



POLITECNICO
MILANO 1863

PowerEnjoy

Requirement Analysis and Specification Document

Luca Scannapieco - 877145
Andrea Pasquali - 808733
Emanuele Torelli - 876210

13/11/2016

Table of contents

1. Introduction	3
1.1. General description of the problem	3
1.2. Stakeholders identifying	3
1.3. Actors identifying	3
1.4. Goals	3
1.5. Text assumptions	4
1.6. Domain assumptions	5
1.7. Glossary	5
1.7. Constraints	6
1.7.1. Regulatory policies	6
1.7.2. Interfaces	6
1.7.3. Others	7
2. Requirements	7
2.1. Functional requirements	7
2.1.1. Guests	7
2.1.2. Users	7
2.1.3. Assistance coordinator	9
2.2. Non-functional requirements	11
2.2.1. Mobile interface	11
2.2.2. Web interface	12
2.2.3. Car screen interface	13
3. Functional Modeling	14
3.1. Possible scenarios	14
3.1.1. Scenario 1	14
3.1.2. Scenario 2	14
3.1.3. Scenario 3	14
3.1.4. Scenario 4	14
3.2. Use case diagram	15
3.3. Use cases description	16
3.3.1. Sign up	16
3.3.2. Login	16
3.3.3. See available cars	17
3.3.4. Make a reservation	17
3.3.5. Unlock the car and start a ride	17
3.3.6. See all the information about a ride	18

3.3.7. Activate the money saving option	18
3.3.8. Finish a ride.....	18
3.3.9. Get a discount.....	19
3.3.10. See position and battery of cars.....	19
3.3.11. Tag/untag a car as out of order	19
4. Object Modeling	20
4.1. Class diagram.....	20
4.2. Statechart diagram	20
5. Dynamic Modeling.....	21
5.1. Sequence diagrams.....	21
5.1.1. Login	21
5.1.2. Reserve a car.....	22
5.1.3. Unlock the car.....	23
5.1.4. Activate the money saving option	24
5.2. Activity diagrams	25
5.2.1. Process of unlocking a car	25
5.2.2. Process of finishing a ride	26
6. Alloy modeling.....	27
6.1. Model.....	27
6.2. Alloy Analyzer results	33
6.3. World generated	34
7. Other info	36
7.1. Future development.....	36
7.2. Reference documents.....	36
7.3. Used tools.....	36
7.4. Hours of work	36
7.5. Changelog.....	36

1. Introduction

1.1. General description of the problem

We will project PowerEnjoy, a car sharing service that allows people to reserve and drive electrical cars in Milano.

The system allows users to reserve a car via mobile app or via web app, using GPS to identify the position of the user and the position of the available cars around the city.

Registration is mandatory before using the service, this allow to collect all the needed information about people who want to drive PowerEnjoy cars.

Users can drive a car everywhere but they must park within safe areas accurately defined by the company.

The system provides user some eventual discounts, for example if a user shares the car with at least two other people or if he leaves the car with the battery charged or charging in a power station at the end of a ride.

The society has also hired some operators to deal with bad behaviors of the users, like cars left around the city with drain batteries.

1.2. Stakeholders identifying

Our stakeholder is ElectricEngine Inc., a company that has been producing electric cars since 1999 and has decided to invest in car-sharing service in our city.

The company wants to provide a service completely eco-friendly using its model of electric car called “Volta”; its CEO is our Prof Luca Mottola.

However, we can adapt this system to accomplish other requests from other enterprises with the same type of cars.

1.3. Actors identifying

- Guests: people who haven't registered to the service yet, they only can read a description of the service or sign up.
- Users: people who have already signed up so that the system has given them the password that can be used to access the system.
- Assistance coordinator: is a company employee in charge of manage the operators when a car needs assistance (e.g. battery replacement).

1.4. Goals

We want to build a system which is capable to reach the following goals:

1. Allow guests to sign up.
2. Allow users to sign in.
3. Allow users to see the available cars (and their battery level) near them or near to a given address.
4. Allow users to reserve an available car for up to one hour and to know if their reservation went successfully and eventually fine them if the hour expires.
5. Allow users to unlock and have access to a car if and only if they are close to that car and the car is reserved by them.

6. Allow users to end a ride if and only if the car is in a safe area or the car has run totally out of battery or an accident happens.
7. Allow users to receive a 10% discount from the total fee if they carry more than two people.
8. Apply a fine of 30% of the total cost to users if the car has been parked more than 3 km from the nearest power station or with less than 20% of battery.
9. Reward users with a 20% of discount if they leave the car with more than 50% of the battery.
10. Reward users with a 30% of discount if they leave the car charging into a power station.
11. Allow users to use the money saving option (see glossary)
12. Allow users to know in real time all the information (cost, car's battery level, safe areas' location) about their ride.
13. Allow assistance coordinator to sign in.
14. Allow assistance coordinator to see the GPS position of all the available cars and their battery level in order to identify the cars in need of battery replacement.
15. Allow the assistance coordinator tag a car/untag as out of order following an accident or damage report by a user.

1.5. Text assumptions

In this paragraph we are going to describe the assumptions we made according to the given specifications in order to build the clearest and most complete model possible.

- We thought that is realistic to assume that a company which wants to invest in a car sharing service has to build the system starting from scratch.
- Safe areas are intended as a set of zones around the city instead of a boundary that surrounds the whole Milano.
- We assumed that safe areas are the only parking zones allowed by the company. So, unless involvements in accidents or battery drains, users can't finish a ride if they haven't park their PowerEnjoy car in a safe area.
- In case of users who break the road rules, they will be fined according to the law, so we don't have to deal with this issue.
- Power stations are composed by a single plug where to attach the car, a set of power station is called power station.
- We assumed that the company has hired some operators which are meant to retrieve cars in need of assistance, for example cars out of battery (<20% of battery level) or cars involved in accidents.
- In order to speed up the process of retrieving cars with drain battery, we assumed that it's better to send operators to replace batteries on site without taking the car to a power station.
- Operators are managed by an assistant coordinator, in addition only the assistance coordinator has access to the system.
- In our study we overlooked the actual payment phase, because often those kind of transactions are managed by banks using complex systems. We only deal with the process of calculating and showing to users the final fee of a ride, assuming that the real payment transaction will be applied automatically by the bank at the end of the ride.
- According to the assumption just mentioned, users can only use credit cards provided previously in the registration for payment.
- The company already knows how to deal with users in trouble with payments (e.g. users with not enough money on their credit card), so we do not have to deal with this issue.
- We won't allow users cancel neither cancel their reservation nor reserve multiple car, in order to prevent bad behaviors that can reduce the service efficiency.

- We assumed that discounts that users can get during a ride are not cumulative, in order to prevent overdone reduction of the fee, so only the biggest one will be applied. In addition to that, if a user gets fined, the system won't apply any discount.
- We thought it's reasonable to think that the discount related to the money saving option is highly related to the one achieved by leaving the car in a power station, so we decided to fix it at 30% of total cost.
- We thought that in our study we have to deal with accidents or car damages, but in order to maintain things as light as possible, car issues will be reported manually (e.g. by phone call) by users to the assistance coordinator, and then the coordinator will deal with those issues.
- If a serious accident happens or the battery level reaches 0% so that the car is not able to go on anymore, the current ride has to end even if the car isn't in a safe area (e.g. cars have sensors that detect accidents occurring during a ride).
- We thought that is reasonable to let the user 3 minutes starting from the end of the ride to allow him to get out of the car and eventually plug it into a power station.

1.6. Domain assumptions

- Cars have a unique ID number.
- The maximum of other passengers excluding the driver during a ride is 4.
- If a sensor detects the presence of a person in the car, it means that the person is actually inside the car (sensors can't be cheated for examples using heavy objects).
- Only the owner of a reservation will drive the car he has reserved.
- Once a user unlocks his reserved car, he will actually get in and start a ride.
- All the GPSs always give the right position of the cars and must be always working.
- The company already handles the information about the operators, so we don't have to deal with them.
- All the power stations are in a safe area.
- At the end of a ride, users definitely get out of a car in 3 minutes.
- Whenever a car is left with the battery level less than 20% an operator will go to replace that battery with a fully charged one within 2 hours.
- Only operators can replace the battery of a car.
- Users never reserve a car when its battery level is at 0%.
- Only cars belonging to the company can be parked at a power station.
- If car's sensors detect that a car crash occurred during a ride, it actually took place.

1.7. Glossary

- Guest: person who hasn't registered yet to the service. He can only sign up to take benefits from the service
- Sign up: process that allows a guest to provide his password so that he can access to the system. In the registration process he must compile a form giving these information:
 - Name
 - Surname
 - Phone number
 - Email
 - Address
 - SSN

- Credit card number
- Driving licence's number

the system will reply sending him an email containing a password.

- User: person who has already registered and can access to the system using his username and password to reserve or unlock an available car.
- Reservation: a process by which a user can reserve an available car up to one hour: since he reserves it, he has exactly one hour to reach it and unlock until the system deletes his reservation and fines him of 1 Euro.
- Ride: by calling ride we mean the time interval from the moment the user starts PowerEnjoy car to the moment he presses the button to terminate the route (after he has turned off the car as well).
- End of ride: state of the car that starts from the moment the user presses the related button after parking in a safe area and goes until the user presses the related button on the mobile app (or automatically after a maximum of 3 minutes). The screen of the car says goodbye to the user and turns off, and he can see a recap screen in his mobile app in which is showed the fee and the discount achieved (or eventually the fine).
- Available car: it's a PowerEnjoy car not reserved by another user and no other user is driving it.
- Out of order car: an out of order car can't be reserved because of damages, in fact it is not available until it will be fixed and the assistance coordinator tags it as available again
- Safe area: a car is parked in a safe area if it is in one of the parks belonging to the set pre-defined by the management system.
- Sharing discount: it's a 10% discount from the total price given to users who carry at least other two people.
- Battery discount: it's a 20% discount from the total price given to users who leave the car parked with more than 80% of battery level.
- Power station discount: it's a 30% discount from the total price given to users who leave the car parked charging in a power station.
- Low battery fine: it's a 30% fine from the total price given to users who leave the car parked with less than 20% of battery level.
- Money saving option: if the user enables the money saving option, he can input his final destination and the system provides information about the station where to leave the car to get a discount.
- Money saving option discount: it's a 50% discount from the total price given to users who follow the money saving option instructions.
- Money saving station: it's the station provided by the money saving option.
- Power station: it's a place in which a car can be recharged.
- Non-fine area: for convenience, we will call non-fine area the zones less distant than 3 km from a power station, since a user pays 30% more if he parks out of them.

1.7. Constraints

1.7.1. Regulatory policies

The system must ask the user the permission to get his position and the permission to manage sensible data (position, mail) according to the privacy law. Furthermore, the systems must not use notifications to send SPAM respecting the privacy law.

1.7.2. Interfaces

- Mobile application
 - 3G/4G/Wi-Fi connection

- GPS
 - Enough space for app package
- Web Browser
 - Modern browser with AJAX
 - Access to GPS of device
- Screen inside car
 - GPS navigator
 - Car battery level indicator
 - Power stations position
 - Real time cost indicator
 - Safe areas position

1.7.3. Others

- The server supports parallel operations from different users.
- The system exploits a database to store all the data needed by the company.

