PowerEnJoy

A.A. 2014-2015

Requirements Analysis and Specifications Document

Author:

Fabio Menzaghi

November 12th 2016

Summary

1. Introduction....4
1.1 – Description of the given problem....4
1.2 – Goals....6
1.3 – Domain properties....9
1.4 – Glossary....9
1.5 – Text assumptions....10
1.6 – Constraints....12
1.6.1 – Regulatory policies....12
1.6.2 – Hardware limitations....12
1.6.3 – Interface to other applications....13
1.6.4 – Parallel operation....13
1.7 – Proposed system....13
1.8 – Identifying stakeholders....13
1.9 – Reference documents....13
2. Actors Identifying....14
3. Requirements....15
3.1 – Functional requirements....15
3.1.1 – Requirements definition....15
3.1.2 – Traceability matrix....17
3.2 – Non-functional requirements....17
3.2.1 – Mobile interface....17
3.2.2 – Desktop interface....18
4. Scenario identifying....19
4.1 – Scenario 1....19
4.2 – Scenario 2....19
4.3 – Scenario 3....20
4.4 – Scenario 4....20
4.5 – Scenario 5....20
4.6 – Scenario 6....20
4.7 – Scenario 7....20
5. UML Models....21
5.1 – Use case....21
5.1.1 – Sign-Up....22
5.1.2 – Sign-In....22
5.1.3 – Car search....22
5.1.4 – Reserve a car....23
5.1.5 – Apply Fee....24
5.1.6 – Unlock a car....24
5.1.7 – Power On....25
5.1.8 – View charging cost....25
5.1.9 – Power Off....26
5.1.10 – Lock....26
5.1.11 – Create transaction....27
5.2 – Class diagram....28
5.3 – Sequence diagram....29
5.3.1 – Sign-Up and Sign-In....29
5.3.2 – Car renting....30
5.4 – State chart diagram....31

5.4.1 – Car states....31
5.4.2 – Transaction states....32
6. Alloy analyzer....33
6.1 – Model....33
6.1.1 – Signatures....33
6.1.2 – Facts....34
6.1.3 – Predicates....36
6.1.4 – Assertions....36
6.2 – Results....37
6.3 – Worlds generated....38
7. Future developments....39
8. Used tools....40
9. Hour of works....41
10. Changelog....42

1. Introduction

RASD (Requirements Analysis and Specification Document) is an analysis of the requirements that the software must meet in order to satisfy goals defined by the stakeholders based on the domain properties of the world and the assumptions made. It is the first step of the process of modeling and formalization of an informative system.

This document is intended to follow the Agile software development philosophy and hence will be updated many times during the modeling phase of the software development. Logs of this changes are available in chapter 10.

The RASD is divided in ten chapters as follows:

Chapter 1 focuses on problem definition, identifying goals and stakeholders. Stating domain properties, assumptions and constraints. It also defines some terms in the glossary that will be used in the whole document.

Chapter 2 identifies actors of the problem.

Chapter 3 enumerates functional requirements according to goals defined in chapter 1 and shows an example of non-functional requirements that the software must meet.

Chapter 4 gives a few example of possible scenarios related to the problem in order to highlight the main use case that are described in the following chapter.

Chapter 5 illustrates the models used to formalize the problem and satisfy the requirements.

Chapter 6 shows the results on the models obtained exploiting alloy analyzer.

Chapter 7 provide some future development of the project.

Chapter 8 list tools used to create this document.

Chapter 9 provides a table of hour needed to redact this document.

Chapter 10 logs all the changes to this document.

1.1 – Description of the given problem

The problem is to develop a digital management system for a car-sharing service that exclusively employs electric cars. The system will allow registered user to list available car near to their GPS location (or a specified address) and reserve one of them. Once reserved the car cannot be reserved by any other user. If not picked up in 1 hour the system will mark the car as available again and charge the user for 1 EUR of fee.

If the user is near the car he/she reserved the system will allow him/her to unlock the car's door through the app. Once engine starts the system will start charging the user for a given amount of money per minute. User are notified about their actual charging cost through a screen on the car.

Once the car is parked in a (predefined) parking area and the user is out of the car, he/she can lock it and the system will stop charging and locks the doors of the car.

Users can only reserve (and pick up) cars that are in their geographical region.

Users will get a discount if they:

1. Bring at least other 2 people. They will get a 10% discount on the last ride.
2. Leave the car with at least 50% of the battery. They will get a 20% discount on the last ride.

1.2 – Goals

Given the above description of the problem taken from the assignment document, what follows are the goals identified.

- **[G1]** Allow guests to sign-up to the PowerEnJoy car sharing service.
- **[G2]** Allow user to sign-in to PowerEnJoy car sharing service.
- **[G3]** Allow user to search for available cars within a certain distance to his/her current GPS location.
- **[G4]** Allow user to search for available cars within a certain distance to a specified address.
- **[G5]** Allow user to reserve a car for 1h.
- **[G6]** Users can reserve only cars in the same geographical area.
- **[G7]** User can reserve only one car at the time.
- **[G8]** Allow user to unlock and enter a reserved car when he/she is near to it.
- **[G9]** Allow user to start engine when he/she is in the car.
- **[G10]** Charge the user an amount per minute for each ride since the engine starts until the car is locked.
- **[G11]** Allow user to monitor the cost of the ride through a screen in the car.
- **[G12]** Allow user to power off the car in a safe parking area.
- **[G13]** When a parked car is empty the system locks it at set it as available.
- **[G14]** Apply a 10% discount to a ride if there are more than 2 people in the car during the ride.
- **[G15]** Apply a 20% discount if at the end of the ride there is at least 50% of the battery in the car.
- **[G16]** Process payment transactions through a gateway.
- **[G17]** Apply a fee of 1 EUR for each expired reservation.
- **[G18]** User with any transaction not paid cannot use the cars.
- **[G19]** Operators can see the list of cars that need to be fixed.

1.3 – Domain properties

All the following statements are assumed to be always true in the world.

- [D] The number of people in the car.
- [D] The car can detect battery level
- [D] A car is able to detect if it is in a parking area.
- [D] The car can detect if there is anyone inside.
- Every car can be in only one location at the same time. The same for users.
- Anything can enter/exit the car if, and only if, it is unlocked.
- Users can start/stop engine only if they are in the car.
- If the company adds a car this will be shown among the available.
- Exists at least one car.

otherwise the service wouldn't even have sense.

- Every Position belongs to one and only one Geographical area.
- Cars and users are unique. (Users can't create duplicated account.)
- If an user press button “Lock”, “Unlock”, “Reserve”, etc. he/she really intends to do the action mentioned.
- Battery level sensor provide always accurate values unless the car is broken.
- A user in a car can always see what is shown on the screen inside the car.
- When a user enter/exit a car always closes the doors.
- Password are stored safely by the users so that authentication can be trusted. {USEFUL?}
- All GPS devices provide the right position of the relative car/user. {WRONG}
- All the mechanical parts of the cars are fully working.{WRONG}
- Each registered user has a valid driving license to drive the car shared by the company. {WRONG}

1.4 – Glossary

- **User**: a person registered to the system. He/she has a password provided by the system and valid payment credentials.
- **Position**: is a position on Earth coded as two variable: longitude and latitude.
- **Parking area**: a position that is safe for a car to be parked.

