Politecnico di Milano

AA 2016-2017
Software Engineering 2 project: "*Power EnJoy*"
Prof. Mottola Luca

RASD

<u>R</u>equirement <u>A</u>nalysis and <u>S</u>pecification <u>D</u>ocument

11/12/2016 - version 2.1

Edited by:
Giulia Pennati **878686**
Paola Sanfilippo **809689**
Marco Siracusa **878083**

Contents

1. Introduction

1.1. Purpose

This document represents the Requirement Analysis and Specification Document (RASD). Its main purpose is to describe the requested system, from the basis of

- functional requirements, that represents all the functionality offered by the system both on interfaces interacting with the user and on the ones interacting with system components, and
- non-functional ones regarding performance and quality of the software we want to implement.

At first we provide a detailed analysis about the domain of the system, goals, assumptions, and the requirement themselves, then we provide a graphic representation of all these through Use cases, class diagram, activity diagram (we chose BPMN diagram, because it seemed to be the most suitable for our needs), state charts and sequence diagram, until the model verification through alloy. This document is addressed to all developers and programmers that will implement the system, to stakeholders that commissioned it, and it could be used as a contractual basis between the customer and the developer.

1.2. Description of the given problem

This project aims to realize an electric car sharing service.

This service makes available to the users an entire fleet of electric cars, power grid stations furnished with plugs to recharge the cars and a maintenance team ready to intervene in case of need.

It also provides both a website and a mobile application, with which the registered users can access the system functionalities.

In order to access these system functionalities, a person should first register, inserting all the required data (that has to be correct), and then wait until the reception of an email containing a password.

The email (s)he provided during the registration with the password received via email are his/her credentials to log into the system and access all the functionalities.

A logged in user can make a research of a car within a certain area, the "Safe Area", choosing the one that most fits his/her needs, and reserve the desired one.

(S)He also can use the car according to the PowerEnJoy company's rules (that will be explained in the next pages) and can both enjoy the different available discounts and be charged with extra fees, limited to a discount of 30% (i.e. +charges-discounts ≤ 30% total discount).

Discounts and charges policy is thought to reward those behaviors that simplify the availableness of usable cars (i.e. in the right place and with a sufficient battery level) and discourage the others.

1.3. Actual system

PowerEnJoy's company has no previous system so it is required to create the whole system.

1.4. Actors

- **Visitor**: a visitor can only see the registration phase. In particular, from this (s)he is able to fill a form and complete the registration to become a registered user and to access all the system functionalities.
- **User**: a registered user that, after the login, can access all the system functionalities.
- **Car**: an electric car from the PowerEnJoy's fleet.
- **Data checking system**: a third-part system that checks the user data to validate the registration
- **Payment system**: a third-part system that checks whether there's enough money to complete a transaction and, if so, charges the user of the due amount.
- **Maintenance system**: a component of the system that deals with the maintenance of the service.

1.5. Goals

1.5.1. Visitors:

[G|v01] Should be able to insert all the data mandatory for the account registration (see [A|v01]). Should also be able to provide payment method information (see [A|v01]) to the system in order to be charged automatically for the use of the service.

[G|v02] Eventually, once the data has been checked by an external data checking system, should have the account accessible giving a password back (generated randomly) to use for accessing the system functionalities or, otherwise, an error describing the issue.

1.5.2. Users:

[G|u01] Should be able to log into the system (through email address inserted in the registration phase and password provided back) to use its functionalities.

[G|u02] Once logged in, should be able to find the location of not reserved cars within a 500m range from his/her current location or from an address specified each time.

[G|u03] Once the not reserved cars have been shown, if (s)he has enough money (as pledge) to reserve a car and all the previous transaction have been carried out (i.e. if the user is "not owing") and has not another session open, should be able to request the reservation of one car and have a confirmation back

[G|u04] Once reserved a car, should have 1€ fine and the deletion of the reservation in case (s)he does not unlock the car within one hour from the receiving of the reservation's confirmation.

[G|u05] Once within 100m from the reserved car, should be have the car unlocked through a request from the mobile application

[G|u06] Should have a 10% discount on the price of the ride if (s)he has more than two passengers onto the car at the beginning of the ride.

[G|u07] Should have a 20% discount on the price of the ride if (s)he ends the ride and the car has a level of percentage of battery charge more or equal than 50%.

[G|u08] Should have a 30% discount on the price of the ride if (s)he, within 2 minutes starting from when the system locks the car, plugs that car in the proper power grid station.

[G|u09] Should have a 30% charge on the price of the ride if (s)he ends the ride and the car is more than 3Km farther from any power grid station or the level of percentage of battery charge less than 20%.

[G|u10] Should be charged of the deploy time at the net amount of the charges/discounts to be applied on the ride. The procedure starts just after 2 minutes from when the system locks the car, when all the charges/discounts have been assessed

[G|u11] When the user enables the Money saving option through the car terminal, should fetch from the car interface the destination address that the user wants to reach and provide to the car the address of a power grid station near the specified address but with a constraint on the availability of the plugs.

1.5.3. Cars:

[G|c01] Should be marked as "reserved" by a certain user or "not reserved"

[G|c02] Should be able to retrieve information about the current charge to show to the user through a screen in the cabin

[G|c03] Should be locked as soon as all the people are out of the car (and the doors are closed) and the engine has already been switched off.

1.5.4. Payment system:

[G|p01] Should receive a request and be able to give a feedback on whether a user has enough money (as pledge) to start a reservation

[G|p02] Should receive the coordinates of the transactions to carry out, due to the use of the service by the user, and able to communicate the result of the transaction (done/denied). If the user does not have enough money, the payment request is repeated over just after a fixed time slot.

1.5.5. Data checking system:

[G|d01] Should receive the visitor's data to check and be able to give a feedback on whether those data are correct or not.

1.5.6. Maintenance team:

[G|m01] Should receive a notification when a car has a battery charge percentage less than 20% in order to draw that in a power grid station, plug it (and it will be the car that notifies the system when it is plugged).

1.6. Glossary

1.6.1. Definitions

[D01] **Visitor**: a person who has not already completed the registration phase. An inaccessible account is assigned to him/her.

[D02] **Account**: a record into the system that identifies each visitor/user and contains all his/her data. Each account can be either "Not owing" or "Owing" depending on whether the user has carried out all payments or not.

[D03] **Registration phase**: procedure that starts with a visitor that sends the required data and ends receiving a password back from the system (being allowed to log into that and use all the provided functionalities).

[D04] **User**: any person that, consequently a registration phase, has received back the password and so is able, after the login phase, to access all the system functionalities.

[D05] **Login**: procedure with which a user (already registered), inserting his/her username and password, accesses into the system in order to use all the provide functionalities.

 [D06] **Car**: any electric vehicle of the provided fleet. This vehicle is equipped with all the required sensors to provide the information necessary to the system.

The car can be:
- Not reserved: has not been reserved yet, so it's ready to be reserved
- Reserved: has already been reserved, so there's another user that has reserved the car or is using it

And
- Plugged (to a power grid station)
- Not plugged (to any power grid station)

Parts of the car that are named throughout the text are:
- Doors
- Trunk
- Battery
- Internal screen
- Terminal

A car provides an interface to communicate with the system.

[D07] **Reservation**: action that makes the cars change their state from "reserved" to "Not reserved" and vice versa. It **starts** when the user receives the confirmation of the requests of the car's reservation done through the mobile application and **ends** when the user unlocks the car

[D08] **Reservation-time**: time elapsed from the beginning to the end of the reservation.

[D09] **Locking**: action requested by the system that consists in the locking of the car's doors and trunk too.

[D10] **Unlocking**: action requested by the system that consist in the unlocking of the car's doors and trunk too.

[D11] **Deploy**: action that intends the effective use of the service that the user is charged for. It **starts** from the unlocking of the car and **ends** on the locking.

[D12] **Deploy-time**: time elapsed from the beginning to the end of the deploy.

[D13] **Ride**: action that **starts** when the engine is switched on and **ends** when the engine is switched off.

[D14] **Ride-time**: time elapsed from the beginning to the end of the ride.

[D15] **Passenger**: person who, for a single ride, travels with the user inside the reserved car.

[D16] **Power grid station**: place equipped with company's plugs whereby the user can put the car on charge. These are placed only in the safe area and well distributed. Each one has a total number of plugs.

[D17] **Safe area**: places within the urban city area where the user is allowed to park the car in. It has been already defined before the deploy phase.

[D18] **Payment system**: external payment system that is supposed to carry out the transactions of the due amounts of money of the several users. It communicates with the developed system through an interface.

[D19] **Pledge**: Amount of money (100€) used as security for the fulfillment of the reservation debt. It is liable to forfeiture the damage of the car or the inability to afford the payment of the reservation.

[D20] **Data checking system**: system that checks the validity and completeness of the user's data. The feedback is positive if and only if the inserted data are completed with respect to the required ones and valid with respect to the laws applied in the country where the system is running.

[D21] **Money saving option**: option that, when selected, drives the user to the power grid station that is nearest to the destination (s)he selected but with not less than half of the plugs available. If the user leaves and plugs the car in the suggested place, (s)he receives the discount.

[D22] **Maintenance team**: team that manages the cars that has been left with a percentage battery level less than 20%

[D23] **Research**: procedure that allows the user to see which cars (s)he can reserve. The showed cars have battery charge level more or equal 20%, are not reserved and within a 500m range from the user's current provided location or from an address specified.

[D24] **2 minutes wait**: time elapsed from the locking of the car and the moment in which the system checks whether the car has been plugged or not and computes the total amount of discounts and recharges

1.6.2. Gantt chart for time subdivision

Session	SESSION
Log in	LOG IN
Research	RESEARCH
Reservation	RESERVATION
Deploy	DEPLOY
Unlock	UNLOCK
Drive	DRIVE
Lock	LOCK
2 minutes wait	2 MINUTES WAIT
Discounts computation	DISCOUNT AND RECHARGES COMPUTATION
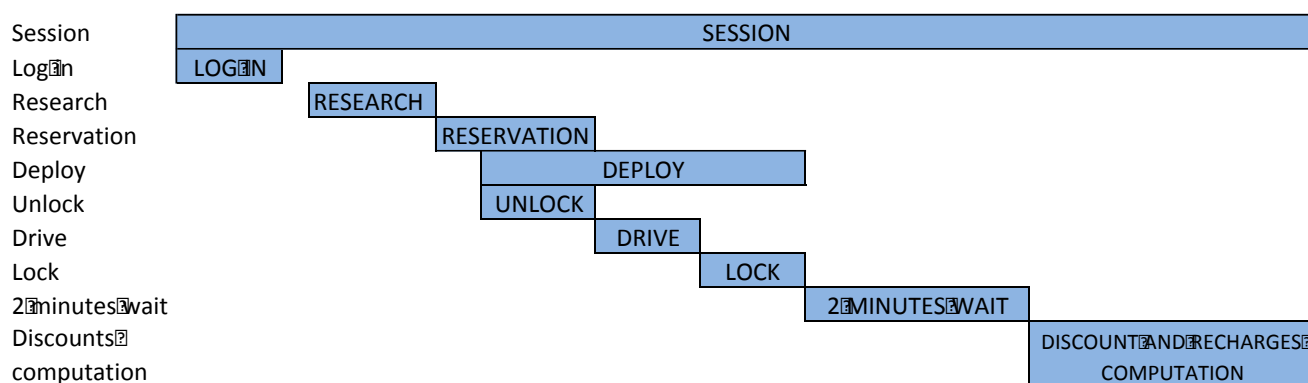
2. Overall description

2.1. Assumptions

2.1.1. Visitor:

[A|v01] For the registration phase the company requires to the visitor information regarding:

- Personal data (i.e. first name, last name, sex, e-mail address, date of birth, birthplace, actual address, city, phone number)
- Driving license data (i.e. number)
- Identity card data (i.e. tax id code)
- Payment card data (i.e. card number, security code)

(S)he is also required, during the registration phase, to upload a scanned copy of his/her ID card and driving license.

2.1.2. User:

[A|u01] If the system requires it, always provides his/her current position to the system with a 10m accuracy

[A|u02] The user is always able to end the ride and therefore left a car in a safe area without running out of battery

[A|u03] The users are reliable: they won't try to cheat in order to have discounts and behave unfairly.

[A|u04] The user is supposed to drive and park respecting the Highway Code.

