|  |
| --- |
| Politecnico di Milano A. A. 2015-2016 |
| Requirements Analysis and Specification Document |
| Software Engineering 2: “myTaxiService” |

|  |
| --- |
| Stefano Sartini (Mat. 853697)  06/11/2015 |

SUMMARY

[1. Introduction 3](#_Toc434521073)

[1.1 Purpose 3](#_Toc434521074)

[1.2 Scope 3](#_Toc434521075)

[1.3 Actors 3](#_Toc434521076)

[1.4 Goals 4](#_Toc434521077)

[1.5 Definitions 4](#_Toc434521078)

[1.6 References 5](#_Toc434521079)

[1.7 Overview 5](#_Toc434521080)

[2 Overall Description 6](#_Toc434521081)

[2.1 Product Prospective 6](#_Toc434521082)

[2.2 User Characteristics 6](#_Toc434521083)

[2.3 Constrains 6](#_Toc434521084)

[2.4 Assumptions and Dependencies 6](#_Toc434521085)

[3 Specific Requirements 8](#_Toc434521086)

[3.1 The World and the Machine 8](#_Toc434521087)

[3.2 External Interfaces Requirements 9](#_Toc434521088)

[3.2.1 User Interface 9](#_Toc434521089)

[3.2.2 API Interfaces 11](#_Toc434521090)

[3.3 Functional Requirements 12](#_Toc434521091)

[3.4 Non-functional Requirements 13](#_Toc434521092)

[3.4.1 Performance Requirements 13](#_Toc434521093)

[3.4.2 Design Constraints 13](#_Toc434521094)

[3.4.3 Software System Attributes 13](#_Toc434521095)

[4 Scenarios 15](#_Toc434521096)

[5 UML Models 17](#_Toc434521097)

[5.1 Use Case Diagram 17](#_Toc434521098)

[5.1.1 Registration 18](#_Toc434521099)

[5.1.2 User Log In 18](#_Toc434521100)

[5.1.3 Driver Log In 19](#_Toc434521101)

[5.1.4 Request a taxi 19](#_Toc434521102)

[5.1.5 Upload Location 20](#_Toc434521103)

[5.1.6 Receive a taxi request 20](#_Toc434521104)

[5.1.7 Receive a taxi notification 21](#_Toc434521105)

[5.1.8 Reply to a Request 21](#_Toc434521106)

[5.1.9 Reply to Incoming taxi 23](#_Toc434521107)

[5.1.10 Receive user response 23](#_Toc434521108)

[5.2 Class Diagram 24](#_Toc434521109)

[5.3 Sequence Diagrams 25](#_Toc434521110)

[5.3.1 Registration 25](#_Toc434521111)

[5.3.2 User Log in 26](#_Toc434521112)

[5.3.3 Taxi Driver Login 27](#_Toc434521113)

[5.3.4 Request a Taxi 28](#_Toc434521114)

[5.3.5 Upload Location 29](#_Toc434521115)

[5.3.6 Reply to Request 30](#_Toc434521116)

[5.3.7 Reply to taxi 31](#_Toc434521117)

[5.4 State Chart Diagrams 32](#_Toc434521118)

[5.4.1 Registered user State Chart 32](#_Toc434521119)

[5.4.2 Taxi Driver State Chart 33](#_Toc434521120)

[6 Alloy Modelling 34](#_Toc434521121)

[7 Appendix 40](#_Toc434521122)

# Introduction

## Purpose

The purpose of this Requirement Analysis and Specification Document (RASD) is to describe and model the different aspects of the myTaxiService application. The system will be defined through the study of functional and non-functional requirements, with regards to constrains and limitation of the application to be and its main characteristics will be modelled using both UML and alloy. The needs of both taxi drivers and passengers will be taken in account in the process of modelling the system. This document’s audience are all the programmers and developers who will want to implement the application or to those who will want to modify it or add functionalities in the future but are not familiar with its structure.

## Scope

The myTaxiService application’s aim is to simplify and optimize the access of passengers to a city’s taxi service. In order to achieve this the city is divided into zones and all the city taxis are assigned to the closest zone from their location, and inserted in a zone specific queue. Passengers have to login or register before using all of the application’s functionalities. Taxi drivers also have to login but not register. Once logged in, a passenger can request a taxi, both via a mobile application or a web application. The system will determinate the zone in which the requesting passenger is and consequently inform the first taxi in the zone queue, which can accept the call or reject it. If no taxis are in the zone’s queue, a taxi from a neighbouring zone is notified. If the taxi accepts the request, the passenger will be informed with the number of the taxi and the waiting time. If the taxi declines the request it is moved to the end of the queue and the next taxi in the queue is informed. Once the passenger receives the notification he can accept it, and the taxi driver is notified again and starts going towards the passenger location. The user can also decline the taxi. None of the payment is done through the application; the application is only used to ask for a taxi and to organize the taxi distribution in the city.

## Actors

* Visitors: all the users that visit the application without having logged in; they can both log in if they are already registered, or register in order to gain access to the application’s functionalities.
* Registered Users: all users who have logged in and can request a taxi. They can complete the request form for a taxi and they are able to access all the functionality of the application.
* Taxi drivers: These are particular users who use this application to obtain information about their next ride.

## Goals

This is the list of all the goal the application has to fulfil.

Visitors should be able to:

* Register to the application, becoming a registered user.

Registered Users should be able to:

* Send a taxi request to the system;
* Be notified by the system with the number of the incoming taxi and the waiting time;
* Accept or decline the taxi;

Taxi drivers should be able to:

* Send their location to the system;
* Be notified by the system with the request of a passenger;
* Accept or decline the request;
* Be notified by the system that the passenger accepted or declined his taxi;

## Definitions

* REGISTERED USER: all the users that have already registered in the system by giving their Name, Surname, Email address, Password and phone number. They have access to all the functionalities described in the goals.
* TAXI DRIVER: all the users that work as taxi driver and are registered in the system as such.
* USER LOCATION: the current location of a registered user from which the users would like to start travelling through the taxi service. This location is defined by a street name and a house number.
* USER DESTINATION: the destination where the registered user would like to travel using the taxi service. As for the user location, it is defined by a street name and a house number.
* TAXI LOCATION: the current location of a taxi driver, based on the GPS information the system receives from a taxi. This is also a street name and a house number.
* TAXI NUMBER: this in the univocal number associated to every taxi which this service uses. It is also associated with a Taxi Driver.
* WAITING TIME: the time it will take a taxi to reach the user if the users accepts the incoming taxi. This does not take in consideration traffic on the way to the customer.
* LOGIN: The login procedure is the same both for registered user and for taxi drivers. The system is the one who has to understand the difference between the two and acts accordingly.
* PASSENGERS (VISITORS): With passengers are described all those customers that would like to request a taxi in order to reach their destination and choose to use the application.

## References

* Specification Document: Assignments 1 and 2.pdf
* IEEE standard for requirement specification.pdf
* IEEE Std. 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
* IEEE Std. 1016™-2009 Standard for Information Technology-System Design-Software Design Descriptions.

