***PowerEnJoy***



**Requirement Analysis**

**And**

**Specification Document**

Authors:

Emanuele Chilà

Giorgio Lazzarinetti

November 13th 2016

SUMMARY

[1. INTRODUCTION 4](#_Toc468196328)

[1.1 PURPOSE 4](#_Toc468196329)

[1.2 DESCRIPTION OF THE GIVEN PROBLEM 4](#_Toc468196330)

[1.3 GOALS 5](#_Toc468196331)

[1.4 DOMAIN PROPERTIES 5](#_Toc468196332)

[1.5 OTHER ASSUMPTIONS 5](#_Toc468196333)

[1.6 GLOSSARY 6](#_Toc468196334)

[1.7 STAKEHOLDER IDENTIFICATION 6](#_Toc468196335)

[2. ACTORS IDENTIFICATION 7](#_Toc468196336)

[3. REQUIREMENTS IDENTIFICATION 7](#_Toc468196337)

[3.1 FUNCTIONAL REQUIREMENTS 7](#_Toc468196338)

[3.2 NON FUNCTIONAL REQUIREMENTS 8](#_Toc468196339)

[4. SCENARIOS IDENTIFICATION 9](#_Toc468196340)

[5. UML MODELS 11](#_Toc468196341)

[5.1 USE CASE DIAGRAMS AND DESCRIPTIONS 11](#_Toc468196342)

[5.1.1 BASIC FUNCTIONALITIES 12](#_Toc468196343)

[5.1.2 CAR MANAGEMENT 14](#_Toc468196344)

[5.1.3 EXTRAORDINARY SITUATION MANAGEMENT 21](#_Toc468196345)

[5.1.4 NOTIFICATION MANAGEMENT 23](#_Toc468196346)

[5.2 CLASS DIAGRAM 27](#_Toc468196347)

[5.3 SEQUENCE DIAGRAMS 28](#_Toc468196348)

[5.3.1 SIGN UP 28](#_Toc468196349)

[5.3.2 LOG IN 30](#_Toc468196350)

[5.3.3 SEARCH FOR AVAILABLE CARS 31](#_Toc468196351)

[5.3.3.1 USING YOUR POSITION 31](#_Toc468196352)

[5.3.3.2 INSERTING A POSITION 32](#_Toc468196353)

[5.3.4 RENT A CAR 33](#_Toc468196354)

[5.3.5 LOCK\_UNLOCK OPTION 34](#_Toc468196355)

[5.3.5.1 UNLOCK FOR THE FIRST TIME 34](#_Toc468196356)

[5.3.5.2 LOCK IN ORDER TO FINISH THE RUN 35](#_Toc468196357)

[5.3.6 FORWARD PAYMENT TO THE BANK 36](#_Toc468196358)

[5.4 ACTIVITY DIAGRAMS 37](#_Toc468196359)

[5.4.1 CAR ACCIDENT and BATTERY LOW 37](#_Toc468196360)

[5.4.2 RENT A CAR 38](#_Toc468196361)

[6. ALLOY MODELING 39](#_Toc468196362)

[6.1 CODE 39](#_Toc468196363)

[6.2 GENERATED WORLDS 47](#_Toc468196364)

[7. USED TOOLS 53](#_Toc468196365)

[8. TIME SPENT 53](#_Toc468196366)

## 1. INTRODUCTION

### 1.1 PURPOSE

The RASD document is intended for describing the general functionality of the PowerEnJoy project assigned as the project of the Software Engineering course.

The intended audience is the people that are participating in the Software Engineering course.

The persons that participate actively in the course are: the professors, professor assistants and the different groups taking the current version of the course.

### 1.2 DESCRIPTION OF THE GIVEN PROBLEM

The software product that is going to be developed is PowerEnJoy, a new vehicle-sharing service application.

The product has to allow users, once registered in the system, to log in, locate and rent an electric car. The system should also manage the calculation of the cost of the service benefited by the user and the billing procedures.

### 1.3 GOALS

The software will satisfy the following main goals:

* Allow users to search for available car;
* Allow users to reserve a single car;
* Registration and login of a person to the system;
* Correctly charge a user that uses the service;
* Make sure that the user behaves well;

### 1.4 DOMAIN PROPERTIES

We assume that the following facts hold in the world in which PowerEnJoy will operate:

* The cars are always traceable via GPS;
* The system identifies the location of the cars in the most possible accurate way;
* The only payment option in order to use the service is the Credit Card;
* The sensors in the car such as weight detector, seat belts, battery charge, odometer are connected to a terminal installed on each car;
* The system uses Goggle Map API in order to locate the position of the cars giving it a meaning in terms of street name and house number.

### 1.5 OTHER ASSUMPTIONS

There are some points that are not very clear in the specification document, so we will assume the following facts:

* A user that has an unsolved payment can’t use the service again until he has fulfilled the last payment request;
* For “safe areas” we mean areas owned by PowerEnJoy and equipped to park cars and recharge them;
* A user may signal the car to open and close at will until it is reserved by him and when his position is identified by the system to be within 50 m from the car;
* The maximum distance from a certain position within the system has to find the available cars is defined by the user;
* In case of accident PowerEnJoy provides a toll free number that handles the accident. An operator of the call center will charge the cost due to the accident at the user;
* A user that has rented a car cannot cancel the reservation made.
* If there are more discounts applicable they are calculated on the discounted previous cost of the trip following the order of increasing discounted rate.
* The discounts are applied before the further charges.
* The user can’t use the rented car for more than 24h: the system blocks the car if that happens.
* All the users of the cars are supposed to own a valid driving license and they are directly responsible if they drive without it.

### 1.6 GLOSSARY

Here are some words we will frequently use in the following pages of this document. In order to make things clear, we now specify their meanings:

* **User:** a user is a person who has filled in a registration form with his/her credentials (i.e., e-mail, password, nickname etc) and a valid Credit Card. Once registered, a user can benefit from the services offered by the platform.
* **Car**: If it is a PowerEnJoy car is implied that this car is an electric one.
* **Reservation**: It’s a booking made by a user and paid to the use of a certain selected car.
* **Safe Area**: We mean the areas where the user can park the rented car
* **Special Parking Area**: We mean the areas where the user can both park the rented car and recharge it (all the special parking area are safe area, but not viceversa);

### 1.7 STAKEHOLDER IDENTIFICATION

The intended stakeholders are the professors and the assistants who gave us this didactical project.

Our scope is to show that we have understood the development process in all its different parts and that we can carry out a project from the beginning to the end passing from all its development phases, providing all the documents that developers use in developing real software.

Our application is intended to satisfy the need of a person that want to benefit on the car sharing services provided by PowerEnJoy.

## 2. ACTORS IDENTIFICATION

We have identified three main actors of the platform:

* **VISITOR:** A visitor is person not registered into PowerEnJoy yet.

A visitor can:

* + - Sign up into the platform;
* **USER**: A user is a person who is registered to the system.

A user can:

* + - Log into the platform and log out of the platform;
    - Search for available car in a certain distance from a given point (i.e. own position, house number etc);
    - Make a reservation for a selected car;
    - Open and close the reserved car at will;
    - Use and then park the car in a safe zone;
* **CALL CENTER:** Isthe infrastructure that the user has to call in case of accident or discharge of the battery.

The call center can:

* + - Change the status of a car;
    - Manage the accident;

## 3. REQUIREMENTS IDENTIFICATION

### 3.1 FUNCTIONAL REQUIREMENTS

Supposing that all the domain properties in the paragraph 1.4 hold, here we list the functional requirements needed to ensure the satisfaction of the goals in the paragraph 1.3; the requirements are sorted by the actors of the system.

* **VISITOR:** 
  + **Registration of a person into the system:** 
    - The system has to provide a signup functionality.
    - The system has to provide a Credit Card check of validity.

* **USER:** 
  + **Log of a person into the system:** 
    - The system has to provide a login functionality.
  + **Allow Users to search for available cars:** 
    - The system has to provide localization functionalities.
    - The system has to provide functionalities to keep track about the status of each car.
    - The system has to provide functionalities in order to interact with the Google Map service.
  + **Allow Users to reserve a single car:** 
    - The system must keep track about the user and the car status.
    - The system must provide functionalities to handle a reservation, in particular the date and the expiration time.
    - The system has to provide functionalities in order to remotely control a certain car in particular opening/locking the doors.
  + **Correctly charge a user that uses the service:**
    - The system has to provide functionalities in order to remotely acquire information from the car’s sensor so the system can define how much the user has to pay and if the user is or not entitled to discounts.
    - The system has to manage the payments records solved and unsolved and be able to charge the cost of the trip to the user credit card.

* **USER and CALL CENTER** 
  + **Make sure that the user behaves well:**
    - The system has to provide functionalities in order to charge more the user if he doesn’t park the car in a safe zone or doesn’t use the car in the way it is supposed to be used.
    - The system has to manage extraordinary event in particular accident, car maintenance and recovery.

### 3.2 NON FUNCTIONAL REQUIREMENTS

The system will use an application for the phone users: in this way the application should result more available and user friendly.

The system will also include a web site accessible from any browser in order to let even the non-phone users to use the service.

