

# Requirements Analysis and Specifications Document (RASD)

Simone Mosciatti & Sara Zanzottera

November 12, 2016

# Contents

1	Introduction	4
1.1	Purpose . . . . .	4
1.2	Scope . . . . .	4
1.3	Definitions, acronyms, abbreviations . . . . .	4
2	Overall Description	5
2.1	Product Perspective . . . . .	6
2.2	Product Functions . . . . .	6
2.3	Parallel Operation . . . . .	6
2.4	Actual Systems . . . . .	7
2.5	Constraints . . . . .	7
	Constraints on the user . . . . .	7
	Platform Constraints . . . . .	7
	Privacy regulations . . . . .	7
2.6	Actors and stakeholders . . . . .	8
	Actors . . . . .	8
	Stakeholders . . . . .	8
2.7	Text Assumptions . . . . .	8
	Assumptions on the final user . . . . .	8
	Assumptions on the car system . . . . .	9
	Statuses assumptions . . . . .	9
	External Services . . . . .	11
2.8	Domain Properties . . . . .	11
2.9	Reference Documents . . . . .	12
3	Specific Requirements	13
3.1	Goals . . . . .	13
3.2	Functional Requirements . . . . .	13
3.3	Non Functional Requirements . . . . .	19
	Client's App Interface . . . . .	19
	Car's Onboard System . . . . .	20
	Staff's App Interface . . . . .	20
4	Scenarios, Use Cases and UML Diagrams	21
4.1	Scenarios . . . . .	21
	Scenario 1 . . . . .	21
	Scenario 2 . . . . .	21
	Scenario 3 . . . . .	21
	Scenario 4 . . . . .	22
	Scenario 5 . . . . .	23
4.2	Use Case Diagram . . . . .	24
	Customer's Use Cases Description . . . . .	25

	Staff Operator's Use Cases Description . . . . .	28
4.3	Other UML Diagrams . . . . .	30
	Sequence Diagrams . . . . .	30
4.4	Activity Diagrams . . . . .	34
	State Diagrams . . . . .	36
5	Alloy Model . . . . .	37
5.1	Model . . . . .	37
5.2	Results of Execution . . . . .	40
5.3	Generated graph . . . . .	41
5.4	Dynamic Modeling . . . . .	41
6	Conclusions . . . . .	44
6.1	Future Development . . . . .	44
6.2	Tools used . . . . .	44
6.3	Hours of work . . . . .	44

# 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 constraints 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 incentivize people to be environment cautious drivers.

## 1.3 Definitions, acronyms, abbreviations

**User** A customer of PowerEnJoy using the service.

**Staff Operator** An employee of PowerEnJoy which takes care of maintains the car in good conditions.

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

**Running Time** The time that an user is using the PowerEnJoy service.

**Issue** Any problem a car may incur in.

**Nearby Cars** Car close to a specific position.

**Nearby Issues** Issues that are affecting car close to a specific position.

**Booking (Reservation)** The act to reserve a car for a limited amount of time, only the user that reserved the car will be able to unlock it.

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

**Passenger** Whoever is inside a PowerEnJoy car but is not the user.

**Driving License** The state issued driving license of the user, it is mandatory to use the services.

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

**Issue Report** An incoming notification that signal some issues to the cars.

**Fine** A fine issued by the local law enforcing officers.

**Pending Bills** Bills that a user still needs to pay to PowerEnJoy due to previous ride.

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

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

**Car's Onboard System** The control system of the car, it is able to exchange data with the central system and to relevant operation parameters.

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

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

**Central System** The central system for PowerEnJoy, all the commands 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.

## 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 penalties in terms of discounts or extra fees depending on the behavior of the user, in order to incentivize a virtuous behavior. Some examples are:

- a discount of 10% for users who bring other passengers with him during the ride.

- a discount of 20% if the car is left with at least 50% of the battery still full, or a 30% if the car is left near a charging station and plugged.
- if the car is left more than 3km from the nearest charging station or with less than 20% of charge left, the user is charged 30% more.

PowerEnjoy is divided in two main part: a frontend, used by the customers, and a backend that provides the service. In addition there is a reserved frontend, used exclusively by the staff members to better organize their job.

## 2.1 Product Perspective

PowerEnjoy provide users the ability to rent cars for a limited amount of time in a bounded area, for example a city. The frontend part, the app, provides the user the ability to look for available cars, reserve cars for up of one hour and unlock the cars once they are near the reserved vehicle; it is used also to lock the vehicle once the run is over and to pay the fee.

The staff's reserved frontend tells them which cars 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 to pay.

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 will be able to 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 he wants to use, they will be able to book the car for one hour. After the car is been booked, the user can unlock the car only when in proximity of the car itself.

The system starts charging the user when the engine starts running, and will stop once the car is been parked, the user exit and locks it. Several forms of discount or overfees will be applied depending on the behaviour of the user when riding and parking the car.

## 2.3 Parallel Operation

The system is capable to operate and serve users concurrently.

## 2.4 Actual Systems

As for now there exists 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

### Constraints on the user

Users who wish to use PowerEnjoy are required to fulfill some specific requirements:

1. The user must be registered to the service.
2. The user must have a valid driving license.

### Platform Constraints

Also, the system has some platform constraints coming from the device that is running the app:

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 be sustainable.
4. The app should be able to get information from the smartphones' GPS.

### Privacy regulations

The system should comply 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 its 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 team that takes care of the digital infrastructure and a **staff** that takes care of the cars in some special situations:

- 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 government**, 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
3. The final user has a smartphone with a GPS
4. The final user has a smartphone with internet connectivity



### Assumptions on the car system

1. Each car is provide with internet connectivity and capable to send and receive data to the main server
2. Each car is provided of a GPS of reasonable accuracy.
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. An empty car gets locked automatically after a short time (namely 5 minutes)

### 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:
  - 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.

## 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 power grid station
- Determinate the distance between two different GPS signals
- Scan and read driving licences' informations
- Ensure validity of Identity IDs and their coupling with personal informations provided by the user.
- 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.

- The user will provide a valid driving license during the registration to the service.
- A driving license will not expire nor be revoked for the whole time the user is registered.
- Different people will not use the same account, never, neither in different moment in time. Each account uniquely identify a physical person.
- It is possible for the company to bring back cars from not-safe areas into safe areas, and from outside the city boundaries into the boundaries themselves.

- A car cannot change its position until it is locked.
- 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 power stations.
- The car instrumentation works as expected:
  - It is possible to determine whether the car is inside the city boundaries at any moment.
  - 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 a car at any moment.
  - Each and any message that the car need to send to the main server will arrive and be processed.
  - It is possible to read a driving licence's informations.
  - Once the car is plugged to the power station the battery starts to increase its charge.
  - Once the car's engine is switched on, the battery starts to decrease its charge.
- The total running time of an user is counted from either when the user starts the engines or after 3 minutes of the user unlocking the car, whichever comes first.
- The total running time of an user ends when the user locks the car.