2.1.3. Car:

[A|c01] Is a third party component that communicate with system through an interface, providing information about:

- the percentage of the current battery level
- the number of people inside the car
- when the engine is switched on/off (messages to the system)
- whether all the doors are closed
- when the car is plugged in a power grid station
- current position with a 10m accuracy
- interaction with its terminal and the user

[A|c02] Receives directives by the system to lock/unlock it

[A|c03] Can be plugged only in the company's power grid station

[A|c04] Has a screen into the cabin, visible by the user, managed by the car

[A|c05] Receives information about the current charge of the user that is driving and displays it through an internal screen

[A|c06] Receives the address to which navigate the user that enabled the Money saving option

[A|c07] If marked as "plugged" and providing a position that is within 50m the position of a power grid station, that car is considered plugged to that power grid station

2.1.4. Payment system:

[A|p01] Is a third party component that communicate with system through an interface, providing information about:

- whether a user has enough money (as pledge) to afford a reservation
- when a payment transaction requested by the system is carried out (sending of a message to the system)

Through the same interface it is also able to receive the data of the payment transaction to carry out in order to make the user pay the use of the service provided by the system

2.1.5. Data checking system:

[A|d01] Is a third party component that communicate with system through an interface, providing a feedback on whether the user's data sent to him are valid and complete or not.
The data have to be valid and complete with respect to the law in the country where the system works in.
Therefore, the system checks on whether the inserted data correspond to the document information and that the documents are valid.
[A|d02] Is supposed to check the data in at most 24 hours from data's reception.

2.1.6. Maintenance team:

[A|m01] When an intervention is required by the system, the maintenance team is supposed to reach the position of the car that has to be recharged, draw it up to a power grid station and plug it.

2.1.7. Other:

[A|g01] Any exception can be managed calling a free-toll number
[A|g02] The safe areas have already been inserted in the system
[A|g03] All the unit of measurement are expressed according to the International System of Units
[A|g04] The documents, once checked, are valid for the whole period in which the user has an account in the system
[A|g05] The communication among the several systems is always possible, correct and considered as immediate
[A|g06] The position and total plugs of the power grid stations have already been inserted into the system

3. Specific requirements

3.1. Requirements

Requirements are divided in functional (f) and non functional (n)

[R01f] The system should allow the visitors to insert all the data required for the registration (see [A|v01]) of a new account and create the account initially marked as "Not Owing" since no transactions have to be paid yet.
Note: the account is not accessible until a password to access is provided to the user.
[R02f] The system should also allow the visitor to insert the payment method information (see [A|v01]).
[R03f] The system should eventually send to the data checking system the data inserted by the visitor. The Data checking system is supposed to check that data and give a feedback message to the system. If the feedback is positive, the system generates and gives a password back to the visitor in order to allow him/her to access the system functionalities; otherwise, if the feedback is negative, the system sends a message to the visitor with the occurred issue and provides a link that carries to a proper page where the user can do the request over just modifying the wrong fields.
[R04n] The system should provide a page (web and mobile) to the visitor that allows him/her to fill in the fields mandatory for the account registration and allows him to modify the wrong fields in case of doing the operation over. The account password or the error message and link (for doing the registration over) are sent to the user through email as soon as the data have been checked by the Data checking system.
Note: The system's response time closely depends on the Data checking system's response time since the data elaboration and forward time is considered not significant.

[R05f] The system should allow a user to insert the credential (email and password) to access the system functionalities.
[R06f] The system should check the inserted login credentials (that match with an email password pair in the database) to allow the user to access the system functionalities or deny him/her the access otherwise, notifying the issue.
[R07n] The system should provide a page (web and mobile) that allows the users to insert the access credential (link).
The response time should be within 30 seconds.

[R08f] The system should allow logged in users to find the locations of cars:
- marked as "not reserved" and
- having a battery charge level more or equal 20% and
- within a 500m range from his/her current provided location or from an address specified each time that, in case, should be inserted in the proper field of the form.

[R09n] The system should provide a page (mobile or web) that allows the users to select whether searching the cars using his/her current position or a specified address (link).
[R10n] The system should provide a page (mobile or web) that shows, in a map, the cars that come out by the research. Relevant information is shown for each car.

[R11f] Once the not reserved cars have been shown to the user, the system checks on whether the user's account is marked as "Not owing" and does not have any other session open and that the user has enough money (as pledge) to afford the ride through a query to the Payment system, checking the provided feedback.

[R12f] If the feedback is positive, the system should allow him/her to request the reservation of one car, marking that car as "reserved". The system should also notify the confirmation or rejection of the request.

[R13n] The system should provide a page (mobile and web) that allows the logged in user to pick out a car from the research result and notify the confirmation or rejection of the reservation.

The system should display the results of the research within 120 seconds starting from when the user provides his/her position.

[R14f] Starting from when the request is confirmed to the user, the system should give that user 1 hour of time to unlock the car. If the time goes over (end of the 60th minute) the system should charge that user of a 1€ fine, mark the car as "not reserved" again, and notifying the user of the event.

Note: the payment procedure is defined as if the user does not have enough money, (s)he can not reserve any other car until the covering of the debt

[R15f] If the reservation of the car by a user is still valid (time is not over), the system should allow the user to request the unlocking of the reserved car and, only afterward, check whether his/her position is within a 100m range from the car's position and, if so, unlock the car.

[R16n] The system should provide a mobile application page that allows the user, who requested a car, to unlock it.

The system should unlock the car within 10 seconds from when the user requests it.

[R17f] When the engine is turned on, the car sends to the system this information and the number of people inside: if it is more or equal than 3, the system keeps track that the cost of the ride needs a 10% discount to be applied. The system also keeps track of when the ride begins (beginning of the deploy time) to eventually assess the cost.

[R18f] When the system locks the car, the car sends to the system the percentage of battery charge: if it is more or equal than 50%, the system keeps track that the cost of the ride needs a 20% discount to be applied.

[R19f] The car, after 2 minutes from when it has been locked, tells the system whether it is plugged to a power grid station and, if so, the system keeps track that the cost of the ride needs a 20% discount to be applied.

[R20f] When the car has been locked, sends to the system its position and the percentage of battery charge and, if the position is more than 3Km farther than the position of any power grid station or the percentage of battery charge is less than 20%, the system keeps track that the cost of the ride needs a 30% charge to be applied.

[R21f] The system, when all the discounts/charges have been assessed, marks the car as "not reserved" again, request a maintenance intervention on the car if it has a percentage of battery charge less than 20% and if it is not plugged to any power grid station and eventually computes the price of the reservation as the deploy-time cost at the net of the amount of the discounts/charges. This value, that is the cumulative amount of the discounts and charges limited to a discount of 30% (i.e. +charges-discounts $\leq$ 30% total discount), is notified to the user. Then the system should charge that user of the due amount of money and notify the event to him/her.
Note: the payment procedure is defined as if the user does not have enough money, (s)he can not reserve any other car until the covering of the debt

[R22f] The system should receive the address sent by the car identifying the destination that the user wants to reach using the Money saving option
[R23f] When the money saving option is enabled, the system should respond providing the address of the power grid station nearest to the specified destination but with not less than half of the plugs available
Being the Power grid stations passive objects, the number of available plugs is retrieved from the position of the plugged cars (the ones with position within a 50m range from the power grid station) and the power grid station.
[R24n] The system should provide a response within 120 seconds starting from the request

[R25f] While the engine is on, the system should communicate to the car the current cost of the ride.
[R26n] The system should send new data every minute.

[R27f] As soon as the car notifies that the engine has been switched off, the system starts asking to the car on whether there are no people inside and all the doors are closed, when it happens, the system locks the car.
[R28n] As soon as the engine is turned off, the system, every 10 seconds, checks whether there are people inside the car and all the doors are closed.

[R29f] When the system should charge that user of the due amount of money, it sends to the Payment system the ID of the user and the due amount. The system should have a feedback on whether the transaction has been carried out.
If the feedback is positive, it notifies it to the user but if the feedback is negative it marks the user as "Owing" and notifies it to him/her.
[R30f] In case of negative feedback, the Payment system checks frequently and so the system sets the user as "Not owing" only once the transaction has been carried out successfully.
[R31n] The system should do the requests over each hour.

[R32f] The maintenance team should be notified when a car needs recharging and therefore its position is attached.
The maintenance team takes care of drawing the car to the next power grid station and once plugged the system should change the status of the car and consider it as plugged.

3.2. Goals-requirement-assumption link table

G|v01 ⇐ R01f + R02f + A|v01   Register an account
G|v02 ⇐ R03f + R04n + A|d01   Register an account

G|u01 ⇐ R05f + R06f + R07n   Log in
G|u02 ⇐ R08f + R09n + R10n + A|u01 + A|c01   Research a car
G|u03 ⇐ R11f + R12f + R13n + A|p01   Reserve a car
G|u04 ⇐ R14f + A|p01   Close the reservation
G|u05 ⇐ R15f + R16n + A|c02   Reserve a car
G|u06 ⇐ R17f + A|c01   Drive a car
G|u07 ⇐ R18f + A|c01 + R27f   Drive a car
G|u08 ⇐ R19f + A|c01 + A|c03   Drive a car
G|u09 ⇐ R20f + A|c01   Drive a car
G|u10 ⇐ R21f + A|p01 + A|u02 + A|g02   Drive a car
G|u11 ⇐ R22f + R23+ R24n + [A|c01] + [A|c03-7]   Enable the Money saving option

G|c01 ⇐ R12f + R14f + R21f   Reserve a car/
                                                                              Close the reservation/
                                                                              Drive a car

G|c02 ⇐ R25f + R26n + A|c04 + A|c05   Drive a car
G|c03 ⇐ R27f + R28n + A|c01 + A|c02   Drive a car

G|p01 ⇐ R11f + A|p01   Reserve a car
G|p02 ⇐ R21f + R29f + R30f + ~~R31n~~ + A|p01   Carry out a transaction

G|d01 ⇐ R03f + A|d01   Register an account

G|m01 ⇐ R21f + R32f + [A|m01]   Maintenance assistance
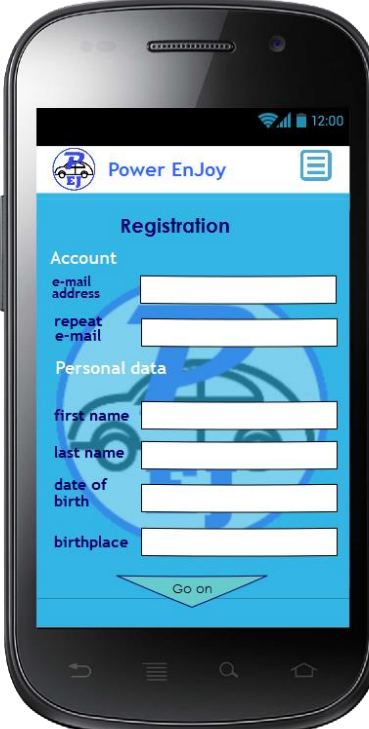
3.3. External interface requirements

3.3.1. User interfaces

Here we present a set of mockups that represents the interface provided to the user both on website and mobile application.

3.3.1.1. From mobile application

- **Registration**: This mockup shows the interface the user has to interact with during the registration phase by mobile app.

- **Login**: This mockup shows the interface the user has to interact with during the registration phase.

- **Research**: By this interface the user can research a "not reserved" car in a specific assigned position, and reserve it

- **Unlock**: When the user is nearby the reserved car, he can unlock it, by using this interface provided only by the mobile application.

- **Error**: It may happen that the user payment card doesn't have enough money to cover the pledge during the reservation or that a user with an account marked as "owing" tries to reserve a car, therefore the system gives an error message.

- **Fee recap**: Once the reservation is up, the user is able to see the paid amount, with all the possible discount voices (or charges) he collected.

