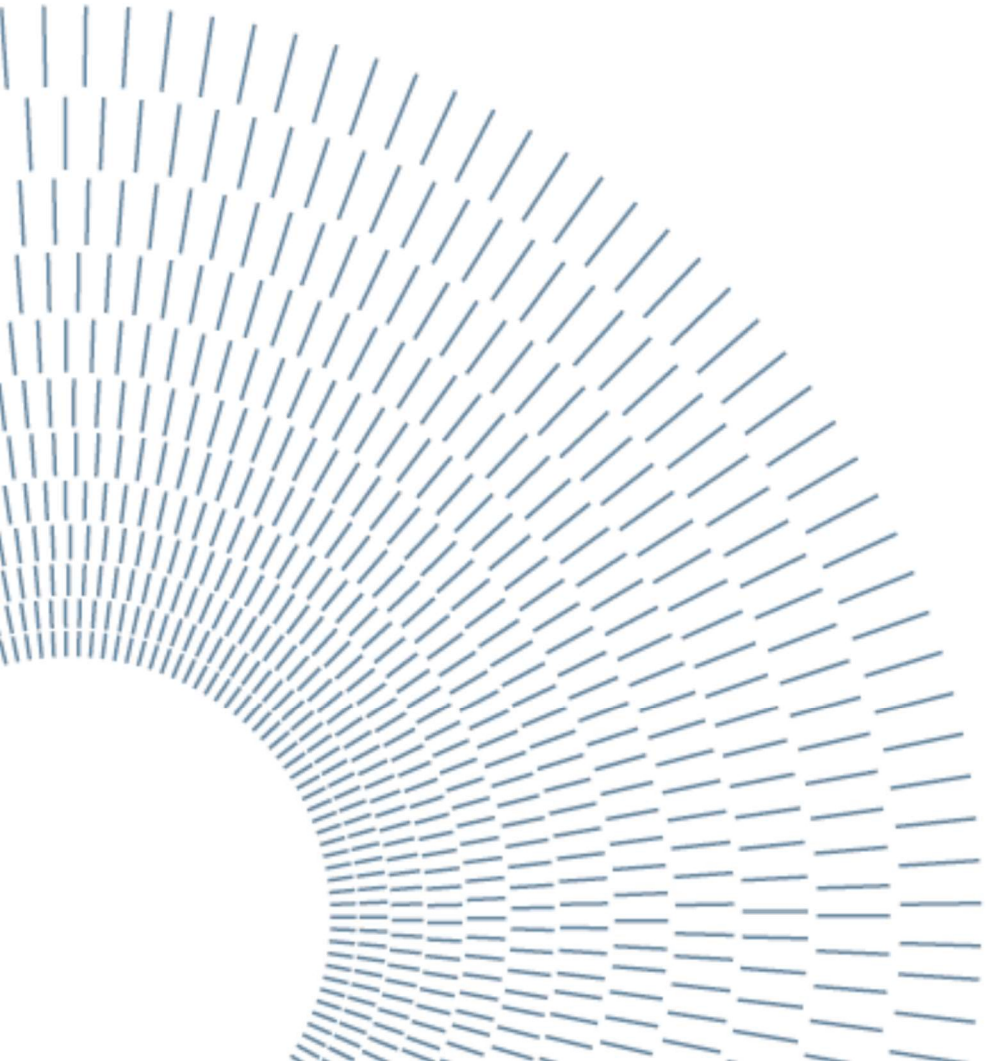
POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2 2016/2017
Project Power EnJoy

Requirement Analysis and Specifications Document

V 1.0



Redacted by:

Leonardo Chiappalupi
CP 10453536

Ivan Bugli
CP 10453746

Table of contents

1. Introduction

1.1 Description of the given problem

The purpose of the project is to develop a digital management system for a car sharing service that has as peculiar aspect the fact that it only offers electric cars. This means that on top of common car-sharing systems features (that are, for instance, users registering and management, car localization and activation, vehicle reservation, billing, etc.) we are also required to implement additional characteristics that are proper of electric cars.
These features are aimed to reward virtuous users, by providing bonuses depending on how responsibly they use the system. The system must therefore be able to keep track of how many passengers share the same ride, how much battery is left when the car is eventually parked, and whether the user chooses to plug the car to the power grid before leaving.

On top of this, even if not explicitly required by the client, management features will also be designed and specified to allow the day-by-day maintenance of the service: since the system will deal with electric cars, issues related to low-battery cars and customers requesting assistance will be part of this analysis, in order to predispose the entire platform to be flexible enough to handle these scenarios.

1.2 Goals

1.2.1 Core Goals

As stated in the initial description, we must guarantee that all the basic features of a car-sharing system are present in our project development. This will be, from now on, "Core goals", and they will be classified as follow:

Label	Schematic goal	Detailed goal
CG1	Users account management	Allow the users to register to the system by providing their credentials and payment information. A password is then sent them to access the system.
CG2	Car finding	Allow registered users to localize available cars within a certain range around them or close to a specified address.
CG3	Car reservation	Allow the reservation of a single car among the ones found in a specific area (i.e. as specified in CG2), up to one hour in advance of when they actually pick it up.
CG4	Car reservation reset & fee	If a car is not picked up within one hour from the reservation (CG3), the car must be made available again and the user must be charged of EUR 1.
CG5	Car unlock/ activation	Allow users nearby a previously reserved car (CG3) to signal it, in order to unlock the car and make it usable.
CG6	Charges	Provide a charge computation system for the service that starts as soon as the user ignites on; notify the user about charges via a screen located inside the car itself. Charge the user at the end of a ride.
CG7	Charges termination and lock	Stop charging the user when the car is parked in a safe area, and lock it.
CG8	Safe areas management	Allow system managers to add, edit and remove safe areas for parking; allow users to know where these parking zones are located.
CG9	Assistance and maintenance	Allow users to report issues, and provide features for basic car maintenance and fixes (low-power car management, fleet management).

1.2.2 Additional exclusive Goals

On top of the *Core Goals*, we want also to achieve some specific goals that strictly relate to how this car sharing system is conceived, that is using electrical cars. From now on, we will refer to this kind of specifications as *Additional (exclusive) Goals* and therefore we will focus about:

Label	Schematic goal	Detailed goal
AG1	**Discount for ride sharing**	Apply a 10% discount to users that carry at least two other passengers in the same car (discount applies to that very ride).
AG2	**Discount for battery saving**	Apply a 20% discount to users that leave the car with at least the 50% of battery capacity full (discount applies to last ride).
AG3	**Discount for plugging**	Apply a 30% discount to users that, at the end of their journey, leave the car in special parking spots and plug the car to the power grid (discount applies to last ride).
AG4	**Additional charge for compensation**	Apply a 30% penalty on journeys that terminate either with battery below 20% or further than 3Km from the nearest power grid station

Goals have been labelled with short tags to better identify them in the following processes of requirements analysis and specifications description. This system will also be used further on in the document (see Glossary for more information).

1.3 Domain properties

The previous goals will be pursued by considering assumed that:

Label	Property
D1	Users always provide valid, up-to-date and official information about their data, with particularly strong assumptions when talking about their documents: users that register to the service have a valid driving license and the data they provide to the system is consistent and original (no fake documents whatsoever).
D2	Users have a valid e-mail account and use it to register
D3	Users that use this service possess GPS or equivalent geo-localization services enabled devices.
D4	Users that unlock a car, always step in and start the ignition.
D5	Users signal that they are nearby a reserved car if and only if they are actually close to a previously reserved car.
D6	Operators actually do what they report taking charge of
D7	Users actually leave the car when they park it and request a lock.
D8	Users always have a valid payment method associated with their account (either working or not)
D9	Users carry on a ride in a fair way, driving the car from start to end, without sharing accounts or cars, and without using a single ride for multiple journeys and drivers.
D10	System managers add, edit and remove safe parking areas only in locations where there are actual parking spots

1.4 Glossary

1.4.1 Definitions and vocabulary

- **User:** generally speaking, the end user that is using the service. From our point of view, users are those who have an active account on the platform, or that are about to create a new one. Depending on the case, 'user' could also be used to abstract from different kind of actors that can access the same feature.
- **System manager:** employee in charge of managing the service remotely, e.g. updating information about parking areas or about cars fleet, responding to assistance requests, etc.
- **Operator:** employee in charge of the maintenance of the service *on-site*, that is moving low-power cars to plug parking areas and intervening when an user issue cannot be solved via phone.
- **System/Service:** whenever we talk about *System* or *Service*, if not clearly stated otherwise, we are talking *as a whole* about the software we are to develop. Less often we could also use the word **Platform**.
- **Ride:** a single "session" of the service consisting of vehicle pick-up, journey and a final stop in a "parking safe area". Less often interchanged with **Journey**.
- **Parking (safe) area:** an area in which users are allowed to park their car. **Plug parking area(s)** are the same, but also equipped with cables to leave cars recharging.
- **Geo-localization services:** integrated solutions (both hardware and software) that allow to locate a user on a geographical map. Since our system will use such services as "black-boxes", we only focus on what they provide and not on how they should work.

1.4.2 Abbreviations

- **CG*i* (where *i* stands for an integer):** Core Goals reference tags/labels, as defined properly in section 1.2.1
- **AG*i* (where *i* stands for an integer):** Additional Goals reference tags/labels, as defined properly in section 1.2.2
- **D*i* (where *i* stands for an integer):** Domain properties reference tags/labels, as defined properly in section 1.3
- **R*i* (where *i stands* for integers):** Requirements reference tags/labels that will be defined and used in section 3 and further.

1.4.3 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **OS:** Operative System
- **DB:** Data Base
- **JVM:** Java Virtual Machine
- **JEE:** Java Enterprise Edition
- **UI:** User Interface

1.5 Assumptions

Client requests and wills are ambiguous or incomplete about some of the features the system will provide; therefore, we hereby explain how we completed or integrated all the inconsistencies or misses that are critical for the system development.

First, it is not clear how the users associate their documents (ID and Driving License), and whether we should validate them or not. We suppose that during the registration, this data is actually requested and collected, but as stated in **D1**, **we will not check its integrity.** We believe this is a more complex duty that relies on external services that depend on country administration and other details that we will not take into account.

Then, we do not know anything about how we will charge the users for their rides: again, we suppose that during the registration phase, the users are asked to provide a valid payment method (we will consider PayPal or a Credit Card) and that, once registered, our system will be able to charge that very account with no additional authorization or data to input. **We hereby move all problems with payment to the selected partner (the bank of client's choice or PayPal) that will handle payments**; since the service is charged every minute, there is no way of checking the actual payment flow every time, and **there is no way of preventing a user with insufficient funds to continue using it** (unless stopping the car all of a sudden, that would be impractical and dangerous). Furthermore, banks and external services can charge fees or similar penalties when users exceed their plafond, so we rely on them taking charge of insolvent people. **Nonetheless, we will provide an external feature for payment providers, that will allow them to report for users that did not have enough funds for their last ride.** We will disable those users then, and request further calls and contacts to re-enable them.

The car parking and locking process is not sharply defined in the goals, so we need to make some assumptions as well. As for other already existing systems of this

kind, we suppose that a user parks the car, leaves it (closing all doors and shutting down the engine), and **then uses a feature of the service (being a button in the app or something similar) that asks the service to terminate their ride.** The platform handles this request by locking the car and making it available again (if the battery is enough), in addition to actually stopping the user from being charged for the ride. Even if not completely sticky to client's request, **we think that introducing this logic will allow future service improvements** like continuing paying a car even if parked (to keep it reserved), improvements that greatly justify the additional hassle of having a manual locking request operation.

We also know pretty much nothing about the *regional* part of the service, that is: whether the system should work in a pre-defined area or if it can work globally, with the only constraint of the parking areas. Examining other comparable services, we assume that parking areas are located in a pre-defined area (e.g. the metropolitan area of Milan), and that **users can take the cars they reserved wherever they want, but they are allowed to park in those areas only.**

Another aspect that requires further specifications is one related to an Additional Goal, specifically **AG1**: we are not told how other users can be detected when sharing a ride when an actual driver, that is the one who also reserved and picked-up the car. As also partially explained in **D7**, we assume that eligible users for this *bonus* are only the ones that are already registered to our service, and that **use a special feature on their app/interface to claim they are on the same car of the main account**, that is the driver one.

Even **AG3** introduces some discrepancies, since it is not clear how special parking spots relate to normal authorized parking spots: we reasonably thought about introducing a very sharp distinction between the two kinds of parking when showing them to the users, assuming that **they are different things located in different places.**

All *Additional Goals* do not clarify how discounts/surcharges are applied when they are more than one: in our system, **variations in price will be calculated applying them one after another, from the most penalizing to the most convenient** (in the interest of the client's revenue).

Our final thought is about the nature of these cars, that are powered by batteries so they are likely to have a limited working time before they get charged. **It is our intent to provide the user as available *only* those cars that have an adequate amount of battery life still available.** Since provided specification do not put constraints in this direction, we will try to integrate into the system a feature that monitors cars' batteries and mark as *Low battery* or *Not enough power* the vehicles that have these "conditions". **Cars will report their battery level every time they are parked, together with whether they are plugged in or not.** Cars

with battery below 20% and not plugged will not be made available again when parked and locked. This is also why we introduced **CG9**: if users run out of battery when still in a ride, **we will provide them with an assistance channel** (being, on top of a call center, a button both inside the car and the app to report a problem) that will hopefully sort out the situation.

To predispose a maintenance service on cars, that is carried out by a department of workers in charge of moving and plugging cars with low battery, we will also provide a feature to filter for this kind of cars (» more details at **CG9** requirements). The idea is that unavailable "dead" cars are manually moved by human operators to the closest plug parking spots, and when their battery reach 80%, they ask the system to be marked as "available" again.

1.6 Proposed System

Considering all aspects examined in previous sections, the system for Power EnJoy will follow a general outline that will have this structure:

We will design a software system made of a core back-end and several interfaces (or front-ends) that will be made available for users.
The back-end will store and retrieve all the content that is needed by the front-ends, including account data, rides reports and geographical data for the service itself. Users will be provided with a web interface (either a mobile application or a responsive website, the key concepts for their design would not change) that allows them to sign up, edit their account data, browse available cars and perform all the features listed in section 1.2.
System managers and operators will be provided with a simple application to accomplish all the duties they are in charge of: managing low-power cars, managing the car fleet, responding to assistance request, managing parking areas.
Cars, on their side, will be considered already equipped with all the necessary sensors and transmission devices that we need to make the system work: at minimum, cars must be able to transmit their battery level and whether they are plugged or not, they must be able to report the system when they are ignited, and they must have a way to ask the system for being made available again when charged enough.

1.7 Stakeholders Identification

Our "financial" stakeholder is the professor, which here resembles an actual company that required our consultancy to design the platform. Both considering reality and a hypothetical case, who commissioned this work probably wants a

reliable system that performs all the required functionalities and so they need a detailed RASD document as well as other development cycle documentation papers. Concerning the actual didactical purposes, the aim here is to show that we understood the key concepts in software engineering and that we are able to apply them to a likely case.

Our typical user, on other hand, is a casual (probably young) person that needs car-sharing to improve their personal transportation on a daily/periodic base, but every time it will be an "on demand" usage. So it is important that we design the platform keeping in mind this kind of customers, that will need a fast and easy service.

2. Actors identification

- **Guest:** all non-registered users that access the service via the web or the app. Guests can only visualize the map of available cars nearby (to have a glimpse of how the platform works) and, obviously, register to Power EnJoy.
- **Registered user:** the most common type of actor, a registered user can access almost every feature of the service. They can manage their account, find cars, reserve them, signal the platform they are sharing a ride, browse the history of their charges and journeys, and so on.
- **System manager:** this actor models the personnel in charge of adding, removing and editing parking areas, as well as first responding to users assistance requests and maintaining updated the car fleet information.
- **Operator:** these are those in charge of taking care of low-battery cars and intervene on-site when users need further assistance, that is when system manager cannot solve an issue via phone. Operators can see a list of low-battery cars, move them to parking areas and plug them (one by one).
- **Cars:** this is a pretty atypical actor. We model cars this way because they are to be seen as a source of inputs and outputs to and from the system core, and they act responding to the actions of other actors in their scenario. Cars send their battery level, information about rides, signals about whether they have been parked and closed (and where this happens), but they also receive commands from the platform - like unlocking/locking instructions and so on.
  Cars also notify the system when their battery level is over 80% (after being disabled for low battery and then plugged somewhere).

3. Requirements

3.1 Functional Requirements

For each goal specified in section 1.2, we will now illustrate the main functional requirements that will ensure all those goals will be covered, possibly combined with some of the domain properties of section 1.3.

3.1.1 Functional Requirements for *Core Goals*

CG1	**Users account management**
	Allow the users to register to the system by providing their credentials and payment information. A password is then sent them to access the system.
R1	Non-registered users can (only) login, register, or browse for nearby cars
R2	Non-registered users can only register if they choose a username not already selected by another user
R3	Registered users cannot register again
R4	Users receive a confirmation email with the password only after they requested a registration
R5	Users are allowed to register if and only if they fulfill all the required document data as well as a valid payment method
D1	Users always provide valid, up-to-date and official information about their data, with particular strong assumptions when talking about their documents: users that register to the service have a valid driving license and the data they provide to the system is consistent and original (no fake documents whatsoever).
D2	Users have a valid e-mail account and use it to register

CG2	**Car finding**
	Allow registered users to localize available cars within a certain range around them or close to a specified address.
R6	If users choose to locate nearby cars, they get a map with all available cars in the selected range
R7	If users choose to locate cars that are nearby a defined address, they get a map with all available cars close to that address
R8	Users cannot locate cars both nearby and around a specified address
R9	Only registered users can search for cars customizing the location
R10	If users try to locate cars outside of the service working region, they receive an error message
D3	Users that use this service possess GPS or equivalent geo-localization services enabled devices.

CG3	**Car reservation**
	Allow the reservation of a single car among the ones found in a specific area (i.e. as specified in CG2), up to one hour in advance of when they actually pick it up.
R11	Users can only reserve cars that have been previously located
R12	Only registered users can reserve cars
R13	If a user reserves a car, the system starts counting for a 1-hour interval
R14	Users can reserve only one car at a time
R15	Only cars not already reserved can be reserved
R16	When a car is reserved, no other user can also reserve it
R17	Low-battery or out of order cars cannot be reserved

CG4	**Car reservation reset & fee**
	If a car is not picked up within one hour from the reservation (CG3), the car must be made available again and the user must be charged of EUR 1.
R18	A reserved car whose relative timer reaches 0 must be made available for reservations again
R19	If a reserved car remains locked for more than 1 hour, the user that made the reservation is charged of EUR 1
R20	If a user must be charged with a fee, this fee is taken using the payment method specified during registration
D5	Users signal that they are nearby a reserved car if and only if they are actually close to a previously reserved car.
D8	Users always have a valid payment method associated with their account (either working or not).

CG5	**Car unlock/activation**
	Allow users nearby a previously reserved car (CG3) to signal it, in order to unlock the car and make it usable.
R21	If a user signals he is nearby a previously reserved car and the 1-hour timer has not terminated yet, the car is unlocked
R22	If a car is unlocked, it is also made usable (ignition available)
R23	If a car is unlocked, the relative 1-hour timer is reset
D5	Users signal that they are nearby a reserved car if and only if they are actually close to a previously reserved car.

Functional requirements continue on the next page »

CG6	**Ride and charges**
	Provide a fare system for the service that starts as soon as the user ignites the car; notify the user about charges via a screen located inside the car itself.
R24	If a user ignites a car, the system starts counting for time and charges, applying fares every minute and depending on currently valid prices
R25	Every time charges are increased, the counter inside the car is updated
R26	Only the user that is driving the car gets charged for the journey
R27	Charges are applied on the user's payment method specified during registration
R28	The car updates its position every minute, and the system stores it
D4	Users that unlock a car, always step in and start the ignition.
D8	Users always have a valid payment method associated with their account (either working or not).
D9	Users carry on a ride in a fair way, driving the car from start to end, without sharing accounts or cars, and without using a single ride for multiple journeys and drivers.

