Requirements Analysis and Specifications Document (RASD)

Simone Mosciatti & Sara Zanzottera

November 29, 2016

Contents

1 Introduction

In this section we are providing an overview of the PowerEnJoy Project, we are highlighting what the product does, what functions it provides, what constrains it has and what assumption we made during the design process.

1.1 Purpose

The purpose of this document is to analyze the requirements for the project and provide detailed specifications for PowerEnJoy, a digital management system for a car sharing service that features only electric cars.

1.2 Scope

PowerEnJoy is a car sharing platform focused exclusively on electric cars. It aims to improve the mobility inside the city in a eco-friendly way. To achieve its goal, it will encourage people to be environment aware drivers.

1.3 Definitions, acronyms, abbreviations

User A customer of PowerEnJoy using the service.

Staff Operator An employee of PowerEnJoy which takes care of the cars.

Ride The action of getting onboard of a PowerEnJoy car, start its engine, drive to destination and park.

Running Time The time an user spends using the PowerEnJoy service.

Issue Any problem a car may incur in, or a user may face while using the service.

Nearby Cars Cars located within a maximum distance to a specific position.

Nearby Issues Issues that are affecting cars close to a specific position.

Booking (Reservation) The act to reserve a car for a limited amount of time for future use by a user.

Reservation's maximum time The maximun amount of time a car can be reserved.

Driver Whoever is driving a regularly booked PowerEnJoy car.

Passenger Whoever is in inside a PowerEnJoy car but is not the driver.

Driving License The state's issued driving license of the user.

Notification A form of comunication where the user is actively notified of some event.

Issue Report An incoming notification that states a car incurred in an issue.

Fine A fine issued by the local law enforcing officers to a user while driving a PowerEnJoy car.

Pending Bills Bills that an user still need to pay to PowerEnJoy .

Safe Area An parking area, predefined by the company, where is possible to safely park the cars of the PowerEnJoy fleet.

Battery Charge The amount of charge that is kept inside the car's battery.

Charging Station Dedicated areas where is possible to plug the PowerEnJoy cars to charge their batteries.

Car's Onboard System The controll system of the car that is able to exchange data with the central system and to relevate operation parameters.

Customer's App An implementation of the system frontend tailored to the need of the customers.

Operator's App An implementation of the system frontend tailored to the need of the staff.

Central System The central system for PowerEnJoy . All the command and all the data are streamed, analyzed and used here.

Credentials Pair {Username, Password} necessary to access the PowerEnJoy system.

GPS : Global Positioning System is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

System's Frontend The interface provided to the user of the PowerEnJoy system.

System's Backend The whole technical infrastructure necessary to PowerEnJoy .

1.4 Motivation

In this small subsection we explain the rationale behind some choices that we made in the subsequent sections of the document.

The client server architecture

Even if the choice of a specific architecture should be postponed in successive development phases, for sake of simplicity and clarity we have already commit ourselves to a client server architecture. Given the kind of application we are developing, a client server architecture is one of the most basic choice we could made.

App

In the whole document we will refer to an app. Again, this choice should be postponed to successive development phases; however, given we already set a Client/Server architecture, we convey that considering a mobile app to provide the service to customers is a reasonable choice.

Using the term "App" we do not mean an application for a specific mobile vendor, but a broader interface to the PowerEnJoy services. Such interface could be an iOS app, an Adroid app, a Windows Phone app, a responsive HTML page or even all of them .

2 Overall Description

PowerEnJoy is a digital management system for car sharing that exclusively employs electric cars to provide its service. The system provides all the functionalities normally provided by a car sharing service, like registering to the service, find the location of nearby available cars, reserve cars up to a short amount of time (namely one hour), unlock the chosen car once found, ride it and then park it in a safe area, when it will be automatically locked and the fee paid.

In addition, the system gives bonuses and penalities in term of discounts or overprices depending on the behavior of the user, in order to promote virtuous behaviors. Some examples are:

- a 10% discount for users who brings at least two other passangers with him during the ride.
- a 20% discount for users who leaves the car with at least 50% of the battery still full.
- a 30% discount if the car is plugged to a charging station at the end of the ride.
- a 30% overprice if the car is left more than 3km from the nearest charging station or with less than 20% of battery charge left.

(Note that these percentages, even if enforced through the whole document, are indicative customizable by the owners of the service, according to the systems' performance on the long run).

PowerEnJoy is therefore a inherently distributed system, based on a central server interactions with many distributed nodes. In detail the system can be divided into four main parts:

- a public app, used by customers to access the service
- a centralized backend that provides the service
- the cars' onboard system, that communicates only with the centralized backend

- a reserved fronted, used exclusively by the staff members to better organize their job

All these four components will be examined in more detail in the subsequent sections of the document.

2.1 Product Perspective

PowerEnJoy provide users the ability to rent electric cars for a limited amount of time. The customer's app allows users to look for available cars, to reserve cars for up to one hour, to lock the vehicle once the run is over and to pay fees and, in case, fines.

The staff's reserved app allows them to locate cars which need to be plugged or recharged in-place, which cars need to be brought to a safe parking area, or if some users need help for some technical issue with their car. It also provides them with the ability of registering fines sent to the company by local police officers, as for the system to assign them to the correct user.

The car's onboard system monitors all the events that happen to the car and react according to the instructions sent by the central backend. It can unlock the car when the backend asks so, monitor the internal state of the car, find out the coordinates of the car using an integrated GPS, and tell the backend any kind of data it may ask for.

The backend part takes care of providing all the required data to both frontends. It also tracks the position and the statuses of every single car, the duration of each ride and charges users accordingly.

2.2 Product Functions

PowerEnJoy is provided to the user via a mobile application. A registered user can log in into its account, while a new user is prompted to register and then to log in.

After the user logged in they can see the position of each car in the city or in a more strictly bounded area, for example, within ten minutes by foot. Once the user has decided which car they wants to use, they can book the car for one hour. After the car has been booked and the user reaches it, the system automatically unlocks the car for the user.

The system starts charging the user when the engine starts running and stops once the car is parked, the user exited and locked the car. Several forms of discounts and overfees are applied depending on the behaviour of the user when riding and especially when parking the car.

2.3 Parallel Operation

The system is capable to operate and serve users concurrently. This feature is strictly necessary as the system is inherently distributed.

2.4 Actual Systems

As for now there exits other car sharing services operating into the city, but none of them involves purely electric cars: or they rent traditional cars, or they use hybrid models, that have different needs comparing to purely electrical vehicles.

Because of these considerations, we decided to not interact with these different car sharing systems and offer a completely separated service, specifically focused on leveraging the specific advantages and issues that electric cars offer.

2.5 Constraints

Platform Contraints

The system has some platform constraints coming from the device that is running the app. These are:

1. The app should be small enough to fit the memory of ideally all smartphones.
2. The app should be thin enough to fit the computational capabilities of ideally all smartphones.
3. The app should send and receive only small quantity of data through the smartphone's internet connection, in order to not add significant connectivity costs for the users.
4. The app should be able to get information from the smartphones' GPS.

Privacy regulations

The system should complain to the most recent privacy regulations in managing the data that the user are generating, thus ensuring:

1. The user must be aware that its data is being recorded by the system, including their GPS position.
2. The user must be able to see which data the system collected about him and, in case, to delete it (even if this may require the user to unsubscribe from the service).
3. No third parties should be able to gain access on users' data.
4. Users' data must be protected from third-parties attacks.
5. Users' data should not be possible to obtain by reverse-engineering the behavior of the system.

2.6 Actors and stakeholders

Actors

The actors involved in our system are mainly **final users**, that is, people who are renting cars. Indeed the system needs support from a **technical staff** that mantains the digital infrastructure, manages the fines that local autorities emits related to PowerEnJoy cars and takes care of the cars themselves in some special situations, like:

- Cars left far from the power grid with an empty battery, that needs to be charged in-place
- Cars left far from a safe parking area, that needs to be brought back to one of them
- Cars not plugged to the power grid, that needs to be plugged
- Cars which has been reported by the user or by the onboard system itself having some technical issues
- Fines received by the company that needs to be assigned to the correct user

Stakeholders

Other than the company behind PowerEnJoy, the main stakeholder for our system are **users** themselves, who requires a easy-to-use service of electric cars sharing. Other stakeholders are the **city governement**, who supports our service as a way to improve sustainable mobility inside the city and takes care of the road infrastructures, and the **energy companies**, which provide power for the cars.

2.7 Text Assumptions

Assumptions on the final user

1. The final user has reached the legal age and has a driving licence.
2. The final user has a smartphone with the PowerEnJoy app installed, a GPS and Internet connectivity.

Assumptions on the car system

1. Each car is provided with internet connectivity and capable to send and receive data to the main server.
2. Each car is provided of a GPS of reasonable accurancy.
3. Each car can capture all these events correctly:
    - the ignition of the engine

- the status of the engine
- the users getting into the car
- the users exiting the car
- the number of passengers.
- the user plugging the battery to the power grid.
- the locked status of a car

4. Each car is able to monitor the residual charge of its battery.
5. Each car is provided with a device that can scan and read a driving licence.
6. The system is able to prevent the ignition of each car's engine remotely.
7. The system is able to switch off every car's engine remotely.
8. The system is able to unlock each car remotely.
9. The system is able to lock each car remotely.
10. The total running time of an ride is counted from either when the user switches the engine on.
11. The total running time of an ride ends when the user locks the car.

Statuses assumptions

The cars are assumed to have some internal, discrete statuses describing their conditions. These are:

- Availability Status: describe if the car is interacting with some users of if it is available for rent. Possible values are:
  - Available
  - Booked
  - Unlocked
  - Running
  - Parked
  - Not Available
- Charging Status: describe if the car is connected to the power grid or not. Boolean values (True or False)
- Exception Status: describe if the car has some issues. Possible values are:
  - No Issue

  - Out Of Power
  - Unsafely Parked
  - Out of City Boudaries
  - Technical issue
  - Mechanical issue
  - Other issue

Here we list some text assumptions about these parameters.

- The Availability status can change to Booked only if it was Available.
- The Availability status can change to Unlocked only if it was Booked.
- The Availability status can change to Running only if it was Unlocked.
- The Availability status can change to Parked only if it was Running.
- The Availability status can change to Available only if it was Parked or Not Available.
- The Availability status of a car with less than 20% of residual charge is Not Available regardless of the status it was supposed to have, and its Exception status is set to Out Of Power.
- The Availability status of a car parked outside a safe area is Not Available and its Exception status is set to Unsafely Parked.
- The Charging status of a car can be True only if the car is plugged to the power grid.
- If Charging status is True, the charge must increase in time. Otherwise, the Availability status must be set to Not Available and the Exception status must be set to Mechanical Issue.
- The Charging status of a car cannot be set to True if the Availability status of the car is Running. Otherwise, the Availability status must be set to Not Available and the Exception status must be set to Mechanical Issue.
- If the Availability status of the car is different from Not Available, the Exception status must be No Issue.
- If the Availability status is Not Available, the Exception status must be different from No Issue.
- If the Availability status of a car is different from Available, the car is associated with one single user and that user cannot change until the Availability status goes back to Available.

