# MyTaxiService - Requirements Analysis and Specification Document

Gabriele Giossi (Id: 854216)

06/11/2015

# Contents

# 1 About the application

## 1.1 Disclaimer

This Requirement Analysis and Specification Document (RASD) is part of the documentation for the MyTaxiService project; it is to be consulted by any and all involved in the development steps, and taken as a formal specification document that lists all the requirements, goals, functionalities, and aspects of the application in a formal way; any and all modifications during the development steps must be noted in the Appendix.

This Document is to be consulted in any development of any add-on that is made on the release versions of the applications.

## 1.2 Purpose

The purpose of MyTaxiService is to provide a reliable, performing and easy-to-use tool so that a city's taxi service is optimized with regard to taxi ride and queue management and provide a base program that can be improved by means of APIs in order to meet the specific requirements of the context it is operating. The application is designed to be implemented and used in any city that has a taxi service, and is meant for use by both taxi drivers and passengers.

## 1.3 Scope

MyTaxiService will be a web-based application, supported both on smartphones and PCs; its aim is, as said, optimizing the taxi service of a city: the system divides the city in which it operates into "zones" of about 2 square kilometers, and to each of them computes a taxi queue

When a passenger, by means of the application, requests a taxi ride, the system assigns an available taxi, the first in queue near the starting point of the ride; the taxi driver will confirm a ride before actually carrying it out.

Some of the benefits of implementing the MyTaxiService system are that it is possible to achieve a real-time balance of numbers of rides requested in a given zone with respect to the available taxis in a zone, providing taxi drivers with an almost-real-time response to spikes of ride requests, improving service quality while simplyfing the interaction of requesting a taxi ride and finding a taxi for any passenger, wherever in the city he might be.

## 1.4 Actors

- Passengers: these actors are the clients of the city's taxi service; they require to be driven by a cab from, usually, one point of the city to the other in the shortest, most comfortable (and cheapest) way.
- Taxi drivers (often simply referred to as 'driver'): these actors are the drivers of the city's taxi service's cars; they are required to have a taxi driver license, driver's license and they are formally employed by the taxi

service. They are required to be professional in all aspects of the work of driving passengers through the city. They have, by policy of the taxi service (Check the "Assumptions and Dependencies" section), the hard requirement of registering on the MyTaxiService application.

- Developers: As the informal specification explicitly stated the required presence of APIs to improve the system funcionality, another actor will be the developers who will use these APIs;
- Server administrators: These actors represent the people that manage the server the application is deployed in; they take care of general server management and they are the only individuals that are able to operate the application's driver databases.

## 1.5 Goals

- User-related goals are:
  - Non-registered users must be allowed to sign up;
  - Users must be able to request a taxi;
  - Users must be allowed to register in the application;
  - Taxi drivers (one type of users) must be able to register from their application interface the code of the taxi they're driving, associating the car's GPS to their app;
  - Taxi drivers must be able to list their availability;
  - Taxi drivers must be notified of the ride the system forwards to them, and this has to be done only if they have listed their availability;
  - Taxi drivers must be able to confirm/decline rides;
- System-related goals are:
  - The system must interface between multiple platforms (mobile, PC, etc.);
  - The system must be provided with a map of the domain (city) and the zone division;
  - The system must manage an arbitrary number of queues, one for each zona the city is divided in;
  - The system must store registered passengers and taxi drivers' personal data;
  - Since the system stores personal data submitted by drivers and passengers, a privacy system must be implemented to safely protect personal data;
  - The system must notify about storage and handling of personal data in compliance with current privacy laws;

– The system must provide an administrator-level server-side interface for driver data management, general system diagnostics and statistics showing;

## 1.6 Definitions, acronyms, abbreviations

- Users: any human actor that interfaces with the app in order to request or carry out a ride, in short passengers and taxi drivers;
- System: the server-side portion of the application that manages queues, assigns taxis to requests, tracks the cab GPS systems, and manages/stores user data;
- HTTP: HyperText Transfer Protocol, the cornerstone of the web as we know it, it is the procotol used to send and receive requests that MyTaxiService will need to interface with in order to be a proper web-based service;
- DBMS: DataBase Management System, a system that is used to organize, interrogate to retrieve and present information retrieved from an arbitrarily large organized data amount; MyTaxiService will require interfacing with one;
- GPS: Global Positioning System, a triangulation-based system to provide real-time localization in the world of an entity; MyTaxiService will interface with the taxi cars' GPS system to track the cars' presence in each city zone, and also to draw and compute the route for each taxi request;
- API:Application Program Interface, a set of routines, protocols and tools for building software applications; MyTaxiService will have several APIs that allow developers to implement functionalities to tailor the application to each context it is deployed in;

## 1.7 Document Overview

This document formalizes the considered application's requirements, constraints, characteristics, and functionalities while attempting to achieve maximum compliance to the IEEE standard for Specifications Documents; there are, besides textual formalizations of goals and requirements, UML and Alloy diagrams and code to further clarify as much as possible what is required of the application, given the specifics and the assumptions that we hold true. Moreover, there will be an appendix that is to be used as a changelog of the document during the application's development steps, that will inevitably require modifications to this RASD.

# 2 Overall description

## 2.1 Application Perspective

The application is thought to be able to be deployed and used on its own, without the need to interface with other applications, besides interfacing with GPS systems such as Google Maps and its API; MyTaxiService will be self-contained and independent of any other application, besides the aforementioned GPS interaction and other obvious constraints such as the need to interface with the Internet.

## 2.2 Application Functions

MyTaxiService will function as an on-demand taxi manager and dispatcher: taxis are assigned to a zone of the city (approx. 2 square kilometers each) and are given a position in a queue based on the GPS information of the cab; when a request from a passenger arrives, the system forwards it to the first taxi in the queue in that zone: the taxi driver can either accept and carry out the ride, or decline it, in which case his taxi is moved to the last position in the queue and the system forwards the request to the second taxi in the queue; in the event a driver leaves the zone he queued for, he is automatically removed from the queue (the application on his smartphone automatically sends a request to the central system to be removed from the queue).

## 2.3 User Characteristics

Users of this application will be of two types: passengers who are in need of a taxi, and taxi drivers who drive these taxis and answer to calls. In order for them to make use successfully of the application, it is requires of them to have at least a smartphone or PC/notebook with Internet access.

## 2.4 Constraints

### 2.4.1 Regulatory Policies

Given that MyTaxiService will store and manipulate sensitive data about taxi drivers and passengers, and will manage sessions of registered users, there will be the need to comply with current laws that regulate usage of cookies and storage of personal data.

