



POLITECNICO DI MILANO

SOFTWARE ENGINEERING II PROJECT

SAFESTREETS

Requirements Analysis and Specifications Document

Authors:

Mattia CALABRESE
Federico CAPACCIO
Amedeo CAVALLO

Professor:

Elisabetta DI NITTO

October 23, 2019

version 1.0

Contents

1	Introduction	1
1.1	Purpose of this document	1
1.2	Scope	1
1.2.1	Goals	1
1.3	Glossary	2
1.3.1	Definitions	2
1.3.2	Acronyms	3
1.3.3	Abbreviations	3
1.4	Document overview	3
2	Overall description	4
2.1	Product perspective	4
2.1.1	System interfaces	4
2.1.2	User interfaces	5
2.1.3	Hardware interfaces	5
2.1.4	Software interfaces	5
2.1.5	Communication interfaces	5
2.2	Product functions	6
2.3	User Characteristics	6
2.4	Constraints	7
2.5	Assumptions	7
3	Specific Requirements	8
3.1	Functional Requirements	8
3.1.1	Goals	8
3.2	Performance Requirements	10
3.3	Software System Attributes	10
3.3.1	Availability	10
3.3.2	Security	10
3.3.3	Portability	10
4	Use cases identification	11
4.1	Scenarios	11
4.1.1	Scenario 1	11
4.1.2	Scenario 2	11
4.1.3	Scenario 3	11
4.1.4	Scenario 4	11
4.1.5	Scenario 5	11
4.1.6	Scenario 6	11
4.2	Use case diagram	12
4.3	Use cases description	14
4.3.1	Registration	14
4.3.2	Authentication	16
4.3.3	View cars on the map	18
4.3.4	Car reservation	20
4.3.5	Car unlock	21
4.3.6	Car rent	23
4.3.7	Rent payment	26

4.3.8	Money saving option	29
4.3.9	Visualization of not available cars	31
4.3.10	Tag a car as available	33
4.3.11	Visualization of users information	35
4.3.12	View users payments and rents history	36
4.3.13	Mark or unmark a user as banned	37
4.3.14	Tag a car as not available	38
4.4	UML class diagram	40
Appendices		41
A Alloy model		41
A.1	Source code	41
A.2	Generated worlds	49
B Software and tools used		51
C Hours of Work		51
D Changelog		51

List of Figures

1	Class Diagram	4
2	Overview of system interfaces	4
3	Use case diagram	12
4	<i>Registration</i> sequence diagram	15
5	<i>Authentication</i> sequence diagram	17
6	<i>View cars on the map</i> sequence diagram	19
7	<i>Car unlock</i> sequence diagram	22
8	<i>Car rent</i> sequence diagram	24
9	<i>One Euro fee</i> sequence diagram	24
10	Car status FSM	25
11	<i>Rent payment</i> sequence diagram	28
12	<i>Money saving option</i> sequence diagram	30
13	<i>Visualization of not available cars</i> sequence diagram	32
14	<i>Tag a car as available</i> sequence diagram	34
15	<i>Tag a car as Not Available</i> sequence diagram	39
16	UML class diagram	40
17	Alloy execution result	48
18	First alloy generated world	49
19	Second alloy generated world	50

List of Tables

1	<i>Registration</i> use case description	15
2	<i>Authentication</i> use case description	16
3	<i>View cars on the map</i> use case description	18
4	<i>Car reservation</i> use case description	20

5	<i>Car unlock</i> use case description	21
6	<i>Car rent</i> use case description	23
7	<i>Rent payment</i> use case description	27
8	<i>Money saving option</i> use case description	29
9	<i>Visualization of not available cars</i> use case description	31
10	<i>Tag a car as not available</i> use case description	33
11	<i>Visualization of users information</i> use case description	35
12	<i>View users payments and rents history</i> use case description	36
13	<i>Mark or unmark a user as banned</i> use case description	37
14	<i>Tag a car as not available</i> use case description	38

1 Introduction

1.1 Purpose of this document

The purpose of a **Requirement Analysis and Specifications Document** is the process of discovering the purpose for which a software system was intended, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. [1] It is also concerned with the relationship of software's factors such as goals, functions and constraints to precise specifications of software behavior, and to their evolution over time and across software families.[2]

1.2 Scope

SafeStreets is a crowd-sourced application that intends to provide users with the possibility to notify authorities when traffic violations occur, and in particular parking violations.

The application allows users to send pictures of violations, including suitable metadata, to authorities. Examples of violations are vehicles parked in the middle of bike lanes or in places reserved for people with disabilities, double parking, and so on. In addition, the application allows both end users and authorities to mine the information that has been received, for example by highlighting the streets (or the areas) with the highest frequency of violations, or the vehicles that commit the most violations.

Users are allowed to consult only certain information, in fact, to guarantee different levels of visibility, the system will provide a *restricted access* API that connects to the municipality's service to create a secure bridge for data retrieval. This connection will enable SafeStreets to cross its data with municipality's and make analysis with the aim of identifying potentially unsafe areas and proposing feasible solutions.

Moreover, through the restricted access API, the system will offer back to the municipality the possibility to retrieve information about the violations and generates traffic tickets from it. Of course we will take extreme care of the security of the information shared to prevent any possible alteration and leak.

In addition, SafeStreets will gather back the information about issued tickets to build statistics, for example about the most egregious offenders, or the effectiveness of the SafeStreets initiative (e.g., by looking for trends in the issuing of tickets).

1.2.1 Goals

- G1** Allow guest users to register to the system
- G2** Allow registered users to authenticate to the system
- G3** Allow users to provide data: upload a picture, specify the type of violation, manually insert the licence plate number and possibly the street in which the violation occurred
- G4** Allow users to consult a map showing unsafe areas and view statistics about previously submitted violations

- G5** Allow municipality to retrieve violation data using a restrict access API provided to them
- G6** Cross SafeStreets data with information about the accidents that occur on the territory of the municipalities to identify potentially unsafe areas, and suggest possible interventions
- G7** Ensure that the chain of custody of the information provided by the users is never broken, and the information is never altered or manipulated
- G8** Access the municipality archives of emitted tickets and build statistics based on the retrieved information

1.3 Glossary

1.3.1 Definitions

System: the SafeStreets software we are to develop

Municipality: a city, a town or a village, or a small group of them

Local authorities or authorities: the local authorities of the municipality for example the local police

Guest or Guest user: person who access the system as non logged user

Logged user or Authenticated user: authenticated person who is interfacing with the system

User: guest user or logged user

Registration: interaction between a non registered user and the system in which the user, providing all of the information required by the system for the creation of an account, receives from the system the credentials needed to authenticate to the system

Authentication or Login: interaction between guests and the system that grants authenticated user's privileges to a guest user

Upload procedure: process which realizes the transfer of data between the user and the system

Restricted access API: API that can be used only by authorized person or system through an access token

GPS Coordinates: GPS coordinates are a unique identifier of a precise geographic location on the earth

Chain of Custody or Chain of Evidence: process of validating how any kind of evidence has been gathered, tracked, and protected on its way to a court of law. It guarantees that the data presented is "as originally acquired" and has not been tampered with and is authentic prior to admission into evidence.[\[4\]](#)

1.3.2 Acronyms

RASD: Requirements Analysis and Specification Document

API: Application Programming Interface

GPS: Global Position System

DBMS: Data Base Management System

FSM: Finite-State Machine

GIS: Geographic Information System

1.3.3 Abbreviations

m: meters (with multiples and submultiples)

w.r.t.: with respect to

i.f.f.: if and only if

e.g.: exempli gratia

etc.: et cetera

1.4 Document overview

According to the IEEE standard [5], this document is structured as

1. **Introduction:** it provides an overview of this entire document and product goals
2. **Overall description:** it describes general factors that affect the product providing the background for system requirements
3. **Specific requirements:** it contains all system's functional and non-functional requirements
4. **Use cases identification:** it contains the usage scenarios of the system with the use case diagram, use cases descriptions and other diagrams
5. **Appendices:** it contains the Alloy model, software and tools used, hours of work per each team member

2 Overall description

2.1 Product perspective

description

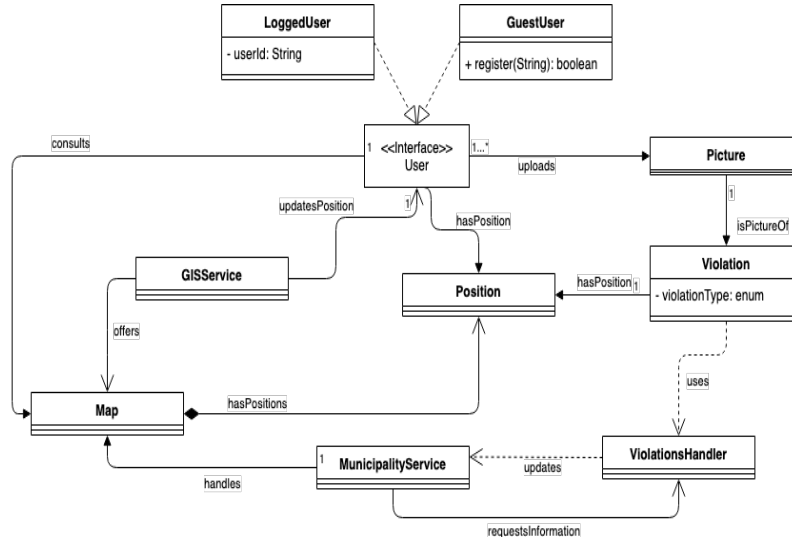


Figure 1: Class Diagram

2.1.1 System interfaces

The system we are to develop will have some external interfaces (represented in Figure 2) to accomplish the goals stated before.