The login interface will be the same for all the users, and, after having insert his/her own credentials, the user will be allowed to use all the features offered by the system. Using the buttons, user can navigate through the different windows, each one of them dedicated to a specific function.

However, the complete description of the user interface will be specified better in the design document.

## 4. SCENARIOS IDENTIFICATION

Here we describe some possible scenarios of PowerEnJoy:

1. Bob needs a way to travel trough the city quickly and cheaply and so he decides to search Google for “car-sharing service” and, among the voices, he finds PowerEnJoy. He clicks on it and discover a new web platform. In the home page he find an explanation of what this service does and information about how to register. Looking at the reviews left by other users and visitors about this service, he decides to try it. He sees even that he can download the app for mobile and, once he has downloaded it, he registers on the system filling in all the fields required for the registration and receives back a password that can be used to access the system.
2. Some days later Bob, leaving the office, needs to reach the mall to run errands. He decides to try the new service he has registered. So he opens his app on the mobile and asks for an available car. The system asks him if he wants to use his position as point from which research a car or another specified position. He decides to use his position, so the system searches for available cars and in few seconds gives him back a list of available car ordered by increasing distance from him. He selects a car, the systems confirms the reservation communicating the position of the car selected and the remaining time to reach the car. Then he goes at the car.
3. Once he is near the car (within 50 meters) the system provides on the same page in which there is the timer a button to unlock the car. Bob push the button and the car is immediately open and so Bob goes on it and starts using it. Reaching the mall, he can see on the car interface the progressive cost the system will charge on the credit card inserted during the registration. Bob reaches the mall, pushing another time the button (this time to lock) communicates the service to lock the car, runs his errand and then goes back to the car. Bob asks the system to unlock it always with the same procedure and then, as he has finished to use it, decides to go parking it. He decides to park the car in an area, indicated by the system as “special parking areas”, in which he takes car parking the car into a power grid. So, the system applies a discount of 30% on his ride. He parks the car with 38% of the battery.
4. Mary, which is a client of the PowerEnJoy system, has rented a car and is going to get it in order to reach the railway station. However, while she’s going to the car she meets her friend Giulia that asks her to have a coffe together. Since she has 48 minutes to reach the car yet in order not to pay the fee and not to lost the reservation she accepts. She spends time with Giulia till she does not realize she has just 3 minutes to reach the car. She tries to run to the car but she arrives later, so she look to the system which confirms she has lost the reservation and she has to pay the fee of 1€.
5. The next day Mary, after having get her 2 sons from school, still needs to reach the railway station. She has already rent a car and reaches it with her sons in time. She asks the system to unlock it and then they go to the “special parking area” near the station and she plug in the car into the power grid. She parks the car with 63% of battery and so when she arrives the cost of the trip is discounted before of the 10%, than of the 20% and in the end of the 30%. They get out from the car, and finally reach the station on foot.
6. Bob, after having rent a car, needs to go home to his family. Knowing of the discount, he asks two friends of him, who live near him, if they want to go home with him. They accept and so, after works they all go the car. Bob asks the system to unlock it, and so they begin their ride to home. However, while they’re arriving at home, Bob’s wife call him and says she has an emergency for which he has to reaches her immediately. She has her own car, so she says that she will bring all the passengers of the car home after they have reached her. So they all go to Bob’s wife. However when they arrive, there is no “special parking area” and the nearest power grid station is 3,4km far. So they park a car in a safe are which is not special parking area and so the system charges 30% more on the ride.
7. Mary, after having rent a car, go to a grocery store in a shopping center a bit far from where she is, on the way back she manages to park the car in a special area near her home just after receiving from the car display the message of “battery running low” stating that the battery is under 20% of full charge. Mary parks the car and remembers to plug the car into the power grid. The system discount the ride for the 30% rewarding Mary for the attention on plugging the car in the power grid, then charges 30% more on the ride cause the battery charge was too low.
8. Bob is driving distractedly his rented car while on his cell phone, does not notice that car in front of him is braking and runs into it. Bob at this point call the toll-number provided by PowerEnJoy and report the incident, the call center assistant asks Bob to fill in the insurance form and leave it in the car, if the car cannot move anymore the assistant takes the position of the car, turn off the timer for the payment of the rented car and sends a tow truck to get the car then Bob can leave the car in the accident location and go away.
9. Serena has just arrived in town and hired a car for a ride, she remains so much fascinated by what she sees on the outside that does not notice the repeated messages on the display of the machine referring to the battery running low and reporting the nearest charging point, after a while the battery is depleted and the machine stops. Serena call the toll-number provided by PowerEnJoy, the call center assistant asks Serena to leave the car there and send a tow track to get the car. The call center assistant stops the payment per minutes on the Serena’s rent but apply a fine for the inconvenient.

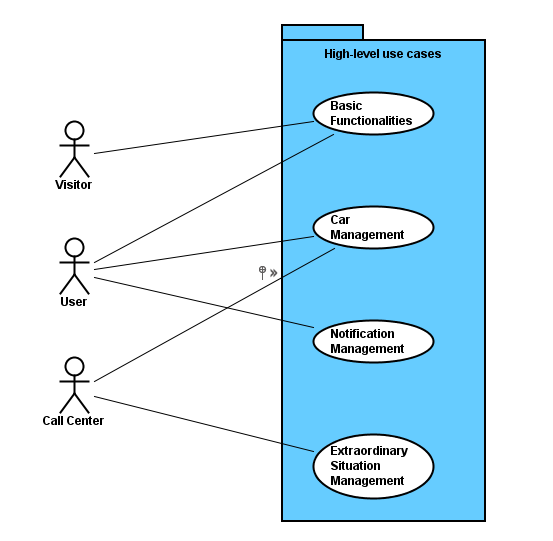
## 5. UML MODELS

### 5.1 USE CASE DIAGRAMS AND DESCRIPTIONS

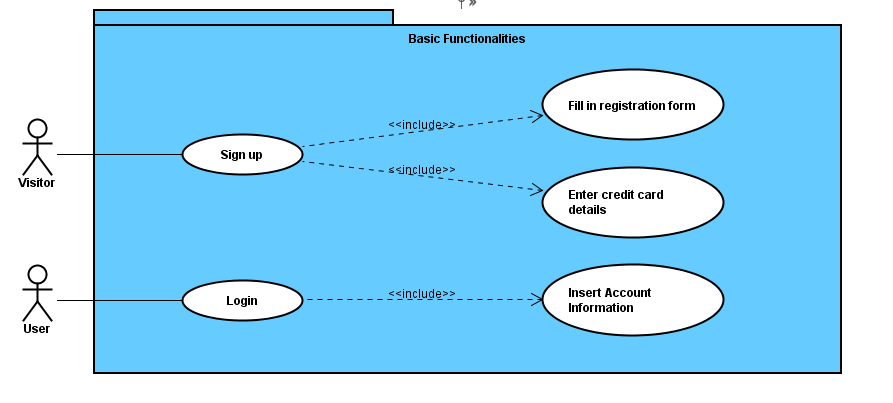
Considering the scenarios described above, we have identified the following use cases:

* Sign up;
* Log in;
* Search for available cars using your position;
* Search for available cars inserting a position
* Rent a car;
* Miss the reservation done because exceeding the timer;
* Unlock the car in order to use it for the first time;
* Unlock the car after having already used that
* Lock the car to reuse it after,
* Lock the car in order to end the rental;
* Forward payment to the bank;
* Show “Battery running low” warning;
* Show “Nearest charging station” notification;
* Show “Impossible to rent a car” notification;
* Show “Max time available for rental” warning;
* Car accident;
* Low battery accident;

We present the first use case diagram as an overview of the main functionalities of the system, each high-level use case shown will be described in detail in another use case diagram.



