Politecnico di Milano
A.Y. 2015/2016

My Taxi Service

Requirement Analysis and Specification Document

Bernardis Cesare matr. 852509 Dagrada Mattia matr.852975

November 6, 2015

Contents

1 Introduction

1.1 Purpose

The main goal of this document is to completely describe the system in terms of functional and non-functional requirements, to analyse the real need of the customer modelling the system, to show the constraints and the software limits and simulate the typical use cases that will occur after the development. This document is intended to all developers and programmers who have to implement the requirements, to the system analysts who want to integrate other system with this one, and could also be used as a contractual basis between the customer and the developer.

1.2 Actual system

The government of a large city wants to optimize its taxi service. We suppose that the actual taxi service is based on simple phone calls from customers to the taxi's call centre.

1.3 Scope

The aim of this project is to create a brand new taxi application that is used by both the taxi drivers and the passengers to access the taxi service. Passengers can access the service either via mobile or web application. They can request a taxi without having to register to service but they have to insert personal information while registered passengers, once logged in, can directly request a taxi. The system confirms a taxi request by sending the passenger the taxi code only when a taxi is found and an estimated arrival time. Registered passengers can also make taxi reservations and they will receive the code as soon as a taxi is found available. Taxi drivers can access the service only from mobile application. Once logged in they can give the system their availability, or revoke it, and they will receive from the system ride requests that they can either accept or refuse. The login interface of the application is shared by passengers and taxi drivers. The system has the city map divided into areas of $2km^2$ and holds a taxi queue in each area. It receives GPS coordinates from taxis, it assigns them to their corresponding area queue, placing them in the last position. Following a FIFO (First In First Out) logic, the first taxi receives a request and is then removed from the queue. In case of rejection, the taxi is placed in the last position of the queue.

1.4 Actors

Guest a guest is able to request a taxi without having to be registered to the service, but simply adding some essential personal information. A

guest can register to the service filling in a registration form, becoming a Registered Passenger.

Registered Passenger a registered passenger, once logged in, is able to request a taxi without having to add additional personal information. A registered passenger can also make reservations, specifying origin and destination of the ride, at least 2 hours before the desired time for the ride.

Taxi Drivers a taxi driver, once logged in, has a different user interface than registered user, from which he will receive ride requests that he can accept or refuse. He will also be able to give his availability, which means that he is willing to pick up ride requests.

1.5 Goals

These are the goals of MyTaxiService application:

- Permit a guest to register to the service.
- Permit a guest to request a taxi.
- Permit a guest to sign in and become a registered passenger
- Permit a registered passenger to require a taxi.
- Permit a registered passenger to make a reservation.
- Permit a registered passenger to cancel a reservation.
- Permit a taxi driver to give the system his availability.
- Permit a taxi driver to revoke his availability.
- Permit a taxi driver to accept a ride request.
- Permit a taxi driver to refuse a ride request.

1.6 Reference Documents

- Specification Document: MyTaxiService Project A.Y.2015-2016.
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

1.7 Document overview

This document is structured as following:

1. **Introduction**: this section represents a generic description of the project, highlighting actors and goals.
2. **Overall Description**: this section gives further informations about the project, focusing more on what are the assumptions made about the software and its constraints.
3. **Specific Requirements**: this section lists the requirements of the software, both functional and non-functional ones. There are also some typical scenarios and use cases related to sequence diagrams along with a class diagram.
4. **Alloy Modelling**: this section contains Alloy code and Alloy worlds.
5. **Appendix**: this section contains extra information and also which software and tools has been used to write this document.

2 Overall Description

2.1 Product perspective

We want to release both a web and a mobile application, avoiding deep integrations with other existing systems in order to keep a high level of compatibility, especially with the mobile application, that we want to release for all major mobile operative systems. The applications will not have any internal interface for administration but they will be only user based. The applications will provide APIs for a faster, safer and better integrated implementation of new features, based on the basic functionalities offered by the system.

2.2 User characteristics

We expect to have two different types of user with distinct experiences.

Passenger who needs to call a taxi for a ride. Passengers can simply be guests or registered users, that have advanced features. Passengers must have access to the Internet and be able to use a web browser or have installed our mobile application on their smartphones;

Taxi Driver who wants to offer his service through our system. Taxi Drivers must have access to Internet and have installed our mobile application on their smartphones.

2.3 Constraints

2.3.1 Regulatory policies

Personal information (locations included) obtained from the user will be stored in our databases if strictly necessary, to provide the best possible experience with our service. No information will be disclosed to any other company or used for any other purpose.

2.3.2 Hardware limitations

MyTaxiService web application will be available on every device with an Internet connection and a browser installed. The mobile application (necessary for taxi drivers) will have the following requirements:

- 256MB total RAM
- 50MB of free space on Disk
- Internet Connection
- Operative System:

- Android 2.3 "Gingerbread" or later
- Windows Mobile 7.0 or later
- iOS 6.0 or later

It is also mandatory to activate the GPS.