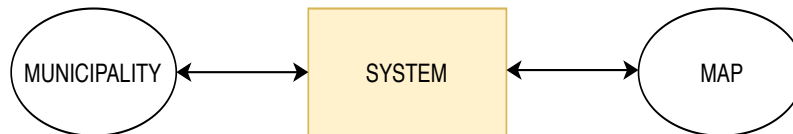


Figure 2: Overview of system interfaces

Municipality The system will interact with municipalities. Our system will retrieve the information about the accidents that occur on the territory of the municipality and cross this information with its own data to identify potentially unsafe areas. It will also retrieve the information about issued tickets from the municipalities to build statistics, for example about the most egregious offenders, or the effectiveness of the SafeStreets initiative (e.g., by looking for trends in the issuing of tickets).

In addition, our system will expose via a *restricted access API* the stored information about the violations to the municipalities, so that the local authorities can generate traffic tickets from it, and receive suggestions for possible

interventions (e.g., add a barrier between the bike lane and the part of the road for motorized vehicles to prevent unsafe parking).

Map Highlighted streets (or areas) with the highest frequency of violations, and unsafe areas are displayed to all users on an up-to-date map which relies on an external GIS.

2.1.2 User interfaces

Guest user Using the interfaces of the system users can:

1. Register to the system or request an account for a specific municipality
2. Log-in to the system

Logged User Using the interfaces of the system users can:

1. Submit a violation with all the required and optional metadata
2. View a map through an external GIS with highlighted streets (or areas) with the highest frequency of violations
3. View statistics on vehicles that commit the most violations, the most egregious offenders, or the effectiveness of the SafeStreets initiative.
4. View and edit personal information

2.1.3 Hardware interfaces

The only hardware interfaces with which the system cooperates are the smart-phones of the users and the hardware that supports the system that offers municipality's service.

2.1.4 Software interfaces

In order to reach the goals highlighted in the [goals section](#) the system need to interface with:

1. Databases and DBMSs, clearly required in order to store data about users, violations, safe/unsafe areas etc.
2. The software offered by the municipality, required in order to exchange data with its services
3. The map API offered by the GIS

2.1.5 Communication interfaces

We want to ensure the encryption of data shared with our system. This involves recording of metadata information as well as issues of access control and security for all the handling digital chain of custody.

Siamo sicuri di voler fare che tutti gli utenti possono vedere queste staitistiche?

2.2 Product functions

Violation upload procedure Provide logged users with the possibility to notify authorities when traffic violations occur and, in particular, parking violations. The application allows users to send pictures of violations, including their date, time, and position, to authorities. SafeStreets then stores the information provided, completing it with suitable meta-data. In particular, when it receives a picture, it runs an algorithm to read the license plate number (the user can help with the recognition) and it stores the retrieved information with the violation, including also the type of the violation (submitted by the user) and the name of the street where the violation occurred.

Retrieve data from municipality Retrieve the information about the accidents that occur on the territory using the service offered by the municipalities and cross these data with SafeStreets data, to identify potentially unsafe areas. This will also allow the system to understand which violations are more likely to cause accidents in a particular zone and elaborate suggestions on possible interventions, later communicated to the municipality via a *restricted access API* provided to them.

Show information and statistics The application allows logged users to mine the information that has been received, highlighting the streets (or the areas) with the highest frequency of violations, considered unsafe areas, or the vehicles that commit the most violations. In addition, statistics about issued tickets, for example about the most egregious offenders, or the effectiveness of the SafeStreets initiative, are shown to the user if requested.

Restricted access API The system will expose via a *restricted access API* the stored information about the violations to the municipalities, so that the local authorities can generate traffic tickets from it and receive suggestions for possible interventions to carry out (e.g., add a barrier between the bike lane and the part of the road for motorised vehicles to prevent unsafe parking), in order to decrease the risk of those areas, so increasing their safety.

2.3 User Characteristics

Users can use our system when they notice a violation and want to communicate it to the authorities.

Necessary conditions for the user in order to use the system are:

- He must have a smartphone with a working connection to the internet and he must be able to properly use it
- He must be in the age of majority
- He must be able to identify a violation and its type

The user agrees to these conditions during the registration to the system.

2.4 Constraints

We assume that these constraints are always met:

- C1** GPS position is supposed to be accurate (max error $\pm 5\text{m}$)
- C2** The quality of the picture is sufficient to recognise the plate number (min resolution 320x240)
- C3** Internet connection must be strong enough to allow the upload of the picture in a reasonable amount of time (supported technologies are 3G, 4G and 5G due to the performance requirement)

2.5 Assumptions

We assume that these assumptions hold true in the domain of our system

- DA1** GPS position of all users is always obtainable
- DA2** Internet connection always works correctly
- DA3** Municipality services are always reachable
- DA4** The maps provided by the GIS are always reachable and up to date
- DA5** The smartphone of the user runs iOS (9 or later) or Android (Jelly Bean or later)

3 Specific Requirements

3.1 Functional Requirements

The following requirements are derived in order to achieve the specified goals.

3.1.1 Goals

G1 Allow guest users to register to the system

R1 The system must require the *guest* user to insert his fiscal code, a username, a valid e-mail and a password to identify him

R2 The system must check that the validity of the data inserted by the *guest* user namely avoid duplicates, invalid fiscal codes and too weak passwords

R3 The system must send an e-mail to the *guest* user to verify the e-mail address given during the registration

DA2 Internet connection always works correctly

DA5 The smartphone of the user runs iOS (9 or later) or Android (Jelly Bean or later)

G2 Allow registered users to authenticate to the system

R4 The system must require the user to insert his username and password to authenticate to the system

R5 The system must be able to check if the username and password pair correspond to a user correctly registered to the system

R6 The system must allow access to a user if username and password inserted corresponds to a user correctly registered to the system

R7 The system must allow access to the services provided by the system only to authenticated users that and have confirmed their own e-mail address

G3 Allow users to provide data: upload a picture, specify the type of violation, manually insert the licence plate number and possibly the street in which the violation occurred

R8 The system must require the user to login in order to allow him to provide data

R9 The system must require the user to upload a picture of the vehicle subject of the violation

R10 The system must allow the user to manually insert the license plate number in order to help the recognition algorithm

R11 The system must be able to retrieve the license plate of the vehicle running an algorithm to recognise it

R12 The system must be able to verify that the license plate number is valid and registered to a vehicle

R13 The system must require the user to specify the type of violation

- R14** The system must allow the user to provide the location of the violation, manually specifying the address, picking it up from the map or using the GPS of the device
- ??** Allow users to consult previously received data served with different levels of visibility based on their role
- R15** The system must require the user to login in order to allow him to consult information about previous violations
- R16** The system must allow the authenticated user to request summary information on the past violations
- R17** The system must allow the authenticated authority to request detailed information on the past violations
- R18** The system must be able to provide the requested information to user
- R19** The system must allow the user to consult the requested violations information
- G6** Cross SafeStreets data with information about the accidents that occur on the territory of the municipalities to identify potentially unsafe areas, and suggest possible intervention
- R20** The system must be able to retrieve data about accidents from municipality systems
- R21** The system must be able to process data retrieved from municipality
- R22** The system must be able to elaborate accidents and violations information to extract data about unsafe areas
- R23** The system must be able to provide data to municipality systems to suggest possible interventions to increase safety in a specific area
- G7** Ensure that the chain of custody of the information provided by the users is never broken, and the information is never altered or manipulated
- R24** The system must adopt security measures to prevent malicious accesses and to protect sensible data
- R25** The system must provide a secure channel to communicate with the users
- R26** The system must encrypt the connection with the users in order to protect the process of providing data
- G8** Access the municipality archives of emitted tickets and build statistics based on the retrieved information
- R27** The system must be able to retrieve data about tickets issued by the municipality
- R28** The system must be able to process data retrieved from municipality
- R29** The system must be able to elaborate issued tickets information to generate statistics about useful violations provided by users

3.2 Performance Requirements

The system should ensure acceptable response times in the interactions with the user, which strictly depends on the number of concurrent users and the connection speed.

The processes of providing data and loading the map of safe and unsafe areas shouldn't be too slow.

3.3 Software System Attributes

3.3.1 Availability

The system must be available 99,9% of the time (up to 8,76 hours per year of downtime). The system should be accessible 24 hours per day.

3.3.2 Security

Users personal information and payment information are encrypted and must be protected during transmission, as already stated the PTPP protocol will be used to ensure encryption through the network. Restricted access APIs must check that who tries to use them is actually allowed to do so.

3.3.3 Portability

The system must be also accessible by the most common mobile platforms (iOS and Android devices).

4 Use cases identification

4.1 Scenarios

Here are some scenarios that describe the usage of the system.

4.1.1 Scenario 1

Davide is walking down the street and notices a car parked over the crosswalks. He opens the SafeStreets app, registers to the system with his fiscal code, and takes a picture from the in-app camera. Before uploading he adds information such as the type of violation (i.e. "Bad Parking") and the street in which the violation occurred. As soon as SafeStreet processes the uploaded data the authorities are alerted.

4.1.2 Scenario 2

Carlo wants to teach his son to drive and wants to find the safest area of Isernia in order to avoid exposing him to difficult situations in his first drives. He opens the SafeStreets app and consult the map with the streets that have the greatest number of incidents and violations, and will try to avoid them allowing his son to have a safe drive.

4.1.3 Scenario 3

SafeStreets automatically retrieves data from the municipality's service, and after a while notices that extremely frequently cars are parked in a restricted area of Milan. After further analysis and with the help of the users they come to the conclusion that don't realise that they can't park in that area because of a misleading signal, so they contact the municipality and offer a possible solution to the problem.

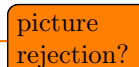
4.1.4 Scenario 4

After receiving a notification of "Dangerously parked car" with the picture showing a car parked in such a way that could risk a possible collision with the tram line, SafeStreets alerts the authorities in order to facilitate the process of car removal and the consequential ticket generation.

4.1.5 Scenario 5

4.1.6 Scenario 6

Non so more ce ne verranno altri



picture
rejection?