- **Registration**: This mockup shows the interface the user has to interact with during the registration phase by mobile app.
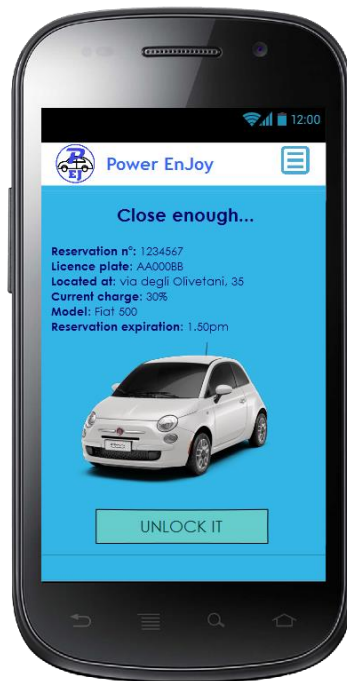
- **Login**: This mockup shows the interface the user has to interact with during the registration phase.
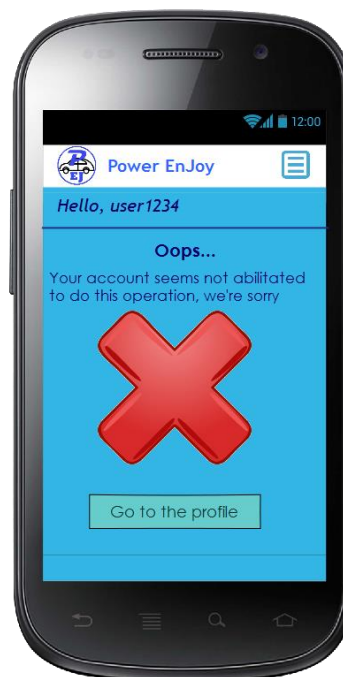
- **Research**: By this interface the user can research a "not reserved" car in a specific assigned position, and reserve it
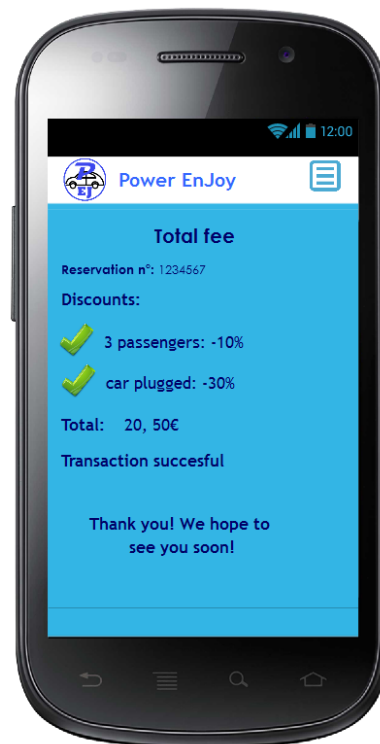
- **Reservation**: Once researched, the user can reserve a car

Power EnJoy

Payment history | Reservation history | Profile

Hello, user1234
Via Lanzone, Milano

OUR FLEET

RATES

BOOK A VEHICLE

INSTRUCTIONS

PARTNERSHIP

Reservation data

Licence plate: AA000BB
Placed at: via degli Olivetani, 35, Milan
Current charge: 30%
Model: Fiat 500
Hypotetical reservation expiration: 1.50pm

RESERVE IT

3.3.1.3. On the car's screen

At last we provide a mockup of the car screen during the ride.
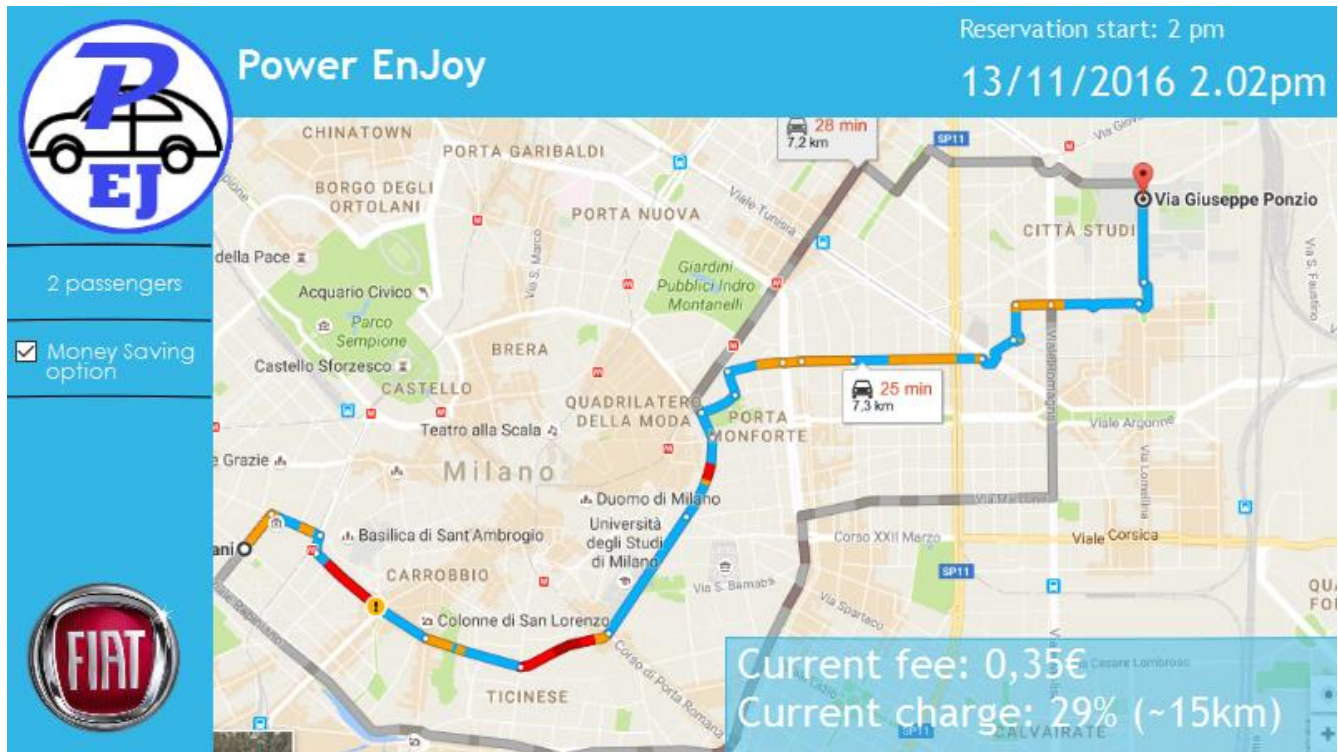
3.4. Scenarios
3.4.1. Scenario 1: registration
Steven is a student that has just moved in the city of the University he is going to attend. At the end of the semester he is supposed to hand in a project done with the collaboration of some classmates. Steven realizes that, in order to reach quickly the different locations where the team meets in, he should use a car. That leads him to register an account on the PoweEnJoy car sharing. For this reason, he is asked to fill in the registration form and therefore he starts inserting his email and personal data (first name, last name, sex, date of birth, birthplace, actual address, city and phone number). Then he takes the driving license from the wallet and copies carefully the driving license number written on the card. He also takes the ID card to fulfill the tax ID code field. The last step requires him to provide his bank account coordinates in order to carry out the payments for the using of the service: he is asked to insert the card number and the security code. Steven uploads also a scanned copy of his ID card and driving license. In a few, he receives an e-mail explaining him that the tax ID code's field has been filled wrongly. He takes the ID card and does the request over, inserting the right tax ID code this time. The system sends him an e-mail in which they told him that the registration has been completed successfully and now he can access as an authorized user to the system functionalities using his e-mail and the attached password.

3.4.2. Scenario 2: Reservation closed by the system
Now Steven wants to reserve a car, hence he opens the mobile application, in the login page he inserts the email provided in the registration phase and the password given back by the system and accesses into the car research page.
Living out of the urban area, and therefore the area of the service coverage, he is aware that no cars can be found and so he decides to look for cars nearby the next subway station within the urban area. So he inserts an address near the station and chooses the car that better fits his needs. He eventually receives the confirmation of the reservation. Steven cannot reach the destination address by an hour, therefore he loses his reservation on the car and furthermore he has a 1€ fine charged directly on his bank account.

3.4.3. Scenario 3: Reservation
As soon as he reaches the place where the car was supposed to be, he sees that it is still in place and he decides to do the request over. Once he's next to the car's door, he unlocks the car from the mobile application and he eventually goes into the car.

3.4.4. Scenario 4: End of the ride
Steven finally gets to the final destination then he parks the car in a safe area, switches the engine off and drops off the car noticing that the car is locked as soon as he is out of that.

3.4.5. Scenario 5: Money saving option
On the way back home, having a little more time, Steven and two other friends decide to reserve a car. They reach it and unlock it, then Steven selects the "Money saving options" in order to let the system suggests them a station where to plug the car in, that is nearby a provided destination address. Therefore he drives toward the station suggested by the system.

3.4.6. Scenario 6: Discounts 1
Reached the power grid station, Steven plugs the car in (within 2 minutes from when the doors are locked) and consequently he could enjoy the 30% discount on the ride and the 20% discount for carrying 2 other passengers and also a 30% charge due to the ending of the reservation with a battery charge percentage less than 20%, for a total maximum discount of 30%.
After 2 minutes from the door's locking and therefore the ending of the reservation, Steven can see that he has been charged for the deploy-time of the service usage at the net of the several charges/discounts.

3.4.7. Scenario 7: Discounts 2

The next meeting is hold in an area not so furnished with power grid stations, so Steven reserves a car, drives toward his destination but has to leave the car 3Km from any power grid station and also with a battery percentage level less than 20%. According to the system policies he is charged for the deploy time with a 30% charge on top.

Coming back home, he decides to pick up a car that is farther from the one previously left but with more battery charge percentage. For this reason, he can reach his final destination with a battery charge percentage that is more or equal to 50% and therefore have a discount of 20% on the current ride.

3.4.8. Scenario 8: User out of money for the pledge payment

George is in a hurry and so he decides to take a PowerEnJoy car. He does a research, filtering the cars by his current position, selects a car and tries to reserve it. But he receives a notification from the system telling him that he hasn't enough money to cover the entire pledge's amount, so he can't reserve a car. As soon as he will have enough money on his payment account, the system will allow him to do another reservation.

3.4.9. Scenario 9: User out of money for the service deploy payment

Sarah yesterday used a PowerEnJoy car, without having enough money on her credit card to cover the entire due amount. Today she needs to go to her office, but the system, when she tries to do a new reservation, doesn't allow her to complete the operation and notifies her that she must complete the previous payment, before being able again to do a new reservation.

3.4.10. Scenario 10: Maintenance team call

William left the car he was using with the battery level equal to 7%, for this reason he was charged of a 30%. So the system requests the maintenance team's intervention: they reach the car and take care of it. When they've plugged the car, they send a notification to the system.

3.5.   UML Models
3.5.1. Use case model
3.5.1.1.   Register an account

USE CASE NAME:
- Register an account

PARTICIPATING ACTORS:
- Visitor
- Data checking system

FLOW OF EVENTS:
- Entry condition: The visitor opens the page containing the registration form either from the website or the mobile application
- The visitor inserts all the required data
- The visitor uploads a scanned copy of ID card and driving license
- The visitor submits all the data to the system
- The system forwards all the inserted data to the Data checking system
- The Data checking system controls the data, obtained by the system, and gives a feedback about the correctness.
- Termination condition: with a positive feedback, the visitor has completed the registration phase and therefore the system adds the account in the database, initializing the account as "Not owing", and sends to the visitor an email containing the password for the system accessing

EXCEPTIONS:
- If the feedback from the Data checking system is negative, the visitor's registration is rejected and (s)he has to do it over. The system sends him/her an email explaining the issue occurred and a link that let him/her to do the request over just modifying the wrong fields

SPECIAL REQUIREMENTS:
- Page available on both the website and the mobile application

3.5.1.2. Log in

USE CASE NAME:

- Log in

PARTICIPATING ACTORS:

- User

FLOW OF EVENTS:

- Entry condition: a user (i.e. someone who is already registered) goes on the login page
- The user inserts and submits to the system his/her personal credentials, that is the e-mail set in the registration phase and the password given back by the system
- The system receives the credentials and checks their correctness
- Termination condition: if they're correct, the system allows the user to access the system functionalities. The user is now logged in.

EXCEPTIONS:

- The inserted email and password pair does not match any registered account. The system notifies it to the user.

SPECIAL REQUIREMENTS:

- The login interface is available both in web and mobile version
- The system checks the credential and sends a response in at most 30 seconds from the submit operation
- Page available both on the website and the mobile application

3.5.1.3.   Reserve a car

USE CASE NAME:

- Reserve a car

PARTICIPATING ACTORS:

- User
- Car
- Payment system

FLOW OF EVENTS:

- Entry condition: a user, once logged-in, opens the research page
- The user selects whether to find cars nearby
  - His/her "current position" or
  - A specified address that has to be inserted in the proper field
- The system looks for the not reserved cars having a battery level charge percentage more or equal than 20% and parked within a 500m range starting from the position wanted by the user (current one or specified address)
- The system displays to the user the result, specifying for each shown car the position, battery level and driving plate, through a map on the same page.
- The user selects the car that most fits his/her needs
- Once the car is selected, the user selects the "request reservation" option
- The system checks if the account is set as "not owning"
- The system asks the Payment system if the user has enough money (as pledge) to reserve a car and waits for a feedback
- If the account is "Not owing" AND the Payment system's feedback is positive, the system notifies the user that his reservation has been accepted, then sets the status of the requested car as "Reserved" and starts counting the reservation time for the car unlocking
- As soon as (s)he reaches the reserved car, the user requests the system to unlock the car via mobile application
- The system receives the request to unlock the car
- Termination condition: The system checks if the user is at most 100m far from the car (s)he has reserved and the reservation is still valid (i.e. the time is not over) and, if so, unlocks that reserved car.

