Requirement Analysis and Specification Document

Alessandro Comodi, Davide Conficconi, Stefano Longari

November 6, 2015

Contents

Chapter 1

Introduction

This section aims to show in general terms the purpose and a brief description of the project, with the integration of a glossary which will be useful for the better understanding of the terms. The software to be developed is called myTaxiService.

1.1 Purpose

This is the Requirement Analysis and Specification Document of the project. Its purpose is to give fairly detailed specifications for the system to be. It illustrates a complete declaration for the development system, explain constraints and possible interactions with other external applications. The aim of this document is to be proposed to stakeholders for their approval and to be as a reference for the development team.

1.2 Scope

Users with prior registration, through a web or a mobile application, are able to request a taxi, giving the starting position of the ride and the number of passengers. The overall area covered by taxis is divided in smaller zones by the

system, which picks the first available taxi in the zone selected by the user or place the user in queue if no taxi is available at the moment. It then sends a notification to the chosen taxi driver through the mobile app. The taxi driver has to decide whether to accept or decline the intervention. Once a taxi driver confirms the user is notified of the needed time for the taxi to arrive and is given a code in order to verify his identity. Users can reserve taxis giving the system the starting and the arrival position besides the starting time. It is not possible to reserve within the last two hours before the ride. If the request for a reservation is accepted by the system it confirms the user about his ride and places a taxi at the starting position 10 minutes before the meeting time. Users can also enable the taxi sharing option when sending a request. The system, in this case, searches for other users who are willing to share a taxi. If there are users who go in the same direction the system alerts all the users and the taxi that the ride will be shared and calculates the fee for each user depending on the length of their ride.

1.3 Stakeholders

The government of the city, the institution which submitted and funded the project for the most part. Its main objective is to reorganize and optimize the city taxi service under the request of the inhabitants that complained about the bad quality of the service. The taxi companies which joined the project and funded the remaining part. Their main goal is to earn the most from through a better management of their service. The inhabitants / future users of the application who wants an easier access to the service and a reduced waiting time in queue for a taxi.

1.4 Glossary

This section explain the meaning of frequently used terms in this document.

- MTS: acronym of the application and the full name is “MyTaxiService”;

- Guest: person that needs to register to the system in order to access all of the functionalities;
- User: a person that is registered to the system and can access to the costumer's services;
- Taxi driver: the person who is logged in as a taxi driver. It is needed to distinguish between normal users (customers) and the taxi drivers.
- Application: with this term it is intended both the web and mobile application. When a different specification for the web and mobile is needed, it will be explicitly said.
- Request: an immediate need of a taxi.
- Reservation: prenotation of a taxi in a postponed time and date.
- Ride: used to identify both requests and reservations.
- Busy: when a taxi driver has accepted a ride and therefore he is not in the queue.
- Available: when a taxi driver is in a queue waiting to accept a ride.

Chapter 2

Overall description

2.1 Product Perspectives

The application to be implemented is going to be mostly independent and self-contained although it will have to be supported by some external systems. The management of the taxi queues and the allocation of the taxi for each passenger is done automatically by the application. MTS is going to be developed as both a Mobile and Web application and so, in order to correctly work, it will need to be integrated with Internet. Another integration is the use of GPS for the navigation system of each taxi. It is going to be used to gather information about the position of the taxis, so that the system can calculate the optimal route and the number of taxis in each zone.

2.2 Product Functionalities

The actors and the relative features for each one that myTaxiServer offers are:

A1 Guest: guests can:

- register;

- log in.

A2 User: users can:

- modify personal information;
- request a taxi;
- check taxi status;
- make a reservation;
- dismiss a reservation;
- abilitate sharing option.

A3 Taxi Driver: taxi drivers can:

- change availability;
- confirm or decline a request.

2.3 User characteristics

Users of the application are required to have a basic web knowledge and a device, like a personal computer or a smartphone, to be able to utilize the service and an identification document in order to avoid the registration of multiple accounts by the same person. Taxi drivers aside from the previous specifications and the strict necessity of a smartphone with integrated gps system are registered by the taxi company to avoid any kind of illegal use.

2.4 Constraints and General Assumptions

2.4.1 Regulatory policies

The system has to follow the laws about the privacy of the data given by the users and the policies regarding cookie storage.

2.4.2 Hardware limitations

On the user side the application has to run on a device connected to the internet while on the taxi driver side the device has to be equipped also with a GPS system. The system has to run on a rented or owned server.

2.4.3 Software limitations

- If it is run on a mobile device, the application needs to be compatible with the following operating systems: iOS, Android and Windows Phone, covering at least the 90% of the devices.
- The application, if used via the web, has to be compatible with the most important browsers (Chrome, Firefox, Internet Explorer and Safari).

2.4.4 Interfaces to other applications

API's for maps, GPS management and the navigation system, besides the use of the different application markets are necessary. An e-mailing interface is also needed to send confirmation of the various reservations and requests.

2.4.5 Parallel operations

MTS needs to support parallel operations from different users and taxi drivers

2.5 Domain Assumptions

All the possible ambiguities and the needed hypotheses are clarified in this section and explained with all their reasons.

[A1] Guests who want to register are provided with an ID document in order to register to the system.

[A2] The system is able to check wether an ID document is valid or not in order to verify the authenticity of the guest.

[A3] All the taxi drivers are provided with a smartphone in order to receive the taxi requests.

[A4] It is assumed that the city is a metropolis and the potential customers are around 2-3 millions.

[A5] The taxi companies that will use MTS have available a number of taxis that can cover the whole city requests.

[A6] The mobile application for drivers isn't available on marketplaces, this owing to prevent from any kind of cyberattack or illegal use. Every taxi company provides the application and a unique identification code, among all companies, to each driver.

[A7] The city is divided in zones, each one with two square kilometers of extension approximately.

[A8] All the GPS data incoming from different devices are 100% correct.

Chapter 3

Specific Requirements and Scenarios

In this chapter there are the main functional requirements with the relative scenarios, event flows and scheme diagrams. The reader can also find the non-functional requirements of the application.

3.1 Functional Requirements

In this section are described the main functional requirement.

3.1.1 Registration and login

[R1.1] Guests, who are not yet logged in the system, are only able to login or to register.

[R1.2] If a guest tries to login with wrong credentials he is requested to retry and sent to the previous page.

[**R1.3**] Guests has to fill the form with their personal data: name, surname, address and phone number.

[**R1.4**] During the registration guests have to give a valid email address to which will be sent a confirmation email and a valid ID document code to demonstrate their legitimacy. The system checks for the authenticity of the document before accepting the registration.

[**R1.5**] When registering a guest has also to insert a password with at least eight characters.

3.1.2 Profile management

[**R2.1**] Users are able to add and modify their email, password, telephone number, home address.

[**R2.2**] Users can change the ID document code but it has to be verified by the system.

[**R2.3**] If the users modifies the password, the system must ask for the old password in order to verify the authenticity of the user. This to increment the security of the application.

[**R2.4**] Users cannot add more than one home address and ID document code.

[**R2.5**] Users are not able to modify their name and surname.

3.1.3 User side taxi request

[**R3.1**] If users give non existing starting or destination addresses the system notifies them to try again and refuses the request.

[**R3.2**] If the user has already a pending request and asks for another one he is notified that the second request will not be forwarded and it is refused.

[**R3.3**] If the request is vaild the user is asked to wait for the first taxi that will accept it.

[**R3.4**] When a taxi driver accepts the request, the user is given the taxi ID and a code to show to the driver in order to avoid other passengers to take the incoming taxi.

[**R3.5**] A user is able to see where the taxi is and the expected time for its arrival.

