Requirements Analysis and Specifications Document

Project: Power EnJoy

Course: Software Engineering 2

Document: RASD (v. 1.2)

Authors: Crovari Pietro, Gnecco Enrico

Contents:

    a. The model
    b. Our world
    c. Some results
7. Hours of work

1. Introduction

a. Informal description

Power EnJoy implements the concept of taking cars just for what needed: the users are able to look for a car, eventually use it and leave it whenever they want; Power EnJoy is the answer to almost everyday problem.

Friendly. Fast. Secure. Cheap. Eco.

Power EnJoy aim to solve problems, not to generate them.

What does this mean?

Every car in the system is eco-friendly, in fact they all are electric.

Moreover the correct usage of those is incentivated by discounts and hints appropriately chosen to give the maximum effort to the user.

The system allows users to reserve cars. The system will provide a way to find cars within a certain area from the place either inserted by the user or suggested using user's position, found by GPS.

As soon as the vehicles are used a bill is increased.The payment method is provided using information previously given by users.

Discounts or extra payments are strictly related to the usage of cars, in particular respectively to the number of people inside, the final level of the battery and how far from a grid station the car is left.

i. The world before and after the application

Pollution, parking problems, comfort, queues and traffic jams must seriously considered, in order to fully understand the benefits deriving from Power EnJoy. Indeed, the flexible usage of electric cars, the service offered, aims to at least decrease such these problems, giving users the chance to move inside the region covered by the system without

consuming oil, or creating traffic jams, but, instead, having free parkings and a comfort that only modern cars can offer.

b. Goals

The application has the following goals:

- To create a user-friendly system that lets people book and use cars, offering to people the possibility of using vehicles only for their actual needs
- To manage a system able to face problems regarding the distribution of cars, the work by the operators and the user demands.

c. Domain Properties and Assumptions

In order to work within a certain environment it is necessary to have particular details on certain views of the world or of the system already present, however the choice of using the old system or not is postponed to the design step, defined in the Design Document:

1) The system owns cars; each one is connected to the central module of the system through a always-valid connection; each car also has a screen to communicate with the user and show him a map with eventually some underlined points given by the central module; each car has engine sensors, battery sensors and all of them are connected to a small calculus unit in order to give the right information to the user and, through the connection, to the system; each car has an intelligent camera inside the cabin, which is always monitoring the situation and is able to understand the number of the people inside in every moment; each car has a little QR code printed on the front that uniquely identifies the car; each car has an assurance for incidents, damages even if the system will keep track of the users and the used cars (such as “kasko” policy).

2) The locking logic of the car is already present: if the camera detects that the car is empty, it is idle, the engine is turned off for 1 minutes and the vehicle is parked in a safe area, then the vehicle is locked. Otherwise, the user goes on paying until all these conditions are respected.

3) There is an already existent management system which provides "safe" areas, the ones in which the user can park, and "recharging-safe" areas, the ones in which the vehicles can be parked and recharge. In addition, it's assumed that there are some operators that can go to recharge cars or to take them to be fixed up. The interface for those operators will be provided by Power EnJoy system.

4) All the cars are always able to communicate with the central module in every moment though their connection. In other terms, the connection is always reliable.

5) We assume that the town has a city center, where the traffic is limited only for authorized vehicles (not Power EnJoy ones). "Safe parkings" and "Recharging safe areas" are assumed to be with a uniform distribution, all over the covered area.

6) Since the environment where we are going to deploy is a city, the traffic is assumed not to be stable; the concentration of cars near the city centre will increase in the mid of the day, while it will decrease during the morning and the evening. It could be further modeled with a shape similar to a gaussian, whose standard deviation changes during the day. As we mentioned before, this underline the concentration of cars around the city center during the mid of the day. This could be a serious problem for QoS of Power EnJoy, because this distribution generates a lower supply of vehicles as we get further from the city center. However, we state that this won't be a problem: during that hours, in stationary stochastic conditions, we can imagine to have a situation very similar to the one described by the Little's law: as many cars go towards the city center as many move in the other direction.

Since this trend is leaded by the demand the conclusion is that this attitude won't be a problem for the system.

7) The system can check the validity of every driving license in every moment. This is done through the police system which provides an appropriate interface.

8) We can check the validity of every payment method. Those can be used in every moment through an interface offered by the respective bank systems, or something similar to PayPal, and it's APIs

9) It's assumed not to be possible for car batteries to go down or for engines to break up when parked; in other terms there can't be a situation in which a user turns on the car and this doesn't start.

10) We assume that every car is always able to be easily recharged (e.g. by connecting a plug to the recharging column) and the recharging system can't be stolen or damaged.

11) The battery of the cars can't go down if they are parked. In other words, sensors and all the data management system consume a negligible amount of energy.

12) It's assumed that the operators will fix a battery problem simply by changing it in loco. Once the operation is completed, the car will become available gain.

13) It's assumed that for every other kind of problems related to maintenance the operators will take the cars to the appropriate place where to fix them.

14) It's assumed that if the car is moved by an operator his aim is to fix some kind of maintenance problem, that could be heavy, light, or both of them. Once the car has been fixed, every kind of problem is solved.

d. Glossary

- Car: all the electric vehicles belonging to Power EnJoy's fleet, that can be booked by Users
- User: every signed up person who wants to interact with the System
- Operator: technicians hired by Power EnJoy, encharged to repair broken or damaged vehicle or charge out of batteries one
- Maintenance: all the action aimed to improve, or fix the situation, of a car
- To start a job: an operator takes a car to repair it to fix a problem on it
- To end a job: an operator finishes working on a car.
- System: the one we are going to deploy
- Segnalation form: a form that Users must fill in before being allowed to ignite the motor of a car
- Final form: a form that the user must compile at the end of a rent if there are some relevant problems to the car which prevent him from a normal usage
- Fee: the amount a User has to pay when a rent finishes

e. Assumptions and Decisions to avoid Ambiguity

Once that all the environmental details are clarified or assumed, it's necessary to face problems deriving from ambiguous situations, which must be cleared up singularly:

1)Every car must be booked first and then taken to be used (the booking time can be just 1 sec), this aspect will be formally modeled later.

2) Every available car can be booked, no matter the distance from the user. Even if the booking time last in one hour and the time to reach the car is greater than that.

3) The payment of a fee of 1€, due to the timeout of the booking time, is done just after that hour. There is no extra time.

4) If the user finds the car with less than the 50% of the battery full, then he has the discount if and only if he goes to recharge it.

5) The user is notified when the battery goes under the 20% and when the car is left 3km away from the nearest power station, so that he can move it in order to avoid the extra-charge.

6) While searching for a car, the user is able to see an overview for each vehicle; in particular: the position on the map, the level of the battery and the plaque number.

7)If a user unlocks a car and uses the car without moving it, or also without turning up the engine, then after 10 minutes he will have to pay per minute, as if the engine had started. This politics we decided to adopt, aims to avoid an improper use of vehicles

8) If a User books a car, it is not able to undo it until the reservation expires, one hour later. In this way we would like to encourage people to a good behaviour, avoiding flash and useless reservations.

f. Constraints and Problems

Power EnJoy presents some problems that we assumed not to block the natural flow of the system. The most important of them are here exposed in order to stress, from a certain point of view, weak aspects of the whole

system. In addition, further developments for the application will be compromised if they won't consider these aspects, also considering the limited budget.

i. Hardware limitation for cars

Hardware is expensive. In particular we assume to have "smart" cars, which can always communicate with our central module, our server, through an internet connection. In addition, we stated that every car has a small calculus unit and a "smart" camera. All of these are related to a screen that notify the user. Here the situation is particularly stressed, indeed there is a great investment to provide those and this can be one of the weakest points of the system. However it has been assumed that all of these aspects are already present in our world, so that is not a problem for the application, but it must be taken into account for further development.

ii. Safe areas

The location of safe areas is already present in the world, but it's necessary to underline the possible problems coming from an extra flow of cars. As it has been assumed, during central hours of the day a great percentage of the cars tends to go towards the city center; it's evident how the lack of safe areas could cause a great deal to our application and to the QoS.

iii. Interfaces to other applications

Power EnJoy system manages the interfaces with other applications, in particular, credit cards and driving licenses checks, so costs for maintenance of the system must consider also these aspects.

iv. Trust in the users

Car assurances are provided, but a log must be built. The reason is simple: the system assumed various situations not to create problems, but those situations are almost avoided if the cars are not physically damaged. How can the system detect who damaged a certain car? The simple answer is keeping track of all travels, for example by implementing a log of user-car tuples with attributes like starting point and time, end point and time, number of people inside, discounts or similar, eventual changes during the usage. Every user is responsible for damages of the car inside and outside during the usage, and this is done through the log built as what concerns user fines.

g. Basic idea of the suggested system

Even if the architectural design will be postponed to the Design Document (DD), here it's possible to observe some useful aspects in order to better understand the possible answers to the requirements that Power Enjoy application needs.