EXCEPTIONS:

- If the account is "Owing" OR the Payment system's feedback is negative, the system communicates to the user the reason that (s)he can't reserve a car and deletes the reservation.
- If all the cars, shown by the research, get reserved before the user can make a reservation, the system shows an error message inviting the user to try again later
- If the user is more than 100 m far from the car, the system denies the user's request and invites him/her to do it again

SPECIAL REQUIREMENTS:

- The system has to perform the research and send the result to the user in no more than 120 seconds.
- Research page available both on the website and the mobile application

3.5.1.4. Close the reservation

USE CASE NAME:

- Close the reservation

PARTICIPATING ACTORS:

- User
- Car
- Payment system

FLOW OF EVENTS:

- Entry condition: The user does not unlock the car within the reservation time
- The system requests a 1€ payment for the user through the Payment system
- The system sets the car as "Not reserved" again
- Termination condition: the system notifies the user through the mobile application that the reservation has been closed

SPECIAL REQUIREMENTS:

- The user is notified through the mobile application

3.5.1.5. Drive a car

USE CASE NAME:
- Drive a car

PARTICIPATING ACTORS:
- User
- Car
- Payment system

FLOW OF EVENTS:
- Entry condition: a user has unlocked his/her reserved car
- The user enters in the vehicle and turns the engine on
- The system keeps track of the current time (at which the deploy begins) in order to charge the user accordingly
- As long as the engine is on, the system notifies to the car the current cost of the ride in order to display that to the user
- The system asks to the car the number of people inside in order to keep track on whether the user needs a 10% discount on the cost of the ride
- When the system receives the notification that the engine is switched off, it starts asking the car on whether there are no people inside the car and the doors have already been closed; when so, it locks the car
- Once the car is locked:
  - the system stores the current time to keep track of the ending of the deploy time of the service in order to charge the user accordingly
  - the system requests at the car his position and level of battery charge in order to:
    - apply a 20% discount on the cost of the ride if the level of the battery charge is more or equal than 50%
    - apply a 30% charge on the cost of the ride if the level of the battery charge is less than 20% or the position of the car is 3Km farther any power grid station
- The system, after a 2-minute delay from when the car is locked, requests the car whether it has been plugged to a power grid station in order to, if so, apply a 20% discount on the cost of the ride
- If the car has a battery charge percentage level that is less than 20% and is not plugged in, the system requests a maintenance intervention
- The system sets the car as "Not reserved"
- The system assesses the cost of the ride (considering the elapsed time between the beginning and the ending of the deploy time)
- The system requests the assessed due amount of money to the user through the Payment system
- Termination condition: The system notifies the user of the cost of the ride through the mobile application

SPECIAL REQUIREMENTS:
- The system sends the updated charge data every minute
- The user is notified with the total charge of the ride through the mobile application

3.5.1.6.   Enable the Money saving option

USE CASE NAME:
- Enable the Money Saving Option

PARTICIPATING ACTORS:
- User
- Car

FLOW OF EVENTS:
- Entry condition: the user starts the deploy and select the Money saving option through the car terminal
- The user inserts the final destination
- The car sends the information to the system
- The system computes a route that depends both on the selected destination and whether the power grid station has not less than half of the plugs available.
- Termination condition: The system drives the user to the suggested power grid station

EXCEPTIONS:
- If the system can't find a power grid station that satisfies the requests, the system notifies the user that this functionality it's not available at the moment.

SPECIAL REQUIREMENTS:
- The system should compute the route in no more than 120 seconds

3.5.1.7.   Carry out a transaction

USE CASE NAME:
- Carry out a transaction

PARTICIPATING ACTORS:
- User
- Payment system

FLOW OF EVENTS:
- Entry condition: the system requires to the payment system the amount due by a certain user
- The payment system performs a money transaction and then gives a feedback to the system notifying the status of the transaction
- Termination condition:
  - If the payment is carried out successfully, the system sets the user as "not owing"
  - If the payment hasn't been carried out successfully, the system waits an hour and then repeat the request to the system

SPECIAL REQUIREMENTS:
- In case of an unsuccessful payment, the system waits an hour before repeat all over.

3.5.1.8. Maintenance team call

USE CASE NAME:
- Maintenance team call

PARTICIPATING ACTORS:
- Maintenance team
- Car

FLOW OF EVENTS:
- Entry condition: An intervention has been called by the system
- Once there, the maintenance team draws the car up to a power grid station and plugs it on.
- Termination condition: The maintenance team notifies the end of the intervention.

3.5.2. Use case diagram
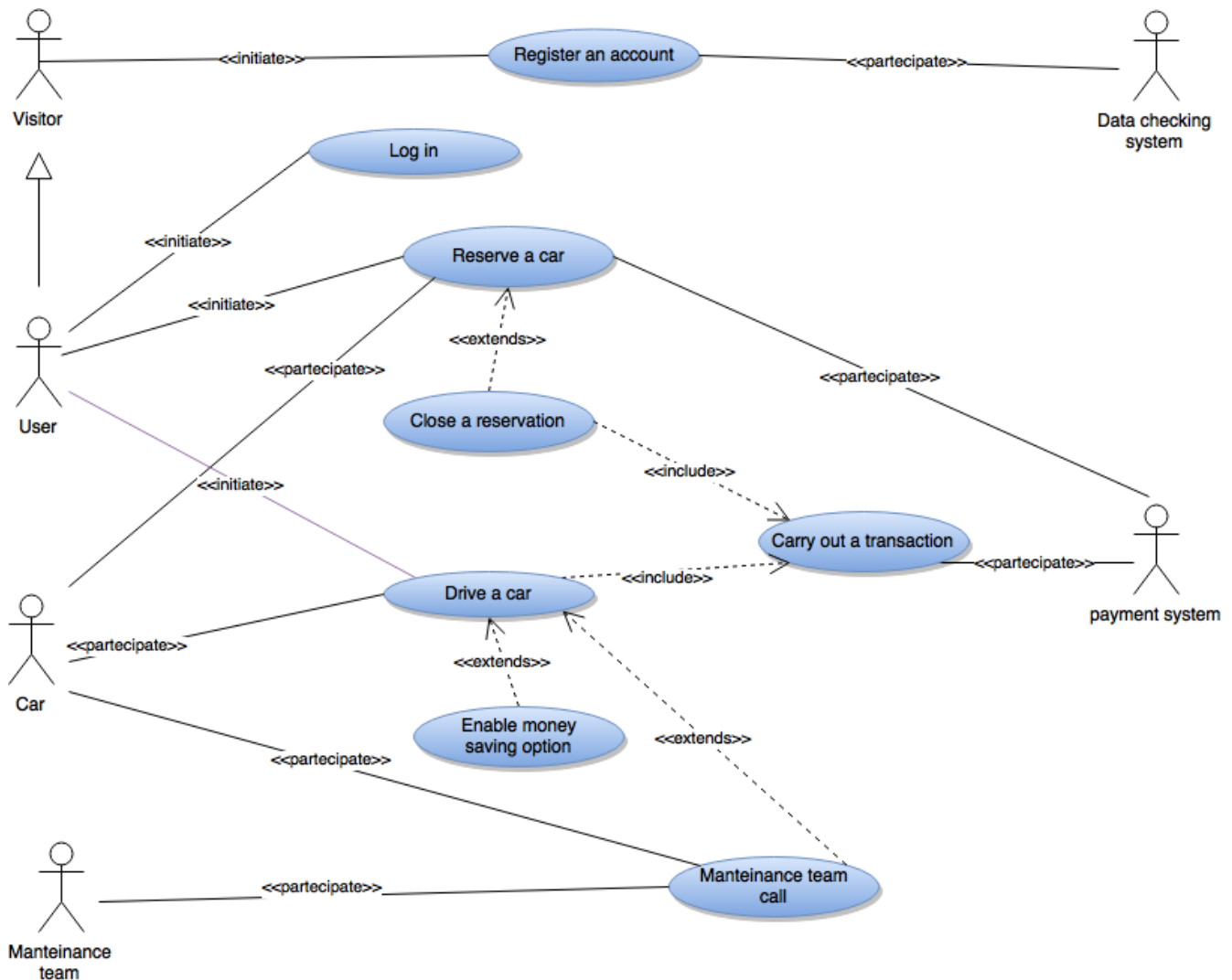
3.5.3. Class diagram
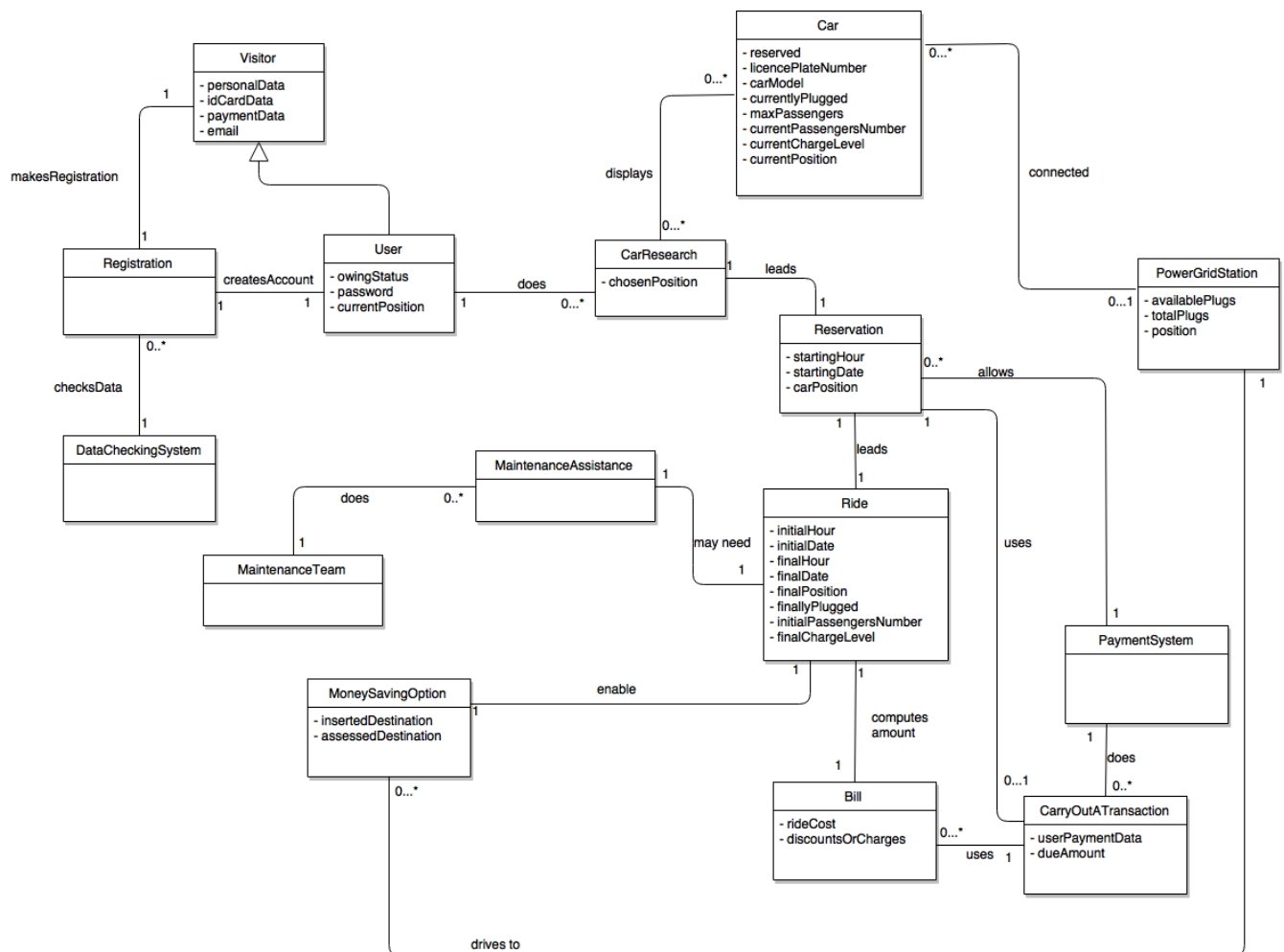
3.5.4. State charts
3.5.4.1. Car's states