2. Requirements

2.1. Functional requirements

Assuming that the domain properties stipulated in the paragraph 1.6. hold, and, in order to fulfill the goals listed in the paragraph 1.4., the following requirements can be derived. The requirements are grouped under each goal from which it is derived.

2.1.1. Guests

1. Allow guests to sign up:

- The system must be able to save the information of all the people who sign up.
- the system must be able to check the correctness of the registration info provided by the user.
- The system replies to every correct registration with a password that the user must use to access.
- The system replies to every incorrect registration by notifying the error.
- The system must prevent users to sign up more than once.

2.1.2. Users

2. Allow users to sign in:

- The system must be able to check if the credentials are correct.
- The system must allow the user to sign in if and only if the provided credentials are correct

3. Allow users to see the available cars (and their battery level) near them or near to a given address:

- The system must have access to the GPS position of all the available cars.
- The system must have access to the GPS position of the user.
- The system must be able to detect all the available cars within a certain distance from the user and show them on a map.
- The system must be able to detect all the available cars within a certain distance from a position selected by the user and show them on a map.

- The system must show the battery level of each available car displayed to the user.
4. Allow users to reserve an available car for up to one hour and to know if their reservation went successfully and eventually fine them if the hour expires
 - The system must tag the car as not available as soon as the reservation is performed.
 - The system must notify the user when a reservation goes successfully.
 - The system activates the reservation timer for the reserved car as soon as the reservation is performed.
 - The system must tag the car as available as soon as the reservation timer is expired.
 - The system must notify the user when the reservation timer expires.
 - The system must send a 1 Euro fine to users who haven't taken the cars they reserved within one hour.
 - The system must show the position of the reserved car only to the user who made the reservation as long as the reservation timer is running.
 5. Allow users to unlock and have access to a car if and only if they are close to that car and the car is reserved by them.
 - The system must detect if the user is less than 5 meters distant from the reserved car.
 - The system must be able to unlock the car once the user is less than 5 meters away from it.
 6. Allow users to end a ride if and only if the car is in a safe area or the car has run totally out of battery or an accident happens:
 - The system must have access to the GPS position of the car.
 - The system must authorize the user to end the ride if and only if the car is turned off and its GPS position is in a safe area or an incident occurs or the car runs totally out of battery.
 - The system must alert the user if he attempts to end a ride but he is not in a safe area or the car is not turned off.
 - The system must stop charging the user once that he ends the ride.
 - The system must display the final total cost of the ride when the ride is ended.
 - The system must tag the car as available after 3 minutes that the ride is ended.
 7. Allow users to receive a 10% discount from the total fee if they carry more than two people:
 - The system must detect the number of passengers that stay in the car for at least half of the duration of the ride.
 - The system must apply a 10% discount on the total fee if that number is at least two and if the user hasn't got neither a greater discount nor a fine.
 8. Apply a fine of 30% of the total cost to users if the car has been parked more than 3 Km from the nearest power station or with less than 20% of battery.
 - The system must detect the battery level of a car.
 - The system must have access to the GPS position of the car.
 - When the ride ends, before calculating the total fee, the system must detect whether the car is plugged into a power station or not.

- The system must apply a fine of 30% of total cost to a user if it detects that the user has ended the ride leaving the car with less than 20% of battery and not plugged into a power station.
 - The system must apply a fine of 30% of total cost to a user if it detects that the user has ended the ride leaving the car in safe area that is more than 3 Km away from a power station.
9. Reward users with a 20% of discount if they leave the car with more than 50% of the battery
- The system must detect the battery level of a car.
 - When the ride ends, before calculating the total fee, the system must detect whether the car is plugged into a power station or not.
 - The system must apply a 20% of discount on the total fee if it detects that the user has ended the ride leaving the car with more than 50% of battery and he hasn't got neither a greater discount nor a fine.
10. Reward users with a 30% of discount if they leave the car charging into a power station.
- When the ride ends, before calculating the total fee, the system must detect whether the car is plugged into a power station or not.
 - The system must apply a 30% of discount on the total fee if it detects that the user has ended the ride leaving the car plugged into a power station and he hasn't got neither a greater discount nor a fine.
11. Allow users to use the money saving option (see glossary).
- The system must ask to the user for money saving option activation once the user unlocks the car.
 - Once the user activates the money saving option, the system must ask to the user his destination.
 - The system must be able to calculate the availability of power plugs in all the power stations.
 - The system must be able to calculate the distribution of cars in the city.
 - The system must be able to determine the money saving station (see glossary).
 - The system must show to the user the selected money saving station on a map.
 - When the ride ends, before calculating the total fee, the system must detect whether the car is plugged into the selected power money station or not.
 - The system must apply a 50% of discount on the total fee if it detects that the user has ended the ride leaving the car plugged into the selected power money station.
12. Allow users to know in real time all the information (cost, car's battery level, safe areas and power station's location) about their ride.
- The system must be able to calculate how much money the user is spending during a ride.
 - The system must be able to display in real time the current fee of the ride.
 - The system must be able to detect the battery level of a car during the ride.
 - The system must be able to display in real time the current battery level of the car during the ride.
 - The system must be able to display the location of all the predefined safe areas during the ride.

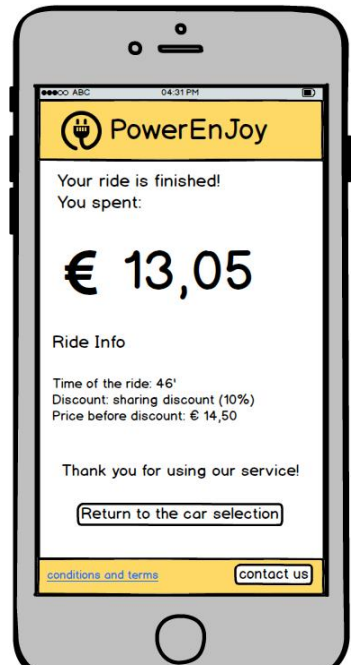
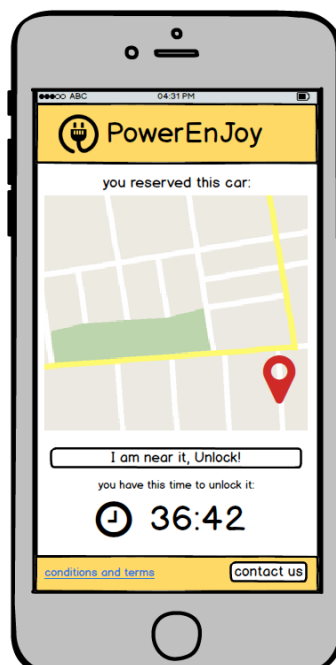
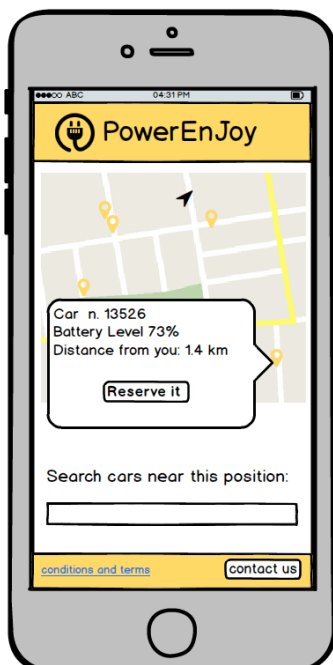
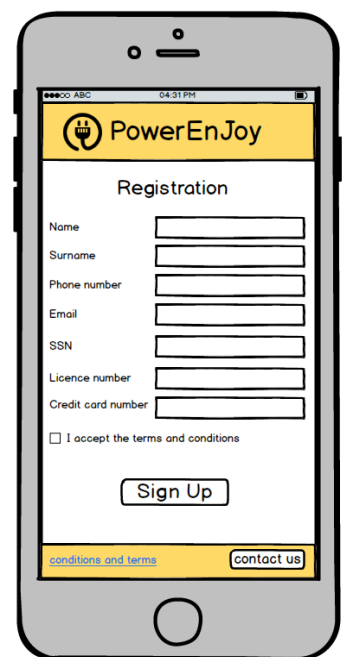
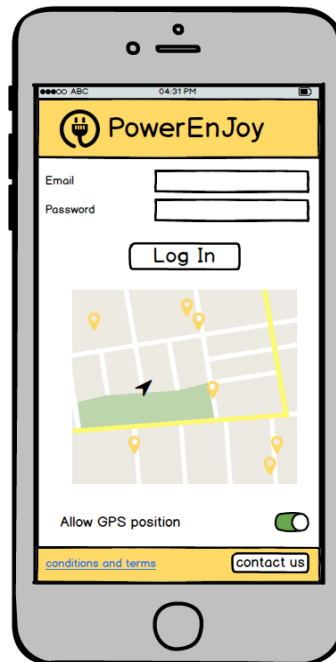
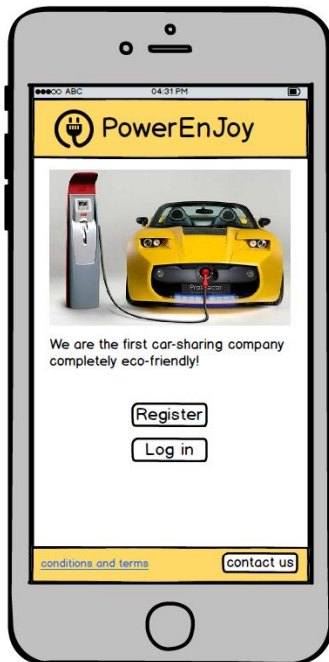
2.1.3. Assistance coordinator

13. Allow assistance coordinator to sign in.
- The system must be able to check if the assistance coordinator credentials are correct.

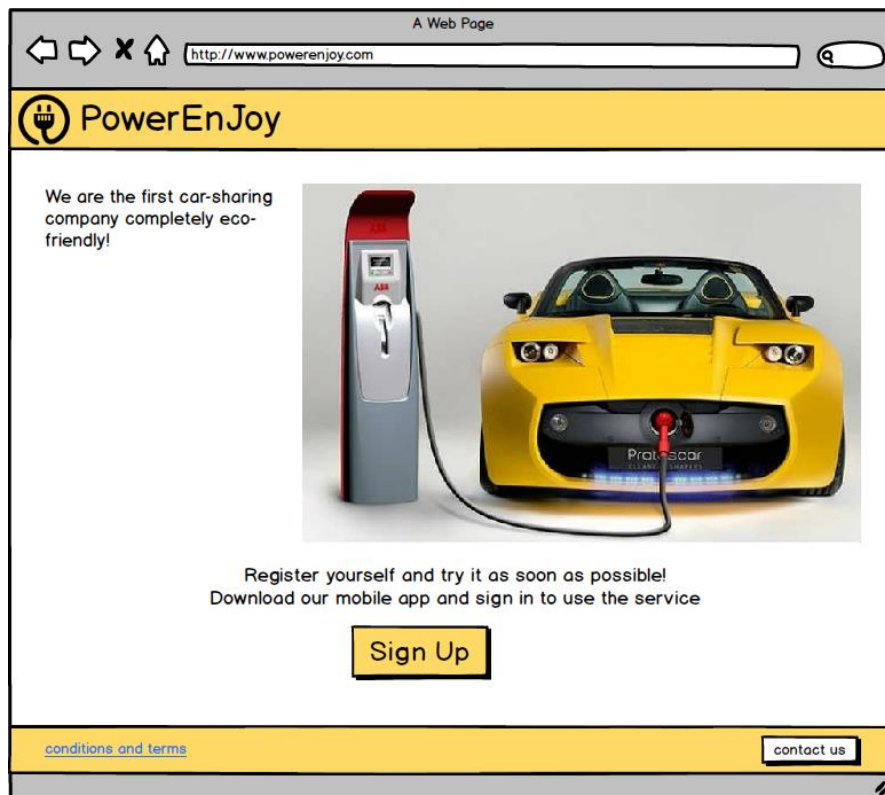
- The system must allow the assistance coordinator to sign in if and only if the provided credentials are correct.
14. Allow the assistance coordinator to see the GPS position of all the available cars and their battery level in order to identify the cars in need of battery replacement.
- The system must have access to the GPS position of all the available cars.
 - The system must detect the battery level of all the available cars.
 - The system must show to the assistance coordinator all the available cars on a map, highlighting the ones with less than 20% of battery level.
 - The system must be able to notify the assistance coordinator that a battery replacement went successfully.
15. Allow the assistance coordinator tag a car/untag as out of order following an accident or damage report by a user.
- The system must show to the assistance coordinator all the available cars on a map.
 - The system must allow the assistance coordinator to select a car.
 - The system must allow the assistance coordinator to tag the car as out of order.
 - The system must allow the assistance coordinator to tag cars as available once fixed.