- If the Availability status of a car is Available, the car cannot be associated with any user.
- Each user can book up to one car at a time, and cannot book any other vehicle until they finished the ride.

External Services

The system depends on external services that are served by third-parties. These services are used to:

- Charging fees to the users.
- Determinate if the car is been parked in a safe area.
- Determinate the distance from the closest charging station.
- Determinate the distance between two different GPS signals.
- Scan and read driving licences' informations.
- Ensure validity of Identity IDs and if they match with personal informations provided by the user and the provided driving licence's data.
- Regularly check and maintain the cars.

The decision to rely on external partners to run these functions has been made since those services are not the core business for PowerEnJoy, and other external companies has already found very efficient and cost-effective solution to them.

2.8 Domain Properties

The following domains properties are assumed during the development of the project.

- Each user has only one unique Identity ID related to them.
- Each user has only one unique driving licence related to them.
- A driving license will not expire nor be revoked for the whole time the user is registered.
- Each account uniquely identify a physical person: account cannot be shared or used by different people, even in different moments.
- It is possible for the staff to bring back cars from not-safe areas into safe areas.
- A car cannot change its position while the engine is switched off.
- A car's engine cannot be switched on while the car is locked (the contrary may happen).

- A car's residual charge does not increase if it is not plugged to the power grid.
- A car's residual charge does not decrease if it is not running, i.e. the engine is off.
- Users are able to plug and unplug the car from the charging stations.
- It is possible to determine whether the car is parked in a safe area or not, at any moment.
- It is possible to determinate the actual level of charge of the battery with neglitable error.
- It is possible to determinate how many people are in each car at any moment.
- Each and any message that the car sends to the main server arrives and is processed.
- It is possible to read a driving licence's informations.
- Once the car is plugged to the charging station, the battery starts to increase its charge.
- Once the car's engine is switched on, the battery starts to decrease its charge.

2.9 Reference Documents

- *Assignments AA 2016-2017.pdf* (Assignments document given by the teacher)
- *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications*
- Other Sample documents:
  - *RASD sample from Oct. 20 lecture.pdf*
  - *Libra: An Economy-Driven Cluster Scheduler Software Requirements Specification*

3 Specific Requirements

In this section we are going to illustrate the specific requirement of PowerEnJoy .

We analyze the goals that the application should fullfill, then moving on explicit functional and not functional requirements.

3.1 Goals

REGISTRATION Users can register to PowerEnJoy .

LOGIN Users can login to PowerEnJoy .

LOOKUP Users can find cars nearby a given position, it could be its position or a point in the map.

BOOK Users can book a car for a short amount of time.

UNLOCK When users are in proximity of the car they booked, the system can unlock it.

RIDE Users can drive to their destination.

SAFE_AREAS Users can locate safe parking areas.

UNSAFE_PARKING The system must react to an unsafe parking.

POWER_STATIONS Users can locate charging stations.

CHARGE At the end of the ride, users are charged a fee.

PAYMENTS Users can pay bills through the app.

FIND_ISSUES The staff can locate cars that need their intervention.

SUPPORT The staff can identify and solve car's issues.

FINES The system can manage fines sent to the company by local authorities.

3.2 Functional Requirements

We are deriving our functional requirements from the goals we listed in the previous section, under the hypotesis that all domain and text assumptions always hold.

REGISTRATION Users can register to PowerEnJoy .

- **REG1** The system must be able to create new accounts with data provided through the client's application.
- **REG2** The system must validate the data the user provided to create a new account.

REG3 The system must check whether the user has no other accounts with a matching Identity ID or driving licence (see Text Assumptions).

REG4 The system must ensure that, at registering time, users are aware of how their data are going to be processed and stored, and how the system complies the current privacy regulations.

REG5 The system must be able to deal with situations where the user cannot open a new account.

LOGIN Users can login to PowerEnJoy .

LOG1 The system must check whether a username-password pair corresponds to an existing account.

LOG2 The system must inform users whether the given username-password pair does not corresponds to any existing account.

LOG3 The system must login users who provides a correct username and password pair

LOG4 The system must prevent not-logged users from using all other app's functions, except for registering and login.

LOOKUP Users can find cars nearby a given position, it could be its position or a point in the map.

LOOK1 The system must send to each user a list of the available cars that locates within a specified walking distance from the position specified, when requested.

LOOK2 Users can change the maximum walking distance required to reach a car, and the system is able to use this parameter when retrieving the list of nearby available cars.

LOOK3 The system is able to change the ranking metric of the nearby cars according to users' preferences.

LOOK4 The app must show to users the position of all the retrieved available cars on an interactive map.

LOOK5 The app must show to users additional informations on each car when requested.

LOOK6 The app must show to users the shortest path to reach their car by foot on an interactive map, and to show them how long the walk is approximately going to take.

BOOK Users can book a car for a short amount of time.

BOOK1 Users can select the car they want to book.

BOOK2 The system must change the status of a car from Available to Booked when a user books it.

BOOK3 A booked car cannot be booked again until its status goes back to Available.

BOOK4 A booked car must be associated with the user who booked it.

BOOK5 Users must be associated only with the car they booked.

BOOK6 If a user does not unlock the car after a short period of time (one hour), a small fee is charged to them and the car's status is set back to Available.

UNLOCK When users are in proximity of the car they booked, the system can unlock it.

UNLK1 The system is able to determine which user booked which car.

UNLK2 The system is able to determine when the user is near to the car they booked.

UNLK3 The system is able to unlock the correct car remotely.

UNLK4 The system must unlock the car the user booked when the user and the car shows approximately the same position on the map.

UNLK5 The system must notify the user their car has been unlocked.

UNLK6 The system must change automatically the status of the car from Booked to Unlocked when a car is unlocked.

UNLK7 Users can lock back an unlocked car from the app before starting the ride. In this case the car's status gets back to Booked and the timer restarts from the point it stopped. In order to perform this operation, no passengers can be inside the car.

RIDE Users can drive to their destination.

RIDE1 Once the engine is turned on, the system must automatically change the car's Availability status to Running.

RIDE2 The system must determine whether the scanned driving licence is associated with the user who booked the car.

RIDE3 The system must prevent the ignition of the car's engine if the scanned driving licence is not the driving licence of the user who rented the car.

RIDE4 The car's onboard system can show to users some informations:

- Residual charge.
- How long and how far the car can approximately run with that charge.
- Distance covered.
- Time spent onboard.
- Fee as calculated at that moment.
- Nearby safe areas where to park the car.
- Nearby power station where to plug the car.

RIDE5 At the end of the ride, when all passengers exit, the car can be locked by the user with a button on the app's interface.

SAFE_AREAS Users can locate safe parking areas.

SAFE1 The system must send to the car all informations about safe areas near its position when requested.

SAFE2 The car can show safe areas' location to the driver.

SAFE3 The app must confirm the driver they parked in a safe area.

SAFE4 Once the car's engine stops in a safe area, the system must automatically change the Availability status of the car from Running to Parked.

UNSAFE_PARKING The system must react to an unsafe parking.

UNSF1 If the car's engine stops outside a safe area, the car should notify the user that they are parking outside a safe area and they are going to be fined if they leave the car.

UNSF2 If a user leaves the car while parked outside a safe area (regardeless of the status of the engine) the car's Availability becomes Not Available, its Exception status is set to Unsafely Parked, and the user is fined.

UNSF3 If a car is in Unsafely Parked status with the engine on, its engine is switched off remotely by the system.

UNSF4 If a car is in Unsafely Parked status, it gets automatically locked by the system.

POWER_STATIONS Users can locate and use charging stations correctly.

PWRS1 The system must send to the car all informations about power stations near its position, when requested.

PWRS2 The car must show power stations' location to the driver when requested.

PWRS3 The system must be aware of which car gets plugged to the power grid and by which user, and must store this information for later use.

PWRS4 The system must confirm the driver that he plugged the car correctly.

CHARGE At the end of the ride, the user is charged a fee.

FEE1 The fee is calculated on the distance the car covered during the ride and the time spent to cover that distance.

FEE2 Carrying at least two passengers, driver excluded, implies a discount of 10% of the total fee.

FEE3 Leaving the car with at least 50% of battery charge implies a discount of 20% of the total fee.

FEE4 Plugging the car to a charging station implies a discount of 30% of the total fee. Users must plug the car before locking it.

FEE5 Leaving the car with less than 20% of battery charge implies a 30% overprice.

FEE6 Leaving the car more than 3KM from the closest charging station implies a 30% overprice.

FEE7 Users receive the bill immediately when its car gets locked.

PAYMENTS Users can pay bills through the app.

PAY1 Users can set a preferred payment method for their account through the app.

PAY2 Under each bill there must be a button that allows users to pay directly with their preferred payment method.

PAY3 The app must show a payment confirmation after each payment.

PAY4 If a payment fails, the app must notify the user and prevent they from using the service until the last payment is completed successfully.

FIND_ISSUES The staff can locate cars that need their intervention.

ISS1 The system must be able to create a list of all the cars that are in Not Available status but not yet taken by any operator, and to send them to all the frontend apps that requests so.

ISS2 The staff's app must show to the operator all informations about all cars that are in Not Available status but not yet taken by any operator.

ISS3 The staff's app can filter and sort the cars basing on their distance from the user's location, time of Non Availability and issue type (the Exception status).

ISS4 The staff's app must give directions to the user to reach the selected car if requested.

SUPPORT The staff can identify and solve car's issues.

SUP1 The staff's app must allow operators to take in charge a certain issue.

SUP2 The staff's app must allow operators to change the status of the car they are in charge of back to Available.

SUP3 When an operator changes the status of their car back to Available, the system must re-check all the car's parameters and then confirm the status or brings it back to Not Available. In the latter case, the system informs the operator involved and another, different issue is generated.

SUP4 The staff's app must allow operators to change the Exception status of the car they are in charge of.

SUP5 The staff's app must allow operators to unlock and lock the car they are in charge of.

SUP6 The staff's app must allow operators to switch their car on, switch off, plug to the power grid, eventually open the car's bonnet or perform any other operation that may be required to solve an issue.

