Politecnico di Milano
School of Industrial and Information Engineering
Master course in Computer Science and Engineering
DEIB Department

Requirements **A**nalysis and **S**pecification **D**ocument (RASD)

Moreno SARDELLA - 859239

Academic Year 2015–2016

Abstract

Purpose: this document represent the Requirement Analysis and Specification Document (RASD) of MyTaxiService project.

Work area: Requirements Engineering (RE).

Scope: explain the MyTaxiService project requirements in order to explicit the releated software behavior, according to the stakeholders needs.

Brief summary: the main activity concerned with collecting and analyzing the stakeholders needs, constantly comunicating with them (requirements elicitation and modeling).
It was necessary to know which components are included in a RASD, therefore there was a study about the «The World & the Machine» model (M. Jackson & P. Zave, 1995) and two modeling languages: UML 2.0 (Unified Modeling Language, OMG, 2005) and Alloy (MIT, 1997).

Contents

List of Figures

List of Tables

Introduction

0.1 Purpose of the system

This document represent the Requirement Analysis and Specification Document (RASD), describe the requirements (functional, non-functional), constraints and assumptions, modeling the MyTaxiService project, according to the stakeholders needs. The audience of this document is composed by: the customer and the users, which are interested in validating the system goals and in a high-level description of the project functionalities, the analyst developers, which define the needs to be satisfied, the programmers, which will implement the requirements, the testers, which will check if the requirements will be met.

0.2 Scope of the system

The government of Milan aims at optimizing its taxi service.
In particular, it wants to:

- simplify the access of passengers to the service
- guarantee a fair management of taxi queues.

A person can register to MyTaxiService giving him/her cellphone number and personal data (also identification card and taxi license identifiers in case of taxi driver).
Passengers can request a taxi either through a web application or a mobile app.
A passenger is reached by a taxi through him/her GPS coordinates, automatically sent to the system, which answers to the request by informing the passenger about the code of the incoming taxi (taxi driver ID) and the waiting time (ETA).
Taxi drivers use a mobile application in order to inform the system about their availability and to confirm whether they are going to take care of a certain call.
The system guarantees a fair management of taxi queues.
In particular, the city is divided in taxi zones (figure 1).
Each zone is associated to a queue of taxis.
The system automatically computes the distribution of taxis in the various zones

based on the GPS position it receives from each taxi.
When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.
When a request arrives from a certain zone, the system forwards it to the first available taxi queuing in that zone.
If the taxi confirms, then the system will send a notification to the passenger.
If not, then the system will forward the request to the second available taxi in the queue and will, at the same time, move the first taxi in the last position in the queue.
Besides the specific user interfaces for passengers and taxi drivers, the system offers also programmatic interfaces to enable the development of an additional service, taxi sharing, on top of the basic one.