2.2. Non-functional requirements

2.2.1. Mobile interface




2.2.2. Web interface



A screenshot of the PowerEnJoy registration page. The browser's address bar shows the URL <http://www.powerenjoy.com>. The page features a yellow header with the PowerEnJoy logo and name. Below the header, the title "Registration" is displayed. The registration form consists of several input fields for the following information: Name, Surname, Phone number, Email, SSN, Licence number, and Credit card number. Below these fields is a checkbox labeled "I accept the terms and conditions of the service". A "Sign up" button is located at the bottom of the form. At the bottom of the page, there is a yellow footer containing a link to "conditions and terms" and a "contact us" button.

2.2.3. Car screen interface

 **PowerEnJoy** User Tom Brown Car number 345216

Save Money Option

Activated

Disactivated

Target position:

Start the navigation


Continue without GPS navigator

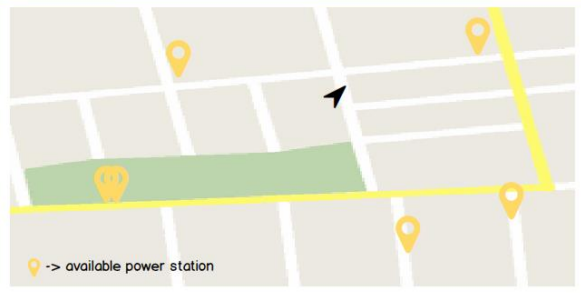
QWERTYUIOP

ASDFGHJKL

↑ZXCVBNM!/?↑

123 456 789 0

 **PowerEnJoy** User Tom Brown Car number 345216




-> available power station

-Turn left in 350 mt, the selected power station is on the right.

current cost of the ride: € 7,35

Assistance number: 199 199 199

End your ride

 **PowerEnJoy** User Tom Brown Car number 345216

You terminated your ride!
you have 3 minutes of time to plug the car in charge to a power station if you want to receive a 30% discount.

The current cost of your ride is: € 13,45

Check the final cost on your mobile app!

thanks for driving our car!

3. Functional Modeling

3.1. Possible scenarios

Here are described some possible scenarios of PowerEnJoy system usage.

3.1.1. Scenario 1

Mario has to go to work, but someone has parked in front of his garage, so he is unable to use his personal car. Fortunately, Mario is registered to PowerEnJoy, so he picks his smartphone and opens the PowerEnJoy app, then he inserts his credentials to log in the system. After that, he takes a look at the map to see if there is any available car near him. He notices that there is an available car parked two minutes walking from him, so he immediately reserves it to prevent other users to take it away before him. Once he is close to the car, he opens the app again and presses a button in order to unlock it. Since the system recognizes that Mario actually is the user who has made the reservation and he less than 5 m distant from the car, the car unlocks the door so that Mario can get into it, ignite the engine with the keys provided inside the dashboard, and go to work on time.

3.1.2. Scenario 2

Mario is driving a PowerEnJoy car. Once he arrives to his home, he looks at the monitor to see if he is in a safe area, but he figures out that his car has the battery very low (10%). Since the football match on the TV is starting, he doesn't want to look for a power station to recharge the car, so he leaves it as it is, out of battery. Once Mario gets out of the car, the system detects that Mario's ride is over, but since the car has been left with less than 20% of battery charged, in addition to the cost of the ride the system will withdraw an additional amount of money as a fine from Mario's credit card.

3.1.3. Scenario 3

Mario has an appointment to the cinema with his friends Ugo and Anna, but today there is a transport strike and the cinema is quite far from their houses. So, Mario, who is a PowerEnJoy user, decides to go taking his friends up to share the route. Car's sensors detect that in the car there are more than two passengers in addition to the driver, so at the end of the ride the system will apply a discount to the total fee. Once Mario and friends arrive to the cinema, they decide to leave the car in the nearest power station, in order to get a bigger discount. Once the car is attached to the power charger and everyone is out, the system detects that the ride is over and calculates the total amount of money that will withdraw from Mario's credit card, considering also the biggest of the discounts from the ones which he achieved. Since Mario got both the 10% discount for carrying at least 2 people and the 20% discount as he left the car in charging in a power station, the system will eventually apply a 20% discount from the total fee of the ride.

3.1.4. Scenario 4

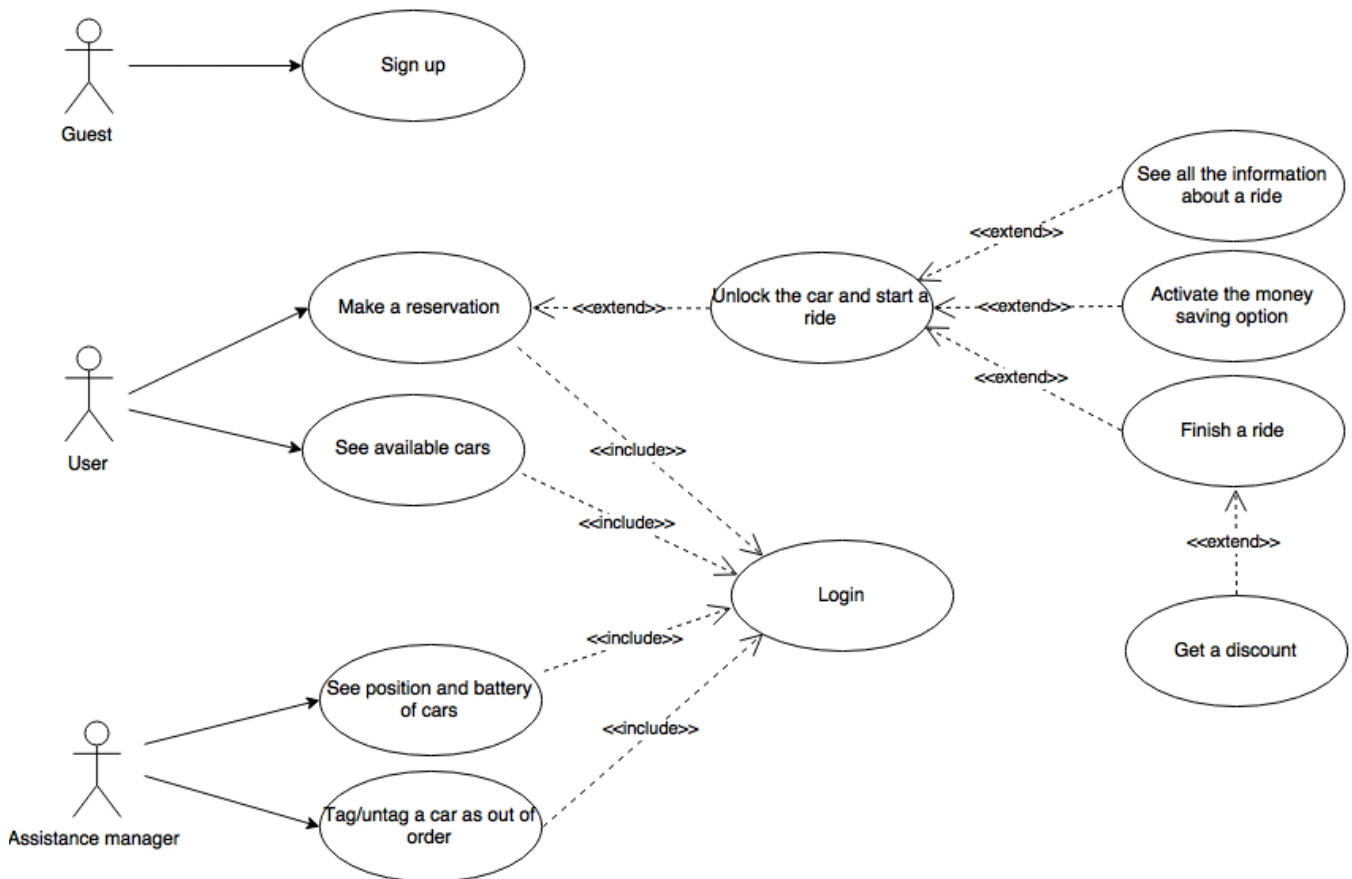
Mario is an operator at the Electric Engine Inc. and one of his main tasks is to monitor the cars in order to prevent the company to have cars out of battery left around the city. The procedure is simple: all that Mario has to do is login to the system via web or mobile app, and select the option which allows him to see all the available cars on a map. In order to facilitate Mario's search, the system highlights the cars whose battery level is under 20%. In fact, cars with less than 20% of battery are supposed to be almost unusable, so Mario contacts an operator and tells him to reach the car to do a battery replacement. The operator is therefore meant to go immediately at the location of the car and replace its battery with a fully charged one.

3.2. Use case diagram

In this section we derive the use case diagram according to the goals listed in paragraph 1.4. and the scenarios mentioned above.

In our model most of the use cases are linked with an «extend» relationship because it stands for a use case that can be executed starting from certain conditions of the use case at the end of the arrow. For example in our case once a user has made a reservation he can unlock the car and start a ride, or once the user started a ride he can finish it.

On the other hand some use cases are linked with the login one with an «include» relationship because it stands for a use case that is always executed during the execution of the one at the base of the arrow. For example in our case before seeing the available cars or make a reservation a user must necessarily login.



3.3. Use cases description

In this section we are going to describe in a detailed way the use cases that we derived from the scenarios. It is important to understand that all the references to “pages”, “buttons” or “input forms” are only hypothesis to make the situation as clear as possible and to help the reader to draw a visual picture in his mind of what we plan to do, but the real structures will be well defined in the Design Document.

3.3.1. Sign up

Name	Sign up
Actors	Guests
Entry conditions	The guest isn't registered to the service
Flow of events	<ul style="list-style-type: none">• The guest accesses to the service via web app or mobile app• The guest clicks on the “sign up” button• The guest fills in the form where he has to write:<ul style="list-style-type: none">○ Name○ Surname○ Phone number○ Email○ Address○ SSN○ Credit card number○ Driving licence's number• The guest clicks the “done” button• System reply to the user with his new password
Exit conditions	Registration successfully done, the guest becomes a registered user
Exceptions	An exception can be caused if the email address of the guest already exists or if some fields of the form aren't filled properly, in this case the user is asked by the system to try again.