SUP7 The staff's app must allow operators to give up over an issue if they realize they are not able to solve it.

FINES The system can manage fines sent to the company by local authorities.

FINE1 The staff's app can receive and send to the system fine informations.

FINE2 The system must be able to find out which user was driving a specific car in a specific moment.

FINE3 The system must associate each fine to a responsible user.

FINE4 The app must show fine's information only to the correct user.

FINE5 The app must allow users to pay the fine directly through the app.

FINE6 The app must prevent users from using the service until they have paid all their fines.

3.3 Non Functional Requirements

Technical Non Functional Requirement

Latency: 99% of request should receive an answer in less than 300ms

Availability: 99%

Reliability: 99%

Since our system's availability is not the most critical aspect of the system, a little downtime is manageable, especially if it provides a faster development cycle and helps containing the costs of the overall project maintanance.

However such a tradeoff demand an extensive monitoring of the system and auto rollback capability in case of a bad deploy.

Client's App Interface

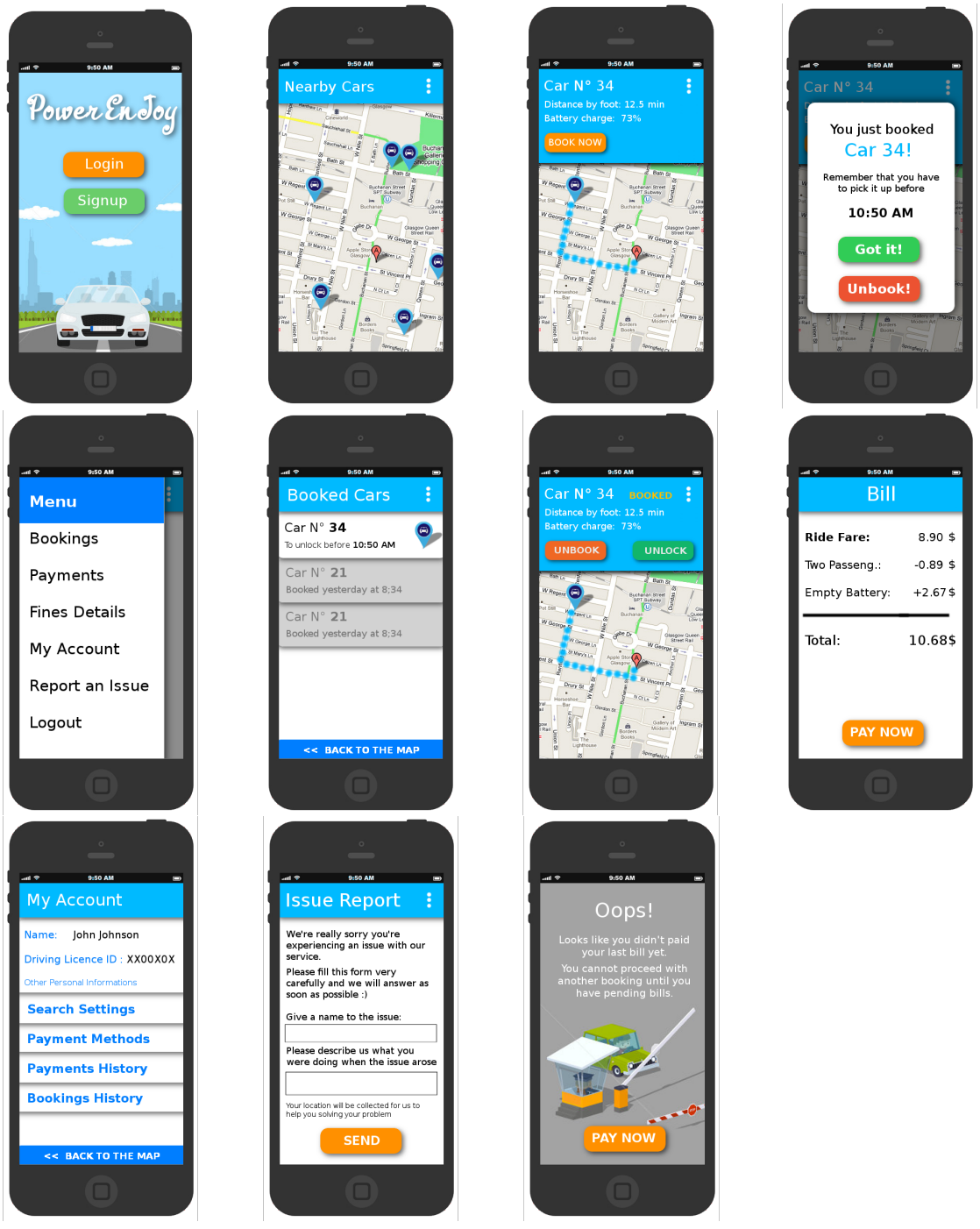

Figure 1: Mockup of the client's app main interfaces

Car's Onboard System

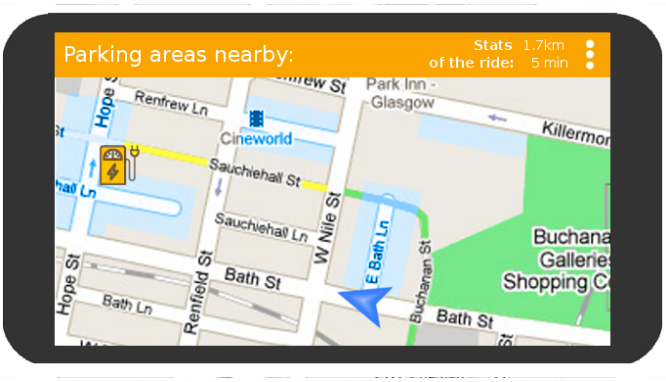

Figure 2: Mockup of the car's onboard system as it should look while driving

Staff's App Interface

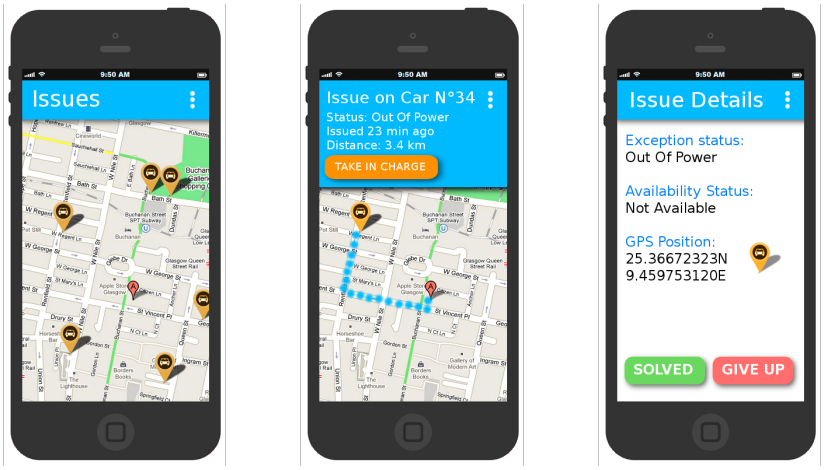

Figure 3: Mockup of the staff's operators reserved interfaces

4 Scenarios, Use Cases and UML Diagrams

4.1 Scenarios

Scenario 1

John has been abroad for a business trip, and now is at the airport, deciding how to go home. He is a user of PowerEnJoy, so he opens the app and looks if there is any available car nearby.

At the beginning the app tells him that, unfortunately, there are no available cars within ten minuted by foot; knowing the airport's parking is reachable in 10 minute by shuttle bus, he changes the settings to look for cars near a specific point on the map, and selects the airport's parking area.

The app immediately finds many more cars: 6 with the battery almost fully charged and 2 with about half charge left. He chooses a random full charged car and reserves it; then takes his luggage, takes the shuttle bus and reach the car within 15 min, leaving its phone's GPS active. Once near the car, the car is automatically unlocked: John can leave his luggage in the luggage van, enter the car and drive back home.

Once there, John parks correctly the car in a safe area, as indicated by the car's onboard system. He exits the car, takes his luggage out and uses the app to lock the car, which has still 60% of battery charge left. Once the lock is confirmed he receives the bill, calculated on the lenght and duration of the ride, and a 20% discount as a bonus to have left the car with more than 50% of battery charge. The nearest charging station locates within 3 km, so he gets no overprices.

John pays the bill immediately through the app.

Scenario 2

Anne want to go from his home to the airport to bring two friends home. Using PowerEnJoy app she books the nearest car and within 10 minutes she started her ride to the airport.

Once there, the car's onboard system tells her that there is a charging station nearby the airport. So Anne drives to the the charging station, parks, exits and plugs the car to the power grid. Then she locks the car. Once the lock is confirmed, she receives the bill and a 30% discount as bonus for plugging the car to the grid.

While waiting for her friends, she reserves another car for coming back, but the plane is late and the reservation, which lasts one hour, expires. Anne receives a little penalty for the expired reservation and pays it immediately; then reserves again the car.

When her friends arrive, they all reach the car within 30 minutes from the second booking: they get on the car and drive to her friends' home. At arrival, the car has about 40% of charge left, but there are no charging stations nearby (even if there is one at 2,1 km from their destination), so Anne parks it in a safe area, exit, take luggages out and locks the car. The app sends her the bill and applies a 10% discount as a bonus for bringing at least two friends on the same ride.

Scenario 3

Bob has an exam at 8 a.m., but the morning train is very late. So he decides to user PowerEnJoy and go to the university by car. He look for some cars near him, but he finds there is only one car left with about 40% of residual charge. He reserves it anyways and within 5 minutes he started his drive to the university.

He is in a hurry, so drives very badly and commits some infractions, that gets recorded by the police. The policeman who is taking care of Bob's infractions looks in the police's databases for the owner of the car he spotted and sends the fine to PowerEnJoy offices.

Once near his destination, the car's onboard system tells him that there is a charging station at about 10 minutes by foot from the university, and a parking area right in front of the entrance of the university. At the end of the ride, the car has only 15% of residual charge.

Bob doesn't care and parks the car in front of the university, then exits and locks the car. The app confirms the lock and send him the bill, 30% overpriced for having left the discharged car not plugged to the power grid, but at least he is in time for the exam.

At the end of the day, the fine has been recorded by the PowerEnJoy staff, so Bob receives a notification with the fine he received, some details about it, and the choice to pay it immediately or to go to the nearest police office to make an inquiry about it.

Scenario 4

it is a sunny Sunday and Charlie wants to go with his family in the countryside. He has no car, but he doesn't want to go there by train: so he decides to use PowerEnJoy to do his trip. He finds a fully charged car near his house, reserves it and within 20 minutes he started his drive.