### 2.4.2 Hardware Limitations

Given that MyTaxiService will be a strictly web-based service, it is required of the application to have a web server and application server that are dedicated to receiving requests from the Internet, understand them, and send replies.

### 2.4.3 Parallel Operation

MyTaxiService will need to support parallel and concurrent operations on the taxi queues and taxi requests, and also on its registered users database.

## 2.5 Assumptions and Dependencies

We hold granted, in the given domain, the following assumptions:

1. We assume users can make use of the application both in an anonymous way, or by registering;
2. We assume taxi drivers MUST register on the application;
3. We assume a single taxi car can be driven, on different days, by different drivers;
4. We assume that a taxi driver can pick up rides in the "traditional" way, and is not forced to exclusively use the system;
5. Furtherly, we assume the system stores data about the drivers, the passengers who register, and keeps track of the rides of the last month.
6. We assume, for drivers, that they are registered on the application (as drivers) by the application's server administrators;
7. We assume the system solely manages and handles queues and requests and has does not interact with payments and such;

## 2.6 Architectural Ideas

Based on what was formalized thus far, we are able to fix some ideas that are recommended for the architectural design step of the application:

- Given the nature of the application, a client-server architecture is mandatory;
- A division of "fat server, thin client" is recommended to reduce at the minimum the load on the client application, allowing it to run more smoothly on portable devices;
- Server architecture is to have a web server and application server for receiving requests and computing and sending responses; given the "fat" nature of the architecture, it is recommended the computational logic and data management allows for the maximum concurrency possible in all aspects (database handling, queue management.....).
- Clients are not to exchange information between each other; every communication is from client app to server and vice versa;

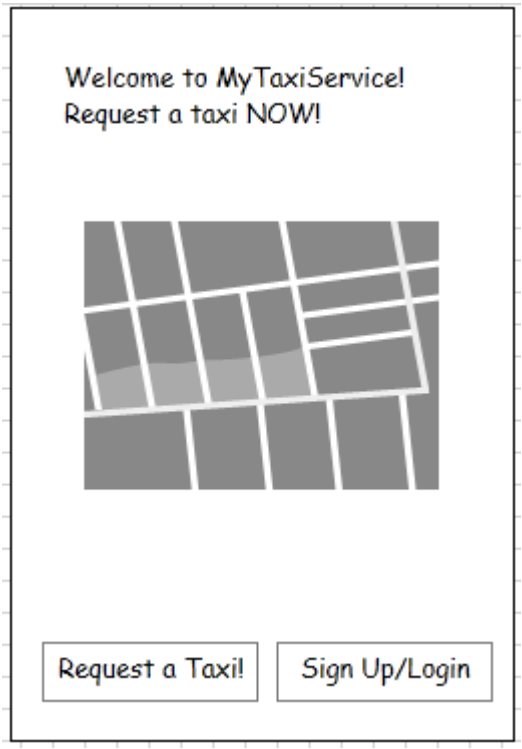
- Redundancy is strongly recommended for taxi queues and requests management: in case of a failure during operation, there must be as little disruption to the service as possible.

# 3 Specific Requirements

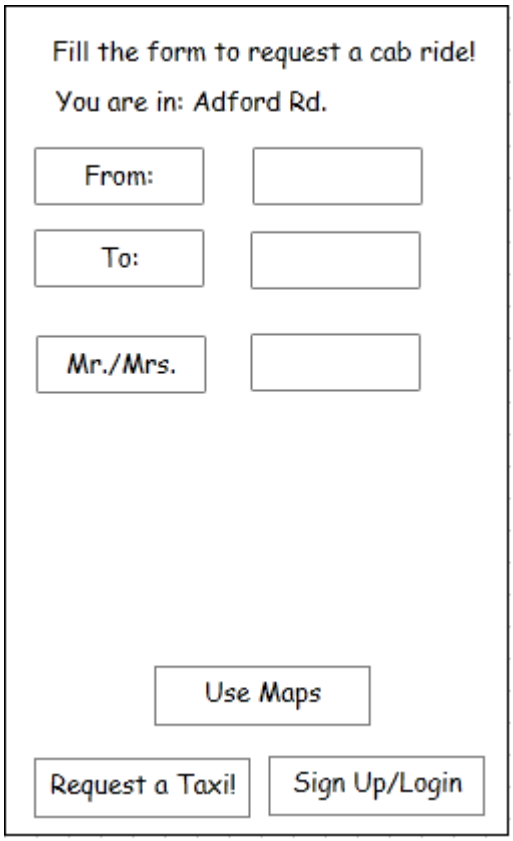
## 3.1 Non-functional Requirement: User Interfaces

Several mock-ups of the user interface for MyTaxiService will follow; they are to give an idea of how the application will show and work. Given the nature of MyTaxiService, the mock-ups are built with the interface of a modern smartphone in mind.

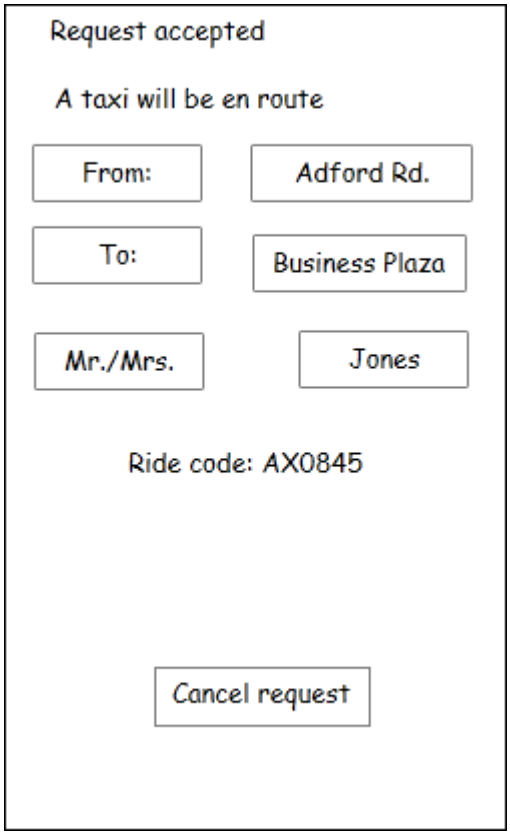
### 3.1.1 Initial Screen mock-up:



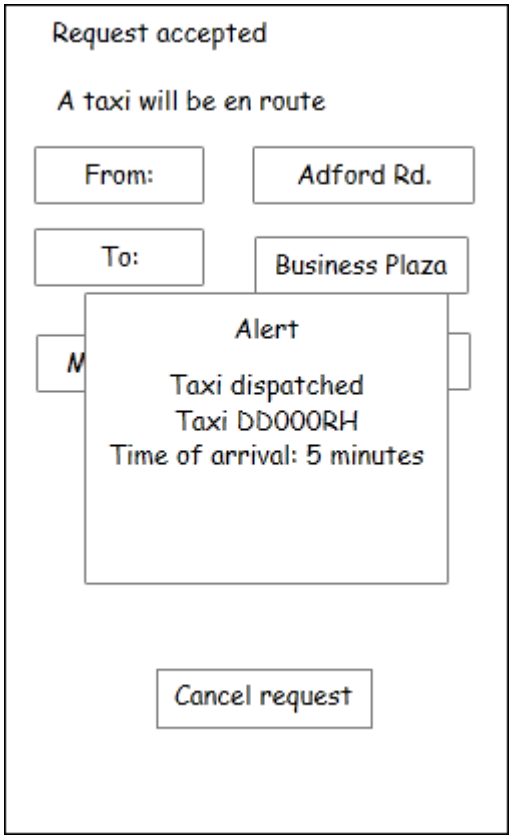
### 3.1.2 Taxi request form (anonymous user):



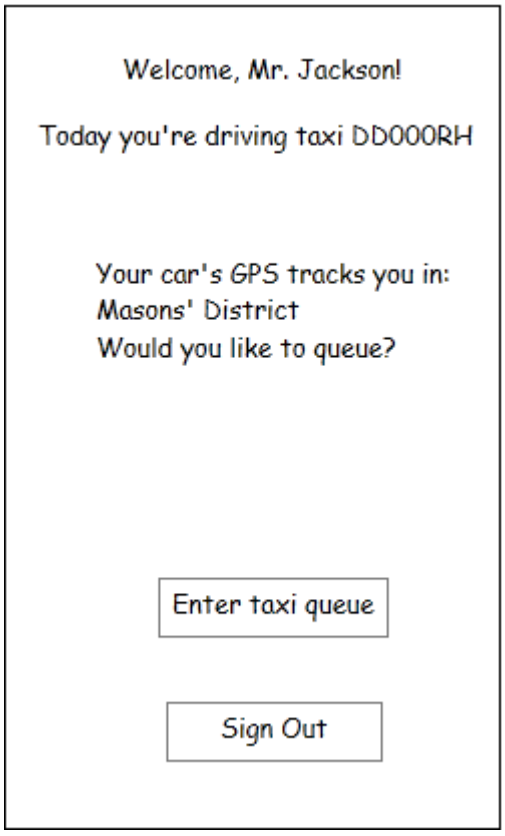
### 3.1.3 Request acceptance:



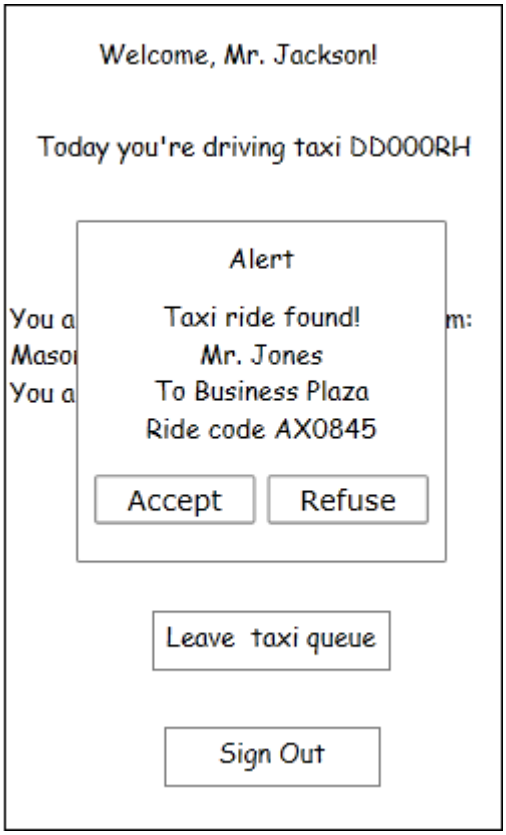
### 3.1.4 Taxi assigned to request:



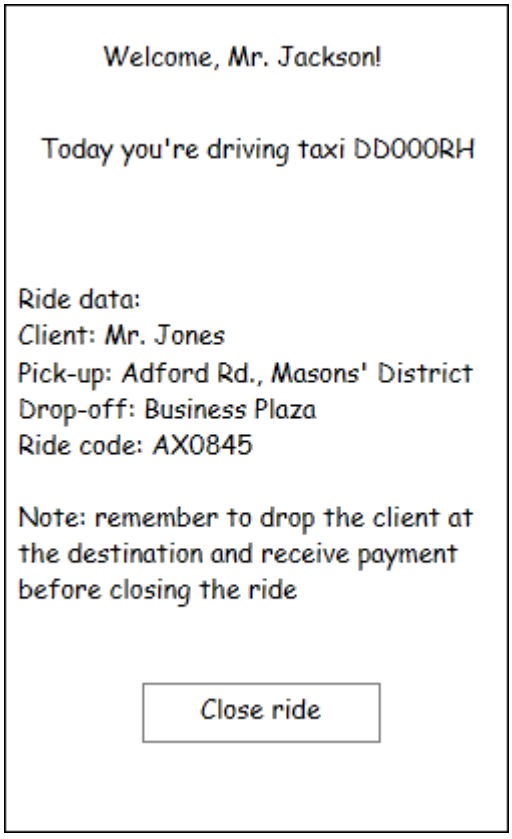
#### 3.1.5 Driver page after login and driven taxi code insertion:



### 3.1.6 Driver is queued and has been assigned a ride:



### 3.1.7 Driver is carrying out the ride:




## 3.2 Non functional requirements: Documentation

The release version of MyTaxiService is expected to be provided with the following documents:

- This RASD for reference and future modifications;
- An Installation Guide to help installing, developing and testing the application;
- A User Manual to detail every possible operation
- A Design Document (DD) to help understand the base design of the application and for future reference;
- Final project report detailing all steps of the development process and team evaluations;

