Requirements Analysis and Specifications Document

Version 1.2

- Andrea Pace
- Tommaso Paulon
- Lorenzo Petrangeli

Table of contents

1-Introduction

1.1-Description of the given problem

We are going to project and implement PowerEnJoy.

PowerEnJoy is a new car-sharing service that exclusively employs electric cars, and is based on a mobile application and a web application.
The system allows the client to sign up for the service, providing some informations (including driving licence and payment informations) . A non-registered user only navigate a map supplied by an external service, and check the available cars by a research system.
An authenticated user, after the login, can reserve a car, searching for it with the same method used by a non-registered user. After the reservation is made, the user can unlock the car (if he is close enough) or cancel the reservation. There is also a “Money saving option”, that the user can activate after unlocking the car.
The payments are managed by a payment handler: the user can choose extinguish his debts directly by the application.

1.2-Regulatory policies

The system must require to the users the permission to get their position and to process their personal data. The system must manage those sensible data respecting the privacy law. The system must send push notifications for recommendations. We assume that every client will be given some reference to the PowerEnjoy operators in case of car malfunctioning.

1.3-Interface to other applications

The system must have an interface with the map provider and a payment handler.

1.4-Parallel operation

The system requires to support parallel operations from different users.

1.5-Reference documents

-Assignments AA 2016-2017 pdf
-RASD sample from Oct. 20 lecture pdf

1.6-Actors Identifying

-Unregistered user
-User
-Operator (just involved in some process)

2-Goals and requirements

[G1] Allow users to sign up, providing their driver license number, payment information and a username

The system must:

[R1] check whether the username is consistent

[R2] check whether the driving license number is consistent

[R3] check whether the payment information is consistent

[R4] check whether the username, the driving license ad the payment information are not already registered

[R5] send back a (valid) password only if the conditions listed above are true

[R6] store credentials, payment information, drive license number, username and password

[G2] Allow users to sign in

The system must:

[R1] let the user log in only if the provided username and password are correct

[G3a] Show available cars within a certain distance from current location

The system must:

[R1] detect the position of every PowerEnjoy car

[R2] detect the current position of the user

[R3] be provided with a map of the area covered by PowerEnjoy

[R4] check whether the specified distance is positive

[R5] provide a map with available cars within the specified distance from the user's position

[R6] Include battery charge among the information about a car

[G3b] Show available cars within a certain distance from a specified address

The system must:

[R1] know the position of every PowerEnjoy car

[R2] check whether the specified distance is positive

[R3] be provided with a map of the area covered by PowerEnjoy

[R4] check whether the address is consistent

[R5] provide a map with available cars within the specified distance from the specified address

[R6] Include battery charge among the information about a car

[G4] Let users reserve a single car

The system must

[R1] allow the user to choose a car from the ones available

[R2] prevent the user from choosing a car if he's already reserved one

[R3] once the car is reserved , it shouldn't be shown in the map between available cars

[G5] If a car is not picked up within an hour from the reservation, the system tags the car as available again, and the reservation expires; the user pays a fee of 1 EUR;

The system must be able to

[R1] start a countdown of a predefined time when the user selects a car

[R2] show the user how much time is left

[R3] detect if the user reaches the car he reserved

[R4] make the car visible in the map to every user if the countdown ends and the user has not notified the system

[R5] increase the amount of money that the user has to pay if the car si not reached before the countdown ends

[R6] cancel the user's reservation after the countdown ends

[G6] A user that reaches a reserved car must be able to tell the system he's nearby, so the system unlocks the car and the user may enter. From that moment on, the user periodically pays an additional amount of money after a predefined time if he/she doesn't start the engine

The system must:

[R1] let the user ask to unlock the car he reserved

[R2] unlock a car when asked to and when the user's distance(via GPS) from the car is lesser than a predefined value

[R3] charge the user with a predefined amount of money every time a predefined time has passed

[G7] As soon as the engine ignites, the system starts charging the user for a given amount of money per minute; the user is notified of the current charges through a screen on the car.

The system must:

[R1] detect when the engine ignites

[R2] be able to measure how long the user keeps the engine running

[R3] charge the user for an amount of money directly proportional to how long the engine is running, starting when the engine ignites

[R4] show the current charges through a screen on the car

[G8] The system stops charging the user as soon as the car is parked in a safe area and the user exits the car; at this point the system locks the car automatically. Then the system submits to the payment handler the total amount of money that the user has to pay.

The system must:

[R1] detect when the engine turns off

[R2] check whether the car is left in a safe area with the engine turned off

[R3] detect when all the passengers leave the car

[R4] stop charging the user when the conditions listed above are satisfied

[R5] be able to lock the car

[R6] interact with the payment handler by submitting the final amount of money and the payment information of the user

[G9] The final amount of money can be increased or decreased depending on the user behaviour. Parking too far from a power grid station or leaving a car with low battery will result in a penalty. Leaving the car with a certain amount of battery charge, taking other passengers onto the car or plugging the car into the power grid inside a safe zone will result in a benefit. An operator must be notified when a car is left with low battery charge and when a car is left ouside the safe area form more than a predefined time.

The system must:

[R1] be able to detect the battery charge of the car

[R2] be able to check whether the car is plugged into the power grid

[R3] calculate the distance between the car and a power grid station

[R4] detect the number of passengers during the travel

[R5] notify an operator if a car is left with battery charge below a predefined percentage

[R6] be able to detect when the engine is turned off

[R7] notify an operator when a car is left with its engine turned off outside the safe area

[G10] If the user enables the money saving option, he/she can input his/her final destination and the system provides information about the station where to leave the car to get a discount. This station is determined to ensure a uniform distribution of cars in the city and depends both on the destination of the user and on the availability of power plugs at the selected station

The system must:

[R1] detect when the money saving option is chosen

[R2] let the user select a destination

[R3] choose the power station closest to the user's destination with a number of free plugs greater or equal to the average number of free plugs among all the power station owned by PowerEnjoy

G[11] If the payment handler notifies the system about incorrect payment informations or about an insufficient balance, the user is prevented from reserving other cars. He can change his payment information and communicate to the system that he wants to pay off his debt. If the payment doesn't go wrong, the user can reserve cars again

The system must:

[R1] be able to receive notifications about successful payments, wrong informations or insufficient balance from the payment handler

[R2] prevent the user from reserving a car if the payment handler sends a notification about wrong informations or insufficient balance

[R3] let the user change his payment informations and ask for a new payment

[R4] let the user reserve cars again if the payment is successful

G[12] The user must be able to cancel his reservation until he starts the engine. He still has to pay a fee of 1 EUR

The system must:

[R1] check if the user has really reserved a car

[R2] check if the engine has been started

[R3] increase the amount of money that the user has to pay if the reservation is canceled

[R4] lock the car as soon as soon as no one is inside

[R5] make the car visible to every user again

[G13] Every operator must be able to change any car status to unavailable. Then they must be able to revert the status back to available.