4.2 Use case diagram

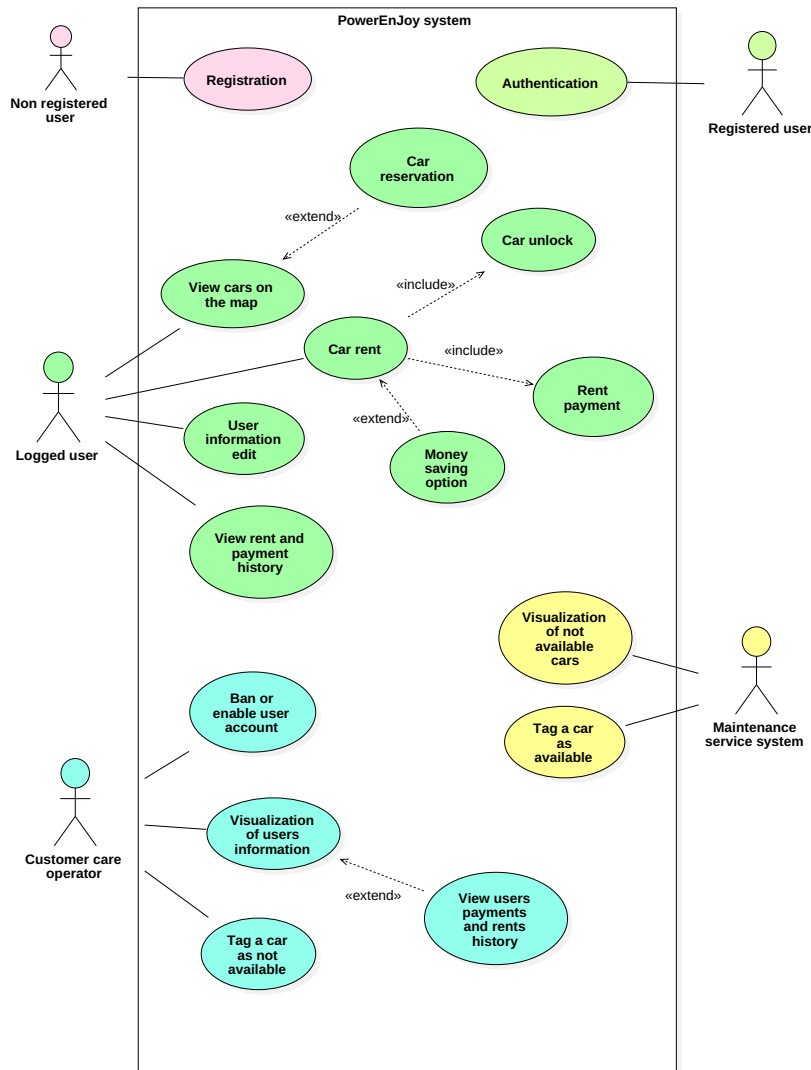


Figure 3: Use case diagram

Notes to read the diagram The use case diagram represents the possible interactions of actors with the system and the different use cases in which the actors are involved.

The "Maintenance service system" is an external software which the system-to-be needs to interact with.

We do not consider the external payment handling system an actor since it does not start any interaction with our system, it simply reacts when our system requests its services; this interaction is encapsulated as sub-procedure in the flow of events of the "Rent payment" use case.

We do not consider the car an actor for the same reason, moreover the only interactions started by the car are trigger events which are very simple interactions, which we do not consider use cases.

4.3 Use cases description

4.3.1 Registration

Name	Registration
Actors	Non registered user
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. The user asks the system to register to its services 2. The system shows the appropriate form to fill to register to the system 3. The user inserts an username to be uniquely identified by the system 4. The user inserts his own email address 5. The user inserts his name, surname, birth date and place and current domicile 6. The user inserts his driving license ID code 7. The user inserts payment information 8. The user confirms data inserted are correct e submit the form 9. The system checks the username to be unique 10. The system checks the email to be unique 11. The system checks the driving license ID to be unique 12. The system sends an email to the user with a unique link to verify the email address inserted by the user really belongs to him 13. The user clicking on the link received confirms his email address 14. The user is notified by mail the registration procedure is correctly completed and provided with a password bound to his username to access the system
Exit conditions	The user is able to authenticate to the system as <i>registered</i> user with its own credentials

Exceptions

- If the username inserted by the user is already used by another user, the system displays an error message asking the user to insert another username
- If the mail inserted by the user is already used by another user, the system displays an error message asking the user to insert another mail
- If the user notices to have entered wrong informations he could edit them at the end of the process of registration in his personal page

Table 1: *Registration* use case description

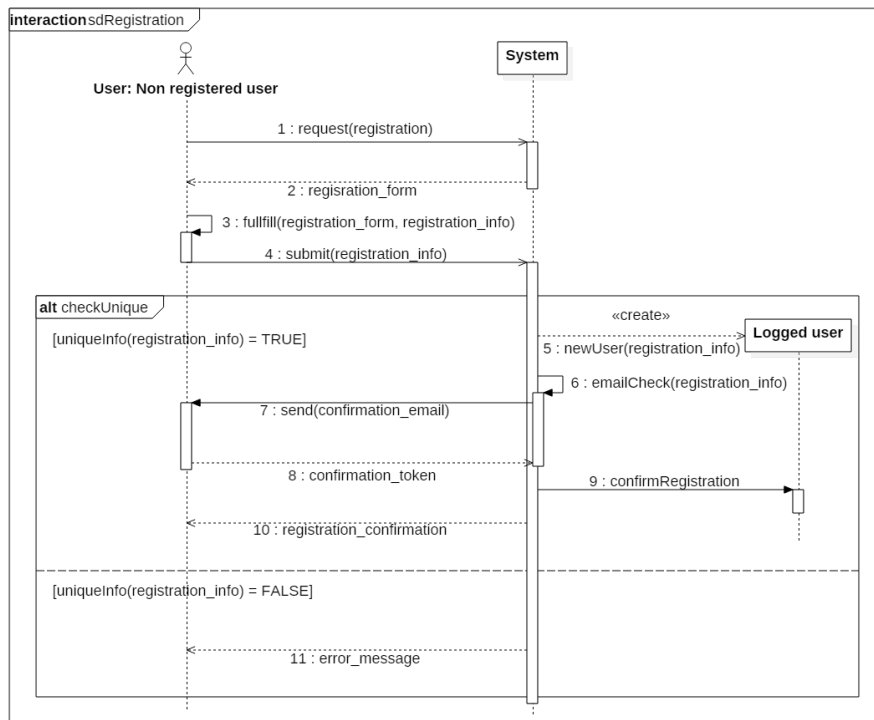
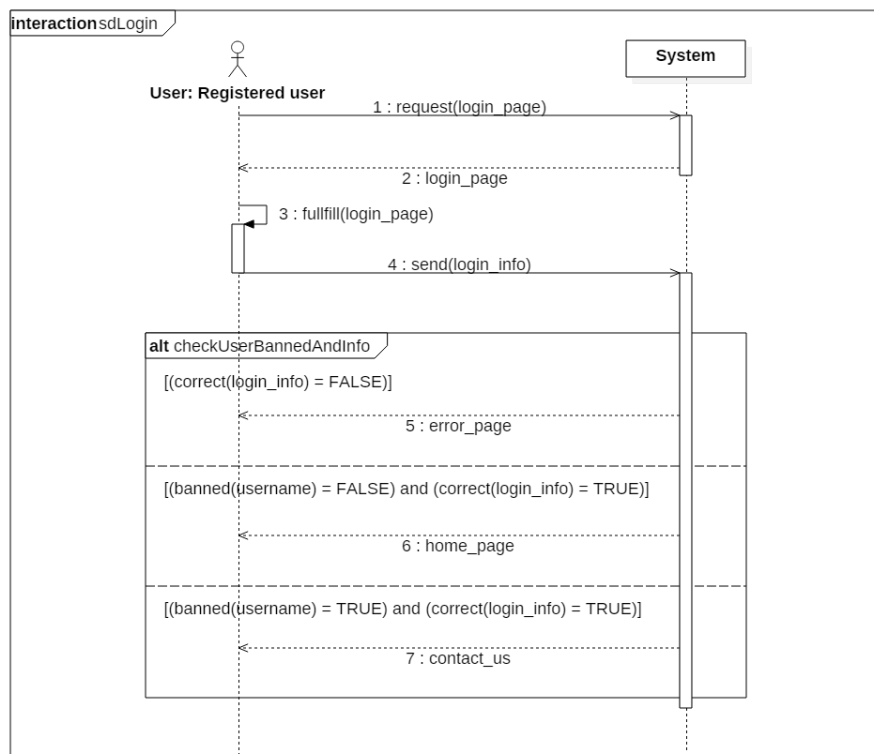


Figure 4: *Registration* sequence diagram

4.3.2 Authentication

Name	Authentication
Actors	Registered user
Entry conditions	The user must know his username and password
Flow of events	<ol style="list-style-type: none"> 1. The user inserts his username and password in the appropriate form and submit it 2. The system validates the inserted credentials checking also if the user has confirmed his own email address 3. The system checks if the user is banned
Exit conditions	If the credential validation is successful and the user is not banned he is granted the proper privileges
Exceptions	<ul style="list-style-type: none"> • If the credential validation failed an error message is displayed • If the credential validation is successful and the user is banned a message providing assistance is displayed and the system doesn't allows the user to access to the system