The main idea is to have a module and many devices interacting with it: on a side all the users and operators' devices, on the other all the car devices. This is the application to develop, and this must communicate with an already present management system.

The simplest conclusion is to build a central module, strictly connected to the management one, which deals with all the connections coming from the devices.

As anticipated, the recommended system is centralized: a server with a connected database is suggested in order to manage all the information.

From the user point of view the suggested portal is a web application because it offers services, like geolocalization, maps and others, and because of its portability and its capability to adapt to other applications or middlewares.

From the cars' point of view, it's not easy to understand when the user is near to the car in order to unlock it. This problem will be solved in the next

more detailed chapter. However, the cars basically offer information strictly related to the sensors and the camera. Except from information displayed on the screen, there are no other interaction between car and user, apart from the passive usage of the car: the user can't communicate to the server through the car.

h. Some more details

Here special cases or more details are explained.

One of them is related to the unlocking logic of the car: indeed the system, as it will be shown later, needs to understand whenever the user is close to the reserved car, in order to unlock it.

The suggested answer is to insert on the front part of each car a QR code previously generated; once the user is near to the car he can take a picture of the code through the web app, and send the translated information to the server. This will send an unlock message to the IP address of the correspondent car.

This choice offers a simple solution to a more sophisticated problem. Other solutions could be:

- GPS location, but it's not as precise as needed;
- A small keyboard on the car, but it's pervasive and expensive;
- Bluetooth connection (or similar), but the user can have problems with interferences, due to the rain or to a problem related to the antenna.

In conclusion, we strongly believe that the qr code is the simplest an most effective solution

Another important detail is related to the car information, indeed we assume that those are always reliable as the connection. In particular, our server needs to know at least these data for every car: the code of the car, the number of people inside, the battery level, the temporary fee,

calculated only on the duration of the ride. If those are missing there can be errors in coherence or issues that will interfere with the user experience. The messages are sent by the car every time that a relevant event occurs.

The last consideration is the one related to the damages to the car. Indeed, every time a car is unlocked, the respective user has to compile a little form where there is declared:

- If the car is okay or not
- If yes, trivial case
- If not, which type of damage and if that is blocking the usage of the car
- If yes, the car will stop, the server after the submission will block the car and mark it as “out of order”
- Otherwise the user can go on but the car will be marked as “need_maintenance”

If the user lies, he will risk to be in a dangerous situation: if the car breaks up while he is using it, the user will have to pay for that, unless a further analysis proves that the failure fault isn’t related to the user.

The user is responsible only for the damages that occurs during his ride, other kinds of damages are covered by the assurance.

The engine can start if and only if the compilation of the form is successfully completed, the car will be in an idle state up to that moment.

i. Stakeholders

The identified stakeholders are the owners of Power EnJoy: people that own the cars and those who want to improve the city services. Overall the stakeholders are the owners of the company and the government of the city, because they can improve the quality of life of citizens, the transport one and they can create new workplaces. Moreover, the can win several subsidies for bringing an eco-friendly system in their cities

In addition, we could consider as minor stakeholders users who join the services, and companies that supply energy and vehicles, because they will sign very important contracts.

j. Reference Documents

- “Assignments AA 2016-2017.pdf”
- “IEEE standard on requirement engineering.pdf”

2. World and Machine: the actors

The problem of identifying the actors of the world is as simple as we go on splitting the functionalities of the world into modules or objects.

These means that we have:

- Cars, and the information they provide. we decided to consider them as actors, even if not real people because, as we said, they are “smart”, so they have a crucial part in all the processes we are going to implement.
- Users, and their relation with the portal provided by the system
- Operators, who interact with the system in order to work to damaged or broken cars

3. Requirements

Following the document given, requirements are extracted and formalised in order to fulfill the goals assumed particular properties of the world.

a. Functionalities

Functional requirements are extracted and grouped here in order to underline the specifications that Power EnJoy has to satisfy.

1) During the signup process, the system, once that the user gave all the valid information needed, must communicate the password to him in order to let the log in.

2) At the end of the registration, all the data collected must be convalidated by the system and all the information must be coherent; that means that driving licence information must be coherent with the identity information declared.

3) Among these information there must be something to let the system being able to communicate somehow the password. Since a web application is suggested, an email address is recommended.

4) The system must provide a way for storing all the information. A database is suggested.

5) The system will have to be able to generate passwords.

6) Every user must be able to look for cars within a certain distance from a certain place, either the actual position of the user or one given by him; however finding one is not assured.

7) Users must have the possibility of accessing to the system in mobility, for instance when they reach the car they booked and they want to unlock it.

8) Every available car must be able to be booked for at most one hour and either released or used.

9) The system must be able to count the bill for the user.

10) The user must be notified through the screen of the car about the amount to be paid. At the end of the course that amount will be sent to the server which, once having calculated possible bonus and/or malus on the fee, will provide the correct payment.

11) The system must understand when the car is left in a safe area and when the user goes out of the car, then, after 1 minute the car must be locked.

12) The system will provide the correctness of the payments.

13) The system must be able to store all the positions of the safe areas and where to recharge the cars.

14) The system must build an interface for other applications, one for the payments, one for the validity of the driving licenses.

15) The system must also build a basic interface for operators: a simple map, or something equivalent, to show to operators the location of those cars which need assistance or maintenance and their actual needs; in addition, there must be a start and a finish button, through which the operators will be able to declare the beginning and the end of their works touching them.

16) The system must allow the operators to declare the beginning of a work on a specific car, preventing it from resulting in available cars set.

17) Operators must have the possibility of reaching information about a car near to him

18) The system must allow the operators to declare the end of a work on a specific car, either making it available again, if it is fully repaired, or preventing it from resulting in available cars set, highlighting its needs.

19) System must provide operators a Form to report ends of jobs. These data must be collected in order to have an history of the interventions.

We think that resuming previous points about the operators is important to a better accomplishment: the system must provide an interface to the operators. This interface must give the information regarding the position and the situation of all the cars, a map is suggested. In addition, there must be 2 other features: a button to start and one to end a job. The first one will let the operator take a picture of the QR code; connecting to the central module of the system, all the information of the car will be available and the job on that car by that operator will be registered as started. The second one will register on the central module the end of the job of that car by that operator, but there will be the need to specify if the car is available or if further work on it is needed; moreover the operator must declare what was needed, what has been done and a second picture of the QR code will be mandatory in order to know which vehicle is being considered and to simplify coherence checks.

20) if the user is not able to accomplish a payment, the system won't allow him to reserve a new car until the payment is not completed successfully.

21) Every time that the state of a car changes, this must be recorded both by the server and the corresponding car.

22) An User must have the possibility only to book one vehicle per time. This helps keeping a higher number of available cars.

23) Before a car which is charging becomes available again, its battery must be fully charged. In this way we can decrease the number of charging required, also minimizing the possibility that an Operator has to intervene.

24) An event of "battery down" must be differentiated by an "out of order" one, because the first one requires an immediate and quick intervention on the spot, whereas the second one requires a longer and more complex operation, that must be executed in a suitable location.

25) Cars must be able to count the bills, according how much time users spend on them.

b. Non-functionalities

Non-functional requirements are here grouped, QoS specifications and other properties that Power EnJoy have to satisfy are the following:

1) The system must be able to count the bill for the user as soon as the engine ignites.

2) The user must be notified through the screen about the amount to be paid in real time.

3) There must be coherence among the bills on all the in-use cars.

4) Once the system understands that the user took at least two other people, then a 10% discount will be applied on the last ride.

5) Discounts are applied on the whole time of car usage.

6) The system has to calculate and apply the 20% discount before the user payment if the battery left is more than 50% of the total.

7) If a car is left at special parking areas where they can be rechar ged and the user takes
care of plugging the car into the power grid, the system applies a discount of 30% on the last ride.

8) If a car is left at more than 3 KM from the nearest power grid station or with more than 80% of the battery empty, the system charges 30% more on the last ride to compensate for the cost required to charge the car on-site.

9) If the car remains turned off for more the 10 minutes once it has been unlocked, it starts to charge the user with a fee consisting of 80% of the one used when the vehicle is in use.

10) User should see the amount of the discount in order to feel satisfaction in good behaviour

4. Scenarios

a. Scenario 1 – A new user

Mark is studying Computer Science and Engineering at Politecnico di Milano.

After the lessons Mark wants to go home, which unfortunately is on the other side of the city; it's raining and he doesn't want to get wet and not feel uncomfortable on the bus with all the other people.

Mark decides to use Power EnJoy through his mobile phone but he's not registered.

Through the web application he succeeds to reach the registration form.

Name, surname, driving license and credit card information are needed. In addition all the information about fees or discounts are now given.

Then he is be able to access to the map where all the available cars near his position are given.

He will go back home perfectly dry.

b. Scenario 2 – Booking a car

Pablo is a great user of Power EnJoy, everyday he chooses a car near his house to go to work.

