myTaxiService
Requirements Analysis and Specifications Document

Jacopo Strada Luca Riva

November 6, 2015

Version 1.2

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document is meant to describe a software solution for the *myTaxiService* problem. The document includes a description of the problem, functional and non-functional requirements and it is addressed to the IT department of the city administration.

1.2 Scope

The *myTaxiService* software focuses on helping the clients benefit from the service and ensures a fair management of taxi queues.

The software will:

- give the possibility to request a taxi either through a web application or a mobile app.
- compute the distribution of taxis in the various zones based on the GPS information it receives from each taxi.
- offer programmatic interfaces to enable the development of additional services.
- allow to book a taxi by specifying the origin and the destination of the ride.

1.2.1 Goals

G1: Allow clients to register an account in the system

G2: Allow taxi drivers to register an account in the system

G3: Allow clients to log in the mobile application or in the website

G4: Allow taxi drivers to log in the mobile application

G5: Allow clients to request a taxi

G6: Inform the passenger about the code of the incoming taxi

G7: Inform the passenger about the waiting time

G8: Inform the passenger about a possible call rejection

G9: Retrieve the position of each taxi from their mobile application using GPS

G10: Allow taxi drivers to set their state as *available* or *unavailable*

G11: Notify the taxi drivers about a client's request

G12: Allow taxi drivers to accept or decline a certain client's request

G13: Guarantee a fair management of taxi queues

G14: Allow a user to reserve a taxi by specifying the origin and the destination of the ride at least two hours before

G15: Allow a driver to accept a client reservation

G16: Confirm the reservation to the user

G17: Supply APIs for further implementation of additional services in the future

1.3 Glossary

1.3.1 Definitions

Client / Passenger / User : Is a person who signed up for this service and their interest is to call a taxi or reserve a ride.

Taxi Driver : Is a person who drives a taxi and would like to be called or reserved for a ride through this service.

1.3.2 Acronyms

API: Application Programming Interface

BPMN: Business Process Model and Notation

CPU: Central Processing Unit

GPS: Global Positioning System

HTTPS: Hyper Text Transfer Protocol over Secure Socket Layer

IEEE: Institute of Electrical and Electronics Engineers

IT: Information Technology

RAM: Random Access Memory

SSD: Solid State Drive

UML: Unified Modeling Language

1.3.3 Abbreviations

A*n*: Assumption number n

D*n*: Domain Property number n

G*n*: Goal number n

R*n*: Requirement number n

1.4 References

- ”myTaxiDriver” Specification Document
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications

1.5 Document Overview

- In section 2 many factors influencing the system and its requirements are explained without emphasizing specific requirements, but focusing on what lays behind the process of defining these requirements and making them easily understandable.
- In section 3 all the requirements mentioned before are carefully described: inputs and outputs of the system are accounted as well as the processes performed by the system from the reception of a given input to the resulting output. In order to make the requirements and the processes as clear as possible this section includes many models that may help the reader figuring out every important detail.
- In the appendix of the document it is possible to find an Alloy model of the system's structure with various demonstrations of how the specified requirements guarantee the desired behaviours and properties of the software.

2 Overall Description

2.1 Product Perspective

The software will consist in three different user interfaces: two different apps for the clients and the taxi drivers and one web application.

In order to make this working there will be a web server containing the application logic and a DBMS that stores all the data (such as users' credentials, taxi's positions, taxi's availability . . .). Mobile applications, web application and the server communicate through the Internet.

The system will also provide programmatic interfaces for other new services. At the moment the city has not any similar system, so the application will be completely independent.

2.1.1 User interfaces

- Taxi Driver
  - Smartphone App on
    - Android
    - iOS
    - Windows Phone
  - Web Browser (any operating system)
- Clients
  - Smartphone App on
    - Android
    - iOS
    - Windows Phone
  - Web Browser (any operating system)

2.1.2 Hardware Interface

Client Side:

- Smartphone and Tablet
  - Android
    - CPU: DualCore 1Ghz
    - RAM: 700Mb
    - GPS
  - iOS
    - iPhone 4 or greater
    - iPad 2 or greater
  - Windows Phone
    - CPU: DualCore 1Ghz
    - RAM: 512Mb
    - GPS

- Web Browser (any operating system)
  - CPU: 1Ghz
  - RAM: 1Gb
  - GPS

Server Side:

- 4 Cores, 2.8-3.0 GHz each (2.8 GHz minimum speed)
- 4 GB RAM per core
- SSD, 100 GB

2.1.3 Software Interfaces

- Google Maps API V3
- Taxi drivers' licences database V1

2.1.4 Communications Interfaces

All the applications will use HTTPS protocol.

2.2 User characteristics

This application is conceived for every taxi driver in the city area and for adult clients (18+) with a compatible device which satisfies the specification listed in the previous section. There are no specific levels of education or expertise needed apart from a decent knowledge of English language.

2.3 Constraints

Interfaces to other applications: Google Maps will be integrated in the system in order to give clear information about a client position and allow the client to modify a wrong position calculated by their smartphone's GPS. Another important feature that this service will offer is the computation of an estimated waiting time for the taxi to reach the client.

Parallel Operations: The system is studied for a large city with an high usage of taxi service. So parallelism is very important because a lot of taxi calls may income in the same time and a lot of taxi drivers have to been indexed by the system.

Reliability requirements: In order to keep an high level of service the system must be reliable. Each call must reach a taxi driver and a response has to arrive to the user in a reasonable time (5 min).

2.4 Domain properties

D1: If a client requests a ride they will wait the taxi's arrival

D2: If a client makes a reservation for a certain time they will be punctual waiting for the ride

D3: If a driver accepts a reservation or a request they will actually try to reach the client location

2.5 Assumptions

A1: Possibility to gain access to the taxi driver's licences database of the city

A2: A client is able to make only one taxi request at once

A3: A client is not able to make two overlapped reservations

A4: If a request is rejected by all the taxi drivers in the queue the request will be discarded: it will not iterate the queue for a second time.

A5: As soon as the request is discarded the client will be notified and able to make a new call

A6: The system will not give the client any information regarding the cost of the ride

A7: If a client calls a taxi and the request is accepted the client will not be able to modify or erase the request.

In order to assure an efficient service it is necessary a slight modification to the customer request:

A8: When a customer reserves a taxi the system allocates the request to all registered driver

A9: If a driver accepts the request the system confirms the reservation to the user

A10: If no driver confirms the reservation in at most an hour since the request the system deletes the reservation and notifies the client

A11: A taxi reservation's origin must be located inside one of the zones of the city

A12: A client is able to cancel a reservation at least ten minutes before the ride.

A13: The system notifies the driver about the ride they accepted ten minutes before the hour of the appointment with the client

Assumptions A8, A9, A10, A13 were made in order to avoid the risk of having an empty queue or no driver who would accept the reservation only ten minutes before the fixed hour of the ride.

2.6 Apportioning of requirements

Reporting: A client may report the lack of service of a driver so that the city's government would be able to take action against the said driver. At the same time a driver may be able to report a client who misses fixed appointments, thus after few reclaims the client's account could be suspended or deactivated.

Payments: It would be interesting to give the client the possibility to pay the reservation service directly through the system, which could also provide an estimated price for the ride given the origin and the destination.

Taxi sharing: An user may be ready to share a taxi, so the application could mark their requests in order to allow other clients to participate in the ride if they have a similar starting point and a similar destination. Obviously the advantage wold be the possibility to split the cost of the ride, which should be calculated by the system. The system should also compute the route for the taxi driver.

3 Specific requirements

3.1 Functional Requirements

G1: **R1:** Clients must provide a valid e-mail address. It will become the identifier for the account.
R2: Clients must provide their Name, Surname, Date of Birth and a valid Mobile Phone Number
R3: Clients must provide a password with a minimum length of 8 characters
R4: User cannot sign up twice
R5: Visitor can just see log in page and registration form

G2: **R1:** Taxi drivers must provide a valid e-mail address. It will become the identifier for the account.
R2: Taxi drivers must provide their Name, Surname and a valid Mobile Phone Number.
R3: Taxi drivers must provide a password with a minimum length of 8 characters.
R4: Taxi drivers must provide a valid license number which will be verified by the system.
R5: User cannot sign up twice.
R6: Visitor can just see log in page and registration form.

G3: **R1:** Users must be already registered to success login process.
R2: Users must provide their e-mail and password used during the registration process.
R3: If credentials are wrong, the system will reject the login.
R4: The service cannot be used before the login.

G4: **R1:** Taxi drivers must be already registered to success login process.
R2: Taxi drivers must provide their e-mail and password used during the registration process.
R3: If credentials are wrong, the system will reject the login.

G5: **R1:** Users must be already logged in the application.
R2: Users must provide their position.
R3: If the position cannot be retrieved from GPS, the user will have to write the address manually.
R4: If the position retrieved from GPS is not correct or sufficiently precise, the user will have to confirm the detected position or provide the correct one.

G6: **R1:** Users must be already logged in the application.
R2: A taxi driver in the area of the client must have accepted the request

G7: **R1:** Users must be already logged in the application.
R2: A taxi driver in the area of the client must have accepted the request

G8: **R1:** Users must be already logged in the application.
R2: No taxi driver in the area has accepted the client's request

G9: **R1:** Drivers must be already logged in the application.
R2: The application installed on the taxi driver's smartphone must be running in background.
R3: Taxi's current positions must be stored in the central DB with the last update time.

G10: **R1:** Drivers must be already logged in the application.

G11: **R1:** Drivers must be already logged in the application.
R2: Drivers' state must be set as *available*.
R3: Drivers must be in the same zone as the client.
R4: The Smartphone or Tablet must ring or vibrate and display a pop-up for the notification.

G12: **R1:** Drivers must be already logged in the application.
R2: Drivers' state must be set as *available*.
R3: Drivers' state will become *unavailable*.
R4: Drivers must be in the same zone as the client.

G13: **R1:** System considers the city divided in zones
R2: For each zone exists a queue of taxi drivers
R3: System assigns a driver to a zone basing its decision on their GPS position
R4: System adds a driver to a zone's queue if they are available

G14: **R1:** System receives taxi requests
R2: System forwards taxi request to the first driver of the queue of client's zone
R3: System sends a confirmation to the client if the driver accepts
R4: System forwards the request to the second driver in the queue if the first one declines
R5: System moves the first driver who has declined to the last position in the queue

G15: **R1:** Clients must be already logged in the application.
R2: Clients must specify a valid origin (see assumption A8).
R3: Clients must make a reservation at least two hours before the ride.

G16: **R1:** System receives a taxi reservation
R2: System forwards request to all the registered drivers
R3: System waits thirty minutes
R4: System stops waiting if a driver accepts the reservation
R5: System sends request's confirmation to the client if a driver has accepted the reservation
R6: System sends a rejection notification to the client if thirty minutes have passed and no driver has accepted the reservation
R7: System notifies the driver who has accepted the reservation ten minutes before the appointment with the client

G17: **R1:** Provide taxi's positions and availability state.

3.2 Non Functional Requirements

3.2.1 Performance Requirements

As already stated before, performance is an important point for this application and so is also necessary to give an idea of the big amount of users who will be serviced by the application. From a little analysis we have derived that *myTaxiService* has to handle at least 2 Millions of users on line at the same time.

The time of response must be less or equals to 1 second for at least 90% of transactions.

3.2.2 Software System Attributes

Reliability: The system is a work's provider for Taxi drivers and so it is important not to loose any opportunity.
Keeping this in mind a system of acknowledge will be implemented and it will guarantee that all notifications will arrive to their proper destination.

Availability: This application may become a complete substitute of telephone taxi calls and so it is very important that it will always be available. To do so the system will be hosted on multiple servers in order to be resistant to some faults.]

Security: The security of the communications is guaranteed by the use of HTTPS with SSL encryption. Attacks are prevented thanks to a structure with more than one level of servers with always a firewall between them.

3.3 External Interfaces

In this section there is a set of mockups illustrating the most important screens of *myTaxyService*. These mockups are divided into two sections in order to divide the Clients Application from the Taxi Drivers Application.

What is presented is only a first idea of the applications that is useful for a better understanding of the whole system described in next sections.

3.3.1 Clients' User Interfaces

Here are shown mockups for the Client Application. In the project it is also planned a website for them which is not represented here but it will have the same features proposed for the mobile application.

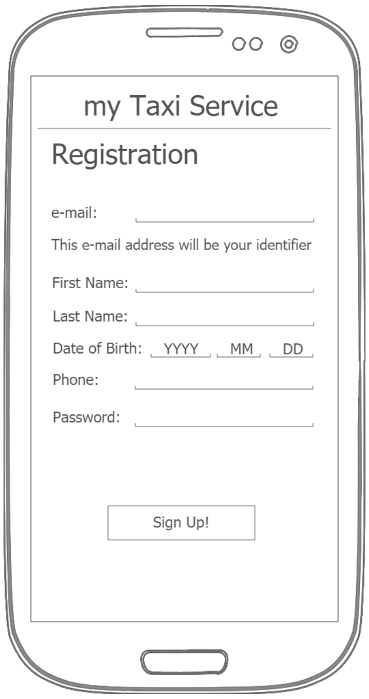


Figure 1: Clients Registration Mockup:
This is the registration page for a client. All the fields must be filled in correctly in order to register to the service.

