PowerEnJoy

Riccardo Cattaneo 873647
Fabio Chiusano 874294

Version: 1.0.0
Release date: 10-11-2016

1 Summary

2 Introduction

2.1 Description of the given problem

We will project the digital management system for PowerEnJoy, which is a car-sharing service that exclusively employs electric cars.
The system, first, has to provide functionalities normally provided by car-sharing services such as the possibility for a new user to register and log in, to find locations of nearby available cars and to reserve them.
It has also to guarantee that a user who has used the service pays a fee that should be as fair as possible.
In order to strengthen the ecological mission of PowerEnJoy, the system aims to incentivize virtuous behaviours of the users by adapting the final bill for every ride.
For example, if there have been at least three people on the car, or if the car has been left charging at special parking areas, the system has to apply a discount. Instead, if the car has been left far from a charging station with a low battery level, it has to apply a charge on the bill.

2.2 Goals

- Users can see and select an available car close to them, or close to a specified address, and reserve it for up to one hour before they pick it up;
- Users can get in a car only if they are near it and they have reserved it;
- Users should pay proportionally to minutes they have used the car, and they should see in real time the amount of the bill;
- Users could register to the system and have their personal area;
- Virtuous behaviours by users should be incentivized.
  In particular, the system could achieve that by charging user, on the last ride:
  - 10% less if they share their trip with at least other two passengers;
  - 20% less if the car is left with at least 50% of battery level;
  - 30% less if the car is left plugged in at special parking areas;
  - 30% more if the car is left at more than 3km from the nearest power grid station with less than 30% of battery level.

2.3 Domain properties

We suppose that these conditions hold in the analysed world:

- All the users have a device connected to the Internet, possibly with a GPS built in;
- All the electric cars have an on-board computer that allows execution of Java software;
- All the electric cars have a GPS to indicate their actual position, that cannot be turned off, and a sensor for every seat which detect the presence of a passenger;
- All the electric cars have Internet connection, that is always working and can't be turned off;
- All the electric cars on-board computer can't be turned off or sabotaged;
- GPS position is always accurate;
- All the cars can carry a maximum of 4 passengers;
- A car can be in only one zone at the same time and this is the actual real zone;
- In a special parking area with power grid stations there is always space for a car to be plugged in;
- Users behave politely and have no intention of cheating;
- After that the Payment Handler has validated the Payment Method of one user, payments using that method always succeed;
- The company never reach the limit of requests per day for the external services.

2.4 Glossary

- *Available Car*: a car parked in a Safe Area and not already reserved;
- *Battery level*: how much in percentage the car battery is charged;
- *Bill*: compensation to be paid for a ride by the user;
- *Car*: for "car", "electric vehicle" or "electric car" we mean an electric car involved in PowerEnJoy™, ready to be reserved and used;
- *Guest*: a guest is a person that probably for the first time accesses the system or that has not already signed up;
- *Management System*: the management system of the car share service;
- *Passengers*: people that are in a car during a ride. The user that drives the car is included in the passengers count.
- *Payment information*: everything that the external payment handler needs in order to run its services.
- *Reservation current cost*: the current cost of the reservation, without discounts applied;

- *Reservation final discharged cost*: the total cost of the reservation with discounts applied;
- *Reservation start area*: area where the user takes the electric car, from which he can starts the ride;
- *Reservation start time*: the instant of time in which the user reserved the car;
- *Reservation*: is the ability of a user to reserve a car at most one hour prior to the pick up;
- *Ride pickup time*: the instant of time in which the user enters the car he has reserved and the car engine ignites;
- *Ride release area*: area where the user leaves the electric car, terminating the reservation;
- *Ride release time*: the instant of time in which the car unlocks itself after the user left it.
- *Ride*: usage of a car, by one user, who has to pay it with a bill. It starts when the user picks up a car after a reservation is made and it ends when the user leaves the car in a safe area;
- *Safe Area*: area where a user can leave the car he's renting;
- *Search-on-map service*: a service that provides the possibility to search locations and see distances on a digital map.
- *Special Safe Area* or *Safe Area with power grid station*: Safe Area where the user can plug the car into the power grid station in order to get a discount on the ride;
- *Unlocked Car*: a car that can be entered since the user who reserved it is near it;
- *User credentials*: everything that a user needs in order to log into the system;
- *User*: a user is a person already registered in the system, so that has a profile, and sometimes is interested to reserve and use a car;

2.5 Text assumptions

We will assume few things about the specification document:

- It is said that "users must be able to reserve a single car for up to one hour before they pick it up". Since we do not know in advance when a user will give the car back, we assume that it is not possible to reserve a car for a future time. It is only possible to reserve a car from the current time and keep the reservation for one hour. The user can pick up the car in that period of time and use it for how long he/she wants, or let the reservation expire and pay a small fee.
  It is important to choose carefully the amount of the fee, otherwise we would have given the users the possibility for some bad behaviours where most of the cars are reserved and a few of them is actually used.

- If the user gets close to the car he rented and the car unlocks itself, then the car should be also able to lock itself if the user distances himself.
- Power grid maintenance is always ok.
- The payment for a ride is carried out when the user quits the car.
- The payment for the reservation fee is carried out as soon as the reservation hour expires.
- Nothing is said about the payment handler, therefore we decided to use an external payment handler and we assume that it is able to manage the cases where there is something wrong with a payment (e.g. the user has no money).
- The discount for having at least two other passengers onto the car is applied only if those passengers are in the car before the engine ignites.
- It is said that "A user that reaches a reserved car must be able to tell the system she's nearby" and we assume that GPS is not necessary. Therefore the user can use the system portal to explicitly say that she's nearby, without satellite connections.
- It is said that "The system stops charging the user as soon as the car is parked in a safe area and the user exits the car". Since "the user exits the car" is ambiguous, we assumed that this is true when the user is not near the car anymore, namely when the distance between user and car is bigger than a certain amount.

2.6 Constraints

2.6.1 Regulatory policies

The Management System must ask the users the permission to get their position and to manage sensible data (position). The user is not forced to accept them in order to use the service because everything can be done without satellite connections:

- He/she can search near available cars explicitly writing his/her current location.
- He/she can tell the system that he/she is near the rented car in order to unlock it.

2.6.2 Hardware limitations

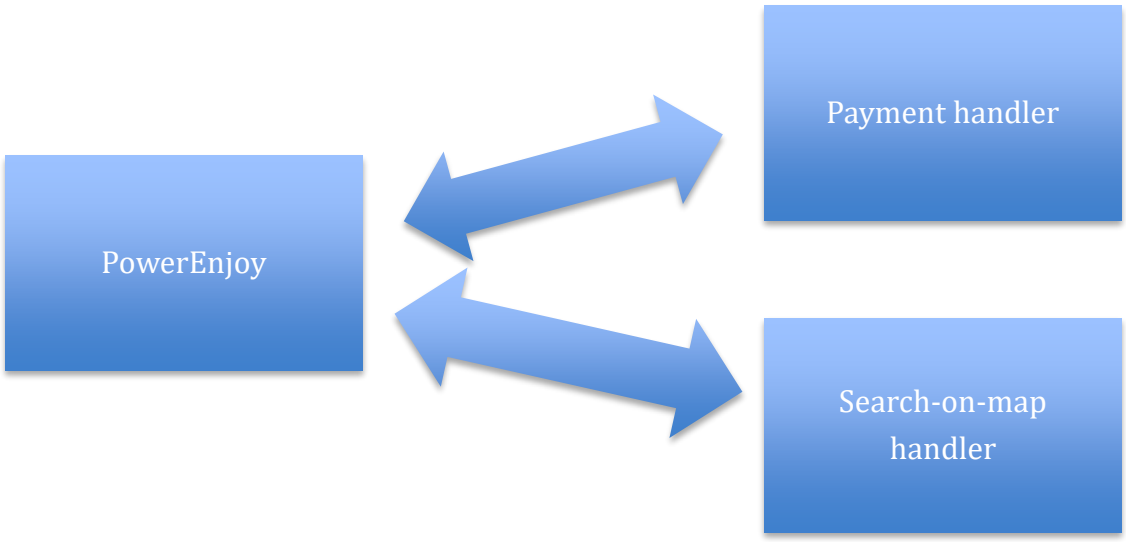
Internet 3G/4G connections is required since the system must be usable on top of a platform built for mobile systems. GPS is not required since every operation can be made without it.
Users must have enough storage space to install the PowerEnjoy app.

The server must be able to run a web server application, for instance Apache Web Server.

System component	Hardware limitations
Client mobile device	• 3G/4G connection • GPS connection (optional) • Enough memory for the app package
Server	• Internet connection • Should run an OS that allows Apache Web Server to run or similars
Car on-board computer	• Internet connection • GPS connection • should run Java software

2.6.3 Interfaces to other applications (system boundaries)

The system relies on an external payment handler (e.g. PayPal) and on an external search-on-map service (e.g. Google Maps). In both cases, we must register our application to the service provider in order to take advantage of it. Sometimes it is also possible to buy a pro licence for these services so that the cost for the service with many requests is smaller. However, this step can be made once the managers find it necessary, as it does not compromise the scalability of our system.



2.6.4 Parallel operation

The server supports of course parallel operations from different clients. In the future, it will be possible to have multiple redundant logic layers distributed on more than one server.

2.7 Proposed system

We propose to make a web app that will give users a comfortable way to use our service. Since the user have to pick up the reserved car in only one hour since the moment of the reservation, we considered the PowerEnjoy service as an “on the fly” service and therefore we preferred the web app instead of the web site. However, we will take the necessary precautions to make it easy to build the web site in the future.

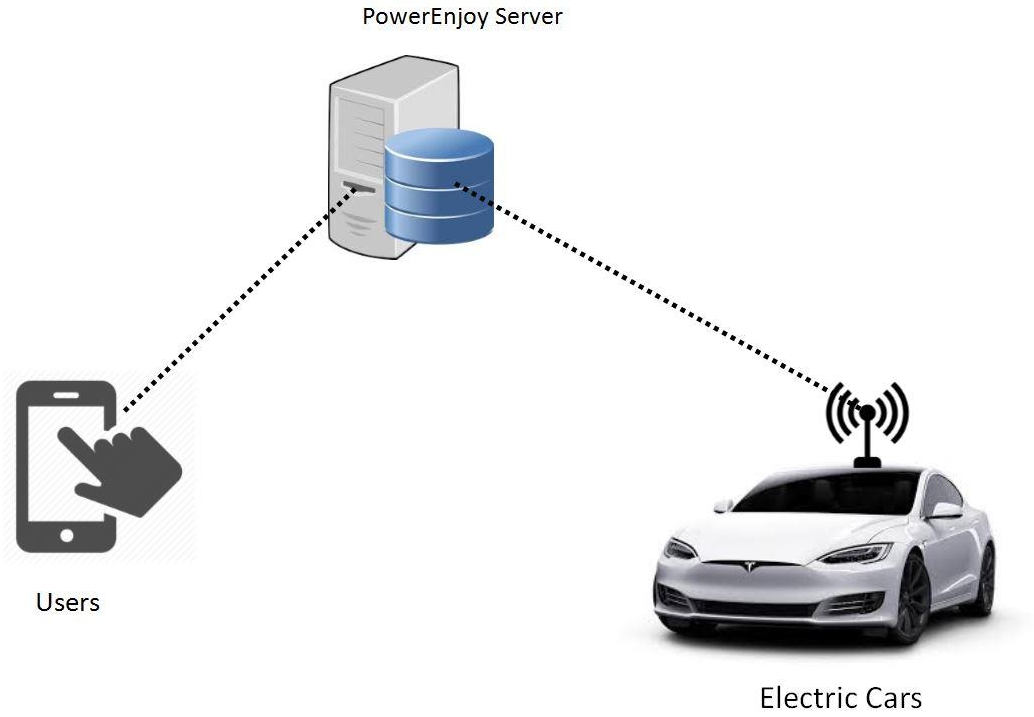
The web app will be available to all the major mobile operating systems and will be developed in a way to communicate with the PowerEnjoy servers through API and Http requests, consequently the front-end will lie on the server. This decision brings to a trade-off between the app performance and the overall flexibility of the system in terms of UI and functional requirements, since, in the case with the front-end inside the app, we would have to make the user update it every time PowerEnjoy proposes new functionalities. Moreover, by having the front-end on the server, it will be easier in the future to make a desktop front-end to be used in a web site.