Using the map he see all the possible cars, with their information.

In order to book one of those he has just to touch the button over the corresponding vehicle.

Once that the car is reserved he has up to one hour to reach its position, otherwise he knows that he would have to pay a 1€ fee, but he knows that's not a problem though because the car is near him enough.

He will reach the work office in a more comfortable, faster and cheaper way.

c. Scenario 3 – Unlocking a car

Jim is trying for the first time the Power EnJoy system, he hasn't understand very well how it works but he succeeded in booking a car.

He goes to the position given by the application service. Once there he notices a QR code on the car and uses the corresponding service on the web application in order to try to understand what to do.

The guess is right, indeed he receives a feedback and the car is unlocked.

Once inside he has to compile a little questionnaire about the situation of the car. After that, the engine can start.

d. Scenario 4 – Parking a car

Katy's been driving for the whole afternoon picking up her little brothers.

Almost exhausted she notices that the battery is close to be down, then, since she wants an additional discount, she decides to go to the nearest station to recharge the car.

Once that she's in the safe area for recharging the car she can go out of the car, the system will lock automatically.

After having attached the car to the recharging system a discount will be applied on the last ride of Katy.

e. Scenario 5 – Idle situation

Giorgia is a very busy woman, she is always using her mobile phone.

She wants to move in the city, she uses Power EnJoy.

Once on the car, the mobile phone rings and she has to answer; unfortunately it's her boss and she has to argue her last project for more than 10 minutes, before having the possibility of turning the car on.

She's using a Power EnJoy car, even if that is idle. After some time the system starts charging the reduced fee.

At the end of the call she will know the bill, but fortunately, that's not too much.

f. Scenario 6 – The new operator

Bob is a new operator at Power EnJoy. He has been told to use the web app to find all the cars that need assistance or maintenance, so Bob starts with the login, giving his id_number and his password, then he reaches the map and understands where all the cars are and the needs they have.

Now Bob can go to repair or to recharge them.

g. Scenario 7 – The operator in action

Ralph is an operator at Power EnJoy.

He has already reached a car which needs something, but he doesn't remember what's needed. How can he do now?

Ralph is in the section of the web app reserved to the operators, he starts a new job choosing the corresponding button. This allows him the usage of a QR code reader. Doing so, all the related information of the vehicle are transferred. Now he knows what the car needs.

Unfortunately the car needs a great help, maintenance is needed.

Ralph drives the car to the maintenance area, where other operators will work on the problem.

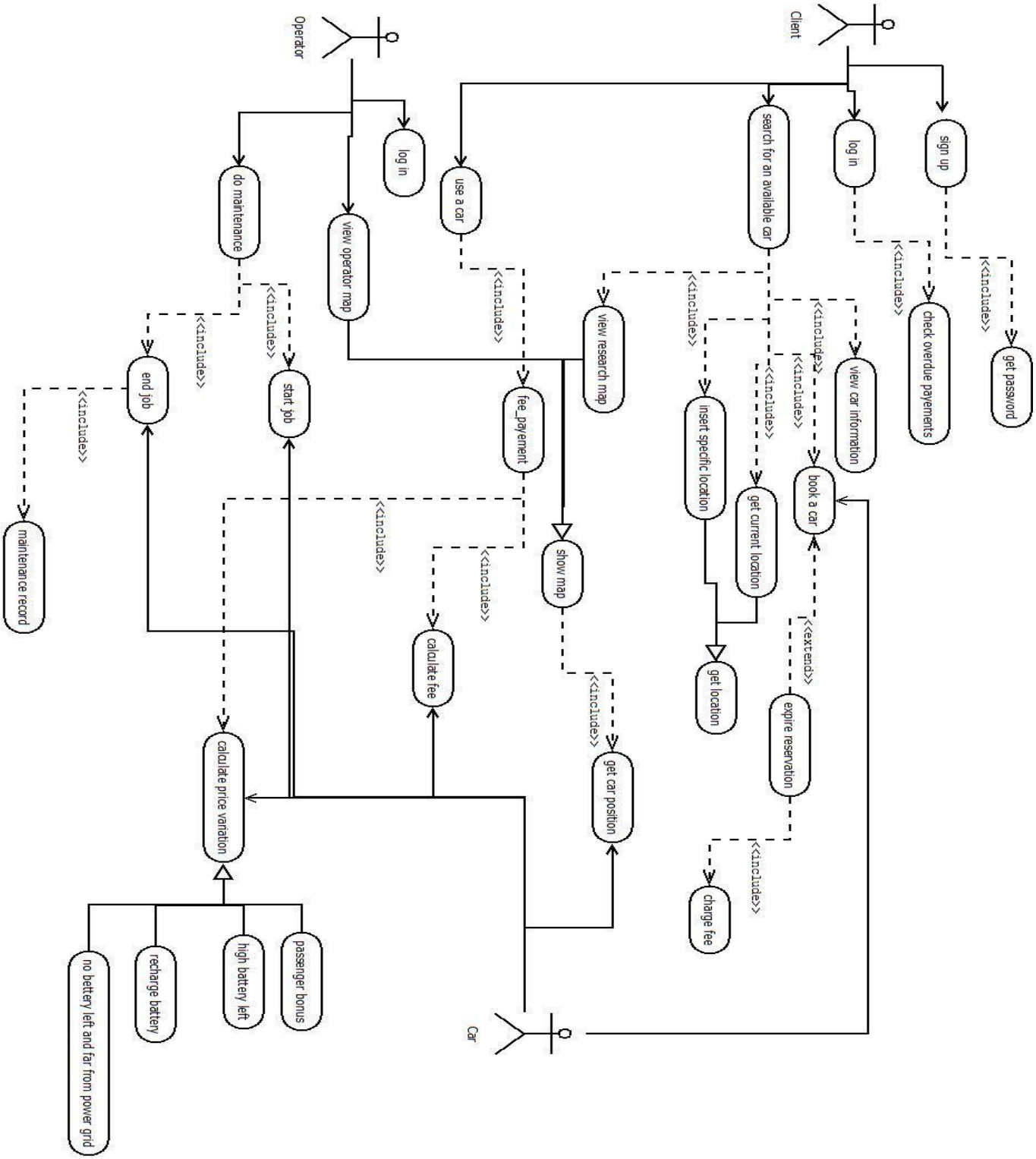
After that, Ralph will have to declare the finish of his job on the reserved part of the system, then to compile the final form, in order to notify the executed operations and the availability of the car. In this case the car isn't available yet but Ralph can go on with his work.

5. Abstracting the world

a. UML: presentation of the models used

In this chapter there will be presented some diagrams describing the main properties of Power EnJoy; from the states to the interactions all the objects considered. A formal analysis will be given with Alloy Analyzer in the following chapter.

b. Use case diagrams and description



Use Case Description:

Name: Client signs in
Actors: Client
Entry conditions: Client clicks on ‘’sign in’’ button
Flow of events:

- The Client arrives to the subscription form of the internet application;
- The Client fills the page with all his personal information;
- The Client insert all the information about his/her driving license;
- The Client insert his payment information;
- The Client clicks on “register’’ button;
- The System redirect the Client to a page where it advises him to check emails;
- The System randomly generates a password and sends it to Client’s mail address and a confirmation link;
- The Client access to the website through the link.

Exit conditions: The Clients manage to access to the application through the link.
Exceptions: If information inserted by the user are not correct, he is asked to correct them before being redirected.

Name: Client search for a vehicle and books it
Actors: Client, Car
Entry conditions: Client clicks on ‘’search’’ button
Flow of events:

- The Client is asked to insert a position, or chooses to use his current one, and a maximum range;
- The System searches for available cars within the selected area;
- All vehicles which match the request are shown on a map on the Client’s device;
- The Client selects one and sees all its information, such as its position, battery left, etc.;
- The Client clicks on a button to book;

- The System marks the Car as booked;
- The System communicates the reservation to the Car;
- The System sends an email to the Client to confirm the reservation, also communicating the expiration hour, the exact position of the vehicle and its license plate;
- The Client is redirected to a page in which he is explained to wait for confirmation email;

Exit conditions: The vehicles and the Client receive Reservation data, the Client is correctly redirected;
Exceptions: If the car results no more available, the system communicate it to the Client, inviting him to choose another vehicle. If the expiration hour occurs, the Car becomes available again, the client is notified via email and he is charged of 1EUR fee.

Name: Car is taken
Actors: Client, Car
Entry conditions: Client has booked a car and arrives nearby
Flow of events:

- The Client takes his smartphone and clicks on "take your car'' button;
- The Client takes a picture of the qr code on the car;
- The System check if the user is picking the right vehicle;
- Hence, it notifies the correct matching and order the car to unlocks the door;
- The System marks the vehicle as busy and updates the drivers' log with Client's information;
- The Client access the vehicle;
- The Client compiles the reporting form and submits it;
- The System notifies the compilation to the Car, and notifies to the Client the possibility to ;
- The Car allows the ignition, notifying it on the display;

Exit conditions: The Clients manage to access the vehicle.
Exceptions: If the Client tries to rent a car that he has not booked, the System alerts him preventing the car from unlocking improperly.

