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)

6th November 2015

Moreno SARDELLA - 859239

Academic Year 2015–2016

Contents

List of Figures

Introduction

0.1 Revision History

Table 1: Detailed history of the document revisions

Name	Date	Note
RASD-version3	28/02/2015	- The project no longer refers to a specific city. - The project no longer refers to Google APIs. - Constraint relaxations about timeouts
RASD-version2	11/11/2015	Document completed (Alloy model added)
RASD	06/11/2015	Document creation

0.2 Purpose

This document explain an high level *Requirement Analysis and Specification* of MyTaxiService project, in order to explicit the releated software behavior, according to the stakeholders needs.

0.3 Scope

The government of a city 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 identication card and taxi license identiers 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 (e.g. 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 identier 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 notication 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.

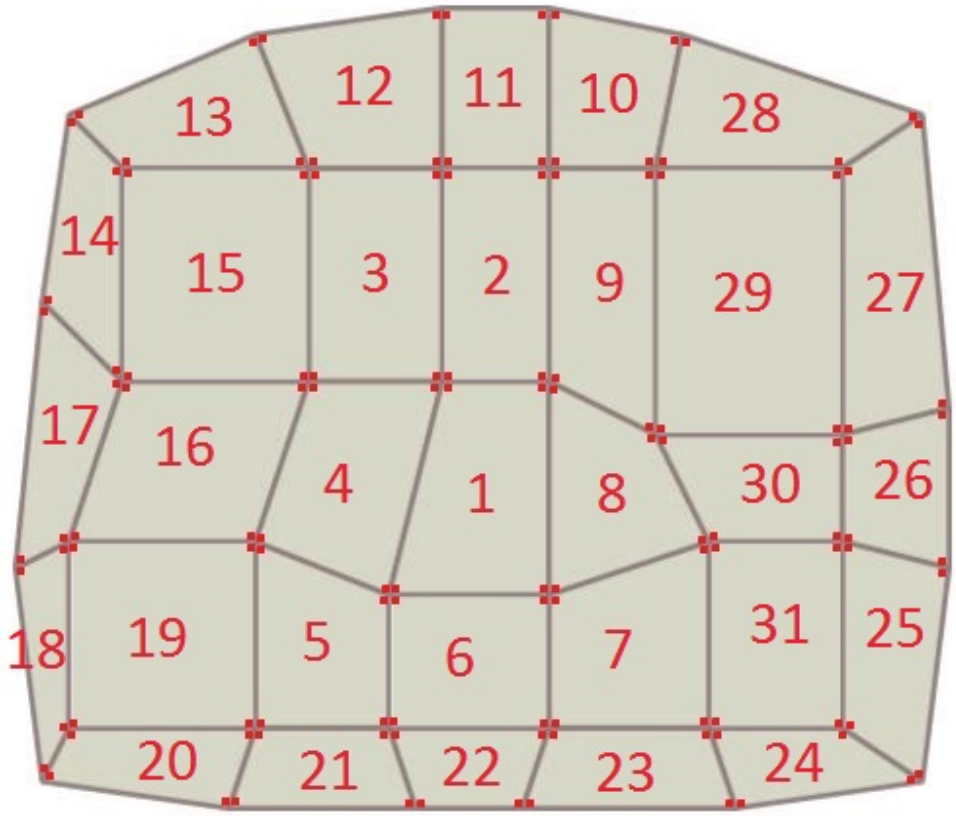

Figure 1: Example of taxi zones

0.4 Definitions, acronyms and abbreviations

0.4.1 Definitions

- **Login**: the procedure through which a guest, entering his/her credentials, authenticates as a user.
- **Logout**: the procedure though which a user disconnects himself/herself from the system.
- **System:** the whole MyTaxiService service.

0.4.2 Acronyms

- **CC**: credit card
- **DD**: Design Document
- **ETA**: Estimated Time of Arrival
- **GUI**: Graphic User Interface
- **OS**: Operating System
- **RASD**: Requirements And Specications Document
- **TBD**: To Be Defined
- **UML**: Unied Modeling Language
- **WRT**: With Respect To

0.4.3 Abbreviations

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

0.5 Reference documents

- [1] Assignments 1 and 2
- [2] IEEE Recommended Practise for SoftwareRequirements Specifications

0.6 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. Specic 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

If there is no GPS connection, the mobile app and the website allow only to login and to sign up.