## Overview

According to the IEEE standard, this document is divided in these main parts:

* Part 1: Introduction: the main characteristics of the application to be and the structure of the document it’s self are described in this part, alongside the definition of goals and actors.
* Part 2: Overall Description: in this second part the application will be described in bigger detail with particular attention to constrains and assumptions derived from the reference document.
* Part 3: Specific Requirements: This major part contains the description and analysis of the application’s functional and non-functional requirements.
* Part 4: Scenarios: In this part all the necessary scenarios for the understanding of the application’s functionalities will be described with detail, in order to give an idea on how the application will have to work once it will be developed.
* Part 5: UML Modelling: All the UML tools will be used to give a good representation of the structure of the application.
* Part 6: ALLOY Modelling: Through the use of Alloy the application’s properties will be analysed and specified.

# Overall Description

## Product Prospective

The application here described is completely autonomous and as such not part of an integrated system. There will not be an administrator account or interface, since it will only be focused on taxi driver and passengers as users. Some programmatic interfaces will be described in the document, in particular the ones which will enable the future development of other additional services such as the possibility of choosing a taxi sharing option.

## User Characteristics

The two different types of users that will have access to the application are the taxi drivers and the passengers. For what regards the passengers, they should only be accustomed to mobile and web applications since the application will not require specific technical skills to be used. As for the taxi drivers a brief procedure will have to be taught to them to guarantee the correct usage of the application.

## Constrains

* If the application doesn’t have Internet access its functionalities are not available, since it cannot communicate with the system.
* The system has to support parallel operations from different users.
* Taxi drivers must have a mobile device with GPS.

## Assumptions and Dependencies

Some unclear points from the specification will be analysed and the chosen solution will be explained in the following paragraph.

* A user might decide not to take the taxi once he has received the waiting time and it is too high, so he has to be able to cancel the request he sent to the system. To make this work, once a taxi driver has accepted a request, the user will be notified as described in the specification, but once he receives the notification he can either accept it or decline it. If he accept, the taxi will be despatched, but if the user declines the taxi, the driver must be notified. Once the taxi has departed, there is no way to decline the request.
* There is no description on how and when a taxi is added to the queue for future service, so when a taxi driver logs in the application he will be able to send his location to the system. From this point on the system identifies the correct zone and consequently adds the taxi to the correct queue. When a taxi accepts the request and the customer confirms, the system removes the taxi from the queue and the taxi will be added only when the driver sends his location again to the system, since the taxi might be in a new zone. If the user does not accept the taxi, it is not removed from the queue.
* No solution is proposed in the specification on what happens if there are no taxi in the zone where a user is requesting a taxi. The solution adopted here is to start searching in the closest zones to the one with no taxi and to forward the request to one of the taxis in these neighbouring zones.
* The zones in which the city is divided are approximately 2 square kilometres each, and are a list of the streets name and house number which are in those 2 square kilometres. Also the GPS signal sent from a taxi driver to the system is a street name and a house number, in this way, by having all locations expressed with a street name and a house number in the system, we achieve consistency in the definition of location.
* The application requires a log in in order to have access to the functionalities because it has to be able to distinguish between normal users and taxi drivers. The log in procedure is the same but the system knows which of the accounts belongs to a taxi driver, and which belongs to a normal user. Only visitors can register from the application, taxi driver accounts are added manually to the system. This avoids people form registering as taxi drivers while they should be registered users.
* The registered users requesting a taxi must always specify both his location and the destination. In this way the taxi driver has more information to decide if he wants to accept the request of decline it.
* Both taxi drivers and registered user are shown a waiting page while the system works to pair them. During this waiting time they cannot use any functionalities of the application and if they close the application the requests are removed from the system.

# Specific Requirements

## The World and the Machine

The methodology used in order to study the domain of the myTaxiService application is based on “The World and Machine” model by M. Jackson and P. Zave. It divides the domain in two portions:

* The Machine: the portion of the system to be developed;
* The World: the portion of the real-world affected by the machine.

All the phenomena can be classified in one of these two sets, some of these, called Shared Phenomena, belong to both the sets. These are particular events of the world influenced by the machine or vice versa.

The World

The Machine

Shared Phenomena

Database queries

User requests taxi

Taxi driver sends location

System running code

User location

User destination

Taxi location

Assigning taxi to a queue

Existing email address

User data

## External Interfaces Requirements

The myTaxiService will run both on a web application but also on mobile devices as described in the specification document, but there is no need to have different interfaces for the two. In fact using two different interfaces wouldn’t add any benefit or deficit to the application. The following interfaces will be a guide line for both type of application. The interfaces here proposed are only used to give an idea of the final product and are not in any way the final version which the developed application will have.

### User Interface

The first interface that will be presented is the home page, from which the user can either log in or to register to the application. All the needed information about the application are displayed in the centre.

Email

Password

Log in

Confirm

MyTaxiService

Application description and all needed information on functionalities.

Register Now Here

This is the registration form interface which will allow visitors to become registered users.

Phone Number

Confirm Registration

Registration

Email (Username)

Name

Password

Surname

The next interface is specific for taxi drivers since they will be the only ones to see it, and allows them to send their location to the system by using the integrated GPS on their mobile phone. Once the GPS signal is found the application will automatically fill the form and the driver only has to submit his location through the button.

Submit Location

Welcome, Name Surname

Street Name

House Number

Use GPS

The last user interface will be specific for registered users who want to request a taxi and contains the form they have to fill in order to send the request.

Submit Taxi Request

Welcome, Name Surname  
please insert your destination and location down here

Current Location Street Name

Current Location House Number

Destination Street Name

Destination House Number

### API Interfaces

Since the taxi drivers also have the possibility to submit their location using the GPS inside their mobile devices, GPS API will be used. These interfaces depend on the operating system the mobile device is using. Other APIs for future development must be included, especially one that will allow a taxi sharing service.

## Functional Requirements

As for the goals in chapter 1.4, functional requirements will be organized based on the different actors.

* **Visitors**:
  + They can only see the home page with information about the application, login form and registration page.
  + They can access the registration form and become a registered user.
    - They should not be already registered to perform registration process.
    - In order to register to the service visitors must give the application a Name, Surname, email address which will also be the username, a password and a valid phone number.
  + They cannot request a taxi and have access to the application’s functionalities.
* **Registered User**: He can:
  + Log in to the application through the appropriate form, using his email address and password.
    - Wrong email and password won’t give the user access to the service.
  + Enter his current location and their desired destination.
    - These are represented as a house number and street name.
  + Send the request to the system with the above mentioned information.
    - User cannot submit multiple requests after the first one.
  + See the system response, which contains taxi number and waiting time.
    - Taxi number received by registered users must refer to one and only one taxi.
  + Accept or decline the taxi.
* **Taxi Driver**: He can:
  + Log in to the application through the appropriate form in the application, using his email address and password.
  + Enter his current location using the GPS of his mobile device to localize his position in the city.
    - The GPS information must be composed of a street and a house number.
  + Submit his position to the system.
    - Once the system receives a taxi location it recognizes its zone and puts the taxi in the correct queue.
  + Receive a taxi request form the system containing customer location and destination.
  + Accept or decline the customer request.
    - If the taxi driver declines a customer, he is moved to the last position of the zone’s queue.
    - If the taxi driver accepts the request he is removed from the queue until his next localization process.

## Non-functional Requirements

### Performance Requirements

The system performance must be as close to real time as the development of the application and the hardware it runs on allow it to be. In this way the only real constraints on the application performance will be the user’s internet connection and the mobile devices’ hardware.

### Design Constraints

All locations in the application are to be considered as the couple street name and house number. Also the GPS information used by Taxi drivers returns a street name and house number. This is used to have a univocal way to identify a location in the city. As for hardware constraints, all taxi drivers’ mobile devices must have a GPS function in order to submit their position to the system.