3.1.4 User side taxi sharing

[**R4.1**] While sending a request or reserving a taxi the user must have the possibility to share the ride with other users.

[**R4.2**] If the ride will be shared the client is advised from the system with a notification containing the percentage of the owed fee.

3.1.5 User side taxi reservation

[**R5.1**] To reserve a taxi the user must provide date and time of the reservaiton.

[**R5.2**] Date must be greater or equal than the one in which the reservation is submitted.

[**R5.3**] Users cannot reserve in the last two hours before the ride and so those hours must be unavailable.

[**R5.4**] Users must insert starting point and final destination of the ride.

[**R5.5**] If users give non existing starting or destination addresses the system notifies them to try again and refuses the reservation.

[**R5.6**] As the system can compute the average time of the reserved ride, if the user asks for another ride for a time in which he should still be on the first one it is refused and the client notified.

[**R5.7**] Users are notified when the taxi driver accepts the ride.

[**R5.8**] Users are able to cancel the reservation until 10 minutes before the meeting time, not later.

3.1.6 Taxi driver side notifications

[**R6.1**] Drivers are told by the system in which zone they are working.

[**R6.2**] When ready, the drivers have to alert the system that they're waiting for a request and are put in the queue of their zone.

[**R6.3**] When a request arrives to the system the chosen taxi driver is given the informations about the ride.

[**R6.4**] A taxi driver has to answer the request within 1 minute otherwise the reqeust passes to the next available taxi and he is moved to the bottom of the queue.

[**R6.5**] Taxi drivers must be informed if a ride is shared or not.

[**R6.6**] If a ride is a reservation the taxi driver must be informed of it.

[**R6.7**] Taxi drivers can only see their own requests and reservations and not the ones of other drivers.

[**R6.8**] Taxi drivers cannot receive notification while busy.

[**R6.9**] When a taxi driver accepts a request he is automatically set to busy and removed from the queue.

3.1.7 Queue Management

[**R7.1**] The city is divided in virtual zones.

[**R7.2**] Each zone of the city has an assigned queue of taxis.

[**R7.3**] The system is aware of the positions of each taxi through GPS informations.

[**R7.4**] The system knows the statistic distribution of taxi requests and reservations in the city and analyzes through these data the optimal division of taxis in different zones.

[**R7.5**] When a taxi driver becomes available he is informed by the system in which zone he is assigned.

[**R7.6**] In case a request cannot be accepted by any taxi in the zone the system searches for the nearest taxi in the surrounding zones.

[**R7.7**] Taxis rotate their positions during the week to avoid unfair management and significant difference in rides and payment.

3.2 Scenarios

In this section are described some of the most frequent use case of the application, divided in several different scenarios which will help the reader in understeanding the above specific requirements.

3.2.1 Registration

John has just downloaded the application from the app store and he would like to make use of it. He decides to register to the system in order to take advantage of the offered services and so he clicks on the “sign up” button. A registration

form shows up and he fills it with all the necessary data. If the information are correct he becomes a new user of the product, otherwise he is redirected to the home page with an error notification. From now on John can log in the system with his credentials.

3.2.2 Login

Mark, who has registered and is a user of MTS needs a taxi ride. He starts the application and clicks on the "login" button. After doing that he is redirected to the login page and so he fills the form with his e-mail address, the one he used to register to the service, and his password. If the informations are correct he is able to access to the home page of the application, otherwise an error message appears and he is asked to try again.

3.2.3 Taxi request

Robert wants to request a taxi to pick him up at the central station. He logs in the applicaiton and clicks on the "request" button and fills the form, with the starting and destination point. He then waits for the confirmation. When a taxi driver confirms the request a notification with the taxi ID, a code that has to be shown to the driver and the approximate waiting time is sent to Robert.He then walks to the place where the taxi will arrive.

3.2.4 Check for taxi status

Sarah has just requested a taxi and a driver confirmed the ride. She wants to know how much time is left until the arrival of the taxi and the position of this one. Sarah then clicks on the "check taxi status" button and selects the request she made. Sarah is able to see the position of the taxi and she can be sure that the vehicle will arrive on time.

3.2.5 Taxi reservation

Philip wants to reserve a taxi for the next day in order to reach the airport. He logs in the application and clicks on the "reserve" button and then completes the form with the time and date, the starting point and the final destination. The confirmation of the insertion of the ride in the system is sent to Philip. The system manages the reservation as a taxi request, allocating a taxi ten minutes before the meeting time. The following day Philip goes at the meeting point and finds the taxi waiting for him to take him to the airport.

3.2.6 Delete reservation

Tom has made a reservation for a taxi for the next day. He then decides to delete his reservation because a friend of his goes in the same direction with his own car. He goes then to the home page of the application, clicks on the "check taxi status" then he selects the reservation he made for the next day and deletes it. The system removes it from the database and no taxi will be requested to be at the meeting point the next day. If Tom changes his mind again he has to make another reservation.

3.2.7 Share a taxi

Bruce wants to reach the pub to meet his friends and Richard wants to go to the cinema, which is situated near the pub in which Bruce wants to go. Both of them decide to use MTS. Bruce and Richard decide to share the ride. The system calculates the road and the fee percentage for each user and sends a notification to the first available taxi driver. A confirmation arrives to both the users who shares the ride. And that's how Batman and Robin knew each other and joined in order to save Gotham city many times.

3.2.8 Taxi driver availability

Julia, who is a taxi driver, has just completed a ride and she is able to receive other requests. She then goes to the availability options page on the application and sets herself as “available”. From now the Julia is inserted in a queue of taxis until she accepts another ride.

3.2.9 Taxi driver confirmation

Ross, a taxi driver, receives a request for a ride. Even if he is about to end his working turn he decide to accept the ride and take the passenger to his destination. Ross then clicks on “confirm ride” button and reaches the meeting point.

3.3 Event flow and Scheme Diagrams

In this section the reader can find the event flow of the above scenarios are analyzed in detail and are accompained by the relative schemes.

3.3.1 Use Case Diagram

Below is presented the Use Case Diagram which represents all the main use cases of the application.

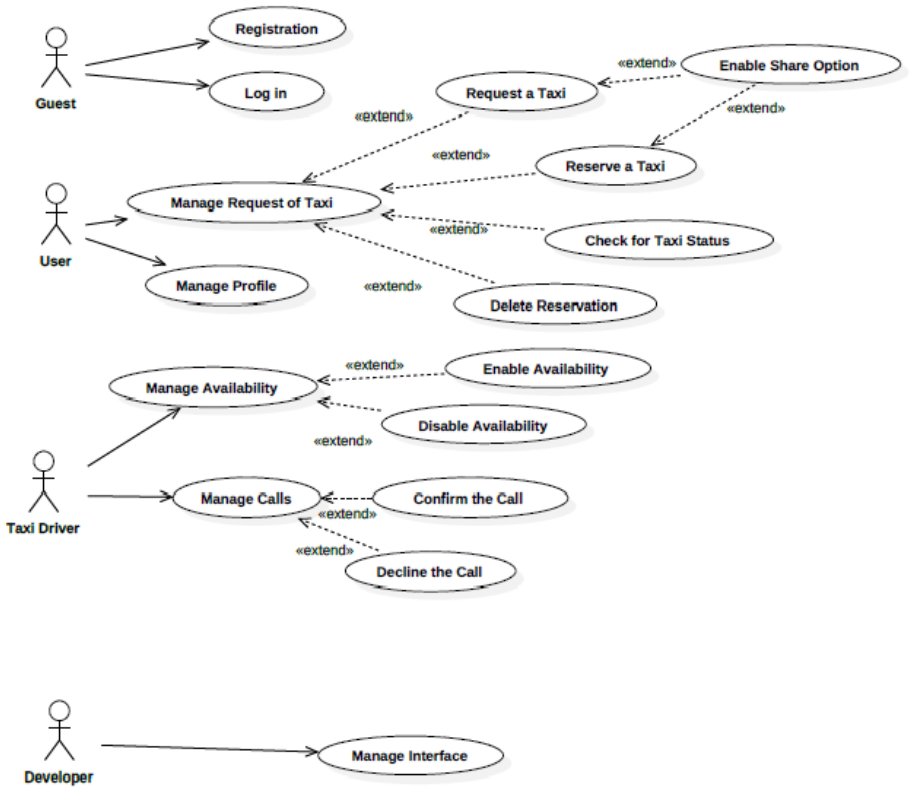

Figure 3.1: Use Case Diagram for MTS

3.3.2 Class Diagram

Below is presented the Class Diagram which represents the main classes of the application

