Requirement Analysis and Specification Document for PowerEnJoy

Enrico Migliorini, Alessandro Paglialonga, Simone Perriello

December 3, 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 his/her 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 he/she 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 he/she is in a distance range of 15 meters from the same Car.

G7 The system should allow each User to drive a Car which he/she has previously unlocked.

G8 The System should be able to know the time usage of the Car, misured in minutes.

G9 The System should allow Users to know where are the Parking Areas.

G10 The system should allow each User to end the ride in a Parking Area.

G11 If the System detects the User took at least two other passengers onto the Car, 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 at special parking areas where they can be recharged 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.

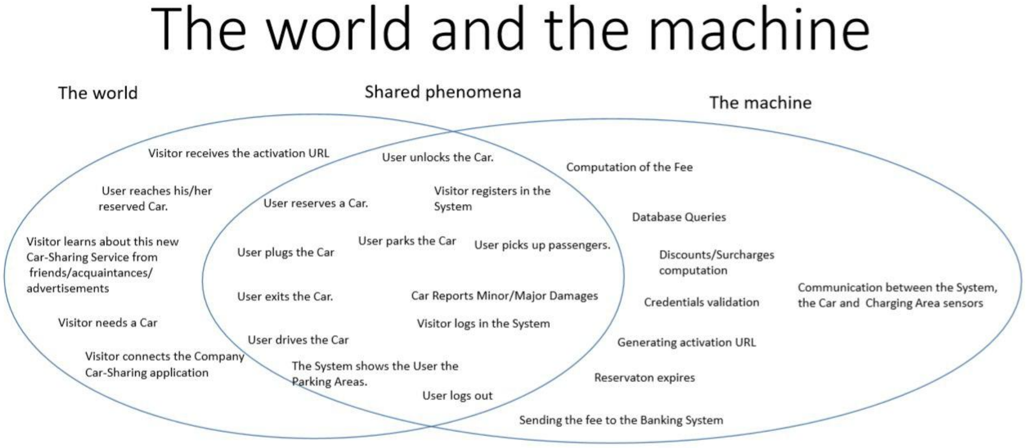

The world and the machine
The world
Shared phenomena
The machine
Visitor receives the activation URL
User reaches his/her reserved Car.
Visitor learns about this new Car-Sharing Service from friends/acquaintances/ advertisements
Visitor needs a Car
Visitor connects the Company Car-Sharing application
User unlocks the Car.
User reserves a Car.
Visitor registers in the System
User plugs the Car
User parks the Car
User picks up passengers.
User exits the Car.
Car Reports Minor/Major Damages
Visitor logs in the System
User drives the Car
The System shows the User the Parking Areas.
User logs out
Computation of the Fee
Database Queries
Discounts/Surcharges computation
Credentials validation
Communication between the System, the Car and Charging Area sensors
Generating activation URL
Reservaton expires
Sending the fee to the Banking System

Figure 1: The World And The Machine

1.4 Definitions, Acronyms and Abbreviations

1.4.1 Business terms glossary

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

- ***Application***
  It's the software interface between the User and the System, which allows the User to access the System functionalities.

- ***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.The Car communicates to the System its position, the status of its battery, its damages, the connection to an electrical socket and the number of seats occupied. A Car has a status and a Plugged flag, the status can be:

  - *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, mainteinance, ...)

  Additionally, the *Plugged* flag indicates if the Car is plugged or not to the socket of the Charging Area.

- ***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 an Increase.

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

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

- ***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.

- ***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.

- ***Employee***
  He's an employee of the Company which is charged of every kind of maintenance of the Car (Charging the Car battery on-site, moving a Car to a Charging Area and so on). The employees are handled by an External System

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

- ***GPS***
  Global Positioning System, it's widely used in our System.

- ***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.

- ***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.

- ***Passenger***
  A person, different from the User, who travels in a Car together with an User.

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

- ***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.

- ***Reservation***
  A User performs a Reservation in order to book an Available Car for a maximum of 1 hour. In this time period the Car is assigned to the specific User only. An User can only have one active Reservation at time.

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

- ***System***
  The software structure this document is about.

- ***User***
  A person registered on the System, who has access to the Car-Sharing Service functionalities.

- ***Visitor***
  A person who needs to log in the System to access the Car-Sharing Service functionalities.

1.4.2 External systems

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

- ***Mail 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.

1.4.3 Document specific terms

- ***Alloy*** A descriptive language that allows to describe a set of structures through constraints.

- ***DBMS.*** Data Base Management System. A software interface allowing to interact with the ***Database***.

- ***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.

The System will be able to communicate with all the involved external systems, such as the Database,in which all the information are stored and can be modified by the System, 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. The System will adopt the HTTP(S) protocol to communicate with the above specified systems managing the interactions through a set of shared protocols and APIs.

End users will access the System functionalities through an Internet connection, using the User intended interface of the System, the Application. Hence, the System should also be able to implement the Application Layer of the Internet Protocol Suite. Users will communicate their position to the System using the GPS coordinates in order to unlock the previously reserved Car.

At last, the System should be able to communicate with the wide variety of sensors placed inside the Cars, in order to know, in every moment, their position, the status of their battery, their possible damages, their plugged status and the number of seats occupied.

2.1.1 Class Diagram

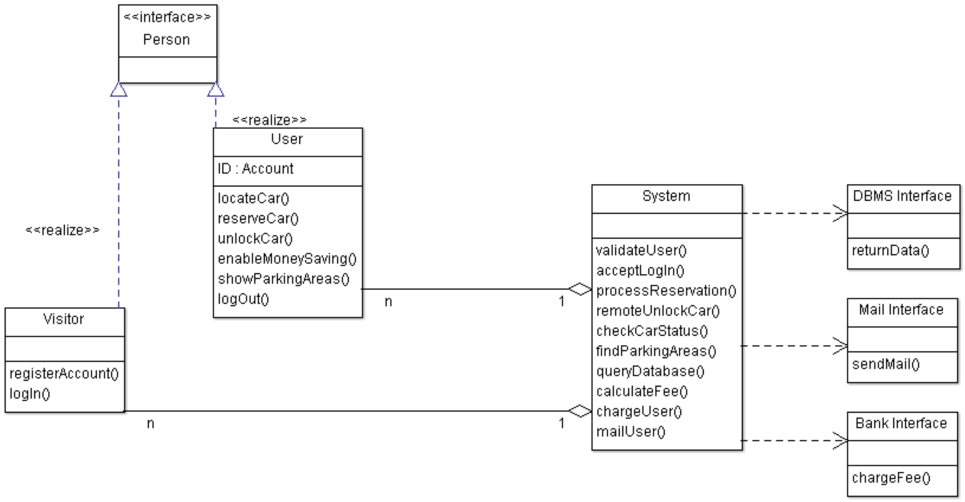


Figure 2: Class Diagram

2.1.2 Statechart

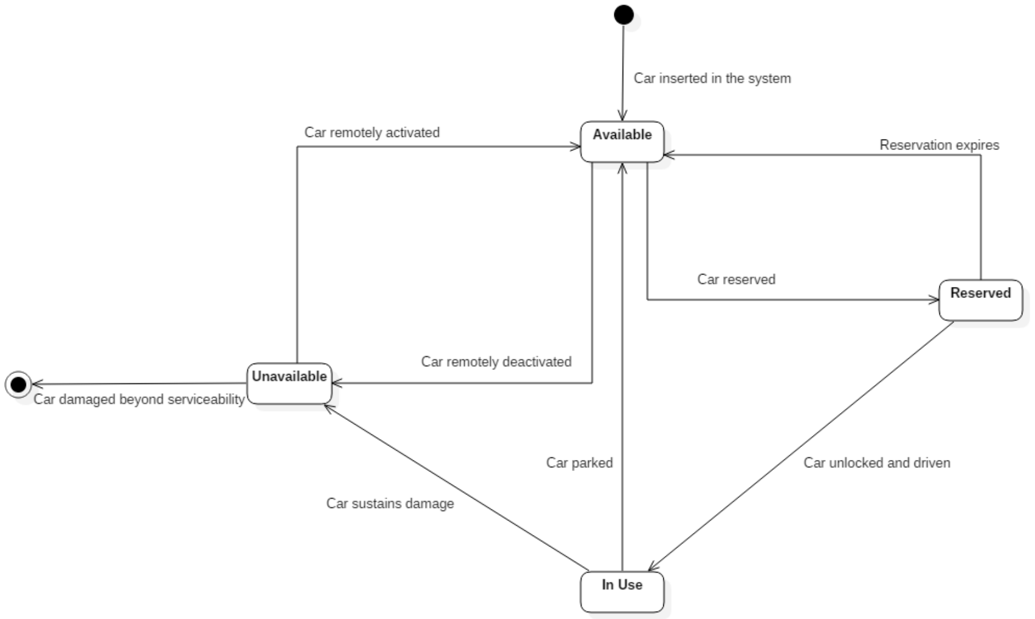


Figure 3: Statechart diagram of Car

2.2 Product Functions

Using the the Application, the User will be able to register an account in the System, log in the System and so finally access the System functionalities dedicated to users. The User can now locate all the Cars, whose state is *Available*, specifying a desired address or asking the System to locate him/her through the GPS coordinates.
The Mapping System is now asked to check for the existence of the specified address or locate the User's position, showing on the map all the *Available* Cars, together with their intrinsic information, in a distance range specified by the System.

