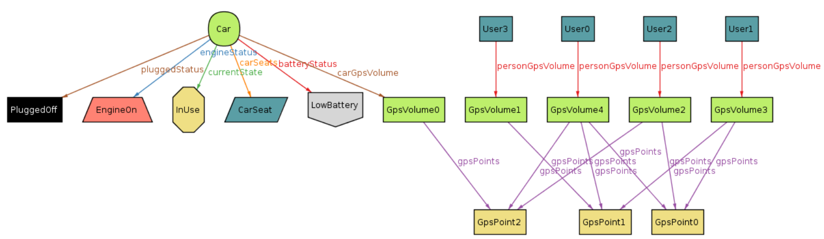


Figure 26: Used cars with no person inside

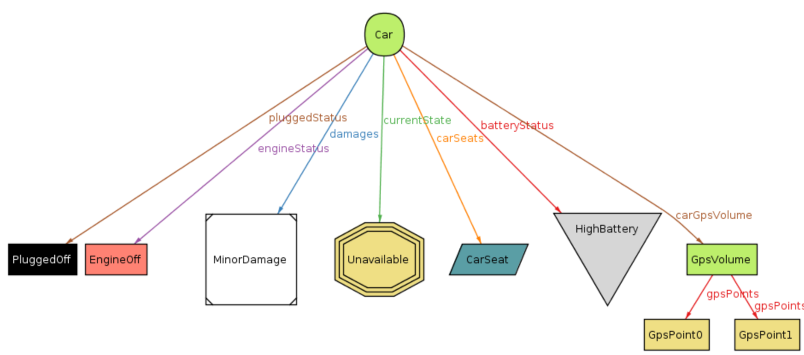


Figure 27: Unavailable functioning cars

### 4.1.4 Areas

```
module Areas
open Cars
open GeoUtilities

/**
  SIGNATURES
*/
abstract sig CompanyCarSlot {}
sig ParkingSlot, ChargingSlot extends CompanyCarSlot {}

abstract sig CompanyArea {
  // We assume that a CompanyArea is composed by a non empty set of Points
  // This is enough for our modelation of the world
  areaGpsPoints: some GpsPoint
}

sig ParkingArea extends CompanyArea {
  parkingSlots: set ParkingSlot,
  parkedCars: Car lone -> lone parkingSlots
}

sig ChargingArea extends ParkingArea {
  chargingSlots: some ChargingSlot,
  chargingCars: Car lone -> lone chargingSlots
}

/**
  FACTS
*/
// Trivial
fact parkingSlotsAreaAssociatedToExactlyOneArea {
  all ps: ParkingSlot | one pa: ParkingArea | ps in pa.parkingSlots
}

fact chargingSlotsAreaAssociatedToExactlyOneArea {
```

```
  all cs: ChargingSlot | one ca: ChargingArea | cs in ca.chargingSlots
}

// Areas do not overlap
fact areaPositionsAreAssociatedToExaxtlyOneCompanyArea {
//  Gps volumes for company area are predefined, so there is no way different
//  areas overlap
  all disj a1, a2: CompanyArea |
    a1.areaGpsPoints & a2.areaGpsPoints = none
}

// Parked Cars are nearby Parking Areas
fact allParkedCarsAreInsideThoseAreaPositions {
  all pa: ParkingArea, c: Car |
    c in (pa.parkedCars).(pa.parkingSlots) implies
    c.carGpsVolume.gpsPoints & pa.areaGpsPoints ≠ none
}

//Charging Cars are nearby Charging Areas
fact allChargingCarsAreInsideThoseAreaPositions {
  all ca: ChargingArea, c: Car |
    c in (ca.chargingCars).(ca.chargingSlots) implies
    c.carGpsVolume.gpsPoints & ca.areaGpsPoints ≠ none
}

// If a Car is inside an Area but not occupying a slot, it should be in use
fact allCarsInsideAreasButNotParkedOrChargingAreInUse {
  all c: Car |
    (c.carGpsVolume.gpsPoints in ParkingArea.areaGpsPoints and
     c not in
     ( (ParkingArea.parkedCars).ParkingSlot +
       (ChargingArea.chargingCars).ChargingSlot )) implies
       c.currentState = InUse
}
```

```
// I.e. a ParkingArea has always a parkingCapacity > 0
fact parkingCapacityZeroCanOnlyBeAssociatedToChargingArea {
  all p: ParkingArea | p.parkingSlots = none implies
    p in ChargingArea
}

// N.B.: Implies and not Iff bcz a car in a ParkingArea can also be Unavailable
fact carStateAvailableOrReservedImpliesCarAtOneParkingArea {
  all c: Car, pa: ParkingArea, ca: ChargingArea |
    (c.currentState = Available or c.currentState = Reserved) implies
    ( (c in (pa.parkedCars).ParkingSlot) or
      (c in (ca.parkedCars).ParkingSlot) or
      (c in (ca.chargingCars).ChargingSlot ))
}
// If a car is plugged ≤> it must be in one charging area
fact carStatePluggedIffCarInOneChargingCars {