3.3.2. Login

Name	Login
Actors	Users, assistance coordinator
Entry conditions	The user is already registered to the service, and of course the assistance coordinator has already got his special credentials to access the system
Flow of events	<ul style="list-style-type: none">• User/assistance coordinator accesses the system via web app or via mobile app• User/assistance coordinator fills in the text fields in the home page with email and password• User/assistance coordinator clicks on the “login” button.
Exit conditions	Login successfully done
Exceptions	An exception can be caused if the email or the password aren't correct, then an error message is displayed and the credentials are asked again

3.3.3. See available cars

Name	See available cars
Actors	Users
Entry conditions	The user is already logged into the service
Flow of events	<ul style="list-style-type: none">• The user opens the map to see all the available cars near him or he selects a destination to see all the available cars near that address
Exit conditions	The user can see all the available cars displayed in a map and their battery level
Exceptions	No exceptions

3.3.4. Make a reservation

Name	Make a reservation
Actors	Users
Entry conditions	User is already logged into the system
Flow of events	<ul style="list-style-type: none">• User selects the car which he wants to reserve• User presses on the “reserve” button• A message is displayed to the user to notify that the reservation went successfully
Exit conditions	The user can see the information and the position of the car he has reserved
Exceptions	No exceptions

3.3.5. Unlock the car and start a ride

Name	Unlock the car and start a ride
Actors	Users
Entry conditions	The user is less than 5 m away from the car he has reserved
Flow of events	<ul style="list-style-type: none">• User selects in the application the option of unlocking the car• The system unlocks the car• The screen inside the car welcomes the user• The user starts the car
Exit conditions	The user is performing a ride and he can see all the information about the ride on the screen
Exceptions	User is more than 5 m away from the car, in this case the system shows an error message and doesn't unlock the car

3.3.6. See all the information about a ride

Name	See all the information about a ride
Actors	Users
Entry conditions	The user has unlocked the car and started a ride
Flow of events	<ul style="list-style-type: none"> The user is performing his ride
Exit conditions	The screen inside the car displays the details of the ride to the user (cost, battery, safe areas, power stations)
Exceptions	nothing

3.3.7. Activate the money saving option

Name	Activate the money saving option
Actors	Users
Entry conditions	The user has unlocked the car
Flow of events	<ul style="list-style-type: none"> The user selects in the car's screen the money saving option by pressing the related button The user inserts the address of his destination The user presses a button in order to confirm his choice
Exit conditions	The screen is highlighting the path leading to the "money saving station" according to the destination inserted by the user
Exceptions	nothing

3.3.8. Finish a ride

Name	Finish a ride
Actors	Users
Entry conditions	The user has stopped the machine in a safe area during a ride
Flow of events	<ul style="list-style-type: none"> The user turns off the engine of the car The screen of the car displays a recap of the ride performed and says goodbye to the user telling him that he can see the discount applied after 3 minutes on his mobile The car waits for 3 minutes in this state to let the user get out and eventually plug into a power station After 3 min the user can see the discount applied to the cost of his ride User is charged of the total amount of the fee including the eventual discount or fine
Exit conditions	The car is tagged as available again from the system
Exceptions	The user has not parked in a safe area, then the car will make an acoustic signal and display a message on the screen and it will not be possible to finish the ride.

3.3.9. Get a discount

Name	Get a discount
Actors	Users
Entry conditions	The user has finished his ride 3 minutes ago
Flow of events	<ul style="list-style-type: none">• The system checks if the user has to pay 30% more for parking 3 km away from a power station• If the user has parked less than 3 km away from the nearest power station, the system calculates the biggest discount achieved by him
Exit conditions	The user can see the total fee of the ride considering the discount
Exceptions	nothing

3.3.10. See position and battery of cars

Name	See position and battery of cars
Actors	Assistance coordinator
Entry conditions	Assistance coordinator is already logged into the system
Flow of events	<ul style="list-style-type: none">• The assistance coordinator presses a button from the web app or the mobile app
Exit conditions	The assistance coordinator can see all the available cars displayed on a map and their battery level, cars with less than 20% of battery are highlighted
Exceptions	nothing

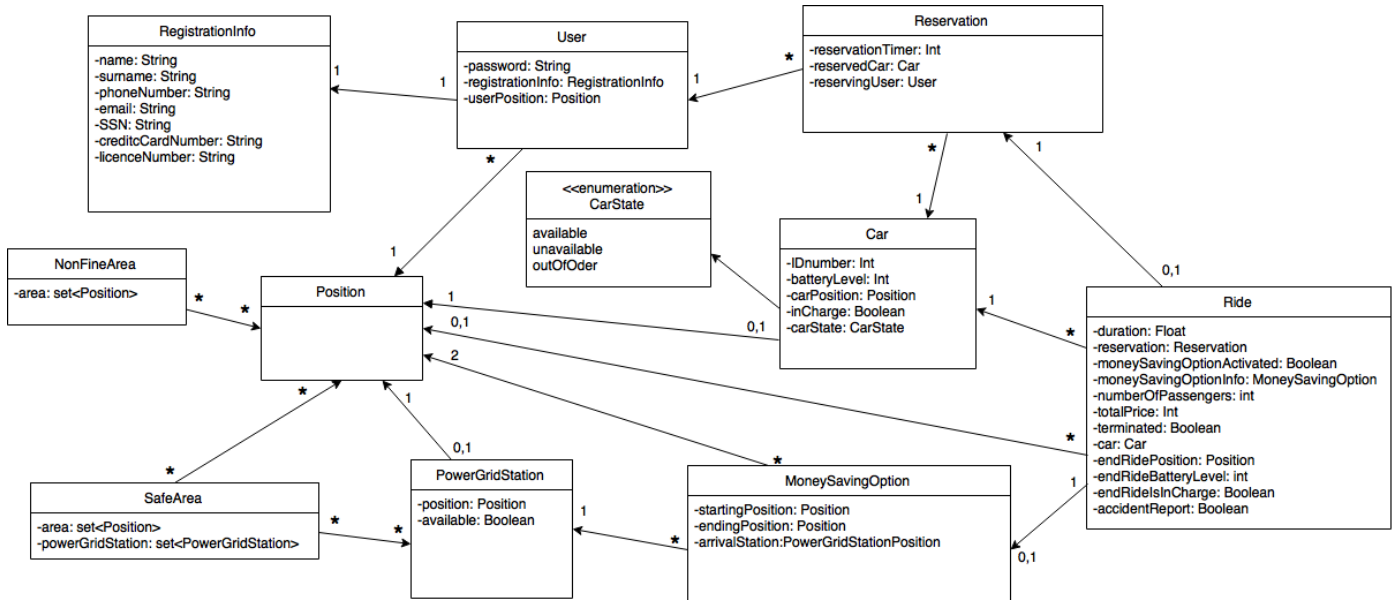
3.3.11. Tag/untag a car as out of order

Name	Tag/untag a car as out of order
Actors	Assistance coordinator
Entry conditions	Assistance coordinator is already logged into the system
Flow of events	<ul style="list-style-type: none">• The assistance coordinator presses a button from the web app or the mobile app• The assistance coordinator selects the car indicated as damaged by clicking on it or by searching it with the id provided• The assistance coordinator tags/untags the selected car as out of order by clicking on the related button• The assistance coordinator confirms his choice
Exit conditions	The car is available again
Exceptions	nothing

4. Object Modeling

4.1. Class diagram

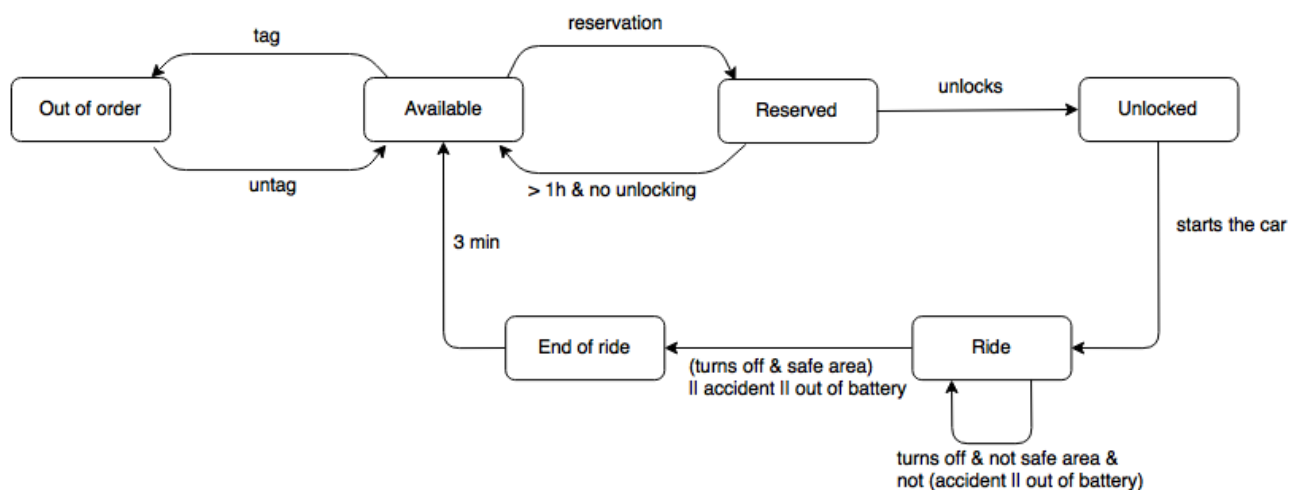
This class diagram reflects the structure of the Alloy model described in section 6.



4.2. Statechart diagram

Here is a representation of all the states in which PowerEnjoy cars are supposed to be and the conditions of moving from one state to another.

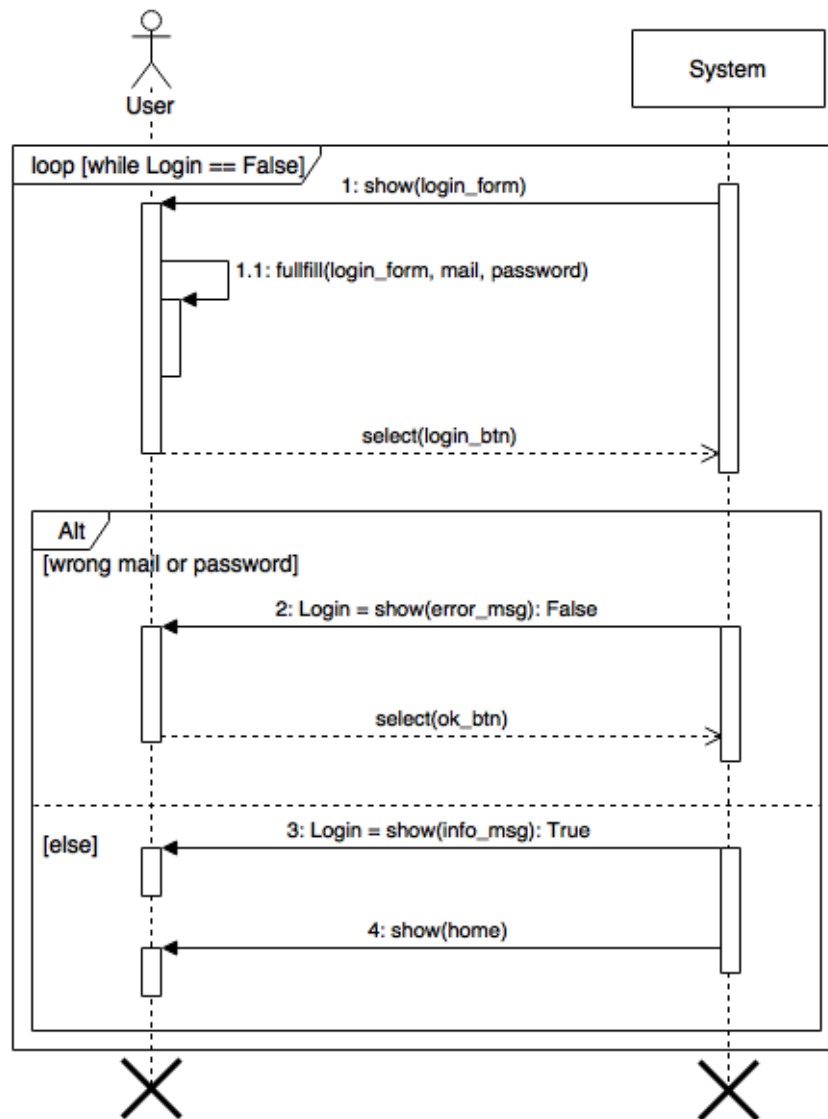
Note that each condition is performed by a user, except the ones entering and exiting the out of order state, which are performed by the assistance coordinator.



