Power EnJoy

Requirements **A**nalysis and **S**pecifications **D**ocument

Software Engineering 2

A.A. 2016/2017

Version 1.0

Emanuele Ghelfi (mat. 875550)
Emiliano Gagliardi (mat. 878539)

November 12, 2016

Contents

1 Introduction

1.1 Description of the given problem

The purpose of this document is to support the development of PowerEnJoy, a software that manages a car sharing service for electric cars. The aim of this software is to make simple and quick the reservation of cars. So the system should provide users with real time information about availability of cars, their status and their positions. Users, after the reservation, can directly get their car in pre-defined parking areas.
The service will be accessible only to registered users, giving some personal information and data needed to the payment. The price of the ride is computed with a fixed amount of money per minute, displayed by the car, and finally charged. To avoid useless reservation, where a user doesn't pick up the car, the reservation expires after a fixed time, the car returns available, and the user is charged with a fee. Cars must be locked in the safe areas, and only users that have made the reservation can unlock them.
The software has to provide also management functionality for administrators and operators in order to ensure a simple managing of the system.

1.2 Glossary and abbreviations

- Virtuous behaviour:
  - Behaviour that makes simple the maintance of the cars and reservation areas.
  - Behaviour helps to reduce the traffic impact.
  - To park in defined safe areas.
- Car status, that can assume the following meanings:
  - Free: it can be reserved.
  - Reserved: has alreasy been reserved.
  - Busy: a user is driving the car.
  - Under maintenance: an operator is fixing some car problem.
  - In charge: the car is plugged in a charging station.
- Car information: technical information about car, useful for mantainance.
- Safe area: special, pre-defined parking area where cars can be parked. It can be also a charging area.
- Charging area: an area where cars can be recharged. It is always a Safe Area.
- Malfunction: the car has some problems that can be repaired by an operator.

- Damage: it's like a malfunction but caused by the user.
- Ride: when a user unlocks the car and then use it to go from a place to another.
- Ride Status:
  - Active: the user is driving the car.
  - Paused: the user has paused the ride and so he can maintain the reservation paying a certain amount of money per minute.
- Payment method: a method of payment the user can use in order to pay Power EnJoy services.
- API: Application Programming Interfaces, a set of methods accessible from an external system.

1.3 Actors

These are the people that will be involved in the use cases of the software to be:

- GUEST: a person that isn't registered to the system.
- USER: a person that has already registered to the system.
- OPERATOR: an employee of the company that takes maintenance of cars and charging area.
- ADMIN: a person that can manage system status with privileged actions.

1.4 Goals

The objectives of the software to be are the following:

- [G1] Users should be able to reserve a car and use it.
  - [G1.1] Users should be able to reserve a car.
  - [G1.2] Users should be able to unlock reseved car when they are close to it.
  - [G1.3] Users should be aware of how much they are going to pay during the ride.
  - [G1.4] If reserved car pass under maintenance, the User that made the reservation must be notified and the reservation should be deleted.
  - [G1.5] User should be able to set Pause status during a ride.
  - [G1.6] User should be able to restart a paused ride.
- [G2] The reservation of two (or more) cars at a time must be forbidden.
- [G3] Users and Guests must be able to search cars.

  - [G3.1] Users and Guests should be able to search cars near his position.
  - [G3.2] Users and Guests should be able to search cars near a selected position.

- [G4] Induce users to keep a virtuous behaviour.
  - [G4.1] A discount of 10% should be applied to rides with at least two passengers.
  - [G4.2] If a car is left with no more than 50% battery empty, a discount of 20% should be applied to the last ride.
  - [G4.3] If a car is left at more than 3 km from the nearest charging area, the system should charges 30% more on the last ride.
  - [G4.4] If a car is left with more than 80% battery empty, the system should charges 30% more on the last ride.
  - [G4.5] If a car is left in a charging area and the user plugs the car into the power grid, a discount of 30% should be applied to the last ride.
  - [G4.6] If a car is not picked up within one hour from the reservation, the user should pay a fee of 1 EUR.
  - [G4.7] If a payment fails because of insufficient money availability of the user, the user should be suspended from the service until the payment is done.
  - [G4.8] If a user parks in an area that is not a safe area, the user should pay a fee.

- [G5] Users have to pay an amount of money based on the ride's duration.

- [G6] Guarantee a ready maintenance of cars.
  - [G6.1] Operators should be able to do car mantainance, knowing their position and status.
  - [G6.2] Users should be able to comunicate to the system cars damages and malfunctions.

- [G7] Admins must be able to manage the system.
  - [G7.1] Admins must be able to manage safe areas.
  - [G7.2] Admins must be able to manage operators.

1.5 Stakeholders identification

The principal stakeholder is the customer, a company that offers a car sharing service and needs a platform to manage it and make it easy for possible users. Other involved people are the employees of the company, who will work with the software. Evidently also the users of the application are stakeholders.
In the end, related to more technical purposes, other possible developers that will consult the software documentation are stakeholders.

1.6 Document overview

The document is structured in the following four parts:

- Introduction: this is the introduction of the document, where is given a general view of the problem, and are exposed the fundaental purposes of the software to be developed
- Overall description: it describes the assumptions made on the domain in wich the software will work, and its dependecies with the used interfaces
- Requirements: it describes the functional and non functional requirements, and their relation with the goals and the domain assumptions
- Scenario: it describes some possible situations that involve the system
- UML model
- Alloy modeling

2 Overall description

2.1 Product perspective

In this section the system boundaries are described. Are described the interfaces provided by external systems that the software to be will use to provide the required functionalities.

2.1.1 Payment methods

The software relies on external transactional system in order to administer in a good way the payment process and to be able to manage exceptional situations too (like user unavailability of money).
So the system doesn't care about how the payments are done, but only about the result of the payment (success or failed).

2.1.2 Car technology

The software to be will use the technologies provided by a system already present on cars, in order to carry out the needed actions. The used APIs provide a secure connection between cars and the central system via internet, and the following commands that the central system can impose on car:

- Lock: locks the car doors.
- Unlock: unlocks the car doors.
- Get position: the car sends to the central system its position.
- Get status: the car sends to the central system detailed information about his current status.

2.1.3 Operators, maintenance, malfunctions and damanges.

The cars availability depends heavily on the presence of a staff that does cars maintenance. So the sotware to be relies on the presence of the staff in order to guarantee availability of cars, and it will provide to operators information that helps the cars maintenance. Operators will manage damanges to cars and consequence for the responsible, so the system will only help the operators in maintenance, and guarantee a way to comunicate malfunction and damanges.

2.2 Domain assumptions

- [D1] Cars positions provided by GPS are accurate.
- [D2] The user and guest position provided by GPS are accurate.
- [D3] Cars provide remote comunication API.
- [D4] Cars provide screen to display current charge to user.

- [D5] Cars can detect how many passengers are on it.
- [D6] Cars can detect if they are plugged.
- [D7] Cars can detect their battery level.
- [D8] Cars can detect information about maintenance needs.
- [D9] If a user detect a malfuncion comunicate it.
- [D10] If an operator is notified by the system about a maintenance need, he takes care of that.
- [D11] Operators do not share their credential.
- [D12] Admins do not share their credential.
- [D13] Users do not share their password.

2.3 Assumptions

There are few point that aren't well specified in the assignement document. The purpose of this section is to formalize some facts that complete the specification.

- Also guests are able to search cars, in order to induce them to create an account.
- There are no information about cars' maintenance, so in the document is made the assumption that cars need operators presence to keep an acceptable status.
- In order to avoid useless resarvations, users can't delete reservations.
- The system relies on external payment methods. The accepted methods are Mastercard, Visa, PayPal, PostePay.
- If a payment fails because of an insufficient money availability of the user, the user is suspended from the service until the payment is done.
- In order to induce users to park in safe areas, users that do not park in safe areas will pay a fee.
- There are no assumptions on how the safe areas are managed, so it is assumed that it is possible for admins to add or remove them in order to ensure flexibility of the system.
- In order to improve the quality of our system has been added the possibility of "pausing" a ride. So a user can "pause" a ride, he can maintain the reservation on the car and he can pay a reduced amount of money. It could be a good function for user. Pause status expires after 1h and then car returns in the Available state. This is done in order to avoid very long pauses.

- A car can go "Under Maintenance" even if it's reserved by a user, in this case the user is notified and his reservation is deleted.
- The user is free to reserve a car even if its battery is low.
- Discount aren't cumulative: in a ride with more discount applicable it's applied only the highest one.
- Fee are cumulative.

2.4 Documentation

These documents will be released in order to well-organize the work in the way to obtain the best quality-cost ratio as possible:

- ***RASD***: Requirements Analysis and Specification Document, to well-understand the given problem and to analyze system's boundaries and goal. RASD is also important to understan system's requirements and specifications in order to reach the goals.
- ***DD*** :Design Document, to define the structure of the the system, his tiers, and the interaction between them.
- ***ITPD***: Integration Test Plan Document, to describe the planning to accomplish the integration test. This document is supposed to be written before the integration test really happens. Moreover this document needs to explain to the developer team what to test, in which sequence, which tools are needed for testing and wich need to be developed.
- ***PP***: Project Plan, aims at defining a planning for the project. It regards in particular the estimation of effort and cost, the scheduling for project's task and the allocation of resources to tasks.

3 Requirements

In order to satisfy goals in section 1.4 under domain assumptions in Section 2.2 we've derived requirements for our system.

- [G1.1] Only user should be able to reserve a car.
  - [R1] The system can modify car's status (available, occupied, in use).
  - [R2] The system shall be able to know car's status.
  - [R3] The system shall reserve a car only if the car is available.
  - [R4] The system shall be able to associate a car to the user who has reserved it.
  - [R5] The system shall provide the login functionality.
  - [R6] The system shall provide sign up functionality.
  - [D13] Users doesn't share their password.
- [G1.2] User should be able to unlock reserved car when they are close to it.
  - [R7] The system shall know cars' location according to cars' GPS.
  - [R8] The system shall know users' location according to his GPS.
  - [R9] The system shall provide a functionality to unlock the car.
  - [D1] Cars positions provided by GPS are accurate.
  - [D2] The user and guest position provided by GPS are accurate.
  - [D3] Cars provide remote comunication API in order to manage them remotely.
- [G1.3] User should be aware of how much they are going to pay during the ride.
  - [R10] The system shall be able to calculate ride's cost based on duration.
  - [R11] The system shall be able to communicate to the car ride's cost.
  - [D4] Cars provide screen to display current charge to user.
- [G1.4] If reserved car pass under maintenance, the user that made the reservation must be notified.
  - [R12] The system shall be able to notify the user when his reserved car switches to under maintainance.
- [G1.5] User should be able to set pause status during a ride.
  - [R13] The system shall offer the functionality to pause a ride.
  - [R14] The system shall be able to manage paused ride.