2.3.3 Interfaces to other applications

MyTaxiService will provide an API library to allow external developers to integrate their applications with our system.

2.3.4 Parallel operation

The system must support parallel operations by different users. Information integrity is fundamental to provide a top-notch service, so the mechanism of supply and demand has to be highly affordable.

2.3.5 Documents related

- RASD (Requirements and Analysis Specification Document)
- DD (Design Document)
- User's Manual
- Testing report

2.4 Assumptions and Dependencies

- There is not an administrator or a privileged user. We think that a hierarchy of users is not necessary to keep the system safe;
- There is not any dependence between users;
- Guest user is identified through his public IP address;
- The position of the user can always be determined, via browser or GPS (see HTML5 Geolocation), and has a good approximation. Otherwise the service will not be accessible;
- A taxi driver can access the service only if he is signed up. The registration is made directly by the taxi provider, therefore taxi drivers will simply have to sign in in order to access the service. The taxi provider assigns an ID number to taxis.
- Through the ID number of a taxi it is possible to retrieve some informations (like the location) of the taxi itself;

- A taxi driver can give and revoke his availability in every moment;
- If a taxi driver does not accept a call within 1 minute after the notification, a rejection will be automatically recorded in the database and the request will be forwarded to another taxi;
- After 3 consecutive rejections (voluntary or involuntary) the taxi availability will be revoked automatically;
- When a user (registered or guest) requests a taxi, his position is taken automatically. If he does not agree, his request will not be forwarded;
- After a request, a user (registered or guest) can not make other requests in the next 30 minutes;
- A user must be signed up to make a reservation;
- Reservations must be taken at least 2 hours in advance from the time of taxi request, otherwise the reservation will not be accepted;
- Reservations must have 30 minutes of distance between them, otherwise they will not be accepted
- A user that registered a reservation for a taxi can not make requests within 30 minutes before the reservation;
- A reservation can be cancelled until 10 minutes before the request time.

2.5 Future possible implementation

The taxi sharing option: this means that the user is ready to share a taxi with others if possible, thus sharing the cost of the ride. In this case the user is required to specify the destination of all rides which he/she wants to share with others. If others are willing to start a shared ride from the same zone going in the same direction, then the system arranges the route for the taxi driver, defines the fee for all persons sharing the taxi and informs the passengers and the taxi driver.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.1.1 Home Page - Mobile Application