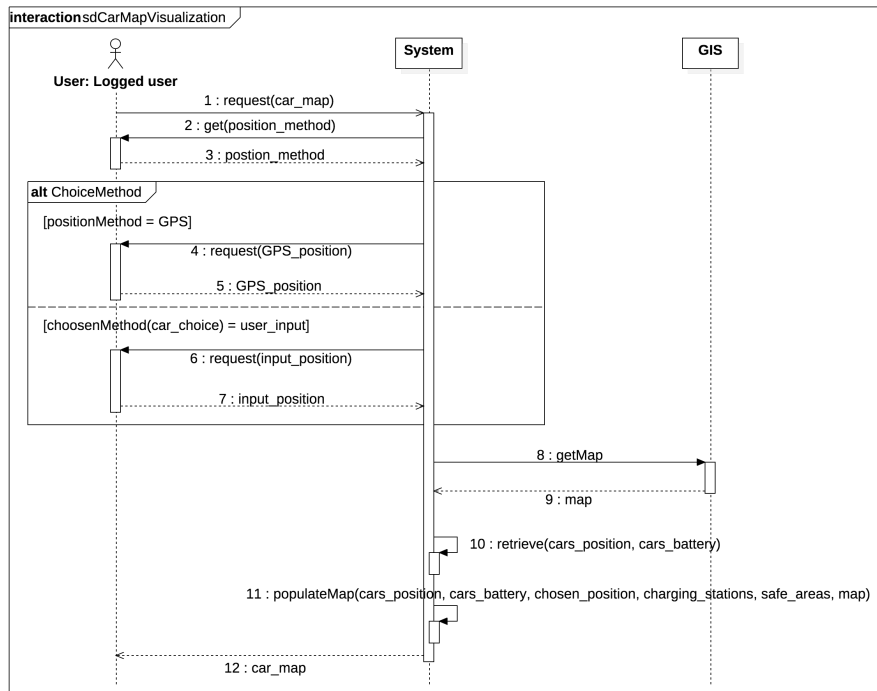
Table 2: *Authentication* use case description

Figure 5: *Authentication* sequence diagram

4.3.3 View cars on the map

Name	View cars on the map
Actors	Logged user
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. The user chooses if he wants to use his GPS position or insert a different one manually <ol style="list-style-type: none"> a. The system retrieves the user's GPS position b. The user inserts a position 2. The system retrieves the position of all <i>Available</i> cars and their battery level percentage 3. The system shows a map with all available cars, charging stations position and safe areas near the position indicated 4. The user can click on a car on the map to see its battery level percentage
Exit conditions	The user can navigate a map with all available cars near the position indicated by him
Exceptions	If the position inserted by the user is not correct an error message is displayed

Table 3: *View cars on the map* use case description

Figure 6: *View cars on the map* sequence diagram

4.3.4 Car reservation

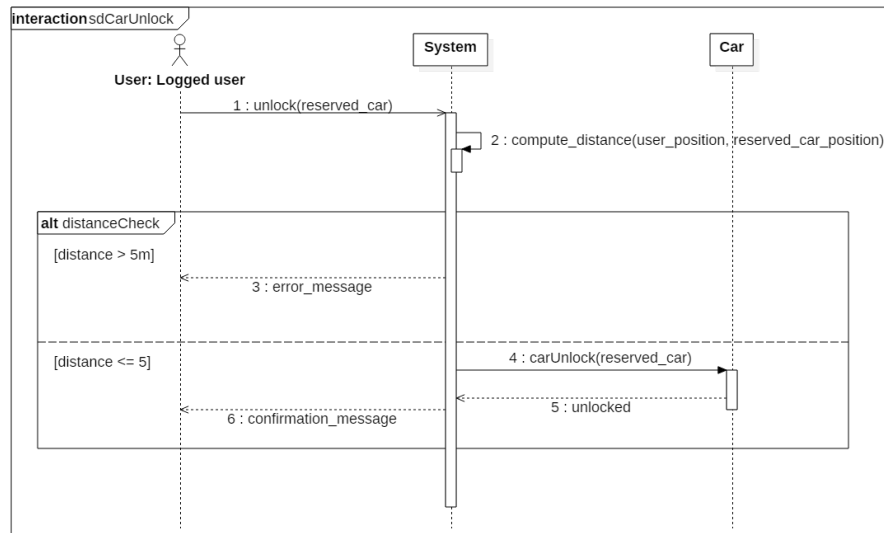
Name	Car reservation
Actors	Logged user
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. View cars on the map 2. The user selects the car he wants to reserve 3. The user confirms he wants to reserve that car
Exit conditions	The system set the state of the chosen car as <i>Reserved</i> paired with the user who made the reservation
Exceptions	If the user has already reserved a car, the system shows an error message and doesn't allow him to reserve another car

Table 4: *Car reservation* use case description

4.3.5 Car unlock

Name	Car unlock
Actors	Logged user
Entry conditions	The user reserved car
Flow of events	<ol style="list-style-type: none"> 1. The user asks the system to unlock the car he reserved 2. The system checks if the user's position is at most 5 meters away from the position of the car he reserved 3. The system unlocks the car with the state set as <i>Reserved</i> paired with the aforementioned user 4. The system sends a message to the user, confirming that the car is unlocked
Exit conditions	The car is unlocked and the user can pick it up
Exceptions	<ul style="list-style-type: none"> • If the position of the user is not at most 5 meters away from the position of the car he reserved the system displays an error message

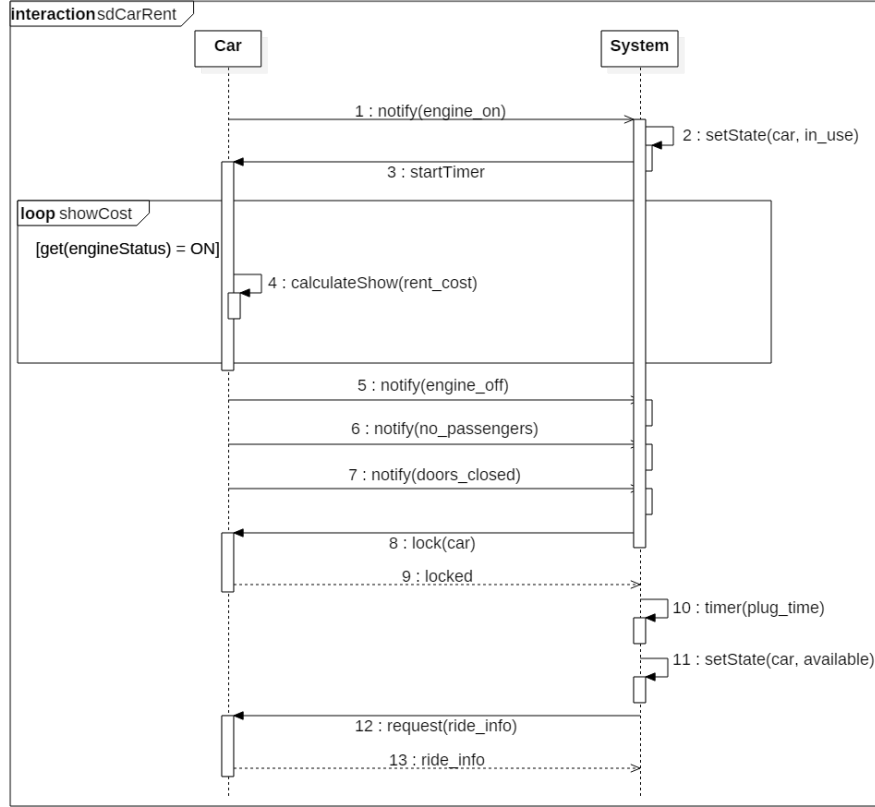
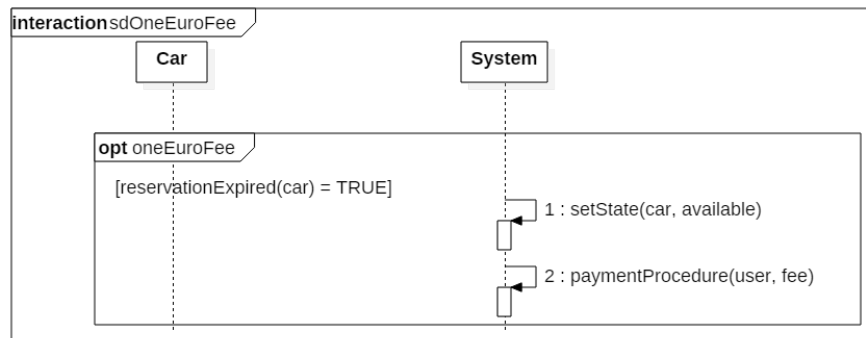
Table 5: *Car unlock* use case description

Figure 7: *Car unlock* sequence diagram

4.3.6 Car rent

Name	Car rent
Actors	Logged user
Entry conditions	The user is paired with the <i>Reserved</i> state of a car
Flow of events	<ol style="list-style-type: none"> 1. Car unlock 2. The user ignites the car engine 3. The system sets the state of the <i>Reserved</i> car to <i>In Use</i> paired with the same user 4. During the rent the user is informed about the current charge and whether he is or not inside a safe area 5. The user leaves the car turning off the engine and closing the doors 6. The system locks the car 7. The system activates a timer to allow the user to plug the car into a charging station if it is near one of them 8. When the timer expires: <ol style="list-style-type: none"> 8.1 The system retrieves informations about the ride from the car: number of passengers detected during the ride, position of the car and battery level at the end of the ride and if the car is or not on charge 8.2 The system sets the car as <i>Available</i> 9. Rent payment
Exit conditions	The user is charged of the correct amount for the ride and at anytime could perform another rent, the car is available again
Exceptions	<ul style="list-style-type: none"> • If the user doesn't start the engine up to one hour after the reservation, he is charged of 1€(through a payment procedure), the car state is set as <i>Available</i> and the user is notified his reservation is expired

Table 6: *Car rent* use case description

Figure 8: *Car rent* sequence diagramFigure 9: *One Euro fee* sequence diagram

The overall status of a car can be represented by the FSM in [Figure 10](#)

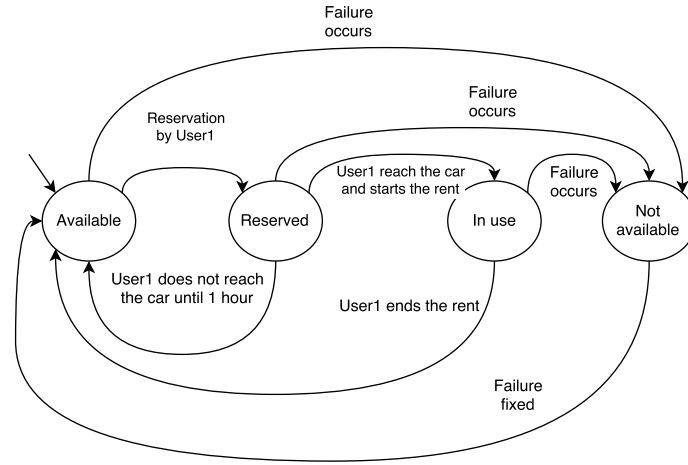


Figure 10: Car status FSM

Notes to read the diagram The *Not Available* state includes the cases in which the car is either broken or a user left it with a critical battery level and not on charge.