Name: Client picks up some passengers
Actors: Client, Car
Entry conditions: The Car starts revealing a change in the number of passengers in the car
Flow of events:

- The Car checks how many people are in the vehicle;
- If the number overtakes the previous maximum, its value is updated with the number of passengers in the car;
- At the end of the ride the car communicate the maximum number of passengers;

Exit conditions: The Car communicate the value correctly.
Exceptions: there is no exceptions for this case study.

Name: Client finishes the rent
Actors: Client, Car
Entry conditions: Client turns down the car in a safe area.
Flow of events:

- The Client and all his passengers leaves the vehicle;
- The Car waits the "end of service time", leaving the time to pick up eventual things stored in the luggage compartment, and potentially to connect the vehicle to the power grid;
- The Car locks itself;
- The Car communicate to the System the end of the service, and the fee calculated and all the information necessary to calculate potential discount (position, battery left, potential in charge event etc.);
- If one or more of the conditions are verified, the System modifies properly the fee;
- The System charges the final fee to the Client;
- The System notifies per email the fee, with the description of all bonus/malus details;

Exit conditions: The Clients receive the fee email.
Exceptions: If the passenger leaves a door not completely closed the car advises him with a noisy signal. If the operation is not successful, the system prevents him from hiring a new car until the payment is completed correctly.

Name: Operator checks manutention required by a car
Actors: Operator
Entry conditions: Operator opens his dedicated interface.
Flow of events:

- The Operator inserts his credentials;
- The System shows with the location of all the cars on a map, highlighting the ones that requires an intervention;
- The Operator clicks on a highlighted car;
- The System shows him the details about it (general information such as position and battery charge, plus specific information about the manutention required);

Exit conditions: The Operator visualizes the information about a Car.
Exceptions: If the Operator inserts wrong credentials, he is asked to insert them again to access to the service.

Name: Operator intervenes for a job
Actors: Operator, Car
Entry conditions: The Operator clicks on “take in maintenance” button;
Flow of events:

- The Operator takes a picture of the qr Code on the car;
- The System shows him all the information about that Car, including the maintenance required;
- The Operator confirms the operation;
- The Car changes status to “under_maintenance”, unlocks doors and enables the potential ignition (even if the engine is broken);

Exit conditions: The Operator has access to the vehicle and the status is correctly changed
Exceptions: If the engine is broken, or the damage prevents from the ignition, it will not turn on because of its status even if it is enabled.

Name: Operator finishes the job
Actors: Operator, Car
Entry conditions: The System clicks on “end maintenance” button;
Flow of events:

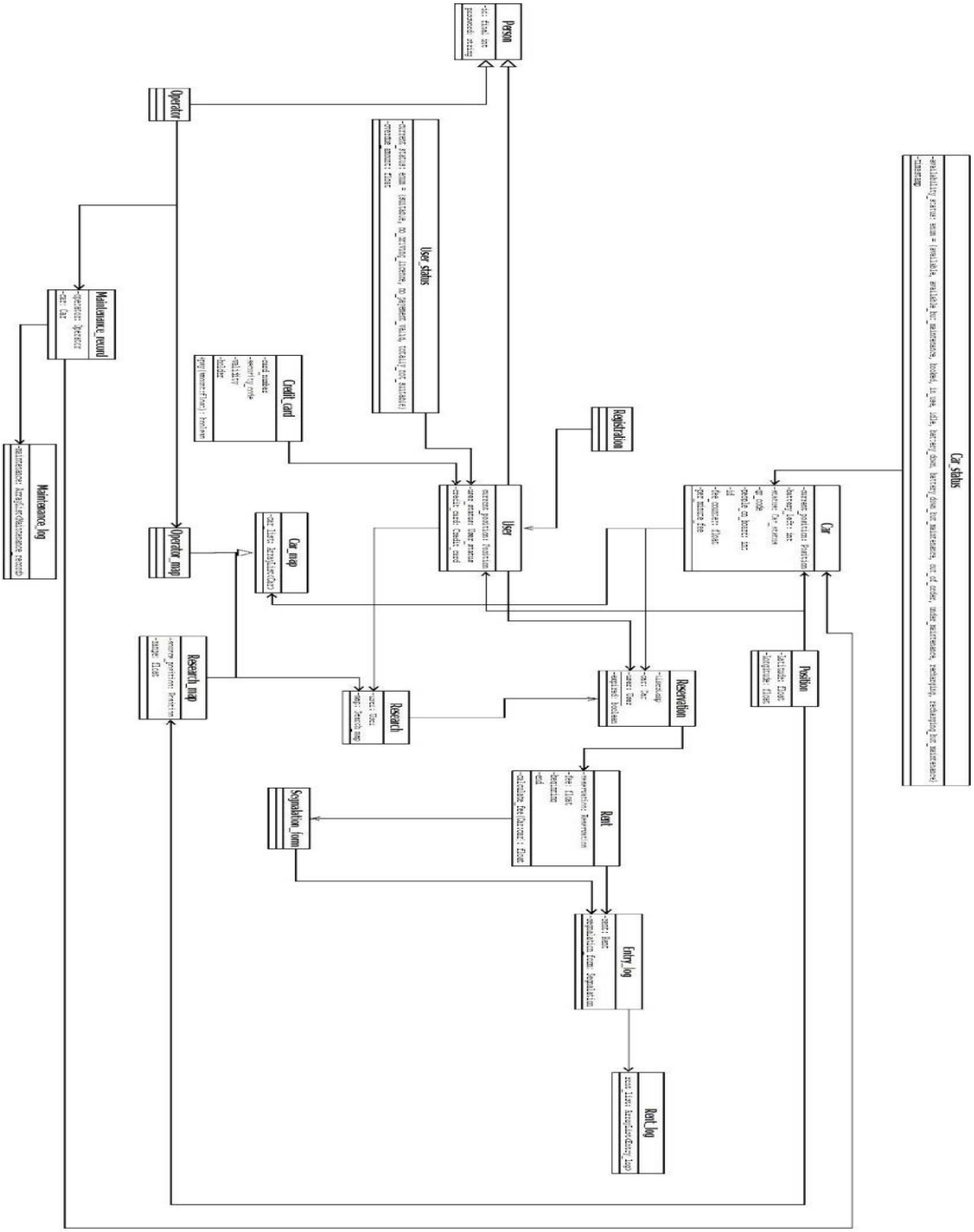
- The Operator takes a picture of the qr code on the car;

- The System shows him all the information about that Car;
- The Operator confirms the operation;
- **If** the vehicle is repaired:
    - The Operator clicks on “maintenance successful”
    - The Operator brings the car in a Safe Area;
    - The Operator leave the vehicle;
    - The Car locks the doors and change its status to available;
- **Else** (in other words, if the Car needs an intervention by another specialist):
    - The Operator clicks on “need more maintenance”
- The Operator complies the form describing all the interventions done and the possible other intervention to do;

Exit conditions: The Car change its status to available.

