POLITECNICO
MILANO 1863

Requirement Analysis and Specifications Document

Joan Ficapal Vila (876805), Nicolò Vendramin (879113)

November 13, 2016
v1.0

Revision History

Revision	Date	Author(s)	Description
0.1	31.10.16	N and J	First draft of the introduction
0.2	31.11.16	N and J	Adding the Domain Properties
0.3	1.10.16	N and J	Definition of Goals and Constraints
0.4	2.11.16	N and J	Conclusion of the overall section
0.5	3.11.16	N and J	Introduction to the third section
0.6	4.11.16	N and J	Adding the mockup examples of the UI
0.7	5.11.16	N and J	Adding the Functional requirements
0.8	6.11.16	N and J	Adding more details about non functional req
0.9	7.11.16	N and J	Describing the scenarios
0.10	8.11.16	N and J	Adding the Use Case Diagram
0.11	9.11.16	N and J	Adding the Use Case Description
0.12	10.11.16	N and J	Adding the Sequence Diagrams
0.13	10.11.16	N and J	Realizing the allow signatures
0.14	11.11.16	N and J	Realizing the alloy facts
0.15	12.11.16	N and J	Realizing the alloy assertions
0.16	13.11.16	N and J	Realizing the alloy predicates
1.0	13.11.16	N and J	First release of the complete document

Hours of work

- Joan Ficapal Vila : 25 hours
- Nicolò Vendramin : 26 hours

Contents

1 Introduction

1.1 Description of the given Problem

We will project and implement the PowerEnJoy system for the car-sharing service of electric cars. The service will be based on a mobile application interacting with an online server that can be accessed by anyone in possess of an iOS or Android smartphone.

The system allows the user to log in or, in case he/she is a new user, to register providing credentials and payment information. After logging into the application it is also possible to locate the available cars within a certain distance from the user's position or from a specified address. The system also provides the possibility to book one of the shown cars. After the booking is completed the user must reach the vehicle within one hour, otherwise the booking expires, the car is tagged again as available and the user is charged with a fee. When a booking is active the system provides the user with the possibility to ask the system to unlock the car in order to start the rent. During the rent the client is informed of the real-time cost of the rent through a screen placed in the car; moreover, as soon as the car is parked in one of the predefined safe areas, the system calculates the final charge for the user applying some discounts in case of certain positive behaviours being detected. When the rent is over and the user has left the car the system locks it again and adds it to the available cars.

1.1.1 Target User

The only actor that is going to interact with the system is the final customer of the rental service, above called: the user. The systems allows only registered users to access the rental service, and offers to the new users nothing but the possibility to register.

1.2 Existing system

The mobile application which composes the new system interacts with a previously designed system which has been developed in order to administrate the vehicles that are used for the rental service. This existing system periodically retrieves data from the cars and populates a database in which for every car is stored the information about power, position and everything else that can be useful for the service. In addition it offers to the employees of the company the possibility to manage the available vehicles manually, for example removing them by the system if a problem is detected, and some features for automatic problem handling such as removing a car with unsufficient power and signaling it to an operator that can move the car and plug it in in one of the service areas.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- **User:** It is used to indicate the generic person that access the application.

- **Client:** This term is used to identify each user registered in the application, that has been correctly identified by the system through the log in procedure. In order to register to the service the client need to provide the following information:
  - Name
  - Surname
  - Birthday
  - Driving License Number
  - Driving License Expiration Date
  - e-Mail (unique)
  - Password for the service
  - Credit/Debit Card Number
  - Credit/Debit Card Security Code
- **Car:** It is a vehicle that can be used for the rental service offered by the application. Every car is equipped with GPS, Power level sensor, pressure sensors, plug-detection sensor (distinguish whether the car is plugged or not) ,a timer that monitors the duration of each rent and a small monitor.
- **Ride:** It is the unit of use of the rental service. The period between the ingition of the car and the end of the rent. The client is charged proportionally to the duration of the ride.
- **Reservation:** It is the booking of one of the available cars so that other people cannot use it. A reservation lasts for a period of maximum one hour after which it expires. In order to open a vehicle the client must reserve it before. It is identified by the expiration time, the user id and the car identifier.
- **Notification:** It is a pop up message that is displayed on the client's smartphone. Notifications are used by the application to comunicate important messages.
- **System:** It is the new system, composed by a mobile application for devices running either iOS or Android. The new System access the data contained in the database of the old system.
- **Legacy:** It is the existing system used to manage the vehicles and their state. The Legacy system is not going to be replaced by the new System. The two system are going to interact in order to offer to the final customer access to the service.
- **Safe Area:** It is one of the zones of the city in which is possible to leave the car at the end of the rent.
- **Re-charge Site:** It is a Safe Area in which there is the possibility to plug the car to refill the battery.
- **Billing Service:** It is the service that is used from the application to charge the user with the amount needed to pay the ride.

- **Client Module:** It is the software component installed on the phone of the user. It has the role of a remote interface to the server.
- **Server Module:** It is the software component running on the servers of the company that provides the functionalities.

1.3.2 Acronyms

- **API:** Access point of Interface. This term is used to indicate the interfaces exposed to access a software service from another software service.
- **GPS:** Global Positioning System. This is the most common localization tool mounted on a wide variety of devices such as phones and laptops.
- **DBMS:** Data Base Management System. Is the software used to query the database.
- **UI:** User Interface. The interface whit which the user access the functionalities provided by a software product.

1.4 Identifying Stakeholders

The main stakeholder of the system is the company commissioning its realization. Other stakeholders are the city (both governance and population) in which the system has to be used, the other companies offering similar services and also the public transportations providers. Ecologists organizations can also be considered as stakeholders since this new service is going to provide a "green" alternative for moving in the city.

1.5 Reference Documents

This document has been written following the IEEE standard number 830-1998 "Recommmended Practice fir Siftware Requirements Secifications".

1.6 Overview

The rest of this document contains an overall description of the software product (section 2), the specific requirements for the system (section 3) and the appendix (section 4). The appendix contains all UML diagrams useful to describe the functionalities of the software, some scenarios and the Alloy modeling.

2 Overall Description

2.1 Domain Properties and Assumptions

In this section we analyize all the properties of the application domain of the system in order to describe the environment in which it operates.

- All the GPS (both on the user phones and on the cars) always give the right postion.

- All the power level and pressure sensors mounted on the cars are giving the correct data.
- The previous system which manages the cars is already populated and working.
- The credentials provided by the user are unique (e-mail as id).
- If a car is marked as available, it is present on the map and it is actually available for the rent.
- All the cars present on the system are working properly.
- A car present on the system can be only in one of the following states: available, booked or in rent.
- The client is billed for the use of the service using an external service.
- Any information about the vehicles is gathered by the new System via direct access to the database of the existing system.
- The other system works correctly without problems.
- In case of any extraordinary event handling (car accidents, system not working) the operators that works for the existing service will manually intervene and manage the problem.
- Any time a car is left unplugged with very low power level, it is not retagged as available by the existing system until the power level reaches again a minimum level.
- We assume that is possible to check the corrispondence between Name+Surname+Birthday and Driving License Number+Expiration Date using an external institutional service.
- We assume that when a car is unlocked, the one who opens it and uses it is always the owner of the reservation. For this reason no further cross-check is needed.
- We assume that when a car is opened the user will start using it immediately.
- We assume that when the user concludes the session he is out of the car and the old system manages to lock the vehicle.
- We assume that is possible to identify without ambiguity the cars using their plate.
- The billing service never fails or if it does it handles internally any problem.
- The operators of the existing service periodically checks the cars in order to ensure that they are in proper conditions.
- We assume that the database with the data related to the cars is associated to a middleware that is able to notify the system when the status of a variable changes (e.g.: engine ignition sends a control message to the existing system which changes the corresponding variable in the database. The change is advertised by the middleware to the other system)

- Between the moment in which the map with available cars is updated the most recent time and the moment in which the user performs the booking none of the available cars has changed its status.

2.2 Goals

Here is the list of goals, intended as the actions that the system allow the user to do:

- **[G0]** Allows everyone to register on the system
- **[G1]** Allows the client to login
- **[G2]** Allows the client to check the cars available within a certain distance from a specified position
- **[G3]** Allows the client to check the cars available within a certain distance from their position
- **[G4]** Allows the client to book a vehicle for a rent
- **[G5]** Allows the client to open the booked vehicle if the booking has not expired yet and he is close to it
- **[G6]** Allows the client to update his/her credentials or payment details
- **[G7]** Conclude the rent and be notified when the rent is over and the payment is successfully completed
- **[G8]** Allows the client to inspect the power level of a selected car before booking it
- **[G9]** Allows the client to identify the localization of the safe areas where the vehicle can be left
- **[G10]** Allows the client to cancel an existing booking up to the expiration time.

2.3 Actors Identifying

- **User:** Everyone willing to use the application that has not been registered yet. They can't access the application functionality but the registration form.
- **Client:** Every registered user willing to use the application. They can access the complete functionality of the application.
- **Online Application Server:** This is the main brain of the application system which executes the queries of the users and clients through the Mobile Application, taking also into account the situation in the other actors such as Billing system and Existing System.
- **Mobile Application:** This is the interface from where the users can query query demands and visualise the answers by the Online Application Server.

- **Billing Service:** Provides APIs to charge the user with the cost of the ride.
- **Existing Service:** It is the previous system which handles the majority of situations that our application takes for granted.