Figure 1: Home Page - Mobile Application

3.1.1.2 Home Page - Web Application

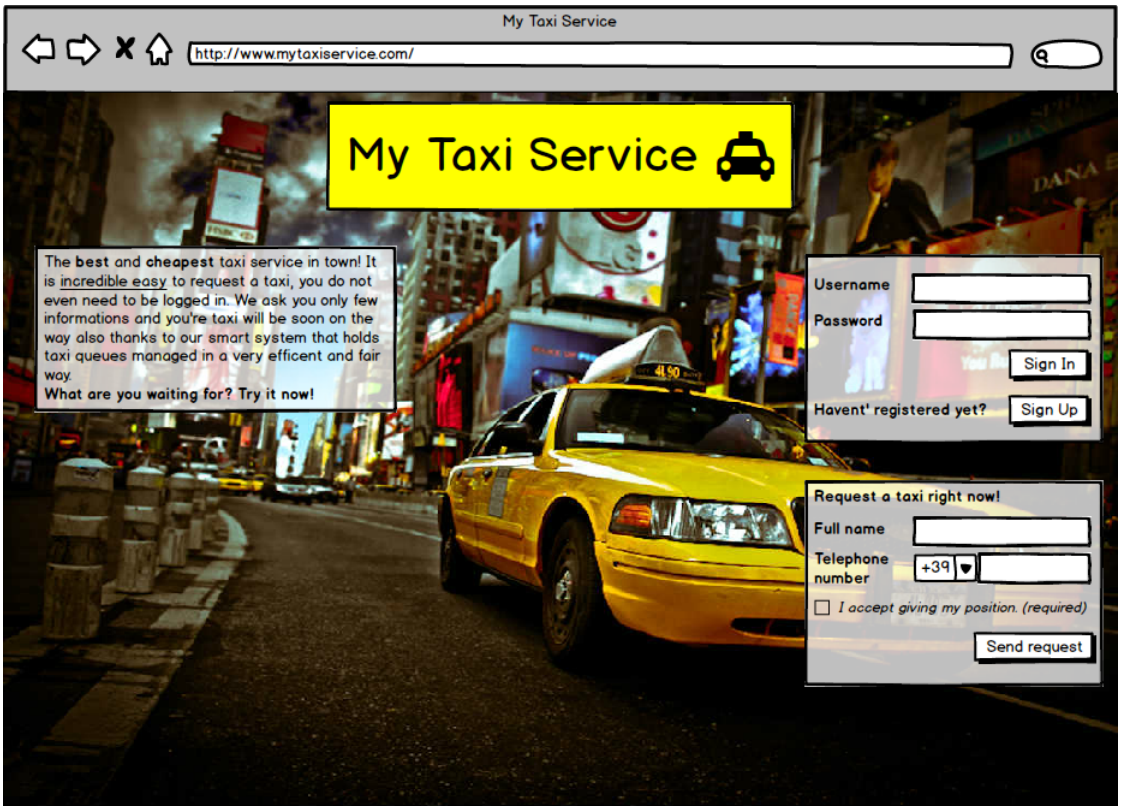

Figure 2: Home Page - Web Application

3.1.1.3 Registration Form - Mobile Application

Figure 3: Registration Form - Mobile Application

3.1.1.4 Registration Form - Web Application

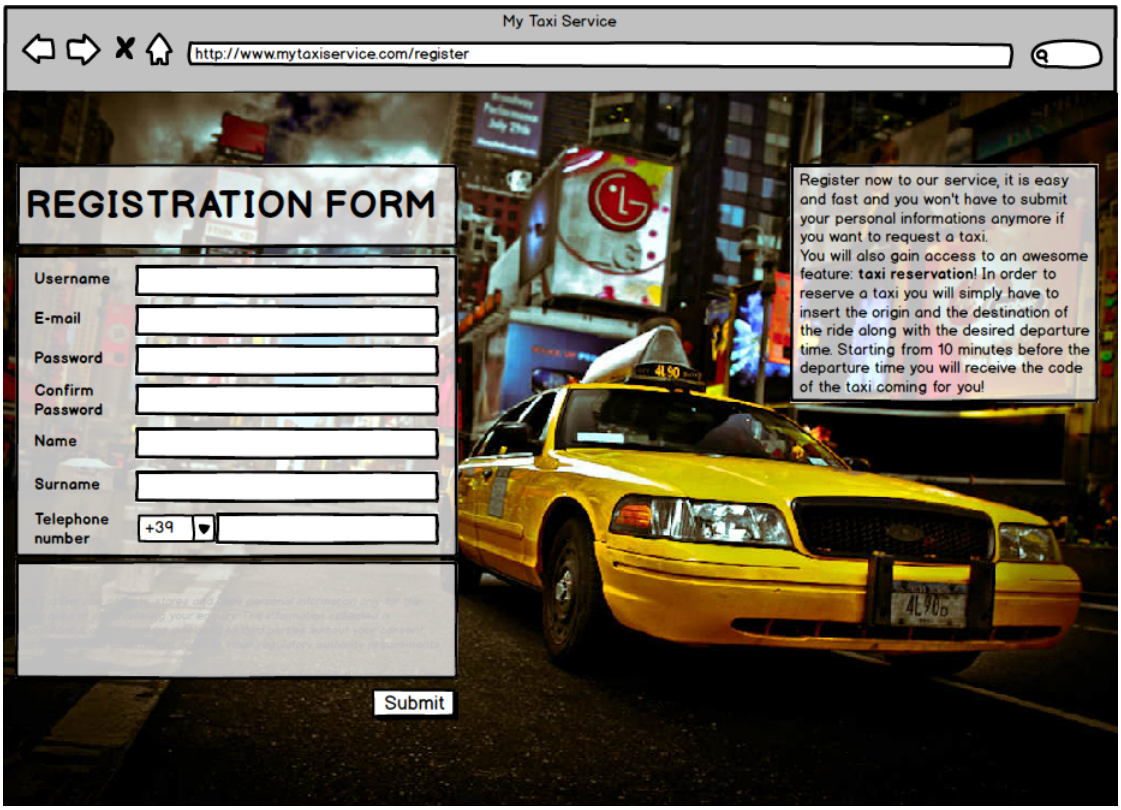

Figure 4: Registration Form -Web Application

3.1.1.5 Passenger Area - Mobile Application

Figure 5: Passenger Area - Mobile Application

3.1.1.6 Passenger Area - Web Application

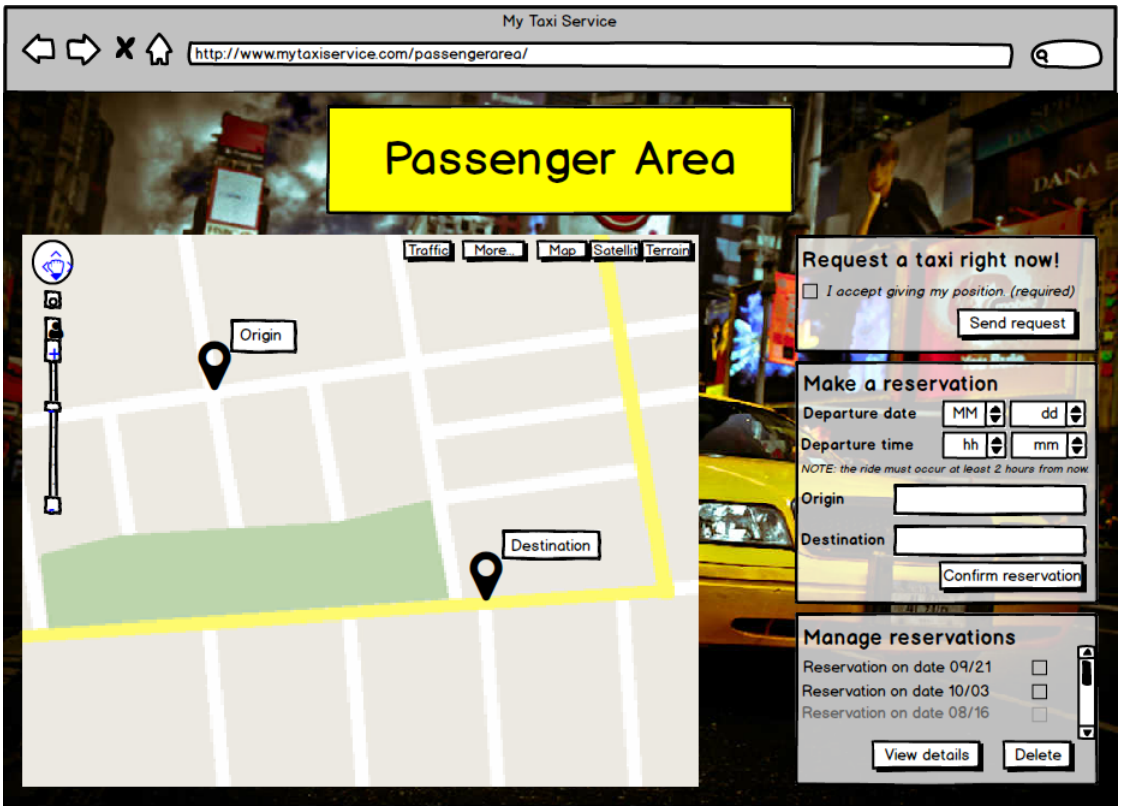

Figure 6: Passenger Area - Web Application

3.1.1.7 Request Confirmation - Mobile Application

Figure 7: Request Confirmation - Mobile Application

3.1.1.8 Request Confirmation - Web Application

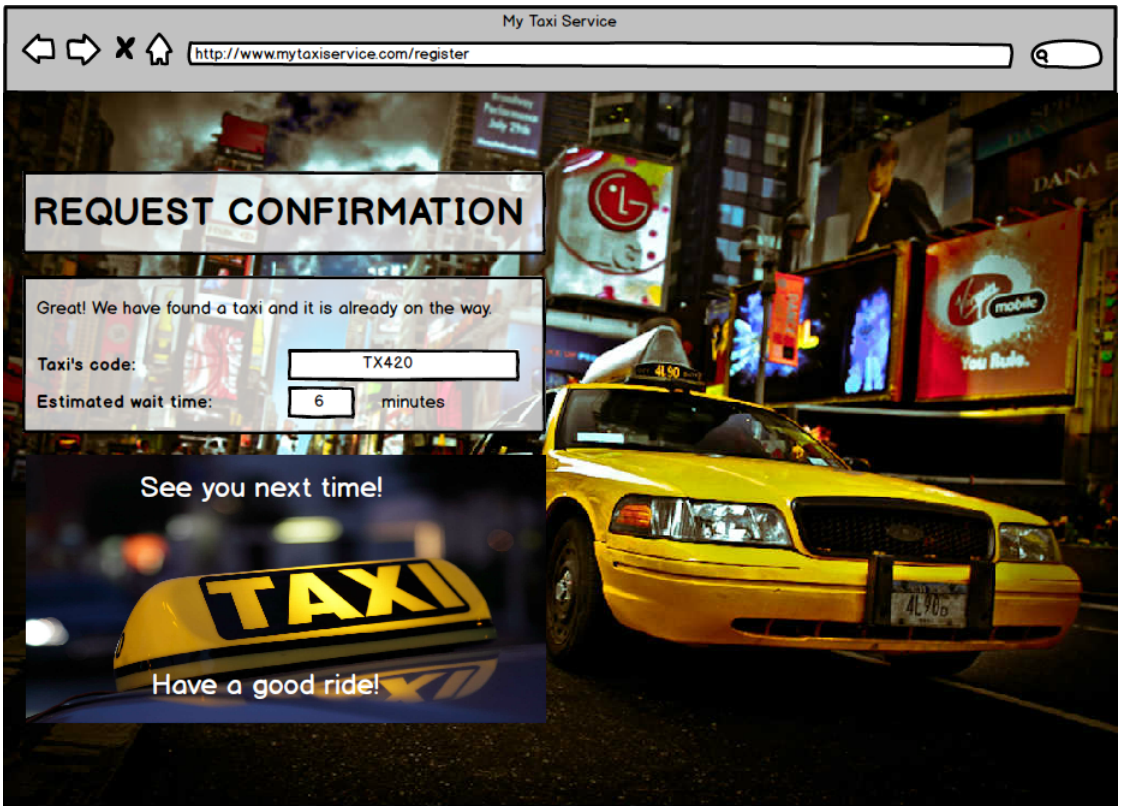

Figure 8: Request Confirmation - Web Application

3.1.1.9 Taxi Driver Area - Mobile Application

Figure 9: Taxi Driver Area - Mobile Application

3.1.2 API Interfaces

For taxi waiting time and other minor features we decided to use Google Maps API. It is a very powerful library that perfectly fits our needs. When a taxi accepts a request, the system retrieves its position and, through those APIs, is able to process either the hypothetical route that the taxi will follow to reach the passenger's position and, consequently, the arrival time. The system will, then, forward those information to the passenger.

3.1.3 Hardware Interfaces

Our project includes a web and a mobile application, so it does not require any external hardware interface.

3.1.4 Software Interfaces

- Database Management System (DBMS):
  - Name: MySQL.
  - Version: 5.6.21
  - Source: http://www.mysql.it/
- Java Virtual Machine (JVM).
  - Name: JEE
  - Version: 7
  - Source: http://www.oracle.com/technetwork/java/javaee/tech/index.html
- Application server:
  - Name: Glassfish.
  - Version: 4.1.
  - Source: https://glassfish.java.net/
- Operating System (OS).
  - Application must be able to run on any SO which supports JVM and DBMS specified before.