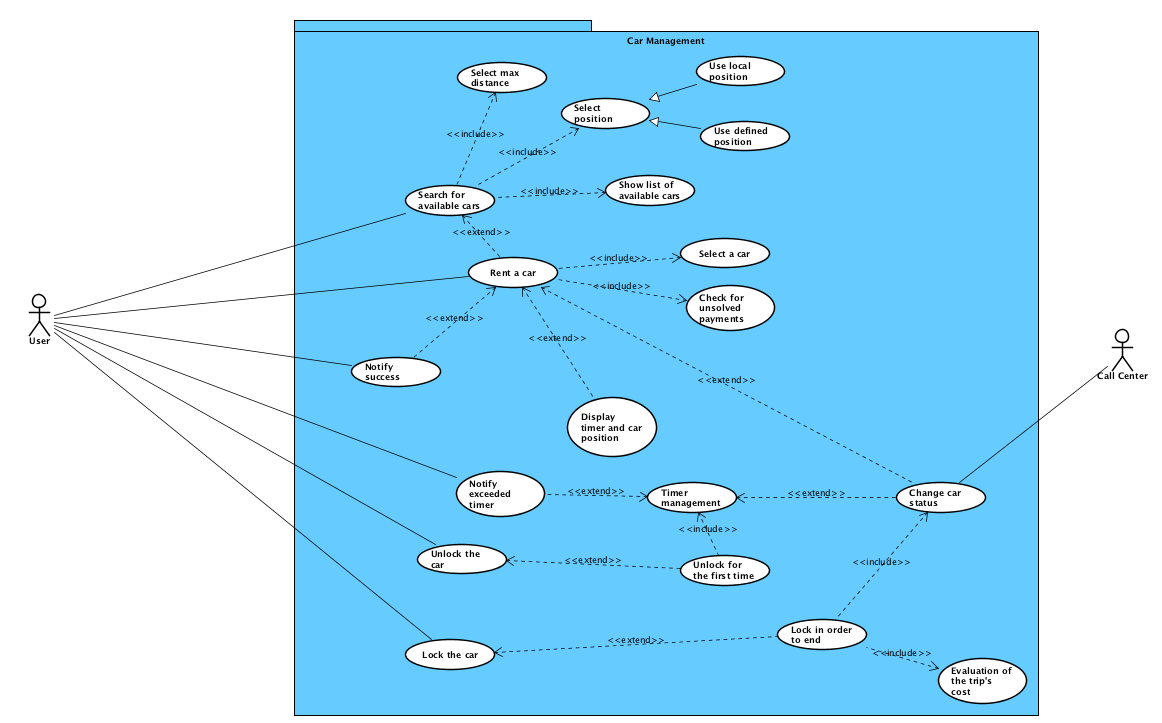
### 5.1.1 BASIC FUNCTIONALITIES



|  |  |
| --- | --- |
|  | **Use case: Sign up** |
| **Actors** | Visitor. |
| **Entry condition** | A person that has not registered to the system yet. |
| **Flow of events** | * The visitor opens the home page or the application of PowerEnJoy and clicks the "Register now" button; * The system displays the registration form; * The visitor fills in the form with username, password and email and submits; * The system shows the credit card credential form; * The visitor fills in the form with: credit card number, expiration date, CW2, name and surname of the holder; * The system notifies the user of the outcome of the operation. |
| **Exit condition** | The person receives a password back and is now registered, so can benefit from the services offered by the platform. |
| **Exceptions** | * The visitor types inappropriate credentials; the system asks the user to insert different data; * The visitor types not valid credit card credentials, the system asks the user to insert valid data; * The system cannot successfully accomplish the registration of the person and notifies him/her. |

|  |  |
| --- | --- |
|  | **Use case: Log in** |
| **Actors** | User registered to the platform. |
| **Entry condition** | The user has successfully signed up to the system. |
| **Flow of events** | * The user opens the home page of the platform or the application; * The user enters his username and password in the input form provided; * The user clicks the button “log in”. * The system shows his/her personal page. |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | The information inserted in the form is wrong (account or password not valid), an error message is shown. |

### 5.1.2 CAR MANAGEMENT



|  |  |
| --- | --- |
|  | **Use case: Search for available cars using your position** |
| **Actors** | User. |
| **Entry condition** | The user is logged into PowerEnJoy. |
| **Flow of events** | * The system displays user’s personal page; * The user clicks the button “Search for available car near your position”; * The system shows max distance option selection; * The user select the max distance for the research; * The system shows a map with the position of each car near him; |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The user provides a not valid distance; |

|  |  |
| --- | --- |
|  | **Use case: Search for available cars inserting a position** |
| **Actors** | User. |
| **Entry condition** | The user is logged into PowerEnJoy. |
| **Flow of events** | * The system displays user’s personal page; * The user clicks the button “Search for available car near a position”; * The system shows insert position option selection; * The user insert an address; * The system shows max distance option selection; * The user select the max distance for the research; * The system displays a map with the position of each car near the selected position; |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The user provides a not valid address; * The user provides a not valid distance; |

|  |  |
| --- | --- |
|  | **Use case: Rent a car** |
| **Actors** | User. |
| **Entry condition** | The user has searched for available cars into PowerEnJoy. |
| **Flow of events** | * The system shows a map with the available cars which are also buttons; * The user select a car he wants to rent by pressing on it; * The system checks if the user has an unsolved payment; * The system creates a new instance of rental; * The system changes the status of the rented car from “available” to “rented”; * The system sets up the rental expiration timer; * The system notifies the user about the success of the operation; * The system display the user the expiration timer and the position of the car; |
| **Exit condition** | The status of the rented car is “rented”. |
| **Exceptions** | * The system cannot ultimate the requested operation. * The user has an unsolved payment about a previous rental. |

|  |  |
| --- | --- |
|  | **Use case: Miss the reservation done because exceeding the timer** |
| **Actors** | User. |
| **Entry condition** | The user has an active rental instance. |
| **Flow of events** | * The event “timer exceeded” occur for an instance of a reservation and the system is informed; * The system informs the user about the expired reservation; * The system charges the cost of a fee of 1€ on the user’s page. * The system changes the status of the car from “rented” to “available”. |
| **Exit condition** | The status of the car is “available”. |
| **Exceptions** | The system cannot ultimate the requested operation. |

|  |  |
| --- | --- |
|  | **Use case: Unlock the car in order to use it for the first time** |
| **Actors** | User. |
| **Entry condition** | The user has an active rental instance, is within 50 m from the car and the timer is not finished yet. |
| **Flow of events** | * The system displays a button to unlock the car; * The user clicks the button “Unlock the car”; * The system opens the car; * The system displays other two buttons “Lock the car to reuse it” and “Lock the car to finish the trip” * The system changes the status of the car. |
| **Exit Condition** | The status of the car is “in use” |
| **Exceptions** | * The system cannot ultimate the requested operation; |

|  |  |
| --- | --- |
|  | **Use case: Lock the car in order to reuse it later** |
| **Actors** | User. |
| **Entry condition** | The user has a rental instance and has already unlocked the car for the first time. |
| **Flow of events** | * The user clicks the button “Lock the car to reuse it”; * The system close the car; * The system displays the button “Unlock the car”; |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The system cannot ultimate the requested operation. |

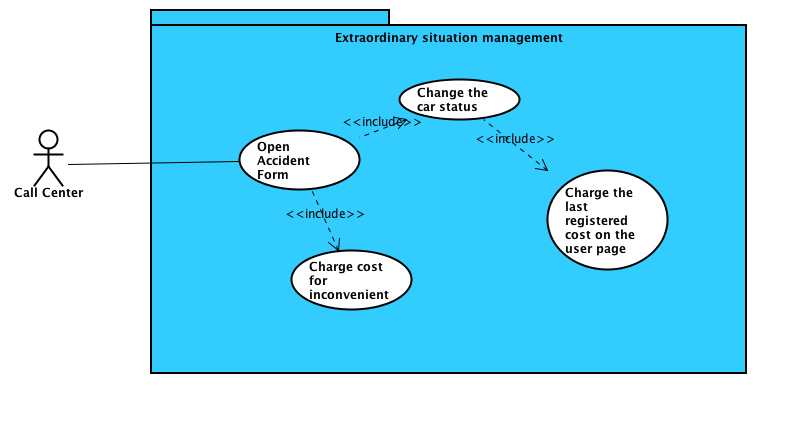
|  |  |
| --- | --- |
|  | **Use case: unlock the car after having already used that** |
| **Actors** | User. |
| **Entry condition** | The user has a rental instance, has locked the car after having already used that and it’s within 50 m from the car. |
| **Flow of events** | * The system displays the button “Unlock the car”; * The user clicks the button “Unlock the car”; * The system opens the car; * The system displays other two buttons “Lock the car to reuse it” and “Lock the car to finish the trip” |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The system cannot ultimate the requested operation. |

|  |  |
| --- | --- |
|  | **Use case: Lock the car to finish the run** |
| **Actors** | User. |
| **Entry condition** | The user has a rental instance, has already unlocked the car for the first time. |
| **Flow of events** | * The user clicks the button “Lock the car to finish the trip”; * The system closes the car; * The system evaluates the cost of the trip. * The system charges the cost of the trip registered on the user’s page. * The system changes the status of the car. |
| **Exit condition** | The status of the car is “available” |
| **Exceptions** | * The system cannot ultimate the requested operation. |

|  |  |
| --- | --- |
|  | **Use case: Forward payment to the bank** |
| **Actors** | User. |
| **Entry condition** | The user has locked the car in order to finish the trip. |
| **Flow of events** | * The system, using the credit card information, sends the last registered cost to the user’s bank; * The bank charges the cost, if it is possible; * The bank sends a summary message to the system confirming the success of the operation; * The system deletes the cost from the user’s page; |
| **Exit condition** | There are no exit condition |
| **Exceptions** | * The system cannot ultimate the requested operation. * The bank cannot confirm the charge of the trip’s cost. |

|  |  |
| --- | --- |
|  | **Use case: exceeding the 24 hour renting time** |
| **Actors** | User. |
| **Entry condition** | The user has an active rental instance. |
| **Flow of events** | * The user exceeded the 24 hours of reservation; * The event “available renting time exceeded” occur for an instance of a reservation and the system is informed; * The system informs the user about the expired reservation and asks the user to park the car in the nearest safe area provided on the screen; * The system activate another timer of 30 minutes in order to let the user park the car; * The system charges the cost of the trip on the user’s page. * The user parks the car; |
| **Exit condition** | The status of the car is “available”. |
| **Exceptions** | If the user doesn’t park the car in 30 minutes, the call center is informed and can decide whether to lock the car and stop it or give more time to the user and charge him. |