## 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 functional and not functional requirements.

#### 3.1 Goals

**REGISTRATION** The user is able to register to PowerEnJoy .

**LOGIN** The user is able to login to PowerEnJoy .

**LOOKUP** The user is able to find cars nearby their own position.

**BOOK** The user is able to book a car for a short amount of time.

**UNLOCK** When in proximity of the car, the user is able to unlock it.

**RIDE** When the user enters a car, they can drive to their destination while the car tracks the distance and the time of the ride.

**OUT\_CITY** If a user drives out from the city's border, they are fined accordingly.

**SAFE\_AREAS** The user is able to locate safe parking areas.

**UNSAFE\_PARKING** The system can react to an unsafe parking.

**POWER\_STATIONS** The user is able to locate charging stations.

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

**PAYMENTS** The app allows the user to pay bills through the app.

**FIND\_ISSUES** The staff is able to locate cars that need their intervention.

**SUPPORT** The staff is able to identify and solve car's issues.

**INFRACTIONS** The system must be able to process a fine sent to the company by a local police officer.

#### 3.2 Functional Requirements

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

**REGISTRATION** The user is able to register to PowerEnJoy .

**REG1** The system must be able to create new account with data provided through the client's application.

**REG2** The system must be able to validate the data the user want to use to create a new account.

**REG3** The system must be able to check whether the user has no other accounts, i.e. if the person registering to the service has already an account.

**REG4** The system must ensure that, at registering time, the user is 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** The user is able to login to PowerEnjoy .

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

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

**LOG3** The system must be able to login the user, given 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** The user is able to find cars nearby some position, it could be its position or a point in the map.

**LOOK1** The system is able to send to each user a list of the available cars that locates within a specified walking distance from the position specified.

**LOOK2** The user 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 change the ranking metric of the nearby cars according to the user's preferences.

**LOOK4** The app is able to show to the user the position of all the available cars on an interactive map.

**LOOK5** The app is able to show to the user additional informations on each car if the user requests so.

**LOOK6** The app is able to show to the user the shortest path to reach his car by foot on an interactive map, and to show him how long the walk is approximately going to take.

**BOOK** The user is able to book a car for a short amount of time.

**BOOK1** The user is able to select the car they want to book.

- BOOK2** The system is able to change the status of a car from Available to Booked.
- 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.
- BOOK6** If the user does not unlock the car after a short period of time (one hour), a small fee is charged to the user and the car's status is set back to Available.
- UNLOCK** When in proximity of the car, the user is able to 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 changes automatically the status of the car from Booked to Unlocked.
- UNLK5** The user can ask the system to unlock their car at any time, even if they are still far from the car.
- UNLK6** If a user unlocks the car and does not enter within a short period of time (namely 5 minutes) the car gets back to Booked and the timer restarts from the point it stopped.
- UNLK7** The user can lock back an unlocked car from the app before starting the ride. In this case the car 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** When the user enters the car, they can drive to their destination while the car tracks the distance and the time spent.
- RIDE1** Once the engine is turned on, the car's Availability status is automatically changed to Running.
- RIDE2** Before turning the engine on, the user who booked the car must scan his driving licence into the car's system.
- RIDE3** The system is able to determine whether the scanned driving licence is associated with the user who rented the car.
- RIDE4** The system is able to 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.
- RIDE5** The car's onboard system is able to show to the user 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 charge the car.

**RIDE6** 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.

**RIDE7**

**OUT\_CITY** If a user drives out from the city's border, they are fined accordingly.

**OUT1** If the car goes outside the city boundaries, the system is alerted and the car can tell the customer that he is gonna be fined if he continues to ride outside the boundaries.

**OUT2** If the alerted user continue to drive far from the city, the car's Availability becomes Not Available and the Exception status becomes Out of City Boundaries

**OUT3** If the car is in Out of City Boundaries status for more than 24 hours, the local authorities are contacted and informations about the car and the user who stole it are sent to them.

**OUT4** If the car comes back into the boundaries within 24 hours, its Availability status gets back to Running, but the user is fined for the time he spent outside the city.

**OUT5** If the car comes back into the boundaries within 10 minutes from the first alert, its Availability gets back to Running and no additional fees are charged.

**SAFE AREAS** The user is able to locate safe parking areas.

**SAFE1** The system is able to send to the car all informations about safe areas near its position.

**SAFE2** The car is able to show safe areas' location to the driver.

**SAFE3** The app is able to confirm the driver he parked in a safe area.

**SAFE4** Once the car's engine stops in a safe area, the Availability status changes automatically from Running to Parked.

**UNSAFE PARKING** The system must be able to react to an unsafe parking.

**UNSF1** If the car's engine stops outside from a safe area, the car should notify the user that he is going to park outside a safe area and that he is going to be fined if he leaves the car there.

**UNSF2** If a user leaves the car for more than 5 minutes outside a safe area (regardless 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** The user is able to locate and use charging stations correctly.

**PWRS1** The system is able to send to the car all informations about power stations near its position.

**PWRS2** The car is able to show power stations' location to the driver.

**PWRS3** The system must be aware of which car gets plugged to the power grid and by which user.

**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** If the user leave the car with no more than 50% of the battery empty a discount of 20% will be applied.

**FEE4** If the user plugs the car to a power plug in the special parking areas, a discount of 30% will be applied. The user is required to plug the car before locking it.

**FEE5** If the user leave the car with less than 20% of battery charge at more than 3KM from the closest charging site a 30% more will be charged on the total fee.

**FEE6** The user receives the fee immediately when its car gets locked.

**PAYMENTS** The app allows the user to pay fees and eventually fines through the app.

**PAY1** The user can set a preferred payment method for their account through the app.

**PAY2** Near each bill there is a button that allows the user 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 prevent the user to do any other action until the last payment is completed successfully.

**FIND\_ISSUES** The staff is able to locate cars that need their intervention.

**ISS1** The staff's app is able to receive informations about all cars that are in Not Available status but not yet taken by any operator, and to show them to the user.

**ISS2** The staff's app can filter and sort the cars basing on their distance from the user's location, priority of the issue, time of Non Availability and issue type (the Exception status)

**ISS3** The staff's app can give directions to the user to reach the selected car.

**SUPPORT** The staff is able to identify and solve car's issues.

**SUP1** The staff's app allows the operator to signal that a staff member is in charge of a certain issue.

**SUP2** The staff's app allows the operator 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 re-checks 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 that was in charge of that issue and another, different issue is generated.