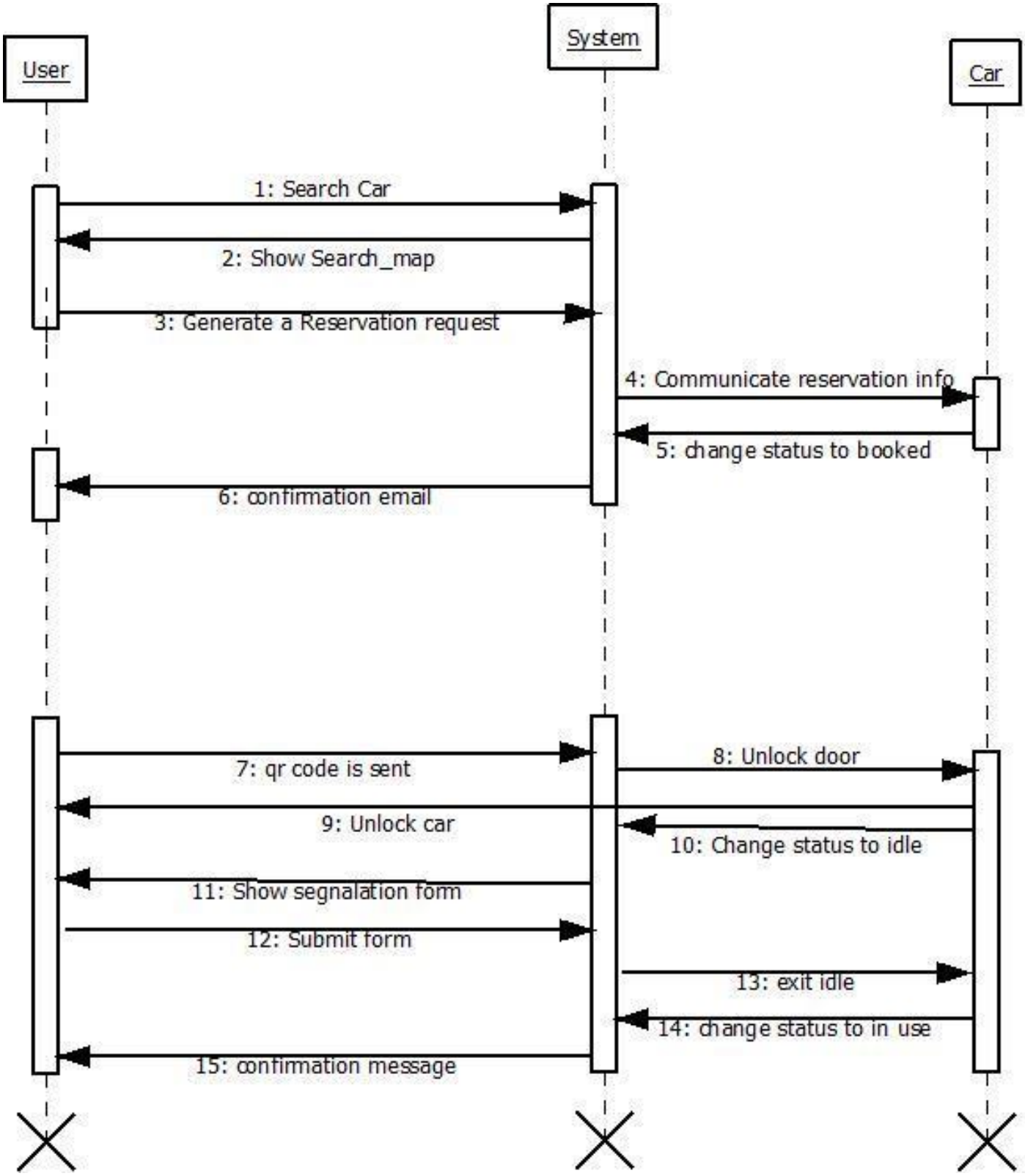
Exceptions: No exceptions for this use case.