### Software System Attributes

The following are the main softer system attributes the system has to have once the application will be developed.

#### **Reliability**

The system has to be reliable in the sense that under any circumstance the system must act accordingly to the behaviour described in the previous paragraphs.

#### **Availability**

The system has to create recovery checkpoints as often as the development will allow, in order to guarantee that if the system has some malfunctioning which requires it to be restarted, the minimum impact will be felt by customers and taxi driver.

#### **Security**

The system can be modified only by accessing directly to the computer it is running on. The same for the registration of taxi drivers to the system. Passwords used by users must meet some characteristics to ensure its strength: the system could ask for the password to be at least 8 letters long and it must contain at least one number and a capital letter. Another security measure that could be implemented is a two phase registration: with this authentication technique the user must complete the registration using a code sent to his email or his phone via SMS.

#### **Maintainability**

Other than the previously mentioned API the application will be fully documented. Future developers will be able to use the documentation for the maintenance of the centralized system and the mobile part of the application.

#### **Portability**

Since the application will be running on both web application and mobile devices it must have a high level of portability. The application has to be developed on the three main mobile operating systems: Android, IOS and Windows Phone, and also on different web browsers.

# Scenarios

In this section all the main scenario will be discussed in order to give a description of how the application will work once it will be developed.

1. Tom has just accepted a new job in a new city and does not know how to move in the city by his own. He decides then to look on the Internet if the city offers a taxi service to help him with the movements his new job requires. From his new home computer he finds out about the myTaxiService and navigates to the main web page, were all the main functionalities of the application are describe. Here he finds out he has to register and since he thinks the application might really help him, he registers by giving the application his email, a password, his name and surname and a phone number. He then also install the mobile application on his smartphone so he can use it whenever he might need to call a taxi.
2. Bob is a taxi driver and has recently joined the other city taxi that use the myTaxiService. Yesterday he visited the taxi station where the application is run and his email and password were added to the database. He starts his work day as every other day by reaching his spot in the city. At this point he starts the application from his mobile phone and logs in with the credentials he gave the day before. At this point he sends to the system his current location, using the GPS integrated in his smartphone, and the system adds him to the correct queue based on the zone he is in.
3. A few days after Tom started his new job, he is required to travel to the legal office which is a few blocks away from his normal office. This is a good chance to try myTaxiService which he previously registered to, so he lunches the application from his mobile phone. He logs in the application and request a taxi. The system receives Tom’s request and forwards it to the first taxi in the queue of Tom’s zone. Bob is in his taxi waiting, when a new request pops on his mobile phone. It’s Tom’s request and Tom’s current location in close to where Bob is, so he decides to accept the request. A few seconds later Tom receives a notification that Bob’s taxi is available to give him a ride, and that the waiting time is only 5 minutes. Since 5 minutes are reasonable, Tom accepts the taxi the system proposed and as soon as Bob receives the confirmation, he starts travelling towards Tom’s location.
4. After several hours working at the legal office Tom has ended his work day and needs to go home. Since his previous experience with myTaxiService was satisfying he decides to use it again. After the login procedure he requests a taxi once more. This time the system response is a waiting time of 30 minutes, since in that hour of the day taxis are very requested and the first one available is in a different zone from the one Tom is. 30 minutes are too many in Tom’s opinion so he decides to decline the proposed taxi and to try getting back home using the underground even if he never used it before.
5. Bob has been working all day as well and his shift ends in just an hour when the system forwards him a taxi request. The passenger location is 10 minutes from where he is, which is fine, but the destination is in the other side of the city, and with the current city traffic it could take up to a few hours to reach it. Bob decides then to decline the request knowing that he will be putted at the end of the queue and will probably not have any more customers from the system this evening.

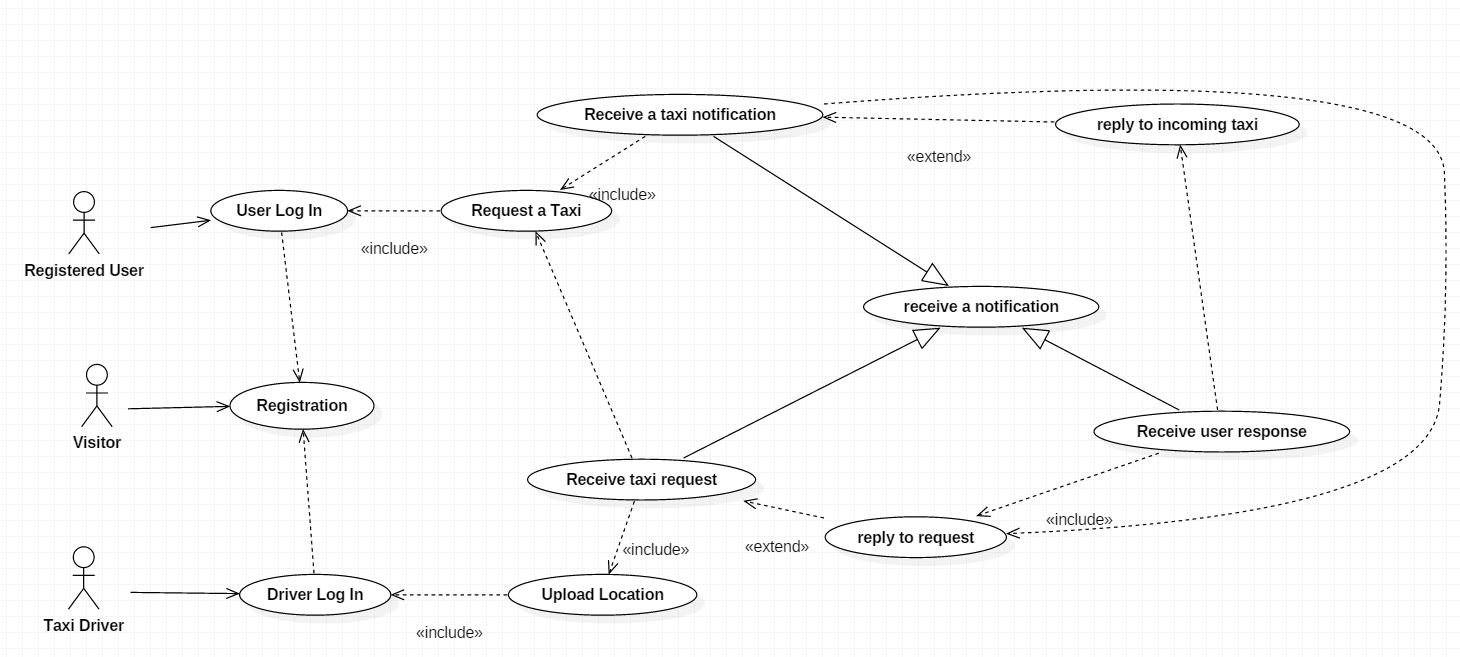
# UML Models

## Use Case Diagram

The main use cases we can derive from the scenarios previously identified are:

* Registration
* User Log in
* Taxi Driver Log in
* Request a Taxi
* Upload location
* Receive a taxi request
* Receive a taxi notification
* Reply to request
* Reply to incoming taxi
* Receive user response

Use Case Diagram:



### Registration

|  |  |
| --- | --- |
| Name | Registration |
| Actors | Visitors |
| Entry Conditions | The visitor isn’t already a registered user. |
| Flow of Events | 1. The visitor reaches the website or opens the mobile application; 2. The visitor clicks the “REGISTER HERE” button; 3. The visitor reaches the registration page; 4. The visitor fills in the form with:    1. Name    2. Surname    3. Email address    4. Password    5. Phone number 5. The visitor clicks “Confirm Registration”; 6. The visitor is sent back to the home page; |
| Exit Conditions | Registration was successful. |
| Exceptions | * Email address is already used. * Phone number not valid. * Password not valid. * Any filed of the form is not filled.   When any of these exceptions happen, the user is alerted and the flow goes back to point 3. |

### User Log In

|  |  |
| --- | --- |
| Name | User Log In |
| Actors | Registered Users |
| Entry Conditions | User has already registered to the system. |
| Flow of Events | 1. The registered reaches the home page; 2. The registered user fills in the log in form with:    1. Email address    2. Password 3. The user click the “Log In” button; |
| Exit Conditions | The system displays the taxi request page to the user and the “Request a taxi” use case starts. |
| Exceptions | * Email address is not associated to any existing registered user. * Password for the corresponding email address is wrong.   The system shows an error message and asks the user to fill the fields and try again and the flow goes back to point 2. |

### Driver Log In

|  |  |
| --- | --- |
| Name | Driver Log In |
| Actors | Taxi Driver |
| Entry Conditions | The taxi driver credentials must already have been inserted in the system. |
| Flow of Events | 1. The taxi driver enters the home page; 2. The taxi driver fills the log in form with:    1. Email address    2. Password 3. The taxi driver clicks the “Log In” button; |
| Exit Conditions | The system displays the localization page for taxi drivers and the “Upload location” use case starts. |
| Exceptions | * Email address is not associated to any existing taxi driver user. * Password for the corresponding email address is wrong.   The system shows an error message and asks the taxi driver to fill the fields and try again and the flow goes back to point 2. |

### Request a taxi

|  |  |
| --- | --- |
| Name | Request a taxi |
| Actors | Registered User |
| Entry Conditions | The registered user must have already logged in the application. |
| Flow of Events | 1. The registered users fills in all the form fields with:    1. The street name of his current;    2. The house number of his current location;    3. His desired destination’s street name;    4. His desired destination’s house number; 2. The registered user clicks the “Submit Taxi request” button. |
| Exit Conditions | The system displays a standby page and the “Receive a taxi request” use case starts. |
| Exceptions | * Any of the fields filled by the user are not valid street names or house numbers. * Any of the field is not filled.   The system shows an error underlining what was wrong and the user is asked to insert the information again, going back to point 1. |

### Upload Location

|  |  |
| --- | --- |
| Name | Upload Location |
| Actors | Taxi Driver |
| Entry Conditions | The taxi driver must already have logged in the application. |
| Flow of Events | 1. The taxi driver clicks the “Use GPS” button; 2. The fields in the application’s form are automatically filled in; 3. The taxi driver clicks the “Submit Location” button. |
| Exit Conditions | The system displays a waiting page for the taxi driver. |
| Exceptions | * The GPS could not localize the device.   The application requires the taxi driver to try again later or try to move to a different location. Flow goes back to point 1. |

### Receive a taxi request

|  |  |
| --- | --- |
| Name | Receive a taxi request |
| Actors | Taxi Driver |
| Entry Conditions | The taxi driver must already have submitted his location to the system. |
| Flow of Events | 1. The system shows the taxi driver a message about the request he was assigned to. 2. The taxi driver clicks the button “Reply to Request”. |
| Exit Conditions | The system displays the request information and the “Reply to Request” use case starts. |
| Exceptions | No exceptions. |

### Receive a taxi notification

|  |  |
| --- | --- |
| Name | Receive a taxi notification |
| Actors | Registered User |
| Entry Conditions | The registered user must already have requested a taxi. |
| Flow of Events | 1. The system sends the user a message about the taxy the system has found; 2. The user clicks on the “Reply to taxi” button. |
| Exit Conditions | The system displays the taxi information and the “reply to incoming taxi” use case starts. |
| Exceptions | No exceptions. |

### Reply to a Request

|  |  |
| --- | --- |
| Name | Reply to a Request |
| Actors | Taxi Driver |
| Entry Conditions | The taxi driver must already done the “Receive a taxi request” use case. |
| Flow of Events | 1. The system displays the information regarding the taxi request:    1. Customer’s current location;    2. Customer’s destination; 2. The taxi driver clicks on the “accept” button if he will take the request; 3. The taxi driver clicks on the “decline” button if he won’t take the request. 4. If the taxi driver declines, he is removed from the zone’s queue. 5. If the taxi driver accepts, the “Receive a taxi notification” use case starts. |
| Exit Conditions | The system displays a waiting page for the taxi driver and the use case “receive a taxi notification” starts. |
| Exceptions | No exceptions. |

### Reply to Incoming taxi

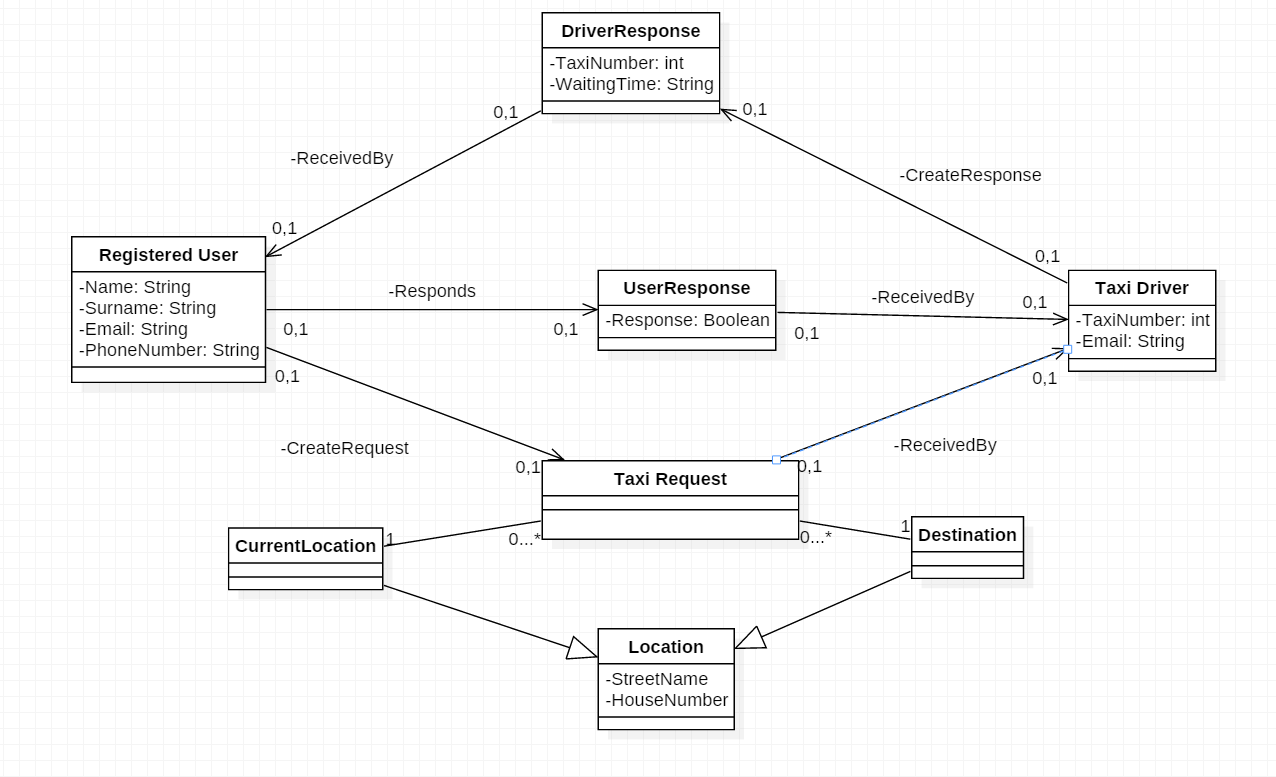
|  |  |
| --- | --- |
| Name | Reply to a Incoming Taxi |
| Actors | Registered User |
| Entry Conditions | The registered user must have done the “Receive a taxi notification” use case. |
| Flow of Events | 1. The system displays the information regarding the taxi:    1. The taxi number;    2. The waiting time; 2. The user clicks the “decline” button if he doesn’t want that taxi; 3. The user clicks the “accept” button if he wants the taxi. |
| Exit Conditions | If the user declines, he is sent back to the Request a taxi page, otherwise the “Receive user response” use case starts. |
| Exceptions | No Exceptions. |