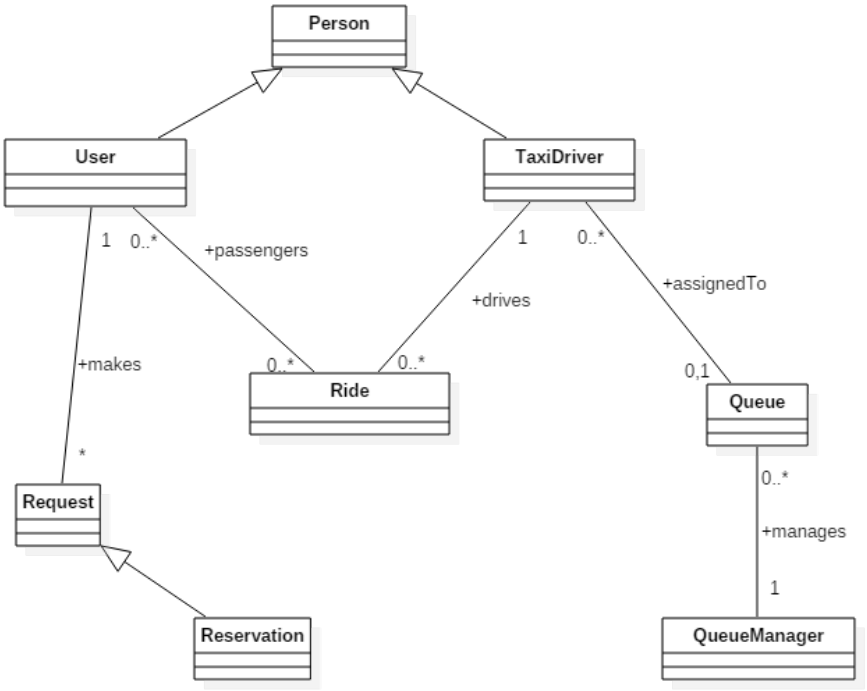

Figure 3.2: Class Diagram for MTS

3.3.3 Registration

Actor	Guest
Precondition	the visitor doesn't have an account and he isn't yet registered
Event flow	1. the guest accesses to the application or to the web page 2. he then clicks on the "sign up" button 3. he fills the form with his personal informations, a valid e-mail and ID document 4. he clicks on "register" button 5. the system checks the given inputs 6. the guest is inserted in the system as a registered user 7. the guest is sent to the home page of the application
Postcondition	the guest becomes a new user of the application and from now on can log in
Exceptions	the e-mail is already present in the system so it is not anymore unique or the given ID document is not valid

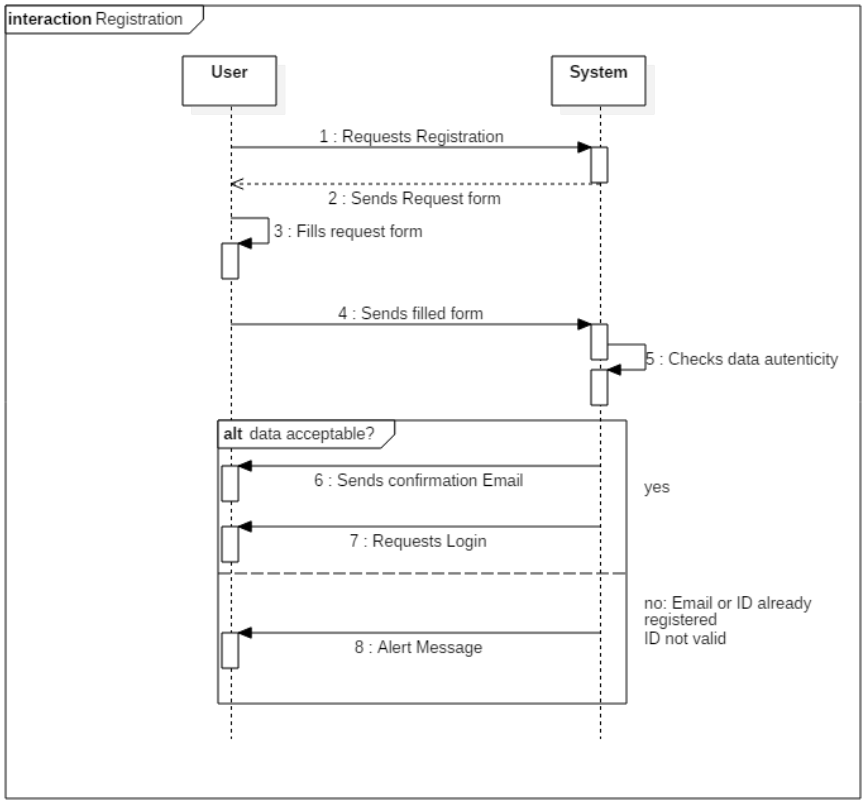

Figure 3.3: Sequence diagram for the registration

3.3.4 Login

Actor	guest
Precondition	the guest is registered in the system
Event flow	1. the guest fills the form, inserting as input his e-mail address and his password 2. the guest clicks on the “log in” button 3. the system checks the correctness of the input 4. the guest becomes a user
Postcondition	the user can access to all the available functionalities
Exceptions	the input password or e-mail are incorrect and the guest cannot log in

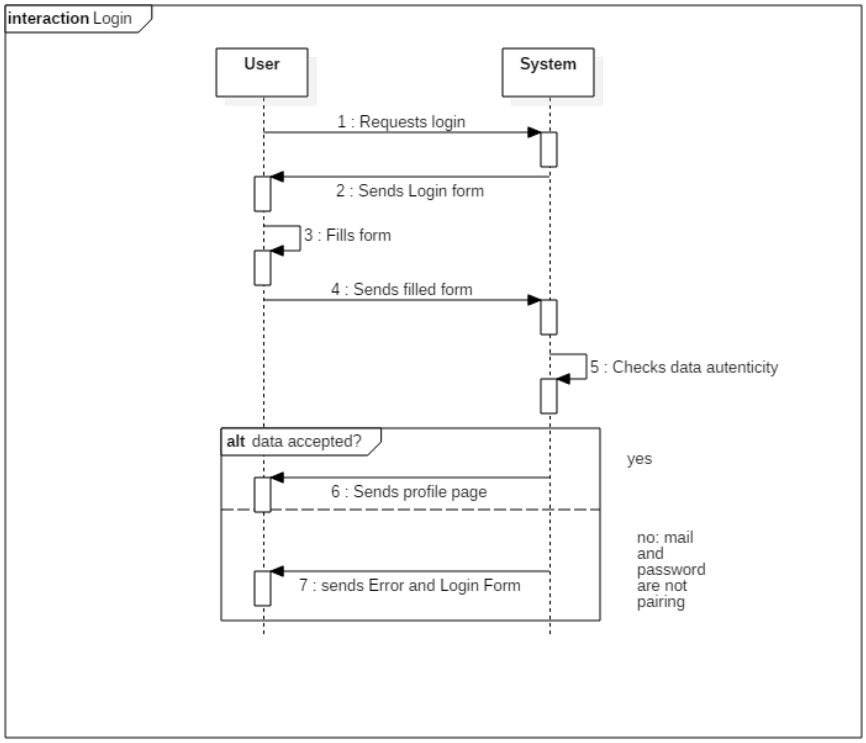

Figure 3.4: Sequence diagram for login

3.3.5 Taxi request

Actor	user
Precondition	the user is logged in
Event flow	1. the user clicks on the “request” button 2. he then fills the form with the starting point and the final destination 3. he clicks on the “submit” button 4. The system analyzes the request and, depending on the zone and the queue of taxis sends a notification to the first available taxi in the queue 5. When a taxi driver confirms the ride a notification is sent to the user
Postcondition	the user successfully requested the ride and waits for the arrival of the taxi
Eceptions	if the time of the ride coincides with a reservation that the user has previously made the request is refused