c. Class diagram



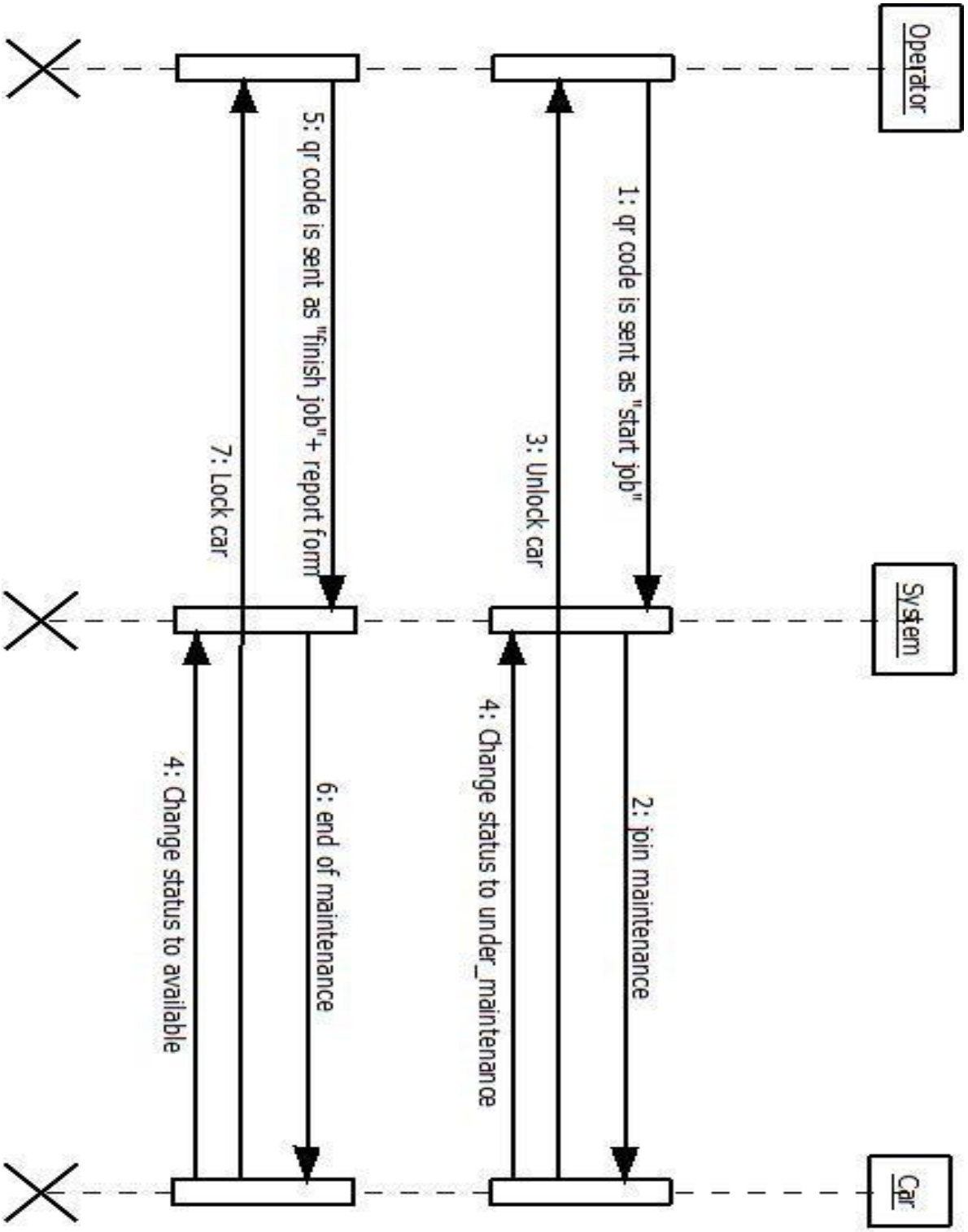
d. Sequence diagrams

Sd1: User searches, books and takes a car

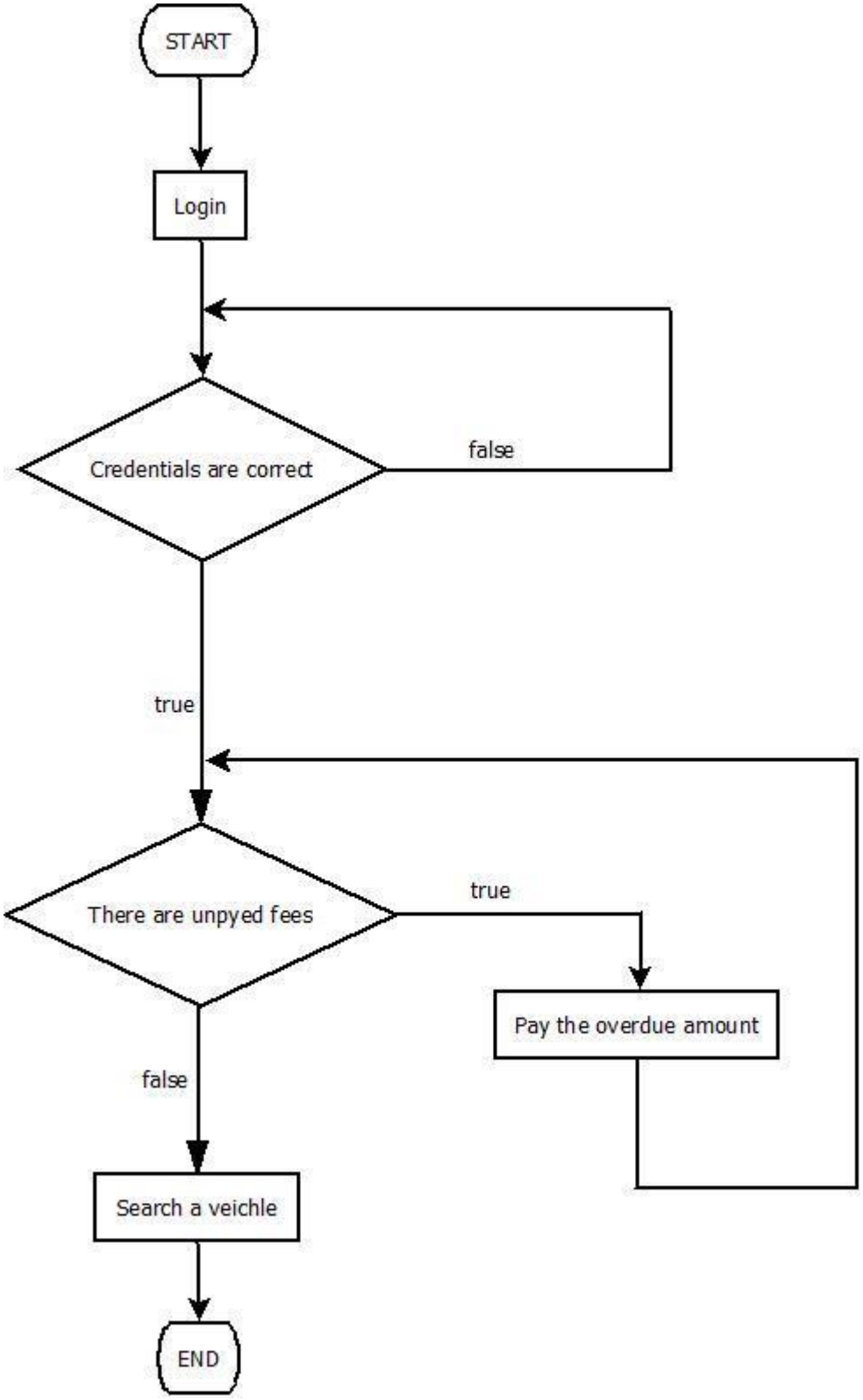


Sd2: Operator completes a job on car

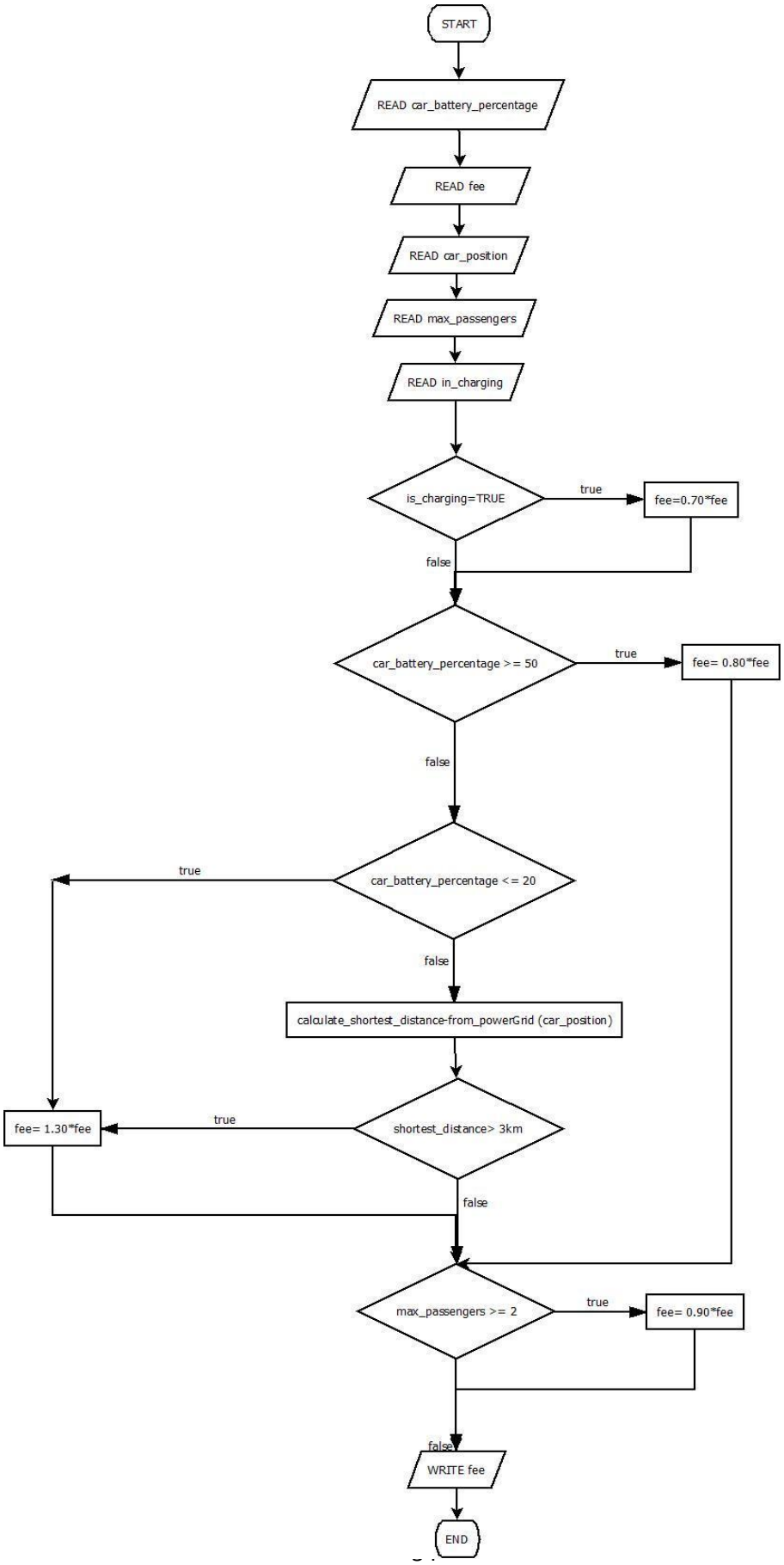
e. Activity diagrams



AD1: User’s login and overdue fees control

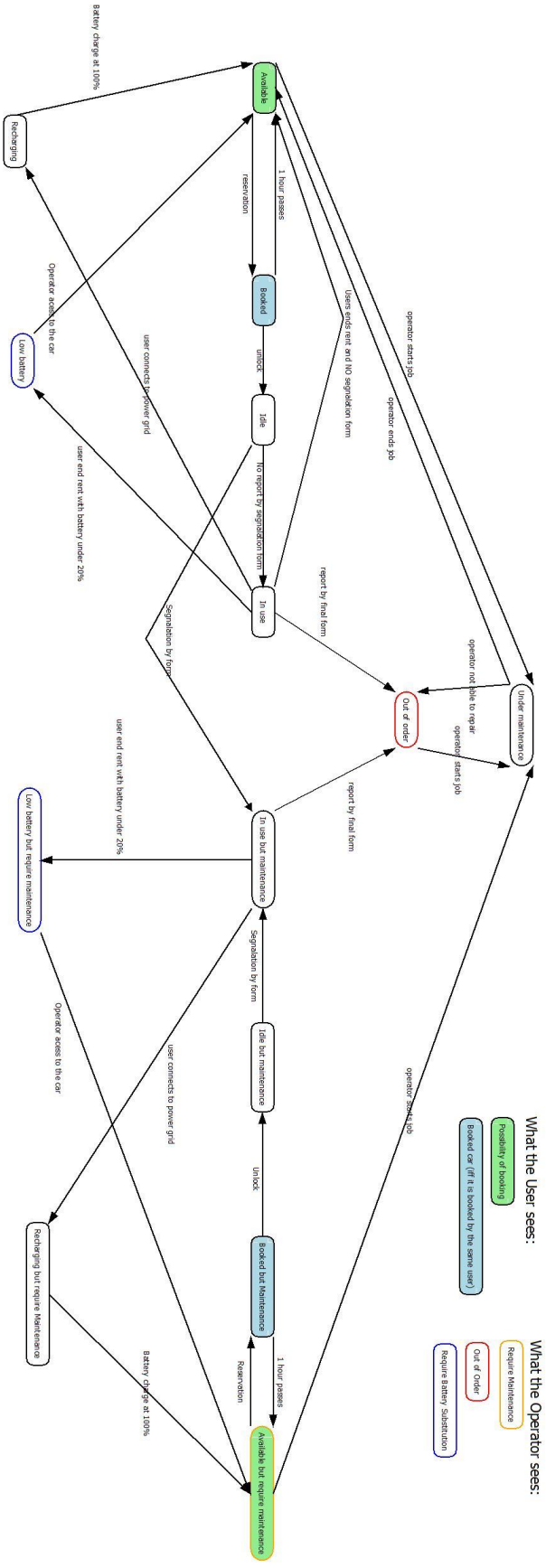


AD2: Fee calculation

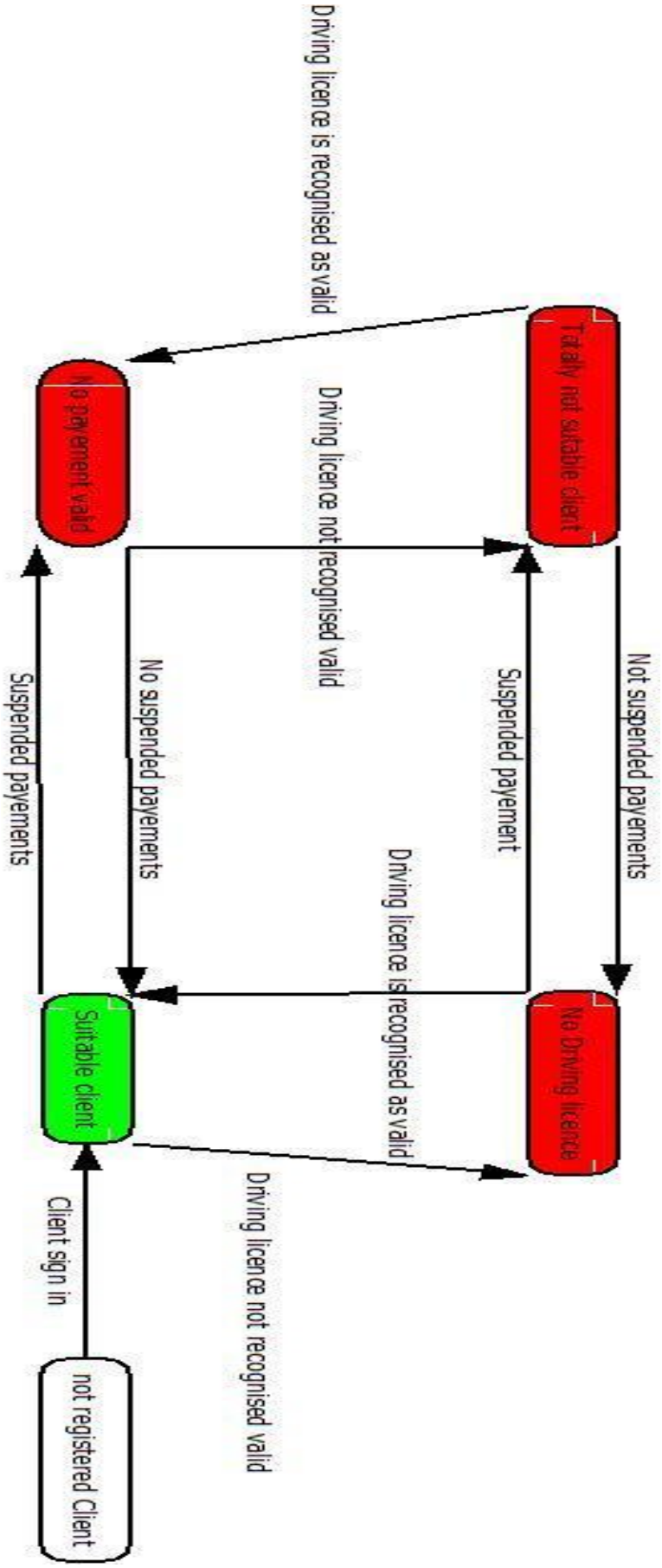


f. State diagrams

SD1: Car status



SD2: User status



6. Formal model: Alloy

a. The model

```
open util/boolean
open util/integer

abstract sig Person{}

sig User extends Person{
  id : one Id,
  status : one UserStatus,
  fiscalcode : one FiscalCode,
  drivinglicense : one DrivingLicense,
   paymentmethod : PaymentMethod,
   password : one Password
} {
   #paymentmethod > 0
  fiscalcode= drivinglicense.fiscalcode
}

sig Id{}
sig FiscalCode{}
sig Password{}
sig PaymentMethod{}

sig DrivingLicense{
  fiscalcode : one FiscalCode,
  validity : one Bool
}

sig UserStatus {
  validity: one Int // 0 okay, 1 need payment, 2 driving license not valid, 3 both not valid
}{
  validity >= 0
  validity < 4
}

fact fiscalCodesAreUnique {
  all t1, t2: User | (t1 != t2) => t1.id != t2.id
}

fact drivingLicenseNotValidImpliesUserStatusNotValid {
  all u : User | !u.drivinglicense.validity.isTrue
  implies
  u.status.validity >= 2
}
```

```
fact univoque_correspondance_users_satus{
 all u1:User, u2:User | u1!=u2
 implies
 u1.status != u2.status
}

fact nr_status_equal_to_nr_users {
 #User=#UserStatus
}

 sig Car{
 id : one Plaque,
 qrcode : one QRCode,
 status : one CarStatus,
 lowBattery : lone LowBatteryCar,
 highBattery: lone HighBatteryCar,
/* They don't add a significant information to the model
    x : one Int, //Float
    y : one Int, //Float
    engine : one Engine,
*/
 maxpeopleinside: one Int,
 peopleinside : one Int
}{
 peopleinside <= maxpeopleinside
 peopleinside > 0 iff (#status.inUse = 1 or #status.inUseButM = 1)
 peopleinside >= 0
 maxpeopleinside > 1
 #lowBattery=1 => #highBattery=0
 #highBattery=1 => #lowBattery=0
 //statusCoherence

 (#status.lowBattery=1 or #status.lowBatteryButM=1) => #lowBattery=1
}


sig LowBatteryCar, HighBatteryCar, Charging{}

sig CarStatus{
      available: lone Available,
      booked: lone Booked,
      idle: lone Idle,
      inUse: lone InUse,
      recharging: lone Recharging,
      lowBattery: lone LowBattery,
      availableButM: lone AvailableButM,
      bookedButM: lone BookedButM,
```

```
    idleButM: lone IdleButM,
    inUseButM: lone InUseButM,
    rechargingButM: lone RechargingButM,
    lowBatteryButM: lone LowBatteryButM,
    outOfOrder: lone OutOfOrder,
    underMaintenance: lone UnderMaintenance
}{
   //In order to guarantee that a car has one and only one status we need to sum all the cardinalities and impose that sum equal to one
   plus[plus[plus[plus[plus[plus[plus[plus[plus[plus[plus[ plus[plus[(#available) , (#booked)] , (#idle)] , (#inUse)] , (#recharging)] , (#lowBattery)] , (#availableButM)] , (#bookedButM)] , (#idleButM)] , (#inUseButM)] , (#rechargingButM)] , (#lowBatteryButM)] , (#underMaintenance)] , (#outOfOrder)] = 1

/* ok