### Receive user response

|  |  |
| --- | --- |
| Name | Receive user response |
| Actors | Taxi Driver |
| Entry Conditions | The taxi driver must already have done the “Reply to a request” use case. |
| Flow of Events | 1. The system displays the information about the customer; 2. The taxi driver is removed from the local queue. |
| Exit Conditions | The taxi driver starts travelling towards the customer’s location. |
| Exceptions | No exceptions. |

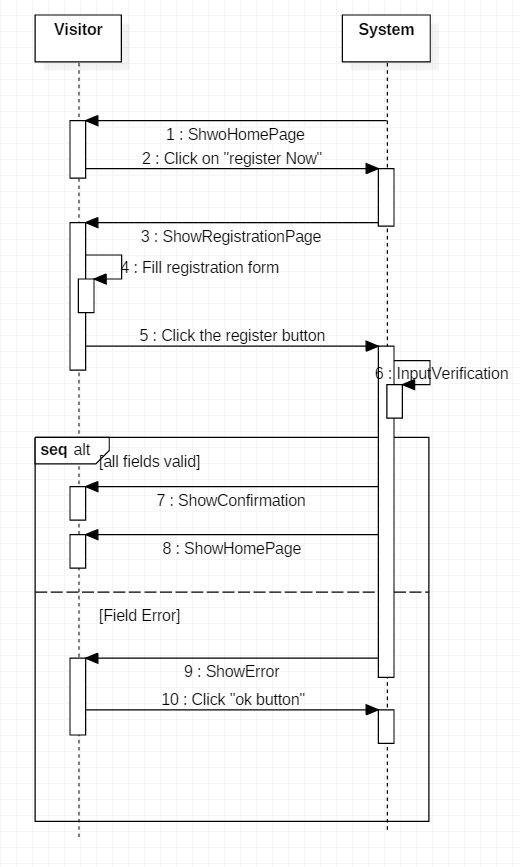
## Class Diagram

This is the representation of the class diagram for the application to be.