3.1.5 Communication Interfaces

Protocol	Application	Port
TCP	HTTPS	443
TCP	HTTP	80
TCP	DBMS	3306 (default)

3.2 Functional Requirements

3.2.1 Permit a guest to register to the service

R1: The system shows the registration form to the user when he presses the sign up button.

R2: When the user inserts the username, the email or the phone number, the system verifies that neither of those have already been selected by someone else.

3.2.2 Permit a guest to request a taxi

R1: The system has to provide a form for storing user basic personal informations.

R2: The system has to ask the user the consensus in order to use his position.

R3: The system has to search for an available taxi in the same area of the user after he presses the request button.

3.2.3 Permit a guest to sign in

R1: The system has to provide the user a sign in form.

R2: The system, during sign in, verifies that the username and the password inserted by the user have a match in the DB.

3.2.4 Permit a registered passenger to request a taxi

R1: The system has to ask the user the consensus in order to use his position.

R2: The system has to search for an available taxi in the same area of the user after he presses the request button.

3.2.5 Permit a registered passenger to make a reservation

R1: User must be already registered and logged in the application.

R2: User must make a reservation at least 2 hours before the request time.

R3: User must provide time, origin and destination of the ride at the moment of the reservation.

R4: User must confirm the reservation process.

R5: The reservation can be deleted by the user himself.

3.2.6 Permit a registered passenger to cancel a reservation

R1: The system shall check that the passenger is already registered and signed in with the correct credentials.

R2: The system shall check that the request the passenger is deleting, exists and belongs to him.

R3: The system shall check that it's not after 10 minutes before the reservation request time.

R4: This process is not reversible

3.2.7 Permit a taxi driver to give the system his availability