 #available=1=>(#booked=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #booked=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #idle=1=>(#available=0 &&#booked=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #inUse=1=>(#available=0 &&#idle=0 &&#booked=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #recharging=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#booked=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #lowBattery=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#booked=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #availableButM=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#booked=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #bookedButM=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#booked=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #idleButM=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#booked=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #inUseButM=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#booked=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #rechargingButM=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#booked=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#outOfOrder=0)
```

```
 #lowBatteryButM=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#booked=0 &&#underMaintenance=0 &&#outOfOrder=0)
 #outOfOrder=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#underMaintenance=0 &&#booked=0)
 #underMaintenance=1=>(#available=0 &&#idle=0 &&#inUse=0 &&#recharging=0 &&#lowBattery=0 &&#availableButM=0 &&#bookedButM=0 &&#idleButM=0 &&#inUseButM=0 &&#rechargingButM=0 &&#lowBatteryButM=0 &&#booked=0 &&#outOfOrder=0)
*/

}

sig Available, Booked, Idle, InUse, Recharging, LowBattery{}
sig AvailableButM, BookedButM, IdleButM, InUseButM, RechargingButM, LowBatteryButM{}
sig OutOfOrder, UnderMaintenance{}

sig QRCode{}

//sig Engine{}

sig Plaque{}

fact plaquesAreUnique {
 all t1, t2: Car | (t1 != t2) => t1.id != t2.id
}

fact QRAreUnique {
 all t1, t2 : Car | (t1 != t2) => t1.qrcode != t2.qrcode
}

/*
fact TwoCarsCan'tBeInSameParking { //Obv in real system there is the precision of GPS to be considered
 all c1, c2 : Car | (c1 != c2) => ((c1.x != c2.x) and (c1.y != c2.y))
}
*/

fact cardinalitiesForCars {
 #Car =#Plaque
 #Car = #QRCode
 #Car = #CarStatus
}

fact coherenceBetweenInstancesOfSameDrivingLicenses{
 all d1, d2 : DrivingLicense | d1.fiscalcode = d2.fiscalcode => d1 = d2
}
```

```
sig Reservation {
  user : one User,
  car : one Car,
  timeLeft : one Int
}{
//  timeLeft <= 60
  timeLeft > 0
}

fact everyUserCanBookAtMostOneCar {
  all r1, r2 : Reservation | r1 != r2 => (r1.user != r2.user and r1.car != r2.car)
}

fact everyUserCanHaveAtMostOneAccount {
  all u1, u2 : User | u1 != u2 => u1.fiscalcode != u2.fiscalcode
}

sig Rent {
    user: one User,
    car: one Car,
    highBatteryDiscount: lone HighBatteryDiscount,
    passengersDiscount: lone PassengersDiscount,
    chargingDiscount: lone ChargingDiscount,
    lowBatteryCharge: lone LowBatteryCharge,
    initialForm : one UserForm,
    finalForm : lone UserForm,
/* They would make the scheme without adding necessary information modeling the world
    initialTime : one Int,
    finalTme : one Int,
    feePerMinute : one FeePerMinute,
*/
}{
    //relation between discounts/charges:
    #highBatteryDiscount = 1 iff #car.highBattery = 1
    #lowBatteryCharge = 1 iff #car.lowBattery = 1
    #passengersDiscount = 1 iff car.peopleinside > 2
}

sig HighBatteryDiscount, PassengersDiscount, ChargingDiscount, LowBatteryCharge{}

fact carMustBeInUseWhenUsed{
     all c: Car, u: Rent | u.car = c => (#c.status.inUse = 1 or  #c.status.inUseButM = 1)
}

fact onlyOneCarInUsePerTime{
    all u1, u2: Rent | u1.user = u2.user
    implies
    u1 = u2
```