Functional requirements continue on the next page »

CG7	**Charges termination and lock**
	Stop charging the user when the car is parked in a safe area, and lock it.
R29	Users can only terminate a ride if the car is parked in a valid spot
R30	When a user asks for ride termination, charges are stopped as well
R31	When a user asks for ride termination, the car is locked
R32	When a car gets locked, it reports its battery level and whether it is plugged in or not
R33	If a car is locked and its battery level is greater than 20%, it is made available again
R34	Only registered users on an active ride can ask for its termination
D7	Users actually leave the car when they park it and request a lock.

CG8	**Parking areas management**
	Allow system managers to add, edit and remove parking areas for parking; allow users to know where these parking zones are located.
R35	Only authorized system managers can edit safe parking areas
R36	Authorized system managers can add safe areas
R37	Authorized system managers can edit/remove safe areas
R38	Only registered users can get the location of safe areas
R39	Registered users can browse parking areas location on a map
D10	System managers add, edit and remove safe parking areas only in locations where there are actual parking spots

CG9	**Assistance and maintenance**
	Allow users to report issues, and provide features for basic car maintenance and fixes (low-power car management, fleet management).
R40	Registered users on a ride can report a problem with their ride
R41	If a problem is reported, the customer is called back and provided with information
R42	If a reported problem requires on-site intervention, the assistance request is forwarded to operators
R43	Operators can browse for positions of disabled cars with battery below 20% or out of order
R44	Operators can browse for assistance requests that need on-site intervention, and relative positions
R45	Operators can choose to either move a car or fulfill an assistance request and the chosen element is no more shown to other operators
R46	If a car is moved to a charging spot and plugged, when its battery goes above 80% the system re-enables it for reservations and rides
R47	System managers can add cars to the service fleet
R48	System managers can remove cars from the service fleet
R49	System managers can edit cars details
D6	Operators actually do what they report taking charge of

3.1.2 Functional Requirements for *Additional Exclusive* Goals

AG1	**Discount for ride sharing**
	Apply a 10% discount to users that carry at least two other passengers in the same car (discount applies to that very ride).
R50	A user that reserved a car cannot apply to a shared ride until either the reservation is canceled or the consequent ride is terminated
R51	Users sharing a ride must confirm they are actually in the same car and their position must be checked at the start of a ride
R52	If a ride is shared with more than 2 passengers, a 10% discount is applied to that ride
R53	If a user is a passenger of a shared ride, he/she cannot reserve cars or share other journeys until that ride is completed
D3	Users that use this service possess GPS or equivalent geo-localization services enabled devices.

AG2	**Discount for battery saving**
	Apply a 20% discount to users that leave the car with at least the 50% of battery capacity full (discount applies to last ride).
R32	When a car gets locked, it reports its battery level and whether it is plugged in or not
R54	After the termination of a ride, if the registered battery level is above 50%, the driver receives a discount of 20% on that ride
D7	Users actually leave the car when they park it.

AG3	**Discount for plugging**
	Apply a 30% discount to users that, at the end of their journey, leave the car in special parking spots and plug the car to the power grid (discount applies to last ride).
R32	When a car gets locked, it reports its battery level and whether it is plugged in or not
R55	After the termination of a ride, if the car has reported to be plugged, the driver receives a 30% discount on that ride
D7	Users actually leave the car when they park it.

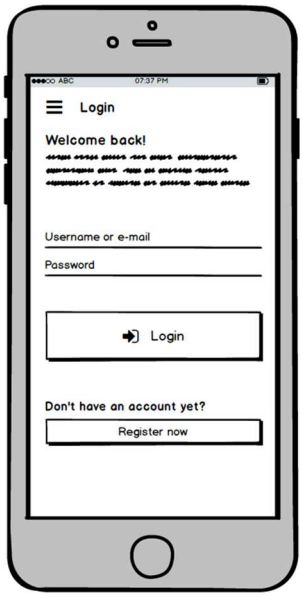
AG4	**Additional charge for compensation**
	Apply a 30% penalty on journeys that terminate either with battery below 20% or further than 3Km from the nearest power grid station
R32	When a car gets locked, it reports its battery level and whether it is plugged in or not
R56	After the termination of a ride, if the reported battery level is below 20%, the driver is charged with a 30% penalty on that ride
R57	After the termination of a ride, if the distance of the car from the nearest power grid parking spot is above 3Km, the driver is charged with a 30% penalty on that ride
R58	The 30% penalty can be applied only once per ride
R27	Charges are taken from user's payment method specified during registration
D7	Users actually leave the car when they park it.
D8	Users always have a valid payment method associated with their account (either working or not).

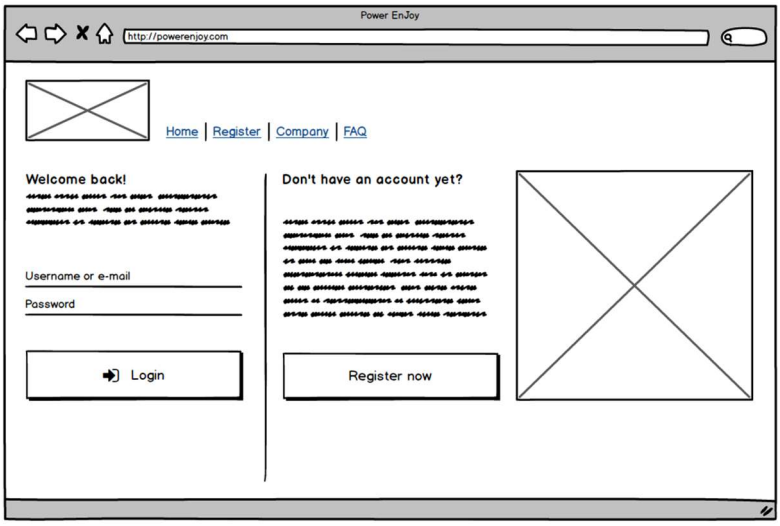
3.2 Non-functional Requirements

3.2.1 User interface

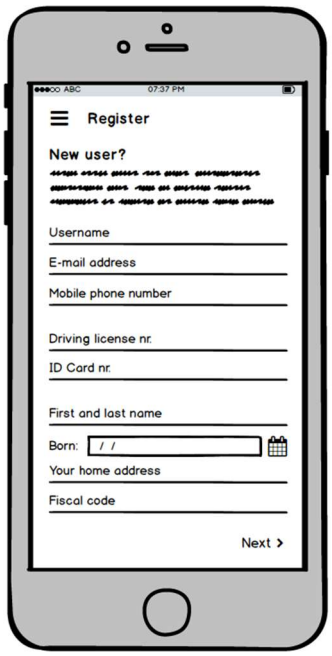
The platform will be consistent across devices, and its interface will allow all the interactions needed to cover all specifications explained so far. The following UI mockups highlight some of the most important interactions scenarios and how the UI follows the uniformity and accessibility of non-functional requirements.

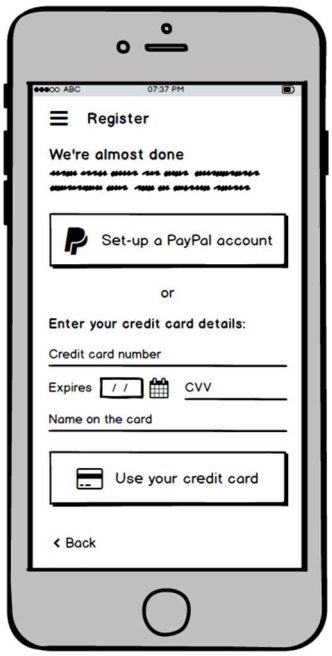
Login screen

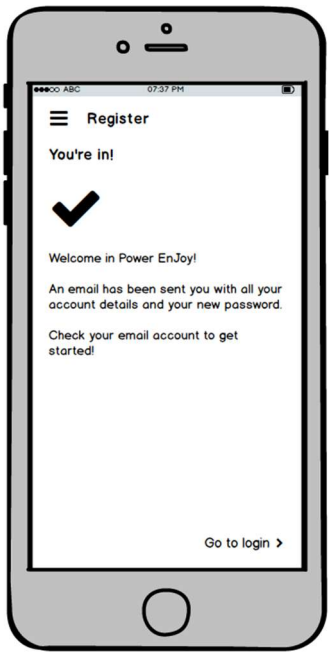




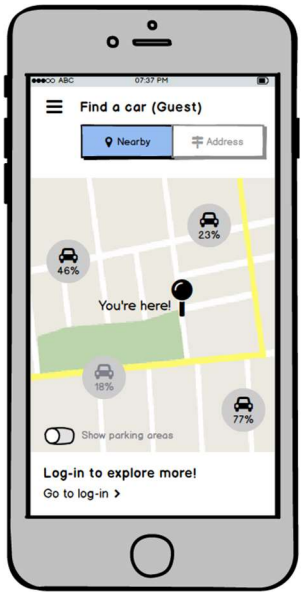
Registration

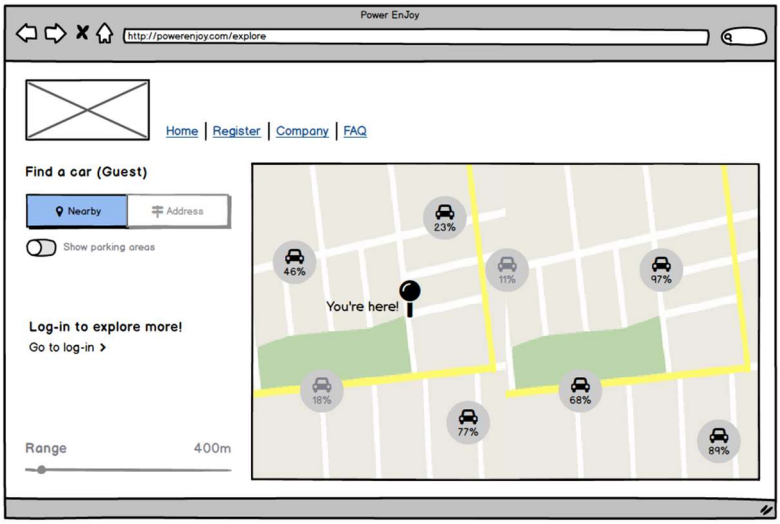




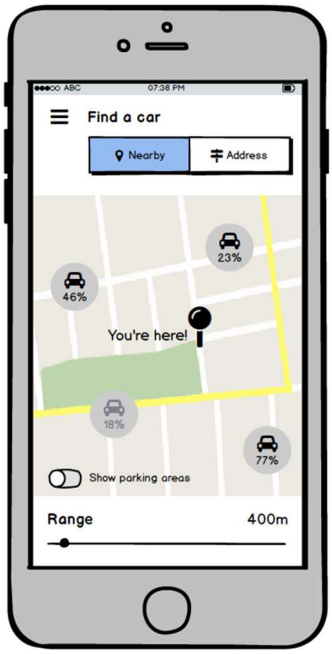


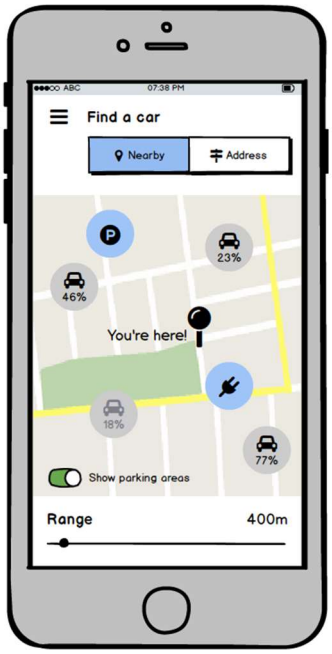
Car map visualization (Guest)



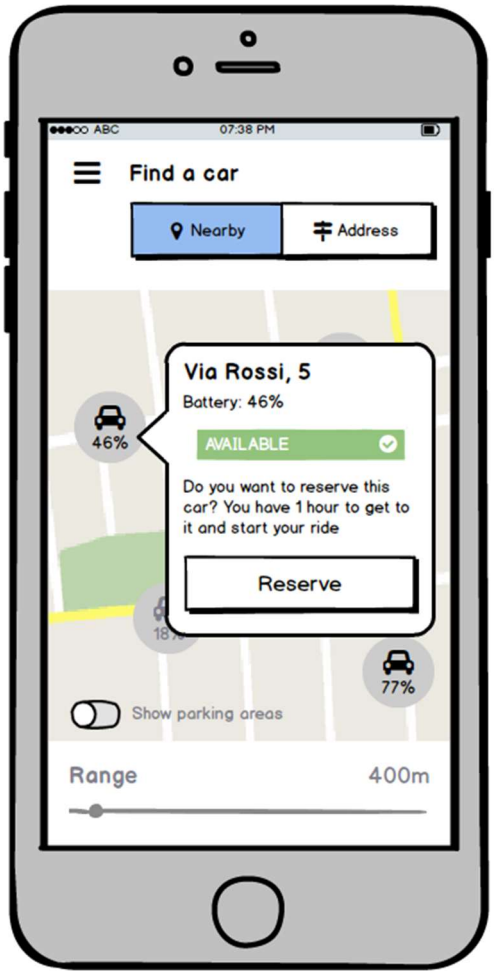


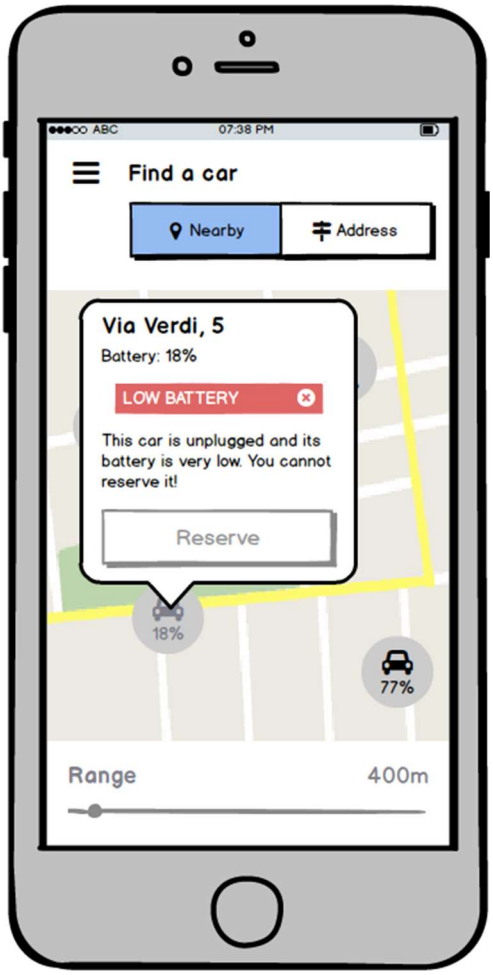
Car map visualization (Registered User)



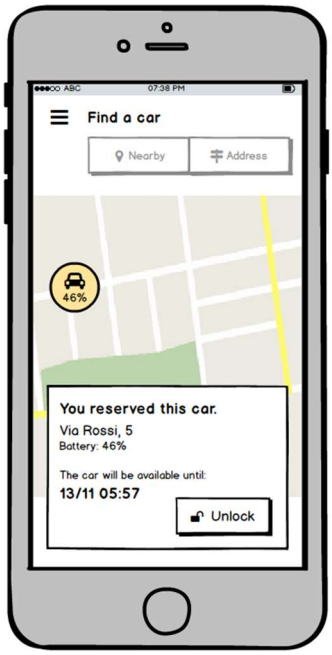


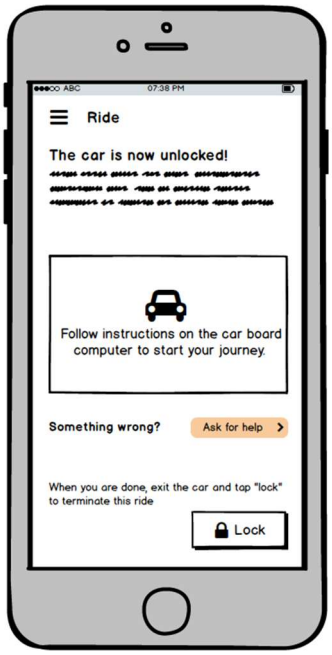
Car information

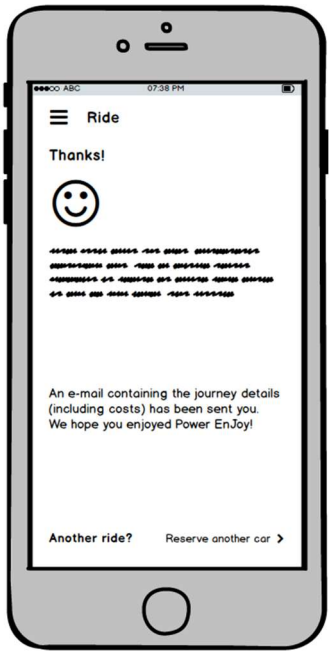




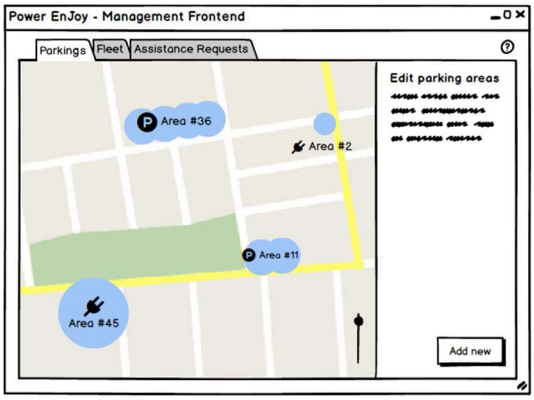
Car reservation

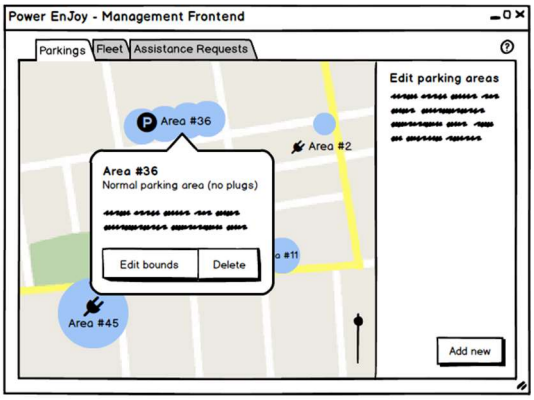


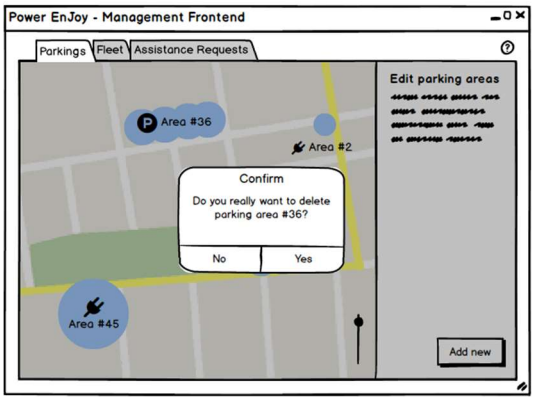


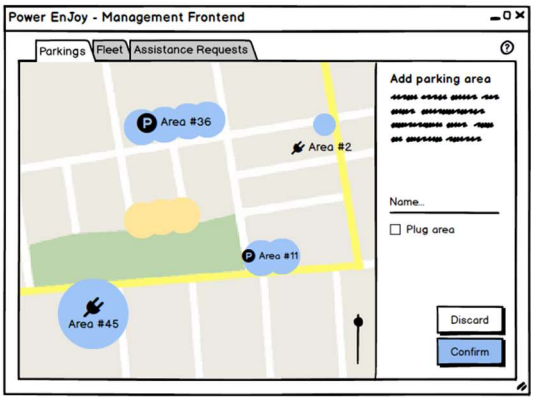


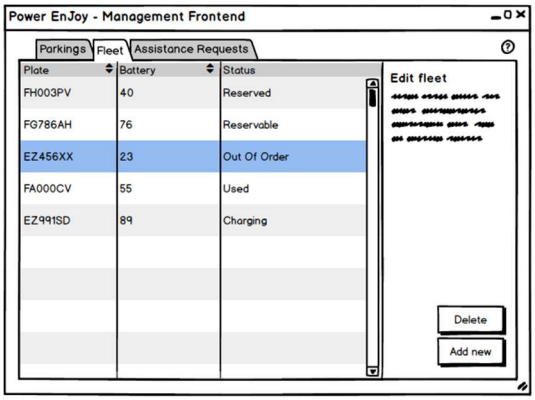
System managers: Parking areas, Fleet, Assistance management