2.4 Constraints

2.4.1 Regulatory Policies

The system requires the user smartphone location, who must agree to provide it together with its e-mail, phone and personal data in order to allow a positive application performance. The form will also explain its privacy law and will make users check the option "I've read and I agree to service terms and conditions" before proceeding to users registration.

2.4.2 Hardware Limitations

- Mobile Application:
  - Smartphone.
  - 3G Connection.
  - GPS enabled.
  - 30 Mb of free space in the users device.
- Online Application Service:
  - Apache Server with permissions to access the existing system's database.

2.4.3 Parallel Operations

The application server must support parallel operations from different users or clients.

2.5 Interfaces to other applications

The mobile application will interact with the web server, querying demands and retrieving the results computed by this one. The application server will interact with the existent system to handle general errors and to retrieves or update the data about the cars used for the service. The application server will interact with the billing system to handle all the tasks related to payments and billing.

3 Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The service is accessed by the user through a mobile application with a simple and immediate user interface. The user interface must be easy to understand and provide access to each functionality with at most 3 taps. Example for the style of the user interface are represented by the following figures: Figure 1, Figure 2 and Figure 3.

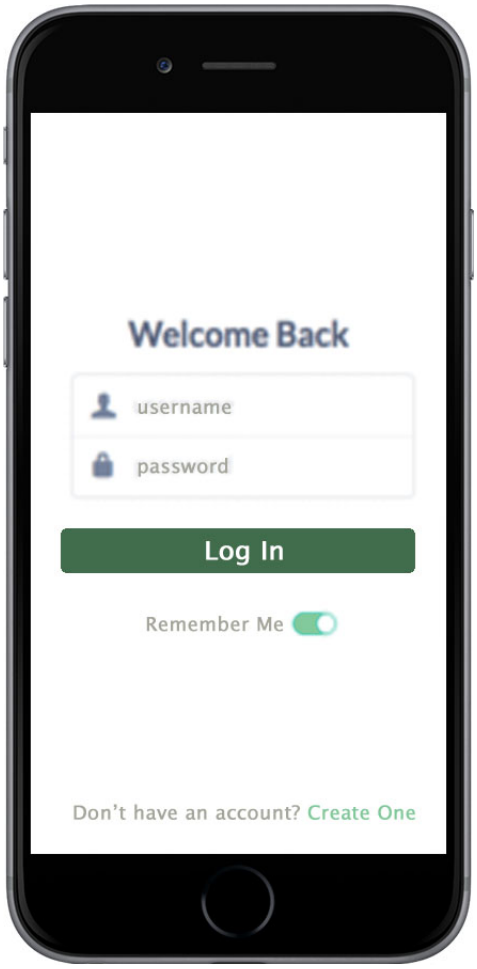

Figure 1: Mock-up of the desirede interface: Log-in

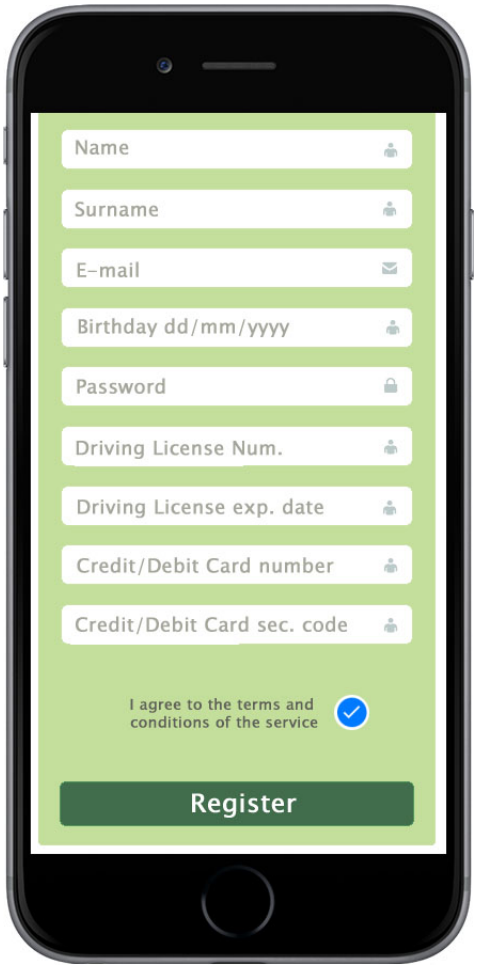

Figure 2: Mock-up of the desirede interface: sign up

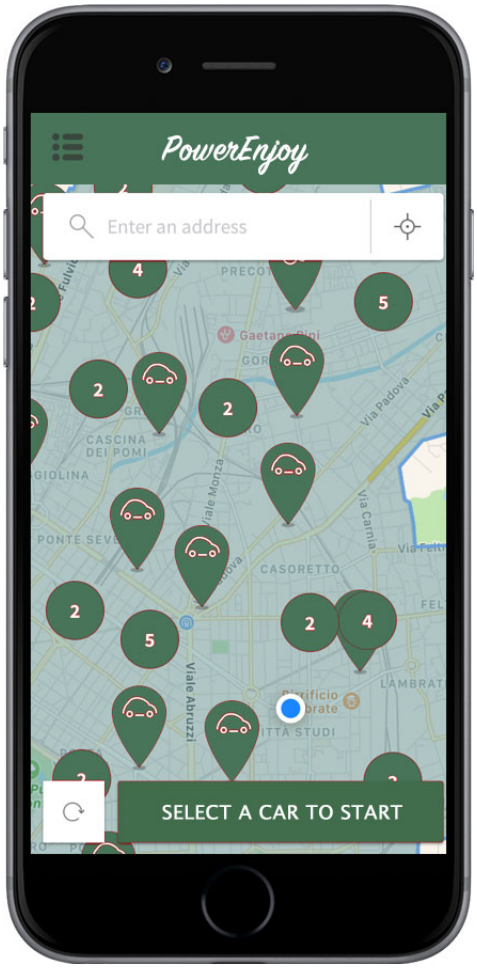

Figure 3: Mock-up of the desirede interface: map

3.1.2 Software Interfaces

The service must communicate with the software interfaces of both the legacy system for the management of the cars and the billing service, which provides support for all the payment and billing operations. In addition the server module must be integrated with a DBMS in order to access the data stored in the data base of the existing service.

3.1.3 Communication Interface

The client module and the server module communicate using the internet and the standard TCP/IP protocol.

3.2 Functional Requirements

Here are listed the requirements of the system needed in order to achieve each of the goals introduced in the section 2.2

- Goal [G0] "Allows everyone to register on the system":
  - [**R0**] Recognise if an e-mail address has already been used.
  - [**R1**] Store the data of the new subscriber.
  - [**R2**] Validate the data relating the driving license.
  - [**R3**] Provide access to the system for the new subscriber.
- Goal [G1] "Allows the client to login":
  - [**R4**] Check the correctness of the password+username couple.
- Goal [G2] "Allows the client to check the cars available within a certain distance from a specified position":
  - [**R5**] Receive an address as an input string from the user.
  - [**R6**] Select all the cars available within 3 km from the address.
  - [**R7**] Display the map of the zone with the available cars on it.
- Goal [G3] "Allows the client to check the cars available within a certain distance from his position":
  - [**R8**] Get the address of the user from the GPS of the mobile device.
  - [**R6**] Select all the cars available within 3 km from the address.
  - [**R7**] Display the map of the zone with the available cars on it.
- Goal [G4] "Allows the client to book a vehicle for a rent":
  - [**R9**] Selecting a vehicle from the map.
  - [**R10**] Set the car as booked.
  - [**R11**] Start the timer associated with the booking.
  - [**R12**] In case of expiration intrust the billing service to charge the client with a fee of 1$
  - [**R13**] Checks the validity of the Driving License of the user

- Goal [G5] "Allows the client to open the booked vehicle if the booking has not expired yet and he is close to it":
  - [**R14**] Check if the client is close to the booked vehicle.
  - [**R15**] Check if the booking is still valid.
  - [**R16**] Set the car as in rent.
  - [**R17**] End the timer of the booking.
  - [**R18**] Delete the record about the reservation.
  - [**R19**] Open the correct vehicle.
  - [**R20**] Create a record of the rent initializing the time at 0.
- Goal [G6] "Allows the client to update his/her credentials or payment details":
  - [**R21**] Display the credentials of the user.
  - [**R22**] Allow the modification of any of the fields.
  - [**R2**] Validate the data relating the driving license.
  - [**R23**] Make the modifications permanent.
- Goal [G7] "Conclude the rent and be notified when the rent is over and the payment is successfully completed":
  - [**R24**] Check when the rent is over.
  - [**R25**] Assign to the billing service the task of charging the client.
  - [**R26**] Notify the client for the conclusion of the ride.
  - [**R27**] The billing system confirms the correct completion of the payment.
- Goal [G8] "Allows the client to inspect the power level of a selected car before booking it":
  - [**R9**] Select a vehicle from the map.
  - [**R28**] Display the details of the vehicle.
- Goal [G9] "Allows the client to identify the localization of the safe areas where the vehicle can be left":
  - [**R29**] Displaying a map of the city with highlighted the safe areas.
  - [**R30**] Displaying the address for each of the safe areas.
- Goal [G10] "Allows the client to cancel an existing booking up to the expiration time.":
  - [**R31**] Display existing booking if present.
  - [**R15**] Check if the booking is still valid.
  - [**R32**] Cancel the booking.
- Other requirements:
  - [**R33**] Apply discounts of 10% to the rides in which the user took at least 2 passengers with him to incentivate car sharing.

