Software Engineering 2

Requirements analysis and system specification document

POLITECNICO
DI MILANO

Version 1.0

Alexsander H. Baldin 876273
Jesús Bermejo herrero 878667

Table of Contents

Introduction

Description of the given problem

The system PowerEnJoy will be a mobile application for car-sharing services, its difference from the others is that it will only employ electric cars.
The users will need to register in the app, inserting the credentials and the payment information, after that he will be able to check for available cars in his area or near an specific address, and depending on his distance from the car, he can reserve the car for an hour until his arrival on the location.
The system starts charging the user once the engine ignites through a request in the mobile app and stops when the car is parked in a safe area. Depending on the circumstances the car is left in the safe area, the system will apply a discount on the ride, but if the car is left with low battery and far from a charge station, the system will charge an extra amount.
The safe areas are pre-defined by the responsible for the system's management in a way that the car will be spread through the whole city.

Goals

Clients:

1) Register in the system
2) Log in the system
3) Find available cars
4) Reserve a car
5) Unlock the requested car once near it

Domain properties

It is supposed that the following properties are true:

- The client's payment information will be always right and he will always have money to pay for the ride;
- The GPS gives the right position of the client;
- The client can reserve a car for one hour, if he doesn't enter it until the time limit, the car will be available for other people to take it;
- Each client has a single login in the system and can reserve only one car at a time;
- If the car is left out of a safe area or without battery, the client will be charged an extra amount and an operator will need to get the car;

Text Assumptions

- The application needs access to GPS or use a location informed by the client to do the search for available cars;
- Once a car is reserved, the client has one hour to enter it and it will not be available for other users during this time;
- The client will send a request in the application to unlock the car when he is near it;
- The system starts charging the client when he ignites the engine and stops when he leaves the car in a safe area;
- If there were at least two passengers in the ride, the system will apply a discount of 10%;
- If the car is left with more than 50% of the battery full, a discount of 20% will be applied;
- If the car is left in a special charge station and is plugged into the power grid, the discount will be 30% of the last ride;
- If the car is left at more than 3km from a charge station or with less than 20% of its battery, the client will be charged an extra of 30% on the ride for the work to recharge it.
- The system assumes that the car will always be parked in a safe area, otherwise the client will be charged an amount to compensate the trouble to send someone to transport the car to a safe area.

Constraints

Regulatory policies

The system will need to manage sensible data from the client, such as his payment information and position, always respecting the privacy law.