- [G1.6] User should be able to restart a paused ride.
  - [R14] The system shall be able to manage paused ride.
  - [R15] The system shall offer the functionality to restart a paused ride
- [G2] The reservation of two (or more) cars at a time must be forbidden.
  - [R16] The system shall reserve a car only if the user hasn't already reserved another car.
  - [R4] The system shall be able to associate a car to the user who has reserved it.
- [G3.1] User or guest should be able to search car near his position.
  - [R7] The system shall know cars' location according to cars' GPS.
  - [R8] The system shall know users' location according to his GPS.
  - [R17] The system shall provide the closest cars to user's position.
  - [D2] Users and guests positions provided by GPS are accurate.
  - [D1] Cars positions provided by GPS are accurate.
- [G3.2] User or guest should be able to search car near a selected position.
  - [R7] The system shall know cars' location according to cars' GPS.
  - [R8] The system shall know users' location according to his GPS.
  - [R18] The system shall provide the closest cars to position provided by the USER.
  - [D1] Cars positions provided by GPS are accurate.
- [G4.1] A discount of 10% should be applied to rides with at least two passengers.
  - [R19] The system shall bel able to detect if there are at least two passengers in the car.
  - [R20] The system shall be able to apply a discount on the ride.
  - [D5] Cars can detect how many passengers are on it.
  - [D3] Cars provide remote comunication API.
- [G4.2] If a car is left with no more than 50% battery empty, a discount of 20% should be applied to the last ride.
  - [R21] The system shall be able to know the battery level of the car.
  - [R20] The system shall be able to apply a discount on the ride.
  - [D3] Cars provide remote comunication API.
  - [D7] Cars can detect their battery level.

- [G4.3] If a car is left at more than 3 km from the nearest charging area, the system should charge 30% more on the last ride.
  - [R7] The system shall know cars' location according to cars' GPS.
  - [R21] The system shall be able to know the battery level of the car.
  - [R22] The system shall be able to apply a charge on the ride.
  - [R23] The system shall be able to calculate distance between car's position and the nearest charging area.
  - [D1] Cars positions provided by GPS are accurate.
  - [D3] Cars provide remote comunication API.
- [G4.4] If a car is left with more than 80% battery empty, the system should charges 30% more on the last ride.
  - [R21] The system shall be able to know the battery level of the car.
  - [R22] The system shall be able to apply a charge on the ride.
  - [D7] Cars can detect their battery level.
- [G4.5] If a car is left in a charging area and the user plugs the car into the power grid, a discount of 30% should be applied to the last ride.
  - [R24] The system shall be able to detect if the user has plugged the car into the power grid.
  - [R20] The system shall be able to apply a discount on the ride.
  - [R4] The system shall be able to associate a car to the user who has reserved it.
  - [D6] Cars can detect if they are plugged.
- [G4.6] If a car is not picked up within one hour from the reservation, the user pays a fee of 1 EUR.
  - [R25] The system shall be able to monitor car's reservations.
  - [R26] The system shall be able to know if a user has picked up a car.
  - [R2] The system shall be able to know car's status.
- [G4.7] If a payment fails for a insufficient money availability of the user, the user should be suspended from the service until the payment is done.
  - [R27] The system shall take care of the result of the payment.
  - [R28] The system shall suspend user that can't afford the payment.
  - [R29] The system shall unsuspend a user when his old undone payment is done.
- [G5] User have to pay an amount of money based on the ride's duration.

  - [R10] The system shall be able to calculate ride's cost based on duration.
  - [R30] The system shall charge an extenernal payment service with doing the payment.
  - [R22] The system shall be able to apply a charge on the ride.
  - [D4] Cars provide screen to display current charge to user.

- [G6.1] Only operator must be able to do car mantainance, knowing their position and status.
  - [R31] The system shall offer a backend panel to Operator
  - [R32] The system shall notify the operator when a car in his competence area needs some kind of maintenance.
  - [R1] The system can modify car's status (available, occupied, in use).
  - [R2] The system shall be able to know car's status.
  - [D7] Cars can detect their battery level.
  - [D8] Cars can detect information about maintenance needs.
  - [D9] If a user detect a malfuncion comunicate it.
  - [D10] If an operator is notified by the system about a maintenance need, he takes care of that.
  - [D11] Operators do not share their credential.

- [G6.2] User must be able to communicate to the system cars damages and malfunctions.
  - [R33] The system shall offer a functionality that let users to comunicate malfunctions and damages.

- [G7.1] Only admin must be able to manage safe areas.
  - [R34] The system shall offer the functionality of add or remove safe areas.
  - [R35] The system shall offer log in functionality for admin.
  - [D12] Operators do not share their credential.

- [G7.2] Only admin must be able to manage operators.
  - [R35] The system shall offer log in functionality for admin.
  - [R36] The system shall offer the functionality of add or remove safe areas
  - [D12] Operators do not share their credential.

3.1 Functional Requirements

After having defined main features of our system we can identify some functional requirements grouped under each defined actor:

- Guest, he can:
  - Sign up.
  - Search for a car near his location.
  - Search for a car near a selected location.
- User, he can:
  - Log in.
  - Reserve cars.
  - Use a reserved car (unlocking it).
  - View his profile (also his discounts).
  - Modify his profile.
  - View his recent reservations.
  - Search cars near to his location.
  - Search cars near a selected location.
  - Communicate malfunctions or damages to the system.
  - Pause a ride.
  - Restart a ride.
- Operator, he can:
  - Log in.
  - View car's status (battery and location).
  - Receive notification when a car need to be charged.
  - Set a car as available or under maintainance.
- Admin, he can:
  - Log in.
  - Add or delete safe areas.
  - Manage car status.
  - Consult car position.
  - Add or delete operators account.
  - Manage other admins.

3.2 Non-functional Requirements

3.2.1 User Interface

In order to make PowerEnJoy available for many people as possible, a Cross-Platform application has to be developed, with a Framework like Xamarin. In this way, a single application could be developed and then deployed for Windows Phone, Android and iOS. A Web Application has also to be developed in order to allow people to use our system in many way as possible.
The User Interface needs to be as user friendly as possible: that's to say that a user who uses Power EnJoy for the first time should be able to learn in a few seconds how to make use of it.
Screens here presented provide an example of the UI and the main functions of the system.