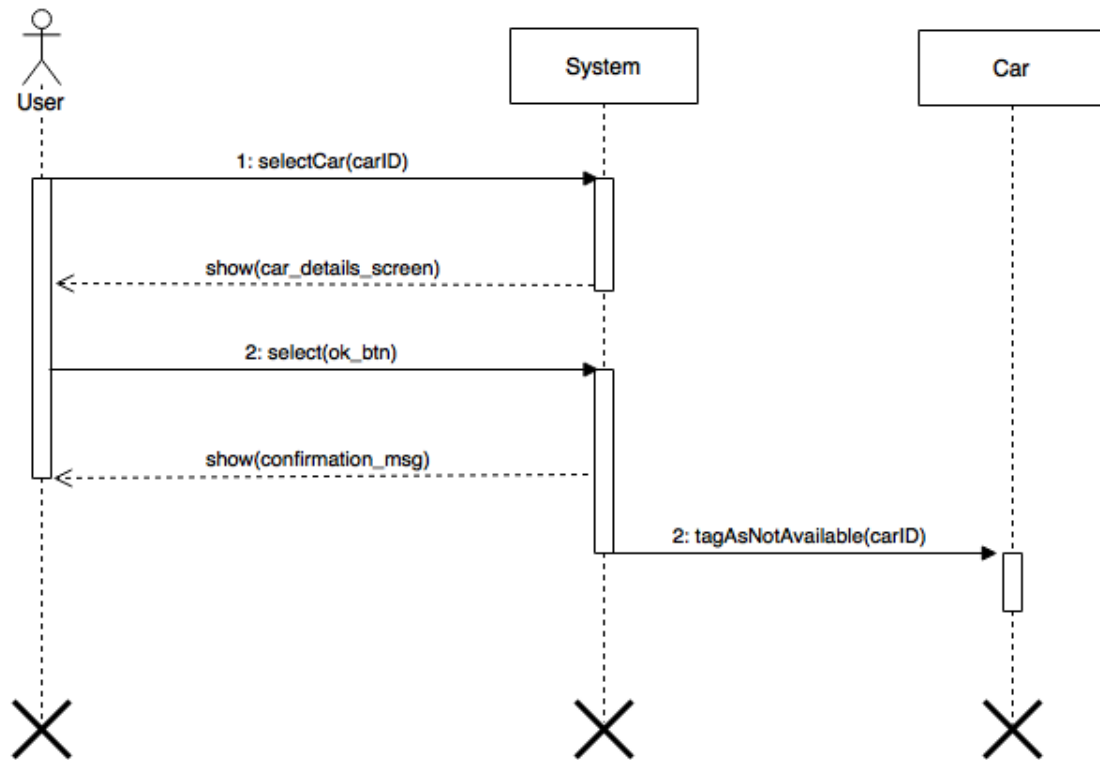
5. Dynamic Modeling

5.1. Sequence diagrams

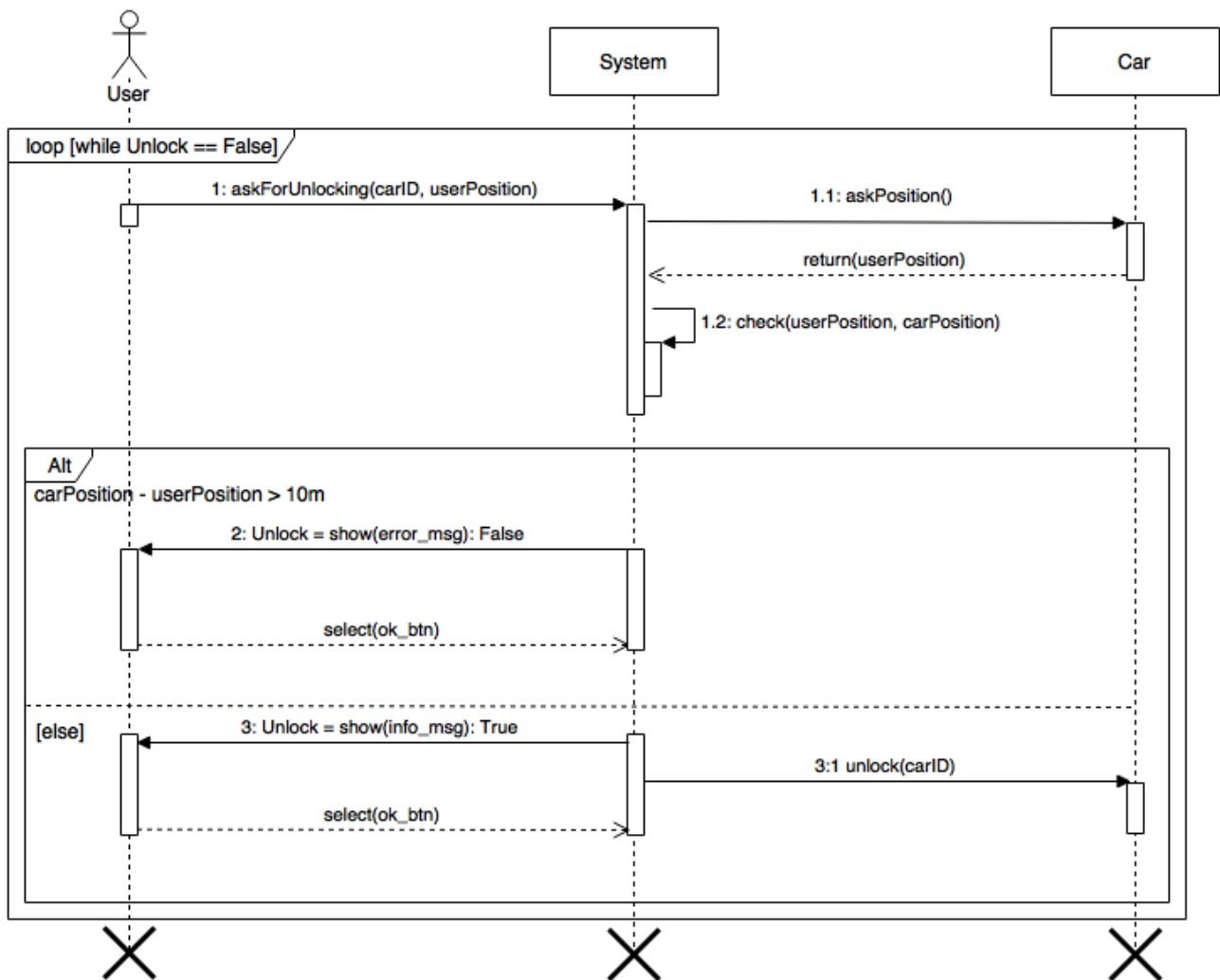
5.1.1. Login



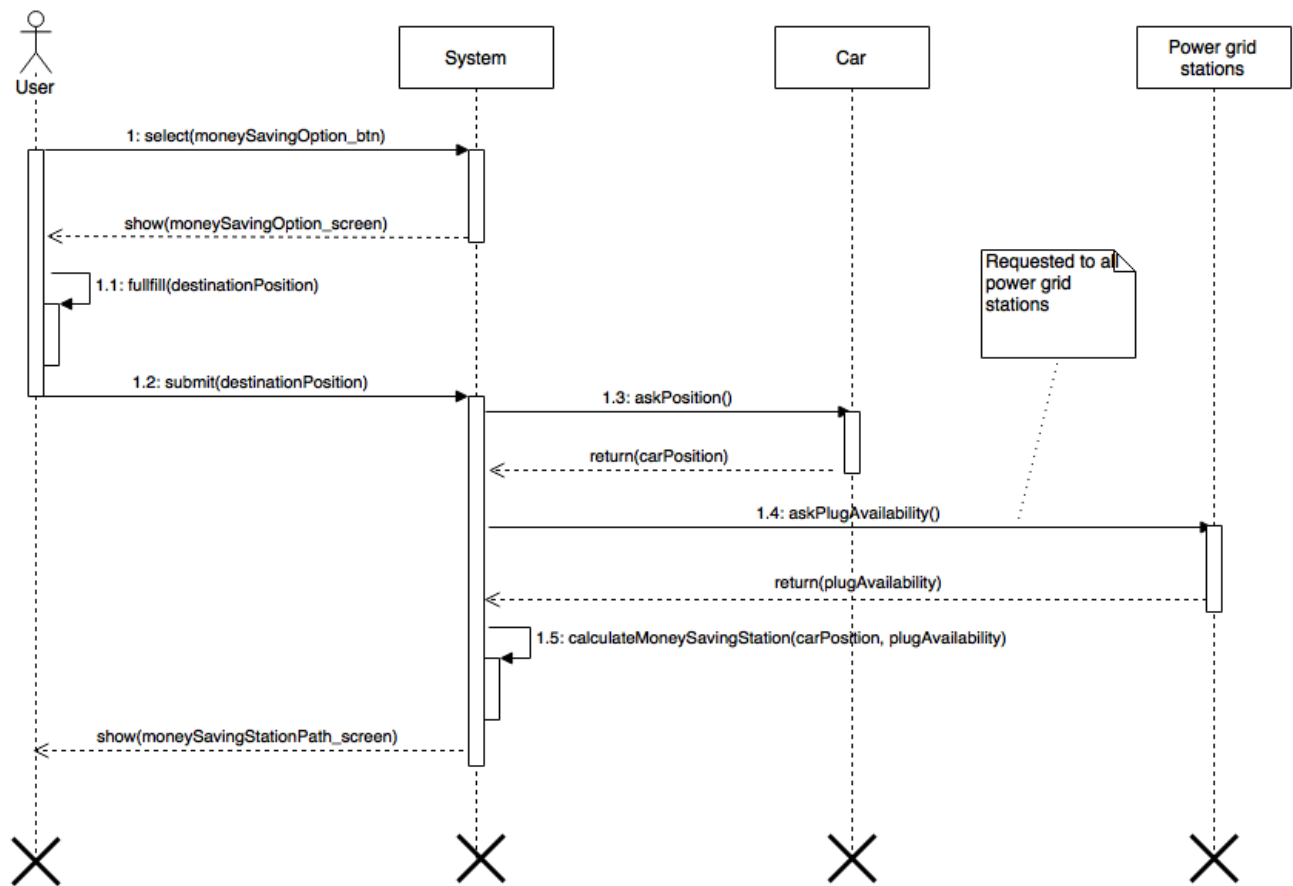
5.1.2. Reserve a car



5.1.3. Unlock the car

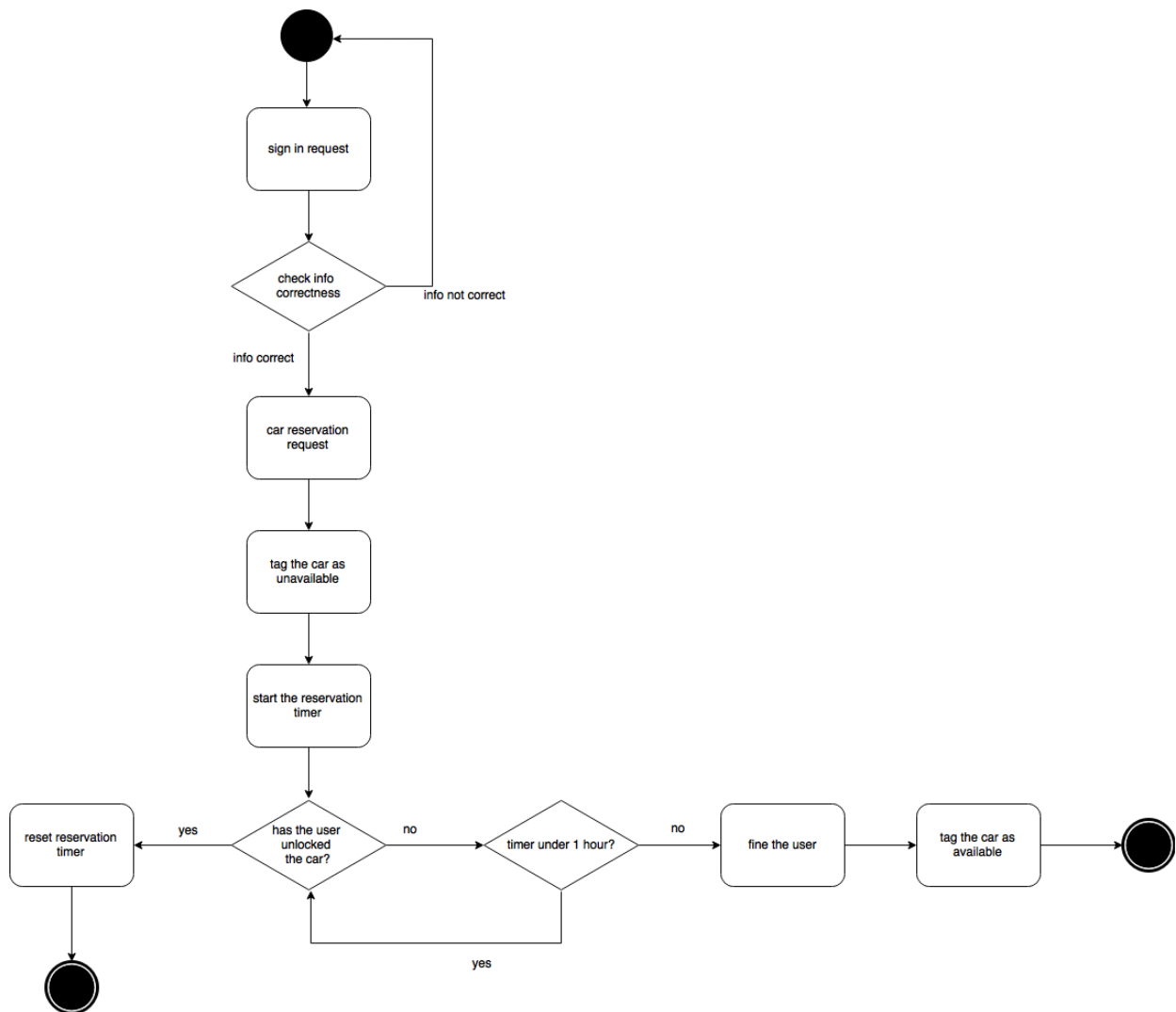


5.1.4. Activate the money saving option

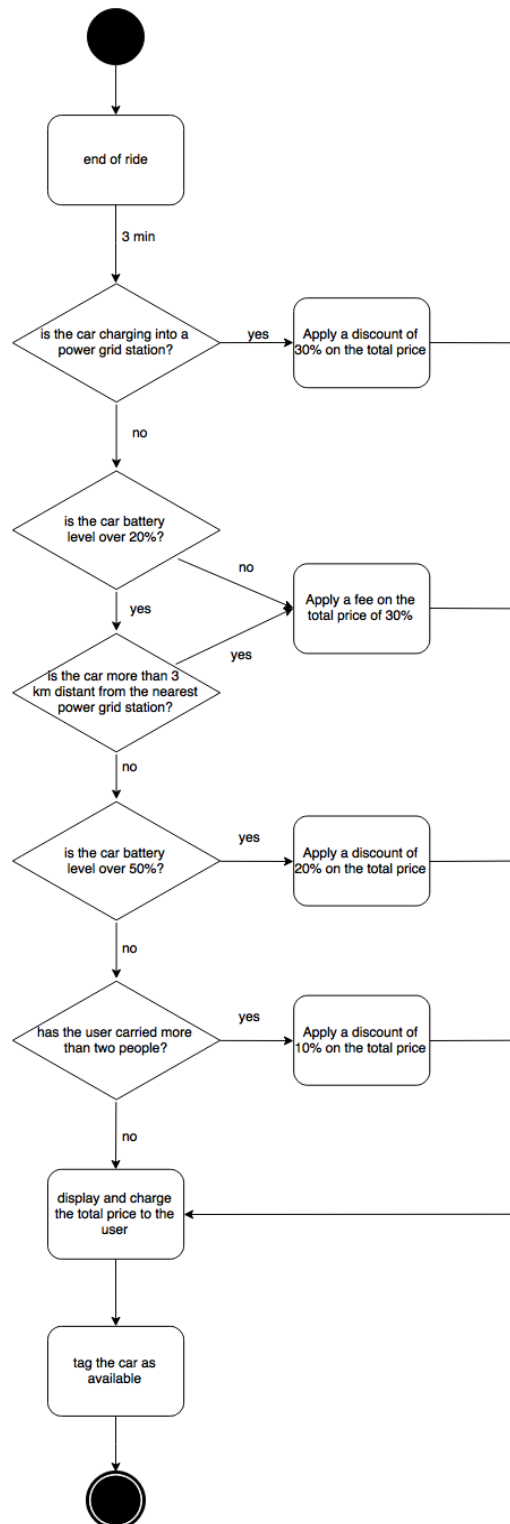


5.2. Activity diagrams

5.2.1. Process of unlocking a car



5.2.2. Process of finishing a ride



6. Alloy modeling

We've used the Alloy Analyzer tool to specify the properties of our application, starting from the representation of the class diagram of section 4.1. In this paragraph we will illustrate the Alloy code used for the analysis and some examples of worlds generated with the analyzer.

6.1. Model

open util/boolean

```
sig Char{}  
sig Name{}  
sig Surname{}  
sig Password{}  
sig Email{}  
sig DrivingLicence{}  
sig CreditCard{}  
sig PhoneNumber{}  
sig Ssn{}
```

abstract sig Position {}

abstract sig CarState {}

one sig Available, Unavailable, OutOfOrder **extends** CarState {}

abstract sig BatteryLevel {}

one sig Zero, Low, Medium, High **extends** BatteryLevel{}

sig Car {

 idNumber: **Int**,
 batteryLevel: BatteryLevel,
 position: **one** Position,
 carState: CarState,
 inCharge: Bool

}

 idNumber > 0
 (**one** unavailable: Unavailable | unavailable **in** carState) => ((**one** res: Reservation |
res.expired.isFalse && res.reservedCar=**this**) && (**no** ride: Ride | ride.terminated.isFalse &&
ride.reservation.reservedCar=**this**)) ||

 ((**one** ride: Ride | ride.terminated.isFalse && ride.reservation.reservedCar=**this**) && (**no** res:
Reservation | res.expired.isFalse && res.reservedCar=**this**))) //unavailable means that the car has been
reserved or is riding

 (**one** outOfOrder: OutOfOrder | outOfOrder **in** carState) => ((**no** ride: Ride | ride.terminated.isFalse