At the end of the day, Charlie wants to go back home with his family. He book the car he brought far from the city with him and shortly thereafter he starts his journey back home. He is halfaway from home when suddenly the car stops running: the engine got too much heat during the day and broke.

Charlie does not need to send a report to PowerEnJoy: the onboard system immediately tells him that an issue report has automatically been sent to the central. So Charlie receives the first part of the bill and, paid it, he can look for another car to finish his trip. He finds only one at half an hour by foot from his location. Charlie books the car and manages to reach it in time to unlock it and go back home.

When he arrives, he parks the car in a safe area near his house. He does not care about charging stations, because he saw that the car has about 80% of charge; this way charlie does not notice that the nearest charging station is at 4.3 km from the place he is parking the car.

So he exits with all his family and locks the car: now he receives the bill, that is the basic price of the ride, plus a 10% discount for having brought many people on the car and a 30% overprice for having left the car too far from a charging station.

Charlie tries to pay, but his payment fails because the rechargeable credit card he connected to the app has not enough money to pay that unexpected overprice. So he gets a notice from PowerEnJoy that he is temporary banned from the service until he manages to pay the last pending bill.

Scenario 5

Harry is a staff operator of PowerEnJoy, in charge of re-chargin cars in-place when required. He has just completed another task, so he opens the app and looks for any other pending issues to solve.

The app gives him a list of issues near him: two discharged cars left far from a power station, one parked near a power station that needs to be plugged, a Technical Issue and a car parked outside from a safe area. Harry filters the issues based on their Exception status, in order to see only issues related to recharging, and gets a list of the three discharged cars. He takes in charge the nearest one pressing the button on his app's interface. The app shows him the way and Harry goes for that car.

Once near the car, Harry plugs the car to his portable charging station and charges it until it's almost full. Then changes manually the status of the car from Not Available to Available: but the system does not confirm the operation, because it cannot detect the residual charge of the battery. The Exception status of the car gets set to Mechanical Issue, but Harry has no means to fix that problem: so he does not take charge of the new issue and goes for another car to charge.

4.2 Use Cases

Use Case Diagram

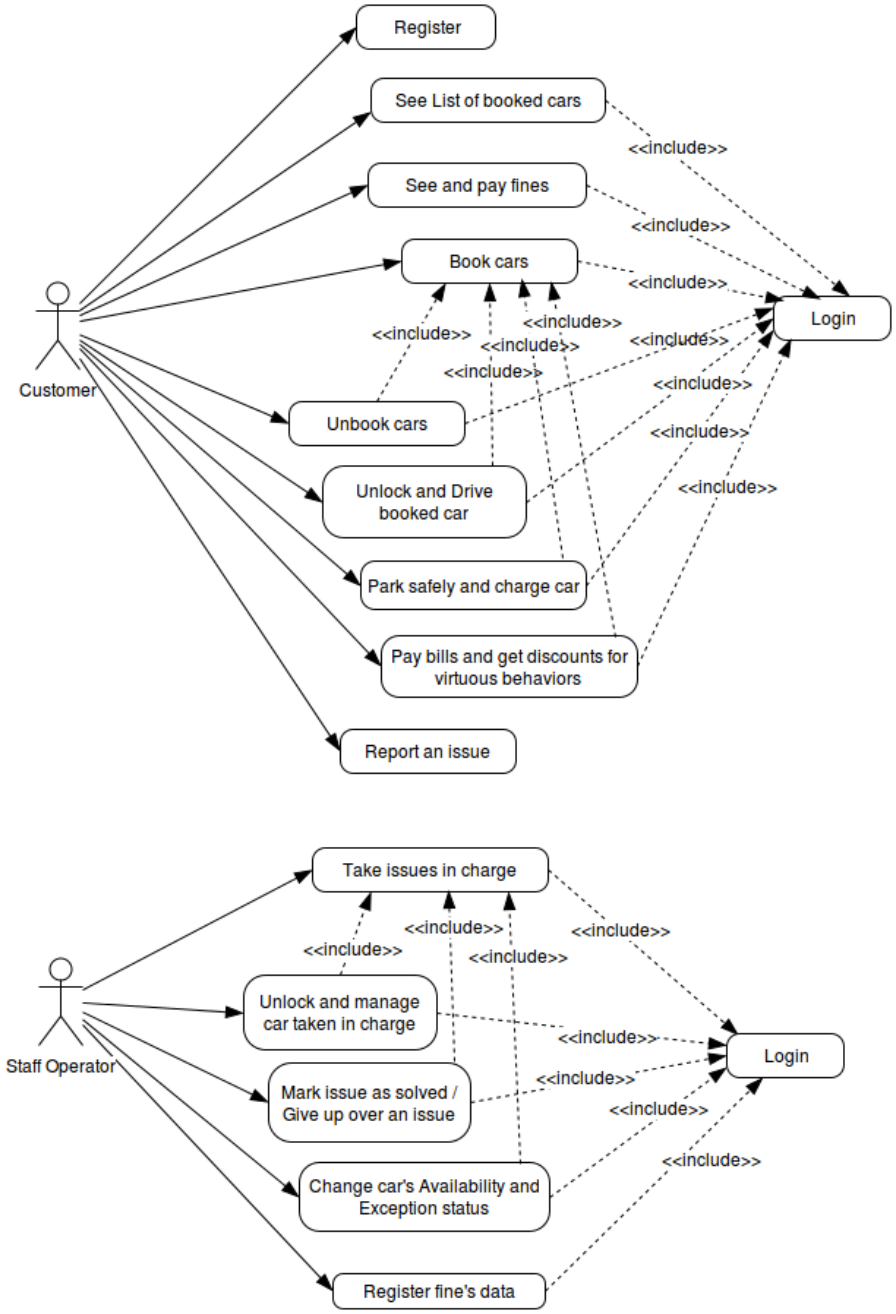

Figure 4: Use Case Diagram for Customers and Staff Operators

Customer's Use Cases Description

Name: Register
Actors Involved: A future user.
Entry Conditions: None
Flow Of Events:

- Opens the app, presses the Register button
- Fills the form with their personal information including a chosen password, Driving Licence Informations and payment informations.
- Confirms they took vision of Terms and Conditions of the Service
- Can press the Confirm button.

Exit conditions: The customer is successfully redirected to the Login page.
Exceptions:

- The user filled the form with invalid data: the app redirects him back to the form and highlights recognized problems.
- The user is found to have another account already active: they are redirected to the Login page with a notice that states they must login with their original credentials.

Name: Login
Actors Involved: A registered user.
Entry Conditions: None
Flow Of Events:

- Open the app, presses the Login button
- Fill the form with their username - password pair.
- Press the Login button.

Exit conditions: The customer is successfully redirected to the Cars Nearby page.
Exceptions:

- The user inputs a wrong username - password pair: they are redirected back to the Login page with a notice.

Name: Book cars
Actors Involved: A registered user, a car.
Entry Conditions: The user logged in and have no pending bills to pay.
Flow Of Events:

- Look at the Cars Nearby page and choses the car they want to book.
- Click on the car they chosed and see additional informations.
- Click the Book button.
- The system changes the status of the car from Available to Booked.

Exit conditions: The customer receives a notification confirming the operation, and see the deadline to unlock the car.
Exceptions: No predictable exceptions at this stage.

Name: Unbook cars
Actors Involved: A user who booked a car, the booked car.
Entry Conditions: The user logged in and has booked a car, but not yet unlocked it.
Flow Of Events:

- Press the Menu icon in the top-right corner of the screen.
- Press Bookings to see the bookings history.
- Select the active booking they want to quit.
- They receive a confirmation notice.
- Press the Unbook button.
- The system changes the status of the car from Booked to Available.

Exit conditions: The customer receives a notification confirming the operation.
Exceptions: No predictable exceptions at this stage.

Name: Unlock and drive booked cars
Actors Involved: A user who booked a car, the booked car.
Entry Conditions: The user logged in and has booked a car, but not yet unlocked it.
Flow Of Events:

- Get near to the car.
- The system unlocks the car.
- The system changes the status of the car from Booked to Unlocked.
- Receive a notice that their car has been unlocked.
- Enter the car.
- Scan their driving licence.
- Receive a confirmation the scanned driving licence matches.
- Switch on the car and drive to the destination.
- The system changes the status of the car from Unlocked to Running

Exit conditions: The user receives a notification their driving time has begun.
Exceptions:

- The car doesn't unlock: the user can try to send again the unlock message pressing the Unlock Manually button.
- The scanned driving licence is not correct for the user who booked the car:1 a notice is sent to the user and a new scan is expected.

Name: Parks safely and charge the car

Actors Involved: A user that is driving a car, the car itself.
Entry Conditions: The user is driving a car.
Flow Of Events:

- See the location of safe areas and charging stations on the map.
- Park in a safe location and switch off the engine.
- The system sees this change and sets the status of the car from Running to Parked.
- If near to a charging station, the driver may plug the car to the power grid.
- Confirm the parking pressing the Lock button.
- The system locks the car.
- The system changes the status of the car from Parked to Available.

Exit conditions: The user receives a confirmation the car has been locked.
Exceptions:

- The car's engine stops out from a safe area: the user is notified and, if they leave, fined.

Name: Pay bills and get discounts for virtuous behavior
Actors Involved: A user who just finished a ride.
Entry Conditions: The user locked a car after a ride.
Flow Of Events:

- Receives the bill, including the regular fare plus eventual discounts or overprices for their behavior.
- Press the Pay Now button at the bottom of the bill.
- Pay the fee.

Exit conditions: The user receives a confirmation that the payment has been completed successfully.
Exceptions:

- The payment fails: all the app's functions are locked until the user manages to pay the pending bill.

Name: See fine's informations and pay fines
Actors Involved: A user that commited an infraction while driving.
Entry Conditions: The user received a fine.
Flow Of Events:

- Receives a notice stating they received a fine.
- Press the Show Info button
- Additional informations on the fine are shown.
- Press the Pay Now button, or

- Press the Pay Later button. In this case all other functions of the app gets locked until the fine is paid.

Exit conditions: The user receives a confirmation that the payment has been completed successfully.
Exceptions:

- The payment fails: all the app's functions are locked until the user manages to pay the pending fine.

Name: Report an issue
Actors Involved: A user who is experiencing an issue.
Entry Conditions: The user notices something wrong with the app or the car.
Flow Of Events:

- Press the button Report An Issue.
- Fill the form and describe the issue.
- Send the Issue Report.

Exit conditions: The user receives a confirmation that their report has been sent successfully.
Exceptions: No predictable issue can happen at this stage.