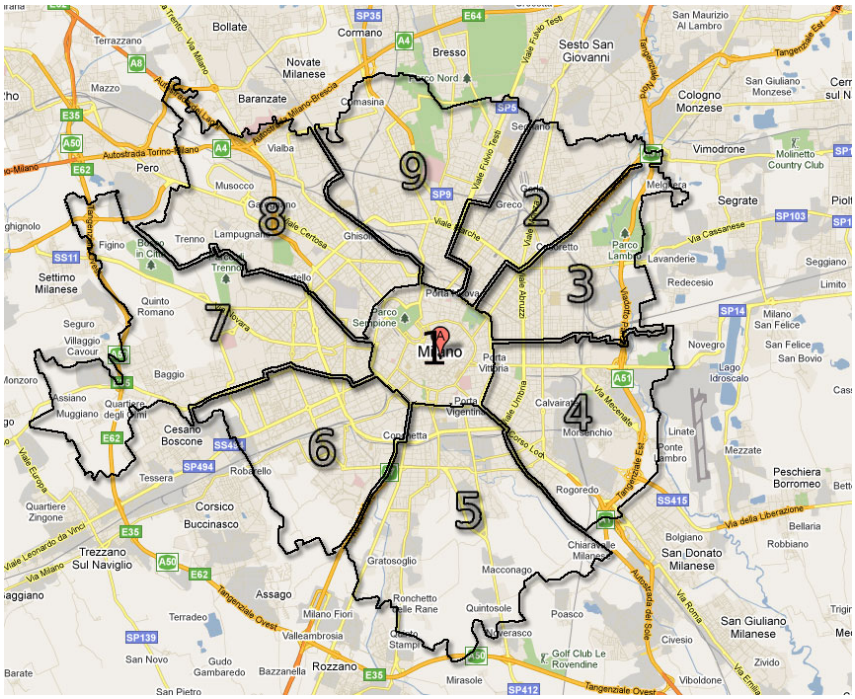

Figure 1: Detailed map of Milan Taxi zones

0.3 Definitions, acronyms and abbreviations

0.3.1 Definitions

- **Notification**: a short SMS sent from the system to a user to signal something.
- **Login**: the procedure through which a guest, entering his credentials, authenticates as a user.
- **Logoff**: the procedure though which a user disconnects himself/herself from

the system.

- **System**: the whole MyTaxiService service (includes app and website).
- **Sign up**: the procedure through which a guest registers herself/himself to the service creating a new account.

0.3.2 Acronyms

- **ID**: Identifier
- **DD**: Design Document
- **ETA**: Estimated Time of Arrival
- **GUI**: Graphic User Interface
- **OS**: Operating System
- **RASD**: Requirements And Specifications Document
- **UML**: Unified Modeling Language

0.3.3 Abbreviations

- **Dn**: n-th domain assumption
- **Gn**: n-th goal
- **FRn**: n-th functional requirement
- **NFRn**: n-th non functional requirement

0.4 References

Specification document: "Assignments 1 and 2".
The structure of this document follows the standard "IEEE Std 830-1993".

0.5 Overview

This document is composed by four part:

1. Overall Description: information about the software product with more focus about actors, goals, constraints and domain properties.
2. Specific Requirements: informations about system requirements.
3. Scenarios: some frequently scenarios.
4. Modeling: UML and Alloy models.

Chapter 1

Overall Description

1.1 Product perspective

The product which will be developed is a new online system whose requirements will not be subject to those of a legacy system.
The only interaction with external systems will concern with payment transactions, identification card and taxi license checking.

1.2 Product functions

1.2.1 Goals

The goals of the project are the following:

1. Provide an easy way to request a taxi.
2. Guarantee a fair management of the taxi queues.
3. Create an extensible system that allows expansion and interactions with other services.

1.3 User Characteristics

Every user must be at least 18 years.

1.4 Constraints

1.4.1 Regulatory policies

The system has to protect the personal data of the users.

1.4.2 Hardware limitations

The mobile app and the website does not work if there is no GPS connection.

1.5 Assumptions and dependencies

1.5.1 Assumptions

1. There is no an administrator user
2. There are no dependences between users
3. A taxi license is provided only if the owner has a driver license
4. A taxi license of a taxi driver is binded to only one car owned by him
5. Every user has a smartphone which is compatible with the app, and with a built-in GPS connector
6. An internet connection and a GPS connection are necessary conditions in order to use the mobile app or the website (except for modifing the personal data)

1.5.2 Dependencies

The system uses:

- government APIs in order to check
  - the identification card ID and the taxi license ID of the taxi drivers
  - the payment transactions through credit cards
- Google APIs in order to calculate the ETA

Chapter 2

Specific Requirements

2.1 Actors

- **Guest**: a person using the service that isn't either logged in or registered.
- **User**: a guest, once logged in the system, becomes a user. A user can be a:
  - **Passenger**: a user who can request a taxi.
  - **Taxi driver**: a user who can supply a passenger request. Also a taxi driver must have a taxi license.