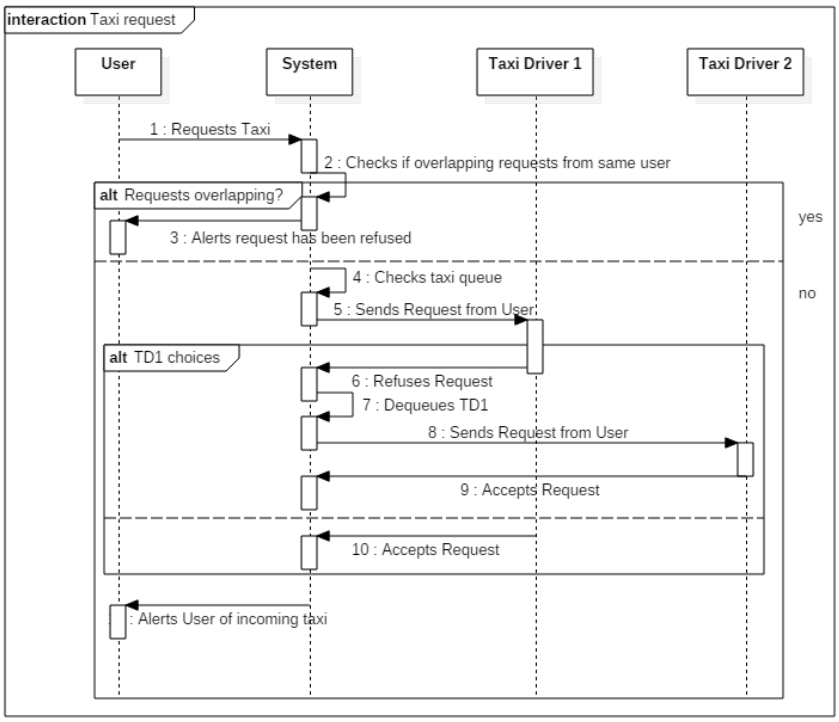

Figure 3.5: Sequence diagram for taxi request

3.3.6 Check for taxi status

Actor	user
Precondition	the user has requested a taxi
Event flow	1. the user clicks on the "check taxi status" on the home page 2. the system uses the GPS information to calculate the position of the incoming taxi and the expected time 3. the system shows to the user the information gathered about the incoming taxi
Postcondition	the user knows all the information about the arriving taxi
Exceptions	none

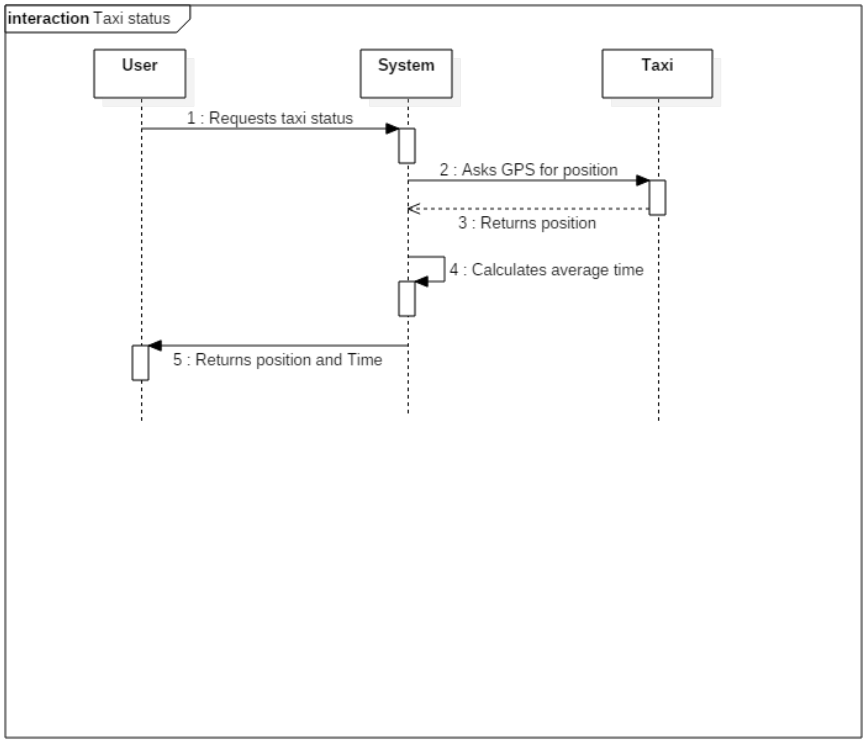

Figure 3.6: Check taxi status sequence diagram

3.3.7 Taxi reservation

Actor	user
Precondition	the user is logged in the system
Event flow	1. the user clicks on the "reserve" button 2. he fills the form with the time, starting point and final destination 3. if the information are correct the reservation is inserted in the system 4. the system informs the user of the occurred insertion 5. the system sends a notification to the first available taxi in the queue 10 minutes before the meeting point 6. when a taxi driver confirms the ride a notification with the summary of the reservation is sent to the user
Postcondition	the user has a new reservation stored in the system
Exceptions	the reservation time coincides with another reservation made in a previous moment

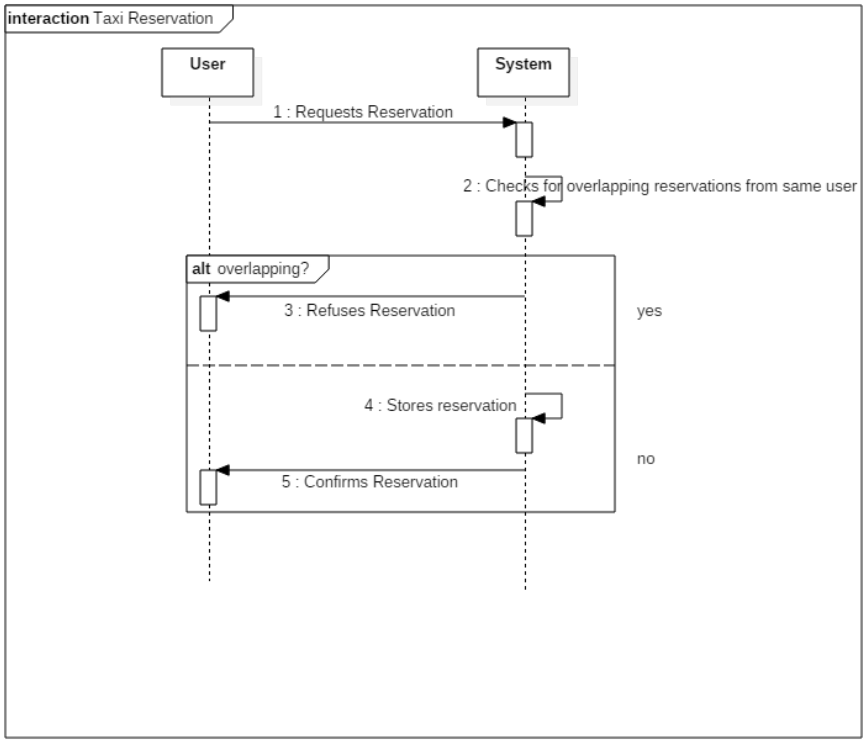

Figure 3.7: Taxi reservation sequence diagram

3.3.8 Delete a reservation

Actor	user
Precondition	the user made a reservation
Event flow	1. the user clicks on the “check reservations” button. 2. he then selects the reservation to be deleted 3. the user clicks on the “delete” button 4. the system removes the reservation
Postcondition	the reservation that the user made is not present anymore in the system
Exceptions	none