&& ride.reservation.reservedCar=**this**) && (**no** res: Reservation | res.expired.isFalse &&
res.reservedCar=**this**)) //outOforder means that the car cannot be reserved
}

fact UsersCannotReserveOutOfBatteriesCars{

no res: Reservation | res.expired.isFalse && (**one** zero: Zero | zero **in** res.reservedCar.batteryLevel)

}

```

fact CarIdNumbersAreUnique{
    all c1,c2: Car | (c1 != c2) => c1.idNumber != c2.idNumber
}

fact DifferentCarsCannotOccupySamePosition{
    all c1,c2: Car | (c1 != c2) => c1.position != c2.position
}

fact CarsInChargingOccupyOnePowerStation{
    all c: Car | c.inCharge.isTrue => (one ps: PowerStation | ps.position=c.position)
}

fact AvailableCarsMustBeInASafeArea{
    all c: Car | (one available: Available | available in c.carState) => (one sa: SafeArea | c.position in
sa.positions)
}

sig RegistrationInfo {
    ssn: one Ssn,
    name: one Name,
    surname: one Surname,
    phoneNumber: one PhoneNumber,
    email: one Email,
    creditCardNumber: one CreditCard,
    drivingLicenceNumber: one DrivingLicence
}

fact AllRegistrationInfoAreRelatedToOneUser{
    all ri: RegistrationInfo | (one u: User | u.registrationInfo=ri)
}

fact PersonsCannotSignUpTwice{
    all r1, r2: RegistrationInfo | (r1 != r2) => (r1.ssn != r2.ssn)
}

fact SomeInfoCanBePartAtMostOfOneRegistration{
    all r1, r2: RegistrationInfo | (r1 != r2) => (r1.phoneNumber != r2.phoneNumber)
    all r1, r2: RegistrationInfo | (r1 != r2) => (r1.email != r2.email)
    all r1, r2: RegistrationInfo | (r1 != r2) => (r1.drivingLicenceNumber != r2.drivingLicenceNumber)
    all r1, r2: RegistrationInfo | (r1 != r2) => (r1.creditCardNumber != r2.creditCardNumber)
}

sig User {
    password: one Password,
    position: one Position,
    registrationInfo: one RegistrationInfo
}

fact DifferentUsersCannotHaveSameRegistrationInfo{
    all u1, u2: User | (u1 != u2) => (u1.registrationInfo != u2.registrationInfo)
}

