Design Document

Authors: Falci Angelo 875123
Lanzuise Valentina 807364
Lazzaretti Simone 875326

Reference Professor: Di Nitto Elisabetta

SUMMARY

1 Introduction

1.1 Purpose
In the first document we have explained in details the purpose of our analysis, fixing on the requirements that our application should have . Here a short summary on the pivotal points of the last document.

We will develop PowerEnJoy, a digital management system for car-sharing service of electric cars.

In our system we manage the service offered to user, whose main functionalities are:
- to sign up to the system and modify personal information
- to book a car
- to rent a car
- to contact assistance
- to use car options on the device inside the car, such as choosing the destination, viewing safe areas and power grids, and using the park function.

In this document instead we are going to deepen what we have already described creating a design document: we will describe better the different elements that are included in our system and the interaction between them, especially using UML diagrams, in order to identify the architectural structure as a whole, but also as single components.

1.2 Scope
The project PowerEnJoy is addressed to any kind of people (that have reached the majority age and with a valid driving license) who want a cheap, comfortable and clean-energy car service. Clients must be registered on the system, and need any device with Internet connection in order to use the service. Our system uses the GPS on the device for localizing users, and provides them the position and the information of the cars nearby. This project is principally thought for a urban area, even if it is possible for users to drive away from it and stop the car there momentarily. Users can leave and take the cars only situated in the safe area, which are disposed in the urban area.

The aim of the system is to provide an efficient and simple service that manages the renting of the cars and supplies convenient discounts in order to improve users to use it.

1.3 Definitions, Acronyms, Abbreviations
Here we write the main terms we will use in the project with their meaning.
- GUEST: identify the person not registered yet.
- REGISTERED USER: identify the person registered who can use the service.
- USER: identify the generic person who is using the service.
- SAFE AREA: identify the area where the user can leave a car to have a
- discount.
- POWER GRID: identify the power station that a safe area can have where the

user can recharge the car to obtain a discount.
- SYSTEM: identify server and database that manage the web-application service, and the software that manages the car.
- CAR: identify every single car provided by PowerEnJoy.
- POSITION: indicate the specific position about car, safe area and power grid using latitude and longitude.
- ASSISTANCE: identify service who user can call if it has a problem with the car.
- ASSISTANT: identify both the operator that manages the maintenance of the cars and the telephone operator who helps clients.
- HOMEPAGE: indicate the page in which are addressed either guests and registered users before signing up.
- USER HOME: identify the page in which users are addressed after login where they have access to the services.
- PERSONAL USER DEVICE: identify a generic device used by user with an Internet connection.
- MSO: identify the "Money Saving Option" function.
- VoIP: is a acronym to identify Voice over Internet Protocol is a methodology and group of technologies for the delivery of voice communications over Internet Protocol (IP) networks, such as the Internet.
- Work Station: identify the computers in the office of PowerEnJoy with which the assistants can communicate with the server.
- Model: identify the abstraction of the data of the DBMS.
- SA: "Safe Area" abbreviation.
- PG: "Power Grid" abbreviation.
- MVC: "Model View Controller" abbreviation.

1.4 Document Structure
This document is principally divided in four parts:
- In the first part we have analyzed the architectural design, in particular the high level description of the components, the software components view in the specific and how they interact.
- In the second part we have given some example of the crucial parts of our algorithm design.
- In the third part we have deepened user interfaces contained in the RASD document, developing some new functionalities and improving the graphics, trough mockups and UX diagrams.
- In the fourth part we have written our conclusions and explained which and how the requirements contained in RASD document traced the realization of our design diagrams.

2 Architecture Design

2.1 Overview
We start to analyze our project giving a panoramic of the entire system.
The picture below shows the high level components and the way they interact.
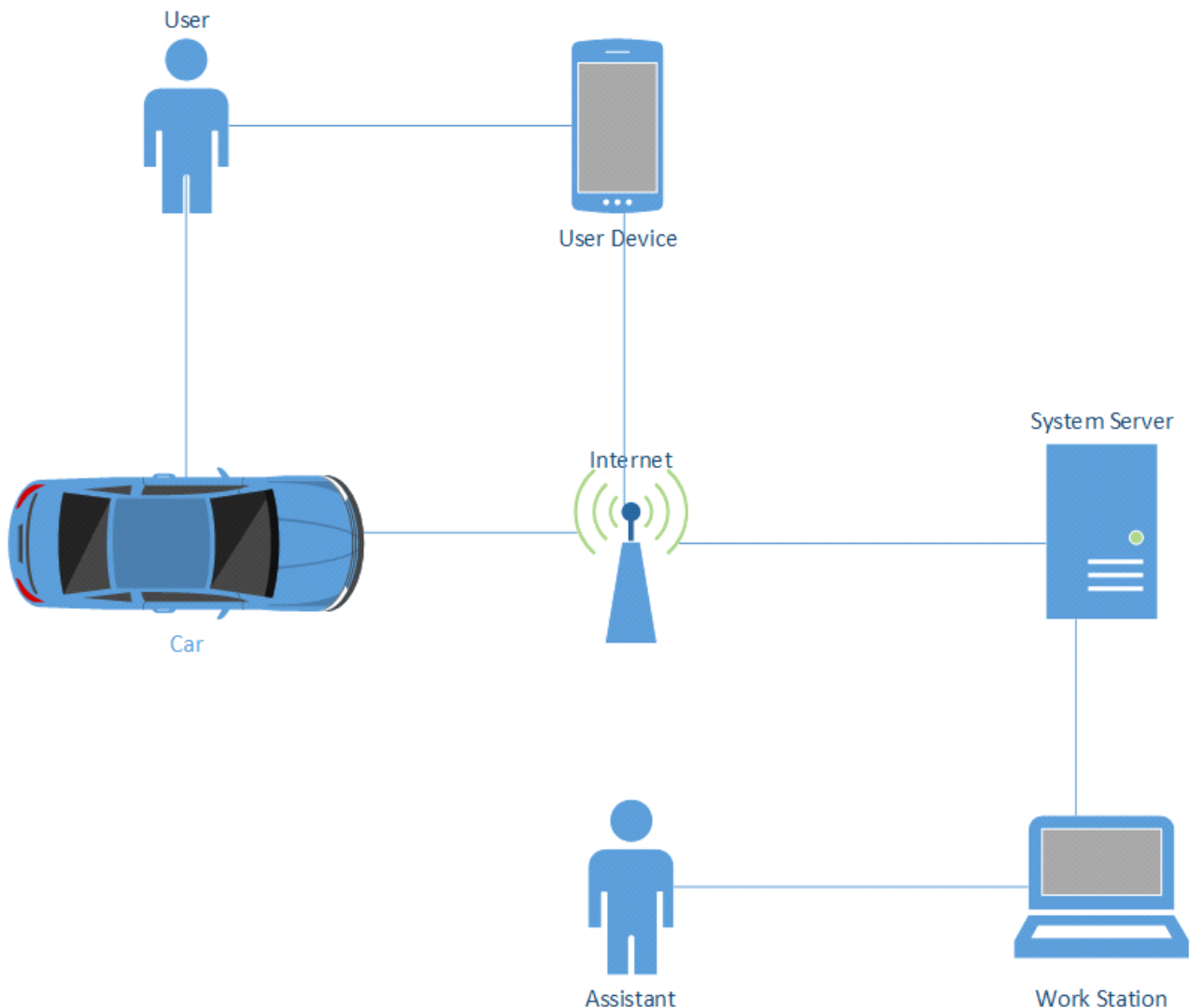
User can interact with the system in two ways:

· Through a "personal user device", equipped with Internet connection, user can register itself at the beginning and then he/she has access to the functions of the service. He/she can see the map with cars available and safe areas, he/she can rent or book a car and modify its personal information.

· Through the "car device" user can turn on the car, view the map (with safe areas or only safe areas that have power grids), insert the destination, view the current change and it is able to use two advanced functionalities, MSO (to find the best safe area near destination in order to obtain the maximum discount) and Park Function (to leave the car temporarily without stopping the renting).

Using Internet, server can communicate with:

· User device in order to satisfy user requests and to send notifications about the

right execution of the operations or of payment occurred.

· Car in order to lock/unlock the door, to lead user to destination, to show safe areas (or only safe areas with power grid) around the car position and to provide advanced functionalities, MSO and Park Function.

Moreover when user need the assistance service the server puts user in touch with the first assistant available using a VoIP.

2.2 Component View

User Client
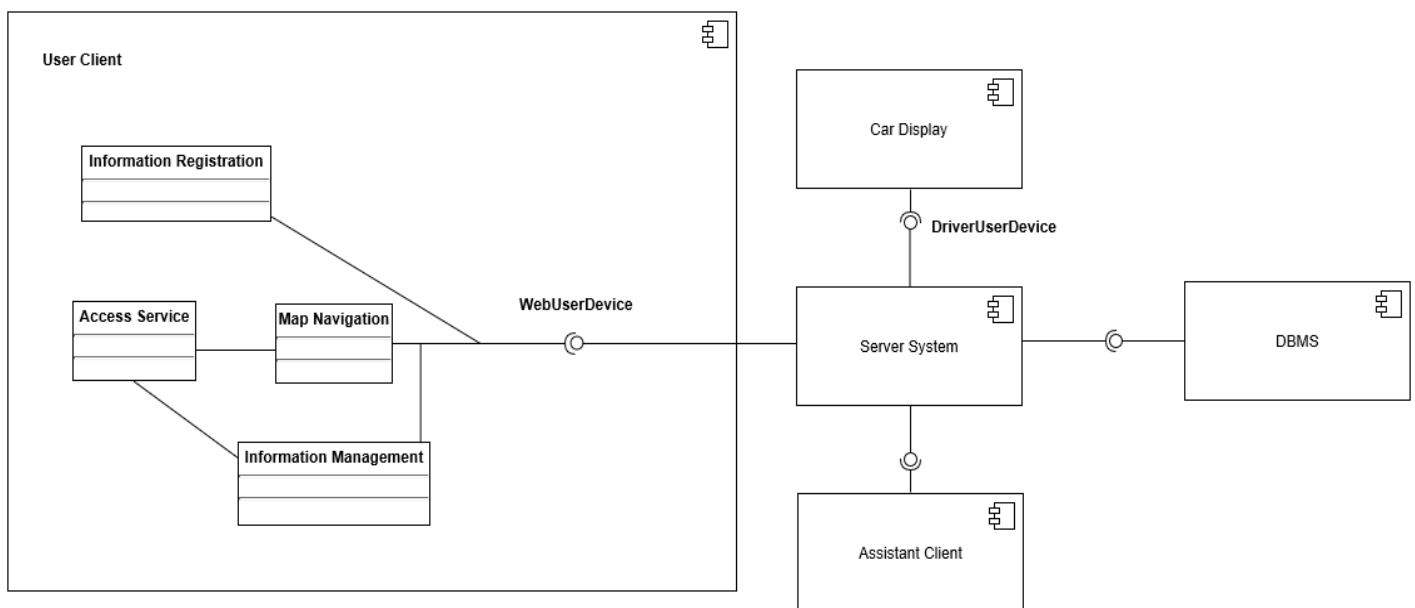
In this graph it is shown the component structure of the client for users, both for PC browser and mobile browser. Here the explanation of the components:

1)Information Registration: this component provides to send the request for the registration to the service, after filling the information fields in the Registration Page

2)Access Service: this component provides to send the request for the access to the service, after filling the information fields in the Home Page

3)Map Navigation: this component shows the map to users, sends all the requests about car-sharing operations (renting, booking, searching address)

4)Information Management: this component shows personal information to users and the list of booking and notifications
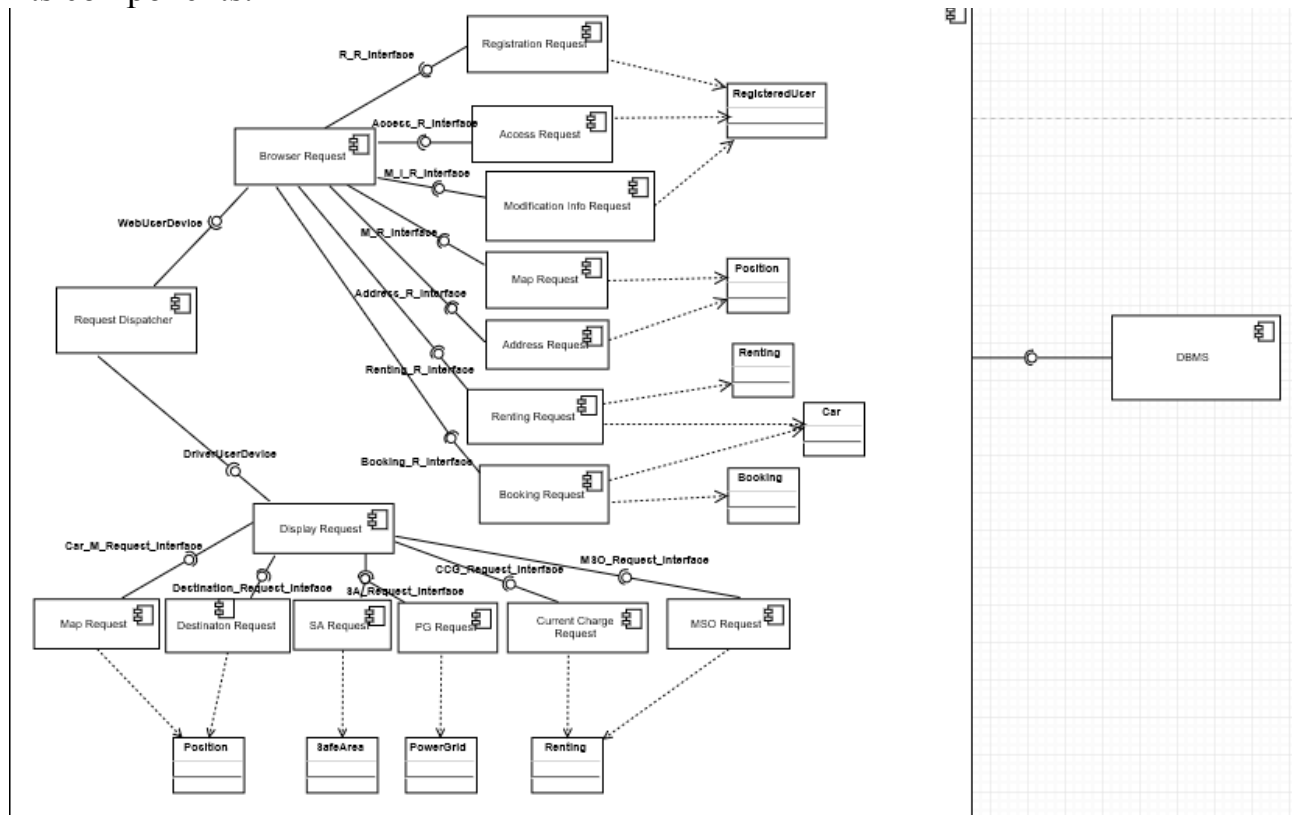
Car Client
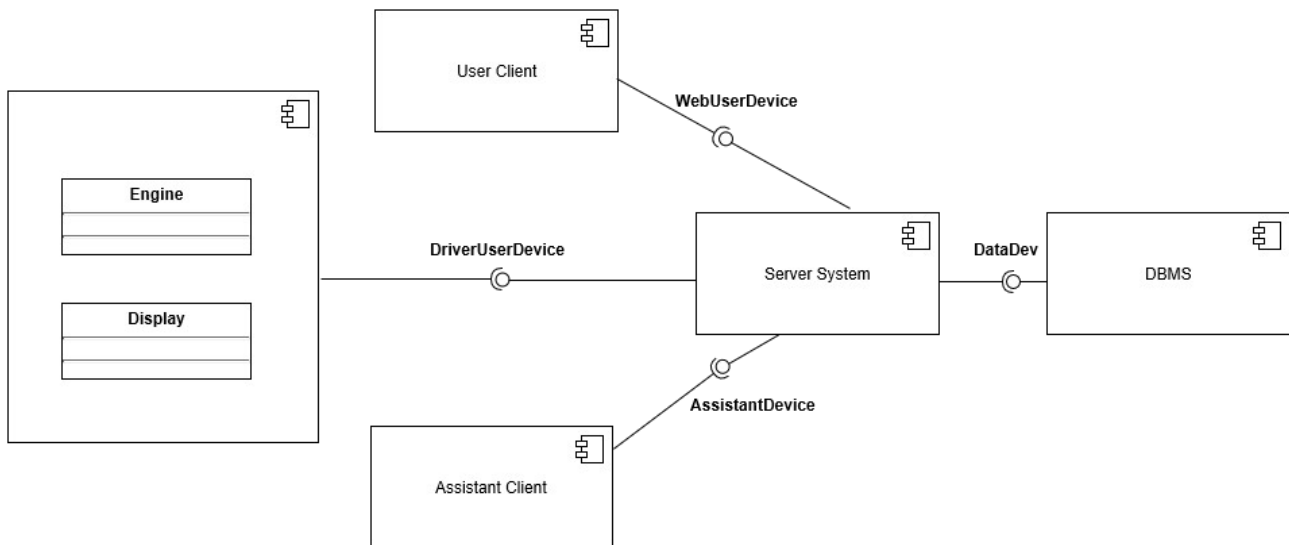
Here it is shown the component structure of car client. Below the description of its components:

1)Engine: this component shows to the user the level of the battery of the car

2)Display: this component has all the main tasks provided by the system to the user, from setting an address as a destination, to activate money-saving-option service

Server System

Here it is shown the component structure of the server logic. Below the description of its components:

1)Request Dispatcher: forwards request received by different users

2)Browser Request: forwards request by Request Dispatcher sub-component originally sent by User Client component

3)Display Request: forwards request by Request Dispatcher sub-component originally sent by Car Client component

4)Map Request: provides service about showing map both on the browser and the car display

5)Registration Request: provides service about sign up

6)Address Request: provides service about research of addresses

7)Access Request: provides service about login

8)Booking Request: provides service about booking

9)Renting Request:provides service about renting

10)Modification Info Request: provides service about modification of personal information (payment method, e-mail address, password, etc)

11)Destination Request: provides service about tracing the itinerary from initial point to the final point one in the map
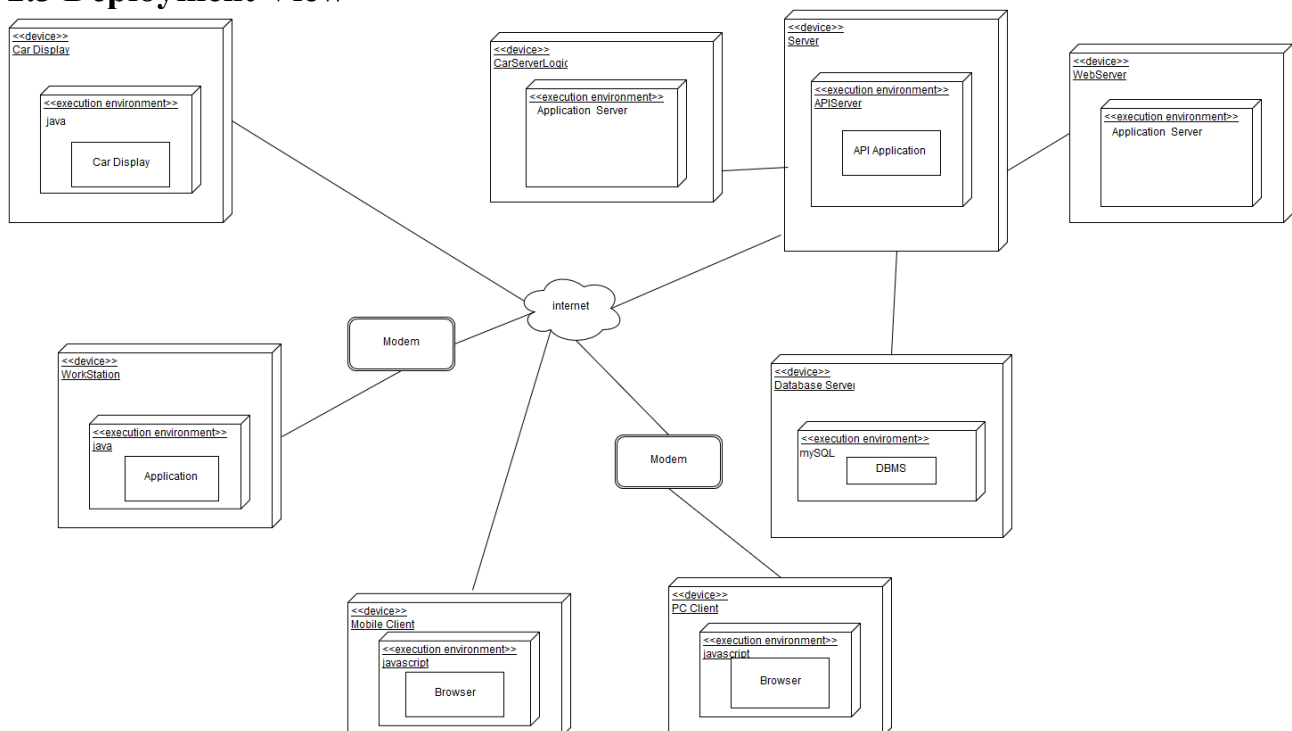
12)SA Request: provides service about displaying safe areas on the display of the car

13)PG Request: provides service about displaying safe area with power grids on the display of the car

14)Current Charge Request: provides service about payment

15)Model: abstraction of the data contained in the DBMS

2.3 Deployment View

This picture shows the deployment view of the software components. We identify the principal division of our Server: Car Server, which includes the logic of application of the car device, Web Server that answers to the requirements of the web pages of

the clients and Database Server that manages the data of the clients and the web database. We indicate also the main device that interact with the system through an Internet connection: the device inside the cars, the users device (PC and mobile) and the workstation of the assistants.
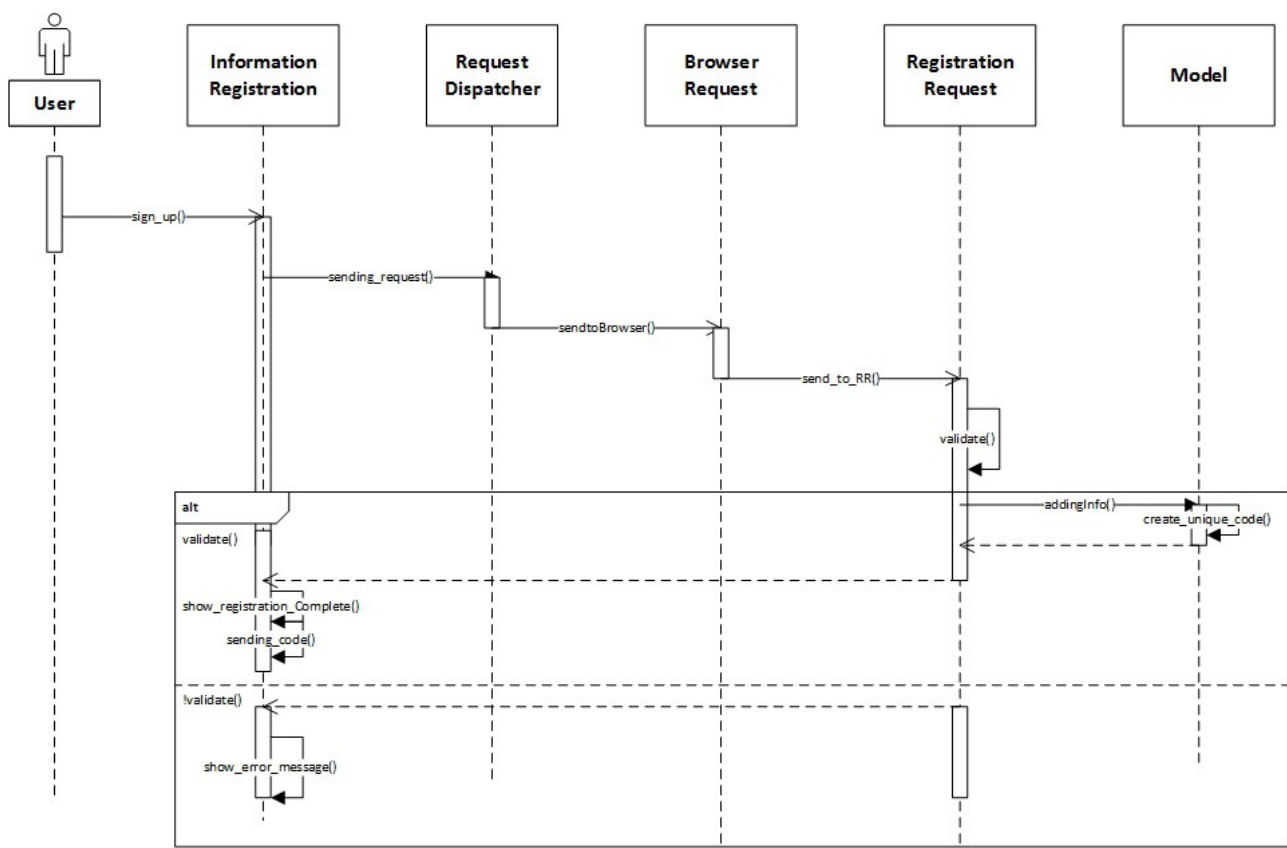
2.4 Run-time View

In these graphs we will see how the system manages the requests from the user with different devices (personal and car device).

We don't show how the system manages all the requests but we focus on some of them and how the information is passed between the different modules.

We derive them from "component view" section.

Sign Up

In the above sequence diagram, user starts the process of registration, filling the forms in the web page. These information are sent to the "Information Registration" component, that saves the information into local variables and sends the copies to the "Dispatcher Request" component with the function send_request(), whose task is to forward the request to the "Browser request" component. This forwards another time the request to the "Registration request" component. Here, information are controlled (if the name exists, the birth date is admissible, etc.) with the function validate(). If the return of this function is true, than the component adds the information to the

database, creates a unique code and returns to the "Information Registration" component with the created code. There the system shows to the user a message saying that the operation has been completed and sending to the user's email a message with the generated code. Otherwise, the information is not added and to user is visualized a message of error.

Log In
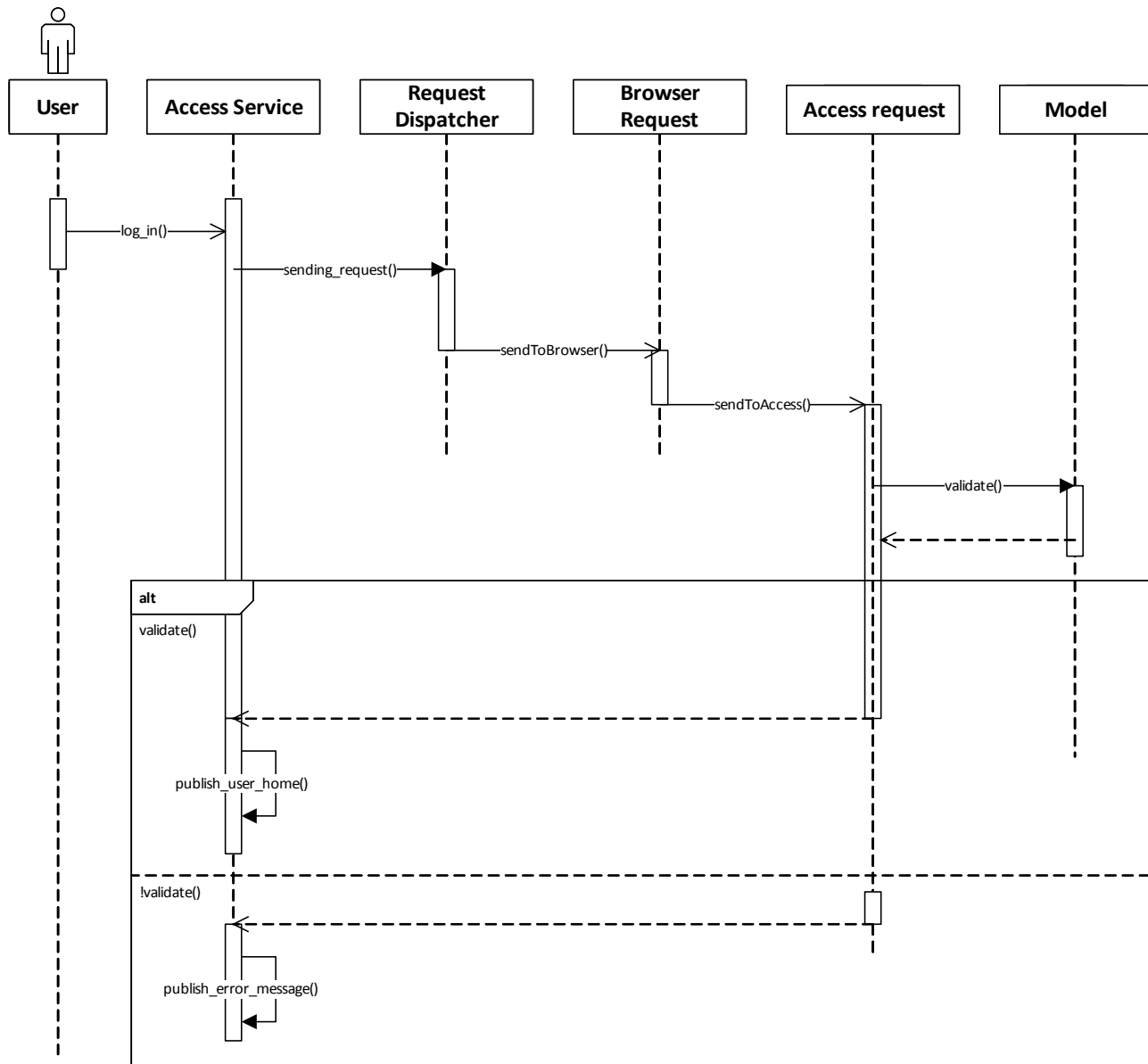
In the above sequence diagram, it's represented the "log in" procedure. The user, after inserting access data, presses on the button "LOG IN", represented by the method log_in(). The the sub-component Access Service start the request forwarding it with the method sending_request() to Request Dispatcher sub-component. After other two forwards, when the request arrives to Access Request sub-component, the data are controlled with the validate() method. If this method returned "true", then user is sent

to User Home . Otherwise, an error message appears on screen, saying that the data combination is incorrect.

Show Map
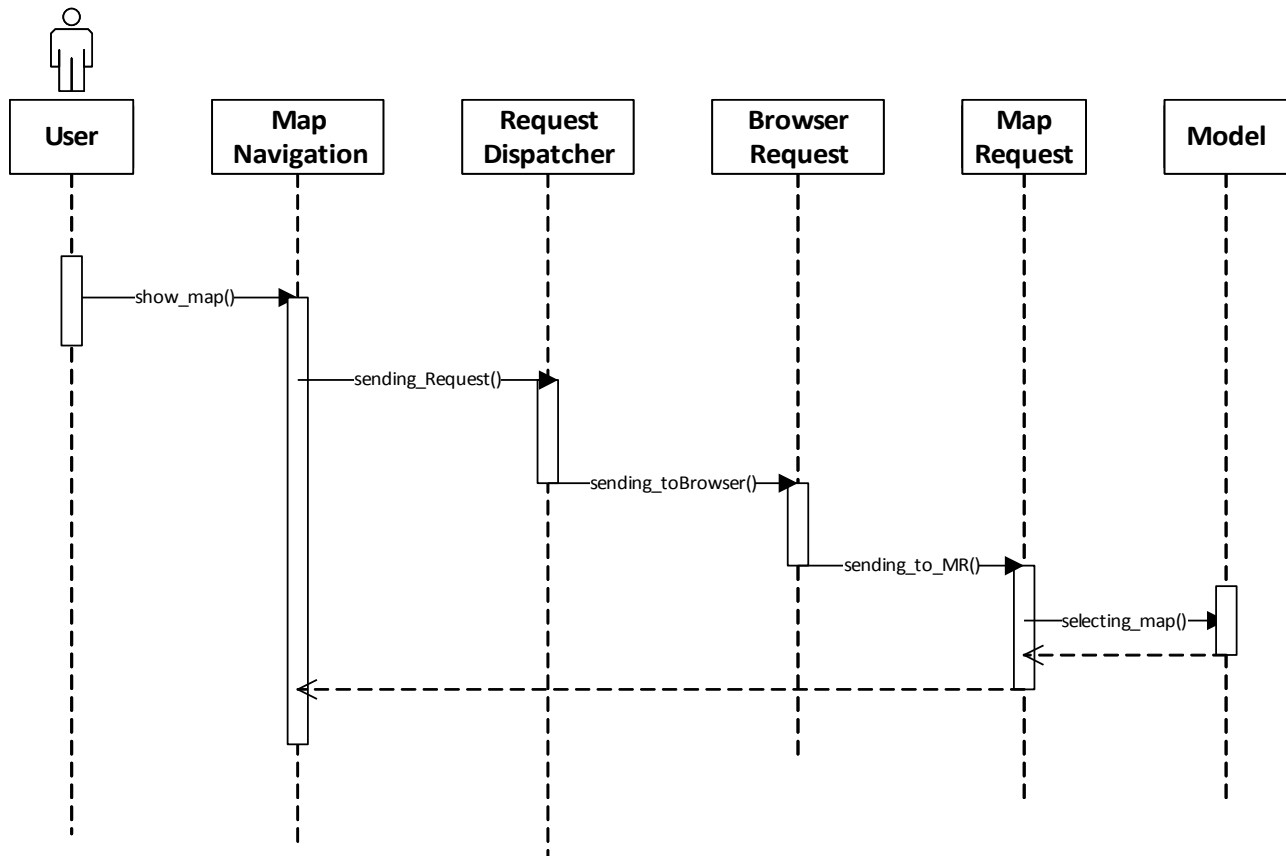
In the above diagram, it is represented the "show map" procedure for the browser user (anyway, the procedure done by user through car display is not so different). The user starts the procedure pressing on the button "Show Map", represented by the show_map() method. Here the sub-component Map Navigation forwards the request till the Map_Request sub component. This sub_component search the data containing map parameters asking it to the Model sub-component (this is done with the selecting_map() method). After the return of this method and the following return to Map Navigation sub-component (with map data), the map is published on the screen user.