### 3.3 Functional Requirements

The goals that do not have

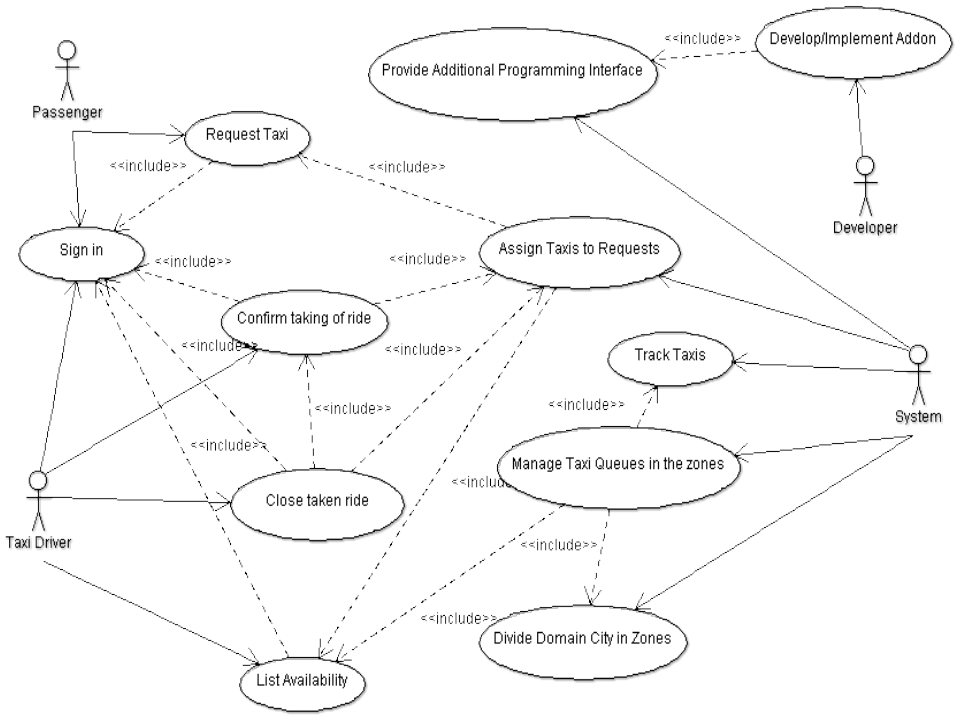
- Non-registered uers must be allowed to sign up;
  - Let anonymous users proceed with requesting taxis, but they must be allowed to sign up;
- Users must be able to request a taxi;
  - Users (both anonymous and registered as passengers) have to be provided a function to fill a form for the ride's data;
  - The start and end point of the taxi ride can be either written by the user in ad hoc fields, or indicated by means of the passenger's GPS system (coordinates);
- Users must be allowed to register in the application;
  - Before or after requesting a ride, there must be always a link to a registration form that will allow the user to register;
- Taxi drivers (one type of users) must be able to register from their application interface the code of the taxi they're driving, associating the car's GPS to their app;
  - After login, if the user is registered as a taxi driver, he has to be prompted to insert his taxi's code before being able to queue for rides;
- The system must provide an administrator-interface for driver data management, general system diagnostics and statistics showing;
  - A server-side interface must be present in order to allow administrators to carry out these operations;
- Taxi drivers must be able to list their availability;
  - Taxis that leave the zone from which they listed their availability are removed from the queue;
- Taxi drivers must be notified of the ride the system forwards to them, and this has to be done only if they have listed their availability;
- Taxi drivers must be able to confirm/decline rides;
  - Implement commands and methoeds that have the system react accordingly;

- The system must interface between multiple platforms (mobile, PC, etc.);
- The system must be provided with a map of the domain (city) and the zone division;
- The system must manage an arbitrary number of queues, one for each zone the city is divided in;
  - A memory space dedicated to managing an arbitrary number of queues must be present in the system server-side, along with the logic to manage and operate on them;
- The system must store registered passengers and taxi drivers' personal data;
  - A Database and associated DBMS must be present in order to store said data, along with all the functionalities needed;
- Since the system stores personal data submitted by drivers and passengers, a privacy system must be implemented to safely protect personal data;
  - Data must be stored in the required database (see previous goal) with some encryption measures, and the database itself must be protected from external attacks;
- The system must notify about storage and handling of personal data in compliance with current privacy laws;
  - Developers must be aware of data protection and privacy laws of the domain they are deploying the pplication in and implement the system and the notifications to the users accordingly;
- The system must provide an administrator-level server-side interface for driver data management, general system diagnostics and statistics showing;

# 4 Scenarios and UML Models

## 4.1 General UML Use Case Diagram

This is a general Use Case UML diagram; the most important use cases will be expanded and there will be further UML diagrams to depict how the system is supposed to work.




## 4.2 Use case expansion

1. Request Taxi:
   (a) Actors: Passenger
   (b) Entry Conditions: Passenger enters the ride request form on his application
   (c) Flow of Events:
      i. The passenger compiles the form with his name, and the start and end addresses of the ride;

ii. Said addresses can be either typed or found by using a GPS device;
iii. The passenger clicks the "Reuqest a taxi!" option, forwarding to the system the request data;
iv. The passegner waits for the system to assign a taxi to his request and then waits for the taxi's arrival;

(d) Exit Conditions: The passenger can either complete the request or abort it before actually sending it or before being assigned a taxi.

(e) Exceptions: The only possible errors are connectivity issues or the system being unable to locate on or both the start and end address.

2. List Availability:

(a) Actors: Taxi Driver

(b) Entry Conditions: Taxi Driver has logged in the application (use case "Log in" resolved)

(c) Flow of Events:
i. The taxi driver requests to be put in the queue with his taxi to the system.
ii. The driver awaits for the System to reply.
iii. The driver receives the confirmation of being queued if everything went correctly, or an error message if else.

(d) Exit Conditions: The taxi driver, once requests to be put in queue, is queued

(e) Exceptions:
i. Connectivity issues prevent queuing.
ii. The system is faults and is unable to queue the driver and notifies him.

3. Assign Taxis to Request:

(a) Actors: System

(b) Entry Conditions: At least one taxi driver has completed the "List availability" use case and at least one passenger has completed the "request a taxi" use case

(c) Flow of Events:
i. First, the system checks if the queue from which he received a request is empty.
ii. If it is not, it fetches the taxi driver's data and forwards him the data from the request; if the driver refuses, it moves him to the bottom on the queue and fecthes data about the second taxi in queue, forwarding him the request; step (ii) is repated until (iii) occurs.

iii. If the taxi driver confirms the ride, the system effectively assigns him to the ride and dequeues him, then notifies the passenger of the incoming taxi and wait time.