The system changes the state of a car from *Available* to *Not Available* when its battery level is critical and the car is not on charge (see ??).

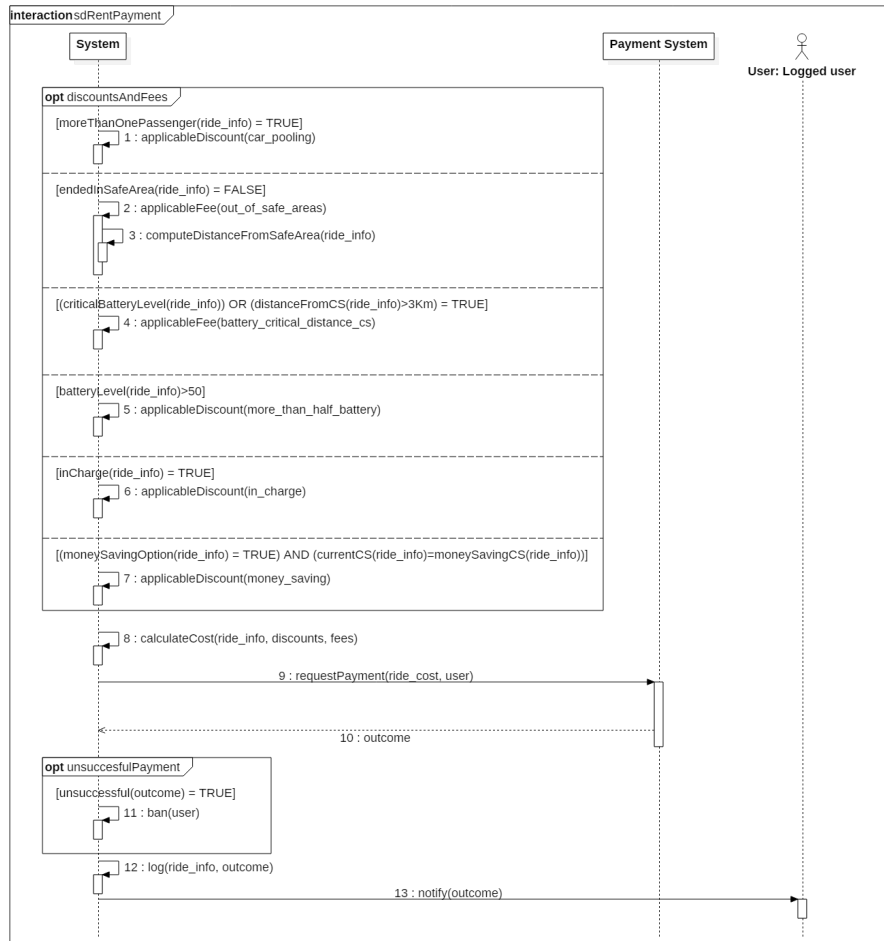
Even if the car is left with no battery left, it is still able to communicate with the system, so the rent can end normally and the maintenance service will take care the car (see ??).

4.3.7 Rent payment

Name	Rent payment
Actors	Logged user
Entry conditions	The user must have completed a rent shutting off the engine and exiting the car. The system has retrieved information about the ride from the car.
Flow of events	<ol style="list-style-type: none"> 1. The system checks if the car position is or is not inside a safe area 2. The system checks if the car has detected more than one passenger during the rent 3. The system checks the car battery percentage 4. The system checks if the car is plugged on a charging station 5. The system checks the distance of the car from the nearest charging station 6. The system calculates the cost of the ride based on the rent time 7. The system determines the applicable discounts/extra fee applying it to the cost of the ride 8. The system starts a payment procedure with user's payment information using an external service 9. The system waits a response from the external payment service 10. The system logs data about the rent and the payment 11. The system notifies the user about the result of the payment procedure and on discount/extra fees applied

Alternative flow	Flow of events as specified upon from 1 to 7
	8 a. The system detects the user has enabled the <i>money saving option</i>
	8 b. The system checks if the car is currently on charge on the charge station determined by the system at the begin of the rent
	8 c. The system determines the applicable discounts/extra fee applying it to the cost of the ride eventually also taking in account the <i>money saving option</i> discount if the car is currently on charge on the charge station determined by the system at the begin of the rent
	Flow of events as specified upon from 9 to 12
Exit conditions	The user is charged of the correct amount for the ride
Exceptions	<ul style="list-style-type: none"> • If the payment procedure is not correctly completed the user is banned, rent information is stored, the payment suspended and the user is informed to contact the customer service.

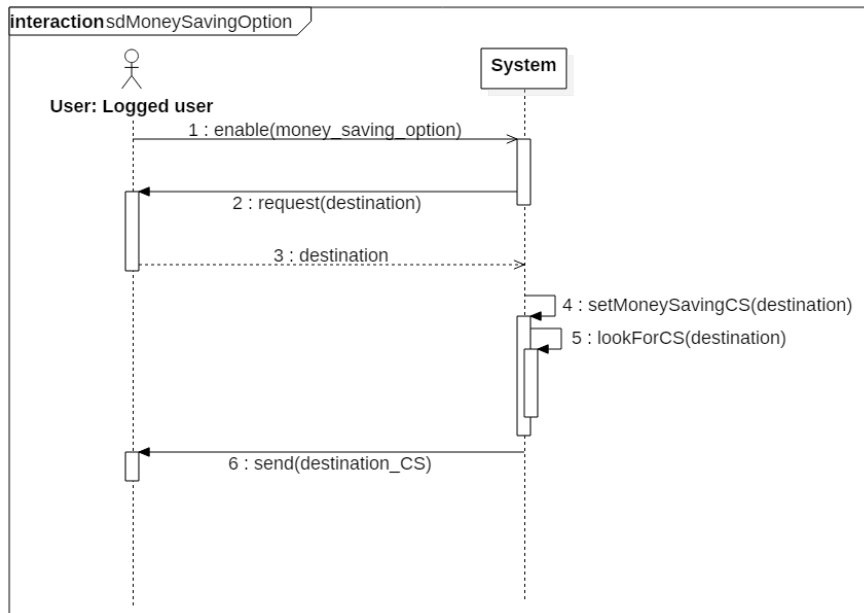
Table 7: *Rent payment* use case description

Figure 11: *Rent payment* sequence diagram

4.3.8 Money saving option

Name	Money saving option
Actors	Logged user
Entry conditions	The user should have enabled the <i>money saving option</i>
Flow of events	<ol style="list-style-type: none"> 1. Car Reservation 2. The system asks the user to insert his destination 3. The user inserts his destination 4. The system searches for charging stations near the destination position inserted by the user with available plugs 5. The system chooses a charging station in order to ensure a uniform distribution of cars in the city and taking in account the destination of the user 6. The system informs the user about the charging station to reach in order to obtain the discount 7. Car Rent (Car Reservation already done)
Exit conditions	<ul style="list-style-type: none"> • If the user has left the car plugged in the charging station suggested by the <i>money saving option</i> he has obtained the correct discount • The user can any time perform another rent • Car is again available
Exceptions	<ul style="list-style-type: none"> • If the user doesn't leave the car in the charging station suggested by the <i>money saving option</i> he doesn't obtain the related discount

Table 8: *Money saving option* use case description

Figure 12: *Money saving option* sequence diagram

4.3.9 Visualization of not available cars

Name	Visualization of not available cars
Actors	Maintenance service system
Entry conditions	Maintenance service system must know the access token to be identified by the system
Flow of events	<ol style="list-style-type: none"> 1. The maintenance service system asks for the list of car with state set as <i>Not Available</i> sending the request paired with the access token 2. The system checks the access token 3. The system retrieves the list of car with state set as <i>Not Available</i> along with the identifier used by the system to identify each car, the GPS position of each car, the description of the problem of each car and the software key to access each car 4. The system sends the information to the maintenance service system
Exit conditions	The maintenance service system receives the list of cars with state set as <i>Not Available</i>
Exceptions	<ul style="list-style-type: none"> • If the access token sent by the maintenance service system is not recognized, the system sent to the maintenance service system an error message