The User can now decide to reserve a Car, from this moment a one hour countdown starts during which the User will be able to unlock his/her reserved Car. The User can unlock the Car asking the System to locate him/her and if the Mapping System 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. 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.

At the end of the ride, the System, basing on the time usage period of the Car, and on bad or good behaviors preset by the System 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 and Administrators 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 Application. In addition, in order to perform the localization, reservation and unlocking of a Car the device must have installed a working GPS module.

2.4.2 Design and Implementation Constraints

The system will employ the HTTP(S) protocol in order to communicate with the Database, the external systems, and with the User through the Application.

2.5 Assumptions and Dependencies

1. The User can only have one Account at time.

2. The Company can decide at any time to block an User from accessing to the System (f.e. for improper behavior, unpaid bill, ...). It will be done by employees or Administrators.

3. The User always provides real correct data in his/her registration form.

4. The Database in which the Cars, Parking Areas, Charging Areas, Users,etc, are stored, is owned and managed from 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 this Database.

5. The Company is responsible for the employees and their actions.

6. 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. We assume that these sensors won't ever break down and that their measures are always correct.

7. The Car GPS always detects its position with absolute precision.

8. The User always enters the Car when he/she unlocks its doors.

9. After a Car is Plugged, it will not be maliciously unplugged by the User himself/herself or by other people.

10. 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.

11. An User can park/stop the Car everywhere and leave the Car at anytime. However, the system will end the ride (i.e. stop charging the User) only if he/she turns the engine off inside a Parking Area.

12. When the User gets at least two Passengers, the corresponding discount on the User's fee will be applied only if the passengers stay together in the Car for more than 3 minutes.

13. When a User will park the Car inside a Parking 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 and is immediately set as *Unavailable*. If the Car status is *Unavailable*, the Car will be reached by an Employee to consider if the Car needs to be taken in the Company's workshop for repairs or just needs to be recharged.

16. The Car has the ability to detect if it has been damaged.

17. If the Car status is *In Use* when a *Minor damage* is detected, the Car status will be set to *Unavailable* at the end of the ride; if a *Major damage* is detected the Car status is immediately set to *Unavailable*. In both cases an employee will reach the car and cope with the damages, deciding if the Car can be immediately used again (sets it to *Available*) or should be moved to the Company's workshop and/or if the User should pay for the damages.

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. 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.

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

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 comunication interfaces.

3.1.1 Software Interface

The Application has to communicate with the GPS module in order to get the GPS position of the User.

The Application has to communicate with the Database

3.1.2 User Interface

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

A generic Visitor of the Application should 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 th e 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.

3.1.3 Hardware Interface

The central server must be provided with one or more sufficiently advanced computers that may run the server-side application, and allow it access to a high-speed network connection.

3.1.4 Communication Interface

As mentioned above, the System heavily uses Internet communications protocols, mainly the HTTP protocol. HTTP requests to and from the server will be mostly carried by mobile network connections.

3.2 Functional Requirements

The functionality for the various users.

3.2.1 Requirements List

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

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

R3 PowerEnjoy shall support Users in locating Parking Areas and their free parking slots.

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

R5 PowerEnjoy shall support Users in reserve Available Cars.

R6 PowerEnjoy 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 PowerEnjoy 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 PowerEnjoy 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 PowerEnjoy 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, PowerEnjoy should apply a percentage Discount of 10% on the final Fee of the last ride.

R10 PowerEnjoy shall allow the User to end the ride in a Parking or Charging Area.

R11 PowerEnjoy 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 PowerEnjoy 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 PowerEnjoy 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 PowerEnjoy 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 PowerEnjoy shall interface with an external Mailing System to send emails to Users.

R16 PowerEnjoy shall interface with an external Banking System to charge Fee to Users.

R17 PowerEnjoy shall interface with an external GPS System to know the positions of Users and Areas.

R18 PowerEnjoy shall interface with an external Mapping System to know the positions of Users and Areas.

R19 PowerEnjoy shall interface with the existing Car to get their GPS position, damages, connection to an electrical socket, the number of seats occupied.

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*** connects to the ***Company's Car-Sharing*** Application.

Flow of events	1. The ***Visitor*** selects the function *“Sign Up”*. 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. 3. The ***Visitor*** fills the form with all the required information. 4. The ***System*** stores the request together with all the data provided with it, generates a random activation URL and asks the ***Mail System*** to forward his/her URL to the email address of the ***Visitor***.
Post Conditions	The ***Mail 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. • 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 saying that some of those credentials were already been inserted into the System for another account. 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. 2. The ***System*** acknowledges that the Visitor has arrived in his/her activation Web Page and activates his/her account.
Post Conditions	The ***Visitor*** is now become an ***User*** which can access the ***System*** using the credentials (Email, password) he provided during the registration phase.
Exceptions	• The Activation URL expires after 10 days 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*** connects to the Company's ***Car-Sharing Application***. The ***Visitor*** has already activated his/her account (UC2)

Flow of events	1. The ***Visitor*** selects the function *“Login”* . 2. The ***System*** shows the ***Visitor*** a login form, asking him to insert the email and password provided in the registration form. 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 that pair (Email,password) and logs the ***Visitor*** in. The ***Visitor*** has now become ***User***
Exceptions	• The System doesn’t find an account associated with that pair (Email, Password) and shows an error message, the flow of events starts 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”* . 2. The ***System*** performs the ***User***’s logout.

Post Conditions	The ***System*** shows the confirmation of the logout to the ***User***. 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”* .
Post Conditions	The ***System*** shows the ***User*** a map with all the ***Parking Areas*** distributed around the city.
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"*. 2. The ***System*** shows a text box asking the ***User*** to provide an address near which they would like to see the ***Cars*** whose state is *Available*. 3. The ***User*** inserts the desired address and selects the *"Locate"* function.
Post Conditions	The ***System*** shows the ***User*** a map containing all the ***Cars*** whose state is Available and which are within a 5KM range from the provided address.
Alternative Flow of Events	The ***User*** selects the function *"Near Me"* instead of inserting a specific address and sends their ***GPS Coordinates*** to the ***System***.
Exceptions	• The System does not find the inserted address and informs the User. The Flow of Events starts from point 1. • 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. 2. The ***User*** selects the function *“Reserve this Car”*.
Post Conditions	The ***System*** stores the ***Reservation*** of the ***Car***, changing its status to ***Reserved***. The ***System*** activates a countdown of 1 hour during which the ***User*** will have the possibility to unlock the ***Reserved Car***.
Exceptions	

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”*. 2. The ***User*** sends their GPS coordinates to the ***System***. 3. The ***System*** checks that the GPS coordinates of the ***User*** are within a 15 metres range from those of the ***Car*** itself.
Post Conditions	The ***System*** has verified that the ***User*** is nearby the car (within the specified distance range) and unlocks the ***Car***’s doors. The ***System*** then changes the ***Car*** status to ***In Use*** and sets the ***Plugged*** Flag to False. The ***User*** enters the ***Car***.

Exceptions	If one hour has passed since the reservation has been made and the ***User*** hasn't unlocked the Car, either because he wasn't within the 15 meters distance range or didn't activate this function, then: • The reservation expires, so that the ***User*** cannot unlock the ***Car*** anymore (unless they reserve it again). • The ***System*** changes the ***Car***'s status to ***Available***. • The ***System*** communicates to the ***Banking System*** the ***Fee*** to charge the ***User*** (this sum amounts to 1 EUR). • The ***System*** allows the ***User*** to perform another reservation.