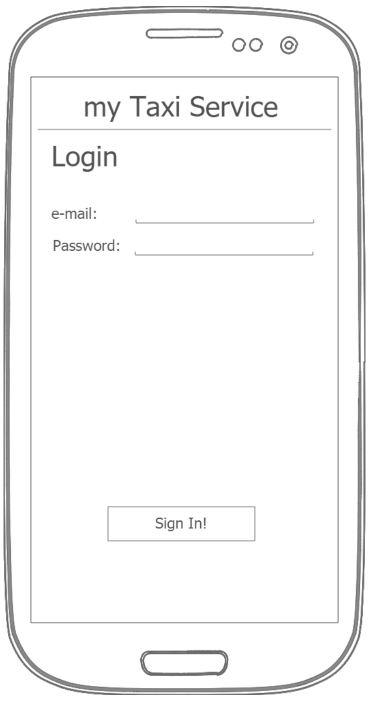


Figure 2: Clients Login Mockup:
This page is used to log in and use the application. This is necessary to use every feature of the application.

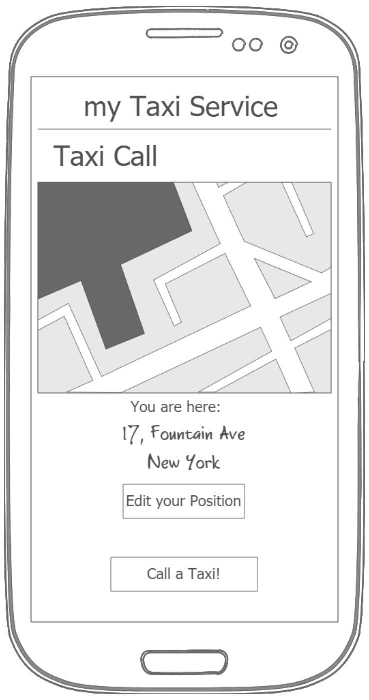


Figure 3: Clients Taxi Call Mockup:
Here is presented the paged used for calling a taxi. The position retrieved from the GPS (if available) is displayed on the map and written below. The client can use the *Edit your Position* button in order to modify the place in which the taxi will arrive; to do this is provided a page like the which one described in Figure 4.

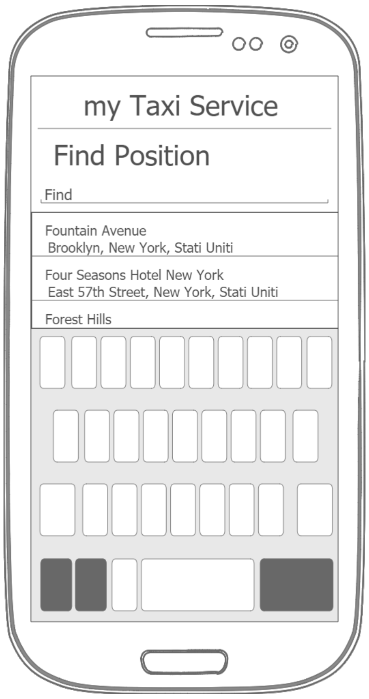
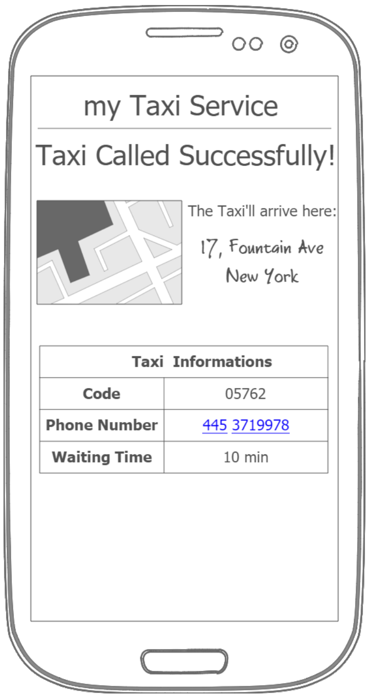

my Taxi Service
Find Position
Find
Fountain Avenue
Brooklyn, New York, Stati Uniti
Four Seasons Hotel New York
East 57th Street, New York, Stati Uniti
Forest Hills

Figure 4: Find Position Mockup:
This page is used for searching a position. It is possible to type booth street names or places names (such as Hotels, Restaurants .... This utility is called from more than one point of our application: see for example Figure 3 or Figure 6.



Taxi Informations	
Code	05762
Phone Number	445 3719978
Waiting Time	10 min

Figure 5: Clients Taxi Call Confirmation Mockup: this is what is displayed when a call made by a client was successfully accepted by a taxi driver. This confirmation let the client to know the number of the taxi which will arrive, the taxi driver's phone number and an estimated waiting time.

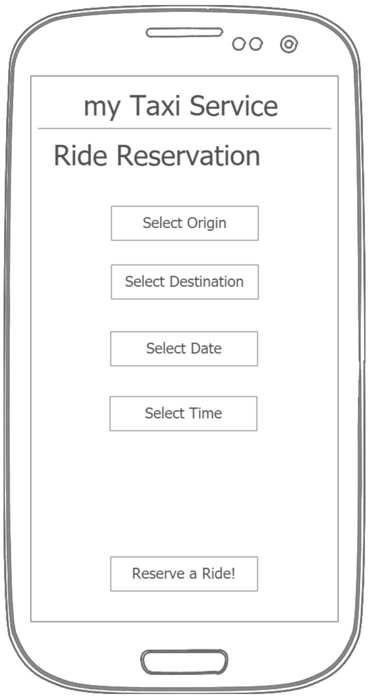


Figure 6: Clients Ride Reservation Mockup:
This page is used to reserve a taxi ride from an origin to a destination specifying a Date and a Time. The two buttons *Select Origin* and *Select Destination* open the utility shown at Figure 4; the other two open respectively a simple date picker and a simple time picker.

3.3.2 Taxi Drivers' User Interfaces

The following mockups represent the key features of Taxi Drivers' Application.

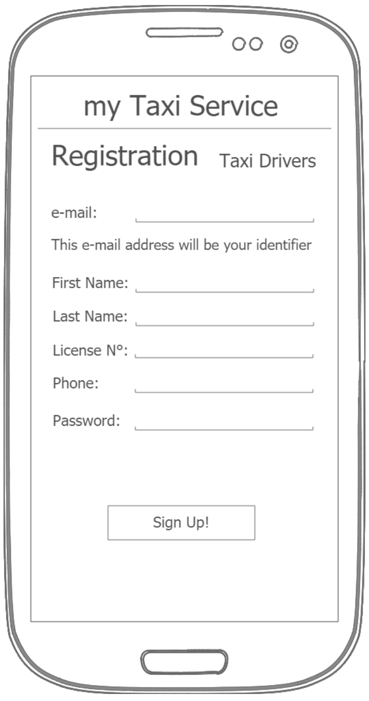


Figure 7: Taxi Drivers Registration Mockup:
This is the registration page for a taxi driver. All the fields must be filled in correctly in order to register to the service.

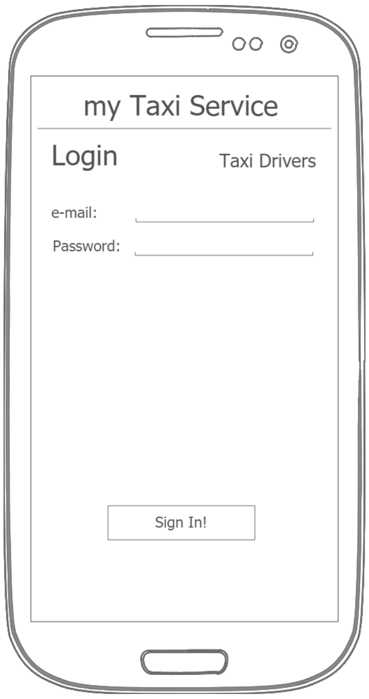


Figure 8: Taxi Driver Login Mockup:
This page is used to log in and use the application. This is necessary to use every feature of the application.

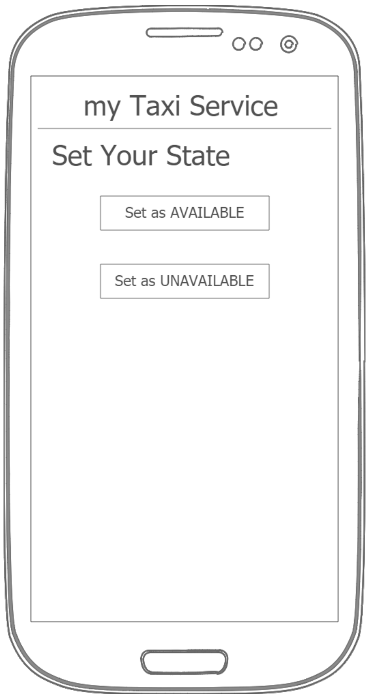


Figure 9: Taxi Drivers Change State Mockup:
This page is used by taxi drivers in order to set their availability state. To do this they have to click on *Set as AVAILABLE* or *Set as UNAVAILABLE* according to their real state. It is not necessary to use this feature when a taxi driver accepts a call; their state will be set as unavailable automatically.

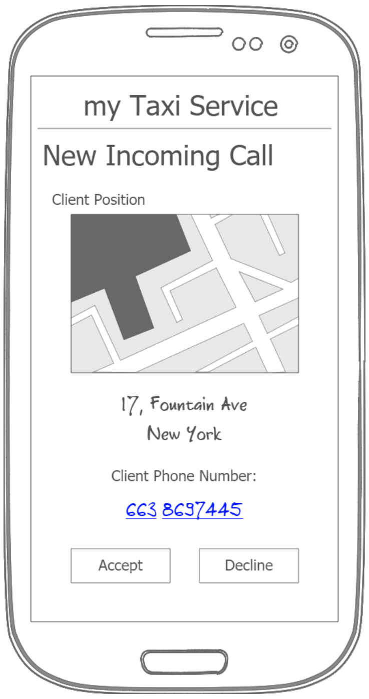


Figure 10: Taxi Drivers Incoming Call Mockup:
This page is shown when a clients makes a call and the system tries to assign that to a taxi driver. The taxi driver has to click on *Accept* or *Decline* before a timeout expires otherwise the effect is the same of *Decline* and the notification will arrive to the next taxi driver of the queue.

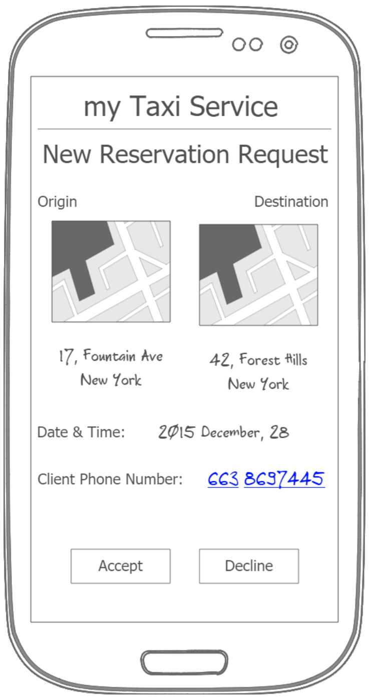


Figure 11: Taxi Drivers Reservation Request:
This page is shown when a clients makes a reservation request and the system tries to assign that to a taxi driver. The taxi driver has to click on *Accept* or *Decline* before a timeout expires otherwise the effect is the same of *Decline* and the notification will arrive to the next taxi driver of the system.

3.4 The World and The Machine

The model by M. Jackson & P. Zave is able to clearly schematize where the entities involved in the project are ideally located and which are the means of interaction between the world and the system at a first glance.

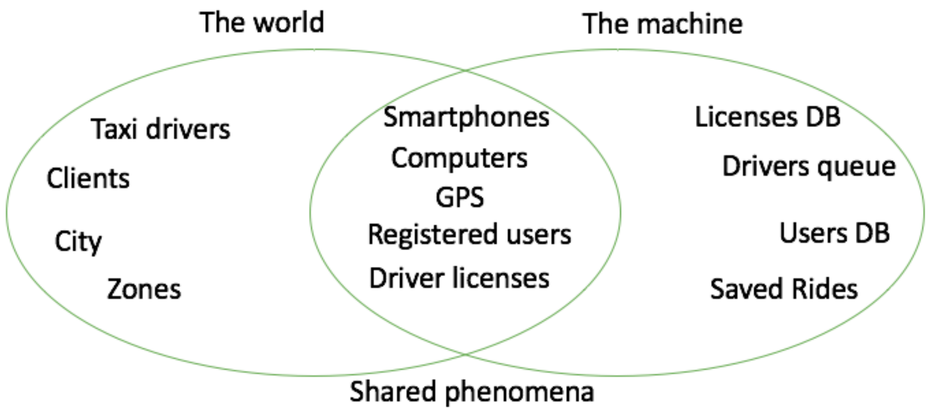


Figure 12: The World and The Machine Representation

3.5 Scenarios

S1: Goffredo is coming back from work and he notices an advertising billboard of a new app: *myTaxiService*. When he arrives home he fires up his computer and finds out that *myTaxiService* has also a website and he decides to create an account. He clicks on "Register" and fills out the necessary fields. Finally, he clicks "Confirm" and a message appears, communicating that the registration has been successfully performed.