Search an Address
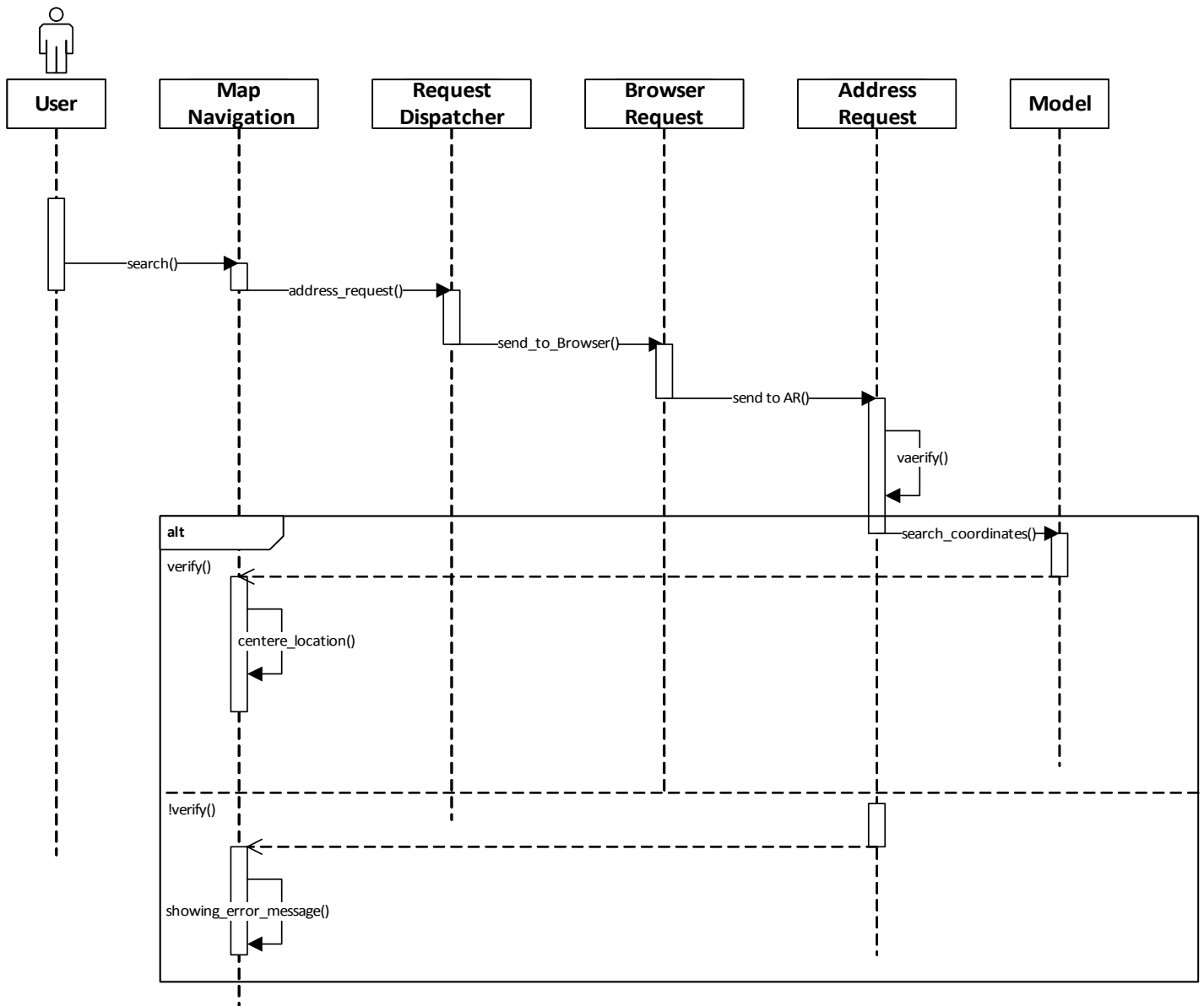
In the above diagram, it is represented the "search an address" procedure. After inserting the desired address, the user presses on the lens icon, represented by the search() method. Here the Map Navigation sub-component forwards the request till the Address Request sub component. This sub-component verifies that the form of the address is correct and starts searching it in the data, with the method verify(). If it returns "true", then the Address Request asks to the Model sub-component the coordinates with the method search_coordinates(). The coordinates are returned at the end to the Map Navigation sub-component and this one continues centering the map

in the address searched. Otherwise, the request shows an error message, leaving unchanged the status of the map.

Renting a Car
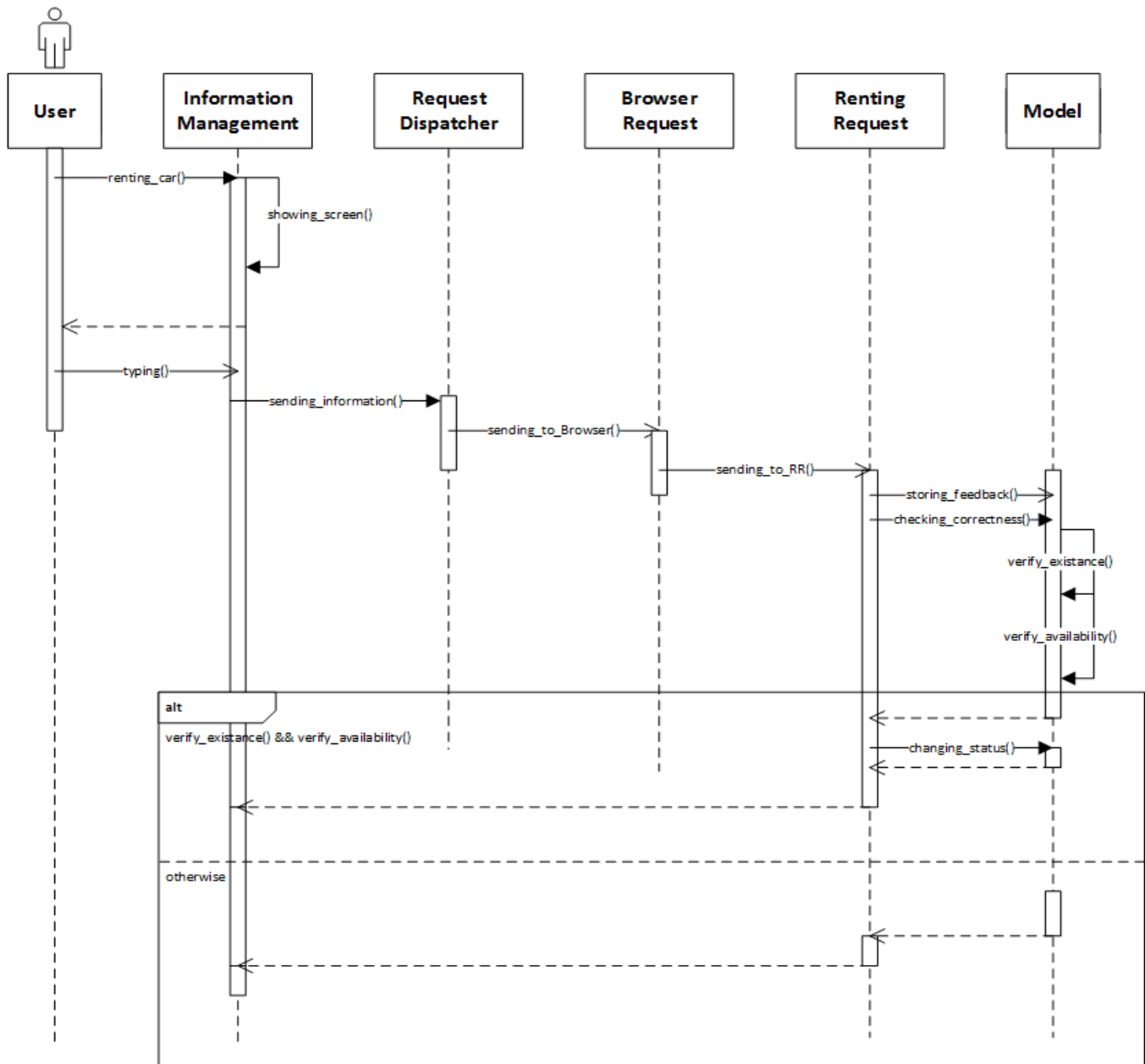
In the above diagram, it is represented the "renting" procedure. The starting situation sees the user pressing "Rent It" button from the page of personal booked car. This action is represented by the method renting_car(). After this, the component Information Management asks to the user to rate the status of the car and to insert the car plate and a code situated in front of the car, loading the dedicated screen. To send the information requested, the user has to insert them in the dedicated blank field and then to press button "UNLOCK"(represented by the function typing()). Doing this,

the Information Management starts the request with the method sending_information(), forwarding it till the Renting Request component. Here, the feedback is stored inside the database, while the plate and the code sent are controlled, sending them to the Model component with the method checking_correctness() and controlling them individually with the method verify_existence(). After this, the component controls if the car is available with the method verify_availability(). If this two last methods return "true", then the Renting Request sub-component asks to the Model sub-component to change the status of the car from "free" to "locked", opening the car for user's usage. If one of this two last methods return "false", the car is not rented and remained close.

Park Function
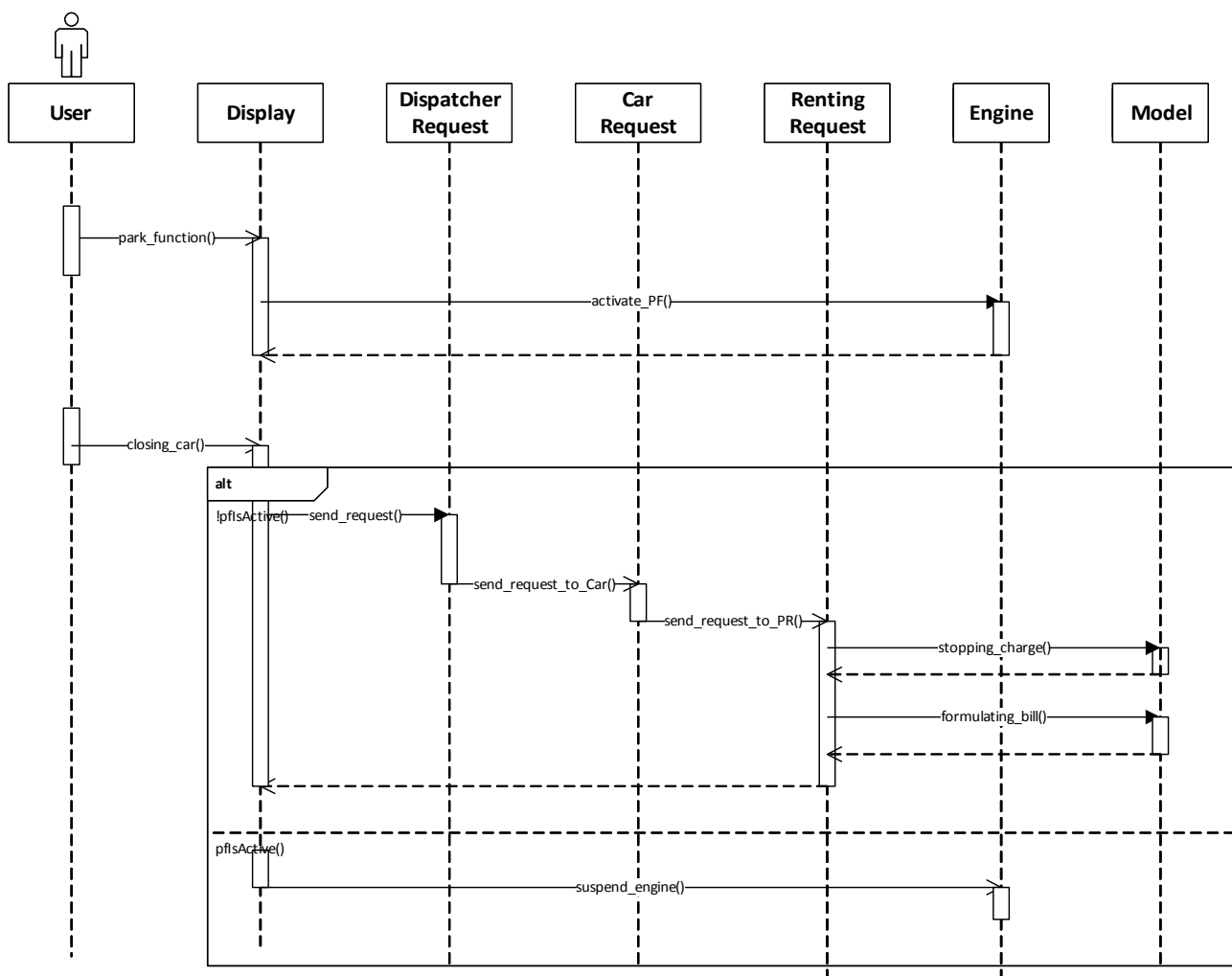
In the above diagram, it is represented the "park function" procedure. Here user presses on the button "Park Function", asking to leave the car without ending the service. The button is represented by the method park_function(). The Display component forwards the request till the Park Request component, that asks to the

Engine component to activate the function in the car and to allow user to remove the car-key. Taking the key, the user is able to re-open the car with it. When user closes the car, the device controls if the service terminates or not. Seeing that the Park Function is activated, the service continues. If the Park Function is not activated, user, closing the car, ends the service, letting the system starts the payment procedure, with a notification (sent to user) with the amount payed

Visualize Current Charging
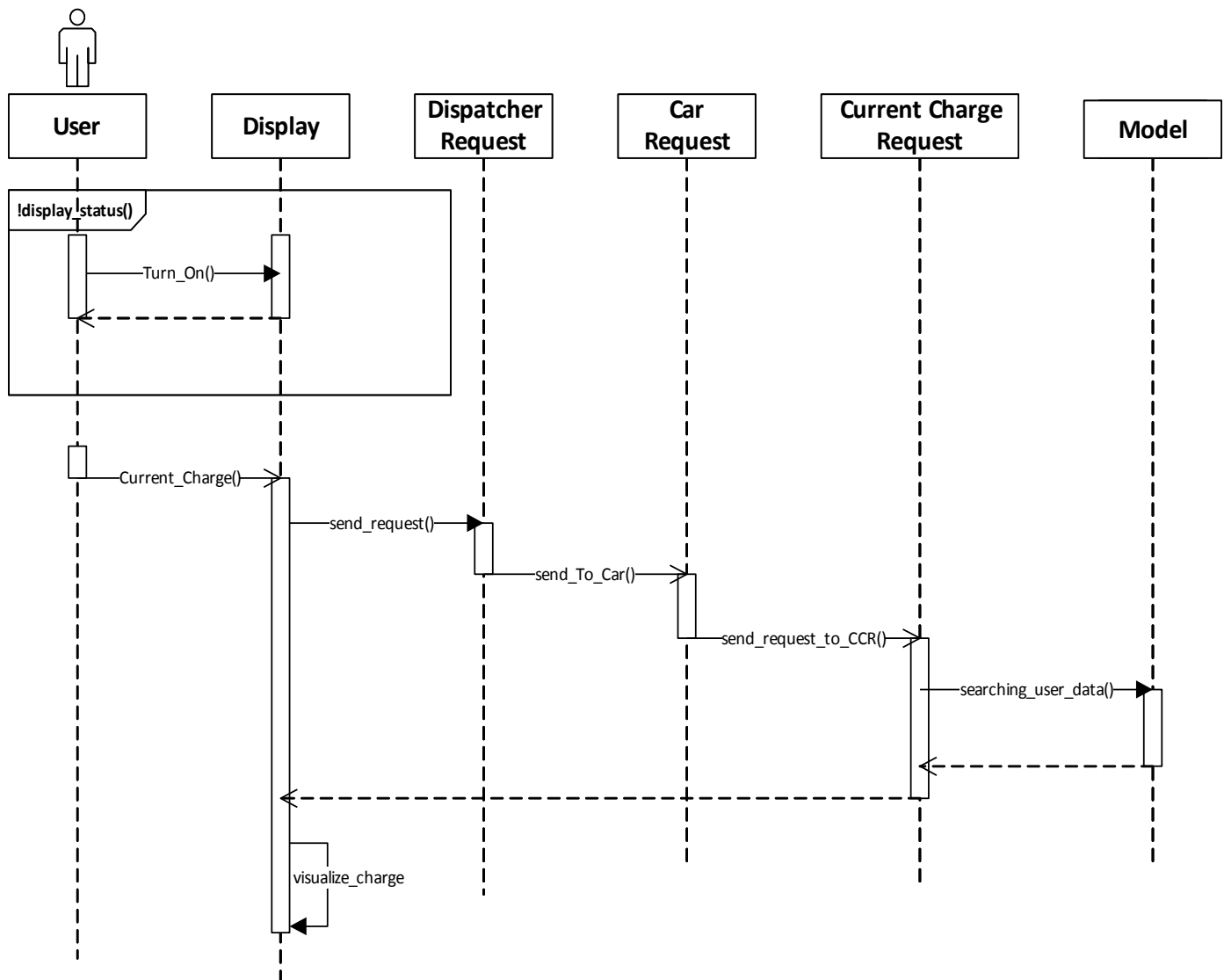
In the above diagram, it is represented the "visualize current charging" procedure. If the display is not turned on, then user must do it. The method Turn_on() expresses this action. After doing this, user presses the button "Current Charge", whose action is expressed by the method Current_Charge(). The Display component forwards the request till the Current Charge Request component. Here the component asks to the

Model one the user data. This action is represented with the method searching_user_data(). It returns the data searched and, when the original request returns, Display component shows the current charge with the method visualize_charge().