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***. 2. The ***System*** starts the Ride Timer which indicates the time usage of the ***Car***. 3. [Extension Point UC 10] 4. [Extension Point UC 13] 5. The ***System*** calculates the current ***Fee*** charged to the ***User*** (calculated as a given amount of money per minute on the Ride Timer) while showing it on the onboard screen.
Post Conditions	The ***User*** drives the ***Car***
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 their ***Reserved Car***.
Flow of events	1. The ***User*** picks up the ***Passengers***. 2. The ***Car*** detects the presence and number of the ***Passengers***.
Post Conditions	The ***User*** is sharing the ride with their ***Passengers***. The ***System*** stores the number of ***Passengers*** who were picked up and whether they stayed in the ***Car*** for at least 3 minutes.
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 one of the ***Parking Areas***.
Flow of events	1. The ***User*** exits the ***Car***. 2. The ***System*** verifies that no one is in the ***Car***. 3. The ***System*** checks the ***Battery*** status. 4. The ***System*** checks, via the GPS coordinates, whether the ***User*** has left the ***Car*** within a 3KM distance range from the nearest ***Charging Area***. 5. The ***System*** checks if the ***User*** drove with ***Passengers*** (UC10). 6. [Extension Point UC12].
Post Conditions	The ***System*** locks the doors of the ***Car*** and sets its status to ***Available***. The ***System*** communicates to the ***Banking System*** the final ***Fee*** to charge the ***User***.

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 3KM 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***. • 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***. • 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***. • 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 3KM 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 3KM 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 3KM 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 3KM 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 3KM 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 3KM 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***. • 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 3KM 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	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***.

Flow of events	1. The ***User*** plugs the ***Car*** into a ***Socket*** of the ***Charging Area***. 2. The ***System*** detects that the ***Car*** has been plugged within 2 minutes since the ***User*** got off the ***Car***.
Post Conditions	The ***Battery*** of the ***Car*** is charging and the ***System*** remembers the ***User***'s action for possible discounts. The ***System*** sets the ***Car***'s ***Plugged*** flag to True.
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. 2. The ***User*** provides the address to the ***System***. 3. The ***System*** computes 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***.
Post Conditions	The result of this algorithm will be sent to the ***User***, providing him the address of the ***Charging Area*** where to leave the ***Car***. The ***User*** will still have to plug the ***Car*** 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

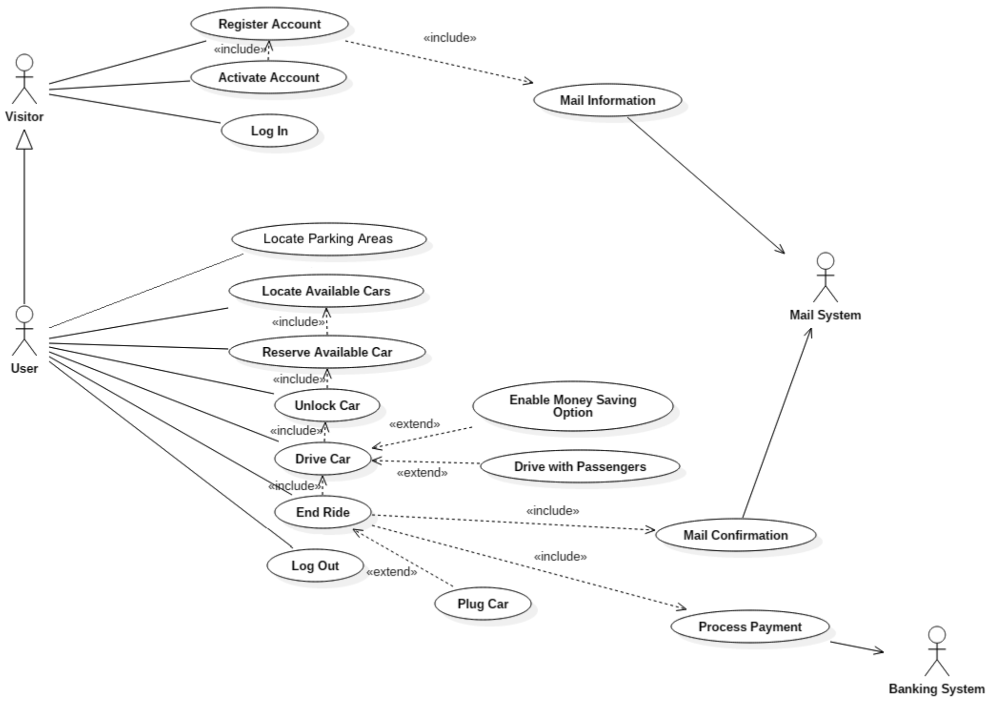


Figure 4: Use Case

3.2.4 Activity Diagrams

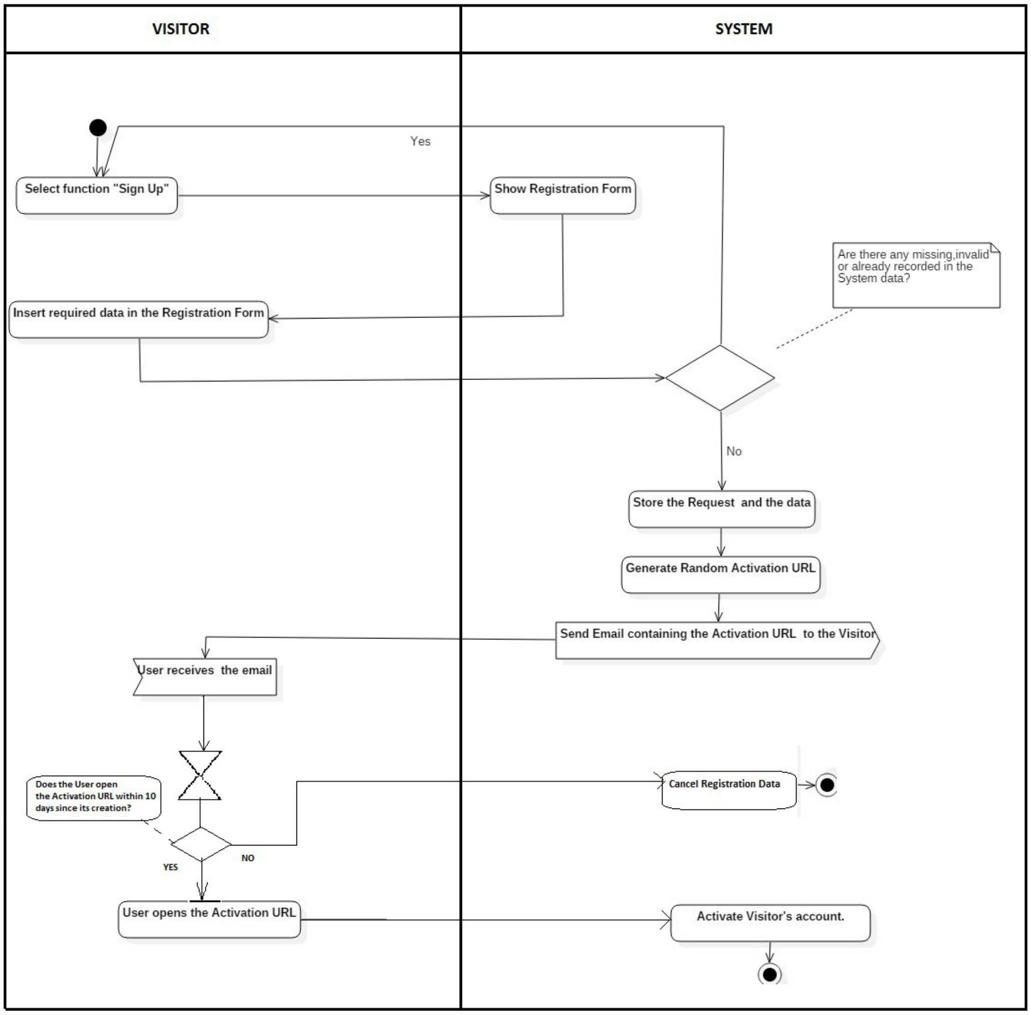

VISITOR
SYSTEM
Yes
Select function "Sign Up"
Show Registration Form
Are there any missing,invalid or already recorded in the System data?
Insert required data in the Registration Form
No
Store the Request and the data
Generate Random Activation URL
Send Email containing the Activation URL to the Visitor
User receives the email
Does the User open the Activation URL within 10 days since its creation?
Cancel Registration Data
NO
YES
User opens the Activation URL
Activate Visitor's account.