S2: Amerigo has just arrived to the city by plane and he needs a taxi to reach his hotel. Amerigo lives in the city so he already has a registered account for *myTaxiService*. He therefore logs in the application and, after being located by the app, he confirms his position, calculated by the system, and requests a taxi. After a minute Amerigo's phone rings, in fact he has received a notification by "myTaxyService" app. Opening the app he finds the request's confirmation and the expected waiting time

S3: Galla Placidia has been trying the new app *myTaxiService* for a week and, as a taxi driver, she is very happy with it. It's been few minutes since her last ride, so she turns on her phone and opens the app to change her state to "Available" clicking on the proper button: now she is going to be notified if someone in her area asks for a ride.

S4: Arcimboldo is waiting at the taxi parking of the train station for a new request from "myTaxyService". When he hears the notification he opens the app which shows the client's address and position on the map. Arcimboldo quickly accepts the new request, starts the car and leaves the parking lot in order to reach his new client.

S5: Gianfilippa has been invited to an high school reunion out of town, but she does not have a car and the place is not reachable by the usual public means of transport. Luckily for her a few weeks ago she joined, mostly out of curiosity, *myTaxiService* and thus she has the possibility to book a custom taxi ride. The app asks her to choose an origin and then the destination. After choosing her home and the restaurant, Gianfilippa selects the hour and the date of the reunion and then forwards the reservation. Finally the application reminds her that she will shortly receive a confirmation.

3.6 UML and BPMN Models

3.6.1 Use Case Models

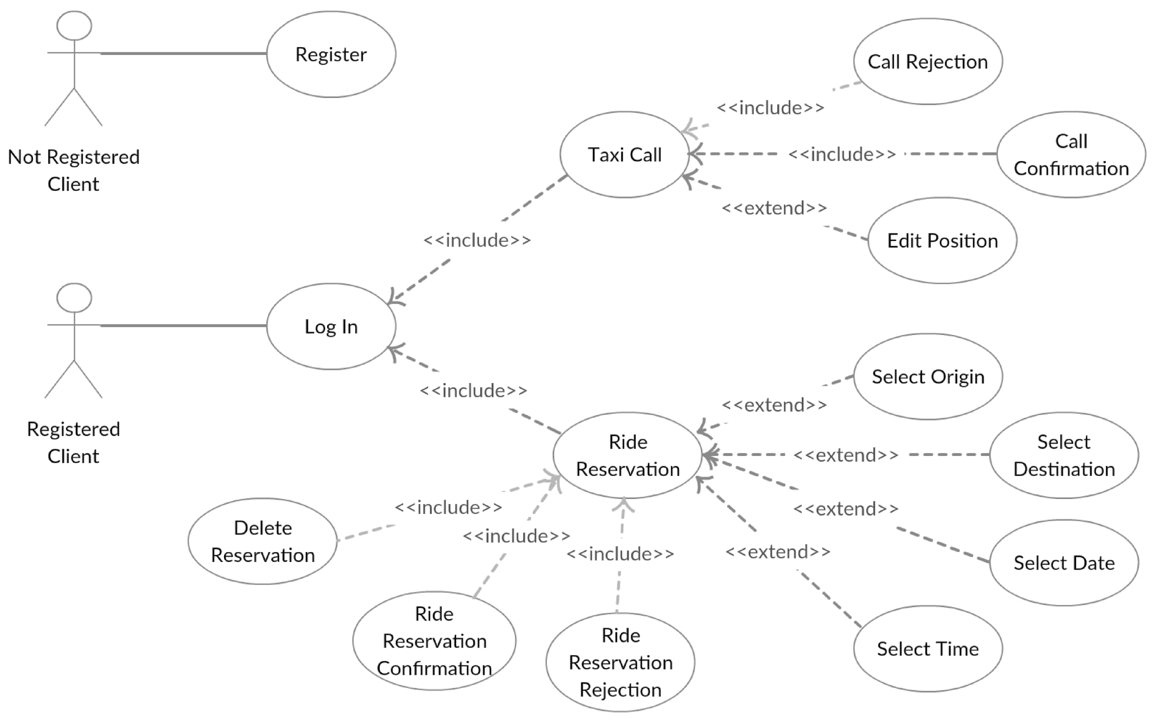


Figure 13: Clients UML Use Case Diagrams

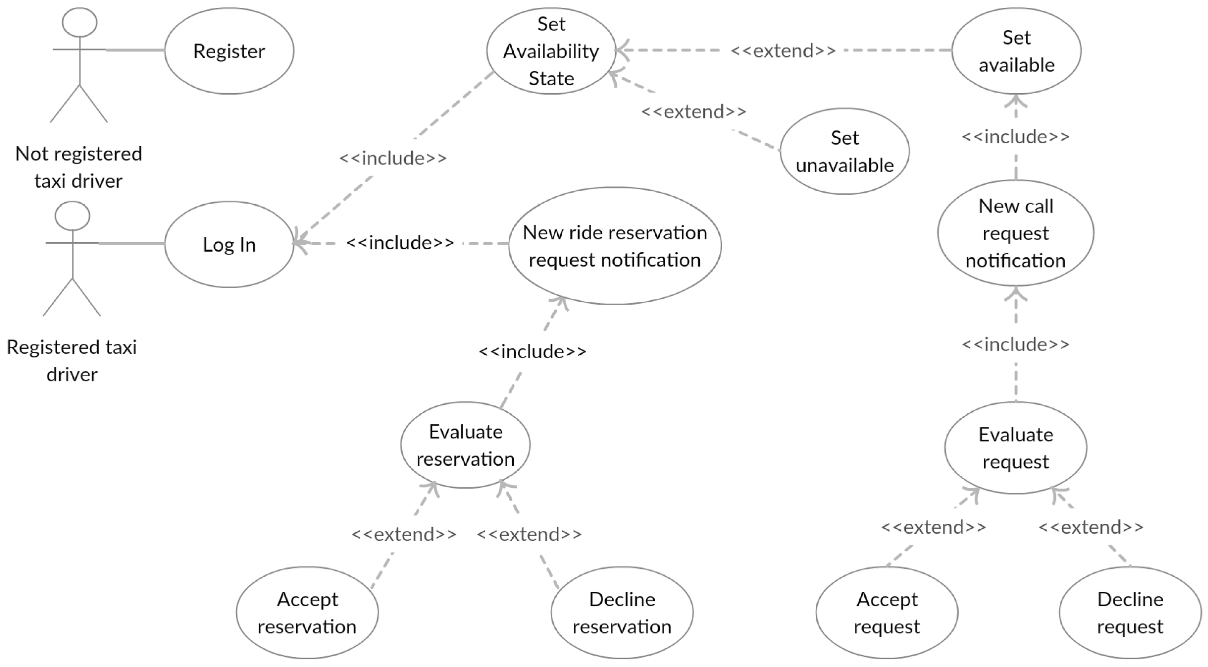


Figure 14: Taxi Drivers UML Use Case Diagrams

3.6.2 Use Case Descriptions

Client Registration	
Actors	Not Registered Client
Entry Conditions	The Client is not registered to the service.
Flow of Events	1. The client browses to the website or opens the mobile app 2. The client clicks on the *REGISTER* button 3. The client inputs the following data: (a) e-mail address (b) First Name (c) Last Name (d) Date of birth (e) Phone Number (f) Password 4. The client clicks on the *Sign Up!* button
Exit Conditions	Client is successfully registered and the login page will be shown. Now they can log in and using e-mail and password inserted before.
Exceptions	An error will be shown if: • some field is empty • e-mail already in use • password is not long enough • date of birth identifies an underage person The application will go back to the *point 3* of the *Flow of Events*

Table 1: Use Case Description of Client Registration

Taxi Driver Registration	
Actors	Not Registered Taxi Driver
Entry Conditions	The Taxi Driver is not registered to the service.
Flow of Events	1. The driver open the mobile app 2. The driver clicks on the *REGISTER* button 3. The driver inputs the following data: (a) e-mail address (b) First Name (c) Last Name (d) Phone Number (e) License Number (f) Password 4. The driver clicks on the *Sign Up!* button
Exit Conditions	Taxi Driver is successfully registered and the login page will be shown. Now they can log in using e-mail and password inserted before.
Exceptions	An error will be shown if: • e-mail already in use • password is not long enough • invalid license number The application will go back to the *point 3* of the *Flow of Events*

Table 2: Use Case Description of Taxi Driver Registration

Client Log in	
Actors	Registered Client
Entry Conditions	The Client must be registered to the service.
Flow of Events	1. The client browses to the website or open the mobile app 2. The client clicks on the *LOGIN* button 3. The client inputs the following data: (a) e-mail address (b) Password 4. The client clicks on the *Sign In!* button
Exit Conditions	Client successfully logged in. Now they can use the service.
Exceptions	An error will be shown if: • e-mail is wrong • password is wrong The client can try to log in again. The client can try to log in again.

Table 3: Use Case Description of Client Log In

Taxi Driver Log in	
Actors	Registered Taxi Driver
Entry Conditions	The Taxi Driver must be registered to the service.
Flow of Events	1. The driver opens the mobile app 2. The driver clicks on the *LOGIN* button 3. The driver inputs the following data: (a) e-mail address (b) Password 4. The driver clicks on the *Sign In!* button
Exit Conditions	Driver successfully logged in. Now they can use the service.
Exceptions	An error will be shown if: • e-mail is wrong • password is wrong The driver can try to log in again.

Table 4: Use Case Description of Taxi Driver Log In

Client makes a Taxi Call	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client clicks on the *Taxi Call* button in the menu 2. The app (or website) shows the current position of the user retrieved by the GPS 3. The client can modify the position 4. The client clicks on the *Call a Taxi!* button
Exit Conditions	The call is successfully forwarded to the system.
Exceptions	There are not exceptions for this use case.

Table 5: Use Case Description of a Taxi Call

The client receives a Taxi Call Confirmation	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client receives a notification 2. The app (or website) shows the number of the taxi which will arrive, the taxi driver's phone number and the waiting time
Exit Conditions	The call will arrive in the specified waiting time.
Exceptions	There are not exceptions for this use case.

Table 6: Use Case Description of a Taxi Call Confirmation

The client receives a Taxi Call Rejection	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client receives a notification 2. The app (or website) shows that their request was rejected
Exit Conditions	The client is able to make another call.
Exceptions	There are not exceptions for this use case.

Table 7: Use Case Description of a Taxi Call Rejection

The client makes a Ride Reservation	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client clicks on the *Ride Reservation* button in the menu 2. The client has to set the following parameters (a) Origin (b) Destination (c) Date (d) Time 3. The client clicks on the *Reserve a Ride!* button
Exit Conditions	The reservation request is forwarded to the system.
Exceptions	An error will be shown if: • the origin is out of the city • the date is in the past • the selected time is not at least two hours after the current time

Table 8: Use Case Description of a Ride Reservation

The client deletes a Ride Reservation	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client clicks on the *Your Reservations* button in the menu 2. The client clicks on the reservation who wants to delete from the list 3. The application shows the reservation's details 4. The client clicks on the *Delete* button
Exit Conditions	The reservation was successfully deleted.
Exceptions	The client cannot delete the reservation if the meeting time is sooner than ten minutes.

Table 9: Use Case Description of a Client who deletes a Ride Reservation

The client receives a Ride Reservation Confirmation	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client receives a notification 2. The app (or website) shows the number of the taxi which will arrive, the taxi driver's phone number, the place and the meeting time.
Exit Conditions	The taxi will arrive at the meeting place at the specified time.
Exceptions	There are not exceptions for this use case.

Table 10: Use Case Description of a Ride Reservation Confirmation

The client receives a Ride Reservation Rejection	
Actors	Registered Client
Entry Conditions	The Client must be logged into the website or the application.
Flow of Events	1. The client receives a notification 2. The app (or website) shows that their ride request was rejected
Exit Conditions	The client is able to make another call or reservation.
Exceptions	There are not exceptions for this use case.

Table 11: Use Case Description of a Ride Reservation Rejection

Taxi Driver changes their availability state	
Actors	Registered Taxi Driver
Entry Conditions	The Taxi Driver must be logged into the application.
Flow of Events	1. The driver opens the mobile app 2. The driver clicks on the *Change State* button 3. The driver taps on the button relative to their state: • *available* • *unavailable*
Exit Conditions	The availability state will be updated and the application will come back to the home.
Exceptions	Nothing will happen if the driver tries to select the same state in which they were before.

Table 12: Use Case Description of Taxi Driver who changes their availability state

The Taxi Driver receives a new call request notification	
Actors	Registered Taxi Driver
Entry Conditions	• The Taxi Driver must be logged into the application • The application must be opened or running in background • The Taxi Driver state must be *Available*
Flow of Events	1. The driver smartphone rings or vibrates (according to user's settings) 2. The driver clicks on the notification 3. The app shows the details of the incoming call: • Client Position • Client Phone Number
Exit Conditions	The applications is showing the details and the taxi driver has to evaluate this request.
Exceptions	There are not exceptions for this use case.

Table 13: Use Case Description of Taxi Driver who receives a call notification

The Taxi Driver evaluates a call request	
Actors	Registered Taxi Driver
Entry Conditions	• The Taxi Driver must be logged into the application • The Taxi Driver must have received a request notification • The application must be opened on the call request
Flow of Events	1. The driver smartphone shows the details of the incoming call 2. The Taxi Driver clicks on *Accept* button
Exit Conditions	The Taxi Driver has to go to take their client who has received the confirmation.
Exceptions	If the taxi driver clicks on *Decline* or the timeout runs out, the request will pass to the next driver present in the queue. If the queue is ended, the client will receive a call rejection notification.