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. Most of the users has an internet and a GPS connection

1.5.2 Dependencies

The system uses:

- government APIs in order to check
  - the identication card ID
  - the taxi license ID
- credit card APIs in order to check
  - the payment transactions through credit cards
- mapping APIs in order to
  - compute the ETA

Chapter 2

Specific Requirements

2.1 Actors

- **Guest**: a person using the service that is not 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 fulfill a passenger request. Also a taxi driver must have a taxi license.

2.2 Functional Requirements

1. Allow a guest to:[1]
   (a) register like a passenger - giving name*, surname*, email address*, password*, cellphone number* and credit card data
   (b) register like a taxi driver - giving name*, surname*, email address*, password*, identification card ID*, taxi license ID* and credit card data
   (c) login - giving email address and password
2. Allow a user to:
   (a) modify his/her data
   (b) log out from the system
3. Allow a passenger to:
   (a) request a taxi

[1] *: mandatory field

(b) pay a ride through credit card

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) conrm/deny the start of the ride with the passenger

   (d) confirm the end of the ride

5. The system automatically has to:

   (a) assign an ID to every taxi driver

   (b) assign dynamically every available taxy driver to a zone queue receiving her/his GPS coordinates every n minutes (TBD in the design document)

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

   (d) send to the passenger a notication about her/his request answer (giving taxi driver ID and ETA whether it is confirmed)

   (e) move in the last position of the queue a taxi driver who either has denied a request or has not answer it in m minutes (TBD in the design document).

   (f) start a taximeter[2] when the ride starts, and stop it when it ends

   (g) compute the bill when the ride ends

2.3 Non-Funtional Requirements

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

1. **Portable**: compatible with all smarthphone OS in the current market.

2. **Usable**: easy to use by every user (a usability study must be done before the GUI design phase).

3. **Reusable**: able to adapt to possible any kind of city (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).

4. **Maintainable**: developed maintainable in order to: fix a bug, add new features, improve usability, increase performance, improve coupling and readability, as easily as possible.

[2] it consists in 2 timers: the timing and the fare

5. **Performant**: able to manage at least y requests per second (TBD in the design document).

6. **Secure**: able to ensure protection from privacy violation 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 in the last few days were in italian.
He find a solution: MyTaxiService.
He goes on the home website, click on the registration button and fills the registration form. Then, accepting the confirmation email, he logs in the website, turns on the GPS and requests a taxi. After just 1 minute he receives a SMS:

«Taxi 102689 will 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 old, 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! Ha ha ha!».*
Bob took the opportunity to exploit the phenomena, so he dowloaded the app and he registered himself.
The following saturday, at 10:00 pm, he turned on his GPS smartphone and he launched the MyTaxiService app. Once logged, he tapped the availability button.

After a while, a push notication on his smartphone notified him *«There is a request! Do you want to accept?»*. Bob accepted it, and the app showed him *«A passenger is waiting you at: c.so Como 7»* with two button: *Start* and *Cancel*.
He went to the received position, but there was no passenger waiting for him! Helpfully the app takes into account this scenario, so he tap on the *Cancel* button, and returns available.
Next times he was more lucky, and he got used to do this job every saturday night. 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 she doesn't want to use the underground, it's for poor people!
Hence 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 *Pay* button, and choose the credit card method, but she didn't fill the releated fields in the registration phase. She fills the form and goes to the job meeting.
At 06:00 pm she goes out from her office, opens the app, and goes to the Noon in order to have an happy hour with her chic 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

