Requirements Analysis and Specification Document

myTaxiService

Luca Nanni (850113) Giacomo Servadei (854819)

November 5, 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

This is the Requirements Analysis and Specification Document to be used in the design of the software system called *myTaxiService*. In this document we describe the specifications and constraints that the software we are to implement must have. The intended audience of this paper is:

- The project manager

- The client which in this case is the Government of the city

- The designers and developers of the application

- The testing team

- The end user

This document has contractual value.

1.2 Scope

Here we summarize the main scope of the application.
The client is the *government of a large city*.
This city already offers a taxi service to its citizens, but the government wants to improve it using a modern and efficient information system.
So we received the request to design and implement this application, called *myTaxiService* which has basically two great objectives:

- Simplify the usage of the taxi service

- Guarantee an efficient management of the taxi queues

1.2.1 Users and interfaces of the application

The system is designed to interact with two kind of users:

- The clients (Passengers) of the Taxi service

- The taxi drivers

For each category of users we must provide an appropriate interface and the client requests for the interfaces are:

- A web application or a mobile application for the clients (Passengers)

- A mobile application for the Taxi Drivers

Moreover, it is required to implement a software interface that allows developers to extend the system with additional taxi services.

1.2.2 High level behavior

Here we report the exact description of the problem the government of the city supplied to us:

Passengers can request a taxi either through a web application or a mobile application. The system answers to the request by informing the passenger about the code of the incoming taxi and the waiting time.

Taxi drivers use a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain request.

The system guarantees a fair management of taxi queues.
In particular the city is divided in taxi zones (approximately $2km^2$ each). Each taxi zone is associated to a taxi queue.
The system automatically computes the distribution of taxi in the various taxi zones based on the GPS information it receives from each taxi. When a taxi is available, its identifier is stored in the taxi queue in the corresponding taxi zone. When a request arrives from a certain taxi zone, the system forwards it to the first taxi queuing (in the taxi queue) in that taxi zone. If the taxi confirms (the request), then the system will send a confirmation to the passenger. If not, then the system will forward the request to the second in the taxi queue and will, at the same time, move the first taxi in the last position in the taxi queue.

A passenger can reserve a taxi by specifying the origin and the destination of the ride. The reservation has to occur at least two hours before the ride. In this case, the system confirms the reservation to the passenger and allocates a taxi to the reservation 10 minutes before the meeting time with the passenger.

Beside the specific user interfaces for passengers and taxi drivers, the system offers also programmatic interfaces to enable the development of additional taxi service (e.g. taxi sharing) on top of the basic one.

1.3 Definitions

We present here the main glossary of the application domain, derived from the client specification already reported:

Term	Description
Available	Status of a Taxi Driver in which he can receive requests
Busy	Status of a Taxi Driver in which he is serving a Passenger's request
Government of the city	It is the client for which we are working. It desires an application for the improvement and simplification of the taxi service.
City	The ambient in which the taxi drivers and the passengers interact. It is divided in taxi zones.
GPS	Technology which manage to get in every moment the position of a vehicle
Meeting point/location	It is the location inside the city at which the passenger and the taxi driver will meet. It is set by the Passenger during the reservation procedure
Meeting time	It is the time at which the passenger and the taxi driver will meet. It is set by the Passenger during the reservation and request procedures
Mobile application	It is one of the interface that passengers and taxi drivers can use to interact with the system. To use it they must have it installed in their smart phone.
Not available	Status of a Taxi Driver in which he cannot receive requests
Passenger	One of the user of the system. He can apply for a taxi service: he can make a request or a reservation
Programmatic interface	It is a software interface to be used by developers to modify and extend the actual software. It is useful for the extension of the application with additional taxi services
Queue management	It is the part of the system that takes care of the organization of the taxi drivers inside the taxi queue of the correspondent taxi zone
Request	It is the action carried out by the passenger when he needs to use the taxi service. It represents an immediate need of the passenger
Reservation	It is the action carried out by the passenger when he needs to use the taxi service in the future. It consists in the specification of the origin and the destination of the taxi ride, reserved at a desired time. The passenger can reserve a journey only if the specified time is at least two hours after the time of reservation.
System	It is the application we have to design. It is constituted by different interfaces: two for the Passengers, one for the Taxi Drivers and one for the future extension of the application (programmatic interface).
Taxi driver	One of the user of the system. They are the people which task is to drive the taxi. They use the application to communicate their availability or not and to be assigned to the taxi queues.
Taxi queue	It is an abstract queue of the available taxis in a taxi zone of the city. It has an order determined by the time in which each taxi driver communicate its availability to the system or by the decision of the taxi driver to accept a request or not

Taxi service	It is a service offered by the government of the large city which enable every person to use the offered taxis. It can be of different nature as a reservation service or a request.
Taxi zone	The city is divided in taxi zones, each one having its taxi queue. The division is provided in order to distribute the taxi availability in all the city territory
Web application	Is one of the interfaces that the passengers can use to interact with the system. To use it the passenger must use a web browser.

Table 1.1: Glossary

1.4 References

- IEEE Std 830-1998: *IEEE Recommended Practice for Software Requirements Specifications*

- Assignment 1 document

1.5 Overview

In the next sessions of this document we will discuss about:

1. (Chapter 2) **Overall Description**

2. (Chapter 3) **Specific Requirements**

Chapter 2

Overall Description

2.1 Product perspective

myTaxyService will not be integrated with any other systems, it will be self-contained.

It will be implemented in a three-tier architecture. Server-side, on the application layer there will be the Queue Manager (will take care of managing taxi queues in the various zones) and the Reservation System (will allow taxis to be reserved in advance). A dedicated server will host the database in which the system will store all the data.

Passengers will be able to access the system after a registration and a following login from both a web application and a mobile application (available for free for the three major mobile operating systems, Android, IOS, Windows Phone from their markets).

Taxi drivers instead, will use a different version of the application that will not be available on the markets but will be provided by the government upon a request. For future improvements some API will be developed and made available.
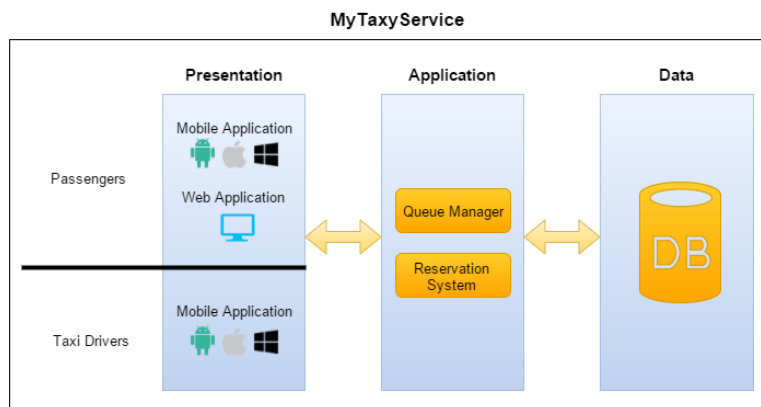
Figure 2.1: Schematic representation of *MyTaxyService*

2.2 Product functions

The system offers the following functions with respect to the different users (Passengers and Taxi Drivers).