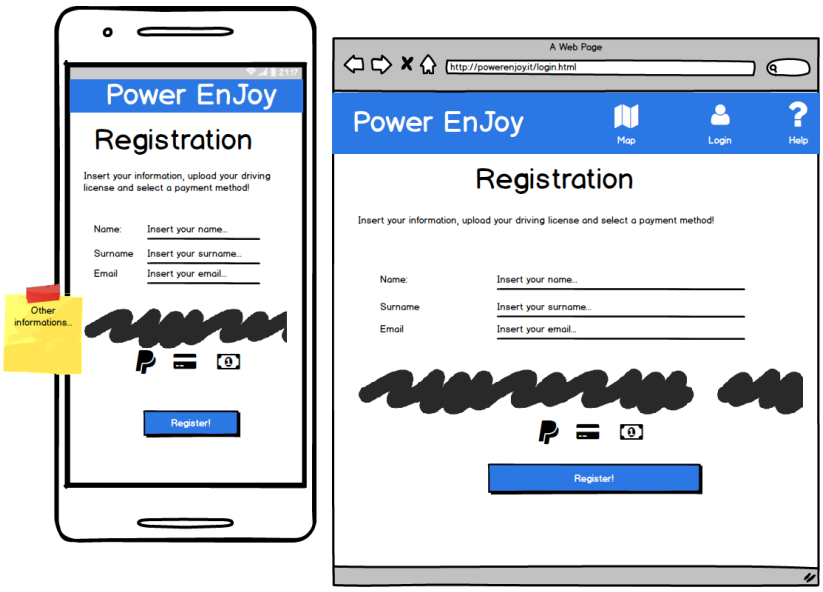


Figure 1: Registration

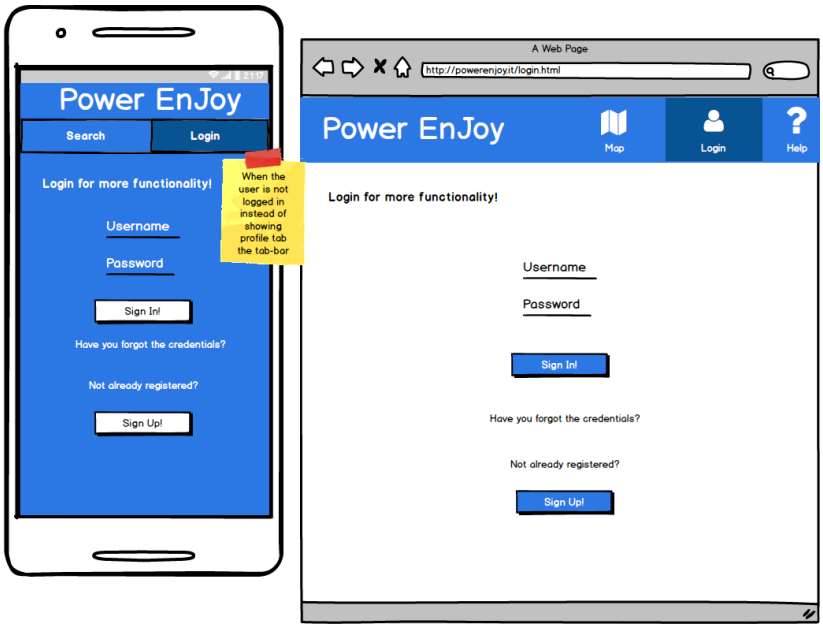


Figure 2: Login

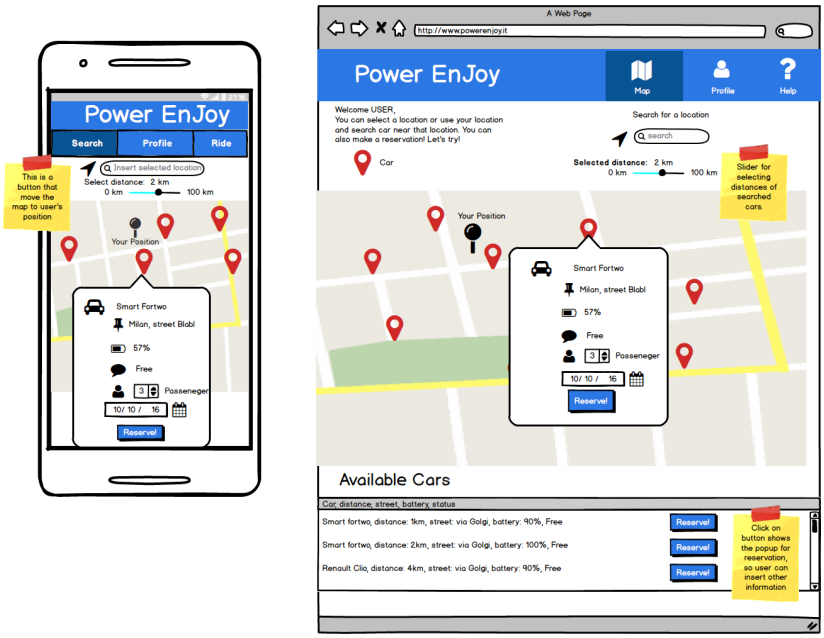


Figure 3: Map UI

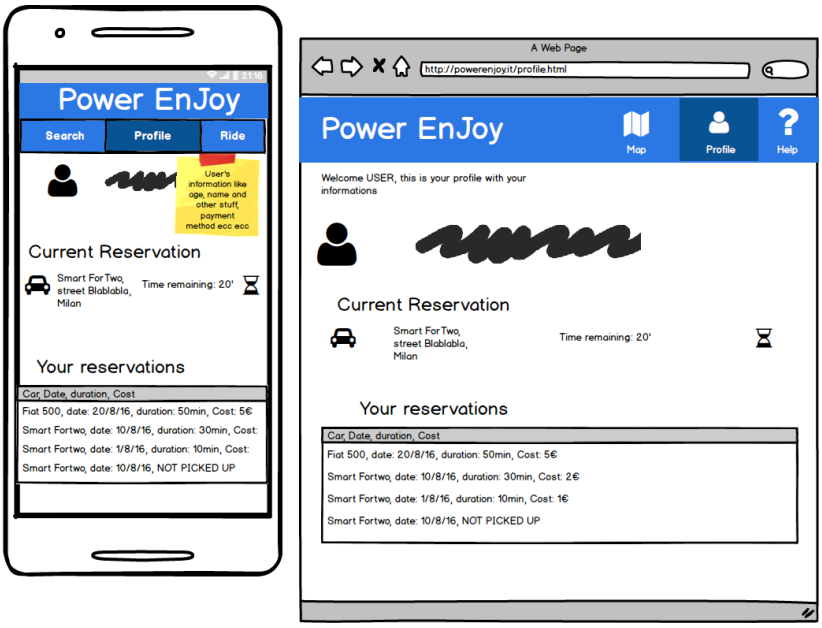


Figure 4: User Profile

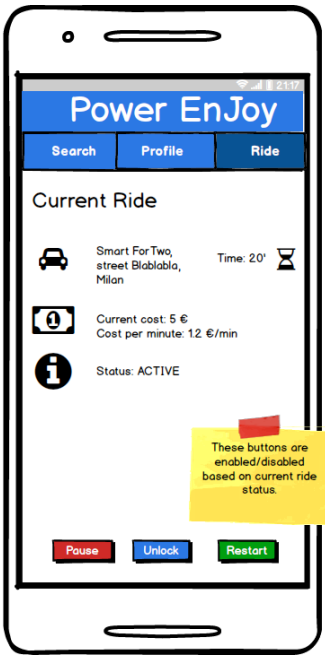


Figure 5: Ride UI

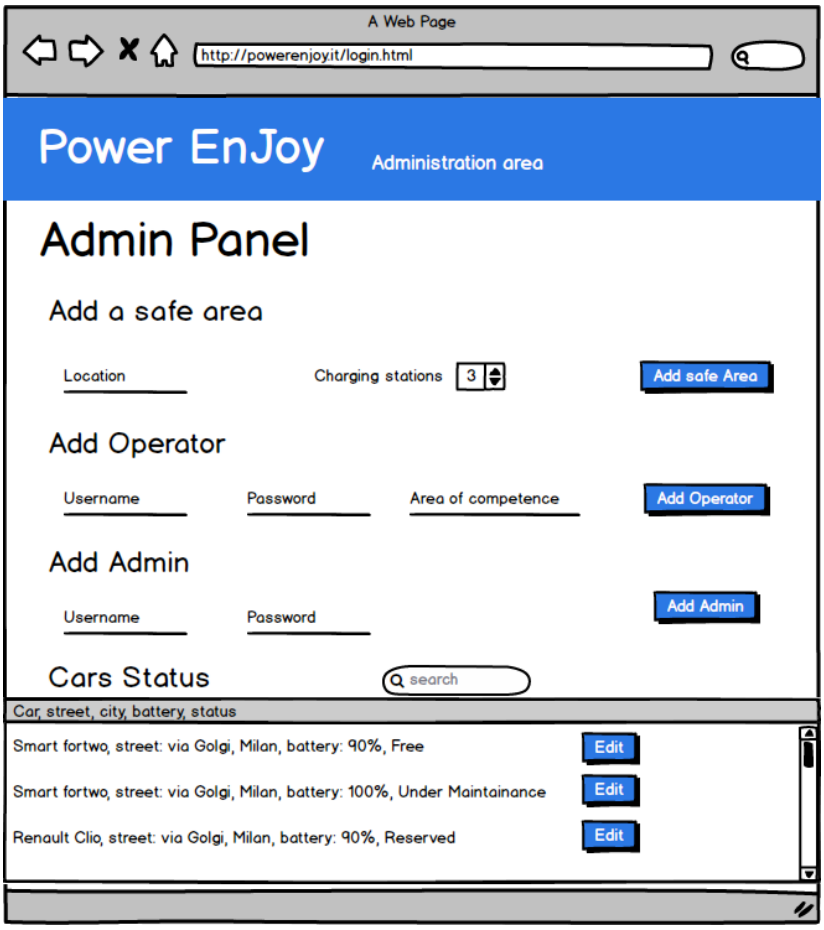


Figure 6: Admin Panel

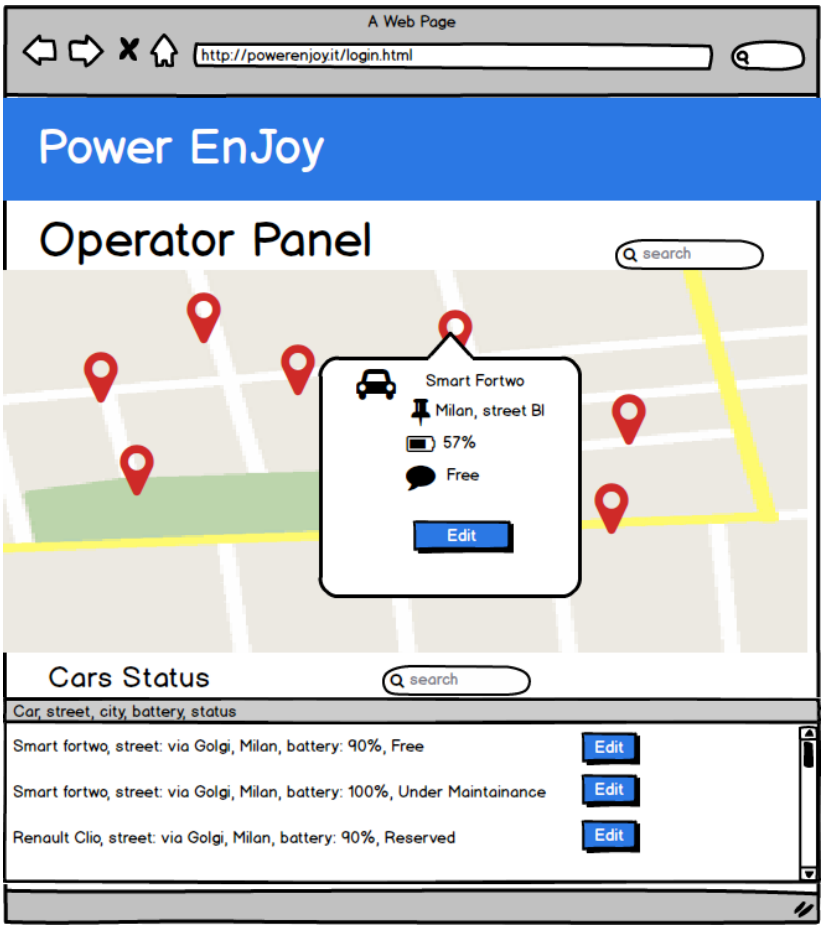


Figure 7: Operator Panel

3.2.2 Security

The system has to satisfy an high level of security regarding the communication protocol with the cars.
The communication with the cars needs to be encrypted in order to avoid unauthorized unlocks and use. Futhermore the system should have the way to stop cars when used in an unauthorized way.
Regarding the security of payment, the system does't care about it because it relies on external payment system.
Another critical security point is the managing of user's information: all user informations need to be encrypted before being sent.

3.2.3 Availability

The application needs to be online 24h/day and 7day/week in order to satisfy all users. This is a constraints because an hour of unavailablity could induce users to use another car sharing service. To achieve an high level of availability could be necessary to use a dedicated server and all the system could be hosted in a cloud platform like OVH. The system should be able to support a high number of users connected at the same time and could be necessary to scale resources depending on this.

3.2.4 Portability

The client-part of the system (application) could be used on any mobile OS in order to make it available for as many people as possible. It is expected a Web Application to increase the possible number of users.
The server-part of the system could be used on any OS which supports JVM and DBMS.

3.2.5 Maintainability

The system code will be documented in order to make easier future improvements or changes by other developers. It needs to be clear how the system works, how the system communicate with cars, and how the system has been developed.

4 Scenario Identifying

4.1 Scenario 1 - Reservation

Paul, a businessman, has to go to Milan by train in order to participate to an important meeting.
As the office where they have to meet is 5 km far from the railway station and taxis are too much expensive, he decides to use *PowerEnJoy* from his smartphone.
He notices, thanks to the map on the app, that there is a parking area next to the railway station with a lot of available cars. He then reserves one of these electric cars so that, as soon as he arrives there, he'll be able to reach the office very quickly and in a green way.

4.2 Scenario 2 - Two Passengers discount

Paul is a student that goes to Politecnico University in Milan. He lives in a city near Milan but unfortunately train's schedule is very prohibitive: he should wake up at 5 o'clock in the morning in order to arrive on time at the lecture that starts at 8:00.
He found out about *PowerEnJoy* from two of his friends in his city that attend the same University. They decide to use together car-sharing service so they can save money using the "two passegners discount".
Since their city is full of electric-car, everyday they make a reservation of a car and go to the University together.
In this way, not only they save money, but they are also much more comfortable in reaching the University.

4.3 Scenario 3 - Registration

Paul has just learnt about *PowerEnJoy*.
This evening he has to go to dinner with his girlfriend, Sarah, so he decides to try it. Then he downloads the app and looks for cars near his location. He finds many cars, he's about to reserve one of them but reservation is not permitted to guests.
For this reason he has to sign up: following the application instructions he provides his personal informations, his driving license and he specifies his payment method.
After that he's able to reserve the car for the special occasion.

4.4 Scenario 4 - Operator maintenance

Jonathan is doing his periodical check of the car in his competence area. He finds out that a dishonest user has left a car with a broken light, without communicate that. So Jonhatan, by his maintenance service system, makes the car unavailable and starts to solve the maintenance problem.

4.5 Scenario 5 - Operator notification

Amy is driving an electric car reserved with *PowerEnJoy*. As always she is very late today, and she notices that, if she wants to arrive on time, she will not be able to bring the car at the station where it can be charged. So she decides to park the car as close as possible to her destination.
After she has parked, the operator Jonathan, receives a notification on his maintenance service view, that tells him where Amy's car is, and that it is almost completely discharged.
Now Jonathan can go to the car and make it available again for the next customer.