Figure 5: Registration flowchart

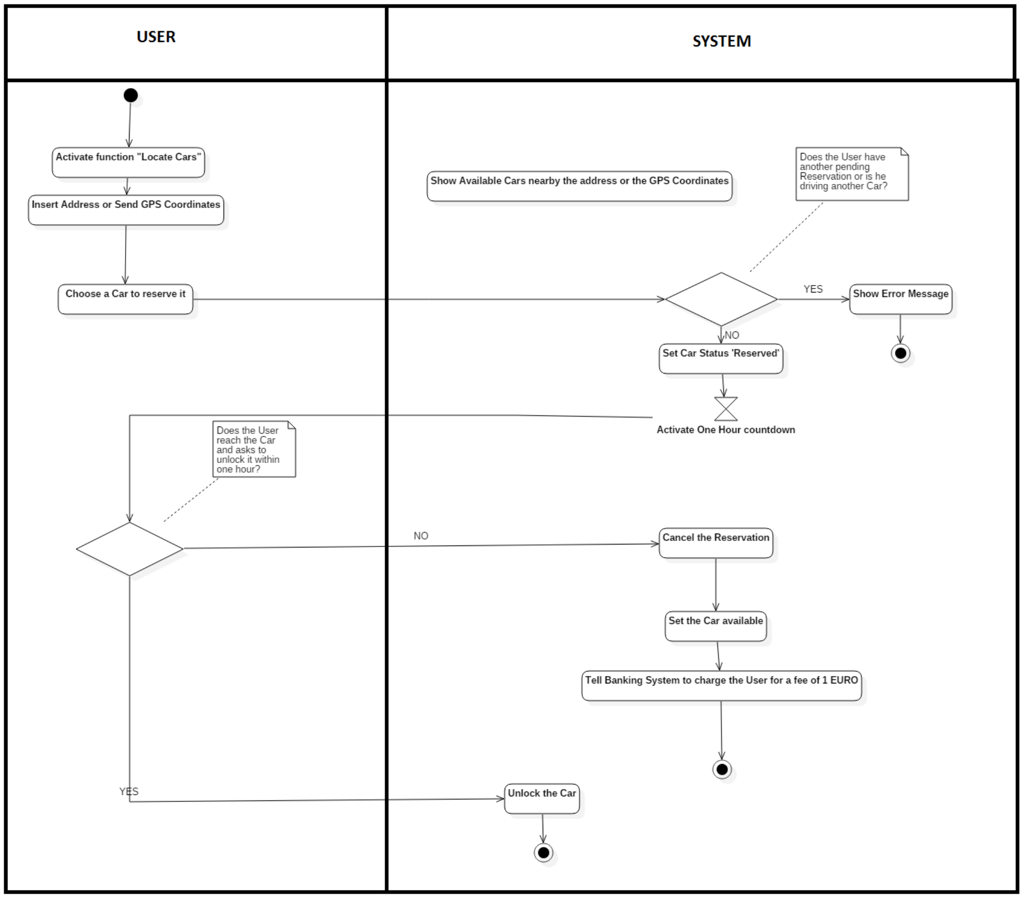

USER
SYSTEM
Activate function "Locate Cars"
Insert Address or Send GPS Coordinates
Choose a Car to reserve it
Show Available Cars nearby the address or the GPS Coordinates
Does the User have another pending Reservation or is he driving another Car?
YES
Show Error Message
NO
Set Car Status 'Reserved'
Activate One Hour countdown
Does the User reach the Car and asks to unlock it within one hour?
NO
Cancel the Reservation
Set the Car available
Tell Banking System to charge the User for a fee of 1 EURO
YES
Unlock the Car

Figure 6: Reservation flowchart

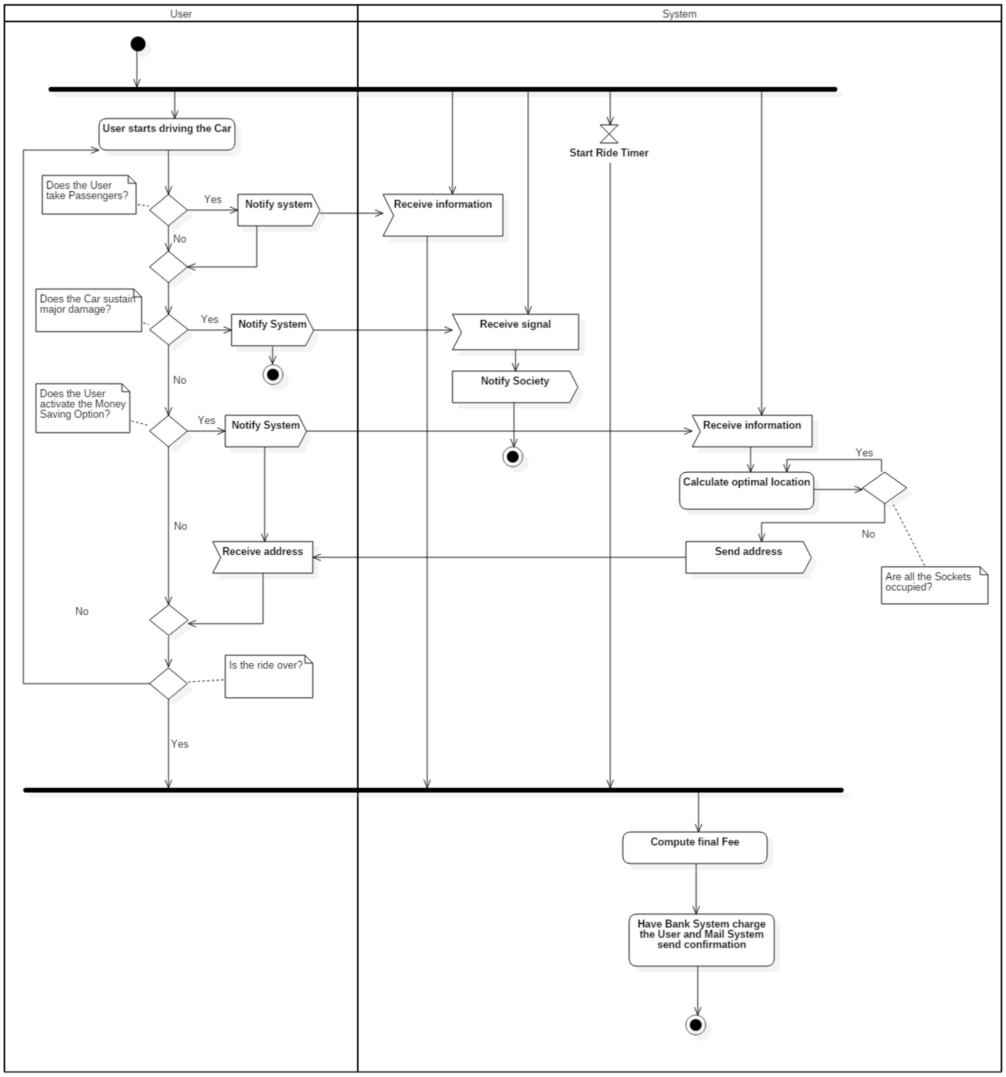

User
System
User starts driving the Car
Start Ride Timer
Does the User take Passengers?
Yes
Notify system
Receive information
No
Does the Car sustain major damage?
Yes
Notify System
Receive signal
Notify Society
No
Does the User activate the Money Saving Option?
Yes
Notify System
Receive information
Yes
Calculate optimal location
No
No
Receive address
Send address
Are all the Sockets occupied?
No
Is the ride over?
Yes
Compute final Fee
Have Bank System charge the User and Mail System send confirmation