Table 14: Use Case Description of Taxi Driver who decides to accept or decline a call request

The Taxi Driver receives a new ride reservation request notification	
Actors	Registered Taxi Driver
Entry Conditions	• The Taxi Driver must be logged into the application • The application must be opened or running in background
Flow of Events	1. The driver smartphone rings or vibrates (according to user's settings) 2. The driver clicks on the notification 3. The app shows the details of the ride reservation request: • Origin • Destination • Date • Time • Client Phone Number
Exit Conditions	The applications is showing the details and the taxi driver has to evaluate this request.
Exceptions	There are not exceptions for this use case.

Table 15: Use Case Description of Taxi Driver who receives a call notification

The Taxi Driver evaluates a Ride Reservarion request	
Actors	Registered Taxi Driver
Entry Conditions	• The Taxi Driver must be logged into the application • The Taxi Driver must have received a request notification • The application must be opened on the ride reservation request
Flow of Events	1. The driver smartphone shows the details of the ride 2. The Taxi Driver clicks on *Accept* button
Exit Conditions	The Taxi Driver has to go to take their client who has received the confirmation.
Exceptions	If the taxi driver clicks on *Decline* or the timeout runs out, the request will pass to the next driver in the system If all the taxi drivers have declined the request, a rejection will be sent to the client

Table 16: Use Case Description of Taxi Driver who decides to accept or decline a Ride Reservation request

3.6.3 Use Case Sequence Diagrams

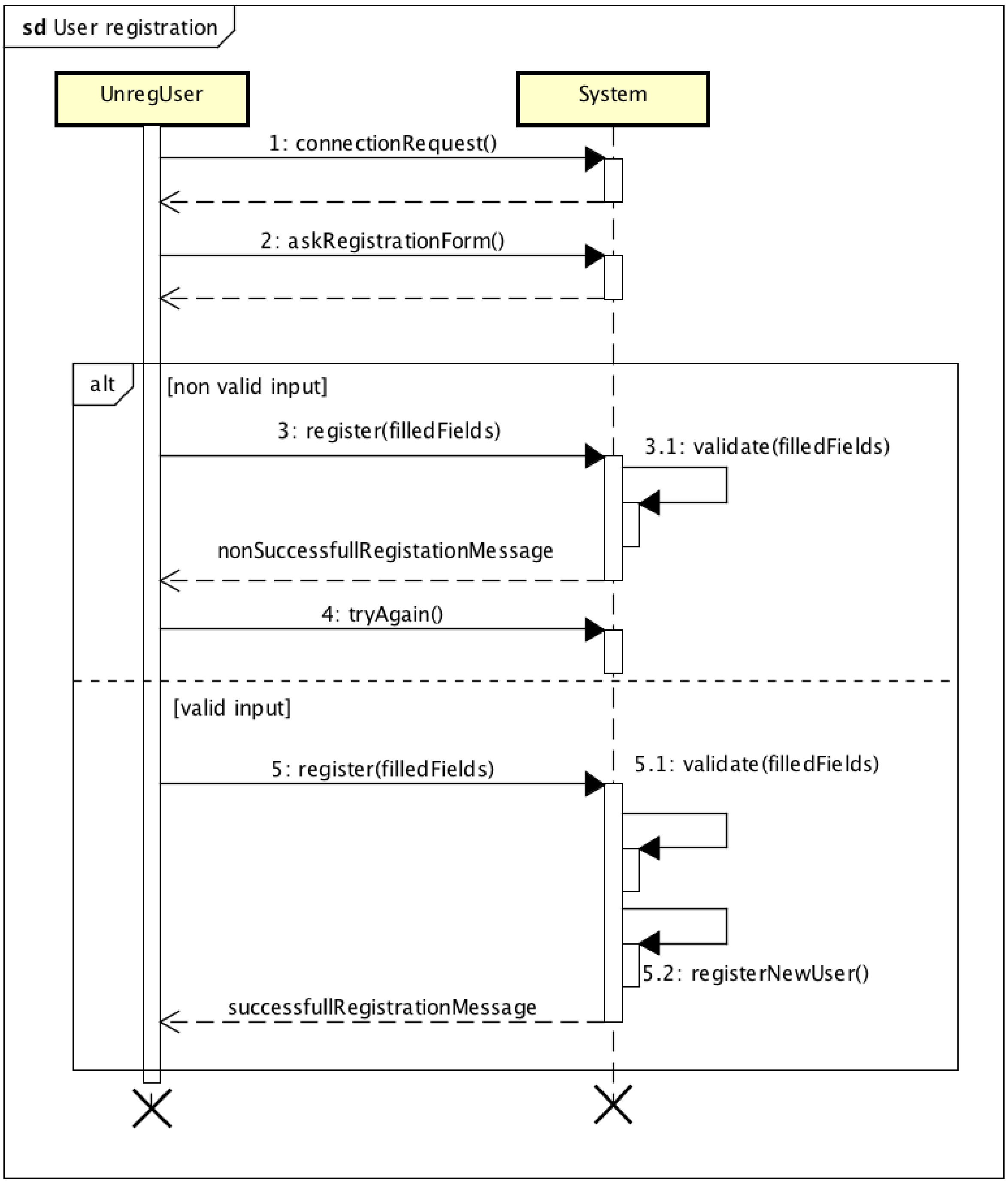

sd User registration
UnregUser
System
1: connectionRequest()
2: askRegistrationForm()
alt
[non valid input]
3: register(filledFields)
3.1: validate(filledFields)
nonSuccessfullRegistationMessage
4: tryAgain()
[valid input]
5: register(filledFields)
5.1: validate(filledFields)
5.2: registerNewUser()
successfullRegistrationMessage

Figure 15: Client Registration UML Sequence Diagram

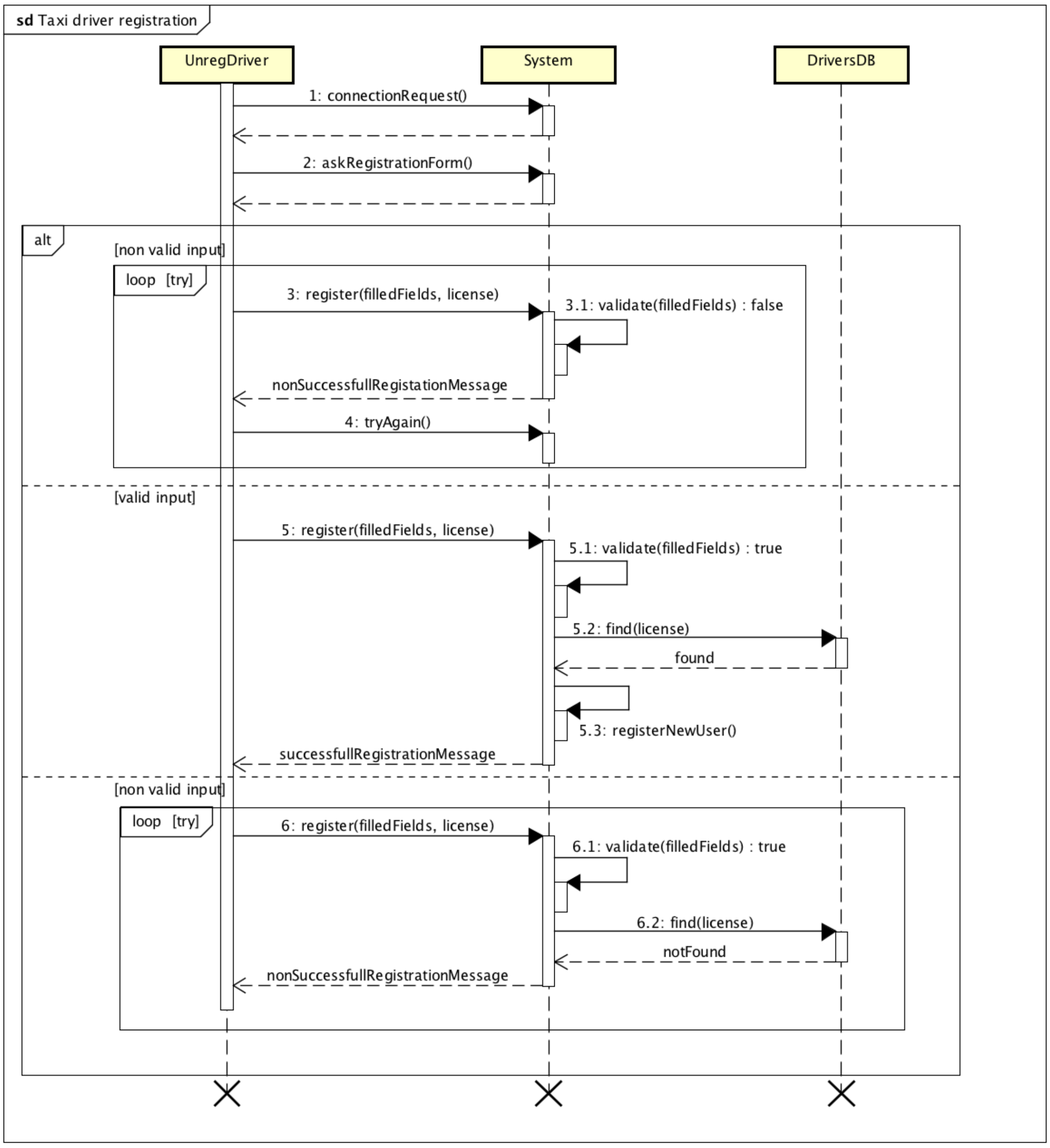

sd Taxi driver registration
UnregDriver
System
DriversDB
1: connectionRequest()
2: askRegistrationForm()
alt
[non valid input]
loop [try]
3: register(filledFields, license)
3.1: validate(filledFields) : false
nonSuccessfullRegistationMessage
4: tryAgain()
[valid input]
5: register(filledFields, license)
5.1: validate(filledFields) : true
5.2: find(license)
found
5.3: registerNewUser()
successfullRegistrationMessage
[non valid input]
loop [try]
6: register(filledFields, license)
6.1: validate(filledFields) : true
6.2: find(license)
notFound
nonSuccessfullRegistrationMessage