- [**R34**] Apply discounts of 20% to the last ride if the car is left with at least 50% of the battery level
- [**R35**] Apply discounts of 30% to the last ride if the client leaves the car in a zone where it can be recharged and plugs it in.
- [**R36**] Apply an extra charge of 30% of the cost of the last ride to those users that leave the car at at least 3 Km from the nearest power grid.
- [**R37**] Apply an extra charge of 30% of the cost of the last ride to those users that leave the car with at least 80% of the battery empty.

3.3 Performance Requirements

The application will properly cut its processes when the user is not checking and the phone screen is black in order to save battery;It must also have a maximum response time of 3 seconds for any operation. Once the user introduces its credentials they will remain kept and he will be logged automatically on future tries when re-opening the application.

3.4 Design Constraints

The application will be developed in Swift and Java for iOS and Android respectively. The application fill 30 Mb at maximum on the user device.

3.5 Software System Attributes

3.5.1 Usability

All functionalities must me reachable by the user within 3 clicks, once logged in. The car map interface [Fig. 2] will englobe different car instances in an unified bubble if they are too close to be represented clearly (Google Maps API criteria), the bubble tag will indicate the number of cars in that zone. The typography and images will follow the responsive model in order to assure a good experience for the user in its device, resizing them depending on the particular screen-size.

3.5.2 Availability

The application system will be available 24h hours per day, for each day of the week.

3.5.3 Security

Data protection in transmission must be guaranteed during authentication phase.

3.5.4 Maintainability

Since is a 24/7 service, it should be continuously maintained without interrupting this policy. If an update or a critical actualization is needed, it would be indicated to users in advance.

3.5.5 Portability

The application can be used in any iOS or Android smartphone.

4 Scenarios and UML Models

4.1 Scenarios

4.1.1 Scenario 1

A random person needs a vehicle in order to reach his home after going out with his friends, this situation happens during night and the public transport is not working and he hasn't brought his car with him. To solve this problem, our character uses the PowerEnJoy application on his smart phone, that he has previously used. After logging in, he finds a car 800 meters away from his position and even if it has just 40 % of the battery he books it. Once he is 2 meters away from the car the application allows him to open it so that the ride can start. Our main character successfully reaches his home, but he forgets to plug the car in at the safe area, and when he sees the confirmation of the realized payment he notices that he payed an additional 30% fee.

4.1.2 Scenario 2

A girl needs a car to arrive at work on time, because she didn't hear the alarm of her clock, and remembers about the PowerEnJoy application. She decides to download and install it and, as soon as she opens the application she is asked to register, After fulfilling the subscription form, she is granted with a full access to the service functionalities. While she is leaving home, she is called by a friend, who gently offers to bring her at work with her car. For this reason, she opens the application again and cancels the booking.

4.1.3 Scenario 3

An important lawyer is in a hurry because she needs to meet with some people in a place, she is worried because she's still on the train which leaves her pretty far from her final destination. Plus, the public transports are on strike. She usually uses PowerEnJoy on those particular situations. After opening it, she logs in and looks for cars in the surroundings of the train station and books a car 300m away from the station location. As soon as she arrives, she receives a call informing that the meeting has been canceled. For this reason, she takes the train again to go back home. She sits down and starts reading a book, forgetting to cancel the reservation . After 1 hour she receives a notification of the expiration of his booking and of the 1$fee while she's on the train.

4.1.4 Scenario 4

An old guy's license expired 2 days ago. He his really concerned about the environment and about city pollution, so he likes to use environment friendly vehicles such as the ones provided in PowerEnJoy, an application in which he is registered since it came out to market. He decides to use his bike to reach the office and renew his license. After his new details are provided, he opens

PowerEnJoy to update his credentials. Once the server verifies them, he is notified about the successful credential update.

4.1.5 Scenario 5

Fabrizio is the singer of a band that plays Italian music. Usually Gigi, the only one of the group's members that had a car, goes to the house of each one of the 4 components of the group to pick them up to go to the studio where they have their rehearsals. Unfortunately, Gigi's car broke last Wednesday they don't know how to go to the studio. Fabrizio remembers about PowerEnjoy, logs into the app, checks the safe areas and sees that luckily there is a safe area with plugs really close to their rehearsal's venue. He proceeds to book the car and start the rent. He picks up his friends and, after arriving to their final destination, he parks and plugs it in. For this reason Fabrizio receives a double discount on it's charge: 30% because he plugged the car in and an additional 10% because he shared the ride with 3 other passengers.

4.2 Use Case Diagrams

Next Page it follows the Use Case Diagram for the system (Figure 4). The subsections of the current 4.2 section are going to explain in detail each of the use cases present in this diagram.

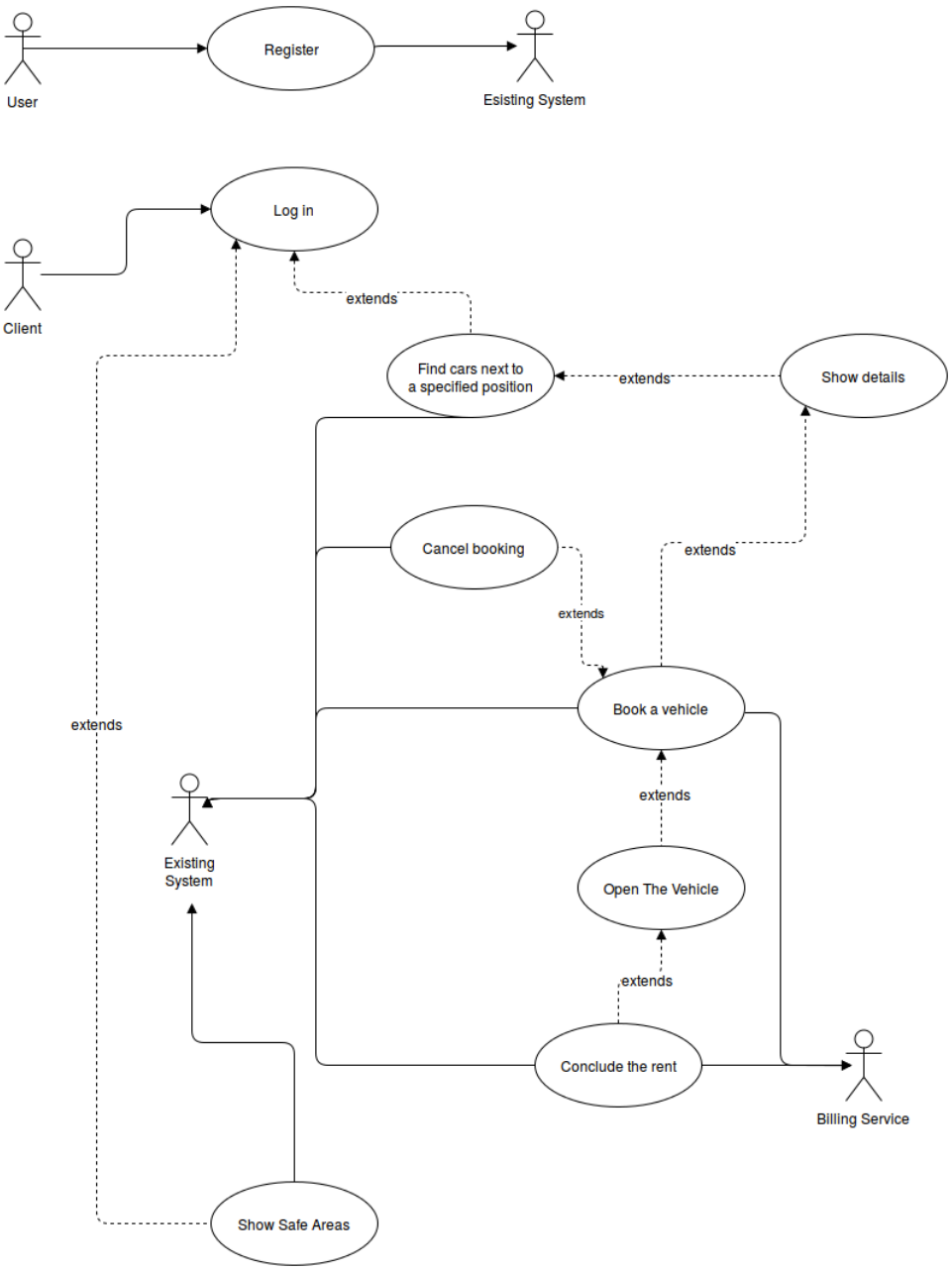

Figure 4: Use Case Diagram

4.2.1 Register

The user registers am account for the service.

Entry Conditions:

1. The user opens the application.
2. The user selects the "create new account" button

Flow of events:

1. The user fulfills the form, accepts the terms of use and sumbits the information.
2. The user information is sent to the existing system in a secure way, using encrypted protocols.
3. The existing system registers the user if the information is correct.
4. The existing sustem informs the application that the registration was successful.

Exit Conditions: The application notifies the client that the registration was successful.

Exceptions: The user doesn't accept the terms of use. A mistake is found in the users registration form. In any of those cases the users is notified of the exception and the process is restarted.