Figure 7: Ride flowchart

3.2.5 Sequence Diagrams

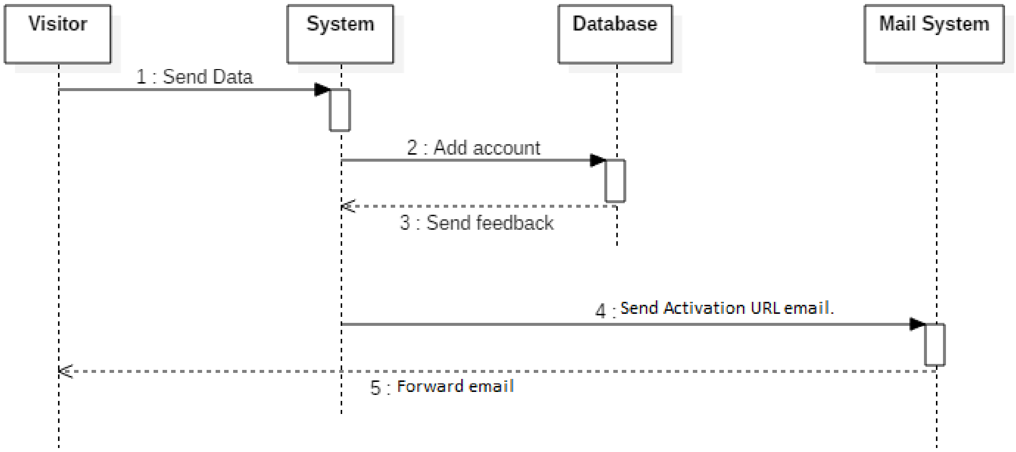


Figure 8: Use Case 1

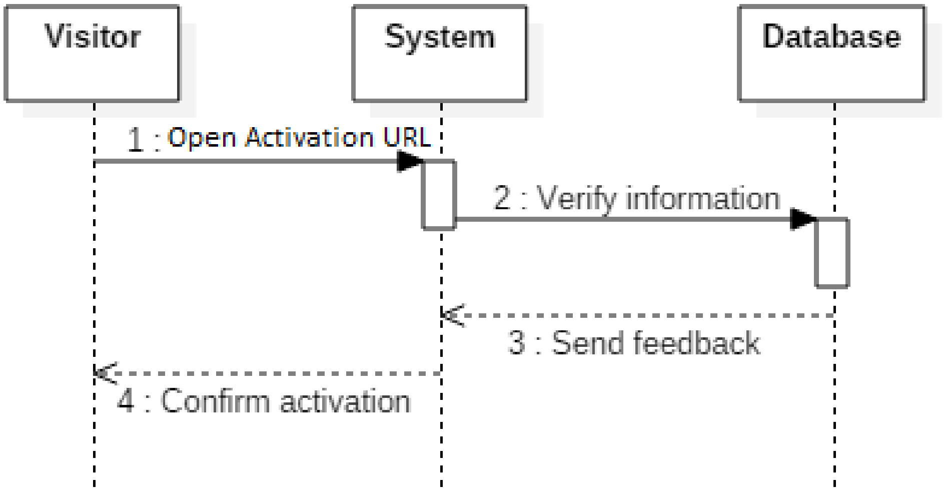


Figure 9: Use Case 2

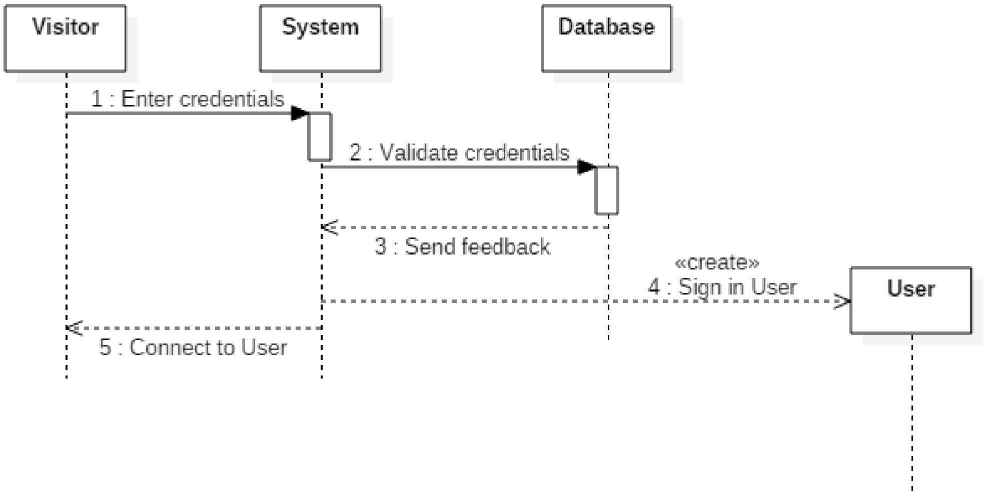


Figure 10: Use Case 3

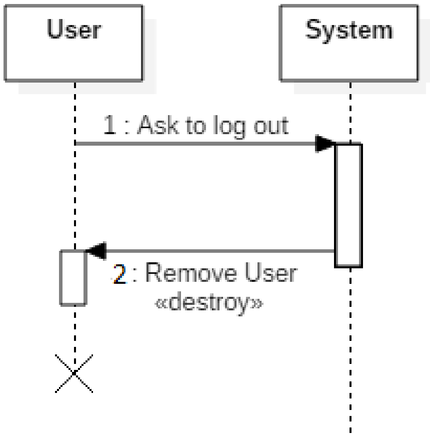


Figure 11: Use Case 4

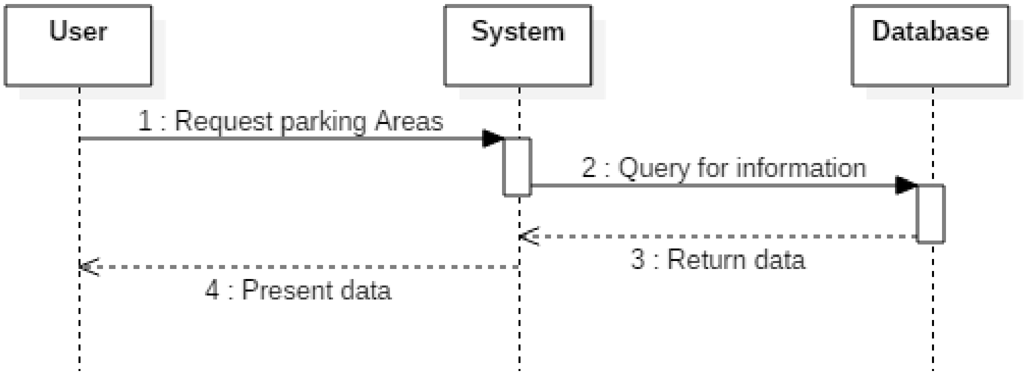


Figure 12: Use Case 5

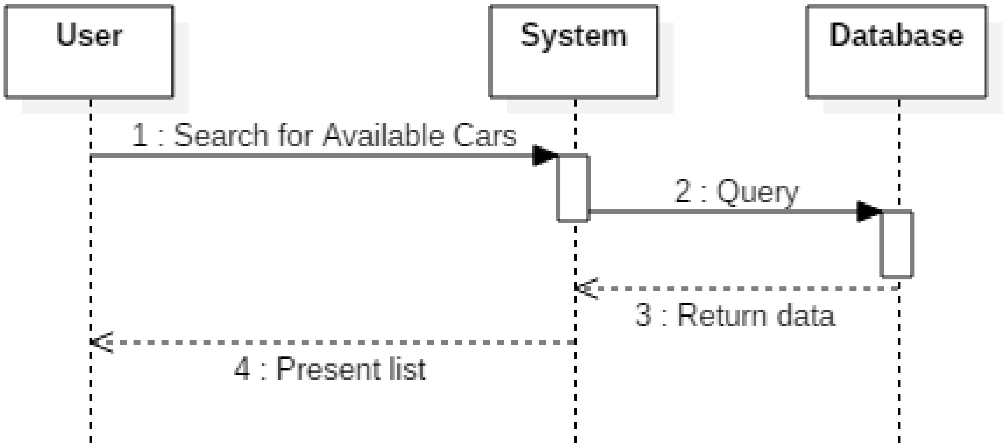


Figure 13: Use Case 6

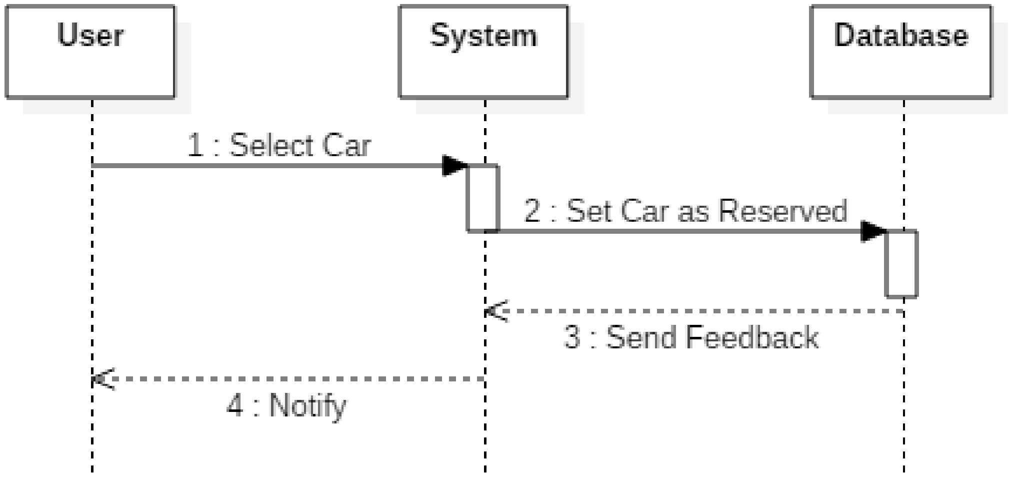


Figure 14: Use Case 7

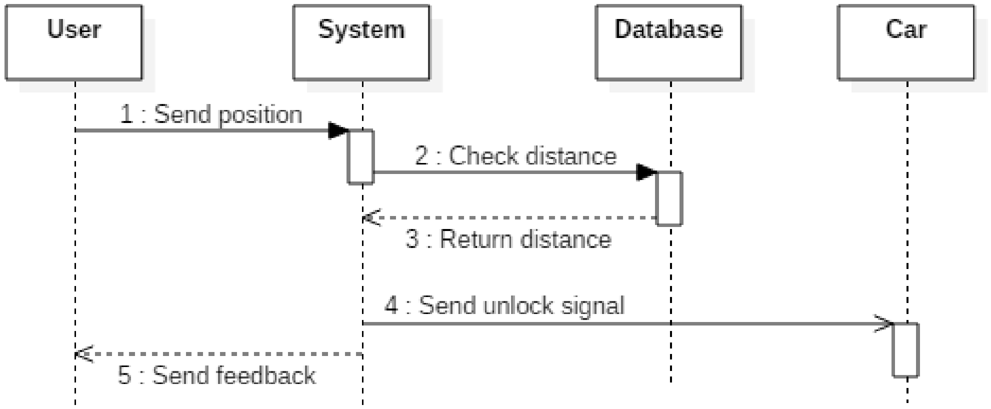


Figure 15: Use Case 8

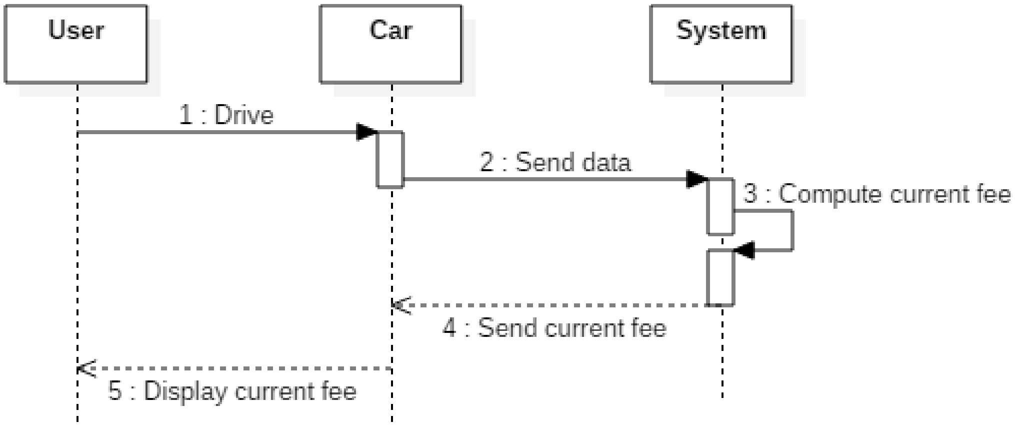