2.2 Functional Requirements

1. Allow a guest to:
   (a) to register [1]giving name^u, surname^u, email address^u, password^u, confirmation password^u, user type (passenger|taxi driver)^u, cellphone number^p, credit card number^t, identification card ID^t, taxi license ID^t
   (b) to login (email address^u and password^u)
2. Allow a user to:
   (a) log in the system
   (b) modify his/her data (except for user type)
   (c) log out from the system
3. Allow a passenger to:
   (a) request a taxi

[1] u: mandatory for every user, p: mandatory if user type is equal to «passenger», optional otherwhise, t: mandatory if user type is equal to «taxi driver», optional otherwhise

(b) choose the payment method for each ride

4. Allow a taxi driver to:

   (a) inform the system about her/his availability

   (b) confirm/deny an incoming request provided by a passenger, visualizing her/his position address

   (c) confirm/deny the start of the ride with the passenger

5. The system automatically has to:

   (a) assign a public ID to every taxi driver

   (b) assign dynamically every taxy driver to a zone queue checking her/his GPS coordinates every 15 minutes

   (c) forward the passenger request to the first available taxy driver in the queue belonging to the zone in which the request comes from (GPS position)

   (d) send to a passenger a notification about her/his request answer (taxi driver ID and ETA whether it is confirmed, «Sorry, retry later.» whether there is no available taxi in her/his zone)

   (e) cancel a ride whether a taxi driver who accepted a request but not confirmed/denied the start of the ride in x minutes, with x=3ETA. Then the passenger will receive a notification «Sorry, something went wrong. Do a request again.»

   (f) move in the last position of the queue a taxi driver who either has denied a request or has not answer it in 5 minutes. Then, 5c

   (g) start a timer when a taxi driver confirm the start of the ride with the passenger, and stop it when a passenger confirms the end of the ride (choosing the payment method)

2.3 Non-Functional Requirements

In order the enlarge the catchment as much as possible, the system must be:

- **Portable**: compatible with all smarthphone OS in the current market.
- **Usable**: easy to use by every user (must be a usability study before the GUI design phase).
- **Reusable**: able to adapt to possible future cities (e.g. the city map and its zone must be external files, in order to make them replaceable with other ones releated to another city).

- **Maintainable**: developed maintainable in order to: fix a bug add new features, improve usability, increase performance, make a fix that prevents a bug from occurring in future, make changes to support new OSs or browser, make it easier for others to maintain the software.
- **Performant**:
  - able to manage at least x requests per second, with x=half number of citizens of the city
  - able to notify a passenger about his/her request at most in 10 minutes (every taxi driver must answer to request at most in 30 seconds).
- **Secure**: able to ensure protection from privacy violantion of sensible data

Chapter 3

Scenarios

3.1 Scenario 1: always the same old Italy

Pavle comes from Serbia, and he attends the master course of computer science at Politecnico di Milano.
It's friday, and he hasn't heard about the public transportation strike because all official voice alerts done in the last few days in every public transportation vehicle were in italian language.
Hence he surf on internet and find a solution: MyTaxiService.
He goes on the home website, click on the registration button and fills the registration form with his name, surname, email address, user type (passenger) password (2 times) and cellphone number.
Then, accepting a confirmation email, he logs in the website, turns on his smartphone GPS and request a taxi.
After just 1 minute he receives a SMS: «Stay there, the taxi driver 2689 is going to reach you in ~5 minutes! MyTaxiService Staff». «Cool!» he thought.
After a while the taxi brings him to the university. All's well that ends well.

3.2 Scenario 2: go home, you're drunk!

Bob is 18 years, and he just passed the driver test and took the driver license.
He loves driving, so he tought to earn some money in order to pay his school fees.
With the help of his parents, he takes the taxi license.
He heard about MyTaxiService from a classmate, who said «Last saturday I went to the disco, I had fun all night long, and I came back home through MyTaxiService. It is very easy to use! I was able to use it even if I was drunk! Ah ah ah!».
Bob took the opportunity to exploit the phenomena, so he dowloaded the app and he registered himself.
The following saturday, at 22:00, he turned on his GPS smartphone and he launched

