Requirements Analysis and Specifications Document- PowerEnJoy

POLITECNICO
MILANO 1863

Version 2.0

- Bagna Francesco Matteo (mat. 878556)
- Barletta Carmen (mat. 877129)

Commit date: 8/12/2016

Summary

1: Introduction

1.1: Purpose

The RASD document’s purpose is to elicitate the requirements, the goals and the general functionality of the assigned project, PowerEnJoy, for the Software Engineering course.

The intended audience is the students of the course and those who are interested in understanding how to manage a project for managing electrical car-sharing services.

1.2: Scope

We have to design and implement PowerEnJoy, an electric car-sharing system, based on a web page and on a smartphone application.
The system allows users to register themselves to it, because the payment must be made with credit cards. The app also lets the users indicate a location from which to search for available cars, or to use their GPS (if they have one) to take their location. Lastly, the system lets the user open a car if they have reserved it.
The system has also some functionalities for the company operators, that can accept or decline a request of recharging a car (which is automatically sent by the system in certain cases).
The system also has an administrator, which can analyze and access to all the information on the database (excluding the user’s payment info).

The system has other functionalities, like giving discounts for users that behave correctly or charging more money users that behave incorrectly.

Our system will be supported by the GEB (Green e-Box) system in the cars, that can detect all the interesting parameters (speed, battery charge, GPS location) of the car, and furthermore give commands to the car (like closing the car door if it was left open, or don’t let the engine start up if the car is charging, or notifying the main system of the ride parameters). These parameters are showed by the car app on an interface always visible to the user.

The product of the project will have these following general objectives:

- Sensitize people in using environmental-friendly means of transport.
- Make people gain a correct behavior towards people who live in their same environment.
- Offer a reliable and widely distributed service of car-sharing.

The system will have the following limitations, that will probably be developed in future versions:

- **Temporary parking**
  It will not be possible to temporary park the car and then continue the ride.
- **Database of the rides**
  Only the last ride of the user will be saved, while the others will be cancelled.
- **Reset password and notify the user**
  The system doesn't let the user reset his password if he has lost or forgotten it.
- **Feedback**
  The system does not let the user send feedbacks about the rides.

1.3: Goals

- [G1] Identify a user in a unique way.
- [G2] For every user easy and fast to find available cars in the proximity
- [G3] Allow users to reserve a single car for up to one hour in advance to the rental
- [G4] Penalize users who reserve a car without taking it in the expected time.
- [G5] Allow user to open, enter and use the car.
- [G6] Allow user to be constantly aware of the cost of his ride.
- [G7] Make easy, for every user, to geolocate all the parking safe areas and all the available special safe areas.
- [G8] Incentivize the virtuous behavior of the users
- [G9] Provide a way to use the cars in a continuative way
- [G10] Maintain an equal distribution of the cars among the areas

1.4: Acronyms, abbreviations and definitions

- User: The person that uses the system to find, reserve and drive a car. He must be registered to the system before he can do any kind of operations.
- Blocked User: A user that did not pay completely for his last ride and is therefore obliged to pay before he can do any more reservations.
- Operator: Employee of the PowerEnJoy company can accept or decline charging requests from cars.
- Car (Electric): Means of transport that needs a valid driving license to be driven. Electric cars only use electricity as fuel, instead of diesel or other polluting fuels. It has a rechargeable battery that can be recharged in special safe areas.
- Safe areas: Areas in which the car can be parked without receiving a fine. It is a predefined list of parking areas that the company can use to park the cars.
- Special safe areas: Safe areas in which there are a certain number of power grid stations, where the electric cars can be recharged. Users receive a discount on their last ride if they connect a car to a power grid.
- Reservation: It's a request, made by a User, to take a car and use it. A reserved car cannot be taken by another user before the reservation is terminated. A reservation must be fulfilled (the engine of the car must be turned on) up to one hour after the reservation was made. If this is not done, the car is freed (can be taken by another user) and the user has to pay 1€ as a fee.
- Ride: Act of using the car, it begins when the user turns on the car's engine and terminates when the user turns it off. A fee per minute is paid during the ride.
- GPS: Global positioning system, used by cars to tell their actual position to the system and by users to start a search for cars from their actual position.

- Position: where the car/user is, calculated by using GPS systems. Identificated by a Latitude and a Longitude.
- Penalty: an incrementation on the charge of a ride for a user that didn't have a good behavior.
- Payment Info: Managed by another company, they're a series of informations about the user and his credit card that are used to make him pay for the rides he does.
- State of the car: A set of parameters of the car, like its availability(available, charging, outOfService, reserved), the status of the engine (on, off), the status of the doors (locked, unlocked, opened), if the car is moving.

1.5: Stakeholders

The main stakeholder is the company itself, that wants to have a better service for its clients and to make a reliable and easy-to-use system, so that as many people as possible can begin to use it. This in order to promote environment-friendly behavior that will help to produce less pollution.

1.6: References

- IEEE Standard for RASD document
- gm.polimi.it (For GEB System)
- Assignments AA 2016-2017.pdf

1.7: Overview

The RASD document will contain a sequence of paragraphs that will explain the entirety of the project.

- **1. Introduction:**
  Introduces the general aspects and the scope of the project.
- **2. Overall description:**
  Describes the factors that can influence the system, like the product's functions, the characteristics of the user that will use the system and the constraints on the project.
- **3. Specific requirements:**
  Contains all the analysis done on the project, like use cases, scenarios, diagrams and software system attributes.
- **4. Appendixes:**
  Contains the alloy modeling of the project, Alloy instances, used software and hours of work.

2: Overall Description

This chapter will describe all the factors that can affect the system (Product's functions, characteristics of the user that will use the system, constraints on the project).

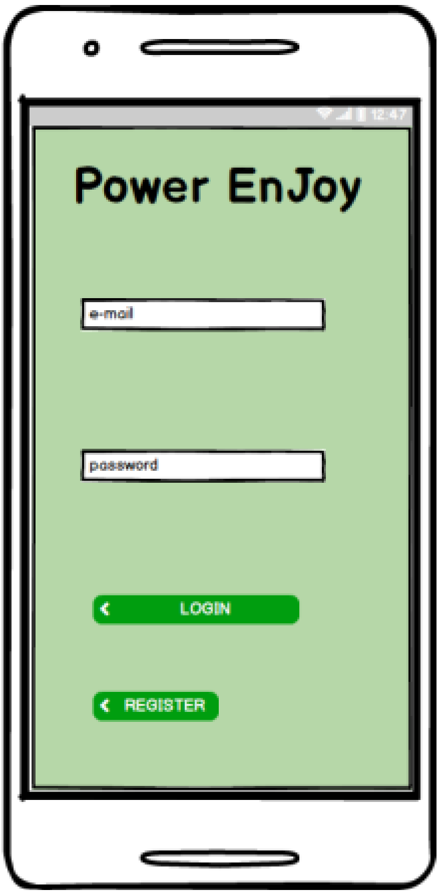
2.1: Product Perspective

Our system will interact with an external society to manage the payment information of the users and to validate their credentials. Also It will interact with the national motorization to validate the driving license of the user, linked to their account. It will integrate the GEB system, that is already embedded in the cars that will be used to provide the car-sharing service.

2.1.1 User Interfaces

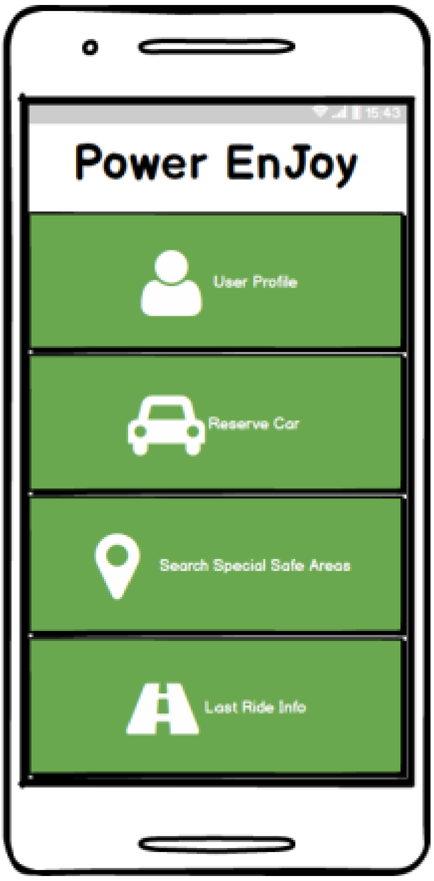
The mobile app will have almost this kind of interface, to interact with the user and let him use the Power EnJoy car sharing service. These mockups have been made, following the "good practices" for designing user interfaces: like depth of navigation when interacting with the system and quick access the main functionalities.

When the user will start the app, the system will show the following login layout:



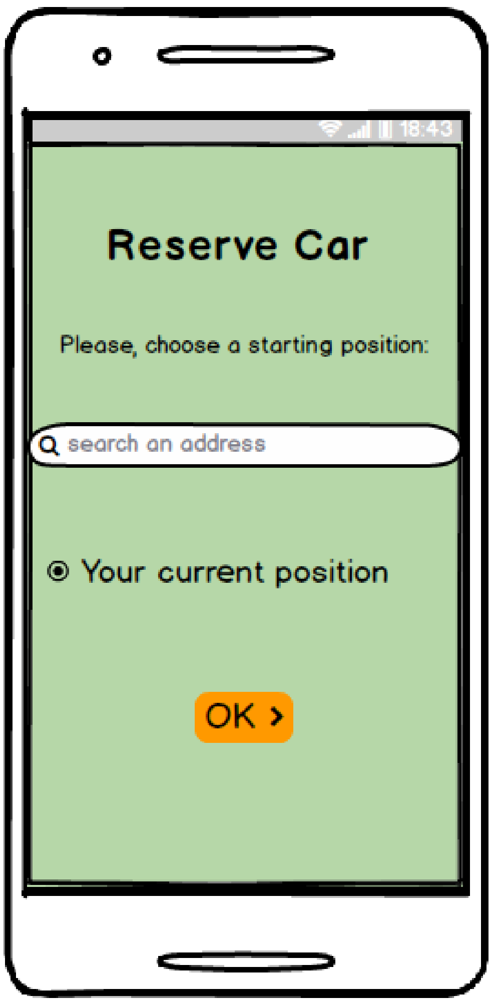
User Interface 1 Login layout

As the user’s credentials are validated and verified, he will see a main menu option as It’s shown by the following image:



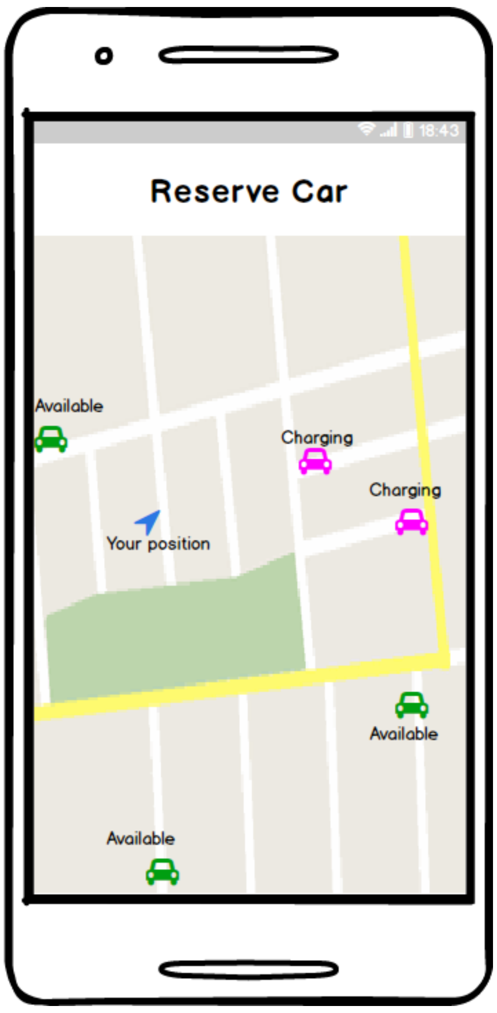
User Interface 2 Full interface main menu

The user profile shows, credentials and payment information, while through the “last Ride Info” he can access to a summary of his last ride if he has already done it and then also manage the pending payment (is he has one) of the ride.
He will also able to search for a special safe area and make a reservation with the “Reserve Car” option, which shows a search bar where the user can enter a specific address of interest for the car’s reservation, as it’s displayed in the following layout:



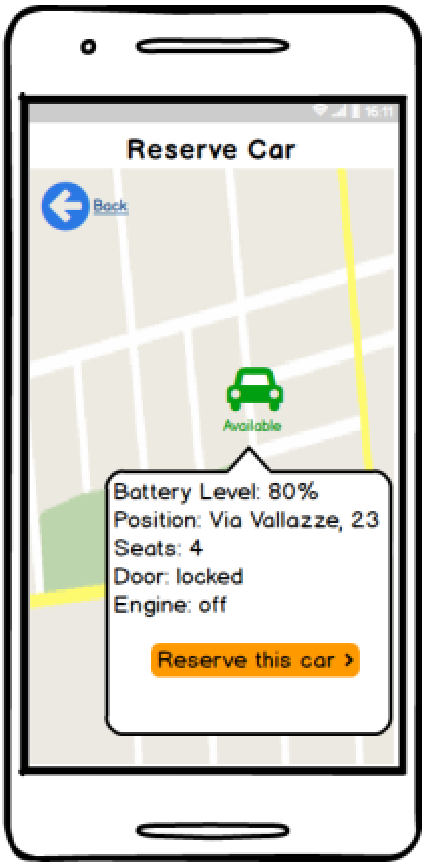
User Interface 3 Reservation Page

After having searched for an address or having given his current position, a map will the nearby cars and their state will be displayed, like in the following layout:



User Interface 4 Showing car nearby

When the user will tap on one of these car, he will see more details about the car’s state and if It’s available he will have the opportunity to reserve it, as the following image shows:



Battery Level: 80%
Position: Via Vallazze, 23
Seats: 4
Door: locked
Engine: off

User Interface 5 Car's state

In general, the system will have a typical navigation style, fast and easy to use.