  all c: Car | one ca: ChargingArea |
    c.pluggedStatus = PluggedOn iff c in (ca.chargingCars).(ca.chargingSlots)
}

fact carCantBeChargingAndParkedAtSameTime {
  no (ParkingArea.parkedCars).ParkingSlot &
      (ChargingArea.chargingCars).ChargingSlot
}

fact carParkedInOneParkingArea {
  all pa1, pa2: ParkingArea |
    (pa1 ≠ pa2 implies
      (pa1.parkedCars).ParkingSlot & (pa2.parkedCars).ParkingSlot = none)
}
```

```
fact carChargingInOneChargingArea {
  all ca1, ca2: ChargingArea |
    (ca1 ≠ ca2 implies
      (ca1.chargingCars).ChargingSlot &
      (ca2.chargingCars).ChargingSlot = none)
}

fact carStateInUseIfItIsNotInAParkingOrChargingSlot {
  all c: Car | c.currentState = InUse implies
    c not in ( (ParkingArea.parkedCars).ParkingSlot +
            (ChargingArea.chargingCars).ChargingSlot)
}

/**
  ASSERTS
*/
assert areaPositionsAreNotOverlapping {
  all disj ca1, ca2: CompanyArea | ca1.areaGpsPoints &
   ca2.areaGpsPoints = none
}
check areaPositionsAreNotOverlapping for 10

assert sameCarShouldNotBePluggedAtDifferentChargingArea
    {
  all c: Car | one ca: ChargingArea |
    c.pluggedStatus = PluggedOn iff
    c in (ca.chargingCars).(ca.chargingSlots)
}
check sameCarShouldNotBePluggedAtDifferentChargingArea
   for 10

assert sameCarShouldNotBeParkedAtDifferentParkingArea {
  all disj p1, p2: ParkingArea |
    (p1.parkedCars).ParkingSlot & (p2.parkedCars).
   ParkingSlot = none
}
check sameCarShouldNotBeParkedAtDifferentParkingArea
   for 10
```

```
// Bcz we assume disjoint sets
assert sameCarShouldNotBeParkedAndChargingAtSameTime {
  no (ParkingArea.parkedCars).ParkingSlot &
      (ChargingArea.chargingCars).ChargingSlot
}
check sameCarShouldNotBeParkedAndChargingAtSameTime for 10

assert carsParkedOrChargingAreNearbyThoseAreas {
  all c: Car |
    c in ( (ParkingArea.parkedCars).ParkingSlot +
        (ChargingArea.chargingCars.ChargingSlot) )
      implies
      (c.carGpsVolume.gpsPoints & ParkingArea.areaGpsPoints ≠ none)
}
check carsParkedOrChargingAreNearbyThoseAreas for 5

assert allParkingOrChargingCarsAreNotInUse {
  all c: Car | c.currentState = InUse implies
    c not in ( (ParkingArea.parkedCars).ParkingSlot +
            (ChargingArea.chargingCars).ChargingSlot)
}
check allParkingOrChargingCarsAreNotInUse for 10


/**
  PREDICATES/FUNCTIONS
*/
pred show() {
  all p: GpsPoint | p in Person.personGpsVolume.gpsPoints or p in CompanyArea.areaGpsPoints or
    p in Car.carGpsVolume.gpsPoints
  GpsVolume in (Person.personGpsVolume + Car.carGpsVolume)
  #GpsVolume > 1

  #Car > 0
  all c: Car | #c.carSeats < 3 and #c.damages < 2
  #Car.usedSeats > 0
```

```

  #Person > 0
  #(Person - User) > 0

  #CompanyArea > 0
  #(ParkingArea - ChargingArea) > 0

  #ParkingArea.parkedCars > 0
  #ChargingArea.chargingCars > 0
}
run show for 3
```