View Safe Area
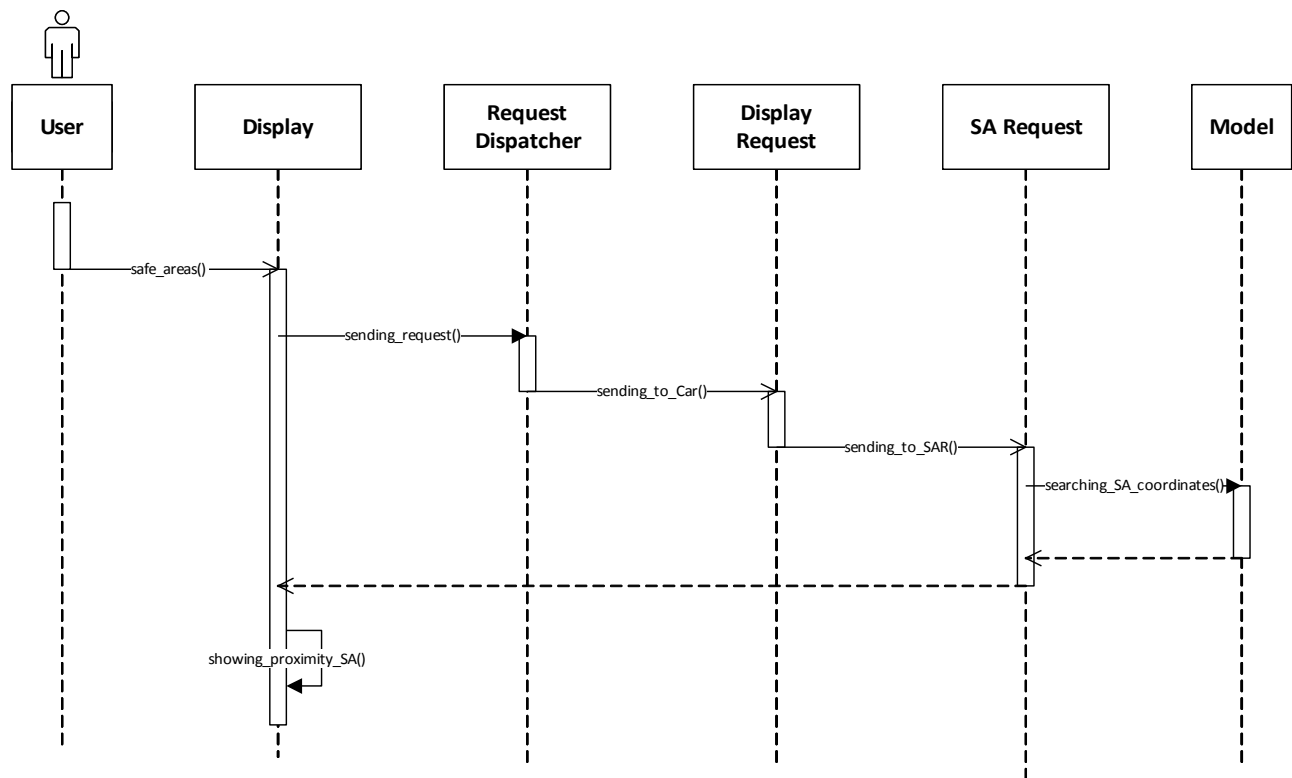
In the above diagram, it is represented the "view safe areas" procedure. The function safe_areas() represent the user pressing the button "Safe Area" on the car display. The Display component forwards the request to components till reaching (through them) the SA Request component. Here, the component search the most significant coordinates (the function searching_SA_coordinates()) in the Model abstraction, returning them to the component who start the request procedure. Than, the Display component shows them to the user.

Booking a Car
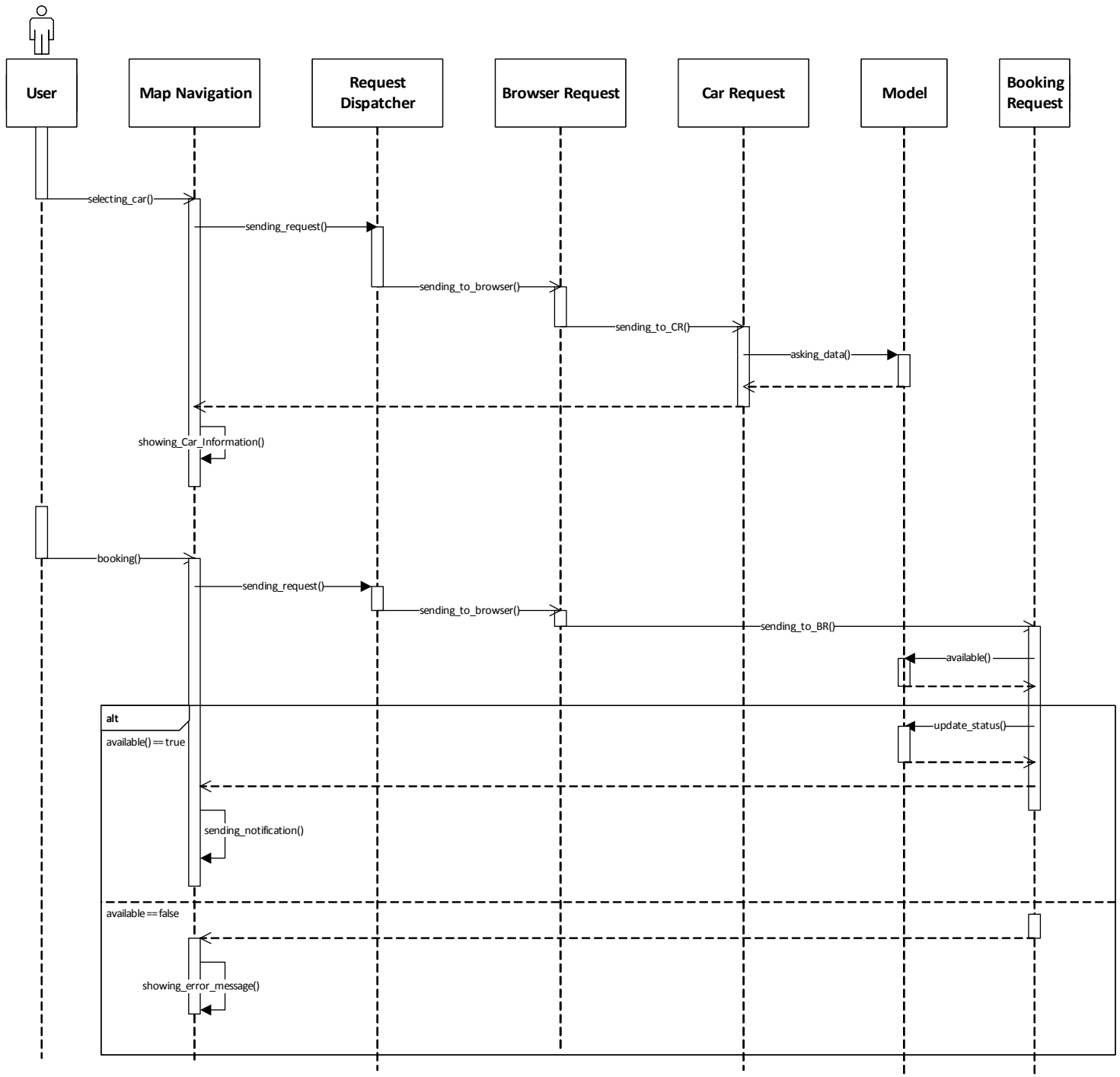
In the above diagram, it is represented the booking procedure. The user, in order to use a car, has to select it on the map page. This is represented with the function selecting_car(). The request is forwarded by the Map Navigation component to the Car Request component (through Dispatcher Request and Browser Request components). Here the component search in the Model abstraction the information about the selected car, returning them to the Map Navigation component. When

received, this component shows to the user a rectangular banner with all the information about the selected car and the button "BOOK IT". Then the user presses on that button to start the booking procedure (this is represented by the booking() function). The requested is forwarded from the Map Navigation component to the Browser Request component (through Dispatcher Request and Browser Request components). This component control the availability of the car: if the available() function returns "true", the Browser Request component has to update the status of the car, from "free" to "locked", returns to the Map Navigation component that sends a notification to the user with all the information about booking; if the function return "false", an error message is shown on the user screen

Active MSO function
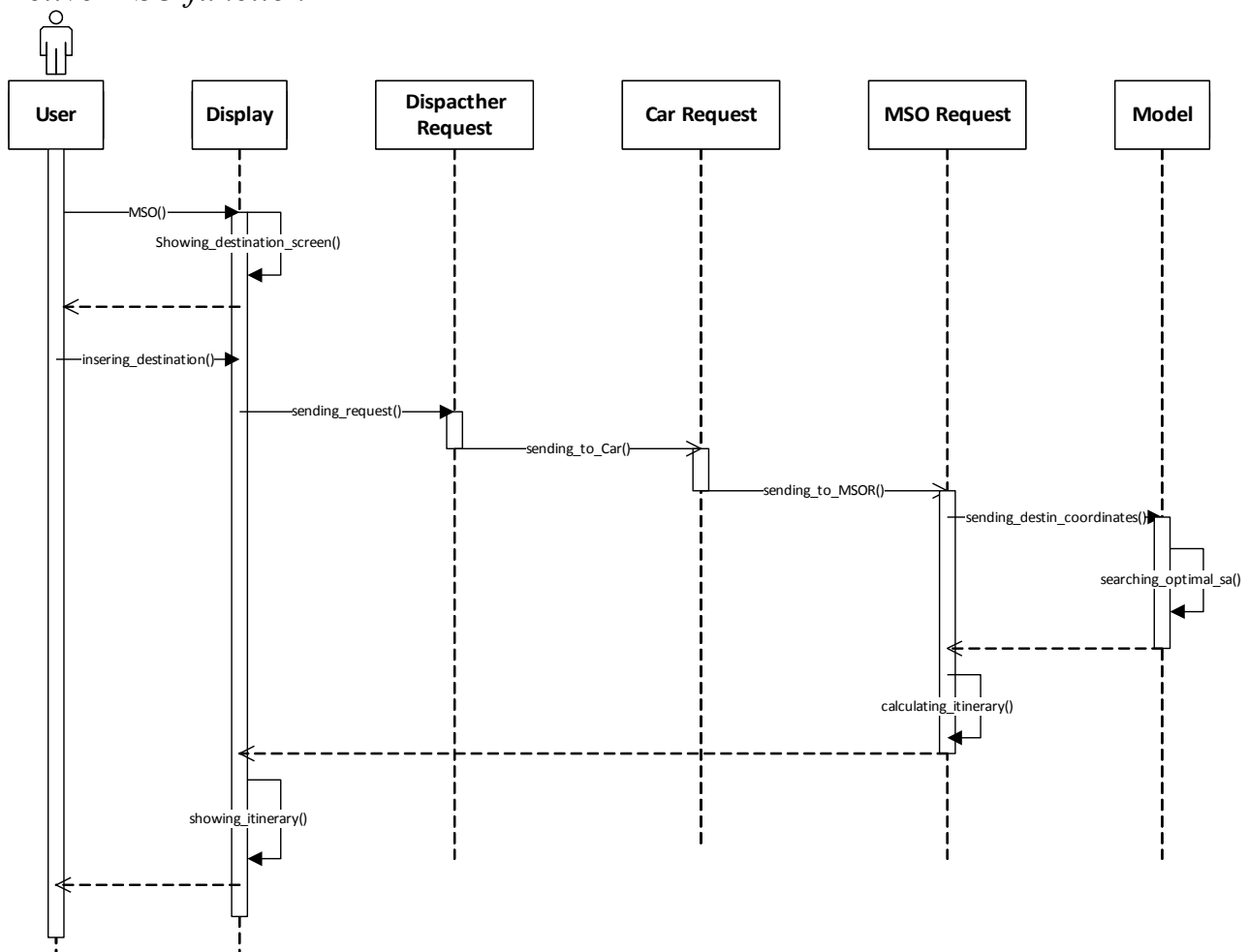
In the above diagram, it is represented the procedure for the activation of the "Money Saving Option" service. The user, as first action, presses on the button "MSO", represented by the function MSO(). Then, if the user didn't insert previously an