Staff Operator's Use Cases Description

Name: Take issue in charge
Actors Involved: An operator.
Entry Conditions: The operator is logged in.
Flow Of Events:

- Looks at the map showing the nearest issues.
- Select a issue tapping on it.
- Press the Take in Charge button.

Exit conditions: The user receives a confirmation that they took that issue in charge and receive directions to reach the involved vehicle.
Exceptions: No predictable issue can happen at this stage.

Name: Unlock and manage car taken in charge
Actors Involved: An operator who took an issue in charge, the car with the issue considered.
Entry Conditions: The operator is logged in and took an issue in charge.
Flow Of Events:

- get near to the issued car.
- Receives a notification that their issued car has been unlocked.

- Get into the car and tries to solve the issue.

Exit conditions: The car gets unlocked.
Exceptions:

- The car doesn't unlock: the operator can try to unlock it manually or give up the issue.

Name: Mark an issue as solved or give up an issue
Actors Involved: An operator who took an issue in charge, the car with the issue considered.
Entry Conditions: The operator is logged in and took an issue in charge.
Flow Of Events:

- Open the Issue page, stating more details about the issue taken in charge.
- Press the Solved button if enabled, or
- Press the Give Up button.
- The system reacts to the solution either checking the car's availability or locking it back for another operator to take it in charge.

Exit conditions: The operator receives a notice confirming the operation.
Exceptions:

- The system doesn't confirm the issue has been solved: the issue status is changed to Technical issue and nother, different issue is generated.

Name: Change car's Exception status
Actors Involved: An operator who took an issue in charge, the car with the issue considered.
Entry Conditions: The operator is logged in and took an issue in charge.
Flow Of Events:

- Open the Issue page, stating more details about the issue taken in charge.
- Changes the Exception status choosing an item from the list.
- Press the Confirm button.

Exit conditions: The operator receives a notice confirming the operation.
Exceptions: No predictable exception at this stage.

Name: Register fines informations
Actors Involved: An operator, a fine sent to PowerEnJoy related to an infraction commited by an user while driving a PowerEnJoy car.
Entry Conditions: The operator is logged in.
Flow Of Events:

- Open the Register Fine page.
- Fill in all required data about the fine.

- Press the Register button.

Exit conditions: The operator receives a notice confirming the operation.
Exceptions:

- The operator input invalid data: they are redirected back to the form page, highlighting the problem.