Table 9: *Visualization of not available cars* use case description

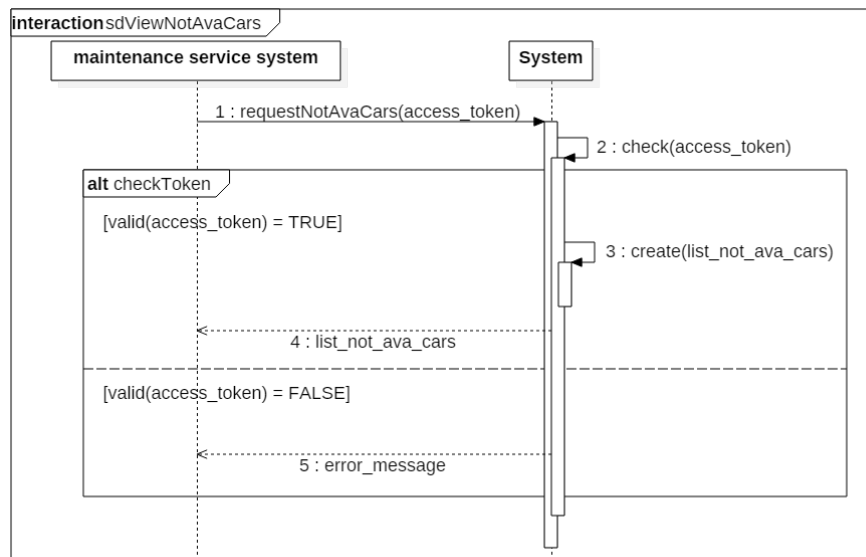
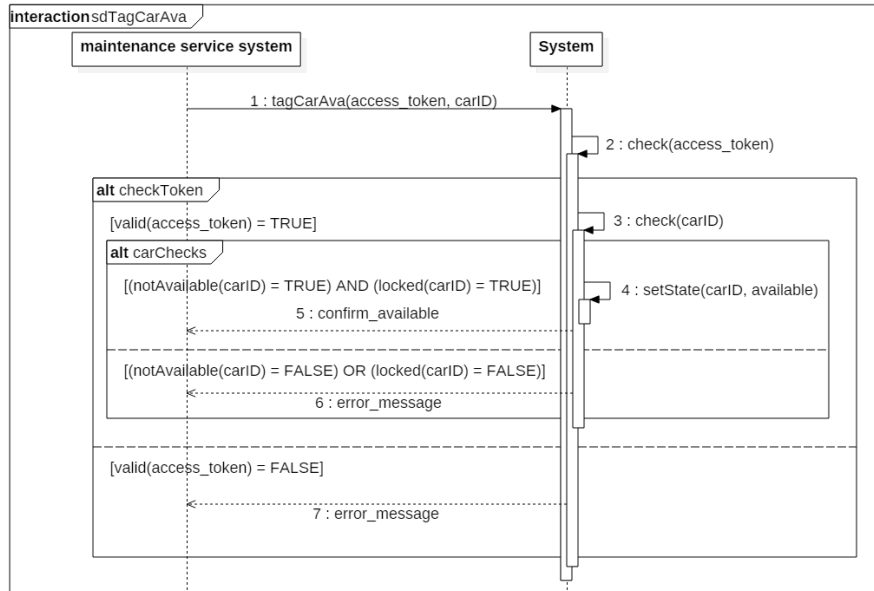


Figure 13: Visualization of not available cars sequence diagram

4.3.10 Tag a car as available

Name	Tag a car as available
Actors	Maintenance service system
Entry conditions	Maintenance service system must know the access token to be identified by the system
Flow of events	<ol style="list-style-type: none"> 1. The maintenance service system asks to tag a car as Available sending the car identifier paired with the access token 2. The system checks the access token sent by the maintenance service 3. The system checks the identifier received corresponds to a car with state set as <i>Not Available</i> 4. The system checks if the car identified by the identifier received is locked 5. The system set the state of the car identified by the aforementioned identifier as <i>Available</i> 6. The system sends to the maintenance service system a confirmation message the car state has been set as <i>Available</i>
Exit conditions	The car state is set as <i>Available</i>
Exceptions	<ul style="list-style-type: none"> • If the access token sent by the maintenance service system is not recognized, the system sends to the maintenance service system an error message • If the car identifier sent by the maintenance service system is not recognized or doesn't correspond to a car set as <i>Not Available</i>, the system sends to the maintenance service system an error message • If the car identifier sent by the maintenance service system corresponds to a car not locked, the system sends to the maintenance service system an error message

Table 10: *Tag a car as not available* use case description

Figure 14: *Tag a car as available* sequence diagram

4.3.11 Visualization of users information

Name	Visualization of users information
Actors	Customer care operator
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. The customer care operator inserts the username or the mail of a registered user 2. The system checks if the username or the mail correspond to a user registered to the system 3. The system retrieves user's data (name, surname, birth date and place, current domicile and driving license information) along with information about the car state the user is actually paired with 4. The system shows to the customer care operator the info about the user
Exit conditions	The customer care operator can view the information required about the user
Exceptions	<ul style="list-style-type: none"> • If no users are found according to the parameters inserted by the customer care operator the system shows an error message

Table 11: *Visualization of users information* use case description

4.3.12 View users payments and rents history

Name	View users payments and rents history
Actors	Customer care operator
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. Visualization of users information 2. The customer care operator asks to view user's payments and rents history 3. The system retrieves the list of user's payments (successful and unsuccessful) 4. The system retrieves the list of user's rents 5. The system shows to the customer care operator user's payments and rents history
Exit conditions	The customer care operator can view the information required about the user
Exceptions	

Table 12: *View users payments and rents history* use case description

4.3.13 Mark or unmark a user as banned

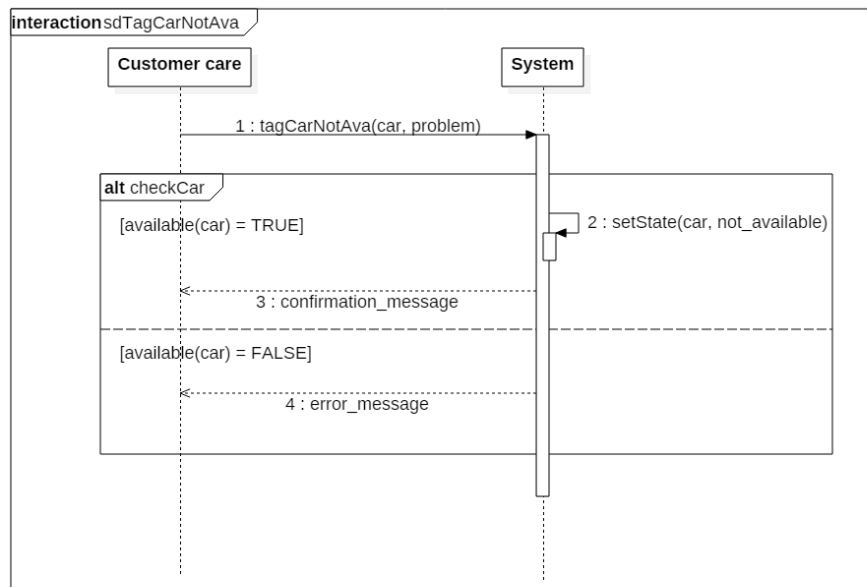
Name	Mark or unmark a user as banned
Actors	Customer care operator
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. The customer care operator inserts the username of a registered user 2. The customer care operator asks to ban or to enable the registered user paired with the inserted username <ul style="list-style-type: none"> • If the operator wants to mark a user as <i>banned</i> he must insert a brief description of reasons why 3. The system checks if the username corresponds to a user registered to the system 4. The system marks or unmarks the user paired with the username as <i>banned</i>
Exit conditions	The state of the user is updated
Exceptions	<ul style="list-style-type: none"> • If the username inserted by the customer car operator is not recognized, the system shows an error message

Table 13: *Mark or unmark a user as banned* use case description

4.3.14 Tag a car as not available

Name	Tag a car as not available
Actors	Customer care operator
Entry conditions	
Flow of events	<ol style="list-style-type: none"> 1. The customer care operator inserts the identifier of the car 2. The customer care operator asks to mark the car as <i>Not Available</i> 3. The customer care operator inserts a brief description of why the car state must be set as <i>Not Available</i> 4. The system checks the car identifier 5. The system set the state of the car identified by the aforementioned identifier as <i>Not Available</i> paired with the description 6. The system shows a confirmation message the car has been tagged as <i>Not Available</i>
Exit conditions	The state of the car is setted as <i>Not Available</i>
Exceptions	<ul style="list-style-type: none"> • If car identifier sent by the customer care operator is not recognized, the system displays an error message

Table 14: *Tag a car as not available* use case description

Figure 15: *Tag a car as Not Available* sequence diagram

4.4 UML class diagram

Based on collected scenarios and on the identified use cases we have developed the following requirements-level class diagram[?]. To ensure a better readability class attributes are not represented.