address as destination, system asks it (in the case shown above, the address was not inserted in the blank space). Pressing the "search" icon, represented by the function destination(), the Display component forwards the request to the dedicated sub component of the server logic (in this case, "MSO Request" component). At this point, the component searches in the Model component all the nearest safe areas, focusing on the coordinates of the final destination and the amount of cars on each safe area (represented by functions sending_destin_coordinates() and searching_optimal_sa()). When the algorithm founds the optimal result, it is returned to the Display component, that calculates the itinerary (calculating_itinerary() function) and publishes it to the user on the screen (showing_itinerary() function).
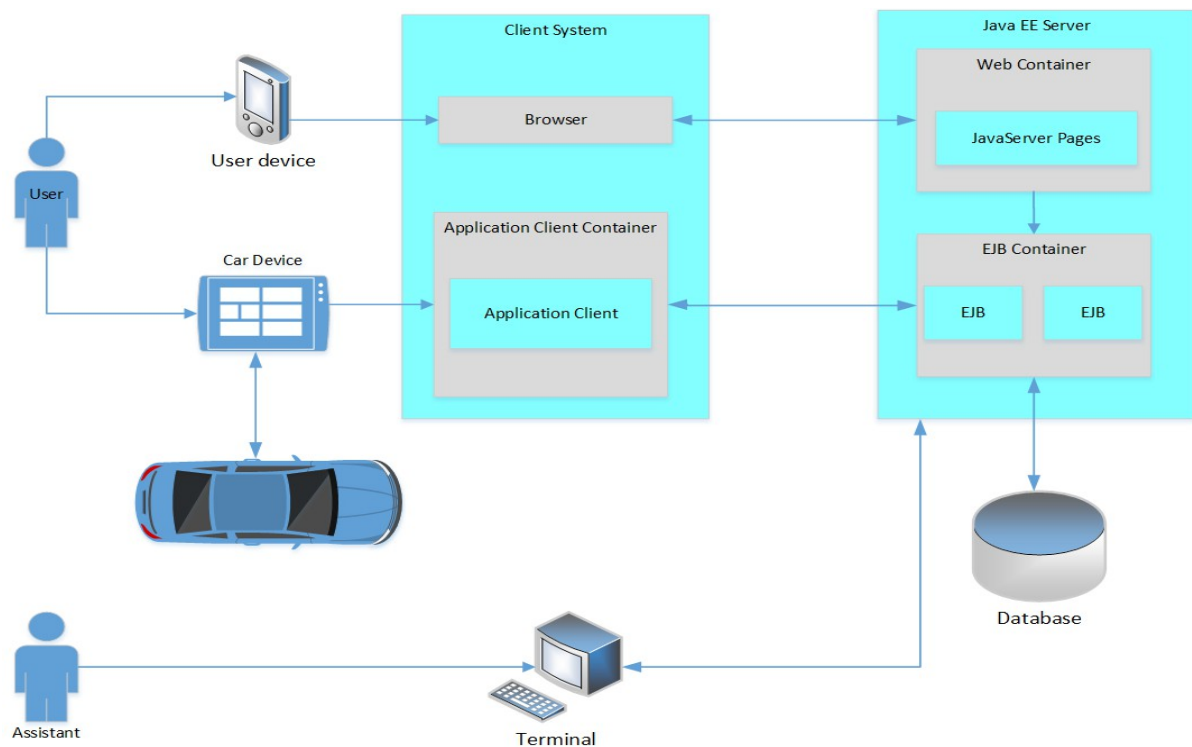
2.5 Component interfaces
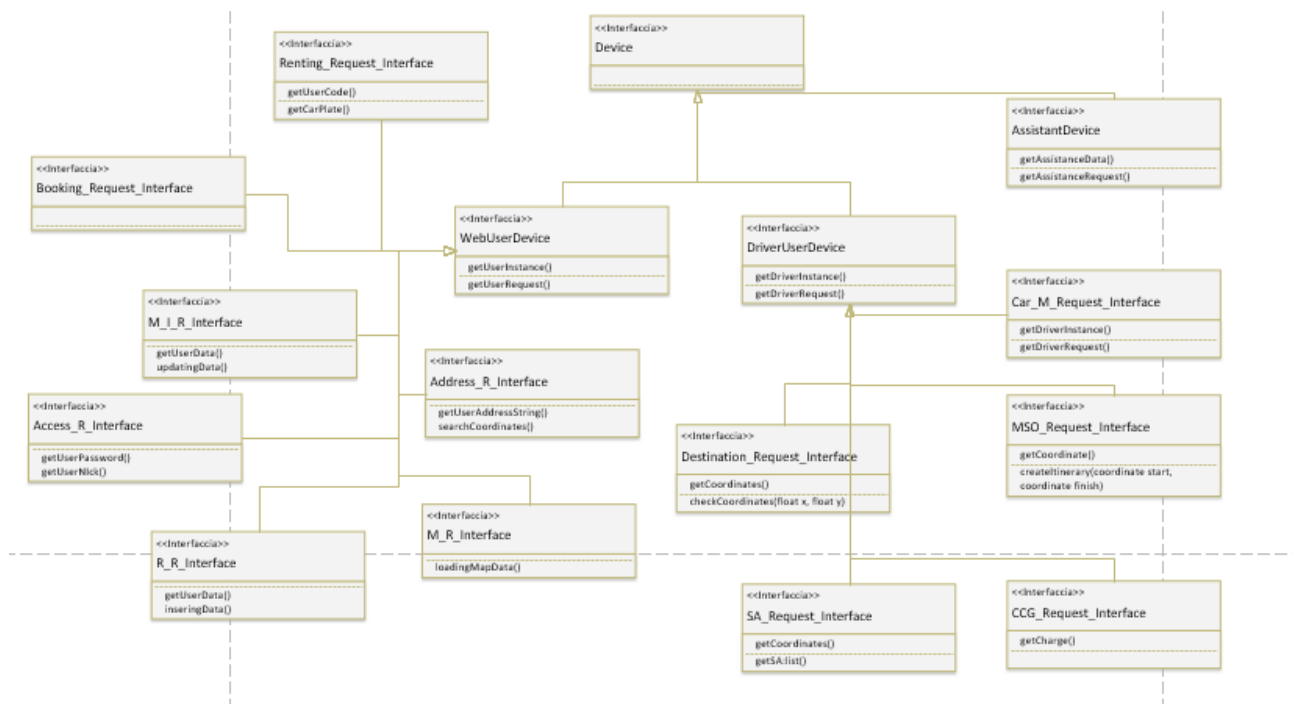
2.6 Selected patterns and architectural style
The image below shows how client and server communicate when user does a request in its personal device.
We see now some consideration about the architecture we have chosen.

User Device
It is convenient to develop the user application in a browser because in all devices equipped by Internet connection it is possible to use a browser.
In opposite, if we create an application it must take into account that there are a lot of system operations and there may be problems of compatibility, so we think browser is the better solution.
In the server the Web Container receives the requests that users send by the browser and it satisfies them using EJB Container.

Car Device
We supposed that each car has the same device. In every car device is installed an application written in Java that communicates directly with the EJB Container. Obviously every device is connected to Internet.
In this case we can use an application because all devices belong to PowerEnJoy so it doesn't exist problems about compatibility.

EJB (Enterprise JavaBeans)
EJB are the software components that implement the "business tear", it means that they contain the logic about how the server must work and they decide who and how it is possible to access to the database.
We use this tear because we want that the logic of our system is independent from the language used in the client, so the system is easily updatable (for example, in the

future, PowerEnJoy can decide to create an app for Android and IOS in order to improve the performance or to offer new services, so with EJB it isn't necessary to modify the logic tear).

JavaServer Pages
This component in the Server builds the HTML page to be shown to user in the browser, leaning to the EJB. Otherwise this component receives the request from the client and uses EJB to reply.

Now we speak about the main pattern we have used.

MVC
Our architecture is strongly depending on the Model-View-Controller pattern.
"View" is formed by all the interfaces and the everything that compose User Client, Car Client and Assistance Client components.
"Model" is, basically, composed by the DBMS.
"Controller" is all the logic of the software, situated in EJB container in the Server System.

Three Tier Client/Server
Our application is build on client-server model.
It is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called server, and service requester, called client.
In our case the clients are user device and car device, while the server is the computer that manages the request of the client.
The whole architecture can also be divided into three layers, or tiers:
1) Application layer: in this layer belonged the browser and the car display, so all the ways the user (both renter and assistant) can interact with the system;
2) Business Logic layer: in this layer belonged the system server, because there it's contained all the logic of the system;
3) Data layer: composed by the DBMS, containing all the data of the system;
All these layer are divided by a firewall in order to provide security to the service's user.
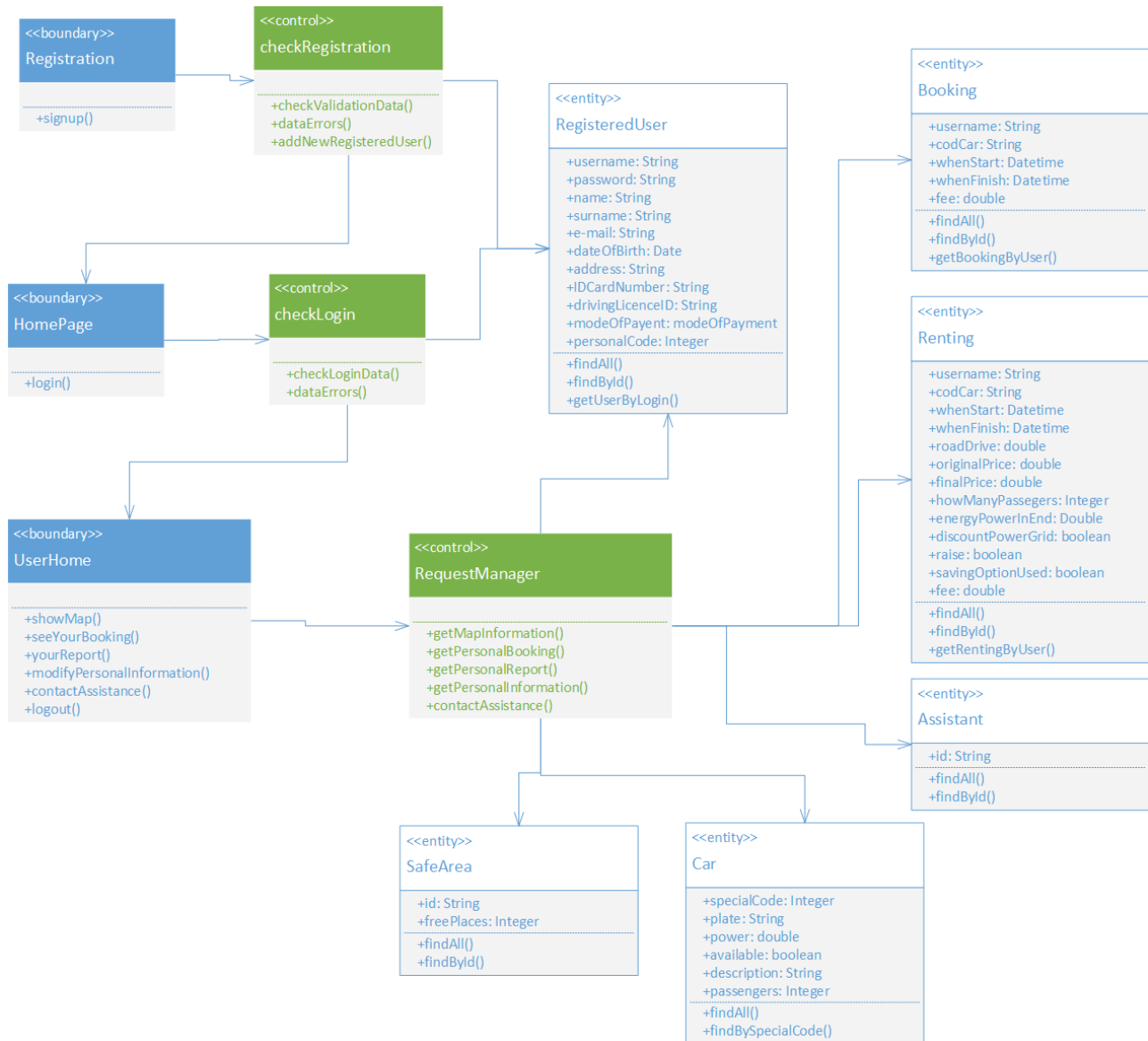
Divide et Impera
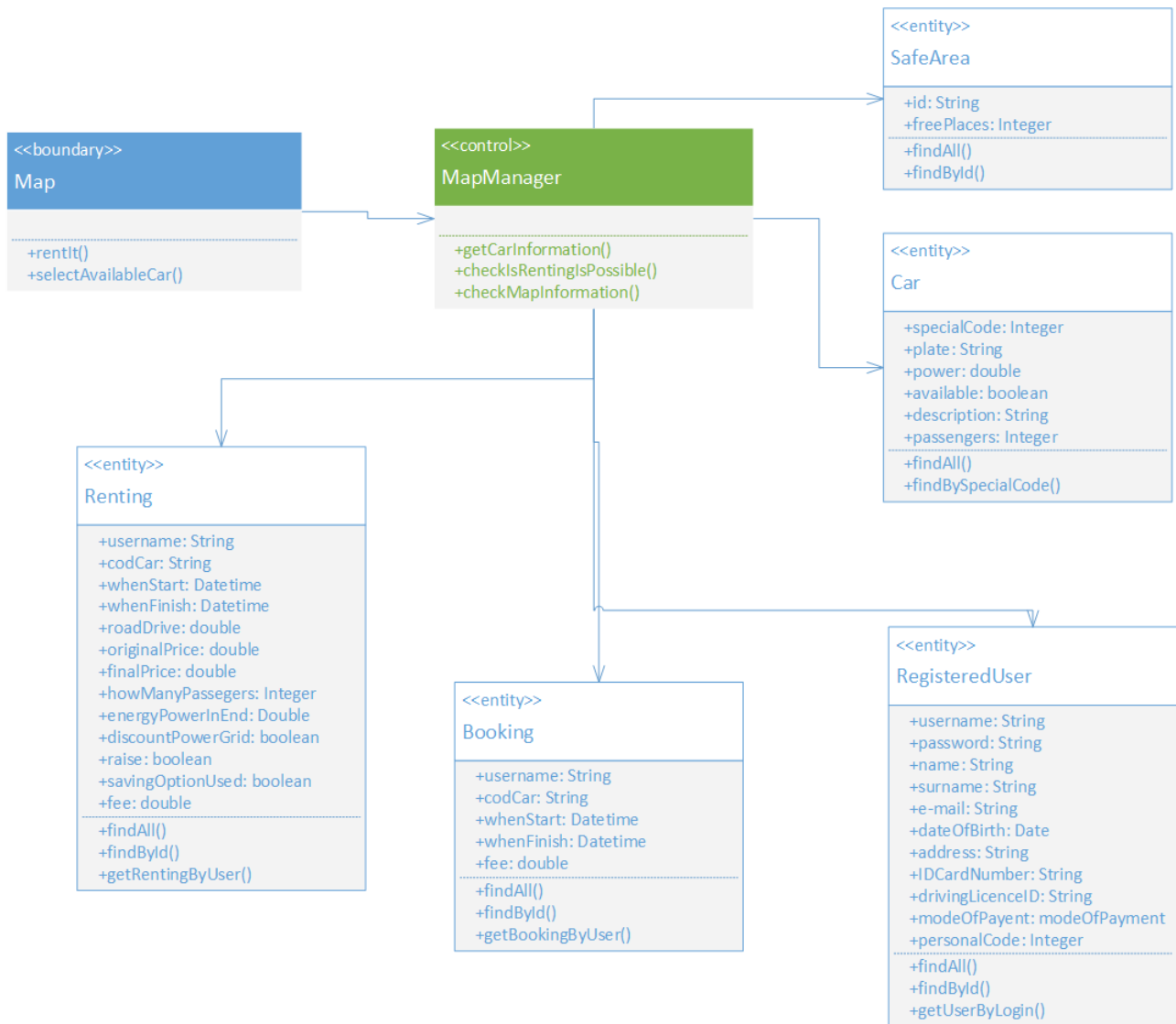The logic has been set in three level:
1) level of the dispatcher, where all the requests come and are divided into two categories, "request from browser" and "request from car";
2) level of the devices, where the two components divided another time the requests into other categories, where the number of categories depends on the number of functionalities provided by our system;
3) level of the request, where every component has the request of its type and the only task is to provide the service.

For greater clarity here we put two BCE diagrams.
The first is about the registration, login and the main functionalities, after the login, on the browser in the user device.
In the second graph we put the main control and entity in the map function on the browser in the user device.

<<boundary>>
Map

+rentIt()
+selectAvailableCar()

<<control>>
MapManager

+getCarInformation()
+checkIsRentingIsPossible()
+checkMapInformation()

<<entity>>
SafeArea

+id: String
+freePlaces: Integer

+findAll()
+findById()

<<entity>>
Car

+specialCode: Integer
+plate: String
+power: double
+available: boolean
+description: String
+passengers: Integer

+findAll()
+findBySpecialCode()

<<entity>>
Renting

+username: String
+codCar: String
+whenStart: Datetime
+whenFinish: Datetime
+roadDrive: double
+originalPrice: double
+finalPrice: double
+howManyPassegers: Integer
+energyPowerInEnd: Double
+discountPowerGrid: boolean
+raise: boolean
+savingOptionUsed: boolean
+fee: double

+findAll()
+findById()
+getRentingByUser()

<<entity>>
Booking

+username: String
+codCar: String
+whenStart: Datetime
+whenFinish: Datetime
+fee: double

+findAll()
+findById()
+getBookingByUser()

<<entity>>
RegisteredUser

+username: String
+password: String
+name: String
+surname: String
+e-mail: String
+dateOfBirth: Date
+address: String
+IDCardNumber: String
+drivingLicenceID: String
+modeOfPayent: modeOfPayment
+personalCode: Integer

+findAll()
+findById()
+getUserByLogin()

3 Algorithm Design

As an example, in this paragraph we implemented in Java some of the most important methods we will put in the EJB container.

In this class obviously we doesn't pay attention about the type of connection between user or car device and system but here we just implement the logic of the system.