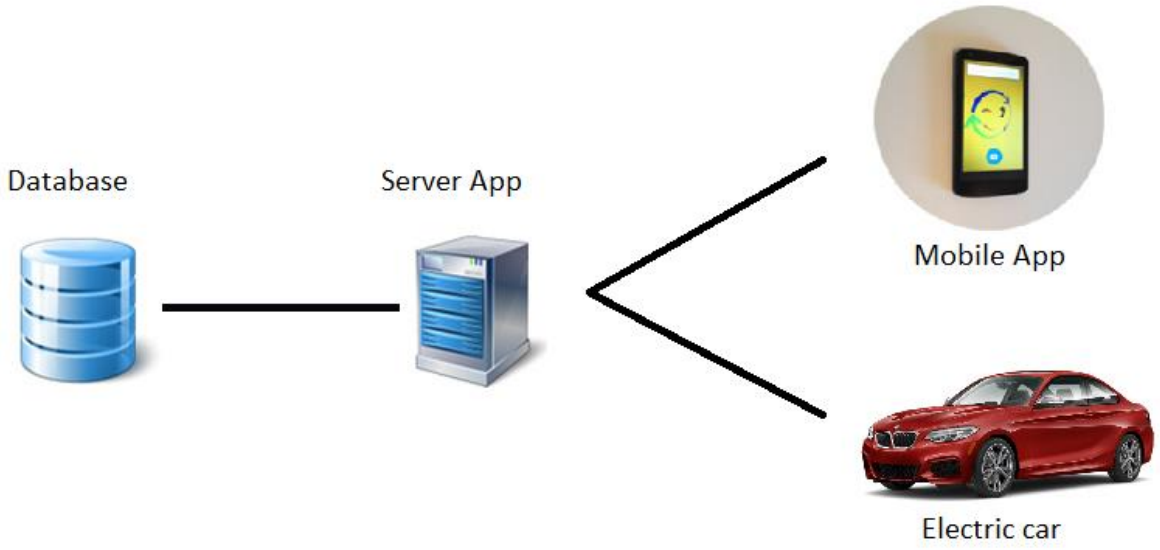
Hardware limitations

Mobile Application:

- Internet connection
- GPS
- Space to install the app

Proposed system

The client-server architecture will be used to implement the system, with only one server that manages the information from the mobile applications and the cars spread through the city.



Identifying stakeholders

The enterprise that needs the PowerEnJoy application would be the main stakeholder of the system, so the it needs to have the best usability, confiability and efficiency possible.

Reference Documents

- Specification Document: Assignments AA 2016-2017.pdf
- IEEE standard on requirement engineering.pdf
- Examples documents:
    - RASD sample from Oct. 20 lecture.pdf

Actor identifying

The system will have only one which is the client:

- The client: He needs to register in the system, when he is identified he can rent an electric car. To be able to drive it, he must reserve the car and give the proximity notice.

Requirements

Functional requirements

The server, the database, the cars and the possible car parks are thanks to PowerEnJoy.

Car:

- The system must be able to check if the driver has the reservation made.
- The system only allows to open the car if the reservation made is correct.
- The system must know the location of the car at all times even if it is occupied.
- The system must provide a quality of service such that two users can not reserve the same car at the same time.
- The system must be connected to the car at all times in order to perform these tasks.

Clients:

- It is necessary that the client registers in the server to be able to make a reservation of an electric car.
- The system must be able to check the position of the client.
- The system must transfer the location of the nearest vehicle to the customer's position.
- Customers can request a reservation of a vehicle with an hour of advance to pick it up.
- The customers have to receive the identifier of the rented vehicle and its location.

Scenarios identifying

Scenario 1

When you have to go on a trip and you have to carry your bags until you get a proper transport you can take a car to carry your suitcase at this time it is convenient to rent an electric car that for very little money allows you to carry your suitcase quickly and effortlessly Since if you put it in the subway and it is very uncomfortable you can have unfortunate inconveniences at this moment from home before leaving for the train you can go you can go to the application to book a car and you will tell the car that is closest Of your location being so you can go for it without loading the suitcase pass with the car by your house and pick up the suitcase so as to take it to the station and there you leave the car and you carefree you have arrived at the station in a comfortable way and fast.

Scenario 2

There are times when you have to go to a polygon that is on the outskirts of the city and in urban transport can take a long time to arrive carrying one of the cars you can quickly go to your destination without needing to use urban transport so you can arrive A shorter time without need to make transshipments since you only have to rent the car and once you get there you can give it to reserve again to leave in less than an hour or when you see that you are finishing and you will need you can reserve it