- **Geographic area**: is a partition of the position on Earth. This means that the intersection of the set of positions of two different geographic areas is empty.
- **Car**: by car we mean one of the PowerEnJoy fleet's cars. Moreover, a car can be in one, and only one of the following states:
    1. **Available**: the car is off and locked but is available to be reserved.
    2. **Reserved**: the car is off and locked but is NOT available to be reserved.
    3. **Ready**: car is unlocked but engine has not been started yet. User and fellows can get on / get off the car in this state.
    4. **Running**: engine is on and the car can move. User and fellows can get on / get off the car in this state.
    5. **Parked**: engine is off and the car is in a Parking area. User and fellows can get on / get off the car in this state (they are supposed to get off and close the door as soon as possible).
    6. **Broken**: the car is broken and cannot be used, location could be unavailable and other things may not work. See assumptions in order to better understand this state.
- **Transaction**: is a payment that the user needs to do either for an invoice such that:
    - **Expiration Fee**: a 1 EUR fee for a reserved car which is not picked up in 1 hour since reservation.
    - **Ride**: a transaction applied for having driven a car. Can be subject to discounts.
- **Payment gateway**: A third party mechanism that provides basic functionality to get payments of the transactions of the users.
- **Discount**: a percentage of discount on the last ride in a Riding transaction. Can be either:
    - **Pooling discount**: if there are at least 3 seats occupied on the car when the engine starts the system will apply a discount of 10%.
    - **Battery discount**: if the car is left with more than 50% of the battery, the system will apply a discount of 20% on the last ride.
- **Power On**:
- **Power Off**:
- **Empty car**: a car is said to be empty when there is no weight on the seats and the doors are closed.
- **Passengers on board**: for each ride the number of passengers on board is calculated as the maximum number of passengers (detected by mean of the number of seatbelt fasten) during the ride. The car will take care of updating this value.

1.5 – Text assumptions

Since the initial written problem statement provided suffers from the typical drawbacks of natural language descriptions: it is informal, incomplete, uses different terms for the same concepts, and the like. The assumptions that follows are intended to solve the incompleteness and ambiguity of the problem statement. It is clearly document the corresponding rationale behind the choices made.

1. Le aree di parcheggio sono individuate dalle auto tramite un sensore
2. The number of passengers of the car measure is based on weight on seats: if on a seat there is a weight higher than a threshold the system will assume that there is a person on it.

   Since passengers can get into the car and leave it while the car is picked up, we assume that the number of passengers of the car is taken when the engine starts. System will count the number of seats with a weight above the threshold and save it as the number of passengers of this ride.
3. Plate is assumed to be the identifier of the cars. If the system has to be implemented in countries where this parameter is not unique an other equivalent identifier can be used.
4. Parking area sono strisce gialle per terra che rappresentano 1 posto auto.
5. A car in one or more of the following states will be considered broken:
   - The car is stolen. (GPS signal or remote connection not available)
   - The car had an accident.
   - The battery level is at 0% (or an other specified threshold).
   - The car is actually broken in such a way that is not good anymore to be part of the service until repaired. For instance a broken engine or wheel.

   The rationale behind this is that in any of these cases the expected behaviour of the system should be the same: stop charging the user if any charging is active and remove the car from the list of available cars.

   The company will then behave accordingly, if the car is stolen call the police, if broken repair it, etc.
6. The distance based on location of the user and the reserved car needed to unlock the car is computed through GPS values in order to avoid the implementation of other proximity detection mechanisms.
7. When people leave a car close the doors behind them. So that when a car is empty, doors are closed.
8. The user who reserve a car is the one who will unlock it. There is no kind of “i take it for you”
9. Locked means doors are also closed. We can suppose this since...
10. Since in the text is not specified if the system must check the user location when

he/she tries to unlock it, it seems reasonable that this check have not to be done since it is in the interest of the user to prevent anyone entering the car until he/she is near to it.

11. It is assumed that on the screen where is shown the charging cost of the ride there is always shown battery level. Even if not explicitly requested this should be implemented due to the discount of 20% if car is left with more than half of the battery: the user needs to know how much is left.
12. In order to fulfill the goal of apply a discount of 10% to the ride with two or more other passenger, a method to determine the number of passenger must be fixed. There are many solutions since passengers are loaded and dropped by the way. The number can be evaluate at the beginning or at the end. But at the beginning it's not fair for the ride home to work since passenger are loaded after the car is started. Viceversa work to home ride. So neither would be fair if we think about a car pooling aim which is the virtuous behavior behind the idea of this discount.

    In this document is assumed that is the maximum number of passenger in the whole ride. The following assumptions regards the method to determine the number of passengers.
13. Diverso tra discount e isVuota() perché se no uno bara oppure c'è il rischio che ci si dorma in macchina.

1.6 – Constraints

Here there are some additional constraints that the system must satisfied implied by the real world.

1.6.1 – Regulatory policies

PowerEnJoy must deal GPS position, user personal information and payment credentials according to the law. It also must meet any regulatory policy related to car driving.

1.6.2 – Hardware limitations

The following systems will need to be equipped with the specified hardware:

1. App on user's mobile device:
    - Must be equipped with a GPS module.
    - Must be connected to the internet.
    - Must have enough space for app package
2. Car device:
    - Must be connected to the system through a tcp/ip connection.
    - Must be equipped with a GPS module.
    - Must be able to lock/unlock doors.
    - Must be equipped with a screen (to show charging cost, battery and messages)

- Must be equipped with a seatbelt sensor.
- Must be equipped with a seat weight sensor.

3. Data center:
    -

1.6.3 – Interface to other applications

The PowerEnJoy system must be able to interface through some API with a payment gateway.

1.6.4 – Parallel operation

The system must be able to operate in parallel to satisfy user requests, car status tracking and process payment transaction.

1.7 – Proposed system

Database

System

Car

User app

Web app

1.8 – Identifying stakeholders

There is only one stakeholder which is PowerEnJoy company who wants to build a electrical car sharing service based on incentives to improve virtuous behaviors. In this case we consider just two types of discounts but more can be added later.

Vogliono una roba leggera perché sono fricchettoni con l'ambiente e fanno le auto elettriche.

1.9 – Reference documents

The following documents have been used as reference during the development of the RASD:

- Assignments AA 2016-2017.pdf
- RASD example SWIMv2.pdf
- RASD meteocal example 1.pdf
- RASD meteocal example 2.pdf
- “Green Move: A Platformfor Highly Configurable, Heterogeneous Electric Vehicle Sharing” (by Andrea G. Bianchessi, Gianpaolo Cugola, Simone Formentin, Angelo C. Morzenti, Carlo Ongini, Emanuele Panigati, Matteo Rossi, Sergio M. Savaresi, Fabio A. Schreiber, Letizia Tanca, and Edoardo G. Vannutelli Depoli)
- “A Flexible Architecture for Managing Vehicle Sharing Systems” (by A. G. Bianchessi, C. Ongini, S. Rotondi, M. Tanelli, M. Rossi, G. Cugola, S. M. Savaresi)

2. Actors Identifying

The actors of our informative systems are:

- **Guest**: people not yet registered to the platform. Their only ability is to sign-up.
- **User**: registered people with valid payment credentials and driving license. He/she is supposed to be equipped with a mobile device with the features described above.
    - Mobile users
    - Desktop users
- **Operator**: is an employed of PowerEnJoy and his/her job consist in repair and recharge the car of the car sharing service.
- **Payment Gateway**: is a third party mechanism providing the ability to receive money giving valid payment credentials.

3. Requirements

3.1 – Functional requirements

3.1.1 – Requirements definition

What follows it's a complete list of the requirements the system must accomplish in order to satisfy the related goal:

[G1] Allow guests to sign-up to the PowerEnJoy car sharing service.

- **[R1] The system must provide a sign-up functionality**
- **[R2] The system must be able to generate passwords.**

[G2] Allow user to sign-in to PowerEnJoy car sharing service.