2.1.2 Hardware interfaces

As mentioned before the system will interface with the GEB (Green e-Box) architecture, with which all vehicles are equipped. This will allow the user to directly connect to the car by using his smartphone's Bluetooth or NFC, and so open the car and manage his ride and reservation, even when there is no 3G/Wi-Fi connection available between the GEB and the central control system.

2.1.3 Software interface

For the development of our software we will rely on these technologies:

- Google Map APIs to show and search cars, safe and special safe areas
- MySQL to store user's passwords, information about cars and special safe areas
- Internet for connection for communication of data, such as keys to unlock the car, notifications to reserve car or call a maintenance operator, so for communication between car, user and central system
- Java to develop the android app
- Browser that support javascript and ajax
- HTML to create the web page

2.2: Product functions

In this section will be described all the functional and non-functional requirements of the software, related to the goals that are satisfied.

2.2.1: Functional Requirements

The functional requirements are divided in four major types:

- Managing user's profile
- Managing user's rental of a car
- Managing car
- Managing ride

2.2.1.1: Managing users' profile

[FR1] Register to the system

[FR2] Consult user profile

[FR3] Update user profile (which we haven't developed so much)

[FR4] Login

[FR5] Block (if he didn't pay the last ride) and unblock the user (when he fulfills his pending payment)

[FR6] Detect user position according to his GPS

[NFR1] The password must be saved securely

[NFR2] The system must manage high number of users

2.2.1.2: Managing users' rental of a car

[FR7] Allow user to reserve a single car

[FR8] Allow user to cancel a rental within one hour from the reservation starting time

[FR9] A car has to be picked up in at most one hour from the reservation starting time. As an extreme case, a car can be reserved and picked up immediately after.

[FR10] Fine a user 1 euro, if he doesn't pick up his reserved car

[FR11] Allow the user to unlock the car by selecting the reserved car on his app and clicking the button "Open car".

[FR12] Reset the one hour timer of the reservation only when the engine starts

[NFR3] The system must manage high number of requests from different users

2.2.1.3: Managing car

[FR13] Detect cars' positions according to their GPS

[FR14] Detect cars' status (Available, Reserved, OutOfService, Charging)

[FR15] Detect cars' battery level

[FR16] Detect if a car is plugged into the power grid

2.2.1.4: Managing ride

[FR17] Calculate the right amount of charge

[FR18] Detect the engine starting

[FR19] Show to the user safe areas for parking

[FR20] Show to the user available special safe areas

[FR21] The system must apply a 10% of discount on the ride, if it's notified of the presence of at least two other passengers in the car

[FR22] Detect the ending of a rental, if the engine is stopped, the doors are open and the car is still

[FR23] The system must apply a 20% of discount on the ride, if the rental has ended and the battery of the car is almost at the 50%

[FR24] The system must apply a 30% of discount on the ride, if it's plugged into the power grid

[FR25] The system must charge 30% more on the last ride, if a car is left at more than 3 KM from the nearest special safe area or with more than 80% of the battery empty

Here there is also a table that maps all the goals to which functional requirements they create.

Goal	Functional Requirements
G1	FR1, FR2, FR3, FR4
G2	FR6
G3	FR7, FR8, FR9
G4	FR10, FR12
G5	FR11, FR14, FR15, FR18
G6	FR17
G7	FR13, FR19, FR20
G8	FR5, FR15, FR16, FR21, FR22, FR23, FR24, FR25
G9	FR15, FR16, FR24
G10	FR13, FR25

2.3: User Characteristics

The users that will use the system normally have these characteristics:

- People with a driving license and a bank account.
- Knowledge in using a web app
- Knowledge in using a browser
- Must be able to read maps
- Have a smartphone

2.4: Constraints

2.4.1: Regulatory policies

The application must ask the user to use his GPS location to find his position. It must manage sensible data (phone number, position and payment information) in accordance with the privacy laws.

2.4.2: Hardware limitations

The user’s smartphone must be able to have a 3G or Wi-Fi connection, must have a GPS and space for the app package.

2.4.3: Interface to other applications

The system will have an interface with a bank application that will manage all the payment information for the users; it will also have an interface with the GEB system.

2.4.4: Parallel Operations

The system must be able to support parallel operations between different users at a time.

2.5: Assumptions

2.5.1: Domain assumptions

We suppose that the following properties are valid in the world:

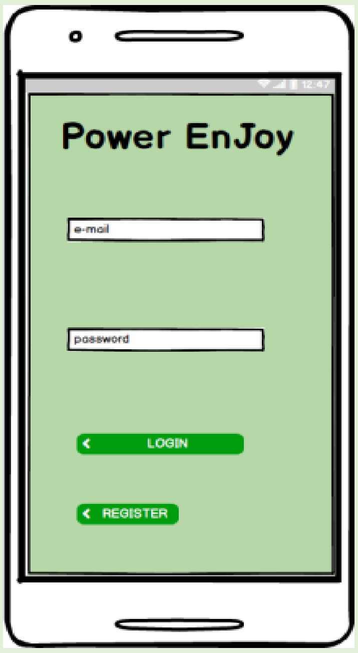
- Only registered users can access to the system
- The password assigned to the user is unique
- Payment information are verified and managed by an external service
- Each user has a single and unique payment information
- All GPS of both cars and users give always the right geolocation
- The cars’ GPS is always switched on
- During the ride the screen in the car is always working
- The car locks when the engine starts
- All cars have sensors which detect the exact presence of the number of passengers
- All special safe areas have a fixed number of powers grid stations
- All special safe areas can detect the number of parked cars
- Accidents are managed through a green number available 24/24 hours
- If an accident occurs it’s up to the maintenance operator, who is called to intervene, to decide how to set the car’s status
- Car’s led that shows its state is always visible and working
- Cars can be recharged by a maintenance operator or if they are into a special safe area
- The user must be constantly aware of the charge during the ride

2.5.2: Text assumptions

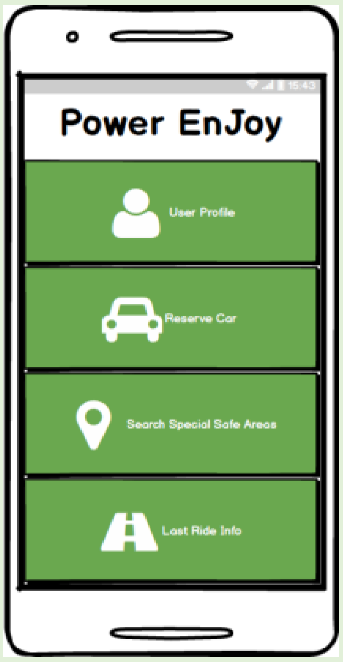
- We should develop a mobile app and a web page. Both the app and the web page let the user register himself and make a reservation, both using a position given by the user, the mobile app also by taking the GPS signal of the user.
- We assume that the fee per minute is fixed
- Safe areas and special safe areas are fixed and predefined
- We assume that the discount for carrying at least two other passengers is applied also even if the passengers don’t share the whole ride with the driver

3: Specific Requirements

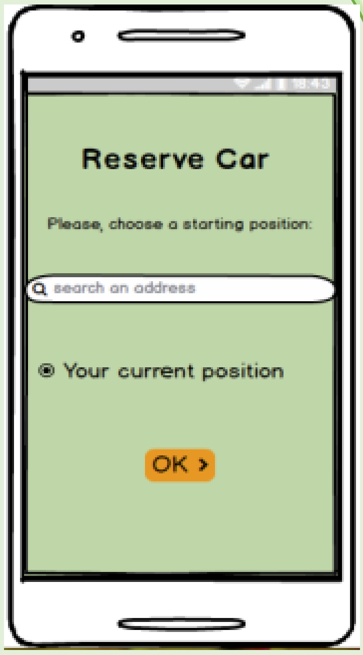
3.1: External Interfaces



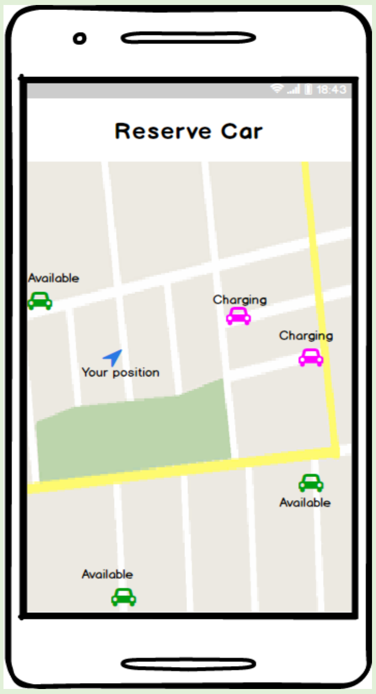
After the login, the user accesses to the main options' menu



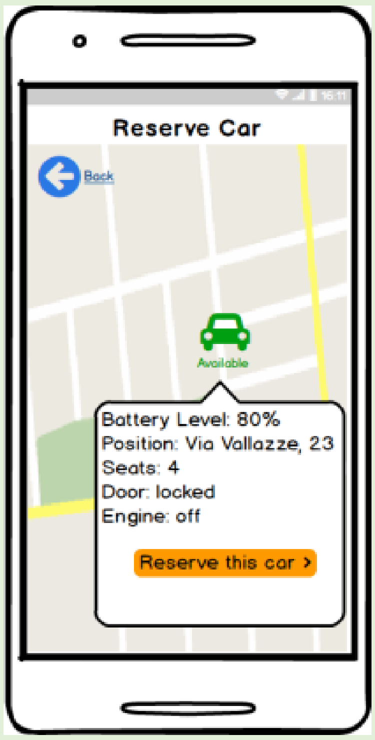
User chooses a starting position



The Reserve Car option shows the map with nearby available or charging cars



The user can see the car's state, by tapping on it and then chose to make a reservation



Battery Level: 80%
Position: Via Vallazze, 23
Seats: 4
Door: locked
Engine: off

3.2: Functional requirements

3.2.1: Scenarios

Here there are some scenarios for the usage of our system. Each of them has a list of which functional requirements it satisfies.

3.2.1.1: Scenario 1

Today Ronald has to go to work. But her daughter, Mary, must use the car too, so he, being a good father, lets her take the car. Since it is raining very hard, Ronald decides to rent a car at PowerEnJoy. He opens his laptop and accesses to PowerEnJoy’s webpage. He logins to the site and indicates his position. The system calculates the position of Available cars near John’s house and returns them to John. He then chooses a nearby car to use for the ride. Today Ronald will arrive dry at the office.

3.2.1.2: Scenario 2

Rick is a middle-aged, Canadian man, grown in the large forests of his nation, and so he is very sensitive about the environmental issues of the Earth. He has recently discovered that his city has now started to use the PowerEnJoy system, and he is so overjoyed that he decides to register himself to the site. He turns on his old-fashioned computer and goes on the webpage of the system. He chooses the option of registering and inserts his name, surname, email and payment information. He then receives his password via e-mail. He decides to try immediately the system. He logins in the system and chooses to reserve a car to go to his favorite electronic goods store. He chooses to reserve a car for tomorrow, but finds out that he can only make a reservation for up to one hour from now. He decides to reserve the car tomorrow. Today Rick is slightly happier than usual.

3.2.1.3: Scenario 3

Alan always uses PowerEnJoy to return home from his office every evening, but today he has forgotten to reserve a car. He logins from his smartphone to PowerEnJoy’s system. He lets the app take his position from the GPS and visualizes all the available cars near him. He reserves the nearest car and exits the office. Arrived at the cars he accesses to his profile and unlocks the car. He enters in the car and starts the engine. The system detects the engine starting, stops the one hour timer and begins calculating the charge, while showing it in the car’s monitor. Alan uses the car’s GPS to locate safe areas and special safe areas for parking. He chooses to park the car in the nearest safe area, which hasn’t a power grid. Being the car at 70% of the total battery charge, Alan will receive a 20% of discount on the ride.

3.2.1.4: Scenario 4

John, Paul, Ringo and George are four good friends. They decide to have a dinner all together, and to use PowerEnJoy in order to save the environment. Ringo logins in his PowerEnJoy’s account and reserves a car.
Half an hour later Ringo takes the car and goes to pick up his friends. After they got all together,

they go to their favorite restaurant, Let It Be. Arrived near the restaurant, they decide to park the car in a special safe zone. Being the car at 57% of the total battery charge, they decide to plug it in one of the power grids. The system detects that four people were in the car and that after the ride the car has been plugged into a power grid, so the total charge amount has a discount of60%. The four friends have a good evening in their favorite restaurant.

3.2.1.5: Scenario 5

Helen is a newly-employed young woman, working on an important piece of work in her office. She has already reserved a car from the PowerEnJoy's system. Selene, her senior colleague, invites her at dinner, because there is no time to talk in the office and she wants to know Helen better. Helen accepts and logins in her PowerEnJoy's system, views her profile and chooses the reservation she already had, and then cancels it.
Before dinner Helen decides to reserve a new car in order to return home, because Selene's house is very distant from hers. She successfully reserves the car. Unfortunately, because of some problem in the restaurant's kitchen, the two women can't finish the dinner in time, and Helen is forced to pay the 1 euro fee for not having taken the car. The restaurant makes them a discount for the inconvenience, and Selene brings Helen home. The two women are now good friends.