SystemClass.java

```java
public class SystemClass{
    private Car car;
    private SafeArea safeArea;
    private Assistant assistant;
    private User user;

    public static void main(String[] args)
    {
        car=new Car();
        safeArea=new SafeArea();
        assistant=new Assistant();
        user=new User();
    }

    /*
    * car device sends the coordinates about the address the user wants to reach
    */
    public Coordinates moneySavingOption(double longitude, double latitude){
        //System return to car device the coordinates of best position for MSO, car device has the map
        //and recognize the position found\end{document}
        return safeArea.safeAreaNearest(longitude, latitude);
    }

    /*
    *   User browser call this method when the user want to login to the system
    */
    public boolean login(String username, String password)
    {
        return user.checkDates();
    }

    /*
    *   This function is called when user try to unlock the car with his personal device
    */
    public boolean startRenting(String username, String carCode, String carPlate, int rating)
    {
        if(user.isFreeForRenting(username, carCode, carPlate) && car.isAvailable(carCode, carPlate))
        {
            boolean result1, result2;\end{document}
            result1=user.startRenting(String username, String carCode);
            if(result1==true)
                result2=car.setNotAvailable(String carCode, String carPlate, rating);
            if(result1==true && result2==true)
                //No error occurred, renting gets success
                return true;
        }
        //Some error occurred
        return false;
    }

    /*
    *   This function is called when user stop rent and close the door of the car.
    *   Return the message that the system will send to the user.
    */
    public String stopRenting(String username, String carCode, double longitude, double latitude,
                              double roadDriven, int howManyPasseger, double finalPower)
    {
        String notify=user.calculatePrice(username, carCode, longitude, latitude,
                              roadDriven, howManyPasseger, finalPower);
    }
}
```

SystemClass is the main class in the EJB container. When user requires something, system calls a method of this class. The other classes support this one and interact with the database.

SafeArea.java

```java
public class SafeArea{
    Connection con;

    public SafeArea(){
        try{
            //Setup the connection with url, username and password to connect to the db
            con =DriverManager.getConnection ("URLDataBase", "myUserName", "myPassword");
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    /*
     *   This method return the safe area (with some free places) nearest to the position that user want to reach
     *   If 2 safe area has the same distance this method chose the safe area that has more
     *   power grid available
     */
    public Coordinate safeAreaNearest(longitude, latitude)
    {
        try{
            //We create an object Statement to query the DB
            Statement cmd = con.createStatement ();
            //Execute a query and save the result in a the object "ResultSet"
            String qry = "SELECT * FROM SafeArea WHERE freePlaces>0;";
            ResultSet allSafeAreas = cmd.executeQuery(qry);
            Coordinate c=new Coordinate();
            double distance=Double.MAX_VALUE;
            int powerGridAvailable=0;
            while(allSafeAreas.next())
            {
                double safeLatitude=allSafeAreas.getInteger("latitude");
                double safeLongitude=allSafeAreas.getInteger("longitude");
                double dist=Math.pow((latitude-safeLatitude),2)+Math.pow(longitude-safeLongitude,2);
                dist=Math.sqrt(dist,2);
                qry = "SELECT count(*) AS powerGridAvailable FROM SafeArea JOIN PowerGrid ON SafeArea.ID=PowerGrid=SafeAreaID WHERE "+
                        +"PowerGrid.SafeAreaID ="+allSafeAreas.ID+
                        +"AND PowerGrid.available=TRUE GROUP BY SafeArea.ID;";
                ResultSet result = cmd.executeQuery(qry);
                int pGA=result.getInteger("powerGridAvailable")
                if(dist<distance || (dist==distance && pGA>powerGridAvailable))
                    {
                        distance=dist;
                        c.longitude=longitude;
                        c.latitude=latitude;
                        powerGridAvailable=pGA;
                    }
            }
            allSafeAreas.close();
            cmd.close();
            return c;
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Car.java

```java
public class Car{
    Connection con;

    public Car(){
        try{
            //Setup the connection with url, username and password to connect to the db
            con =DriverManager.getConnection ("URLDataBase", "myUserName", "myPassword");
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public boolean carIsAvailable(String carCode, String carPlate)
    {
        boolean free=false;
        try{
            //We create an object Statement to query the DB
            Statement cmd = con.createStatement ();
            String qry = "SELECT * FROM Car WHERE specialCode="+carCode+" and plate="+carPlate+" and available=true and power > 20;";
            ResultSet check = cmd.executeQuery(qry);
            //if the query didn't find anything in the tables car isn't available, in opposite way car is available
            if(check.next())
                free=true;
            check.close();
            cmd.close();
            return free;
        }catch(Exception e)
        {
            e.printStackTrace();
            return false;
        }
    }

    public boolean setNotAvailable(String carCode, String carPlate, rating)
    {
        try{
            //We create an object Statement to query the DB
            Statement cmd = con.createStatement ();
            String qry = "UPDATE Car SET available=FALSE where carCode="+carCode+" and carPlate="+carPlate+";";
            cmd.executeQuery(qry);
            String qry = "UPDATE Car SET rating="+rating+" where carCode="+carCode+" and carPlate="+carPlate+";";
            cmd.executeQuery(qry);
            check.close();
            cmd.close();
            return true;
        }catch(Exception e)
        {
            e.printStackTrace();
            return false;
        }
    }
}
```

User.java

```java
1   public class User{
2       Connection con;
3
4       //In this example we put the price variable here in the code like final static private but
5       //in real case we can also put them in a DB and read them so in this way assistant can modify them throug DB
6       /*
7        * If the renting lasts more than one day user pay for every day
8        */
9       final static private priceForMinute=0.20;
10      final static private priceForDay=100;
11
12      /*
13       *   If user uses the car for more than maxDistance he pays for each Km over maxDistance
14       */
15      final static private maxDistance=50;
16      final static private priceForEveryKm=1;
17      /*
18       *   User pay a fee if he doesn't leave the car in a safe area
19       */
20      final static private feeForKm=5;
21
22      public User(){
23          try{
24              //Setup the connection with url, username and password to connect to the db
25              con =DriverManager.getConnection ("URLDataBase", "myUserName", "myPassword");
26          }catch(Exception e)
27          {
28              e.printStackTrace();
29          }
30      }
31
32      public boolean checkDates(String username, String password)
33      {
34          boolean correct=false;
35          try{
36              //We create an object Statement to query the DB
37              Statement cmd = con.createStatement ();
38              //Execute a query and save the result in a the object "ResultSet"
39              String qry = "SELECT * AS c FROM RegisteredUser WHERE username="+username+" and password="+password+";";
40              ResultSet check = cmd.executeQuery(qry);
41              if(check.next())
42                  correct=true;
43              check.close();
44              cmd.close();
45              return correct;
46          }catch(Exception e)
47          {
48              e.printStackTrace();
49              return false;
50          }
51      }
52
53      public boolean isFreeForRenting(String username, String carCode)
54      {
55          boolean free=true;
56          try{
57              //We create an object Statement to query the DB
58              Statement cmd = con.createStatement ();
59              //With the first query we check that user isn't renting any car (in other words he finished all renting he started)
60              String qry1 = "SELECT username FROM Renting WHERE username="+username+" and whenFinish=NULL;";
61              ResultSet check1 = cmd.executeQuery(qry);
62              //With the second query we check that user isn't booking any car or if he is tring to rent a car he has booked yet
63              String qry2 = "SELECT username FROM Booking WHERE username = "+username+" and (whenFinish = NULL and carCode != "+carCode+");";
64              ResultSet check2 = cmd.executeQuery(qry);
65              //if the query didn't find anything in the tables user is free
66              if(check1.next() || check2.next())
67                  free=false;
68              check.close();
69              cmd.close();
70              return free;
71          }catch(Exception e)
72          {
73              e.printStackTrace();
74              return false;
75          }
76      }
77
78      public boolean startRenting(String username, String carCode)
79      {
80          try{
81              //We create an object Statement to query the DB
82              Statement cmd = con.createStatement ();
83              /*
84               *   With the first query we insert a new renting record, in this record for the moment just three column
```

```java
 *   have a value, the other values are NULL and the system will set them when the renting will finish
 */
DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Date date = new Date();
String whenStart=dateFormat.format(date);
String qry1 = "INSERT INTO Renting (username, carCode, whenStart) VALUES ("+username+","+carCode+","+whenStart+")";
cmd.executeQuery(qry);
/*
 *   in the second query we set the finish of booking if the user is renting a car he has booked. Otherwise
 *   this query doesn't do anything
 */
qry = "UPDATE Booking SET whenFinish="+whenStart+" WHERE username="+username+" and whenFinish=NULL;";
cmd.executeQuery(qry);
cmd.close();
return true;
}catch(Exception e)
{
    e.printStackTrace();
    return false;
}
}

public String calculatePrice(String username, String carCode, double longitude, double latitude,
                            double roadDrive, int howManyPasseger, double finalPower)
{
    try{
        String notification="";
        //We create an object Statement to query the DB
        Statement cmd = con.createStatement ();
        //First System sets the finish of the renting
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        Date date = new Date();
        String whenFinish=dateFormat.format(date);
        String qry = "UPDATE Renting SET whenFinish="+whenFinish+" WHERE username="+username+" and whenFinish=NULL;";
        cmd.executeQuery(qry);
        //Now System calculates the price
        //How many time last the renting?
        double price, finalPrice, fee;
        qry = "SELECT whenStart FROM Renting WHERE username="+username+" and whenFinish="+whenFinish+";";
        check=cmd.executeQuery(qry);
        Date whenStart;
        if(check.next())
            whenStart=check.getDate("whenStart");
        else
            return "Error";
        if(whenFinish.getDay()-whenStart.getDay()==0)
        {//rent lasts less than 1 day
            int duration=(whenFinish.getHour()-whenStart.getHour())*60+Math.abs(whenFinish.getMinute()-whenStart.getMinute());
            price=duration*priceForMinute;
        }
        else
        {//rent lasts more than 1 day
            price=(whenFinish.getDay()-whenStart.getDay())*priceForDay;
        }
        //put in DB roadDrive
        qry="UPDATE Renting SET roadDrive="+roadDrive+" WHERE whenFinish="+whenFinish+";";
        cmd.executeQuery(qry);
        //User did over maxDistance?
        if(roadDrive>maxDistance)
        {
            price=price+(roadDrive-maxDistance)*priceForEveryKm;
        }
        notification="Rent from "+dateFormat.format(whenStart)+" to "dateFormat.format(whenFinish)+"\n";
                    +"Your original price= "+price+"\n";
        //put original price in DB
        qry="UPDATE Renting SET originalPrice="+price+" WHERE whenFinish="+whenFinish+";";
        cmd.executeQuery(qry);
        finalPrice=price;
        //update in DB how many passeger there were
        qry="UPDATE Renting SET howManyPasseger="+howManyPasseger+" WHERE whenFinish="+whenFinish+";";
        cmd.executeQuery(qry);
        //System applies the discount for passegers
        if(howManyPasseger>2)
        {
            finalPrice=finalPrice*0.9;
            notification=notification+"Discount for passegers was applied\n";
        }
        //update in DB the final power
        qry="UPDATE Renting SET energyPowerInEnd="+finalPower+" WHERE whenFinish="+whenFinish+";";
        cmd.executeQuery(qry);
        //System applies the discount for power
        if(finalPower>=50)
        {
            finalPrice=finalPrice*0.8;
```