R1: The system shall check that the taxi driver is already registered and signed in with the correct credentials.

R2: The system shall check that the taxi driver is actually unavailable and then let the taxi driver set his status as unavailable through the mobile application.

3.2.8 Permit a taxi driver to revoke his availability

R1: The system shall check that the taxi driver is already registered and signed in with the correct credentials.

R2: The system shall check that the taxi driver is actually available and then let the taxi driver set his status as unavailable through the mobile application.

R3: The system shall revoke it automatically if it registers three consecutive request rejections.

3.2.9 Permit a taxi driver to accept a ride request

R1: The system shall check that the taxi driver is already registered and signed in with the correct credentials.

R2: When the system receives a taxi request, it shall remove the first taxi driver of that area from the relative queue and notify him, through the mobile application, that someone is requesting his taxi.

R3: When the taxi driver receives the notification, he accepts

R4: The system shall register the answer and inform the passenger that this taxi is reaching his position.

R5: The system shall set the taxi driver status as unavailable

3.2.10 Permit a taxi driver to refuse a ride request

R1: The system shall check that the taxi driver is already registered and signed in with the correct credentials.

R2: When the system receives a taxi request, it shall remove the first taxi driver of that area from the relative queue and notify him, through the mobile application, that someone is requesting his taxi.

R3: When the taxi driver receives the notification, he declines the request.

R4: If this taxi driver is at the third consecutive rejection, the system shall set his status as unavailable, notifying him

R5: If this taxi driver is at the first or second consecutive rejection, the system shall put him at the end of his area queue.