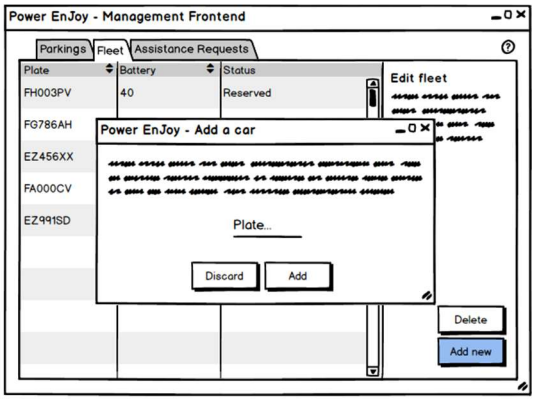


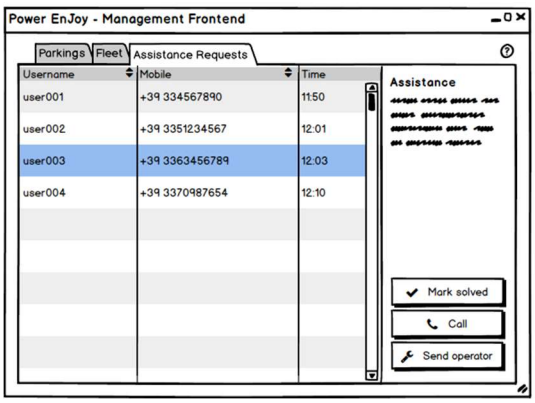


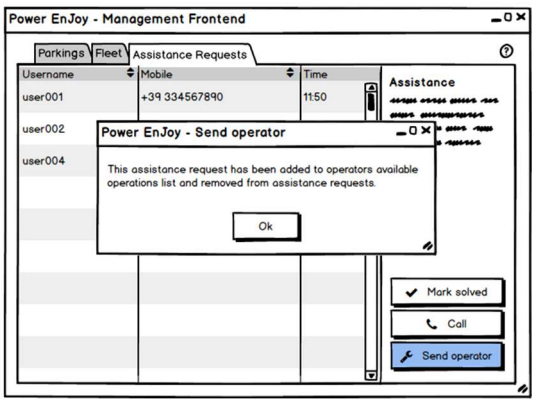












3.2.2 Documentation

During the development process of Power EnJoy, several documents will be released in order to accompany the design, the deployment and the maintenance of the platform:

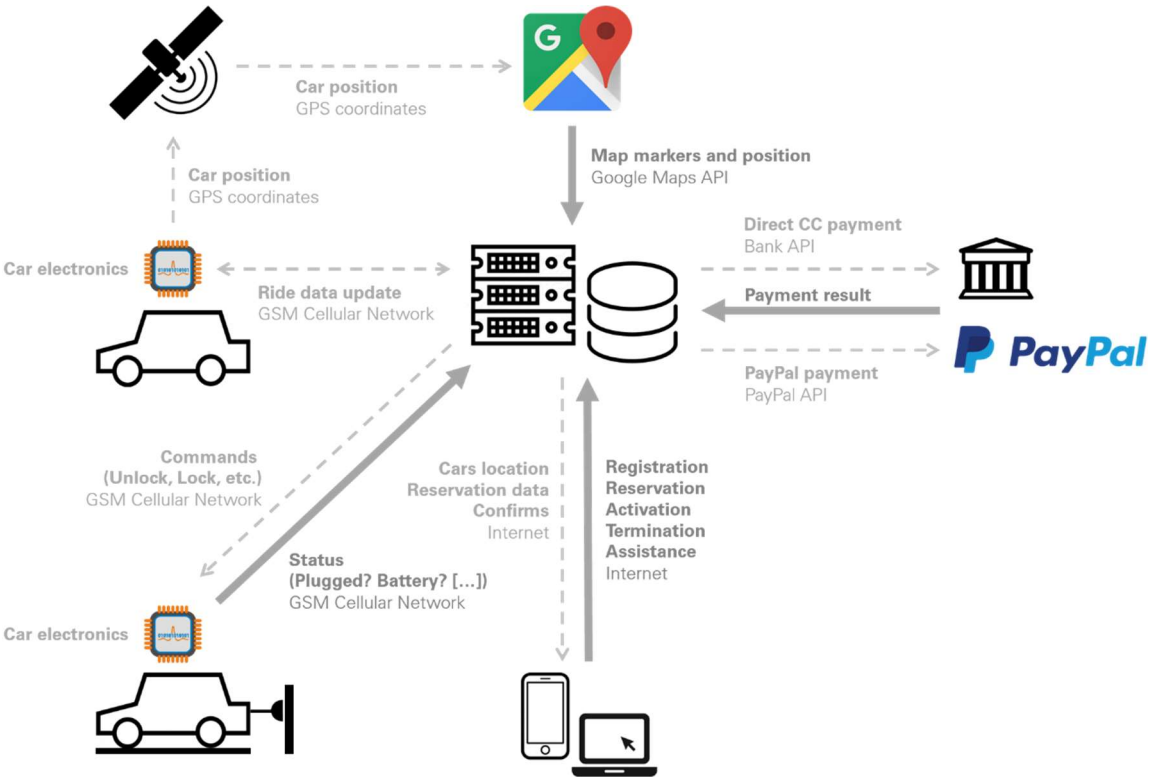
- **RASD (Requirements Analysis and Specifications Document)**, which contains an overall description of the system, the main goals, the requirements to satisfy, as well as other general considerations about the service;
- **DD (Design Document)**, which is aimed at providing a functional description of the system, using various types of modules;
- **ITPD (Integration Test Plan Document)**, which describes how the integration test will be carried out;
- **PP (Project Plan)**, that will finally describe a full planning for this project;
- **User's manual**, with all the crucial information about how to use the platform;
- **Testing report**.

3.2.3 Architectural outline

See also: Section 1.6, Proposed System

The main idea that we will be focusing on during the planning of this platform is the creation of a J2EE back-end that uses a convenient DBMS to retrieve and store information about cars, users, journeys and locations in a central redundant DB.

Following sections in this document will provide even more detailed information about the data structure used in the database, as well as about relations that hold among this data. This scheme proposes an overall outline of how the main components of the system will interact:



This back-end also exposes all the network services needed from the system to work, that will be used by all the web-based interfaces to be developed (a responsive website, a mobile application for the most common smartphone OS, etc.).

This interfaces will allow all interactions necessary for satisfying requirements specified above, and they will be consistent both in graphic and functional design on every platform they will be released to.

The system will need to communicate with payment external systems and with a geo-localization support service (Google Maps), therefore we will be using PayPal and Google APIs, as well as APIs or interfaces provided by the client's bank of choice for performing Credit Card payments.

External APIs to provide other developers with data from the service will be possibly considered, but at this time security concerns make us postpone this idea.

5. Scenarios identification

In this section, various possible scenarios are presented, in order to provide a wide range of circumstances our system is able to manage.

5.1 Scenario 1

Brian is a commuter who every day comes to Milan to work. He often complains about public transportation, because of delays and strikes. One day, speaking with a friend, he discovers the existence of the new car-sharing service Power EnJoy, and so he decides to try it out, thinking it would be convenient.
He searches the application with his mobile phone and downloads it. Launching it Brian is prompted to register as a new user, considering that it is the first time he accesses the platform. He is asked to provide all the basic information: name, surname, phone number, fiscal code, birth date, address and a valid e-mail account (later verified with a confirmation mail automatically sent by the system after the registration, which also contains the password to log in). Then a new form is displayed and Brian is compelled to provide a valid (meaning here also not expired) driving license; document number and expiring date are requested. At the end Brian is also required to choose a payment method between PayPal and Credit Card to pay the service provided by Power EnJoy.
After having inserted all the required information, Brian is automatically logged into the system and can now access all the features thanks to the "friendly" UI of the app.

5.2 Scenario 2

Brian needs to reach his office, located in Milan outskirts, from the railway station where he often arrives. He decides to use the Power EnJoy service to rent a car and reach it, so he opens the app and logs in, filling the login form with his name and password. After the successful login operation Brian presses the search button to find the nearest car possible to his current location; in so doing he is requested to activate his phone's GPS sensor so that he can be localized. After a bit of waiting several dots appear on the map shown in the main app page: these are the available cars nearby.
Brian then clicks on the closest dot, only 130 meters from the station, and reserves the car pressing on a specific button contained in a new page that is promptly shown. The system notifies him that the vehicle is reserved and that he

must reach and ignite it by an hour, or his reservation will be cancelled and he'll be charged a EUR 1 fee.

5.3 Scenario 3

Being satisfied with the service Brian keeps on using it and starts suggesting it to friends. Knowing that on November 19th an important exhibition will take place in Milan, he suggests two friends of him, John and Eleanor, interested in the aforementioned exhibition, to share a ride on one of the electric cars provided by Power EnJoy, in order to reach the location.
On the established day, they meet at the railway station where Brian helps them creating an account, which is necessary to share a ride and obtain a bonus discount. That done, Brian reserves a car and then Eleanor and John use the app to signal they are going to share a ride with the user "Brian". The system registers the event associating Eleanor and John to the reservation Brian requested.
The three friends reach the exhibition location and Brian parks the car in an allowed parking area. As soon as he locks the car using a specific button in the app, the system notifies him, via the app, that a 10% discount has been applied on the amount charged for the ride.

5.4 Scenario 4

John, some days after the exhibition, has to come back to Milan for a job interview. Having liked the car-sharing service advised by Brian he decides to reserve a car to reach faster the offices in which the interview is going to take place.
Two stops before Central Station, still on the train, he opens the app and reserves a car, expecting to arrive in a few minutes. Unfortunately, due to a breakdown at the very stop before Central Station, his train stops between the two stations.
More than one hour passes before the line is restored and so, when the time granted to reach the vehicle after the reservation expires, John is notified by the system, via the app, that his reservation has been cancelled and that his account has been charged a EUR 1 fee for the inconvenient, automatically paid thanks to the payment method supplied during the registration process.

5.5 Scenario 5

Eleanor too, some days after the exhibition, decides to try the service again. She needs to reach La Creta theater, located in the south-west part of Milan, to attend a comedy she was looking forward for months. Arrived in Milan she reserves a car

and reaches it far within the allowed time. Approaching the car, she notifies the system, using the app, that she reached it, so that the vehicle is unlocked.
The system then records the moment in which the car is ignited and starts to charge Eleanor every minute that passes. Thirty-six minutes later, Eleanor arrives near La Creta theater and searches for a parking zone nearby. After having parked and locked the car, she is notified the amount she has been just charged for the ride, with a thanksgiving for having used the service.

5.6 Scenario 6

Brian, now used to the service, tries to save as much money as possible with every ride, sharing the car with friends when possible. One morning, however, he decides to park a little bit further from his office, in a special parking zone supplied with recharging stations, so that he can take advantage of another bonus. Arrived in the parking he stops the car near one of the recharging stations and then plugs the vehicle. The battery of the electric car immediately starts to refill and the system, notified of the virtuous behavior of Brian, applies a discount of 30% on the amount charged for the ride. Brian receives immediately a notice stating that the discount has been applied.

5.7 Scenario 7

Some days after the comedy, Eleanor comes back to Milan with two friends of her to attend another comedy and decide to exploit the situation to show them the Power EnJoy car-sharing service. She makes them register to the platform and then helps them to share with her one ride. Being the theatre not very far from the station, Eleanor parks and locks the car leaving 66% of the battery unused. Soon she receives the notification regarding the ride just taken in which is reported she received both the special discounts for having shared the ride with at least two people and for having left more than 50% of the battery charged. The payment is then processed automatically via the payment method she selected while registering to the platform.

5.8 Scenario 8

John, after having obtained another appointment for the job interview he missed last time, decides to try again the Power EnJoy service; this time, however, he waits to arrive at Central Station before reserving a car. He reaches it in a matter of minutes and ignites the engine. Arrived near the offices the interview is going to take place in, he realizes that the nearest power grid station is far 3.6Km from

his position, but decides to park here anyway, in an allowed parking area. As soon as he stops and locks the car, the system is notified that he left it too far from a power grid station and so John is notified that a 30% penalty has been applied to the amount charged for the ride. The total amount is then automatically paid via the payment system provided during registration.

5.9 Scenario 9

Bruce is a plumber living in Milan. He doesn't own a car and always uses public transportation to reach his clients. However, it would be far easier for him driving a car, so he decides to use the service offered by Power EnJoy. He therefore registers to the platform and then presses the button "search by address", filling the textbox with his own address.
In a matter of seconds several dots appear near his address and he selects the nearest one, even if it has only 23% of battery remaining, reserving it immediately. He reaches the location in less than ten minutes and unlock the car, igniting it soon after.
The location is not very far, but Bruce remains stuck in a big traffic jam due to an accident. The battery quickly run too low to maintain the engine ignited, so it stops. Bruce is compelled to use the assistance service of Power EnJoy. He presses the "assistance" button in the app and is notified that a manager is going to call him to solve the problem.
A minute later Bruce receives a phone call from the system manager Andrew and is prompted to explain the issue. After having explained that the car is stuck without battery left, Andrew tells him an operator will reach his current position soon, and to wait there.
About twenty minutes later an operator, that chose to take charge of this issue, arrives at Bruce location and carries the car away with a tow truck, plugging then it to the nearest power grid.

5.10 Scenario 10

Andrew is a Power EnJoy system manager. He has to modify a parking area because it has recently been enlarged, adding 30 new parking spots. He accesses his personal page and enters the functionality allowing him to edit a parking area.
A map, with every parking area marked in red, appears on Andrew screen. He presses the edit button after having chosen the desired one.
The area is promptly magnified and Andrew can now edit it thanks to various options offered by the edit page UI. After having enlarged the area borders and having added the single parking spots inside it, Andrew saves and confirms the modifications. The system updates data in the database and the manager is shown

an updated map where now the edited area appears enlarged, giving Andrew the final confirmation that the operation succeeded.
Andrew's work, however, is not over; he also has to add five new cars to the system and to remove an old one that needs to be demolished. The manager enters the "car fleet management" section and adds the new cars one by one, inserting, for each one, the plate in an appropriate textbox. He confirms and the cars are added to the database. Before quitting he also delete from the database the old car, searching it by plate and pressing a specific button of the UI.

5.11 Scenario 11

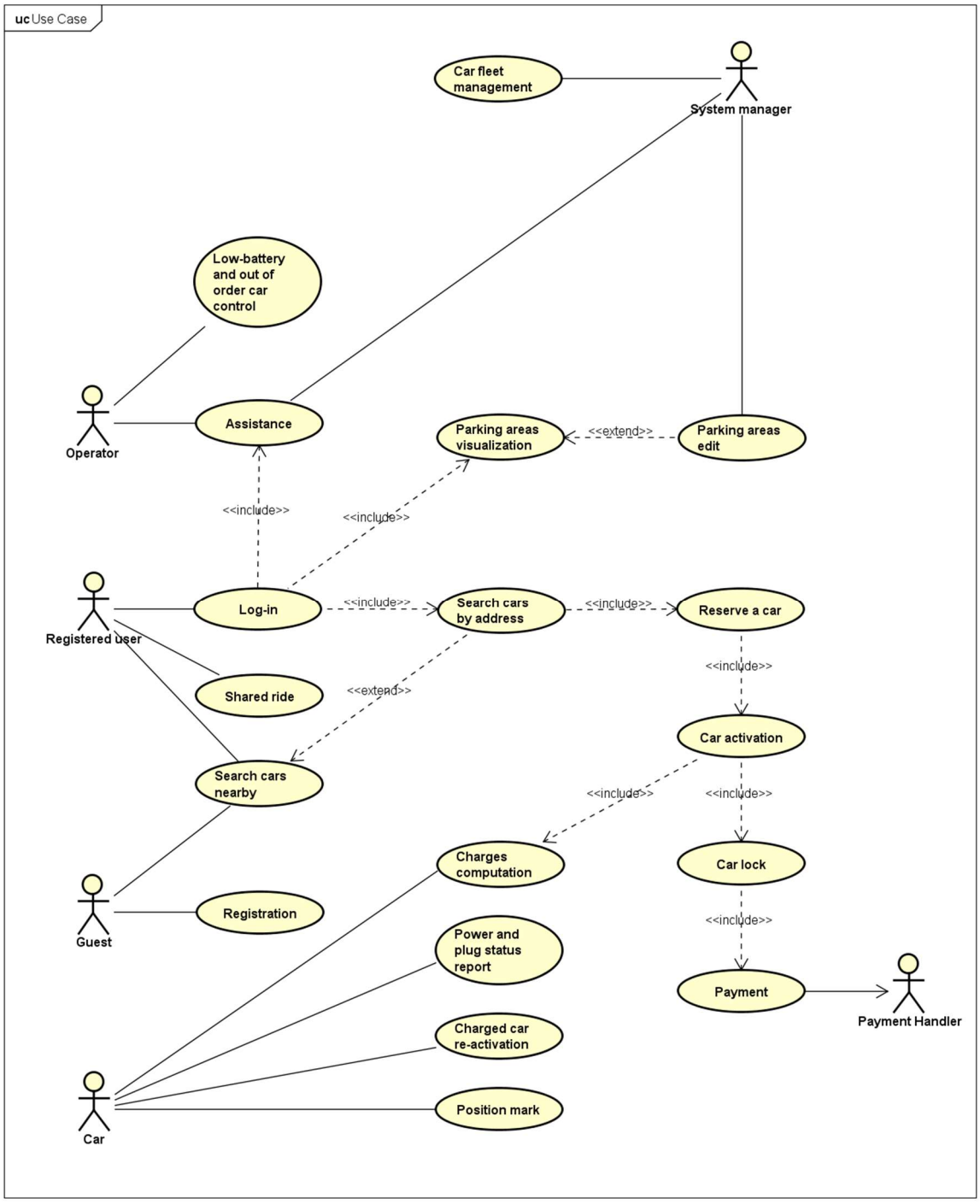
During his working turn Bob, a Power EnJoy operator, needs to choose the next work to accomplish with his tow truck.
Bob opens the operators' app and accesses the section allowing him to visualize the list of low-battery and out of order cars as well as assistance requests forwarded from system managers. Bob selects the nearest car, signaling the system he is taking care of it so that it becomes unavailable for other operators, and reaches its position to carry it to the nearest power grid. Once connected to the plug the battery starts to refill and Bob leaves it, signaling he is going to take care of another one not too far.
As soon as the battery of the car plugged by Bob reaches 80%, it signals the system its battery level by itself. The system, received the signal, modifies the database marking the car as available again. Now the car is ready to be reserved and used by the users.

6. UML models

6.1 Use Case

- Register
- Log-in
- Search cars nearby
- Search cars by address
- Reserve a car
- Car activation
- Car lock
- Parking areas edit
- Parking areas visualization
- Power and plug status report
- Shared ride
- Position mark
- Charges computation
- Payment
- Assistance
- Low-battery and out of order car control
- Charged car re-activation
- Fleet management

6.1.1 Use Case Diagram



6.1.2 Use Case Descriptions

Registration	
Actors	Guest
Goal(s)	CG1 (partial)
Entry conditions	-
Event Flow	1) Guests access the registration page 2) Guests fill the registration form with their information 3) Data is saved in the database 4) A password is sent to the new user (using the specified e-mail address) 5) The user receives the passwords
Exit conditions	The user that was previously a guest is now a Registered User, that can access almost all the features of the platform by logging in.
Exceptions	The user could abort the registration himself: if this happens, no information is saved in our system and the user remains a guest.