Scenario 3

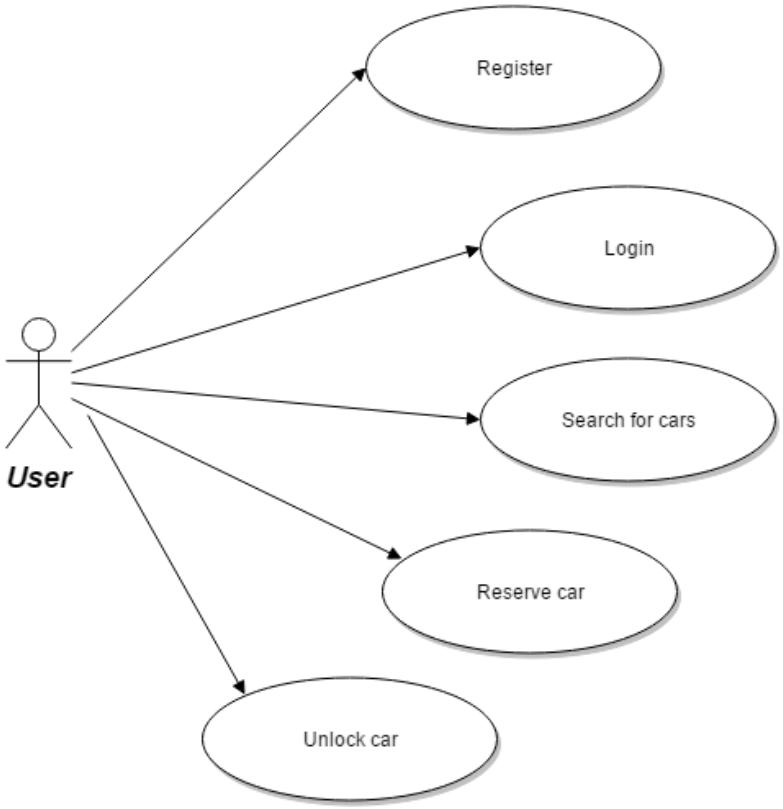
In some moments you can stay with other people to go for dinner usually you are going to drink some wine or alcohol in general therefore it is not convenient to return from with the car therefore you can rent one of our electric cars to go up Dinner and once there you forget about him since you do not have to need more so this is how you can go in a car to the place of dinner and then not worry about not having to drink or police checks.

Scenario 4

You can also happen that someday you fall asleep in bed and when you wake up is a little late therefore you will be late to work at this time can replace the public transport that is always slower than the car, Wake up you can go where the nearest vehicle is and reserve to get to work at your normal time,

UML models

Use case diagram



Use case description

Client registers himself

Name: Client registers himself
Actor: Client
Entry conditions: There are none
Flow of events:

- The client enter in the mobile application
- The client clicks in the register button
- The client inputs his informations (email, password, payment info, etc.)
- The client clicks the confirmation button
- The system creates a login for that client and return to the home page

Exit conditions: The system creates a new user in the system and go to the home page.
Exceptions: The client inputs inconsistent information, if this happens, the system tells the client which informations are wrong and wait new inputs.

Client logs in the system

Name: Client logs in the system
Actor: Client
Entry conditions: There are none
Flow of events:

- The client enter in the mobile application
- The client inputs his credentials (username and password)
- The client clicks the button to log in
- The system redirects the client to the map screen

Exit conditions: The system redirects the user to the map.
Exceptions: The client credentials are wrong, so the system warns the client that his username or password are invalid.

Client searches for cars

Name: Client searches for cars
Actor: Client
Entry conditions: The client needs to have logged in the system
Flow of events:

- The client enter the screen with the map
- The client clicks to button to search for available cars nearby or from an address
- The system shows in the map the locations of the cars that are available

Exit conditions: The system shows icons where the available cars can be found.
Exceptions: If the system doesn't find any car, he shows a message for the client that no car was found.

Client reserves a car

Name: Client reserves a car
Actor: Client
Entry conditions: Client is logged in
Flow of events:

- The client selects one of the available cars in the map
- The system changes the status of that car to reserved for one hour until the client gets to it

Exit conditions: The system sets the car as reserved and the client receives a confirmation
Exceptions: If the client doesn't reach the car in one hour, the car becomes available again and the client needs to pay a fee of 1 euro.