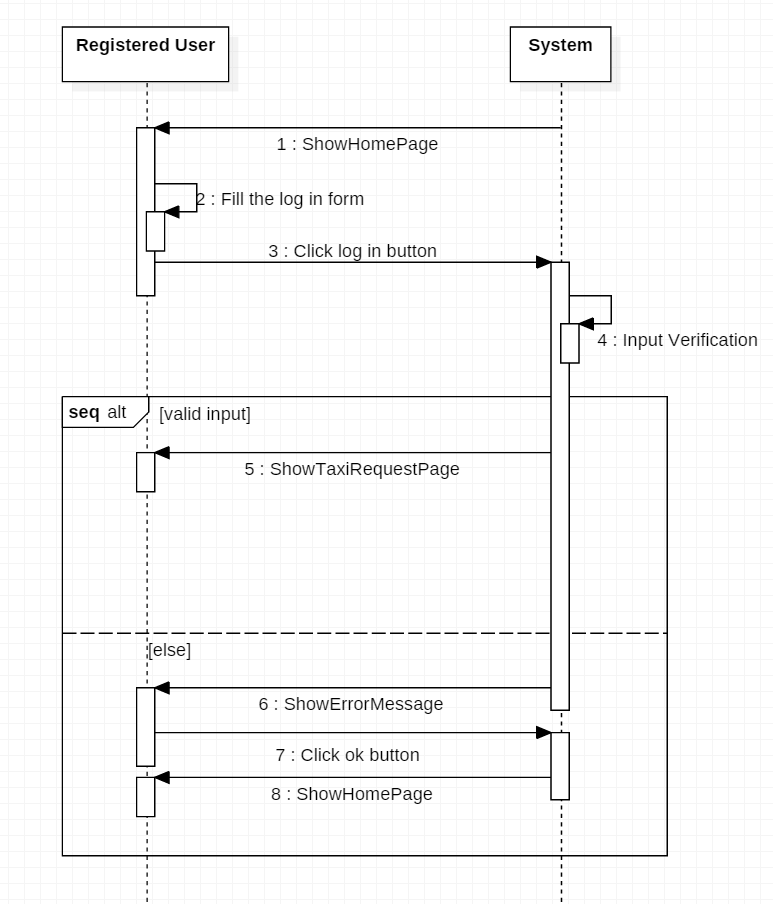
****

## Sequence Diagrams

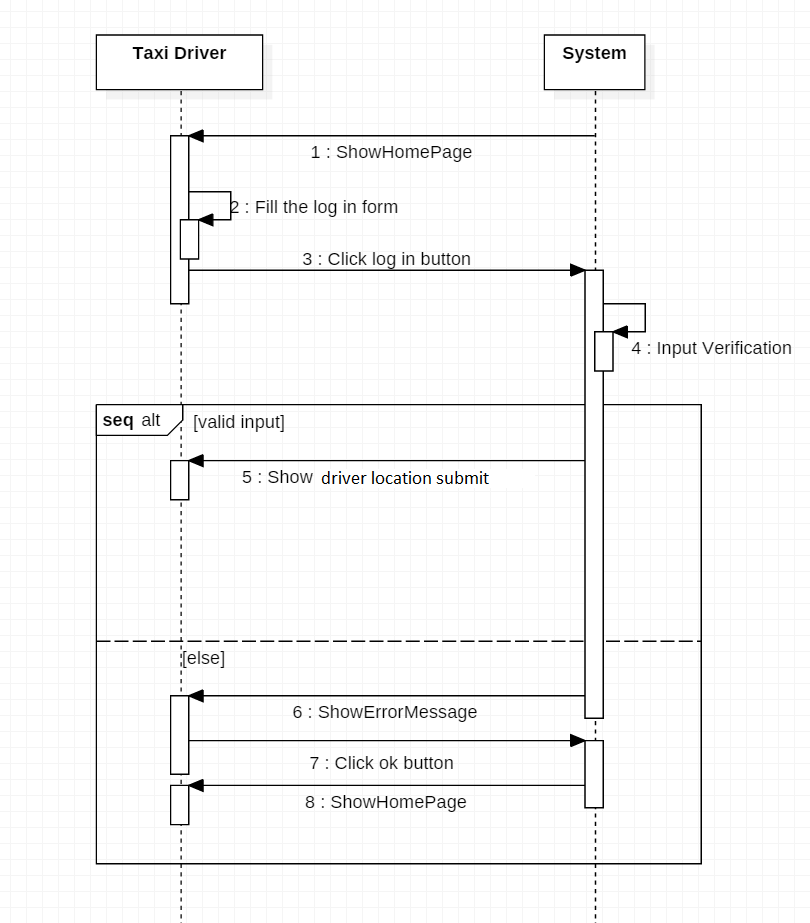
### Registration

****

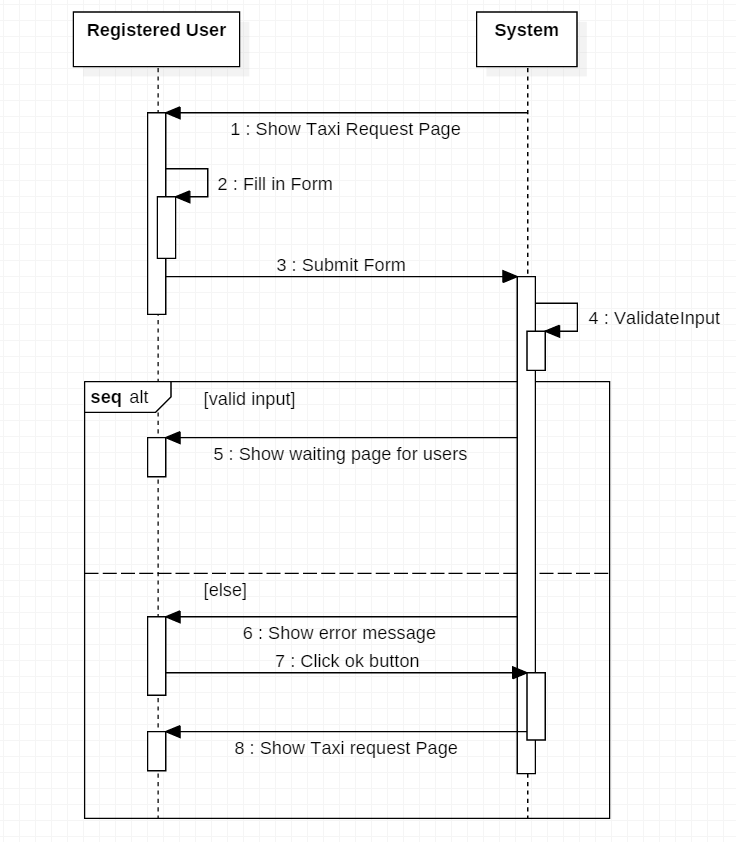
### User Log in

****

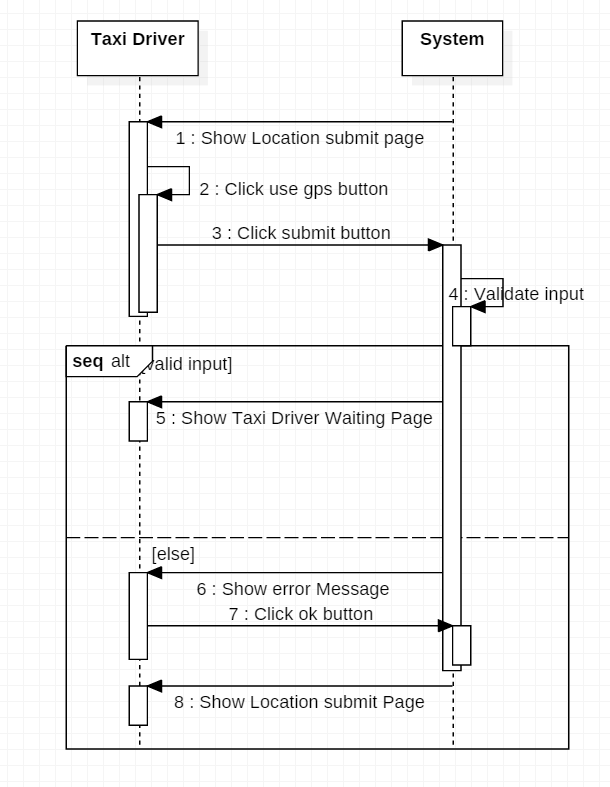
### Taxi Driver Login

****

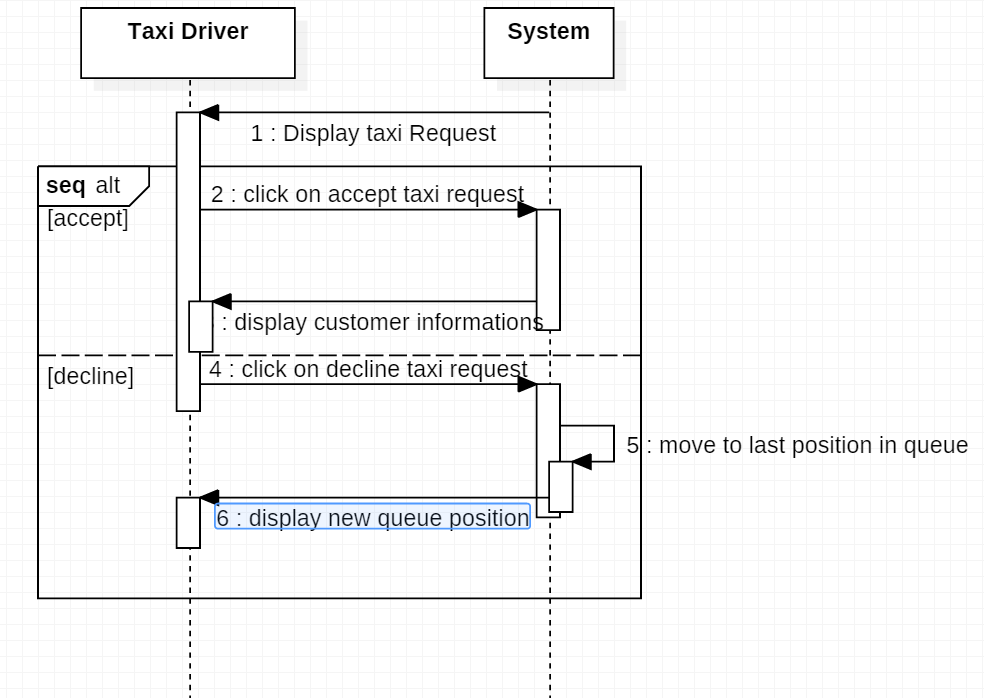
### Request a Taxi

****

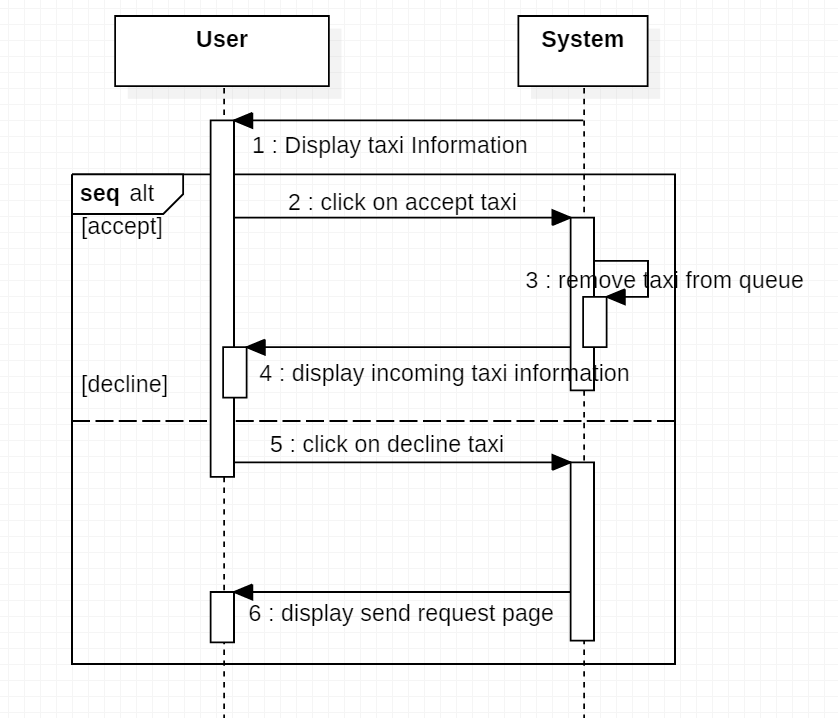
### Upload Location

****

### Reply to Request

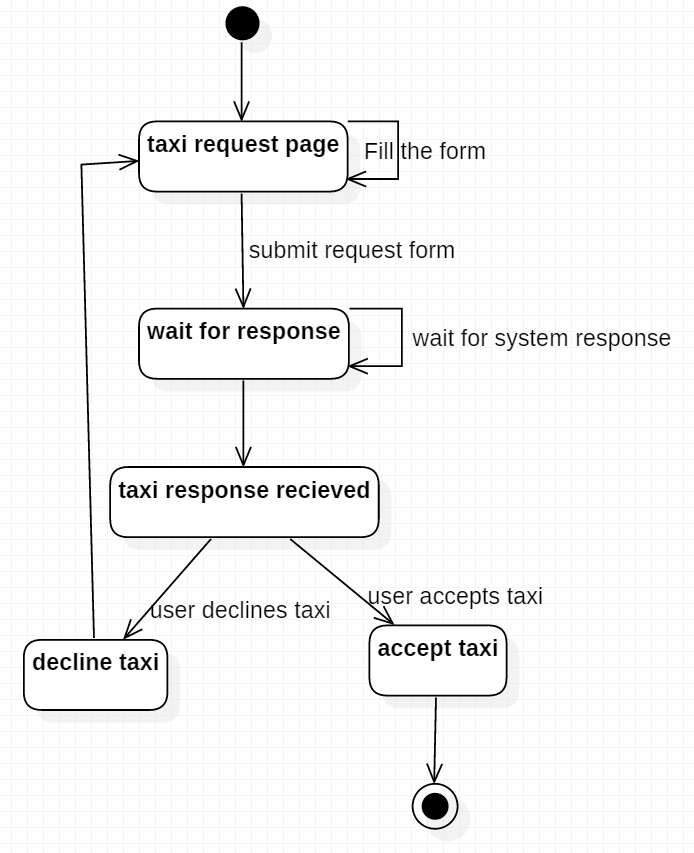
****

### Reply to taxi

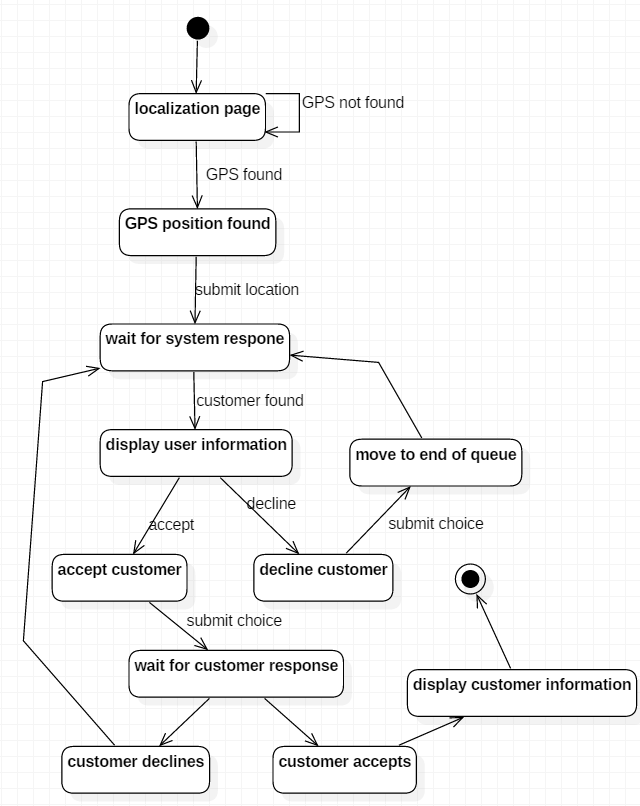
****

## State Chart Diagrams

### Registered user State Chart

****

### Taxi Driver State Chart

****

# Alloy Modelling

Here will be reported the alloy code used to verify that the model is consistent. It shows a representation of the world in which the application will work and contains elements of both the world and shared phenomena, in reference to the model by M. Jackson and P. Zave.

//SIGNATURES

sig RegisteredUser {

position: one Location,

destination: one Location

}

sig TaxiDriver {

position: one Location,

taxiNumber: one TaxiNumber

}

sig Request {

creator: one RegisteredUser

}

sig DriverResponse{

request: one Request,

driver: one TaxiDriver,

waitingTime: one Time

}

sig UserResponse{

user: one RegisteredUser,

response: one DriverResponse

}

sig Location {

street: one StreetName,

number: one HouseNumber,

zone: one Zone

}

sig Queue{

taxi: set TaxiDriver

}

sig StreetName {}

sig HouseNumber {}

sig Zone {

queue: one Queue}

sig TaxiNumber {}

sig Time{}

//FACTS

fact OneRequestPerUser { //Every user has only one request

all u: RegisteredUser | one r: Request | r.creator = u

}

fact DestinationNotPosition{ // Every user has a different position and destination

all u: RegisteredUser | u.position != u.destination

}

fact DriverResponseDifferent{ //No two driver responses can have the same driver

no disj d1,d2: DriverResponse | d1.driver = d2.driver