4.6 Scenario 6 - User comunicates a malfunction to the system

Amy is driving an electric car reserved with *PowerEnJoy*. Arrived at her destination, she starts a parking maneuver, but she hurts a light pole, doing a damage to the vehicle. Amy is an honest person, and following the application instruction she communicates to the system that she has damaged the car. The system will communicate to an operator about the damage.
Paul is an operator of Power EnJoy, when Amy communicates the damages he receives a notification in the operator panel.
Paul, through the operator panel can see car location and can edit car status. He changes the status into "under maintainance", in this way no one can reserve the damaged car.
Then he goes to the car and repairs it. After the reparation he edits the status into "Available" so the car in reservable by users.

4.7 Scenario 7 - Pausing a ride

Paul is going to the supermarket with his fiancée with an elecrtic car. They know that it will not take to much time.
As soon as they arrived, Paul, using a *PowerEnJoy* mobile application functionality, pauses the ride. In this way, they manage to do shopping without losing their car reservation. During the shopping they are in a hurry, because they are going to pay also the pausing time, and the pause expires after a period.
After the shopping, they return to the car, and Paul unlock that restarting the ride.

5 UML Models

5.1 Use Case Diagram

With the Use Case Diagram are highlighted the main functions of our system and the interactions between actors and the related Use Cases.

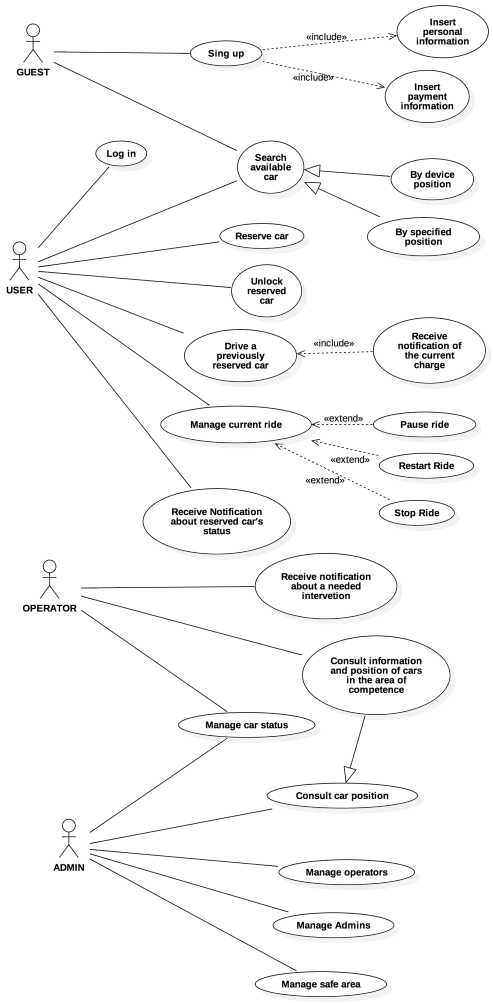


Figure 8: Use Case Diagram

5.2 Sequence Diagram

With Sequence Diagram are highlighted the main functions of the system. Are described the High-Level sequences of actions focusing on the interaction between actors and system.

5.2.1 Registration

Name	Registration
Actors	Guest
Goal	All
EntryConditions	The guest isn't already registered to the application.
EventFlow	1. Guest opens the app. 2. Guest click on Sign Up button. 3. Guest fills in all mandatory fields like payment informations, his vital statistics etc. 4. Form is sent to the server that saves all data in the DB. 5. Email with registration confirmation is sent by the server to the email address provided in the form.
Output Condition	The guest ends registration procedure and become a user. Now he/she can login and use all functionalities offered by the system.
Exception	1. Payment informations are not valid. 2. Driving License in not valid. 3. Username is already used by another user. 4. One or more mandatory fields of the form are not valid. In all this case the visitor is alert and the app explains the problem. Then the app goes back to point 3 of Event Flow.

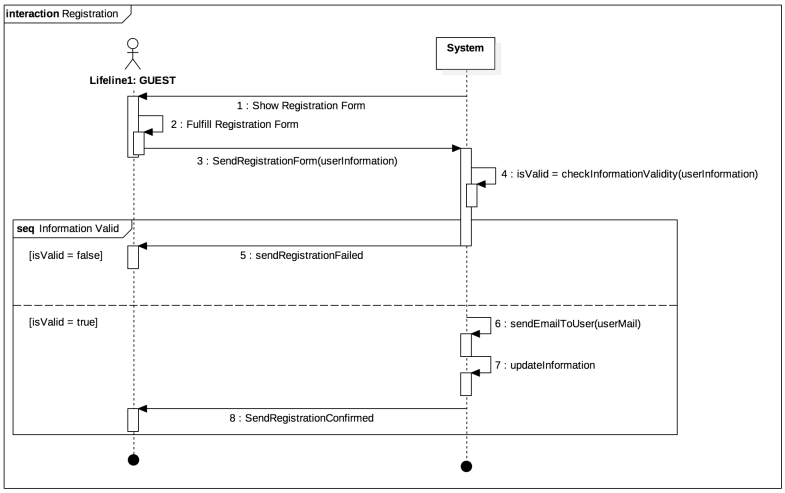


Figure 9: Registration

5.2.2 Login

Name	Login
Actors	User
Goal	All
EntryConditions	The user is registered into the system
EventFlow	1. User opens the app. 2. App shows login page. 3. User inserts his email and his password. 4. Email and password are sent to server in a secure way. 5. Server verifies user's credentials and confirms login.
OutputCondition	User can use the app, reserve cars and consult his profile.
Exception	1. User's credentials are incorrect. 2. User is banned to the system (due to an incorrect behaviour). In the first case the user is notified that his/her credentials are incorrect. In the second case the user is notified that he can't use the service because of his behaviour.

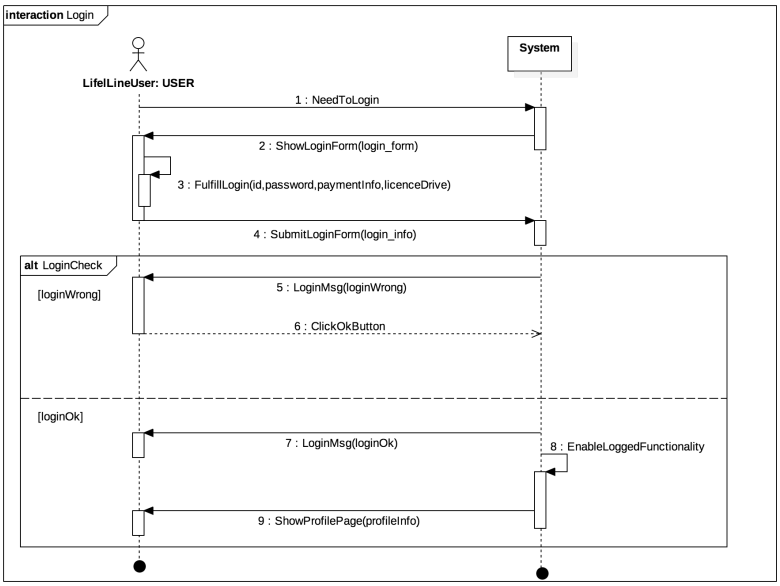


Figure 10: Login

5.2.3 Car Reservation

Name	Car Reservation
Actors	User
Goal	[G1.1]
EntryConditions	The user is registered into the system and has already logged in.
EventFlow	1. User selects a car. 2. User fills the reservation form with the reservation hour. 3. Form is sent to the server. 4. Server verifyies the form and the validity of reservation. 5. Server updates his informations with the new reservation. 6. App confirms reservation to the user. 7. App adds to current user's reservation the actual reservation.
OutputCondition	User has reserved a car and he can use it.
Exception	1. Car is already reserved by another user. 2. User has already reserved another car. In the first case app notifies the user that he needs to select another car in order to make his reservation. App returns to point 1 of Event Flow. In the second case appp notifies user that he has already reserved another car and that he can't reserve more than one car at the same time.

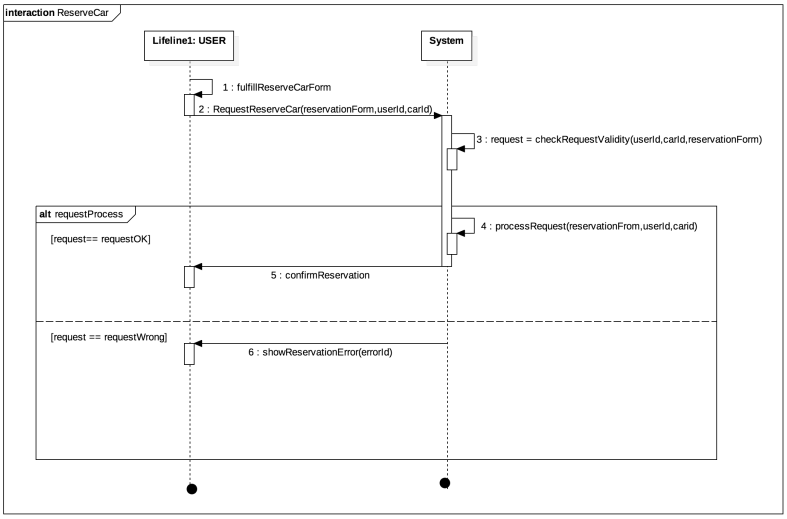


Figure 11: Car Reservation

5.2.4 Search Cars

Name	Search for Cars
Actors	User or guest
Goal	[G3.1], [G3.2]
EntryConditions	NULL
EventFlow	1. User opens the app. 2. User select the search method: search car near his position or near a selected position. 3. User inserts the needed parameter. 4. The request is sent to the server. 5. Server searches for cars that satisfy requested parameter and sends them. 6. The App shows the received information.
OutputCondition	User or Guest is aware of cars position near to the requested position.
Exception	No exception.

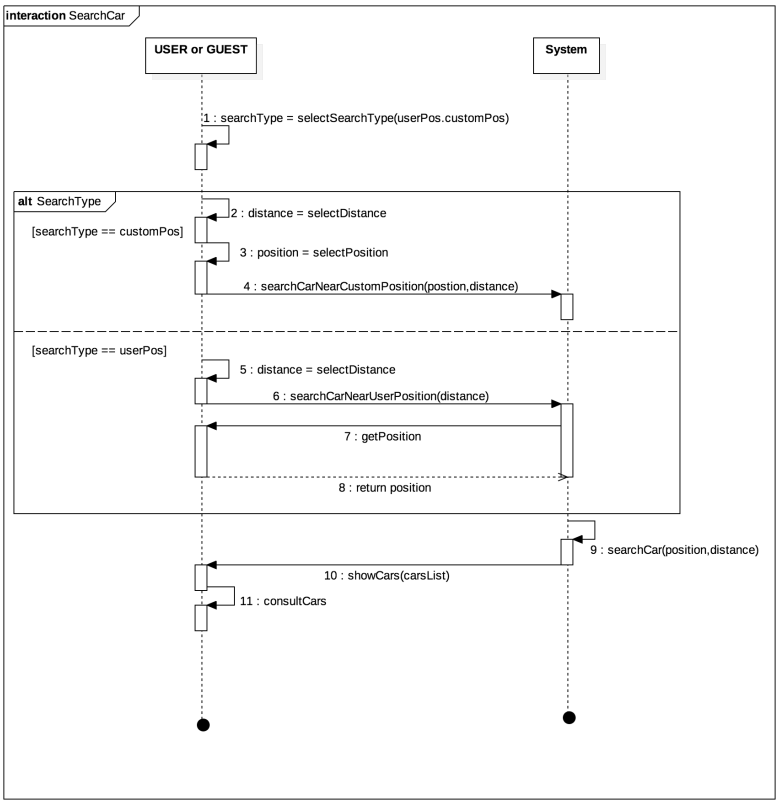


Figure 12: Search for a car

5.2.5 Unlock Car

Name	Unlock Car
Actors	User
Goal	[G1.2]
EntryConditions	The user has already reserved a car or has paused a ride.
EventFlow	1. User opens the app. 2. User make the request, so the client application sends user position to the server. 3. Server search for the position of the user reserved car. 4. Server check if the distance between car and user is acceptable. 5. If acceptable the car is unlocked.
OutputCondition	The user has unlocked is reserved car.
Exception	1. The user has not reserved any car. 2. The user is not enough close to the car. In the first case the error is shown to the user. In the second case the app shows to the user that he has to go near the car if he want to unlock it.