**SUP4** The staff's app allows the operator to change the Exception status of the car they are in charge of.

**SUP5** The staff's app allows the operator to unlock and lock the car they are in charge of.

**SUP6** The staff's app allows the operator 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.

**INFRACTIONS** The system must be able to process a fine sent to the company by a local police officer.

**INF1** The staff's app can receive and store fine informations.

**INF2** The system is able to find out which user was driving a specific car in a specific moment.

**INF3** The app is able to show fine's information to the correct user.

**INF4** The app allows the user to pay the fine directly through the app.

**INF5** The app prevent the user from using any function of the service until they have paid all their fines.

### 3.3 Non Functional Requirements

#### Client's App Interface



Figure 1: Client's app principal pages

## Car's Onboard System

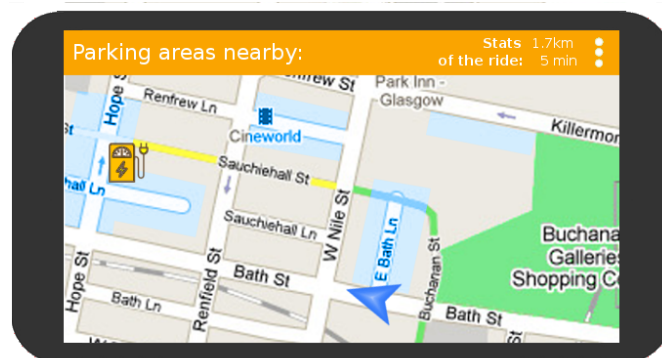


Figure 2: Car's Onboard System as it looks while driving

## Staff's App Interface

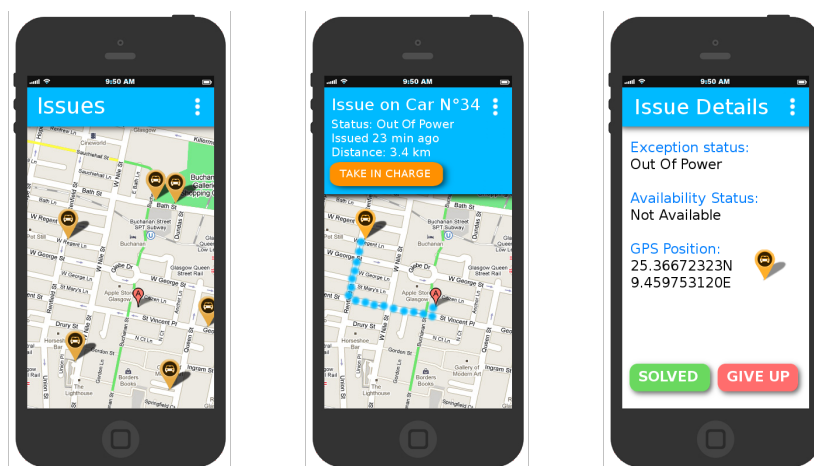


Figure 3: Staff's Operators special pages

## 4 Scenarios, Use Cases and UML Diagrams

### 4.1 Scenarios

#### Scenario 1

John needs to go from the airport, which locates inside the city's boundaries, to his home. He is already a user of PowerEnjoy, so he opens the app and looks for some available cars in its area. He opens the map and finds there are 8 available cars in a 10-minutes walk distance from his home: 6 almost fully charged and 2 with about half charge left. He chooses the nearest full charged car and reserves it; then takes his luggage 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 to the airport.

Once there, John parks correctly the car in a safe area, as indicated by the car's onboard system. John 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.

John pays the bill immediately through the app.

#### Scenario 2

John want to go from his home to the airport, which locates inside the city's boundaries, to bring two friends home. He reserves the nearest car and within 10 minutes he started his ride to the airport.

Once there, the car's onboard system tells him that there is a charging station nearby the airport. So John drives to the parking area nearby the charging station, parks, exits and plugs the car to the power grid. Then he locks the car. Once the lock is confirmed, he receives the bill, calculated on the distance and the time of the ride, and receives a 30% discount as bonus for plugging the car to the grid.

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

When his friends arrive, they all reach the car within 30 minutes from the second booking: they all enter and drive to his friends' home. When exiting, the car has about 40% of charge left, but there are no charging stations nearby, so John parks it in a safe area, exit, take luggages out and locks the car. The app sends him 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 to go to the university by car. He look for cars nearby, 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 processed by both the police officers and PowerEnjoy, 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 a notice 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.

He parks the car in a safe area near his house, 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
- a 30% overprice for having left the car discharged far from a charging station

Charlie tries to pay, but his payment fails because the rechargeable credit card he connected to the app is now empty. 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-charging cars in-place when required. He has just completed another task in Charlie's neighborhood, 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 Charlie's discharged car, which is the nearest, 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 battery 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 Case Diagram

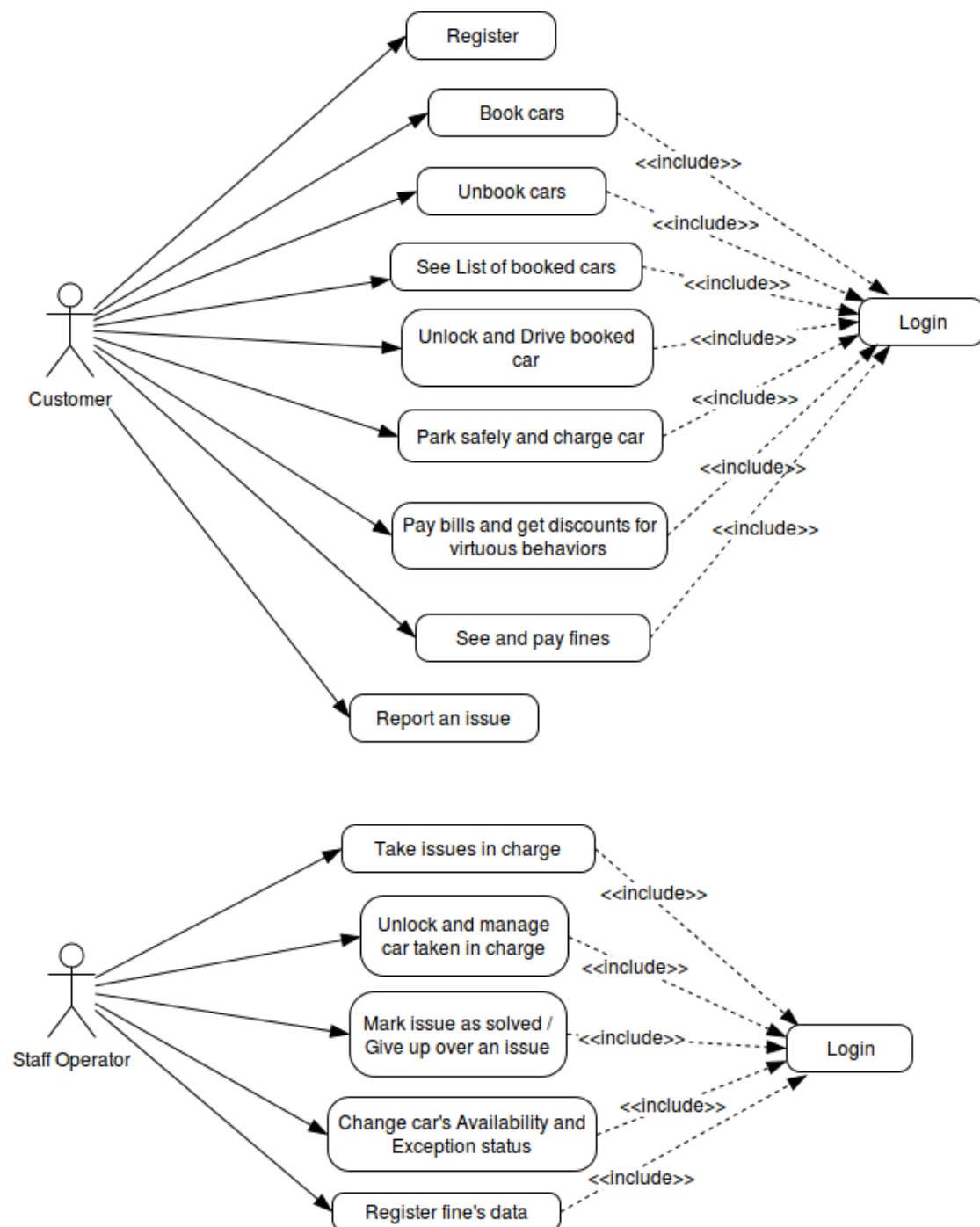


Figure 4: Use Case Diagram for Customers and Staff Operators



## Customer's Use Cases Description

**Name: Register**

**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: then the app redirects him back to the form and highlights recognized problems.
- The user is found to have another account already active: then they are redirected to the Login page with a notice that states they must login with their original credential or try to recover them.

