Requirement Analysis and Specification Document

Michele Madaschi Lidia Moioli Luca Martinazzi

November 5, 2015

Contents

Introduction

1.1 Purpose

This document aims to describe, specify and analyze the software requirements for *My Taxi Service*.
My Taxi Service is needed to provide a passenger-friendly interface to interact with the city's taxi service, and ensure a fair management of the city-wide taxi deployment.

1.2 Scope

The passengers should be able to use an application (either mobile or browser based) to request a taxi through the system, which in turn should answer with the ETA and identification code of the incoming tax.
The passengers should be compelled to provide their current location to the system, for their request to be accepted.
The taxi drivers should be able to use a mobile application to communicate their availability to the system, and accept or refuse incoming calls.
The system shall manage a queue of taxis for each taxi zone[1].
The system shall receive GPS location data from each taxi, and use that information to assign each taxi to a taxi zone; an available taxi is automatically placed into the taxi queue belonging to the taxi zone it currently occupies.
The system shall remove a taxi from the queue upon receiving a confirmation in which the driver accepts an incoming call. If a taxi (driver) does, on the other hand, refuse an incoming call, the system shall move it to the bottom of its taxi queue.
The system matches a passenger's position to a taxi zone, and uses that information to forward the call to the first taxi available in the relative taxi queue.
The system shall provide an Application Programming Interface, to make room for future improvements.
The system shall also provide the possibility of requesting the reservation of

[1] Partition of the city

a taxi; said reservation must occur at least 2 hours before the actual time of the ride; the time of the ride has to be specified by the passenger during the reservation procedure, as well as the passenger's location and destination. However, the system will actually allocate a taxi (by means of removing it from the queue) only 10 minutes before the requested time of the ride.

1.3 Domain properties

In this section we will analyze the background laying behind My Taxi Service:

- Passengers will pay at the end of the ride the amount of money demanded by the taxi driver
- Passenger reserve a taxi only in a period of three month starting from the current time
- Taxi drivers must own a valid taxi driving license
- A taxi can reach every position within the same zone in less than 10 minutes
- ETA is estimated with a maximum error margin of 5 minutes
- We assume that, for each zone, if its queue is empty, the system must notify the unavailability of taxis
- We assume GPS coordinates reliable

1.4 Goals

The passengers must be able to:

G1 Transmit its position and the desired destination to the system, thus initiating the Request of a taxi

G2 Receive the code and the ETA of the incoming taxi

G3 Reserve a taxi for a time period, starting at the time specified during the reservation[2], and ending after the ride is complete.

G4 Request a shared ride

The taxi driver must be able to:

G5 Answer a passenger's request

[2]the starting of the reserved ride must occur at least 2 hours after the time of the reservation

G6 Render him/herself available to the scheduler

G7 Receive informations regarding the fee defined for each passenger

The system must be able to:

G8 Offer a programmatic interface to enable the development of additional services

1.5 Definitions, Acronyms and Abbreviations

Passenger: the user who sends a taxi request.

User: an human interacting with the system. Users are split in 2 classes: 'passengers' and 'taxi drivers'.

System: the automatic part that manages the service.

ETA: estimated time of arrival.

Taxi zones: geographical partitions of the city, non overlapping, with an average size of $2Km^2$.

Queue: a list of all available taxis in the corresponding taxi zone. It is managed as a FIFO queue. There is exactly one taxi queue associated to each taxi zone.

GPS: global position system.

Shared ride: a passenger shares the ride with other people that origins from the same zone, and go to the same direction

Active reservation : a reservation is considered active if it is in the reservation scheduler and isn't occured yet.

Same direction : two taxi rdies are considered to be going in "the same direction" if and only if the destination taxi zone is the same.

•

1.6 Reference documents

1.7 Overview

Overall description

2.1 Product perspective

The system must interact with a map service, to retrieve information about the route to send to the taxi driver in case of shared ride.

2.2 Constraints

We will develop a unique mobile application that can be used by both passengers and taxi-drivers.
The web application will include only passengers functions.
The mobile application must be available for Android, Windowsphone and iOS.

2.3 User characteristics

The users must be connected to the network to use the application. Passengers can interact with the service through a web browser or a mobile application; they don't need any particular ability or foreknowledge to use it.
Taxi driver must access to the application with a device provided of GPS; since they must follow a standard procedure they must attend a formation course before starting (2 hours will be enough).