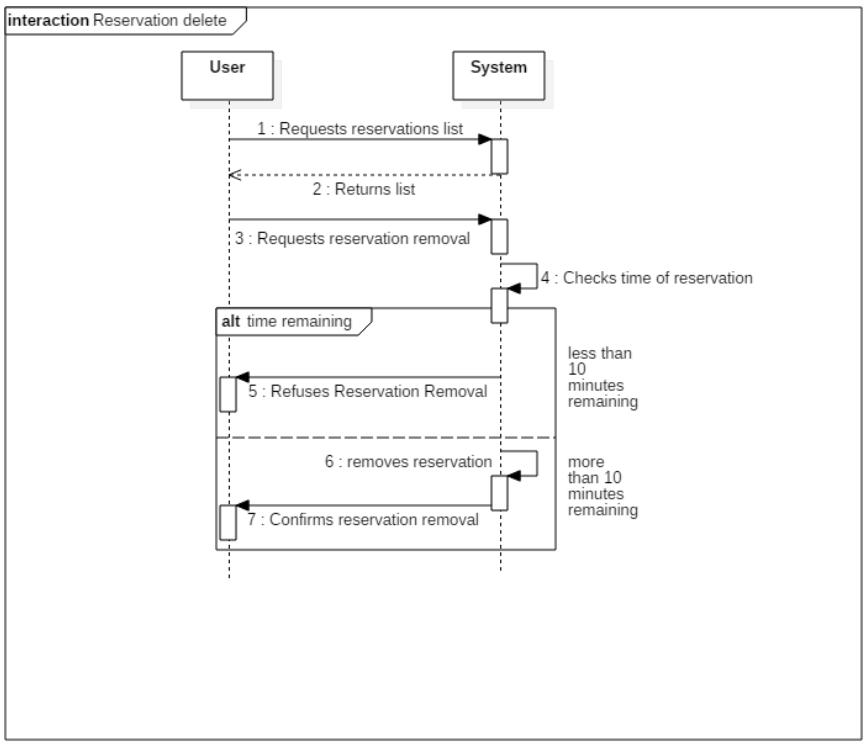

Figure 3.8: Delete a reservation sequence diagram

3.3.9 Share a taxi

Actor	user
Precondition	the user is logged in
Event flow	1. the user clicks on the “request” button 2. he then fills the form with starting point and final destination 3. he checks the “share a taxi” option 4. the system searches for possible matches with other users 5. if a match is found, the system sends a notification to the taxi driver 6. when a taxi driver confirms the ride, a confirmation is sent to all the users that share the ride
Postcondition	the taxi ride is shared among two or more users
Exceptions	there is no user departing from the same zone or going in the same direction. In this case the user cannot share the ride with anyone

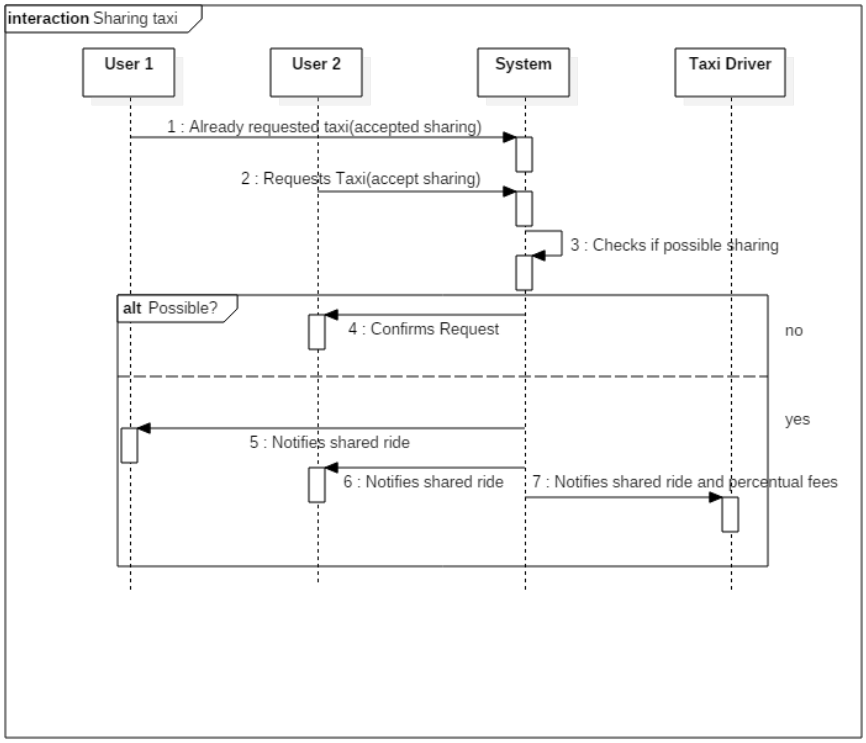

Figure 3.9: Share a taxi sequence diagram

3.3.10 Taxi driver availability

Actor	taxi driver
Precondition	he is logged in the system
Event flow	1. the taxi driver clicks on the “manage availability” button 2. he is redirected to a page that shows him his availability options 3. he can choose to set himself as “busy” or “available” 4. in case the taxi driver switches to “busy” the system removes him from the queue 5. in case the taxi driver switches to “available” the system adds him to the queue of taxis
Postcondition	if the taxi driver is “busy” the system will not send him request notifications
Exceptions	none