**Name: Login**

**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 Near You page.

**Exceptions:**

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

**Name: Book cars**

**Entry Conditions:** The user logged in and have no pending bills to pay.

**Flow Of Events:**

- Look at the Cars Near You page and choses the car they want to book.
- Click on the car they chosed and see additional informations.
- Click the Book Now button.

**Exit conditions:** The customer receives a notice confirming the operation, and see the deadline to unlock the car.

**Exceptions:** No predictable exceptions at this stage.

**Name: Unbook cars**

**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 Now button.

**Exit conditions:** The customer receives a notice confirming the operation, and receive directions showing how to reach the vehicle.

**Exceptions:** No predictable exceptions at this stage.

**Name: Unlock and drive booked cars**

**Entry Conditions:** The user logged in and has booked a car, but not yet unlocked it.

**Flow Of Events:**

- Get near to the car.
- Receive a notice that their car has been unlocked.
- Get into the car.
- Scan their driving licence.
- Receive a confirmation the driving licence has been read and it is the correct one.
- Can switch on the car and drive to the destination.

**Exit conditions:** The user receives a notice their driving time has begun.

**Exceptions:**

- The car doesn't unlock: then 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: then a notice is sent to the user and a new scan is expected.

**Name: Parks safely and charge the car**

**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.
- Exit from the car.
- If near to a charging station, may plug the car to the power grid.

- Confirm the parking pressing the Lock button.

**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**

**Entry Conditions:** The user locked a car after a ride.

**Flow Of Events:**

- Receives the bill, including the regular fare plus eventual discounts or over-prices for his 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**

**Entry Conditions:** The user received a fine.

**Flow Of Events:**

- Receives a notice stating that they received a fine.
- Press the Tell Me More 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**

**Entry Conditions:** None.

**Flow Of Events:**

- Notice something wrong with the app or the car.

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

**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

**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 This button.

**Exit conditions:** The user receives a confirmation that they took that issue in charge and receive directions to reach it.

**Exceptions:** No predictable issue can happen at this stage.

**Name:** Unlock and manage car taken in charge

**Entry Conditions:** The operator is logged in and took an issue in charge.

**Flow Of Events:**

- Press the Unlock car button.
- 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 again or give up the issue.