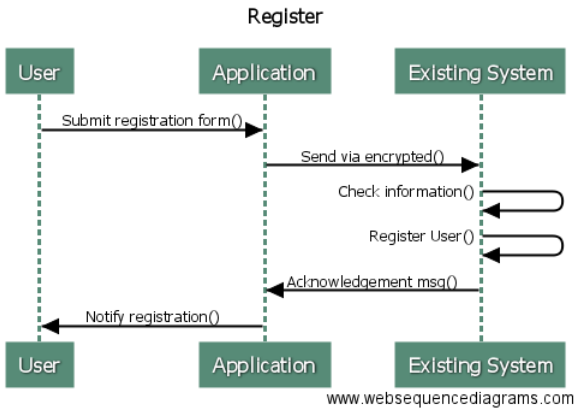

Figure 5: Register Sequence Diagram

4.2.2 Log in

The client logs into his personal account.

Entry Conditions:

1. The client opens the application.
2. The client is already registered.
3. The client is not logged in.

Flow of events:

1. The user fulfills the log in form.
2. The user information is sent to the existing system in a secure way, using encrypted protocols.
3. The existing system log in the user if the credentials provided are correct.
4. The existing sy stem informs the application that the log in was successful.

Exit Conditions: The application shows a screen with a button to start using the service.

Exceptions: The credentials provided don't match anyone registered on the system.

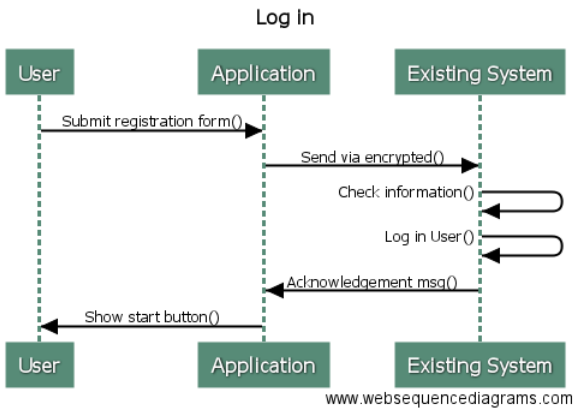

Figure 6: Log in Sequence Diagram

4.2.3 Find Cars Next to a Specified Position

The client looks for the cars near a specified position. It can be either his position or an input address.

Entry Conditions:

1. The client is logged in the application.
2. The client on the main page clicks the botton to start the research.

Flow of events:

1. The application asks the user whether to use the position from the GPS or another provided by the user as input.
2. In case the user wants to use his current location the Application access the GPS data.
3. In case the user wants to input a position, the user inputs the position.
4. The application sends the data to the existing system.
5. The existing system replies sending all the cars that match the condition to the application.

Exit Conditions: The application shows a map on which are displayed the available cars.

Exceptions: No available cars matching the condition.

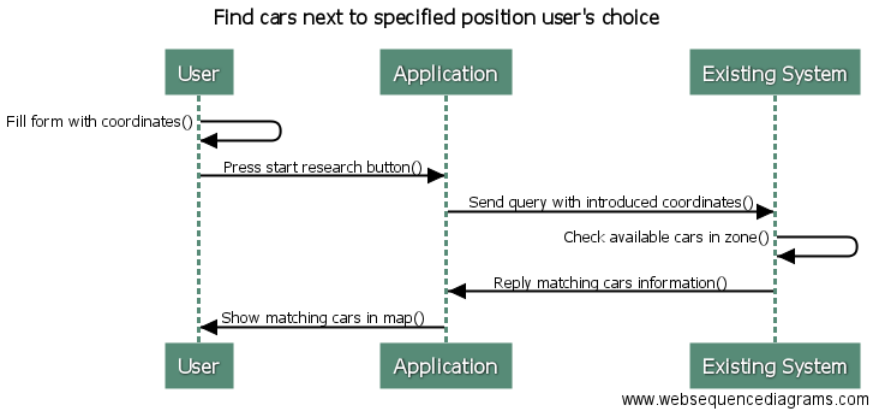

Figure 7: Find Car Next to a Specified Position Sequence Diagram

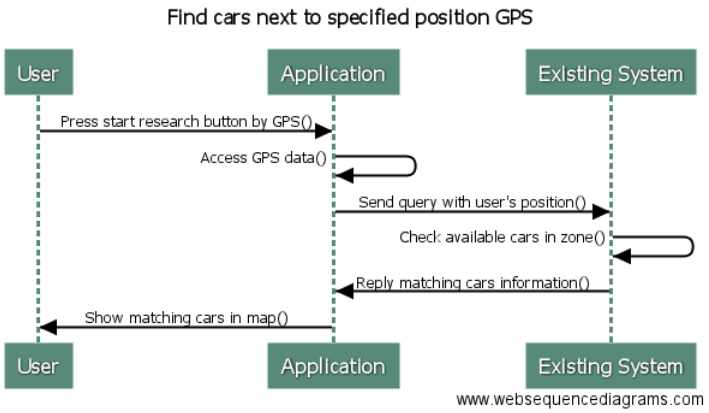

Figure 8: Find Car Next to a your Position Sequence Diagram

4.2.4 Show Details

The client inspect the details of a specific vehicle.

Entry Conditions:

1. The client is logged in the application.
2. The client has performed a research of the cars within a certain position.
3. The client double-taps one of the cars on the present on the map.

Flow of events:

1. The application loads the data cached about the selected car.

Exit Conditions: The application shows a screen with the details of the car and with 2 buttons, one to exit and one 2 book.

Exceptions: No exceptions can happen within this use case.

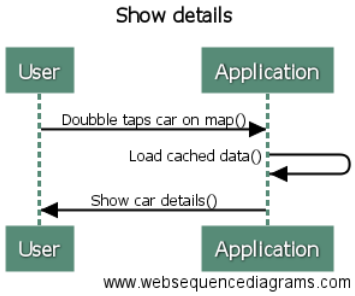

Figure 9: Show Details Sequence Diagram

4.2.5 Book a Vehicle

The client books a vehicle.

Entry Conditions:

1. The client is logged in the application, and is inspecting a vehicle.
2. The client agrees to book the selected vehicle and proceeds to press the accept button.

Flow of events:

1. The application sends a query to the existing system.
2. The existing system books the vehicle, labels the vehicle as booked and the client as owner.
3. The system starts a timer associated with the booking.
4. The existing system informs the system about the successful booking.

Exit Conditions: The application informs the user about the successful booking and shows and shows a screen with the details of the reserved car, including the address and a button to cancel the booking.

Exceptions: The booking timer expires. In that case the billing service is asked to charge the user with 1$. The user is enabled to book vehicles. The system tags the car as available.

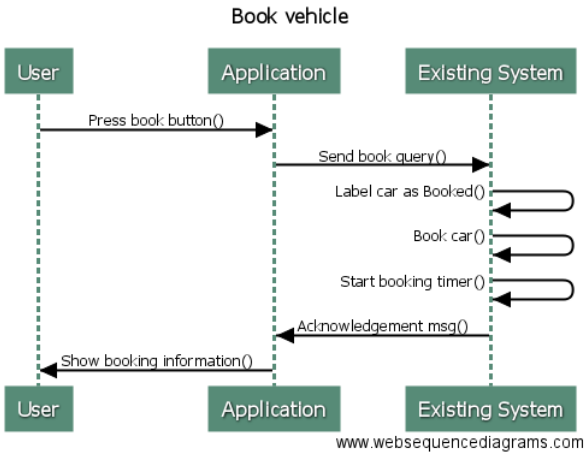

Figure 10: Book Vehicle Sequence Diagram

4.2.6 Cancel

The client cancels an active booking.

Entry Conditions:

1. The client is logged in the application.
2. The client has an active booking.
3. The client presses the cancel button.

Flow of events:

1. The application sends a request to existing system to cancel the booking.
2. The existing system tags the booking as cancelled, the car as free and enables bookings for the client.
3. The existing system stops the booking timer.
4. The existing system confirms to the application the successful completion of the process.
5. The application informs the user of the correct deletion of the booking.

Exit Conditions: The application shows again the main page of the application.

Exceptions: No exceptions can happen within this use case.

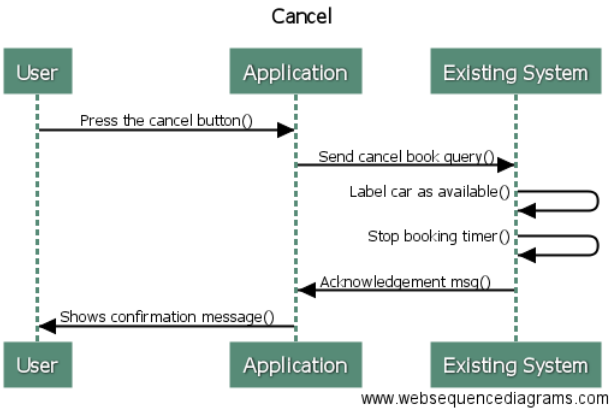

Figure 11: Cancel Sequence Diagram

4.2.7 Open the Vehicle

The client opens the vehicle.

Entry Conditions:

1. The client is logged in the application and has an active and valid booking.
2. The client is close to the vehicle.
3. The client clicks on the "Open" button.

Flow of events:

1. The application sends a query to the existing system.
2. The existing system checks if the client is close enough to the car.
3. The existing system opens the vehicle.
4. The existing system informs the system about the successful opening.

Exit Conditions: The application informs the user about the successful opening of the car. The client enters the car and can start the ride.

Exceptions: The user is not close enough.