Figure 16: Taxi Driver UML Sequence Diagram

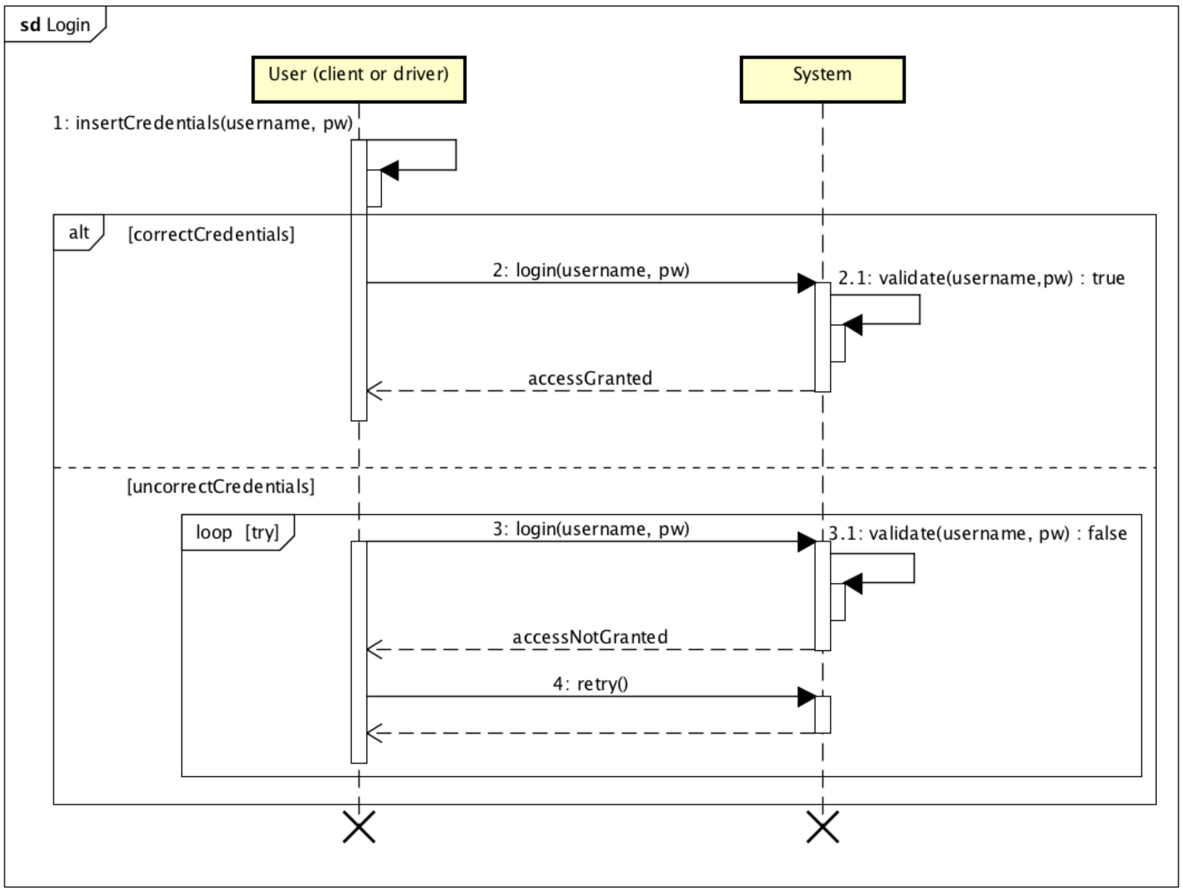


Figure 17: Login Sequence Diagram

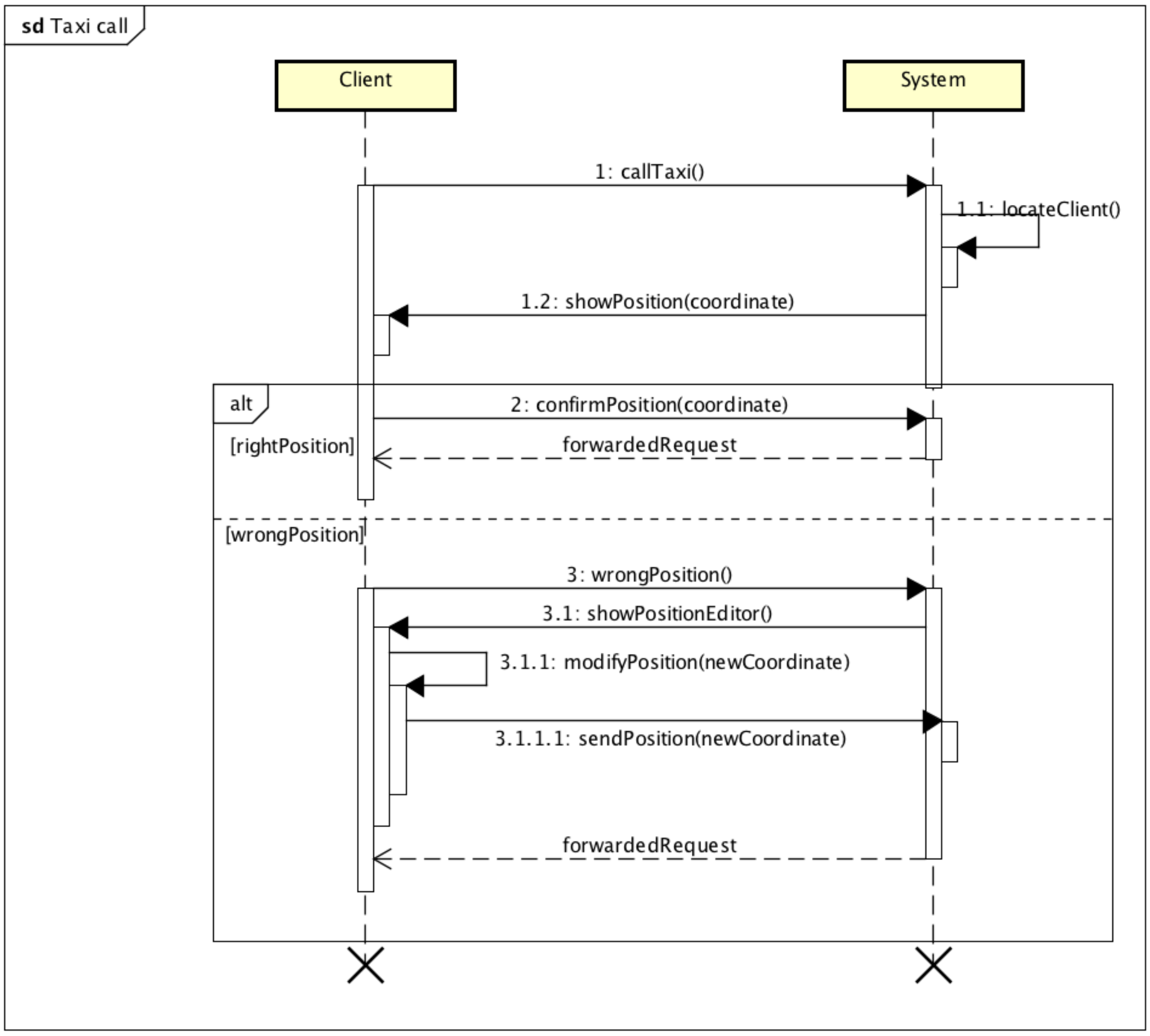

sd Taxi call
Client
System
1: callTaxi()
1.1: locateClient()
1.2: showPosition(coordinate)
alt
[rightPosition]
2: confirmPosition(coordinate)
forwardedRequest
[wrongPosition]
3: wrongPosition()
3.1: showPositionEditor()
3.1.1: modifyPosition(newCoordinate)
3.1.1.1: sendPosition(newCoordinate)
forwardedRequest

Figure 18: Client Taxi Call UML Sequence Diagram

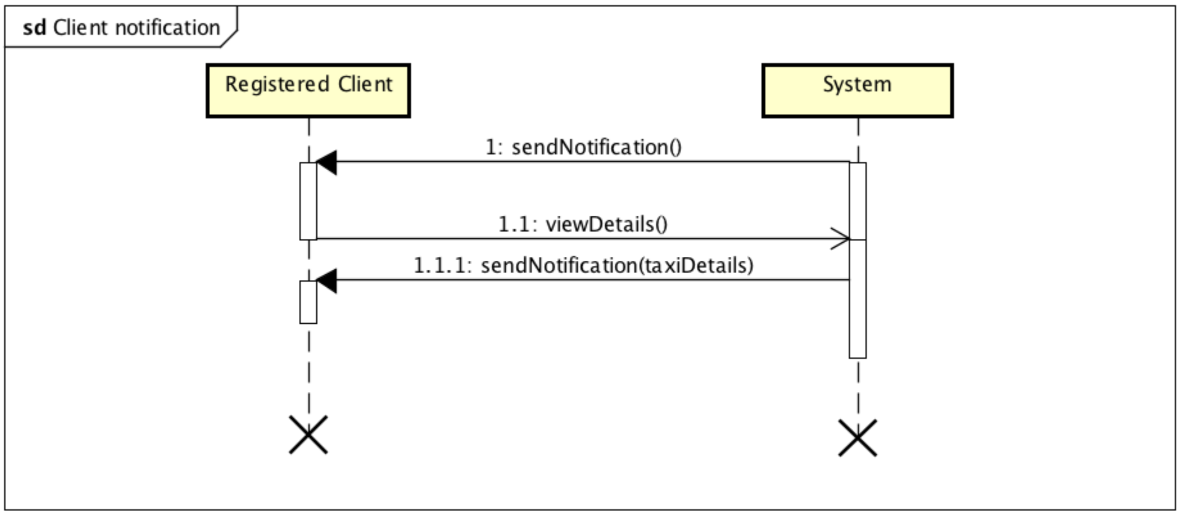


Figure 19: Client Call Confirmation UML Sequence Diagram

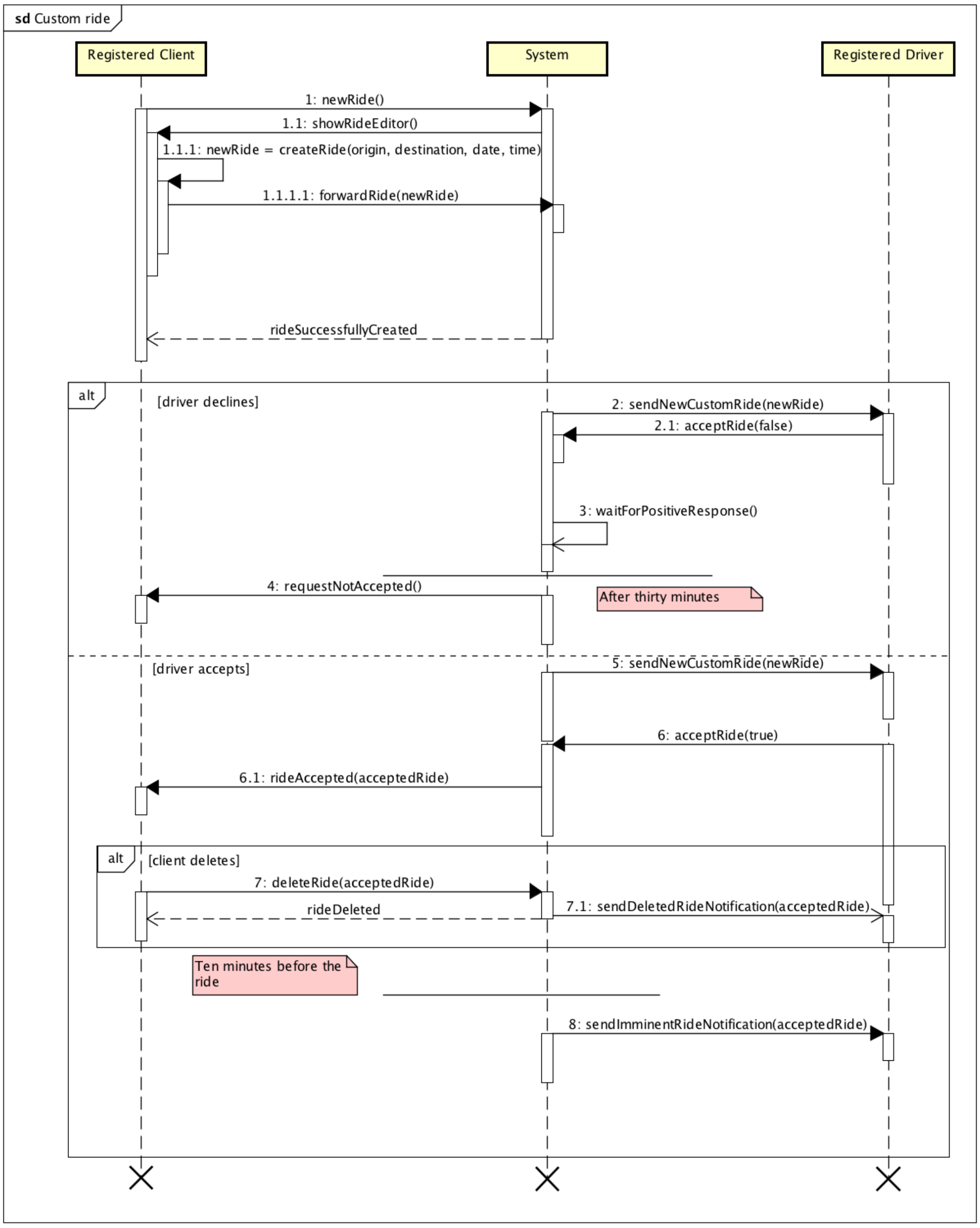

sd Custom ride
Registered Client
System
Registered Driver
1: newRide()
1.1: showRideEditor()
1.1.1: newRide = createRide(origin, destination, date, time)
1.1.1.1: forwardRide(newRide)
rideSuccessfullyCreated
alt
[driver declines]
2: sendNewCustomRide(newRide)
2.1: acceptRide(false)
3: waitForPositiveResponse()
4: requestNotAccepted()
After thirty minutes
[driver accepts]
5: sendNewCustomRide(newRide)
6: acceptRide(true)
6.1: rideAccepted(acceptedRide)
alt
[client deletes]
7: deleteRide(acceptedRide)
rideDeleted
7.1: sendDeletedRideNotification(acceptedRide)
Ten minutes before the ride
8: sendImminentRideNotification(acceptedRide)

Figure 20: Client Ride Reservation and Driver accepts/declines UML Sequence Diagram