We assume that at the moment the user base is small and therefore one server will be enough to serve everyone. However, due to the fact that the user base can become wider in a small amount of time, everything will be made in a way that implementing redundant arrays of them is easy.

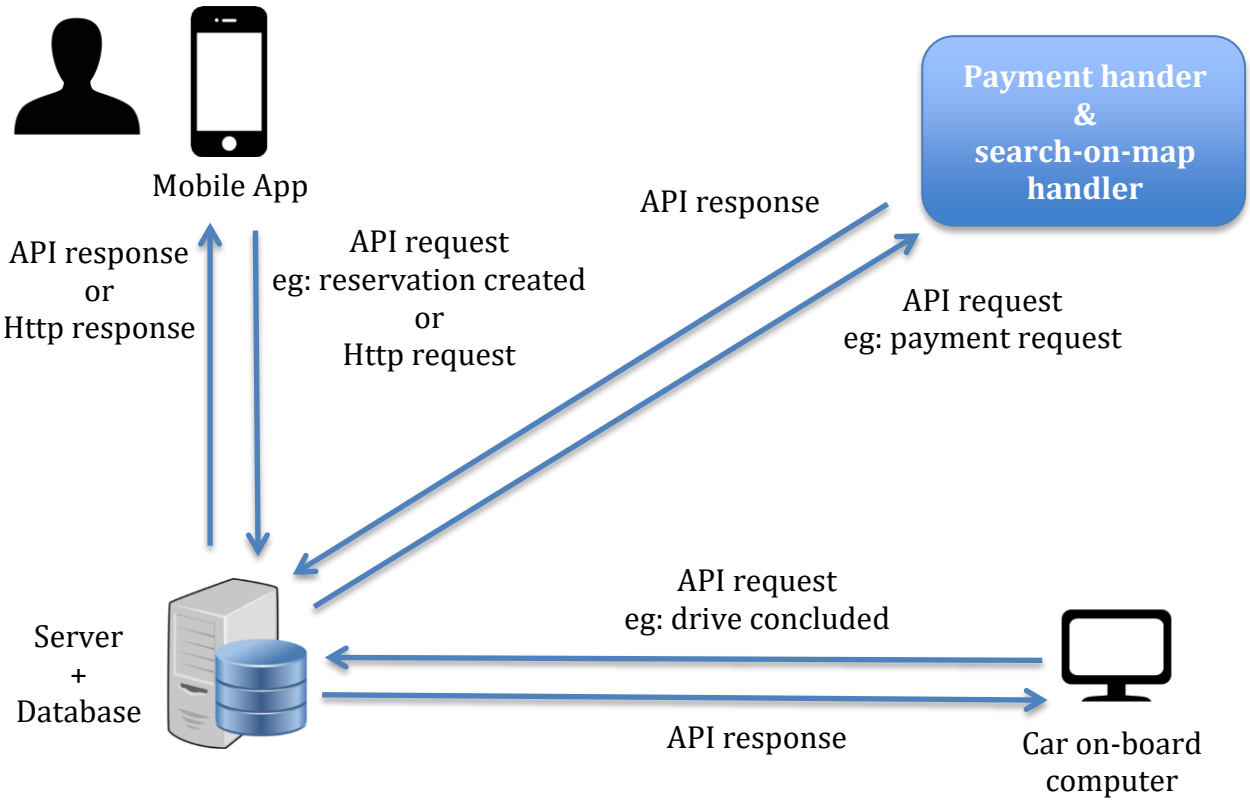
The database will stay on the same machine of the web server at the moment, but nothing will compromise the possibility to move it on another machine in the future.

Of course, the electric cars must be able to communicate with the server, so they must be provided with an Internet connection and an on-board computer that must be able to run Java software. However, the car is only an agent in our system and therefore all the business logic will lie on the server.

The proposed system can be summarized in a high-level way in the image in the following page:



As for the communications, the server will expose a RESTful API to the mobile app and the electric car.



2.8 Identifying stakeholders

Trivial stakeholders are:

- The company that serves the PowerEnjoy service (managers, employers, etc.);
- The customers of PowerEnjoy;
- Possible creditors/suppliers and shareholders, for example:
    - The car company that provides the cars that run the PowerEnjoy service;
    - The company that provides all the hardware needed for the system.

There are many entities that incentivize the use of electric vehicles, since they are less harmful to the environment:

- The government.
- The city in which the service is active (the society).

External service providers and suppliers are stakeholders too (indeed they are suppliers). They are:

- Payment handler.
- Search-on-map handler.

In addition, eventually we are stakeholders too, as software developers/engineers.

Stakeholder	**Needs**
PowerEnjoy	Provide the car sharing ecological service in an efficient, usable, reliable way to customers.
Customers	Use the service.
Possible creditors and shareholders	Get profit.
Government and society	More sustainable and ecological viability.
External service providers	Get profit.
Us	Get profit and the glory for creating an ecological car sharing service.

2.9 Reference documents

- Specification Document: Assignments AA 2016-2017.pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- Example documents:
    - RASD sample from Oct. 20 lecture.pdf
    - SWIMv2 RASD Example.pdf

3 Actors identifying

The main actor of PowerEnJoy™ system is:

- *User*: a person that has already registered and so has provided his personal information and payment method.

There is also another possible actor:

- *Guest*: a person that has not registered and can only perform basic functionalities such as looking for where safe areas are.

Eventually, there are two external actors:

- *Payment handler*: an external service that helps our system with all tasks related to the payments;
- *Search-on-map handler*: an external service that helps our system with all tasks related to the use of maps;

4 Requirements

Assuming that the domain properties stipulated hold, and, in order to fulfill the goals listed before, the following requirements can be derived.

4.1 Functional requirements

- [G1] *Users could see and select an available car close to him, or close to a specified address, and reserve it for up to one hour before they pick it up:*
  - The system has to detect if a car is parked in a Safe Area and its battery level;
  - The system has to detect car position and display it on a map;
  - The system has to be able to identify the location of a user through his/her GPS, if he/she gives the consent;
  - The system has to provide a list of available cars close to a given address;
  - The system has to give the possibility to reserve a car at most by one user at a time;
  - The system has to mark the reservation as expired for a car after one hour if the user has not picked it up;
  - The system has to apply a fee of 1€ if the reservation has expired;
- [G2] *Users could get in a car only if they are near it and they reserved it:*
  - The system has to be able to identify the location of the user and of the car;
  - The system has to unlock the car if the position of the user is really close to the one of the car;
- [G3] *Users should pay proportionally to minutes they have used the car, and they should see in real time the amount of the bill:*
  - The system has to reset trip information when a user get on the car
  - The system has to be able to understand when the car engine ignites;
  - The system has to start charging the user when the car engine ignites;
  - The system has to display the current charge;
  - The system has to identify when a car is parked in a safe area;
  - The system has to identify when there is no one sit in the driver's seat;
  - The system has to stop charging the user when the car is parked in a safe area and there is no one sat in the driver's seat;

- [G4] *Users could register to the system and have their personal area:*
    - The system has to provide log-in functionalities to the users;
    - The system has to provide sing-up form to users:
        - The system has to check that there are not two users with the same username;
        - The system has to store the password and personal information of every user;
        - The system has to provide the possibility to enter a payment method;
        - The system has to check if the payment method provided by the user is valid and usable;
    - The system has to provide the possibility to change personal information or payment methods even after the registration;
- [G5] *Virtuous behaviours by users should be incentivized:*
    - The system has to apply a discount of 10% on the final bill if there were at least three passengers on the last ride:
        - The system has to identify and store how many passengers there were on the car in the last ride;
    - The system has to apply a discount of 20% on the final bill if the car is left with at least 50% of battery level:
        - The system has to be able to identify the battery level of the car;
    - The system has to apply a discount of 30% on the final bill if the car is left plugged-in in a Special Safe Area:
        - The system has to identify if the car is plugged-in;
    - The system has to apply an extra-charge of 30% on the final bill if the car is left at least 3Km from the nearest Special Safe Area and the battery level is less than 30%:
        - The system has to be able to calculate the distance between the actual position of the car and the nearest Special Safe Area

4.2 Non-functional requirements

- The system has to be interoperable with the payment-handler in order to provide effectively payments function;
- The system has to be interoperable with the search-on-map service;
- The system has to be available 24 hours per day, 7 days per week, the same as the time required to develop effectively this fucking document;
- The system has to be available at least as an Android app;
- Modified data about availability of cars in a database have to be updated for all users accessing it within 2 seconds.
- Users' passwords have to be encrypted using SHA256 algorithm.

5 Scenario identifying

5.1 Scenario 1

Nick and his three best friends want to go out at night, but public transport is not serviceable at those hours. They do not want to spend a large amount of money, therefore they decide to take advantage of PowerEnjoy service and its discount.
Nick decides to plan the trip in order to achieve the maximum discount possible, that is by leaving the car in the Safe Area closest to the pub they want to go.
He opens the PowerEnjoy mobile app about one hour before going out, makes a reservation for the car and finds out the best place where to leave the car.
He and his friends can enjoy the night without spend too much.

5.2 Scenario 2

Riccardo and Fabio are two university students. Riccardo is excited about PowerEnjoy since he tried it for the first time and talks about it to his friend Fabio after the lesson. Fabio, who has always been a fond of environment-friendly companies, cannot wait to try PowerEnjoy electric cars and immediately downloads the app of the service. He decides to go home with an electric car, so he registers to the system with his credentials and payment information and makes a reservation for a car in the nearest possible point. Fabio reaches the car and drives it to the closest park to his home. The next day, Riccardo tells Fabio that he should have left the car in one of the special parking of PowerEnjoy with power grid stations so that he would have had a reduction on the fee. Fabio feels sad.