3.2.1.6: Scenario 6

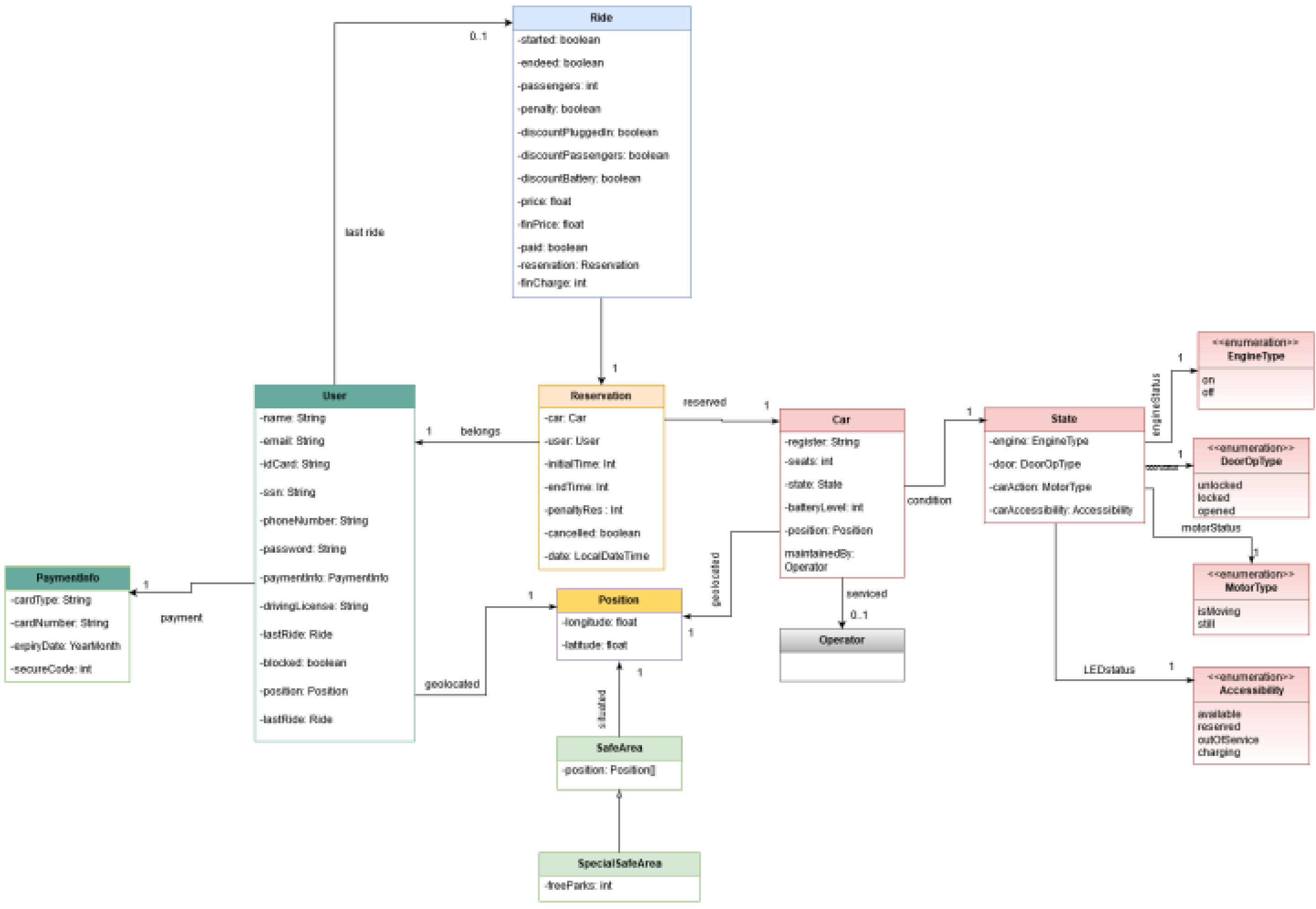
Rocky is the boss of a big company. He is arrogant and likes big, polluting cars. To build a better image of himself he is advised to sometimes take an electric car in order to show how much he cares about the environment. He then logins in his almost non-used PowerEnJoy's account and reserves the nearest car.
He then chooses to do a very long ride in the car circling almost all the city, in order to make as many people as possible his good deed. He then returns at his office and drops the car in a special safe area, with only 5% of the total battery charge and without caring of plugging it in a power grid (even if he had one right in front of him). After five minutes of waiting, the system decides that he doesn't want to plug the car in the power grid, and sends a request of recharging to an operator. Rick will have to pay 30% more for this uncaring behavior.
He then takes his car and returns home. Unfortunately for him, he had registered in the system his old credit card, which hasn't a single cent on it anymore.
Isaac, one of PowerEnJoy's employees, receives a notification from the system that the car is with too little charge. He accepts the recharging request and is forced to reach the car and recharge it manually.
Some weeks later Rocky prepares a new trip around the city, but discovers that, because he hasn't paid the last ride, he is blocked from using the PowerEnJoy's service. He then has to update his profile with the new informations and pay his last ride before he can reserve a new one.

Here is a table that puts in evidence all the scenarios and the Functional requirements that are used in them.

Scenario	FRs
1	FR2, FR4, FR7, FR13, FR14
2	FR1, FR4, FR7
3	FR4, FR6, FR11, FR12, FR13, FR14, FR15, FR17, FR19, FR20, FR22, FR23
4	FR4, FR7, FR11, FR12, FR14, FR15, FR16, FR17, FR18, FR19, FR20, FR21, FR22, FR23, FR24
5	FR2, FR7, FR8, FR9, FR10
6	FR2, FR3, FR4, FR5, FR7, FR15, FR25

3.2.2: Analysis diagram

This is the UML model for our system (An image of the UML model can also be found in the repo of the project):



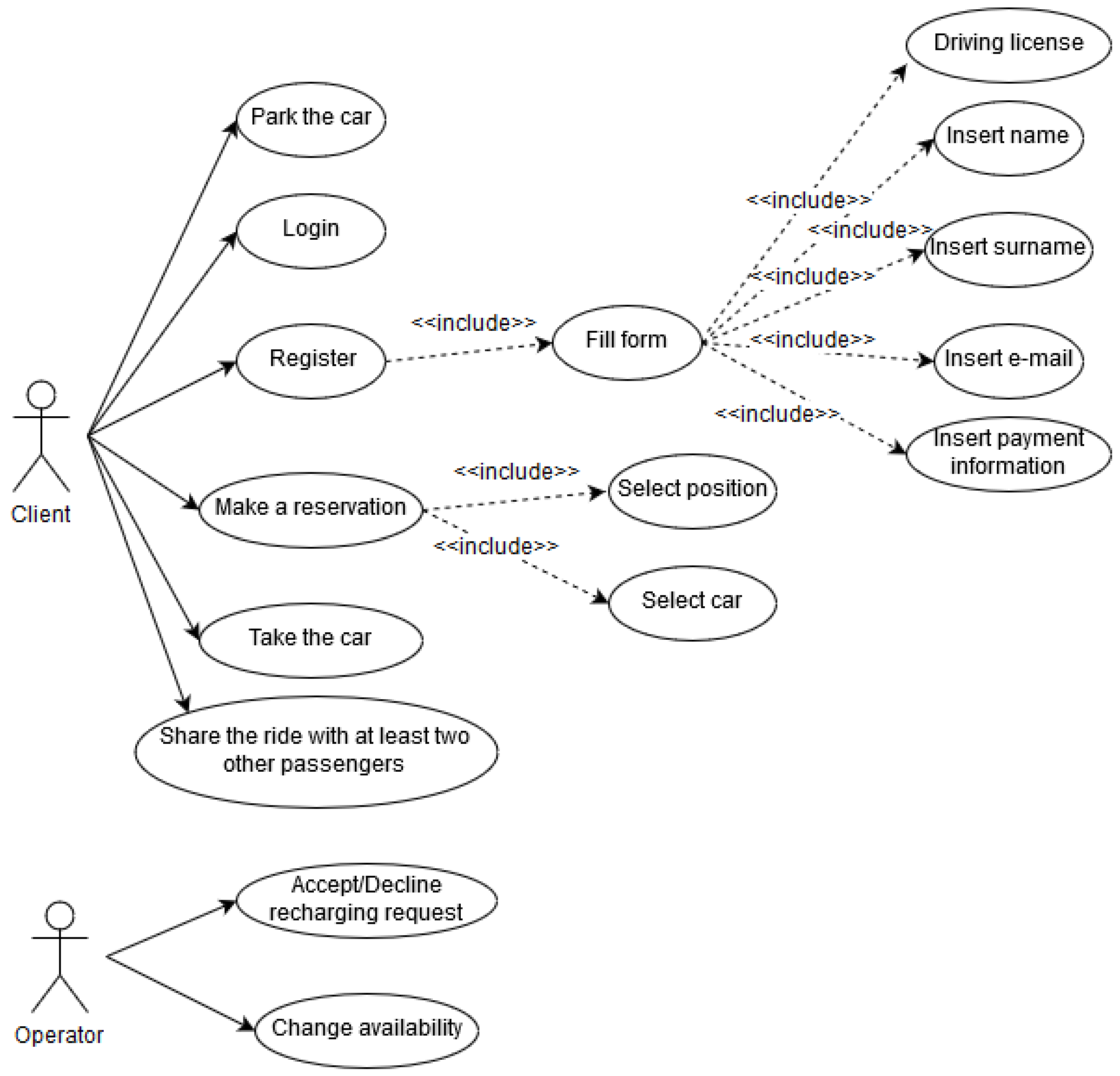
3.2.3: Use cases

This phase began with the identifications of all the actors of our system. These actors are:

- The user: a registered person that is able to make reservations.
- The operator: an employee of the company that works on maintaining the cars and that can receive and accept or decline recharging requests.

3.2.3.1: Use case diagram

Given these two actors we were able to identify these main use cases:



3.2.3.2: Description of the use cases:

Here is a description of the use cases for our system. Everyone of them has a name, some entry conditions, a flow of events, some exit conditions and some exceptions for that use case.

3.2.3.1: Use case 1: User registers

Name: User registers.

Actors: User.

Entry Conditions: None.

Flow of events:

- The user opens the app or the webpage and arrives at the HomePage.
- The user inputs his name, surname, email and payment information.
- The user clicks on the registering button.
- The system sends to the user's email his new password.

Exit conditions: The system successfully sends the user his password.

Exceptions: The email does not exist or the payment informations are not correct. In this case, the system does not send any email and notifies him that an error has occurred, then lets him input his email and payment information again.

3.2.3.2: Use case 2: User Logins

Name: User logins.

Actors: User.

Entry Conditions: None.

Flow of events:

- The user opens the app or the webpage and arrives at the HomePage.
- The user inputs his email and password.
- The user clicks the log in button.
- The system redirects the user to his personal page.

Exit conditions: The user is successfully redirected to his personal page.

Exceptions: The email and password are not correct. In this case, the user is not redirected and is notified of the error. Then the system lets him input his email and password again.

3.2.3.3: Use case 3: User makes a reservation

Name: User makes a reservation.

Actors: User.

Entry conditions:

- The user must be logged in.
- The user must not have already done a reservation.

Flow of events:

- The user indicates the position from where to start the research or lets the app take his GPS signal.
- The system returns the available cars in the area and their position.
- The user selects the car that he wants to reserve and clicks the “reserve” button.
- The system tags the car as “Reserved”.
- The system sends an email to the user indicating the details of his reservation, and redirects him to a page with the same informations.

Exit conditions: The system successfully sends the email and redirects the user.
Exceptions:

- The application cannot read the user’s GPS or cannot find the location specified by the user. In this case, the system does not make the user select a car, notifies him of the error and lets him retry to use the GPS or to input a different location.
- There are no cars available in the area. In this case the system does not make the user choose a car and notifies him of the error, asking him to try to search from a different starting point, then lets him input a new location.

3.2.3.4: Use case 4: User takes the car

Name: User takes the car.
Actors: User.
Entry conditions:

- The user must have reserved a car.
- The user must have his Bluetooth turned on

Flow of events:

- The user logins in his profile.
- The user selects from his profile on the app the reservation.
- The user clicks on the “Connect to car” button
- The user clicks the “Open car” button.
- The system through the GEB opens the car.
- The user enters in the car and starts the engine.

Exit conditions: The system detects the car’s engine starting and stops the one hour reservation timer.
Exceptions: The car’s engine isn’t started for one hour after the reservation. In this case, the car is tagged as “Available”, the reservation is cancelled and the user pays 1 Euro as a fee.

3.2.3.5: Use case 5: Operator receives a recharging request

Name: Operator receives a recharging request.

Actors: Operator.

Entry conditions:

- A car must have 20% or less of charge or must be at least at 3 KM from the nearest power grid.
- The operator must be Available

Flow of events:

- The operator receives a notification of the request.
- The system shows the operator the details of the request and shows him two buttons: Accept and Reject.
- The operator chooses one of the two buttons.

Exit conditions:

- If the operator has chosen Accept, the operator’s status is now “Not Available”. Then the operator goes to the car and recharges it on site. Once this is complete, the system tags the car as “Available”.
- If the operator has chosen Decline, this use case restarts from the beginning with another operator.