3.3 Scenarios

3.3.1 Occasional user

Mario, an important businessman, has just arrived by train in our city for the first time. His train had technical issues and now he is a bit late for the meeting on the opposite side of the town, so he does not want to take public transports. The best solution is to call a taxi. He opens the browser on his smartphone and he searches for our service. Once he has found the web page he inserts name and surname, sends a taxi request to our system. As soon as a taxi is found, he receives a notification with the taxi ID number and the estimated waiting time before the taxi arrival. He is then able to reach the meeting location faster than he thought.

3.3.2 Habitual user

The meeting of Mario went very well: he started an important collaboration with a local company. He will have to come back often in our town. He found our service very useful, so he decides to download the mobile application and to sign up. After a couple of weeks he has to come back. Unfortunately his train had technical issues, and he risks to be late to the meeting again. This time he has our application installed on his mobile device, so with only few taps and no additional information provided requests a taxi. As soon as a taxi is found he receives a notification with the taxi ID number and the estimated waiting time before the taxi arrival. He is able once again to reach the meeting location faster than he thought.

3.3.3 Taxi reservation

After another couple of weeks, Mario has to come back in our town again. He thinks that he will never be so unlucky to find a defective train for the third consecutive time, so he decides to make a reservation before leaving his place, in order to find a taxi waiting for him at the train station at his arrival. Using the mobile application, he fills in all the necessary fields, indicating hour and place of the request. Now he can relax and enjoy his journey.

3.3.4 Deleting a taxi reservation

Mario's train has technical issues. Poor Mario, he's so unlucky! He will be very late, so he does not want to let the taxi wait so long. He opens our application and, 30 minutes before the taxi request, he selects the reservation and deletes it. Once arrived he will request a taxi as usual.

3.3.5 Taxi driver

A taxi driver, Luigi, is having a little break, eating his favourite chocolate bar in his taxi. Suddenly his phone rings: a notification has just arrived. But he's having a break, so he does not answer. After 1 minute, the system automatically records a negative answer from him. Luigi slowly finish his chocolate bar and starts answering some messages on his smartphone. Suddenly his phone rings. He sees another notification, but he's very busy, so he declines the request. Luigi greets his friend on the chat and goes back to work. After a few minutes his phone rings again. A man called Mario is requesting a taxi in his area, right next to the train station. He accepts the request and starts driving towards the location of the customer. After the ride, Luigi sets his status back to available and waits for another request.

3.3.6 Taxi availability

After a hard morning, Luigi decides that it is the right time to have lunch. He opens our mobile application and sets his status to unavailable: he does not want to be disturbed during such an important moment. After a hearty lunch, he goes back to work, so he picks up his phone, opens our application and sets his status to available. He works hard for the whole afternoon and, after the last passenger, he's so tired that he forgets the status on available. The system sends him up to 3 request notifications, but Luigi has a huge headache and can not hear the phone ringing. After the third request without answer the system automatically sets him to unavailable. Finally Luigi will get his deserved peace.