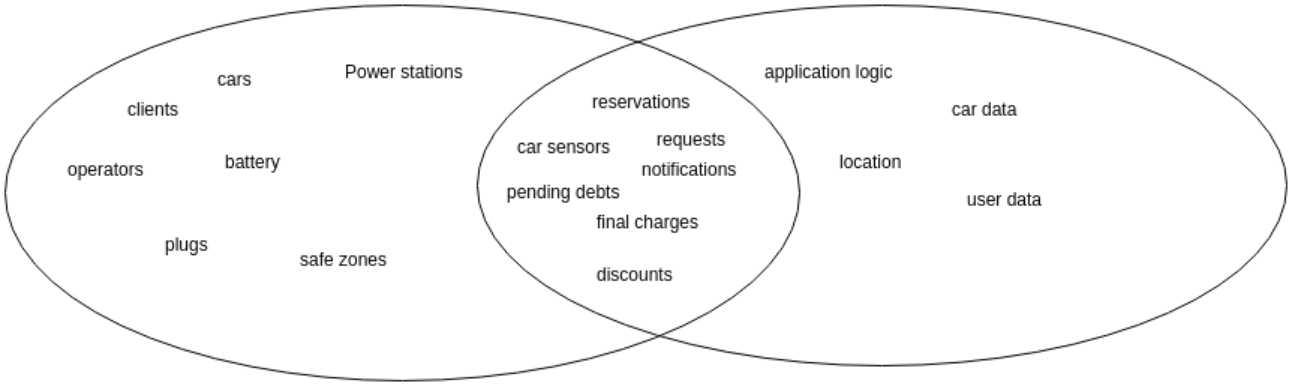
The system must:

[R] be able to change the status of the car

3-Domain assumptions

- the user keeps localization turned on all the time
- the provided driving licence number is correct
- GPS coordinates are always correct
- GPS inside cars can be switched off only remotely by an operator
- The car is equipped with a sensor that detects if a car is plugged into the power grid
- The car is equipped with a sensor that detects the number of passengers
- the car is equipped with a screen
- battery charge measurement is always accurate
- power grids always work
- the car can be locked remotely
- the set of safe areas for parking cars is predefined

Here is a picture of The World And The Machine model that we refer to:



4-Scenario identifying

Scenario 1

Someone stole the car to John Doe, but he needs one to go to work. John is already registered to the service. He finds a car on the app, make his reservation and approaches it by walking. He unlocks the car and then drive up to his office.

Scenario 2

Sarah has an important meeting and decides to use PowerEnJoy. She is already registered to the service. After a few minutes, she discovers that she can't reach the car in less than an hour, so she decides to cancel the reservation. She has to pay a fee of 1 EUR.

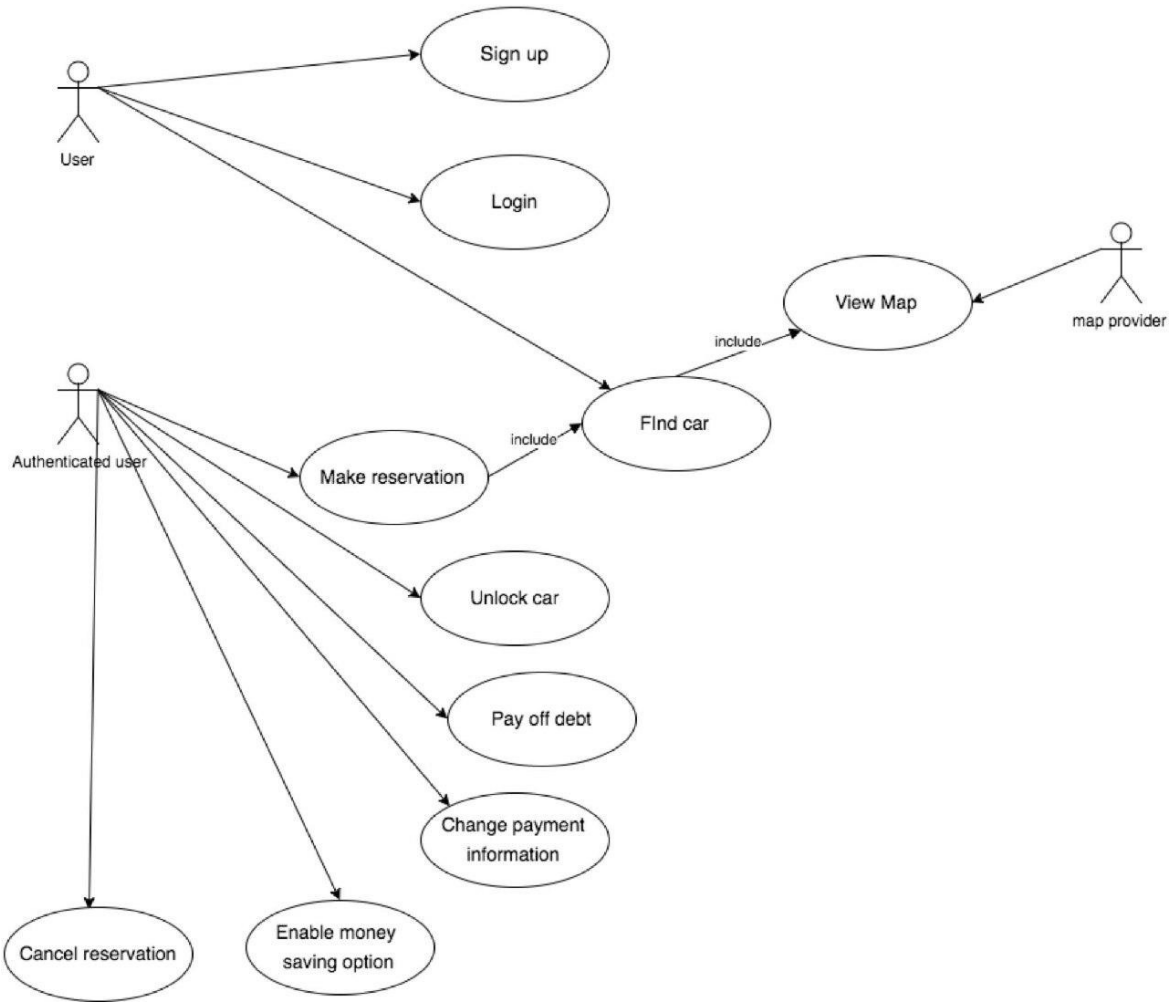
Scenario 3

After going to dance Jane decides to reserve a car via PowerEnJoy. She is not registered to the service. She finds a car through the app, and decides to sign up while approaching it. When she reach the car, she discovers, after the login, that she can't make a reservation on that car because is no longer available. She calls for a taxi, and while waiting two girls approaches to the car and unlock it. She ask them a lift, and they accept, explaining her that they will have a 10% discount because there are 3 users on it.

Scenario 4

Mr. Smith doesn't own a car, and uses PowerEnJoy daily. He decides to pay off the debt that he owe to PowerEnJoy: he opens the mobile app and presses the "Pay Off debt button". After a few seconds a notification reveals him that his payment method is invalid, so he decides to change his payment informtions by the app. After that, he presses on the "Pay off debt button" again, extinguishing his debt.

5-Use case diagram



5.1-Use case description

Name: Sign Up.
Actors: User.
Entry conditions: There are no entry conditions.
Flow of events:

-The user presses the sign up button from the homepage.

-The user inputs his credentials, his payment information, an username and a password.

Exit conditions: The account is successfully created.
Exceptions: The username is not available. In this case, the system notifies the user and ask him to choose another username, until an available one is chosen.

Name: Login.
Actors: User.
Entry conditions: The user must have registered.
Flow of events:

-The user presses the login button from the homepage.
-The user inputs his username and password.
-The system redirects the user to the homepage for authenticated users.

Exit conditions: The user is successfully redirected to his homepage.
Exceptions: The username or the password are not correct. In this case, the system notifies the user and ask him to reinsert them.