Log in	
Actors	Registered User
Goal(s)	CG1 (partial)
Entry conditions	The user is already registered to the system
Event Flow	1) The user loads the login page/screen 2) The user enters their username and the password received at the end of the registration 3) The user logs in 4) The system checks data correctness
Exit conditions	The user is recognized by the system and can now access all the features of the platform
Exceptions	The user provides invalid credentials: the user is then notified of the error and taken back to the homepage/home screen

Search cars nearby	
Actors	Registered User, Guest
Goal(s)	CG2 (partial)
Entry conditions	-
Event Flow	1) The user loads the homepage/home screen 2) (alt) If the user is registered and has already logged in, the "Nearby" view mode is selected 3) The system checks user position 4) The system checks for cars marked as available within a standard 1Km range from that position 5) A map is shown with markers representing cars and their battery levels 6) (alt) If the user changes the range using a slider, the system checks available cars within the new range and updates the shown data.
Exit conditions	The user can look at a map with available cars marked on it
Exceptions	The user is located outside the working range of Power EnJoy: in this case the platform shows an error and a "try again" button replaces the map

Search cars by address	
Actors	Registered User
Goal(s)	CG2 (partial)
Entry conditions	The user has logged in
Event Flow	1) The user loads the homepage/home screen 2) The "By address" view mode is selected, or the user selects it 3) The system asks the used a valid address 4) The system checks for address validity 5) The system checks for cars marked as available within a standard 1Km range from that address 6) A map is shown with markers representing cars and their battery levels 7) If the user changes the range using a slider, the system checks available cars within the new range and updates the shown data.
Alternative flow	Since this use case extends "Search case nearby", at point 2) the registered user could also choose "Nearby" and continue with the event flow in that use case (starting from 2)
Exit conditions	The user can look at a map with available cars marked on it; those cars are around a specified address
Exceptions	The address is not valid: the system asks the user for a new, valid address, and restarts the flow from point 4). The address is valid, but outside Power EnJoy working range: the system shows an error message and asks for a new address. Then the flow restarts from point 4).

Reserve a car	
Actors	Registered User
Goal(s)	CG3, CG4
Entry conditions	The user has logged in and is in front of the map with available cars, as detailed in previous UC.
Event Flow	1) The user selects a car on the map 2) The system prompts the user with car data and asks for confirmation 3) The user confirms the reservation 4) The car is marked as "reserved" in the database, and not available anymore for other users 5) The system starts a 1 hour countdowns and shows it to the user
Exit conditions	The car the user selected is now reserved for 1 hour
Exceptions	The user does not confirm at step 3): the car is not reserved and the user is taken back to the nearby cars view. The 1-hour timer expires and the user did not unlock the car yet: the car is made available again for all users and the reservation is canceled.

Car activation/unlock	
Actors	Registered User, Car
Goal(s)	CG5, CG6 (partial)
Entry conditions	The user has logged in and reserved a car (as detailed in previous UC), and the 1-hour timer has not expired yet. The user is close to the reserved car.
Event Flow	1) The user activates the app feature to request car unlock 2) The system checks if the reservation is valid 3) The system asks the car to unlock 4) The car receives the input and unlock doors/engine 5) The user ignites the car 6) The car reports engine activation to the system 7) The system discards the previously set timer (for the reservation)
Exit conditions	The user has ignition on in a previously reserved car
Exceptions	The reservation is not valid or its integrity cannot be checked: the user is prompted with an error and asked to try again.

Car lock	
Actors	Registered User, Car
Goal(s)	CG7
Entry conditions	The user has stopped the car somewhere and turned the ignition off. The user is going through a regular journey of Power EnJoy.
Event Flow	1) The user exits the car and uses the app feature to ask for ride termination and car locking 2) The system asks the car for its position 3) The car uses GPS to send back its current location 4) The system checks that the car is in an authorized parking spot 5) The system asks the car to lock 6) The car locks and reports its battery level and if it is plugged or not 7) If the battery level is above 20%, the car is made available again for other users
Exit conditions	The user has terminated a ride
Exceptions	The car is outside service bounds or not in a valid parking zone: the user is prompted with an error message and the system keeps on counting and charging. The battery level is low: the car is marked as unavailable and not shown to next users.

Parking areas edit	
Actors	System manager
Goal(s)	CG8 (partial)
Entry conditions	The system manager is logged in
Event Flow	[extends from point 5) of *Parking areas visualization*] 6) The system manager chooses to edit the parking areas with a specific button 7) The system manager browses the map and selects an area he wants to edit 8) The system shows him the selected area magnified with an interface allowing him to edit its boundaries 9) The system manager decides whether to edit the area or to completely delete it 10) The system asks for confirmation 11) The system manager confirms 12) The system processes the modifications and saves them into the database 13) The updated parking area is now shown to the system manager into the map along with the others
Alternative Flows	3) The system manager decides to add a new parking area and clicks the appropriate button 4) The system manager is asked to define the boundaries of the new area on a map 5) The system manager defines the new area 6) The system manager is asked to state if the new area is one with plugs or not 7) The system manager answers and confirms [As 10) in the standard event flow, and so on]
Exit conditions	Parking spots have been edited
Exceptions	The system manager does not confirm the modifications: the update is cancelled and nothing is saved into the database.

Parking areas visualization	
Actors	Registered user, System manager
Goal(s)	CG8 (partial)
Entry conditions	The registered user/system manager has logged in.
Event Flow	1) The user loads his homepage 2) The user opens the parking spot visualization map 3) A map containing all allowed parking areas is opened 4) The user browses the map, selecting a specific parking area if he wants more information about it 5) The selected parking area is magnified and some information about it is shown in a specific panel
Exit conditions	The user is visualizing information regarding a specific parking area
Exceptions	-

Power and plug status report	
Actors	Car
Goal(s)	AG2, AG4 (partial)
Entry conditions	The car has been locked by the user/The car is charging and a minute is passed from last report
Event Flow	1) The car uses a sensor to measure the battery level 2) The car sends the information to the system 3) The car uses a sensor to determine if the battery is charging 4) The car sends the information to the system 5) The system sends the car a confirmation signal, notifying the receipt of the message
Exit conditions	The system knows the battery level of a specific car and if it is plugged or not
Exceptions	The battery level is 0%, and so even too low to send the requested message: the system, receiving no message, marks the car as unavailable.

Shared ride	
Actors	Registered user
Goal(s)	AG1
Entry conditions	The user has logged in
Event Flow	1) The user starts the "Share a ride" feature 2) A list of the current active reservations, together with the username of the user who made them, is shown to the user, ordered from the most to the last recent 3) The user browses the list and chooses the reservation of the user he wants to share a ride with 4) A confirmation message is shown to the user 5) The user confirms 6) The system processes the request and asks the user who reserved the car to confirm the sharing request of the other user 7) The user who reserved the car confirms the request 8) The system saves the information into the database 9) The system sends a success notification to the user who asked to share the ride 10) When starting a ride, every user that is sharing it is asked to confirm their position 11) Users allow the check 12) The system uses GPS to determine if those users are close to the driver
Exit conditions	The system knows a specific car will be shared by multiple users, and the number of them
Exceptions	The user does not confirm at step 6): the system cancels the operation without sending further messages and without modifying the database. The user who reserved the car does not confirm at step 8): the system cancels the operation without modifying the database; then it sends the user who requested to share the ride a error message informing that the owner of the reservation refused his request. Some users do not allow the position check at point 10): those users are not counted as "sharing" that ride anymore.

Position mark	
Actors	Car
Goal(s)	AG4 (partial)
Entry conditions	The car is being used during a ride and a minute passed from last position report/A position request has arrived from the system
Event Flow	1) The car activates the GPS system 2) The car uses the GPS to detect its current position 3) The car sends the information to the system 4) The system sends the car a confirmation signal, notifying the receipt of the message 5) The system stores the position of the car and associates it with the car itself
Exit conditions	The system knows the position of a specific car
Exceptions	The battery level is 0%, and so even too low to send the requested message: the system, receiving no message, marks the car as unavailable. The GPS system cannot be activated: the system marks the car as unavailable and in need of maintenance

Charges computation	
Actors	Car, Registered user
Goal(s)	CG6, CG7
Entry conditions	The car has been unlocked and ignited by the user
Event Flow	1) The car notifies the system the engine started and the system starts to charge for the ride 2) The ride duration and price is shown by the car real time on a specific screen 3) The user parks the car, stops the engine and locks the car 4) The system, notified the car has been locked, halts the calculation of the amount to charge the user 5) The system then asks the car to send position and battery level 6) The car sends the information 7) The system calculates whether some discounts or fees are to be applied or not
Exit conditions	The ride ended and the system knows the final price of a ride
Exceptions	The car is not locked by the user: the system keeps on charging the user even if the engine has been stopped. The car cannot send the information requested by the system about position and battery level: the system marks the car as unavailable and sends the user a notification informing that some problems occurred, and to wait until they are fixed and the final amount to pay correctly calculated.

Payment	
Actors	Payment Handler, Registered user
Goal(s)	CG6 (partial)
Entry conditions	Charges for a ride have been computed
Event Flow	1) The system contacts the payment handler 2) The system provides the final amount to pay and user data to the payment handler 3) The payment handler processes the transaction 4) The payment handler sends confirmation 5) The system sends a mail to the user with details about the final price and possible extras (surcharge or discounts)
Exit conditions	Charges for a ride have been processed
Exceptions	There are problems during the transaction and the payment handler reports a failure at point 4): in this case we disable the user and we send them an email reporting the issue and asking them to contact the service to re-enable their account.

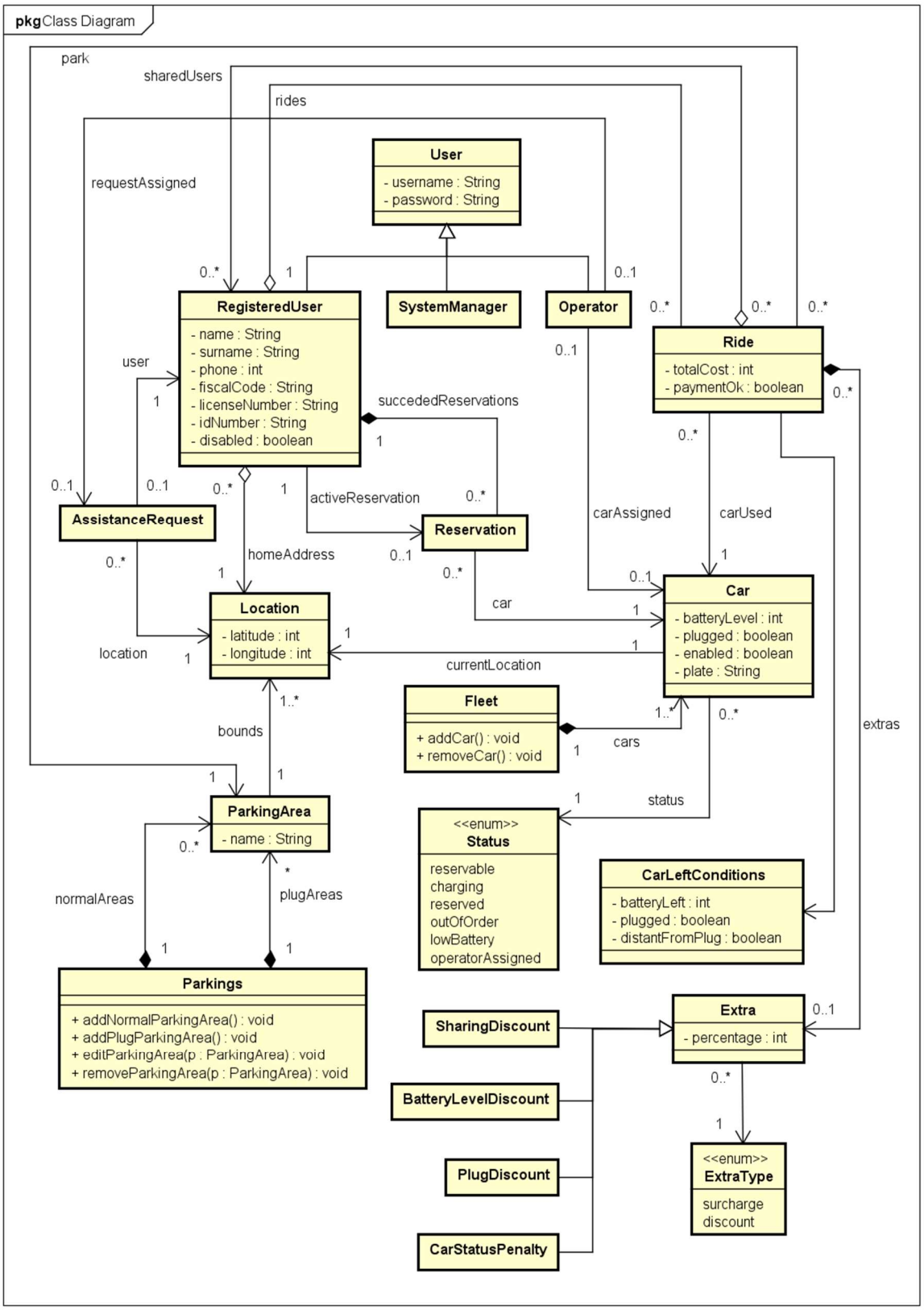
Assistance	
Actors	Registered user, System Manager, Operator
Goal(s)	CG9 (partial)
Entry conditions	The user is using the service and has some issues
Event Flow	1) The user activates the assistance feature using the app 2) The system receives the assistance request and notifies all available system managers 3) One of the system managers takes charge of the request 4) The system manager calls the user that requested assistance 5) The system manager solves the issue
Alternative flows	5) The system manager evaluates that the issue is not solvable via phone 6) The system manager forwards the assistance request to the operators' platform 7) An operator chooses to take charge of that pending request 8) The operator reaches the assistance request coordinates and solves the issue
Exit conditions	The issue of the registered user is solved
Exceptions	The issue is not solvable in a short time: then the car is marked as unavailable and the ride computed and terminated. The customer is not reachable via phone: the assistance request is ignored and canceled and the user is notified.

Low-battery and out of order cars control	
Actors	Operator, Car
Goal(s)	CG9 (partial)
Entry conditions	The operator is logged into the system
Event Flow	1) The operator opens his app 2) The operator accesses the list of cars and users in need of assistance 3) The operator chooses a low-battery car from the list 4) The car is marked as in assistance and not presented to other future operators asking for the cars list 5) The operator uses a truck to move the selected car to a plug parking spot, and plugs the car 6) When plugged, the car reports its position to the system so that the service can update car data
Alternative flows	3) The operator chooses an out of order car from the list 4) [same] 5) The operator uses his truck to take the car to the garage
Exit conditions	A low-battery car is now charging at a plug parking spot, or an out of order car is now in a garage for fixing.
Exceptions	The operator uses a feature to signal that is not able to take care of the previously selected car: the car position is updated and the car is marked again as in need of an operator

Charged car re-activation	
Actors	Car
Goal(s)	CG9 (partial)
Entry conditions	A previously low-battery car has been moved and plugged and its battery is now above 80%
Event Flow	1) The car contacts the system asking to be re-activated 2) The system asks for car position 3) The car uses GPS to determine its new position 4) The car sends the position back to the system 5) The system asks for the updated battery level 6) The car reports its current battery level 7) The system marks the car as available
Exit conditions	A disabled car is now available again for reservations
Exceptions	-

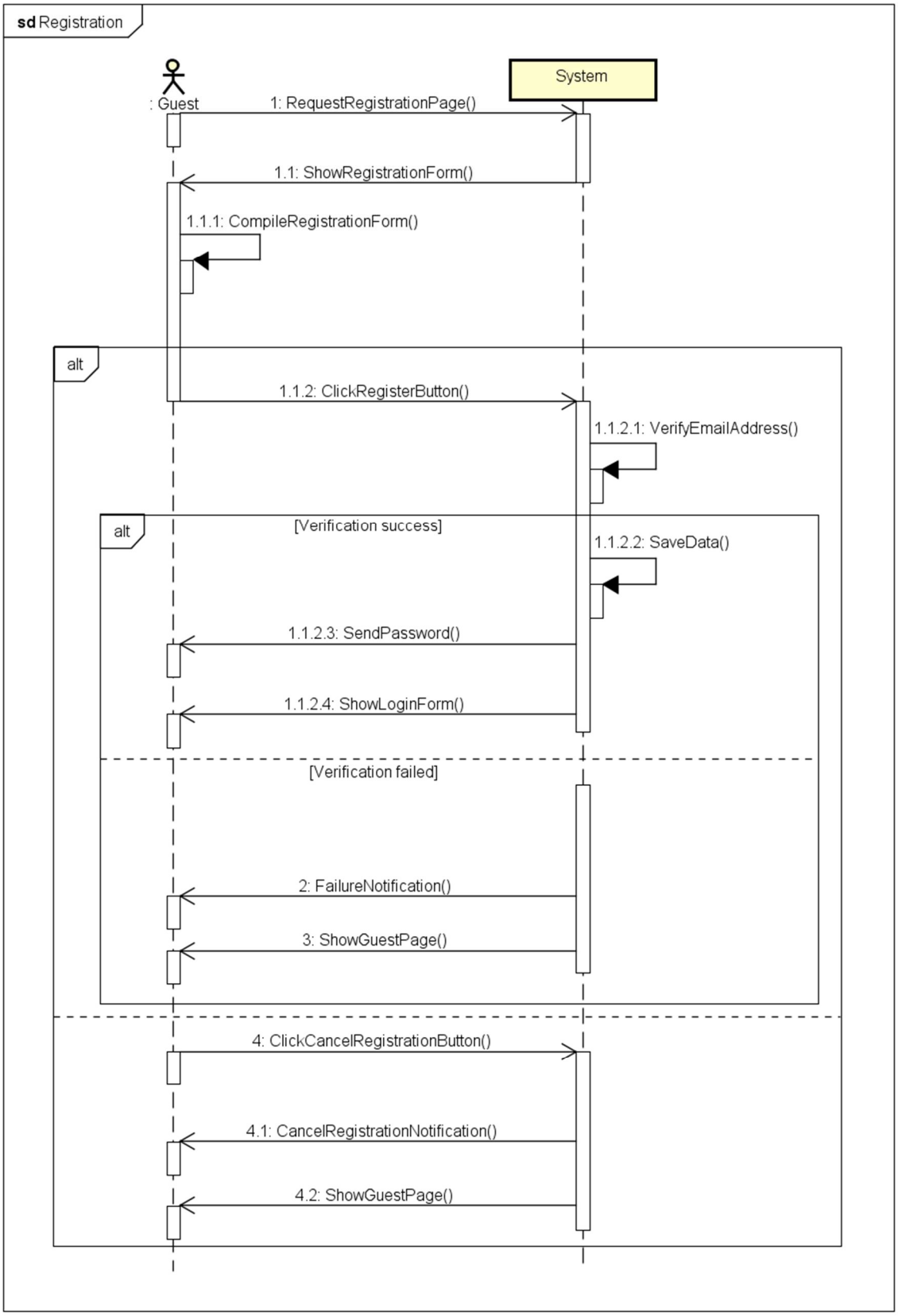
Fleet management	
Actors	System manager
Goal(s)	CG9 (partial)
Entry conditions	The system manager is logged into the system and started the fleet management feature
Event Flow	1) The system manager is presented with a list of all cars managed by the system 2) The system manager chooses to add a car 3) The system manager enters the details of the new car via a form 4) The system asks for confirmation 5) The system manager confirms
Alternative Flows	2) The system manager chooses to remove a car 3) The system asks for confirmation 4) The system manager confirms 2) The system manager selects an existing car 3) The system shows an edit car page 4) The system manager edits the details of the car 5) The system asks for confirmation 6) The system manager confirms
Exit conditions	Modifications to the fleet are applied
Exceptions	The system manager discards modifications at some point: no changes are then made to the system.