5.3 Scenario 3

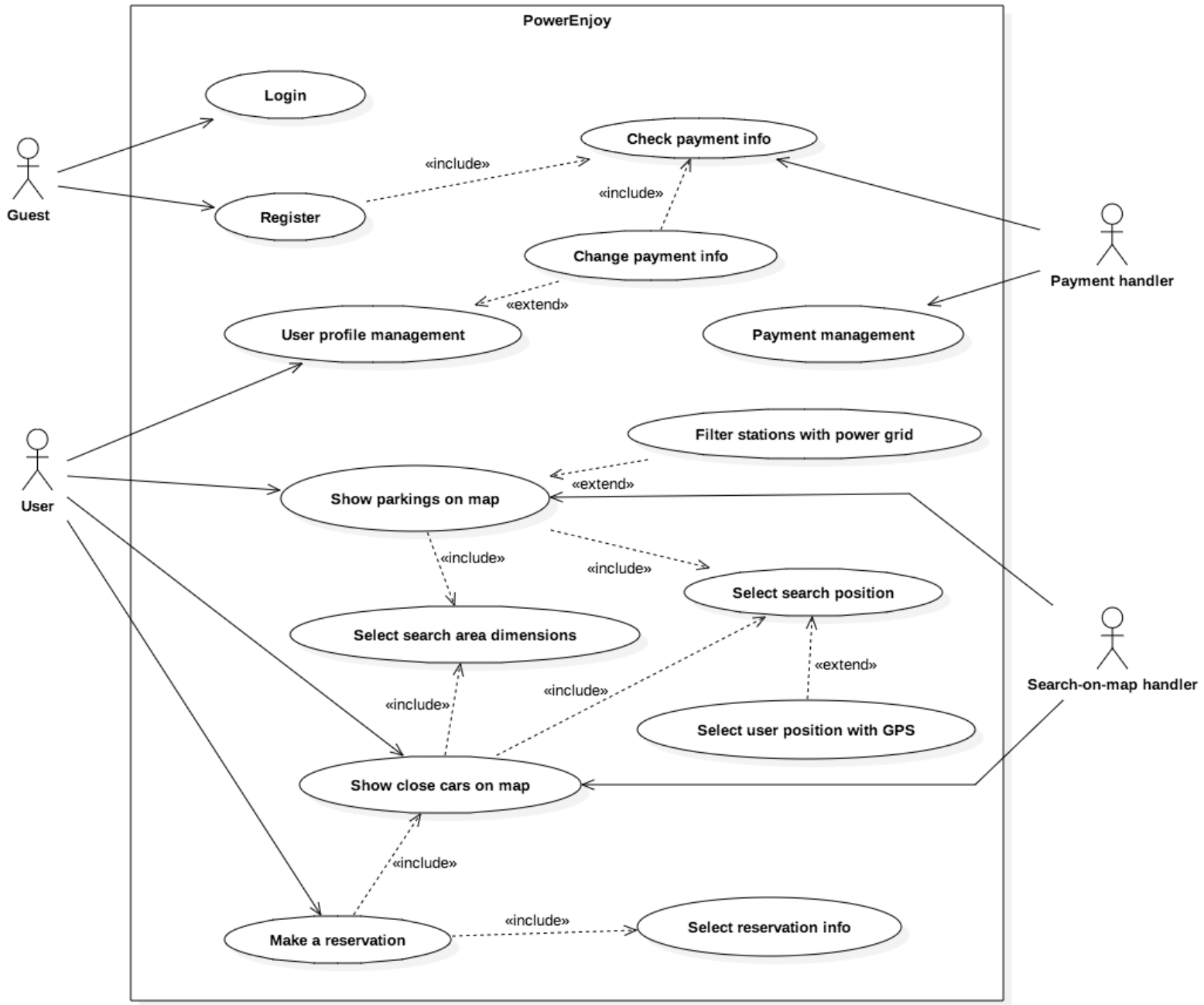
Agnese has just finished shopping. She bought a lot of things and she doesn't have the strength to bring home all her seven full shopping bags by hands. Agnese tried to call home to see if someone was there to help her but nobody answered. Therefore, she decides to use her favourite car-sharing app: PowerEnjoy. She takes her phone and looks for a car near her position. Unfortunately, the nearest car is farther than her house. Next time Agnese will buy less things, or will go shopping by car.

5.4 Scenario 4

Daniel has to meet his friend Adriana in one hour. Since Daniel has no cars, he makes a reservation for a PowerEnjoy electric car. He finishes seeing some episodes of his favourite telefilm and reaches the car he rented. However, the car does not open because Daniel saw to many episodes and more than an hour has passed. So he makes another reservation for a car that fortunately is available in that parking and takes it. The system charges him an extra fee.

6 UML models

6.1 Use case diagram



6.2 Use case description

Name	**Log-in**
Actors	Guest
Entry conditions	Guest has previously signed up to the system.
Flow of events	• The Guest opens the application of PowerEnJoy™ on his/her device; • The system shows him the login page; • The Guest enters his e-mail address and password in the input form provided; • The Guest clicks the button “log in”. • The system shows the map panel.
Exit conditions	Guest gets a confirmation message and is redirected to his/her personal page. He/she is now acknowledged by the system as a User.
Exceptions	The username and password provided by the Guest are not correct. The system notifies him/her that he/she has made an error and allows the guest to input his/her username and password again.

Name	**User changes payment info**
Actors	User
Entry conditions	User has already signed in to the system.
Flow of events	• The user clicks on “Change default Payment” button; • The system redirects the User to a form where he/she has to provide all the information about the new payment method; • The User sets Payment Data and clicks on Submit button; • The systems checks through the Payment Handler module if information provided are valid;
Exit conditions	When Payment Handler confirms the Payment Method provided, the system shows a confirmation message and redirects the User to his Personal Page.
Exceptions	The Payment Handler rejects the Payment Method provided. The system shows an error message and redirects the User to his Personal Page.

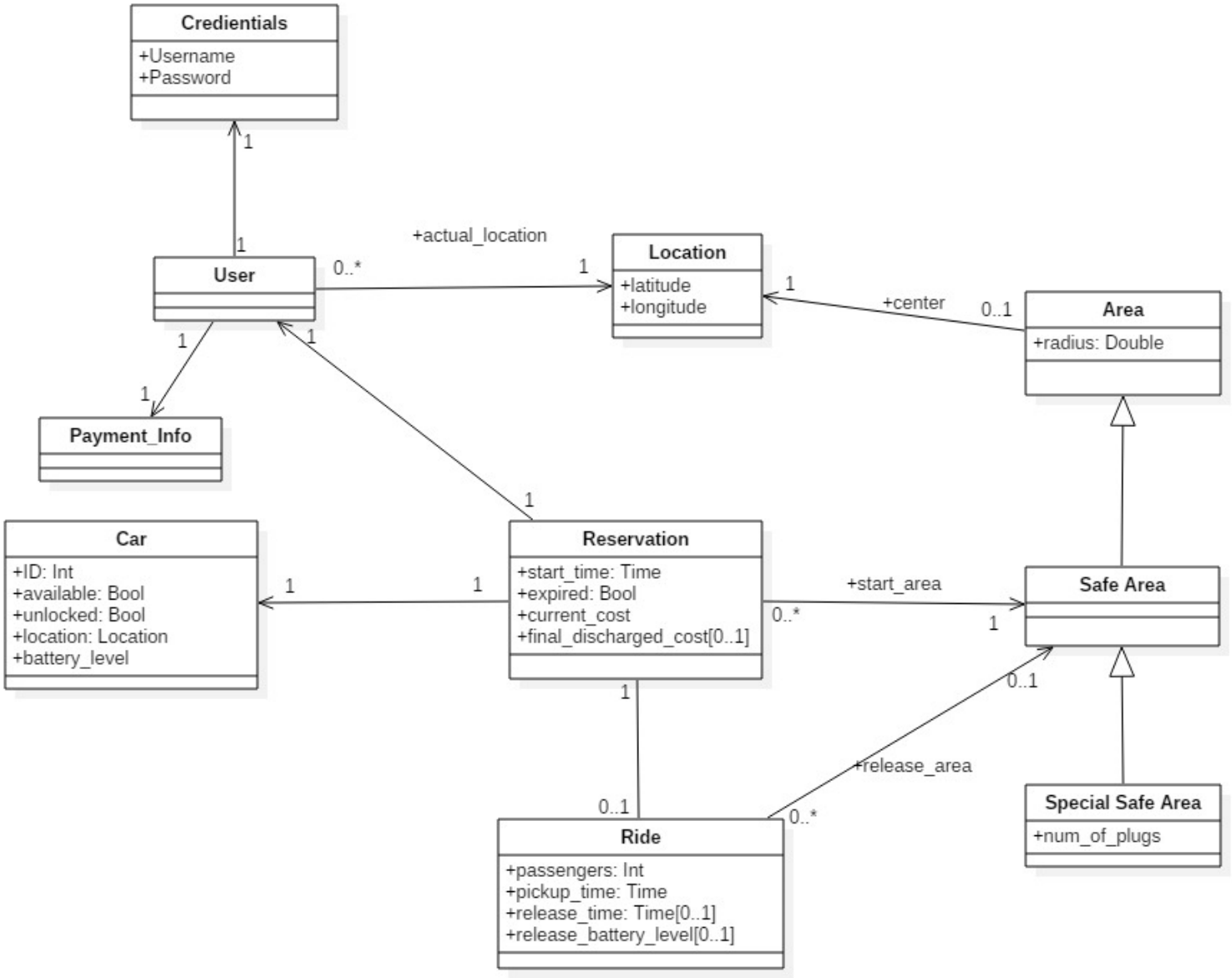
Name	**Show close cars on map through GPS**
Actors	User
Entry conditions	User has already signed in to the system.
Flow of events	• User clicks on “Show Map” button if he’s not in the map panel; • The system shows an updated map on the center of the page; • User clicks on “Find me by GPS” • The system acquires user’s GPS location and shows it by adding a colored dot on the map; • User puts a tick on “Show close available cars”; • The system adds on the map in the right position a little car image for every available car within a default range of distance between location of cars and location of the user; • User moves a slider in order to adjust this distance; • The system adjusts objects on the map according to user’s choice;
Exit conditions	There are no exit conditions.
Exceptions	There are no cars or Safe Areas within the above distance. The system displays a popup message to inform the user that PowerEnJoy™ service is not present in the area.

Name	**Show close Safe Areas with Power Grid on map**
Actors	User
Entry conditions	User has already signed in to the system.
Flow of events	• User clicks on “Show Map” button if he’s not in the map panel; • The system shows an updated map on the center of the page; • User fills the address field with a desired address; • The system shows a colored dot on the map in the provided address; • User puts a tick on “Show close Safe Areas with Power Grid”; • The system adds on the map in the right position an identifier for every available car within a default range of distance between

	location of cars and location of the user;
Exit conditions	There are no exit conditions.
Exceptions	There are no cars or Safe Areas within the above distance. The system displays a popup message to inform the user that PowerEnJoy™ service is not present in the area.