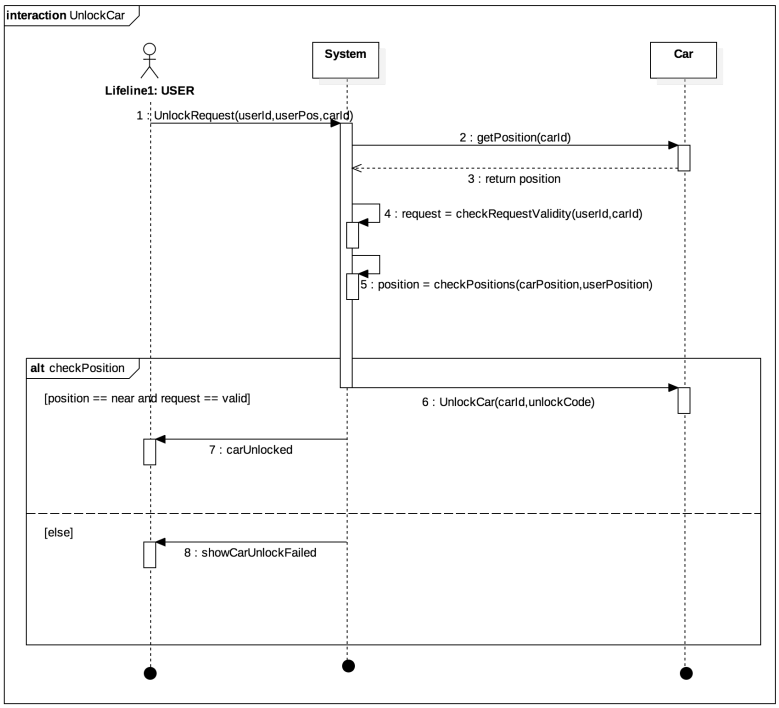


Figure 13: Unlock Car

5.2.6 Communicate Damages And Malfucntions

Name	Communicate Damages And Malfunctions
Actors	User, operator
Goal	[G6.2]
EntryConditions	NULL
EventFlow	1. User opens the app. 2. User fills the form with description of the damage and identification of the car. 3. If the form is accepted the server sent an acknowledge to the user and notificate the operator wich competence area contains the car. 4. If it's a real malfunction the operator change status car to Under Maintenance
OutputCondition	The user has comunicated a damage to a car or a malfunction. The car status is Under Maintenance.
Exception	1. The form is not filled correctly. 2. The car hasn't a malfunction. In the case (1) the request is refused and the user is suggested to check his form. In the case (2) the operator doesn't change car status.

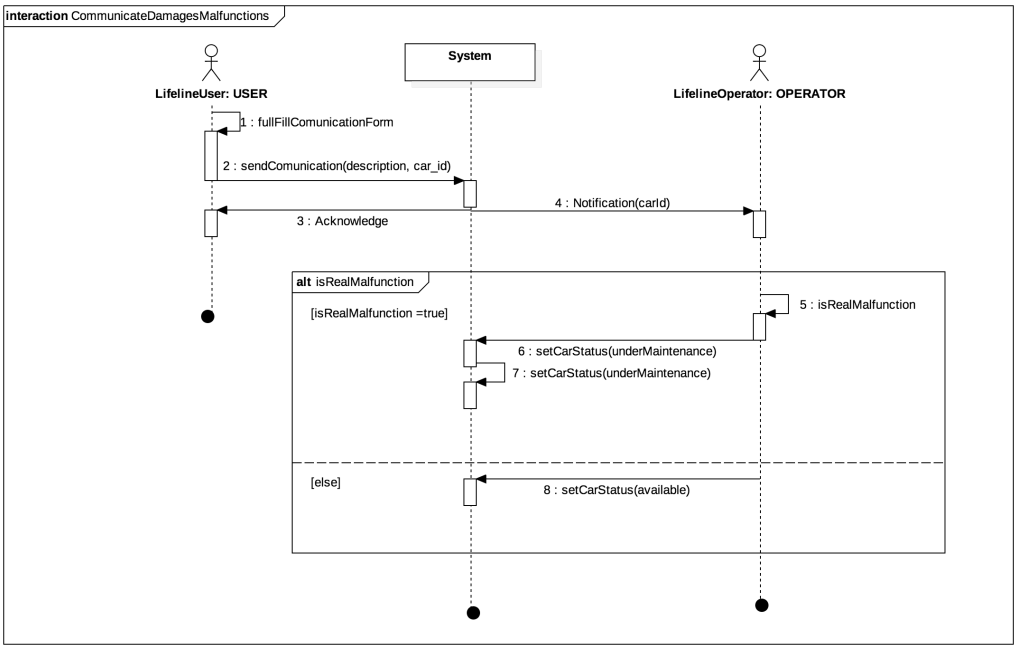


Figure 14: Communicate Damages And Malfunctions

5.2.7 End of the Ride

Name	Communicate Damages And Malfunctions
Actors	User, External Payment service
Goal	[G1], [G5]
EntryConditions	User has reserved and is using the car
EventFlow	1. User click on STOP ride button 2. System checks the validity of the request 3. System locks the car. 4. System asks to the external payment service to complete the payment of the user after having calculated discount and charges. 5. Payment service returns the result of the payment. 6. System notifies payment result.
OutputCondition	The user finishes his ride and the payment is done.
Exception	1. User can't afford the payment. In this case the user is suspended until he can't pay.

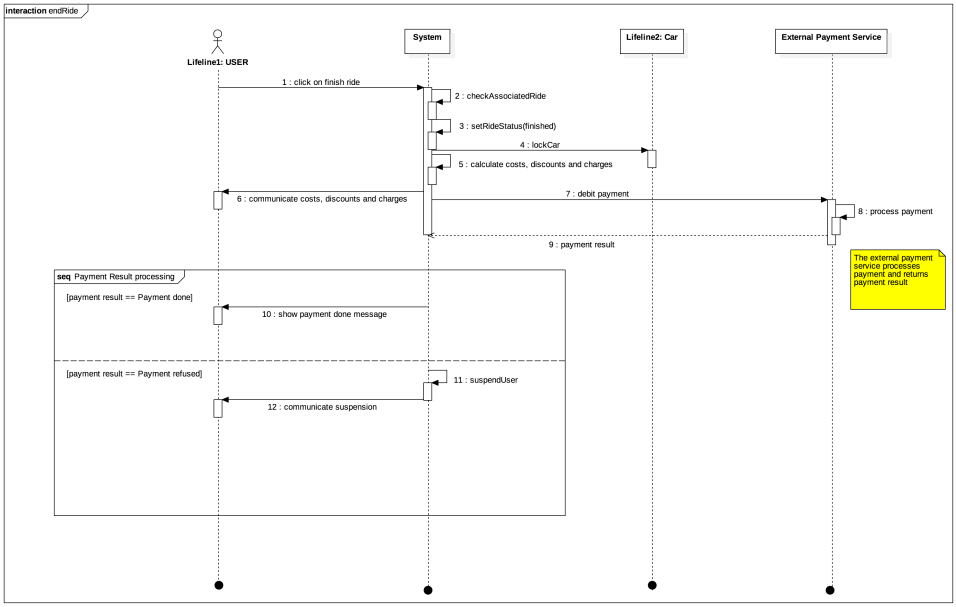


Figure 15: End of the ride

5.2.8 Malfunction detection and notification

Name	Malfunctions detection and Notifications
Actors	User, Operator, Car
Goal	[G1.4]
EntryConditions	NULL
EventFlow	1. System periodically check status of cars. 2. If a possible malfunction has been found by car the list of malfunctions is sent to system. 3. The list of possible malfunctions is sent to Operators. (a) If the operators find a real malfunctions set car status to Under Maintenance. (b) else (if there isn't no real malfunction) the system tells to the car that there isn't no real malfunction. 4. If the car is Reserved the system notifies user that reserved it and deletes the reservation.
OutputCondition	The car has changed state to Under Maintenance and the user that has eventually reserved it is notified. Operator can do car maintenance.
Exception	1. No malfunction has been found. If no malfunction has been found the system continuosly check status of cars.