2.4 Assumptions

- If a request comes from a zone, whose queue is empty, then the system forwards the call to the first taxi in the queue corresponding to an adjacent taxi zone, starting from the northeast.
- A passenger is required to subscribe an account to utilize the taxi services (taxi request, taxi booking, taxi sharing)

- Taxi drivers can create only one account per vehicle ID
- Passengers who reserve a taxi can delete the reservation; if a taxi was allocated for the ride, the system will notify the taxi driver and put him at the top of the queue.
-

Specific requirements

3.1 External interface requirements

The system must be able to comunicate with a map service, in order to retrieve information about the users' position, send the best route to the taxi drivers ,in case of shared ride, calculate the ETA for the passengers that requested a taxi.
Also the application must be able to interact with a map service, due to allow passengers to select the meeting position.

3.2 Functional Requirements

For each goal, we define the specific function that we will have to implement;

G1 :

- Users can create an account;
- Users can log into their account;
- Passengers can select from a menu the option of requesting a taxi as soon as possible;
- Passengers can insert their position filling an input form and confirm it;
- The system will receive the request and identify the zone in which the passenger is in;
- The system will forward the request to the first taxi in the selected zone queue and wait for an answer;
- If the taxist accepts, the system will remove him from the queue; otherwise it will append the taxist to the last position and scan the list for a taxist to accept;

G2 :

- As soon as a taxist accepts a request, the system invokes the support system to calculate the ETA giving the position of the taxi and the position of the passenger;
- The system will communicate the taxi code and the ETA;

G3 :

- Passengers can select from a menu the option of reserving a taxi for a chosen ride and date;
- Passengers can insert the initial and final position, time and date, their email and confirm it;
- The system will receive the reservation and if it respects the 2 hour constraint it will send a confirmation;
- Ten minutes before the ride starts, the system allocates a taxi for it.

G4 :

- The application must have a selectable option labled:"share your ride", that allows passengers to enable the shared ride service. In case of non reserved ride, the application will ask passengers the amount of time they can wait for others people.
- When the system receive a request of a shared ride, it will search for others shared ride requests starting from the same taxi zone, and going in the same direction.
- When a new passenger is added to a shared ride, the system will interact with the map service, in order to retrieve a new route for the taxi driver, and to calculate new fees
- When the timeout of one passengers ,added to the current ride, occur, the system will procede with the allocation of the taxi .
- After the taxi allocation, the passengers who requested the shared ride will receive, not only the taxi ID, but also the fee they have to pay.

G5 :

- The system must forward a taxi request in the following cases:
  1: A passenger has requested a ride.
  2: A taxi reservation is sheduled to begin in 10 minutes.
- If a taxi driver refuses to take care about a call, the system will move him at the end of the queue,and forward the request to the next taxi driver in the queue. If a queue is empty, the system will notify the passenger that there are no taxi available.

- If a taxe driver accepts to take care of the call, the system shall remove him from the queue.

G6 :

- A taxi driver logged in into the system can select the button " Ready ", then the system will notify the system that the logged user is ready to accept some passengr's call. The application also send the taxi driver's position detected with a GPS
- If the application needs to retrieve data from a GPS and this isn't available, it will remind the user to turn it on.
- When the system receive a notification , by a taxi driver, informing that he is ready to take care of some passengers, it will append the user in the queue corresponding to the taxi zone that include the position retrieved by the application.

G7
- When a taxi driver is assigned to a shared ride, the system will send him the route he needs to follow, and the fee amount every passenger have to pay
- When a driver is assigned to a non-shared ride, the system will send him the route he needs to follow, and the fee amount the passenger has to pay

G8 :

- It is also neccessary to develop programmatic interfaces that allow to customize the system, adding new features.

- Passengers can access a section, in which they be able to check the ID of the taxi assigned to their ride and manage (delete or modify) an active reservation.
- When a passenger delete a reservation , the system will remove it from the reservation scheduler and, if a taxi driver is already assigned, notify the taxist.
- A passenger can modify an active reservation changing position, date and time.
- The system will accept modification only if sent before the taxi allocation.
- The system will accept date and time modification if it occur at least two hours after the request or/and after the previous reservation.
- A taxi driver have the possibility to remove himself from the queue by clicking the:"Disable" button.

- The system will remove a taxi from the list if receive the corresponding request by the taxi driver, or if the taxist logged out.

3.3 Scenarios

1. Jon is driving back home while he notices a strange noise coming from his car. He decides to stop at the first garage on the road and then call a taxi using an app he downloaded some weeks ago, MyTaxiService. The mechanic checks the car and tells him that the problem is quite serious and the car must stay there for some maintenance. He opens MyTaxiService app and he accesses his account with his email and password. Then he inserts his position and selects "call a taxi". The app informs him that he has to wait 5 minutes for taxi 13C to come over.