```java
169                        notification=notification+"Discount for power was applied\n";
170                    }
171                    //Update raise in DB, raise is false initially
172                    qry="UPDATE Renting SET raise=FALSE WHERE whenFinish="+whenFinish+";";
173                    cmd.executeQuery(qry);
174                    //System applies raise for power
175                    if(finalPower<20)
176                    {
177                        finalPrice=finalPrice*1.3;
178                        notification=notification+"Raise for power was applied\n";
179                        //update raise in DB
180                        qry="UPDATE Renting SET raise=TRUE WHERE whenFinish="+whenFinish+";";
181                        cmd.executeQuery(qry);
182                    }
183                    //System applies discount or raise for final position
184                    Coordinate c=distanceFromNearestSafeArea(longitude, latitude);
185                    double distance=Math.pow((latitude-c.latitude),2)+Math.pow(longitude-c.longitude,2);
186                    distance=Math.sqrt(distance,2);
187                    if(distance>0)
188                    {//user doesn't leave the car in the safe area
189                        fee=distance*feeForKm;
190                        finalPrice=finalPrice+fee;
191                        notification=notification+"Car is left out of safe area, Fee="+fee+"\n";
192                        //update fee in DB
193                        qry="UPDATE Renting SET fee="+fee+" WHERE whenFinish="+whenFinish+";";
194                        cmd.executeQuery(qry);
195                    }
196                    else
197                    {
198                        //update fee in DB
199                        qry="UPDATE Renting SET fee=0 WHERE whenFinish="+whenFinish+";";
200                        cmd.executeQuery(qry);
201                    }
202                    if(safeAreasHasPowerGrid(c))
203                    {//safe area when user leave the car has power grid
204                        finalPrice=finalPrice*0.7;
205                        notification=notification+"Discount for power grid presence was applied\n";
206                        //update raise in DB
207                        qry="UPDATE Renting SET discountPowerGrid=TRUE WHERE whenFinish="+whenFinish+";";
208                        cmd.executeQuery(qry);
209                    }
210                    else
211                    {
212                        //update raise in DB
213                        qry="UPDATE Renting SET discountPowerGrid=FALSE WHERE whenFinish="+whenFinish+";";
214                        cmd.executeQuery(qry);
215                    }
216                    //System check final price because it doesn't want to apply 30% of raise 2 time
217                    if(distanceFromNearestPowerGrid(c)>3 && finalPower>=20)
218                    {//user leave the car far 3 or more Kilometres from the nearest safe area with power grid
219                        finalPrice=finalPrice*1.3;
220                        notification=notification+"Raise for leaving car far from power grid was applied";
221                        //update raise in DB
222                        qry="UPDATE Renting SET raise=TRUE WHERE whenFinish="+whenFinish+";";
223                        cmd.executeQuery(qry);
224                    }
225                    notification=notification+"Final Price="+finalPrice;
226                    //insert in DB the final price
227                    qry="UPDATE Renting SET finalPrice="+finalPrice+" WHERE whenFinish="+whenFinish+";";
228                    cmd.executeQuery(qry);
229                    check.close();
230                    cmd.close();
231                    return notification;
232            }catch(Exception e)
233            {
234                e.printStackTrace();
235                return "Error";
236            }
237        }
238        private Coordinate distanceFromNearestSafeArea(double longitude, double latitude)
239        {
240            ...
241        }
242        private boolean safeAreasHasPowerGrid(Coordination c)
243        {
244            ...
245        }
246
247        private double distanceFromNearestPowerGrid(Coordinate c)
248        {
249            ...
250        }
251 }
```
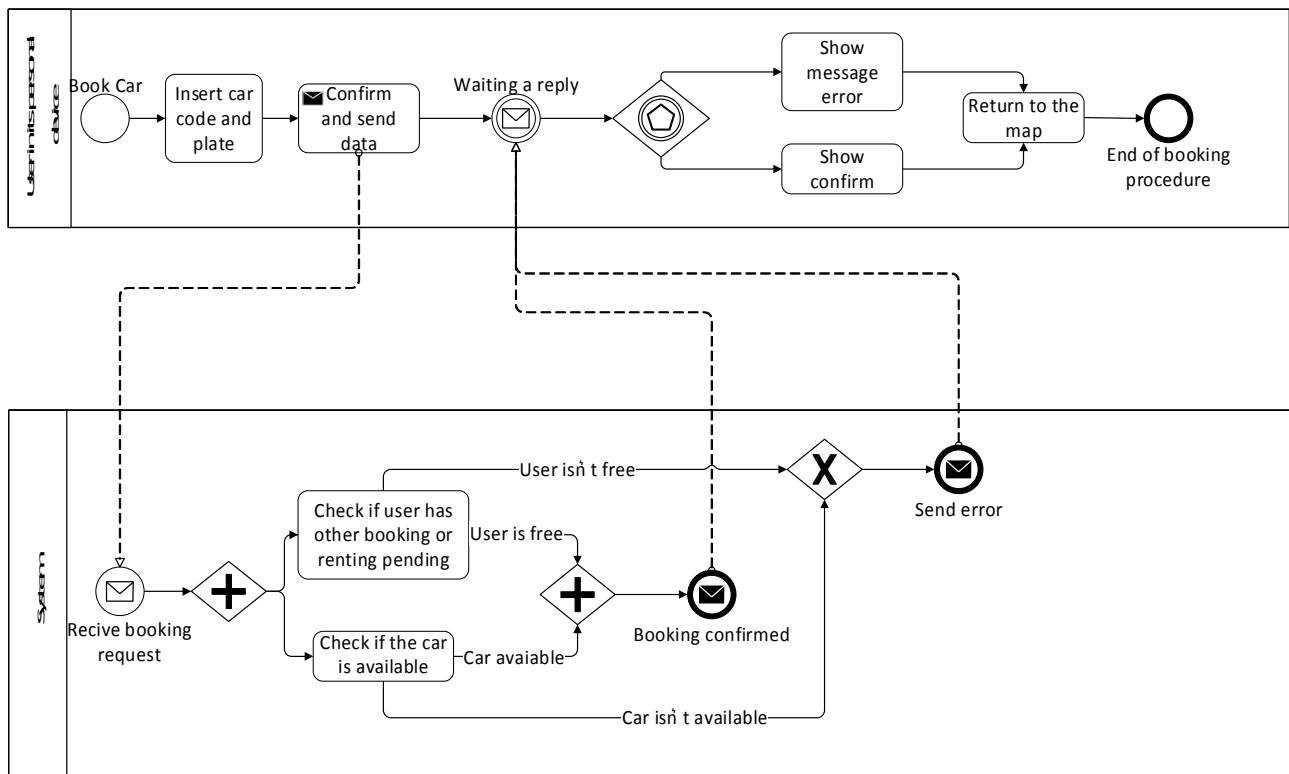
Here we show the booking function through a BPMN diagram.

4 User Interface Design

This application is thought to be compatible with any device, but in order to visualize the website and the main services in a more approachable way, we have developed principally two visualization modes: the portable device mode and the PC version.
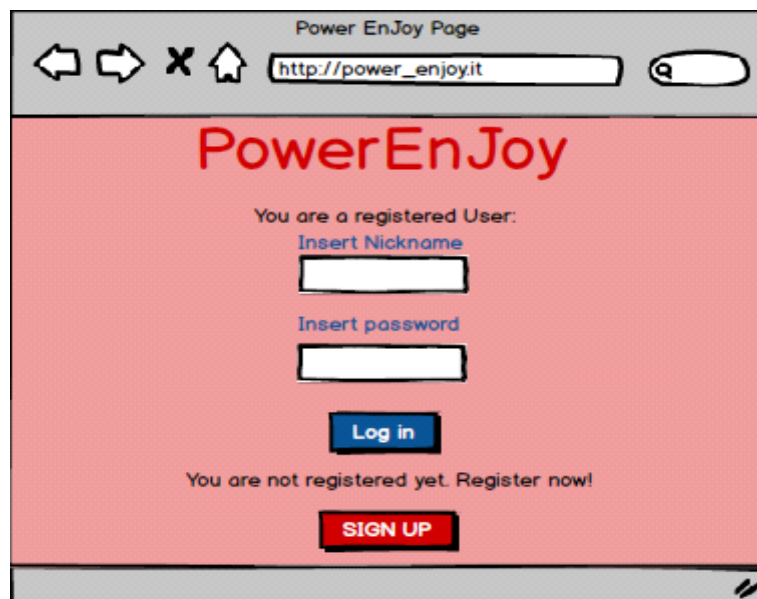
4.1 Mockup

Home Page
At first both guests and users are addressed to these pages. In the first sketch we have included in the RASD, we thought to use a same page both for the forms of login and of registration. We changed our mind and we made two different pages. So in this page users can effect the login inserting a nickname and a password and, if the fields are correct, user is sent to the User Home. Instead, if a user is not registered yet and want to sign up, he/she clicks the special button and she/he is addressed to the Registration Page.

Registration Page
If users wants to sign up they have to fill the forms, inserting their name, surname, nickname, password, identity card number, driving license id and the mode of payment they prefer, and then click the button "Sign up". Eventually the system show them the incorrect fields and a error message, but if the data are correct it is sent an email to user containing a special code using for renting cars. If user wants to cancel his/her registration he/she can always click the top left-arrow that sent him/her to the Home Page. Below the pictures both of the Home Page and the Registration page in the PC version and the portable device view.

User Home

After login, users are addressed to their home, where they can choose what to do pressing the special buttons: they can modify their personal information, view the map with their position and available cars, they can see details about their booking, they can visualize the report about historic payment and system notifications or they can contact assistance if they need support.
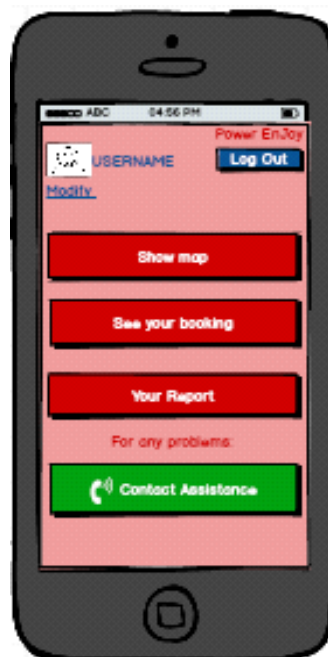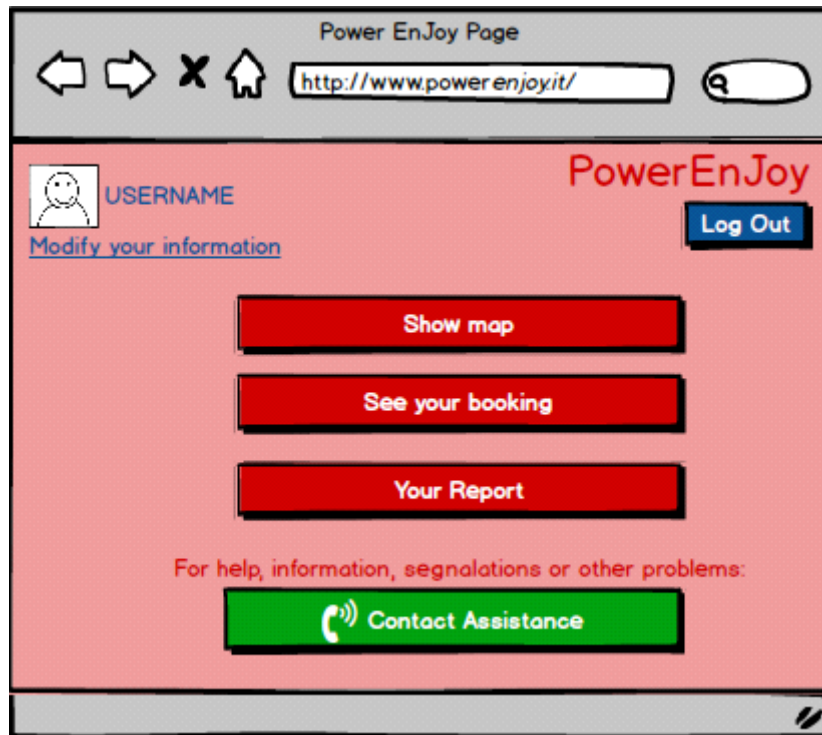
Show Map

Clicking 'Show Map' from User Page, users are able to visualize the map centered on their position (light blue cycle) and cars nearby. If they want to see other cars in a far-away position, they can insert the location address in the special form "Search Location". They can select cars pressing on them (red icons on the map), so users can see more details and information about them. Instead, if they are near the car (either if they have booked it before or not), they can rent it clicking on the button in the bottom page, and they will be readdressed to the Renting Page. At last, clicking on the top left-arrow, they will return to User Home.
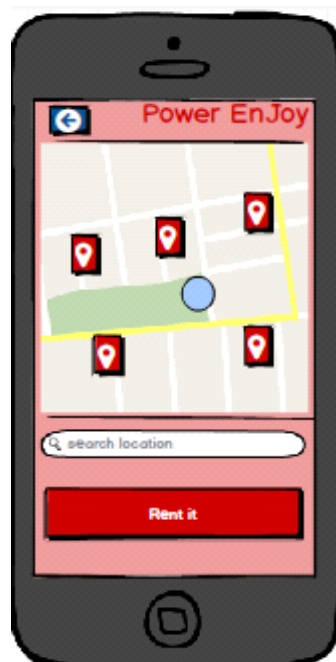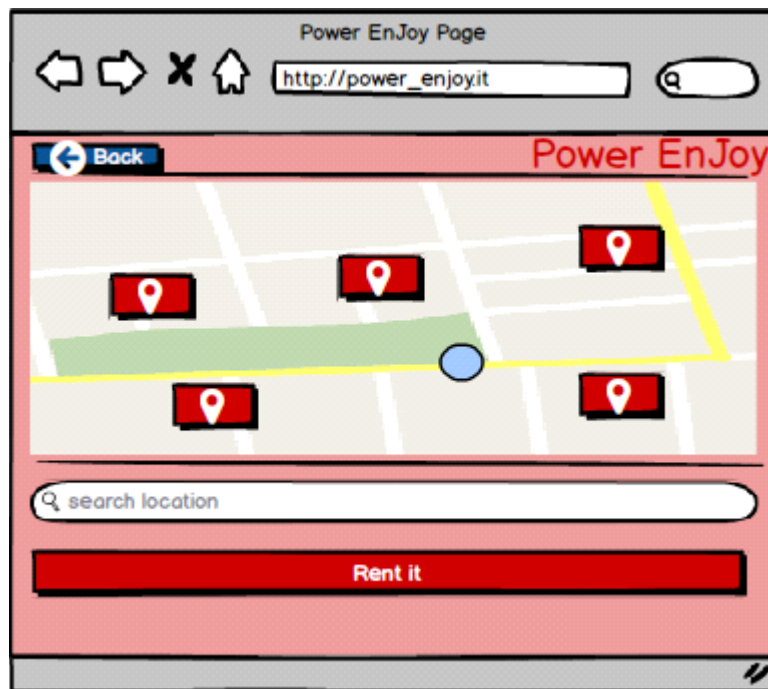
Select car
When users click on the cars (red icons on the map), it appears a window with the main information and giving more details, such as how many seats there are in the car, how much power, the name of the street in which find it, etc. Users can decide to close the window, pressing "X" and return to examine the map, or they can choose to book the car clicking "Booking it". They could visualize information about their booking from the User Home, clicking the appropriate voice "See your booking".
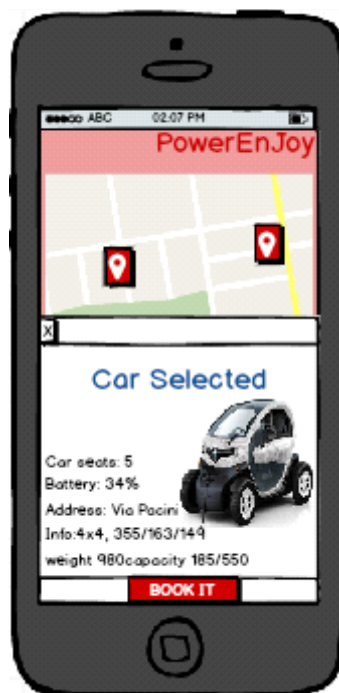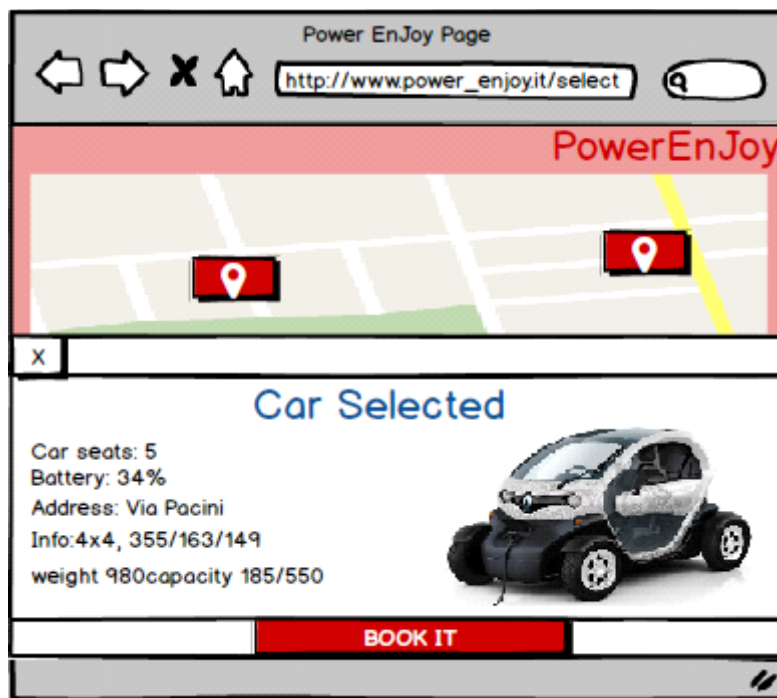
Booking page
Clicking 'See your Booking' from User Home, users are addressed to this page in which they can see more details about their booking, such as car information, position in which finding it and the time that remains before the reservation expires. If users haven't booked nothing, these fields are empty. Users can also rent the car from this page, clicking "Rent it", which address them to the Renting page, or they can delete the booking (if they have booked something) clicking "Delete Booking" button. The "Show map" button, instead, shows the map centered on the position of the car. At last, clicking the left-arrow in the top left screen, users return to User Home.
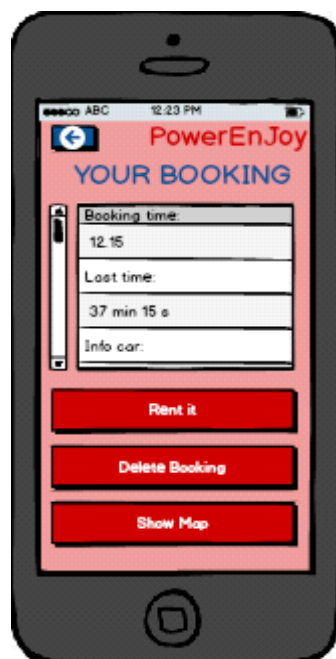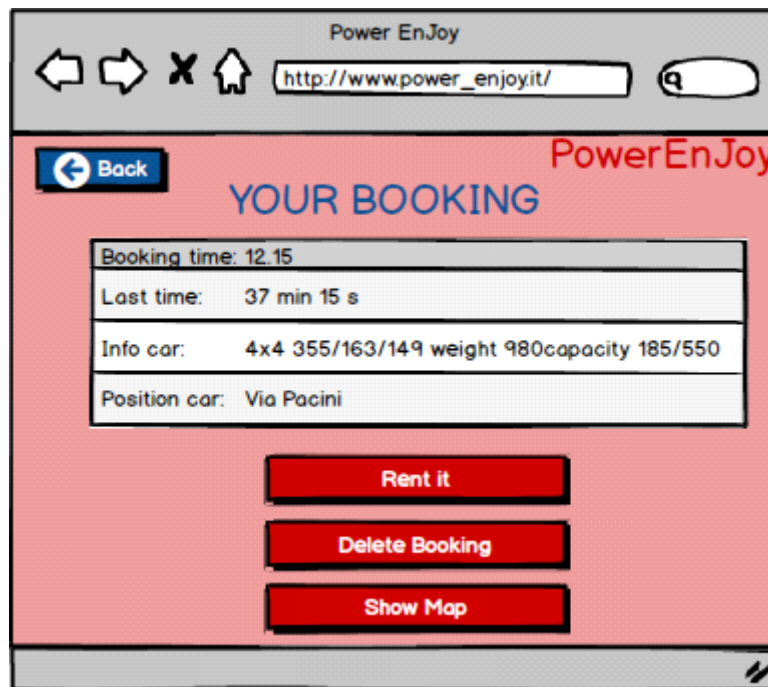
Renting page