}

fact DriversSameZoneUser{ //Drivers should allways be in the same zone as the users they answer to

all d: DriverResponse | d.driver.position.zone = d.request.creator.position.zone

}

fact DifferentUsersResponse{ //no two responses can refer to the same user.

no disj r1,r2 : UserResponse | r1.user = r2.user

}

fact UserResponseCorrect{//The user that creates the request and the one that responds to a driver should be the same

all r: UserResponse | r.user = r.response.request.creator

}

fact DifferentTaxiNumbers{ //Every taxi driver has a different taxi number

no disj t1,t2: TaxiDriver | t1.taxiNumber = t2.taxiNumber

}

fact DifferentLocations{ // no location can have the same house number and street name

no disj l1,l2: Location | l1.street = l2.street && l2.number=l1.number

}

fact NoTwoZoneWithSameQueue{ // two zones can't have the same queue

no disj z1,z2: Zone | z1.queue = z2.queue

}

fact TaxiInHisZoneQueue{ // taxi drivers are in their position's queue

all z: Zone | all t: TaxiDriver | t in z.queue.taxi => t.position.zone = z

}

fact NoZoneWIthouLocation{ //every zone has to contain at least one location

all z:Zone | some l:Location | l.zone = z

}

fact LoneUserResponseForDriver{ // for every driver response, at most 1 user response

all d: DriverResponse | lone u: UserResponse | u.response = d

}

fact TaxiDriverinOnlyOneQueue{ //every taxi driver can only be in one queue

all t: TaxiDriver | no disj q1,q2: Queue | t in q1.taxi && t in q2.taxi

}

fact TaxiDriverinPositionAlsoInQueue{ //every taxi is in thus location queue

all t:TaxiDriver | t in t.position.zone.queue.taxi

}

//ASSERTIONS

assert NoDoubleRequest{ // assert no user has two requests

no disj r1,r2: Request | r1.creator = r2.creator

}

check NoDoubleRequest

assert NoUserWithoutRequest{ //Every request must be associated to a user

no r:Request | (no u:RegisteredUser | r.creator = u)