4.3 Other UML Diagrams

Sequence Diagrams

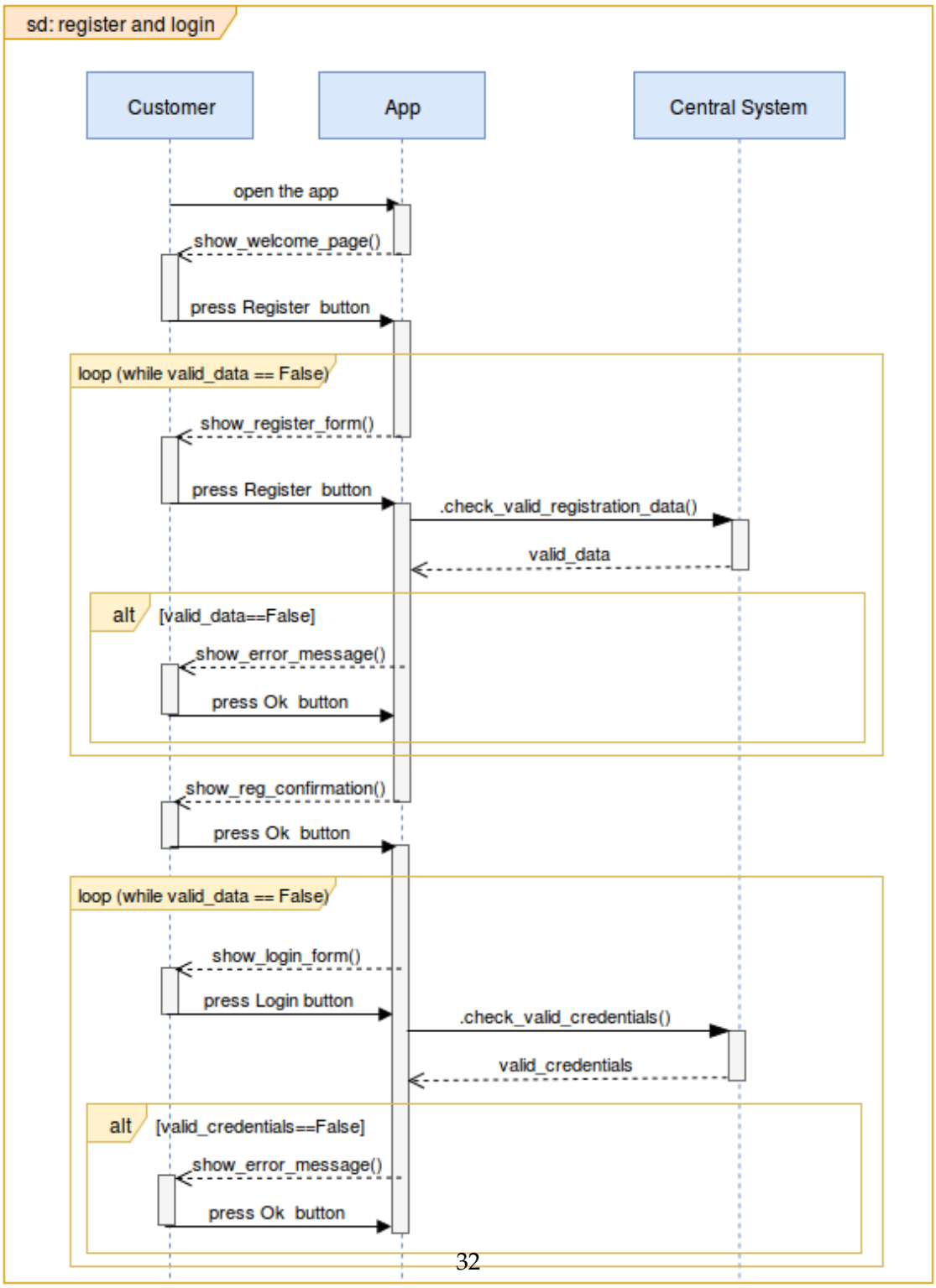

Figure 5: Registration and Login Process

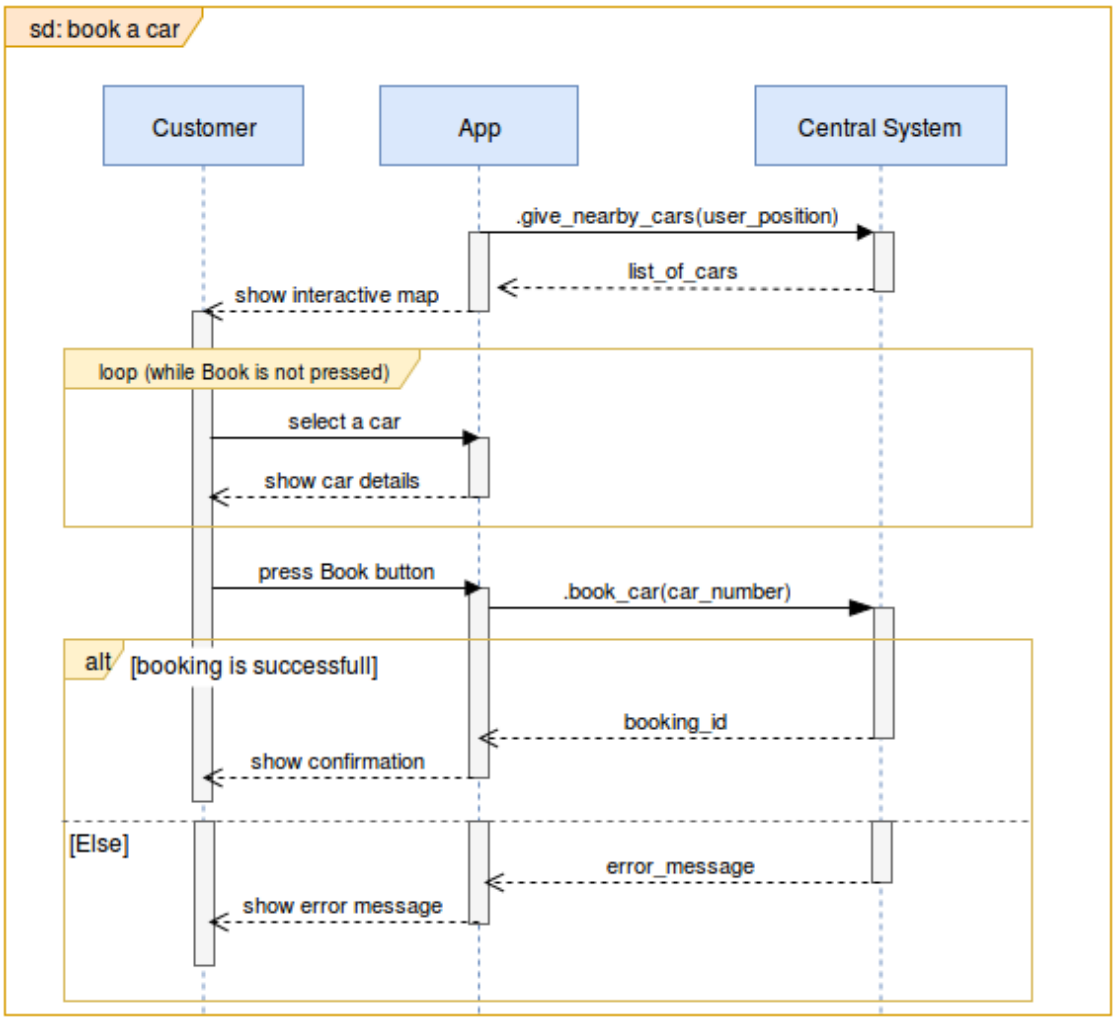

Figure 6: Booking Process

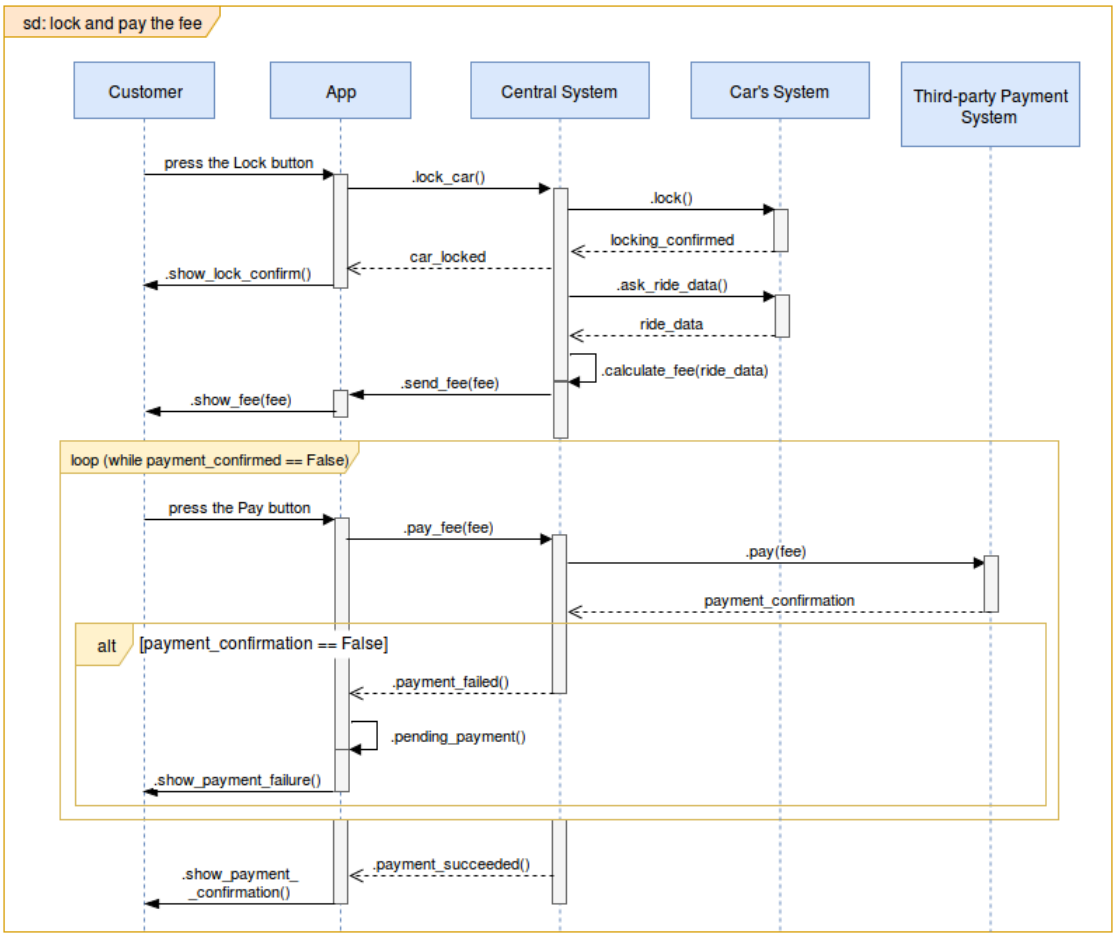

Figure 7: Locking and Paying Process

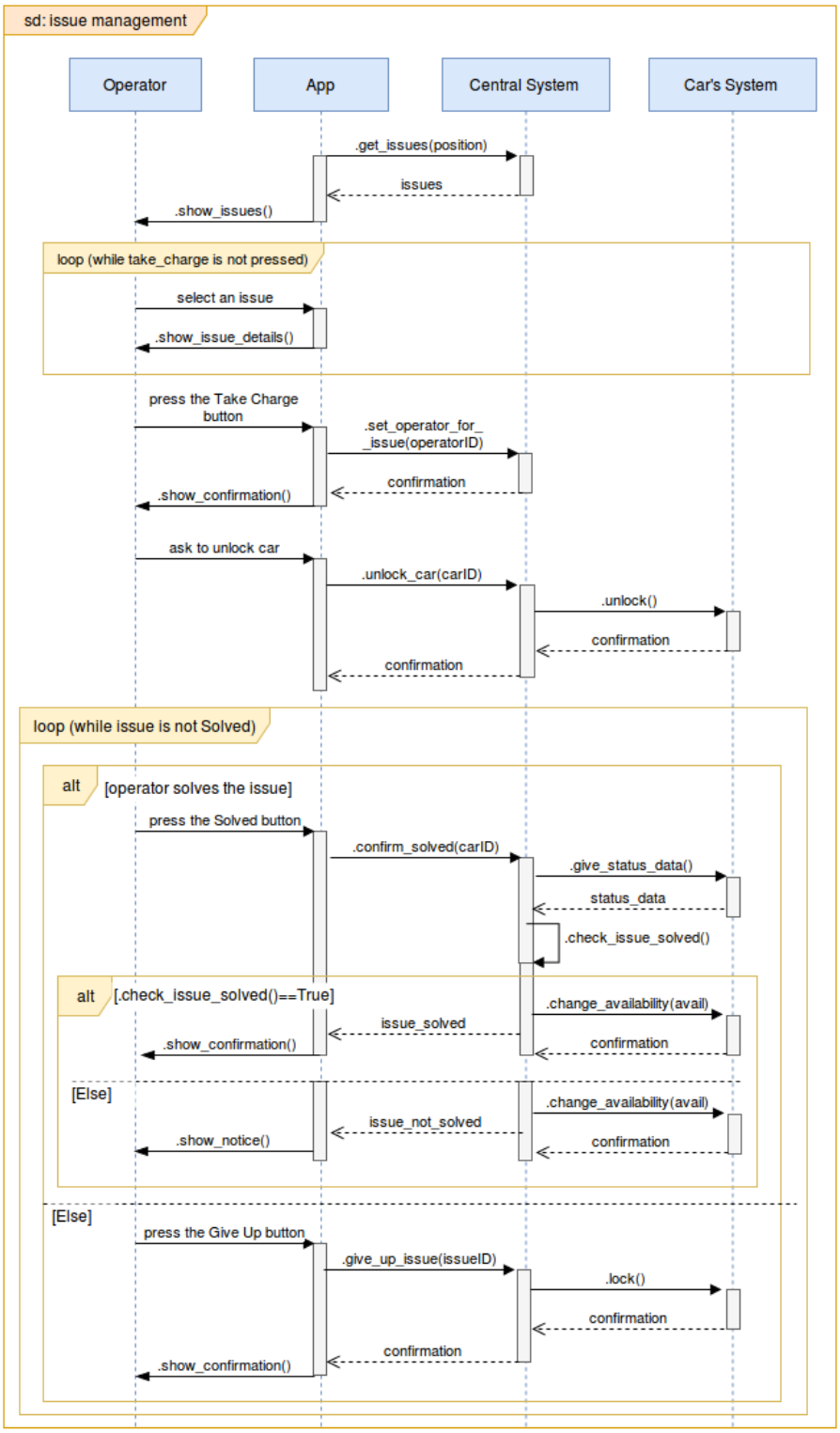

Figure 8: Issues Management Process

Activity Diagrams

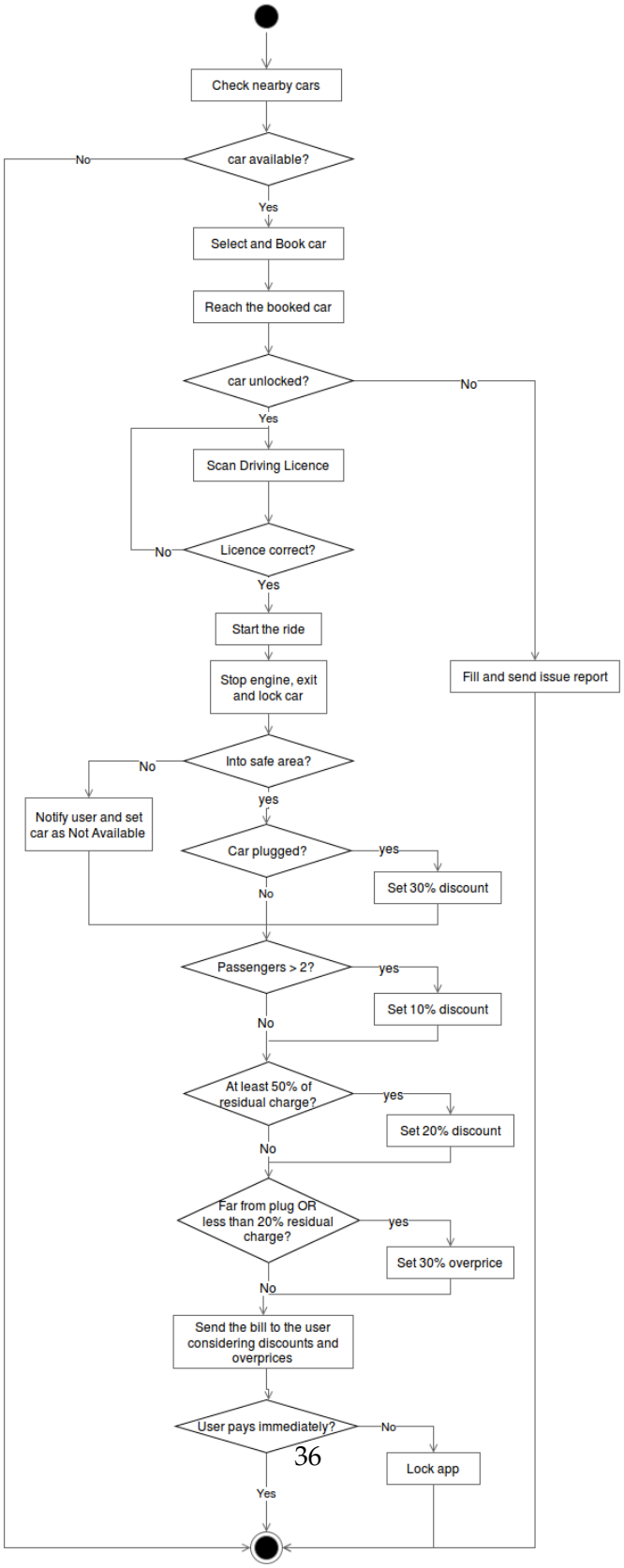

Figure 9: Booking Flow

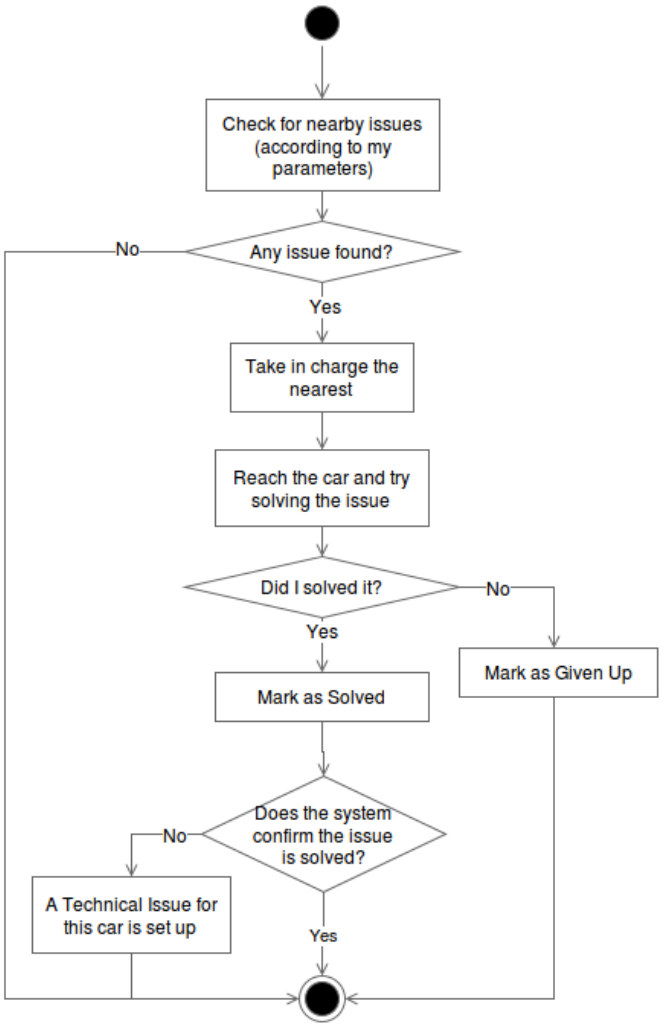

Figure 10: Issue Management Flow

State Diagrams

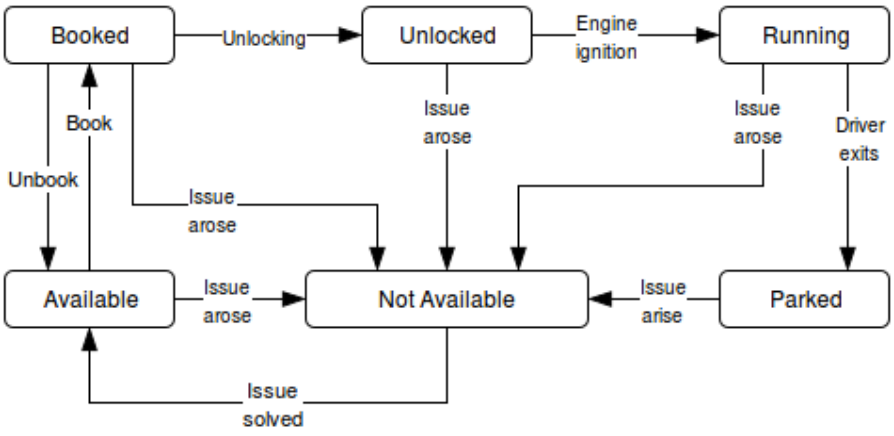

Figure 11: Car's Availability Status State Transitions

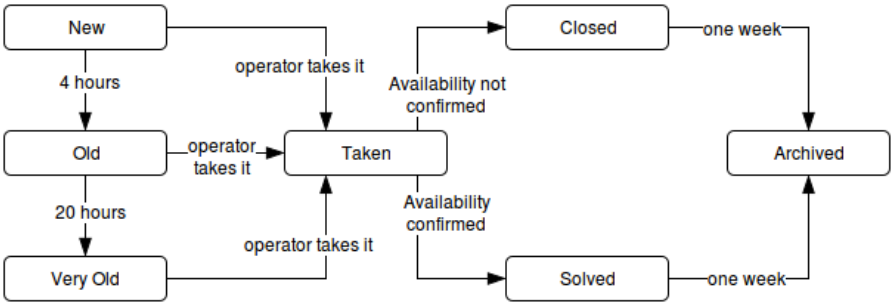

Figure 12: Issue's State Transitions

5 Alloy Model

To check the validity of the proposed solution we implemented a model in Alloy.

In order to keep the model itself manageable, it was first developed in small chucks and their validity checked. Then each chuck has been merged in a bigger model and the validity of the resulting model has been re-check again.

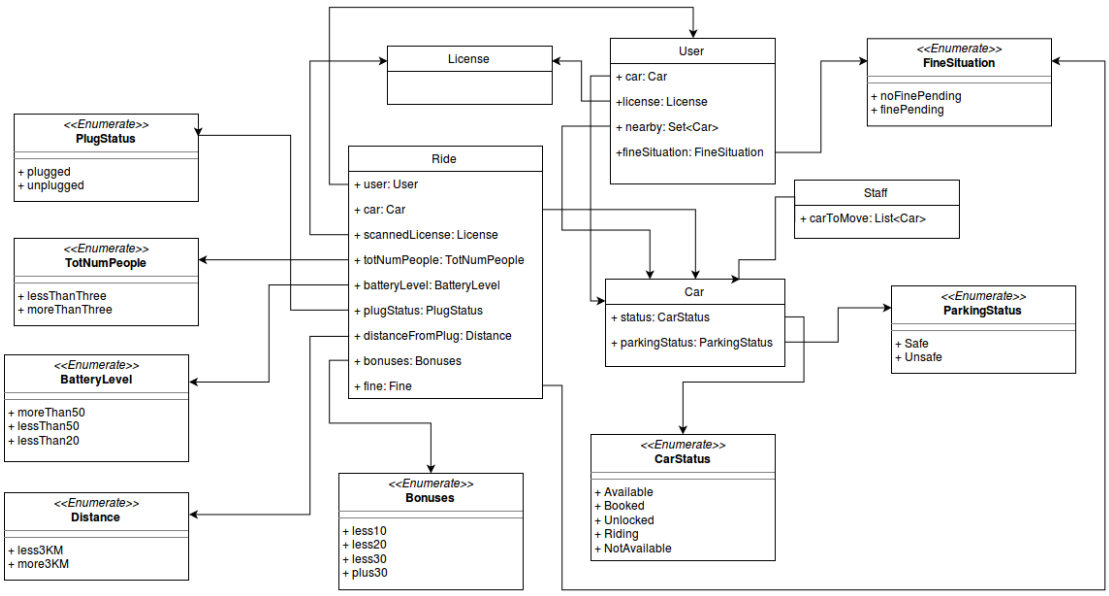

Figure 13: Class Diagram for the Alloy Model

5.1 Main Model

```
enum CarStatus {Available, Booked, Unlocked, Riding, NotAvailable}
enum ParkingStatus {Safe, Unsafe}
enum TotNumPeople {lessThan3, moreThan3}
enum BatteryLevel {moreThan50, lessThan50, lessThan20}
enum PlugStatus {plugged, unplugged}
enum Bonuses {less10, less20, less30, plus30}
enum Distance {less3KM, more3KM}
enum Fine {Fined, NotFined}
enum FineSituation {NoFinePending, FinePending}

sig Staff {
        carToMove: set Car
}

sig License {}

sig User {
        car: lone Car,
        license: one License,
        nearby: set Car,
        fineSituation: one FineSituation
```

```
}

fact NoActionWithAFinePending {
        no u: User | u.fineSituation = FinePending ∧ not no u.car
}

fact licenseIsPersonal {
        no u, u': User, l: License | u ≠ u' ∧ l in u.license ∧ l in u'.license
}

fact OnlyCarRidingDontHaveAParkingStatus {
        ∀ c: Car | c.status = Riding⟺no c.parkingStatus
}

fact CarUnsafelyParkedAreOnListToBeMoved {
        ∀ c: Car, s: Staff | c.parkingStatus = Unsafe⟺c in s.carToMove
}

fact CarUnsafelyParkedAreNotAvailable {
        ∀ c: Car | c.parkingStatus = Unsafe⟺c.status = NotAvailable
}

sig Car {
        status: one CarStatus,
        parkingStatus: lone ParkingStatus
}

sig Ride {
        user: one User,
        car: one Car,
        scannedLicense: one License,
        totNumPeople: one TotNumPeople,
        batteryLevel: one BatteryLevel,
        plugStatus: one PlugStatus,
        distance: one Distance,
        bonuses: set Bonuses,
        fine: one Fine
}

fact ABookedCarIsAssociatedWithAnUser {
        ∀ c: Car | some u: User | c.status = Booked⟺c in u.car
}

fact AnUnlockedCarIsAssociatedWithAnUser {
        ∀ c: Car | some u: User | c.status = Unlocked⟺c in u.car
}

fact AnAvailableCarIsAssociatedWithNoUser {
        no c: Car, u: User | c.status = Available ∧ c in u.car
}

fact AnNotAvailableCarIsAssociatedWithNoUser {