Figure 16: Use Case 9

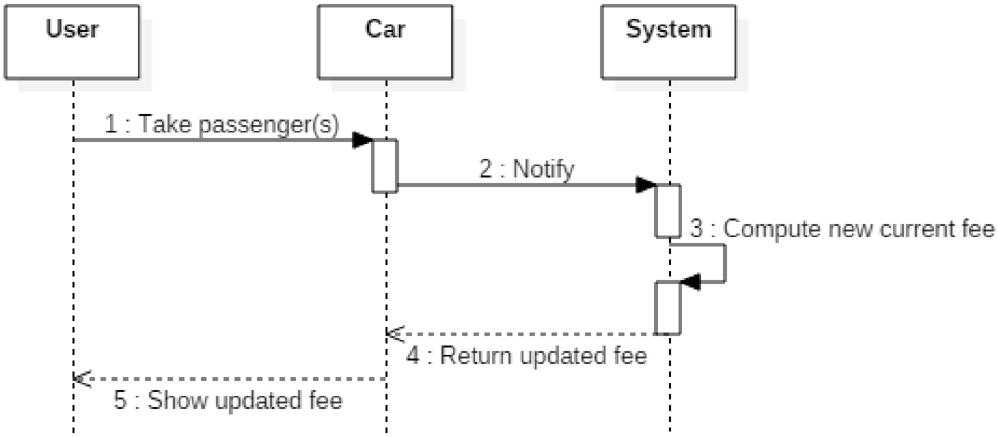


Figure 17: Use Case 10

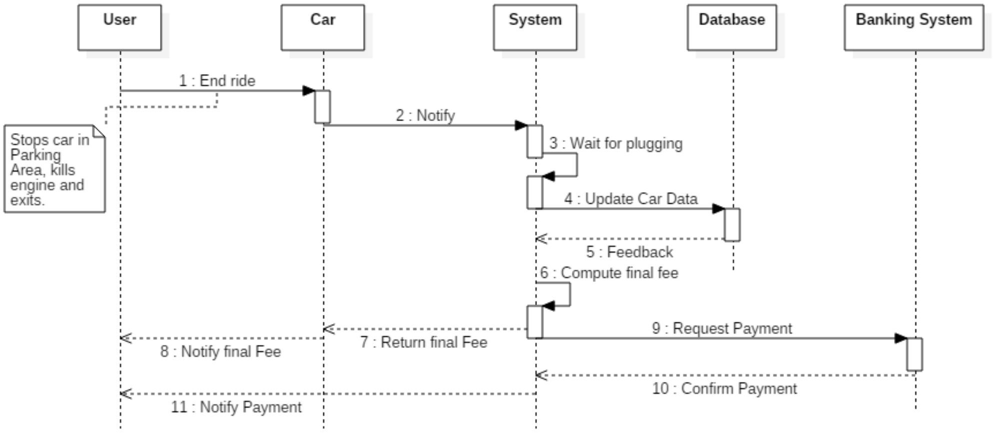


Figure 18: Use Case 11

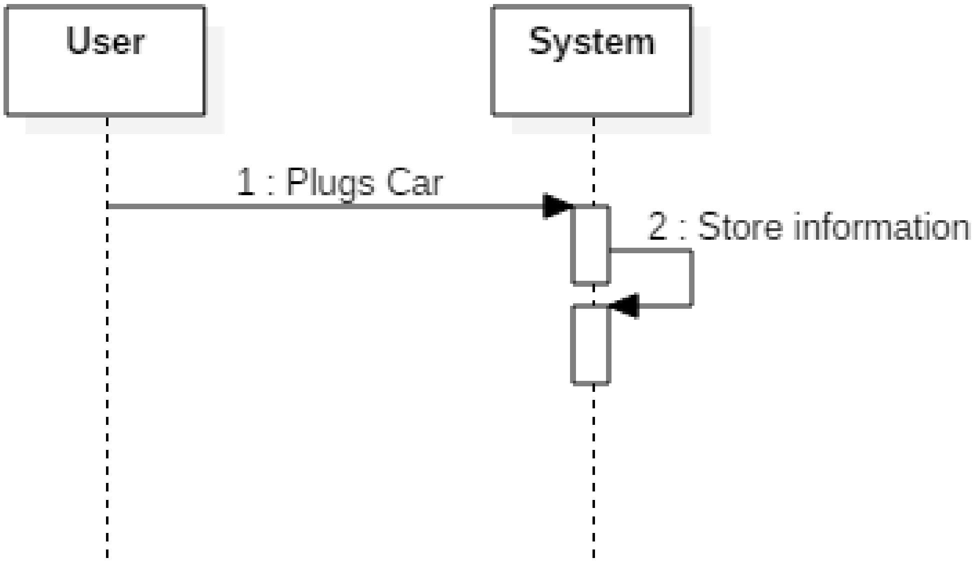


Figure 19: Use Case 12

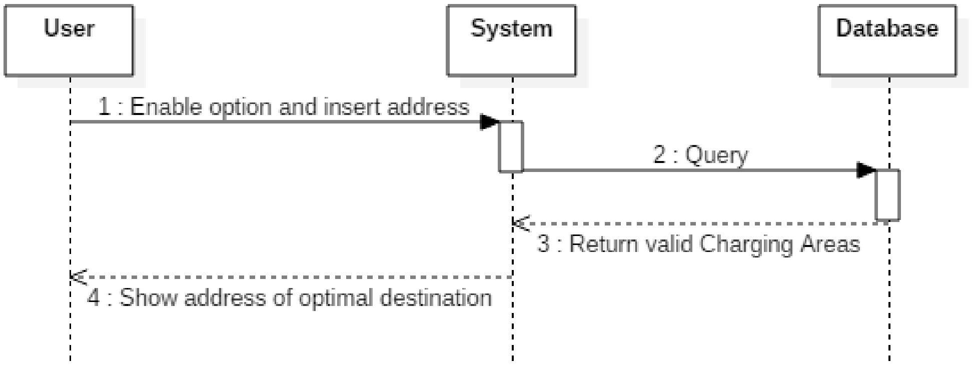

User
System
Database
1 : Enable option and insert address
2 : Query
3 : Return valid Charging Areas
4 : Show address of optimal destination

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 Safety and Security Requirements

There will not exist cases in which sensible data belonging to the User, Visitor, Company, will be passed on a insecure channel.