6.2 Class Diagram

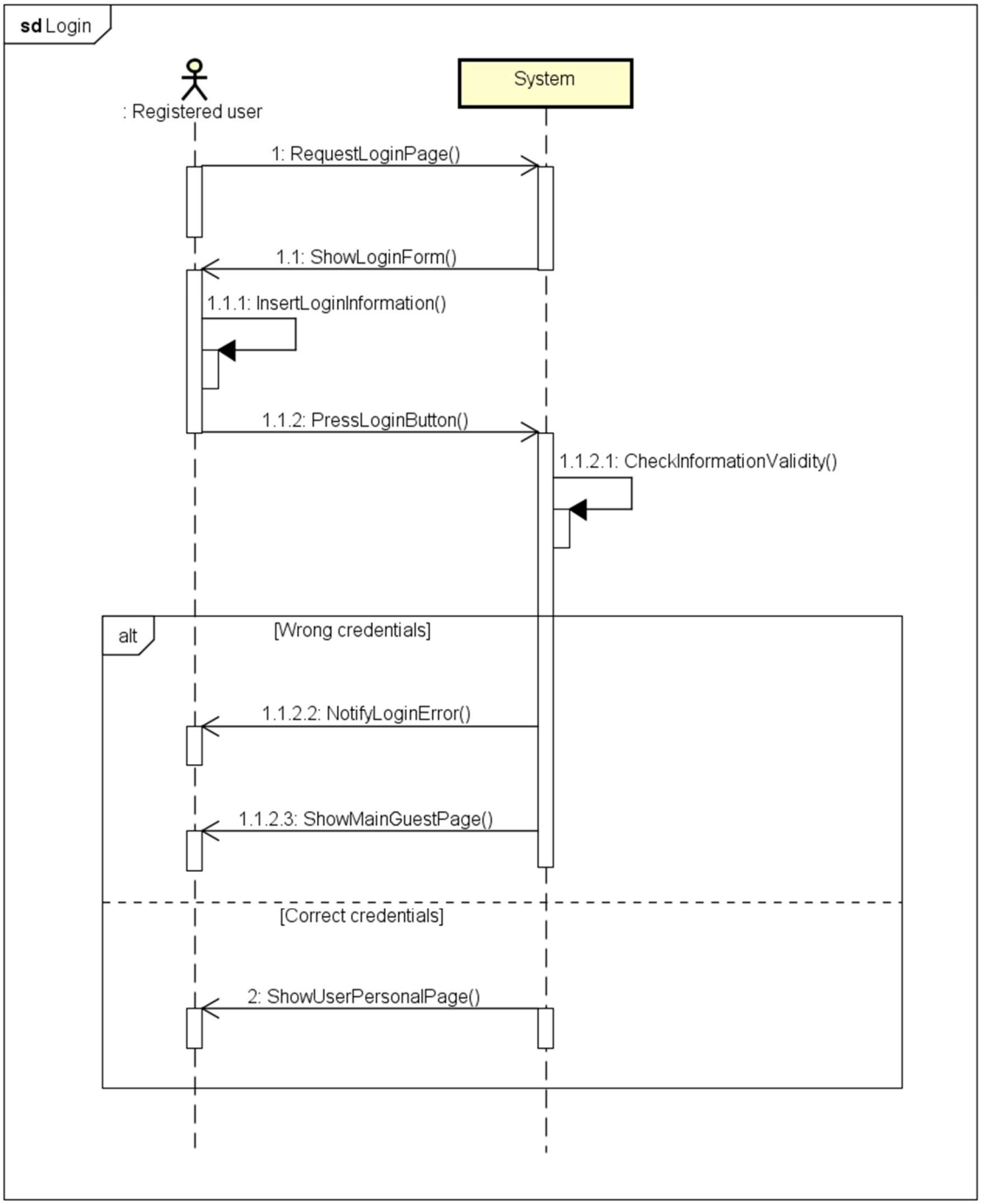


6.3 Sequence Diagrams

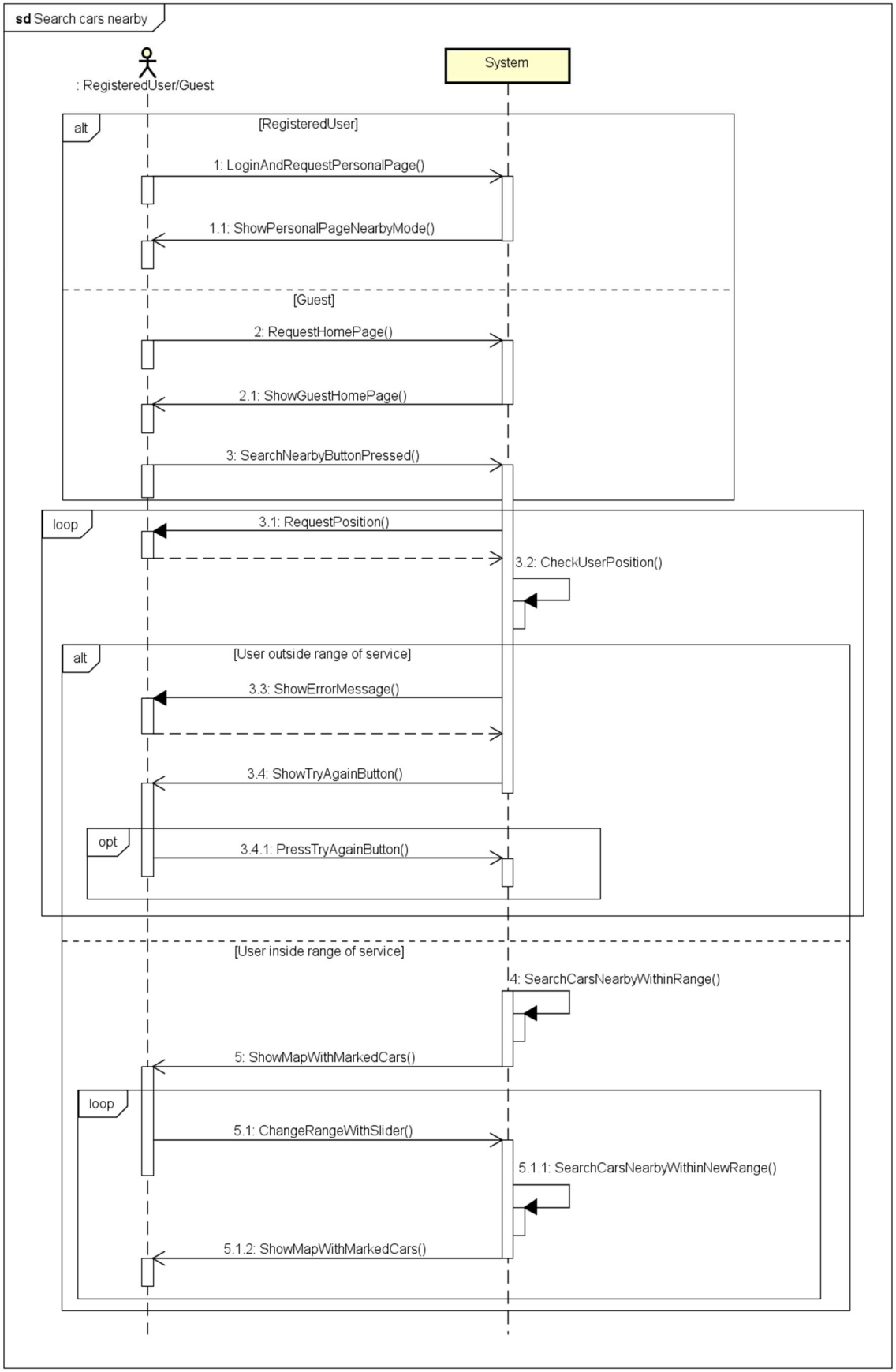
6.3.1 Register



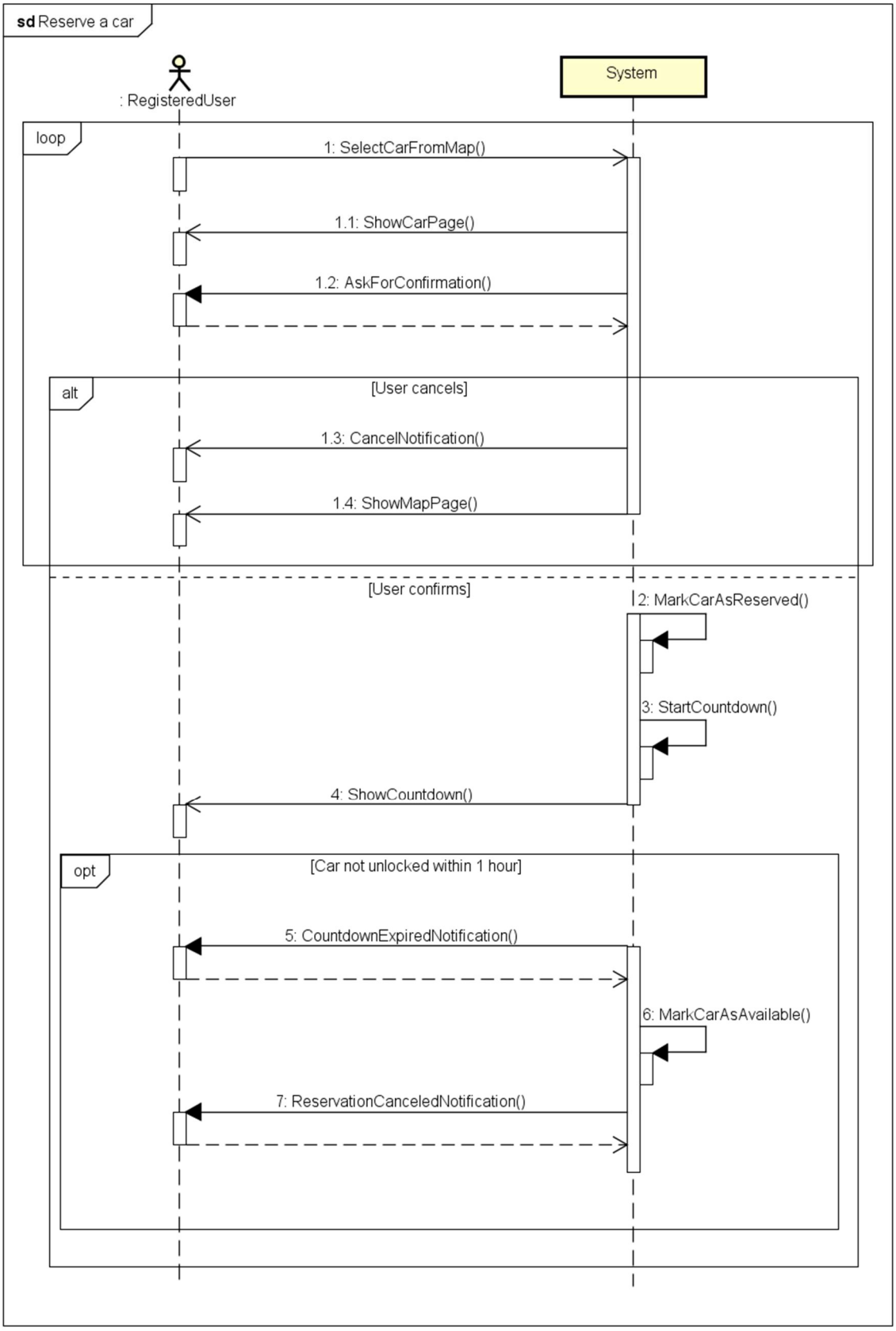
6.3.2 Log-in


sd Login
: Registered user
System
1: RequestLoginPage()
1.1: ShowLoginForm()
1.1.1: InsertLoginInformation()
1.1.2: PressLoginButton()
1.1.2.1: CheckInformationValidity()
alt
[Wrong credentials]
1.1.2.2: NotifyLoginError()
1.1.2.3: ShowMainGuestPage()
[Correct credentials]
2: ShowUserPersonalPage()