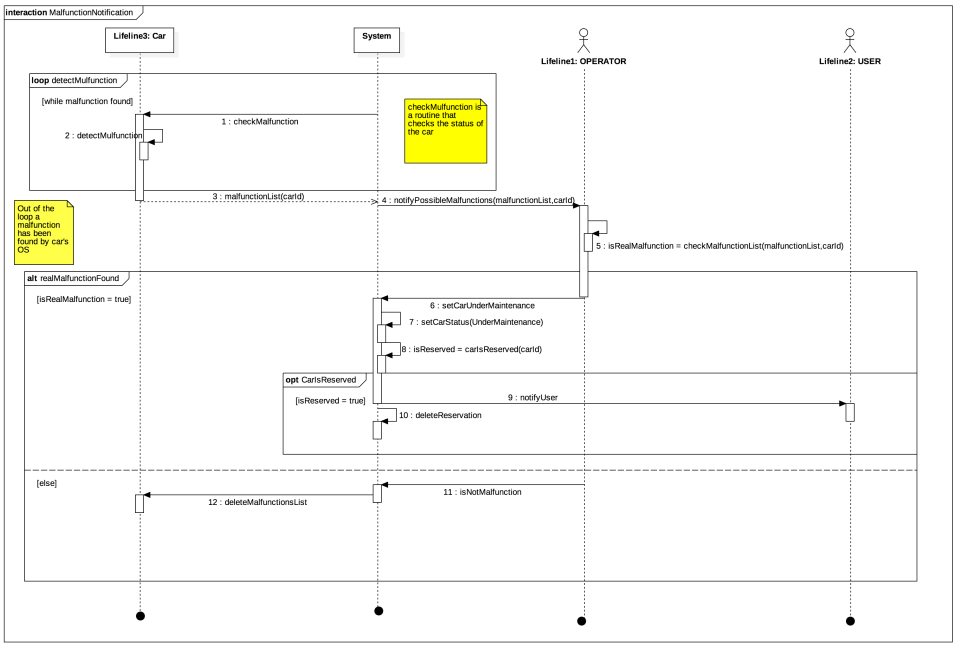


Figure 16: Malfunctions detection and notifications

5.3 Statechart Diagram

5.3.1 Car State Machine

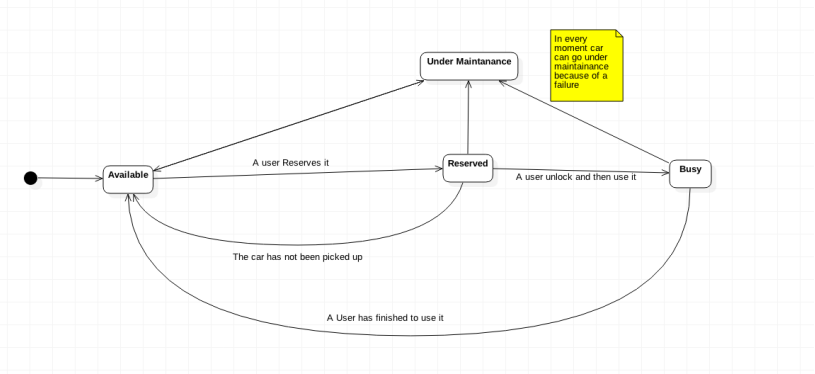


Figure 17: Car State Machine

5.3.2 User State Machine

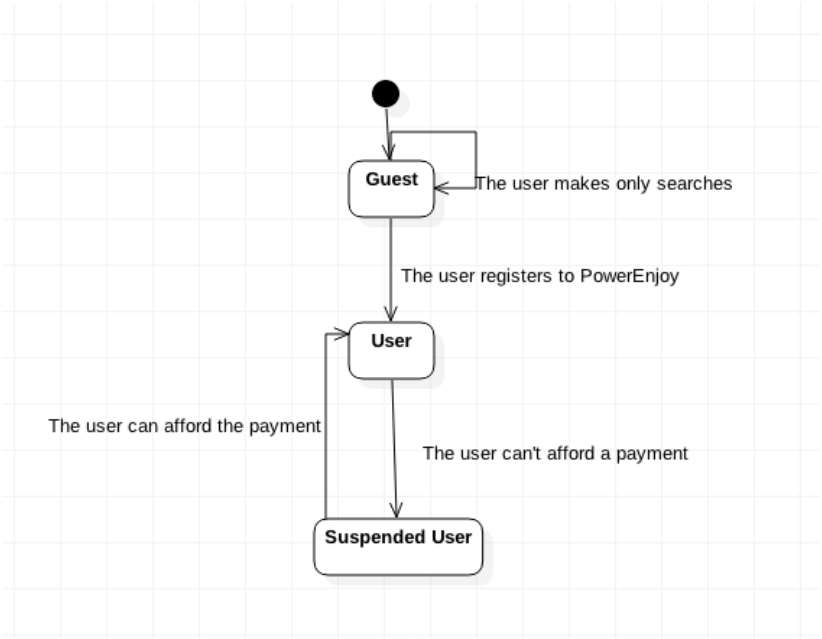


Figure 18: User State Machine

5.4 Class Diagram

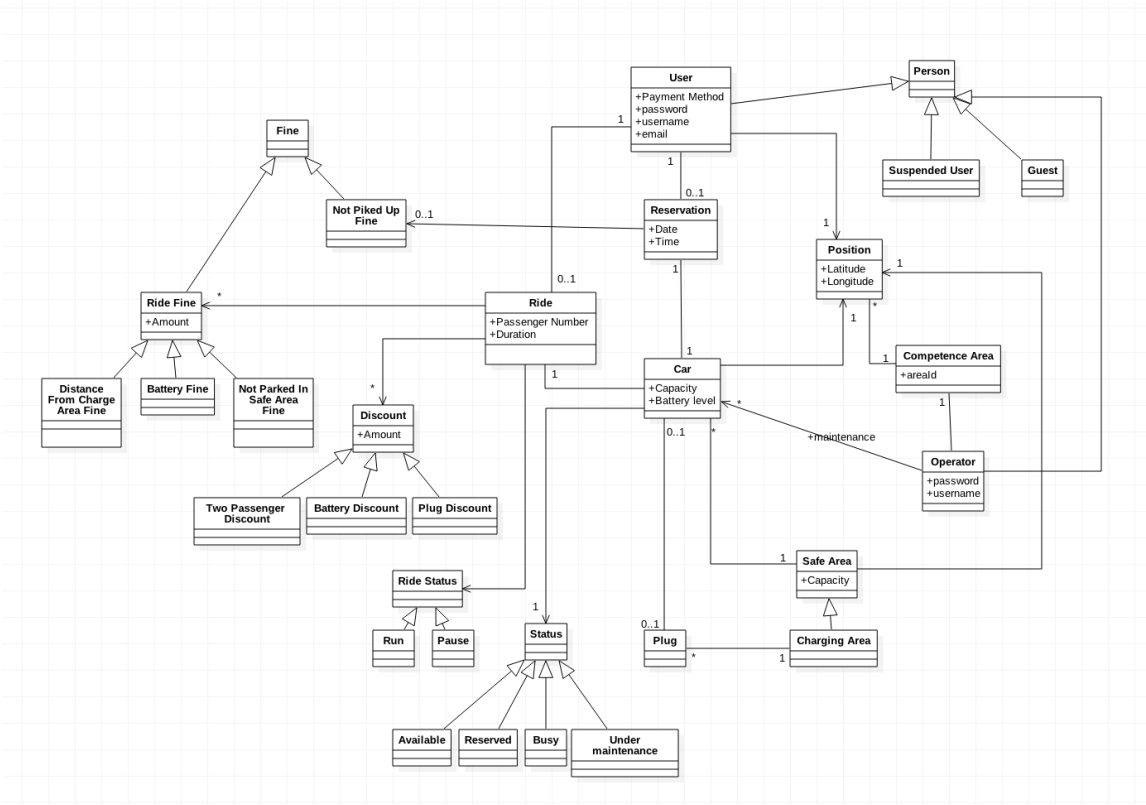


Figure 19: Class Diagram

6 Alloy Modeling

The purpose of the Alloy Modeling is to check the consinstency of the system model.
The abstraction level is fundamental in this point and here aren't described implementative or low level details.
Discounts and charges haven't been modeled because these two details could augment too much the complexity of the model without adding significative information about system status.

6.1 Model

```
1  /*-------UTIL-----------*/
2  sig Position{}
3  /*-------PERSON---------*/
4  abstract sig Person{}
5
6  sig User extends Person{
7          userPosition: one Position,
8          reservedCar: lone Car,
9          inUseCar: lone Car,
10         canUnlock: lone Car
11 }
12 {
13         #reservedCar = 1 implies #inUseCar=0
14         #inUseCar = 1 implies #reservedCar = 0
15 }
16
17 sig SuspendedUser extends User{} {
18         #reservedCar = 0
19         #inUseCar = 0
20 }
21
22 sig Admin extends Person{}
23
24 sig Operator extends Person{
25         competenceArea: one CompetenceArea,
26         carInMaintenance: set Car
27 }
28 {
29
30
31         all c : Car | c in carInMaintenance iff c.status
               = UnderMaintenance
32         all c : Car | c in carInMaintenance
```

```
33                  implies c.carPosition in competenceArea.
                        positions
34
35
36  }
37
38  /*-------CAR-------*/
39
40
41  sig Car{
42          connectedPlug: lone Plug,
43          status: one CarStatus,
44          carPosition: one Position
45   }
46  {
47          this in User.reservedCar iff status in Reserved
48          this in User.inUseCar iff status in Busy
49          status = Busy implies this not in SafeArea.
                carsInSafeArea
50          // car can be plugged only if it's in a charging
                area
51
52          #connectedPlug = 1 implies (connectedPlug.~plugs)
                .safeAreaPosition = carPosition
53          #connectedPlug = 1 implies not status in Busy
54   }
55
56  /*-------CAR STATUS----------*/
57  abstract sig CarStatus{}
58
59  one sig Reserved extends CarStatus{}
60
61  one sig Available extends CarStatus{}
62
63  one sig Busy extends CarStatus{}
64
65  one sig UnderMaintenance extends CarStatus {}
66
67  /*-------AREA----------*/
68
69  sig CompetenceArea{
70          positions : set Position
71  }{
72          // each competence area belongs to an operator
73          this in Operator.competenceArea
74          #positions >0
```

```
75  }
76
77  sig SafeArea{
78          capacity: Int,
79          carsInSafeArea : set Car,
80          safeAreaPosition : one Position
81  }
82  {
83          capacity>0
84          #carsInSafeArea <= capacity
85  }
86
87  sig ChargingArea extends SafeArea{
88          plugs: set Plug,
89  }
90  {
91          #plugs>0
92          #plugs<=capacity
93  }
94
95  sig Plug{
96          carConnected: lone Car
97  }
98  {
99          all p:Plug | p in ChargingArea.plugs
100 }
101
102 /*-------RIDE----------*/
103
104 sig Ride{
105         rideCar: Car,
106         rideUser: User,
107         rideStatus: RideStatus
108 }
109 {
110 /* If the status of the ride is Run, user and his used
       car
111 have the same position */
112         rideStatus in Run implies rideCar.carPosition =
              rideUser.userPosition }
113
114 /*-------RIDE STATUS----------*/
115
116 abstract sig RideStatus{}
117 one sig Run extends RideStatus{}
118 one sig Pause extends RideStatus{}
```

6.2 Facts

```
1  /* Different users have different cars in use and
       different
2  reserved cars
3  */
4
5  fact {
6          no disj u1,u2:User | u1.inUseCar = u2.inUseCar
               and #u1.inUseCar=1
7          no disj u1,u2:User | u1.reservedCar=u2.
               reservedCar and #u1.reservedCar = 1
8  }
9
10 /* A user can unlock a car if is near that,
11 and has reserved that or is in a pause ride
12 */
13
14 fact{
15 // can unlock only if reserved and same position
16         canUnlock = (reservedCar + (rideStatus.Pause).
               rideUser <: inUseCar) & userPosition.~
               carPosition
17 }
18
19 // Different operators have disjoint competence area
20
21 fact{
22         all disj o1,o2:Operator | o1.competenceArea != o2
               .competenceArea
23 }
24
25 // If a car is in a safe area, they have the same
       position
26
27 fact {
28         all c : Car, safeArea: SafeArea | c in safeArea.
               carsInSafeArea iff c.carPosition = safeArea.
               safeAreaPosition
29  }
30
31
32 // CompetenceAreas.positions induces a partition on
       Position
33
34 fact {
```

```
35          // Empty intersection
36          all disj c1,c2: CompetenceArea | c1.positions &
                c2.positions = none
37          // There doesn't exist position that are not in a
                competence area
38          all p:Position | p in CompetenceArea.positions
39  }
40
41  // There doesn't exist different safe areas with same
        position
42  fact {
43          no disj s1,s2: SafeArea | s1.safeAreaPosition =
                s2.safeAreaPosition
44  }
45
46
47  // a plug is only in a one charging area
48  fact plugUnique{
49          no disj c1,c2: ChargingArea | (some p:Plug |p in
                c1.plugs and p in c2.plugs)
50  }
51
52  // connectedPlug is the same of carConnected transposed
53  fact {
54          connectedPlug = ~carConnected
55  }
56
57  // Set user and car in Ride like in the relation User.
        inUseCar
58  fact {
59          (~rideUser).rideCar = inUseCar
60          #Ride = #inUseCar
61  }
```