Stakeholders did not ask for any special requirement.

3.3.3 Portability

Stakeholders did not ask for any special requirement.

3.3.4 Availability and Reliability

The Server on which the System runs must always be reachable, in addition it must provide a relaiability and availability of 99.9The DataBase used by the System must be always must always be reachable, in addition it must provide a reliability and availability of 99.9

Stakeholders did not ask for any special requirement.

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

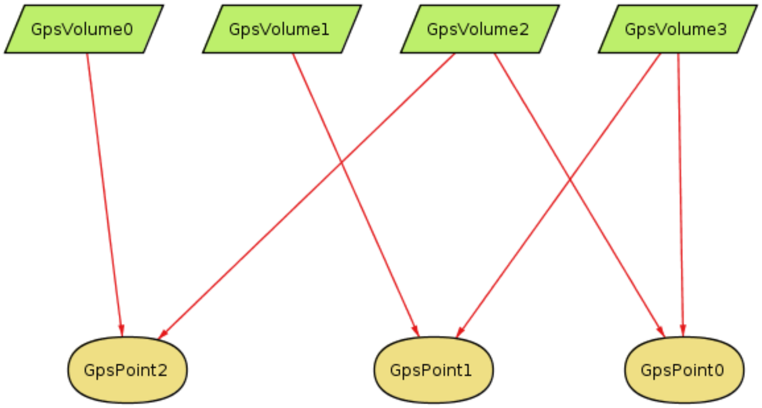


Figure 21: A Gps World

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.

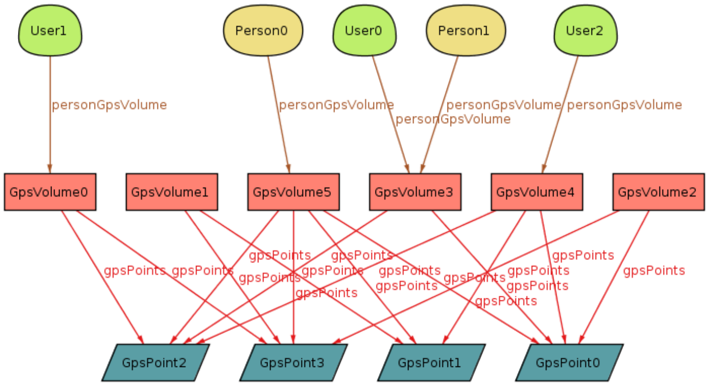


Figure 22: A Persons World

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.

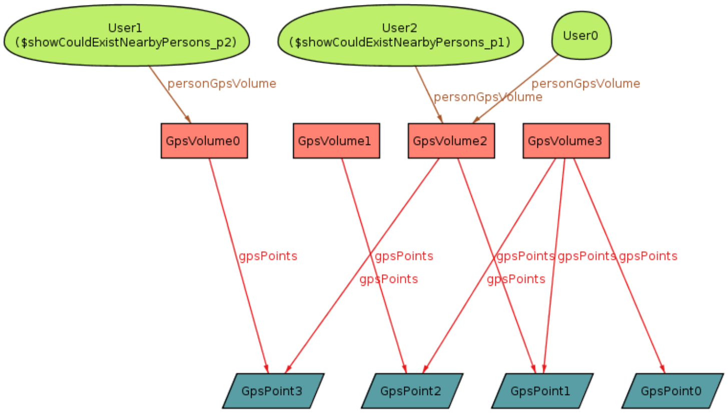


Figure 23: Nearby Persons

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).

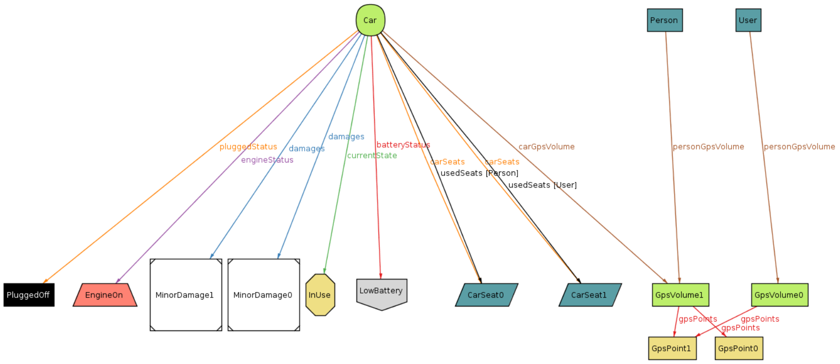
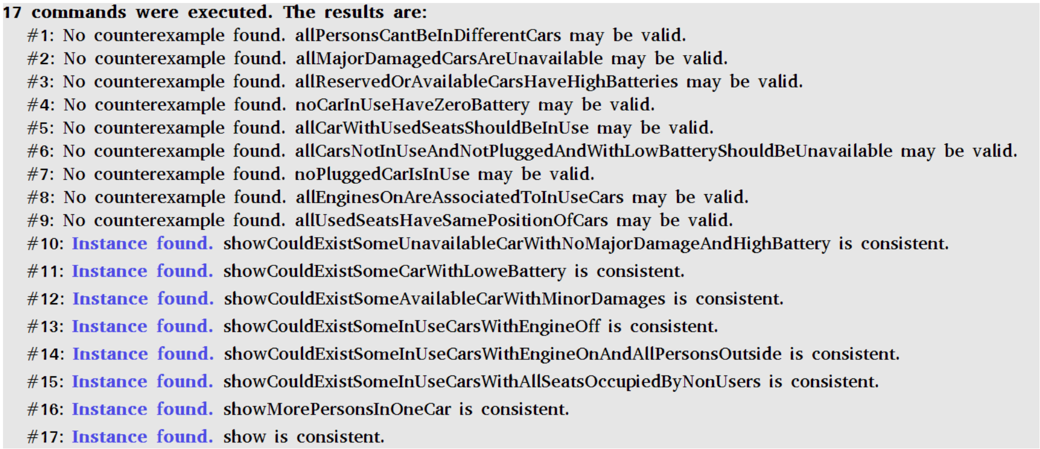


Figure 24: A Cars World

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

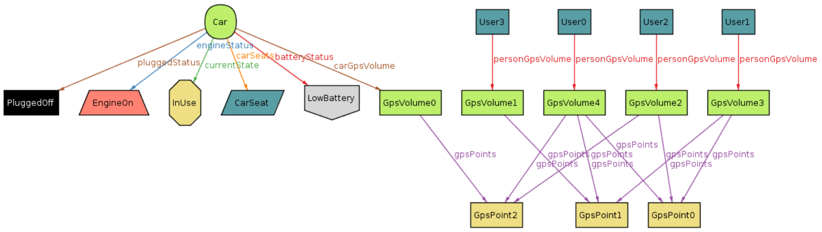


Figure 26: Used cars with no person inside

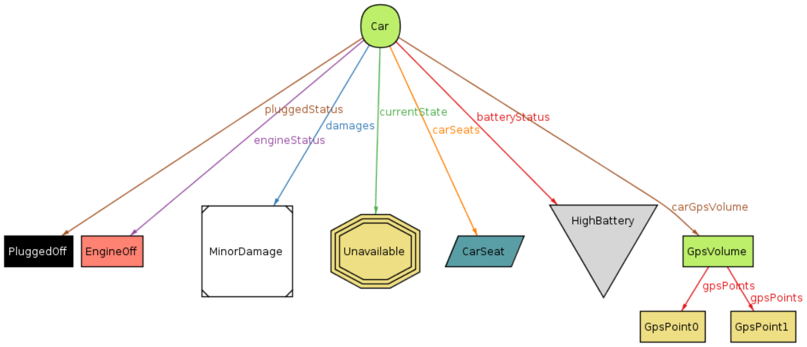


Figure 27: Unavailable functioning cars

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.