Exceptions: There are no exceptions.

3.2.3.6: Use case 6: Operator changes his availability

Name: Operator changes his availability.

Actors: Operator.

Entry conditions: None.

Flow of the events:

- The operator accedes at his computer in PowerEnJoy’s Office.
- The operator selects one of the two buttons “Available” or “Not Available”.

Exit conditions: The system successfully changes the operator’s availability status.

Exceptions: None.

3.2.3.7: Use case 7: Two more people enter in the car

Name: Two more people enter in the car.

Actors: User(s).

Entry conditions: A user has already taken a car.

Flow of events:

- Two more people enter in the car.
- The system detects three people in the car and flags the 10% discount for the ride.

Exit conditions: The system has applied the 10% discount.

Exceptions: None.

3.2.3.8: Use case 8: The user parks in a special safe area

Name: The user parks in a special safe area.

Actors: User.

Entry conditions: A user has already taken a car.

Flow of events:

- The user visualizes on the car's GPS the location of special safe areas.
- The user enters a special safe area with the car.
- The user parks the car and turns off the engine.

Exit conditions:

- If the user plugs the car to the power grid before five minutes pass, he receives a discount on the last ride.
- If the user does not plug the car to the power grid before five minutes and the car has more than 20% of the battery, there are no penalties or discounts applied.
- If the user does not plug the car to the power grid before five minutes and the car has less than 20% of the battery, he receives a penalty of 30% more to the last ride.

Here is the continuation of the table in section 2.2.1, which mapped the goals to the corresponding use cases. Now the table also maps the use cases that satisfy the goals and functional requirements

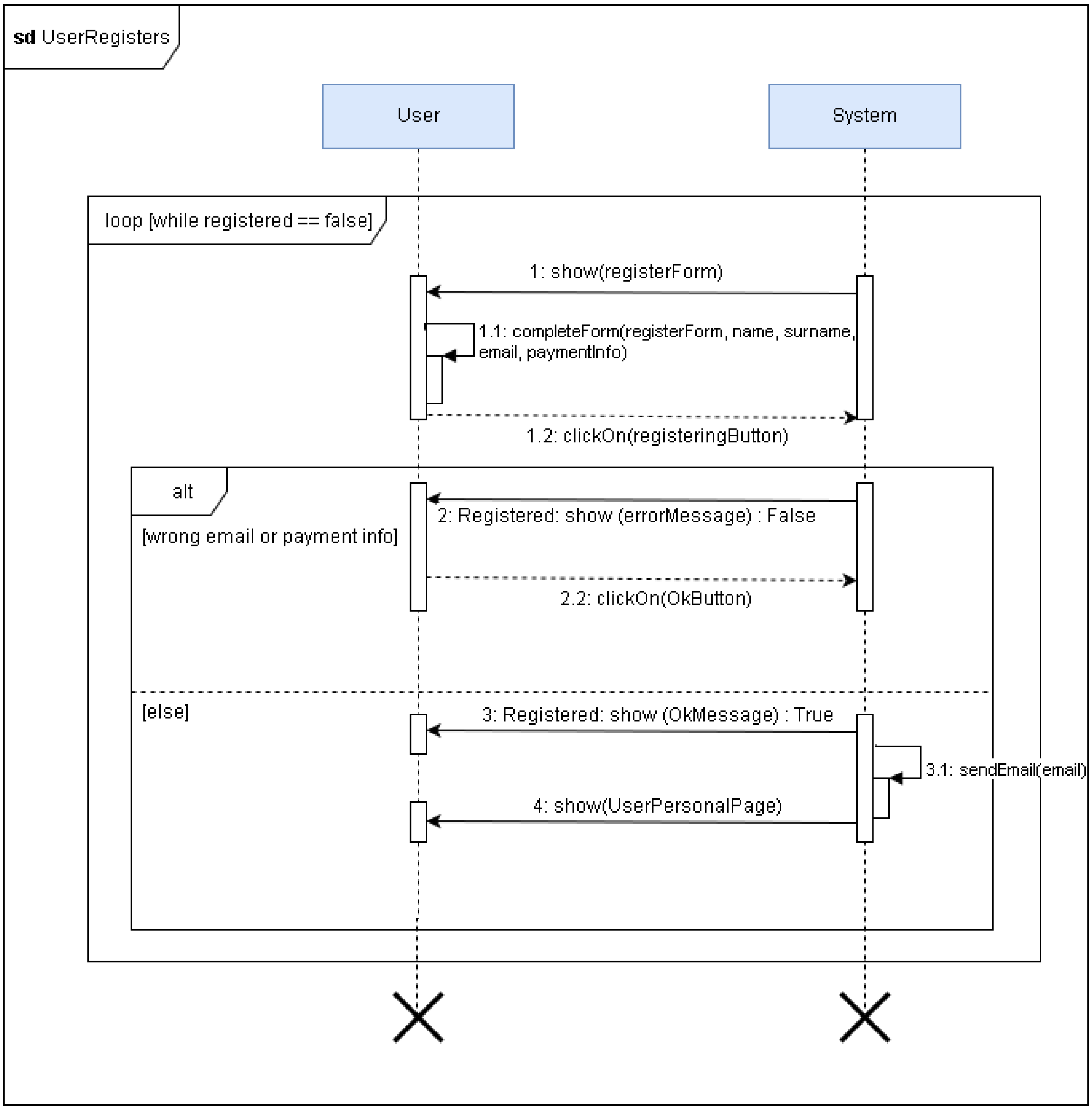
Goal	Functional Requirements	Use Cases
G1	FR1, FR2, FR3, FR4	UC1, UC2
G2	FR6	UC3
G3	FR7, FR8, FR9	UC3
G4	FR10, FR12	UC4
G5	FR11, FR14, FR15, FR18	UC4, UC8
G6	FR17	UC4, UC8
G7	FR13, FR19, FR20	UC8
G8	FR5, FR15, FR16, FR21, FR22, FR23, FR24, FR25	UC5, UC6, UC7, UC8
G9	FR15, FR16, FR24	UC8

G10	FR13, FR25	UC5, UC8

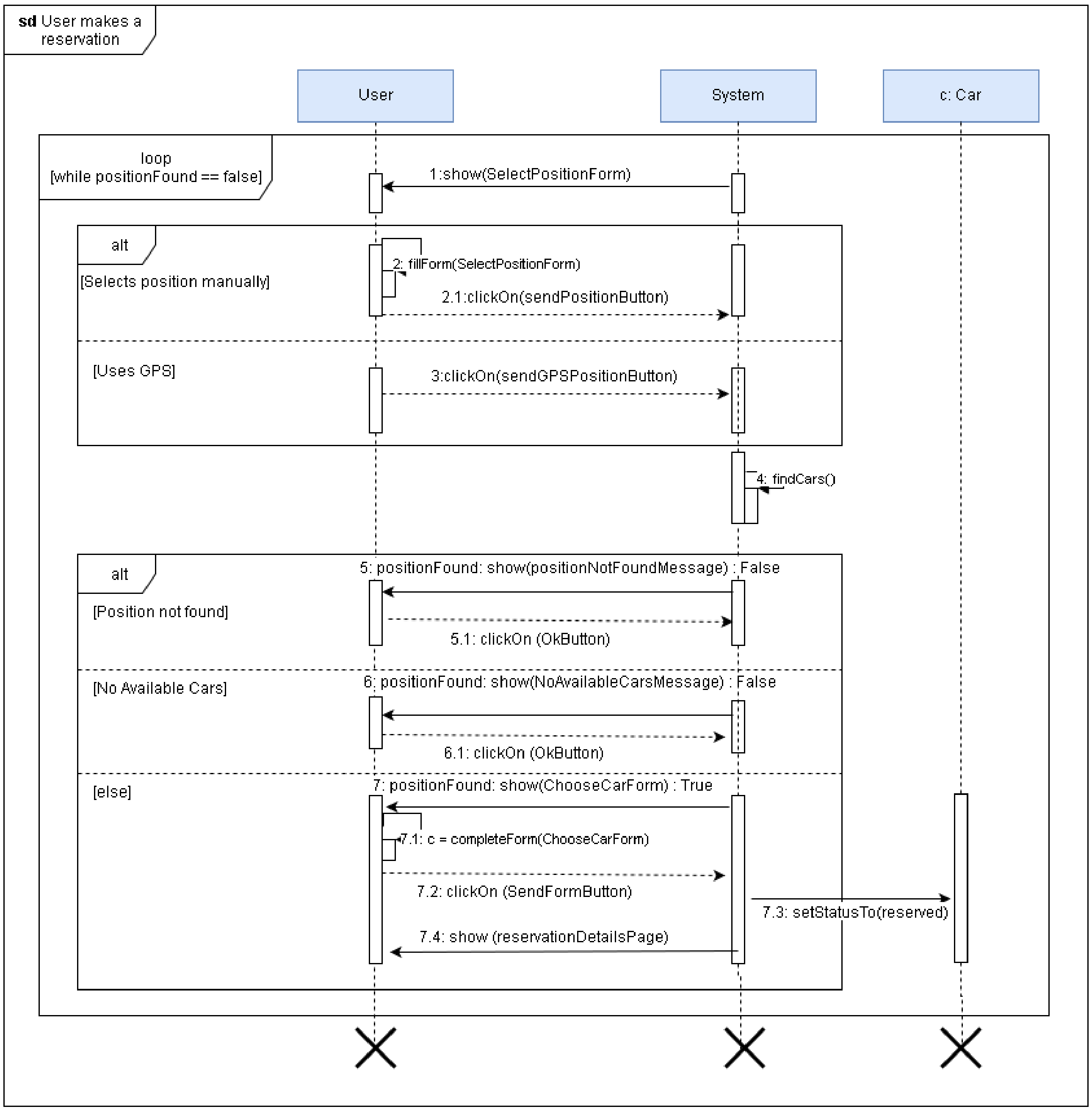
3.2.4: Sequence diagrams

We will now see the main sequence diagrams for all the principal use cases that were already described.

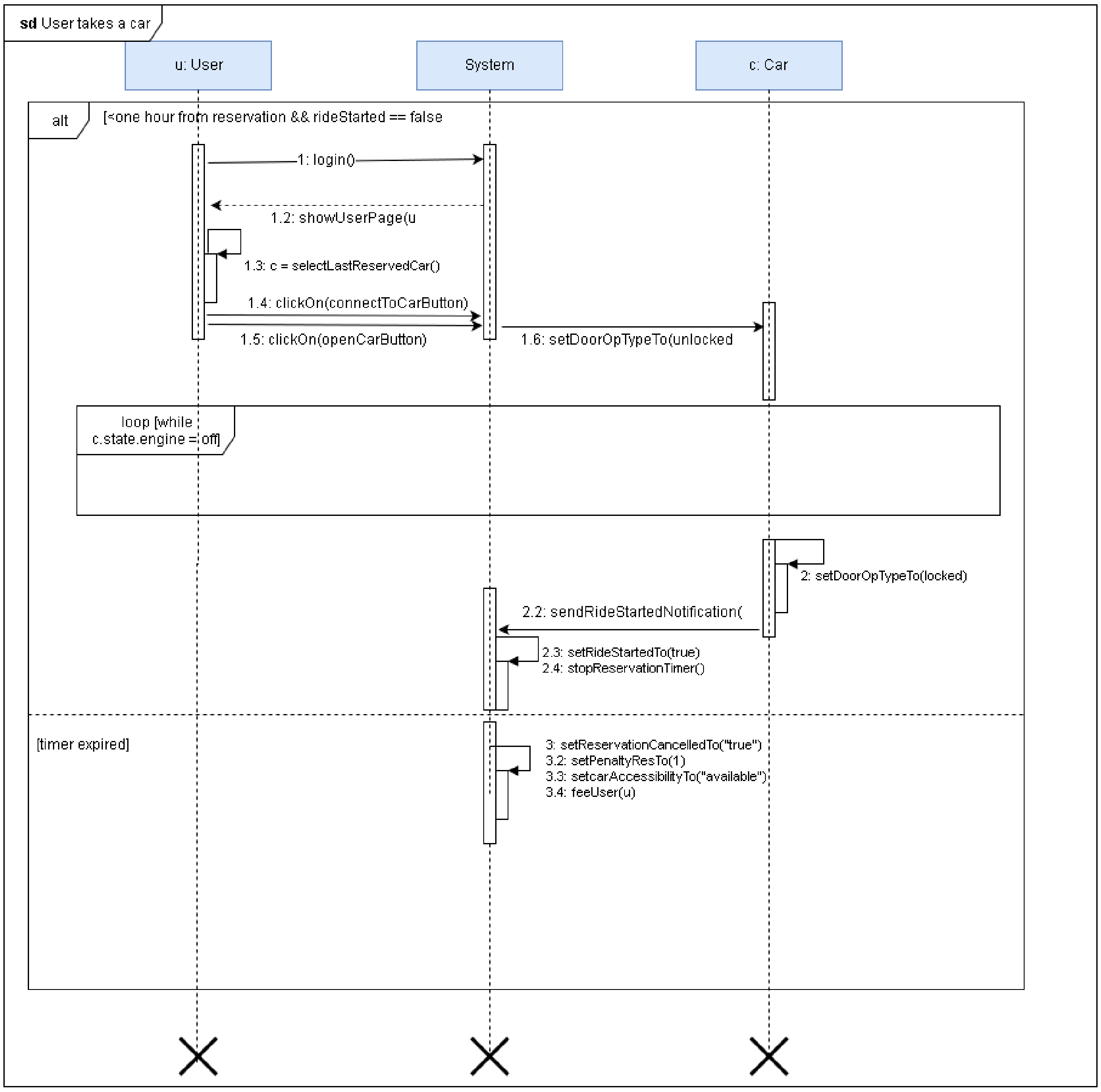
3.2.4.1: Use case 1: User Registers


sd UserRegisters
User
System
loop [while registered == false]
1: show(registerForm)
1.1: completeForm(registerForm, name, surname, email, paymentInfo)
1.2: clickOn(registeringButton)
alt
[wrong email or payment info]
2: Registered: show (errorMessage) : False
2.2: clickOn(OkButton)
[else]
3: Registered: show (OkMessage) : True
3.1: sendEmail(email)
4: show(UserPersonalPage)