(d) Exit Conditions: The taxi driver confirms the ride

(e) Exceptions:

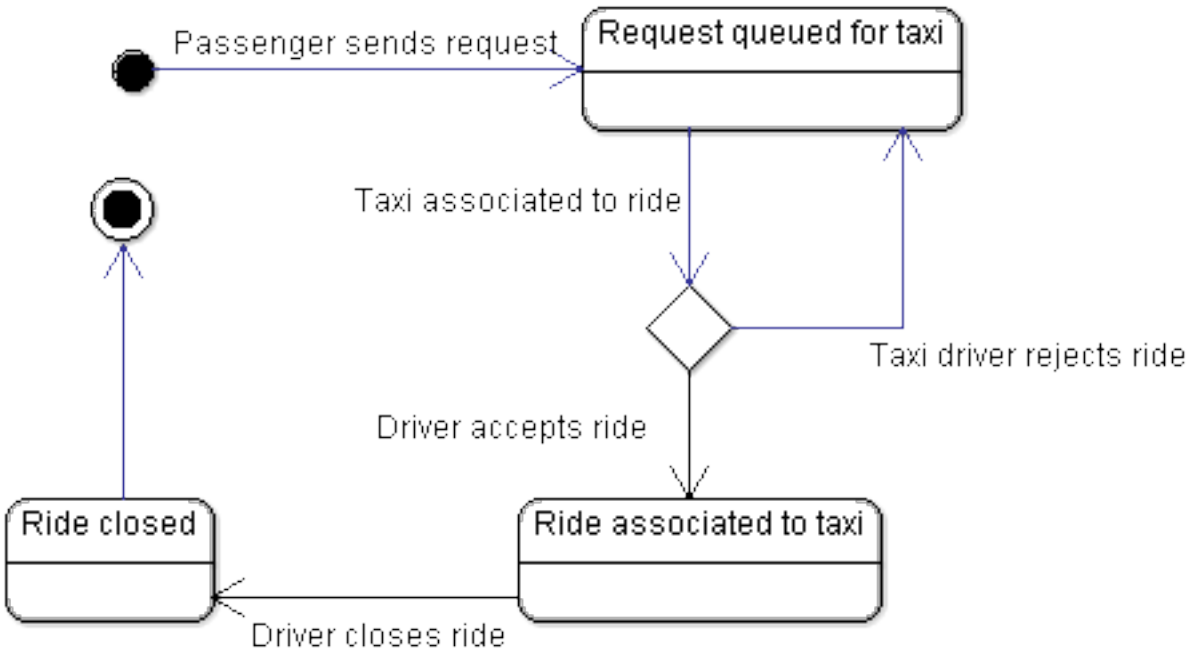
i. If the system faults, requests can be lost without acceptance.

ii. Connection issues can prevent any of the data exchange in the use case.

## 4.3 Other UML Diagrams

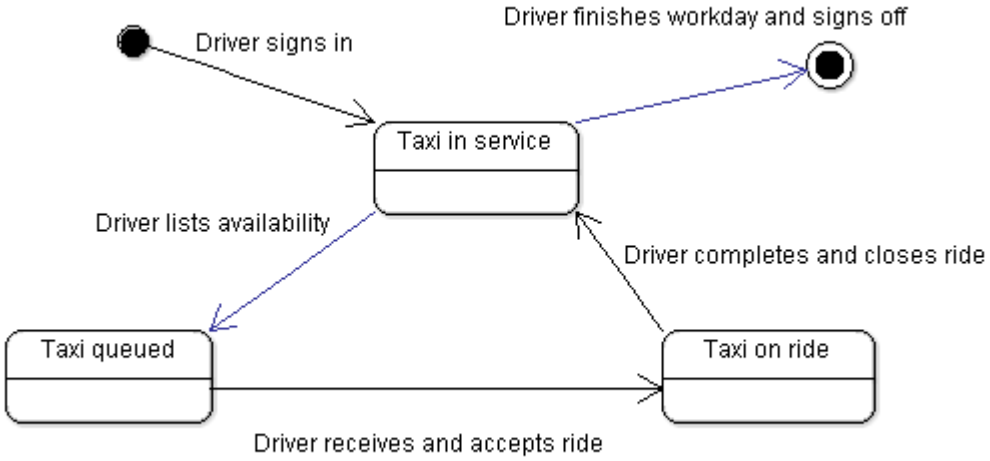
These diagrams depict extra details of the inner workings of the system; it should be noted that they have been made at the highest possible level so that the actual design of the system can be freely analyzed without giving too many costraints.