2.2.1 Functions for Passengers

A Passenger can:

- Register to the service

- Request a taxi for an immediate necessity

- Reserve a taxi for a future necessity
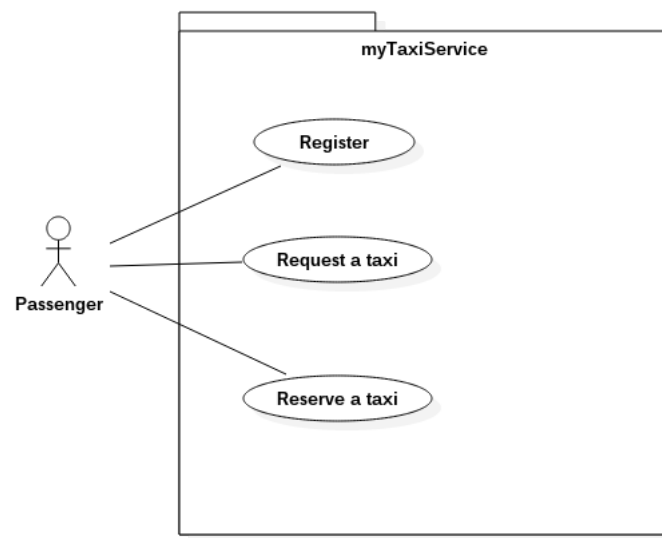
2.2.2 Functions for Taxi Drivers

A Taxi Driver can:

- Start his working activity by putting himself as available

- Accept an incoming request from a Passenger, both requests and reservations

- Refuse an incoming request from a Passenger, both requests and reservations

- Finish his working activity by putting himself as not available
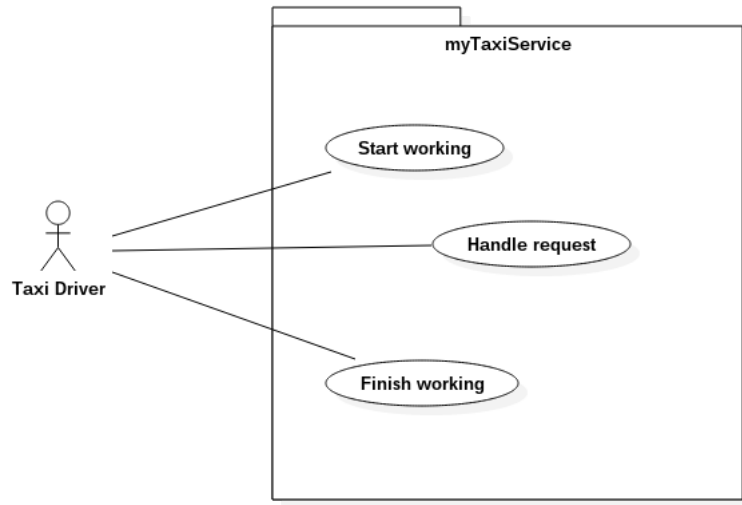
2.3 User characteristics

The application addresses two specific kinds of users: *Passengers* and *Taxi drivers*

2.3.1 User class 1: *Passengers*

A Passenger is a user with these characteristics:

- He owns a smart phone

- He is able to install a mobile application on his smart phone

- He owns/has access to a computer

- He has access to the Internet

- He is able to use a web browser

We can consider a passenger every person who wants to use the taxi service of the large city. They can be citizens of the city, tourists or simply visitors.

2.3.2 User class 2: *Taxi drivers*

A Taxi Driver is a user with these characteristics:

- He is a employee of a taxi enterprise having its location in the city or he works by his own.

- The taxi enterprise has registered his taxi drivers to the system or, in case of a freelance taxi driver, he has already done the registration procedure (external to the software to be)

- He owns/has access to a smart phone (maybe provided by the enterprise)

- He is able to install a mobile application on his smart phone

Taxi Drivers need to respond to every ride request in the shortest time possible. In a large city like the one we are working for this means that their time schedule is always full of requests during their working hours.

2.4 Constraints

2.4.1 Regulatory Polices

The software behavior must respect all the prescription of the local law about:

- Security and Integrity of the users data (both Passengers and Taxi Drivers)

2.4.2 Hardware limitations

The system must be interfaced through two kind of mobile applications (one for the Passenger and one for the Taxi Driver) and a web application.

Mobile applications

Both applications must be developed for the three major mobile Operating Systems (Android, iOS, Windows Phone). The passengers version must be released in the three specific application market for free. The taxi driver version, instead, will only be available upon a request to the government. The applications have to request the minimum amount of authorizations to the smart phone Operating System and of course they must not damage/modify unrelated data stored in the device.

Web application

The web application must be supported by all the most famous web browsers: in particular Google Chrome, Safari, Mozilla Firefox and Internet Explorer.

2.4.3 Parallel Operation

The system must be able to deal with multiple contemporary requests coming from different Passengers. The statistics on the daily usage of the taxi service of the city show that there will be at least 5/6 requests per second during the more congested hours of the day.
The system must provide the right grade of parallelism.

2.4.4 Criticality

The system does not present any critical application. The Passengers are supposed not to use the taxi service for life-critical purposes.

2.5 Goals of the application

From the analysis of client's needs for the software to be we can derive the following list of goals.

2.5.1 Goals Passenger side

A Passenger:

1. must be able to register to the service

2. must be able to login to the service

3. can access *myTaxyService* either from the web application or the mobile application

4. must be able to request a taxi ride from a location he decides

5. must be able to reserve a taxi ride from an origin, to a destination at a specific time and date

6. must be able to cancel a reservation that he made

7. must receive, if a taxi is available for a request, a notification stating the arrival time and the taxi identification number

8. if he has reserved a taxi correctly for a certain meeting time, he must be notified before that time with a notification saying the arrival time of the taxi and it identification number, or, in case of problems, saying the nature of the problem.

2.5.2 Goals Taxi Driver side

A Taxi Driver:

1. must be able to set himself as available (start working)

2. must be able to set himself as not available (stop working)

3. must receive requests (both immediate and reservations) only if he is available

4. must be able to choose if he wants to accept an incoming request or not

5. must receive a notification if he is the first taxi in the queue corresponding to his actual location and if a passenger made an request in that zone

6. when available, must be set in the correct waiting queue

7. must receive the location of the passenger who made the request, if he accepted it

2.6 Derived behavior of the actors

From the domain analysis, the assumptions and the basic requirements for the application we can derive a model of behavior of the agents (Passengers and Taxi Drivers) when they use the future application

2.6.1 Passenger behavior
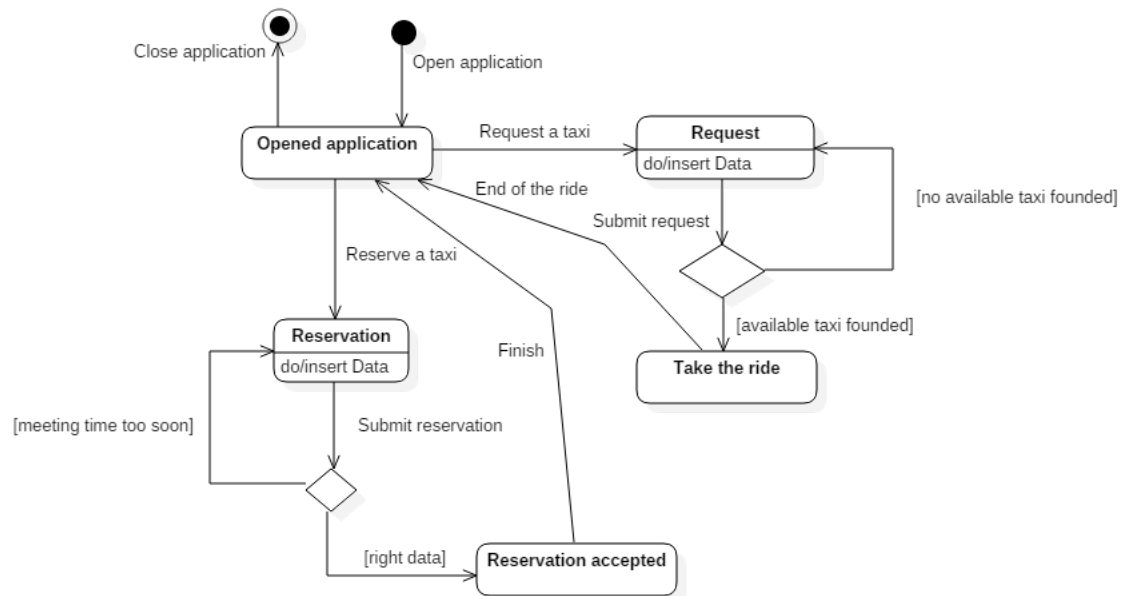
2.6.2 Taxi Driver behavior
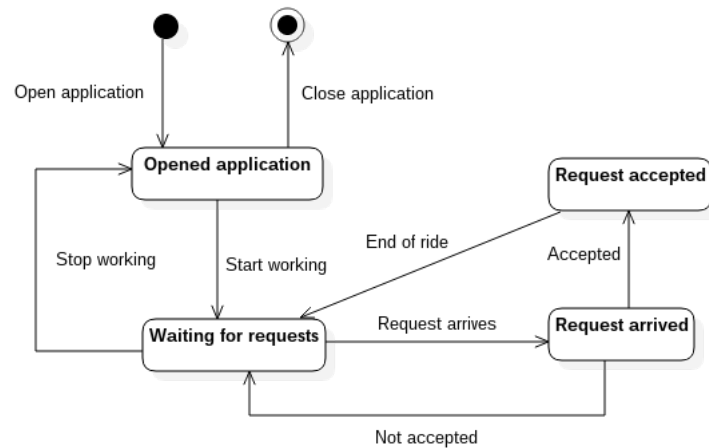
2.7 Assumptions and dependencies

These are our assumptions about the problem domain:

- Each smart phone used by the Taxi drivers has a GPS system installed and the application has the privileges to access it

- Once a passenger made a request and it has been accepted by a taxi, he's going to wait for it until it arrives and he will take the ride

- The taxi drivers will set their availability only when they can accept new passengers

- The Passenger, once ended the ride, will pay the Taxi Driver directly. There is no support for an in-app payment system

- The Passengers need to register to the system (username + password) in order to use it

- Passengers can't access the mobile application used by the Taxi Drivers

- The Taxi drivers will register to the system with a specific request to the Government of the city. There is no support for taxi driver registration in the system.