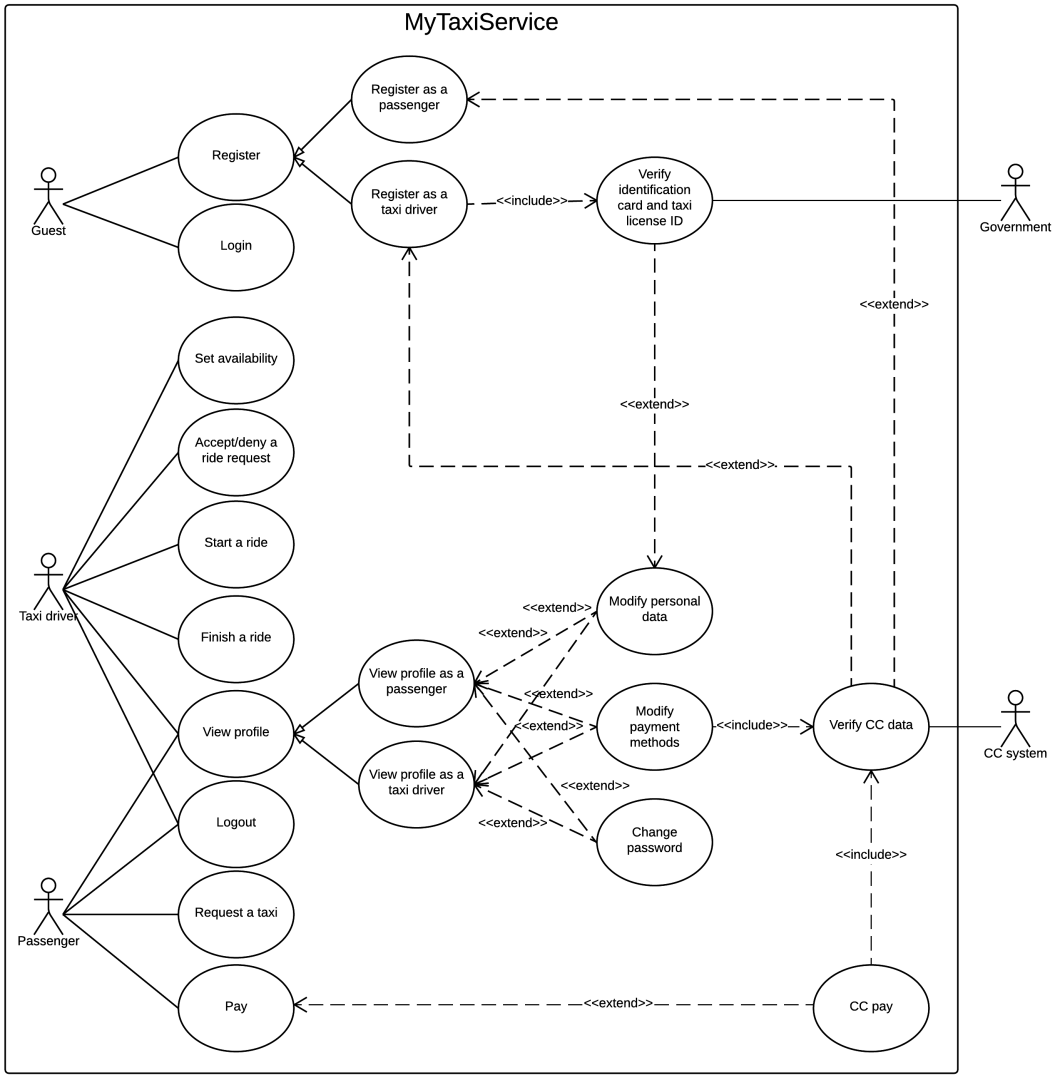

Figure 4.1: Use case model of MyTaxiService project

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 opens the MyTaxiService app 2. He/she chooses to register himself/herself tapping the *Register* button 3. He/She types name, surname, email address, password, conrmation password, the cellphone number and eventually the credit card data. 4. He/she taps on *Complete* 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.
Exceptions:	An error message occours whether: - the 2 passwords are not tha same - there is already an account with the same email address - the credit card data are not valid (CC API) - 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 opens the MyTaxiService app 2. He/she chooses to register himself/herself tapping the *Register* button 3. He/She types name, surname, email address, password, conrmation password, the idetification card ID, the taxi license ID and eventually the credit card data. 4. He/she clicks on *Complete* 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.
Exceptions:	An error message occours whether: - the 2 passwords are not tha same - there is already an account with the same email address - the credit card data are not valid (CC API) - the identification card ID is not valid (government API) - the taxi license ID is not valid (government API) - there is at least one mandatory field not filled