- **[R3] The system must provide a sign-in functionality**

[G3] Allow user to search for available cars within a certain distance to his/her current GPS location.

- **[R4] The system must be able to list all the car within a certain distance to a specified position.**

[G4] Allow user to search for available cars within a certain distance to a specified address.

- Same as [R4]
- **[R5] The system must be able to map an address to a position.**

[G5] Allow user to reserve a car for 1h.

- **[R6] The system must allow the user to reserve an available car.**
- **[R7] When a user reserves a car the system must mark the car as *Reserved*.**
- **[R8] When a reservation for a car expires mark it as *Available* again.**
- **[R9] The system must prevent any other user to reserve a reserved car.**
- **[R10] The system must keep track of the time when the user reserve a car.**

[G6] Users can reserve only cars in the same geographical area.

- **[R11] The system must prevent the user to reserve a car in a different geographical area.**

[G7] User can reserve only one car at the time.

- **[R12] The system must prevent the user to reserve more than one car.**

[G8] Allow user to unlock and enter a reserved car when he/she is near to it.

- **[R13] The system must unlock the door of the car reserved by the user when the user clicks on “Unlock” button.**

- **[R14] When the car is unlocked the system must then mark the car as *Ready*.**

[G9] Allow user to start engine when he/she is in the car.

- **[R15] The system must power on the car if it in status *Ready* and the user presses the button *Power On*.**
- **[R16] When the engine powers on the system must mark the car as *Running*.**

[G10] Charge the user an amount per minute for each ride since the engine starts until the car is locked.

- **[R17] The system must keep track of the time between engine starts and the car is locked.**
- **[R18] The system must create a transaction for the ride.**

[G11] Allow user to monitor the cost of the ride through a screen in the car.

- **[R19] The system must show the cost of the ride on the screen in the car.**
- **[R20] The system must be able to compute the cost at runtime.**

[G12] Allow user to power off the car in a safe parking area.

- **[R22] The system must show if the car is in a safe parking area on the screen in the car.**
- **[R23] The system must power off engine when the user presses the button power-off if the car is in a safe parking area.**
- **[R24] The system must mark the status of the car as *Parked*.**

[G13] When a parked car is empty the system locks it at set it as available.

- **[R25] When a *Parked* car is empty, the system locks it and mark it as *Available*.**

[G14] Apply a 10% discount to a ride if there are more than 2 people in the car during the ride.

- **[R26] The system must add a Pooling Discount if during the the ride there are at least 2 people in the car.**

[G15] Apply a 20% discount if at the end of the ride there is at least 50% of the battery in the car.

- **[R27] The system must add a *Battery Discount* to a car if there is at least 50% at the end of the ride.**

[G16] Process payment transactions through a gateway.

- **[R28] The system must compute the cost of a ride an amount per minute**
- **[R29] The system must apply discounts to the amount of a ride.**
- **[R30] The system must process a transaction to a payment gateway**
- **[R31] The system must mark the transaction as *Paid* or *Not Paid* according to**

the payment gateway response.

[G17] Apply a fee of 1 EUR for each expired reservation.

- **[R32] When a reservation expires create a transaction of 1 EUR.**

[G18] User with any transaction not paid cannot use the cars.

- **[R33] The system must prevent the users with at least 1 transaction not paid to reserve cars.**

[G19] Operators can see the list of cars that need to be fixed.

- **[R34] The system must be able to list the broken cars.**

3.1.2 – Traceability matrix

This table illustrates how goals, requirements and use cases are related to each other. Use cases are described in chapter 5.

	Goal	Requirement	Use case	Comments

3.2 – Non-functional requirements

3.2.1 – Mobile interface

3.2.2 – Desktop interface

4. Scenario identifying

4.1 – Scenario 1

Anni is going to spend the next 6 month in Milan in order to practice her Italian to find a job in her city, Munich, as an Italian teacher. She already found a job and one of her friends, Francesca, can host her at her place. Francesca's house and job are not well connected by public service: it takes more than an hour to get there and she has to change three different lines. So she needs to find an other way to get there every morning. She knows about car sharing services but she is also against the use of the car since she takes care of the environment. Googling “Eco car sharing service”, among the results she founds the PowerEnJoy voice promoting its “ecologic electric car”. She opens the page and in the description of the service she notices that there also discounts for virtuous behavior. She feels totally on the same page with the mission of the company and clicks on sign-up to create an account. The page asks for personal information and a payment method. Anni insert is name and surname and her credit card number. Then she clicks “Sign-Up” and the page is reloaded showing a password to be used to sign-in.

Anni opens PowerEnJoy's homepage and insert her username and the provided password, then clicks on “Sign-In”. The website is now showing a page with a field for inputing an address to find car near it and other functionalities. She inserts the address of Francesca's house to have a clue of how many car there are available in that area. She chooses a range of 150m that she thinks she can walk to get to the car. Once clicked on “Search cars” the system shows a list of 6 available car near to Francesca's house.

Anni is satisfied since she can now go to work in a reasonable time and in an ecological way.

4.2 – Scenario 2

Bob is just landed in the airport of Malpensa and he has just collected his bags. He is walking out the airport when hears a couple nearby worrying about the strike of the airport shuttles that carry people from the city to the airport and viceversa. As expected there is no shuttle outside the airport and he needs to find an other solution to get to the city. He remembers about the car sharing service he signed-up few months ago. He used this service for a couple of times always in the city. He takes is phone hoping to find some of those car near to the airport. He opens the PowerEnJoy's app, insert his login details and clicks on *“Sign-In”*. The website is now showing the personal area page with the form to find cars. Bob inserts 300m range and clicks on *“Search car by current position”*. The app shows a list of a dozen cars available. Since they are all at the same distance, probably because are in the same parking Bob thinks, he chooses the first of the list and clicks on *“Reserve now”*. Then the app shows the car on the map and an *“Unlock”* button to be pressed once the user is near the car. Bob is warned to reach the car in 1 hour, otherwise he will be applied a fee of 1 EUR and the car will be freed again. Bob follows the map and find the car, it took him only 15 min so he has still his car reserved. Clicking on “Unlock” Bob can hear the doors getting unlocked and tries to open it. He now enter the car and close the door. As soon as he clicks on “Power On” button the car turns up and the screen shows an amount of 0.00 € that

is his current charging amount. Near to it he can also see that the battery level is 76%, more than enough to get home. He drives until his destination. When he is near home the screen is showing an amount of 13.74 € . He already knows the parking area of PowerEnJoy's cars near to his house so he goes straight to it. He park the car in the right place and presses the *"Power off"* button. The carn turns off but the screen still shows the charging amount and a new message: *"Thanks for your ride! Please, leave the car and close the doors to let the car lock and stop charging"*.

Bob leaves the car and as soon as he closes the door he hears them lock. On the personal area of PowerEnJoy app he can now see a new transaction with an amount of 13.76 €.

4.3 – Scenario 3

Charlie works in a software company in the city center but he cannot afford to leave in this area so every day he takes the PowerEnJoy car to get there since is less expensive than buying a car and there are no public services that could carry him there. Charlie is not the only one in his office who use PowerEnJoy and since sometimes is hard to find an available car because people leave all at the same hour, Charlie decided to reserve it 45 min before work time end. But today the job assigned an extra task to Charlie that make him leave the office 20 minutes after the work time end. He checks on his app and it shows no more the car reserved and instead there is one more transaction described as "Untaken car fee" of 1 €.