6.3 Dynamic Modeling

```
1   /*-------DYNAMIC MODELING PREDICATES----------*/
2
3   //From available to reserved
4   pred carFromAvailableToReserved [u: User, c: Car, u1:
        User, c1: Car] {
5           //initial conditions for reserving
6           not u in SuspendedUser
7           #u.reservedCar = 0
8           #u.inUseCar = 0
9           c.status in Available
```

```
10          //informations non involved in reservation don't
                change
11          c1.connectedPlug = c.connectedPlug
12          c1.carPosition = c.carPosition
13          u1.userPosition = u.userPosition
14          u1.inUseCar = u.inUseCar
15          //final conditions
16          c1.status in Reserved
17          u1.reservedCar = c1
18  }
19
20  //From reserved to busy
21  pred carFromReservedToInUse [u: User, c: Car, u1: User,
        c1: Car] {
22          //initial conditions
23          u.reservedCar = c
24          #u.inUseCar = 0
25          u.canUnlock = c
26          //final contition
27          #c1.connectedPlug = 0
28          u1.inUseCar = c1
29          c1.status in Busy
30          all r: Ride | r.rideUser = u1 implies r.
                rideStatus in Run
31  }
32
33  //From available to under maintenance
34  pred carFromAvailableToUnderMaintenance [c: Car, c1: Car]
        {
35          //initial conditions
36          c.status in Available
37          //not involved informations don't change
38          c1.connectedPlug = c.connectedPlug
39          c1.carPosition = c.carPosition
40          //final condition
41          c1.status in UnderMaintenance
42  }
43
44  // From busy to available
45  pred carFromBusyToAvailable [c: Car, c1: Car, u:User, u1:
        User]{
46          // initial condition
47          c.status in Busy
48          c in u.inUseCar
49          // not involved information don't change
50          u1.userPosition = u.userPosition
```

```
51          c1.carPosition = c.carPosition
52          not u1 in SuspendedUser
53          #u1.inUseCar = 0
54          // final condition
55          c1.status in Available
56  }
57
58  //Plugging a car
59  pred pluggingCar [c: Car, c1: Car, u1: User] {
60  //the user is required to ensure that car.~reservedCar
        does not change
61          //initial condition
62          #c.connectedPlug = 0
63          c in ChargingArea.carsInSafeArea
64          //not involved informations don't change
65          c1.carPosition = c.carPosition
66          c1.status = c.status
67          c in User.reservedCar implies (all u: User | u.
                reservedCar = c implies (u1.userPosition = u.
                userPosition and u1.inUseCar = u.inUseCar and
                u1.reservedCar = c1 and u1 not in
                SuspendedUser))
68          //final condition
69          #c1.connectedPlug = 1
70  }
71
72  //Stopping a run (to pause status)
73  pred rideFromRunToPause [r: Ride, r1: Ride, u: User, u1:
        User, c: Car, c1: Car] {
74          //initial condition
75          r.rideStatus in Run
76          r.rideUser = u
77          r.rideCar = c
78          //informations that do not change
79          r1.rideCar = c1
80          r1.rideUser = u1
81          c1.connectedPlug = c.connectedPlug
82          c1.status = c.status
83          c1.carPosition = c.carPosition
84          u1.userPosition = u.userPosition
85          //final condition
86          r1.rideStatus in Pause
87  }
88
89  // Restart a run
90  pred rideFromPauseToRun [r: Ride, r1: Ride, u: User, u1:
```

```
    User, c: Car, c1: Car] {
91          //initial condition
92          r.rideStatus in Pause
93          r.rideUser = u
94          r.rideCar = c
95          //informations that do not change
96          r1.rideCar = c1
97          r1.rideUser = u1
98          c1.connectedPlug = c.connectedPlug
99          c1.status = c.status
100         c1.carPosition = c.carPosition
101         u1.userPosition = u.userPosition
102         //final condition
103         r1.rideStatus in Run
104 }
```

6.4 Assertions

```
1  /*------- ASSERTION ----------*/
2
3  // RESERVATION
4
5  // Two different user can't reserve the same car
6  assert noTwoUserSameReservedCar{
7          no c: Car, disj u1,u2: User | c in u1.reservedCar
               and c in u2.reservedCar
8  }
9
10 // A user can't reserve more than one car
11 assert noUserTwoReservedCar {
12         no u: User | #u.reservedCar > 1
13 }
14
15 // If it exists a user that has reserved a car, that car
      status is reserved
16 assert reservedStatus{
17         all c: Car | (some u: User | u.reservedCar = c)
              implies c.status in Reserved
18 }
19
20 // A suspended user can't reserve a car
21 assert noReservationBySususpended {
22         no c: Car | (some u: SuspendedUser | u.
              reservedCar = c)
23 }
24
```

```
25 // A suspended user can't use a car
26 assert noUsedBySususpended {
27         no c: Car | (some u: SuspendedUser | u.inUseCar =
               c)
28 }
29
30 // UNLOCK
31
32 // Users can unlock cars iff the car is in their paused
       ride, or if they have reserved it, and they are near
       it
33 assert canUnlockAssert {
34         all u : User, c : Car | (u.userPosition = c.
               carPosition and u.reservedCar = c) implies u.
               canUnlock = c
35         all r: Ride, u: User, c: Car | (r.rideStatus =
               Pause and r.rideUser = u and u.userPosition =
               c.carPosition and r.rideCar = c) implies u.
               canUnlock = c
36 }
37
38 // UTILIZATION OF CARS
39
40 // A user can't use two machine
41 assert noUserUseTwoCar {
42         no u: User | #u.inUseCar > 1
43 }
44
45 // A user can't use a car reserved by another user
46 assert noUseOfReservedCar {
47         no u: User, c: Car | u.reservedCar = c and c.
               status in Busy
48 }
49
50 //A user can't use a car that is under maintenance
51 assert noUseCarInMaintenance {
52         no u: User, c: Car | u.inUseCar = c and c.status
               in UnderMaintenance
53 }
54
55 // If exist a user that is using a car, that car status
       is Busy
56 assert busyStatus {
57         all c: Car | (some u: User | u.inUseCar = c)
               implies c.status in Busy
58 }
```

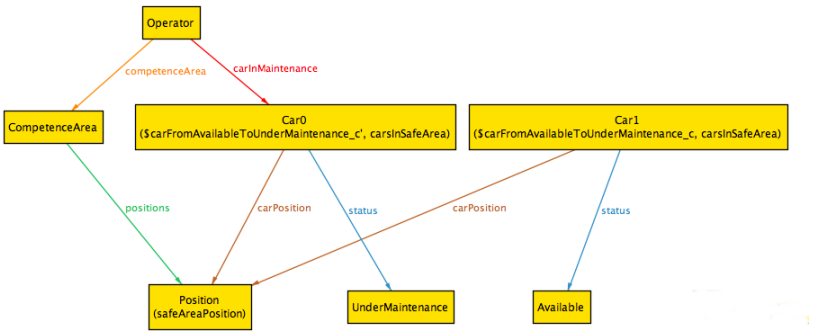
```
59
60 // A car is busy iff exists a ride with that car
61 assert busyCarInARIde {
62         all c: Car | c.status = Busy iff (some r: Ride |
               r.rideCar = c)
63 }
64
65 //MAINTENANCE OF CARS
66
67 // If a car is under maintenance, there exists an
       operator that is repairing it
68 assert inMaintenanceImpliesExistOperator{
69         all c: Car | (c.status in UnderMaintenance)
               implies (some o: Operator | c in o.
               carInMaintenance)
70 }
```

6.5 Run Commands

```
1  run carFromAvailableToReserved
2  run carFromReservedToInUse
3  run pluggingCar
4  run carFromAvailableToUnderMaintenance
5  run carFromBusyToAvailable
6  run rideFromRunToPause
7  run rideFromPauseToRun
8  run show
9  check canUnlockAssert
10 check inMaintenanceImpliesExistOperator
11 check busyCarInARIde
12 check noTwoUserSameReservedCar
13 check noUserTwoReservedCar
14 check reservedStatus
15 check noReservationBySususpended
16 check noUsedBySususpended
17 check noUserUseTwoCar
18 check noUseOfReservedCar
19 check busyStatus
```

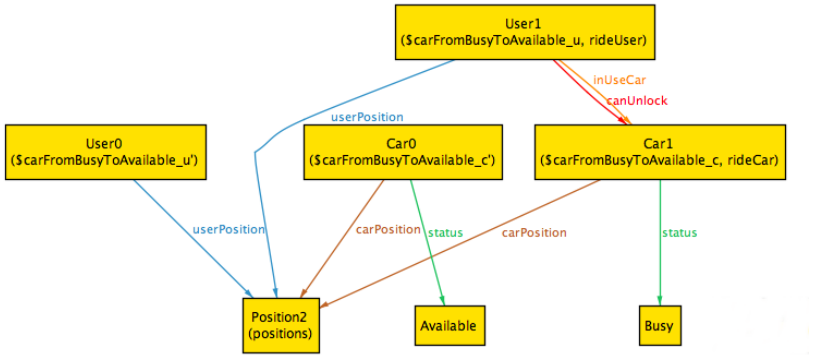
6.6 Generated Worlds

6.6.1 Car changing of status



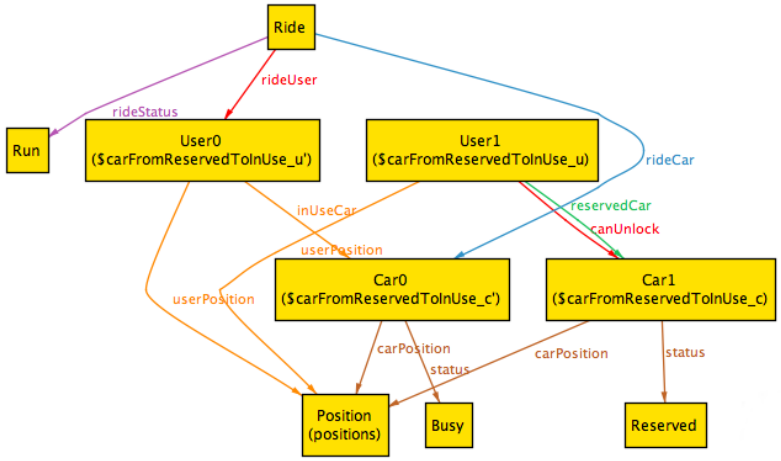
Car1, in the competence area of Operator, is available. After an hypotetic failure, it goes under maintenance (it becomes Car0), and Operator takes car of it.