Name: Find Car.
Actors: User.
Entry conditions: There are no entry conditions.
Flow of events:

-The user presses the “find a car button” from the homepage.
-The user is redirected by the system to a map page, where he can search for a location. The available cars are shown on the map in their location. The user can navigate the map.

Exit conditions: There are no exit conditions.
Exceptions: The user has entered an invalid address. The system notifies him and ask him to reinsert a valid address.

Name: Make Reservation
Actors: Authenticated user.
Entry conditions: The user must be on his “authenticated user” homepage.
Flow of events:

-The user presses the “make a reservation button” from the homepage.
-The user is redirected by the system to a map page, where he can search for a location.
-The available cars are shown on the map in their location. The user can navigate the map.
-The user can indicate a car on the map. If he does, the system notifies him by a popup, where the user can choose between “confirm reservation” and “cancel”.

Exit conditions: If the user confirms the reservation, the system removes the car from the map and creates a reservation for the user. If he doesn’t, there are not exit conditions.
Exceptions: If the user has a debt, he is prevented from confirming any reservation.

Name: Unlock Car.
Actors: Authenticated user.
Entry conditions:

-The user must be on his “authenticated user” homepage.
-The user must have a reservation.
-The user must be near the car.

Flow of events:

-The user presses the “unlock car button” from the homepage.

Exit conditions: The system unlocks the car, delete the reservation and notifies the user.

Exceptions: if more than an hour has passed, the car becomes available again and the user cannot unlock the car.

Name: Pay Off Debt.
Actors: Authenticated user.
Entry conditions:
-The user must be on his “authenticated user” homepage.
-The user must have at least one unsolved payment.
Flow of events:
-The user presses the “Pay Off Debt button” from the homepage.
Exit conditions: The system performs the transaction, removes the unsolved payments and notifies the user.
Exceptions: If there are no enough money, the system notifies the user and redirects him to the homepage.

Name: Change Payment Information.
Actors: Authenticated user.
Entry conditions: The user must be on his “authenticated user” homepage.
Flow of events:
-The user presses the “Change Payment Information” button.
-The user inserts his new payment information.
Exit conditions: The system replaces the old informations.
Exceptions: If the new payment informations are inconsistent, the system notifies the user and ask him to reinsert them.

Name: Cancel Reservation.
Actors: Authenticated user.
Entry conditions:
-The user must be on his “authenticated user” homepage.
-The user must have a valid reservation.
Flow of events:
-The user presses the “Cancel Reservation button”.
-The system asks the user to confirm his choice.
Exit conditions: The system deletes the reservation, make the car available again in the map page and locks the car. If the user was already in the car, the system waits until he exites and then locks the car
Exceptions: There are no exceptions.

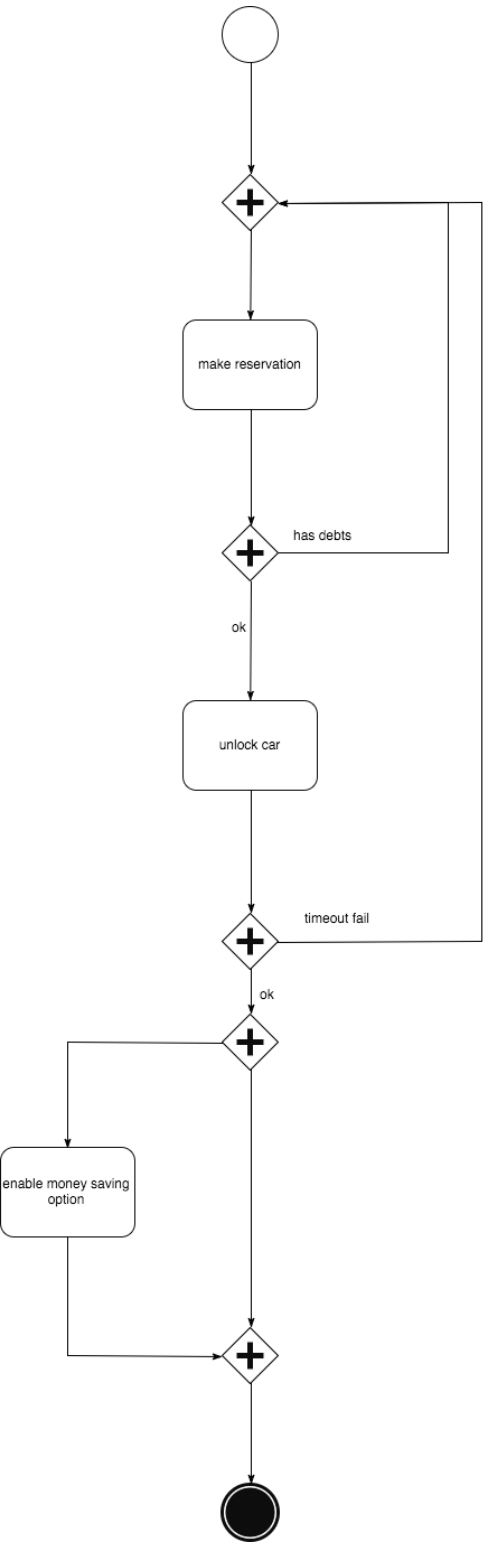
Name: Enable Money Saving Option.
Actors: Authenticated user.
Entry conditions:
-The user must be on his “authenticated user” homepage.
-The user must have unlocked a car.
Flow of events:
-The user presses the “Enable Money Saving Option button”

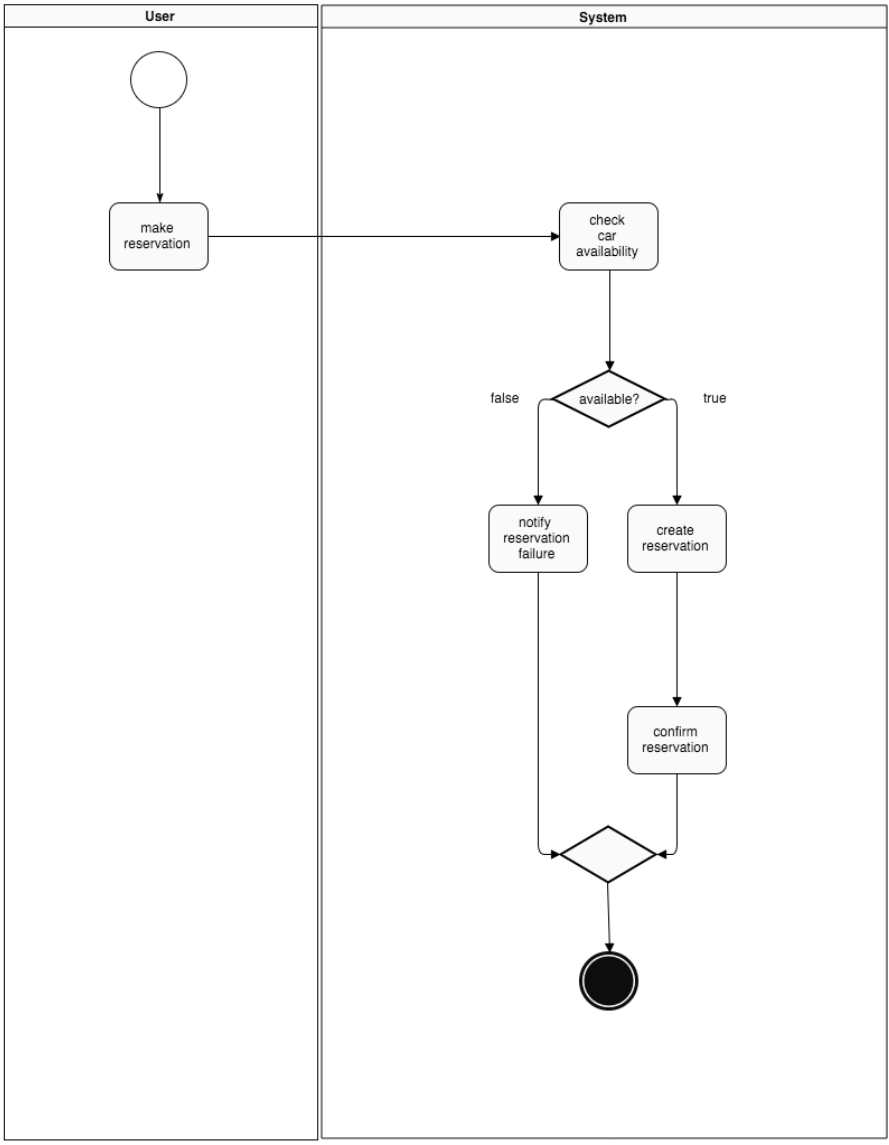
-The user inserts his final destination.
-The system checked the inserted destination.