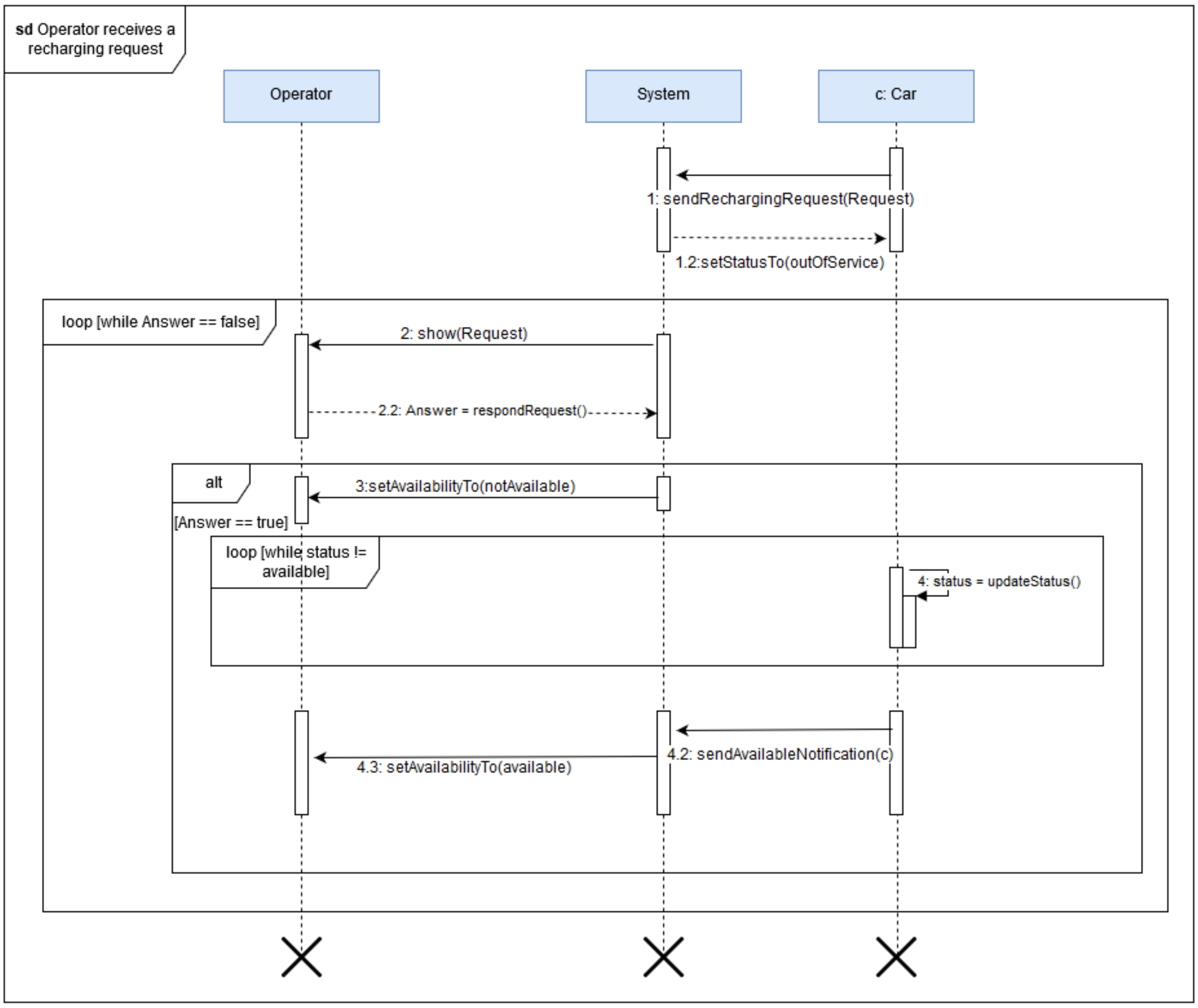
3.2.4.2: Use case 3: User makes a reservation


sd User makes a reservation
User
System
c: Car
loop
[while positionFound == false]
1:show(SelectPositionForm)
alt
[Selects position manually]
2: fillForm(SelectPositionForm)
2.1:clickOn(sendPositionButton)
[Uses GPS]
3:clickOn(sendGPSPositionButton)
4: findCars()
alt
[Position not found]
5: positionFound: show(positionNotFoundMessage) : False
5.1: clickOn (OkButton)
[No Available Cars]
6: positionFound: show(NoAvailableCarsMessage) : False
6.1: clickOn (OkButton)
[else]
7: positionFound: show(ChooseCarForm) : True
7.1: c = completeForm(ChooseCarForm)
7.2: clickOn (SendFormButton)
7.3: setStatusTo(reserved)
7.4: show (reservationDetailsPage)

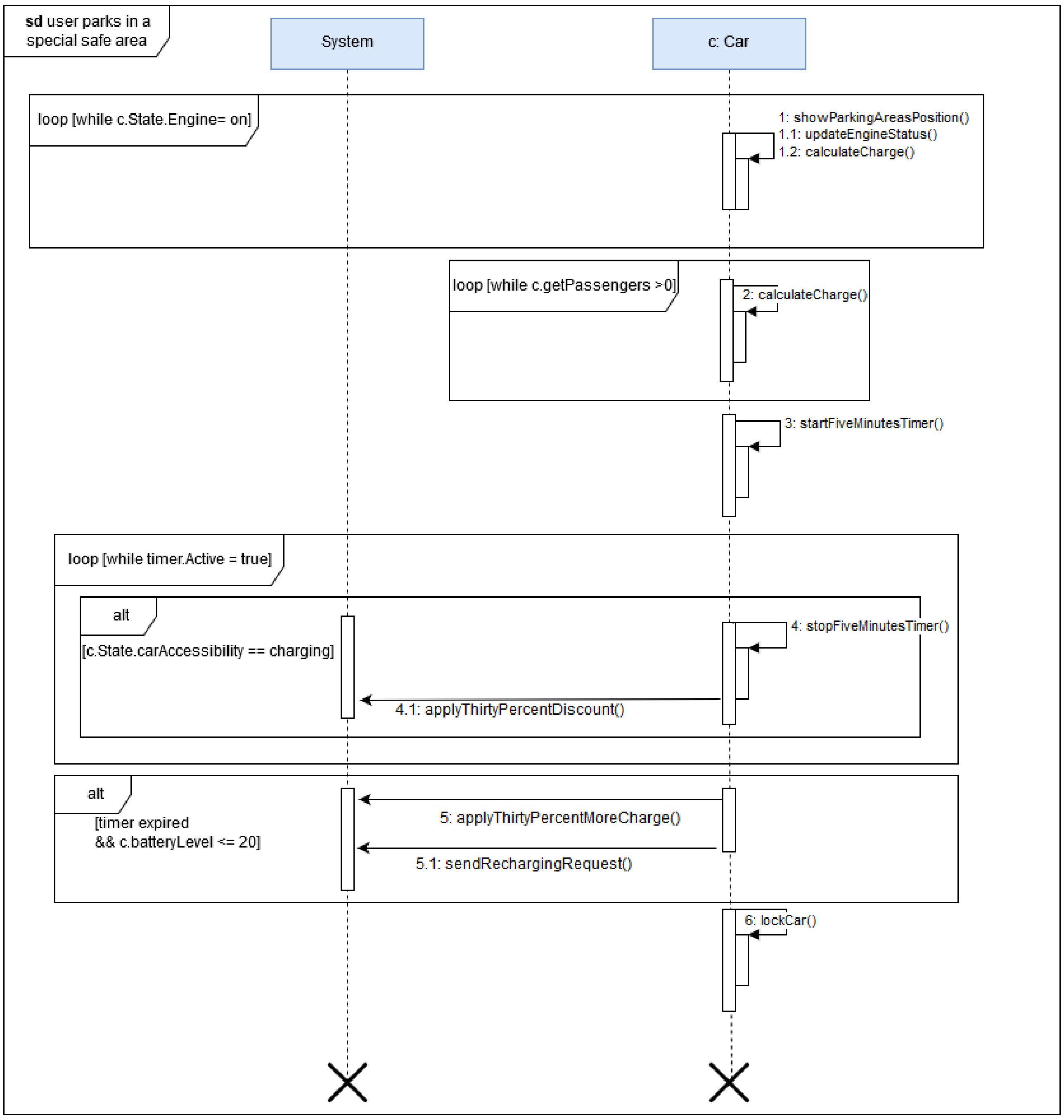
3.2.4.3: Use case 4: User takes the car


sd User takes a car
u: User
System
c: Car
alt
[<one hour from reservation && rideStarted == false
1: login()
1.2: showUserPage(u
1.3: c = selectLastReservedCar()
1.4: clickOn(connectToCarButton)
1.5: clickOn(openCarButton)
1.6: setDoorOpTypeTo(unlocked
loop [while c.state.engine = off]
2: setDoorOpTypeTo(locked)
2.2: sendRideStartedNotification(
2.3: setRideStartedTo(true)
2.4: stopReservationTimer()
[timer expired]
3: setReservationCancelledTo("true")
3.2: setPenaltyResTo(1)
3.3: setcarAccessibilityTo("available")
3.4: feeUser(u)

3.2.4.4: Use case 5: Operator receives a recharging request


sd Operator receives a recharging request
Operator
System
c: Car
1: sendRechargingRequest(Request)
1.2:setStatusTo(outOfService)
loop [while Answer == false]
2: show(Request)
2.2: Answer = respondRequest()
alt
3:setAvailabilityTo(notAvailable)
[Answer == true]
loop [while status != available]
4: status = updateStatus()
4.2: sendAvailableNotification(c)
4.3: setAvailabilityTo(available)

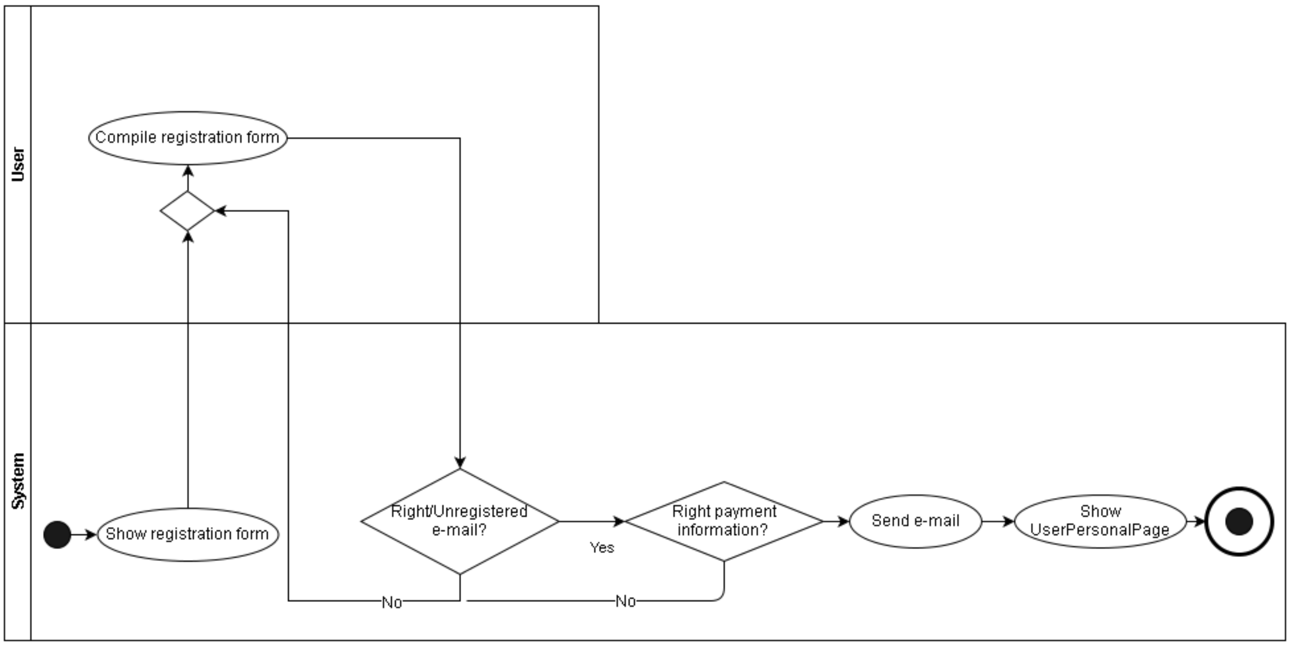
3.2.4.5: Use case 8: User parks in a special safe area


sd user parks in a special safe area
System
c: Car
loop [while c.State.Engine= on]
1: showParkingAreasPosition()
1.1: updateEngineStatus()
1.2: calculateCharge()
loop [while c.getPassengers >0]
2: calculateCharge()
3: startFiveMinutesTimer()
loop [while timer.Active = true]
alt
[c.State.carAccessibility == charging]
4: stopFiveMinutesTimer()
4.1: applyThirtyPercentDiscount()
alt
[timer expired && c.batteryLevel <= 20]
5: applyThirtyPercentMoreCharge()
5.1: sendRechargingRequest()
6: lockCar()