### 4.3.1 State Diagram: ride

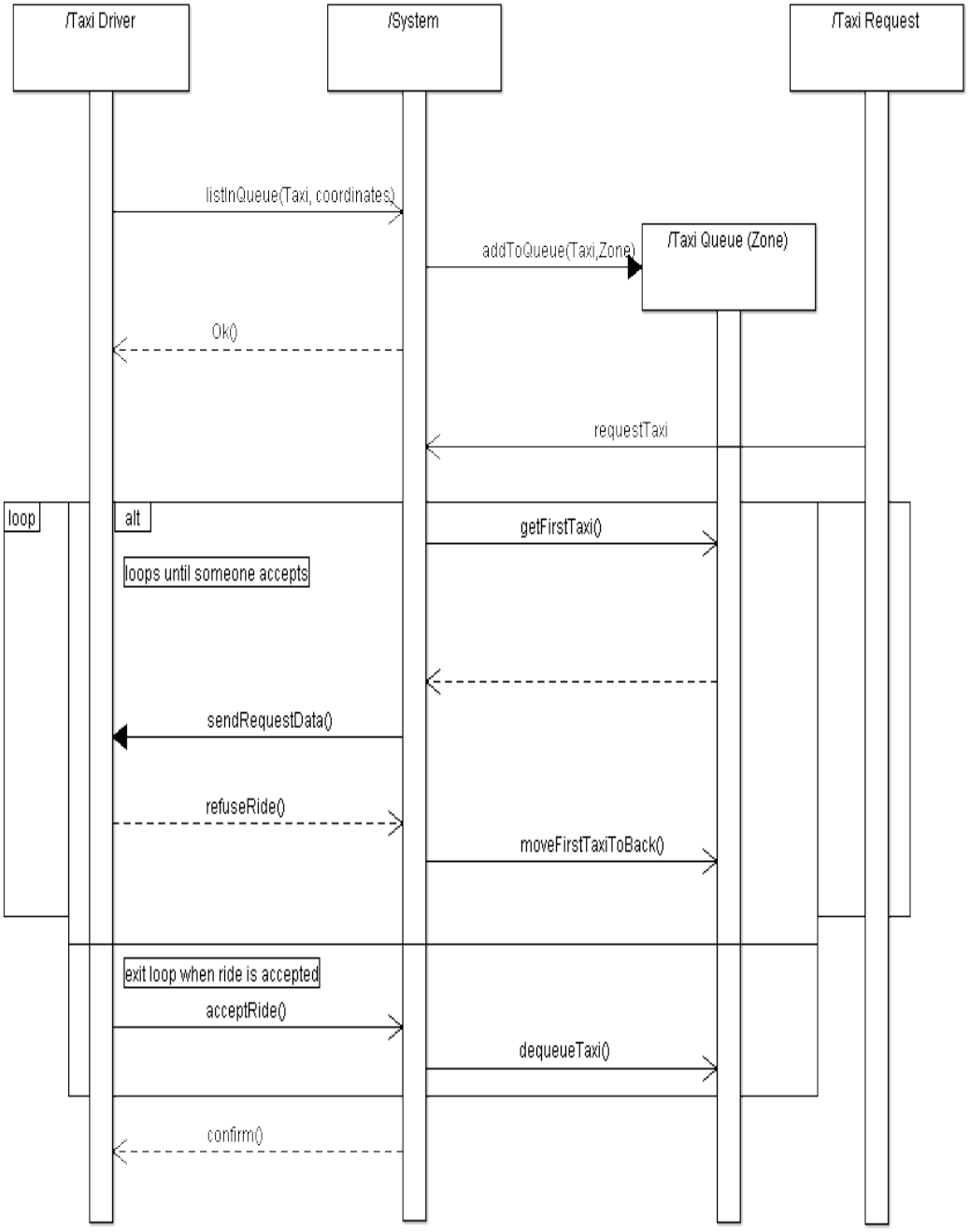


This statechart depicts the different states a ride can pass through, from being a simple request to being closed by a driver once tha passenger is at his/her destination and paid for the ride.

#### 4.3.2 Statechart 2: Taxi

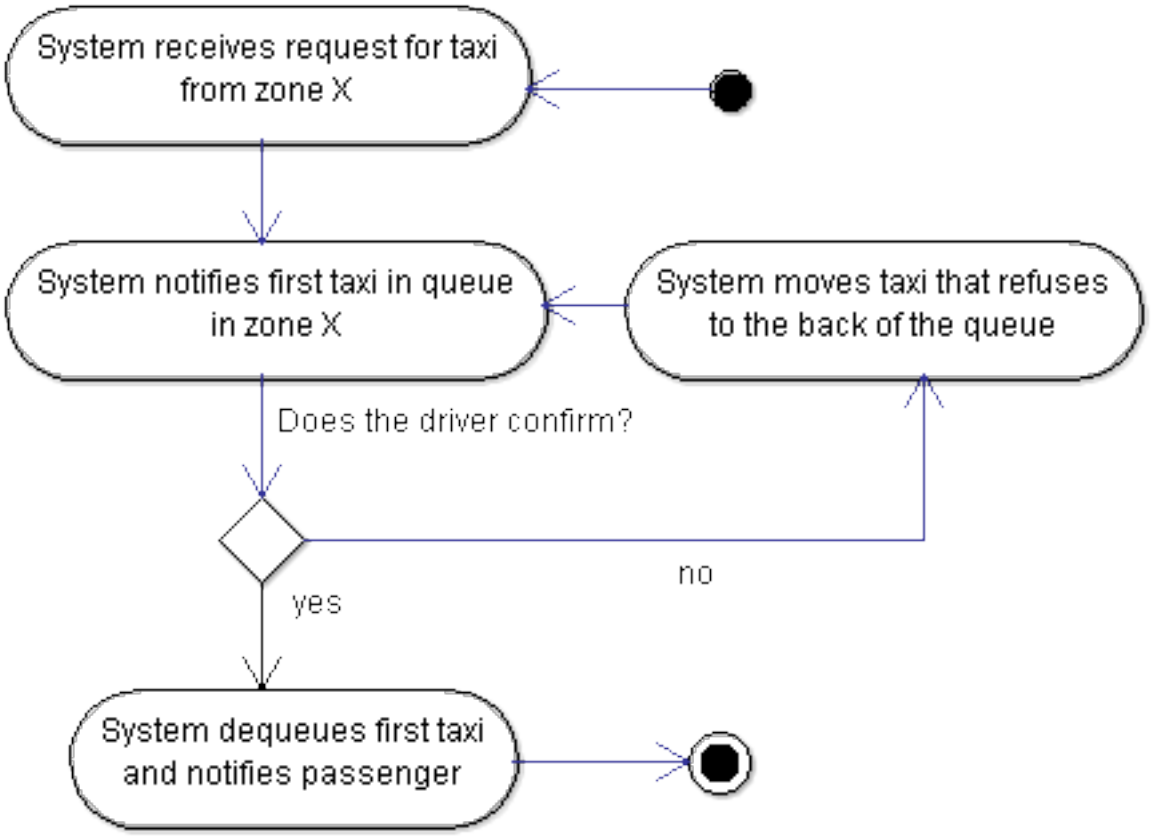


### 4.3.3 Sequence Diagram: request and queue management



This sequence diagram shows how the system is supposed to manage and handle requests and taxi queues, up to a driver confirming the ride.

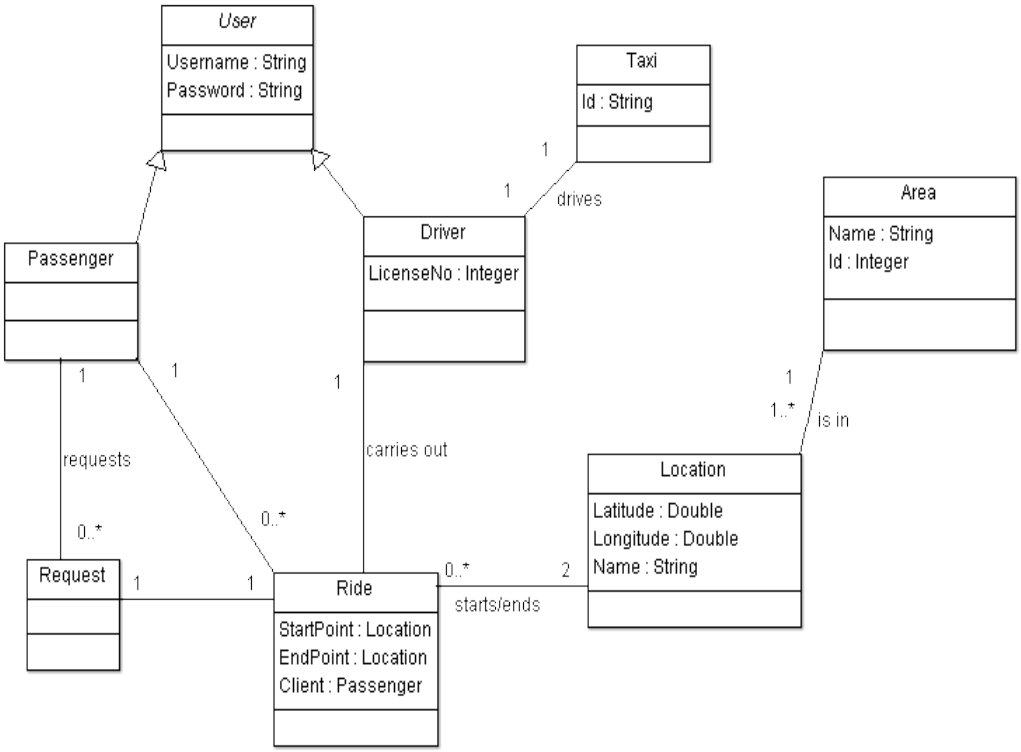
#### 4.3.4 Activity Diagram: system working




This diagram shows the fundamentals of the system's standard operation: request reception, forwarding and handling of queues.