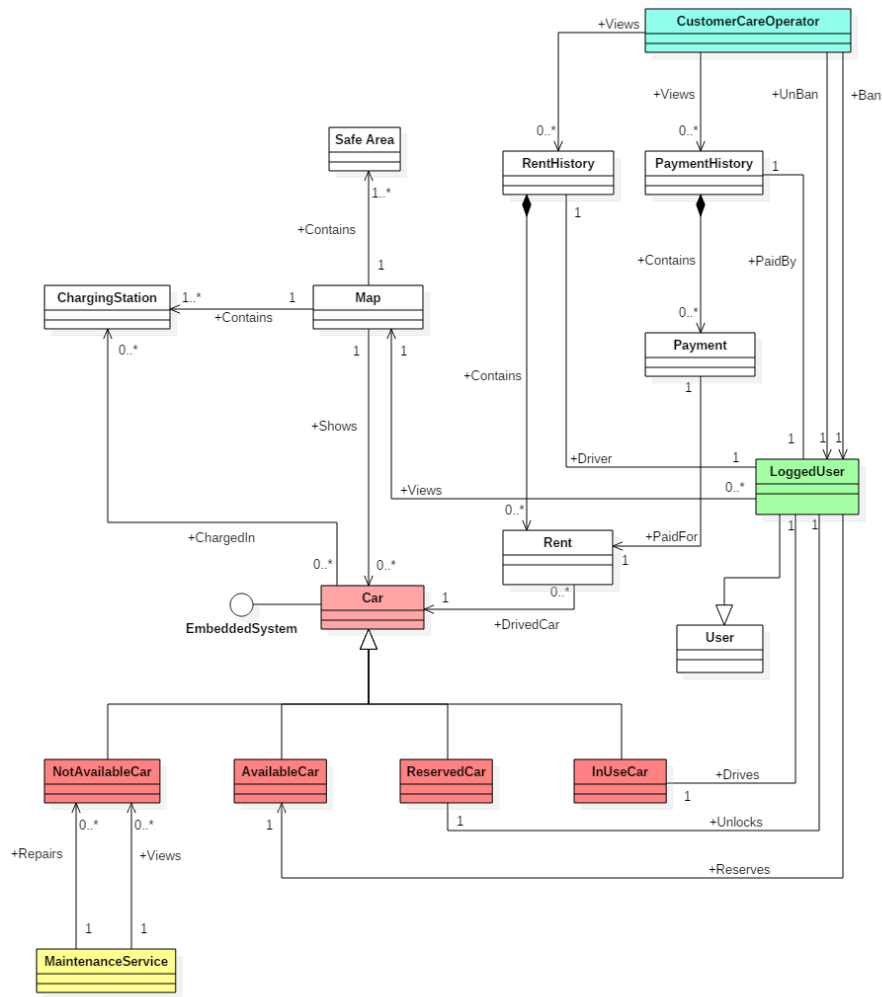


Figure 16: UML class diagram

Appendices

A Alloy model

A.1 Source code

```

1  open util/boolean
2
3  sig Car{
4    batteryLevel: one BatteryLevelPercentage,
5    status: one CarStatus,
6    usedBy: lone LoggedUser,
7    reservedBy: lone LoggedUser,
8    numberOfPassengers: NOType,
9    onCharge: one Bool,
10   engineOn: one Bool
11 }
12
13 //Car statuses
14 abstract sig CarStatus{}
15 one sig Available extends CarStatus{}
16 one sig Reserved extends CarStatus{}
17 one sig InUse extends CarStatus{}
18 one sig NotAvailable extends CarStatus{}
19
20 //Battery level percentage: should be a percentage 0-100%
21 abstract sig BatteryLevelPercentage{}
22 one sig Lower20Full extends BatteryLevelPercentage{}
23 one sig More50Full extends BatteryLevelPercentage{}
24 one sig From20to50Full extends BatteryLevelPercentage{}
25
26 //Number of passengers, we assume to deal with 5 passengers
   cars
27 abstract sig NOType{}
28 one sig Zero extends NOType{}
29 one sig One extends NOType{}
30 one sig Two extends NOType{}
31 one sig Three extends NOType{}
32 one sig Four extends NOType{}
33 one sig Five extends NOType{}
34
35 abstract sig User{}
36 sig LoggedUser extends User{
37   //personal information
38   //other parameters
39   banned: one Bool
40 }
41
42 sig ChargingStation{
43   charging: set Car
44 }
45

```

```

46 //A RentMade models a rent made in the past, so, for example
    , in the world created
47 //the user who made the rent can be banned or the car used
    for the rent can be NotAvailable
48
49 //If a RentMade corresponds to a reservation expired the
    correspondent fee is assigned but
50 //others parameters regarding the end of the rent are set to
    default acceptable values
51
52 //leftMSOstation: true iff money saving option enabled and
    auto left on charge
53 //in the station determined by MSO
54
55 //Choice of discount to be applied is not modeled
56 sig RentMade{
57   userRent: one LoggedUser,
58   carRent: one Car,
59   endPosition: one PositionWrtPowerGrid,
60   endSafeArea: one Bool,
61   reservationExpired: one Bool,
62   endBatteryLevel: one BatteryLevelPercentage,
63   onChargeAtTheEnd: one Bool,
64   passengersDuringTheRide: one NOPTYPE,
65   discountApplicableRent: set Discount,
66   additionalFeeRent: set Fee,
67   leftMSOstation: one Bool
68 }
69
70 abstract sig PositionWrtPowerGrid{}
71 one sig More3kmPowerGrid extends PositionWrtPowerGrid{}
72 one sig Lower3kmPowerGrid extends PositionWrtPowerGrid{}
73
74 //M1PD = MoreThan1PassengerDiscount
75 //BHFD = BatteryHalfFullDiscount
76 //CCD = CarOnChargeDiscount
77 //MSOD = MoneySavingOptionDiscount
78 abstract sig Discount{}
79 one sig M1PD extends Discount{}
80 one sig BHFD extends Discount{}
81 one sig CCD extends Discount{}
82 one sig MSOD extends Discount{}
83
84 //REF =ReservationExpiredFee
85 //OSAF =OutSafeAreaFee
86 //A3BCF =Away3kmOrCSBatteryCriticalFee
87 abstract sig Fee{}
88 one sig REF extends Fee{}
89 one sig OSAF extends Fee{}
90 one sig A3BCF extends Fee{}
91
92 //A user can be only in one car at a given time
93 fact OneUserCanBeInOneCarAtSameTime{

```

```

94   no disjoint c1,c2:Car | c1.usedBy = c2.usedBy and c1.
    usedBy != none
95 }
96
97 //A user can reserve only one car at a given time
98 fact ACarReservedByOnlyOneUser{
99   no disjoint c1,c2:Car | c1.reservedBy = c2.reservedBy and
    c1.reservedBy != none
100 }
101
102 //A car in use cannot be reserved
103 fact ACarInUseCannotBeReserved{
104   all c:Car | c.usedBy != none implies c.reservedBy = none
105 }
106
107 //A user cannot use one car and reserve another car at a
    given time
108 fact NoUsersCanUseAndReserveDifferentCars{
109   no disjoint c1,c2:Car | c1.usedBy = c2.reservedBy and c1.
    usedBy != none and c2.reservedBy != none
110 }
111
112 //Cars set as Available cannot be used or reserved at a
    given time
113 fact AvailableCarsCantBeReservedOrUsed{
114   no c:Car | c.status = Available and (c.usedBy != none or c
    .reservedBy != none)
115 }
116
117 //Cars set as Not Available cannot be used or reserved at a
    given time
118 fact NotAvailableCarsCantBeReservedOrUsed{
119   no c:Car | c.status = NotAvailable and (c.usedBy != none
    or c.reservedBy != none)
120 }
121
122 //Reserved statuts must be paired with only one user
123 fact ReservedStatusMustBePairedWithOneUser{
124   all c:Car | c.status = Reserved implies (c.reservedBy !=
    none and c.usedBy = none)
125 }
126
127 //In Use statuts must be paired with only one user
128 fact InUseStatusMustBePairedWithOneUser{
129   all c:Car | c.status = InUse implies (c.reservedBy = none
    and c.usedBy != none)
130 }
131
132 //Car with battery percentage lower than 20 percent full
    must be set as Not Available
133 fact CarWithBatteryPercentageLower20FullNotAvailable{
134   all c:Car | (c.batteryLevel = Lower20Full and c.onCharge =
    False) implies c.status = NotAvailable
135 }

```

```

136
137 //A car not in use can not detect number of passengers
    greater than zero
138 fact PassengersOnlyOnInUseCars{
139     no c:Car | c.status != InUse and c.numberOfPassengers !=
        Zero
140 }
141
142 //A car In Use must detect at least one passenger
143 fact AtLeastOnePassengerOnInUseCars{
144     no c:Car | c.status = InUse and c.numberOfPassengers =
        Zero
145 }
146
147 //A car In Use has the engine turned on
148 //Note that a In Use car can have the engine turned off
149 fact OnlyInUseCarEngineOn{
150     all c:Car | c.engineOn = True implies c.status = InUse
151 }
152
153 //A car In Use can not be on charge
154 fact InUseCarNotOnCharge{
155     all c:Car | c.status = InUse implies c.onCharge = False
156 }
157
158 // A car is charging when connected to a charging station
159 fact CarIsChargingWhenConnected{
160     all s:ChargingStation, c:Car | c in s.charging implies c.
        onCharge = True
161     all c:Car | some s:ChargingStation | c.onCharge = True
        implies c in s.charging
162 }
163
164 // At most one charging station connected to a car
165 fact NoMoreOneCSForOneCar{
166     all disjoint s1,s2:ChargingStation | s1.charging & s2.
        charging = none
167 }
168
169 //Banned users cannot deal with cars
170 fact NoBannedUsersDealingWithCars{
171     no u:User | some c:Car | u.banned = True and ( c.usedBy =
        u or c.reservedBy = u )
172 }
173
174 //A REF is applicable if the reservation is expired
175 //No other fee are applicable if the reservation is expired
176 fact ReservationExpiredFeeApplicable{
177     all r:RentMade | r.reservationExpired = True iff (REF in r
        .additionalFeeRent and #r.additionalFeeRent = 1)
178     no r:RentMade | r.reservationExpired = False and REF in r.
        additionalFeeRent
179 }
180

```



```

181 //A reservation expired could not be outside safe area
182 fact NoReservationExpiredOutsideSafeArea{
183     no r:RentMade | r.reservationExpired = True and r.
        endSafeArea = False
184 }
185
186 //No discount are applicable if a reservation is expired
187 fact NoDiscountOrFeeIfReservationExpires{
188     all r:RentMade | r.reservationExpired = True implies r.
        discountApplicableRent = none
189 }
190
191 //No passengers can be detected during the ride if the
        reservation is expired
192 fact NoPassengersIfReservationExpires{
193     all r:RentMade | r.reservationExpired = True iff r.
        passengersDuringTheRide = Zero
194 }
195
196 //M2P discount must be applied iff there are at least two
        passengers detected during the ride
197 fact M1PDiscountAppliable{
198     all r:RentMade |( r.passengersDuringTheRide != Zero and r.
        passengersDuringTheRide != One)
        iff M1PD in r.discountApplicableRent
199 }
200
201
202 //BHF discount must be applied iff the car is left with more
        than 50 percent of battery
203 //at the end of the rent
204 fact BHFDiscnountAppliable{
205     all r:RentMade | r.endBatteryLevel = More50Full
        iff BHFD in r.discountApplicableRent
206 }
207
208
209 //CC discount must be applied iff the car is left on charge
        at the end of the ride
210 fact CCDiscnountAppliable{
211     all r:RentMade | r.onChargeAtTheEnd = True
        iff CCD in r.discountApplicableRent
212 }
213
214
215 //If a car is left on charge at the end of the ride it is
        located inside a safe area
216 fact AllCharginStationInSafeArea{
217     all r:RentMade | r.onChargeAtTheEnd = True implies r.
        endSafeArea = True
218 }
219
220 //If a car is left in the charge station determined by the
        MSO at the end of the
221 //ride, it is on charge at the end of the ride
222 fact IfLeftMSOStationIsOnCharge{