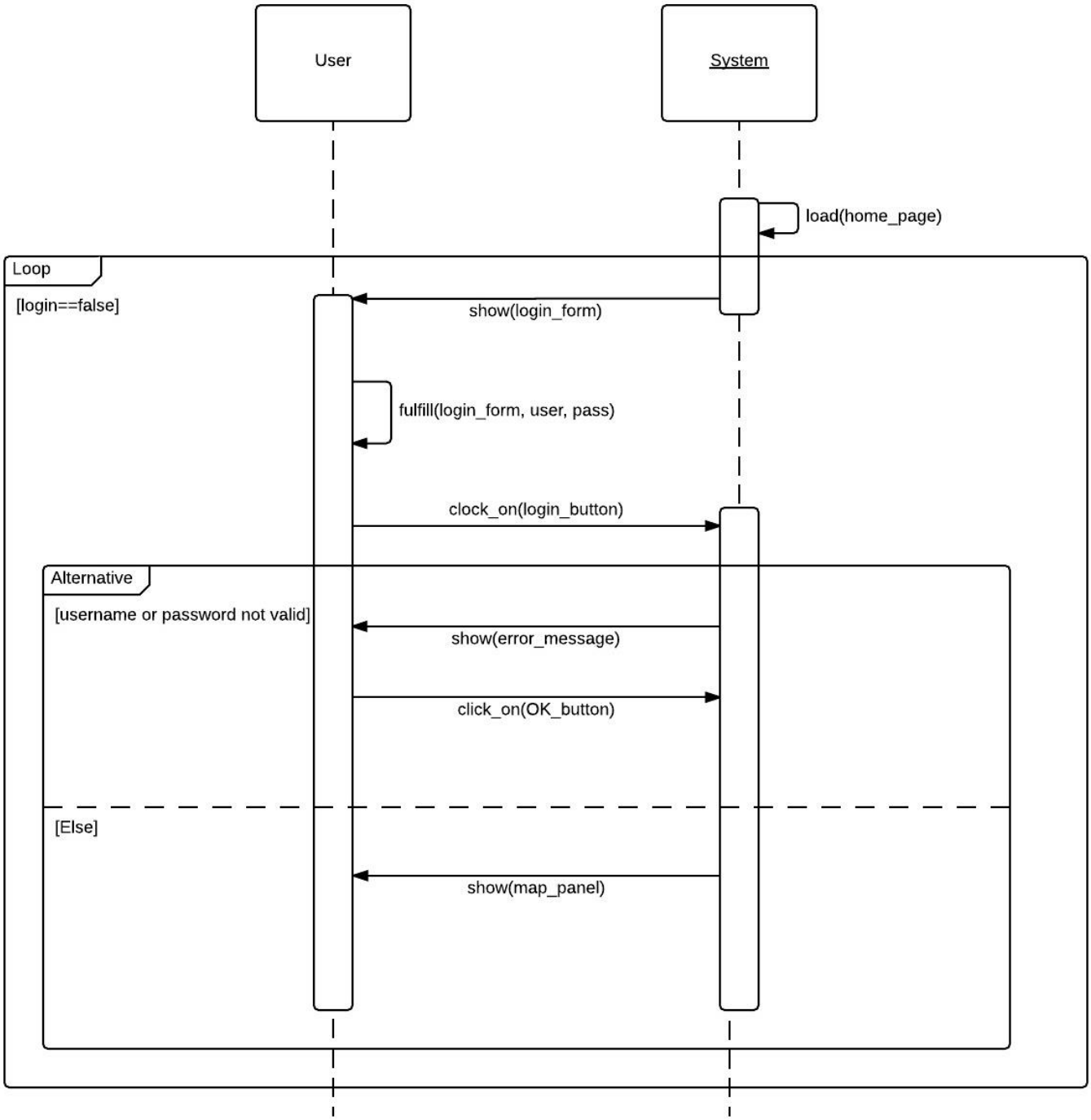
Name	**Make a reservation**
Actors	User
Entry conditions	User has already signed in to the system and is viewing available cars that may be of interest to him.
Flow of events	• User clicks on the car that best meets his/her needs; • The system shows a popup in which there are also information about the car; • User clicks on “Reserve this car”; • The system updates information of the car and marks it as reserved; • The system shows an information message about remaining time to pick-up the car.
Exit conditions	The User clicks OK, the system redirects him/her to the Personal Area.
Exceptions	Another user has reserved the car shortly before the user did. The system shows a message and redirects him/her to the map, displaying currently available cars.