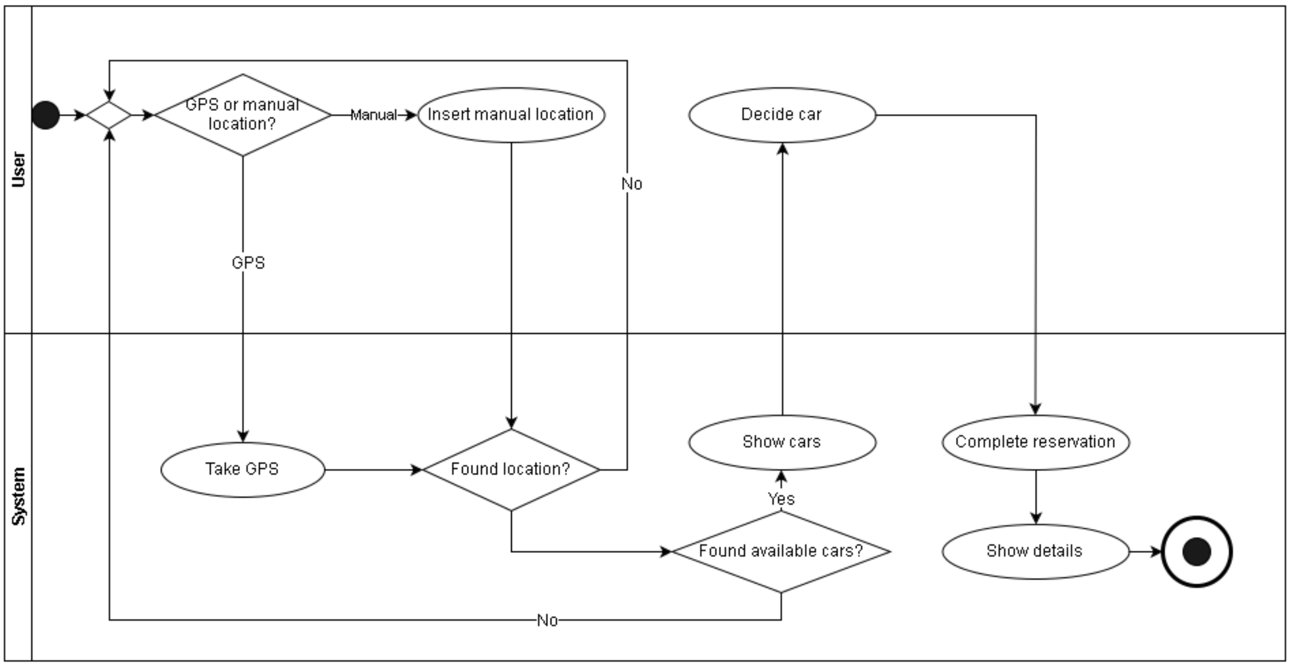
3.2.5: Activity Diagrams

We will now see the activity diagrams for all the sequence diagrams already proposed in the last section of this document.

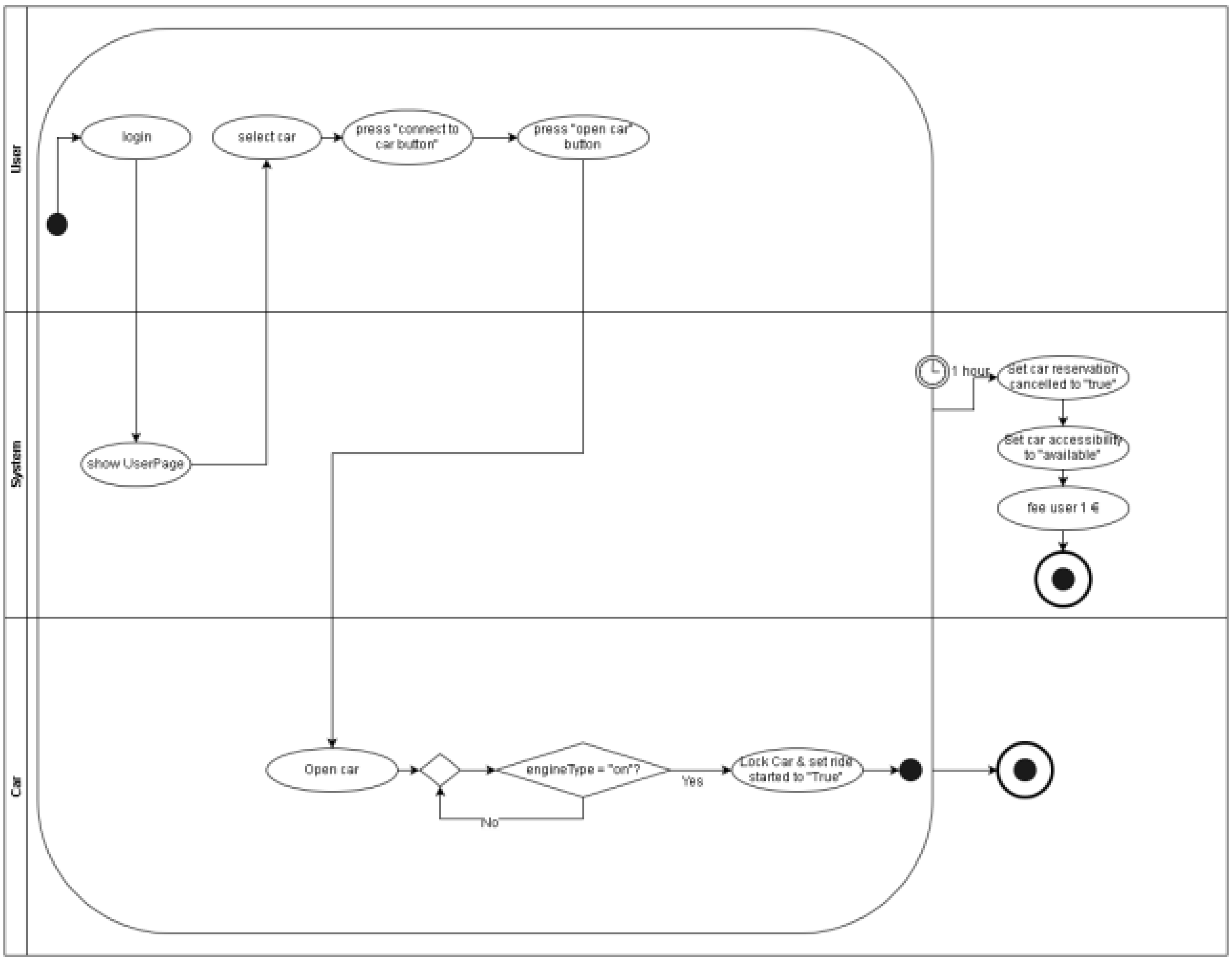
3.2.5.1: Use case 1: User registers


User
System
Compile registration form
Show registration form
Right/Unregistered e-mail?
Yes
No
Right payment information?
No
Send e-mail
Show UserPersonalPage

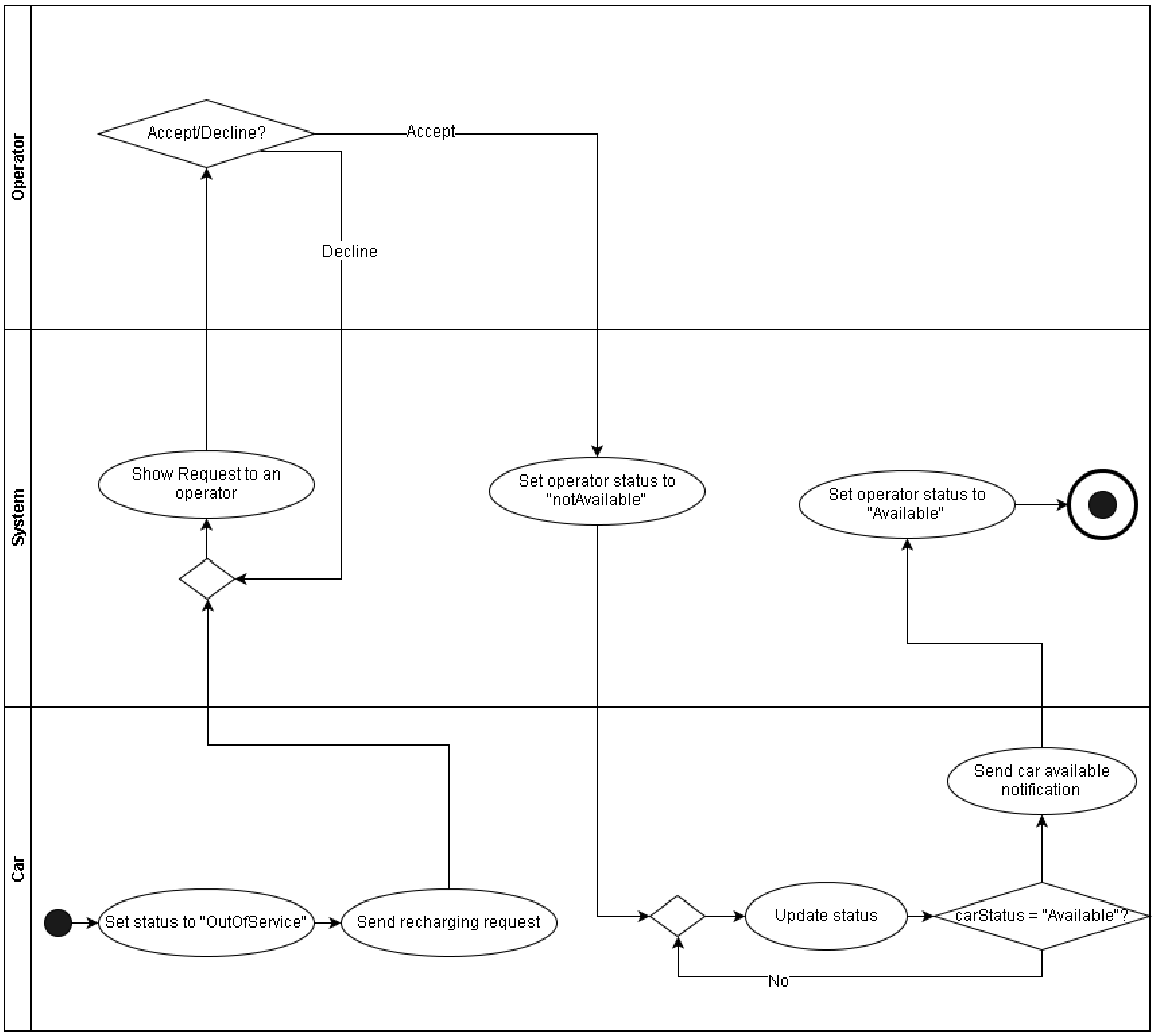
3.2.5.2: Use case 3: User makes a reservation



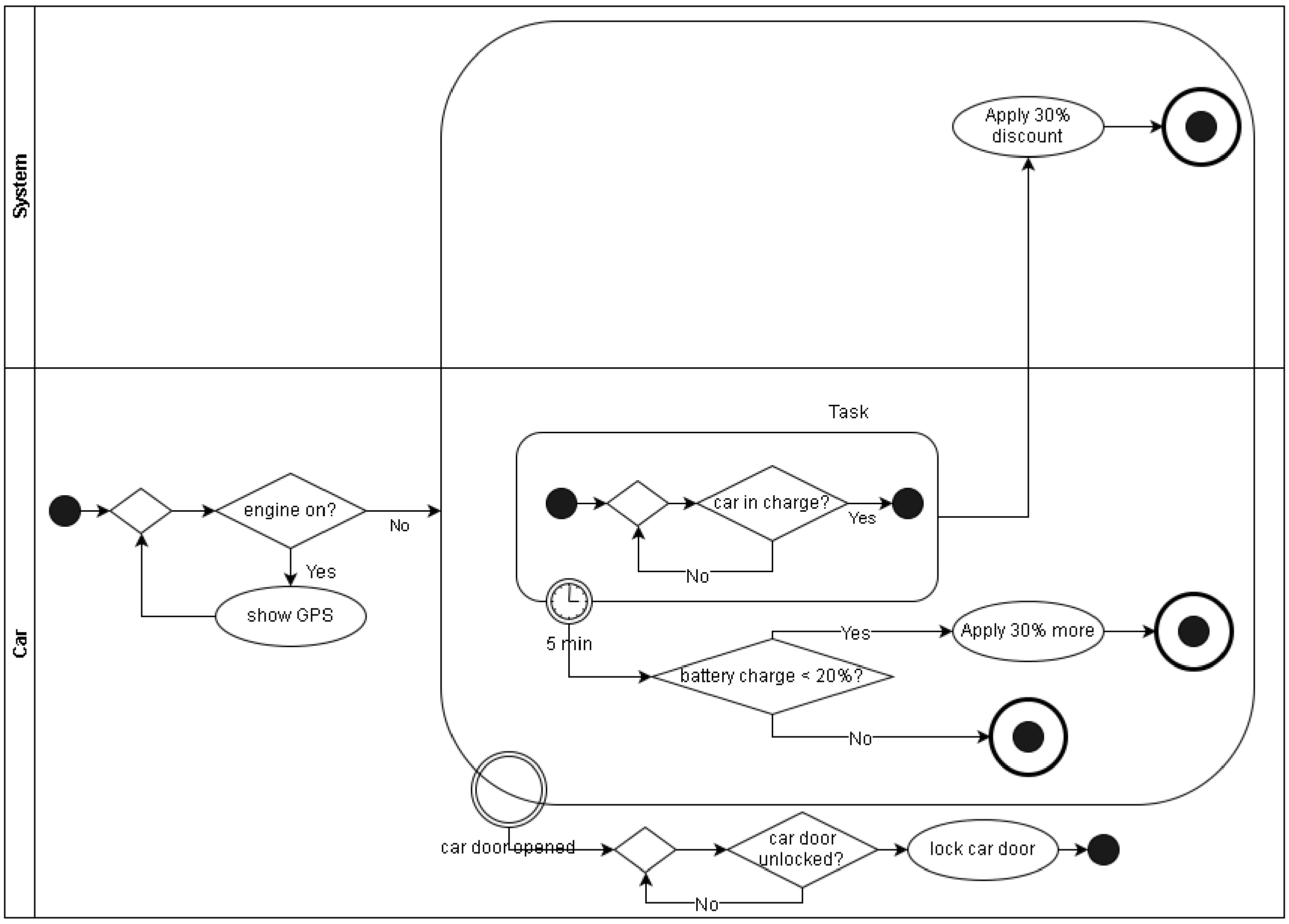
3.2.5.3: Use case 4: User takes the car


User
System
Car
login
select car
press "connect to car button"
press "open car" button
show UserPage
1 hour
Set car reservation cancelled to "true"
Set car accessibility to "available"
fee user 1 €
Open car
engineType = "on"?
Yes
No
Lock Car & set ride started to "True"