4.5 – Scenario 5

Elise is an operator for PowerEnJoy. Her work consist in charging the cars used by the car sharing service when they get out of charge or to carry the broken cars to the mechanic. She is having her morning breakfast and in the meanwhile she opens her PowerEnJoy app for operators. The screen on the mobile shows a list of car along with a message. The first three cars listed are out of battery but parked in a PowerEnJoy parking area. The fourth is a car with a broken engine. Elise clicks on the first car of the list and the app shows a map of the location of the car without battery. She reaches the car and with a portable battery charger provided by PowerEnJoy she charge the battery up to 100%. After the car is charged she clicks on the "Repaired" button on her app and the system removes the car from the list.The car is now available to be reserved and she can move on to the next one.

4.6 – Scenario 6

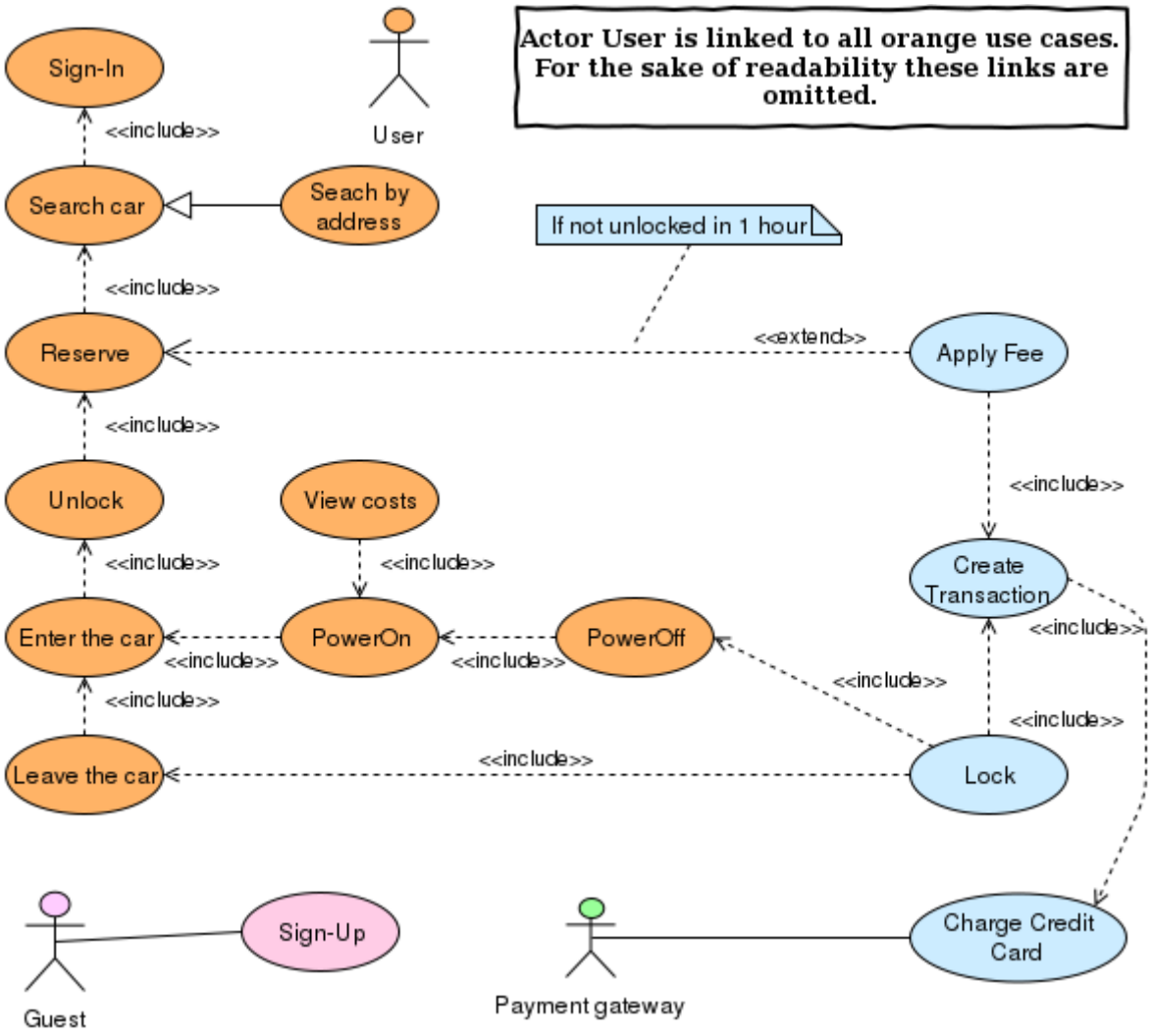
Gary needs to go to the shopping mall in order to buy a new pair of football shoes. Since the shopping mall is 8 km away from his home he decide to use the PowerEnJoy's car sharing service to reach it. It opens the app on his phone and list all the available cars nearby him in a ray of 100 m. The system shows 5 possible cars. He chooses the nearest and when he clicks on "Reserve car" the system prompt an error message displaying "Your last transaction has not been paid. Please click here to retry billing". He clicks on ok and the system contacts the payment gateway with his credential. The payment gateway sends a positive feedback and the transaction is marked by the system as paid. Gary can now rent the car he wants.

5. UML Models

This chapter illustrates the models used to formalize the problem and satisfy requirements. Use cases are related to goals and requirements through traceability matrix in section 3.1.2. The other diagrams are presented with a description that will make clear the link to the underlying requirement.

5.1 – Use case

The goal of the diagram in the figure is to show functions available to the actors and some subsystem module that deals with it. Please note that use cases “Enter the car” and “Leave the car” are not functionality provided by the system but are there in order to get a more complete diagram.



5.1.1 – Sign-Up

This use case is also documented by sequence diagram 5.3.1.

Actors	Guest
Entry Conditions	The guest isn't already signed up and he/she is on PowerEnJoy home page.
Flow of events	1. The user clicks on the button "sign up"; 2. The system shows a page which contains an input form which asks for some personal information and valid payment credential; 3. The guest fulfills the input form and clicks "sign up"; 4. The submission is sent to the system which check inputs and generate a password. 5. The system shows a page containing a confirmation message and the generated password.
Exit conditions	User information are persistently stored in the system and the user can now sign-in.
Exceptions	The user is already registered or some input data are not valid. The system will then show an appropriate error message and the user to re-fill the form.

5.1.2 – Sign-In

This use case is also documented by sequence diagram 5.3.1.

Actors	User
Entry Conditions	The user is already signed up to PowerEnJoy and is on the home page.
Flow of events	1. The user fills the input field "username" and "password" with his/her username and the password provided at registration. Then he/she presses the button "Sign-In". 2. The system checks the inputs provided and initiate a new session. 3. The user is shown the Personal Area page.
Exit conditions	The user can now act as an authenticated user on PowerEnJoy platform.
Exceptions	Username or password are not valid. The system shows the homepage again with an error message about the failure. The user is invited to retry.

5.1.3 – Car search

This use case is also documented by sequence diagram 5.3.2.

The use case described below is analogous to the use case "Car search by address", it only differs in the following points:

- In the "search by address" case there is an extra input field asking for an address.
- No GPS information is sent to the system.
-

Actors	User
Entry Conditions	User is already signed-in and he/she is on the Personal Area page.
Flow of events	1. The user fills the input field asking for the max distance the user want to walk to find a car. Then, he/she submit through the button “Find cars”. 2. Users inputs and his/her GPS location are sent to the system. 3. The system make a look up of all available cars in the geographical area of the user are within the provided distance. 4. The user is shown a list of cars with the distance from his/her current position.
Exit conditions	The user can proceed either by choosing a car to reserve it (see use case “Reserve” below) or abort session.
Exceptions	There is no car that satisfies the requirements: the system shows an error message pushing the user to enlarge distance.