- Passengers, once submitted a request, will never try to cancel it.

- The Passengers will make requests and reservations only from a location inside the city only to a location inside the city

- Taxi Drivers, once accepted a request, will always reach the meeting point

- There is absolutely no overlapping between taxi zones

- Each taxi driver has only one taxi: this means that each taxi driver has a fixed identification number for his taxi

2.8 Entities involved

Here we summarize the entities involved in the system to develop
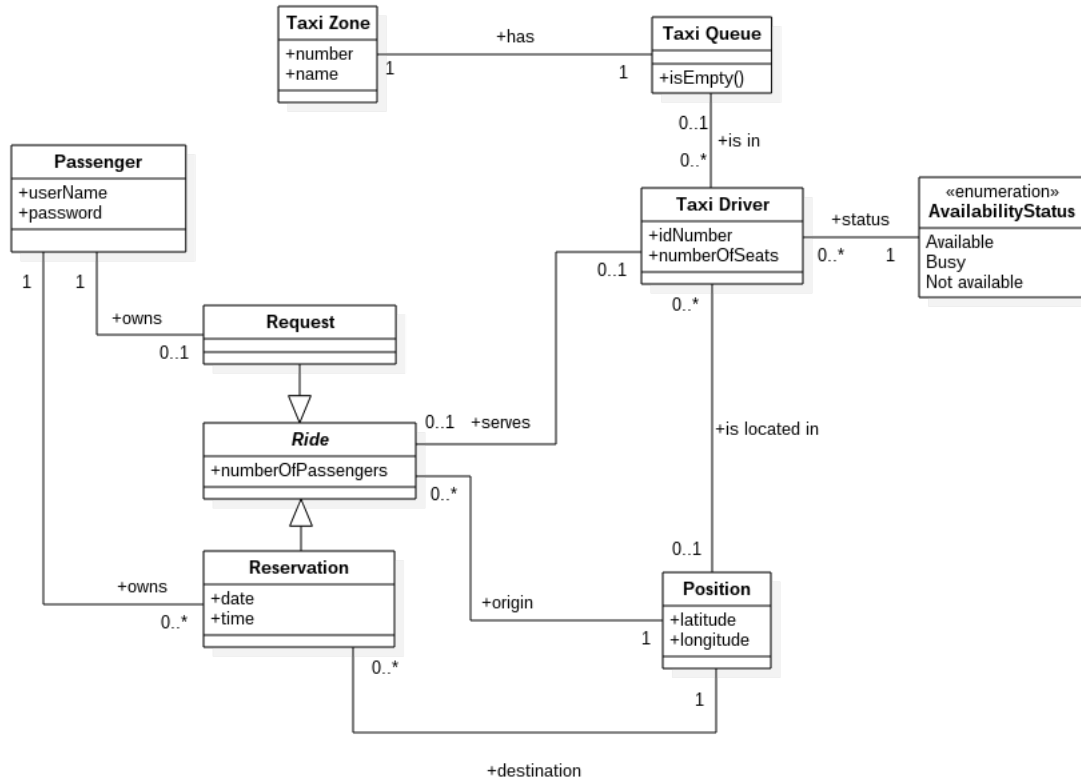
2.9 Future Possible Implementations

- The system will be able to suggest the taxi driver in which taxi zone they should go in order to have better chance to receive requests. These suggestions will be computed based on the statistics regarding the number of requests per hour in each zone.

- The system could also manage the payment of the ride through an in-app interface

- The system will have the possibility to handle the cancellation of the request after its submission to the system.

- The system will allow taxi drivers to accept reservations in advance. In this way, it is guaranteed that a reservation will be served at the right time, without delays.

Chapter 3

Specific Requirements

3.1 Scenarios

3.1.1 Registration to the service

Scenario

Bob has discovered the new service offered by the city called *myTaxiService* and he wants to discover how it works. He goes to the web application site and starts the registration phase.
He insert the username and the password he wants and submit the request of registration. The system replies saying that the username selected is already used and that Bob has to insert another username. Bob insert another username and this time the procedure succeed. Bob is now correctly registered to *myTaxiService*.

Use case

Use case	**Register**
Actors	Passenger
Goals	A passenger must be able to register to the service
Enter condition	None

Event flow	1. The passenger goes to the web application page
	
	2. The passenger clicks on the sign up button
	
	3. The passenger fills the form with username and password desired
	
	4. The passenger clicks the submit button
	
	5. If the username is already present in the system or there are missing data
	
	    (a) Notify the Passenger of the error
	    (b) Go back to Event flow 3
	
	6. The system retrieves the data and stores them
	
	7. The system notifies that the registration has been correctly done

Exceptions	If the user, during the insertion of the registration data (username + password) decides to abort the procedure by clicking on the proper button 1. The system notifies the passenger of the loss of the inserted data 2. The system abort the procedure
Exit condition	The passenger is correctly registered to the service

Table 3.1: Use case: Register

3.1.2 Passenger request

Scenarios

Positive response Tom has already downloaded the mobile application called *myTaxiService* and desires to request a taxi ride from his actual place, located in "Piazzale Gorini 18, Milano" to the "Stazione Centrale of Milano". He enters in the request section of its application, fills the form (source of the journey, number of passengers) and waits for a response from the service. After a few seconds the system responds indicating that the taxi with identification code T345 is arriving at the requested address. The approximate waiting time for the taxi is 5 minutes.

Negative response Tom has already downloaded the mobile application called *myTaxiService* and desires to request a taxi ride from his actual place, located in "Piazzale Gorini 18, Milano" to the "Stazione Centrale of Milano". He enters in the request section of its application, fills the form (source of the journey, number of passengers) and waits for a response from the service. Immediately the system responds indicating that there are no taxi available in the corresponding zone and Tom is invited to retry later.

Use case

Use case	**Request a taxi**
Actors	Passenger, Taxi Driver
Goals	A passenger must be able to request a taxi ride from a location he decides
Enter condition	- The Passenger must already be registered to the service - The Taxi driver must be registered to the service - The Passenger must already have opened the application (mobile or web) and logged in - The Taxi driver must be available

Event flow	
	1. The Passenger goes to the section for requesting a taxi ride
	2. The Passenger fills the form with:
	• Meeting location
	• Number of passengers
	3. The Passenger submit the request to the system
	4. If there are missing data or the specified location is not valid or the number of passengers is greater than 3:
	(a) The system notifies the error to the Passenger
	(b) Go back to Event flow 2
	5. The request is stored in the system
	6. The system checks the location provided by the Passenger and computes the corresponding zone in the city
	7. The system, based on the city zone computed, retrieves the corresponding taxi queue
	8. If there are no taxi in the queue:
	(a) Returns an error to the Passenger and invites him to retry later
	(b) Go back to Event flow 2
	9. The system takes the first taxi in the queue
	10. The system send the request to the taxi driver
	11. If the taxi driver does not accept the request:
	(a) The system puts the Taxi driver at the bottom of the queue
	(b) Go back to Event flow 9
	12. The system sends to the Taxi driver the location of the meeting point with the Passenger
	13. The system computes an approximate waiting time for the Passenger
	14. The system informs the Passenger of the forthcoming arrival of the Taxi and the approximate waiting time.
Exit condition	The passenger is waiting for the taxi driver and the taxi driver is reaching him