3.2.5.4: Use case 5: Operator receives a recharging request


Operator
System
Car
Accept/Decline?
Accept
Decline
Show Request to an operator
Set operator status to "notAvailable"
Set operator status to "Available"
Send car available notification
Set status to "OutOfService"
Send recharging request
Update status
carStatus = "Available"?
No

3.2.5.5: Use case 8: User parks in a special safe area



3.3: Performance requirements

The system is needed to have performance requirements in disk management and in the bandwidth in order to receive as many requests as possible, ideally one per car at every moment. It is so necessary to manage a high number of contemporary requests.

3.4: Logical database requirements

The database should be able to memorize all the different users along with their personal information and the last ride they have done. It also must be able, for future updates, to store all the rides done by a certain user at a time. It must also be able to memorize the cars used in the system and all their information.

3.5: Design requirements

The system must be designed and programmed using JEE technology.

3.5.1: Standard compliance

The code should be implemented by respecting the normal standards in order to make it readable and easily maintainable.

3.6: Software system attributes

3.6.1: Reliability

Our system has to ensure the integrity and durability of all the requests done by users.

3.6.2: Availability

Our system has to be available every month of the year, every day of the month, at every hour of the day.

3.6.3: Security

Our system must be able to securely store the sensible data about the user, like it's info, in particular its payment information. It should use security protocols.

3.6.4: Maintainability

The system should be highly maintainable and it should possibly be maintained without shutting down the server of the system or, if this is not possible, reducing at the minimum the down time.

3.6.5: Portability

The system should be able to be run in any web browser and in all operating systems (Windows, iOS, Android…)

4: Appendixes

4.1: Alloy modeling

```
open util/boolean

open util/ integer as integer


enum DoorOpType{ unlocked, locked,opened}

enum EngineType{on,off}

enum MotorType{ isMoving, still}

enum Accessibility{ available, reserved, outOfService, charging}


sig State{

        engine: EngineType,

        door: DoorOpType,

        carAction: MotorType,

        carAccessibility: Accessibility

}{

        carAction=isMoving => door=locked

        carAction=isMoving=> engine=on

        carAction=isMoving=> carAccessibility=reserved

        carAccessibility=charging => (carAction=still and engine=off)

        carAccessibility=outOfService => carAction=still and engine=off

        carAccessibility=available => carAction=still and engine=off and door=locked


}


sig Car{
        seats: Int,

        state: one State,

        position: one Position,

        maintainedBy: lone Operator,

        batteryLevel: Int

} {

        seats>0 and seats<=4

        batteryLevel>=0 and batteryLevel<=100

        batteryLevel=0 => state.engine=off and state.carAction= still

        #maintainedBy=1 => state.carAccessibility=outOfService

}

sig Position{

        latitude: Int, // float

        longitude: Int, // float

}


sig Reservation{

        initialTime: Int,

        endTime:Int,

        date: Int,

        penaltyRes: Int,

        cancelled: Bool,

        user: one User,

        car: one Car

} {     minus[endTime,initialTime]<=60

        penaltyRes=0 or penaltyRes=1

        //postCondition of a forced and spontaneus cancellation

        (minus[endTime,initialTime]<=60  and (car.state.carAccessibility= available or car.state.carAccessibility=
charging) )=>cancelled=True
```

```
	//Penalty for not picking up a reserved car

	(minus[endTime,initialTime]=60 and car.state.engine=off) => penaltyRes=1 and
(car.state.carAccessibility= available or car.state.carAccessibility= charging) and cancelled=True

	initialTime<endTime

	initialTime>=0

	date>0

}


sig Password{}

sig PaymentInfo{}

sig User{

	password: one Password,

	paymentInfo: one PaymentInfo,

	lastRide: lone Ride,

	blocked: Bool,

	position: one Position

}{ blocked=True <=> (lastRide.paid=False and lastRide.ended=True)

}




sig Ride{

	passengers: Int,

	price: Int,

	finPrice: Int,

	finCharge: Int,

	penalty: Bool,

	discountPass: Bool,

	discountPlug: Bool,

	discountBat: Bool,
            reservation: one Reservation,

            paid: Bool,

            started: Bool,

            ended:Bool

} {    finCharge>=0 and finCharge<=reservation.car.batteryLevel

            passengers>=0 and passengers<=3

            price>=0

            finPrice>=0

            started=False=> finCharge=reservation.car.batteryLevel and price=0 and finPrice=0

            finCharge=0 => ended=True and reservation.car.state.engine=off and  reservation.car.state.carAction=
still

            //Discount and penalty effect on final price

            finPrice=price <=> (discountPass=False and discountPlug=False and discountBat=False and
penalty=False) or (discountPass=True and discountPlug=True and discountBat=False and penalty=True)

            finPrice>price <=> (discountPass=False and discountPlug=False and discountBat=False and
penalty=True) or (discountPass=False and discountPlug=True and discountBat=False and penalty=True) or
(discountPass=True and discountPlug=False and discountBat=False and penalty=True)

            //Discount of 10% if there are at least 2 other passengers

            discountPass=True <=> passengers>=2 and started=True

            //Discount of 20% if rental ended and battery almost 50%

            discountBat=True <=>  ended=True and finCharge>=50

            //Discount of 30% if car plugged into power grid

            discountPlug=True <=>  ended=True and  reservation.car.state.carAccessibility=charging

            //Rental can end only if it has started

            ended=True=> started=True

            //Ride starts when the engine starts

            reservation.car.state.engine=on and reservation.car.state.carAccessibility= reserved=>started=True

            //Car is reserved only during the ride and the reservation

            (started=True and ended=False) or (started=False and
(minus[reservation.endTime,reservation.initialTime]<60) ) <=> reservation.car.state.carAccessibility=reserved
```

```
		//Ride end when engine is off, car stops and doors open
		ended=True => (reservation.car.state.engine=off and reservation.car.state.carAction=still and 
reservation.car.state.door=opened)
		//Ride is linked only to a valid(not cancelled) Reservation
		reservation.cancelled=False
		//Pay after the ride
		 paid=True=> ended=True
		//Possible states after the endig  of a ride
		ended=True => reservation.car.state.carAccessibility= available or reservation.car.state.carAccessibility= 
charging or (reservation.car.state.carAccessibility= outOfService and finCharge<20)
		//outOfServiceCar
		reservation.car.state.carAccessibility = outOfService => (reservation.car.batteryLevel <=20 or finCharge 
<= 20)
		//don'tBeFinalUntilIsEnded
		ended=False => finCharge=reservation.car.batteryLevel
}


sig SafeArea{
		position: seq Position
}{		#position.elems>0}


sig SpecialSafeArea extends SafeArea{
		freeParks: Int,
} {		freeParks>=0
		freeParks <=#position.elems
}
sig Operator{}
//***FACTS***//
//A State must belong to a Car
```

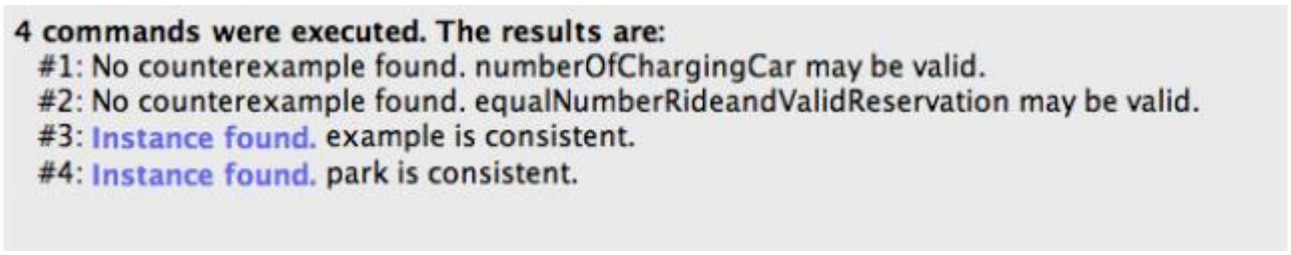
```
fact stateHasCar{

all s: State {some c: Car | s=c.state}}

//A Password must belong to a User

fact passwordHasUser{

all p:Password{ one u:User | p= u.password}}

//A PaymentInfo must belong to one User

fact paymentHasUser{

all p: PaymentInfo { one u: User | p=u.paymentInfo}}

//Users have different passwords

fact passwordsAreUnique{

all u1, u2: User | u1!=u2 => u1.password!= u2.password

}

//A User can do a single reservation at a time

fact singleReservation{

all r1,r2: Reservation| r1!= r2 and r1.user=r2.user and r1.date=r2.date => (r2.initialTime!=r1.initialTime or
r2.initialTime>=r1.endTime)}


//Rides have different Reservation

fact singleReservationforRide{

all rd1,rd2: Ride | rd1!=rd2 => rd1.reservation !=rd2.reservation}

//A Car can belong only to one Reservation in a specific initial time and date

fact uniqueCar{

all r1,r2: Reservation | r1!=r2 and r1.date=r2.date and (r1.initialTime= r2.initialTime or
r2.initialTime<=r1.endTime) => r1.car !=r2.car }


//-----------------------

//Meaning of lastRide

fact defLastRide{
```

```
all u:User { no ri: Ride | ri.reservation.user=u and ri.ended=True and
(ri.reservation.date>=u.lastRide.reservation.date or (ri.reservation.date=u.lastRide.reservation.date and
ri.reservation.endTime>=u.lastRide.reservation.endTime))}}

//Users have different lastRide

fact  uniqueLastRide{

all u1, u2: User | u1!=u2 => u1.lastRide!= u2.lastRide}

//Each"valid" Reservation must have a corresponding Ride

fact validResHaveRide{

all r:Reservation | r.cancelled=False => {one ri:Ride | ri.reservation=r}}

//LastRide is linked to a Reservation of the same User

fact defbisLastRide{

all u: User {one r:Reservation |u.lastRide.reservation=r and r.user=u}}

//A blocked User can't make new Reservations

fact noReservationFromBlockedUser{

no r:Reservation | r.user.blocked=True and (r.date>r.user.lastRide.reservation.date or
(r.date=r.user.lastRide.reservation.date and r.initialTime>r.user.lastRide.reservation.initialTime))}

// No SafeAreas with same Position

fact uniquePositionSa{

all s1,s2: SafeArea | s1!=s2 => (s1.position.elems & s2.position.elems)=none }

//No parks with the same position

fact uniquePositionSaParks{

all s: SafeArea, i,k:s.position.inds| i!=k => s.position[i]!=s.position[k]}

//No Car with same position

fact carInDifferentPosition{

all c1,c2:Car | c1!=c2 => c1.position!=c2.position}

//Car can be recharged in special safe area

fact carInCharge{

all c: Car, ssa: SpecialSafeArea | c.state.carAccessibility= charging => c.position in ssa.parkPos}

//30% penalty for uncorrect behaviour
```

```
fact penaltyRide{

all c:Car, ssa: SpecialSafeArea |{some r:Ride | r.penalty=True <=> (c in r.drivenCar and r.ended=True and
r.reservation.car.state.carAccessibility=outOfService) and ( r.finCharge<20 or c.position in ssa.parkPos)}}

//Constraint car without Reservation

fact carReserved{

all c:Car | c.state.carAccessibility=reserved => { some r:Reservation | r.car=c}}

//An outofService Car can be maintained only by one operator

fact OperatorHasUniqueCar{

all c1,c2: Car | c1!=c2 => c1.maintainedBy!=c2.maintainedBy}

//A car cannot have a reservation if the other hasn't ended

fact carWithatmostoneRidenotended{

all r1,r2:Ride |r1!=r2 and r1.reservation.car =r2.reservation.car => r1.ended=True or r2.ended=True}


//-----Functions---//

fun parkPos[ ssa: SpecialSafeArea] : set Position{

ssa.position.elems}

fun drivenCar[ r:Ride] :Car{

r.reservation.car}


fun notCancelledReserv : set Bool{

 False  & Reservation.cancelled }

fun chargingCar: set Accessibility{

charging & Car.state.carAccessibility }

fun carInSpecialSa: set Position{

Car.position & SpecialSafeArea.position.elems}

//---Assertion--//

assert equalNumberRideandValidReservation{

#Ride = #notCancelledReserv}
```

```
assert numberOfChargingCar{

#chargingCar <= #carInSpecialSa}

//***PREDICATES**//

pred SpecialSafeArea.park[ sa: SpecialSafeArea, c: Car]{

c.position=this.position.last

sa.position = this.position.delete[#this.position]

}

pred example{

#User>1

#Operator>0

#Car >0

#(charging & Car.state.carAccessibility)>0

#(True & Ride.ended)>0

#SpecialSafeArea>0

#Reservation>1

}


check numberOfChargingCar

check equalNumberRideandValidReservation

run example for 8 Int

run park for 8 Int
```