6.3 Class diagram

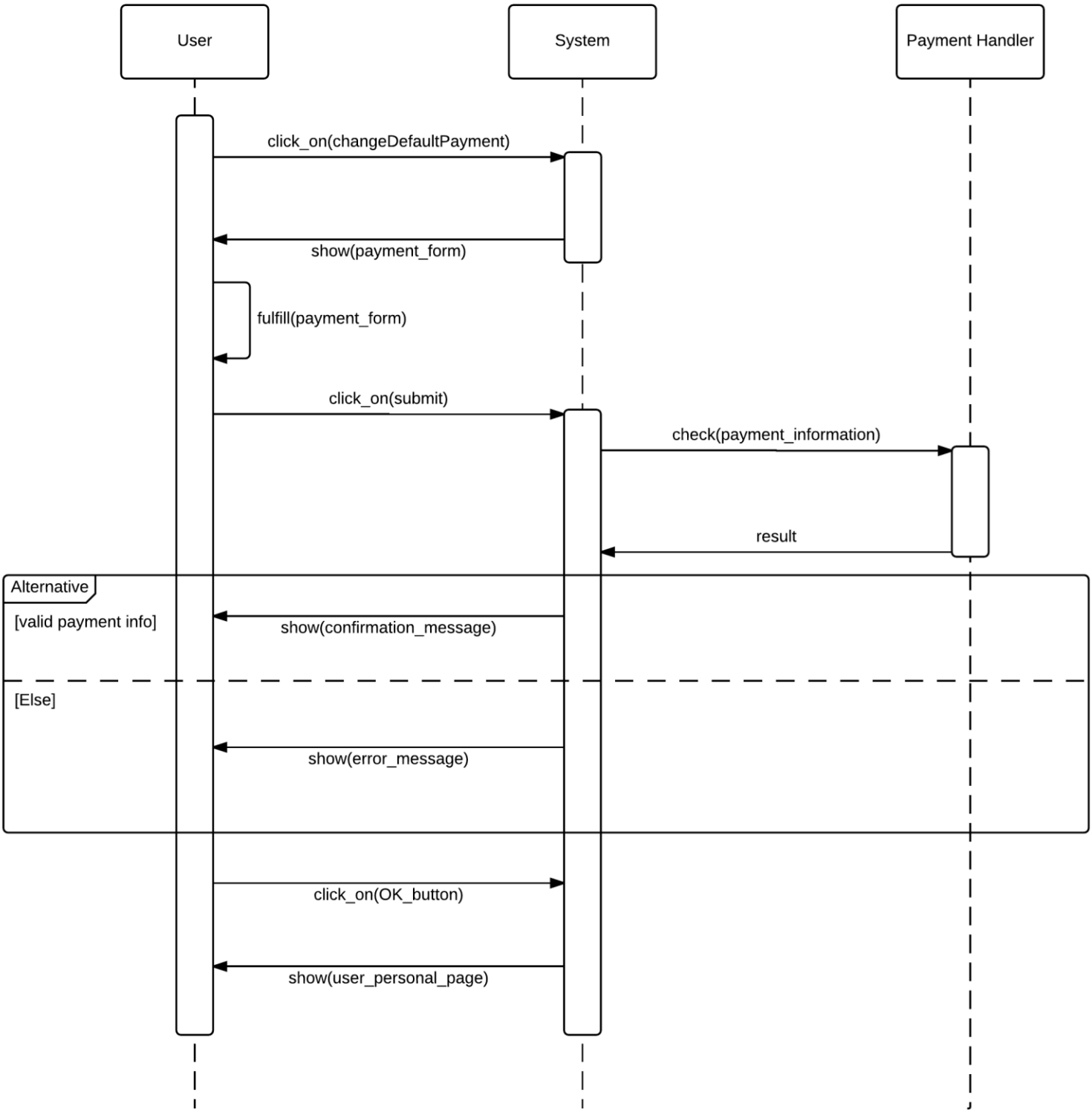


6.4 Sequence diagrams

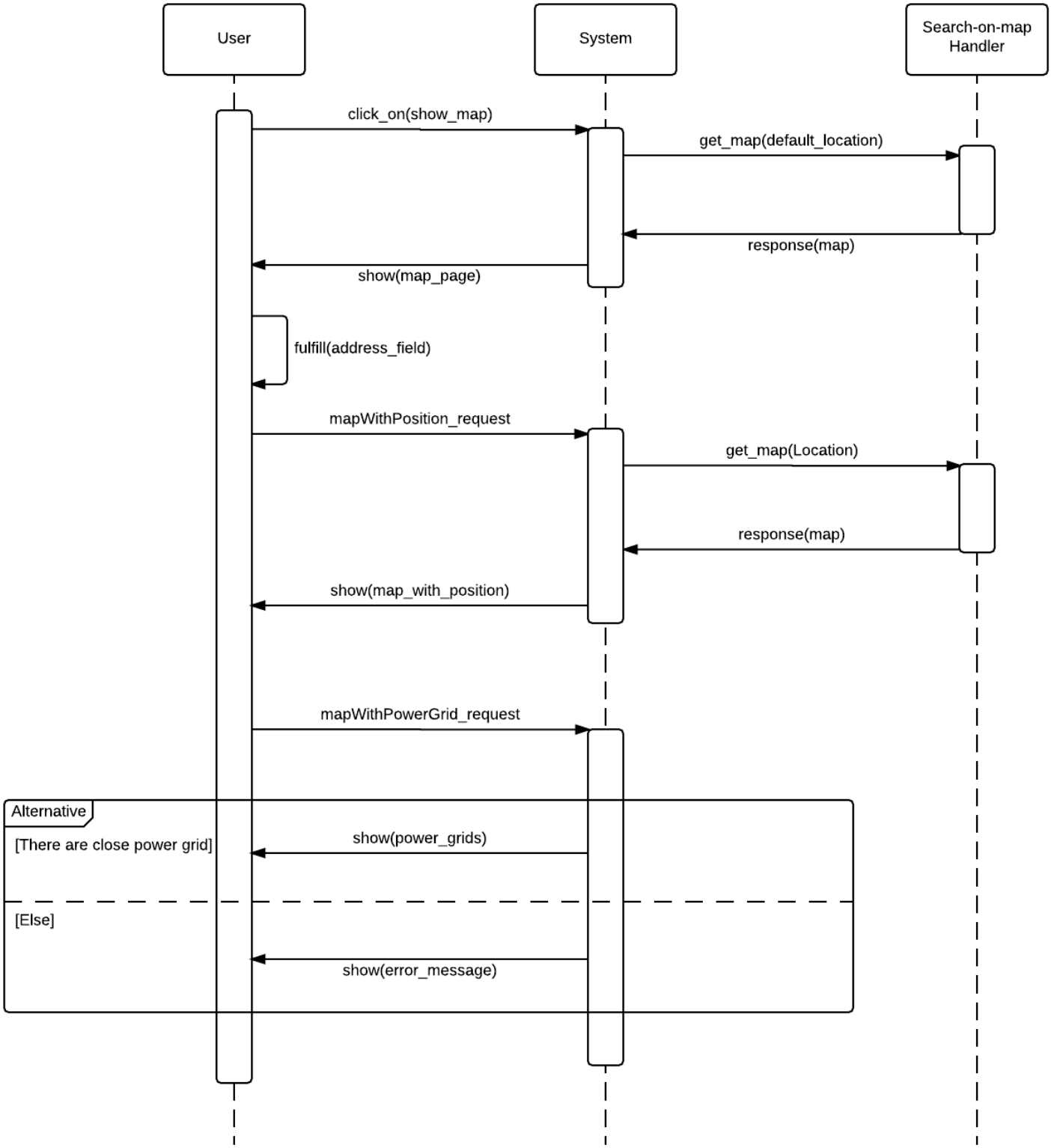
6.4.1 Log-In



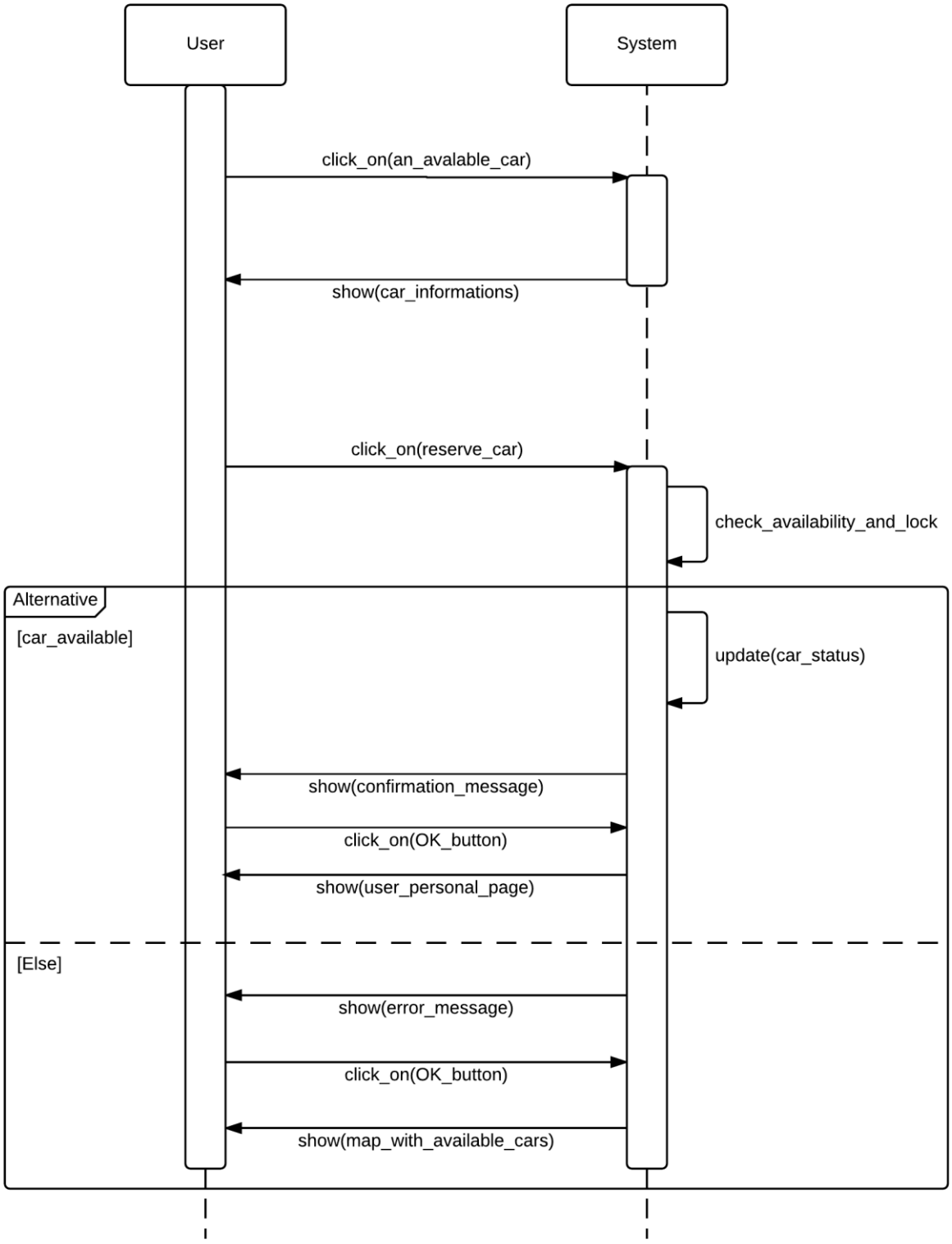
6.4.2 Change Payment Info



6.4.3 Show Special Safe Areas with Power Grid

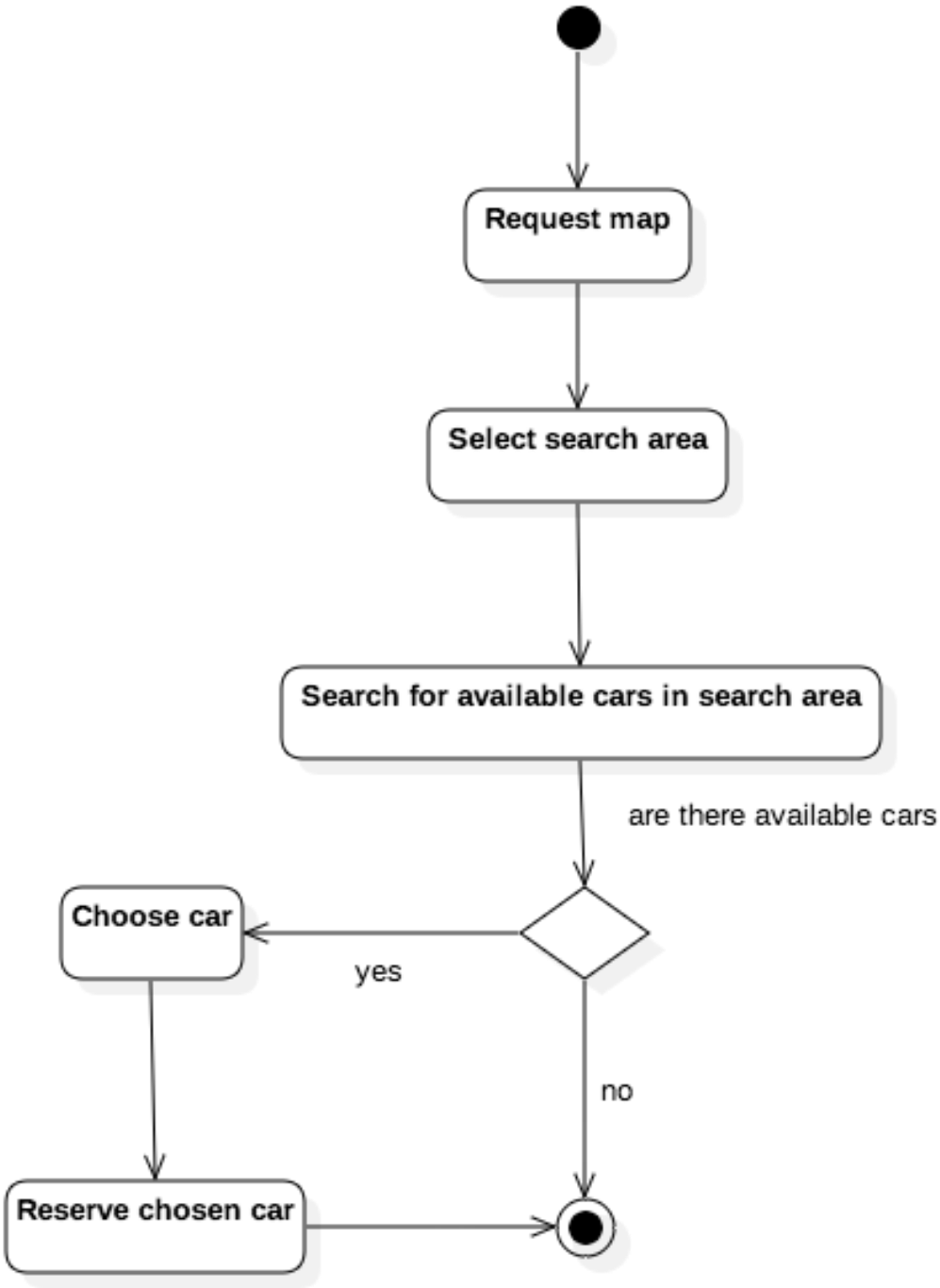


6.4.4 User makes a reservation



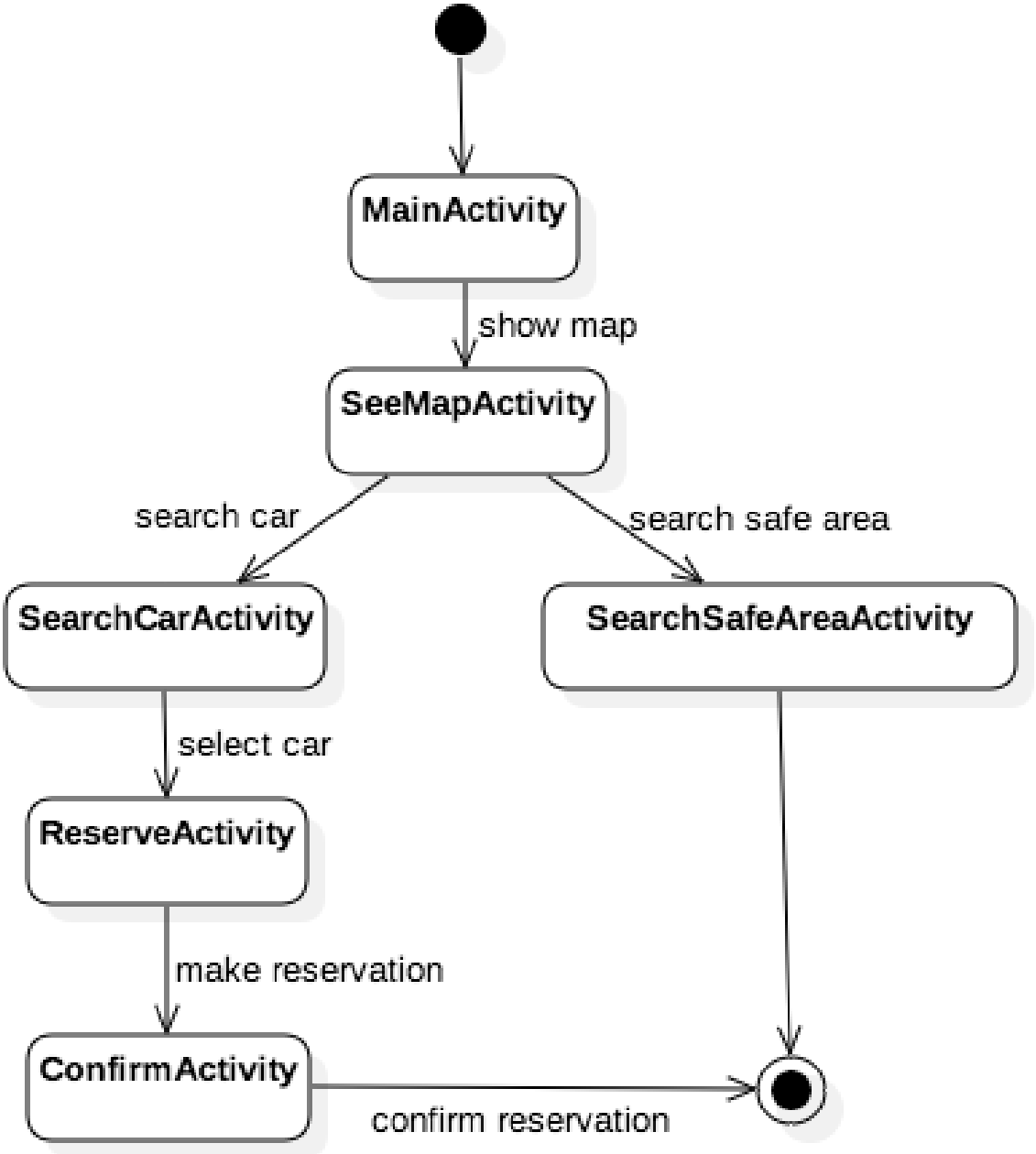
6.5 Activity diagrams

6.5.1 User makes a reservation

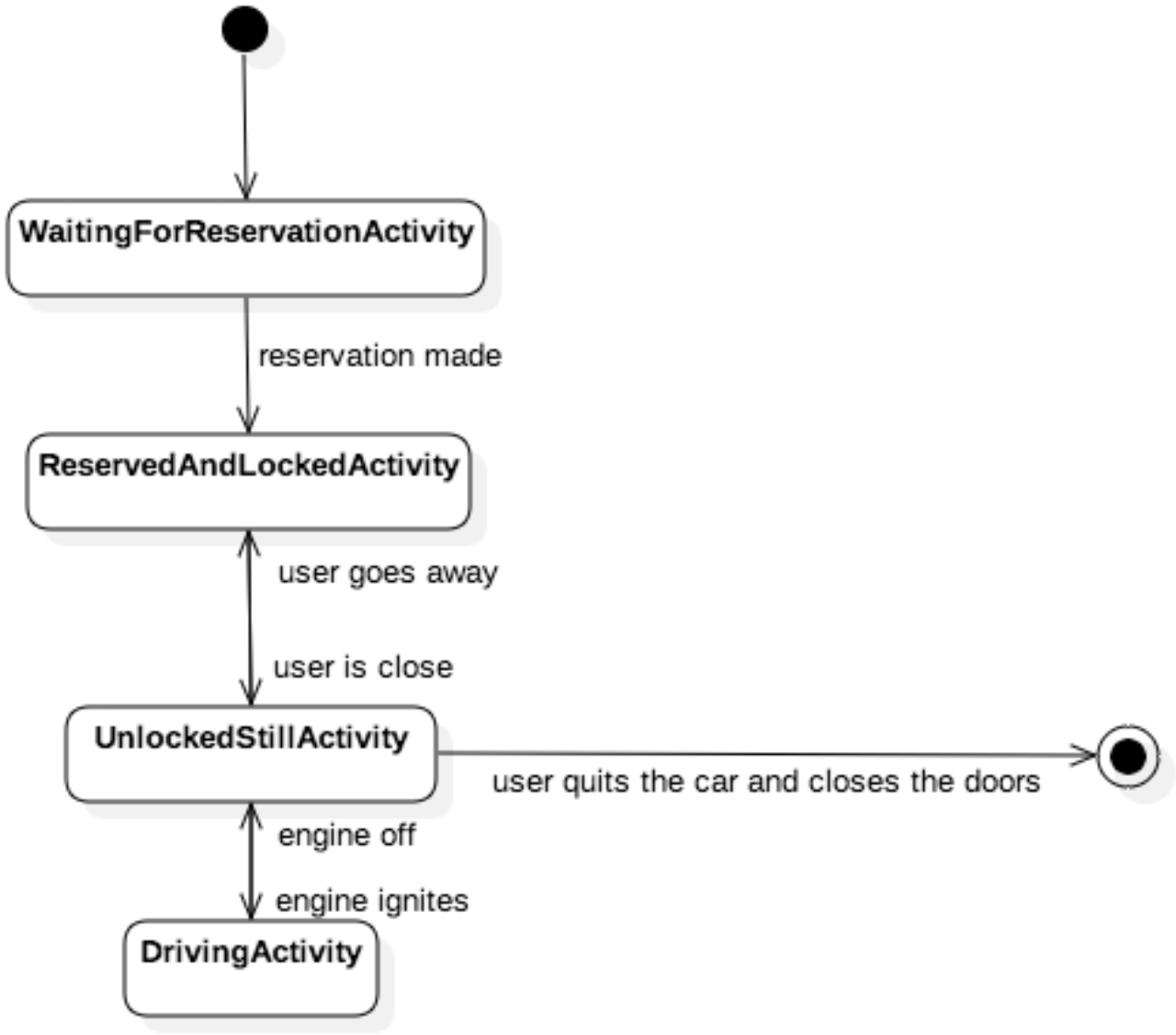


6.6 State diagrams

6.6.1 Mobile app states



6.6.2 Car on-board software states



7 Alloy modeling

7.1 Model

```
open util/boolean

// ----------- Signatures -----------
//A car is avalable if it can be reserved from the PowerEnjoy
//application, it is unlocked if the user who reserved it is close to it
//or he is driving.
sig Car {
	id: Int,
	available: Bool,
	location: Location,
	unlocked: Bool,
	battery_level: Int
} {
	id>0
	battery_level >= 0
	battery_level <= 10
}
//location represents where the user actually is
sig User {
	credential: Credential,
	payment_info: Payment_Info,
	location: Location
}

sig Credential {}
sig Payment_Info {}

//current_cost shows how much is the actual cost of a ride, without any
//discout or extra charge
//A reservation is expired if the user did not pick up the car within an
//hour. A fee is applied.
sig Reservation {
	user: User,
	car: Car,
	start_area: Safe_Area,
	start_time: Time,
	current_cost: Int,
	final_discharged_cost: lone Int,
	expired: Bool,
	ride: lone Ride
} {
	current_cost >= 0
	final_discharged_cost >= 0
}

sig Ride {
	reservation: Reservation,
	passengers: Int,
	pickup_time: Time,
	release_time: lone Time,
	release_battery_level: lone Int,
	release_area: lone Safe_Area,
```

```
      release_plugged: lone Bool
} {
      passengers >= 1
      passengers <= 4
      release_battery_level >= 0
      release_battery_level <= 10
      not areEqual[pickup_time, release_time]
}

// Time represents a generic moment of the day before the CurrentTime
sig Time {
      hour: Int,
      minute: Int
} {
      hour >= 0
      hour =< 23
      minute >= 0
      minute =< 59
}
sig CurrentTime extends Time {}

// Location represents a point in the geographic area. An Area
//represents a special location with some particular feature.
sig Location {
      latitude: Int,
      longitude: Int
}
sig Area {
      center: Location,
      radius: Int
}
//A Safe Area is a place where a car can be left.
sig Safe_Area extends Area {}
//A Special Safe Area is a Safe Area with power grids, where an electric
//car can be left plugged-in and charging.
sig Special_Safe_Area extends Safe_Area {}

/* ----------- Functions ------------ */
//Returns the absolute difference between minutes of two Times
fun minutesDiff[t, t': Time]:Int {
      t.minute>t'.minute
      implies
            sub[t.minute, t'.minute]
      else
            sub[t'.minute, t.minute]
}
//Returns the absolute difference between hours of two Times
fun hoursDiff[t, t': Time]: Int {
      t.hour>=t'.hour
      implies(
            sub[t.hour, t'.hour]
      ) else (
            sub[t'.hour, t.hour]
      )
}

//Returns the cost for a ride from time t to time t'
fun cost[t, t': Time]: Int {
```

```
            t.hour = t'.hour
            implies (
                  minutesDiff[t, t']
            )
            else (
                  t.hour>t'.hour
                  implies(
                        add[add[sub[60, t'.minute],
                        mul[sub[hoursDiff[t,t'], 1], 60]], t.minute]
                        //(60-t'.minute) + (t'.hour-t.hour-1)*60 + t.minute
                  )
                  else(
                        add[add[sub[60, t.minute], mul[sub[hoursDiff[t,t'],
                        1], 60]], t'.minute]   //(60-t.minute) + (t.hour-
                        t'.hour-1)*60 + t'.minute
                  )
            )
}

/* ----------- Predicates ----------- */
// Returns true if t == t'
pred areEqual[t, t': Time] {
      t.hour = t'.hour and t.minute = t'.minute
}

// Returns true if t - t' <= 1h
pred isLessThanOneHourAhead[t, t': Time] {
      areEqual[t, t']
      or
            (t.hour = t'.hour and t.minute > t'.minute)
      or
            (t.hour = add[t'.hour,1] and add[sub[60, t'.minute], t.minute]
            < 60)
}

// Returns true if t >= t'
pred comesAfterOrEqual[t, t': Time] {
            t.hour = t'.hour
            implies t.minute >= t'.minute
            else t.hour >= t'.hour
}
// Returns true if the max distance between latitudes and longitudes is
//4
pred isNear[l, l': Location] {
      (l.latitude >= l'.latitude implies
            sub[l.latitude, l'.latitude] <= 4
      else
            sub[l'.latitude, l.latitude] <= 4)
      and
      (l.longitude >= l'.longitude implies
            sub[l.longitude, l'.longitude] <= 4
      else
            sub[l'.longitude, l.longitude] <= 4)
}
// Returns true if the min distance between latitudes and longitudes is
//15
pred isFarFromSpecialSafeArea[area: Safe_Area] {
                  all ssa: Special_Safe_Area |(
```

```
                (int ssa.center.latitude >= int area.center.latitude
                implies
                      sub[ssa.center.latitude, area.center.latitude] > 15
                else
                      sub[area.center.latitude, ssa.center.latitude] > 15
                )
                and
                (int ssa.center.longitude >= int area.center.longitude
implies
                      sub[ssa.center.longitude, area.center.longitude] >
                      15
                else
                      sub[area.center.longitude, ssa.center.longitude] >
                      15)
      )
}

// Reservation r is open <=> r is not expired and "it has not a ride yet
//or its ride has no release_time