```

```

sig Reservation {
    expired: Bool,
    reservedCar: one Car,
    reservingUser: one User
}

expired.isFalse=> (one unavailable: Unavailable | unavailable in reservedCar.carState) //if a car is
reserved, it is not available
}

sig SafeArea {
    positions: disj set Position,
    powerGridStation: set PowerStation
}

all ps: powerGridStation | ps.position in positions //every power station position is contained in the
set of positions of the safe area to which it belongs
#positions>=1
}

//the set of positions in the range of 3 km from a power station
sig NonFineArea{
    positions: set Position
}

some pos: positions | (one ps: PowerStation | ps.position=pos)
all pos: positions | (one sa: SafeArea | pos in sa.positions)
}

fact AllPowerStationsAreInANonFineArea{
    all ps: PowerStation | some nfa: NonFineArea | ps.position in nfa.positions
}

sig PowerStation {
    position: one Position,
    available: Bool
}

available.isFalse <=> (one c: Car | c.position=position && c.inCharge.isTrue) //a power grid station
is unavailable when a car position matches its position and the car is in charge
}

fact PowerStationsMustBeInASafeArea{
    all ps: PowerStation | (one sa: SafeArea | ps.position in sa.positions)
}

fact DifferentPowerStationsHaveDifferentPositions {
    all ps1,ps2: PowerStation | ps1!=ps2 => ps1.position != ps2.position
}

sig Ride {
    reservation: one Reservation,
    duration: Int, //expressed in seconds
    passengers: Int,

```

```

    totalPrice: Int, //should be float
    terminated: Bool,
    endRidePosition: lone Position,
    endRideBatteryLevel: lone BatteryLevel,
    endRideIsInCharge: one Bool,
    moneySavingOptionActivated: Bool,
    moneySavingOptionInfo: lone MoneySavingOption,
    accidentReport: Bool //after an accident report occurs the ride ends
}

    duration>0
    passengers>0
    passengers<=5
    totalPrice>0
    (moneySavingOptionInfo=none) <=> moneySavingOptionActivated.isFalse //money saving option
info are present only if the option has been activated
    terminated.isFalse => (one unavailable: Unavailable | unavailable in
reservation.reservedCar.carState) && (reservation.reservedCar.inCharge.isFalse) //the car is unavailable
during the ride and cannot be in charge
    (endRidePosition != none) <=> terminated.isTrue //end ride position is saved only once the ride has
been terminated
    (accidentReport.isFalse && terminated.isTrue) => (one sa: SafeArea | endRidePosition in
sa.positions) //in order to terminate the ride the car must be parked in a safe area unless an accident
occurs
    (accidentReport.isTrue) => terminated.isTrue //the accident report implies the termination of the
ride
    reservation.expired.isTrue //since the ride exists, the respective reservation is terminated
    terminated.isTrue => (endRidePosition!=none && endRideBatteryLevel!=none &&
endRideIsInCharge!=none) //whenever a ride is terminated all the field related to the end of the ride are
not empty
    endRideIsInCharge.isTrue => (one ps:PowerStation | endRidePosition=ps.position) //if the car
result in charge at the end of the ride then its position at the end of that ride matches that of a power
station
}

fact OneReservationIsRelatedAtMostToOneRide{
    all res: Reservation | lone ride: Ride | ride.reservation=res
}

sig MoneySavingOption {
    startingPosition: one Position,
    targetPosition: one Position,
    selectedPowerStation: one PowerStation
}

    startingPosition != targetPosition
}

fact EveryMoneySavingOptionBelongsExactlyToOneRide{
    all mso: MoneySavingOption | one r: Ride | r.moneySavingOptionInfo=mso
}

fact UsersCanJustRideOncePerTime {

```

```

    no disj r1,r2: Ride | r1.terminated.isFalse && r2.terminated.isFalse &&
r1.reservation.reservingUser=r2.reservation.reservingUser
}

fact PowerStationSelectedByMoneySavingOptionMustAlwaysBeAvailable{
    all r: Ride | (r.terminated.isFalse && r.moneySavingOptionActivated.isTrue) =>
((r.moneySavingOptionInfo.selectedPowerStation.available.isTrue)&&(no car:Car |
r.moneySavingOptionInfo.selectedPowerStation.position!=car.position))
}

assert RunningsRidesAndReservationsCannotInvolveAvailableOrOutOfOrderCars{
    all c: Car | (one available: Available | available in c.carState) || (one outOfOrder: OutOfOrder |
outOfOrder in c.carState) => ((no r: Ride | r.reservation.reservedCar=c && r.terminated.isFalse) && (no
res: Reservation | res.expired.isFalse && res.reservedCar=c))
}

assert UnavailableCarsAreRunningOrReserved{
    all c: Car | (one unavailable: Unavailable | unavailable in c.carState) => ((one r: Ride |
r.terminated.isFalse && r.reservation.reservedCar=c) || (one res: Reservation | res.expired.isFalse &&
res.reservedCar=c))
}

assert PowerStationsBelongToOneSafeArea{
    no disj sa1,sa2: SafeArea | some ps: PowerStation | ps in sa1.powerGridStation && ps in
sa2.powerGridStation
}

assert NumberOfReservationsGreaterOrEqualThenNumberOfRides{
    #Reservation>=#Ride
}

assert ACarCannotBeInvolvedIntwoNonTerminatedRides{
    no disj r1,r2: Ride | r1.terminated.isFalse && r2.terminated.isFalse &&
r1.reservation.reservedCar=r2.reservation.reservedCar
}

assert ReservedCarsCannotBeInvolvedInANonTerminatedRide{
    no car: Car | (one res: Reservation | res.expired.isFalse && res.reservedCar=car) && (some ride:
Ride | ride.terminated.isFalse && ride.reservation.reservedCar=car)
}

assert RunningCarsCannotBeInCharge{
    no car: Car | car.inCharge.isTrue && (one ride: Ride | ride.terminated.isFalse &&
ride.reservation.reservedCar=car)
}

assert CarsInChargingAreInANonFineArea{
    all car:Car | car.inCharge.isTrue => (some nfa: NonFineArea | car.position in nfa.positions)
}

pred MoneySavingOptionBonusAchieved(r: Ride){
    r.terminated.isTrue &&

```



```

    r.moneySavingOptionActivated.isTrue &&
    r.moneySavingOptionInfo.selectedPowerStation.position=r.endRidePosition &&
    r.endRidelsInCharge.isTrue &&
    r.accidentReport.isFalse
}

pred ChargingBonusAchieved(r: Ride){
    r.terminated.isTrue &&
    r.endRidelsInCharge.isTrue &&
    r.accidentReport.isFalse
}

pred FineForParkingInAFineArea(r: Ride){
    r.terminated.isTrue &&
    r.accidentReport.isFalse &&
    no nfa: NonFineArea | r.endRidePosition in nfa.positions
}

pred PassengersBonusAchieved(r: Ride){
    r.terminated.isTrue &&
    r.passengers>=2 &&
    r.accidentReport.isFalse &&
    one nfa: NonFineArea | r.endRidePosition in nfa.positions &&
    (one medium: Medium | medium in r.endRideBatteryLevel)
}

pred HighBatteryBonusAchieved(r: Ride){
    r.terminated.isTrue &&
    r.endRidelsInCharge.isFalse &&
    one nfa: NonFineArea | r.endRidePosition in nfa.positions &&
    one high: High | high in r.endRideBatteryLevel &&
    r.accidentReport.isFalse
}

pred FineForLowBattery(r: Ride){
    r.terminated.isTrue &&
    r.endRidelsInCharge.isFalse &&
    one low: Low | low in r.endRideBatteryLevel &&
    r.accidentReport.isFalse
}

pred showPred1{
    one ride: Ride | MoneySavingOptionBonusAchieved[ride] &&
    one ride: Ride | ChargingBonusAchieved[ride]
}

pred showPred2{
    one ride: Ride | FineForParkingInAFineArea[ride] &&
    one ride: Ride | PassengersBonusAchieved[ride]
}

pred showPred3{

```

```

    one ride: Ride | HighBatteryBonusAchieved[ride] &&
    one ride: Ride | FineForLowBattery[ride]
}

run showPred1
run showPred2
run showPred3
check RunningsRidesAndReservationsCannotInvolveAvailableOrOutOfOrderCars
check UnavailableCarsAreRunningOrReserved
check PowerStationsBelongToOneSafeAre
check NumberOfReservationsGreaterOrEqualThenNumberOfRides
check ACarCannotBeInvolvedInTwoNonTerminatedRides
check ReservedCarsCannotBeInvolvedInANonTerminatedRide
check RunningCarsCannotBeInCharge

```

6.2. Alloy Analyzer results

This output from the Alloy Analyzer tool shows the consistency of our model:

10 commands were executed. The results are:

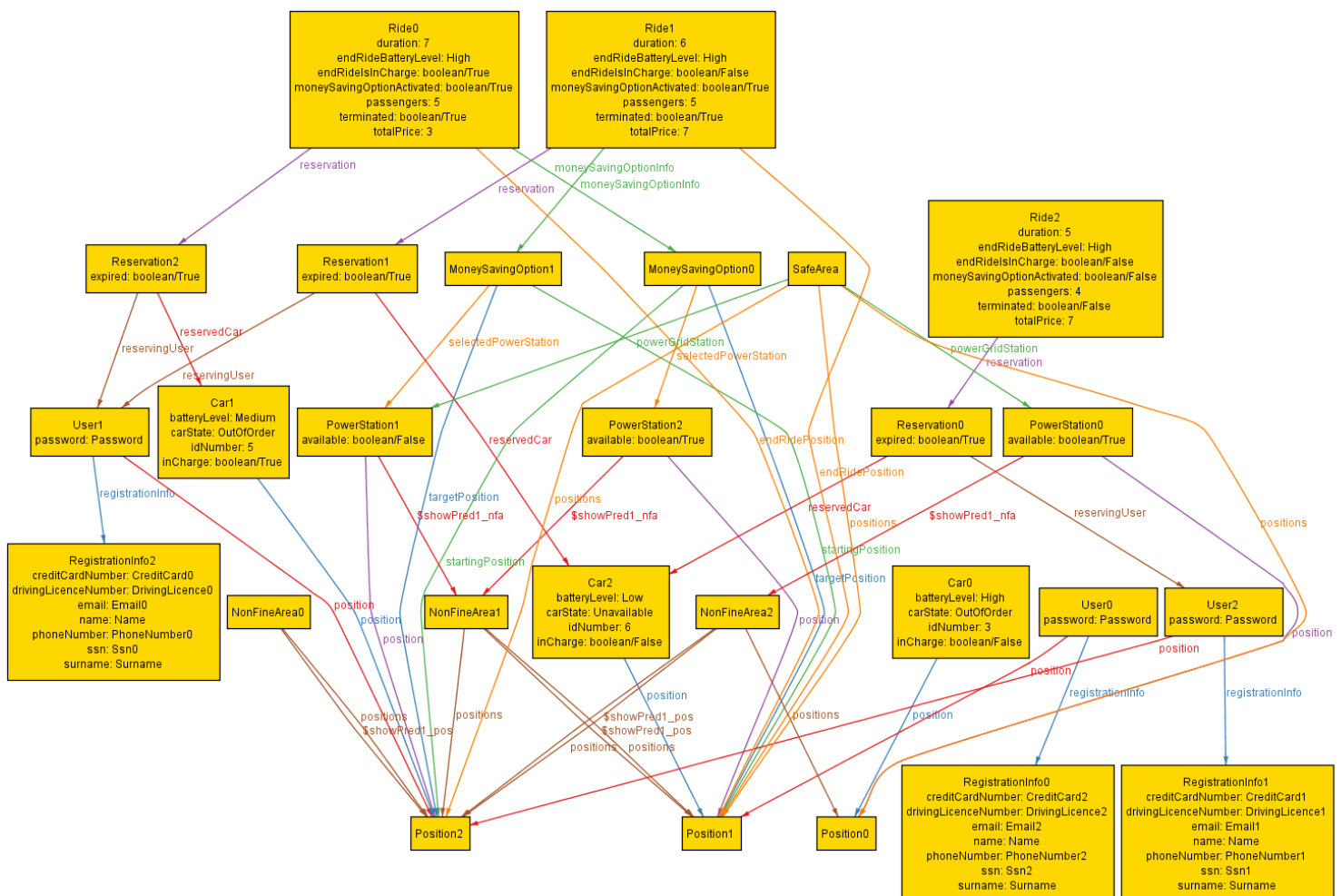
- #1: **Instance Found.** showPred1 is consistent.
- #2: **Instance Found.** showPred2 is consistent.
- #3: **Instance Found.** showPred3 is consistent.
- #4: No counterexample found. RunningsRidesAndReservationsCannotInvolveAvailableOrOutOfOrderCars may be valid.
- #5: No counterexample found. UnavailableCarsAreRunningOrReserved may be valid.
- #6: No counterexample found. PowerStationsBelongToOneSafeAre may be valid.
- #7: No counterexample found. NumberOfReservationsGreaterOrEqualThenNumberOfRides may be valid.
- #8: No counterexample found. ACarCannotBeInvolvedInTwoNonTerminatedRides may be valid.
- #9: No counterexample found. ReservedCarsCannotBeInvolvedInANonTerminatedRide may be valid.
- #10: No counterexample found. RunningCarsCannotBeInCharge may be valid.