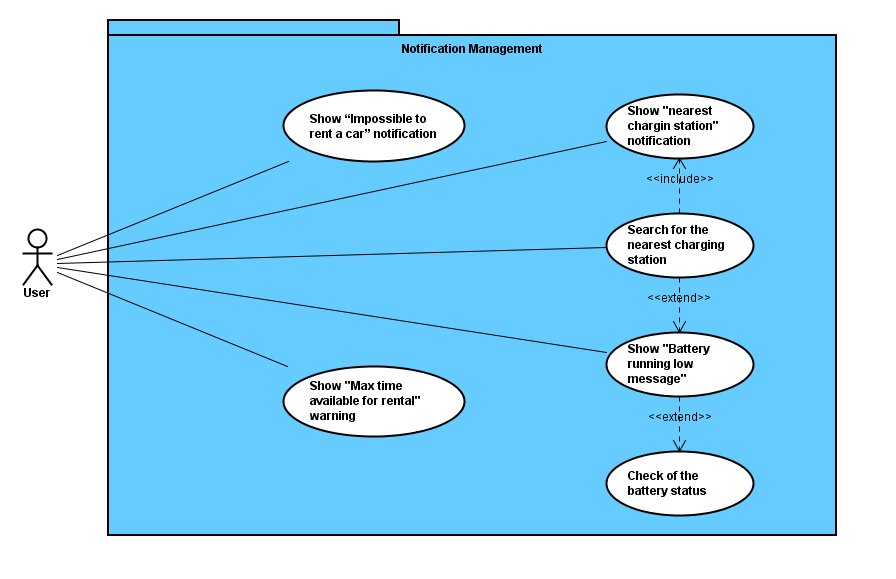
### 5.1.3 EXTRAORDINARY SITUATION MANAGEMENT



|  |  |
| --- | --- |
|  | **Use case: Make an accident** |
| **Actors** | Call Center |
| **Entry condition** | The user, which is using a car (so as a rental instance and the car is in the status “in use” and the battery is not empty), has an accident. |
| **Flow of events** | * The user makes a call to the Call Center. * At the Call center the first free operator answers the call; * The Call Center sends on the place an operator to analyze the situation. * The Call Center opens an accident dossier and changes the status of the car. * The system charges the last registered cost on the user page. * The Call Center finally charges further costs on the user page according to the damages produced by the user and estimated by the operator on the accident place. |
| **Exit condition** | The status of the car is changed from “in use” to “available” |
| **Exceptions** | * The user cannot call for the Call Center |

|  |  |
| --- | --- |
|  | **Use case: Terminate the battery** |
| **Actors** | Call Center |
| **Entry condition** | The user, which is using a car (so as a rental instance and the car is in the status “in use”), terminates the battery. |
| **Flow of events** | * The user makes a call to the Call Center; * At the Call center the first free operator answers the call; * The Call Center sends on the place an operator to analyze the situation; * The Call Center open an accident dossier and changes the status of the car; * The System charges the last registered cost on the user page; * The Call Center finally charges further costs on the user page due to the work of the operator which has to bring the car to a power grid station; |
| **Exit condition** | The status of the car is changed from “in use” to “damaged” |
| **Exceptions** | * The use cannot call for the Call Center |

### 5.1.4 NOTIFICATION MANAGEMENT



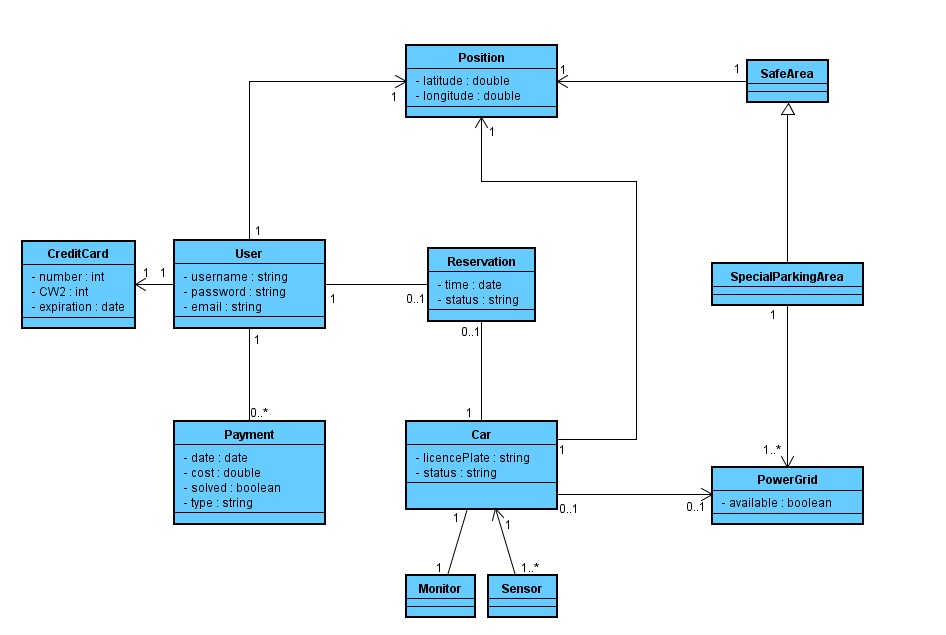
|  |  |
| --- | --- |
|  | **Use case: Show “Battery running low” warning** |
| **Actors** | User |
| **Entry condition** | The user is using his rented car. |
| **Flow of events** | * The system is triggered when the battery charge of the car is below a certain level (20%, 10%, 5%), (so he notices that the car battery is running low); * The system evaluates the precise charge of the car battery; * The system displays on the car monitor a message stating that the battery charge is low |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The system can’t display the message on the car’s monitor |

|  |  |
| --- | --- |
|  | **Use case: Show “Nearest charging station” notification** |
| **Actors** | User |
| **Entry condition** | The user is using his rented car. |
| **Flow of events** | * The system is triggered when the battery charge of the car is below a certain level (20%, 10%, 5%), (so he notices that the car battery is running low); * The system evaluates the precise charge of the car battery; * The system displays on the car monitor a message stating that the battery charge is low; * The user select the button on the screen “Search for the nearest charging station”; * The system detects car position; * The system calculates and find the nearest charging station to the car; * The system shows the nearest charging station on the display of the car; |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The system can’t display the message on the car’s monitor * The system cannot ultimate the requested operation. |

|  |  |
| --- | --- |
|  | **Use case: Show “Impossible to rent a car” notification** |
| **Actors** | User |
| **Entry condition** | The user has searched for available cars into PowerEnJoy. |
| **Flow of events** | * The system shows a map with the available cars which are also buttons; * The user select a car he wants to rent by pressing on it; * The system checks if the user has an unsolved payment; * The system finds out that the user has an unsolved payment; * The system displays a message on the browser/application user’s screen stating: ”impossible to rent the car you have an unsolved payment”. |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The system can’t display the message |

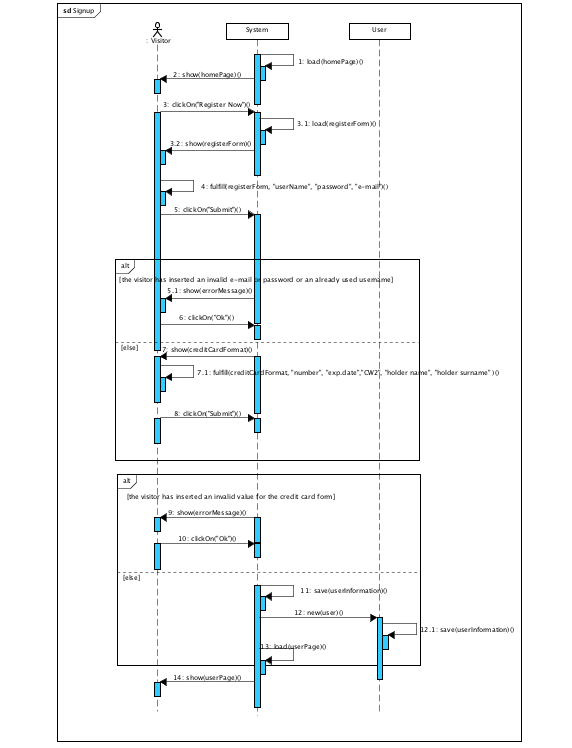
|  |  |
| --- | --- |
|  | **Use case: Show “Max time available for rental” warning** |
| **Actors** | User |
| **Entry condition** | The user is using his rented car. |
| **Flow of events** | * The system is triggered when the time of use of the rented car is approaching the maximum limit of 24 hours; * The system inform the user by displaying the message on the car’s display “Reaching the maximum time available for rental“; * The user clicks the "ok" button and closes the notification window. |
| **Exit condition** | There are no exit conditions. |
| **Exceptions** | * The system can’t display the message. |