pred isActive[r: Reservation] {
      r.expired = False and (
            no r.ride
            or
            no r.ride.release_time
      )
}

/* ----------- Facts ------------- */
// There's only a current time
fact currentTimeForeverAlone {
      #CurrentTime = 1
}

// CurrentTime is after all the times in the model
fact currentTimeIsAlwaysAhead {
      all t: Time |
            all ct: CurrentTime |
                  comesAfterOrEqual[ct, t]
}

// Each credential/password/payment info is used for a user
fact eachCredentialCorrespondToAUser {
      User.credential=Credential
}
fact eachPaymentInfoCorrespondToAUser {
      User.payment_info=Payment_Info
}

// Different users <=> different credentials
fact noUsersWithSameCredentials {
      all u1, u2: User |
            u1 != u2
            iff
            u1.credential != u2.credential
}

// Different cars <=> different IDs
fact noCarsWithSameIDs {
```

```
      all c1, c2: Car |
            c1 != c2
            iff
            c1.id != c2.id
}

// The relation between a Reservation and a Ride is bijective, if the
//Reservation already has it.
fact ridesAndReservationRelation {
      all rid: Ride | (
            rid = rid.reservation.ride
      ) and
      all res: Reservation | (
            one res.ride
            implies
                  res=res.ride.reservation
      )
}

//A car can be released plugged only if it is released in a Special Safe
//Area
fact pluggedOnlyInSafeArea{
      all rid: Ride |(
            rid.release_plugged=True
            implies
            (rid.release_area in Special_Safe_Area)
      )
}

// There can't be two reservations r1 and r2 for the same car
//overlapping.
fact carsCanBeReservedByOneUserAtOnce {
      all c: Car | (
            all r, r': Reservation | (
                  (r.car = c and r'.car = c and r != r' and
comesAfterOrEqual[r'.start_time, r.start_time])
                  implies (
                              (some r.ride and some r.ride.release_time and
comesAfterOrEqual[r'.start_time, r.ride.release_time])
                        or    (r.expired = True and not
                        isLessThanOneHourAhead[r'.start_time,
                        r.start_time])
                  )
            )
      )
}

//A user can reserve a car only if it has a battery level grater than
//80%
fact carReservableCondition {
      all r: Reservation | (
            (no r.ride and r.expired=False)
            implies
            r.car.battery_level>8
      )
}
```

```
// A car is available if it has no reservations open and has enough
//battery
fact carAvailableCondition {
      all c: Car |
            c.available = True
            iff(
            (no res: Reservation |
                  res.car = c and isActive[res])
            )
}

// A reservation is expired if there is not a ride whose pickup_time is
//less than one hour after the reservation start_time.
fact reservationExpiredCondition {
      all res: Reservation |
            res.expired = True
            iff
            (all currentTime: CurrentTime |
                  not isLessThanOneHourAhead[currentTime, res.start_time]
            )
            and
            no res.ride
}

// The pickup time of a ride should be in one hour from its reservation
//start_time
fact pickupTimeConstraint {
      all rid: Ride | (
            isLessThanOneHourAhead[rid.pickup_time,
rid.reservation.start_time]
      )
}

//If a user is driving the reserved car, the location of the user and
//the car are the same.
fact userLocationSameAsCar{
      all res: Reservation |(
            (isActive[res] and one res.ride)
            implies
            (res.user.location = res.car.location)
      )
}

// release_time is always after pickup_time
fact releaseTimeIsAfterPickupTime {
      all rid: Ride | (
            no rid.release_time
            or
            comesAfterOrEqual[rid.release_time, rid.pickup_time]
      )
}

// A car is unlocked if exists an active reservation for it whose user
is near the car
fact carUnlockedConstraint {
      all c: Car | (
            c.unlocked = True
            iff
```

```
            some r: Reservation | (
                  r.car = c and isActive[r] and isNear[c.location,
                  r.user.location]
            )
      )
}

//If a car is locked and not plugged to a power grid, its battery level
//is the same as it has been left after last ride
fact batteryLevelConstraint{
      all res: Reservation | (
            (no res': Reservation | (res!=res' and res.car=res'.car and
comesAfterOrEqual[res'.start_time, res.start_time]))
            implies (
                  (res.ride.release_plugged=False and
res.car.location=res.ride.release_area.location)
                  implies
                        res.ride.release_battery_level =
res.car.battery_level
            )
      )
}

//If there is a release time, there are also a release release_battery
//and a release_area
fact releaseInfoConstraint{
      all rid: Ride| (
            (some rid.release_time
            iff
            some rid.release_battery_level) and
            (some rid.release_area
            iff
            some rid.release_time) and
            (some rid.release_time
             iff
            some rid.release_plugged)
      )
}
//If a Reservation is expired without the user picked up the car, he/she
//has to pay a fee of 10.
fact feeConstraint {
      all r: Reservation | (
            r.expired = True
            implies
            r.current_cost = 10 and r.final_discharged_cost = 10
      )
}

//The actual cost of a reservation not expired is proportional to
minutes spent in //the car
fact costOfAReservationConstraint {
      all res: Reservation | (
            res.expired = False
            implies (
                  no res.ride