### 4.4 UML class diagram

Based on what has been pointed out so far, we're able to produce an idea of a possible class diagram (further refinements are to be done in the design document redaction stage) that will also be used as a starting point to develop the Alloy code:


User
Username : String
Password : String
Taxi
Id : String
1
1
drives
Area
Name : String
Id : Integer
Driver
LicenseNo : Integer
Passenger
1
1
1
1
1..*
is in
requests
carries out
Location
Latitude : Double
Longitude : Double
Name : String
0..*
0..*
Request
1
1
Ride
0..*
2
starts/ends
StartPoint : Location
EndPoint : Location
Client : Passenger

# 5 Alloy

## 5.1 Alloy Code

The Alloy code follows almost to the letter the class diagram idea that has been put in this document: the idea is to give a high-level metamodel, independent of data types and details, in order to show the relation between objects in the system as thought.

### 5.1.1 Signatures

```
abstract sig Person{
}


sig Passenger extends Person{
}

sig Driver extends Person{
license: one Int,
drives: lone Taxi
}{license>0}

sig Taxi{
id: one Int
}{id>0}

sig Request{
madefor: one Ride
}

sig lat{}

sig long{}

sig Ride{
client: one Passenger,
driver: one Driver,
start: one Location,
end: one Location,
}

sig Location{
inarea: one Area,
lat: one lat,
long: one long
}

sig Area{
}
```

### 5.1.2 Facts:

```
fact NoSameLicenseDrivers{
//no drivers with same license
(no d1,d2 : Driver | d1.license=d2.license and d1!=d2)
}

fact NoSameIDTaxis{
//no taxis with same ID
(no t1,t2 : Taxi | t1.id=t2.id and t1!=t2)
}

fact NoDriverInTwoTaxis{
//No drivers can be in two different taxis at once
(no d1, d2 : Driver, t1, t2 : Taxi | d1=d2 and t1!= t2 and d1.drives=t1 and d2.drives=t2)
}

fact noTwoDriversInTaxi{
//Two drivers can't drive the same taxi
(no d1, d2: Driver, t1, t2: Taxi | d1!=d2 and t1=t2 and d1.drives=t1 and d2.drives=t2)
}

fact noLocationInTwoAreas{
//No location must be in two different areas
(no l1,l2: Location, a1, a2: Area | l1.inarea=a1 and l2.inarea=a2  and a1!=a2 and l1=l2)
}

fact noClosedPathRides{
//Rides start and end in different places
 (no l1,l2: Location, r: Ride | r.start=l1 and r.end=l2 and l1=l2)
}

fact NoDiffLocationsSameCoords{
//Locations that are different must have different coordinates
(all l1,l2: Location | l1!=l2 implies l1.lat!=l2.lat or l1.long!=l2.long)
}

fact OneRequestForEachRide{
//Rides originate from a single request
(all r1, r2:Request| r1!=r2 implies r1.madefor!=r2.madefor)
}

fact AllRidesHaveRequest{
//Rides have to originate from a request
(#Ride=#Request)
}
```

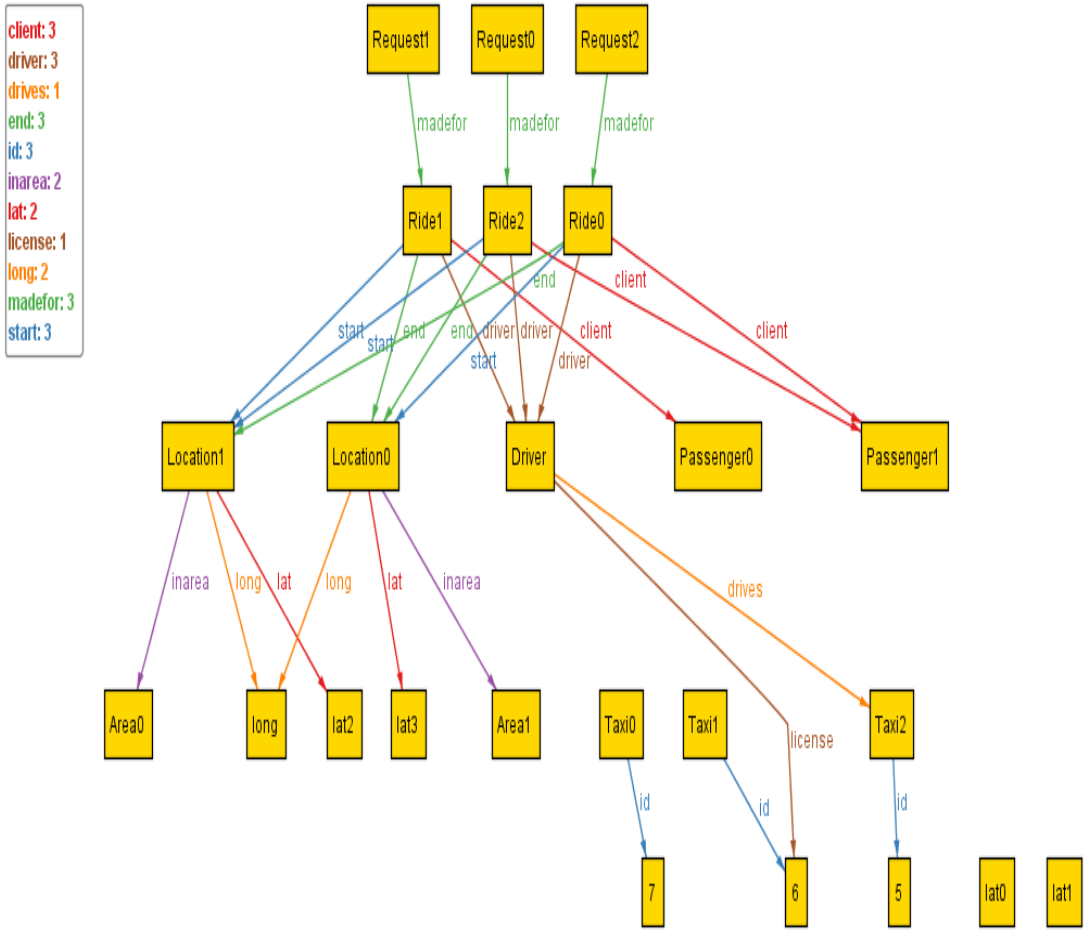
### 5.1.3 Assert/Predicates ran:

```
assert allLocationsInAreas{
(all l: Location | #l.inarea=1)
}
check allLocationsInAreas


pred show{
}

pred showMultiRidesForAPassenger{
#Passenger = 1
#Ride >1
}

pred showMultiplePassengers{
#Passenger>1
}


run show for 8
run showMultiRidesForAPassenger for 5
run showMultiplePassengers for 4
```