3.4 UML models

3.4.1 Class diagram

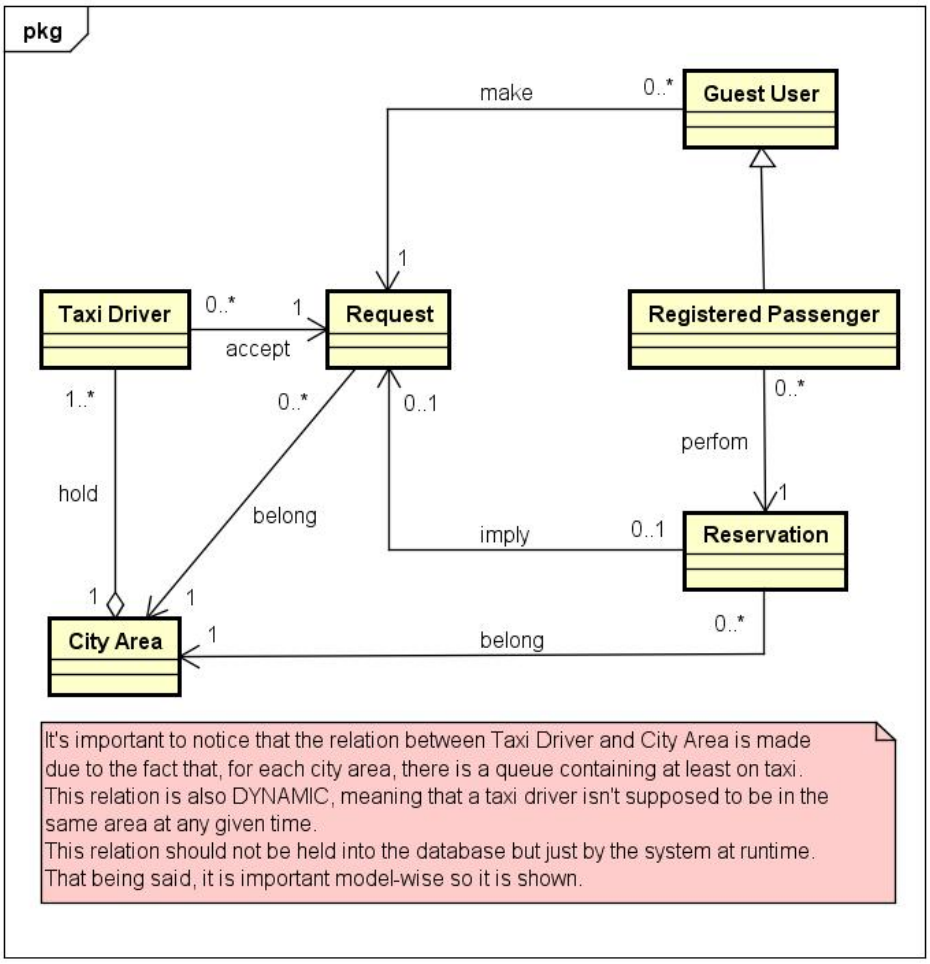

Figure 10: Class Diagram for MyTaxiService

3.4.2 Use Case diagram

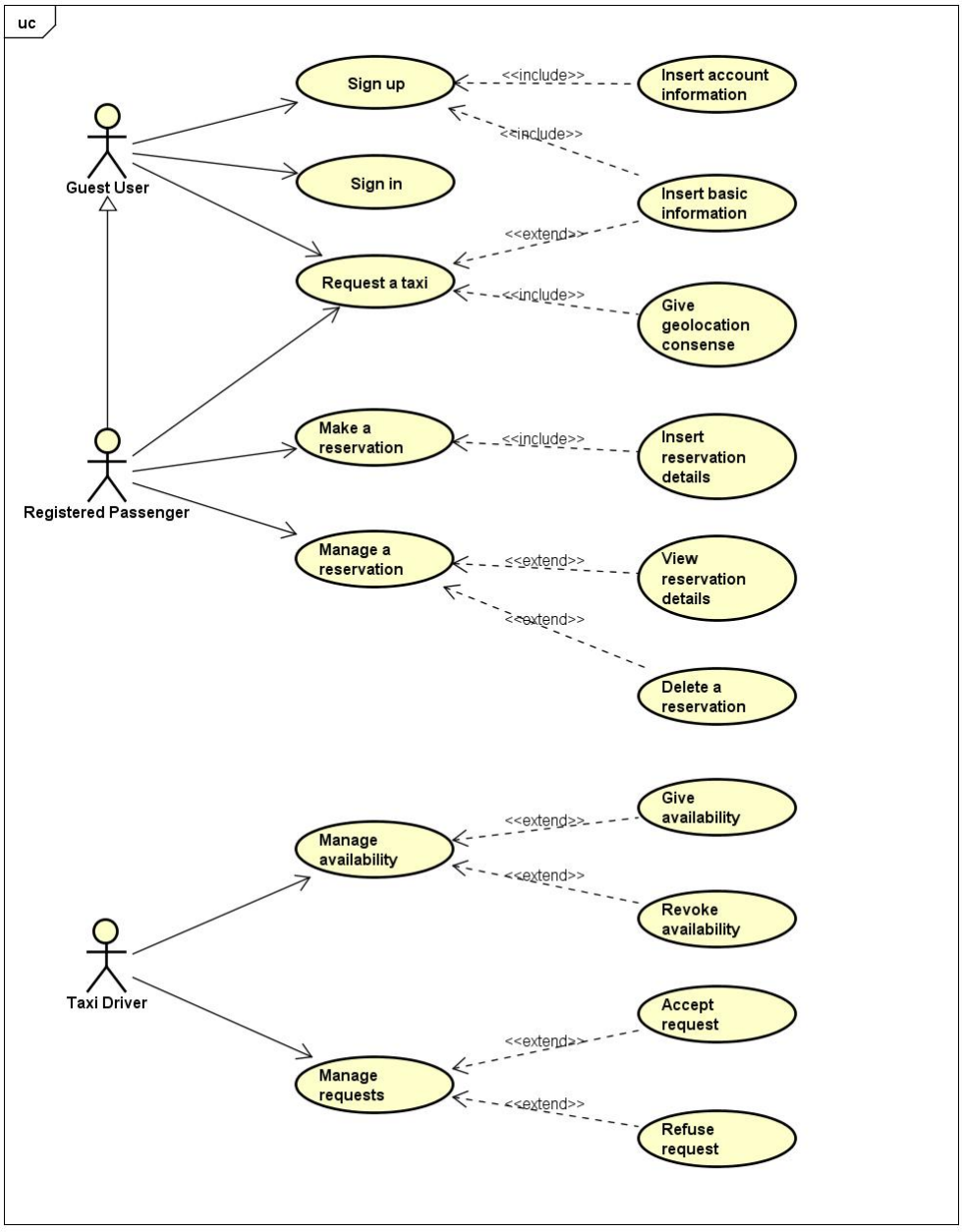

Figure 11: Use Case Diagram for MyTaxiService

3.4.2.1 Guest Registration

Actor	Guest
Preconditions	The guest has not registered yet to the service
Execution Flow	1. The guest presses the sign up button and access the registration form 2. The form requires to insert username, email, password, password confirmation, name, surname, telephone number and to accept terms and conditions 3. The system verifies that username, email and telephone number are uniques 4. The system confirms the registration 5. The system automatically loads the Passenger Area page