        no c: Car, u: User | c.status = NotAvailable ∧ c in u.car
}

fact RideOnlyIfTheScannedLicenseIsTheSameOfTheUser {
        no r: Ride, u: User, l: License |  r.user = u ∧ u.license = l ∧ r.scannedLicense ≠
l
}

fact CarCanBeBookedOnlyByUser{
        ∀ c: Car, b: Booked | some u: User | b in c.status⟺c in u.car
```

```
}

fact CarCanBeBookedOnlyByASingleUser {
        no c: Car, b: Booked, u, u': User | b in c.status ∧ u ≠ u' ∧ c in u.car ∧ c in u'.car
}

fact OnlyOneRideForOneCar {
        no c: Car, r, r': Ride | r ≠ r' ∧ r.car = c ∧ r'.car = c
}

fact RideOnlyWithRidingCar {
        no c: Car, r: Ride | r.car = c ∧ c.status ≠ Riding
}

fact NoRidingCarWithoutRide {
        ∀ c: Car | some r: Ride | c.status = Riding⟺c in r.car
}

fact RidingWithTheSameCarUsedByTheUser {
        no r: Ride, c: Car, u: User | r.car = c ∧ r.user = u ∧ u.car ≠ c
}

fact CarCanBeRidenOnlyByAnUser {
        no r: Ride, c: Car, u, u': User | u ≠ u' ∧ Riding = c.status ∧ u = r.user ∧ c =
r.car ∧ c = u'.car
}

fact UnlockedCarHasIsUserNearby {
        ∀ c: Car | some u: User | Unlocked = c.status⟺(c in u.nearby ∧ u.car = c)
}

fact RidingCarHasIsUserNearby {
        ∀ c: Car | some u: User | Riding = c.status⟺(c in u.nearby ∧ u.car = c)
}

pred UserCanBookAvailableCar[c, c': Car, b: Booked, a: Available, u, u': User] {
        (c ≠ c')
        (u ≠ u')

        a in c.status
        c not in u.car

        b in c'.status
        c' in u'.car
}

fact discountNumPeople {
        ∀ r: Ride | moreThan3 = r.totNumPeople⟺less10 in r.bonuses
}

fact discountBatterLevel {
        ∀ r: Ride | moreThan50 = r.batteryLevel⟺less20 in r.bonuses
}

fact discountPluggedCar {
        ∀ r: Ride | plugged = r.plugStatus⟺less30 in r.bonuses
}

fact overchargeUnloadFarAway {
        ∀ r: Ride | (lessThan20 = r.batteryLevel or more3KM = r.distance)⟺plus30 in r.bonuses
}
```

```
pred show{
        #Ride > 3
}

assert UserWithFinePendingCantBookUnlockRideCars {
        no u: User, c: Car | u.fineSituation = FinePending ∧ u.car = c ∧
                (c.status = Booked or c.status = Unlocked or c.status = Riding)
}

assert NoFineWithoutResponsableUser {
        no r: Ride | r.fine = Fined ∧ no r.user
}

assert AllNotSafelyParkedCarAreInTheListOfTheStaff {
        no c: Car, s: Staff | c.status = NotAvailable ∧ c not in s.carToMove
}

assert OnlyCarAvailableOrNotAvailableAreNotAssociateWithAnUser {
        no c: Car | some u: User | c in u.car ∧ (c.status = Available or c.status = NotAvailable)
}

assert UserAreDrivingACarOnlyIfTheScannedLicenseIsTheSameOfTheUserLicense {
        no r: Ride, u: User, l, l': License | l ≠ l' ∧ u.license = l ∧ r.scannedLicense =
l' ∧ r.user = u
}

check UserWithFinePendingCantBookUnlockRideCars for 10
check NoFineWithoutResponsableUser for 10
check AllNotSafelyParkedCarAreInTheListOfTheStaff for 10
check OnlyCarAvailableOrNotAvailableAreNotAssociateWithAnUser for 10
check UserAreDrivingACarOnlyIfTheScannedLicenseIsTheSameOfTheUserLicense for 10
run show for 5
```