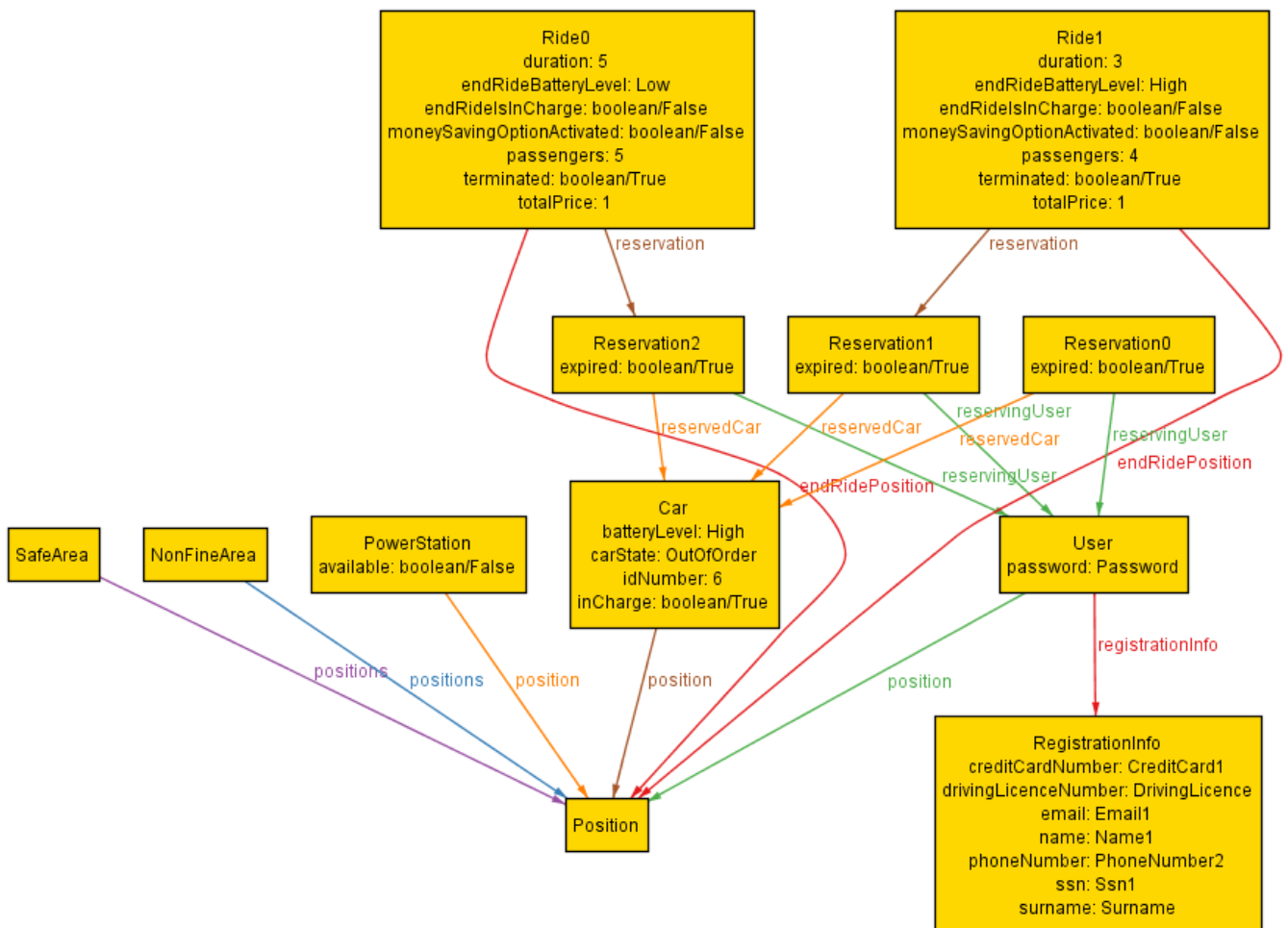
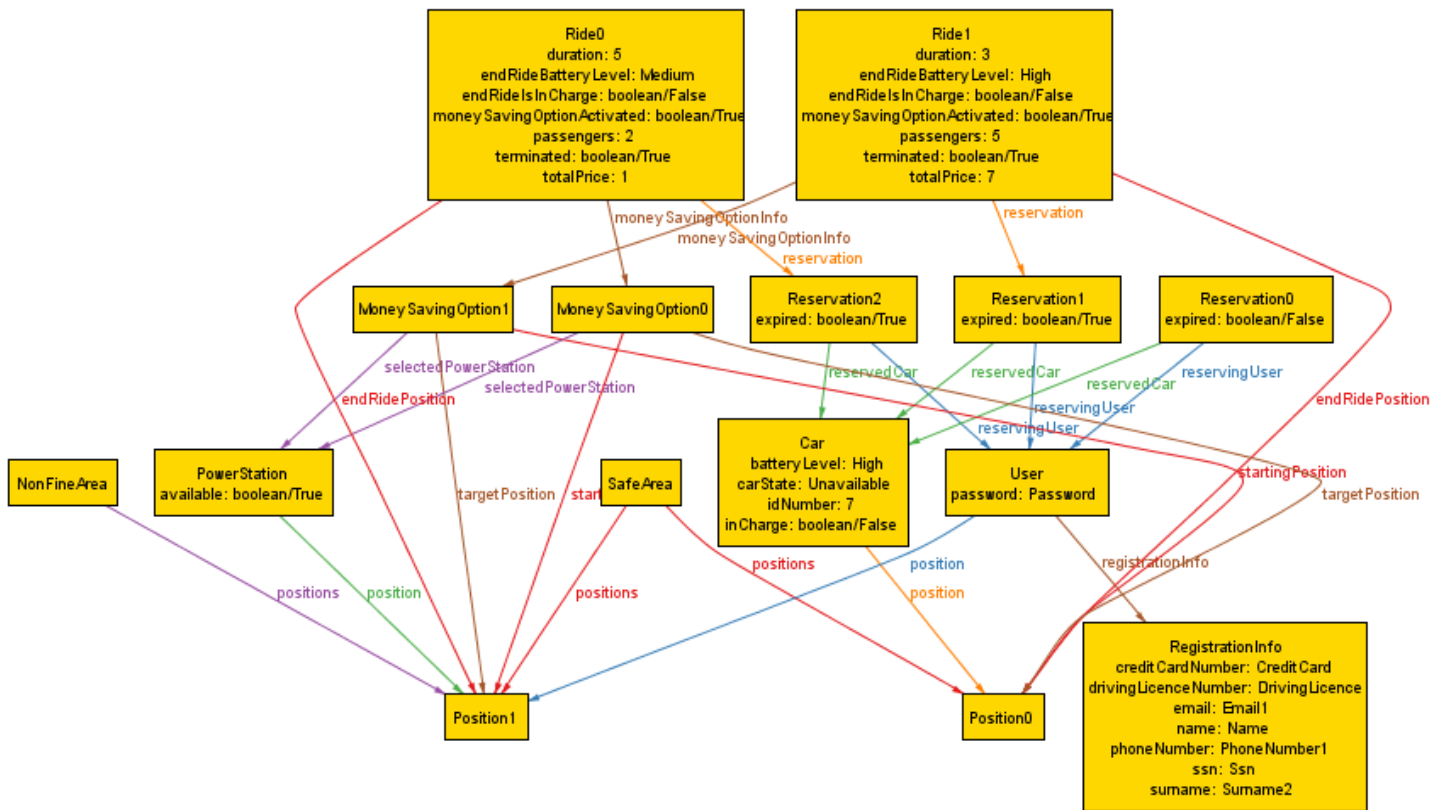
6.3. World generated

Once done the model, we wrote 6 predicates each of them corresponds to a ride in which discounts were achieved or fines have been applied. The three generated worlds derive from three combinations of these six predicates, in particular:

- showPred1 is a world in which there is at least one ride in which the user has used properly the money saving option and at least one when he obtained the discount for finishing the race leaving the machine in a power station.
- showpred2 is a world in which there is at least one ride in which the user was fined for parking too far from a power station and one in which the user got the passengers discount.
- showPred3 in which there is at least one ride in which the user has obtained a discount for finishing the race with battery over 50% and one in which a fine has been applied for having left the car with battery under 20%.

The representations of the worlds generated from showPred1, showPred2 and showPred3, respectively, are shown below.





7. Other info

7.1. Future development

One possible future improvement of PowerEnjoy system can be an option which allows users to report car accidents or damages directly on their mobile or web app, and the possibility to store those reports in the system so that the assistance coordinator can read and manage them in a better way.

7.2. Reference documents

- Assignments AA 2016-2017.pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- Sample documents:
 - RASD sample from Oct. 20 lecture.pdf
 - RASD_meteocal-example1.pdf
 - RASD_meteocal-example2.pdf
 - RASD_meteocal-example3.pdf
- Course slides:
 - Requirement engineering Part I.pdf
 - Requirement engineering Part II.pdf
 - Requirement engineering Part III.pdf

7.3. Used tools

- Microsoft Word 2016, for the drafting of RASD
- Microsoft OneDrive, to allow concurrent editing
- GitHub, to store the project in a repo
- Draw.io, for the drawing of the diagrams
- Alloy Analyzer 4.2, to prove the consistency of our model
- Balsamiq, for the mockups

7.4. Hours of work

For redacting and writing the Requirements Analysis and Specification Document we spent approximately 35 hours per person.

7.5. Changelog

Version 1.1

- Minor changes in the drafting of the text
- Added cardinalities in the class diagram in section 4.1.
- Added a description for the «extend»/«include» relationships of the use case diagram in section 3.2.
- Removed the AccidentReport entity from the class diagram and the Alloy model