5.1.4 – Reserve a car

This use case is also documented by sequence diagram 5.3.2.

Actors	User
Entry Conditions	User is logged and the system is showing a list of car that the user can reserve.
Flow of events	1. The user choose a car and clicks on button “Reserve now” 2. The system marks the car as Reserved and starts a timer of 1 hour. (see use case “Apply Fee”) 3. The user is shown a page with the position of the car. 4. In the page there is a button “Unlock” to be pressed once the user is near the car. 5. See use cases “Unlock” and “Apply Fee” below
Exit conditions	• Car is in Reserved state. • Car cannot be reserved anymore. • The user can unlock the car
Exceptions	The car has been reserved in the time between the list of cars has been created and the time the user clicks on the “Reserve now” button. The system apologizes and send an updated list of cars that meets the previous requirements.

5.1.5 – Apply Fee

Actors	Payment Gateway
Entry Conditions	The user reserved a car but still not unlocked it. Reservation was made more than 1 hour ago.
Flow of events	1. The timer set by the systems fires. 2. The system deletes user reservation and sets the car as Available again. 3. The system creates a transaction for a fee invoice. 4. The system sends payment details to the gateway. 5. See use case below “Create Transaction”
Exit conditions	• The car is Available, any user who match the constraints can reserve it. • System created a transaction of 1 EUR charged to the user through the payment gateway.
Exceptions	There are no exceptions.

5.1.6 – Unlock a car

This use case is also documented by sequence diagram 5.3.2.

Actors	User
Entry Conditions	The user is near a car that he/she reserved. On his/her mobile devices is shown the PowerEnJoy Personal Area page which shows the position of the reserved car and the button to unlock it.
Flow of events	1. The user clicks on the button “Unlock” 2. The system stops the timer for the fee and send a command to the car to unlock the doors. 3. The car receive the signal and unlocks the doors. 4. The user hear and sea the doors unlocking and he can now enter into the car. (this is the use case “Enter the car” in the diagram) 5. See use case “Power On” below
Exit conditions	• The user can get into the car • Timer for fee application is stopped. • Car is in Ready status.
Exceptions	The reservation expired and the user was charged a fee of 1 EUR (see use case “Apply Fee” above).

5.1.7 – Power On

Name	**Power On**
Actors	User
Entry Conditions	The user is near or into an Unlocked car and he/she sees the button “Power On” in the car.
Flow of events	1. The user presses the button “Power On” 2. The car tells the system it is starting and sends the number of seats occupied. 3. The system saves start time and creates a Pooling Discount if the number of seats occupied is more than two. 4. The systems sands a feedback to the car which now powers on. 5. The user can sees the display turn on and show the actual charging cost. He/she understands the car is powered on.
Exit conditions	• The user can now drive the car. • System has persistently saved start time and number of seats occupied. • Car's engine is on. • Car's state is Running.
Exceptions	

5.1.8 – View charging cost

Name	**View cost**
Actors	User
Entry Conditions	
Flow of events	
Exit conditions	
Exceptions	

5.1.9 – Power Off

Name	**Power Off**
Actors	User
Entry Conditions	The user is on a Running car and in a valid Parking Area.
Flow of events	1. The user presses the “Power off” button. 2. The car sends its GPS position to the system. 3. The system checks the position is in a valid Parking Area. 4. The car receive a positive feedback and powers off the engine. 5. The user understands the car is off but can still see the charging cost increase on the screen.
Exit conditions	• Car is in Parked state. • Engine is off. • The user can leave the car.
Exceptions	The GPS position provided doesn't belong to a valid Parking Area: the system sends a negative feedback to the car which keeps the engine on and ask the user through the screen to kindly move to a valid Parking Area. Car is still in Running state.

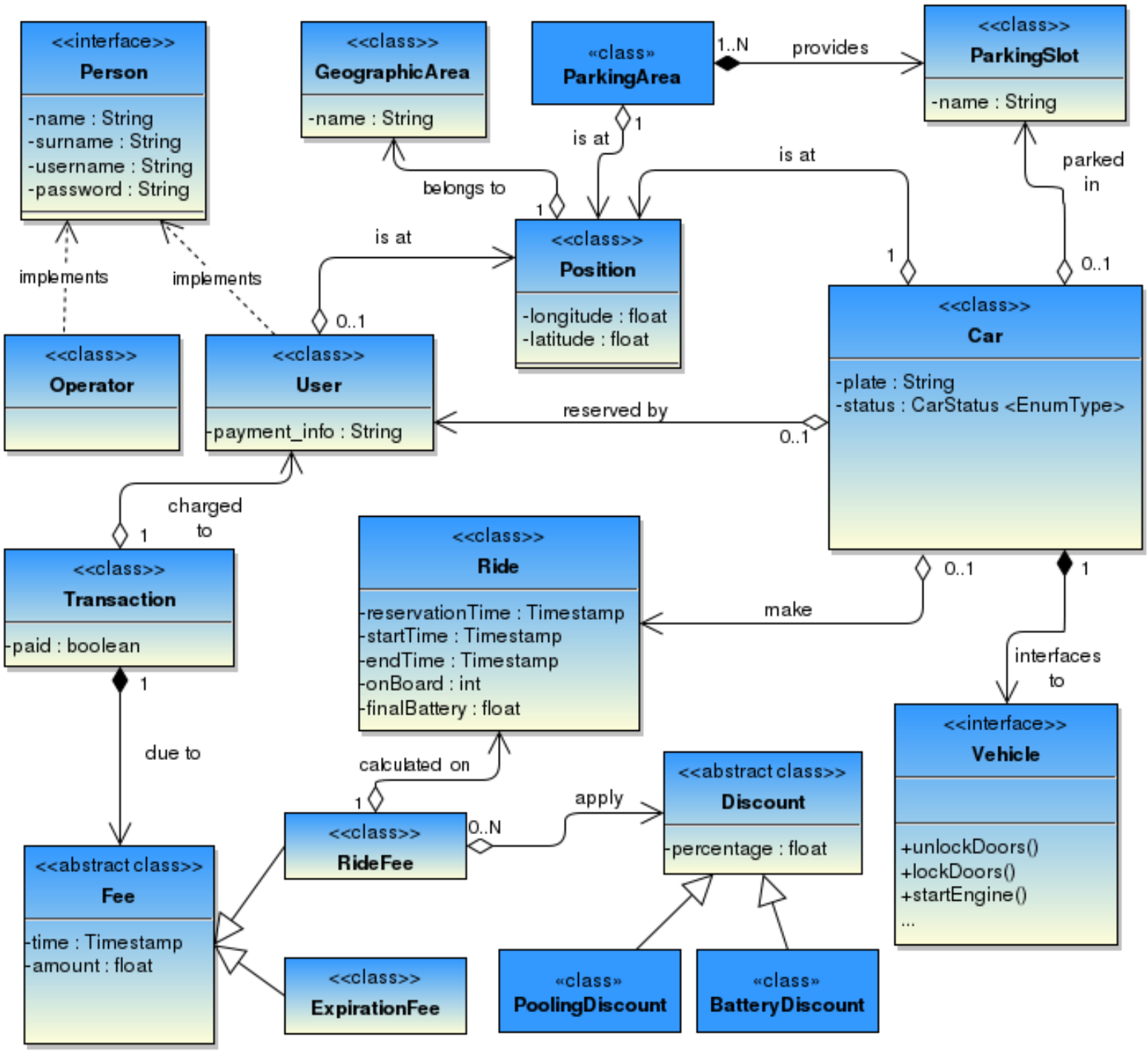
5.1.10 – Lock

Name	**Lock**
Actors	User
Entry Conditions	The user is in a parked car.
Flow of events	1. The user leaves the car and closes the doors. 2. The car detects there is no seats occupied and that the doors are closed. 3. The user sees the doors locked and can move away. 4. The car locks the doors then send to the system its battery status and the notification it locked. 5. The system saves the end time of the ride. 6. The system add a Battery Discount if the battery is more than 50%. 7. The system sets the car in Available state. 8. The system computes the transaction (see use case “Create Transaction” below)
Exit conditions	• Car is Available. • Doors are locked. • The system persistently saved ride ending time. • The user is (probably) no more interacting with the system. • A discount may be created.
Exceptions	