}

check NoUserWithoutRequest

//PREDICATES

pred show{} //generates the world

run show

pred showRequest{ //show the detail of one request

#RegisteredUser = 1

#TaxiDriver = 1

#DriverResponse = 1

#UserResponse = 1

}

run showRequest

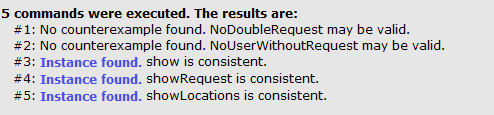
pred showLocations{

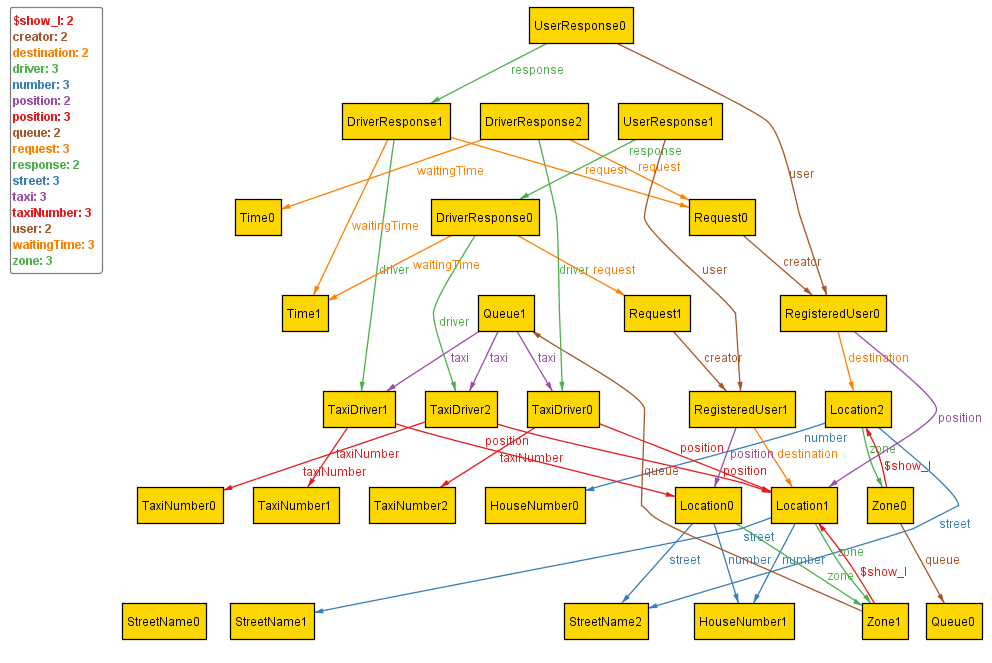
#RegisteredUser =0

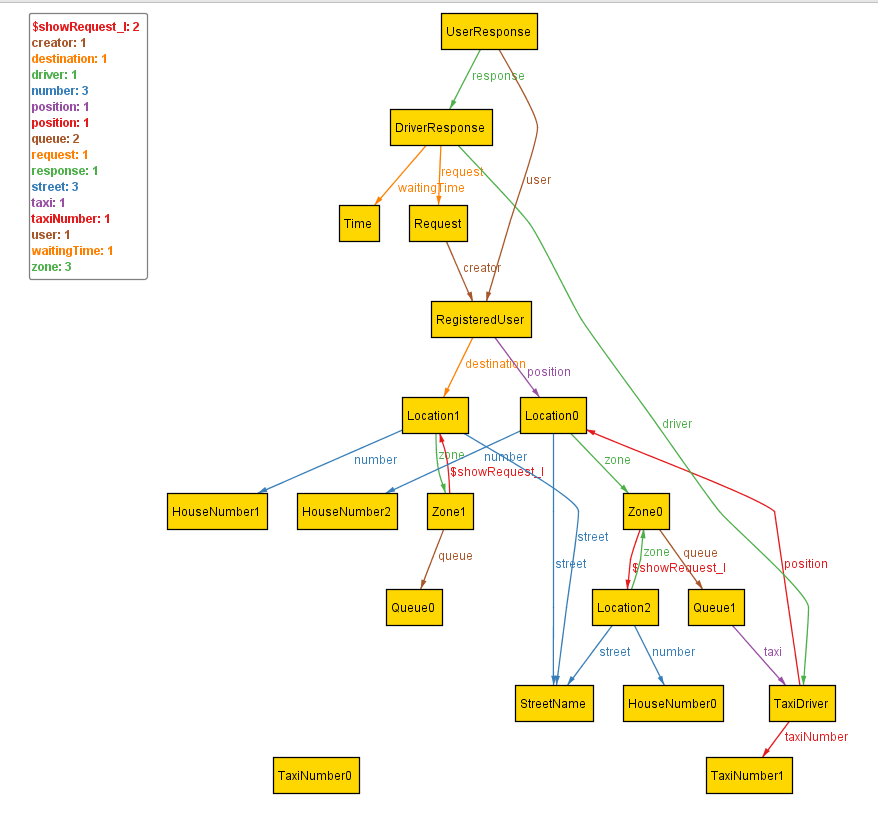
}

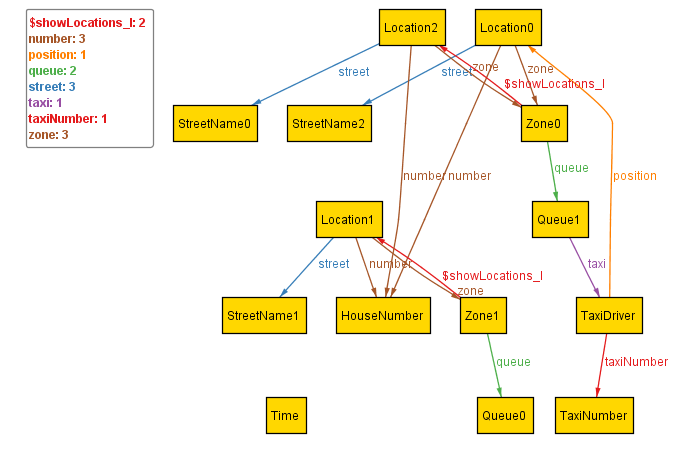
run showLocations

And this is the result given by alloy analyser after running the predicates and assertions:



This is the world generated by the Show Predicate:

This is the model of a single request of a user:

The following is the model of the locations in the world:

# Appendix

The tools that were used to create this RASD documents are:

* Microsoft Office Word 2011
* StarUML for the uml modelling
* Alloy Analizer 4.2 for the alloy models

The amount of hours I worked on this document is approximately 25.