the MyTaxiService app.
After the login phase, he was redirected to his home page. He clicked on the «Are you available?» button, and the app showed him 2 giants buttons: a green «Yes» one and a red «No» one. He clicked «Yes» of course.
After a while, a push notification on his smartphone notifies him «A passenger from c.so Como 7 request a ride. Would you like to accept?». Once accepted, he went to the received position, but there was no passenger waiting for him, so he had to tap on the home page, click on «Something went wrong?» button and choose the «The passenger stood me up» option.
Next times he was more lucky, and he got used to do the saturday night taxi driver. Now Bob is a mechanical engineer, and he is very grateful to his parents to have helped his dreams come true.

3.3 Scenario 3: milanese imbruttito

Rachele is a business woman who has to go the job meeting.
She wants to travel comfortly, and her car doesn't work, so she uses MyTaxiService for the first time.
She registers in the system, logs in, and requests a taxi, which will arrive in time.
Once arrived, Rachele has to pay the taxi driver: she opens the app, tap on «Ride payment» button, and choose the credit card method, but she didn't fill the releated field in the registration phase.
No problem: the app redirects her to a form containing the following fields: circuit, credit card number, owner name, owner surname, expiration date and CVV code.
She fills the form and goes to the job meeting.
At 18:00 she goes out from her office, reuses the app, and she goes to the Noon in order to have an happy hour with her friends.
She was very happy because she didn't have to provide her credit card data another time!

Chapter 4

Modeling

4.1 UML Model

4.1.1 Use Cases

4.1.1.1 Overview

Starting from the scenarios described before, here are the general use cases[1]:

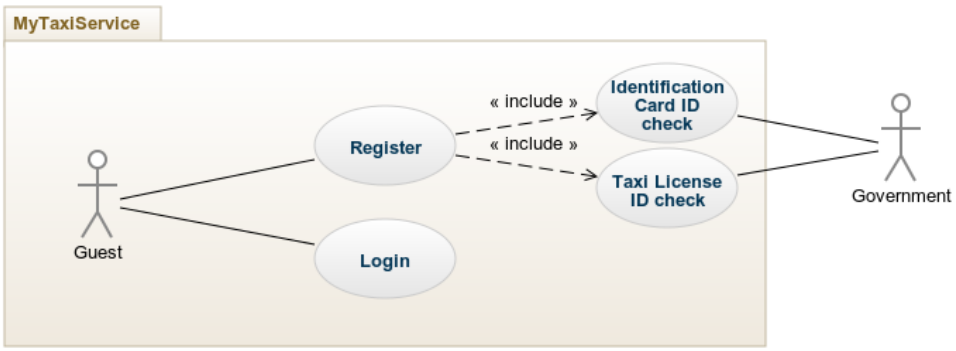

Figure 4.1: Detailed guest use case model

[1] In the next pages it is used the website version of the system, but there are similarities between the website and the app, so it is not a big deal.