5.1.11 – Create transaction

Name	**Create Transaction**
Actors	Payment Gateway
Entry Conditions	A car has just locked.
Flow of events	1. The system computes the invoice taking into account also discounts. 2. The bill is sent to the payment gateway with payment credentials. 3. The payment gateway do its internal job and send a positive feedback to the system. 4. The system stores the transaction as Paid.
Exit conditions	• The transaction is persistently stored. • PowerEnJoy got the money.
Exceptions	Payment is rejected: transaction is stored as Not Paid.

5.2 – Class diagram



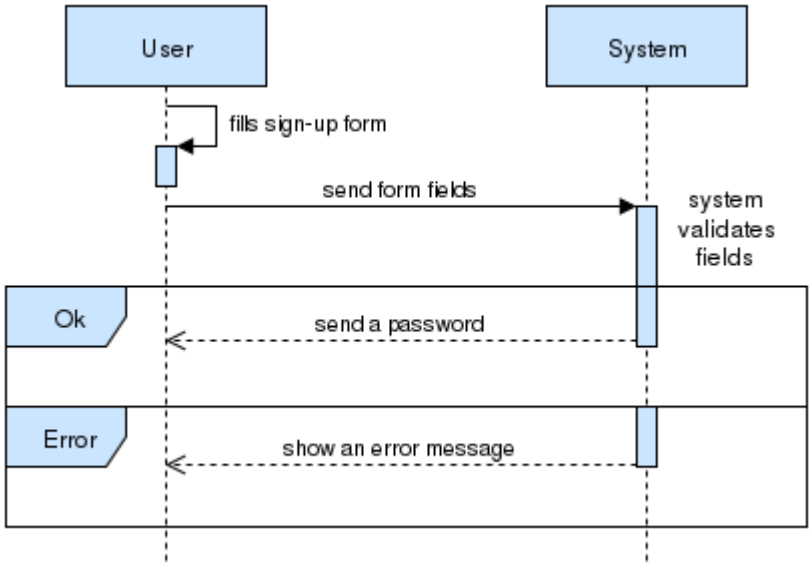
5.3 – Sequence diagram

5.3.1 – Sign-Up and Sign-In

The following sequence diagrams illustrate sequence of action and the message exchanged by the user and the system during registration and login.

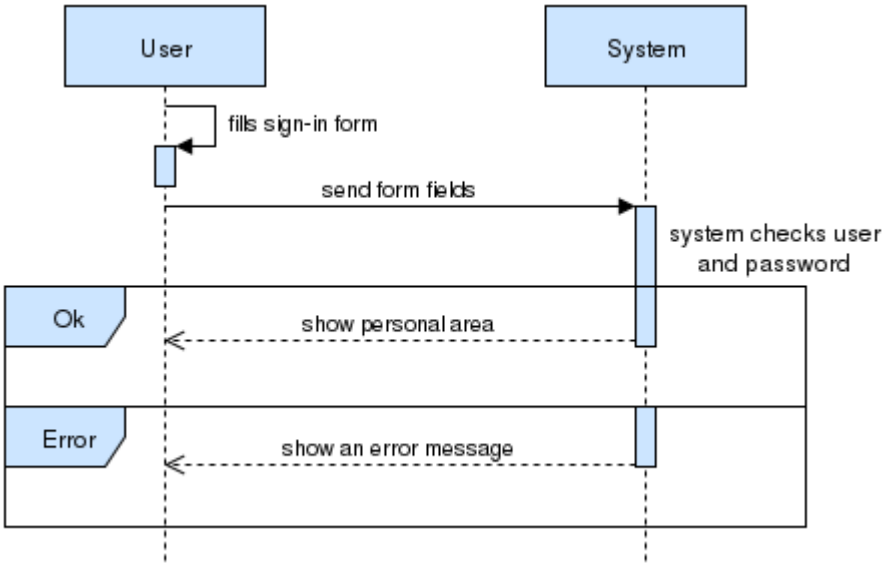
Sign-Up

The diagram shows the flow of action between user and system in use case 5.1.1.



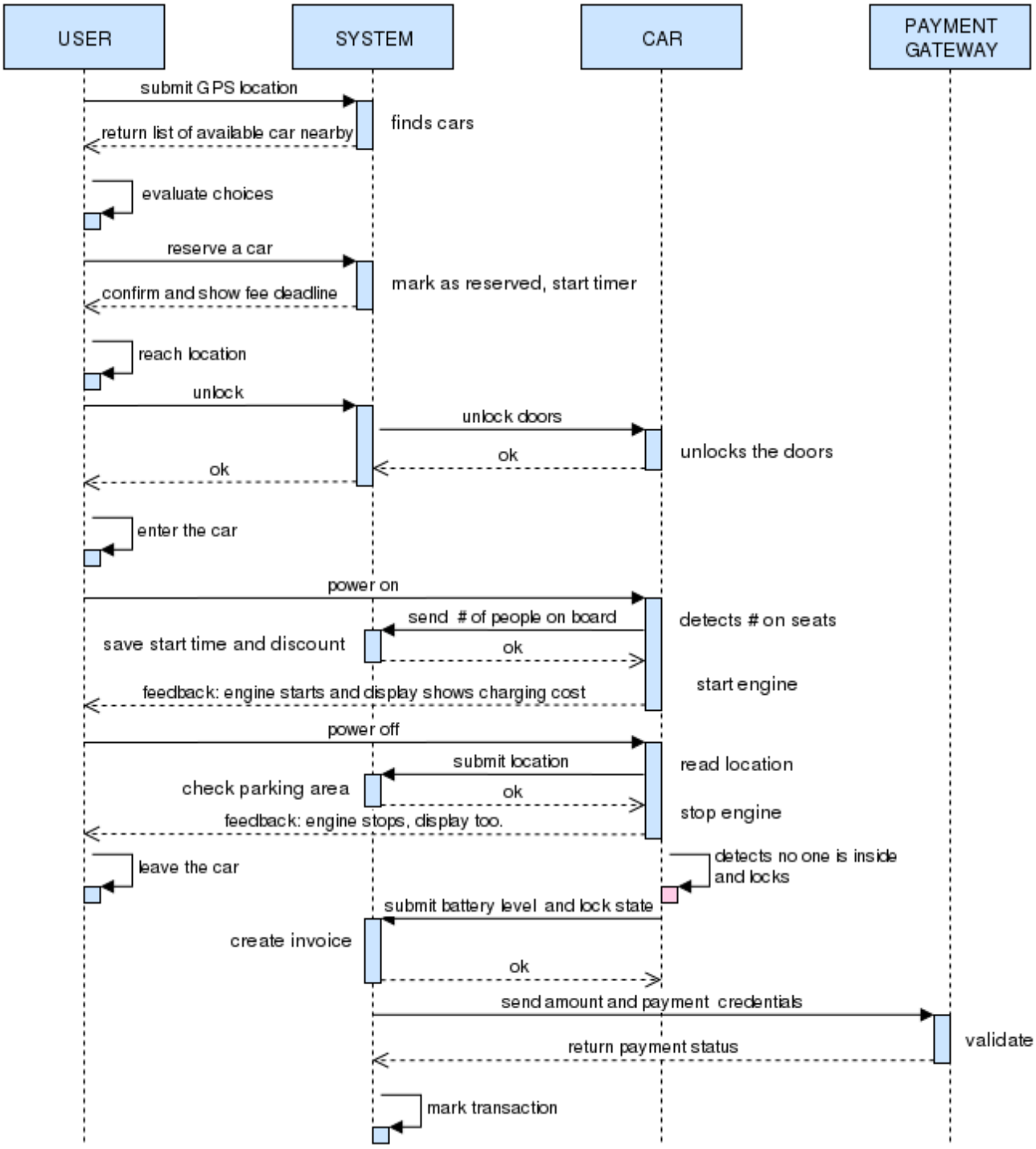
Sign-In

The diagram shows the flow of action between user and system in use case 5.1.2.