#### 5.2 CLASS DIAGRAM

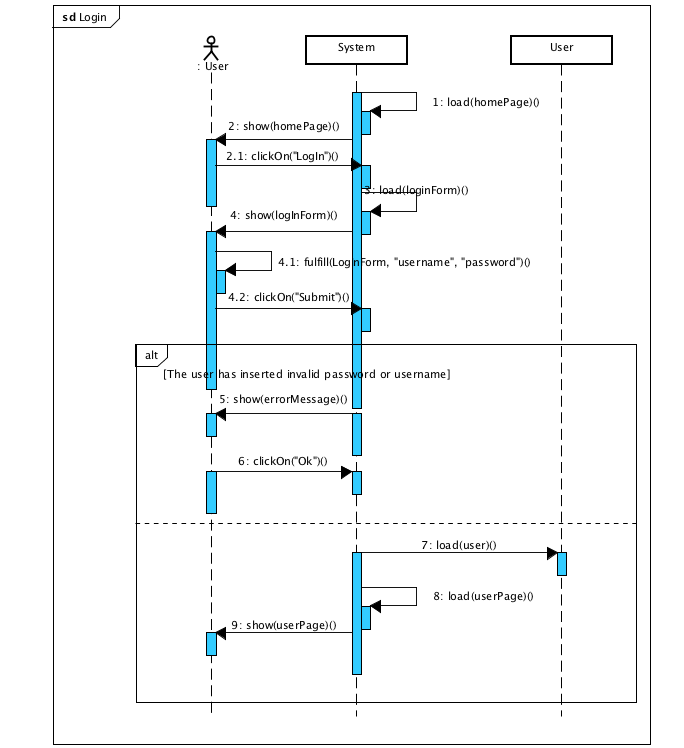


### 5.3 SEQUENCE DIAGRAMS

#### 5.3.1 SIGN UP

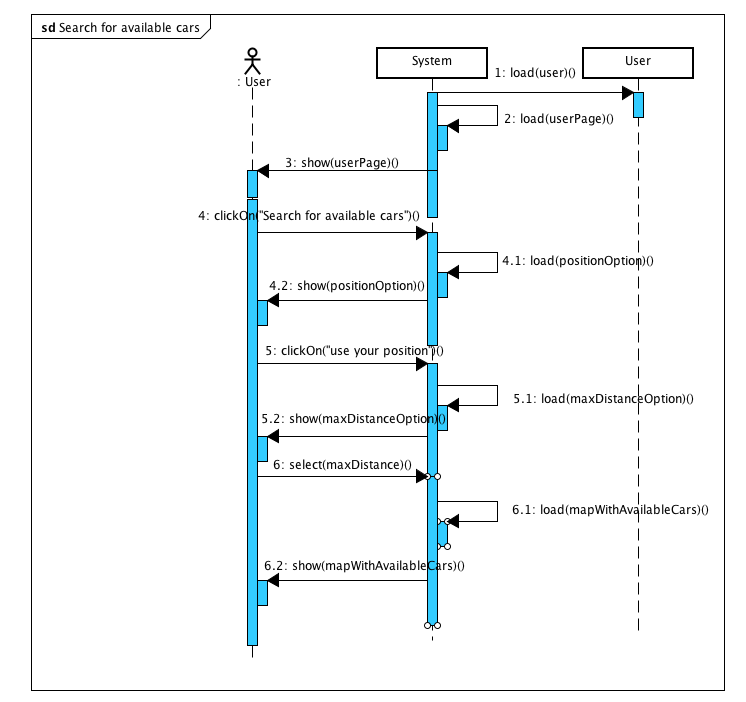


#### 5.3.2 LOG IN

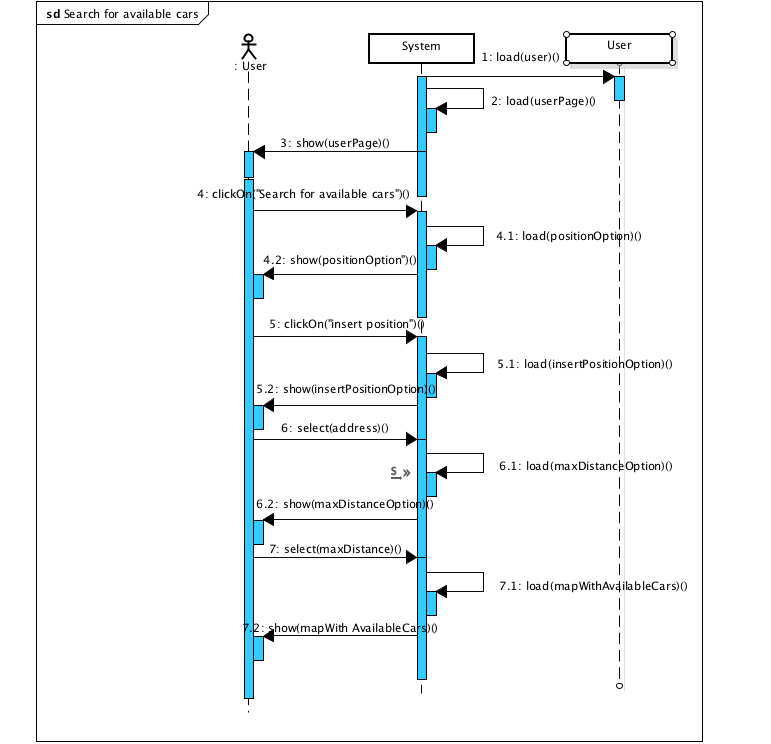


#### 5.3.3 SEARCH FOR AVAILABLE CARS

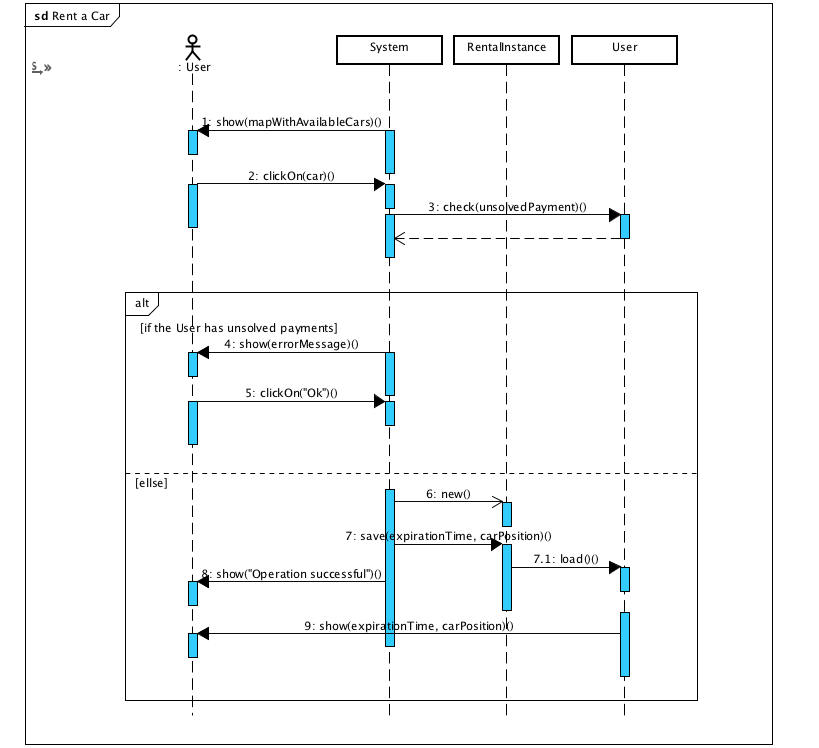
#### 5.3.3.1 USING YOUR POSITION



#### 5.3.3.2 INSERTING A POSITION

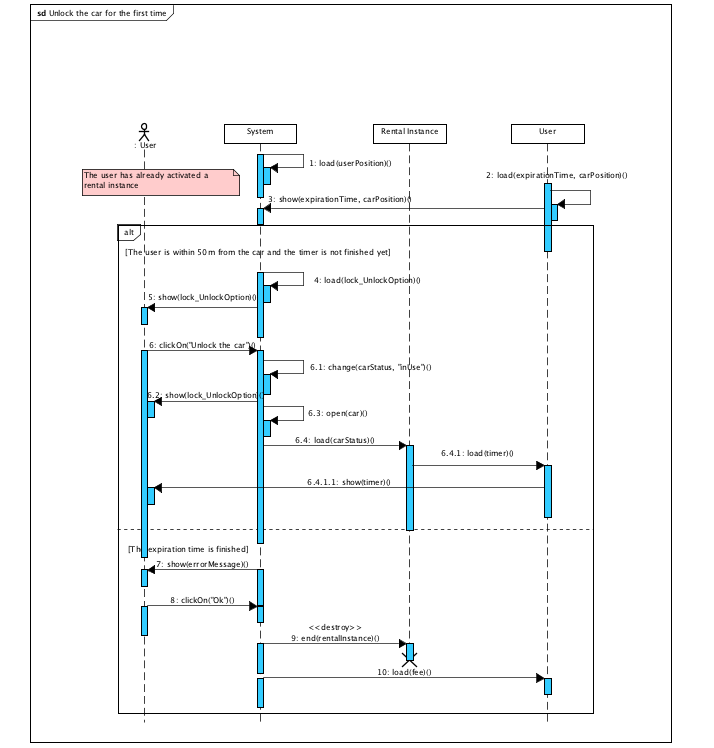


#### 5.3.4 RENT A CAR

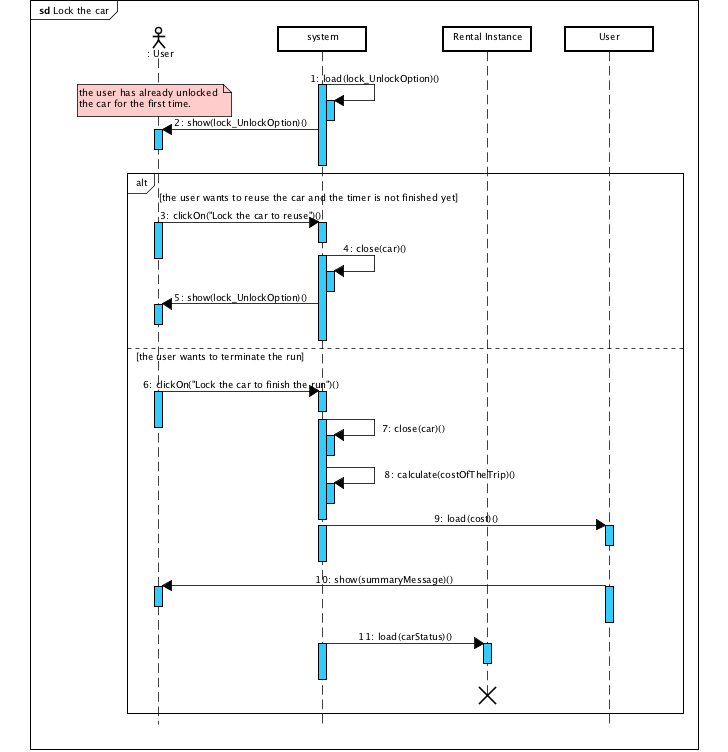


#### 5.3.5 LOCK\_UNLOCK OPTION

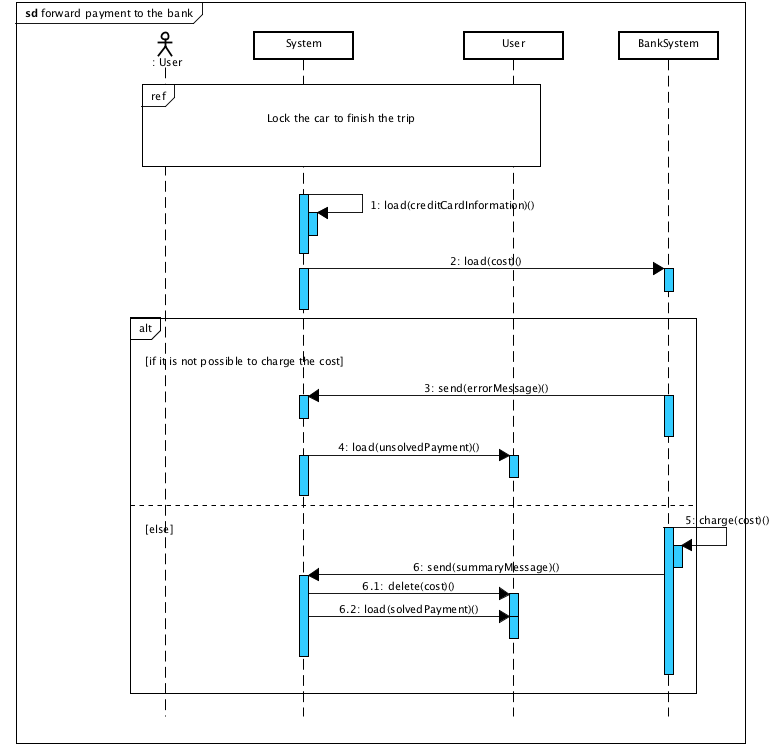
#### 5.3.5.1 UNLOCK FOR THE FIRST TIME



#### 5.3.5.2 LOCK IN ORDER TO FINISH THE RUN

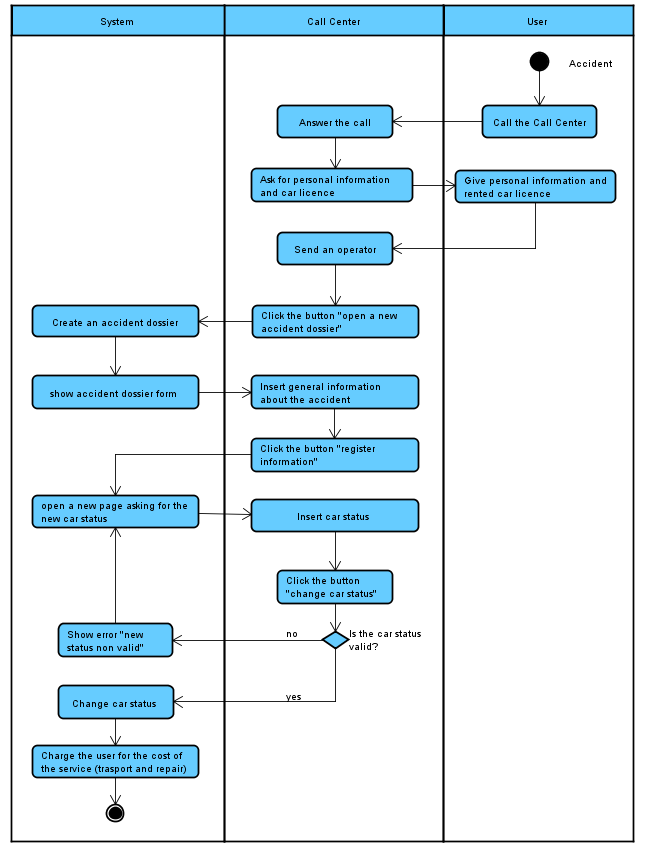


#### 5.3.6 FORWARD PAYMENT TO THE BANK

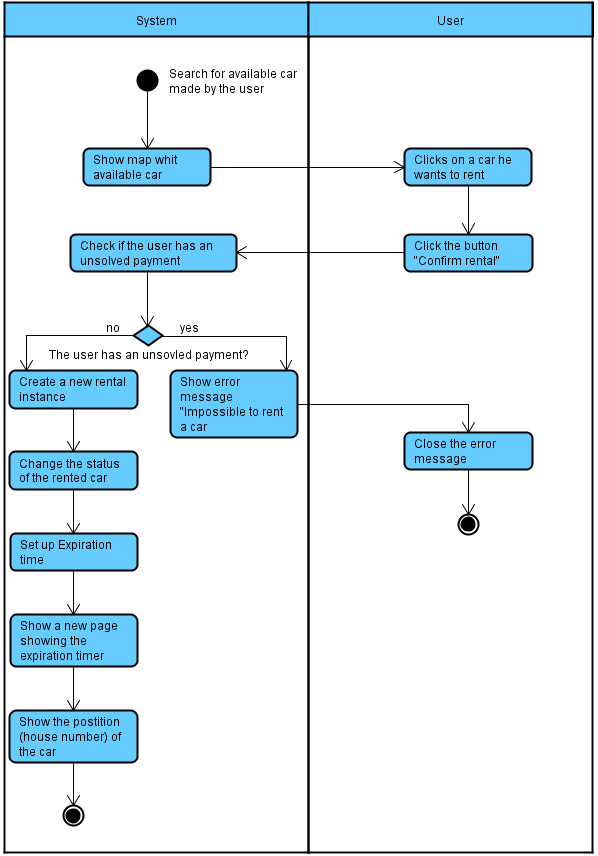


#### 5.4 ACTIVITY DIAGRAMS

##### 5.4.1 CAR ACCIDENT and BATTERY LOW



##### 5.4.2 RENT A CAR



## 6. ALLOY MODELING

In this paragraph we try to understand if the Class Diagram can be consistent by using an Alloy model. We wrote the predicates needed and then we generated the World, in order to verify the consistency of our model. Here we report the used code and some screenshots of the Alloy Analyzer. Facts and assertions are commented so that it is easier to understand what’s their purpose.

### 6.1 CODE

//SIGNATURES