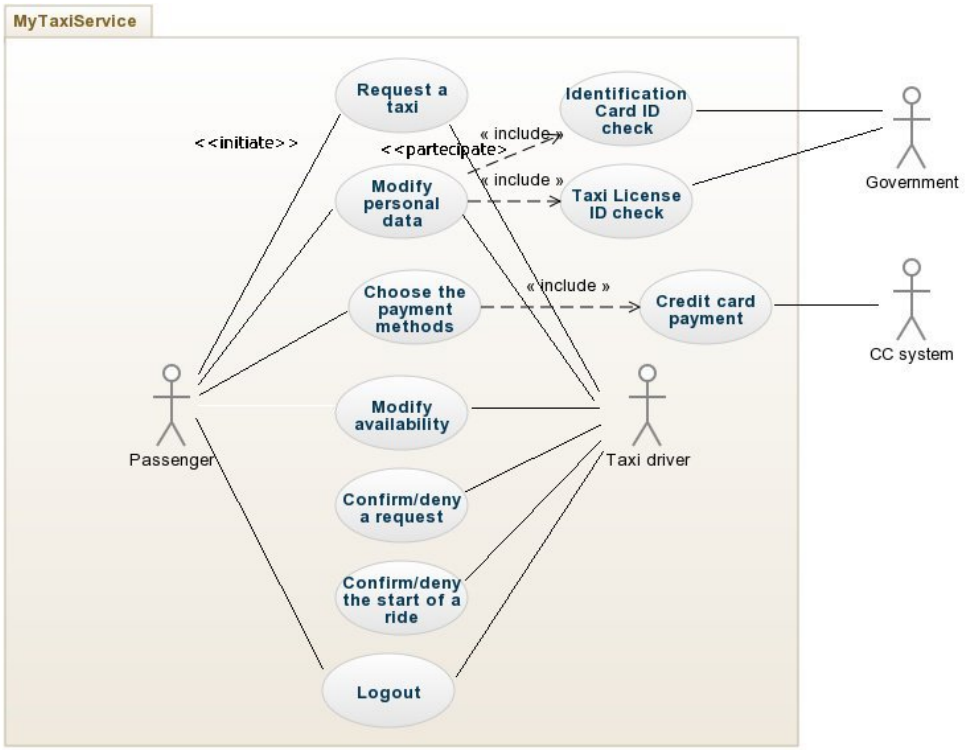

Figure 4.2: Detailed passenger and taxi driver use case model

4.1.1.2 Register (passenger)

Table 4.1: Use case model of guest registration in case of passenger

Actors:	Guest
Pre-conditions:	The guest is not a returning user
Events flow:	1. The guest arrives on the home page of MyTaxiService 2. He/she chooses to register himself/herself clicking the «Register» button 3. He/She types name, surname, email address, password, confirmation password, user type equal to «passenger» and cellphone number. 4. He/she clicks on «Confirm» button 5. The system checks the fields (see Exceptions) 6. If there are no errors, the system sends a confirmation mail to the prompted email address
Post-conditions:	The guest is registered, so there is an account releated to his/her data which expires in 1 day whether the guest doesn't click the confirmation link sent via mail by the system. Otherwhise he/she can login.
Exceptions:	An error message occours whether: - the 2 passwords are not tha same - there is already an account with the same email address - there is at least one mandatory field not filled

4.1.1.3 Register (taxi driver)

Table 4.2: Use case model of guest registration in case of taxi driver

Actors:	Guest
Pre-conditions:	The guest is not a returning user
Events flow:	1. The guest arrives on the home page of MyTaxiService 2. He/she chooses to register himself/herself clicking the «Register» button 3. He/she types name, surname, email address, password, confirmation password, user type equal to «taxi driver», credit card number, identification card ID and taxi license ID 4. He/she clicks on «Confirm» button 5. The system checks the fields (see Exceptions) 6. If there are no errors, the system sends a confirmation mail to the prompted email address.
Post-conditions:	The guest is registered, so there is an account releated to his/her data. Hence he/she can login.
Exceptions:	An error message occours whether: - the 2 passwords are not tha same - there is already an account with the same email address - there is at least one mandatory field not filled - the identification card ID is not valid (government API) - the taxi license ID is not valid (government API)

4.1.1.4 Login

Table 4.3: Use case model of user login

Actors:	Guest
Pre-conditions:	The guest has confirmed his/her registration through the confirmation mail
Events flow:	1. The guest arrives on the home page of MyTaxiService 2. He/she chooses to login himself/herself clicking the «Login» button 3. He/she types email address and password 4. He/she clicks on «Confirm» button 5. The system checks the fields (see Exceptions) 6. If there are no errors, the system redirect to: 6a. a taxi home page whether the system knows his/her identification card ID, taxi license ID, but not the cellphone number 6b. a passenger home page whether the system knows his/her cellphone number, but not the identification card ID and the taxi license ID 6c. an intermediate page whether the system knows both his/her cellphone number, identification card ID and taxi license ID, which shows him/her only 2 buttons: «Login like a passenger» (redirection to the passenger home page) and «Login like a taxi driver (redirection to the taxi driver home page).
Post-conditions:	The guest is a user.
Exceptions:	An error message occours whether the pair email address+password does not corrispond to a registrated users.