Drive

user unlocks

Unlocked Reserved

User leaves the car

Reservation

Locked Reserved

Locking

Locked Reserved

batteryLevel < 20%

after 2 minutes from the locking && batteryLevel < 20%

reserve a car

Locked Not reserved Not enough battery

after 2 minutes from the locking && batteryLevel >= 20%

close reservation

batteryLevel >= 20%

Locked Not reserved Enough battery
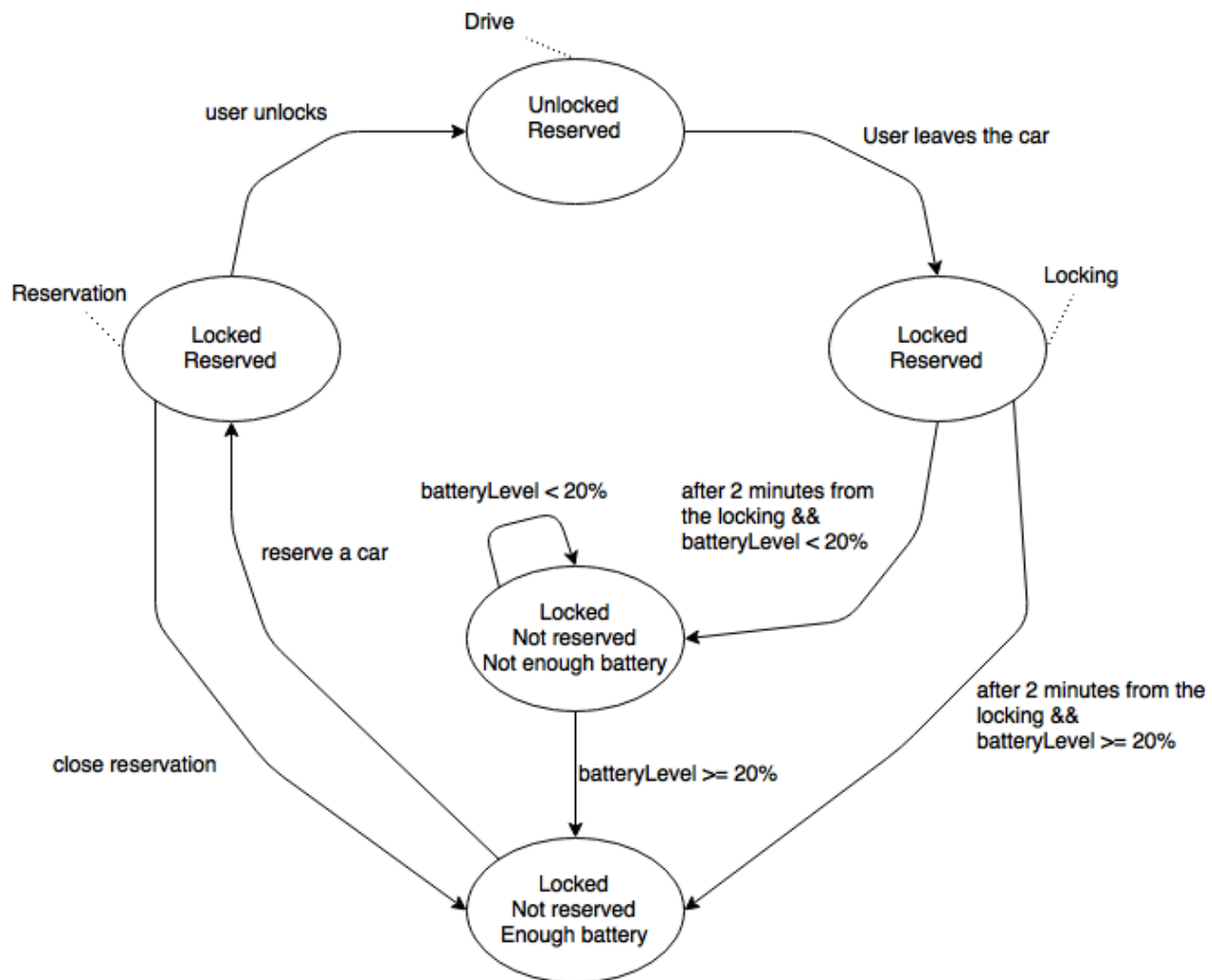
3.5.4.2. User/Visitor's phases
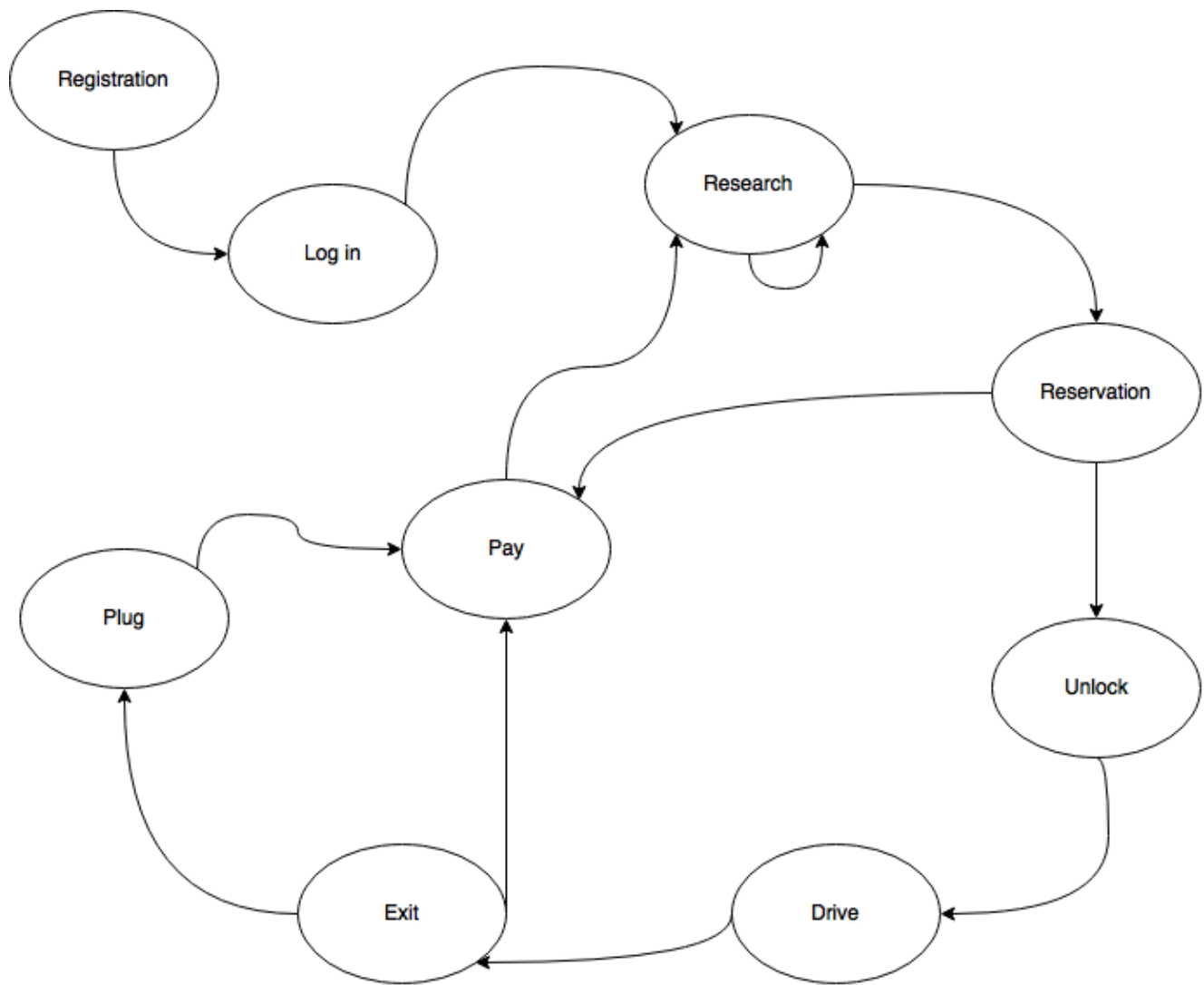
3.5.5. Activity diagrams

3.5.5.1. Registration phase
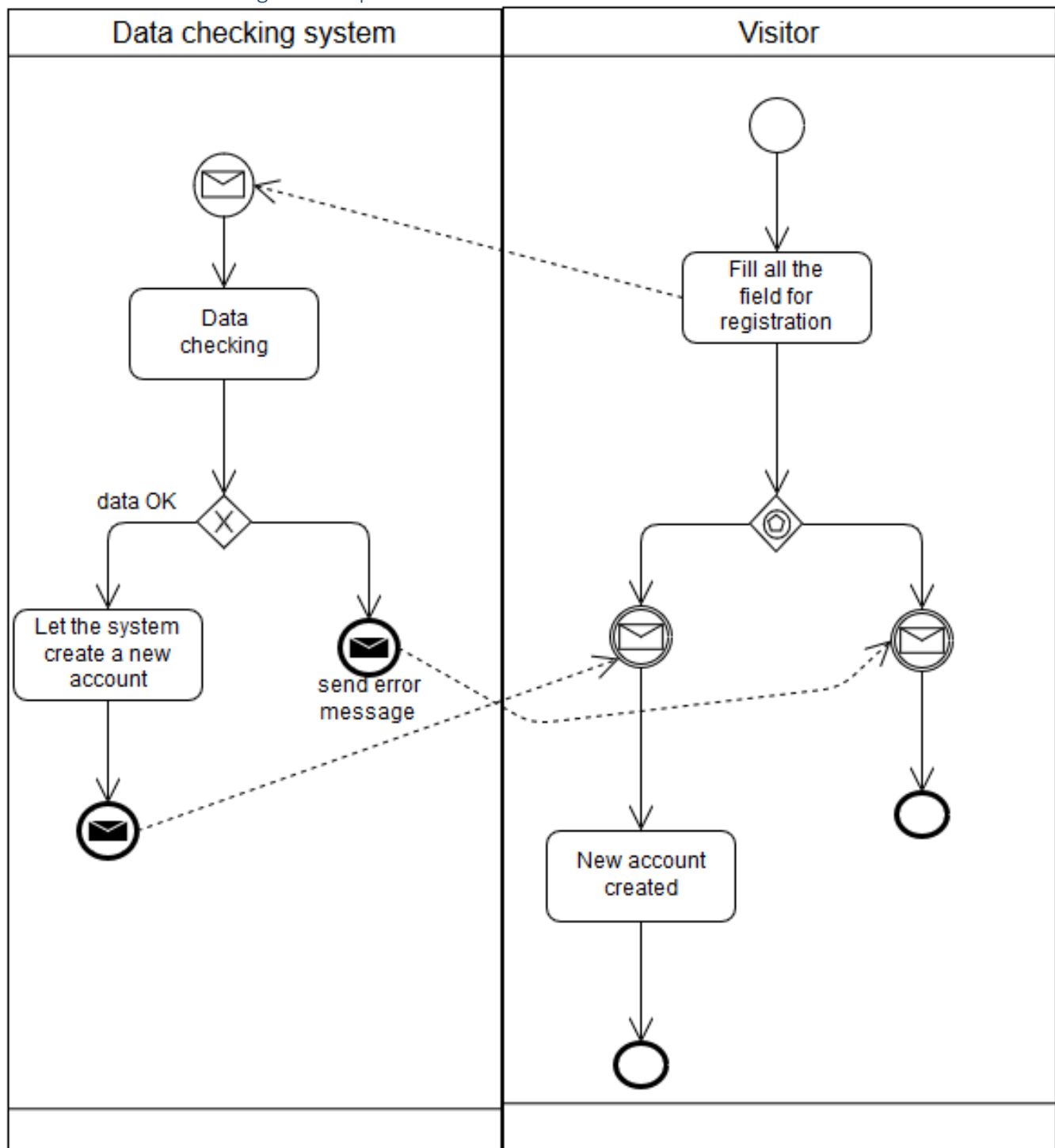
3.5.5.2.   User's session

(continues below…)

Payment system	User	Car

User column:

sending unlock message

Get on the car

turn the engine on

⊗

no money saving option | yes money saving option

⊗

drive

leave the car

⊗

Plug

⊗

Car column:

Engine on

ask for money saving option

enable money saving option

⊗

empty car

Lock

plugged | 2 min

⊗

Compute discounts and recharges

Evaluate car charge

Payment system column:

Carry out a transaction
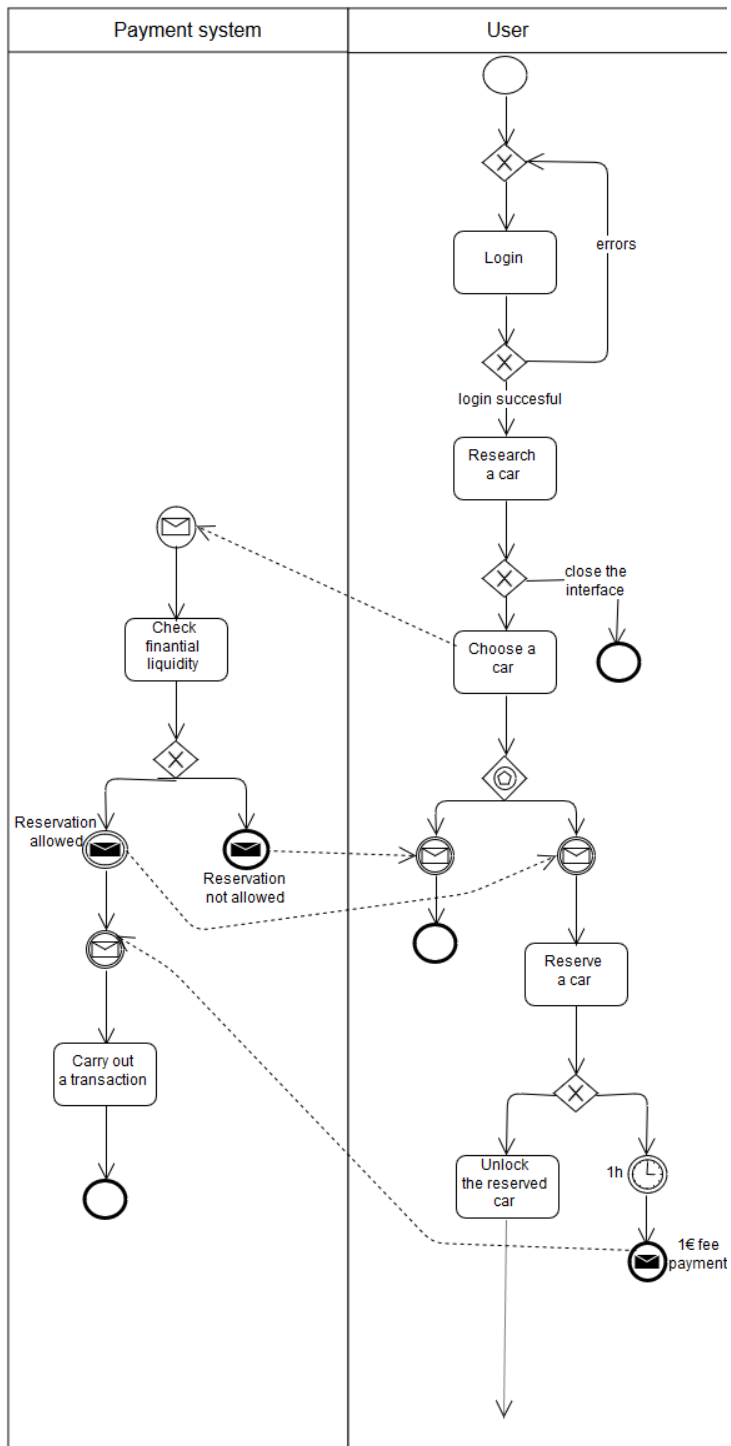
3.5.6. Sequence diagrams

3.5.6.1. Registration phase

sd Registration phase
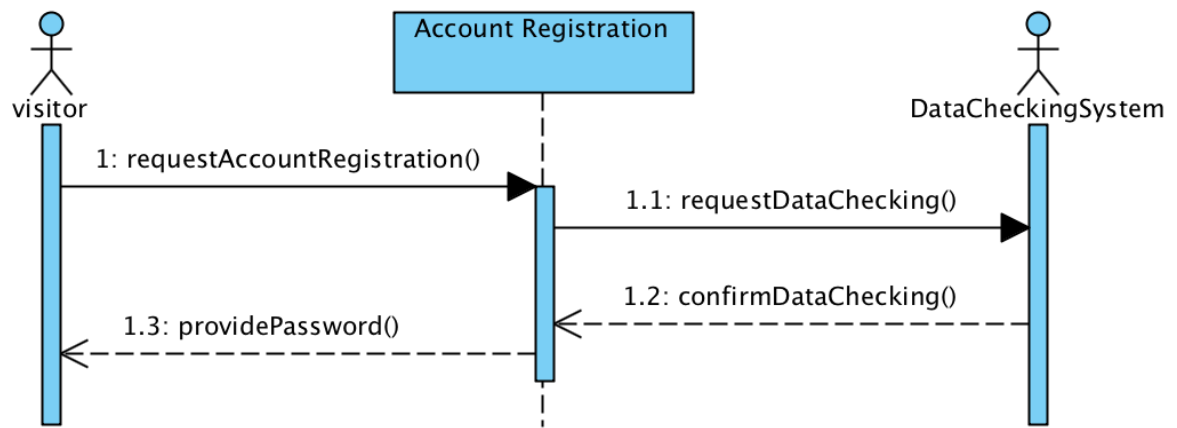
3.5.6.2. Log in

sd Login
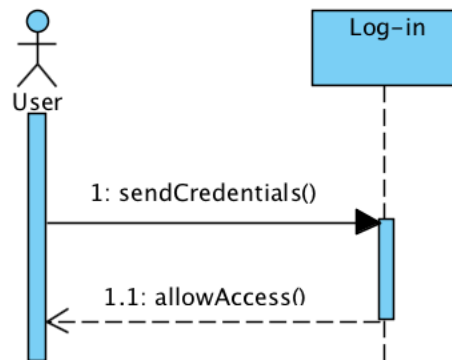
3.5.6.3.   Reserve a car
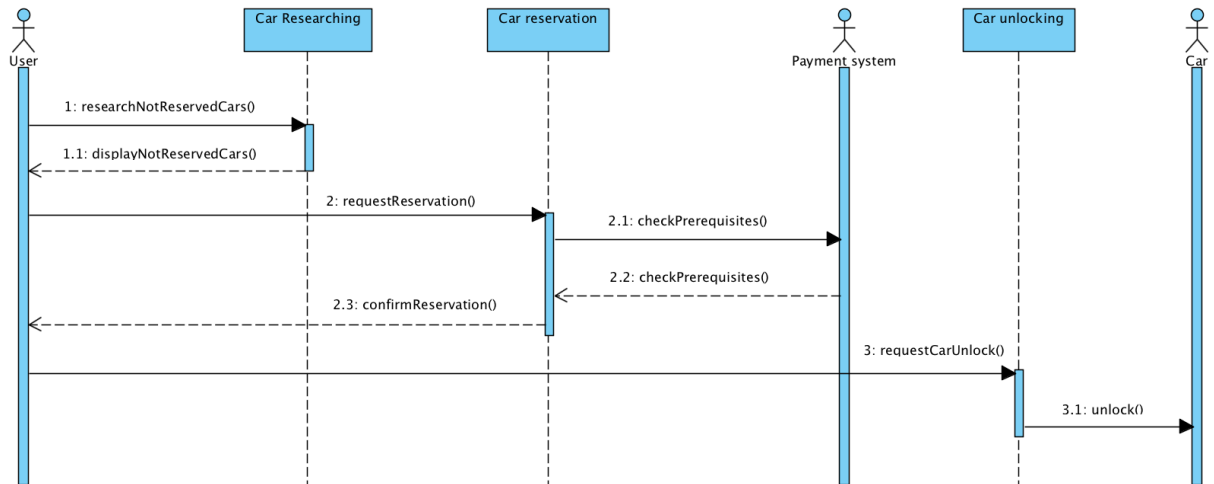
3.5.6.4.   Close a reservation

41

3.5.6.5.   Drive a car

3.6.   Alloy Models

These alloy models files (.als) are fully available on our gitHub repository. We've decided to split the model into two parts: the first describes the model of "Registration and Login" part, and the second represents the whole "Reservation" process, from the research until the end of the reservation session.

3.6.1. Registration and login model
3.6.1.1.   Datatype Definition and abstract entity.

This is the definition of all useful data types and the abstract entity user.

```
open util/boolean

//-----------------DATATYPE DEFINITION----------------//
one sig Email{}
one sig Integer{}
one sig Strings{}
abstract sig Date {
    day: one Integer,
    month: one Integer,
    year: one Integer
}
abstract sig User {}
sig DateBirth extends Date {}

sig PersonalData {
firstName: one Strings,
lastName: one Strings,
sex: one Strings,
emailAddress: one Email,
dateOfBirth: one DateBirth,
birthPlace: one Strings,
actualAddress: one Strings,
city: one Strings,
phoneNumber: one Strings,
drivingLicenceNumber: one Strings,
taxIDCode: one Strings
}

sig PaymentCardData {
paymentCardNumber: one Strings,
paymentCardSecurityCode: one Strings
}


sig Password {}
```

3.6.1.2.   Signatures

This is the definition of all the signatures used in the model.

```
//---------------SIGNATURES---------------------------//
sig Visitor extends User{}

sig RegisteredUser extends User {
personalData: one PersonalData,
paymentData: one PaymentCardData,
newPassword: one Password,
loggedIn: one Bool,
disabled: one Bool
}

one sig PowerEnjoyUserDB {
    registeredUsers: set RegisteredUser,
    loggedInUsers: set RegisteredUser
}

sig Registration {
registrates: one RegisteredUser,
regId: one Integer,
dataChecker: one DataCheckingSystem
}

one sig DataCheckingSystem {
    correctData: one Bool
}
```

3.6.1.3. Facts

Here the facts definition.

```
//---------------FACTS--------------------------------//
fact dateNonempty {
    all d: Date | (#d.year=1) and (#d.month=1) and (#d.day=1)
}

fact dataCheckNonempty {
    all d: DataCheckingSystem | (#d.correctData=1)
}
fact userNonempty {
    all u: User | (#u.loggedIn=1) and (#u.personalData=1) and (#u.newPassword=1) and (#u.paymentData=1)
}
fact registrationNonempty {
    all r: Registration | (#r.registrates=1) and (#r.dataChecker=1)
}
fact personalDataNonempty {
    all p: PersonalData | (#p.firstName=1) and (#p.lastName=1) and (#p.sex=1) and (#p.emailAddress=1) and
                          (#p.dateOfBirth=1) and (#p.birthPlace=1) and (#p.actualAddress=1) and (#p.city=1) and
                          (#p.phoneNumber=1) and (#p.drivingLicenceNumber=1) and (#p.taxIDCode=1)
}
fact paymentNonempty {
    all p: PaymentCardData | (#p.paymentCardNumber=1) and (#p.paymentCardSecurityCode=1)
}

fact noDuplicateUser {
    no disj u1, u2: RegisteredUser | u1.personalData.emailAddress = u2.personalData.emailAddress
}

fact noDuplicateRegistration {
    no disj r1, r2: Registration | r1.regId=r2.regId
}

fact registration {
    all u: RegisteredUser | one r: Registration | r.registrates = u
}

fact accountDisabled {
    all u: RegisteredUser | u.disabled= True => u not in PowerEnjoyUserDB.registeredUsers
}


fact userMemorized {
    all u: RegisteredUser | u in PowerEnjoyUserDB.registeredUsers
}

fact registrationDoneIfDataAreCorrect {
    all r: Registration| r.dataChecker.correctData = True
}

fact login {
    all u: RegisteredUser | u.loggedIn=True
}

fact logoutIssue {
    all u: RegisteredUser | u not in PowerEnjoyUserDB.loggedInUsers => u.loggedIn= False
}
```

3.6.1.4. Predicates

The definition of all predicates.

```
//--------------PREDICATES-------------------------//
pred show {}

pred addNewUser(u: RegisteredUser, pdb1, pdb2: PowerEnjoyUserDB) {
    u not in pdb1.registeredUsers implies pdb2.registeredUsers = pdb1.registeredUsers + u and u.disabled= False
}

pred deleteUser (u: RegisteredUser, pdb1, pdb2: PowerEnjoyUserDB) {
    u in pdb1.registeredUsers and u.disabled = True implies pdb2.registeredUsers= pdb1.registeredUsers- u
}

pred login (u: RegisteredUser, p1, p2: PowerEnjoyUserDB) {
    u not in p1.loggedInUsers and u.loggedIn= True implies p2.loggedInUsers = p1.loggedInUsers + u
}

pred logout (u: RegisteredUser, p1, p2: PowerEnjoyUserDB) {
    u in p1.loggedInUsers and u.loggedIn= False implies p2.loggedInUsers = p1.loggedInUsers - u
}

run show for 4
run addNewUser for 2
run deleteUser for 2
run login for 2
run logout for 2
```