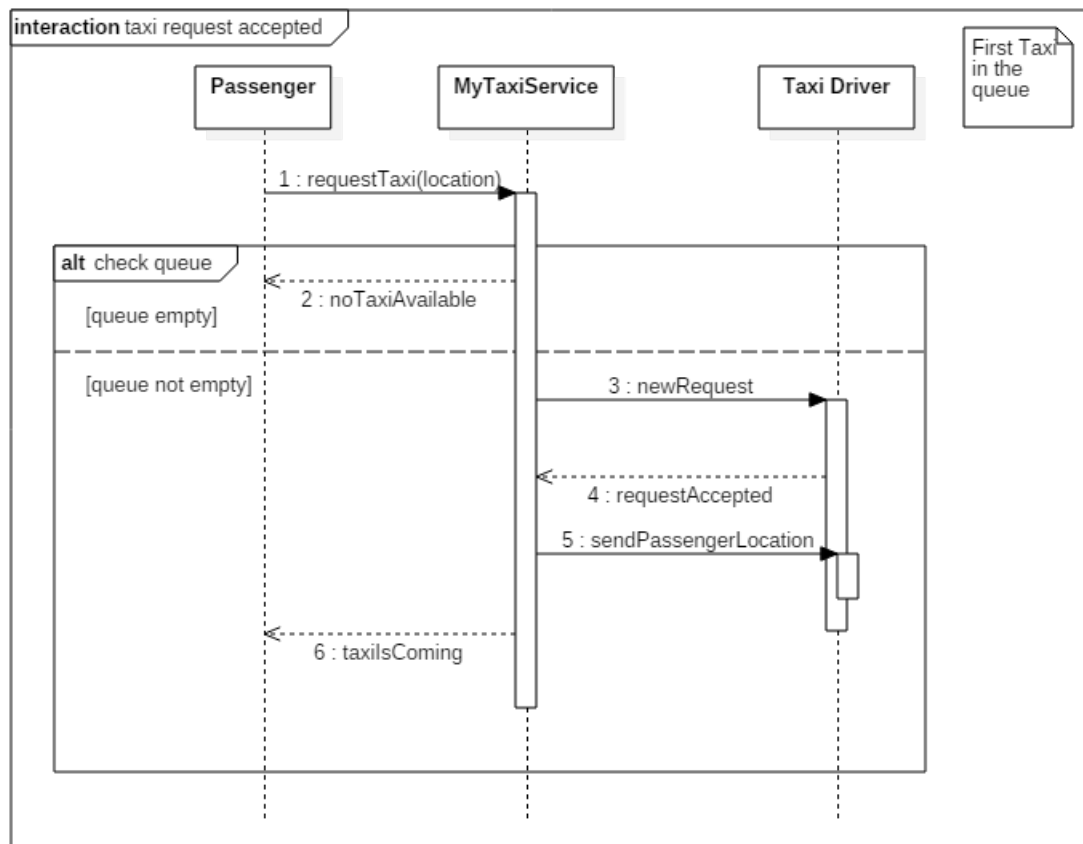Table 3.2: Use case: Taxi request from a Passenger

Sequence diagrams

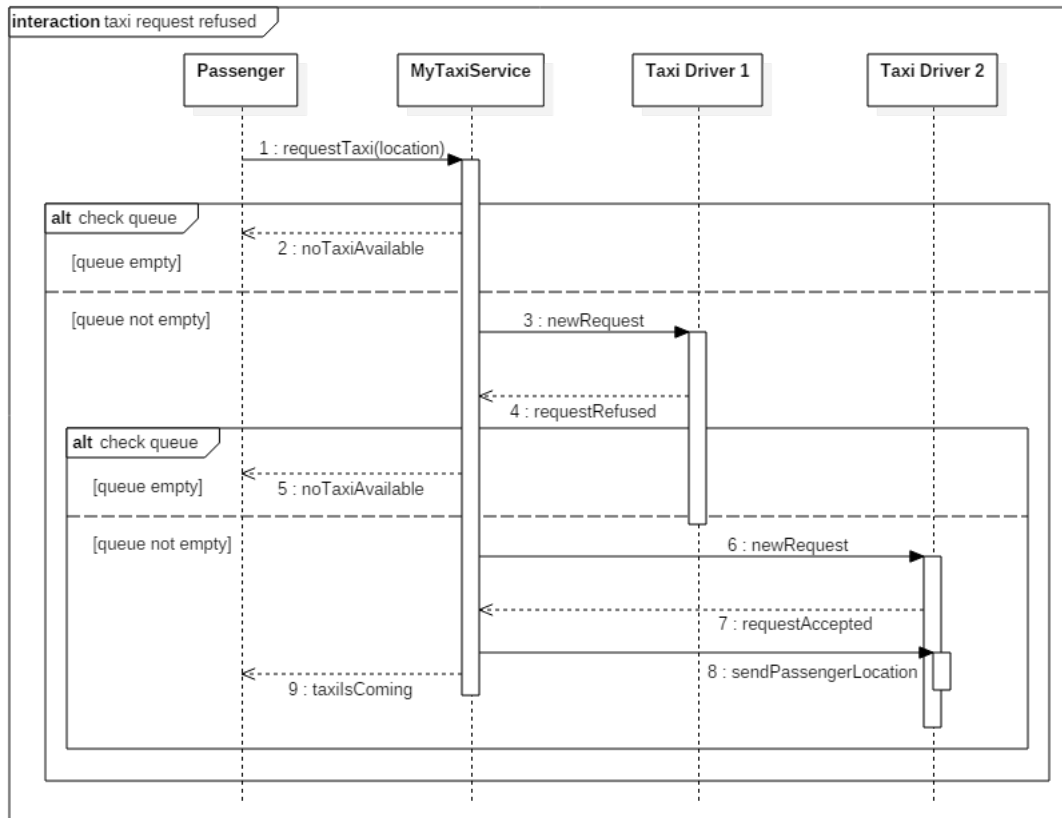Figure 3.1: An immediate positive response to the request of a Passenger

Figure 3.2: A complete positive response to the request of a Passenger

3.1.3 Passenger Reservation

Scenario

Positive Response It is 9.00 in the morning and Tom has already downloaded the mobile application called *myTaxiService* and he has a fixed appointment at 17.00 at "Stazione Centrale of Milano" so he decides to reserve a taxi for the 16.30 to go there. He enters in the apposite section of the application, indicates the source of the journey,the destination, the time of the meeting with the Taxi driver. He also specify the number of passengers that the Taxi will carry. The system, after a few seconds, reply positively to Tom saying that the reservation has been accepted and that the Taxi identification number will be provided 10 minutes before the meeting time.

Negative Response It is 16.00 in the morning and Tom has already downloaded the mobile application called *myTaxiService* and he has a fixed appointment at 17.00 at "Stazione Centrale of Milano" so he decides to reserve a taxi for the 16.30 to go there. He enters in the apposite section of the application, indicates the source of the journey,the destination, the time of the meeting with the Taxi driver. He also specify the number of passengers that the Taxi will carry. The system immediately responses that the reservation cannot be accepted because the meeting time specified is not at least 2 hours later.

Use case

Use case	**Reserve a taxi**
Actors	Passenger, Taxi Driver
Goals	A passenger must be able to reserve a taxi ride from an origin, to a destination at a specific time and date
Enter condition	- The Passenger must already be registered to the service - The Taxi driver must be registered to the service - The Passenger must already have opened the application (mobile or web) and logged in - The Taxi driver must be available

Event flow	
	1. The Passenger goes to the section for reserving a taxi ride
	2. The Passenger fills the form with: origin, location, date, time, number of passengers
	3. The Passenger submit the reservation to the system
	4. If there are errors with the data (missing fields, not valid locations or number of passengers greater than 3) or the date and time provided are not 2 hours in advance:
	(a) The system notifies the error to the Passenger
	(b) Go back to Event flow 2
	5. The reservation is stored in the system
	6. The system waits until 10 minutes before the date and time of the reservation
	7. The system checks the origin location provided by the Passenger and computes the corresponding zone in the city
	8. The system, based on the city zone computed, retrieves the corresponding taxi queue
	9. If there are no taxi in the queue:
	(a) Wait until there is at least one taxi driver in the queue.
	(b) Go back to Event flow 9
	10. The system takes the first taxi in the queue
	11. The system send the request to the taxi driver
	12. If the taxi driver does not accept the request:
	(a) The system puts the Taxi driver at the bottom of the queue
	(b) Go back to Event flow 9
	13. The system sends to the Taxi driver the location of the meeting point with the Passenger
	14. The system computes an approximate waiting time for the Passenger
	15. The system informs the Passenger of the forthcoming arrival of the Taxi and the approximate waiting time.