2. Brandon is going to Moscow in three days and must be at the airport at 3,00 pm. Since the parking fee at the airport is very high he decides to reserve a taxi that will lead him right near the airport entrance. Brandon searches on his personal computer for a taxi service and finds MyTaxiService, creates an account inserting his email address and inventing a password, he waits untill he receives a mail from MyTaxiService at the same email account he gave while signing in. He clicks the link in the mail and now his account is valid. He can reserve a taxi from his house to the airport for that day and see the details in his "active taxi list".

3. Eddard, a taxist, has just started a new day of work. As soon as he gets in his vehicle he logs into his account, inserting his code and password. Now he is connected with the system, a notification pop up informs him to turn on the gps. He activates the gps and he is sending correctly his position. Now he awaits for incoming calls.

4. Robb, a taxist, receives a notification of an incoming call. The request comes from Mario Street, not far from his position. Robb looks at the time, it's 12.58 meaning that his turn is over in two minutes, so he declines the request.

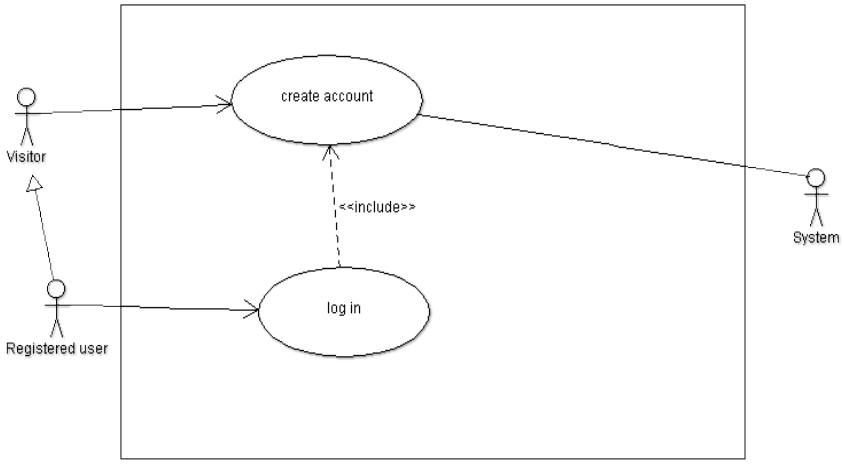
5. Arya wants to go to an exibition downtown. She has heard that the city centre will be closed at traffic but taxis will still be able to access. So she decides to book one, and in order to save some money she reserves a ride and enables the sharing option, hoping that also others will use the taxi and join her. She receives a notification from the taxi service in which they confirm the taxi and that 3 more people will use the same ride so the cost will be only of 5 dollars.

6. Rickon, a taxist, receives a notification of an incoming call. It's a shared ride, the system provides him the route he has to follow, pick up a person in Golgi street then one in Grossich Street and leading them to Piazza Duomo, and the fees every passenger has to pay.

3.4 Use cases

- Create an account

USE CASE	A user can create an account into myTaxiService
ACTORS	Visitor
Entry condition	
Flow events	Visitor creates an account inserting his email address and choosing a password (longer than 8 characters), the system processes the submission and sends a confirmation mail to the given address in which there's a link that the user must click to validate his account.
Exit condition	A new account is created
Exception	If the email address is not correct or the user doesn't confirm his account, the system will not consider any request from it. If the email address already belongs to an existing account or the password is shorter than 8 characters, the system gives an error before the user can continue.



3.5 Software system attribute

3.5.1 Reliability

In order to easly react against failure, the system will make a backup of all the server and save it on a cloud service.

3.5.2 Availability

The system is completely automatized, so it will be available every day at every time, except for the first wednesday of every month from 20:00 to 23:00 , when the server will be disconnected due to maintenance or update.

3.5.3 Security

The servers will be protected ,by external attack , adding two firewall located between the network and the apllication server, and between the applicatione server and a the data server.

3.5.4 Maintainability

The application must porvide set of API for the purpose of adding new features in future time. Those API must be thoroughly documented; the

core system must be documented as well.

3.5.5 Portability

The server side application could be deployed on any platform supporting JRE-7.
The mobile application shall be developed for the major mobile operating systems (Android , iOS, Windows phone). The web application shall be compatible with the most widely used browsers (Mozzilla,Google Chrome, Safari, Internet explorer)