3.6.1.5.   Assertions

Here are all the assertions.

```
//-------------ASSERTIONS-------------------------//


assert addNewUser {
    all u: RegisteredUser, pdb1, pdb2: PowerEnjoyUserDB | (u not in pdb1.registeredUsers)
    and addNewUser[u, pdb1, pdb2] implies (u in pdb2.registeredUsers) and u.disabled=False
}

assert deleteUser {
    all u: RegisteredUser, pdb1, pdb2: PowerEnjoyUserDB | (u in pdb1.registeredUsers) and u.disabled= True
    and deleteUser[u, pdb1, pdb2] implies (u not in pdb2.registeredUsers)
}

assert login {
    all u: RegisteredUser, pdb1, pdb2: PowerEnjoyUserDB | (u not in pdb1.loggedInUsers)
    and u.loggedIn= True and login[u, pdb1, pdb2] implies (u in pdb2.loggedInUsers)
}

assert logout {
    all u: RegisteredUser, pdb1, pdb2: PowerEnjoyUserDB | (u in pdb1.loggedInUsers)
    and u.loggedIn= False and logout[u, pdb1, pdb2] implies (u not in pdb2.loggedInUsers)
}

check addNewUser
check deleteUser
check login
check logout
```

This is the result given by the Alloy Analyzer about the consistency of this part of the model.

```
9 commands were executed. The results are:
   #1: Instance found. show is consistent.
   #2: Instance found. addNewUser is consistent.
   #3: Instance found. deleteUser is consistent.
   #4: Instance found. login is consistent.
   #5: Instance found. logout is consistent.
   #6: No counterexample found. addNewUser may be valid.
   #7: No counterexample found. deleteUser may be valid.
   #8: No counterexample found. login may be valid.
   #9: No counterexample found. logout may be valid.
```

3.6.1.7. Generated world

This is the generated world given by Alloy Analyzer on the predicate show run for 4 instances.
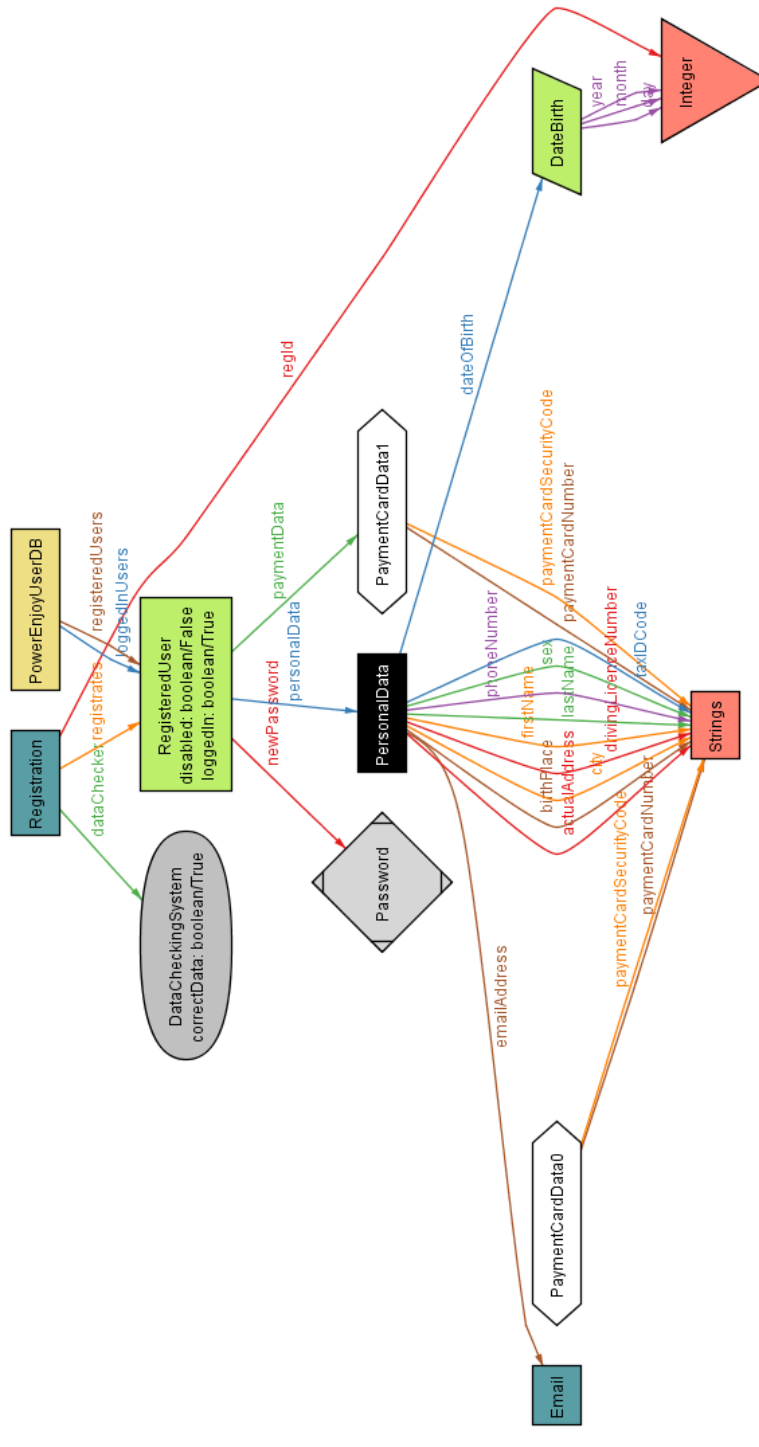
3.6.2. Reservation model

3.6.2.1. Abstract signatures

These are all the abstract signatures used in the reservation model.

```
open util/boolean

abstract sig BatteryLevel {}
    one sig LowBattery extends BatteryLevel {}        // 0%   < BatteryLevel   < 20%
    one sig MediumBattery extends BatteryLevel {}      // 20% <= BatteryLevel   < 50%
    one sig HighBattery extends BatteryLevel {}        // 50% <= BatteryLevel   < 100

abstract sig DistanceCarToPowerGridStation {}
    one sig Close extends DistanceCarToPowerGridStation {}  // Car 50m near a Power grid station
    one sig Medium extends DistanceCarToPowerGridStation {} // Car from 50m to 3Km far away a Power grid station
    one sig Far extends DistanceCarToPowerGridStation {}    // Car 3Km far away a Power grid station

abstract sig DistanceUserCarForUnlocking {}
    one sig InUnlockingRange extends DistanceUserCarForUnlocking {}  // User within 100m from the reserved car
    one sig OutUnlockingRange extends DistanceUserCarForUnlocking {} // User not within 100m the reserved car

abstract sig DistaceCarToResearchPosition {}
    one sig InResearchRange extends DistaceCarToResearchPosition {}  // Car within 500m from the research position
    one sig OutResearchRange extends DistaceCarToResearchPosition {} // Car not within 500m from the research position

abstract sig SessionStatus {}
    one sig Reservation extends SessionStatus {}
    one sig ReservationClosed extends SessionStatus {}
    one sig UnlockingRange extends SessionStatus {}
    one sig Driving extends SessionStatus {}
    one sig CostAssessment extends SessionStatus {}
```

3.6.2.2.   Signatures

These are all the signatures with their conditions on the parameters.

```
sig Person{ }
sig Visitor extends Person{ }

sig User extends Visitor {
    owing: one Bool, // If the user owes the system or not
    loggedIn: one Bool,
    pledgeOk: one Bool,
    session: lone ActiveSession
} {
    this not in session.recordedPassengers

    // A: Only users that have paid all the previous sessions can reserve a car
    session != none implies owing = False

    // A: Only logged in users can reserve a car
    session != none implies loggedIn = True

    // A: An user cannot be a passenger of itself
    session != none && this not in session.recordedPassengers
}

sig PowerGridStation { }

sig Car {
    pluggedAt: lone PowerGridStation,
    batteryLevel: one BatteryLevel,
    locked: one Bool,
    distanceNextPGS: one DistanceCarToPowerGridStation
} {
    #pluggedAt != 0 implies distanceNextPGS = Close
}

sig PossiblePGS {
    pgs: one PowerGridStation,
    distanceToDestination: one Int,
    pluggedCars: one Int
} {
    pluggedCars = pluggedCars[pgs]
    distanceToDestination >= 0
}


//Note: the model does not comprehend the distance of the powerGridStation to the destination
// There would be picked out the nearest one
sig MoneySavingOption {
    ppgs: set PossiblePGS, // possible ones
    cpgs: lone PossiblePGS // Chosen one
} {
    #ppgs !=0 implies #cpgs != 0
    let p = PossiblePGS | p in ppgs iff p.pluggedCars < 2
    all k: PossiblePGS | k in ppgs && cpgs.distanceToDestination <= k.distanceToDestination
}
```

```
sig FinalCost {
    fee: one Bool,              // 1€ fee for having got to the car too late
    discount1: one Bool,       // 10% discount with at least 2 passenger
    discount2: one Bool,        // 20% discount ending the ride with Battery >= 50%
    discount3: one Bool,        // 20% discount plugging the car to a power grid station ending the ride
    charge1: one Bool,         // 30% charge ending the ride with Battery < 20% or 3Km far away any power grid station
}

// Note: the pay session prevents the user reserving other cars since the user can have just one session active
sig ActiveSession {
    status: one SessionStatus,
    car: one Car,
    moneySavingOption: lone MoneySavingOption,
    duc: one DistanceUserCarForUnlocking,
    dcp: one DistanceCarToPowerGridStation, // Distance to the next one
    overPluggingTime: one Bool,
    overReservationTime: one Bool,
    finalCost: lone FinalCost,
    recordedPassengers: set User,
    onBoardPassengers: set User,
    driverOnBoard: one Bool,
} {

    //A: At most 4 passengers
    #recordedPassengers < 4

    // User far away from the car => cannot be on board
    duc = OutUnlockingRange implies driverOnBoard = False
    // User on board => user close to the car
    driverOnBoard = True implies duc = InUnlockingRange

    #recordedPassengers = 0 implies #onBoardPassengers = 0
    #onBoardPassengers != 0 implies onBoardPassengers = recordedPassengers
    status = Driving iff onBoardPassengers = recordedPassengers

    status = Reservation iff (
                car.batteryLevel != LowBattery &&
                #moneySavingOption = 0 &&
                car.locked = True &&
                duc = OutUnlockingRange &&
                //dcp = NOT RELEVANT &&
                overPluggingTime = False &&
                overReservationTime = False &&
                #finalCost = 0 &&
                #recordedPassengers = 0 &&
                #onBoardPassengers = 0 &&
                driverOnBoard = False
            )
```

```
status = ReservationClosed iff (
        car.batteryLevel != LowBattery &&
        #moneySavingOption = 0 &&
        car.locked = True &&
        duc = OutUnlockingRange &&
        //dcp = NOT RELEVANT &&
        overPluggingTime = False &&
        overReservationTime = True &&
        #finalCost != 0 &&
        #recordedPassengers = 0 &&
        #onBoardPassengers = 0 &&
        driverOnBoard = False
    )

status = UnlockingRange iff (
        car.batteryLevel != LowBattery &&
        #moneySavingOption = 0 &&
        car.locked = True &&
        duc = InUnlockingRange &&
        //dcp = NOT RELEVANT &&
        overPluggingTime = False &&
        overReservationTime = False &&
        #finalCost = 0 &&
        #recordedPassengers = 0 &&
        #onBoardPassengers = 0 &&
        driverOnBoard = False
    )

status = Driving iff (
        //car.batteryLevel = NOT RELEVANT &&
        //#moneySavingOption = NOT RELEVANT &&
        car.locked = False &&
        duc = InUnlockingRange &&
        //dcp = NOT RELEVANT &&
        overPluggingTime = False &&
        overReservationTime = False &&
        #finalCost = 0 &&
        //#recordedPassengers = NOT RELEVANT &&
        //#onBoardPassengers = NOT RELEVANT &&
        driverOnBoard = True
    )

status = CostAssessment iff (
        //car.batteryLevel = NOT RELEVANT &&
        #moneySavingOption = 0 &&
        car.locked = True &&
        //duc = NOT RELEVANT &&
        //dcp = NOT RELEVANT &&
        overPluggingTime = True &&
        overReservationTime = False &&
        #finalCost != 0 &&
        //#recordedPassengers = NOT RELEVANT &&
        #onBoardPassengers = 0 &&
        driverOnBoard = False
    )
```

```
// A: A car cannot be plugged when in movement
status = Driving implies ( no pgs: PowerGridStation | car.pluggedAt = pgs )

    finalCost.fee = True iff status = ReservationClosed
    finalCost.discount1 = True iff #recordedPassengers >= 2
    finalCost.discount2 = True iff car.batteryLevel = HighBattery
    finalCost.discount3 = True iff ( #car.pluggedAt != 0 )
    finalCost.charge1 = True iff ( car.batteryLevel = LowBattery or car.distanceNextPGS = Far )


}
```