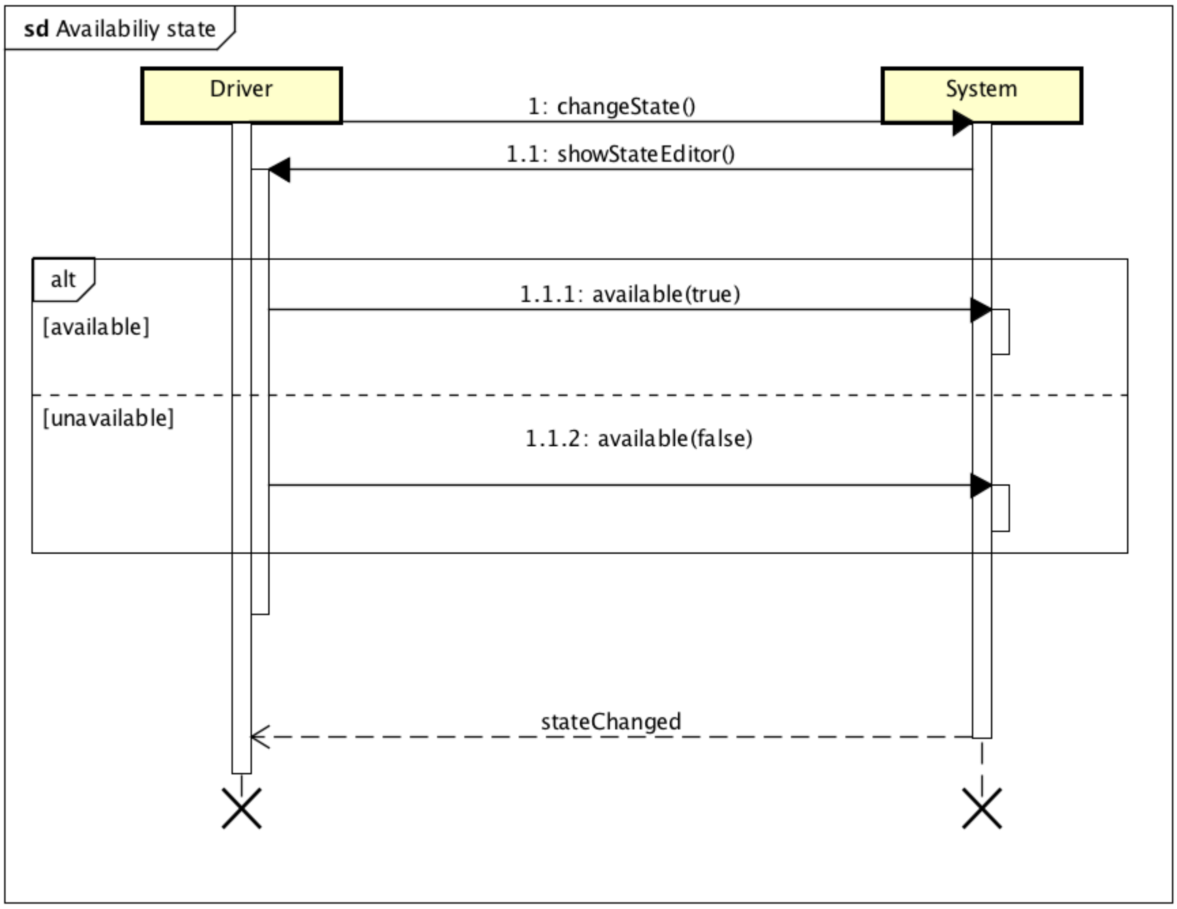


Figure 21: Taxi Driver Change Availability State UML Sequence Diagram

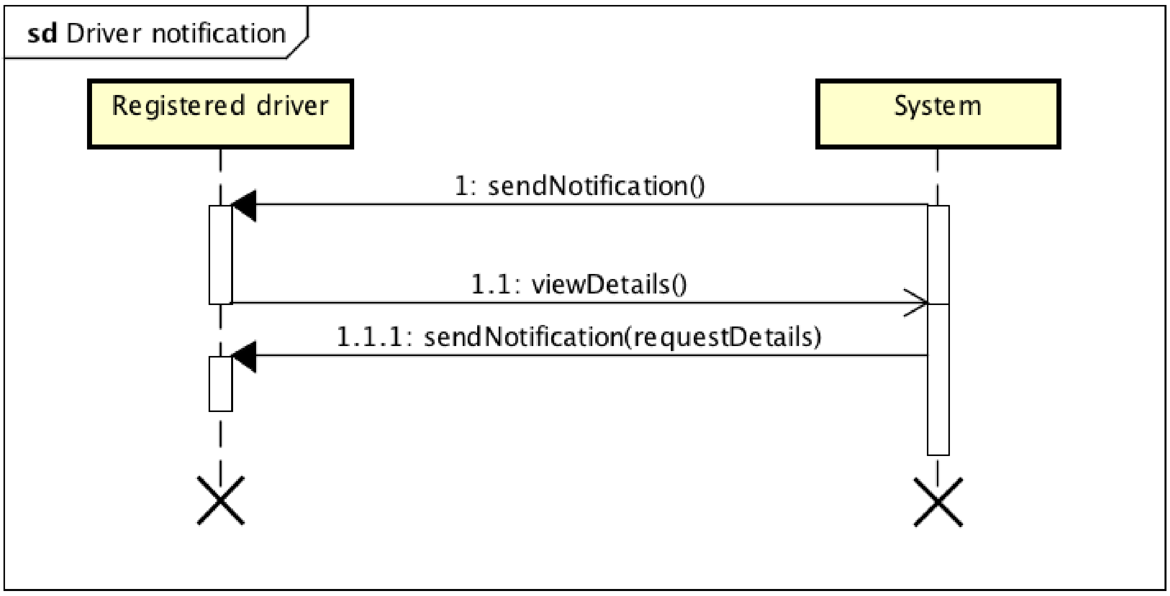


Figure 22: Taxi Driver Request Notification UML Sequence Diagram

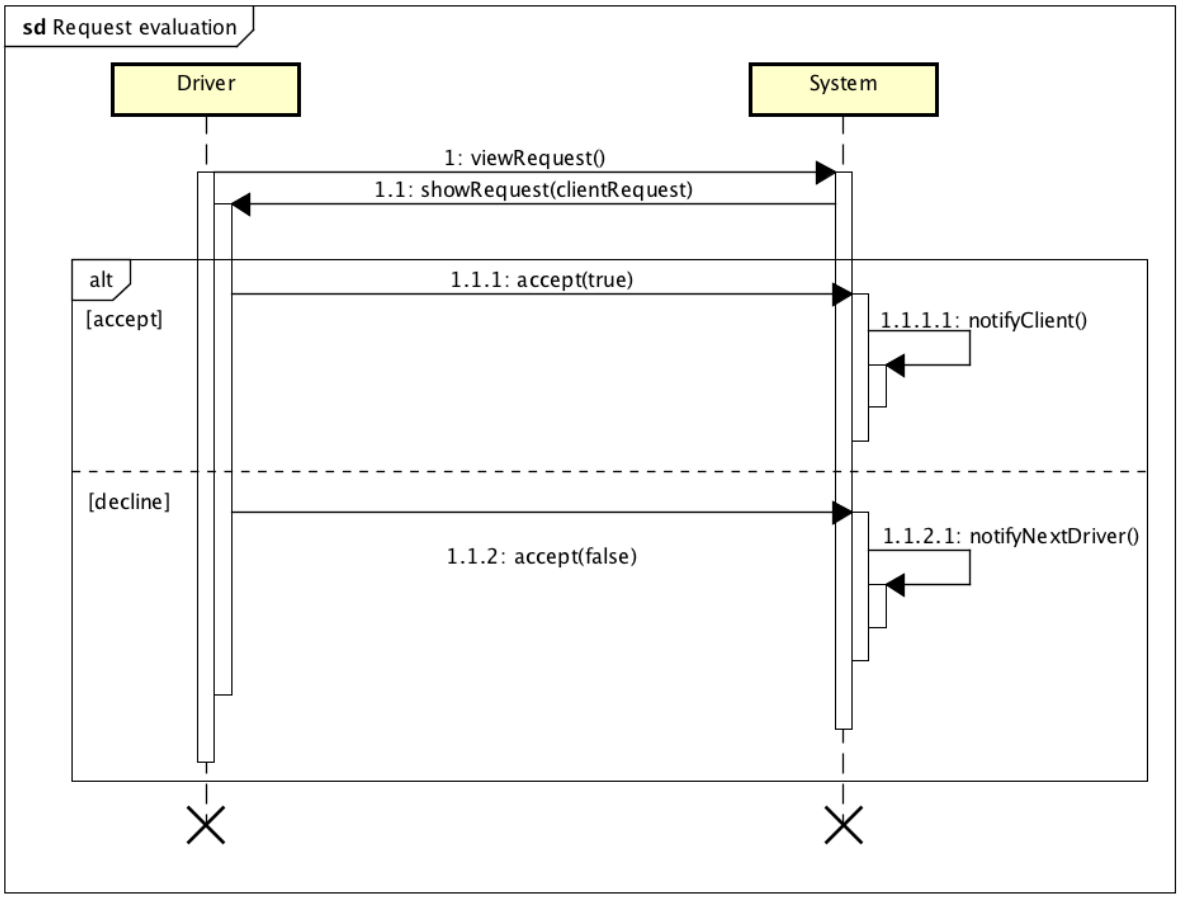


Figure 23: Taxi Driver Request Evaluation UML Sequence Diagram

3.6.4 BPMN Diagrams

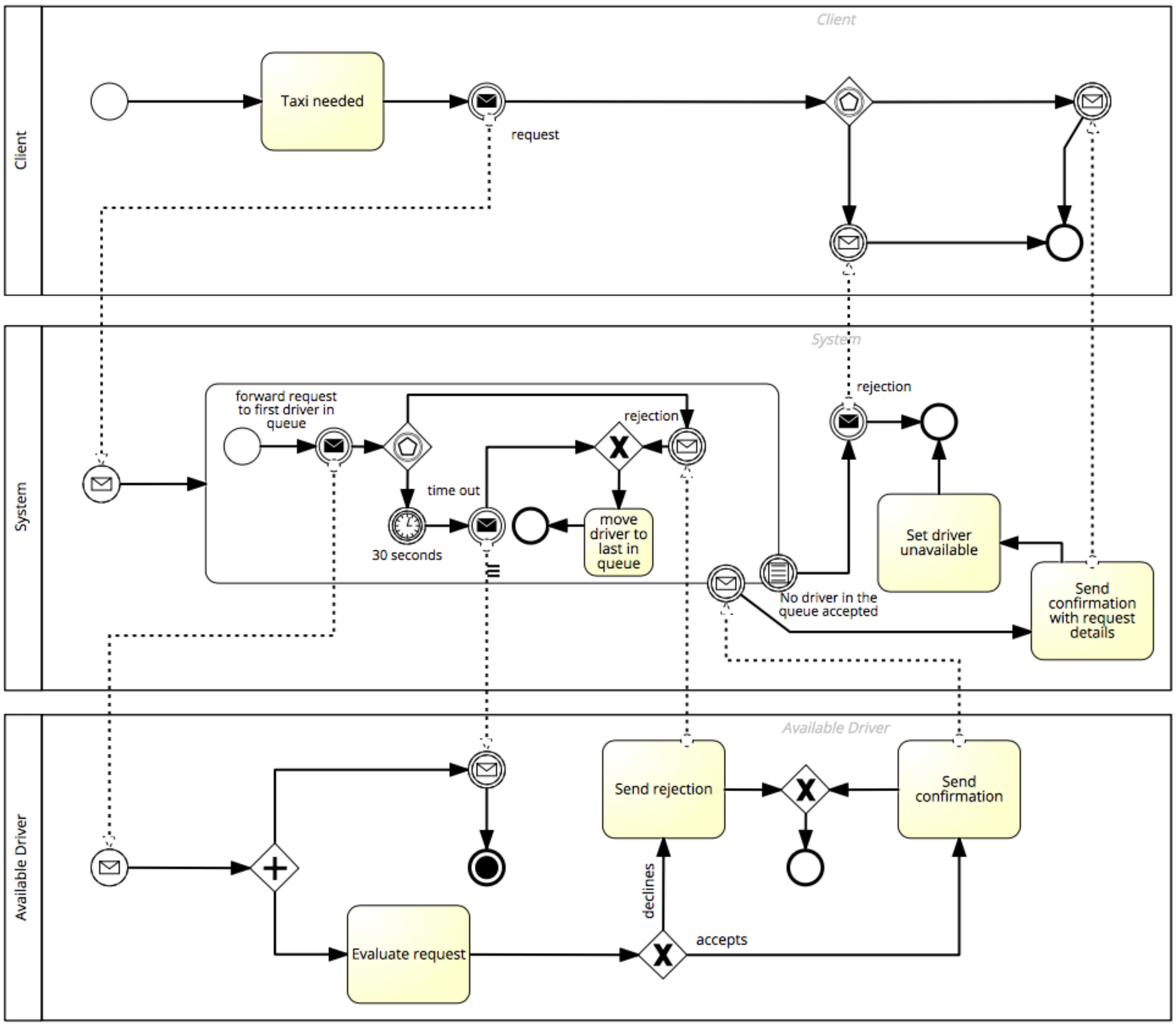


Figure 24: This is a BPMN diagram of the taxi's request process of *myTaxiService*

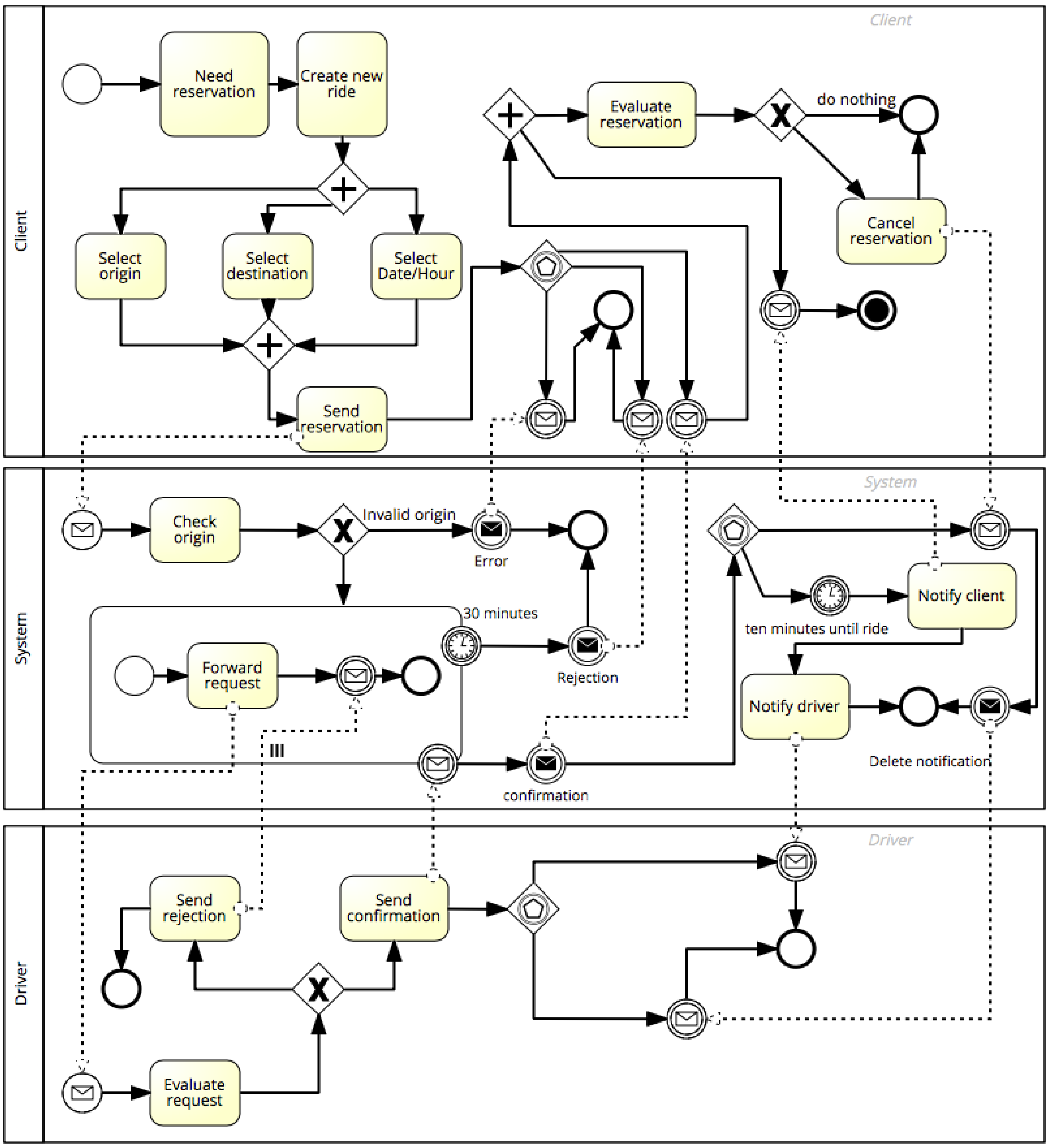


Figure 25: This is a BPMN diagram of the taxi's reservation process of *myTaxiService*