- - -

Appendix

4.1 Alloy

Here is displayed the fully commented alloy code used to model the portion of city our system will interact with

```
sig GeographicalPosition{
}

sig Taxi{
        location: one GeographicalPosition
}

sig TaxiQueue{  // just a set of taxis
        taxi: set Taxi
}

sig TaxiZone{  // taxi-zone as defined in RASD
        queue: one TaxiQueue,
        positions: set GeographicalPosition
}
{
        #positions > 0 // an empty taxi zone would be
           useless
        all t: queue.taxi | t.location in positions
           // if a taxi is the queue, then its
           location must belong to the zone as well
}

sig City{
        // note: the model itself does not impose that
            there is only one city
        // there could be many overlapping cities
        // the limitation #City = 1 is applied
           elsewhere
```

```
        taxis: set Taxi,
        zones: set TaxiZone,
        positions: set GeographicalPosition
}
{
        all z: TaxiZone | z.positions in positions  //
            taxi zones are made of the same positions
           which make up the city
        all z: TaxiZone | z.queue.taxi in taxis  //
           taxi zones contain only taxis which are in
           the city
        all t1, t2: TaxiZone | !(t1 = t2) => t1.
           positions & t2.positions = none  // taxi
           zones do not overlap
        all t1, t2: TaxiZone | !(t1 = t2) => t1.queue
           & t2.queue = none // different zones have
           different queues
}

// other important constraints
fact {
        all q: TaxiQueue | q in TaxiZone.queue  //
           every taxi-queue must belong to a taxi-zone
        all t: TaxiZone | t in City.zones  // every
           taxizone must belong to a city
        all p: GeographicalPosition | p in TaxiZone.
           positions  // every point in the city must
           belong to a taxi-zone (in other words: the
           city is fully covered by taxi-zones)
        // the following constraint is incorrect (
           commented away), since a taxi servicing a
           passenger resides in a zone, without
           appearing in the queue
        //all t: Taxi, z: TaxiZone | t.location in z.
           positions => t in z.queue.taxi  // the dual
            of what specified in TaxiZone's appended
           fact: if a taxi belongs to a zone's queue,
           then its location must be in the zone as
           well
        all q: TaxiQueue | q in TaxiZone.queue  //
           every taxiqueue must be attached to a
           taxizone
        all t: Taxi | t in City.taxis
        #City = 1  // the scope of our project
```

```
}

// arguments:   t: target taxi, dest: its destination,
    fromZone, fromZone': pre and post state of the
   taxi's initial zone
//                      toZone, toZone': pre and post
   state of the taxi's final zone
pred taxiMove[t: Taxi, start, dest:
   GeographicalPosition, fromZone, toZone, fromZone',
   toZone': TaxiZone] {
        start in fromZone.positions and start in
           fromZone'.positions
        dest in toZone.positions and dest in toZone'.
           positions
        fromZone'.queue.taxi = fromZone.queue.taxi - t
        toZone'.queue.taxi = toZone.queue.taxi + t
}

// helper predicates, used to describe the state of a
   taxi
pred taxiIsUnavailable[t: Taxi, z: TaxiZone] {
        t.location in z.positions
        t not in z.queue.taxi
}
pred taxiIsAvailable[t: Taxi, z: TaxiZone] {
        t.location in z.positions
        t in z.queue.taxi
}

// arguments: t: the target taxi, c: city in the pre-
   state, c': city in the post state
// we need a pre/post state city (and not just a pre/
   post state taxizone)
// because of vincula imposed on the taxi zones by
   city's append-fact
pred taxiBecomeUnavailable[t: Taxi, c, c': City] {  //
    TODO? unsure
        t in c.taxis and t in c'.taxis  // t always
           belongs to the city
        c.taxis = c'.taxis  // the taxi set is
           unchanged
        all z: c.zones | t not in z.queue.taxi => z in
            c'.zones  // all zones the same except the
            one with the changed taxi
```

```
        // the availability of t in the pre-state
           implies the unavailability of t in the post
           -state
        taxiIsAvailable[t, c.zones] =>
           taxiIsUnavailable[t, c'.zones]
        c.positions = c'.positions  // the pre/post
           state are physycally the same city
}

// sister function of the above
pred taxiBecomeAvailable[t: Taxi, c, c': City] {
        t in c.taxis and t in c'.taxis  // t always
           belongs to the city
        c.taxis = c'.taxis  // the taxi set is
           unchanged
        all z: c.zones | t not in z.queue.taxi => z in
            c'.zones  // all zones the same except the
            one with the changed taxi
        // the availability of t in the pre-state
           implies the unavailability of t in the post
           -state
        taxiIsUnavailable[t, c.zones] =>
           taxiIsAvailable[t, c'.zones] // changed
           from the other function
        c.positions = c'.positions  // the pre/post
           state are physycally the same city
}

pred show {
        #TaxiZone >= 2 // just to make things
           interesting
}

run taxiMove for 1 City, 1 Taxi, 2
   GeographicalPosition, 5 TaxiZone, 5 TaxiQueue
run taxiIsUnavailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run taxiIsAvailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run taxiBecomeUnavailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run taxiBecomeAvailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run show for 15
```

4.2 Generated world

Diagram of one of the instances generate by alloy, running the *show* predicate