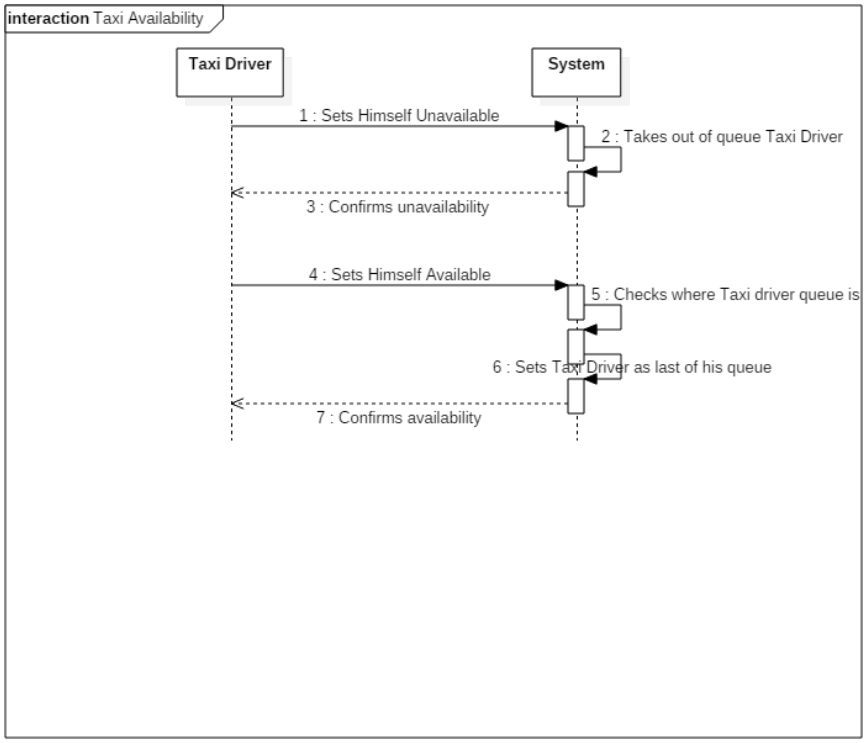

Figure 3.10: Taxi availability sequence diagram

3.3.11 Taxi driver confirmation

Actor	taxi driver
Precondition	he is logged in the system and is available
Event flow	1. the taxi driver receives a notification for a ride 2. He analyzes the allocation request 3. He then decides whether to accept or refuse it 4. If he accepts it a confirmation is sent to the user 5. the system places him out of the queue when the ride is about to begin and sets him as “busy” 6. if the ride is refused the system notifies the next taxi in the queue and he is sent in the end of the queue
Postcondition	the ride is associated with a taxi driver and he is approaching to the meeting point
Exceptions	the taxi driver could not answer in less then one minute, so he is not able to confirm the ride anymore

3.4 Entities behaviours

In this section are exposed the behaviours of some entities taken from Figure 3.2 using UML state chart diagrams

3.4.1 Ride class

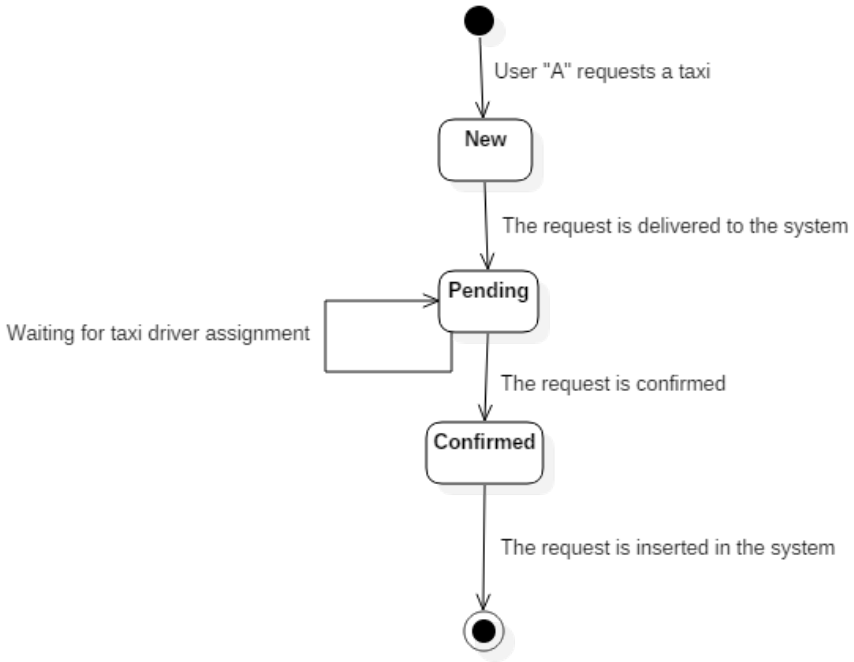

Figure 3.11: Ride state chart diagram

3.4.2 Reservation class

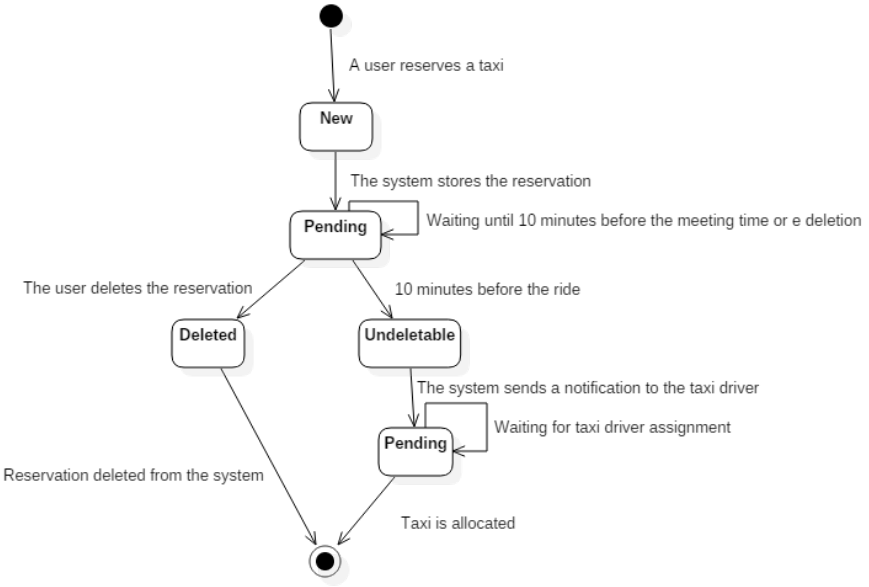

Figure 3.12: Reservation state chart diagram

3.5 Non-functional Requirements

This section contains some of the main non functional requirements that the Software to be should fulfil in order to provide a proper quality of the service and guarantee an accurate stream of operations.

3.5.1 User friendliness

Due to the wide target of people for whom the application is intended , it should present a very intuitive and effective user interface . Also old aged people and people with a poor knowledge of web or mobile applications should understand how to move inside the application and get a taxi.

3.5.2 Portability

The mobile application should be available for all major mobile devices with different kind of hardware and software. Also the web application should be used via all the main browser(e.g. : Chrome,Mozilla) with no kind of restriction of operative system or hardware features.

3.5.3 Maintainability & Reliability

The application need high ease of maintenance due to the fact that it has normally to be operative every day of the week , 7/7 days, and every hour, 24/24. Some exceptions could happen, but any necessary maintenance has to be very rare.

3.5.4 Availability & Performance

1. The system should be reactive and able to manage at least 1'000 of concurrent and simultaneous operations. Because of this and in order to avoid

traffic overloading more than one server will be needed.

2. At least 90% of the operations have to be completed in less than 100 ms.

3. In order to process all the information there is a need to have at least 24 gigabyte of RAM.

4. All the produced data has to be stored, so the servers must have at least 10 terabytes.

3.5.5 Security

1. The security of the system is guaranteed in different ways. A user is identified by the tuple email, password, number of id document, whereas a taxi driver is identified by the tuple email,code of company, number of license. No user has to be able to insert data into the database but all the details of the various rides (like the starting and destination point, user, taxi driver...) are stored and used to compute the distribution of the taxis.

2. passwords associated to users must be cripted in order to avoid security failures of the system.

Chapter 4

Alloy Modelling

This chapter analyzes the consistency of the Class Diagram presented in Figure 3.2 through the use of the Alloy Analyzer. In the following section there will be the Alloy code, the responses and the generated worlds.