3.6.5 Class Diagram

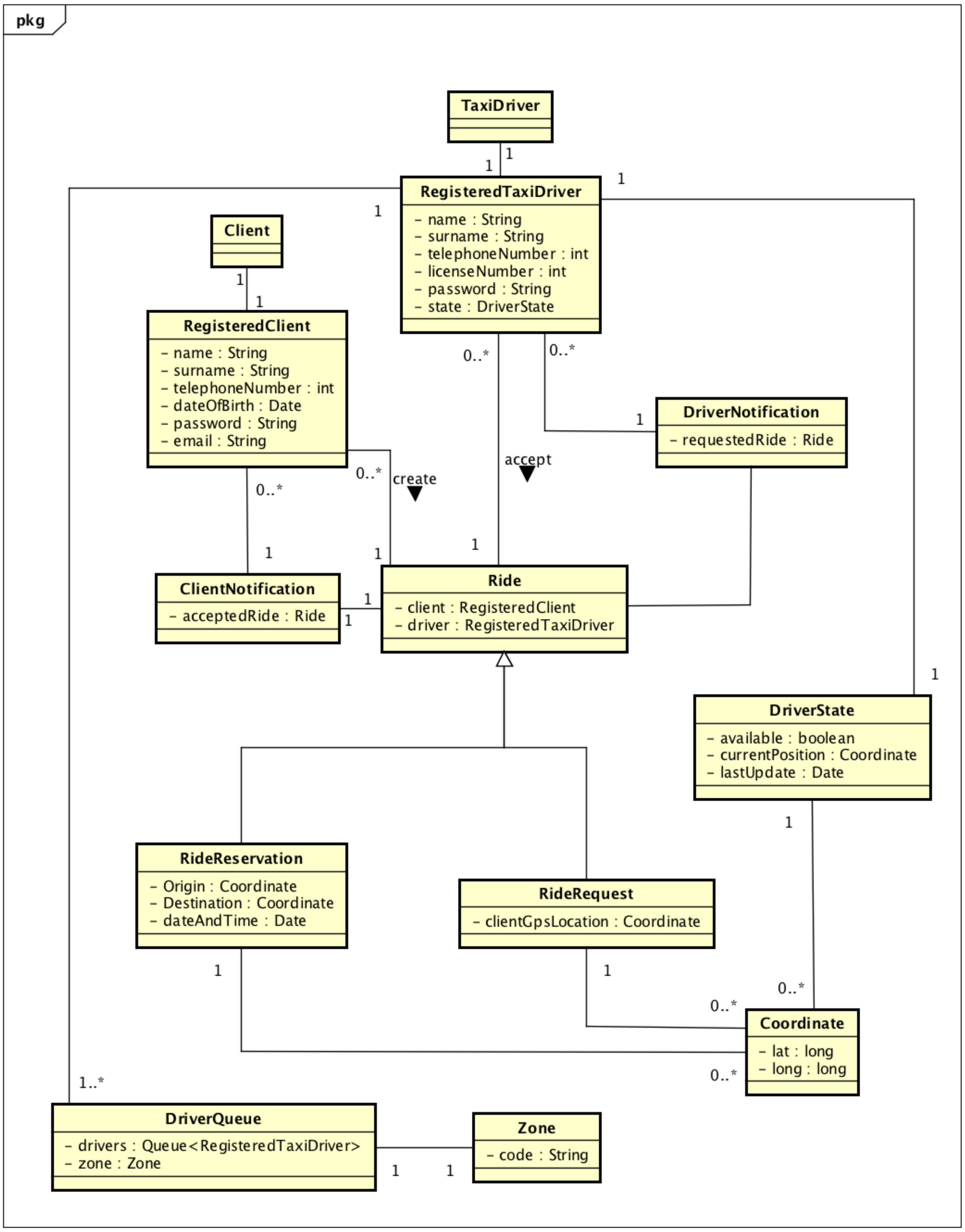


Figure 26: This is an high level class diagram of *myTaxiService*

4 Appendixes

4.1 Alloy

We have used the *Alloy Analyzer* in our project to make a formal definition of the most important properties of our system. In the following sections the model and the results derived using this tool are presented.

4.1.1 Signatures

Here are presented the signatures of our model. Some fields which have not a relevant importance in term of constraint were omitted in order to make the model as simpler as possible.

```
// Import boolean utility
open util/boolean
// Primitive String Type
sig Strings{}

// General user of the service
abstract sig User {
  eMail: one Strings
}
// Taxi Driver
sig Driver extends User {
  isAvailable: one Bool
}
// Client
sig Client extends User {}

// Taxi Drivers Queue
sig Queue {
  root: lone Node
}
// Queue Element
sig Node {
  ndriver: one Driver,
  next: lone Node
}

// City Zone
sig Zone {
  queue: one Queue,
}

// Taxi Call made by a Client
sig TaxiCall {
  client: one Client
}
// Call Confirmation made by a Taxi Driver and related to a Taxi Call
sig CallConfirmation {
  call: one TaxiCall,
  cdriver: one Driver
}

// Ride Request made by a Client
sig RideRequest {
  client: one Client
}
// Ride Confirmation made by a Taxi Driver and related to a Taxi Call
sig RideReservationConfirmation {
  request: one RideRequest,
  rdriver: one Driver
}
```

4.1.2 Facts

In this section the code represent all the constraints necessary to make the model consistent.

```
open signatures

// BEGIN: Definition of a generic Queue:
// A node cannot be connected to itself
fact nextNotReflexive {
  no n:Node | n = n.next
}
// Cycles must not be present
fact nextNotCyclic {
  no n:Node | n in n.^next
}
// Cannot exist a node not in a queue
fact allNodesBelongToAQueue {
  all n: Node | one q: Queue | n in q.root.*next
}
// END

// All the available taxi drivers must be in one and only one queue
fact availableDriverInQueue {
  all d: Driver | (one n: Node | d.isAvailable.isTrue implies d in n.ndriver) or d.
    isAvailable.isFalse
  all d: Driver, n: Node | d in n.ndriver implies d.isAvailable.isTrue
}

// A queue must own to a zone
fact queueMustBeInAZone {
  all q:Queue | one z: Zone | q in z.queue
}

// A Taxi call can have at most a Call Confirmation
fact oneCallConfirmationPerTaxiCall {
  all c: TaxiCall, disj cc1,cc2: CallConfirmation | c in cc1.call implies not c in cc2.
    call
}

// A Client can make at most a Taxi Call at a time
fact oneTaxiCallPerClient {
  all c: Client, disj c1,c2: TaxiCall | c in c1.client implies not c in c2.client
}

// A Taxi Driver can accept at most a Call at a time
fact oneCallConfirmationPerDriver {
  all d: Driver, disj cc1,cc2: CallConfirmation | d in cc1.cdriver implies d not in cc2.
    cdriver
}

// A Taxi Driver who accepts a Taxi Call must become "unavailable"
fact busyDriver{
  all d: Driver, cc: CallConfirmation | d in cc.cdriver implies d.isAvailable.isFalse
}

// A Ride Request can have at most a Ride Reservation Confirmation
fact oneRideReservationConfirmationPerRideRequest {
  all rr: RideRequest, disj rc1,rc2: RideReservationConfirmation | rr in rc1.request
    implies not rr in rc2.request
}

// Two different Users cannot have the same e-mail address but with the same e-mail a
    driver must be registered as a client too.
fact noSameEMail{
  all disj c1, c2: Client | not c1.eMail = c2.eMail
  all disj d1, d2: Driver | not d1.eMail = d2.eMail
}
```

4.1.3 Functions

The following code is a function used to write assertions in an easy way.

```
open signatures

// given a Queue, return all the drivers in this Queue
fun driversInQueue [q: Queue] : set Driver {
  q.root.*next.ndriver
}
```

4.1.4 Assertions

Here there are some assertions to be checked in order to verify the model.

```
open signatures
open functions

// Check that a driver belogs only to one zone
assert noDriverInTwoZones{
  all d: Driver,  disj z1, z2: Zone | d in driversInQueue [z1.queue] implies d not in
    driversInQueue [z2.queue]
}

// Check that in a queue there aren't unavailable drivers
assert noUnavailableDriverInAQueue{
  all d: Driver, z: Zone | d.isAvailable.isFalse implies d not in driversInQueue [z.
    queue]
}

// Check that each the available drivers belog to a zone
assert availableDriverInAZone {
  all d: Driver | (one z: Zone | d.isAvailable.isTrue implies d in driversInQueue[z.
    queue]) or d.isAvailable.isFalse
  all d: Driver, z: Zone | d in driversInQueue[z.queue] implies d.isAvailable.isTrue
}

// Check that two clients have different e-mails
assert differentEMail {
  no disj c1, c2: Client | c1.eMail = c2.eMail
}
```

4.1.5 Predicates

The code below is the declaration of some predicates.

```
open signatures

// Show generic big world
pred show {
}

// Show a generic world
pred showWorld1(){
  #Node = 3
  #Driver = 4
  #CallConfirmation = 1
  #TaxiCall = 2
  #Client =3
  #Zone = 2
  #RideRequest = 0
}

// Show a world with only available taxi driver
pred showWorld2 {
  #Zone = 1
  #Driver = 5
  #Client = 0
  #RideRequest = 0
}

// Show a world with no available taxi driver
pred showWorld3 {
  some d: Driver | d.isAvailable.isFalse and not ( one cc: CallConfirmation | d in cc.
    cdriver ) and not ( one rr: RideReservationConfirmation | d in rr.rdriver )
}

// Show a world in which is enlighted the coexistence between Calls and Reservations
pred showWorld4 {
  #Zone = 1
  #RideReservationConfirmation > 1
  #RideRequest > #RideReservationConfirmation
  #CallConfirmation > 0
}
```

4.1.6 Main

This is the main code of the model.

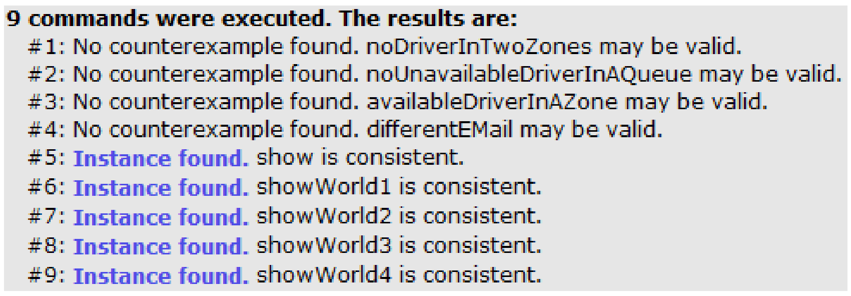
```
// Import all the parts of the model
open signatures
open functions
open facts
open assertions
open predicates

// Check the assertions
check noDriverInTwoZones for 5
check noUnavailableDriverInAQueue for 5
check availableDriverInAZone for 5
check differentEMail for 5

// Generate some Worlds
run show for 20
run showWorld1 for 10
run showWorld2 for 10
run showWorld3 for 10
run showWorld4 for 10
```

4.1.7 Output

The following screenshot represents the output generated by the *Alloy Analyzer* starting from the code presented in the previous sections. It shows that the model is consistent.



```
9 commands were executed. The results are:
  #1: No counterexample found. noDriverInTwoZones may be valid.
  #2: No counterexample found. noUnavailableDriverInAQueue may be valid.
  #3: No counterexample found. availableDriverInAZone may be valid.
  #4: No counterexample found. differentEMail may be valid.
  #5: Instance found. show is consistent.
  #6: Instance found. showWorld1 is consistent.
  #7: Instance found. showWorld2 is consistent.
  #8: Instance found. showWorld3 is consistent.
  #9: Instance found. showWorld4 is consistent.
```

Figure 27: Alloy Output

4.1.8 Worlds

The images in this section represent some *Worlds* generated with *Alloy Analyzer*. These *Worlds* show some configurations of our system.