Exceptions	The passenger can abort the procedure only during the waiting phase. If he decides to abort the procedure:
	1. If the time of cancellation of the reservation is later than 10 minutes before the meeting time:
	(a) The system notifies the Passenger of the impossibility to cancel the reservation
	2. The system cancel the reservation
	3. The system notifies the Passenger of the successful deleting of the reservation
Exit condition	The passenger is waiting for the taxi driver and a Taxi driver is reaching him

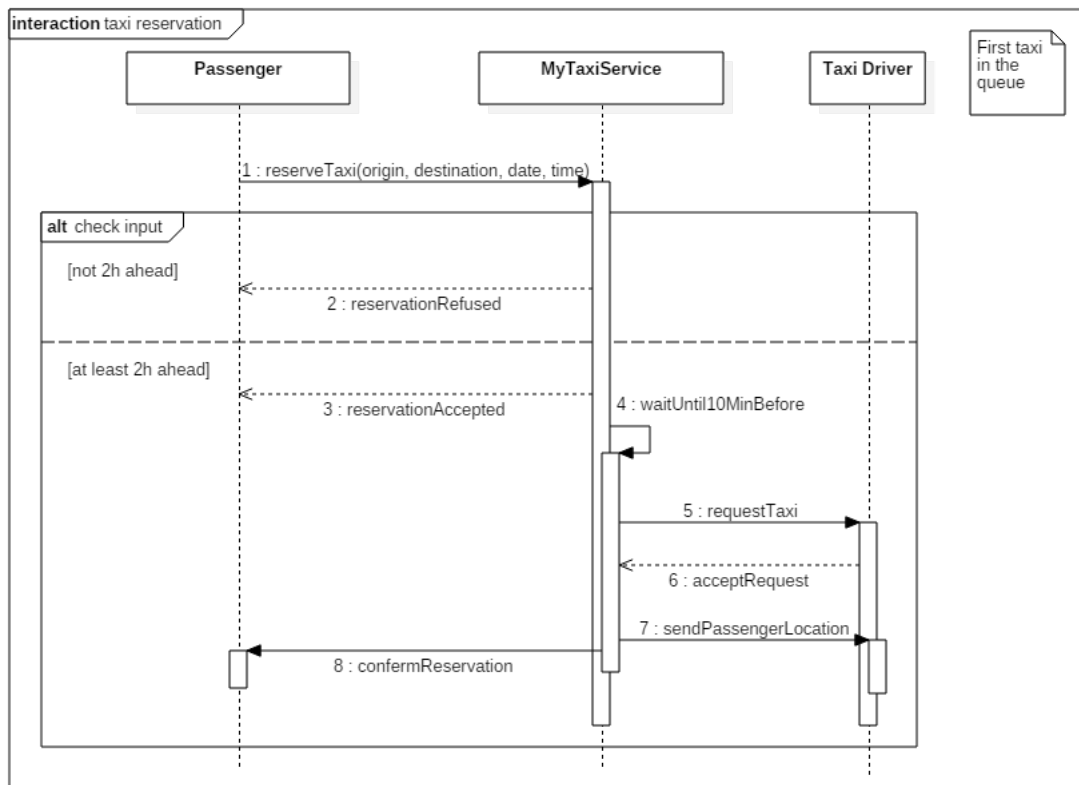Table 3.3: Use case: Taxi reservation from a passenger   +

Sequence diagram

Figure 3.3: Reservation of a taxi

3.1.4 Taxi driver: start of the working day

Scenario

Bob is a Taxi driver that works in the large city and has already installed the application *myTaxiService - Taxi Driver Edition*. He enter in his taxi and opens the mobile application. In the home page of the application he press the button which himself as available. The system automatically stores the identifier of his taxi in the queue corresponding at the location of the taxi (given by the GPS system of his smart phone). A message of the system says to Bob that his taxi has been inserted in a waiting queue and a waiting screen appears.

Use case

Use case	**Start working**
Actors	Taxi Driver
Goals	A Taxi Driver must be able to set himself as available (start working)
Enter condition	• The Taxi Driver must be already registered to the service • The Taxi Driver must already have opened the application
Event flow	1. The Taxi Driver press on the apposite button with the intention of setting himself as available 2. The system receives the updated status of the Taxi Driver 3. The system retrieves the location of the Taxi Driver from the GPS system 4. The system computes the actual zone of the Taxi Driver 5. The system puts the taxi driver at the bottom of the taxi queue relative to the computed taxi zone. 6. The system notifies the Taxi Driver and now he is waiting for a request
Exit condition	The Taxi Driver is waiting in a Taxi Queue

Table 3.4: Use case: Taxi Driver starts working

3.1.5 Taxi driver: acceptance of a request

Scenario

Bob is a Taxi driver that works in the large city and has already installed the application *myTaxiService - Taxi Driver Edition*. He is in the middle of his working day (so he has already executed the procedure at "Taxi driver: start of the working day") and he is waiting in a taxi queue corresponding to his actual location in the city.

Suddenly a pop-up shows up saying that there is a new request for a ride and he is asked to say if he wants to accept it or not. He accepts it and another screen opens specifying the location of the Passenger. The systems automatically sets the Bob's taxi as "busy": the taxi is removed from the taxi queue

Bob go to the location specified, picks up Alice and brings her to her desired location. At the end of the ride, Bob set himself as available. The system, based on his actual location, computes his taxi zone and puts him in the corresponding taxi queue.

Use case

Use case	**Accept a request**
Actors	Taxi Driver, Passenger
Goals	• A Taxi Driver must receive requests only if he is available • A Taxi Driver must be able to choose if he wants to accept an incoming request or not
Enter condition	The taxi driver must have already opened the application and must already be in a taxi queue

Event flow	
	1. The system receives a request from a Passenger, computes the correspondent taxi zone and retrieves the taxi queue
	2. The system picks the first taxi driver in the queue and forwards to the taxi driver the request
	3. If the taxi driver refuses to accept the request:
	(a) The system puts the taxi driver at the bottom of the queue
	(b) Go back to Event flow 2
	4. The system removes the taxi driver from the taxi queue and sets him as busy
	5. The system communicates the taxi driver with information about the meeting point
	6. The taxi driver reaches the meeting point and takes the passenger to the location he desires
	7. The taxi driver set himself as available
	8. Go to "Use case: Start Working", Event flow 2
Exit condition	A taxi driver is on his way to the meeting point with the passenger

Table 3.5: Use case: Accept a request

See also table 3.2 and table 3.3.
Sequence diagram

See figure 3.1, 3.2 and 3.3

3.1.6   Taxi driver: refusing a request

Scenario

Bob is a Taxi driver that works in the large city and has already installed the application *myTaxiService - Taxi Driver Edition*. He is in the middle of his working day (so he has already executed the procedure at "Taxi driver: start of the working day") and he is waiting in a taxi queue corresponding to his actual location in the city.
Suddenly a pop-up shows up saying that there is a new request for a ride and he is asked to say if he wants to accept it or not.
He refuses the request because he *wants to watch the world burn* (Batman citation). The system puts Bob at the bottom of the taxi queue and re-forwards the request to the new first taxi driver of the queue.

Use case

See table 3.2 and table 3.3

3.1.7 Taxi driver: end of working day

Scenario

Bob is a Taxi driver that is finishing his working day. He has just finished his last ride and the passenger has just left the taxi. He takes his phone with the application *myTaxiService* already running. He communicates to the system that he has ended the current ride. The system puts him in the taxi queue corresponding to his actual location.
Bob does not want to accept more requests and so he sets himself as not available. The system removes him from the queue and his taxi is set as not available.