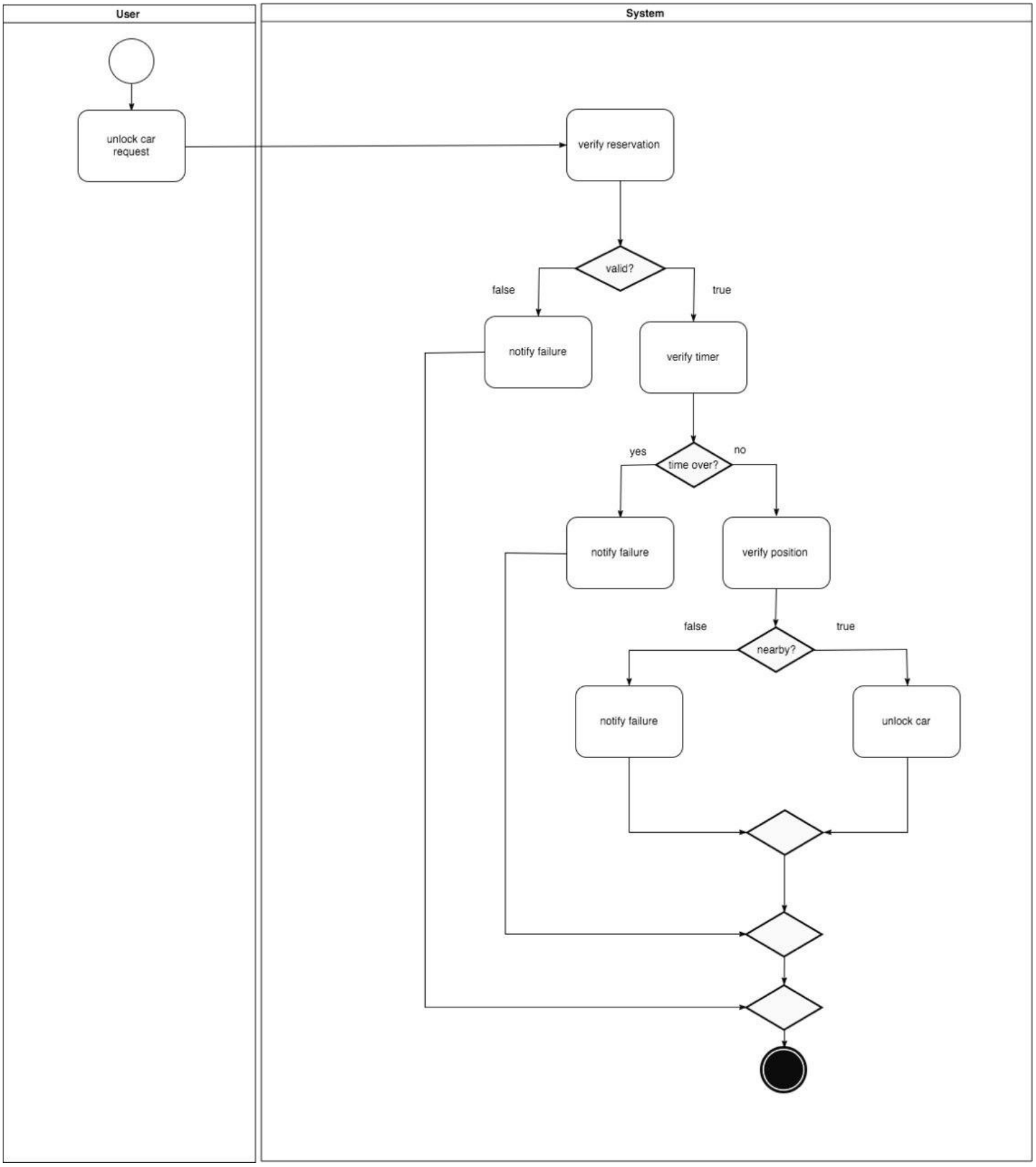
Exit conditions: The system provides information about where to leave the car.

Exceptions: It the destination inserted are not correct, the system notifies the user and ask him to reinsert them.