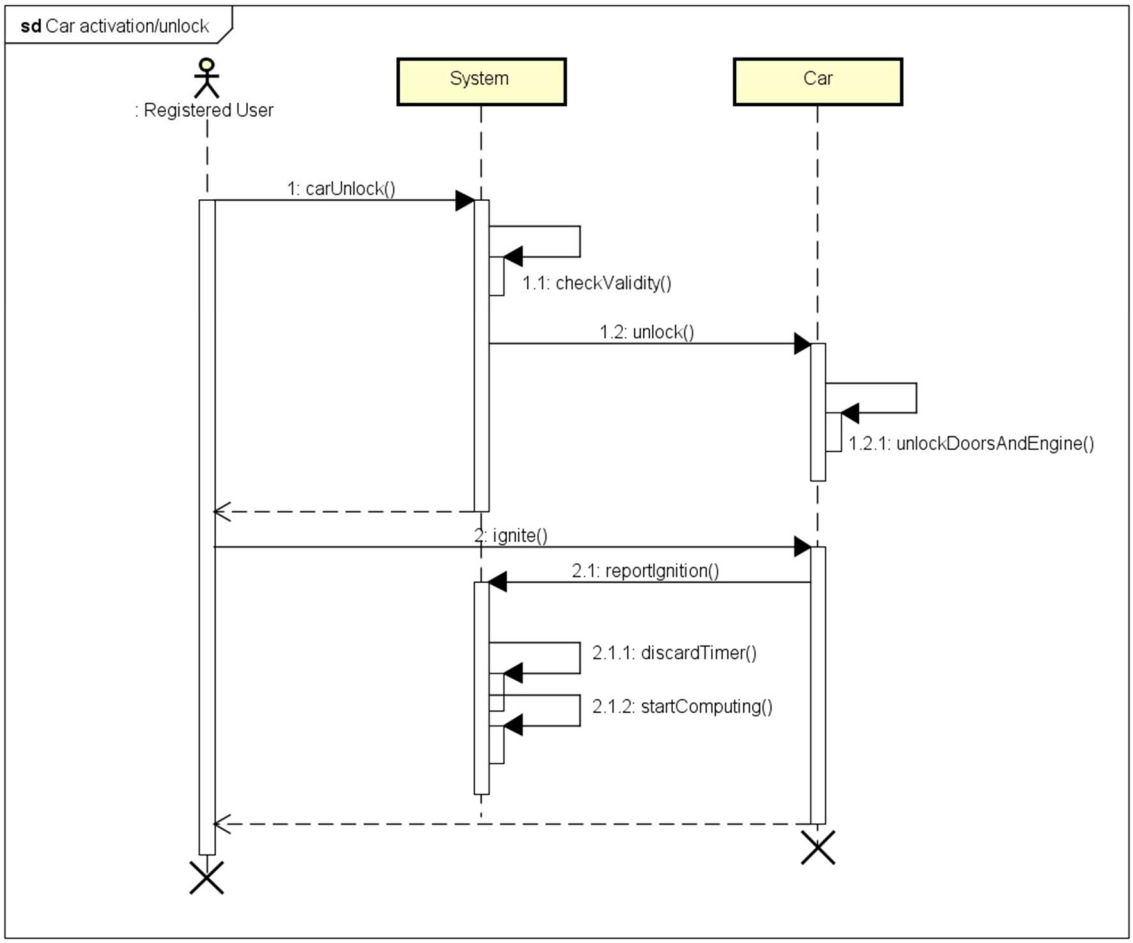
6.3.3 Search cars nearby



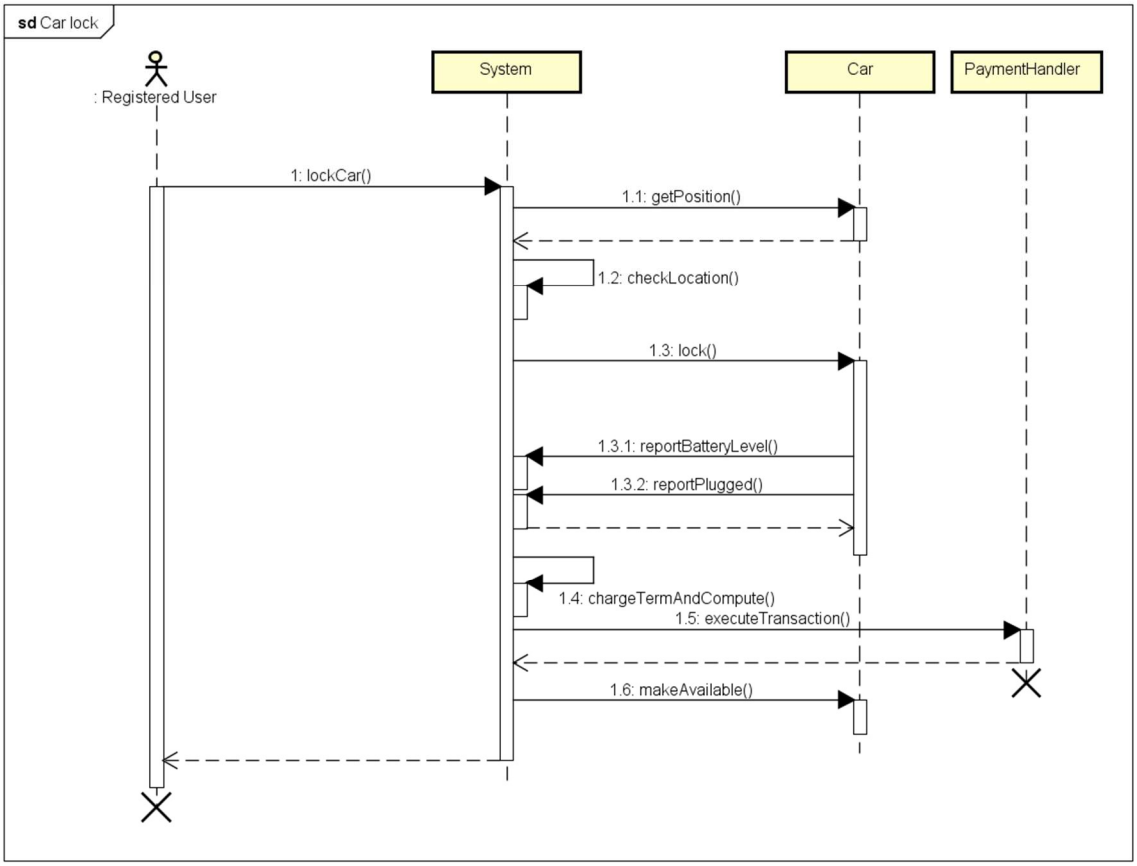
6.3.4 Reserve a car



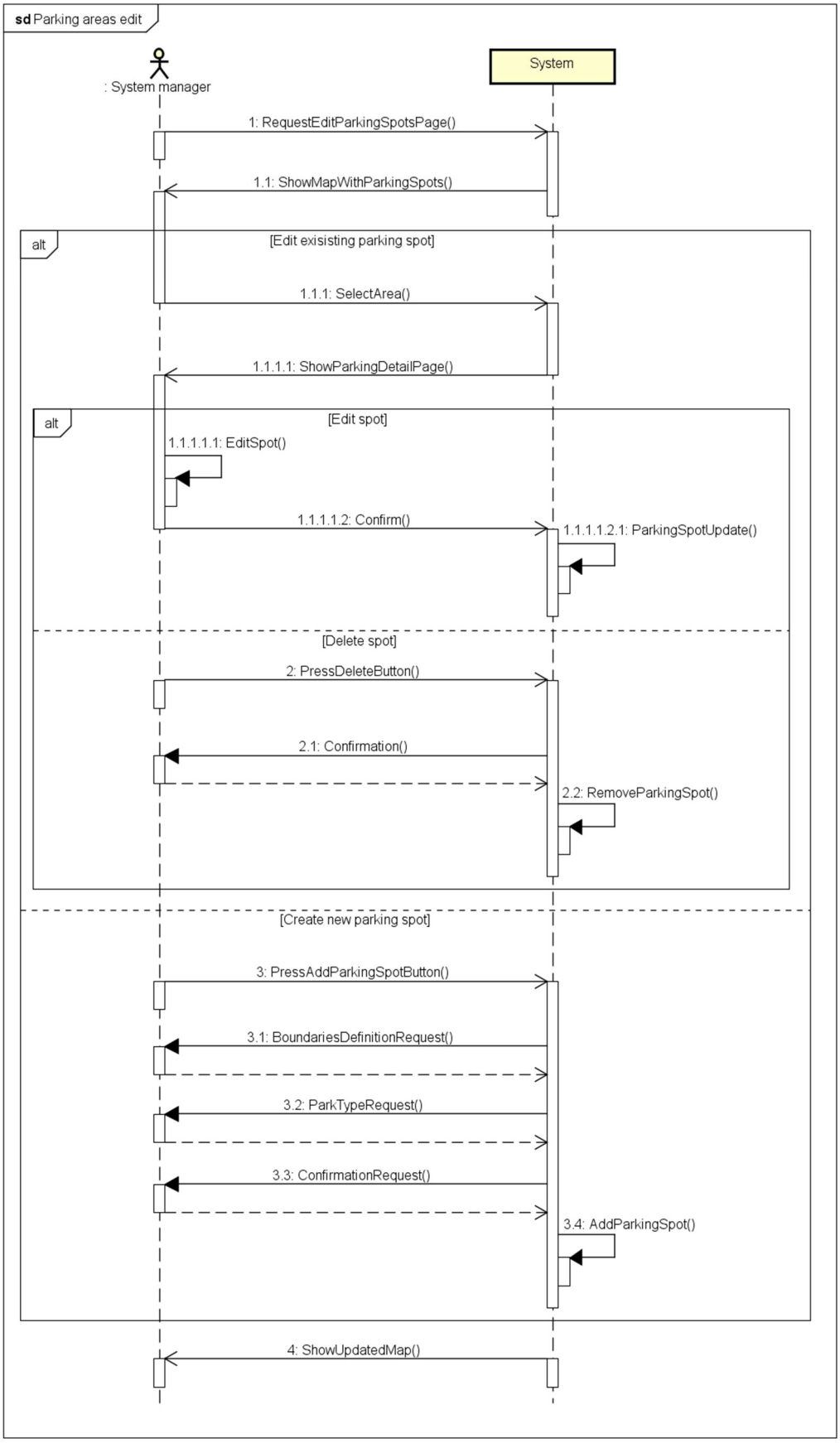
6.3.5 Car activation/unlock



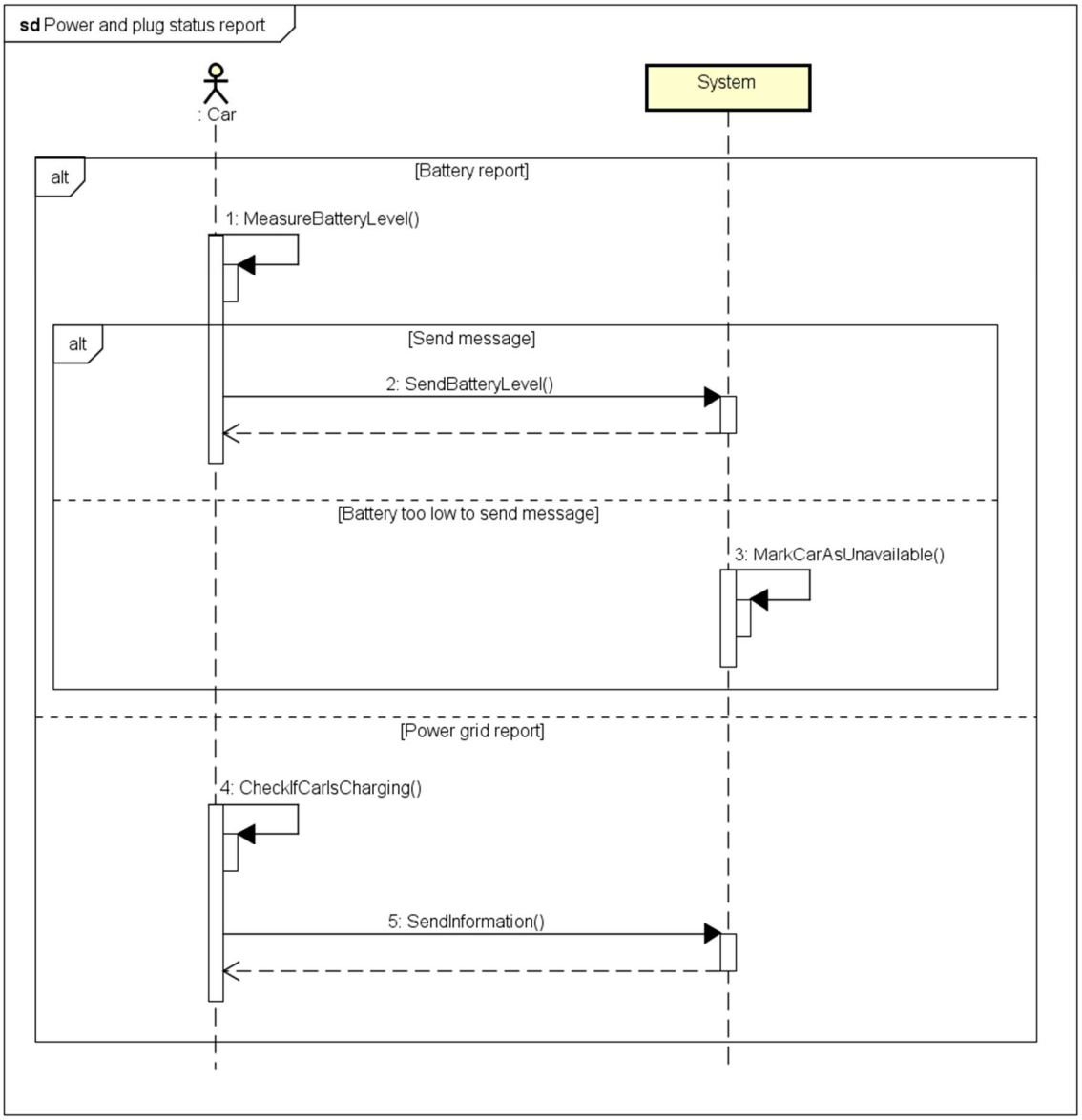
6.3.6 Car lock


sd Car lock
: Registered User
System
Car
PaymentHandler
1: lockCar()
1.1: getPosition()
1.2: checkLocation()
1.3: lock()
1.3.1: reportBatteryLevel()
1.3.2: reportPlugged()
1.4: chargeTermAndCompute()
1.5: executeTransaction()
1.6: makeAvailable()