Here we define the CompanyAreas and all the things related to them.

Examples of possible worlds are shown in the following figures.

In Figure 28 we show a Car which is In Use and at the same time inside a Charging Area without occupying any of its charging slots. This does not come as a surprise: an User can still be inside an Area even if he/she is using the Car. However, we can also notice that, even if the Car is InUse, there is no Person occupying any of the seats. The only User shown in the figure has the same position of the ChargingArea (i.e. he/she is nearby it) and the same position of the Car (i.e. he/she is nearby it).

Figure 29, instead, shows a Charging Area with a Car inside it. The Car is occupying a ParkingSlot of this ChargingArea. Its status, however, is Unavailable, maybe due to the fact that it has ZeroBattery.

Adding more objects to the model, we can see how things get complicated (but still consistent). Possible worlds are shown in figures 30 and 31.

Even in this case, we can see in figure 32 how the execution of all checks has not shown any counterexample for our model.

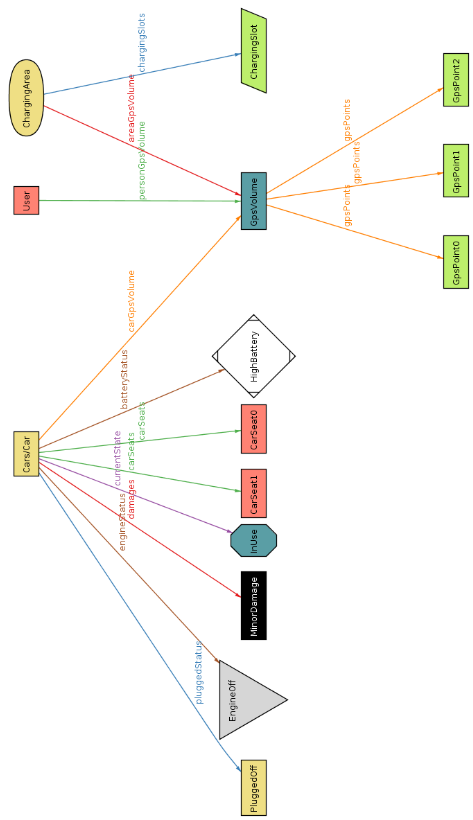


Figure 28: An Areas World