some sig User{

ID: one UserID,

userCreditCard: one CreditCard,

userPayment: set Payment,

userPosition: one Position,

}

sig Payment{

ID: one PaymentID,

status: one PaymentStatus,

}

abstract sig PaymentStatus{}

one sig Solved extends PaymentStatus{}

one sig Unsolved extends PaymentStatus{}

sig CreditCard{}

sig Position{}

sig UserID{}

sig PaymentID{}

sig Reservation{

user: one User,

car: one Car,

}

sig Car{

position: one Position,

powerGrid: lone PowerGrid,

monitor: one Monitor,

sensor: some Sensor,

status: one CarStatus,

}

abstract sig CarStatus{}

one sig Available extends CarStatus{}

one sig InUse extends CarStatus{}

one sig Rented extends CarStatus{}

one sig Damaged extends CarStatus{}

sig Monitor{}

sig Sensor{}

sig SafeArea{

position: one Position,

}

sig SpecialParkingArea extends SafeArea{

powerGrid: some PowerGrid,

}

sig PowerGrid{}

// FACTS

//each payment must be associated exactly to one user

fact paymentAssociatedToOneUser{

all p: Payment | (some u: User |p in u.userPayment)

}

//each position must be associated either to a user or to a car or to a safe area

fact positionAssociated{

all p: Position | (some u: User | p in u.userPosition) || (some c: Car | p in c.position) || (some sa: SafeArea | p in sa.position)