Users can reach this page either from the map or from the Booking Page. They have to insert the number of the plate and the car's code (which is inside the car) in the specific forms, and give a rate of the conditions of the cars, internal and external, giving a vote from 1 to 5, clicking on the star icon. After that, they can click "Unlock" and if the data are correct or incorrect it appears a successful message with the following operations to do (opening the car and turning on it) and users are sent to the map page. While, the data are incorrect, it is shown an error message and they have to fill fields again. In any moment users can choose to return to the previous page, clicking the "Cancel" button.

Your Personal Area

If users press the text "Modify" below their own picture in the User Home, they are addressed to this page in which they can modify their old information. They can save the new ones with the button "Save and Exit" or they can decide to turn back to User Home clicking the top left-arrow.

Report Page
Clicking on "Your Report" from User Home, users are sent to this page. They can visualize all the notifications they received, actions they made and the payment history. Notifications are divided in 5 main typologies, such as payment, confirmation of booking, fee for booking, deleting a booking and generics. Notifications were sent before to users, shown as messages on their device, but they are memorized here in a summary way. Pressing on the voices of the report, it will be displayed the whole message that was sent before. Clicking on the top left-arrow users return to the User Home.


40

Here an example of how the notifications are displayed on the devices (in this case on a portable device). Clicking on the mail icon, users could read the whole message. If they want to read the content of the message again, they will have to go on the Report Page.


Contact Assistance

When users click on "Contact Assistance" button, from the User Home, it is started a call that the system directs to the assistants. This is an example of how the screen is displayed on the devices (in this case, on portable device) during the call.
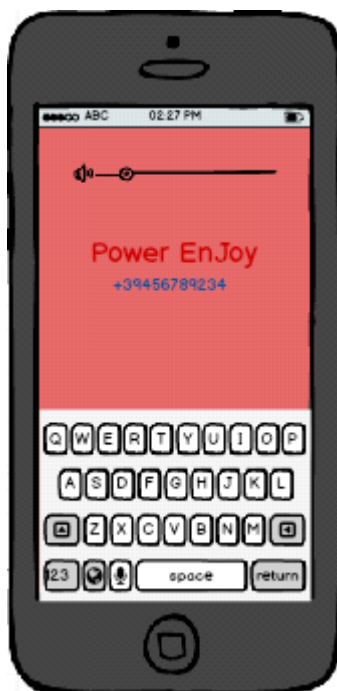
Car's Display

Turn on car

Once entered in the car and turned on the screen, users must insert the code (sent by email after their registration) in order to turn on the car, filling the form below. If correct, it appears a message that say to users to turn the keys already inserted in the car ignition.
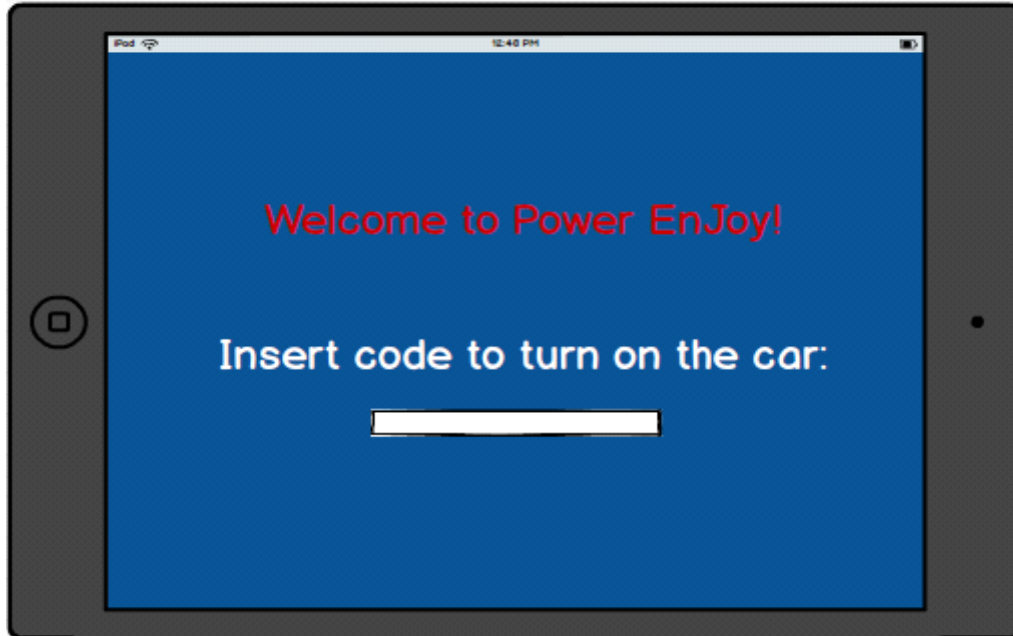
Map View

This is the interfaces which is visible from the screen on the dashboard of the car. Users can insert the destination, which will be visualized on the screen with also the car position and the fastest road to go. Clicking "Safe Area" or "Power Grid" the map would show respectively the safe areas and the safe areas with power grid situated near the car, while clicking "ActiveMSO" the modality "Money saving option" would be activated. The button "View Current Charge" will show the current charge for few second on the screen.
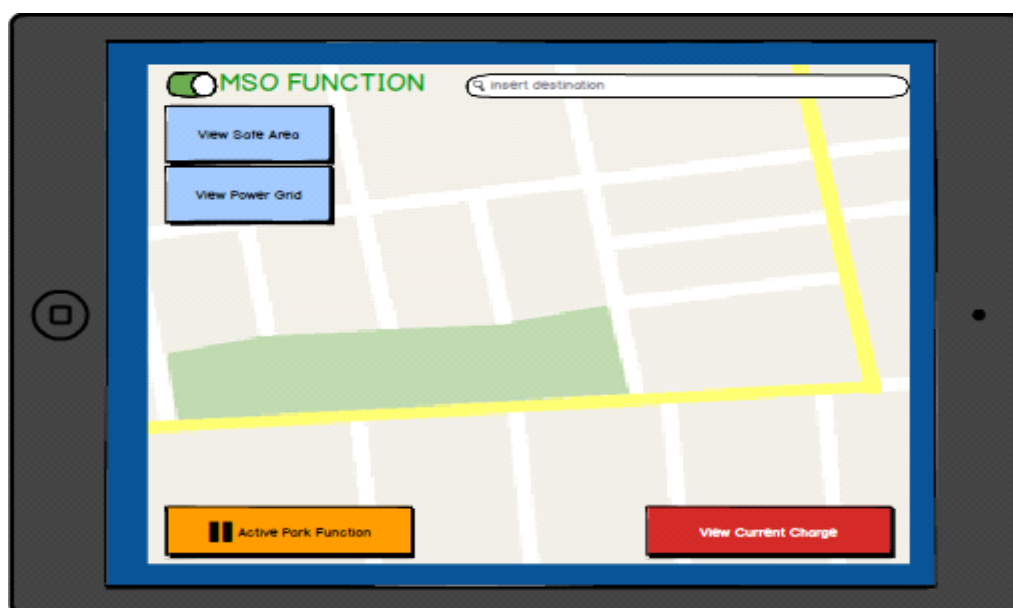
Clicking on "Park function active", it will be active the park function, that permits to users to leave momentarily the car. The screen will show the message that "Park function is active" and unlocks the car, but it locks all the other functionalities (they are shown in gray color because they are not available). Users can exit from the car and lock it with the car's key, the screen will stay on. To turn off the park function users must be inside the car and click "off" on the car device.
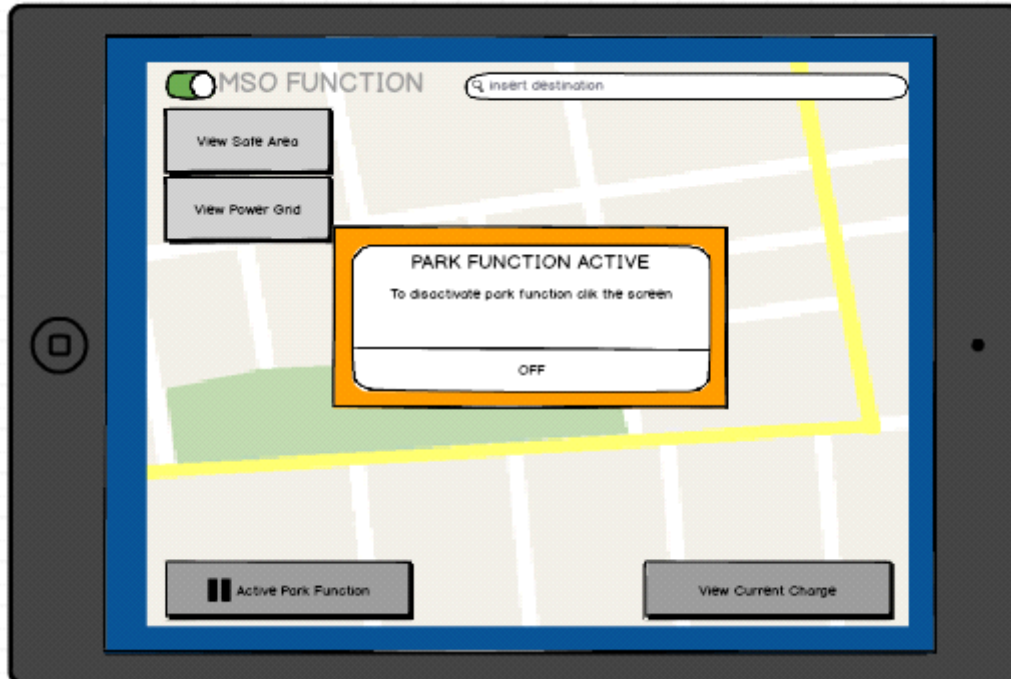
4.2 UX Diagram

Here we put the UX Diagrams. They show the views with which user interact. The first diagram shows the views that users see on the car device, the second one on the browser in their personal device.
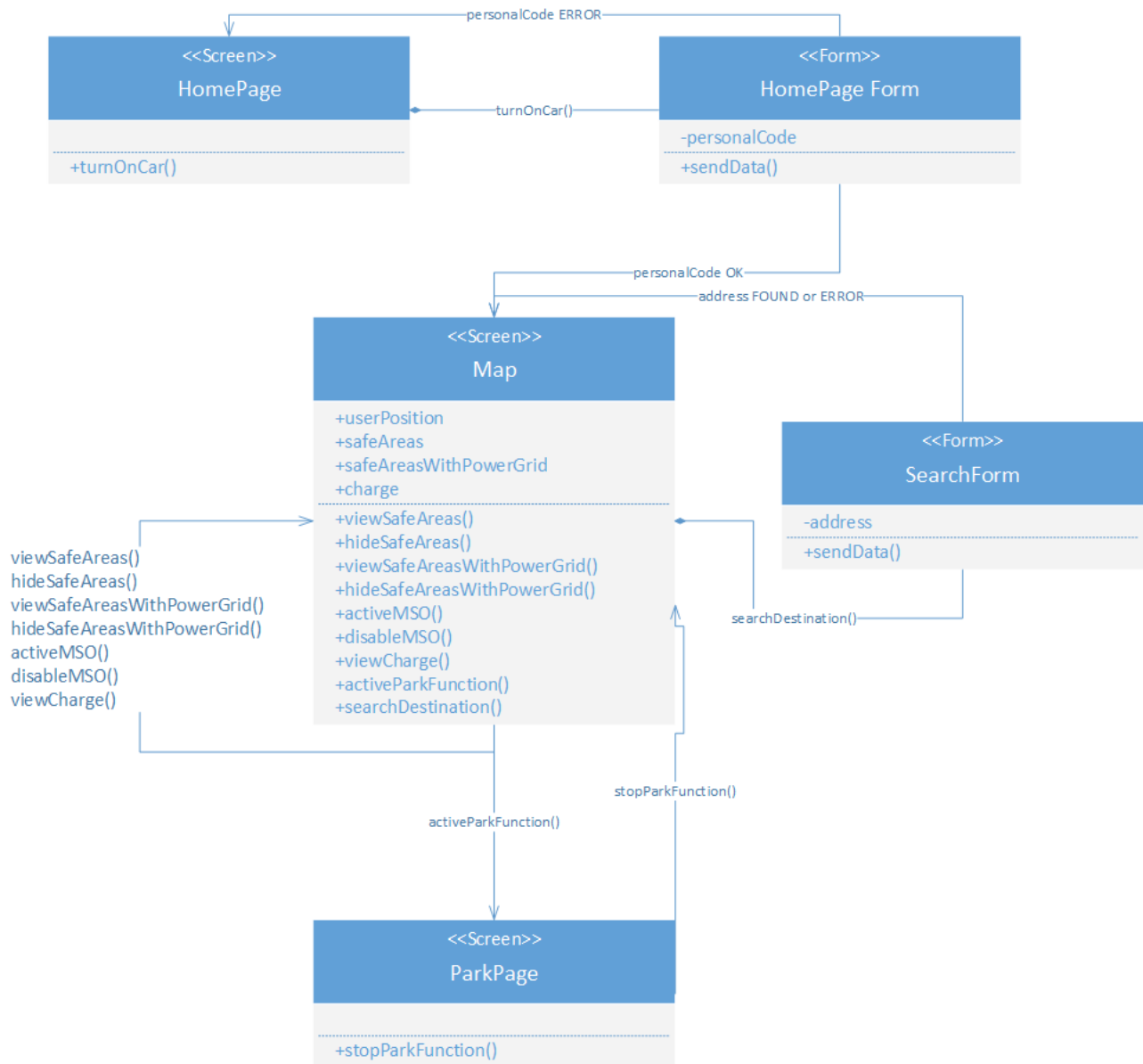
5 Requirements Traceability

The design of this project was made aiming to fulfill optimally the requirements and goals specified in the RASD. The reader can find here under the list of these requirements and goals and the designed component of the application which will assure its fulfillment.

[G1] Registration of a person to the system:

→ Information Registration → Dispatcher Request → Browser Request →
Information Registration → Model;

[G2] Show the map to the user:
Two ways:
1) Map Navigation → Dispatcher Request → Browser Request → Map Request
   → Model
2) Display → Dispatcher Request → Car Request → Map Request → Model

[G3] Searching an address in the map:
Map Navigation → Dispatcher Request → Browser Request → Address Request

G[4] Renting a Car:
Two ways:
1)→ Map Navigation → Dispatcher Request → Browser Request → Renting Request
→ Model
2)→ Information Management → Dispatcher Request → Browser Request →
Renting Request → Model

G[5] Booking a Car:
→ Map Navigation → Dispatcher Request → Browser Request → Booking Request
→ Model

G[6] Turn on Car:
→ Display → Dispatcher Request → Car Request → Current Charge Request →
Model
→ Display → Engine

G[7] Park Function:
→ Display → Engine

G[8] Adding or modifying payment method:
→ Information Management → Request Dispatcher → Browser Request →
Mod.Personal.Inf.Request → Model

G[9] Active Saving Option
→ Display → Request Dispatcher → Display Request → MSO Request → Model

G[10] Checking discount for passengers:
when the user ends his renting, turning off the car, the Engine component sends the
request through dispatcher to Service Payment component. Here the payment is
calculated, taking in consideration discounts and raises. For passenger discount, the
system check if sensors have registered presence of more than two passengers

G[11] Checking discount for battery empty:
when the user ends his renting, turning off the car, the Engine component sends the request through dispatcher to Service Payment component. Here the payment is calculated, taking in consideration discounts and raises. For battery empty discount, the system check if battery level is over the half

G[12] Checking discount for power grid replenishment:
when the user ends his renting, turning off the car, the Engine component sends the request through dispatcher to Service Payment component. Here the payment is calculated, taking in consideration discounts and raises. For power grid replenishment discount, the system check if car is linked to power grid through car sensors

G[13] Checking raises:
when the user ends his renting, turning off the car, the Engine component sends the request through dispatcher to Service Payment component. Here the payment is calculated, taking in consideration discounts and raises. For raises, the component checks the location of the car from the nearest power grid station and/or the power battery level

G[14] View current charge:
→ Display → Request Dispatcher → Display Request → Charge Request → Model

6 Effort Spent
Angelo Falci: 26 hours, Purpose; Definitions, Acronyms, Abbreviations; Overview; Selected patterns and architectural style; Algorithm Design; UX Diagram; BCE

Valentina Lanzuise: 27 hours, Purpose; Scope; Definitions, Acronyms, Abbreviations; Document Structure; Deployment view; User Interface Design; Mockup

Simone Lazzaretti: 31 hours, Definitions, Acronyms, Abbreviations; Component View; Run-time view; Component interfaces; Selected patterns and architectural style; Requirements Traceability

7 Reference Documents
To realize this document we have consulted different sources:
- Lecture slides
- Our previous document about Requirements Analysis and Specification Document
- Paper on the Green Move Project
- Wikipedia
- Old projects

The main programs we used are:
- Visio Standard 2016: to realize the main diagrams.

- MyBalsamiq: to draw the part related to the mokups.
- Notepad++: to write the alloy code in a clear way.
- Paint: to model some pictures.
- Draw.io: to realize other diagrams
- OpenOffice Writer: to write the document
- Eclipse: to execute and control the algorithm part
- Notepad: to write the algorithm part
- Dropbox: to share documents and pictures, and to collaborate

8 Revision History

11/12/16: version 1.0 released

5/2/17: new version 1.1 released

Version 1.1: improved level of details for component view graphs (changed icons, added icons like interfaces, changed direction of some arrows); updated the names of some components in run-time view section; changed description in "Park Function" case inside run-time view section, according to what is written in the latest version of the RASD document; updated component interface section, to provide a better view of interface section; BCE diagram shifted to "Selected patterns and architectural style" section; modified "Sign Up" sequence diagram and related description; modified "Renting Request" sequence diagram and related description;