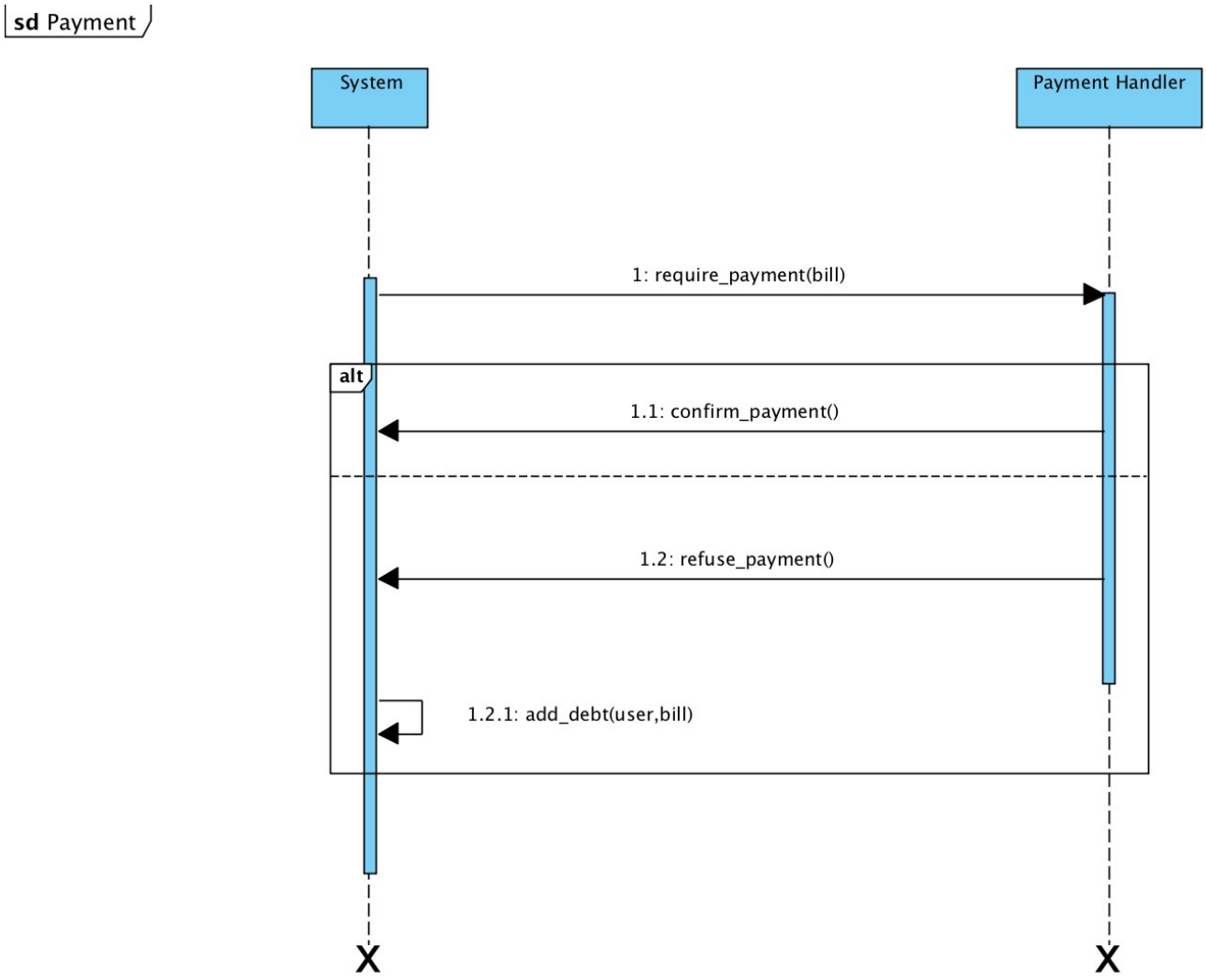
6-Activity diagrams

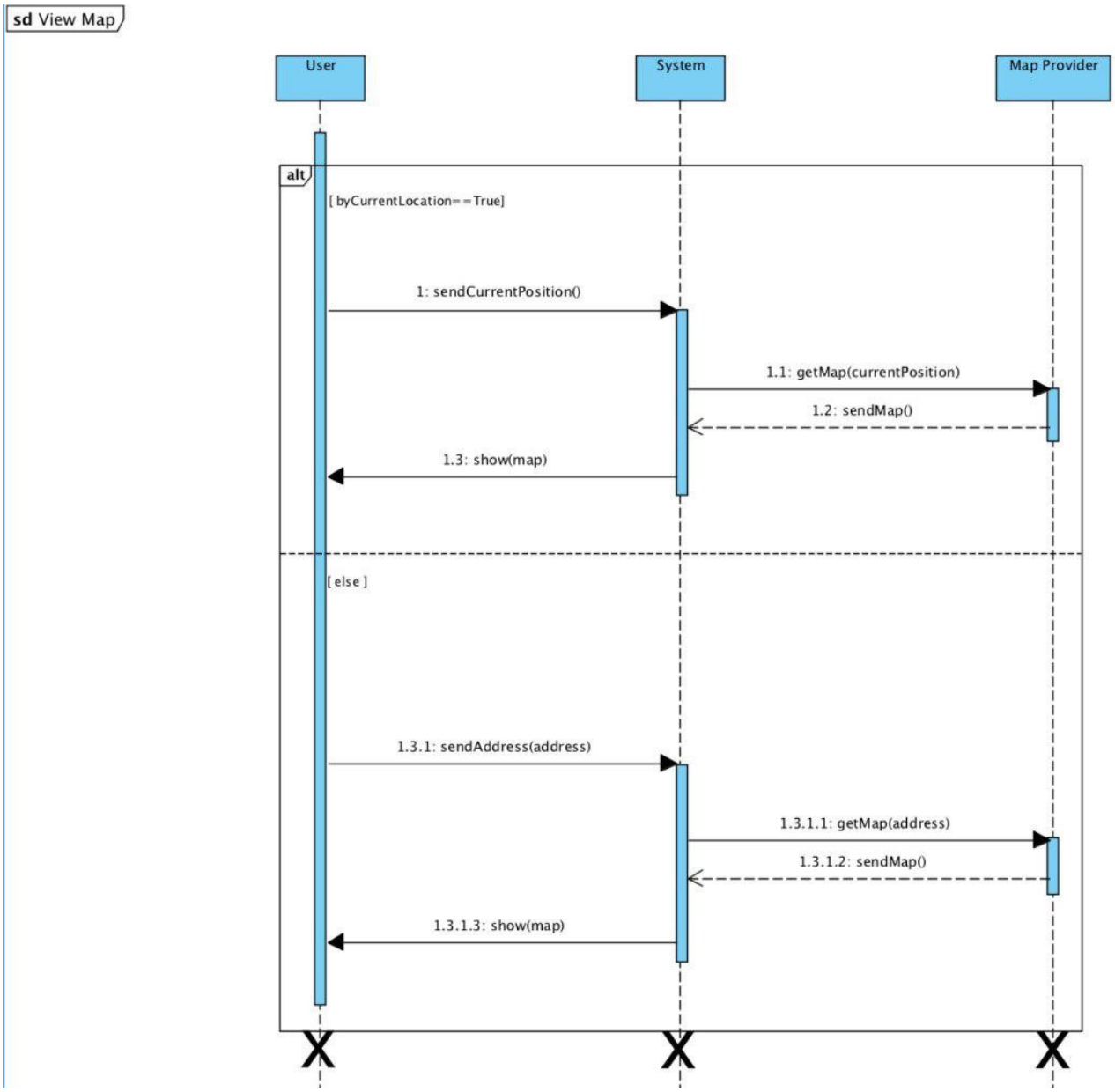

make reservation
has debts
ok
unlock car
timeout fail
ok
enable money saving option


User
System
make reservation
check car availability
available?
false
true
notify reservation failure
create reservation
confirm reservation


User
System
unlock car request
verify reservation
valid?
false
true
notify failure
verify timer
yes
time over?
no
notify failure
verify position
false
nearby?
true
notify failure
unlock car