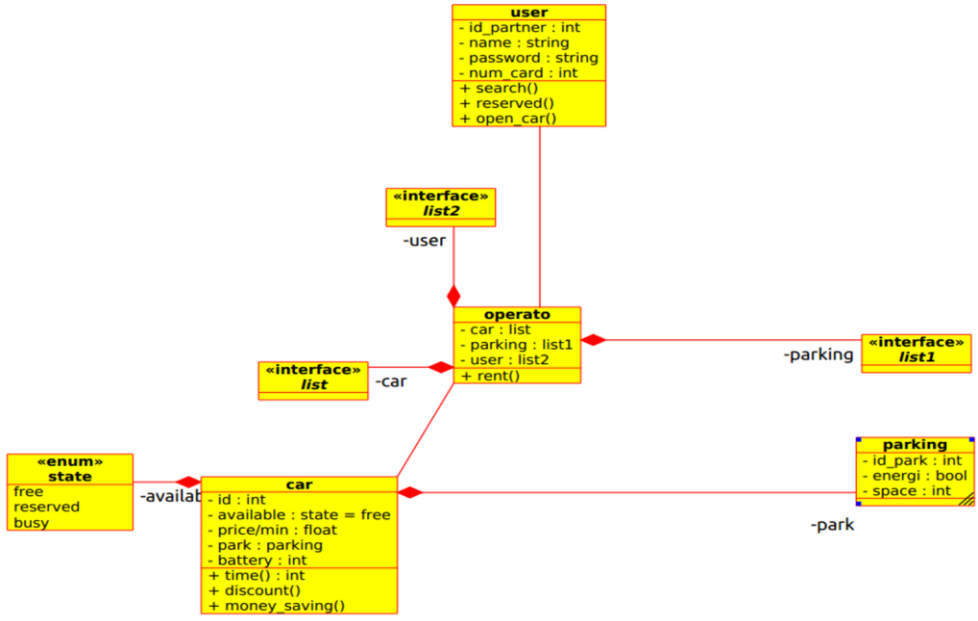
Client unlocks car

Name: Client unlocks car
Actor: Client
Entry conditions: Client is logged in and is near the car he has reserved
Flow of events:

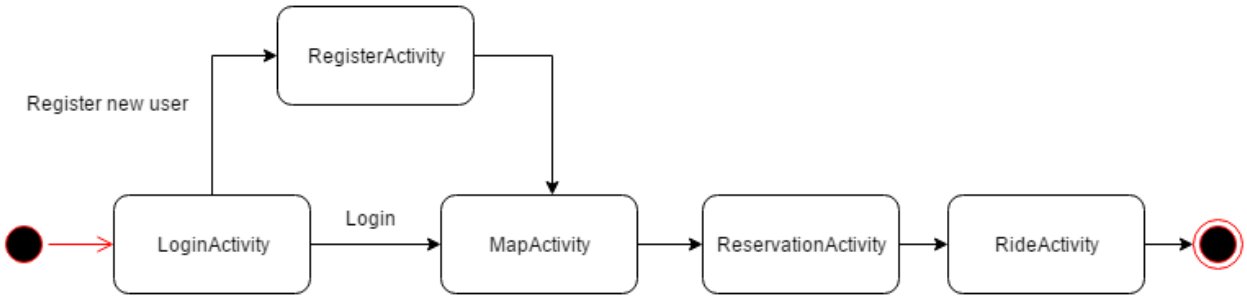
- The client enter the screen of the car he has reserved and click to unlock it
- The system checks if the client's location is close to the car's and unlocks the door and change the status of the car to busy

Exit conditions: The system opens the car.
Exceptions: The client is far from the car, then the system informs the user that he needs to get closer for the car to be unlocked.