Figure 20: From Available to Under Maintenance



User1 is using Car1, so Car1 status is Busy. After the ride, Car1 becomes Car0, and the satus becomes Available. User1 becomes User0 with no car in use.

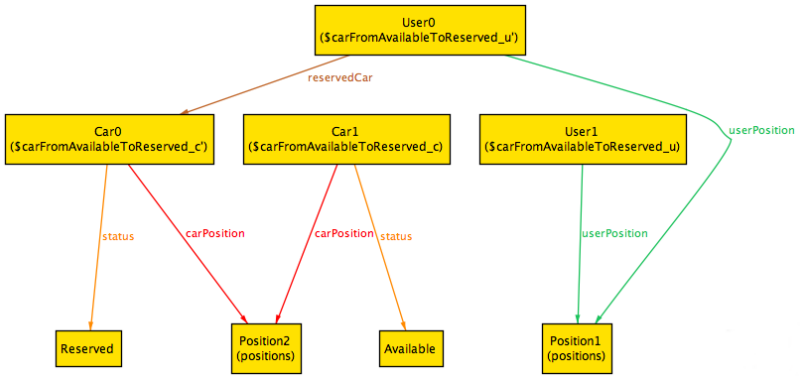
Figure 21: From Busy to Available



User1 has as reserved car Car1 and they are in the same position, so User1 can unlock Car1. After, User1 becomes User0, Car1 become Car0 with status Busy, and User0 is using Car0. The corresponding Ride has been generated.

Figure 22: From Reserved to in Use

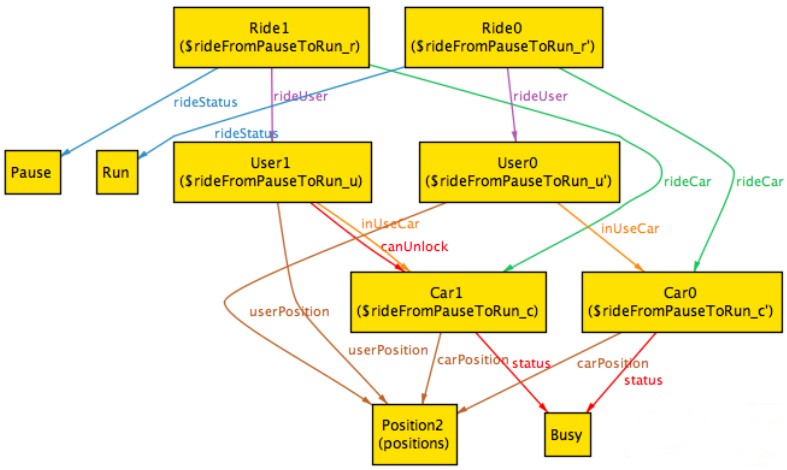
6.6.2 Reservation



Car1 is available, and User0 doesn't have any reserved car. After, User1 becomes User0, Car1 becomes Car0 with status Reserved, and User0 has Car0 as reserved car.

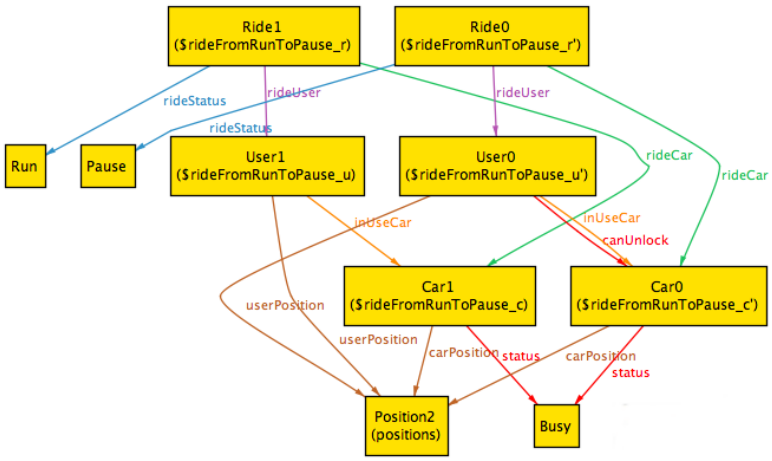
Figure 23: Reservation

6.6.3 Ride changing of status



Ride1 is paused and becomes Ride0 that is Run. A thing to notice is that User1 has as car in use Car1, and they are in the same position, so User1 can unlock the car.

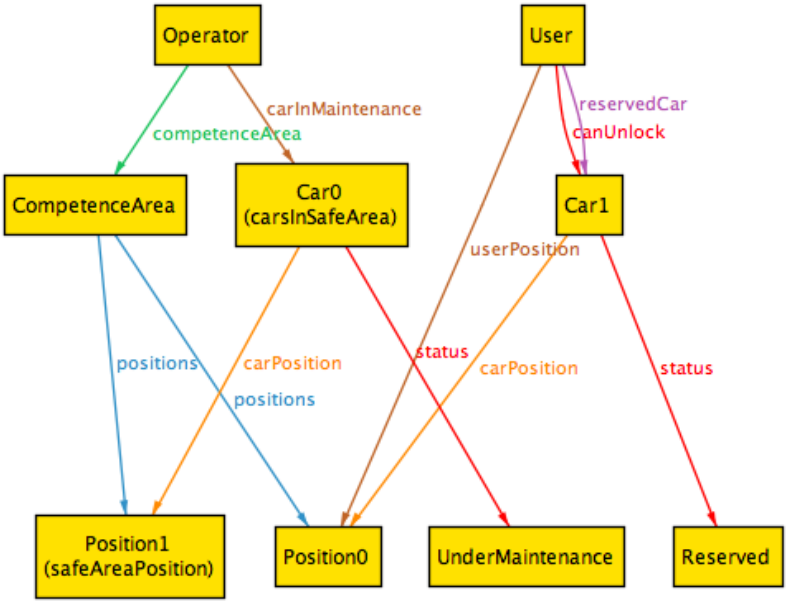
Figure 24: Ride from Pause to Run



Ride1 is in Run status and becomes Ride0, that is paused. User1 becomes User0, and now it can unlock the car.

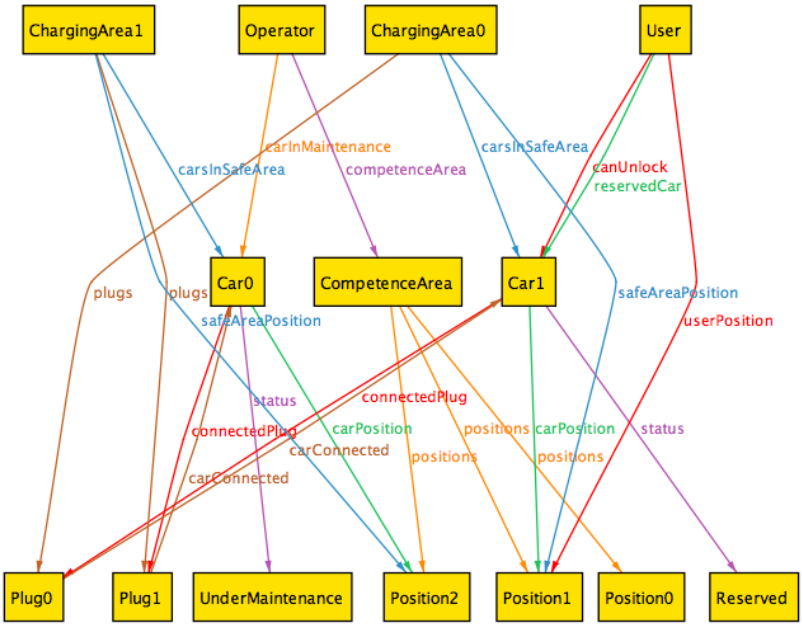
Figure 25: Ride from Run to Pause

6.6.4 General View



Operator is working on Car0, that is under mantainance and in his competence area. User has as reserved car Car1, they are in the same position, so it can unlock the car.

Figure 26: General View 1



Car1 is in CargingArea0, and it is plugged in one of the ChargingArea0 plugs.

Figure 27: General View 2

6.6.5 Model checking poof

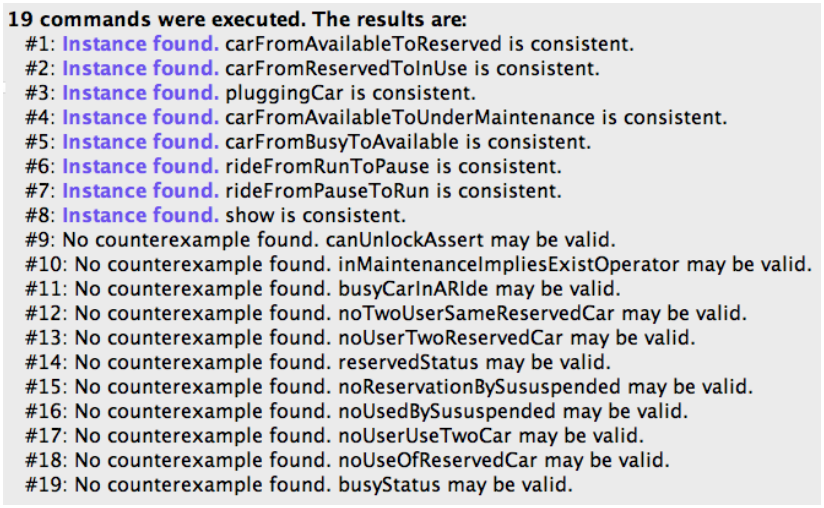


Figure 28: Solver results screenshot

7 Used Tools

The tools used to create this RASD document are:

- Github: for version control.
- Lyx: to redact and organize this document.
- StarUML: to create UML diagrams (class diagram, sequence diagram, statechart diagram, use case diagrams).
- Alloy Analyzer 4.2: to check the correctness and consistency of the model.
- Balsamiq Mockups 3: to create UI Mockups.

8 Hours of Work

Emanuele Ghelfi:

- 21/10/16: 4 h, Project Planning
- 25/10/16: 4 h, Goals and Requirements
- 28/10/16: 6 h, UML Diagrams
- 01/11/16: 2 h, UML Diagrams
- 04/11/16: 4 h, Alloy
- 08/11/16: 4 h, General correction and refactoring
- 11/11/16: 4 h, Final refactoring and correction
- 12/11/16: 4h, Final refactoring and correction
- Other hours: 5 h

Total hours: 37 h

Emiliano Gagliardi:

- 21/10/16: 4 h, Project Planning
- 25/10/16: 3 h, Goals and Requirements
- 28/10/16: 6 h, Introduction and Overall description
- 01/11/16: 2 h, UML Diagrams
- 04/11/16: 4 h, Alloy
- 08/11/16: 4 h, General correction and refactoring
- 11/11/16: 4 h, Final refactoring and correction
- 12/11/16: 4h, Final refactoring and correction
- Other hours: 5 h

Total hours: 36 h