**Name:** Mark an issue as solved or give up an issue

**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.

**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 the operator is given the possibility to give up the issue.

**Name: Change car's Exception status**

**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**

**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

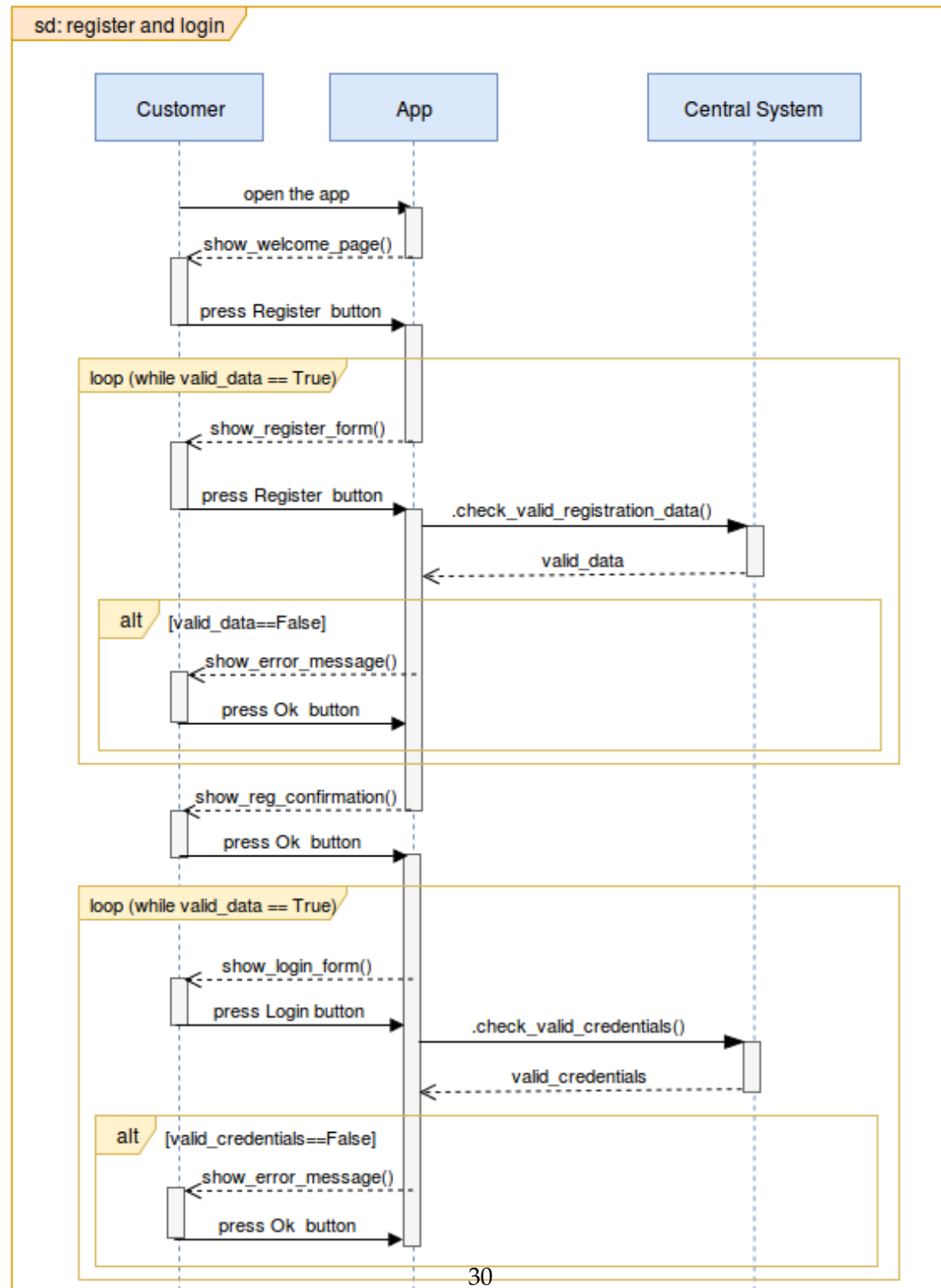


Figure 5: Registration and Login Process

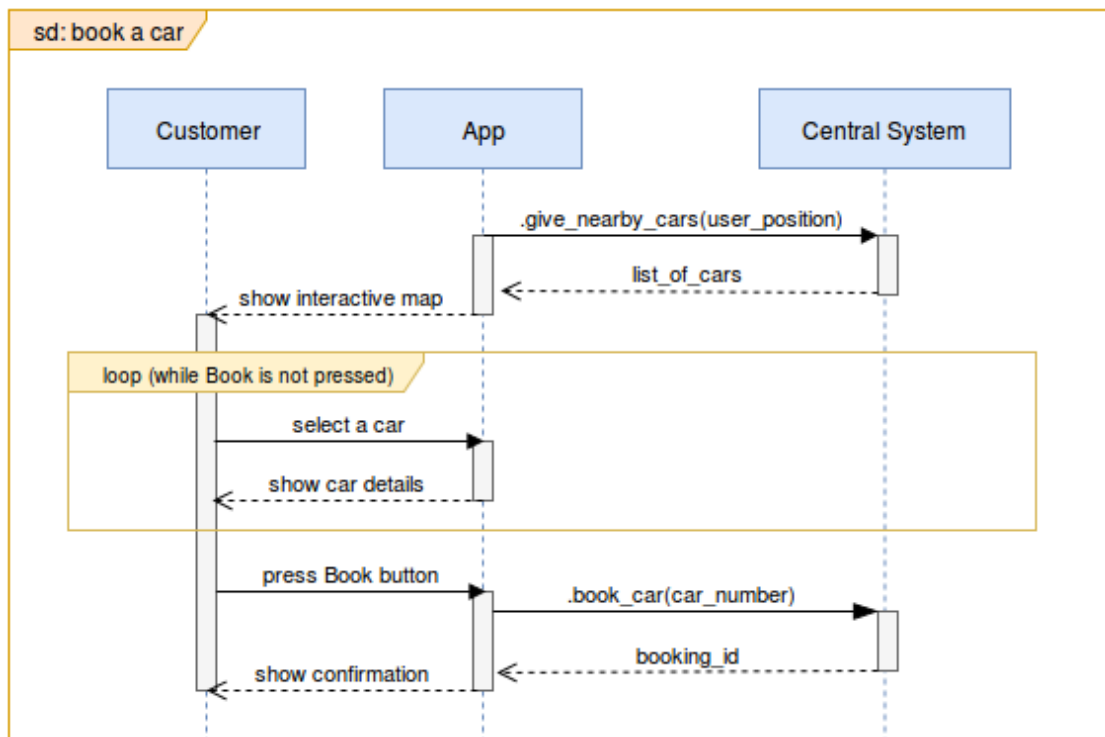


Figure 6: Booking Process

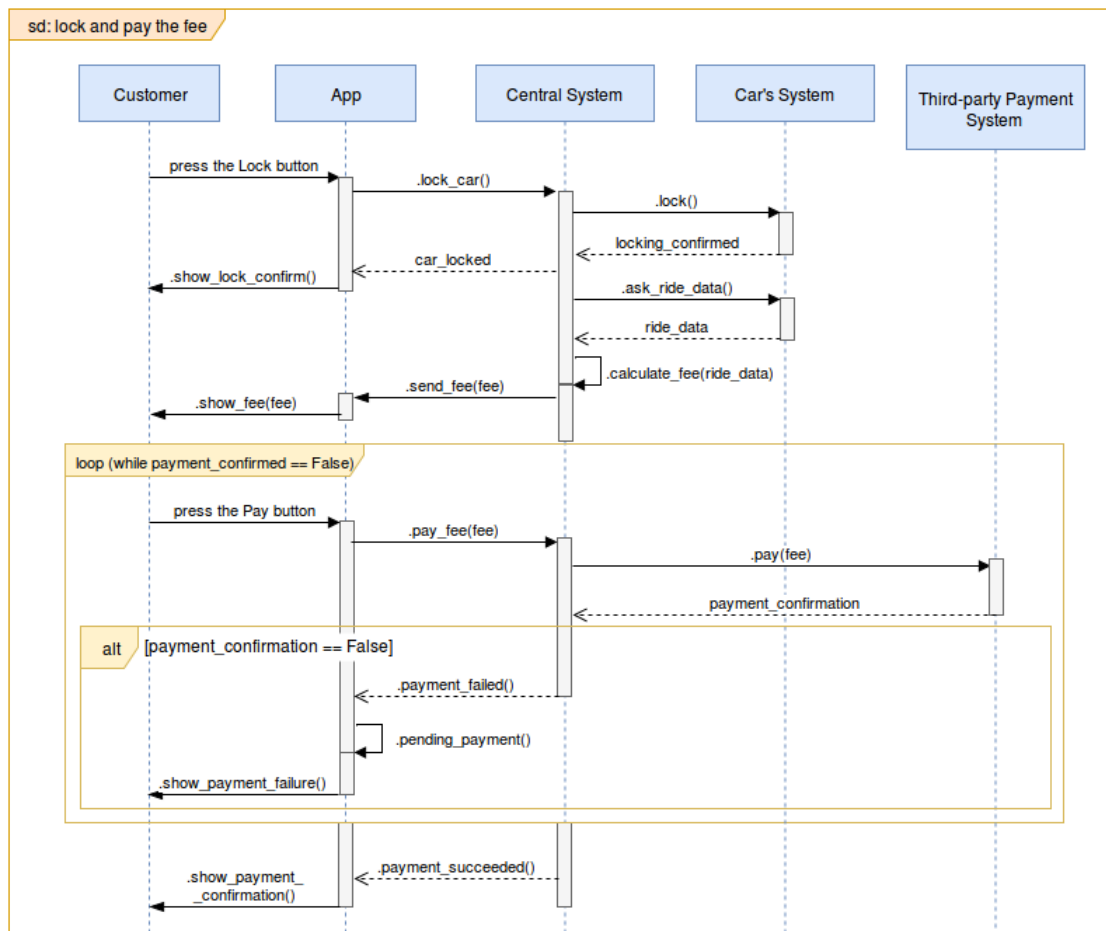


Figure 7: Locking and Paying Process



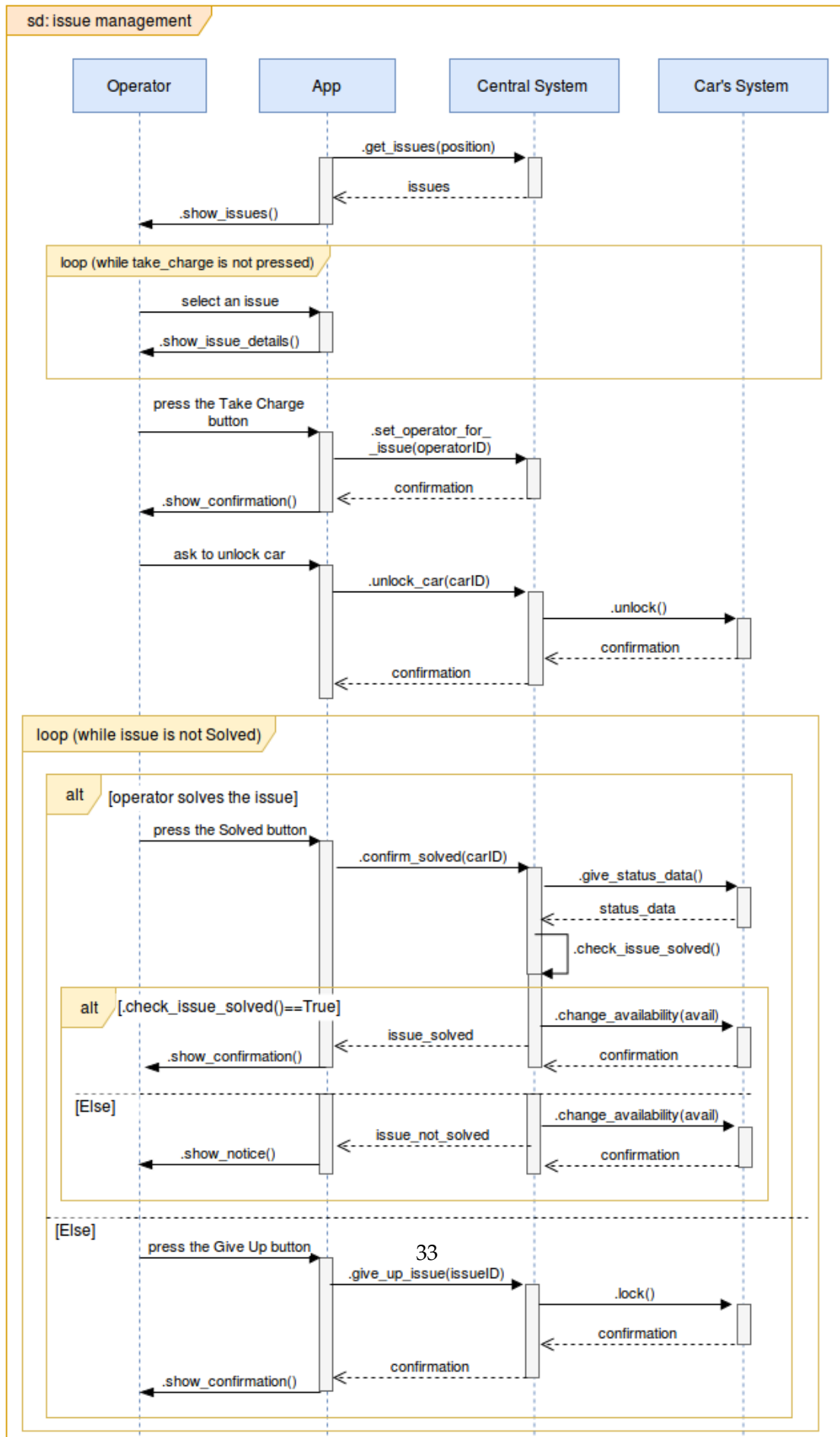


Figure 8: Issues Management Process

## 4.4 Activity Diagrams

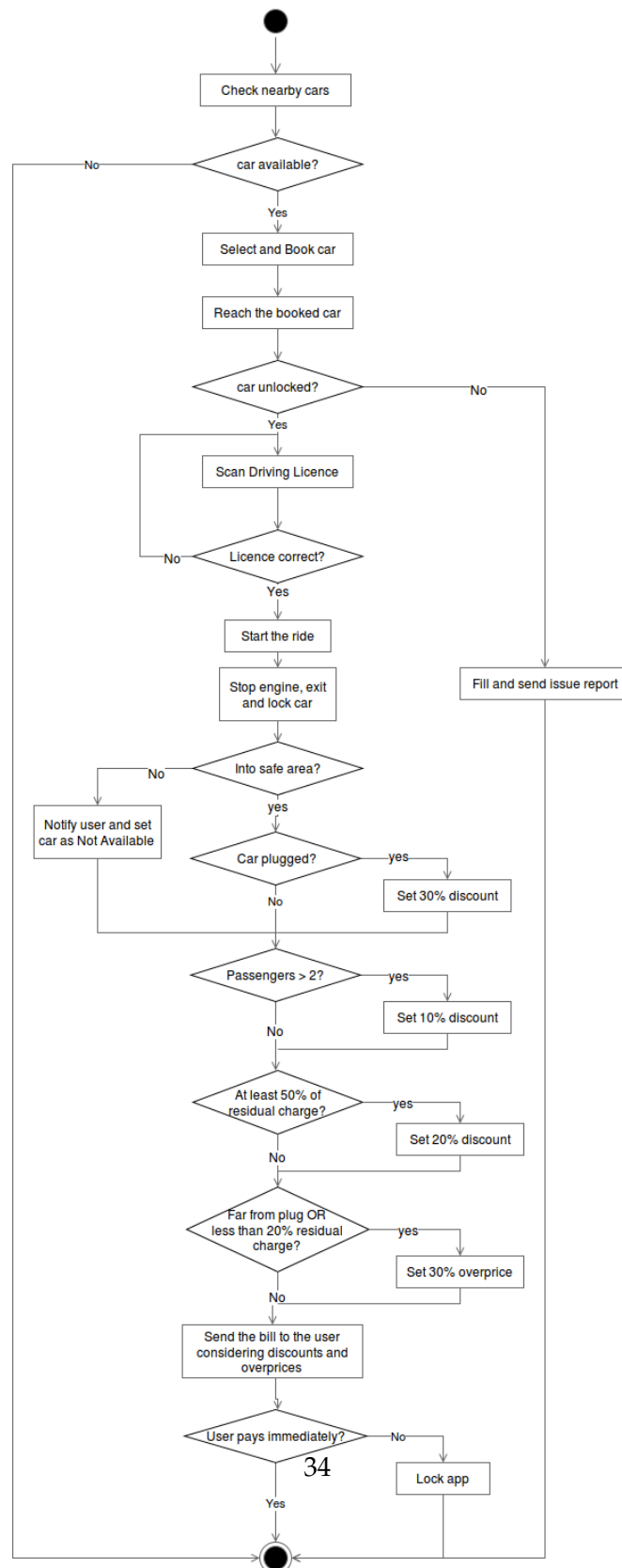


Figure 9: Booking Flow

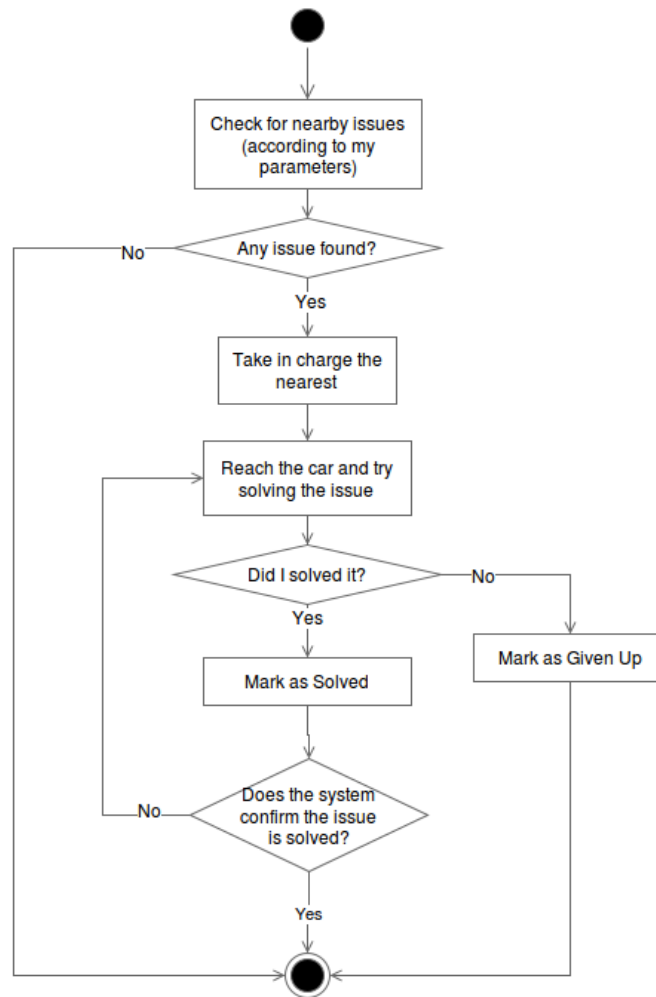


Figure 10: Issue Management Flow