Use case

Use case	**Stop working**
Actors	Taxi Driver
Goals	A Taxi Driver must be able to set himself as not available (stop working)
Enter condition	- The Taxi Driver must be already registered to the service - The Taxi Driver must already have opened the application - The Taxi Driver must be available
Event flow	1. The Taxi Driver press on the apposite button with the intention of setting himself as not available 2. The system receives the updated status of the Taxi Driver 3. The system removes the Taxi Driver from the queue 4. The system set the taxi driver as not available
Exit condition	The Taxi Driver is not available

Table 3.6: Use case: Taxi Driver stops working

3.2 Functional requirements

We divide the functional requirements of the application by the user class (actors) which are (mainly) involved in it:

3.2.1 Passenger

1. The system must not allow an already signed up Passenger to register himself (same username) again to the system

2. The username provided in the registration phase must not be empty

3. The password provided in the registration phase must not be empty

4. The system must notify the Passenger in case:

   - The username provided is empty
   - The password provided is empty
   - The username provided already exists in the system

   and must abort the registration procedure

5. The system must provide for the Passenger with a way to abort the registration procedure

6. If a passenger makes a request and the corresponding taxi queue is not empty, then the system must sooner or later respond positively to the Passenger

7. A passenger request must be refused if and only if there are no taxi driver available in the corresponding taxi queue or the number of passengers of the ride is greater than 3 [1].

8. A reservation must be refused if and only if:

   - Origin and destination are the same location
   - The number of passengers of the ride is greater than 3
   - time(meeting time) − time(reservation) < 2 hours

9. When the system looks for a taxi driver for serving a reservation (10 minutes before the meeting time), if the corresponding taxi queue is empty, it will wait until there is at least one taxi driver in the queue.

3.2.2 Taxi driver

1. A Taxi Driver, when he sets himself as available, must be put at the bottom of the taxi queue relative to his taxi zone.

2. A Taxi Driver, when he is available, must be in one and exactly one taxi queue.

3. A Taxi Driver, when he is not available, must not be in any taxi queue.

---

[1]We notice immediately that if all the taxi drivers in a queue refuse to accept the Passenger request, this will be forwarded again to the first taxi driver in the queue, who is the first one that refused it.
We think that this behavior is fair but can cause a loop and the Passenger could wait for a response a lot of time.

4. At each position of the Taxi Driver retrieved from the GPS data must correspond exactly one and only one taxi zone

5. A taxi queue must have a number n of taxi driver waiting in the range $n \in [0, +\infty[$

6. A Taxi driver must be always in one of this three states, which are mutually exclusive:

   - Available (see the glossary at Table 1.1)
   - Busy (see the glossary at Table 1.1)
   - Not available (see the glossary at Table 1.1)

7. The system must put the taxi driver at the bottom of the queue if:

   - He refuses a request
   - He does not respond to a request (accepting or refusing) within a certain time from the reception of it (10 seconds).

8. A taxi driver can receive requests if and only if he is at the top of a taxi queue.

9. A taxi driver waiting (available) in a taxi queue must receive requests only from passengers which specified a meeting point inside the correspondent taxi zone

10. When a Taxi driver accepts a request from a Passenger, he must be removed from the corresponding taxi queue and set as busy.

3.3 External interface requirements

3.3.1 User interfaces

Here we present the mock-ups of the application user interface.
We will present the user interface regarding only the two mobile applications (one for the Passengers and one for the Taxi Drivers).
The web application interface for the Passenger is a natural extension of the mobile one.

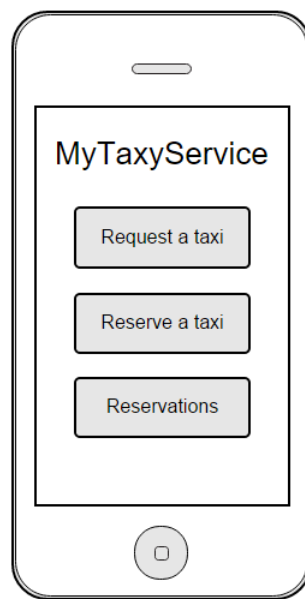The user interface must be as simple as possible: both the users must be able to use it immediately after the installation.

Home page for the Passenger

Figure 3.4: Home page for the Passenger

This screen will present to the Passenger the possibility to

- Make a request

- Reserve a taxi for a future journey

- Check his pending reservations

Request a taxi
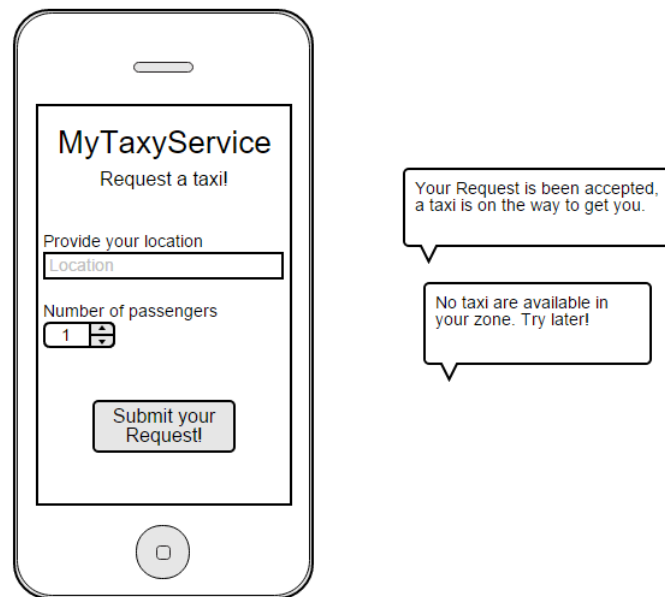
Figure 3.5: Make a request page

This screen is reached by pressing the button "REQUEST A TAXI" from the Home Page. The user can:

- Insert his location

- The number of passengers of the requested ride

- Submit the request

If one or more necessary information is not inserted (missing) the user will be notified with an appropriate pop-up.

Reservation of a taxi

Figure 3.6: Reservation of a taxi

This screen is reached by pressing the button "RESERVE A TAXI" from the Home Page. The user can:

- Specify his actual location

- Specify the destination of the journey

- Specify the date and hour of the meeting time

- Specify the number of passengers

- Submit the reservation request to the system

If one or more necessary information is not inserted (missing) the user will be notified with an appropriate pop-up.
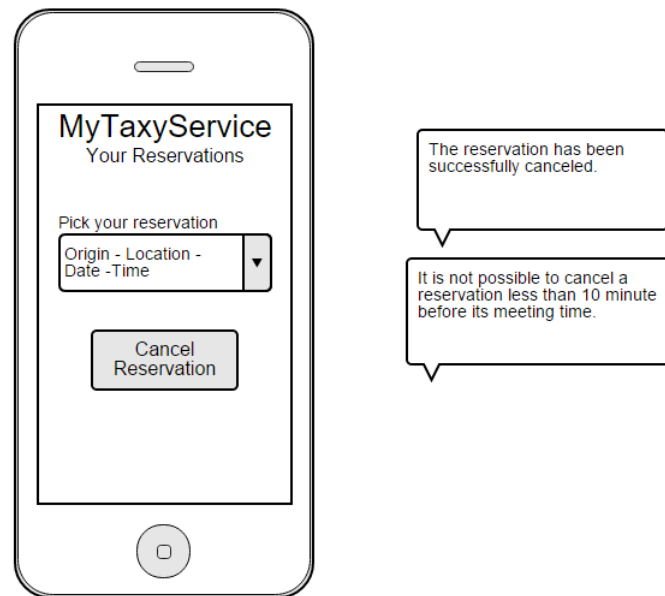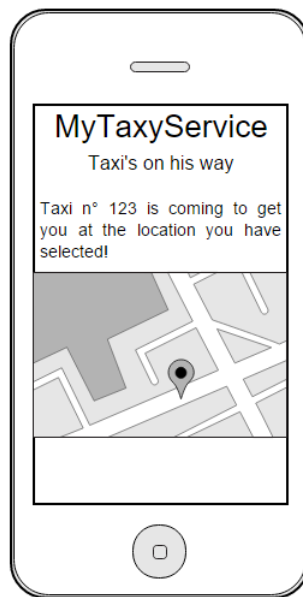
Reservations

Figure 3.7: Reservations page of a passenger

This screen is reached by pressing the button "RESERVATIONS" from the Home Page. The user can:

- View all the reservations

- Cancel a reservation

The permission of cancelling a reservation will be granted only if this is done at least 10 minutes before the meeting time. A pop-up will notify the passenger of the result of the operation.

Acknowledgement of a taxi request/reservation to a Passenger

This screen is reached when a taxi is allocated to the Passenger request, both in case of request and reservation.