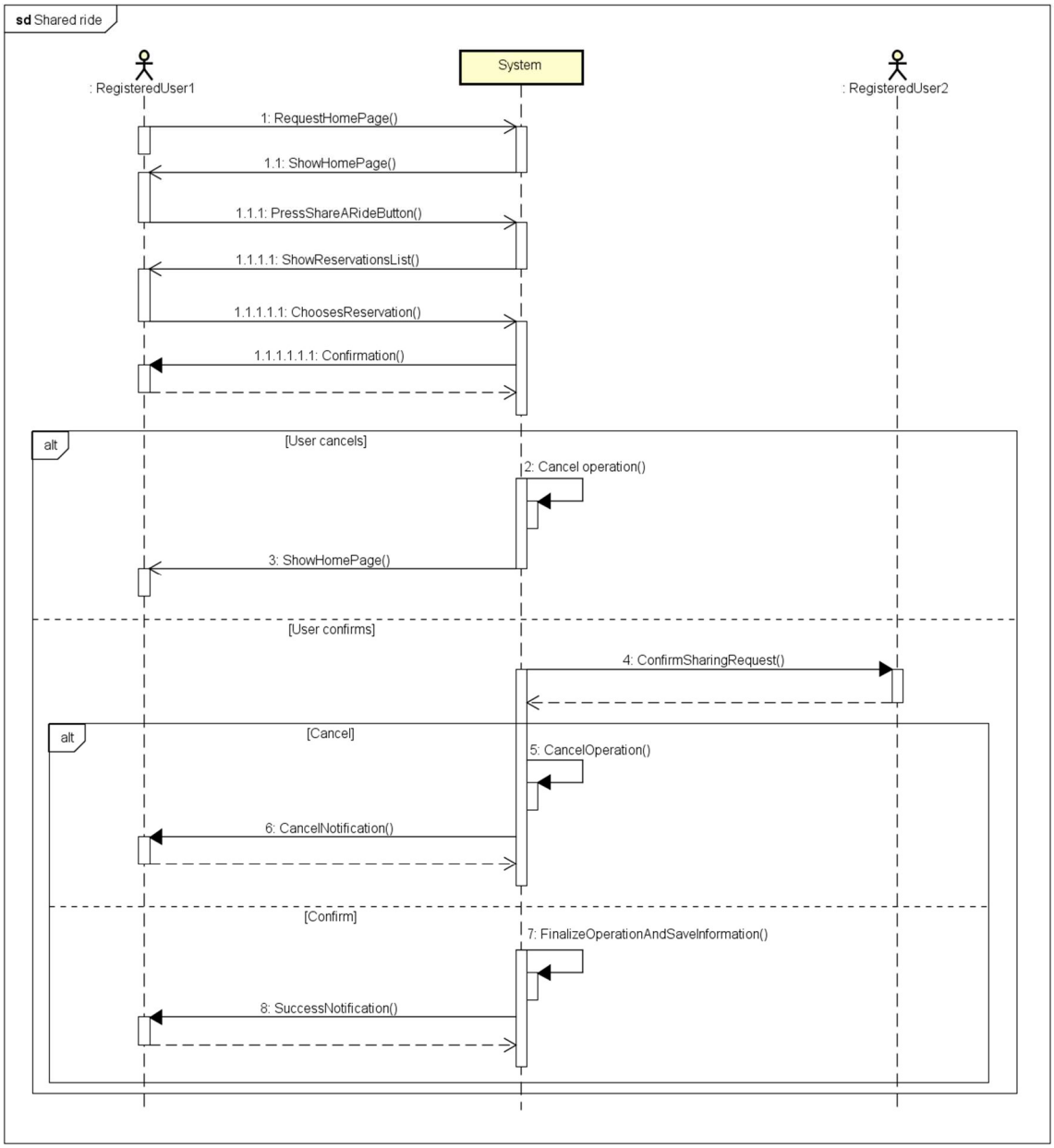
6.3.7 Parking areas edit



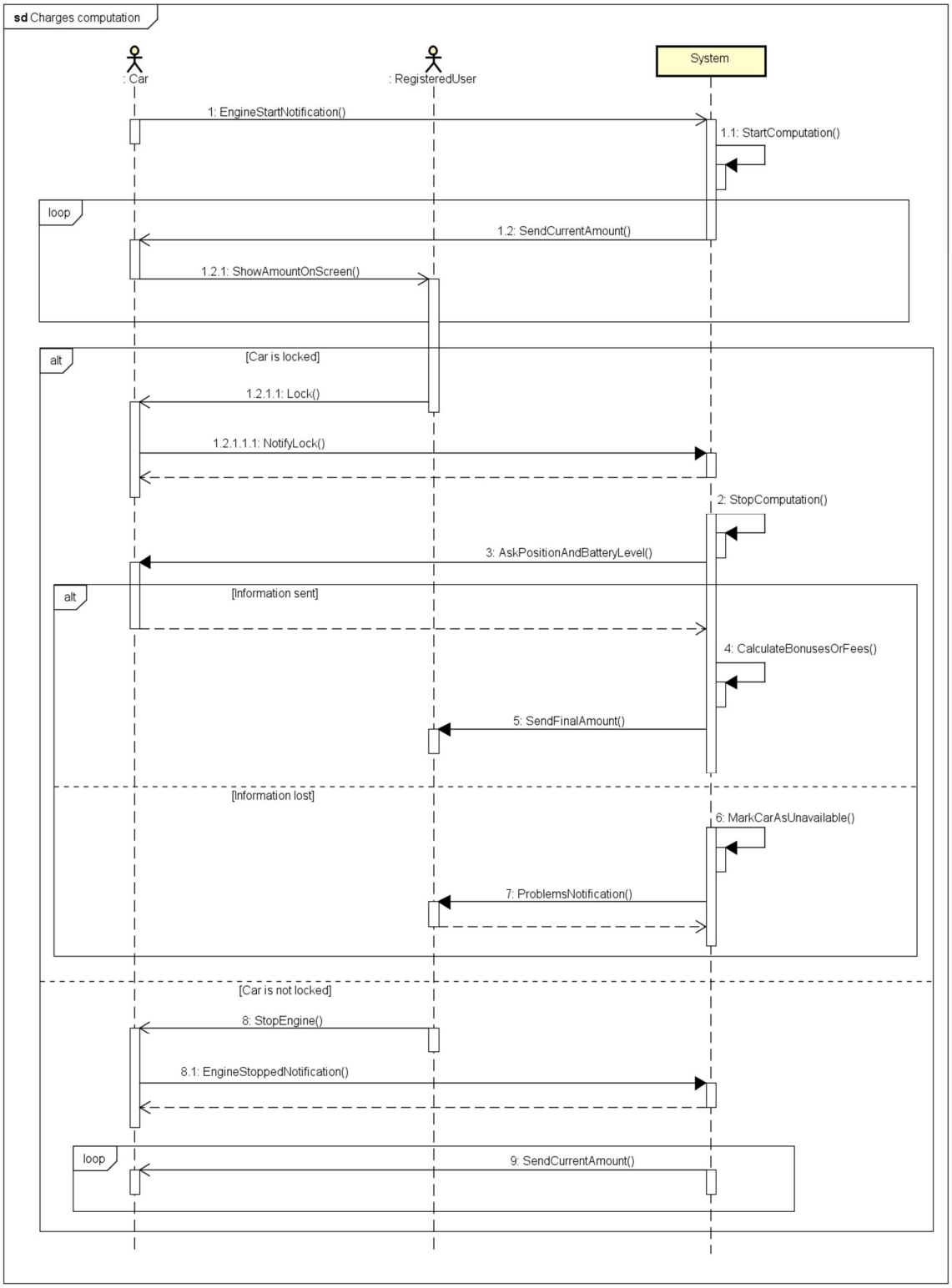
6.3.8 Power and plug status report



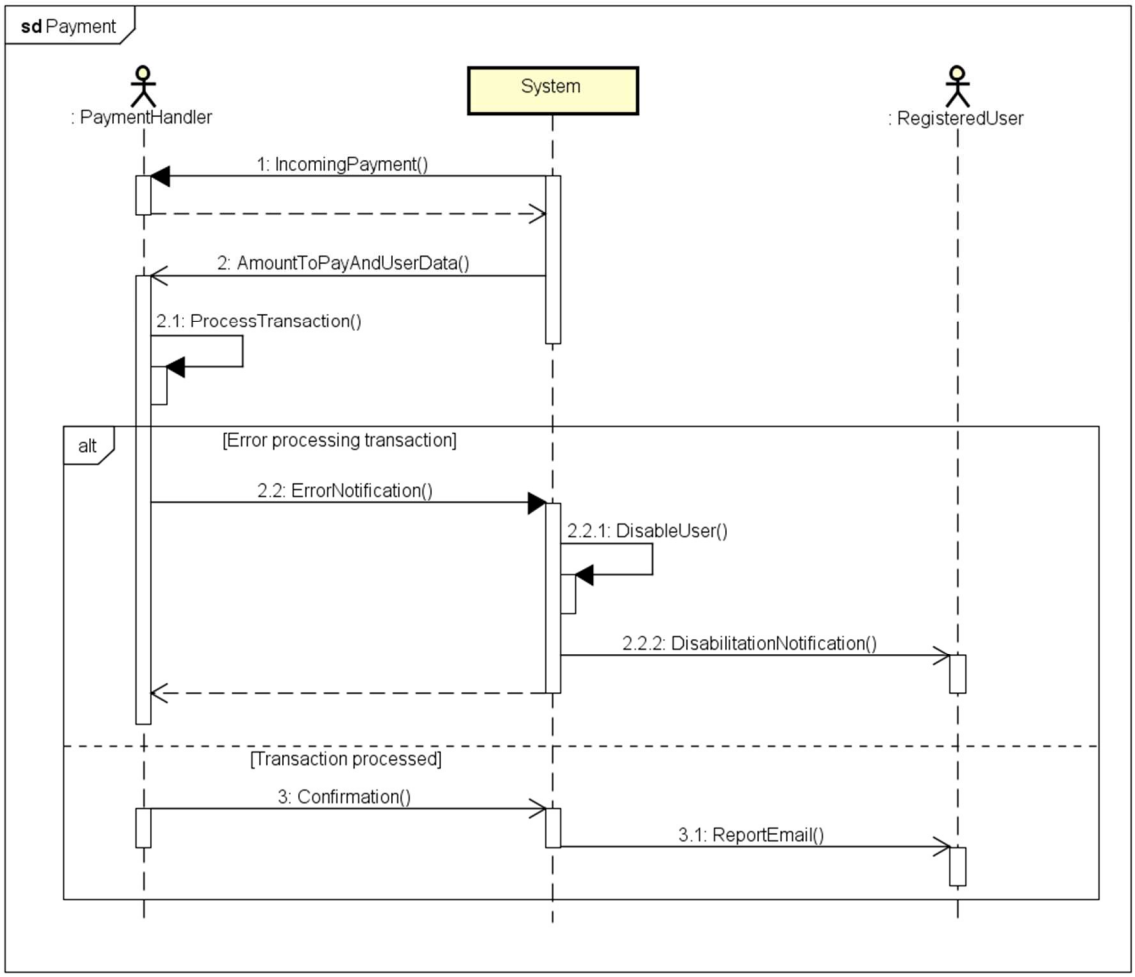
6.3.9 Shared ride



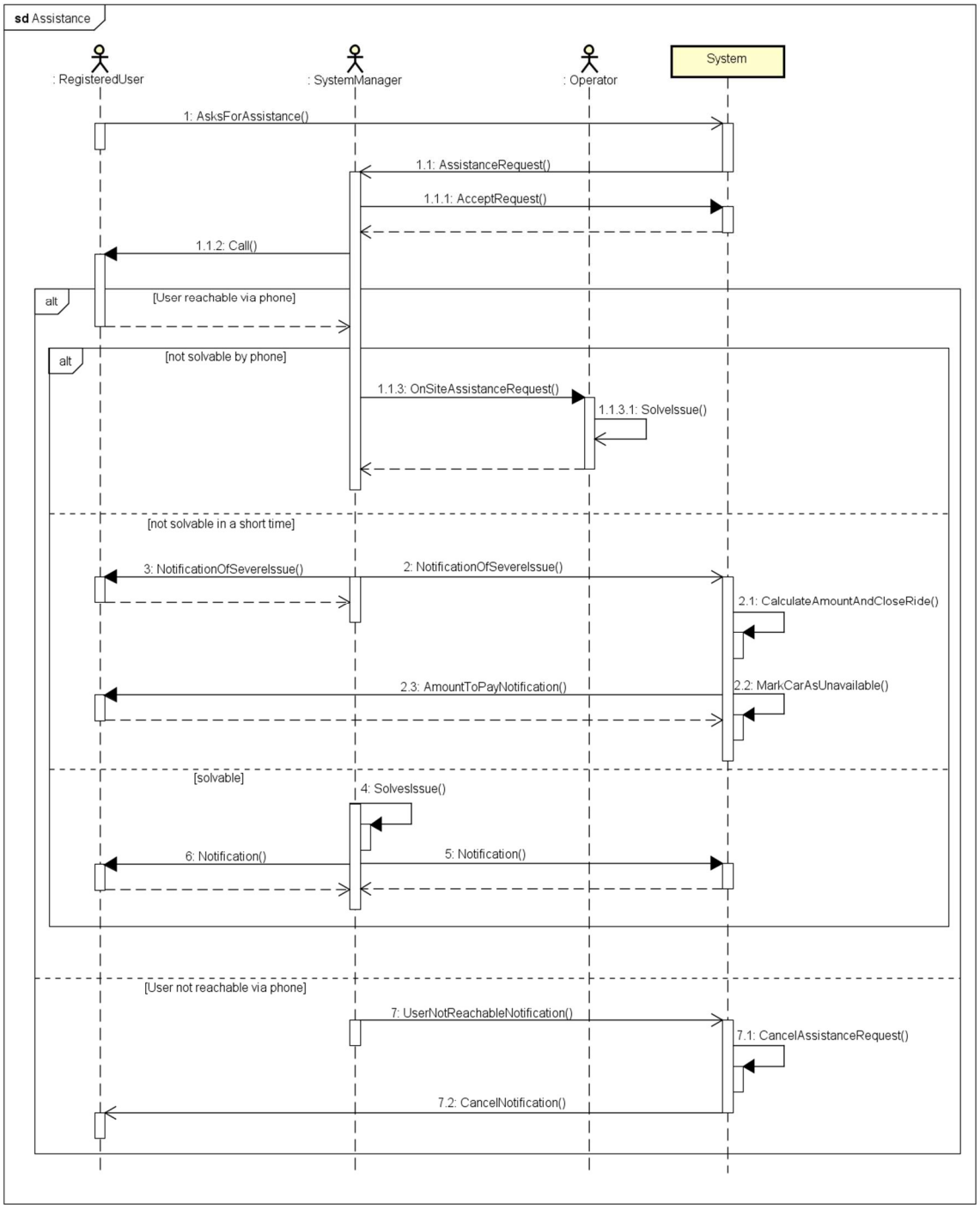
6.3.10 Charges computation



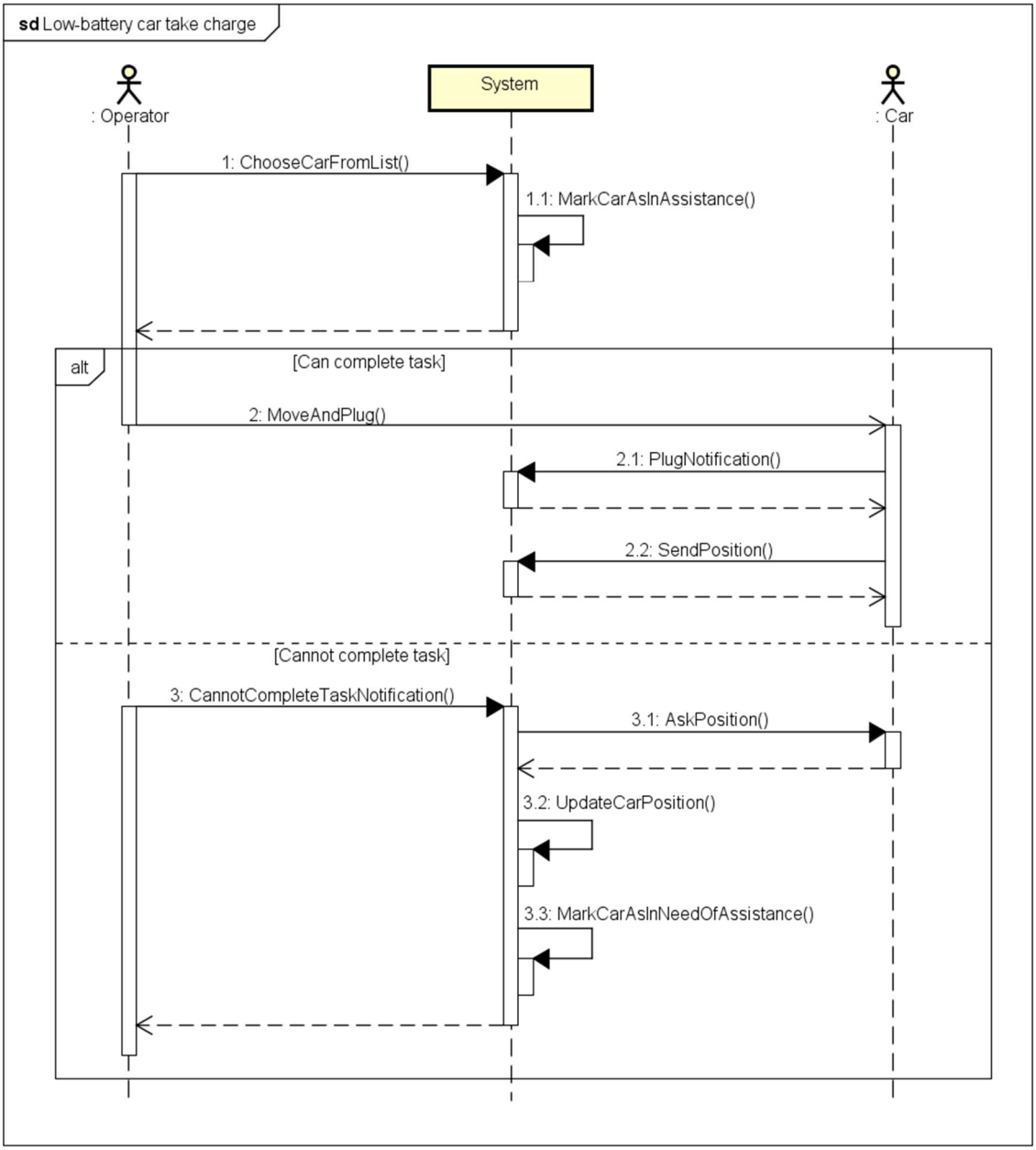
6.3.11 Payment



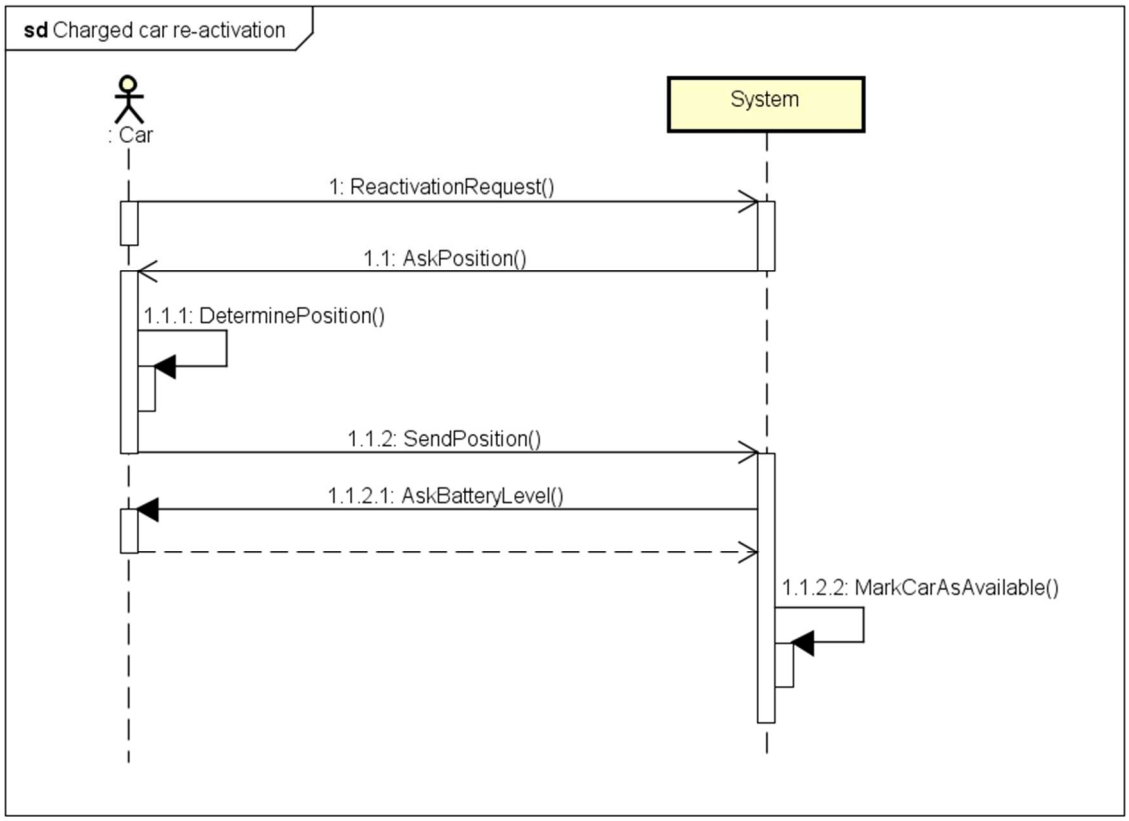
6.3.12 Assistance



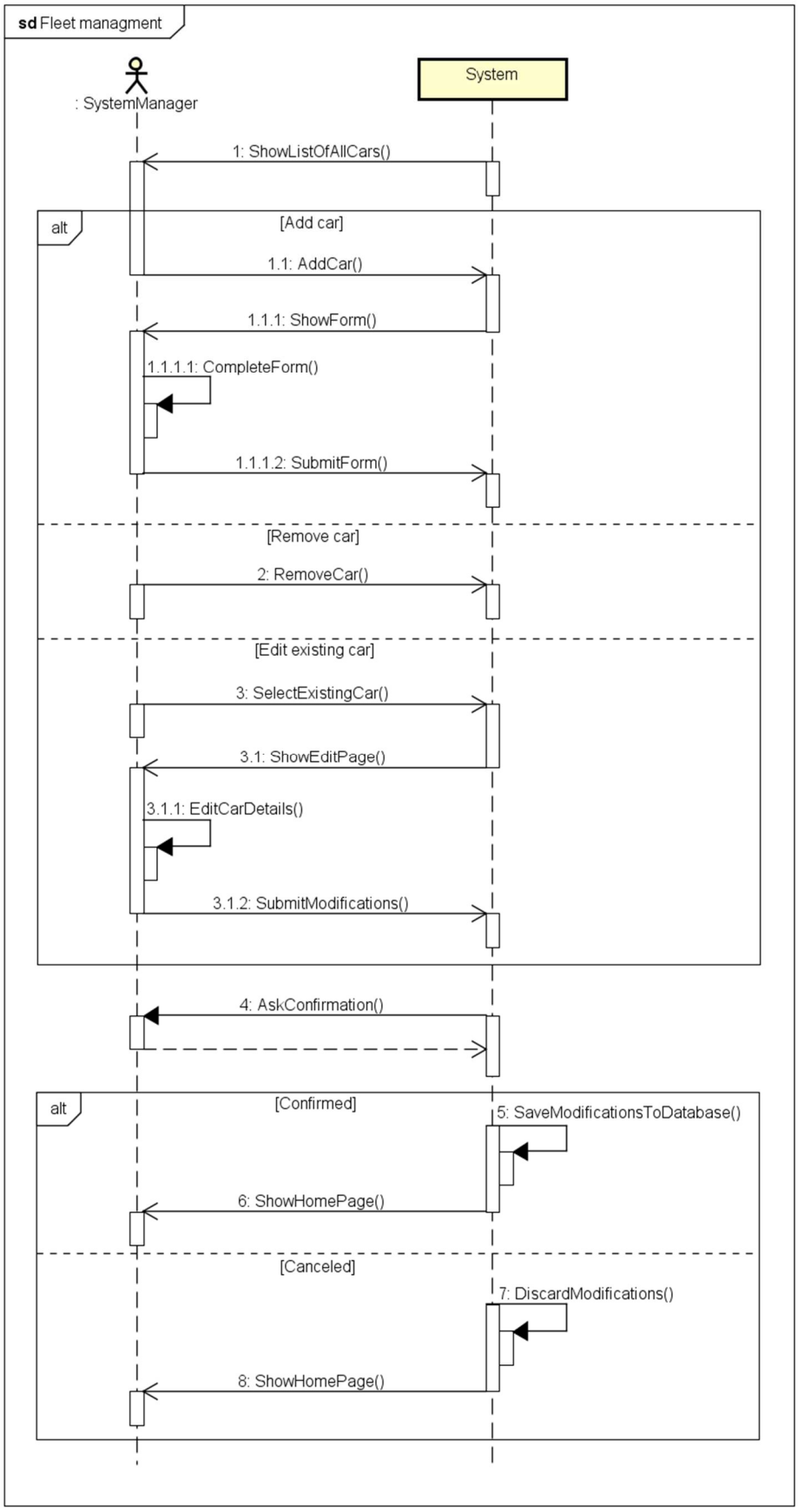
6.3.13 Low-battery car take charge



6.3.14 Charged car re-activation



6.3.15 Fleet management



7. Alloy model

Based upon the class diagram and the use cases defined previously in this document, a coherent Alloy model is hereby presented. Signatures and enums are presented first, then all the mandatory facts are listed, followed by some assertions that cover the most significant flaws avoided by facts themselves. Finally, some example predicates are shown – including the ones that are necessary to generate some example worlds.

Further on in this section, several instances of the model will be reported graphically.

7.1 Alloy code

```
open util/Boolean

-------------------------------------------------------------------------------
////// SIGNATURES //////

sig Plate {}

sig DateAndTime {}

abstract sig User {}

sig Operator extends User {
	carAssigned: lone Car,
	requestAssigned: lone AssistanceRequest
}

sig AssistanceRequest {
	user: one RegisteredUser,
	location: one Location
}

enum CarState {
	reservable, charging, reserved, outOfOrder, lowBattery
}

enum SurchargeOrDiscount {
	surcharge, discount
}

abstract sig ExtraCharge {
	type: one SurchargeOrDiscount,
	percentage: one Int
}
```

```
lone sig SharingDiscount extends ExtraCharge {}

lone sig BatteryLevelDiscount extends ExtraCharge {}

lone sig PlugDiscount extends ExtraCharge {}

lone sig CarStatusPenalty extends ExtraCharge {}

one sig Fleet {
	cars: set Car
}

sig Car {
	plate: one Plate,
	status: one CarState,
	position: one Location,
	batteryLeft: one Int,
	plugged: one Bool
}
{
	batteryLeft > 0
	batteryLeft < 100
}

sig CarLeftConditions {
	batteryLeft: one Int,
	plugged: one Bool,
	distantFromPlug: one Bool
}
{
	batteryLeft > 0
	batteryLeft < 100
}

sig Location {}

sig Guest extends User {}

sig RegisteredUser extends User {
	activeReservation: lone Reservation,
	succededReservations: set Reservation,
	rides: set Ride,
	disabled: one Bool
}

sig SystemManager extends User {}

sig ParkingArea {
	bounds: set Location
}
```

```
one sig Parkings {
	normalAreas: set ParkingArea,
	plugAreas: set ParkingArea
}

sig Ride {
	totalPrice: Int,
	reservation: one Reservation,
	timeStart: one DateAndTime,
	timeEnd: one DateAndTime,
	vehicle: one Car,
	sharedUsers: set RegisteredUser,
	park: one ParkingArea,
	carLeftState: one CarLeftConditions,
	extras: set ExtraCharge,
	paymentOk: one Bool
}
{
	totalPrice > 0
}

sig Reservation {
	time: one DateAndTime,
	car: one Car
}



--------------------------------------------------------------------------------
////// FACTS //////

--------------------------------------------------------------------------------
//Users, rides and reservations relations properties

//Disabled user must have one ride whose payment is not ok and cannot make
reservations
fact DisabledUserMustHaveProblems {
	all u: RegisteredUser | u.disabled = True implies (one r: Ride | r in
	    u.rides and r.paymentOk = False) and #u.activeReservation = 0
	all r: Ride | r.paymentOk = False implies (one u: RegisteredUser | r in
	    u.rides and u.disabled = True)
}

//No more than 1 active reservation for same car
fact OneReservationPerCar {
	no disj u1, u2 : RegisteredUser |
	    (some r1 : u1.activeReservation, r2 : u2.activeReservation | r1.car
	    = r2.car)
}
```

```
//No user is a "shared user" of one of his/her rides
fact UsersShareOnlyWithOthers {
	no r: Ride, u: RegisteredUser | r in u.rides and u in r.sharedUsers
}


//A reservation is either active or past (succeded)
fact ReservationsAreActiveOrPast {
	no u: RegisteredUser | (some r: Reservation | r in u.activeReservation
	    and r in u.succededReservations)
}

//There is no reservation that corresponds to multiple users
fact noOneResForMultiUsers {
	no r: Reservation | (some disj u1, u2: RegisteredUser |
		((r in u1.succededReservations or r in u1.activeReservation) and
		(r in u2.succededReservations or r in u2.activeReservation)))
}

//No reservation for multiple rides
fact ReservationsAreExclusive {
	all disj r1, r2: Ride | r1.reservation != r2.reservation
}

//Car in a ride is the same of the reservation
fact RideAndResSameCar {
	all r: Ride | r.reservation.car = r.vehicle
}

//There is no ride without a user
fact RidesAlwaysHaveAUser {
	all r: Ride | (one u: RegisteredUser | r in u.rides)
}

//There is no ride without corresponding succeded reservation and vice versa
fact RidesAndPastReservationsProperty {
	all r: Ride, res: Reservation, u: RegisteredUser |
		(res = r.reservation and r in u.rides implies
		res in u.succededReservations) and
		(res = r.reservation and res in u.succededReservations implies
		r in u.rides)
}

//There is no ride linked to more than 1 user
fact EveryRideOneUser {
	all disj u1, u2: RegisteredUser | #(u1.rides & u2.rides) = 0
}
```

```
--------------------------------------------------------------------------------
//Parking areas properties

//Normal parking spots and plug parking spots do not collide (i.e. have always
//different locations)
fact PlugAndSpotsDifferent {
	all disj pa1, pa2: ParkingArea | #(pa1.bounds & pa2.bounds) = 0
	    #(Parkings.normalAreas & Parkings.plugAreas) = 0
}

//There is no parking spot that is not neither a normal nor a plug spot
fact AllParkingsHaveType {
	all pa: ParkingArea | pa in Parkings.normalAreas or pa in
	    Parkings.plugAreas
}

//If a car is plugged, then it must be in a plug parking spot
fact PluggedCars {
	all c: Car | c.plugged.isTrue implies (some p: ParkingArea | p in
	    Parkings.plugAreas and c.position in p.bounds)
}

--------------------------------------------------------------------------------
//Service management

//Cars assigned to operators must have some problem
fact OperatorsCars {
	all c: Car | (one o: Operator | c = o.carAssigned) implies c.status in
	    lowBattery + outOfOrder
}

//Operators have only one duty active
fact OperatorsExclusive {
	all o: Operator | #o.carAssigned = 1 implies #o.requestAssigned = 0
	all o: Operator | #o.requestAssigned = 1 implies #o.carAssigned = 0
}

//Cars that need assistance can only be assigned to one operator
fact MaxOneOperatorPerCar {
	all disj o1, o2: Operator | #(o1.requestAssigned & o2.requestAssigned) =
0
	all disj o1, o2: Operator | #(o1.carAssigned & o2.carAssigned) = 0
}

//There are no users that have more than 1 pending requests
fact OneRequestPerUser {
	all disj a1, a2: AssistanceRequest | #(a1.user & a2.user) = 0
}
```

```
-----------------------------------------------------------------------------
//Car properties and constraints

//All cars are in the fleet
fact CarFleet {
	all c: Car | c in Fleet.cars
}

//Cars have different (uniques) plates
fact UniquePlates {
	all disj c1, c2: Car | c1.plate != c2.plate
}

//If a car has low battery and it is not plugged, it must be broken or marked
as lowBattery
fact UnpluggedLowBatteryCarsNotReservable {
	all c: Car | (c.batteryLeft < 20 and c.plugged = False) implies
	    (c.status = outOfOrder or c.status = lowBattery)
}

//If a car has low battery but it is plugged, it may be outOfOrder or
//charging, waiting for its battery to reach 80%
fact PluggedLowBatteryCarsNotReservable {
	all c: Car | (c.batteryLeft < 20 and c.plugged = True) implies (c.status
	    = outOfOrder or c.status = charging)
}

//Two cars cannot be in the same position at the same time
fact CarsHaveDifferentPositions {
	all disj c1, c2: Car | c1.position != c2.position
}

//If a car is reservable or already reserved, it must be in a parking spot
fact ReservableCarsAreParked {
	all c: Car | (c.status = reservable or c.status = reserved) implies
	    c.position in (Parkings.plugAreas.bounds +
	    Parkings.normalAreas.bounds)
}

//Cars in active reserv. can only be "reserved" (working and not reservable)
fact ReservedCarsNotReservableOrBroken {
	no u: RegisteredUser, c: Car, r: Reservation | c = r.car and r =
	    u.activeReservation and c.status != reserved
}

//If a car is "reserved", than there must exist a user that has an active
reservation on it
fact ReservedCarsHaveAReservingUser {
	all c: Car | (c.status = reserved implies (some u: RegisteredUser |
	    u.activeReservation.car = c))
}
```

```
//If a car has no special conditions, it must be either outOfOrder or reserv.
fact CarsArePossiblyReservable {
	all c: Car | c.batteryLeft > 20 and c.status != reserved and c.status !=
	    outOfOrder implies c.status = reservable
}

-------------------------------------------------------------------------------
//Discounts or surcharge properties

//Car left conditions properties
fact CarLeftConditionsProperties {
	all clc: CarLeftConditions | (some rd: Ride | clc = rd.carLeftState)
	all clc: CarLeftConditions, rd: Ride | clc = rd.carLeftState and
	    clc.plugged = True implies rd.park in Parkings.plugAreas and
	    clc.distantFromPlug = False
	all rd: Ride | rd.park in Parkings.plugAreas implies (one clc:
	    CarLeftConditions | clc = rd.carLeftState and clc.plugged = True and
	    clc.distantFromPlug = False)

}

//Share discount
fact SharingDiscountProperties {
	all sd: SharingDiscount | (some r: Ride | sd in r.extras) and
	    sd.percentage = 10 and sd.type = discount
	all r: Ride, sd: SharingDiscount | sd in r.extras implies #r.sharedUsers
	    > 2
	all r: Ride | #r.sharedUsers > 2 implies one sd: SharingDiscount | sd in
	    r.extras
}

//PlugDiscount
fact PlugDiscountProperties {
	all pd: PlugDiscount | (some r: Ride | pd in r.extras) and pd.percentage
	    = 30 and pd.type = discount
	all r: Ride, pd: PlugDiscount | pd in r.extras implies
	    r.carLeftState.plugged = True
	all r: Ride | r.carLeftState.plugged = True implies one pd: PlugDiscount
	    | pd in r.extras
}

//BatteryLevelDiscount
fact BatteryLevelDiscountProperties {
	all bld: BatteryLevelDiscount | (some r:Ride | bld in r.extras) and
	    bld.percentage = 20 and bld.type = discount
	all r: Ride, bld: BatteryLevelDiscount | bld in r.extras implies
	    r.carLeftState.batteryLeft > 50
	all r: Ride | r.carLeftState.batteryLeft > 50 implies one bld:
	    BatteryLevelDiscount | bld in r.extras
}
```

```
//CarStatusPenalty
fact CarStatusPenaltyProperties {
	all csp: CarStatusPenalty | (some r: Ride | csp in r.extras) and
	    csp.percentage = 30 and csp.type = surcharge
	all r: Ride, csp: CarStatusPenalty | csp in r.extras implies
	    (r.carLeftState.batteryLeft < 20 or r.carLeftState.distantFromPlug =
	    True)
	all r: Ride | (r.carLeftState.batteryLeft < 20 or
	    r.carLeftState.distantFromPlug = True) implies one csp:
	    CarStatusPenalty | csp in r.extras
}

-------------------------------------------------------------------------
//Constraints on time (before, after) not possible with this model. We simply
//state some constraint to get more plausible worlds:
fact NoInstantaneousRide {
	all r:Reservation, rd:Ride | r = rd.reservation implies r.time !=
	rd.timeStart and r.time != rd.timeEnd
}

fact NoZeroTimeRide {
	all rd:Ride | rd.timeStart != rd.timeEnd
}

fact NoSimultaneousRides {
	no disj r1, r2: Ride | some u: RegisteredUser | (r1 in u.rides and r2 in
	u.rides) and r1.timeStart = r2.timeStart
}

-------------------------------------------------------------------------
////// ASSERTIONS //////

assert noOneCarMultipleRes {
	no disj u1, u2 : RegisteredUser |
	  (some r1 : u1.activeReservation, r2 : u2.activeReservation | r1.car =
	  r2.car)
}

check noOneCarMultipleRes for 5

assert noSelfShared {
	no r: Ride, u: RegisteredUser | r in u.rides and u in r.sharedUsers
}

check noSelfShared for 5

assert noOneResForMultiUsers {
	no r: Reservation | (some disj u1, u2: RegisteredUser |
	  ((r in u1.succededReservations or r in u1.activeReservation) and
	  (r in u2.succededReservations or r in u2.activeReservation)))
}
```