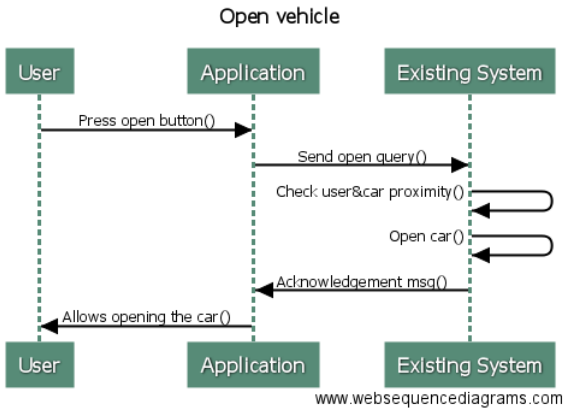

Figure 12: Open Vehicle Sequence Diagram

4.2.8 Conclude the Rent

The client concludes the session.

Entry Conditions:

1. The client is inside the car that he previously booked.
2. The client wants to conclude his rent.
3. The client clicks on the "Conclude" button.

Flow of events:

1. The application sends a query to the existing system.
2. The existing system checks whether the car is in a safa area or not.
3. The ride is ended.
4. The system calculates the appropriate discounts or extra charges.
5. The system prompts the billing service to charge the user with the proper amount.
6. The user is allowed again to book vehicles.
7. The car is tagged as available.

Exit Conditions: The client can leave the car and the application goes back to the main page.

Exceptions: The car is not in a safe area. The car has too low battery, and in that case the existing system would not retag it as available.

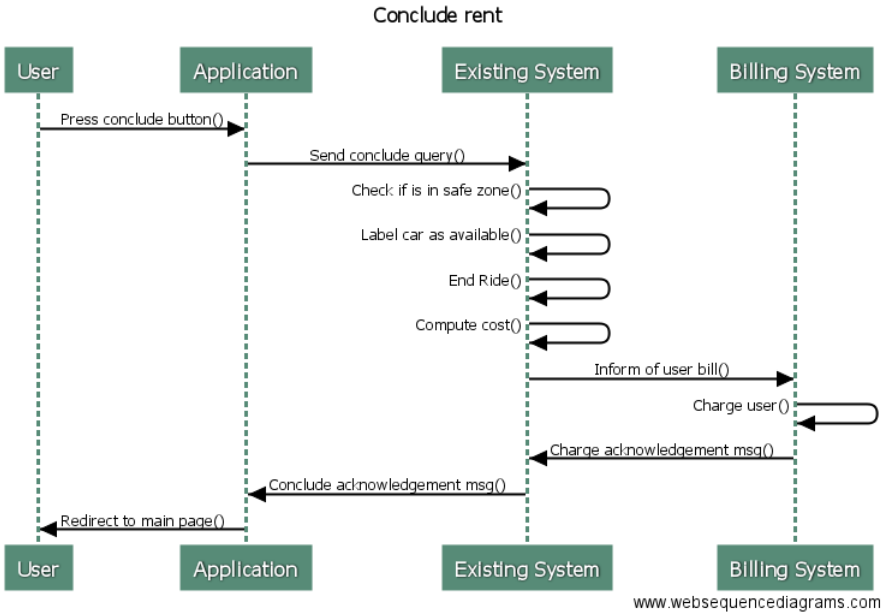

Figure 13: Conclude the Rent Sequence Diagram

4.2.9 Show Safe Areas

The application shows the list of the Safe Areas.

Entry Conditions:

1. The client is logged in into the application.
2. The client clicks on the "Show Safe Areas"

Flow of events:

1. The application sends a query to the existing system.
2. The existing systems replies with a list of adresses.

Exit Conditions: The application shows a page with the list of Safe Areas with the possibility to display them as point on a map.

Exceptions: Nothing.

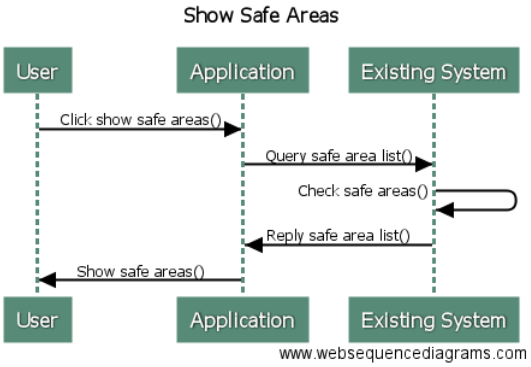

Figure 14: Conclude the Rent Sequence Diagram

4.3 State Chart Models

The following models represent the state chart for the Car and Booking entity. The other entities are not represented with the state chart model because their representation doesn't fit well into a discrete state chart.

4.3.1 Car's State Chart

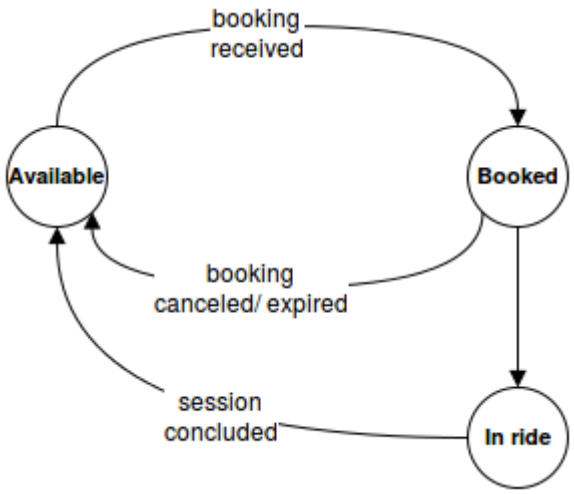

Figure 15: Car's state chart model

4.3.2 Booking's State Chart

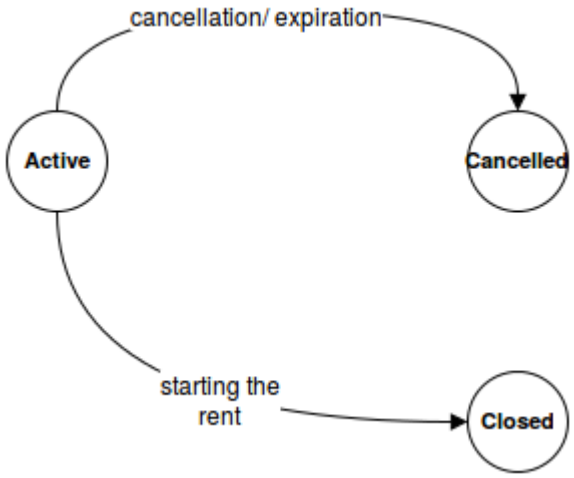

Figure 16: Booking's state chart model

4.4 Class Diagram

In the following page is reported the class diagram of the modellized entities of the world. All the details about the methods are omitted to make it easier to be understood. In addition in some cases also some details about the attributes of the single class is left out in order to make the representation more direct and easy-to-read.