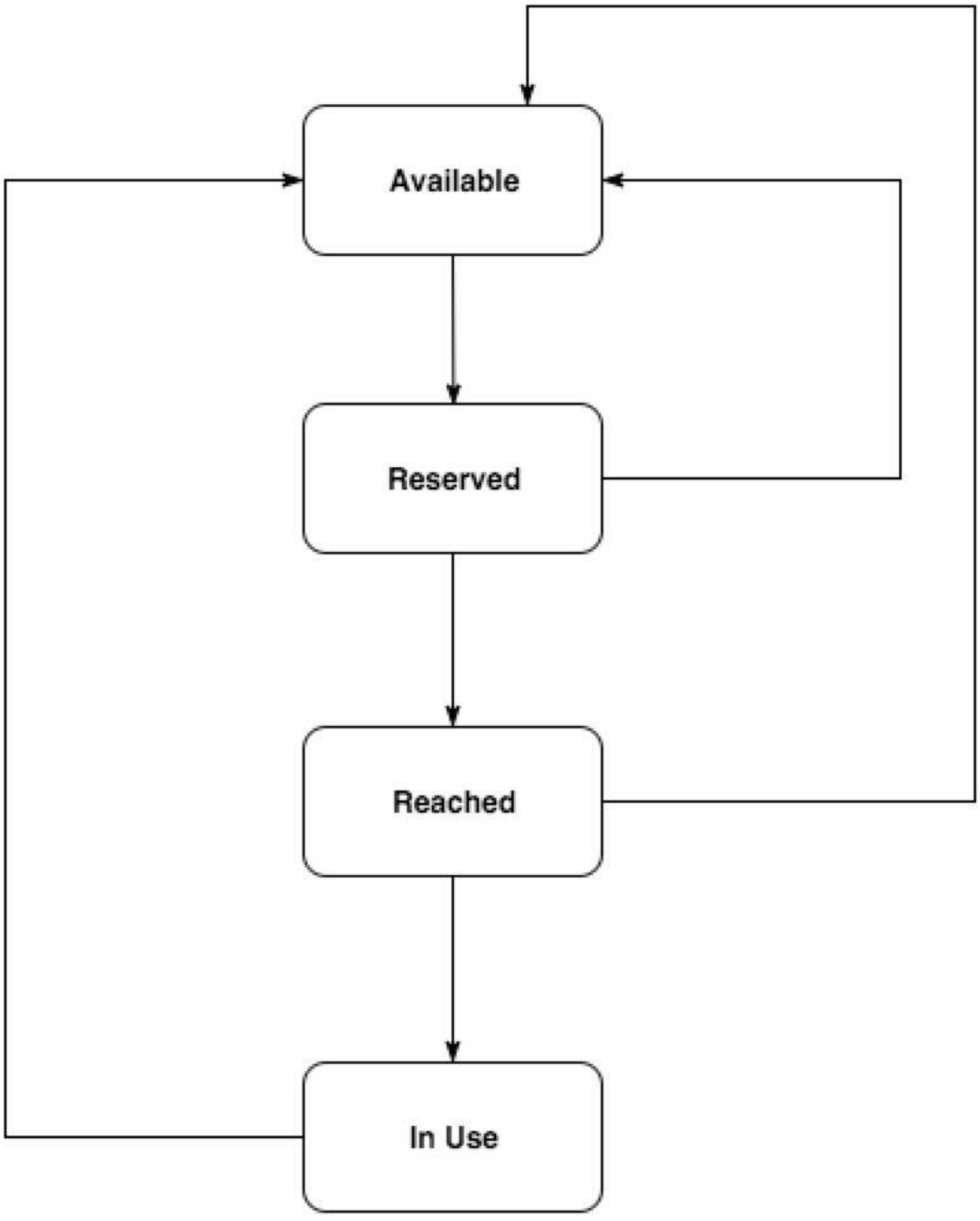
7-Sequence diagrams



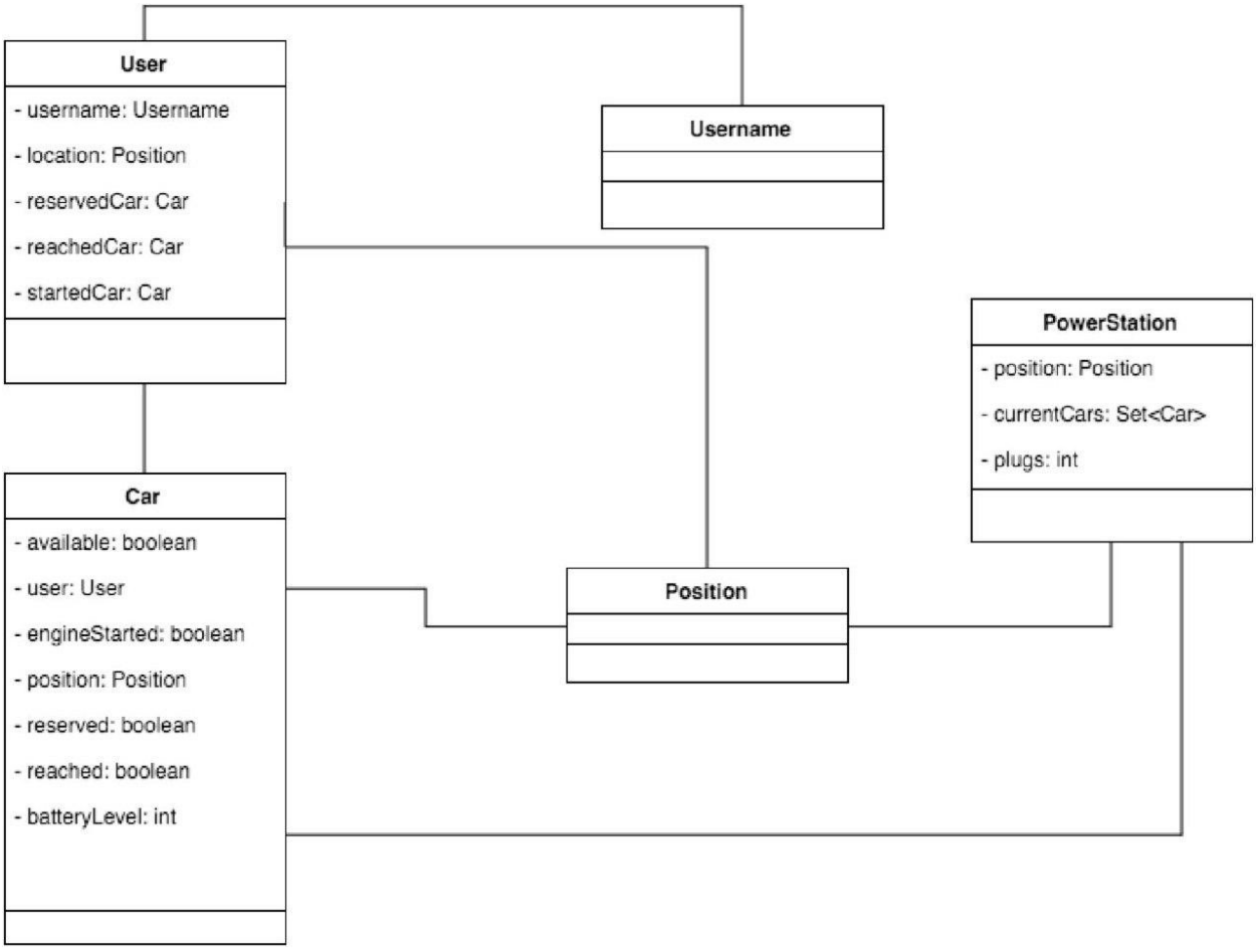

sd View Map
User
System
Map Provider
alt
[byCurrentLocation==True]
1: sendCurrentPosition()
1.1: getMap(currentPosition)
1.2: sendMap()
1.3: show(map)
[else]
1.3.1: sendAddress(address)
1.3.1.1: getMap(address)
1.3.1.2: sendMap()
1.3.1.3: show(map)

8-State diagram

This is the state diagram of our car. There is actually another state, called “unavailable” that can be set only by operators. This state is reachable from any other state since there may be a problem in any moment.