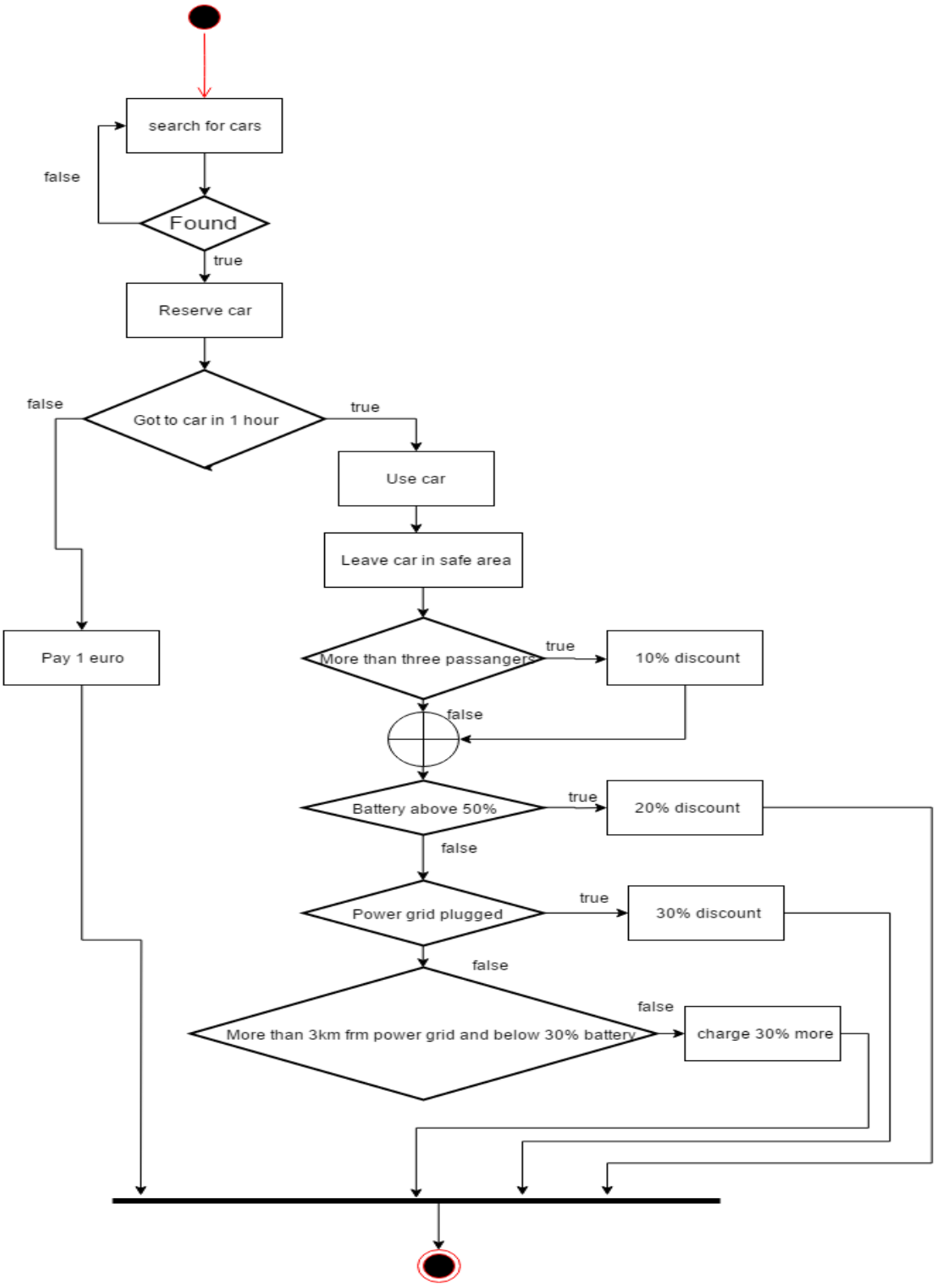
Class Diagram



State diagram



Activity diagram



Alloy modeling

Model

```
open util/boolean
sig operator {
car: ArrayList,
parking: ArrayList,
user: ArrayList
} {
id_user > 0
id_car > 0
id_parking > 0
}
fact IdCarUnique {
all c1,c2: car | (c1 != c2) => c1.id != c2.id
}
sig Car {
car: one car,
available: one state,
park: one parking,
battery: one int
}

sig CardNumber { }
sig User {
cardNumber: cardNumber
}
fact ParkingAreUnique {
all p1,p2: parking | p1 != p2 => p1.id_park != p2.id_park
}
sig Position {
latitude: int,
longitud: int
}
abstract sig CarPosition extends Position {
car: one car
}
```

```
sig Path {
positions: seq CarPosition
} {
#positions = 1
}
fact pathPositionsAreUnique {
all p: Path | not p.positions.hasDups
}

fact pathWithLoad {
all p: Path | p.positions in CarPosition
}

assert clientGetInAndGetOut {
all p: Path| all i1: p.positions.inds | one i2: p.positions.inds |
i2 != i1 and let p0=p.positions[i1] | let p1=p.positions[i2] |
p0.complement[p1]
}
assert pathSameNumberOfLoadAndDrop {
all p: Path | #(p.positions.elems & DropPosition) = #(p.positions.elems & LoadPosition)
}
abstract sig Request {
path: Path,
passengers: Int
} {
passengers > 0
}
sig SingleRequest extends Request {
}
sig SharedRequest extends Request {
}
sig MergedRequest extends Request {
}
abstract sig Queue {
s: seq univ
}
sig CarQueue extends Queue {
} {s.elems in Car
}
sig RequestQueue extends Queue {
} {
```

```
s.elems in Request
}
fact enqueuedCarMustBeAvailable {
all d: carQueue.s.elems | d.available.isfree
}
fact availableCarMustBeEnqueued {
all d: car | d.available.isfree <=> d in DriverQueue.s.elems