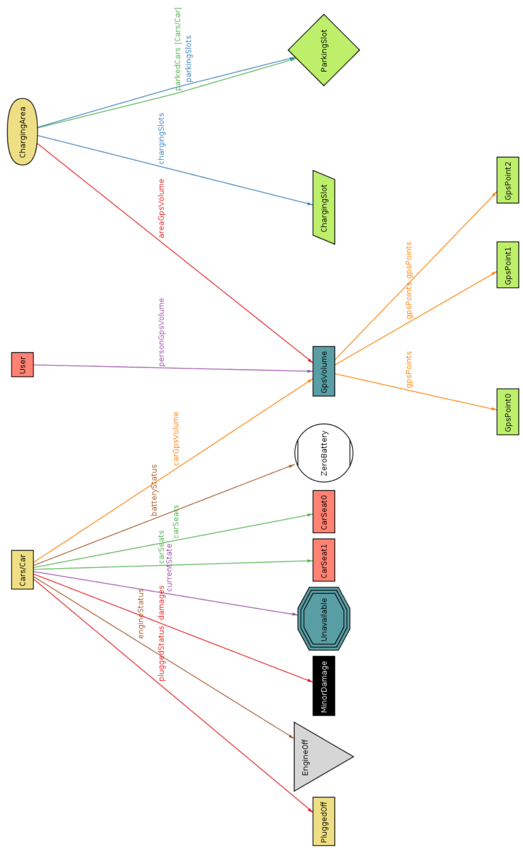


Figure 29: Another Areas World

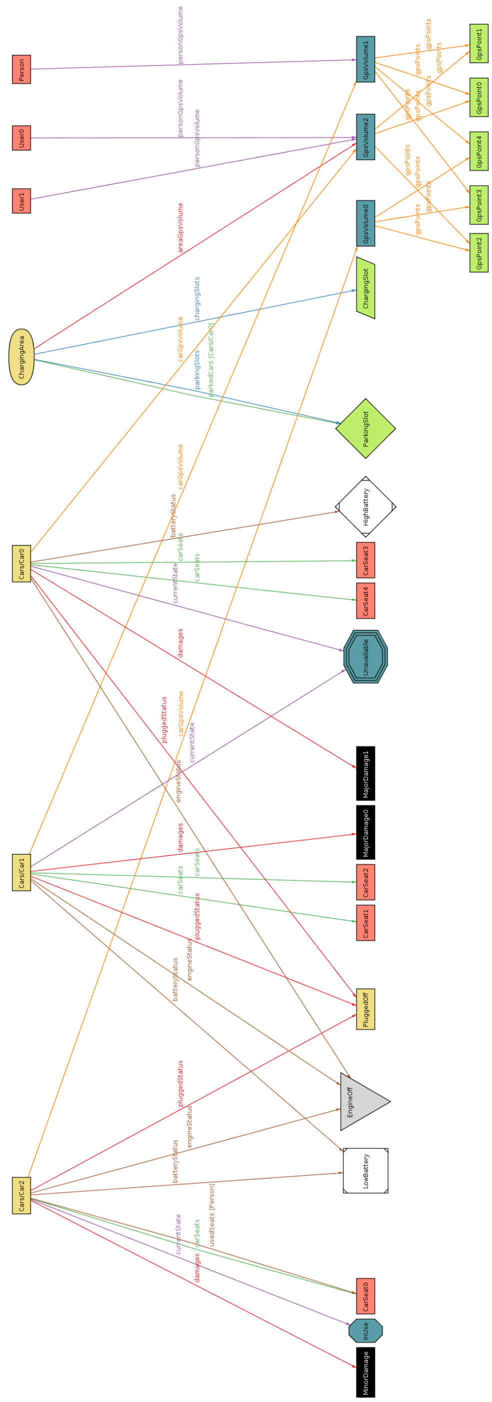



Figure 30: A more complicated Areas World

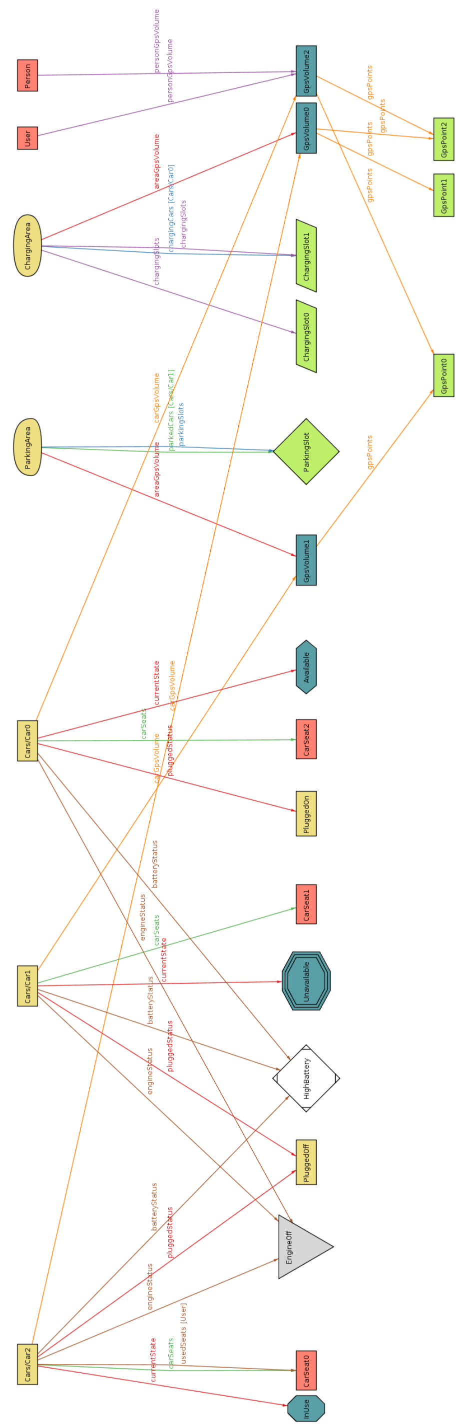




Figure 31: Another more complicated Areas World

```
7 commands were executed. The results are:
  #1: No counterexample found. areaPositionsAreNotOverlapping may be valid.
  #2: No counterexample found. sameCarShouldNotBePluggedAtDifferentChargingArea may be valid.
  #3: No counterexample found. sameCarShouldNotBeParkedAtDifferentParkingArea may be valid.
  #4: No counterexample found. sameCarShouldNotBeParkedAndChargingAtSameTime may be valid.
  #5: No counterexample found. carsParkedOrChargingAreNearbyThoseAreas may be valid.
  #6: No counterexample found. allParkingOrChargingCarsAreNotInUse may be valid.
  #7: Instance found. show is consistent.
```

Figure 32: Execution of checks and predicates for areas

## 4.2 Revision History

**Revision History**

| **Version** | **Date** | **Changelog** |
|---|---|---|
| 1.0 | 13/11/2016 | First release, delivered |