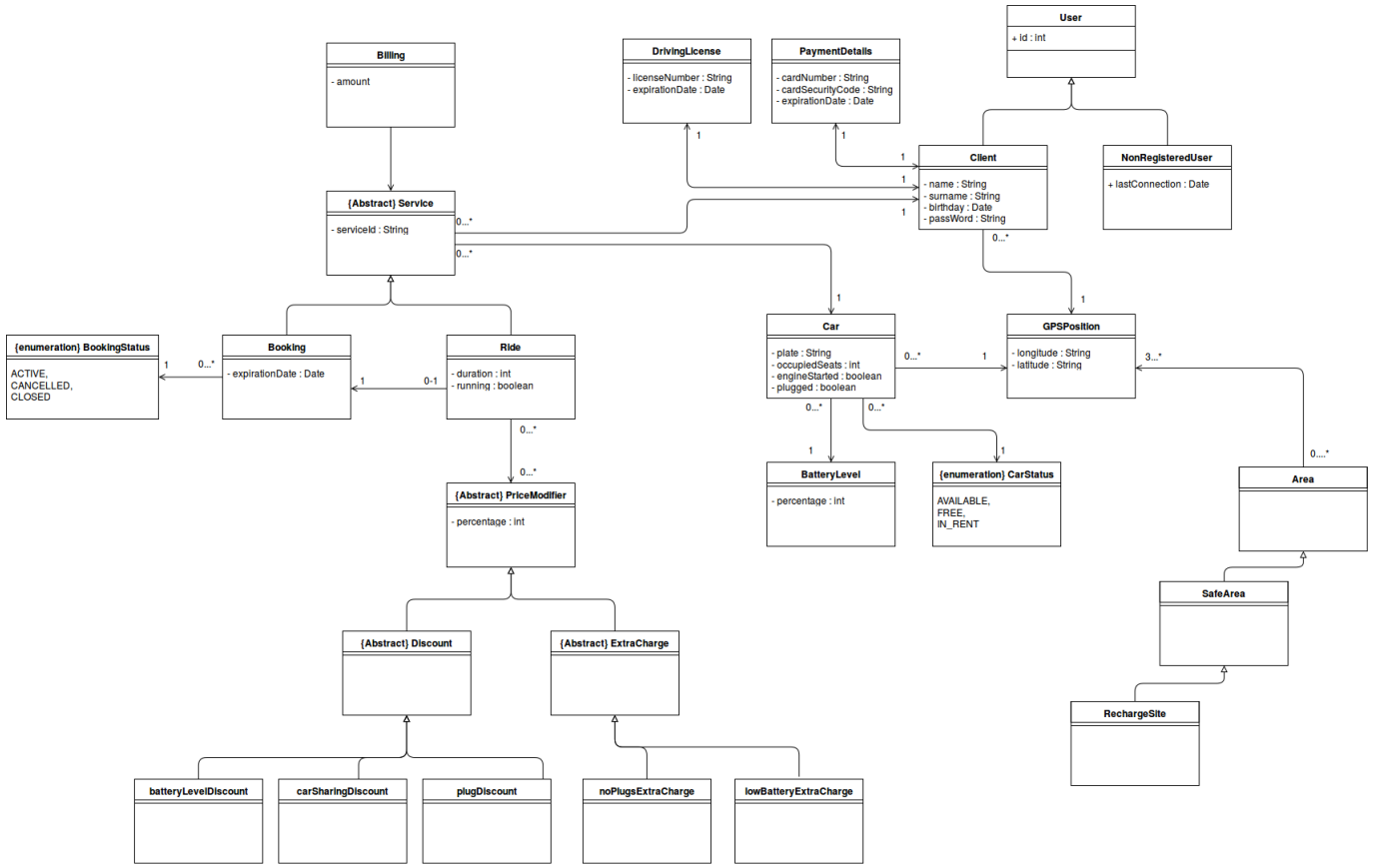

Figure 17: Class Diagram Model

5 Appendix

5.1 Signatures

```
open util/boolean
open util/integer

//DATA TYPES
sig Strings{}
sig Date{}



//SIGNATURES
sig User{
  id: one Int,
}

sig NonRegisteredUser extends User{
  lastConnection : lone Date,
}

sig GPSPosition{
  latitude : one Strings,
  longitude : one Strings,
}

sig PaymentDetails{
  cardNumber : one Strings,
  cardSecurityCode : one Strings,
  expirationDate : one Date,
  client : one Client,
}

sig DrivingLicense{
  licenseNumber : one Strings,
  expirationDate : one Date,
  client : one Client,
}

sig Client extends User{
  name : one Strings,
  surname : one Strings,
  birthday : one Date,
  password : one Strings,
  lastKnownPosition : one GPSPosition,
  drivingLicense : one DrivingLicense,
  paymentDetails : one PaymentDetails,
}

sig Billing{
  amount : one Strings,
  service : one Service,
}
```

```
51
52  abstract sig Service{
53    serviceId : one Int,
54    client : one Client,
55    car : one Car,
56    status : one ServiceStatus,
57  }
58
59  sig Ride extends Service{
60    duration : one Int,
61    booking : one Booking,
62    modifiers : set PriceModifier,
63  }
64
65  sig Booking extends Service{
66    expirationDate : one Date,
67    ride : lone Ride,
68  }
69
70  enum ServiceStatus {
71    ACTIVE,
72    CANCELLED,
73    CLOSED
74  }
75
76  abstract sig PriceModifier{
77    percentage : one Int,
78  }
79  {
80    percentage > 0
81    percentage <= 100
82  }
83
84  abstract sig Discount extends PriceModifier{
85  }
86
87  abstract sig ExtraCharge extends PriceModifier{
88  }
89
90  sig batteryLevelDiscount extends Discount{
91  }{
92    percentage = 20
93  }
94
95  sig carSharingDiscount extends Discount{
96  }{
97    percentage = 10
98  }
99
100 sig plugDiscount extends Discount{
101 }{
102   percentage = 30
103 }
104
```

```
sig noPlugsExtraCharge extends ExtraCharge{
}{
  percentage = 30
}

sig lowBatteryExtraCharge extends ExtraCharge{
}{
  percentage = 30
}

sig Car{
  plate : one Strings,
  occupiedSeats : one Int,
  engineStarted : one Bool,
  plugged : one Bool,
  position : one GPSPosition,
  battery : one BatteryLevel,
  status : one CarStatus,
}

sig BatteryLevel{
  percentage : one Int,
}
{
  percentage > 0
  percentage <= 100
}

enum CarStatus{
  AVAILABLE,
  BOOKED,
  RENT
}

sig Area{
  borders : set GPSPosition,
}
{
  #borders >= 3
}

sig SafeArea extends Area{
}

sig RechargeSite extends SafeArea{
}
```

5.2 Facts

```
//FACTS

//Defines the != for GPSPosition
fact DifferentGPS{
```

```
5     all g1, g2 : GPSPosition | g1 != g2 implies
6     (g1.latitude != g2.latitude || g1.longitude != g2.
        longitude)
7   }
8
9   //Defines the != for Car
10  fact DifferentCar{
11    all c1,c2 : Car | c1 != c2 implies c1.plate != c2.plate
12  }
13
14  //Defines the rules to associate a billing to a service
15  fact ServicesBillingLogic{
16    all s : Booking | s.status = CLOSED
17      implies ((one b : Billing | b.service = s && (no r :Ride
          | s.ride = r))
18        || (no b:Billing | b.service = s && (one r :Ride | s.
        ride = r)))
19    all s : Ride | s.status = CLOSED
20      implies (one b : Billing | b.service = s)
21  }
22
23  //No more than one billing per service
24  fact OneBillPerS{
25    all b : Billing | (lone s : Service | b.service = s)
26  }
27
28  //Defines the != for User
29  fact DifferentUser{
30    all u1,u2 : User | u1 != u2 implies u1.id != u2.id
31  }
32
33  //The car of an active booking must be booked
34  fact IfCarInActiveBookingThenBooked{
35    all b : Booking | b.status = ACTIVE
36      implies b.car.status = BOOKED
37  }
38
39  //Removes the possibility to have a Cancelled status for a
        ride
40  fact RideCannotBeCancelled{
41    all r : Ride | r.status != CANCELLED
42  }
43
44  //Different payment details are associated to different
        clients
45  fact DifferentPayment{
46    all c1,c2 : Client | c1 != c2
47      iff c1.paymentDetails != c2.paymentDetails
48  }
49
50  //Defines the != for PaymentDetails
51  fact DIfferentPaymentDetails{
52    all p1,p2 : PaymentDetails | p1 != p2
53      implies p1.cardNumber != p2.cardNumber
```

```
}

//No modifiers existing without a ride
fact OneRidePerDiscount{
  all pm:PriceModifier | ( some r: Ride | pm in r.modifiers)

}

//Each client has a GPSPosition
fact OneGPSPerClient{
  all c: Client | (one g: GPSPosition  | c.lastKnownPosition
    =g)
}

//Defines the 1:1 association between clients and driving
    licenses
fact DrivingLicenseClient{
   drivingLicense =~ client
}

//Establishes that all discounts are applied at the end of
    the ride
fact NoDiscountsOrExtraChargeDuringRide{
  all r : Ride | r.status = ACTIVE
    implies (no p : PriceModifier | p in r.modifiers)
}

//The only bookings associated with billings are the one
    without a ride
fact payOnlyWastedBooking{
  all b : Booking | (b.status = CLOSED
    && (one bill : Billing | bill.service = b))
      implies (no r : Ride | b.ride = r)
  all b : Booking | ((b.status = CLOSED
    &&  (no r : Ride | b.ride = r))
      implies (one bill : Billing | bill.service = b))
}

//Defines the 1:1 association between clients and payment
    details
fact PaymentClient{
  paymentDetails =~ client
}

//Active services can not share neither the client nor the
    car
fact ServicePerCarAndClient{
  all s1,s2 : Service |
    (s1 != s2 && s1.status = ACTIVE && s2.status = ACTIVE)
      implies (s1.client != s2.client && s1.car != s2.car)
}

//Each service is associated to a client
fact OneClientPerService{
```

```
102   all s: Service | (one c: Client  | s.client=c)
103 }
104
105 //Each billing is associated to one service
106 fact OneServicePerBilling{
107   all b: Billing | (one s: Service  | b.service=s)
108 }
109
110 //Established the possible relations between booking and
      ride
111 fact BookingRide{
112   all r : Ride | one b : Booking | r.booking = b
113   some b : Booking | one r : Ride | b.ride = r
114   all r : Ride | r.booking.ride = r
115   some b : Booking | no r : Ride | b.ride = r
116 }
117
118 //The billing is applied only on closed services
119 fact billingOnlyForClosedServices{
120   all b : Billing | b.service.status = CLOSED
121 }
122
123 //If the billing is associated to a booking, than the
      booking has no ride
124 fact billingOnlyBookingsClosedWithNoRide{
125   all b : Booking |
126     (b.status = CLOSED  && (no r : Ride | b.ride = r))
127       iff ( one bill : Billing | bill.service = b )
128 }
129
130 //The ride of a booking share the same client and car
131 fact BookandRideConsistency{
132   all r:Ride | (r.booking.car=r.car && r.booking.client=r.
      client)
133 }
134
135 //The booking of a ride must always be closed
136 fact RideMeansBookingClosed{
137   all r : Ride | r.booking.status = CLOSED
138 }
139
140 //No double bookings for the same ride
141 fact bookingRideAssociation{
142   all b : Booking | #b.ride = 1 implies b.ride.booking = b
143   all r : Ride | r.booking.ride = r
144 }
145
146 //The car status of a car of an active riding is RENT
147 fact RidingCarMeansCarInRent{
148   all r : Ride | r.status = ACTIVE
149     implies r.car.status = RENT
150 }
151
152 //Defines the != for Service
```

```
153 fact DifferentService{
154   all s1,s2 : Service | s1 != s2
155     implies (s1.serviceId != s2.serviceId)
156 }
157
158 //For every car in RENT exists a ride
159 fact ifCarInRentThenExistsRide{
160   all c : Car | c.status = RENT
161     implies (one r : Ride | (r.car = c && r.status = ACTIVE)
    )
162 }
163
164 //For every car BOOKED exists a booking
165 fact ifCarBookedThenExistsBooking{
166   all c : Car | c.status = BOOKED
167     implies (one b : Booking | (b.car = c && b.status =
    ACTIVE))
168 }
169
170 //Is not possible to take 2 times the same modifier
171 fact NonRepeatablePriceMod{
172   all r : Ride,b,c : batteryLevelDiscount | (b in r.
    modifiers) implies
173    !(c in r.modifiers && c != b)
174
175   all r : Ride,b,c : carSharingDiscount | (b in r.modifiers)
      implies
176    !(c in r.modifiers && c != b)
177
178   all r : Ride,b,c : plugDiscount | (b in r.modifiers)
     implies
179    !(c in r.modifiers && c != b)
180
181   all r : Ride,b,c : noPlugsExtraCharge | (b in r.modifiers)
      implies
182    !(c in r.modifiers && c != b)
183
184   all r : Ride,b,c : lowBatteryExtraCharge | (b in r.
    modifiers) implies
185    !(c in r.modifiers && c != b)
186
187 }
188
189 //No price modifiers without a ride
190 fact OneRidePerPriceModifier{
191   all pm: PriceModifier | (one r: Ride  | pm in r.modifiers)
192 }
193
194 //No battery level without a car
195 fact OneCarPerBatteryLevel {
196   all b:BatteryLevel | (some c : Car | c.battery = b)
197 }
198
199 //If a car is available the no services can be active on it
```

```
fact AvailableCarHasNoActiveService{
  all c : Car | c.status=AVAILABLE
    implies !(some s: Service | s.status=ACTIVE && s.car=c)
}

//Every car in ride has its engine started
fact engineStartedDuringRide{
  all c : Car | c.engineStarted = True iff one r : Ride | r.
    car = c
}
```