}

//each credit card must be associated to at least one user

fact creditCardAssociated{

all c: CreditCard | (some u: User |c=u.userCreditCard)

}

//each monitor must be associated to a car

fact monitoAssociated{

all m: Monitor| (some c: Car |m= c.monitor)

}

//each sensor must be associated to a car

fact sensorAssociated{

all s: Sensor| (some c: Car |s in c.sensor)

}

//each UserID must be associated to a user

fact userIDAssociated{

all id: UserID | (some u: User | id=u.ID)

}

//each paymentID must be associated to a payment

fact paymentIDAssociated{

all id: PaymentID | (some p: Payment | id=p.ID)

}

//each power grid must be associated to one special parking area

fact powerGridAssociated{

all pg: PowerGrid | (some spa: SpecialParkingArea | pg in spa.powerGrid)

}

//two users cannot refer to the same ID

fact uniqueUserID{

all disj u1, u2: User | u1.ID!=u2.ID

}

//two payments cannot refer to the same ID

fact uniquePaymentID{

all disj p1, p2: Payment | p1.ID!=p2.ID

}

//two users must have different payments

fact uniqueUserPayment{

all disj u1,u2:User | u1.userPayment&u2.userPayment = none

}

//two cars must be associated to different power grid

fact uniquePowerGrid{

all disj c1, c2: Car | c1.powerGrid!=c2.powerGrid || c1.powerGrid=none

}

//two cars cannot be associated to the same monitor

fact uniqueMonitor{

all disj c1,c2:Car | c1.monitor!=c2.monitor

}

//two cars cannot be associated to the same set of sensors

fact uniqueSensors{

all disj c1, c2: Car | c1.sensor&c2.sensor= none

}

//two users cannot be associated to the same position

fact uniquePositionForDifferentUsers{

all disj u1, u2: User | u1.userPosition!=u2.userPosition

}

//two reservations cannot be associated to the same user and car

fact uniqeuReservation{

all disj r1, r2: Reservation | r1.user!=r2.user

all disj r1, r2: Reservation | r1.car!=r2.car

}

//two cars cannot be associated to the same position

fact uniquePositionForDifferentCars{

all disj c1, c2: Car | c1.position!=c2.position

}

//two safe areas cannot be associated to the same position

fact uniquePositionForDifferentSafeAreas{

all disj sa1, sa2: SafeArea | sa1.position!=sa2.position

}

//two special parking areas cannot be associated to the same position

fact uniquePositionForDifferentSpecialParkingAreas{

all disj spa1, spa2: SpecialParkingArea | spa1.position!=spa2.position

}

//two special parking areas cannot be associated to the same power grid

fact uniquePowerGrid{

all disj spa1, spa2 : SpecialParkingArea | spa1.powerGrid&spa2.powerGrid= none

}

//two payment unsolved cannot be associated with the same user

fact notTwoPaymentUnsolved{

all u:User, disj p1,p2:Payment | p1.status & Unsolved !=none && p2.status & Unsolved !=none implies

(p1 not in u.userPayment || p2 not in u.userPayment)

}

//users, cars and safe areas cannot have the same position

fact uniquePosition{

no u: User, c: Car | u.userPosition=c.position

no u: User, sa: SafeArea | u.userPosition = sa.position

no c: Car, sa: SafeArea | c.position = sa.position

}

//the status of the car depends from the reservation. If a car is associated to a reservation, it can be in use or rented, else it must be damaged or available

fact carStatus{

all r: Reservation, c: Car | r.car=c implies (c.status& InUse!=none || c.status&Rented!=none)

all c: Car| c not in Reservation.car implies (c.status&Available!=none || c.status&Damaged!=none)

}

//if a user has a Payment unsolved it cannot have a reservation

fact PaymentUnsolvedNoReservation{

all u:User,p:Payment | (p in u.userPayment && p.status in Unsolved) implies (no r: Reservation | r.user=u)

}

//a car that is InUse cannot be plugged into a PowerGrid

fact carInUseNotPlugged{

all c:Car | (#(c.status&InUse)>0) implies c.powerGrid=none

}

// ASSERTION

//if there are no car Rented or InUse the number of reservations is 0

assert reservationCheck{

#(Car.status&InUse + Car.status&Rented)=0 implies #(Reservation)=0

}

//if there are no reservation all the car status are Available or Damaged

assert carStatusCheck{

#(Reservation)=0 implies #(Car.status&InUse + Car.status&Rented)=0

}

//if a user has a reservation all his payments must be Solved

assert paymentCheck{

all u:User,r:Reservation | r.user=u implies #(u.userPayment.status&Unsolved)=0

}

//if a car is plugged into a powerGrid it is not InUse

assert carPluggedCheck{

all c:Car | c.powerGrid!=none implies c.status&InUse=none

}

//if all cars are damaged there cannot be any reservations

assert allCarDamagedCheck{

(all c:Car | c.status&Damaged!=none) implies #Reservation=0

}

//if all car are available there cannot be any reservation

assert allCarAvailableCheck{

(all c:Car | c.status&Available!=none) implies #Reservation=0

}

//if there is at least one unsolved payment and only one user there cannot be any reservations

assert oneUnsolvedPaymentAndUserMeansNoReservation{

(#User=1 and Payment.status&Unsolved!=none) implies #Reservation=0

}

// PREDICATES

pred showAll{

#User=3 and

#Payment>4 and

#Reservation=2 and

#Car=4

}

pred showCars{

#User=2 and

#Reservation=2 and

#(Car.status&InUse)=1 and

#(Car.status&Available)=1 and

#(Car.status&Damaged)=1 and

#(Car.status&Rented)=1 and

#Car=4 and

#Payment=0 and

#SpecialParkingArea=1 and

#PowerGrid=2

}

pred showReservation{

#User=1 and

#Reservation=1 and

#Payment=0 and

#Car=1

}

pred showPayments{

#User=2 and

#Reservation=1 and

#Payment=3 and

#Car=1 and

#(Payment.status&Unsolved)=1

}

check reservationCheck

check carStatusCheck

check paymentCheck

check carPluggedCheck

check allCarDamagedCheck

check allCarAvailableCheck

check oneUnsolvedPaymentAndUserMeansNoReservation

run showAll for 8

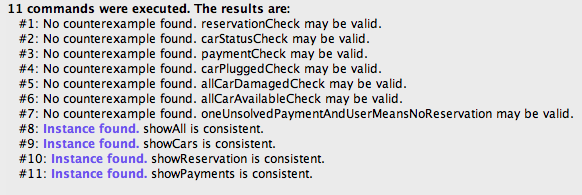
run showCars for 8

run showReservation for 3

run showPayments for 3

### 6.2 GENERATED WORLDS

We show now the results we have obtained running the Alloy Analyzer. Let’s start from the commands executed. As we can see, for every assertion (commands 1 to 7), the analyzer didn't find any counterexamples. Moreover, the analyzer found some consistent instances of the world we created (command 8 to 11): the first one is the general one in which all the relevant elements are presented at least one time. The others represent particular aspects related to cars, reservations and payments



Now we present the instances of the world in more detail, highlighting the most relevant aspects for each of them.

1. The first image represents the result of the predicate *showAll.*
   * Every user has an ID a position and a credit card. We can see also that two different user can be associated to the same credit card. A user can be associated to payments and reservation, with the meaning that a user has already paid the cost of a run and that has rented a car
   * Every car has a monitor and some sensors, a position and a status. The status of the car can be available, when the car is free from any reservation and not damaged, rented, when a user has reserved that but has not already unlocked it, in use, when the user has rented the car and has already unlocked that, and damaged when the car has been involved to an accident and so it cannot be used for a reservation.
   * Every special parking area has a position and at least a power grid;
   * Every safe area has a position;
2. The second image represents the result of the predicate *showCars*.
   * A car which is associated to a reservation must be rented or in use. If it is rented it can be or in a safe area or in a power grid of a special parking area, while if it is in use it cannot be associated neither to a safe area nor to a special parking area;
   * A car which is available or damaged cannot be associated to a reservation.
   * Both and available car and a damaged one can be in a power grid.
3. The third image represents the result of the predicate *showReservation*:
   * A reservation can involve just exactly one user and exactly one car;
   * A car which is involved in a reservation must be rented or in use;
   * A user involved in this kind of relation can have either no payments already done or just solved payments
4. The last image represents the result of the predicate *showPayment*:
   * A user can have either solved payments or unsolved payments or no payments at all;
   * If a user has unsolved payments he cannot be associated to a reservation;