## State Diagrams

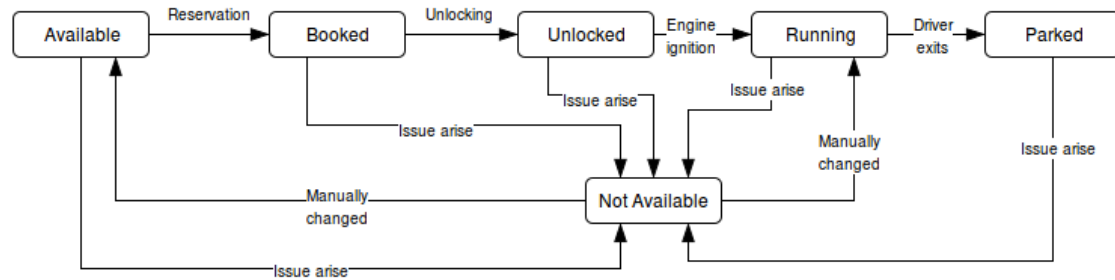


Figure 11: Car's Availability Status State Transitions

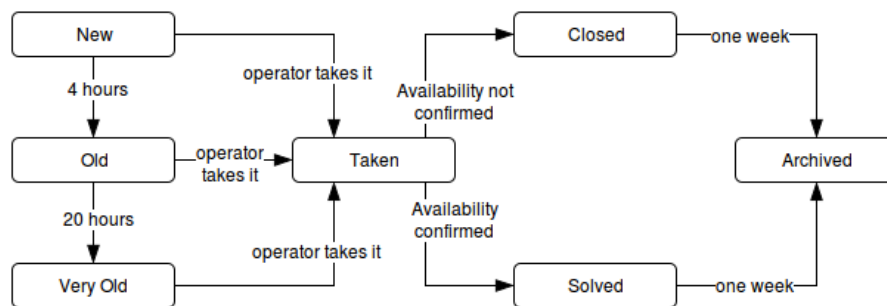


Figure 12: Issue's State Transitions

## 5 Alloy Model

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

In order to keep the model itself manageable it was first developed in small chunk and its validity checked, then each chunk is been merged in a bigger model and the validity of the whole model is been re-check again.

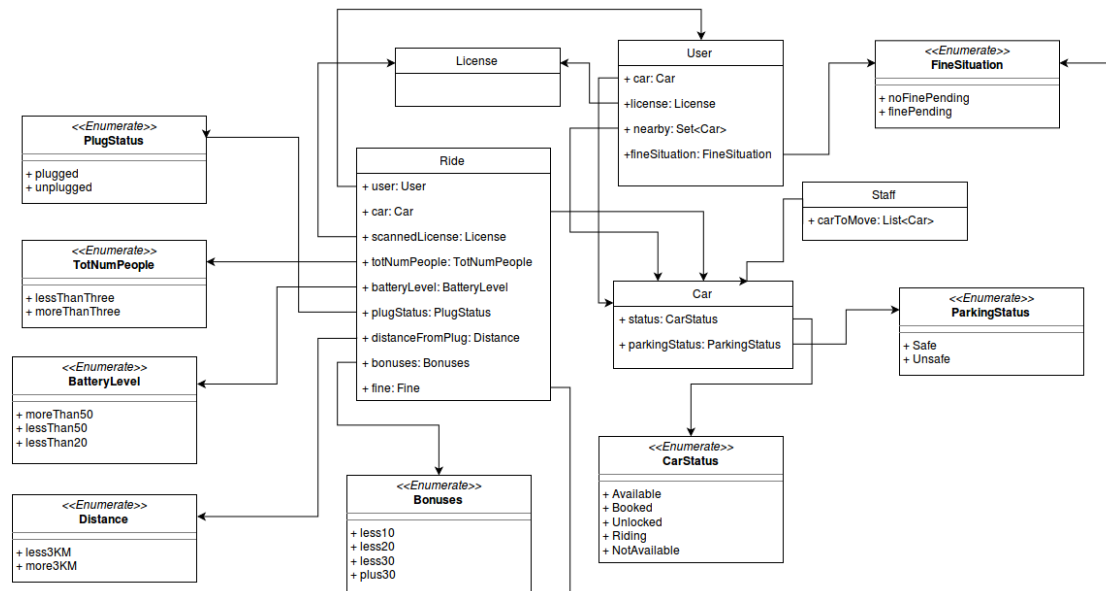


Figure 13: Class Diagram for the Alloy Model

### 5.1 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  $\wedge$  not no u.car
}

fact licenseIsPersonal {
    no u, u': User, l: License | u  $\neq$  u'  $\wedge$  l in u.license  $\wedge$  l in u'.license
}

fact OnlyCarRidingDontHaveAParkingStatus {
     $\forall$  c: Car | c.status = Riding  $\iff$  no c.parkingStatus
}

fact CarUnsafelyParkedAreOnListToBeMoved {
     $\forall$  c: Car, s: Staff | c.parkingStatus = Unsafe  $\iff$  c in s.carToMove
}

fact CarUnsafelyParkedAreNotAvailable {
     $\forall$  c: Car | c.parkingStatus = Unsafe  $\iff$  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 {
     $\forall$  c: Car | some u: User | c.status = Booked  $\iff$  c in u.car
}

fact AnUnlockedCarIsAssociatedWithAnUser {
     $\forall$  c: Car | some u: User | c.status = Unlocked  $\iff$  c in u.car
}

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

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

fact RideOnlyIfTheScannedLicenseIsTheSameOfTheUser {
    no r: Ride, u: User, l: License | r.user = u  $\wedge$  u.license = l  $\wedge$  r.scannedLicense  $\neq$ 
    l
}

fact CarCanBeBookedOnlyByUser{
     $\forall$  c: Car, b: Booked | some u: User | b in c.status  $\iff$  c in u.car
}

```