Results of Execution

```
9 commands were executed. The results are:
  #1: No counterexample found. UserWithFinePendingCantBookUnlockRideCars may be valid.
  #2: No counterexample found. NoFineWithoutResponsableUser may be valid.
  #3: No counterexample found. AllNotSafelyParkedCarAreInTheListOfTheStaff may be valid.
  #4: No counterexample found. OnlyCarAvailableOrNotAvailableAreNotAssociateWithAnUser may be va
  #5: No counterexample found. UserAreDrivingACarOnlyIfTheScannedLicenseIsTheSameOfTheUserLice
  #6: No counterexample found. NoUsersHaveTheSameCarBooked may be valid.
  #7: No counterexample found. NoUsersHaveTheSameCarUnlocked may be valid.
  #8: No counterexample found. UnlockedCarHaveTheirUserNearby may be valid.
  #9: Instance found. show is consistent.
```

Figure 14: Output of the Alloy Analizer

Generated graph

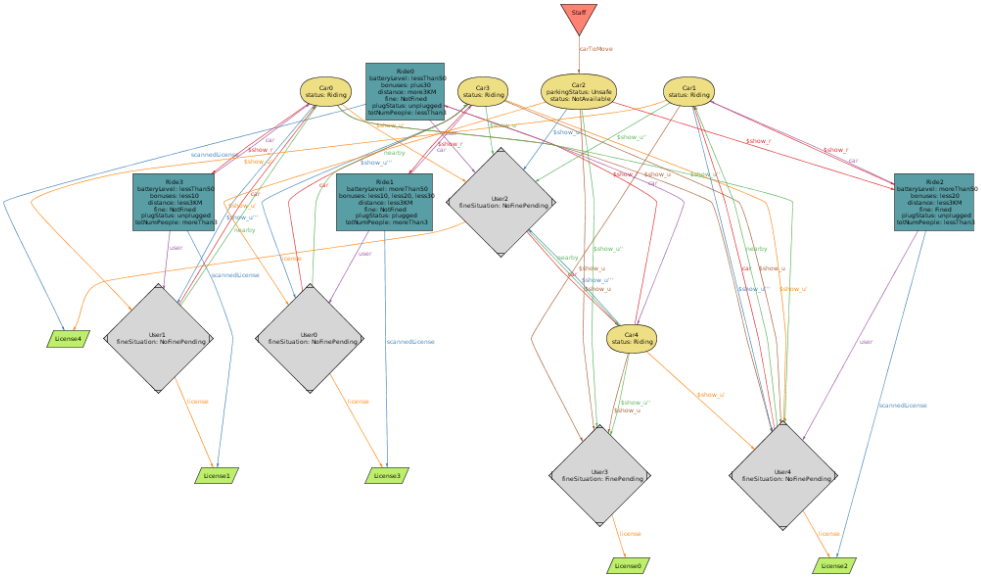

Figure 15: One of the model returned by Alloy, compressed visualization

5.2 Dynamic Model

Finally, in order to check the validity of our solution in a dynamic context, we develop another smaller Alloy model of the dynamic aspects of the system.

Model

```
enum CarStatus {Available, Booked, Unlocked, Riding, NotAvailable}

sig User {
        car: lone Car,
        nearby: set Car,
}

sig Car {
        status: one CarStatus
}

fact IfAnUserIsAssociateWIthACarNoOneElseIs {
        no u, u': User | some c: Car | u ≠ u' ∧ c = u.car ∧ c = u'.car
}

fact ABookedCarIsAssociatedWithAnUser {
        ∀ c: Car | some u: User | c.status = Booked⟺c in u.car
}

fact AnUnlockedCarIsAssociatedWithAnUser {
        ∀ c: Car | some u: User | c.status = Unlocked⟺c in u.car
}
```

```
fact AnUnlockedCarHasIsUserNearby {
        no c: Car, u: User | c.status = Unlocked ∧ c in u.car ∧ c not in u.nearby
}

fact AnAvailableCarIsAssociatedWithNoUser {
        no c: Car, u: User | c.status = Available ∧ c in u.car
}

fact AnNotAvailableCarIsAssociatedWithNoUser {
        no c: Car, u: User | c.status = NotAvailable ∧ c in u.car
}

fun BookableCar[]: Car {
        {c : Car | c.status = Available}
}

fun UnlockableCar[u: User]: Car {
        {c: Car | c.status = Booked ∧ c in u.car ∧ c in u.nearby}
}

pred PossibleToBookACar[c, c': Car, u, u': User] {
        c ≠ c'
        u ≠ u'
        c in BookableCar
        no u.car

        c'.status = Booked
        u'.car = c'
        u.nearby = u'.nearby
        c not in u.nearby⟺c' not in u'.nearby
}

pred PossibleToUnlockACar[c, c': Car, u, u': User] {
        c ≠ c'
        u ≠ u'
        c in UnlockableCar[u]
        c in u.car

        c'.status = Unlocked
        u'.car = c'
        u.nearby = u'.nearby
        c not in u.nearby⟺c' not in u'.nearby
}

assert UnlockedCarHaveTheirUserNearby {
        no u: User | some c: Car | c.status = Unlocked ∧ c not in u.nearby ∧ u.car = c
}

pred PossibleToRideACar[c, c': Car, u, u': User]{
        c ≠ c'
        u ≠ u'
        c.status = Unlocked
        c in u.car

        c'.status = Riding
        u'.car = c'
        c' in u'.nearby
        u.nearby = u'.nearby
        c not in u.nearby⟺c' not in u'.nearby
}
```

```
pred PossibleToReleaseACar[c, c': Car, u, u': User]{
        c ≠ c'
        u ≠ u'
        c.status = Riding
        c in u.car
        c in u.nearby

        c'.status = Available
        no u'.car
        c' in u'.nearby
        u.nearby = u'.nearby
        c not in u.nearby⟺c' not in u'.nearby
}

pred PossibleToBookUnlockRideACar[c, c', c'', c''': Car, u, u', u'', u''': User] {
        PossibleToBookACar[c, c', u, u']
        PossibleToUnlockACar[c', c'', u', u'']
        PossibleToRideACar[c'', c''', u'', u''']
}

pred PossibleWholeCycle[c, c', c'', c''', c'''': Car, u, u', u'', u''', u'''': User] {
        PossibleToBookACar[c, c', u, u']
        PossibleToUnlockACar[c', c'', u', u'']
        PossibleToRideACar[c'', c''', u'', u''']
        PossibleToReleaseACar[c''', c'''', u''', u'''']
}

pred show{}

run show for 5
run PossibleToBookACar for 5
run PossibleToUnlockACar for 5
run PossibleToRideACar for 5
check UnlockedCarHaveTheirUserNearby for 10
run PossibleToBookUnlockRideACar for 4
run PossibleToReleaseACar for 5
run PossibleWholeCycle for 5
```

Result of Execution

8 commands were executed. The results are:
#1: Instance found. show is consistent.
#2: Instance found. PossibleToBookACar is consistent.
#3: Instance found. PossibleToUnlockACar is consistent.
#4: Instance found. PossibleToRideACar is consistent.
#5: No counterexample found. UnlockedCarHaveTheirUserNearby may be valid.
#6: Instance found. PossibleToBookUnlockRideACar is consistent.
#7: Instance found. PossibleToReleaseACar is consistent.
#8: Instance found. PossibleWholeCycle is consistent.

Figure 16: Output of the Alloy Analizer

6 Conclusions

6.1 Future Development

There are a lot of aspects of PowerEnJoy that can be improved in the future.

For example, users may show the need to exit their car for a short amount of time during the ride: for example, to go shopping and then come back home. This can be an improvement in the next versions of the software.

We may also think at more bonuses for virtuous behaviours that at this stage were not considered: for example, leaving the car in an area where there are on average a lot of requests can be seen as a virtuous behavior that can be rewarded.

We can also think at different payment methods for frequent users, like prepaid credit related to user's account, or discounts for very frequent (i.e. daily) and usually virtuous users of the service.

6.2 Tools used

During the development of this document we used the following tools:

- **Github** to version control the project
- **LaTeX** on TeXworks to redact this document
- **Alloy Analyzer v.4.2** to check model's validity
- **www.draw.io** to draw UML graphs
- **Gimp v.2.8** to mockup the application

6.3 Hours of work

- SZ: 4h on 1/11
- SZ: 4h on 4/11
- SM: 5h on 4/11
- SZ: 6h on 5/11
- SM: 2h on 6/11
- SM: 4h on 7/11
- SZ: 4h on 7/11
- SM: 4h on 9/11
- SZ: 2h on 10/11

- SM: 4h on 10/11
- SZ: 2h on 11/11
- SZ: 4h on 12/11
- SM: 6h on 12/11