4.1 Alloy code

```
//************CLASS DEFINITIONS************//

//assuming date as an int dd-mm-yy
//assuming Int as an int hour,minute
//definition of position
sig Position{
  x: one Int,
  y: one Int,
}

sig Text{}

abstract sig Boolean{}

one sig True extends Boolean{}
```

```
one sig False extends Boolean{}

//definition of a generic person
abstract sig Person{
  idDoc: Int,
  currentPosition: lone Position,
}

//definition of a user
sig User extends Person{
  email: Text,
  password: Text,
}

//definition of a taxi driver
sig Driver extends Person{
  code: Int,
  license: Int,
  seatsAvailable: Int,
  availability: one Boolean,
}{
        availability = True =>( some p:Position | p = currentPosition)
}
//definition of a ride
sig Ride{
  driver: one Driver,
  date: one Int,
  startingPoint: lone Position,
  destinations: some Position,
  users: some User,
  numberOfPeple: one Int,
  isShared: one Boolean,
  startingTime: one Int,
  requests: set Request,
}{
  date > 0
  all d: destinations | d != startingPoint
  numberOfPeple >= 1
  startingTime > 0
  isShared = False => #users = 1 and #destinations = 1 and #requests = 1
}
```

```
//definition of a request
sig Request{
  date: Int,
  startingPoint: Position,
  destinationPoint: Position,
  user: User,
  people: Int,
  acceptSharing: Boolean,
  requestTime: Int,
}{
  date > 0
  requestTime > 0
  people >= 1
  startingPoint != destinationPoint
}

//definition of the reservation
sig Reservation extends Request{
  prenotationTime: one Int,
  prenotationDate: one Int,
}{
  prenotationTime >0
  prenotationDate >0
}

//definition of the queue
sig Queue{
  zone: one Text,
  drivers: some Driver,
}

//definition of the queue manager
one sig QueueManager{
  queues: set Queue,
}

//************CONSTRAINTS************//

//uniqueness of the user by the email
fact uniqueUser{
```

```
  no u1,u2 :User | (u1 != u2 and u1.email = u2.email)
}

//uniqueness of the taxi driver by the double code, taxi license
fact uniqueDriver{
  no t1,t2 :Driver | (t1 != t2 and (t1.code = t2.code or t1.license = t2.
      license))
}

//number of available seats >= number of passengers
fact rightPassengersNumber{
  all r: Ride | r.driver.seatsAvailable >= r.numberOfPeple
}

//no ride without destinations or user
fact existanceDestinationAndUser{
  all r:Ride | #r.destinations > 0 and #r.users > 0
}

//constraints on availability
fact taxiAvailbleInQueue{
  all t: Driver | ((t.availability = True) <=>(isInQueue[t]) ) or ((t.
      availability = False) <=> (!isInQueue[t]))
}

//all queue are managed by the queue manager
fact queuesAllStoredInManager{
  all q:Queue | q in QueueManager.queues
}

//each driver is in at most in one queue
fact driversAtMostOneQueue {
  all t: Driver | (lone q: Queue | t in q.drivers)
}

//shared ride has at least two users
fact sharedRide{
  all r: Ride | ((r.isShared = True ) =>(#r.users > 1 and #r.destinations >=
      1 ) )
}
//no existance of user in a ride that hasn't request
```

```
fact allUserOneRequestEachRide{
        all r :Ride | (all u: r.users | one r1: r.requests| r1.user = u )
}

//shared ride has at least two requests
fact sharingAtLeastTwoRequest{
  all r: Ride | r.isShared = True => (some r1,r2: r.requests | r1.user != r2.
       user) and #r.requests >= 2
}

//number of total required seats of a shared ride has to be lesser or equal
     than the available seats
fact sharedSeats{
  all r : Ride | r.isShared = True => ((sum r1:r.requests | r1.people) <= r.
       driver.seatsAvailable)
}

//shared ride only for who accepts sharing
fact sharingCorresponding{
        all r: Request | r.acceptSharing = False => (no ride:Ride | r in ride
             .requests and ride.isShared = True)
}

//all request at most in one ride
fact atMostOneRideForRequests{
        all r: Request | lone ride: Ride | r in ride.requests
}

//there isn't any ride with same time and date completed by the same driver
fact noSimultaneousRideBySameDriver{
  no r1,r2:Ride | r1!=r2 and r1.driver = r2.driver and  r1.date = r2.date and
       r1.startingTime = r2.startingTime
}

//starting point comes from the first request
fact determineStartingPoint{
        all r: Ride | some r1: r.requests | (no r2 : r.requests | r2.
             requestTime > r1.requestTime) => r.startingPoint = r1.
             startingPoint
}
```

```
// shared ride same starting point from requests
fact sharedOneStartingPoint{
        all r: Ride |  r.isShared = True => (all r1,r2: r.requests | r1.
             startingPoint = r2.startingPoint)
}

//************ASSERTIONS************//

// check on the shared ride by more than one user
assert moreUsersIfShared{
  no r: Ride | #r.users>1 and r.isShared = False
}

check moreUsersIfShared for 4

//all available driver must be in a queue
assert noTaxiNoQueue{
  no t:Driver | t.availability = True and (no q: Queue | t in q.drivers)
}

check noTaxiNoQueue for 4


//************PREDICATES************//


// if the driver is in a queue
pred isInQueue[t:Driver]{
 some q:Queue | t in q.drivers
}

//same time and date of a request
pred isSimultaneous[r1: Request, r2: Request]{
  r1.date = r2.date and r1.requestTime = r2.requestTime
}

//same starting point of a request and same destination point
pred isSameStartAndDestination[r1: Request, r2: Request]{
  r1.startingPoint = r2.startingPoint and r1.destinationPoint = r2.
       destinationPoint
}
```

```
pred display(){}

pred show(){
  #User > 1
  #Ride > 1
  #Driver > 1
  #Request > 1
  #Queue > 1
  #Reservation >= 1
  #{r: Ride | r.isShared = True} > 1
}

pred exactlyOneSharing(){
        #User>1
        #Driver=3
        #Queue>1
        #Reservation = 1
        #Request >1
        #Request < 4
        #{r: Ride | r.isShared = True} = 1
}

pred noShare(){
        #{r: Ride | r.isShared = True} = 0
        #User > 1
        #Driver > 1
        #Driver < 4
        #Request > 1
        #Ride > 1
        no u: User |  (all r: Request | u in r.user)
        #{r1: Request | r1.acceptSharing = True } = 0
        #Position >1
        #Position < 5
}

run display for 4
run show for 5
run exactlyOneSharing for 10
run noShare for 7
```

4.2 Alloy response

```
6 commands were executed. The results are:
   #1: No counterexample found. moreUsersIfShared may be valid.
   #2: No counterexample found. noTaxiNoQueue may be valid.
   #3: Instance found display is consistent.
   #4: Instance found show is consistent.
   #5: Instance found exactlyOneSharing is consistent.
   #6: Instance found noShare is consistent.
```

4.2.1 Alloy Worlds

More than one world was created to show different kind of behaviours of the program

4.2.1.1 World 1

In world 1, created through the predicate “noShare” we forced the world to avoid the sharing of rides and to have a significant amount of Actors. Also, the world accepts the case in which a Reservation/Request is not connected to a ride because of some errors in the creation by the User. The world can be seen in Figure 4.1.

4.2.1.2 World 2

In world 2, created through the predicate “exactlyOneSharing” we forced the world to have one shared Ride by two users and a reservation. Also in this case the predicate is consistent and complies to the given requests. The world can be seen in Figure 4.2.