5.3 Assertions

```
//ASSERTIONS
assert NoFreeCarDuringBookingOrRide{
  all c : Car | c.status = AVAILABLE
    implies (all s : Service | s.car = c
      implies (s.status = CLOSED || s.status = CANCELLED))
}
check NoFreeCarDuringBookingOrRide for 5

assert NoClientInTwoActiveServices{
  all s1 : Service | s1.status = ACTIVE
    implies (all s2 : Service |(s1 != s2 && s1.client = s2.
    client)
      implies (s2.status = CLOSED || s2.status = CANCELLED))
}
check NoClientInTwoActiveServices for 5

assert NoCarInTwoActiveServices{
  all s1 : Service | s1.status = ACTIVE
    implies (all s2 : Service |(s1 != s2 && s1.car = s2.car)
      implies (s2.status = CLOSED || s2.status = CANCELLED))
}
check NoClientInTwoActiveServices for 5

assert allBookedCarsOneBooking{
  all c: Car | c.status = BOOKED
    implies (one b : Booking | b.car = c && b.status =
    ACTIVE)
}
check allBookedCarsOneBooking for 5

assert allRentedCarsOneRent{
  all c: Car | c.status = RENT
    implies (one b : Ride | b.car = c && b.status = ACTIVE)
}
check allRentedCarsOneRent for 5

assert everyBookingHasAride{
   all r : Ride | one b : Booking | (b.ride = r && r.booking
    = b)
}
```

```
check everyBookingHasAride for 5

assert noUselessBilling{
  all b : Billing | one s : Service | b.service = s && s.
    status = CLOSED
}
check noUselessBilling for 5
```

5.4 Predicates

```
//PREDICATES
pred show{
  #Booking > 1
  #Car > 3
  #Ride > 1
  #GPSPosition > 3
  #BatteryLevel > 1
  #Client > 1
  some b : Booking | b.status = CLOSED && #b.ride = 0
}
run show for 5

pred twoBillings{
  #Billing = 2
  some b : Booking | b.status = ACTIVE && #b.ride = 0
  some b : Booking | b.status = CANCELLED && #b.ride = 0
}
run twoBillings for 5


pred oneCar{
   #Car = 1
}
run oneCar for 5

pred oneCartwoClients{
   #Car = 1
   #Client = 2
}
run oneCartwoClients for 5
```