```

}

fact AvailableCarDontHaveUser {
    no c: Car, a: Available, u: User | a in c.status  $\wedge$  c in u.car
}

fact CarCanBeBookedOnlyByASingleUser {
    no c: Car, b: Booked, u, u': User | b in c.status  $\wedge$  u  $\neq$  u'  $\wedge$  c in u.car  $\wedge$  c in u'.car
}

fact OnlyOneRideForOneCar {
    no c: Car, r, r': Ride | r  $\neq$  r'  $\wedge$  r.car = c  $\wedge$  r'.car = c
}

fact RideOnlyWithRidingCar {
    no c: Car, r: Ride | r.car = c  $\wedge$  c.status  $\neq$  Riding
}

fact NoRidingCarWithoutRide {
     $\forall$  c: Car | some r: Ride | c.status = Riding  $\iff$  c in r.car
}

fact RidingWithTheSameCarUsedByTheUser {
    no r: Ride, c: Car, u: User | r.car = c  $\wedge$  r.user = u  $\wedge$  u.car  $\neq$  c
}

fact CarCanBeRidenOnlyByAnUser {
    no r: Ride, c: Car, u, u': User | u  $\neq$  u'  $\wedge$  Riding = c.status  $\wedge$  u = r.user  $\wedge$  c =
    r.car  $\wedge$  c = u'.car
}

fact UnlockedCarHasIsUserNearby {
     $\forall$  c: Car | some u: User | Unlocked = c.status  $\iff$  (c in u.nearby  $\wedge$  u.car = c)
}

fact RidingCarHasIsUserNearby {
     $\forall$  c: Car | some u: User | Riding = c.status  $\iff$  (c in u.nearby  $\wedge$  u.car = c)
}

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

    a in c.status
    c not in u.car

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

fact discountNumPeople {
     $\forall$  r: Ride | moreThan3 = r.totNumPeople  $\iff$  less10 in r.bonuses
}

fact discountBatterLevel {
     $\forall$  r: Ride | moreThan50 = r.batteryLevel  $\iff$  less20 in r.bonuses
}

fact discountPluggedCar {
     $\forall$  r: Ride | plugged = r.pluginStatus  $\iff$  less30 in r.bonuses
}

```

```

fact overchargeUnloadFarAway {
     $\forall$  r: Ride | lessThan20 = r.batteryLevel  $\wedge$  more3KM = r.distance  $\iff$  plus30 in r.bonuses
}

pred show{}

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

assert NoFineWithoutResponsibleUser {
    no r: Ride | r.fine = Fined  $\wedge$  no r.user
}

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

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

assert UserAreDrivingACarOnlyIfTheScannedLicenseIsTheSameOfTheUserLicense {
    no r: Ride, u: User, l, l': License | l  $\neq$  l'  $\wedge$  u.license = l  $\wedge$  r.scannedLicense =
l'  $\wedge$  r.user = u
}

check UserWithFinePendingCantBookUnlockRideCars for 10
check NoFineWithoutResponsibleUser for 10
check AllNotSafelyParkedCarAreInTheListOfTheStaff for 10
check OnlyCarAvailableOrNotAvailableAreNotAssociateWithAnUser for 10
check UserAreDrivingACarOnlyIfTheScannedLicenseIsTheSameOfTheUserLicense for 10
run show for 5

```

## 5.2 Results of Execution

**9 commands were executed. The results are:**

- #1: No counterexample found. UserWithFinePendingCantBookUnlockRideCars may be valid.
- #2: No counterexample found. NoFineWithoutResponsibleUser may be valid.
- #3: No counterexample found. AllNotSafelyParkedCarAreInTheListOfTheStaff may be valid.
- #4: No counterexample found. OnlyCarAvailableOrNotAvailableAreNotAssociateWithAnUser may be valid.
- #5: No counterexample found. UserAreDrivingACarOnlyIfTheScannedLicenseIsTheSameOfTheUserLicense may be valid.
- #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: One of the model returned by Alloy, compressed visualization.



### 5.3 Generated graph

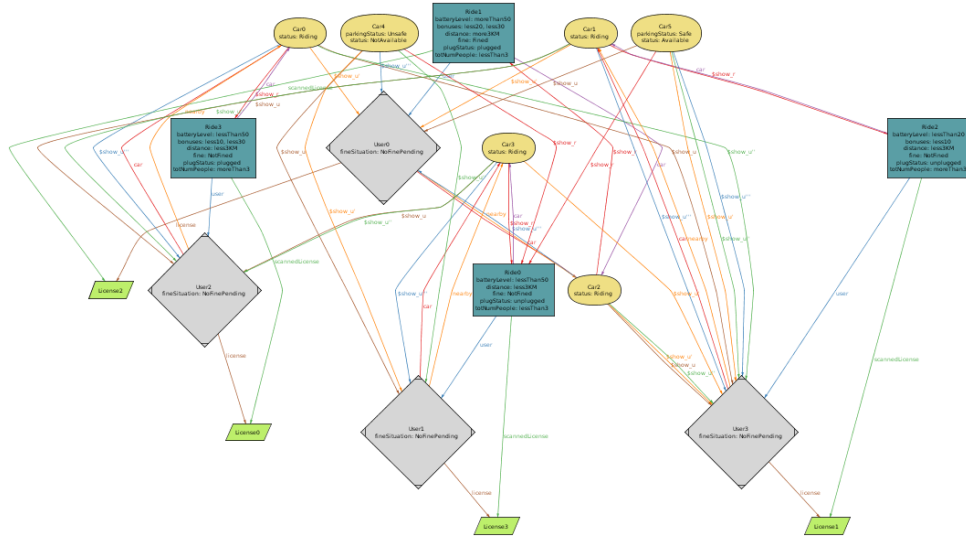


Figure 15: One of the model returned by Alloy, compressed visualization.

### 5.4 Dynamic Modeling

Finally, in order to check also the validity of our solution also in a dynamic case we develop another smaller, but dynamic Alloy 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  $\wedge$  c in u.car
}

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

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

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

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

    c'.status = Booked
    u'.car = c'
    u.nearby = u'.nearby
    c not in u.nearby  $\iff$  c' not in u'.nearby
}

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

    c'.status = Unlocked
    u'.car = c'
    u.nearby = u'.nearby
    c not in u.nearby  $\iff$  c' not in u'.nearby
}

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

pred PossibleToRideACar[c, c': Car, u, u': User]{
    c  $\neq$  c'
    u  $\neq$  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  $\iff$  c' not in u'.nearby
}

pred PossibleToReleaseACar[c, c': Car, u, u': User]{
    c  $\neq$  c'
    u  $\neq$  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  $\iff$  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

```

**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: One of the model returned by Alloy, compressed visualization.

## 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
- **L<sup>A</sup>T<sub>E</sub>X** to redact this document
- **Alloy 4.2** to check model's validity
- **www.draw.io** to draw UML graphs
- **Gimp** to mockup the application

### 6.3 Hours of work

- SZ: 4h on 1/11
- SZ: 4h on 4/11
- SZ: 6h on 5/11
- SZ: 4h on 7/11
- SZ: 2h on 10/11
- SZ: 2h on 11/11
- SZ: 2h on 12/11