3.6.2.3. Facts and functions

Here all the defined facts and a function used in the model.

```
fun pluggedCars[pgs: PowerGridStation] : one Int {
    #pluggedAt :> pgs
}


// A: Any plug station has exactly 4 electric outlets (plugs) available
fact plugsLimit {
    all pgs: PowerGridStation | pluggedCars[pgs] <= 4
}

// A: Any session needs an user (person) associated
fact noSessionsWithNoUser {
    all s: ActiveSession | one u: User | u.session = s
}

// A: Any Money saving option needs a session associated
fact noMSOWithNoSession {
    all mso: MoneySavingOption | one s: ActiveSession | s.moneySavingOption = mso
}

// A: Any PossiblePGS needs a MoneySavingOption associated
fact noPossiblePGSWithNoMoneySavingOption {
    all p: PossiblePGS | one mso: MoneySavingOption | p in mso.ppgs
}

// It comes up from the model sempification of the distance PGS-Car
fact PossiblePGLessEqPGS1 {
    no mso: MoneySavingOption | #mso.ppgs > #PowerGridStation
}
```

```
// It comes up from the model sempification of the distance PGS-Car
fact PossiblePGLessEqPGS2 {
    no disjoint ppgs1, ppgs2: PossiblePGS, mso: MoneySavingOption | ( (ppgs1 in mso.ppgs)
    and (ppgs2 in mso.ppgs) ) implies ( ppgs1.pgs = ppgs2.pgs )
}

// A: No people on board in different places at the same time
fact noOnBoardDifferentPlacesSameTime {
    all u: User | ( ( u.session.driverOnBoard = True ) implies u not in ActiveSession.onBoardPassengers )
    no disjoint s1, s2: ActiveSession | s1.onBoardPassengers & s2.onBoardPassengers != none
}

// A: one car only for each session
fact oneCarForEachSession {
    no disjoint s1, s2: ActiveSession | #s1.car != 0 && s1.car = s2.car
}

// Any car without a session should be locked
fact carWithNoSessionLocked {
    no c: Car | #ActiveSession.car = 0 && c.locked = False
}

//A: The final cost, if exists, should be linked at a session
fact noFinalCostWithoutSession {
    no f: FinalCost | f not in ActiveSession.finalCost
}
```

3.6.2.4.   Predicates

Here the predicates.

```
pred reserveACar[u: User] { // G|u3
    u.owing = False and u.loggedIn = True and u.pledgeOk = True and
    some c: Car | ActiveSession.car != c
}
run reserveACar for 5

pred unlockTheReservedCar[u: User] { // G|u05
    u.session.overReservationTime = False
    and u.session.duc = InUnlockingRange iff u.session.status = UnlockingRange
}
run unlockTheReservedCar for 5

pred show() {}
run show
```

3.6.2.5. Assertions

These are all the assertions used to verify the model.

```
assert notReservableIfBatteryLessThan20 {
    all s: ActiveSession | s.status = Reservation implies s.car.batteryLevel != LowBattery
}
check notReservableIfBatteryLessThan20

assert closeTheReservationIfLate { // G|u04
    all s: ActiveSession | s.overReservationTime = True implies s.finalCost.fee = True
}
check closeTheReservationIfLate

assert discount10p { // G|u06
    all u: User | u.session.status = CostAssessment && #u.session.recordedPassengers >= 2
    iff u.session.finalCost.discount1 = True
}
check discount10p

assert discount20p { // G|u07
    all u: User | u.session.status = CostAssessment && u.session.car.batteryLevel = HighBattery
    iff u.session.finalCost.discount2 = True
}
check discount20p

assert discount30p { // G|u08
    all u: User | u.session.status = CostAssessment && #u.session.car.pluggedAt > 0
    iff u.session.finalCost.discount3 = True
}
check discount30p

assert charge30p { // G|u09
    all u: User | u.session.status = CostAssessment && u.session.car.batteryLevel = LowBattery
    or u.session.car.distanceNextPGS = Far implies u.session.finalCost.charge1 = True
}
check charge30p

assert activeMSO { // G|u11
    all s: ActiveSession | #s.moneySavingOption > 0 && #s.moneySavingOption.ppgs > 0
    iff #s.moneySavingOption.cpgs.pgs > 0
}
check activeMSO
```

3.6.2.6. Result

This is the result given by the Alloy Analyzer for the reservation model.

```
10 commands were executed. The results are:
 #1: Instance found. reserveACar is consistent.
 #2: Instance found. unlockTheReservedCar is consistent.
 #3: Instance found. show is consistent.
 #4: No counterexample found. notReservableIfBatteryLessThan20 may be valid.
 #5: No counterexample found. closeTheReservationIfLate may be valid.
 #6: No counterexample found. discount10p may be valid.
 #7: No counterexample found. discount20p may be valid.
 #8: No counterexample found. discount30p may be valid.
 #9: No counterexample found. charge30p may be valid.
 #10: No counterexample found. activeMSO may be valid.
```

3.6.2.7. Generated World

This is the model generated by the analyzer for the predicate show() on 4 instances.
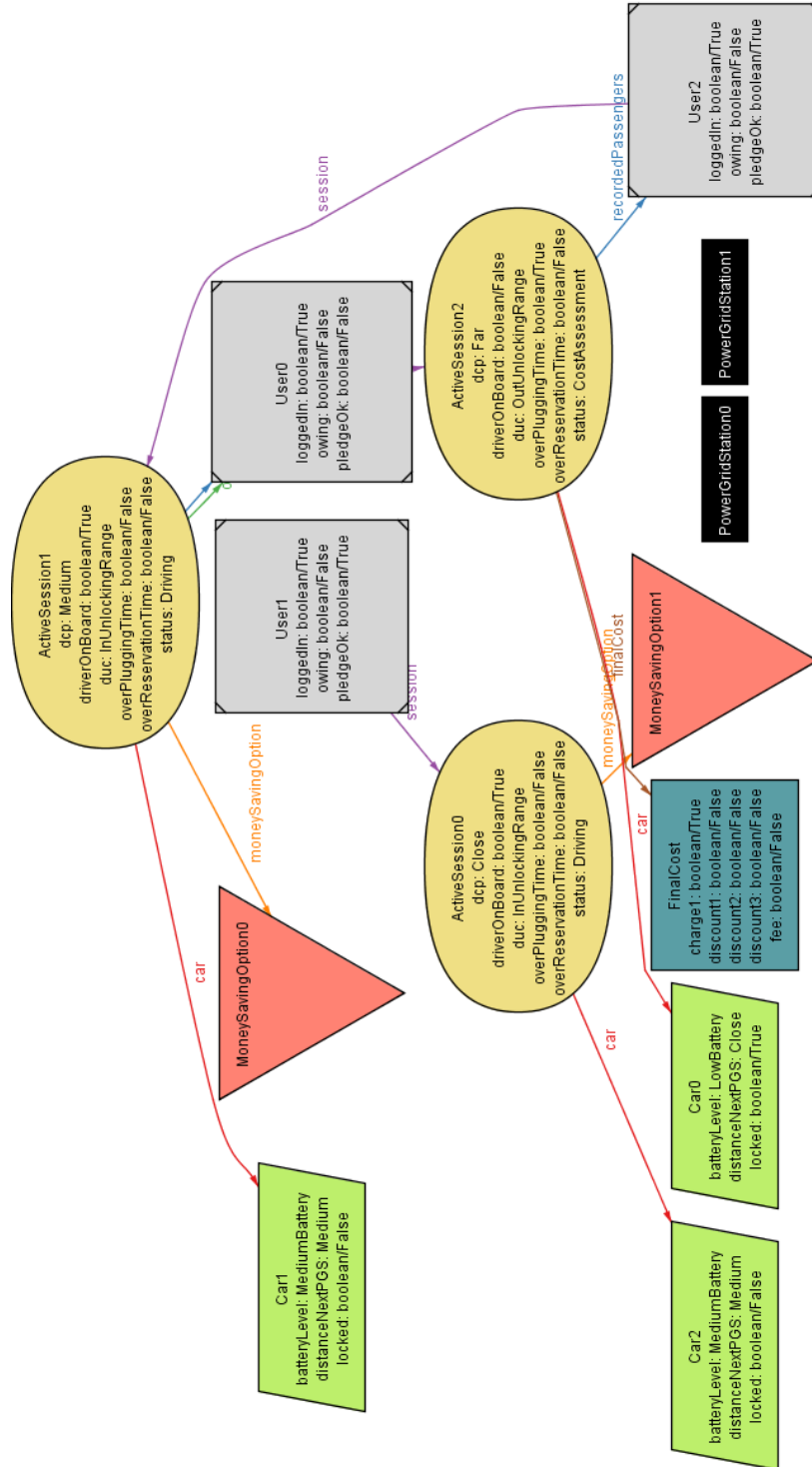
4. Tools used:

- **<u>Microsoft word</u>**: for document redaction
- **<u>Alloy analyzer</u>**: for model consistency analysis
- **<u>Visual Paradigm</u>**: for the sequence diagram realization
- **<u>Draw.io</u>**: for use case diagram, BPMN, statechart and class diagram
- **<u>Gimp</u>**: for some image modification
- **<u>Pencil</u>**: for mockup realization
- **<u>Microsoft Excel</u>**: for hours counting and gantt diagram

5. Hours of work:

Date	Hours	Person	Task
28/10/16	1	Marco	Goals
	1	Paola	Goals
	1	Marco	Requirements, goals and domain assumptions' formalization
29/10/16	2,5	Marco	Requirements, goals and domain assumptions' formalization
30/10/16	2,5	Giulia	Requirements, actors identification, formalized description (own version)
31/10/16	2,5	Giulia	Mockup experiments
31/10/16	1	Marco	Requirements, goals, definitions, constraints (own version)
01/11/16	2,5	Marco	Requirements, goals, definitions, constraints (own version)
02/11/16	2	Giulia	Mockups
02/11/12	2	Marco	Requirements, goals, definitions, constraints (ownversion)
03/11/16	1	Paola	Requirements
	1	Giulia	Mockups
	1	Marco	Requirements
	1	Paola	Scenarios
	1	Giulia	Scenarios
03/11/16	2	Marco	Requirements, goals, definitions, constraints (own version)
	1	Paola	Scenarios
04/11/16	0,5	Marco	Requirements consistency check
	0,5	Paola	Requirements consistency check
	2	Marco	Scenarios, requirements
	2	Paola	Scenarios, requirements
	2	Giulia	Mockups
	1	Marco	Fixes on requirements, assumptions, goals and definitions
	1	Paola	Fixes on requirements, assumptions, goals and definitions
05/11/16	0,5	Marco	Scenarios
	2	Marco	Use cases + DELETING the "valid/not valid" option in the account
	2	Paola	Use cases + DELETING the "valid/not valid" option in the account
	2,5	Paola	Use cases
	2	Paola	Completed use cases, need to be checked
06/11/16	1	Giulia	General review and mockups
	2	Paola	Use cases and fixes on rasd and scenarios
	3	Marco	Use cases/Scenarios
	3,5	Marco	Use cases and requirement consistency
	2	Giulia	Use case diagram
07/11/16	2,5	Paola	Fixes on use cases + first attempt of class diagram
	1	Paola	Final version of the use case
	2	Giulia	UseCase & class
	2	Marco	UseCaseModel/UseCaseDiagram + several fixing
08/11/16	2	Giulia	Activity & Statechart
	2,5	Marco	Several fixing + Class Diagram/Sequence diagram
	1	Paola	Several fixing + Class Diagram/Sequence diagram
	2,5	Paola	Sequence diagrama and state chart
09/11/16	1	Giulia	Brief meeting
	1	Marco	Brief meeting
	1	Paola	Brief meeting
	2	Giulia	activity diagrams formaization
	2,5	Paola	Use case model
	3,5	Marco	Diagrams correction (with Paola) + RASD review and addition (not compl yet)
10/11/16	2	Giulia	Diagrams & mockup
	3,5	Giulia	Diagrams & mockup (hope finished)
	3	Marco	Requirements and UseCaseModel checking
	1	Marco	Diagrams correctness checking and corrections. Consistency checking
	2,5	Paola	Fixes + activity diagram + alloy
	3	Paola	Alloy + fixes
11/11/16	3,5	Giulia	last fixes+alloy
	2,5	Giulia	alloy
	2,5	Marco	Alloy
	2	Paola	Fixes
	1	Marco	Alloy
12/11/16	5,5	Giulia	Alloy
	6,5	Paola	RASD consistency's check + everything's revision
	5,5	Marco	Alloy
13/11/16	3,5	Giulia	Alloy
	2	Giulia	Diagrams last revision and document impagination
	6,5	Paola	Revision and fixes and RASD assembly
	5,5	Marco	Alloy
07/12/16	2	Giulia	Added the Alloy code into the RASD Document too (previously developed)