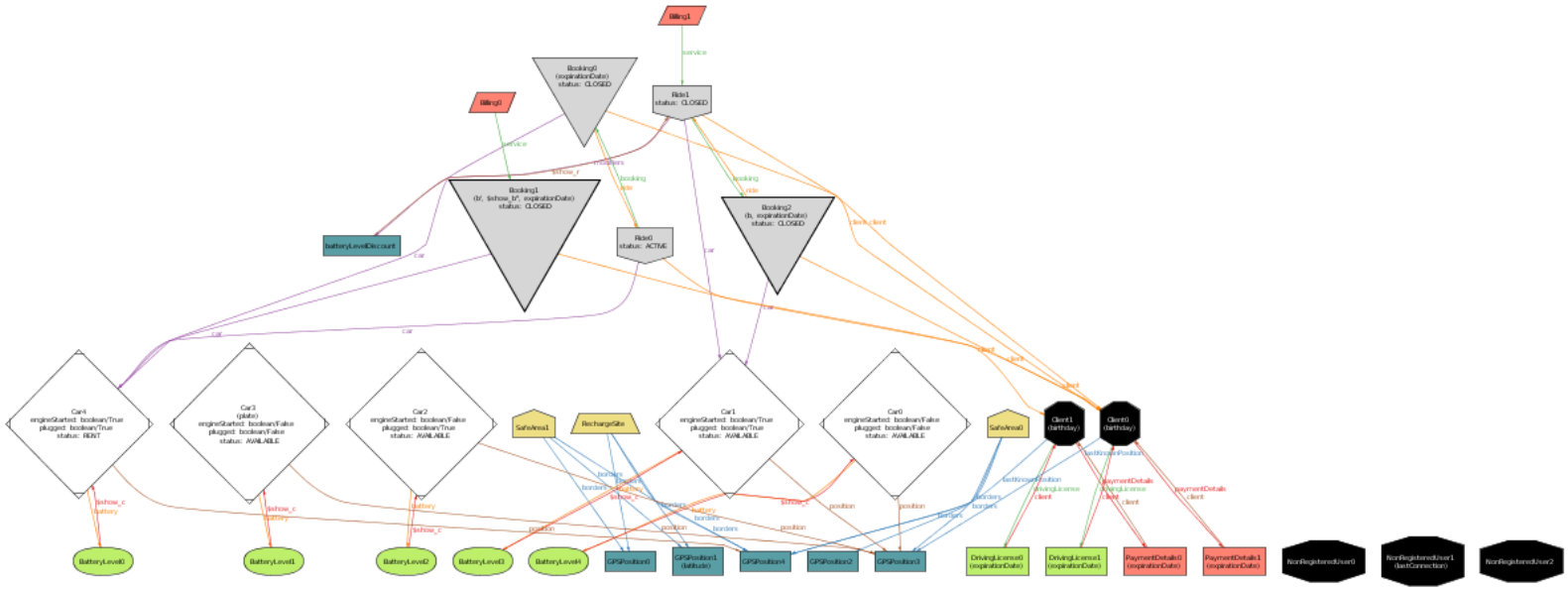

Figure 18: Execution of predicate Show

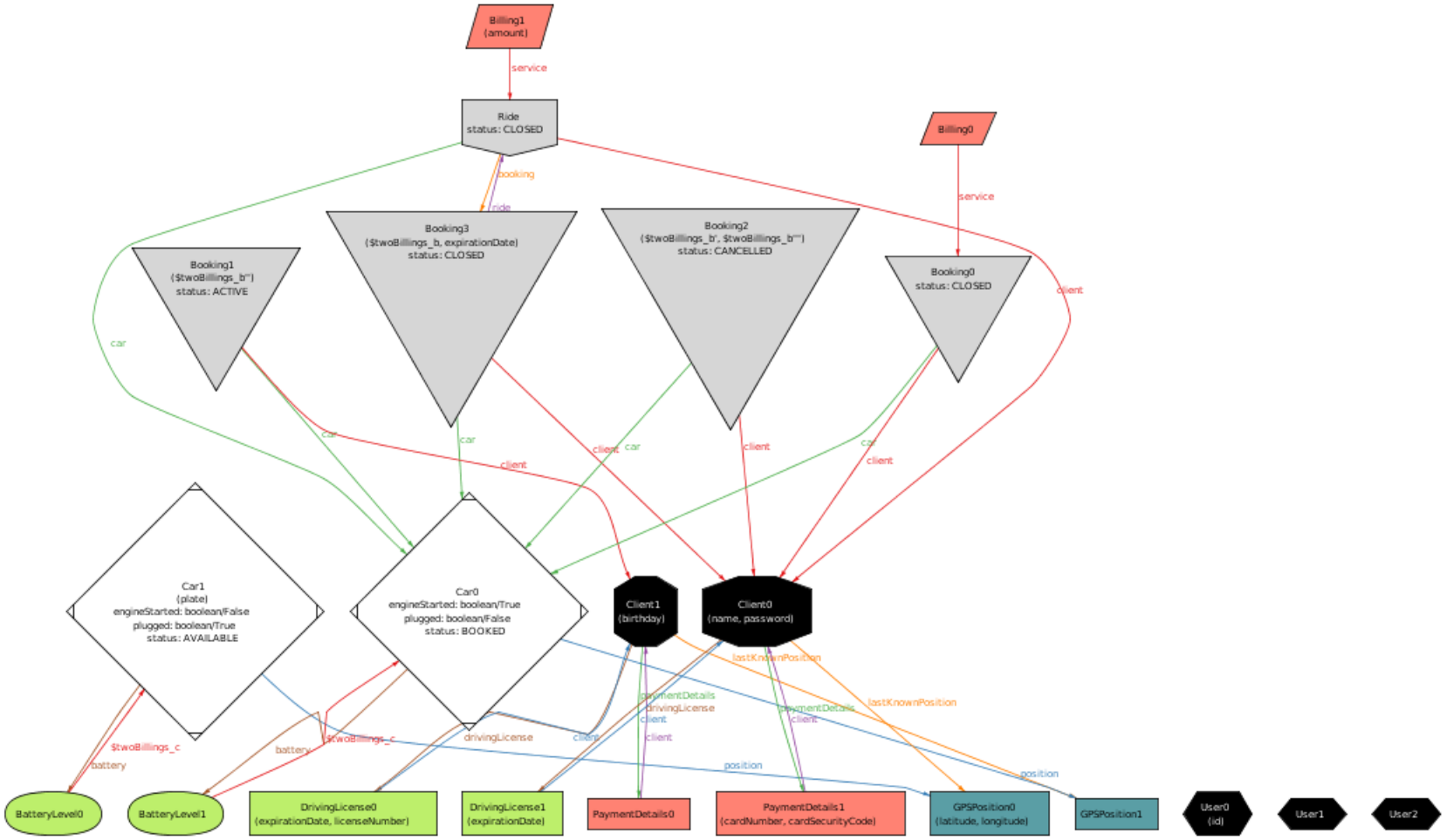

Figure 19: Execution of predicate twoBillings

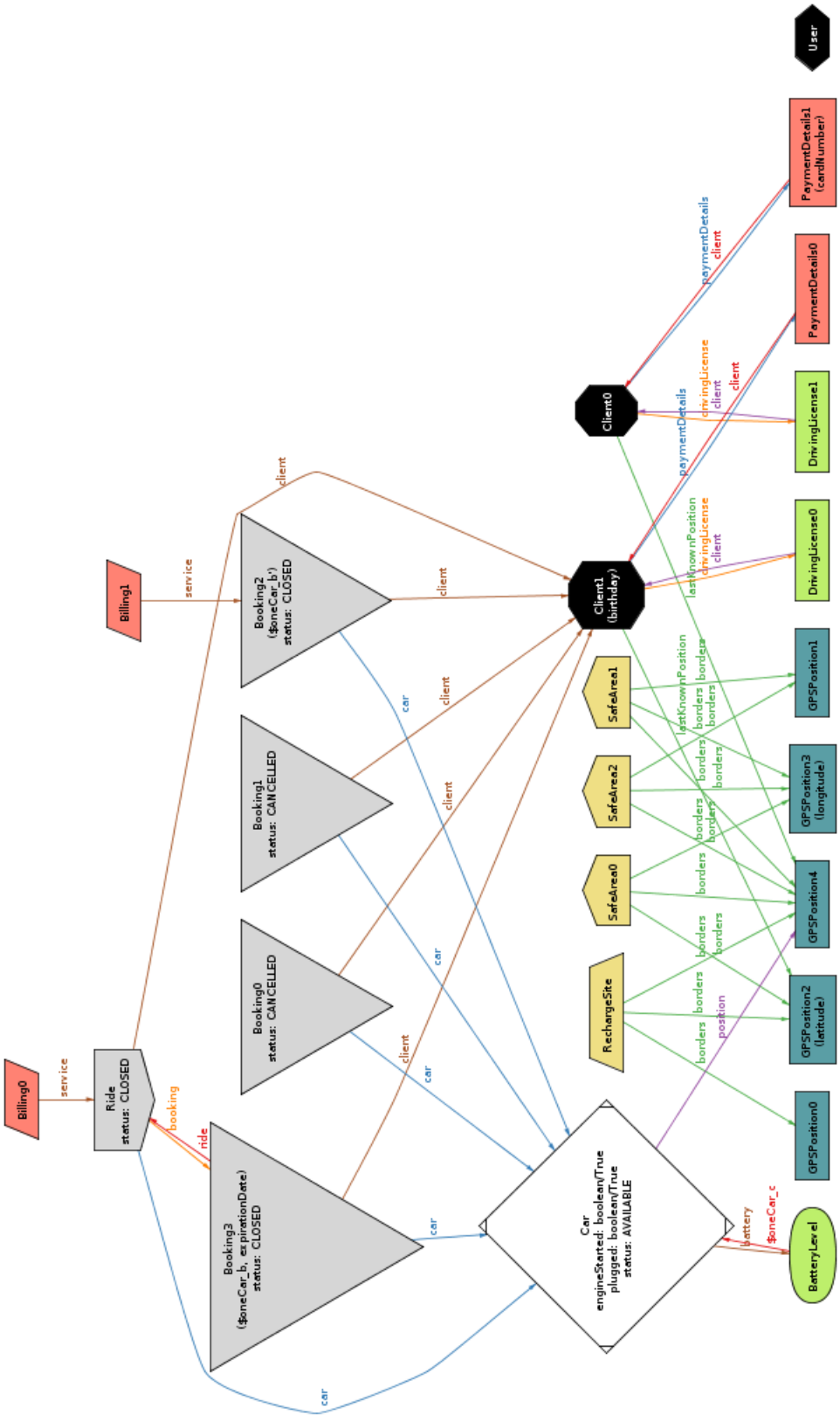

Figure 20: Execution of predicate oneCar

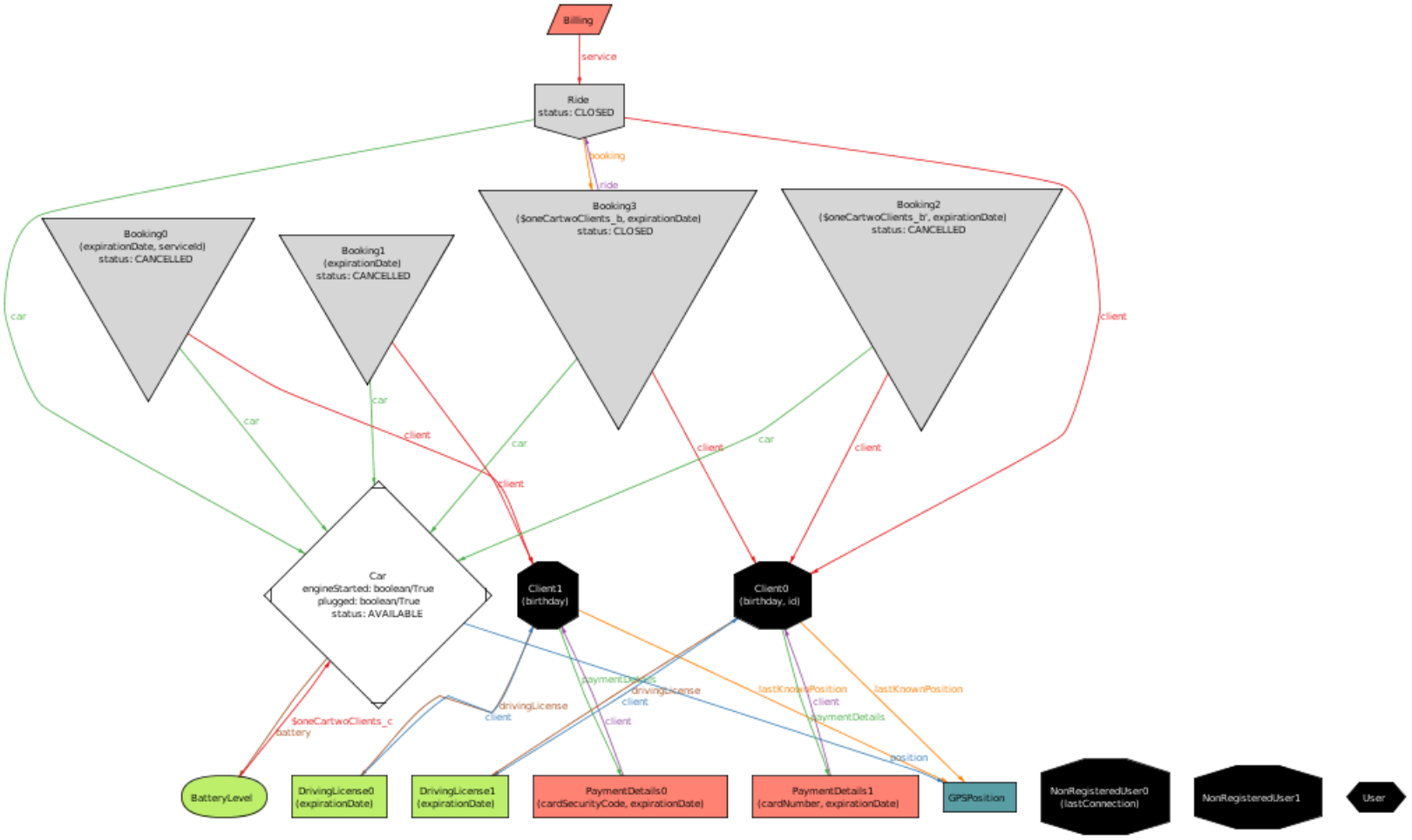

Figure 21: Execution of predicate oneCartwoClients