9-Class diagram



10-Alloy

This section shows a way to model some of the requirements of our system. We omitted the unavailable state in the signature of the car since no requirement holds while the car is in this state.

```
open util/boolean
sig Car {
	available: one Bool,
	user: lone User ,
	reserved: one Bool,
	reached: one Bool,
	engineStarted: one Bool,
	position: one Position,
	batteryLevel: one Int } {
	batteryLevel >=0
}

sig Position {}

sig Username {}

sig User {
	username: one Username,
	location: one Position,
	reservedCar: lone Car,
	reachedCar: lone Car,
	startedCar: lone Car
}

sig PowerStation {
	position: one Position,
	currentCars: set Car,
	plugs: one Int }
{plugs>0}



pred usernameAreUnique {
	all disj u1, u2: User | u1! = u2 => u1.username != u2.username
}
```

```
fact carWithOneState {
	all c: Car | c.available=True iff (c.reserved=False and c.reached=False
and c.engineStarted=False)
	all c: Car | c.reserved=True iff (c.available=False and c.reached=False
and c.engineStarted=False)
	all c: Car | c.reached=True iff (c.available=False and c.reserved=False
and c.engineStarted=False)
	all c: Car | c.engineStarted=True iff (c.available=False and
c.reserved=False and c.reached=False)
	all c: Car | c.available=True or c.reserved=True or c.reached=True or
c.engineStarted=True
}


fact noCarForMultipleUsers {
	all u1, u2: User, c: Car | u1.reservedCar=c and u2.reservedCar=c =>
u1=u2
	all u1, u2: User, c: Car | u1.reachedCar=c and u2.reachedCar=c  =>
u1=u2
	all u1, u2: User, c: Car | u1.startedCar=c and u2.startedCar=c => u1=u2
	no disj c1, c2: Car | c1.user = c2.user and #c1.user=1
}


fact coherenceBetweenCarStatusAndUsers {
	all c: Car | c.available=True <=> #c.user=0

	all u: User, c: Car | (u.reservedCar = c)<=> c.user=u and
	c.reserved=True
	all u: User, c: Car | (u.reachedCar = c) <=> c.user=u and
	c.reached=True
	all u: User, c: Car | (u.startedCar = c) <=> c.user=u and
	c.engineStarted=True
}



fact noMoreCarsThanPlugs {
	all p: PowerStation | #p.currentCars <= p.plugs
}

fact allCarsPluggedInPowerStationAreNotMoving {
	no p: PowerStation, c: Car | c in p.currentCars and c.engineStarted=True
}

fact noSharedCarsBetweenStations{
	all disj p1, p2: PowerStation | disj [p1.currentCars, p2.currentCars]
}

fact StationsHaveDifferentPositions {
	all disj p1, p2: PowerStation | p1.position != p2.position
```

```
}

fact CarsInStationHaveTheSamePosition {
	all c: Car, p: PowerStation | (c in p.currentCars) => c.position = p.position
}

pred makeReservation(u, u': User, c, c': Car) {
	c.available=True
	c'.reserved=True
	u'.reservedCar= c'
	c.position=c'.position
	#u.reservedCar=0
	#u.reachedCar=0
	#u.startedCar=0
	u'.username=u.username
	u'.location=u.location
}

pred deleteReservation(u, u':User, c, c': Car) {
	c.reserved=True or c.reached =True
	c'.available=True
	c.position=c'.position
	u.reservedCar=c or u.reachedCar=c
	#u'.reservedCar=0
	#u'.reachedCar=0
	#u'.startedCar=0
	u'.username=u.username
	u'.location=u.location
}




assert userCanCancelAReservationBeforeTheEngineIgnition {
	no u, u': User, c, c': Car | deleteReservation[u, u', c, c'] and
	c.engineStarted=True
}

assert onlyAvailableCarsCanBeReserved {
	no u, u': User, c, c':Car | makeReservation[u, u', c, c'] and
	c.available=False
}

assert usersCanReserveASingleCar {
	no disj u1, u3: User, disj u2, u4:User, c, c': Car | makeReservation[u1, u2,
	c, c'] and makeReservation[u3, u4, c, c']
}


pred showGeneralScenario(){
	usernameAreUnique
```

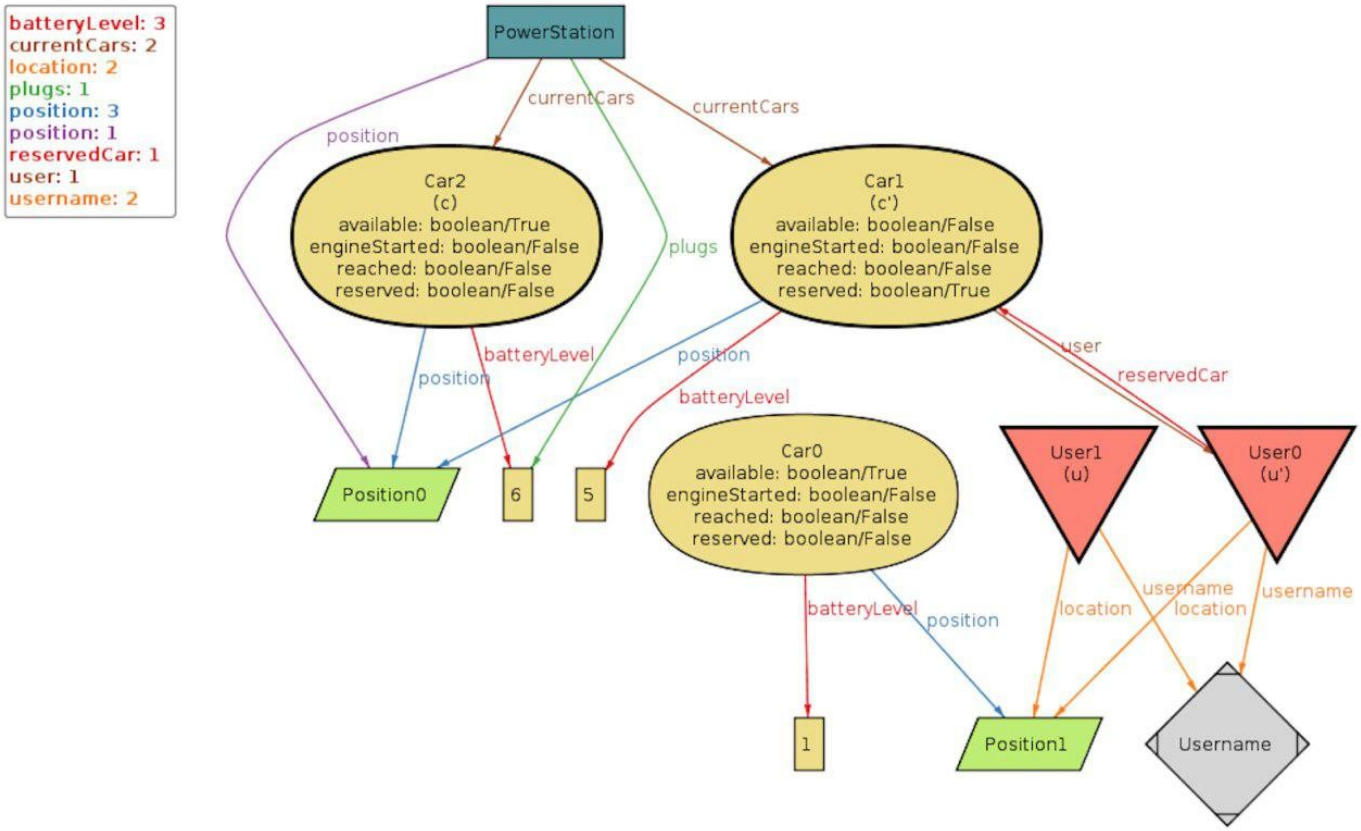
}

check onlyAvailableCarsCanBeReserved for 5
check usersCanReserveASingleCar for 5
check userCanCancelAReservationBeforeTheEngineIgnition for 5