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 opens the MyTaxiService app 2. He/She types email address and password 3. He/she chooses to login himself/herself tapping the *Login* button 4. The system checks the fields (see Exceptions) 5. If there are no errors, the app redirect him/her to the personal home page
Post-conditions:	- The guest becomes a user - The system starts a user session
Exceptions:	An error message occours whether: - there is at least one mandatory field not filled - the pair email address+password does not corrispond to a registrated user - there is already a user logged with the same credentials

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 user is logged in
Events flow:	1. The passenger taps the *Request a taxi* button in his/her home page 2. He/she confirms 3. The system looks for an available taxi driver 4. Once founded, the app shows the taxi driver ID and the ETA 5. The passenger can wait or cancel his/her request tapping the *Cancel* button.
Post-conditions:	The taxi driver becomes unavailable.
Exceptions:	- If there are no available taxi in the zone, the system keep on searching it in neighbor zones for n minutes (TBD in the design document), then it notifies the passenger - Either the taxi driver refuses a passenger request, or doesn't answer in m seconds (TBD in the design document), the system automatically redirects the request to the following available taxi, and moves the taxi driver to the last position of the queue.

4.1.1.6 Finish a ride

Table 4.5: Use case model about the request of a taxy made by a passenger

Actors:	Passenger, Taxi driver
Pre-conditions:	The user is logged in
Events flow:	1. The taxi driver, once brought the passenger to the destination. taps the *Finish* button and confirms 2. The system notify the passenger, who has to pay 3. The passenger pays
Post-conditions:	The taxi driver becomes available.
Exceptions:	If the passenger chooses the CC payment, an error message occours whether the transaction goes wrong

4.2 Class Diagram

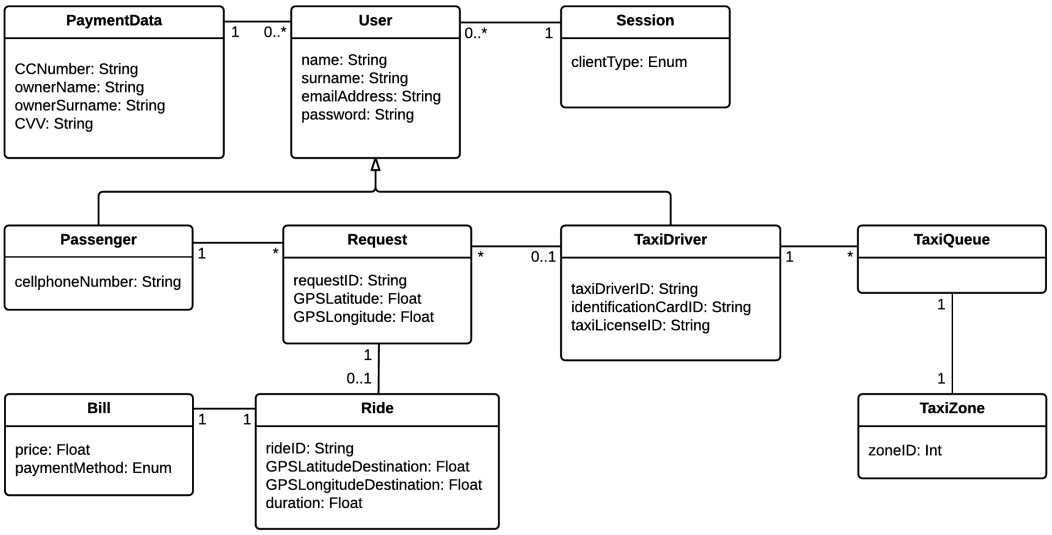

Figure 4.2: High level of class diagram model

4.3 Alloy Model

4.3.1 Model Code

```
open util/boolean
//signatures sig Stringa{} abstract
sig User {
        name: one Stringa,
        surname: one Stringa,
        emailAddress: one Stringa,
        password: one Stringa,
        paymentData: set PaymentData }
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 PaymentData{
        CCNumber: Stringa,
        ownerName: one Stringa,
```

```
        ownerSurname: one Stringa ,
        CVV: one Int }
sig Float{}
sig GpsCoordinates {
        latitude: one Float ,
        longitude : one Float }
sig Request {
        passenger: one Passenger,
        taxiDriver: lone TaxiDriver,
        ride: lone Ride,
        origin: one GpsCoordinates }
sig Ride {
        destination: one GpsCoordinates,
        duration: Float }
sig TaxiQueue {
        taxiDrivers: set TaxiDriver }
sig TaxiZone {
        zoneID: 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.zoneID = tz2.zoneID) }
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)}
fact NoMoreThanOneRidePerTimeForEveryPassenger {
        no r1, r2: Request | (r1.passenger = r2.passenger)}

//assertions
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
```