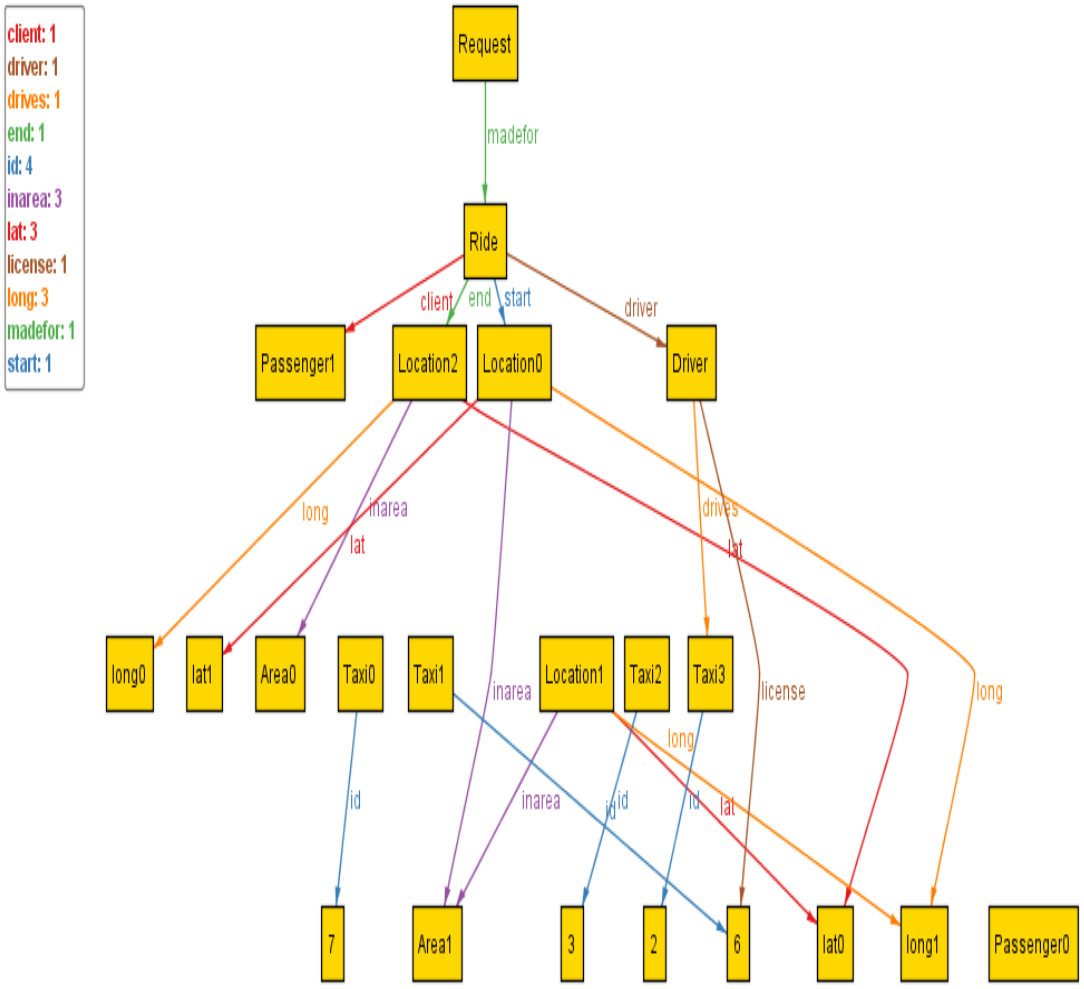
## 5.2 Generated Alloy Worlds

Several worlds have been generated, some of which refliect only parts of the system in order to prove the consistency of both these sections of the model and the model as a whole; these are example world, so there may not be all possible cases (as is with Alloy), but it should be explicative and prove the model's consistency well enough.

### 5.2.1 World 1


client: 3
driver: 3
drives: 1
end: 3
id: 3
inarea: 2
lat: 2
license: 1
long: 2
madefor: 3
start: 3
Request1
Request0
Request2
madefor
madefor
madefor
Ride1
Ride2
Ride0
start
start
end
end
driver
driver
client
client
client
start
driver
end
Location1
Location0
Driver
Passenger0
Passenger1
inarea
long
lat
long
lat
inarea
drives
Area0
long
lat2
lat3
Area1
Taxi0
Taxi1
Taxi2
license
id
id
id
7
6
5
lat0
lat1

### 5.2.2 World 2


client: 1
driver: 1
drives: 1
end: 1
id: 4
inarea: 3
lat: 3
license: 1
long: 3
madefor: 1
start: 1
Request
madefor
Ride
client
end
start
driver
Passenger1
Location2
Location0
Driver
long
inarea
lat
drives
lat
long0
lat1
Area0
Taxi0
Taxi1
inarea
Location1
Taxi2
Taxi3
license
long
long
id
inarea
id
id
lat
7
Area1
3
2
6
lat0
long1
Passenger0

# 6 Appendix

## 6.1 Changelog

1.0: Initial document release

## 6.2 Document redaction

The redaction of this document and the amount of work towards the deadline of the analysis team was:

Gabriele Giossi: 30 hours

## 6.3 Used Tools

- To create and export the UML diagrams the tool used was ArgoUML `http://argouml.tigris.org/`
- Alloy Analyzer was employed to formalize and analyze the Alloy code and generate example worlds.
- To format and redact this document LyX was the used tool.
- Creating the mock-ups of the application was done by the online tool mockupbuiler: `http://www.mockupbuider.com/App`