                  implies
                        res.current_cost=0
                  else(
```

```
                        no res.ride.release_time
                        implies(
                              all ct : CurrentTime |
                                    res.current_cost=cost[ct,
                                    res.ride.pickup_time]
                        )
                        else
                              res.current_cost=cost[res.ride.release_time,
                              res.ride.pickup_time]
                  )
            )
      )
}

//The final discherged cost corresponds to the final current cost with a
//discount or an extra fee
fact finalDischargedCostCalculation {
      all res: Reservation | (
            (one res.ride.release_time and res.ride.passengers>2)
            implies(               //Enabled to 10% discount
                  (res.expired = False and
res.ride.release_battery_level>=5)
                  implies(         //Enabled to 10%+20% discount
                        (res.ride.release_plugged = True)
                        implies(  //Enablet to 10%+20%+30% discount
                              res.final_discharged_cost=div[mul[res.current
                              _cost, 2],5]
                        )
                        else(       //Enabled to 10%+20% discount
                              res.final_discharged_cost=div[mul[res.current
                              _cost, 7],10]
                        )
                  )
                  else(
                        (res.ride.release_plugged = True)
                        implies(  //Enablet to 10%+30% discount
                              res.final_discharged_cost=div[mul[res.current
                              _cost, 3],5]
                        )
                        else(//Enabled to 10% discount, but 30% extra fee

      (isFarFromSpecialSafeArea[res.ride.release_area] and
res.ride.release_battery_level<3)
                              implies (

      res.final_discharged_cost=div[mul[res.current_cost, 6],5]
                              )
                              else(

      res.final_discharged_cost=div[mul[res.current_cost, 9],10]
                              )
                        )
                  )
            )
            else(
                  (res.expired = False and
res.ride.release_battery_level>=5)
                  implies(         //Enabled to 20% discount
```

```
                        (res.ride.release_plugged = True)
                        implies(  //Enablet to 20%+30% discount

      res.final_discharged_cost=div[res.current_cost,2]
                        )
                        else(       //Enabled to 20% discount
                              res.final_discharged_cost=div[mul[res.current
                              _cost, 4],5]
                        )
                  )
                  else(
                        (res.ride.release_plugged = True)
                        implies(  //Enablet to 30% discount
                              res.final_discharged_cost=div[mul[res.current
                              _cost, 7],10]
                        )
                        else(       //There are no discounts

      (isFarFromSpecialSafeArea[res.ride.release_area] and
res.ride.release_battery_level<3)
                              implies ( //Extra-charging of 30%

      res.final_discharged_cost=div[mul[res.current_cost, 13],10]
                              )
                              else( // No discounts orr extra-charging

      res.final_discharged_cost=res.current_cost
                              )
                        )
                  )
            )
      )
}




// ------------- Assertions ---------------
assert availabilityAndLockingChecking {
      all c: Car | ((c.available = True implies c.unlocked = False)
              // car available => car locked
                                    and (c.unlocked = True implies
c.available = False))   // car unlocked => car not available
}


// reservation expired => no ride
assert expiredEntailsNoRide {
      all r: Reservation | r.expired = True implies (no r.ride and
r.final_discharged_cost=10)
}

// reservation active => car not available and there aren't any other
active reservations for that car
assert activeEntailsNotAvailable {
      all r: Reservation | isActive[r] implies (r.car.available = False
```

```
          and no r': Reservation | r'!=r and isActive[r'] and
r.car=r'.car)
}

// ------------ Show world! --------------
pred showCarLockedAndCarUnlocked {
      some c: Car | c.unlocked = False
      some c: Car | c.unlocked = True
}

pred showCarAvailableAndCarNotAvailable {
      some c: Car | c.available = False
      some c: Car | c.available = True
}

pred showReservationsTypes {
      some r: Reservation | r.expired = True
      some r: Reservation | isActive[r]
      some r: Reservation | not isActive[r] and r.expired = False
}

check availabilityAndLockingChecking for 8 Int
check activeEntailsNotAvailable for 8 Int
check expiredEntailsNoRide for 8 Int
run showCarLockedAndCarUnlocked for 8 Int
run showCarAvailableAndCarNotAvailable for 8 Int
run showReservationsTypes for 8 Int
```

7.2 Alloy result

```
6 commands were executed. The results are:
  #1: No counterexample found. availabilityAndLockingChecking may be valid.
  #2: No counterexample found. activeEntailsNotAvailable may be valid.
  #3: No counterexample found. expiredEntailsNoRide may be valid.
  #4: Instance found. showCarLockedAndCarUnlocked is consistent.
  #5: Instance found. showCarAvailableAndCarNotAvailable is consistent.
  #6: Instance found. showReservationsTypes is consistent.
```

Assertions explanation:

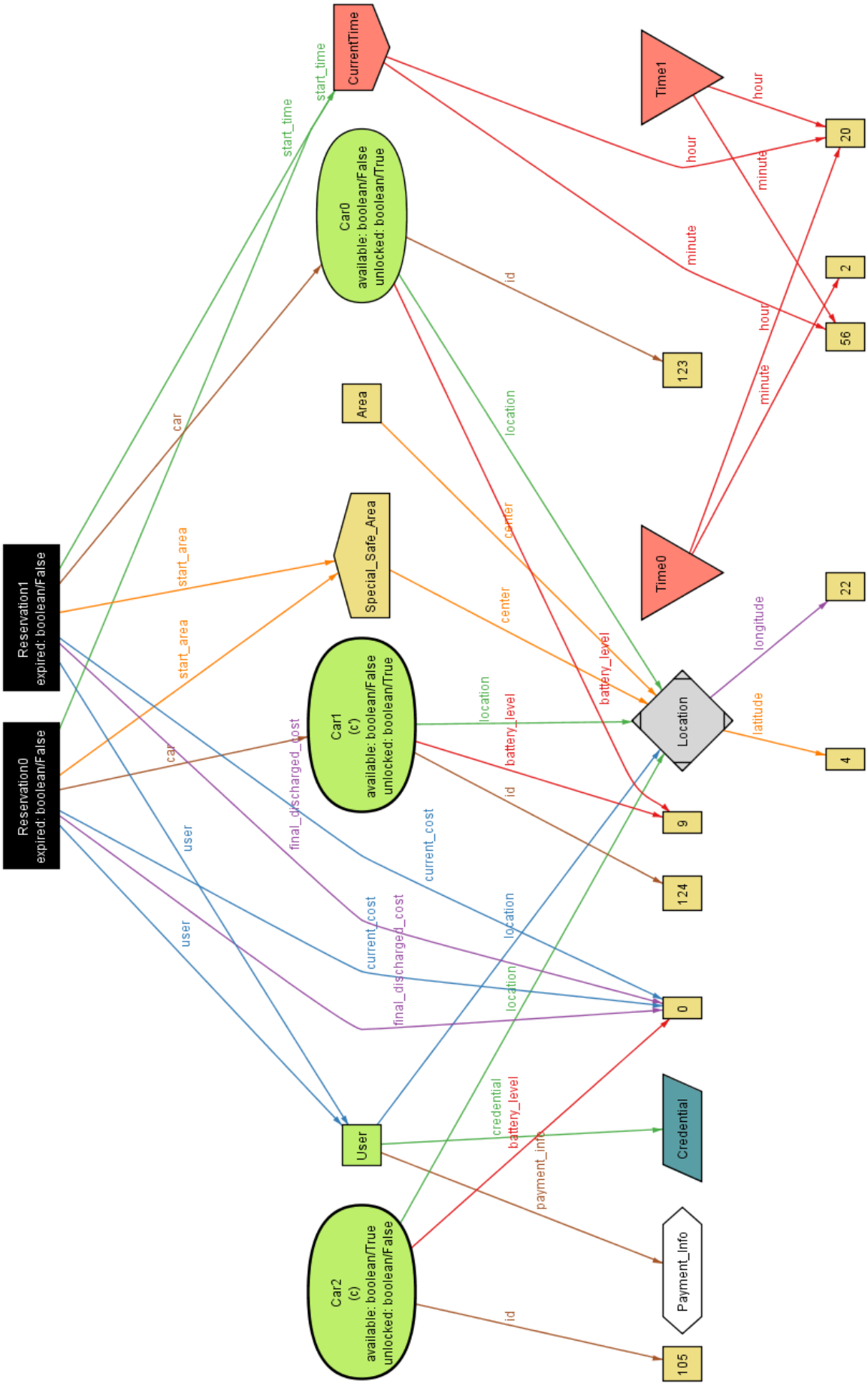
- *availabilityAndLockingChecking*: checks that:
  - If a car is available, then it's locked;
  - If a car is unlocked, then it's not available.
- *activeEntailsNotAvailable*: checks that if a reservation for a car is active, then the car is not available and there aren't any other active reservations for that car;
- *expiredEntailsNoRide*: checks that, if a reservation has expired, then there is no ride associated with it;

Worlds explanation:

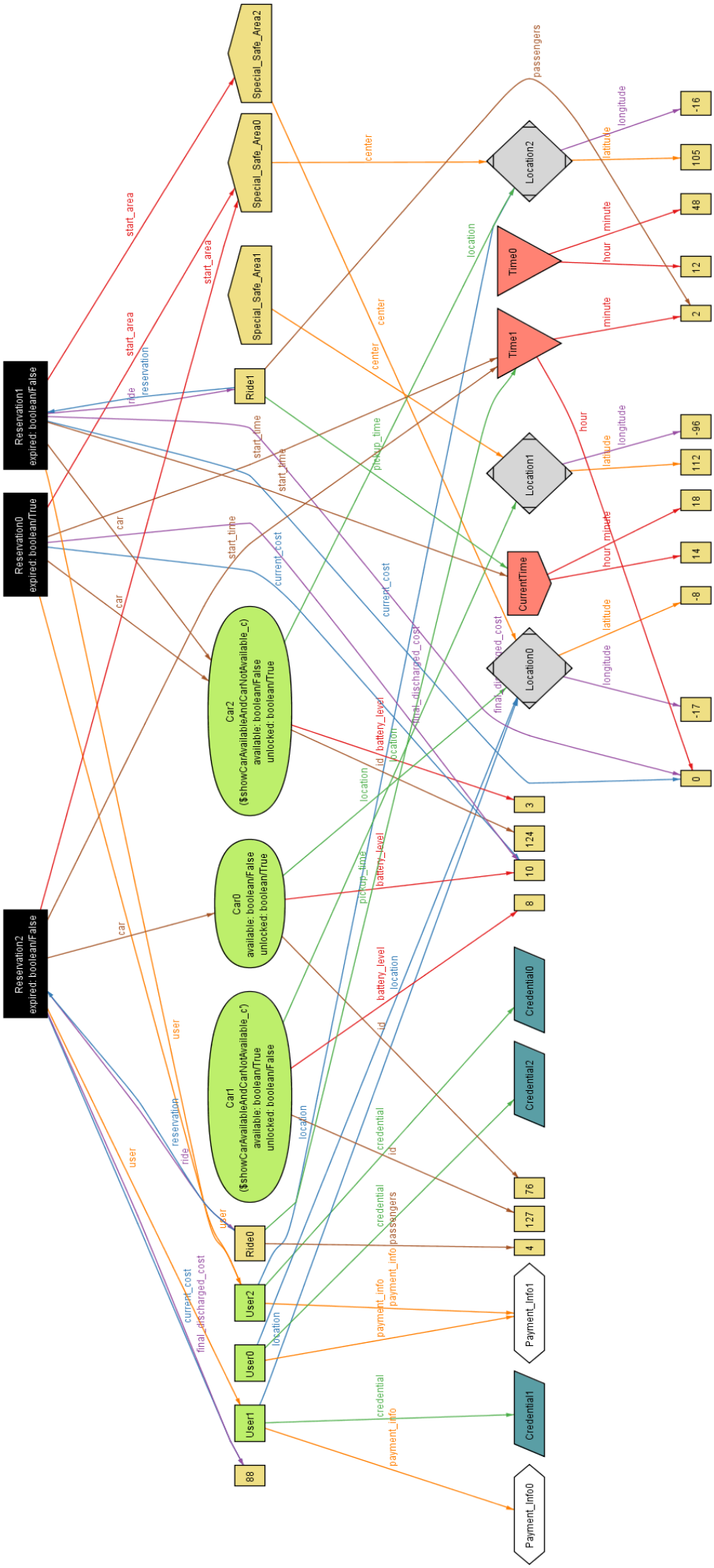
- *showCarLockedAndCarUnlocked*: shows a world that contains at least one car locked and at least one car unlocked;
- *showCarAvailableAndNotAvailable*: shows a world that contains at least one car available and at least one car not available;
- *showReservationsTypes*: show a world that contains at least:
  - a reservation expired;
  - a reservation not expired and not concluded yet;
  - a reservation not expired and concluded;

7.3 Worlds generated

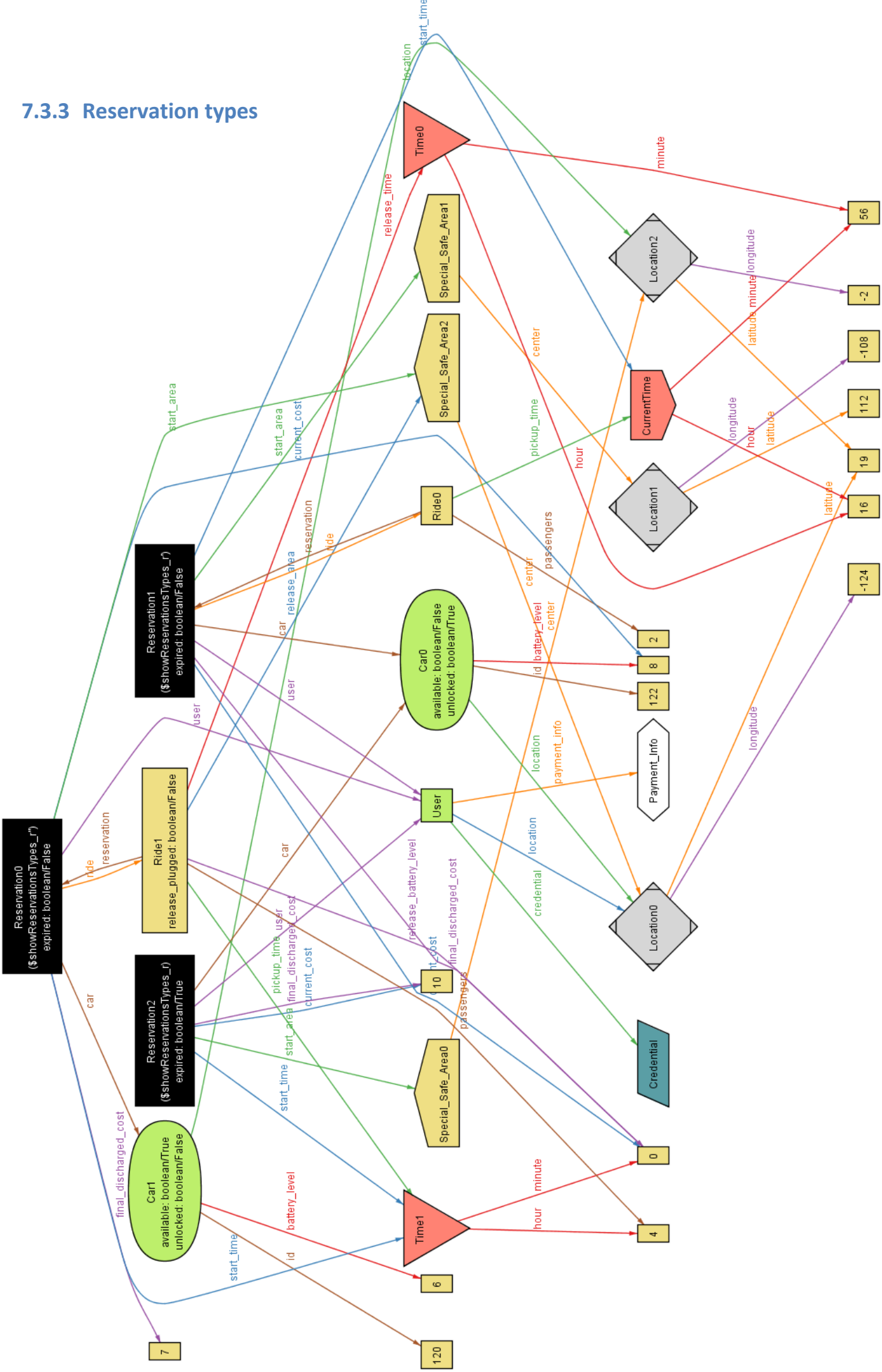
7.3.1 At least one car locked and one car unlocked



7.3.2 At least one car available and one car not available



7.3.3 Reservation types



8 Future development

There are a lot of possible improvements in the system-to-be:

- Accident management;
- Push notifications or SMS for reservation timer alerts;
- Add a web site (create desktop front-end with the help of the mobile front-end);
- Different type of cars and different fees for them;
- Share car reservation info with friends through mail, SMS or Whatsapp.

9 Used tools

The tools used for this document are:

- Microsoft Word: for text editing;
- Github, Git: for version control and synchronization between team members;
- StateChart.com: to create Sequence UML Diagrams
- StarUML: for creating other UML Diagrams;
- Alloy Analyzer 4.2: to prove consistency of our model;

10 Hours of work

We managed to distribute the workload fairly between days and team members in a way that allowed us to finish a few days before the deadline and have time for an accurate check in the last days.
We worked together most of the times and trying to do something every day, as it can be shown by Git commits.
The total amount of time required to build this document is about 37 hours of work per person.