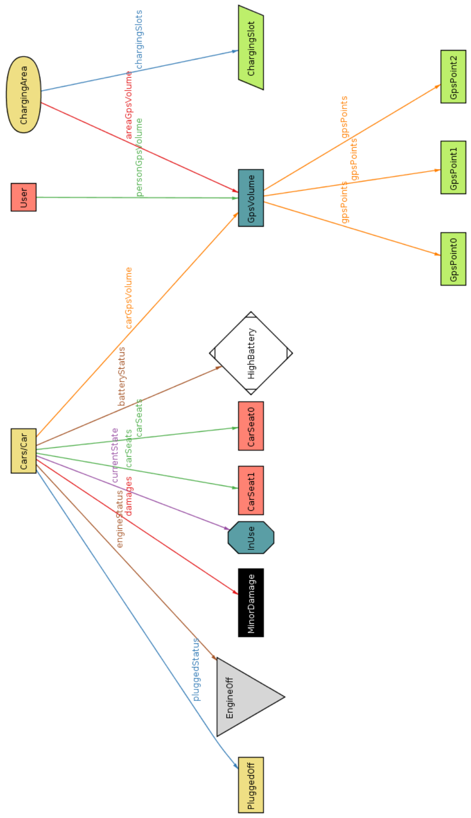


Figure 28: An Areas World

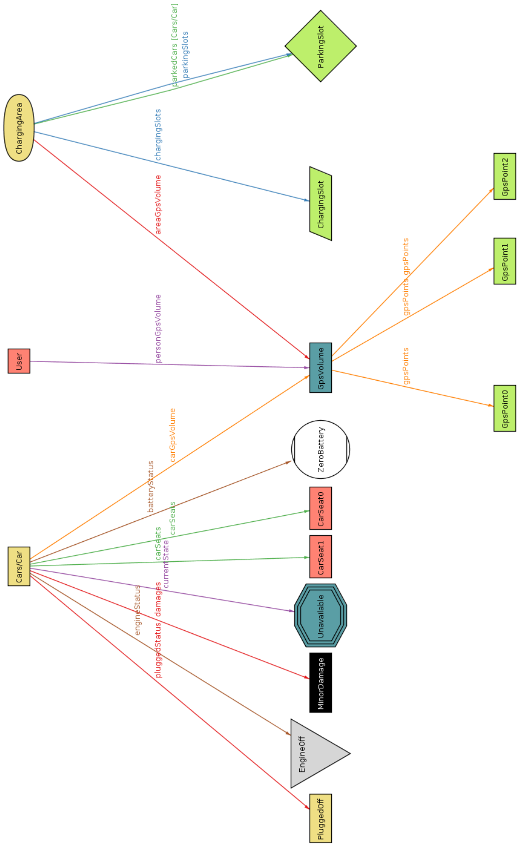


Figure 29: Another Areas World

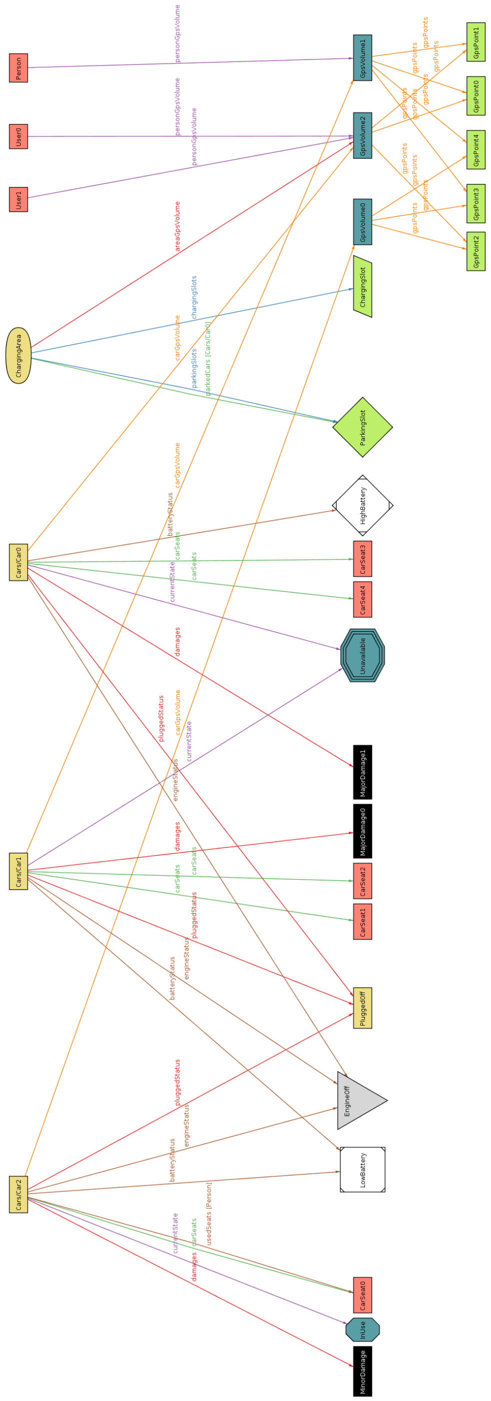


Figure 30: A more complicated Areas World

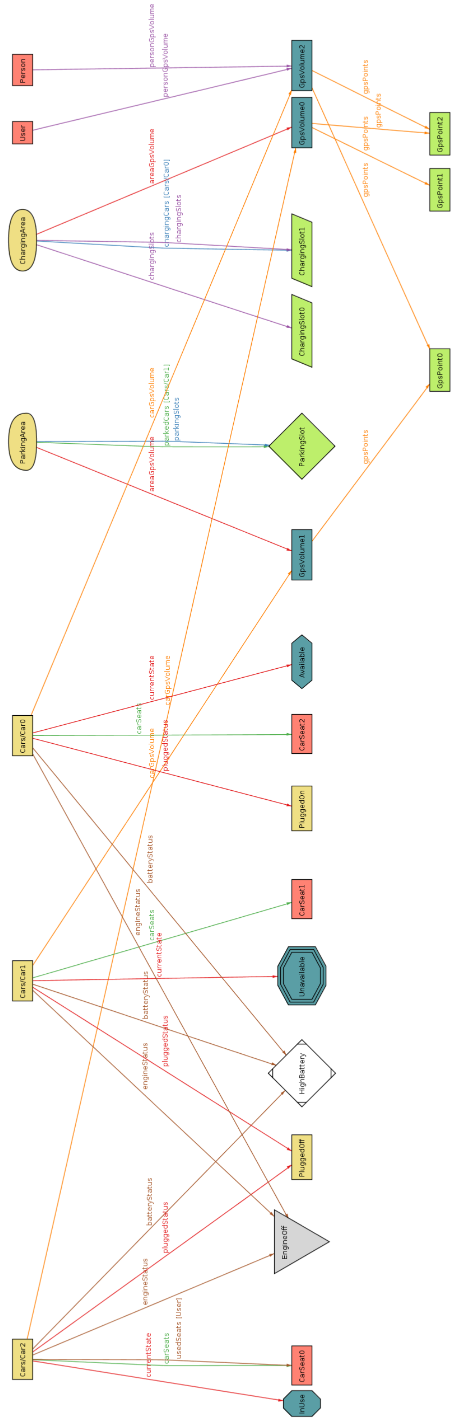
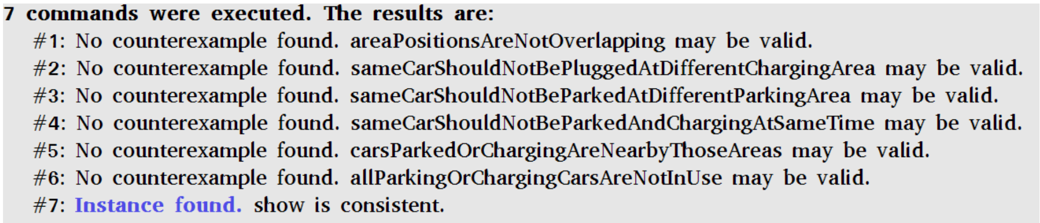


Figure 31: Another more complicated Areas World



```
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 Working Hours

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

4.2.1 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

4.2.2 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

4.2.3 Enrico Migliorini

50 total hours.