4.1.1.5 Request a taxi

Table 4.4: Use case model about the request of a taxy made by a passenger

Actors:	Passenger, taxi driver
Pre-conditions:	The users are logged in
Events flow:	1. A taxi driver and a passenger, once logged in, are arrived on thei releated home pages 2. The taxi driver sets his/her availability clicking the «Availability» button, and he/she is redirected to a page in which there are the pending taxi requests provided by passengers in the taxi driver zone 3. The passenger chooses to request a taxi clicking the «Request a taxi» button 4. The system starts a timeout of 10 minutes and detects the zone from which the request is sent through his/her GPS position, and then it searchs for an alavailable taxi in the releated taxi queue. 4. The taxi driver sees the request and accepts it (this action trigger his/her availability status from «free» to «occupied») 5. The system redirects him/her to a page in which there is the following question: «Is the ride started?» 6. The system send a notification to the passenger about him/her request accepting containing the taxi driver ID, the ETA (Google API), and a link in order to pay the ride at the end of it 7. The taxi driver reach him/her, and answers to the previous question clicking «Yes» (this action triggers a ride timer in order to calculate the payment) 8. Once arrived, the passenger click on the link suggested in the prevoius notification, chooses the payment method, and after a few minutes (payment transaction API checking), says goodbye to the taxi driver (this action triggers the availability status of the taxi driver from «occupied» to «free»)
Post-conditions:	The taxi driver is available.
Exceptions:	- If there are no available taxi in the zone, the system keep on searching for 10 minutes, then it notifyies the passenger the message «Sorry, retry later» - Either the taxi driver takes more than 3ETA to answer the «Is the ride started?» question, or if he/she answer «No», the system automatically cancel the ride, resets the avalability taxi driver status to «free», and notifies the passenger with the following message: «Sorry, something went wrong. Do a request again.» - Either the taxi driver refuses a passenger request, or doesn't answer in 30 seconds, the system automatically redirects the request to the following available taxi in the same zone, and moves the taxi driver to the last position of the queue. - If the payment transaction fails, the system prompt a message to the passenger «Sorry, you have to use cash»