The screen will present:

- Taxi identification number

- Waiting time

- Resume of the submitted data by the Passenger (meeting location, ecc...)

Start working day for a Taxi Driver

This is the first screen a Taxi Driver will see at the opening of the application:

Figure 3.8: First screen of the application for Taxi Drivers

In this page the Taxi Driver can:

- Set himself as available
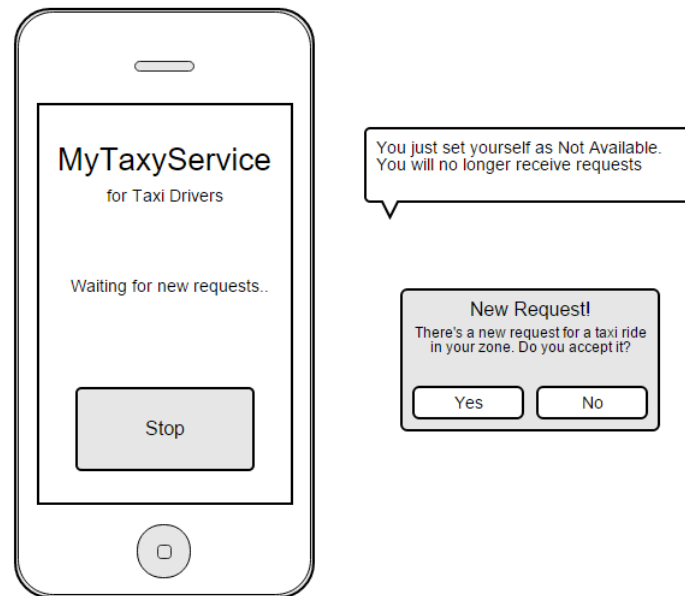
Taxi Driver's waiting for a requests

Figure 3.9: Waiting screen for a taxi driver

This screen is reached by pressing on the "START" button on the previous screen.
Once a request is submitted to the taxi driver the application notifies him with a pop-up saying that there is a request for a ride in the current zone.
It is asked to the taxi driver if he wants to accept the request or not.
Before receiving a request, the taxi driver can press the "STOP" button if he wants to set himself as not available. This action will bring him to the home page.

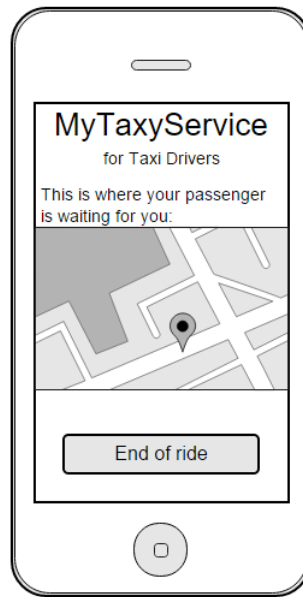**Communication of the location to a Taxi Driver**

Figure 3.10: Communication of the Passenger location to the Taxi Driver

This screen is reached by pressing the "YES" button on the pop-up displayed on the previous screen. It will be present a map indicating the position of the Passenger (meeting location). When the ride will be done the Taxi Driver will press the "END OF RIDE" button to notify his availability.
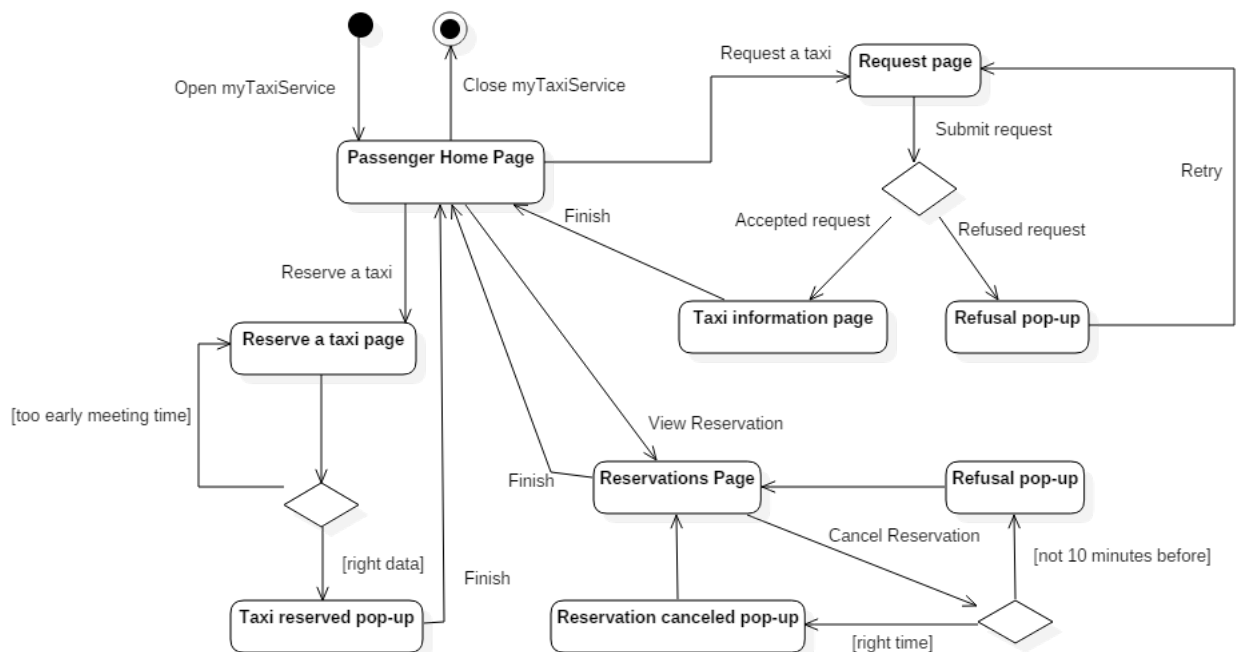
3.3.2 GUI state chart

We can summarize the behavior of the user interface through a state chart in which every state represents a specific screen of the application and each transition is basically a user input or a system communication to the application.

Taxi driver usage of the user interface
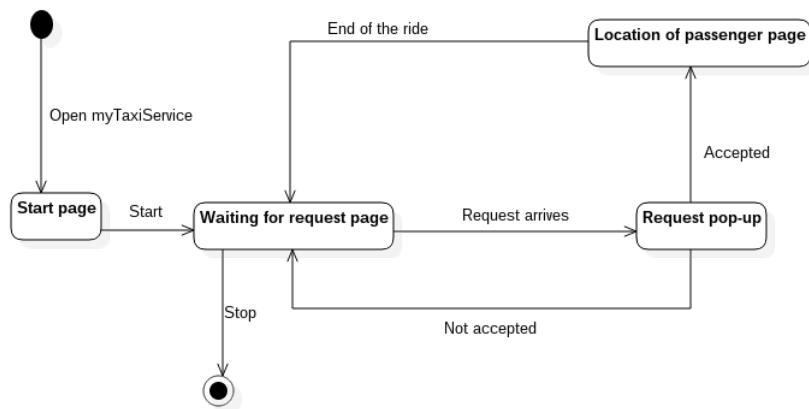
Passenger usage of the user interface

3.3.3 Software interfaces

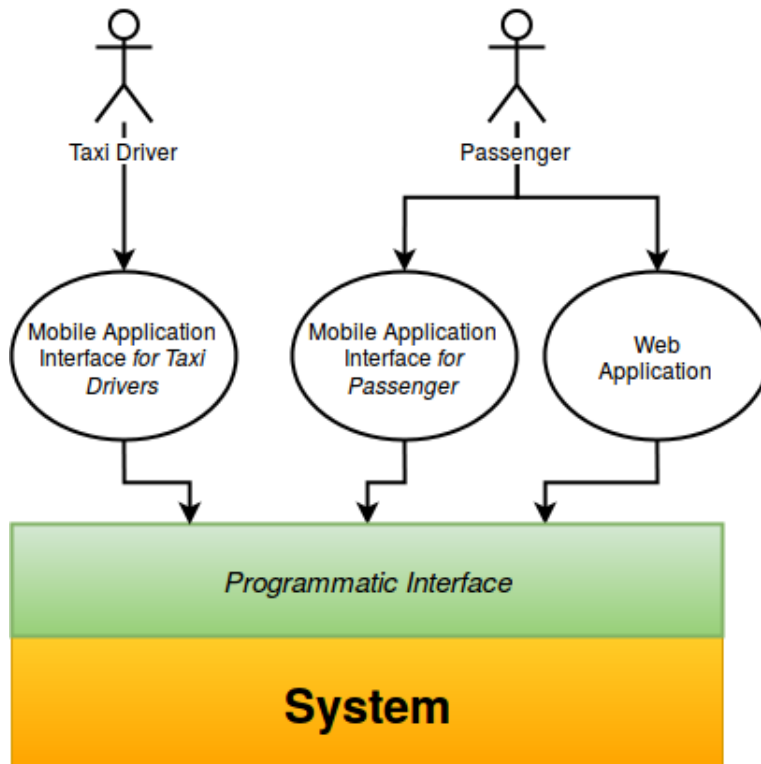The software to be must provide a *programmatic interface* to enable easy future extensions.