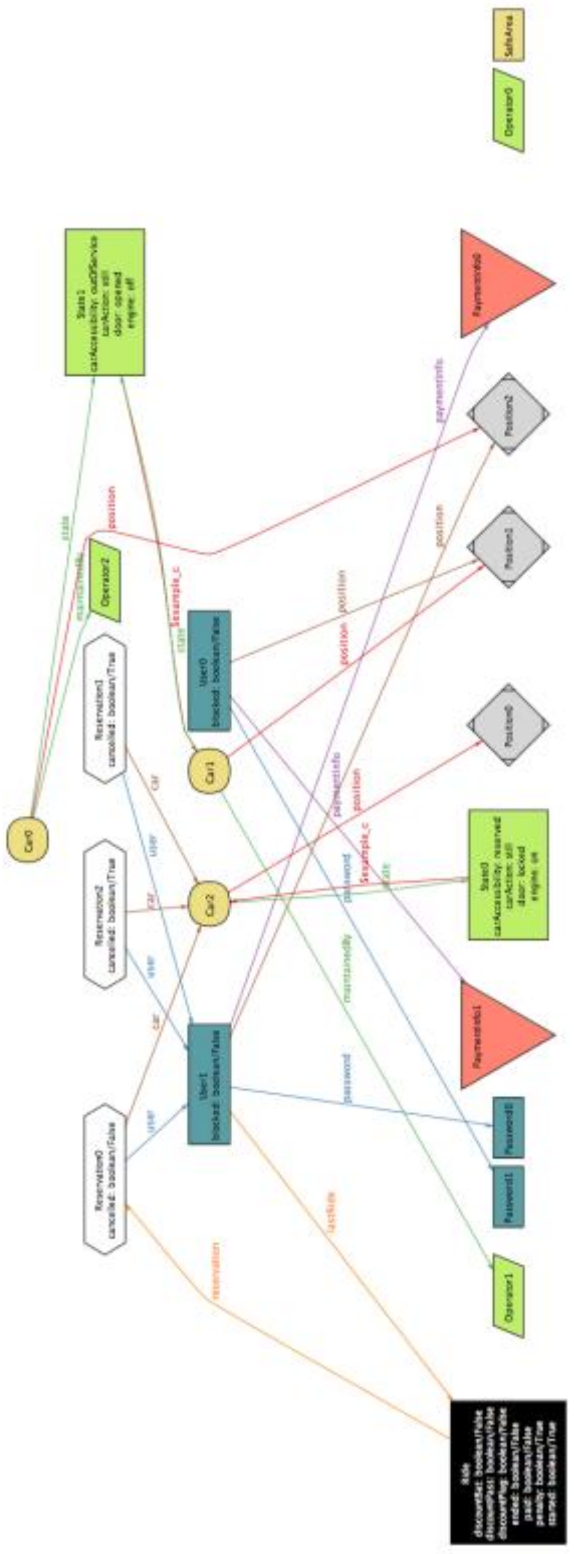
4.2: Alloy Results



```
4 commands were executed. The results are:
  #1: No counterexample found. numberOfChargingCar may be valid.
  #2: No counterexample found. equalNumberRideandValidReservation may be valid.
  #3: Instance found. example is consistent.
  #4: Instance found. park is consistent.
```

4.3: World generated

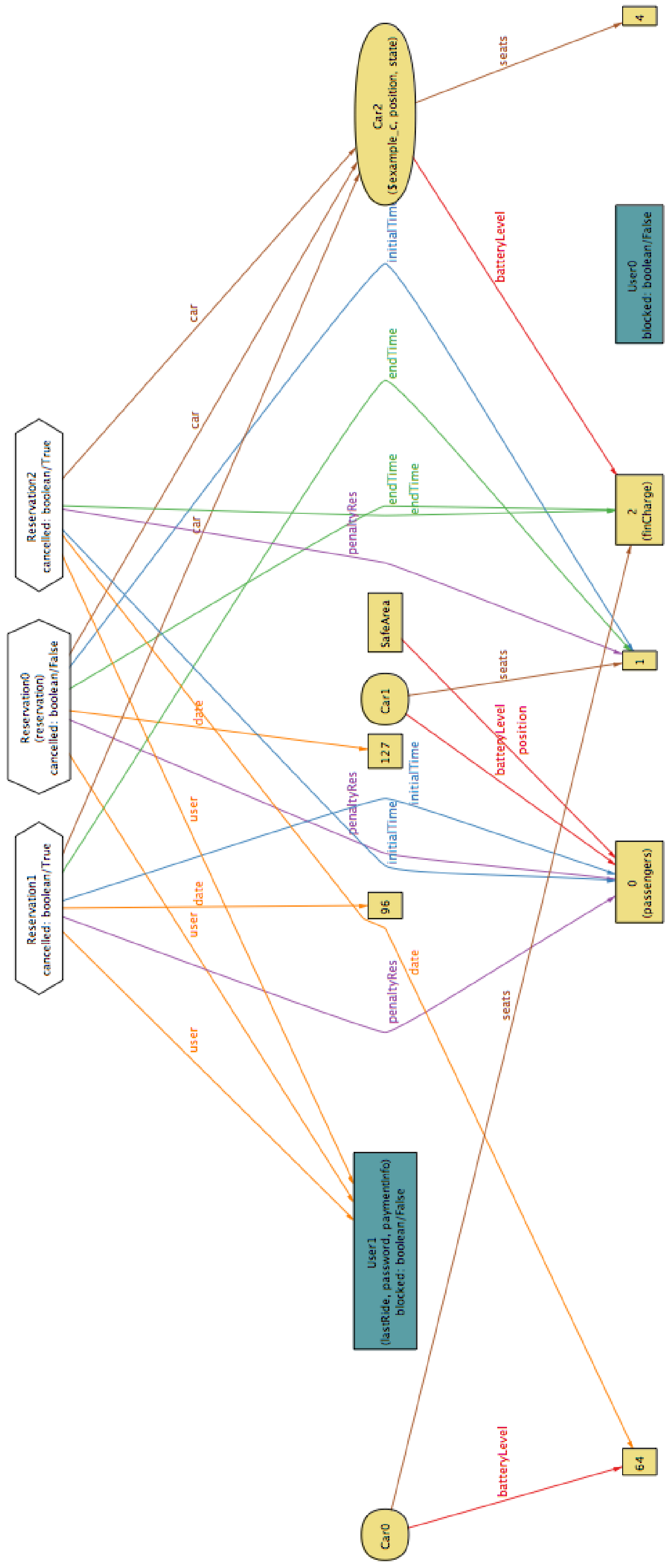
(Better images of the alloy instances can be found in the repo of the project)



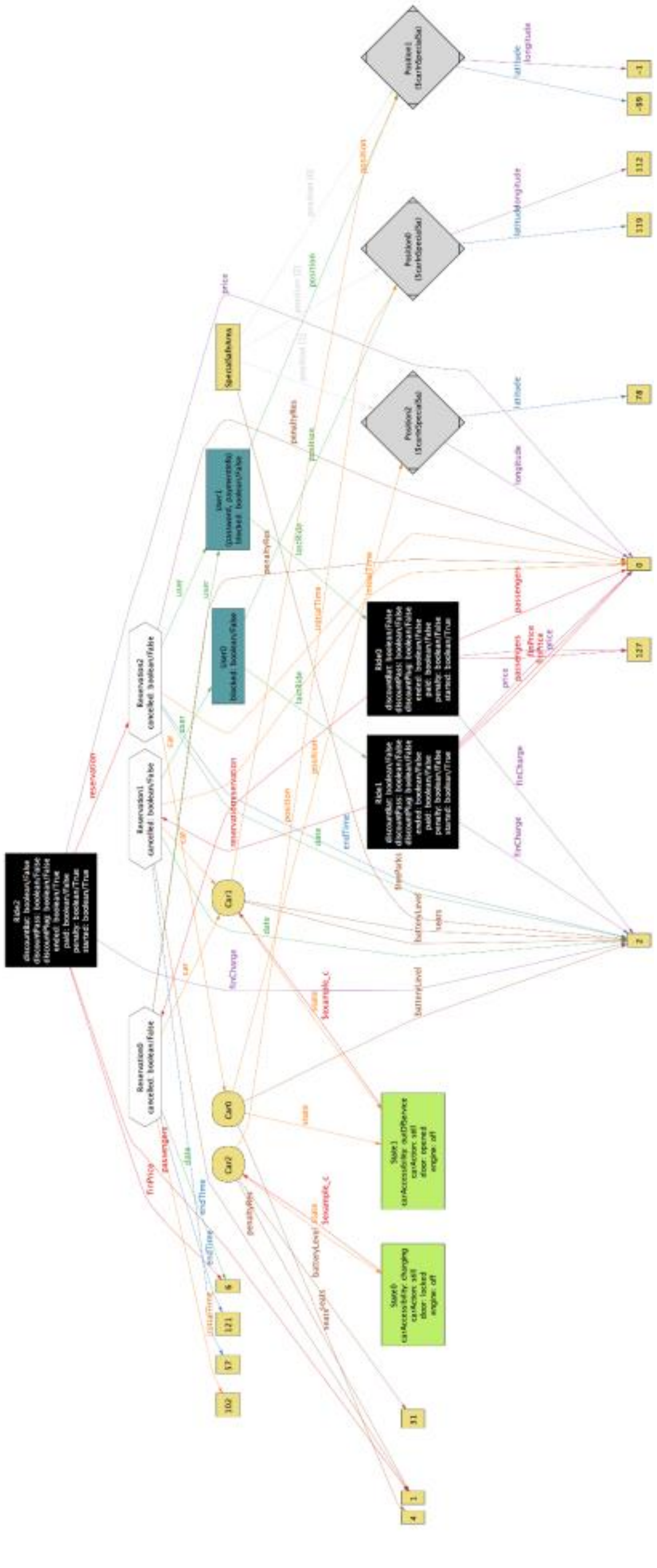
This alloy instance shows for example that User1 has started his ride, for this reason it’s marked has his last ride and linked to the reservation which of course corresponds to a reservation (not cancelled) made by the same User1.

Car2 is linked to different Reservations but they occur in different date/time, these relations have been hidden in this image, only to make it more readable.
Car0 instead is out of service and for this reason, it’s maintained by Operator 2.

All the hidden relations of the previous instance, are instead shown in the following image:


Reservation2
cancelled: boolean/True
Reservation0
(reservation)
cancelled: boolean/False
Reservation1
cancelled: boolean/True
car
car
car
user
user
user
date
date
date
initialTime
initialTime
initialTime
endTime
endTime
endTime
penaltyRes
penaltyRes
penaltyRes
Car2
($example_c, position, state)
SafeArea
Car1
127
96
User1
(lastRide, password, paymentInfo)
blocked: boolean/False
Car0
seats
seats
seats
batteryLevel
batteryLevel
batteryLevel
position
4
User0
blocked: boolean/False
2
(finCharge)
1
0
(passengers)
64

This shows, in fact, that User1 is linked with the 3 Reservations but, these occur in different time because have a different date.


Ride2
Ride1
Ride0
Reservation0
Reservation1
Reservation2
Car0
Car1
Car2
User0
User1
State0
State1
Position0
Position1
Position2
SpecialSafeArea

Car2 is charging while It's parked into a special safe area and User1 is linked to Reservation2 (that has Car0) which is linked to Ride2 that has the penalty attribute equals to True. In fact User got the penalty because Car is parked into a special safe area with a finCharge equals to 2, but without plugging the Car into the power grid, for this reason Car0 is marked as out of service.

4.4: Used softwares

During the development of the RASD document were used the following programs:

- Microsoft Office Word
- Balsamiq Mockups 3 (for the graphical interface prototypes)
- Alloy 4.2 (for Alloy Modeling)
- Draw.io (for diagrams and uml models)
- Trello (as a notice board for the team)

4.5: Hours of work

Carmen Barletta
26/10: 2h
27/10: 1h30m
29/10: 2h
31/10: 5h
1/11: 5h
4/11: 1h
6/11: 4h
7/11: 3h
8/11: 3h
9/11: 1h30min
10/11: 2h
11/11: 3h

Francesco Matteo Bagna
26/10: 2h
27/10: 1h30m
28/10: 2h
31/10: 5h
1/11: 5h

5/11: 3h
7/11: 3h
8/11: 3h
9/11: 1h30min
10/11: 2h
11/11: 3h

4.6: Changelog

Version 2.0:
-Updating the Sequence Diagram of Use Case 5
-Updating the Sequence Diagram of Use Case 8
-Updating the Mockups
-Inserting the administrator
-Correcting the traceability matrix in which G10 was missing