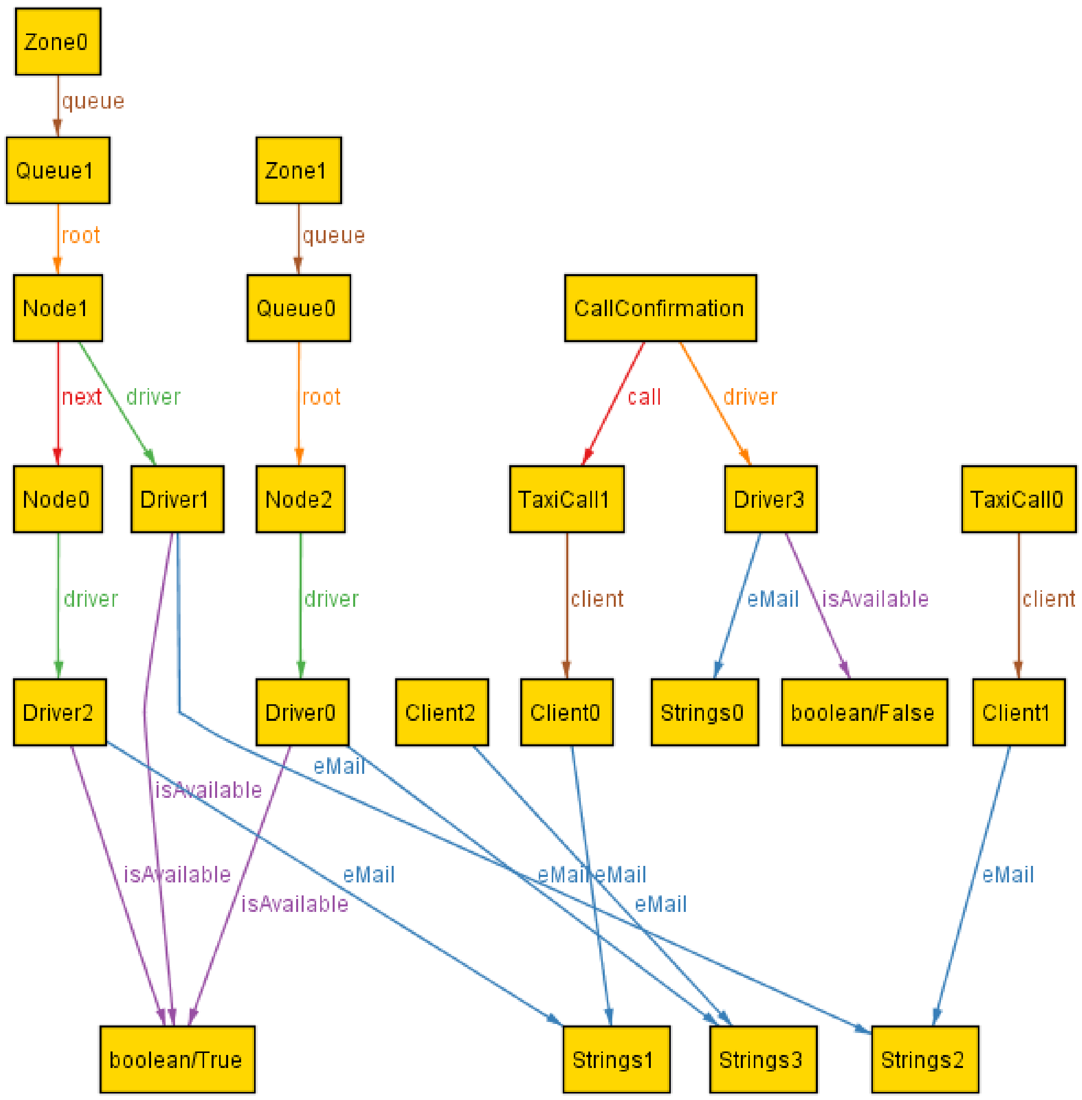


Figure 28: This alloy world represents a generic situation in which there are two zones, some drivers are available and other are busy in a Call. Here is not present any Reservation

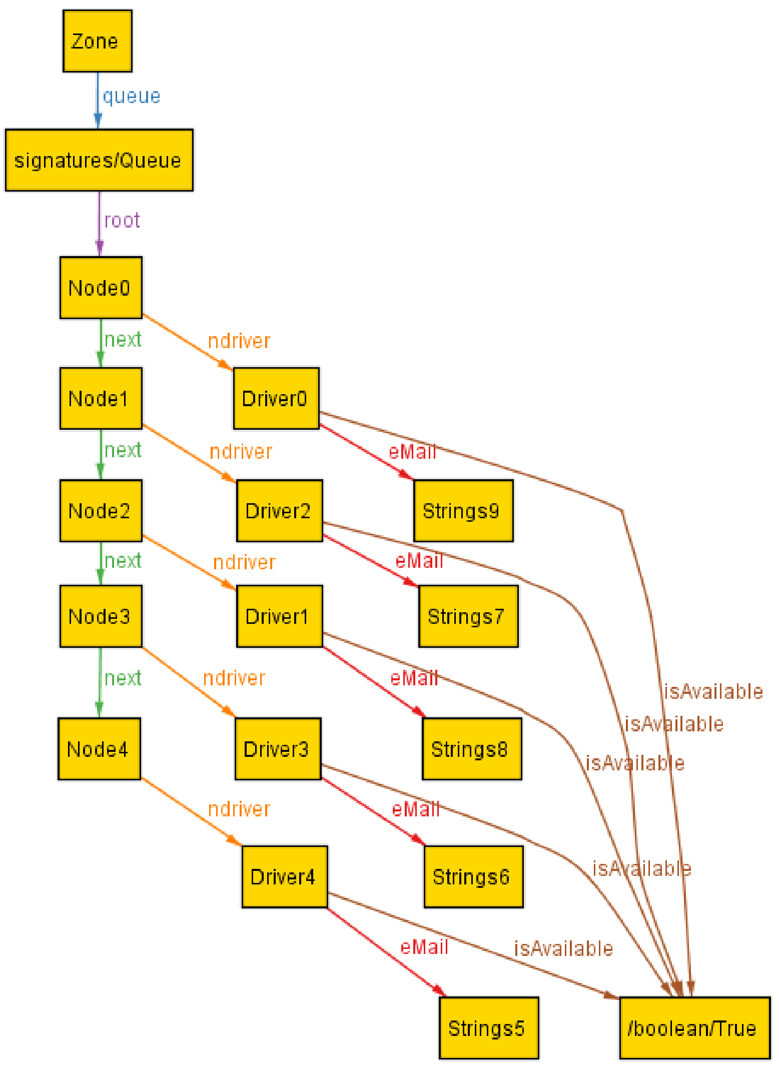


Figure 29: This alloy world represents a particular situation in which all the Taxi Drivers are available. This is useful to see how the queue is implemented. Here is not present any Reservation

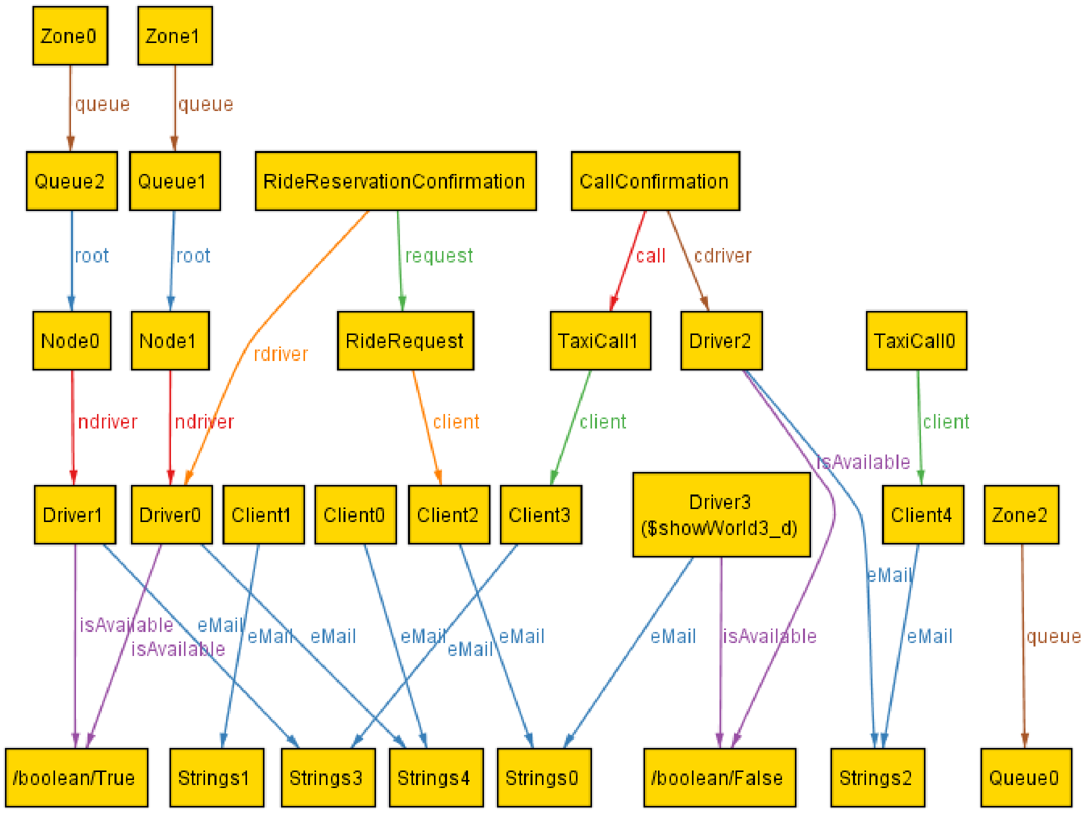


Figure 30: This alloy world represents a situation in which you can show all the possible states of a Taxi Driver: Driver 0 is available but has a reservation accepted, Driver 1 is available, Driver 2 is busy in a Call and Driver 3 is at rest.

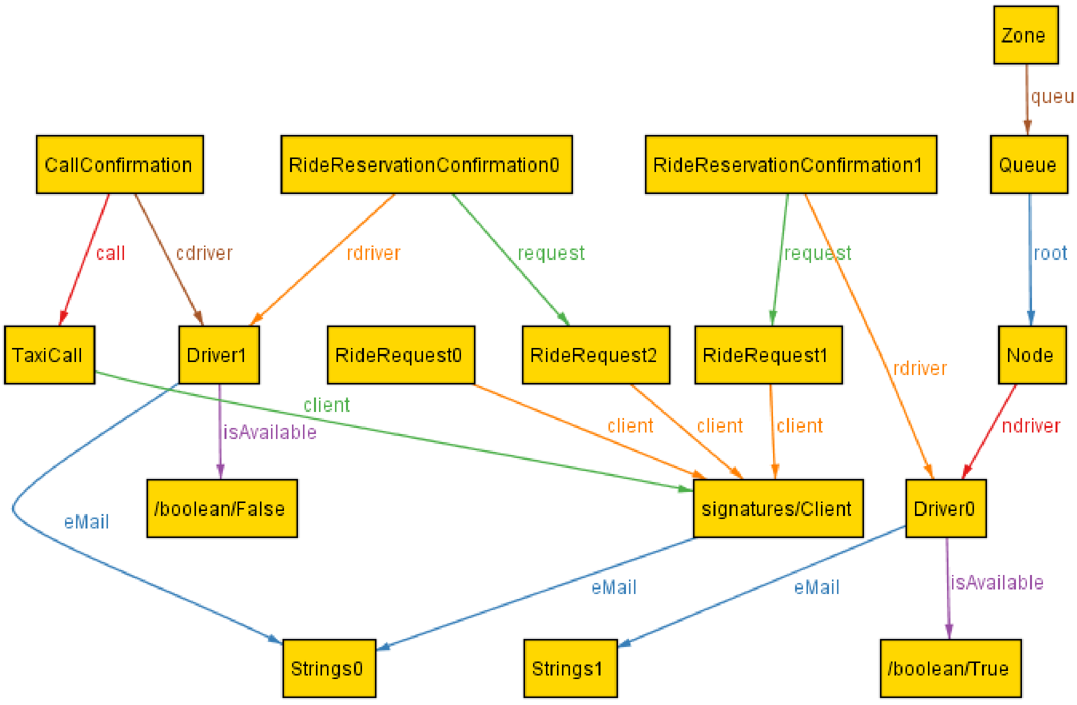


Figure 31: This alloy world was made to show a situation with some Ride Reservations mixed with Calls

4.2 Software and Tools used

ShareLatex: This web application was used to redact this document in a collaborative way. (`https://it.sharelatex.com/`)

NinjaMock: This online service was useful for design the mockups present in this document. (`https://ninjamock.com/`)

Creately: This web service helped us to create Use Case Diagrams. (`https://creately.com`)

Astah Professional: This desktop application was used to create all the others UML Diagrams. (`http://astah.net/`)

Signavio Academic: This web application was used to create the BPMN diagrams (`http://academic.signavio.com`)

Alloy Analizer: This tool was used to prove the consistency of our model. (`http://alloy.mit.edu/alloy/`)

4.3 Hours of Work

We spent approximately the following amount of hours to redact this document:

Riva Luca: 35

Strada Jacopo: 35

5 Revisions

v1.0 First final release

v1.1 Spelling corrections in mockups

v1.2 Added version number and *Revisions* section