}
fact enqueuedElemntsMustBeUnique {
all q: Queue | not q.s.hasDups
}
fun peek[s: (seq/Int -> univ)]: univ {
s.first
}
fun peek[q: Queue]: univ {
q.s.peek
}
fun peek[q: carQueue]: car {
q.s.peek & car
}
fun peek[q: RequestQueue]: Request {
q.s.peek & Request
}
pred Queue.add[q: Queue, e: univ] {
q.s = this.s.insert[#this.s, e]
}


pred Queue.get[q: Queue, e: univ] {
e in this.s.elemsq.s = this.s.delete[0]
not e in q.s.elems
}
assert getInverseOfAdd {
all q0,q1,q2: Queue, e: univ | q0.s.isEmpty and
q0.add[q1, e] and q1.get[q2, e] => q0.s = q2.s
}
sig Zone {
car: one carQueue,
requests: one RequestQueue,
bounds: seq Position
} {
#bounds > 2
```

```
all p: bounds.elems | not p in StopPosition
not bounds.hasDups
}
fact oneQueueForZone {
all z1, z2: Zone | z1 != z2 =>
z1.car != z2.car and z1.requests != z2.requests
}
fact zoneOwnAllCarQueues {
Zone.cas = carQueue
}
fact zoneOwnAllRequestQueues {
Zone.requests = RequestQueue
}
sig Ride {
car: some car,
path: Path,
prices: Client -> Int
}
fact ownAllCar {
operato.car = car
}
fact ownAllClients {
operato.clients = Client
}
fact ownAllParking {
operato.parking = Zone
}
pred show() {
#operato = 1
#Ride = 1
#SingleRequest = 1
}
run show for 4
check pathPositionsAreEven
check clientGetInAndGetOut
check pathSameNumberOfLoadAndDrop
check getInverseOfAdd
run add for 4
run get for 4
```

Used tools

We have used the following tools to create this RASD

- Text editor Google Drive.
- Draw.io
- We have used umbrello for the creation of uml

Hour of work

Jesús Bermejo Herrero

6/11/16 → 5 h
12/11/16 → 6 h
13/11/16 → 6 h

Alexsander H. Baldin

5/11/16 → 2h
6/11/16 → 3h
11/11/16 → 6h
12/11/16 → 8h
13/11/16 → 8h