Figure 3.11: General schema of the architecture of the programmatic interface

Functional Requirements

1. The interface must have the possibility to add to the system new functionalities (for example: a taxi sharing function)

3.3.4 Communication interfaces

The only communication protocol used is HTTP.

3.4   Non Functional Requirements

3.4.1   Performance requirements

The performance of the web and mobile applications are strictly related on the performances of the internet connection of the devices they are running on. The interfaces must not cause any delay in the interaction.

3.4.2   Availability

The system must work 24/24 7/7 with a maximum fault time of 1 hour / week

3.4.3   Design constraints

We don't have any particular design constraint.

Chapter 4

Appendix

4.1 Alloy

For the evaluation of the model and the elicitation of the requirements we used the specification language *Alloy* which enabled us to express the structural and behavioral constraints of the software system *myTaxiService*.

```
/*
        ALLOY MODEL
        myTaxiService
*/

/******************ENTITIES***************************/
sig Passenger
{
}

sig TaxiDriver
{
        hasStatus: one Status
}

enum Status
{
        Available, NotAvailable, Busy
}

sig Position
{
}

sig TaxiZone
{
        hasQueue: one TaxiQueue
}
```

```
sig TaxiQueue
{
        contains: set TaxiDriver
}


abstract sig Ride
{
        ownedBy: one Passenger,
        isServedBy: lone TaxiDriver,
        origin: one Position
}

sig Request extends Ride
{
}

sig Reservation extends Ride
{
        destination: one Position
}

/*****************FACTS***************************/

//A Passenger can only have one request at a time
fact onlyOneRequestForPassenger
{
        all p: Passenger | lone r: Request | r.ownedBy = p
}


//To one taxi zone correspond exactly one taxi queue
fact queueInOnlyOneZone
{
        all q: TaxiQueue {one z: TaxiZone | q in z.hasQueue}
}


//A Taxi Driver can be in only one queue
fact taxiDriverInOnlyOneQueue
{
        all d: TaxiDriver {lone q: TaxiQueue | d in q.contains}
}
//All the taxi drivers that are in a queue are available and
//all the taxi that are available are in a taxi queue
fact allTaxiDriverInQueueAreAvailable
{
        all d: TaxiDriver |
        {one q: TaxiQueue | d in q.contains} iff d.hasStatus = Available
}
//A taxi driver can serve a ride only if he is busy and
```

```
//a ride can be served only by a busy taxi driver
fact taxiDriverHasRideOnlyIfBusy
{
        all d:TaxiDriver |
        { some r:Ride | r.isServedBy = d } iff d.hasStatus = Busy
}
//A Taxi Driver  can serve only one ride at time
fact onlyOneRidePerTaxiDriver
{
        all d: TaxiDriver |
        #{r: Ride | r.isServedBy = d} <= 1
}
//The origin and the destination of a reservation must be different
fact originDifferentFromDestination
{
        all r: Reservation | r.origin != r.destination
}




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

assert availableTaxiDriverInQueue {
        all d: {d1: TaxiDriver | d1.hasStatus = Available } |
         one q: TaxiQueue | d in q.contains
}


assert queueHasOnlyAvailableTaxiDriver
{
        all q: TaxiQueue {all d: q.contains | d.hasStatus = Available}
}



assert allRidesHasBusyTaxiDriver
{
        all r: {r1 : Ride | #(r1.isServedBy) = 1} |
         r.isServedBy.hasStatus = Busy
}

assert ifTaxiDriverBusyServesRide
{
        all d: {d1: TaxiDriver | d1.hasStatus = Busy} {
                one r: Ride | r.isServedBy = d
        }
}

assert onlyOneRidePerTaxiDriver
{
        all d: TaxiDriver | #{r: Ride | r.isServedBy = d} <= 1
```

47

```
}


check allRidesHasBusyTaxiDriver for 8
check ifTaxiDriverBusyServesRide for 8
check availableTaxiDriverInQueue for 8
check queueHasOnlyAvailableTaxiDriver for 8
check onlyOneRidePerTaxiDriver for 8




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

pred show
{
#Passenger > 1
#TaxiQueue > 1
#Request > 3
#{d: TaxiDriver | d.hasStatus = Available}>0
#{d: TaxiDriver | d.hasStatus = Busy}>0
#{d: TaxiDriver | d.hasStatus = NotAvailable}>0
}

run show for 5
```
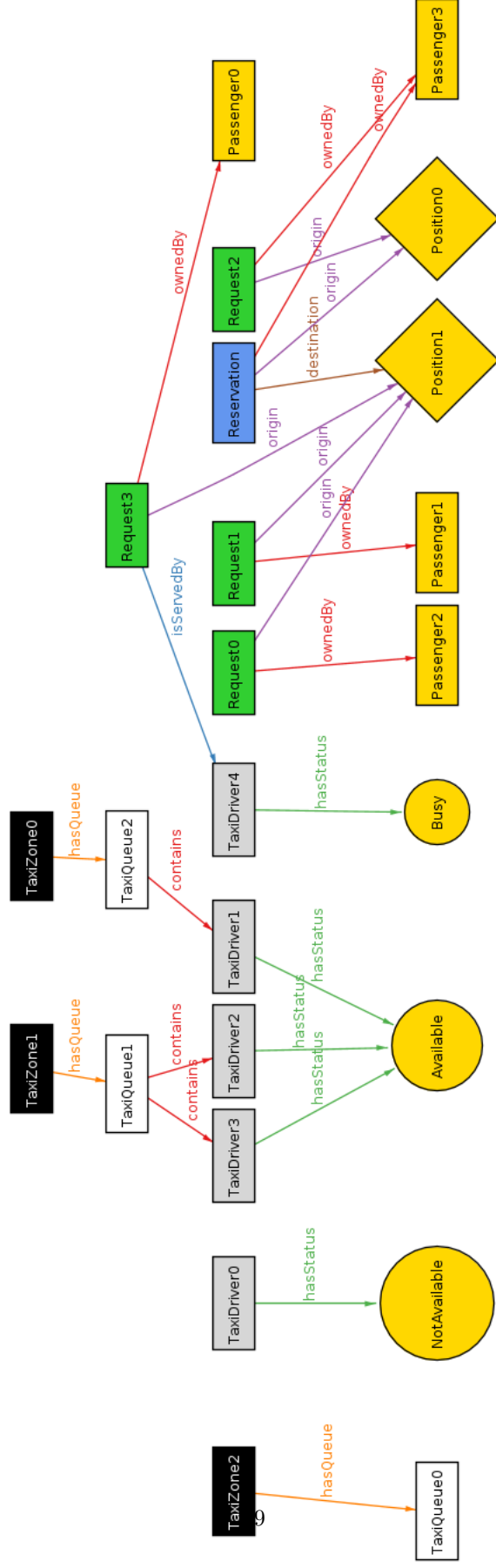
4.1.1 Notes about the model

Obviously we did not model all the requirements stated in the previous parts.
We mainly used Alloy in the first part of the elicitation of requirements in order to understand the **main constraints and relations** between the *entities* involved in our problem.
For these reasons the model above in of course incomplete and simplified. We are not interested, at this level of abstraction, in modeling every aspect of the system.

Our alloy model does not represent the history of the entities involved but a specific point of time. Of course, in the real application, is necessary to provide a support for the storing of information.

4.2 Working hours

This document was written in a total amount of around 40 hours divided so between the two group elements:

- Luca Nanni: 20 hours

- Giacomo Servadei: 20 hours

The two group elements were not in charge of a particular section/chapter but both worked together at the whole document.