5.3.2 – Car renting

In the diagram below are represented two actors (User and Payment Gateway) and two parts ot the system: the module installed on the car and the cloud (denoted as system). This diagram focuses on the exchange of messages between this modules. As expected the user starts to interact with the car only after having unlocked it, then the car notify its status updates to the main system while the user interacts only with the car. Finally the system interact with the payment gateway and updates transaction state.

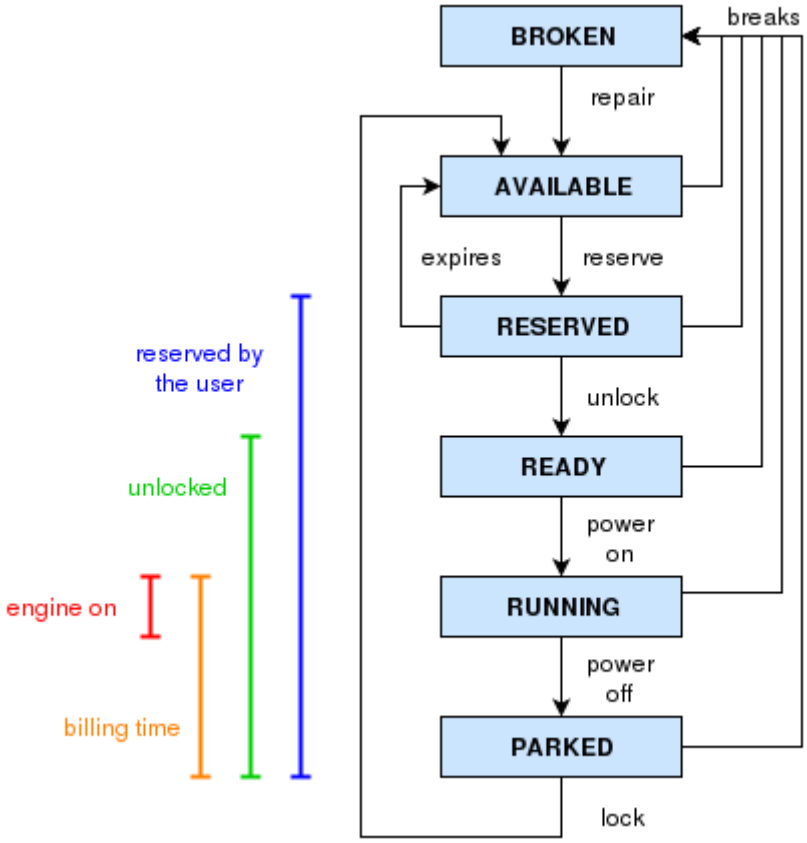


5.4 – State chart diagram

The only two object of the system who has a state are Car and Transaction. Here are their diagrams.

5.4.1 – Car states

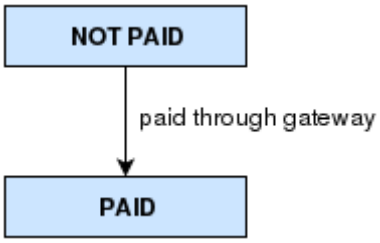
This diagram shows the evolution of the state of the car. On the left are highlighted the properties that hold for the relative set of states.



- **reserve**: the user action of reserving a car make the car change state to reserved.
- **expires**: the alarm set after 1 hour since reservation fired and the car is set to available.
- **unlock**: the user unlocks the car through the system and he can now enter into it.
- **power on**: the user presses the button “Power On” and the system start charging the ride.
- **power off**: the user presses the button “Power Off” in a parking area and the car turns off.
- **lock**: the user left the car and doors are closed. User is charged and car is set to available.
- **breaks**: in any moment, in any state, the car can break leading it to the broken state.
- **repair**: a broken car that is repaired becomes available again to be reserved.

5.4.2 – Transaction states

A transaction can only be in two states: “Paid” and “Not Paid”. A transaction can never go from the Paid state to the Not Paid state by assumption. Instead a Not Paid transaction once processed by the payment gateway with a positive feedback becomes Paid forever.



6. Alloy analyzer

The model described in previous chapters is now analyzed exploiting Alloy analyzer. In the first section of this chapter the model is formalized in alloy language. In the second section are shown results obtaining running the analyzer. In the last section there a few sketches of the generated worlds.

6.1 – Model

6.1.1 – Signatures

```
sig Position{ geo: one GeographicArea }
sig GeographicArea{}
sig ParkingArea{}
sig ParkingSlot{ belongs_to : one ParkingArea, geo_park : one Position}

sig User{ usr_gps : lone Position }
sig Car{
   car_gps : one Position,
   status : one CarStatus,
   reserved_by : lone User,
   current_ride : lone Ride,
   parked_in : lone ParkingSlot,
   empty : lone Int,
}{ empty=0 or empty=1}

abstract sig CarStatus{}

one sig Available extends CarStatus{}
one sig Reserved extends CarStatus{}
one sig Ready extends CarStatus{}
one sig Running extends CarStatus{}
one sig Parked extends CarStatus{}
one sig Broken extends CarStatus{}

sig Timestamp{}
sig Ride{
   reserve_time : one Timestamp,
   start_time : lone Timestamp,
   end_time : lone Timestamp,
   max_on_board : lone Int,
   final_battery_level : lone Int,
}{max_on_board>0  and final_battery_level>=0 and final_battery_level<=5}

one sig Transactions{ trans : User -> Fee }

abstract sig Fee {
   amount : one Int,
   fee_status : one FeeStatus
}{amount>=0}
```

```
one sig ExpiredFee extends Fee{}
sig RideFee extends Fee{
  ride : one Ride,
  discounts : set Discount,
}
abstract sig FeeStatus{}

one sig Paid extends FeeStatus{}
one sig NotPaid extends FeeStatus{}

abstract sig Discount{}

one sig Pooling extends Discount{}
one sig Battery extends Discount{}
```