Image 1: result of the predicate *ShowAll.*

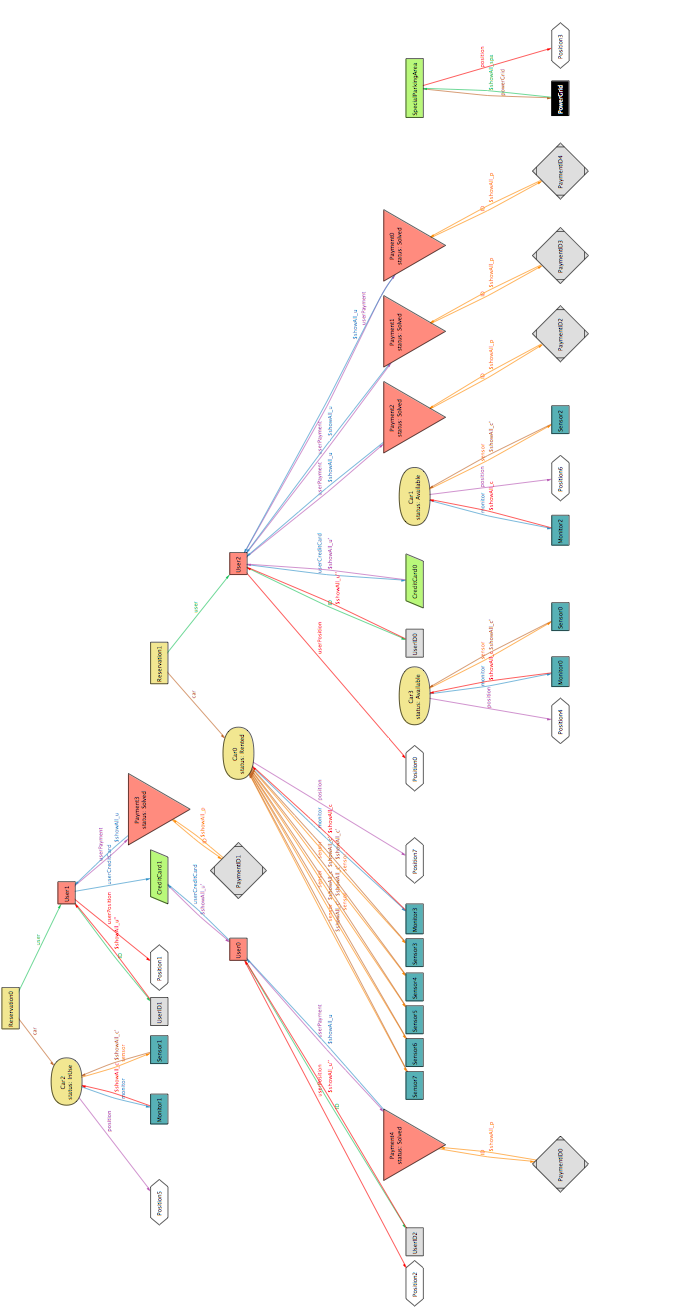


Image 2: result of the predicate *ShowCars.*

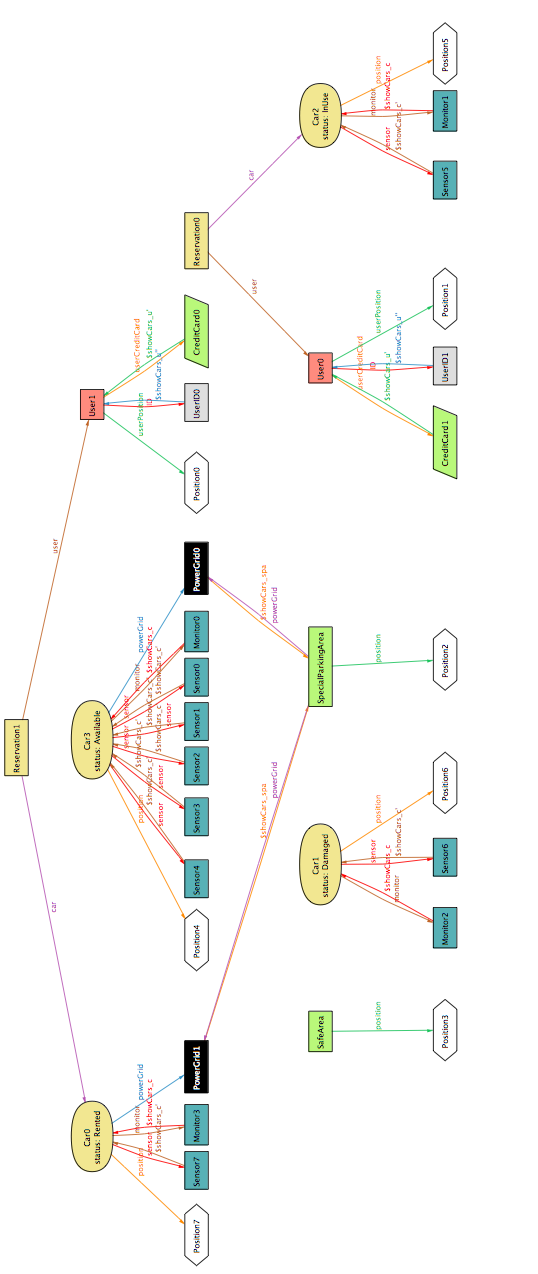


Image 3: result of the predicate *ShowReservation.*

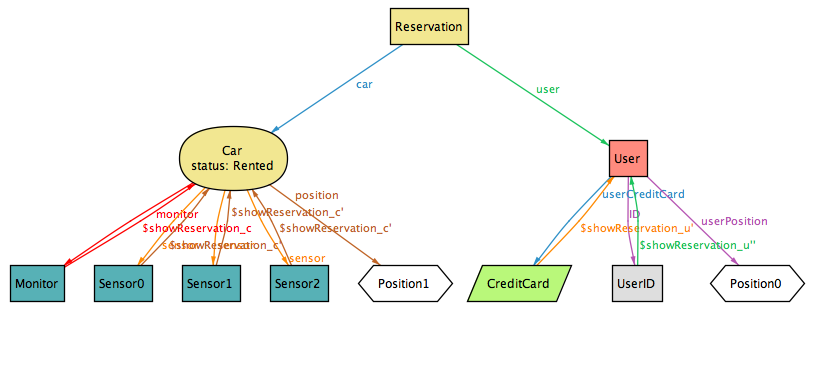
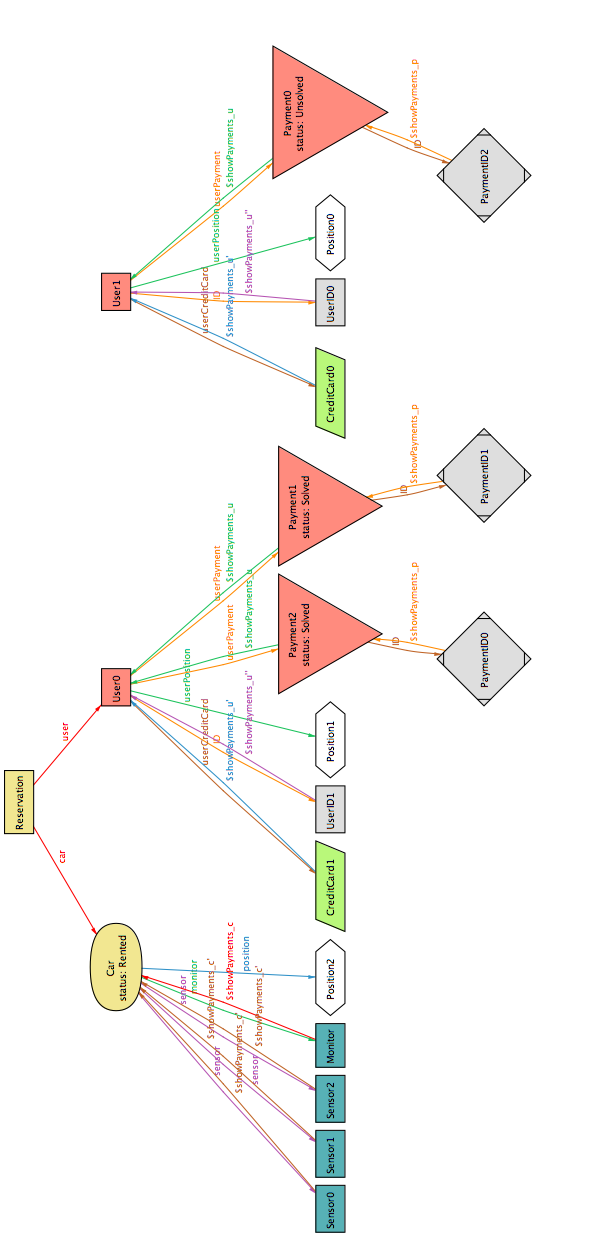


Image 4: result of the predicate *ShowPayments*.



## 7. USED TOOLS

Here are listed the tools used for creating this RASD document:

* Microsoft Office Word 2011: to redact and format this document;
* Alloy Analyzer 4.2: to prove the consistency of our model
* Astah professional: to write UML documents;

## 8. TIME SPENT

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time spent [hours]** | |  |
| **Activities** | **Emanuele Chilà** | **Giorgio Lazzarinetti** | **Total** |
| Goals, domain assumptions and requirements identification | 4 | 3 | 7 |
| Scenarios identification and description | 3 | 4 | 7 |
| Use cases identification and detailed description | 6 | 6 | 12 |
| UML Models (Use Case Diagrams,  Class Diagram, Sequence Diagrams and Activity Diagrams) | 6 | 6 | 12 |
| Alloy (Coding + Verification) | 6 | 6 | 12 |
| Overall document formatting | 2 | 3 | 5 |
|  | **Total RASD** | | **55** |