4.1.2 Class Diagram

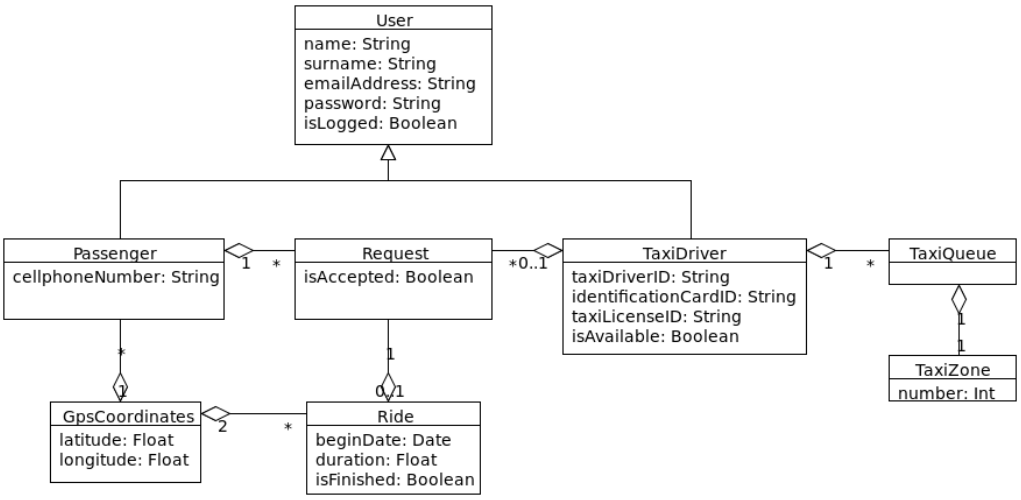

Figure 4.3: High level of class diagram model

4.2 Alloy Model

4.2.1 Model code

```
open util/boolean
//signatures
sig Stringa{} abstract
sig User {
        name: one Stringa,
        surname: one Stringa,
        emailAddress: one Stringa,
        password: one Stringa,
        isLogged: one Bool
}
sig Passenger extends User {
        cellphoneNumber: one Stringa,
        currentPosition: one GpsCoordinates
}
sig TaxiDriver extends User {
        taxiDriverID: one Stringa,
        identificationCardID: one Stringa,
        taxiLicenseID: one Int,
        isAvailable: one Bool
```

```
}
sig Float{}
sig GpsCoordinates {
        latitude: one Float,
        longitude: one Float
}
sig Request{
        passenger: one Passenger,
        taxiDriver: lone TaxiDriver,
        ride: lone Ride,
        isAccepted: one Bool
}
sig Ride {
        origin: one GpsCoordinates,
        destination: one GpsCoordinates,
        beginDate: one Int, // since there is no Date type, I use Julian D
        isFinished: Bool
}
sig TaxiQueue {
        taxiDrivers: set TaxiDriver
}
sig TaxiZone {
        number: one Int,
        queue: lone TaxiQueue
}

//facts
fact NoNeither {
        Passenger + TaxiDriver = User
}
fact UniqueTaxiDrivers {
        no t1, t2: TaxiDriver | ((t1 != t2) and
(t1.taxiDriverID = t2.taxiDriverID or
t1.identificationCardID = t2.identificationCardID or
t1.taxiLicenseID = t2.taxiLicenseID ))
}
fact UniqueTaxiZone {
        no tz1, tz2: TaxiZone | (tz1 != tz2 and
                        tz1.number = tz2.number)
}
```

```
fact UniqueUsers {
        no u1, u2: User | (u1 != u2 and
                        (u1.emailAddress = u2.emailAddress))
}
fact UniquePassengers {
        no p1, p2: Passenger | (p1 != p2 and
                        (p1.cellphoneNumber = p2.cellphoneNumber))
}
fact NoMoreThanOneRequestPerRide {
        no r1,r2: Request | (r1!=r2 and r1.ride=r2.ride)
}
fact NoMoreThanOneRidePerTimeForEveryTaxiDriver {
        no r1,r2: Request | (r1.taxiDriver = r2.taxiDriver and
}
fact NoMoreThanOneRidePerTimeForEveryPassenger {
        no r1,r2: Request | (r1.passenger = r2.passenger and
}
fact OriginAndDestinationMustBeNotEqualForTheSameRide{
        no r: Ride | (r.origin=r.destination)
}

//assertions
assert AcceptedRequestHasOneTaxiDriverAndOneRide{
        all r:Request | r.isAccepted=True
                        implies (one r.taxiDriver and  one r.ride)
}
assert NotAcceptedRequestHasNoTaxiDriver{
        all r:Request | r.isAccepted=False implies (no r.taxiDriver)
}
assert FinishedRideHasDuration{
        all r: Ride | r.isFinished=True implies (one duration)
}
assert NotFinishedRideHasNoDuration{
        all r: Ride | r.isFinished=False implies (no duration)
}
assert AvailableTaxiDriversAreInQueue {
        all t: TaxiDriver | t.isAvailable=True
                        implies (some tq: TaxiQueue | t in tq.taxiDrivers)
}
assert NonAvailableTaxiDriversAreNotInQueue {
```

```
        all t: TaxiDriver | t.isAvailable=False
                        implies (no tq: TaxiQueue | t in tq.taxiDrivers) }

// predicates pred show(){
        #Passenger >= 2
        #Ride >= 1
        #TaxiDriver >= 2
        #{t: TaxiZone | #t.queue >=1 } >= 1
        all p: Passenger | (some r: Request | p in r.passenger)
}
run show for 10
```