6.1.2 – Facts

```
fact AvailableStatus{
  all c : Car | (c.status=Available) implies
    (c.reserved_by=none and
     c.current_ride=none and
     c.parked_in!=none)
}
fact ReservedStatus{
  all c : Car | (c.status=Reserved) implies
    (c.reserved_by!=none and
     c.current_ride!=none and
     c.parked_in!=none and
     c.current_ride.reserve_time!=none and
     c.current_ride.start_time=none and
     c.current_ride.end_time=none)
}
fact ReadyStatus{
  all c : Car| (c.status=Ready) implies
    (c.reserved_by!=none and
     c.current_ride!=none and
     c.parked_in!=none and
     c.current_ride.reserve_time!=none and
     c.current_ride.start_time=none and
     c.current_ride.end_time=none)
}
fact RunningStatus{
  all c : Car | (c.status=Running) implies
    (c.reserved_by!=none and
     c.current_ride!=none and
     c.current_ride.reserve_time!=none and
     c.current_ride.start_time!=none and
     c.current_ride.end_time=none)
}
fact ParkedStatus{
  all c : Car | (c.status=Parked) implies
    (c.reserved_by!=none and
     c.current_ride!=none and
     c.parked_in!=none and
     c.current_ride.reserve_time!=none and
     c.current_ride.start_time=none and
     c.current_ride.end_time=none)
}
```

```
fact BrokenStatus{
   all c : Car | (c.status=Broken) implies
      (c.reserved_by=none and
       c.current_ride=none)
}

fact AllSlotsInAnAreaHaveTheSameGeo{
   all ps1, ps2 : ParkingSlot | ps1.belongs_to=ps2.belongs_to implies (ps1.geo_park.geo=ps2.geo_park.geo)
}

fact SingleReservation{
   no c1,c2 : Car | c1.reserved_by = c2.reserved_by and c1!=c2
}
fact RideInTransactionOrCar{
   no r : Ride | (r in Car.current_ride and r in RideFee.ride) or !(r in Car.current_ride or r in RideFee.ride)
}
fact NoDuplicatedRide{
   all rf1,rf2 : RideFee | rf1.ride=rf2.ride implies rf1=rf2
   all c1,c2 : Car |  c1.current_ride=c2.current_ride implies c1=c2
}
fact RideIsCompleteInRideFee{
   all r : Ride | (r in RideFee.ride) implies
      (r.reserve_time!=none and
      r.start_time !=none and
      r.end_time!=none and
      r.max_on_board!=none and
      r.final_battery_level!=none)
}

fact CarGPSMustMatch{
   no c : Car, p : ParkingSlot | c.parked_in=p and c.car_gps!=p.geo_park
}
fact OneCarPerParkingSlot{
   no c1,c2 : Car | c1.parked_in = c2.parked_in and c1!=c2
}
fact OneCarPerParkingSlot{
   no ps1,ps2 : ParkingSlot | ps1.belongs_to=ps2.belongs_to and ps1.geo_park.geo!=ps2.geo_park.geo
}
fact NoFeeOutOfTransactions{
   all f : Fee | f in User.(Transactions.trans)
}

fact NoCarForUntrustedUser{
   no c : Car, u:User,f:Fee | c.reserved_by=u and f.fee_status=NotPaid and (u->f) in Transactions.trans
}
fact ExpiredFeeAmount{
   all f : ExpiredFee | f.amount=1
}

fact ApplyPoolingDiscount{
   all r : RideFee | (r in User.(Transactions.trans) and r.ride.max_on_board>=3) iff (Pooling in r.discounts)
   /*all r : RideFee , d : Discount |
         (r in User.(Transactions.trans) and r.ride.max_on_board>=3 and d=Pooling)
      implies
         (d in r.discounts)*/
}

fact ApplyBatteryDicount{
   all r : RideFee | (r in User.(Transactions.trans) and r.ride.final_battery_level>=3) iff (Battery in r.discounts)
}


fact ClosedCarIsEmpty{
   all c : Car | (c.status=Reserved or c.status=Available) implies c.empty=1
}
```

6.1.3 – Predicates

```
pred Reservable[ c: Car, u : User]{
  c.status=Available
  no c1: Car | c1.reserved_by=u
  c.car_gps.geo=u.usr_gps.geo
  no f:Fee | f.fee_status=NotPaid and (u->f) in Transactions.trans
}
pred Unlockable[ c: Car, u : User]{
  c.status=Reserved
  c.reserved_by=u
}
pred Parkable[ c: Car]{
  c.status=Running
  c.parked_in!=none
}
pred Lockable[ c: Car]{
  c.status=Parked
  c.empty=1
}

pred SatisfiesPoolingDiscount[ r : Ride]{
  r.max_on_board>=3
}

pred SatisfiesBatteryDiscount[ r : Ride]{
  r.final_battery_level>=3
}
```

6.1.4 – Assertions

```
assert ReserveOnlyInSameGeo{
  no u : User, c : Car | Reservable[c,u] and u.usr_gps.geo!=c.car_gps.geo
}
assert OneCarPerUser{
  no u:User | #reserved_by.u>1
}
assert AllowUnlock{
  all u : User, c : Car | (u=c.reserved_by and c.status=Reserved) implies Unlockable[c,u]
  no u : User, c : Car | u!=c.reserved_by and Unlockable[c,u]
}
assert PowerOffInParkingAreas{
  all c : Car | Parkable[c] implies c.parked_in in belongs_to.ParkingArea
}
assert LockTheCarWhenNoOneInside{
  all c : Car | (c.status=Parked and c.empty=1) implies Lockable[c]
}
assert ApplyPoolingDiscount{
  all rf : RideFee, d : Discount | (SatisfiesPoolingDiscount[rf.ride] and d=Pooling) implies d in rf.discounts
  all rf : RideFee, d : Discount | (!SatisfiesPoolingDiscount[rf.ride] and d=Pooling) implies d not in rf.discounts
}
assert ApplyBatteryDiscount{
  all rf : RideFee, d : Discount | (SatisfiesBatteryDiscount[rf.ride] and d=Battery) implies d in rf.discounts
  all rf : RideFee, d : Discount | (!SatisfiesBatteryDiscount[rf.ride] and d=Battery) implies d not in rf.discounts
}
assert pUserWithUnpaidTransactionCannotReserve{
  no u: User, c: Car , f :Fee| Reservable[c,u] and f.fee_status=NotPaid and (u->f) in Transactions.trans
}
```

6.2 – Results

This are the results of the Alloy Analyser.

```
Executing "Check ReserveOnlyInSameGeo"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   0 vars. 0 primary vars. 0 clauses. 8ms.
   No counterexample found. Assertion may be valid. 0ms.

Executing "Check OneCarPerUser"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5064 vars. 363 primary vars. 11385 clauses. 17ms.
   No counterexample found. Assertion may be valid. 5ms.

Executing "Check AllowUnlock"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   0 vars. 0 primary vars. 0 clauses. 13ms.
   No counterexample found. Assertion may be valid. 0ms.

Executing "Check PowerOffInParkingAreas"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5098 vars. 363 primary vars. 11413 clauses. 12ms.
   No counterexample found. Assertion may be valid. 3ms.

Executing "Check LockTheCarWhenNoOneInside"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   0 vars. 0 primary vars. 0 clauses. 9ms.
   No counterexample found. Assertion may be valid. 0ms.

Executing "Check ApplyPoolingDiscount"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5030 vars. 360 primary vars. 11304 clauses. 11ms.
   No counterexample found. Assertion may be valid. 3ms.
```

```
Executing "Check ApplyBatteryDiscount"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5030 vars. 360 primary vars. 11304 clauses. 11ms.
   No counterexample found. Assertion may be valid. 4ms.

Executing "Check UserWithUnpaidTransactionCannotReserve"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5233 vars. 369 primary vars. 11703 clauses. 12ms.
   No counterexample found. Assertion may be valid. 1ms.
```

6.3 – Worlds generated

7. Future developments

8. Used tools

- Alloy Analyser 4.2
- Writer
- draw.io

9. Hour of works

Introduction 5

Actor 1

Requirements 3

Scenarios 2

UML 3

- Use case 5
- Class diagram 2
- Sequence 1
- State 1,5

Alloy 6

All the rest 2,5+

10. Changelog

Here are listed the revisions of this document containing a description of the change made and the reason behind that.

- 01/01/9999 – RASD document version 1.0