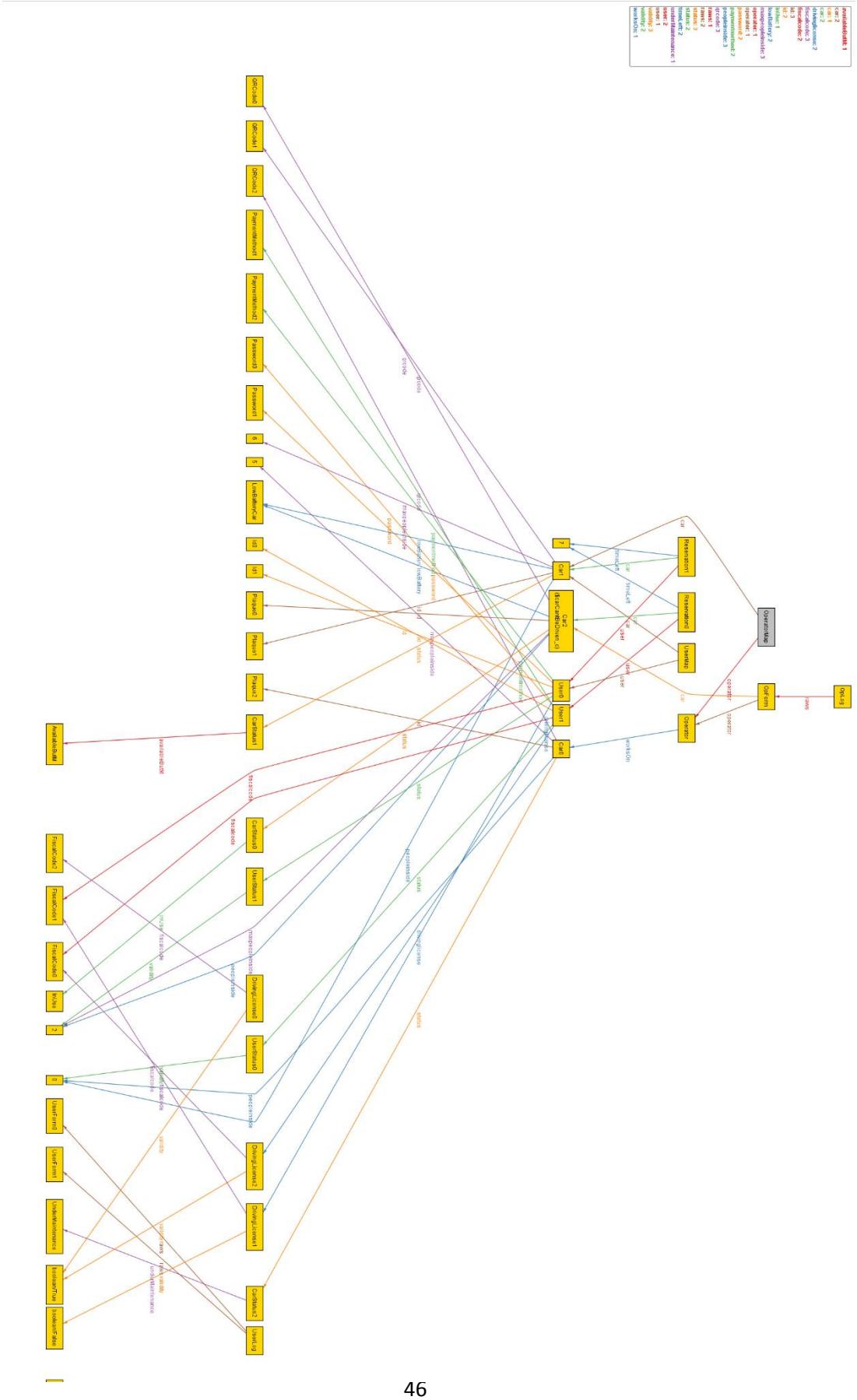
```
}

fact userCantUseIfBook{
    all u : User, rent : Rent , reservation : Reservation | (rent.user = u and reservation.user = u)
    implies
    (#reservation = 0 or #rent = 0)
}

fact onlyOneUsePerCar{ //this not impede the possibility of an user to enter a car as passenger, but avoid 2 users are able to use the same car!
    all u1, u2 :  Rent | u1.car= u2.car
    implies
    u1=u2
}

abstract sig Map{
    car: some Car
}{}

sig UserMap extends Map{
    user : some User
}{}

fact allCarInUserMapMustBeAvailable{
    all c : Car, um : UserMap | (#c.status.available = 1 or #c.status.availableButM = 1)
    iff
    c in um.car
}

sig OperatorMap extends Map{
    operator : some Operator
}{}

fact opMapMustShowAllCarsHeNeedsToSee{
    all c: Car, om:OperatorMap | (#c.status.availableButM = 1 or #c.status.lowBattery = 1 or #c.status.lowBatteryButM = 1 or #c.status.outOfOrder = 1)
    iff
    c in om.car
}

sig Operator extends Person{
     worksOn : lone Car,
}{
     #worksOn.status.underMaintenance = 1
}

sig OpForm extends Form{
    car : one Car,
```

```
    operator : one Operator
}

abstract sig Log{}

sig UserLog extends Log{
    raws : some UserForm
}{
    #raws = #UserForm
}

sig OpLog extends Log{
    raws : some OpForm
}{
    #raws = #OpForm
}

fact everyFormIsContainedEitherInUserLogOrOperatorLog {
    #UserLog = 1
    #OpLog = 1
}

abstract sig Form{}

sig UserForm extends Form{}

fact OnlyOneOperatorWorksOnACar{
     all o1, o2 : Operator | o1.worksOn = o2.worksOn
     implies
     o1 = o2
}

fact constraintOverPossibleStates {
     #Available=1
     #Booked=1
     #Idle=1
     #InUse=1
     #Recharging=1
     #LowBattery=1
     #OutOfOrder=1
     #UnderMaintenance=1
     #AvailableButM=1
     #BookedButM=1
     #IdleButM=1
     #InUseButM=1
     #RechargingButM=1
     #LowBatteryButM=1
     #LowBatteryCar=1
```

```
    #HighBatteryCar=1
    #UserMap = 1
   #OperatorMap = 1
}

assert drivingLicensesAreUniqueInTheSystem {
 no d1, d2 : DrivingLicense | d1.fiscalcode = d2.fiscalcode and d1 != d2
}

pred showOnlyUsers{
  #User > 1
  #Car = 0
}

pred showOnlyCars{
  #Car > 1
  #User = 0
}

pred showExampleOfReservation{
 #Reservation > 1
  #User = 2
  #Car = 2
}

pred showAll{
 #Car > 1
 #User > 1
 #Rent >1
}

pred showMap{
 #Car > 1
 #User > 1
 #Rent >1
 #UserMap=3
}
pred showOperator{
 #Car >1
 #User>1
 #Operator>1
}

assert carCantBeDriven {
 no c : Car | #c.status.inUse = 1
}

assert sameCarIsn'tBookedByMoreThanOneUser{
```

```
    no r1 : Reservation, r2 : Reservation | ((r1.car != r2.car) and (r1.user !=r2.user) and (r1 != r2))
}

assert anUserCan'tBookACarWithEmptyBattery{
    no c : Car | #c.status.booked = 1 and #c.status.lowBattery = 1
}

//check carCantBeDriven
//check drivingLicensesAreUniqueInTheSystem
//check sameCarIsn'tBookedByMoreThanOneUser
check anUserCan'tBookACarWithEmptyBattery




/*
pred show{}

run show
*/
```

b. Our world



c. Some results

```
Executing "Check anUserCan'tBookACarWithEmptyBattery"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  10625 vars. 720 primary vars. 25309 clauses. 188ms.
  No counterexample found. Assertion may be valid. 142ms.

Executing "Check carCantBeDriven"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  10605 vars. 720 primary vars. 25246 clauses. 64ms.
  Counterexample found. Assertion is invalid. 107ms.

Executing "Check drivingLicensesAreUniqueInTheSystem"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  10637 vars. 723 primary vars. 25286 clauses. 51ms.
  No counterexample found. Assertion may be valid. 16ms.
```

7. Future Project

A possible future development for Power Enjoy could be the extension of the fleet to other types of vehicles, for example scooter and vans. Otherwise, implementing a system for paying high fidelity customers, for example some points to collect to receive gift and discounts, can be a great way to suggest people to use the system.

8. Hours of work

Enrico Gnecco: 25 hrs.
Pietro Crovari: 25 hrs.

9. Change log

V1.2 - 11/01/2017 Fixed a typo