4.2.2 Generated Worlds

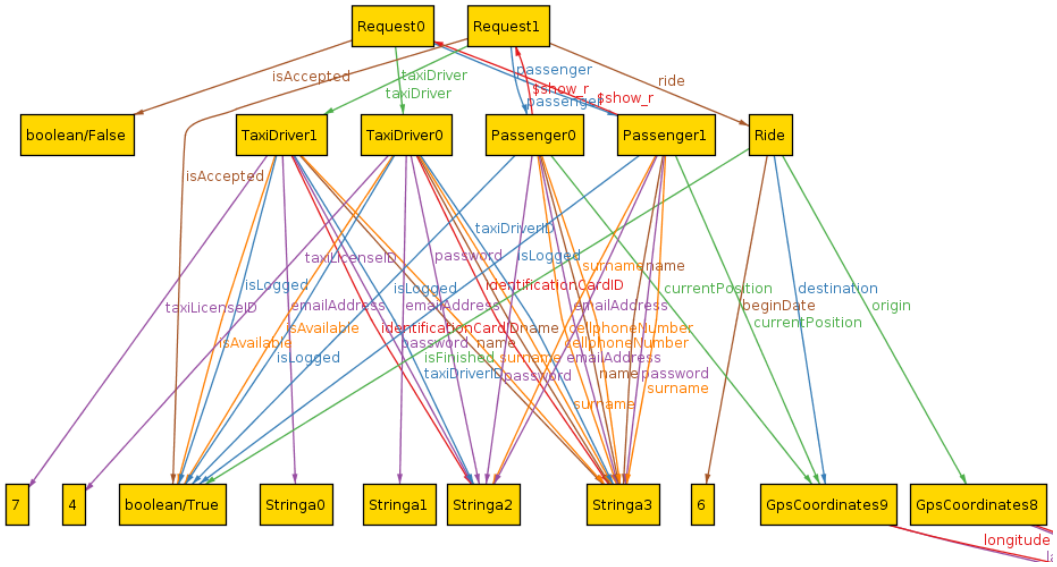

Figure 4.4: One World generated through Alloy- part 1

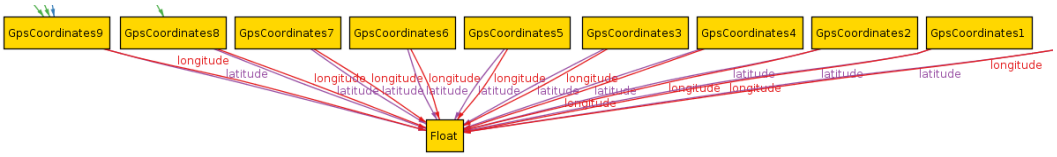

Figure 4.5: One World generated through Alloy - part 2

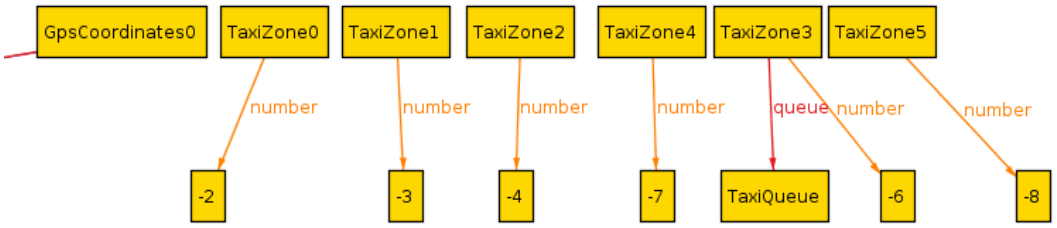

Figure 4.6: One World generated through Alloy - part 3

Appendix A

Document Informations

A.1 Effort

Until 9th November 2015, approximately **50 hours** have been spent making this document.

A.2 Tool Used

- **LyX**: www.lyx.org
- **GenMyModel**: www.genmymodel.com
- **Alloy Analyzer**: http://alloy.mit.edu/alloy