4.3.2 Generated World

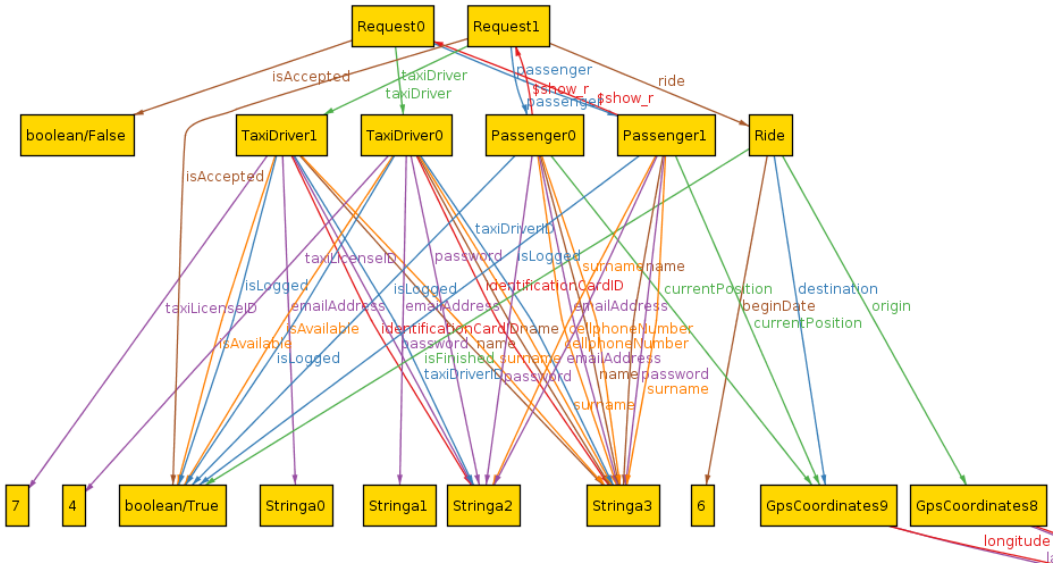

Figure 4.3: One World generated through Alloy - part 1

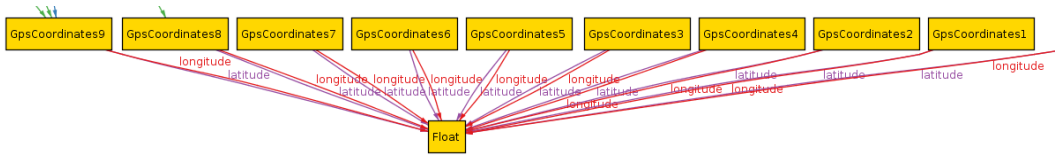

Figure 4.4: One World generated through Alloy - part 2

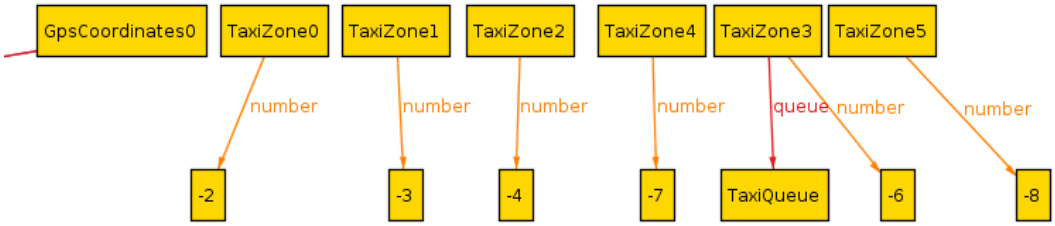

Figure 4.5: One World generated through Alloy - part 3

Appendix A

Document Informations

A.1 Effort

Approximately **55 hours** have been spent making this document.

A.2 Tool Used

- **LyX**: www.lyx.org
- **LucidChart**: https://www.lucidchart.com/
- **Alloy Analyzer**: https://alloy.mit.edu/alloy