| 1.1 | 29/12/2016 | • Global changes after Design phase and laboratory sessions.<br>• Small changes in Alloy code.<br>• Modified Alloy section in order to better explain code and output figures.<br>• Changes in Glossary.<br>• Modified Class diagram.<br>• Fixed typos. |

## 4.3 Working Hours

This is the comprehensive list of the working hours reported by each member.

### 4.3.1 Alessandro Paglialonga

- 21/10/16 : 1h and 30mins (Meeting with Simone, planning tasks division and choosing shared tools with other teammates)
- 24/10/16 : 1h and 30 mins
- 25/10/16 : 1h and 40 mins
- 31/10/16 : 2h
- 01/11/16 : 4h
- 02/11/16 : 4h
- 03/11/16 : 4h
- 04/11/16 : 5h
- 05/11/16 : 2h and 30 mins
- 06/11/16 : 3 and 40 mins (1h and 30mins meeting with Simone)
- 07/11/16 : 4h
- 08/11/16 : 4h and 40 mins
- 09/11/16 : 4h and 20 mins
- 10/11/16 : 6h (3h meeting with Simone)
- 11/11/16 : 4h (2h meeting with Simone )
- 12/11/16 : 3h
- 13/11/16 : 3h

Total: 59h

### 4.3.2 Simone Perriello

- 21/10/16 : 1h 30mins (meeting)
- 24/10/16 : 1h
- 27/10/16 : 1h
- 31/10/16 : 2h
- 01/11/16 : 3h
- 02/11/16 : 4h
- 03/11/16 : 5h
- 04/11/16 : 3h
- 05/11/16 : 4h
- 06/11/16 : 6h
- 09/11/16 : 3h
- 10/11/16 : 3h
- 11/11/16 : 8h
- 12/11/16 : 8h
- 13/11/16 : 10h

Total: 62h 30m

### 4.3.3 Enrico Migliorini

50 total hours.