```

```

223   all r:RentMade | r.leftMSOstation = True implies r.
      onChargeAtTheEnd = True
224 }
225
226 //MSO discount must be applied iff the car is left in the
      charging station
227 //determined by the MSO
228 fact MSODiscountApplicable{
229   all r:RentMade | r.leftMSOstation = True iff MSOD in r.
      discountApplicableRent
230 }
231
232 //OSA fee must be applied iff the car is left outside a safe
      area at the end of the rent
233 fact OSAFeeMustBeAdded{
234   all r:RentMade | r.endSafeArea = False
235     iff OSAF in r.additionalFeeRent
236 }
237
238 //A3BC fee must be applied if the car is left more than 3km
      away from the nearest
239 //charging station or with battery percentage lower than 20
      percent
240 fact A3BCFeeMustBeAdded{
241   all r:RentMade | r.endPosition = More3kmPowerGrid
242     implies A3BCF in r.additionalFeeRent
243   all r:RentMade | r.endBatteryLevel = Lower20Full
244     implies A3BCF in r.additionalFeeRent
245   all r:RentMade | A3BCF in r.additionalFeeRent
246     implies (r.endPosition = More3kmPowerGrid or r.
      endBatteryLevel = Lower20Full)
247 }
248
249 //A3BC fee cannot be applied if the car is left on charge
250 fact NoA3BCFeeIfOnCharge{
251   no r:RentMade | r.onChargeAtTheEnd = True and A3BCF in
      r.additionalFeeRent
252 }
253
254 //If CC discount is applied the end car position can not be
      more than 3km away
255 //from the nearest power grid
256 fact NoCCDMoreThan3km{
257   no r:RentMade | CCD in r.discountApplicableRent and r.
      endPosition = More3kmPowerGrid
258 }
259
260 // Assertions
261 //Can not exists reserved car with engine turned on
262 assert NoReservedCarWithEngineOn{
263   no c:Car | c.engineOn = True and c.status = Reserved
264 }
265 check NoReservedCarWithEngineOn
266

```

```

267 //Can not exists a car in charge with engine turned on
268 assert NoCarInChargeWithEngineOn{
269     no c:Car | c.engineOn = True and c.onCharge = True
270 }
271 check NoCarInChargeWithEngineOn
272
273 //If a car is left on charge at the end of the rent the
274 //outside safe area fee (OSAF)
275 //can not be applied because alla charging stations are
276 //inside a safe area
277 assert NoOSAFIfOnChargeAtTheEndRent{
278     no r:RentMade | r.onChargeAtTheEnd = True and OSAF in r.
279     additionalFeeRent
280 }
281 check NoOSAFIfOnChargeAtTheEndRent
282
283 //If CCD is applied A3BCF can not be applicable and
284 //viceversa
285 assert NoCCDAndA3BCF{
286     no r:RentMade | CCD in r.discountApplicableRent and A3BCF
287     in r.additionalFeeRent
288 }
289 check NoCCDAndA3BCF
290
291 //If CC discount is applied the end car position can not be
292 //more than 3km away
293 //from the nearest power grid
294 assert NoMSODMoreThan3km{
295     no r:RentMade | MSOD in r.discountApplicableRent and r.
296     endPosition = More3kmPowerGrid
297 }
298 check NoMSODMoreThan3km
299
300 //If MSOD is applied A3BCF can not be applicable and
301 //viceversa
302 assert NoMSODAndA3BCF{
303     no r:RentMade | MSOD in r.discountApplicableRent and A3BCF
304     in r.additionalFeeRent
305 }
306 check NoMSODAndA3BCF
307
308 pred show{#charging > 2 some u:LoggedUser | u.banned = True}
309 run show for 10 but exactly 2 ChargingStation, exactly 4 Car
310 , exactly 4 LoggedUser, exactly 2 RentMade

```

7 commands were executed. The results are:

- #1: No counterexample found. NoReservedCarWithEngineOn may be valid.
- #2: No counterexample found. NoCarInChargeWithEngineOn may be valid.
- #3: No counterexample found. NoOSAFIfOnChargeAtTheEndRent may be valid.
- #4: No counterexample found. NoCCDAndA3BCF may be valid.
- #5: No counterexample found. NoMSODMoreThan3km may be valid.
- #6: No counterexample found. NoMSODAndA3BCF may be valid.
- #7: **Instance found.** show is consistent.

Figure 17: Alloy execution result

A.2 Generated worlds

Note that in [Figure 18](#) LoggedUser3 has been banned *after* completing Rent-Made0.

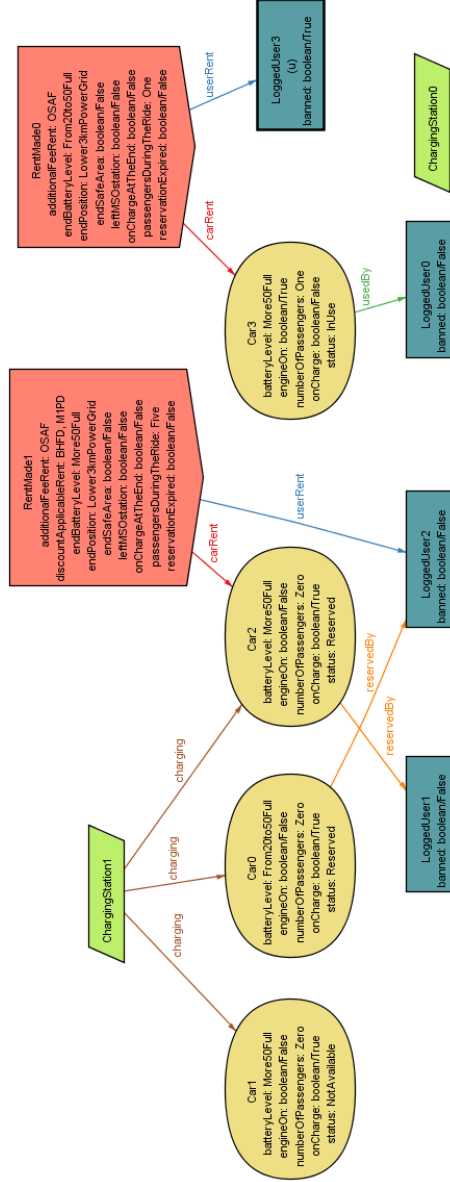


Figure 18: First alloy generated world

Note that in Figure 19 RentMade1 is actually a reservation expired of Car3 made by LoggedUser2. He now has reserved Car1.

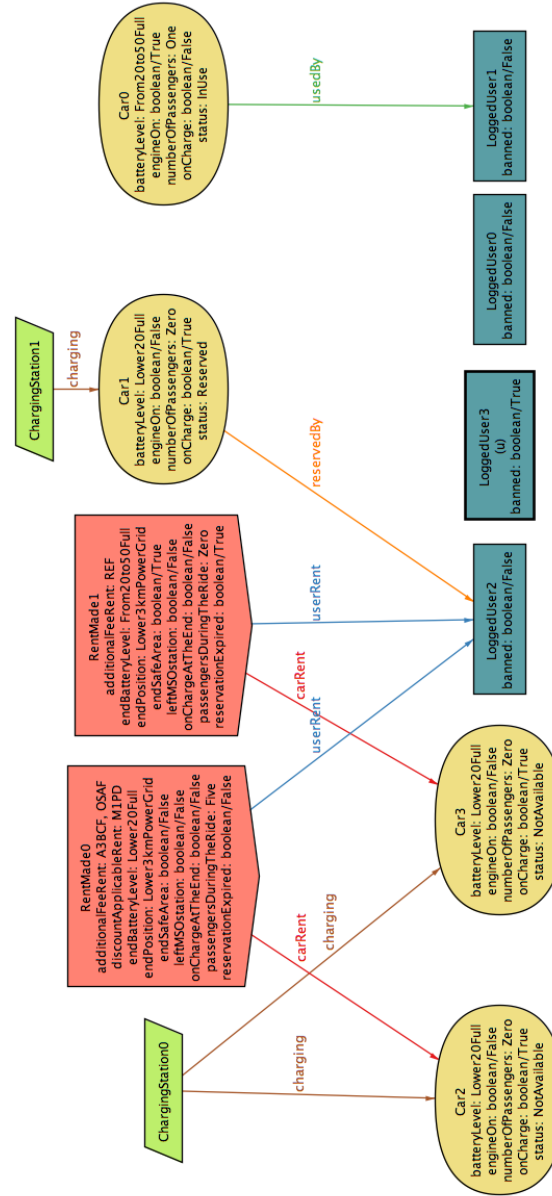


Figure 19: Second alloy generated world

B Software and tools used

For the development of this document we used

- L^AT_EX as document preparation system
- GitHub as version control system
- Draw.io for graphs

C Hours of Work

This is the amount of time spent to redact this document:

- **Section 1 - Introduction**
 - Amedeo Cavallo - 2 hours
 - Mattia Calabrese - 1 hour
 - Federico Capaccio - 1 hour

D Changelog

- **v1.0** October 23, 2019
 - Initial RASD document structuring and redaction
 - Introduction (Purpose and Scope sections)

References

- [1] B. Nuseibeh, S. Easterbrook, *Requirements Engineering: A Roadmap*, 2000
- [2] P. Zave, *Classification of Research Efforts in Requirements Engineering*, ACM Computing Surveys, 1997
- [3] E. Di Nitto, L. Mottola, *Software Engineering 2 Assignment*, AA 2019-2020
- [4] A. Stone, “Chain of custody: How to ensure digital evidence stands up in court,” September 2015
- [5] IEEE Std 830:1993, *IEEE Recommended Practice for Software Requirements Specifications*, 1993