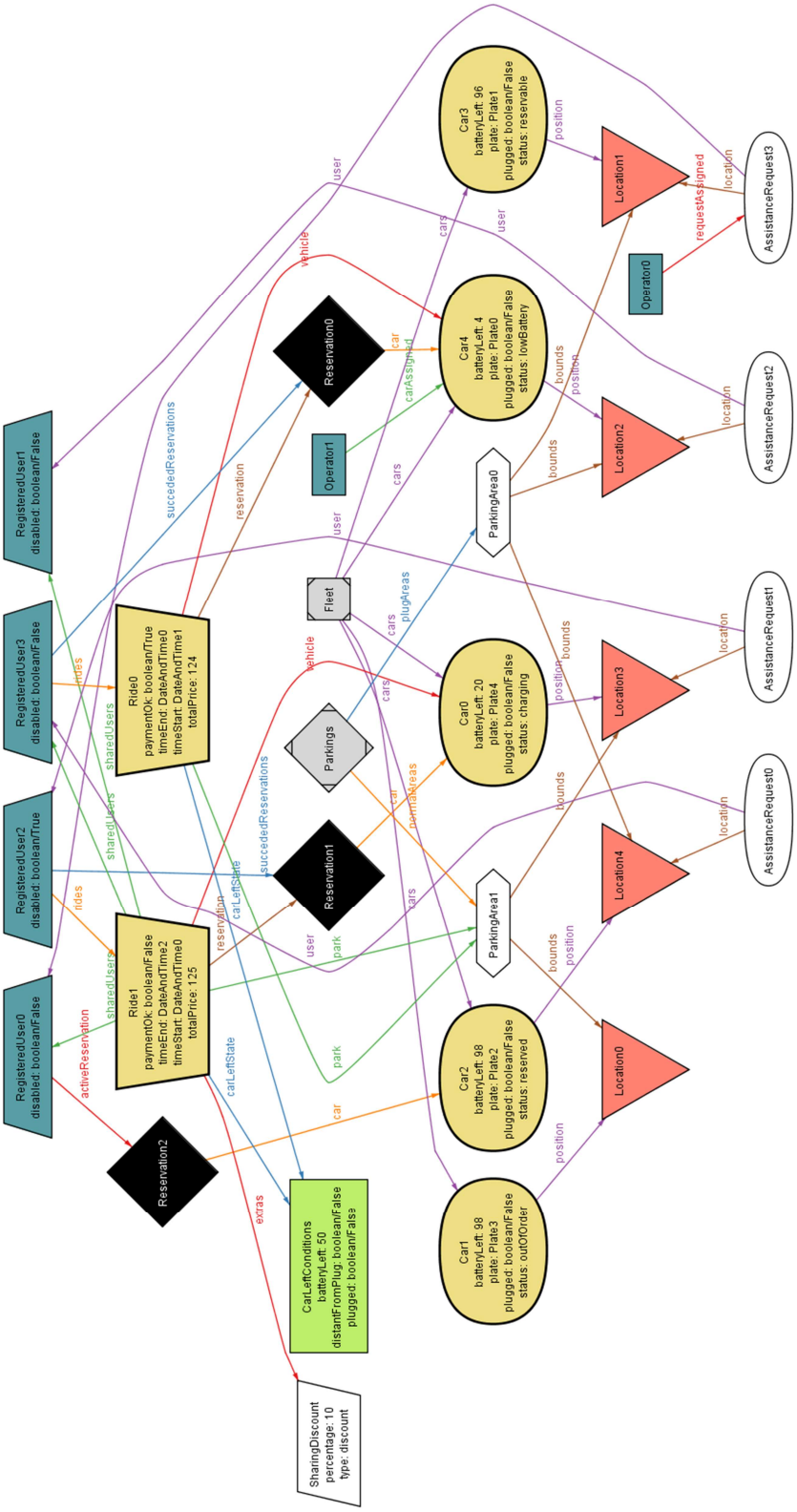
```
check noOneResForMultiUsers for 5

assert noDifferentCarsForResAndRide {
	no r: Ride, res: Reservation | res = r.reservation and not res.car =
	  r.vehicle
}

check noDifferentCarsForResAndRide for 5

assert NoRideWithoutUser {
	no r: Ride | (no u: RegisteredUser | r in u.rides)
}

check NoRideWithoutUser for 5

assert NoRideWithoutReservation {
	no r: Ride, res: Reservation, u: RegisteredUser |
	    ((r in u.rides and res = r.reservation and not res in
	    u.succededReservations) or
	    (res in u.succededReservations and res = r.reservation and not r in
	    u.rides))
}

check NoRideWithoutReservation for 5

assert NoCollidingSpots {
	no disj pa1, pa2: ParkingArea | some l: Location | l in (pa1.bounds &
	  pa2.bounds)
}

check NoCollidingSpots for 5

--------------------------------------------------------------------------------
////// PREDICATES //////

--------------------------------------------------------------------------------
//Service management predicates

//Predicate: add and remove parking spots (for service managers)
pred addNormalParkingArea(p: ParkingArea, pp, pp': Parkings) {
	p not in pp.normalAreas implies pp'.normalAreas = pp.normalAreas + p
}

pred addPlugParkingArea(p: ParkingArea, pp, pp': Parkings) {
	p not in pp.plugAreas implies pp'.plugAreas = pp.plugAreas + p
}

run addNormalParkingArea for 5

run addPlugParkingArea for 5
```

```
pred removeParkingArea(p: ParkingArea, pp, pp': Parkings) {
	p in pp.normalAreas implies pp'.normalAreas = pp.normalAreas - p and
	p in pp.plugAreas implies pp'.plugAreas = pp.plugAreas - p
}

run removeParkingArea for 5

//Predicate: add and remove cars to fleet
pred addCarToFleet(c: Car, fl, fl': Fleet) {
	c not in fl.cars implies fl'.cars = fl.cars + c
}

run addCarToFleet for 5

pred removeCarFromFleet(c: Car, fl, fl': Fleet) {
	c in fl.cars implies fl'.cars = fl.cars - c
}

run removeCarFromFleet for 5


-------------------------------------------------------------------------------
//Show predicates

pred show() {
	#RegisteredUser > 1
	#Ride > 1
	#Reservation > 1
	#activeReservation > 0
	#SharingDiscount > 0
	//some c: Car | c.batteryLeft < 20 and c.plugged = True
	some c: Car | c.status = lowBattery
	some c: Car | c.status = reservable
	some c: Car | c.status = reserved
	some c: Car | c.status = outOfOrder
	some c: Car | c.status = charging
	//all r: RegisteredUser | some rd: Ride | rd in r.rides
	#Operator > 1
	#carAssigned > 0
	#requestAssigned > 0

}

run show for 6 but 8 int

pred showDisabled() {
	some u: RegisteredUser | u.disabled = True
}

run showDisabled for 6 but 8 int
```

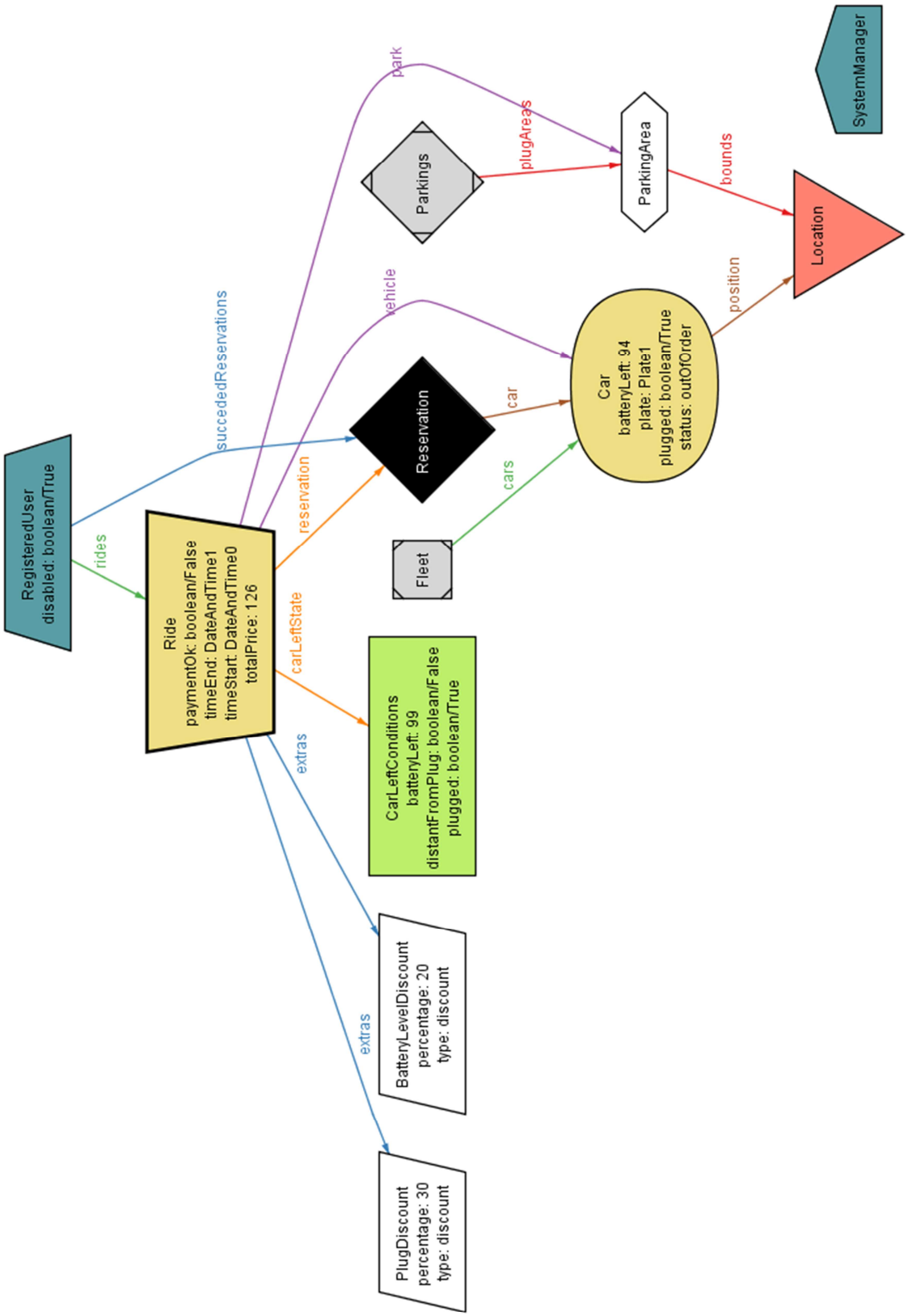
```
pred showExtras() {
       #SharingDiscount > 0
       #CarStatusPenalty > 0
       #PlugDiscount > 0
       #BatteryLevelDiscount > 0
}

run showExtras for 6 but 8 int
```

7.2 Alloy generated worlds

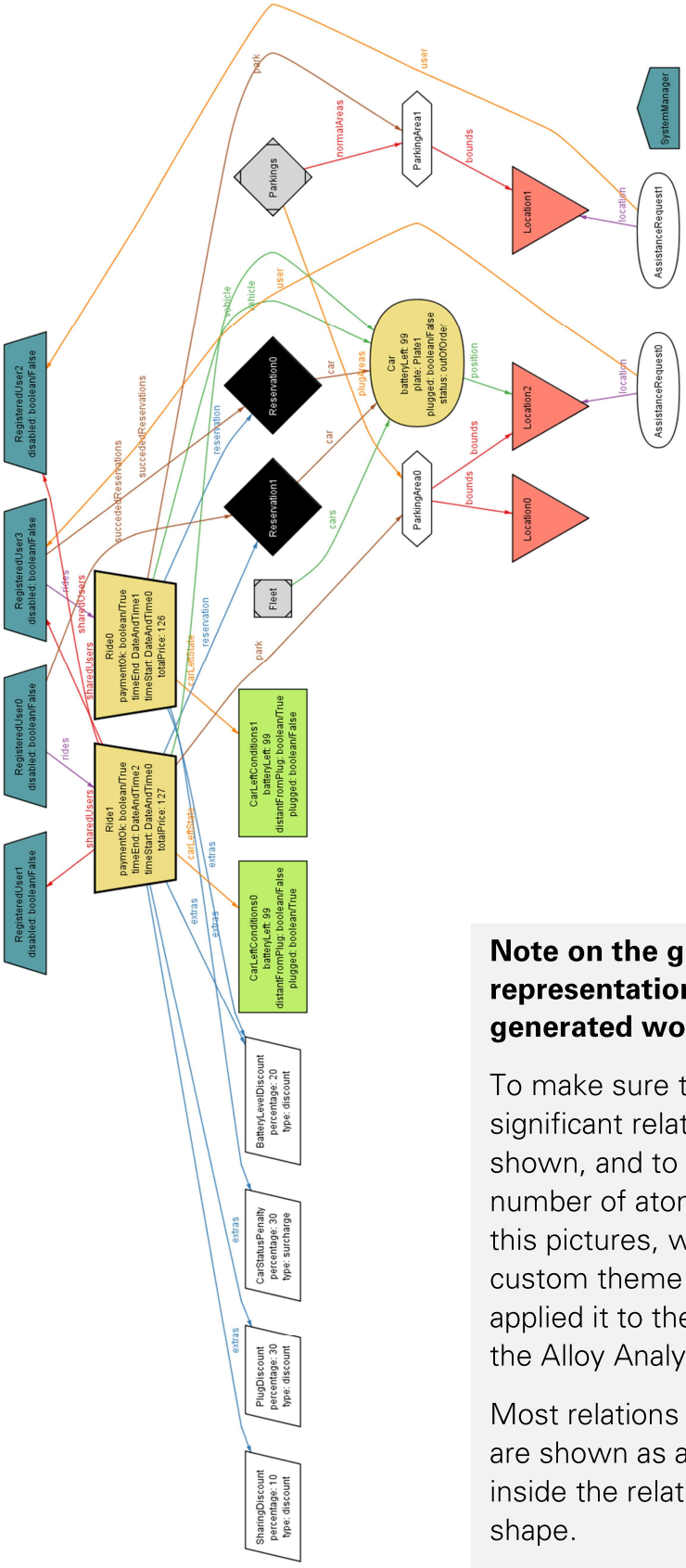
Running the first show predicate, an almost complete world is generated: the aim here is to show a situation where almost all signatures and relations are involved.



The second show predicate is aimed at showing a much simpler situation instead, where a user has been disabled because of some issue in paying a ride.



Finally, the third show predicate is useful to show how *extras* in user payments work, by exploiting all possible discounts and/or surcharges.



Note on the graphical representation of generated worlds:

To make sure that only significant relations are shown, and to reduce the number of atoms present in this pictures, we created a custom theme and then we applied it to the outputs of the Alloy Analyzer by MIT.

Most relations and atoms are shown as attributes inside the relative signature shape.

8. Appendices

8.1 Tools used

- Microsoft Office Word 2016
  https://products.office.com/it-it/home

  For redacting, reviewing, layout and graphic design of this document

- Astah Professional
  http://astah.net/editions/professional

  For UML modeling, Use Case Diagram, State Charts, Class Diagram, Sequence Diagrams

- Balsamiq Mockups
  http://balsamiq.com/products/mockups/

  For UI mockups design

- Alloy Analyzer
  http://alloy.mit.edu/alloy/

  For alloy modeling and consistency tests

8.2 Hours of work

The present document required almost the same time for both the curators (Leonardo Chiappalupi, Ivan Bugli).

The total time spent on the creation of the paper is: ~40 Hours.