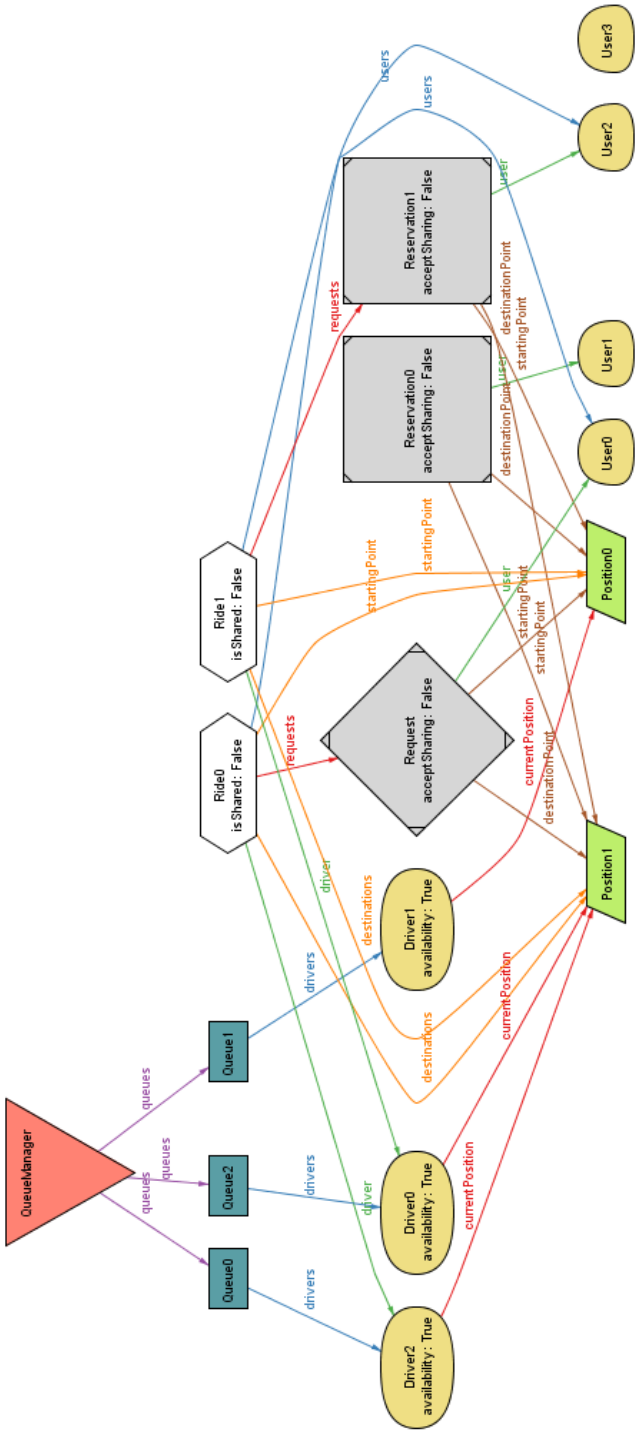

Figure 4.1: Alloy world 1

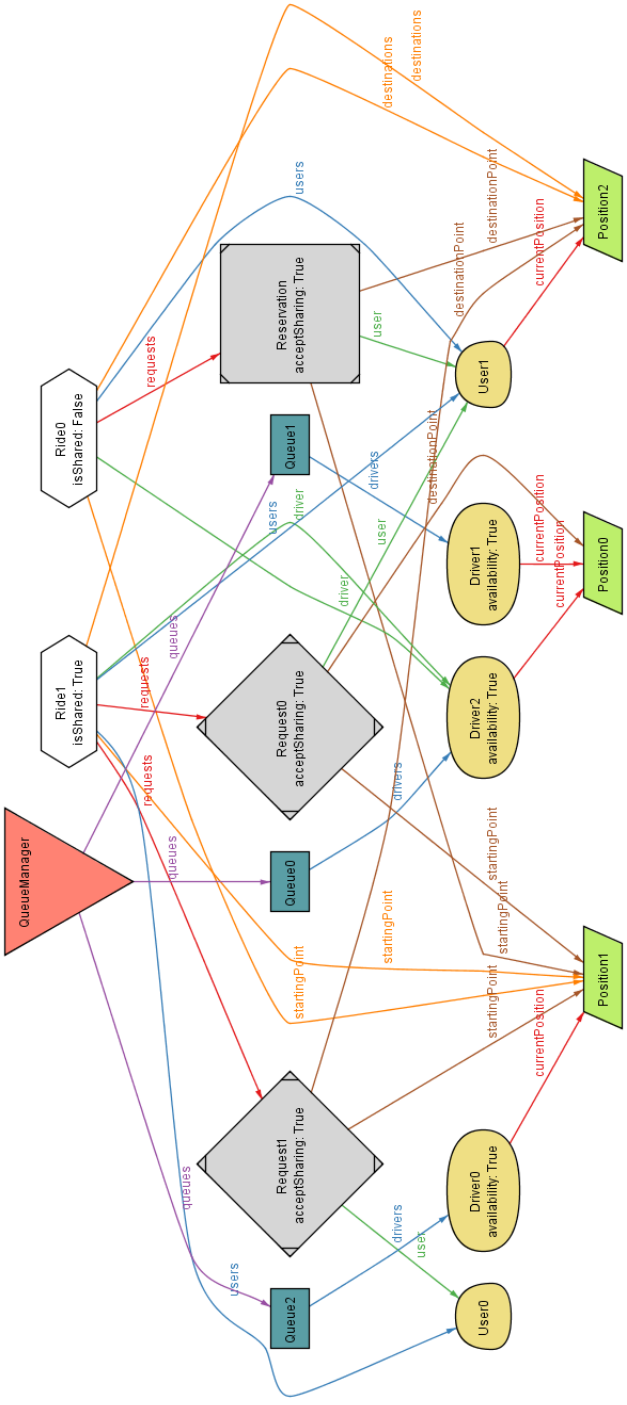

Figure 4.2: Alloy world 2

4.2.2 Alloy metamodel

The metamodel shows the interactions between signatures in Alloy. We decided to show this diagram too as a comparison with the class diagram which has the same classes. The only considerable exception is "Position". We purposely avoided to add it in the class diagram as we don't see the necessity of it in the RASD document while it was necessary in the Alloy model as it helps defining some of our constraints.

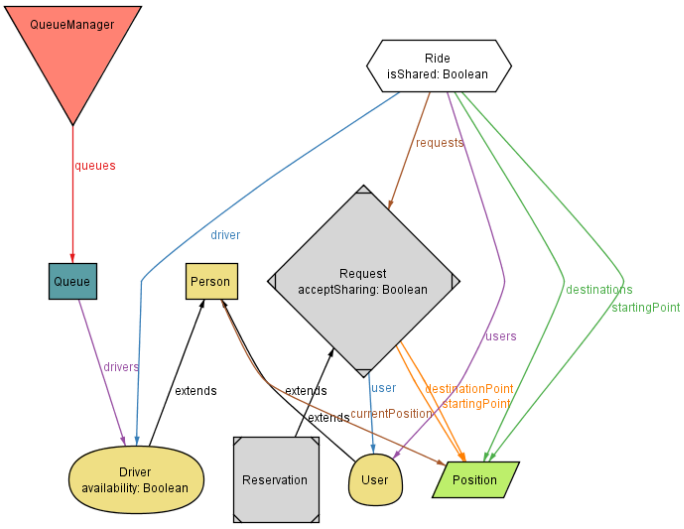

Figure 4.3: Alloy metamodel

Chapter 5

Other information

This chapter presents information about the developement and the redaction of this document.

5.1 Working hours

Giorno	Alessandro	Davide	Stefano
22/10/2015	1	1	-
23/10/2015	2	2	2
26/10/2015	3	3	3
28/10/2015	3	3	3
30/10/2015	2	3.5	2
02/11/2015	3	1	2
03/11/2015	-	3	-
04/11/2015	4	2	4
05/11/2015	2	4	3
06/11/2015	4	4	4
Total hours	24	26.5	23

5.2 Used tools

In order to redact this document the following tools have been used:

1. LaTeX and LyX editor
2. Alloy Analyzer
3. Google Drive
4. GitHub
5. StarUML for diagrams