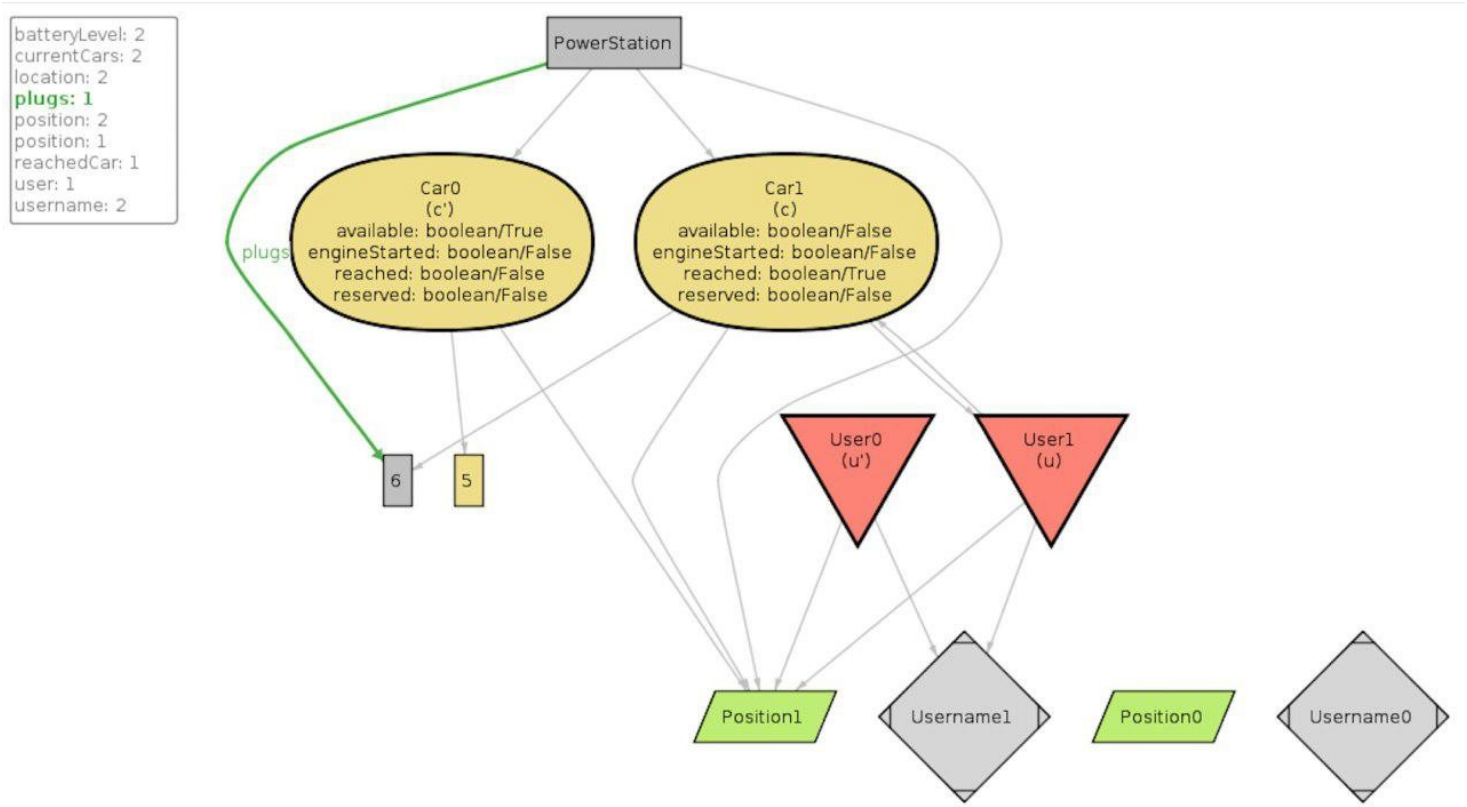
run makeReservation for 2 User, 2 Position, 3 PowerStation, 2 Username, 3 Car
run deleteReservation for 2 User, 2 Position, 3 PowerStation, 2 Username, 3 Car

run showGeneralScenario for 4

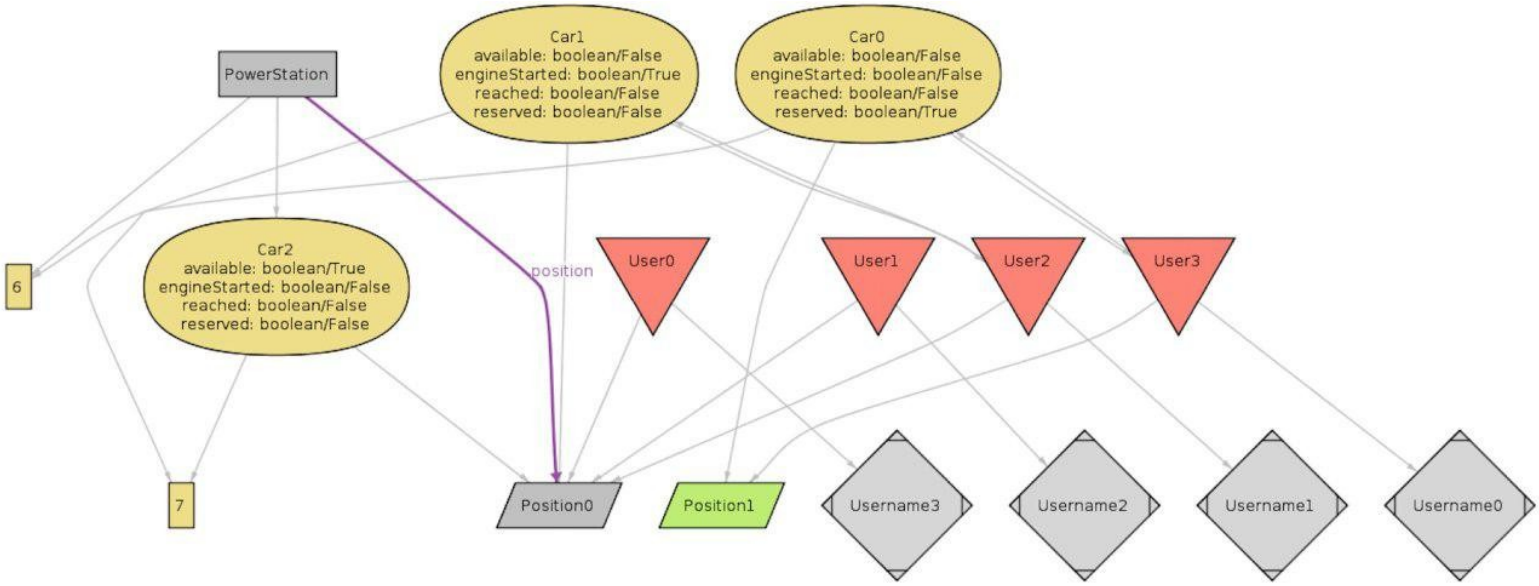
run makeReservation for 2 User, 2 Position, 3 PowerStation, 2 Username, 3 Car



run deleteReservation for 2 User, 2 Position, 3 PowerStation, 2 Username, 3 Car



run showGeneralScenario for 4



11-Other

Hours of work

Andrea Pace

26/10 1h
28/10 1h
31/10 2h
1/11 1h30
4/11 3h
6/11 1h
9/11 2h
10/11 2h
11/11 3h
12/11 9h
13/11 9h

Lorenzo Petrangeli

21/10 5h
24/10 1h
25/10 1h
27/10 2h
28/10 4h
4/11 2h
5/11 3h
6/11 3h
9/11 1h
10/11 1h
12/11 2h
13/11 3h

Tommaso Paulon

17/10 4h
18/10 2h
21/10 1h
23/10 3h
24/10 1h

25/10 1h
26/10 1h
29/10 2h
31/10 1h
1/11 4h
6/11 1h
8/11 2h
10/11 3h
11/11 1h

Changelog

1.1 Add goal [G15]
1.2 Add picture of The World And The Machine, specify unavailable state in state diagram and alloy