Postconditions	The guest is now a Registered Passenger and is logged in to the service
Exceptions	Username, password or email are not uniques; the registration fails

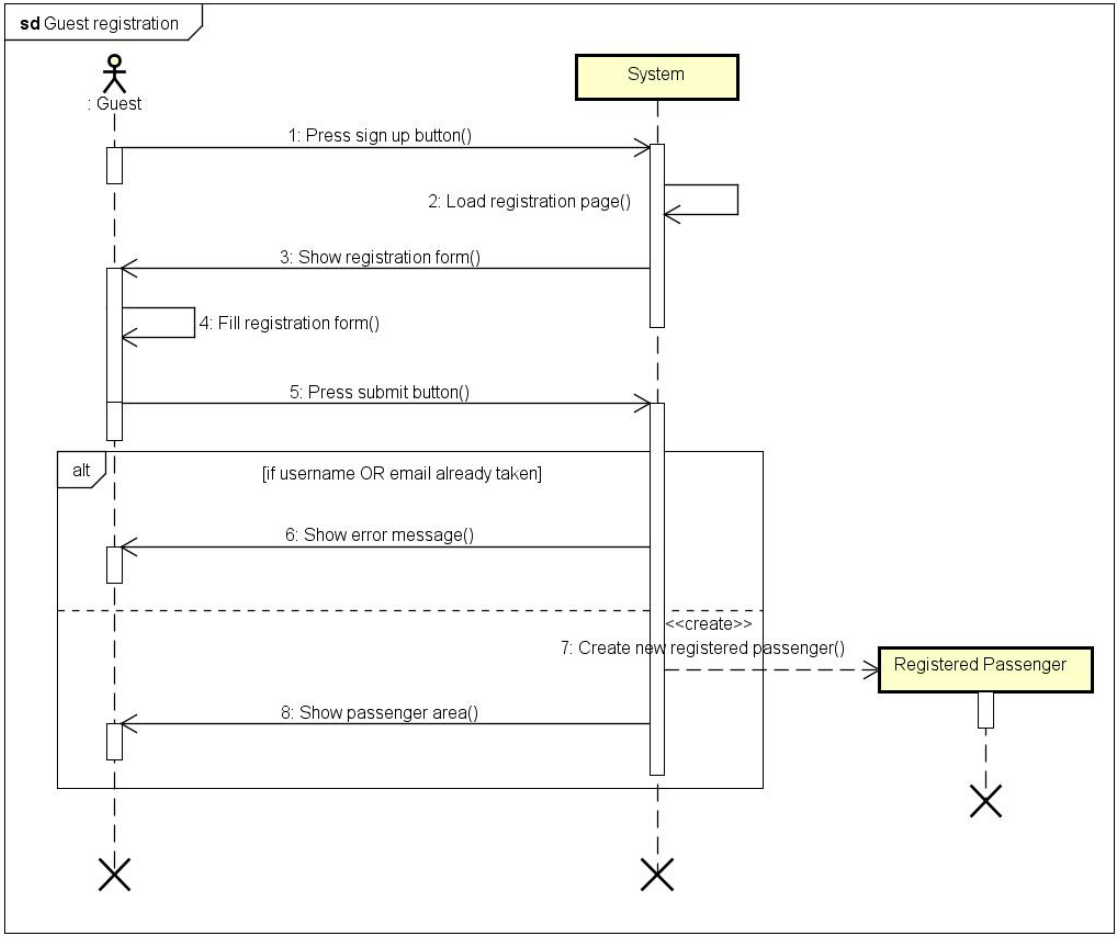

Figure 12: Sequence Diagram - Guest Registration

3.4.2.2 Registered Passenger Sign In

Actor	Registered Passenger
Preconditions	The registered passenger is registered to the service
Execution Flow	1. The registered passenger fills the sign in form, which requires to insert username and password 2. The guest presses the sign in 3. The system verifies that username and password matches 4. The system loads the Passenger Area page
Postconditions	The registered passenger is logged in to the service
Exceptions	Username not existent or username-password mismatch; the sign in fails

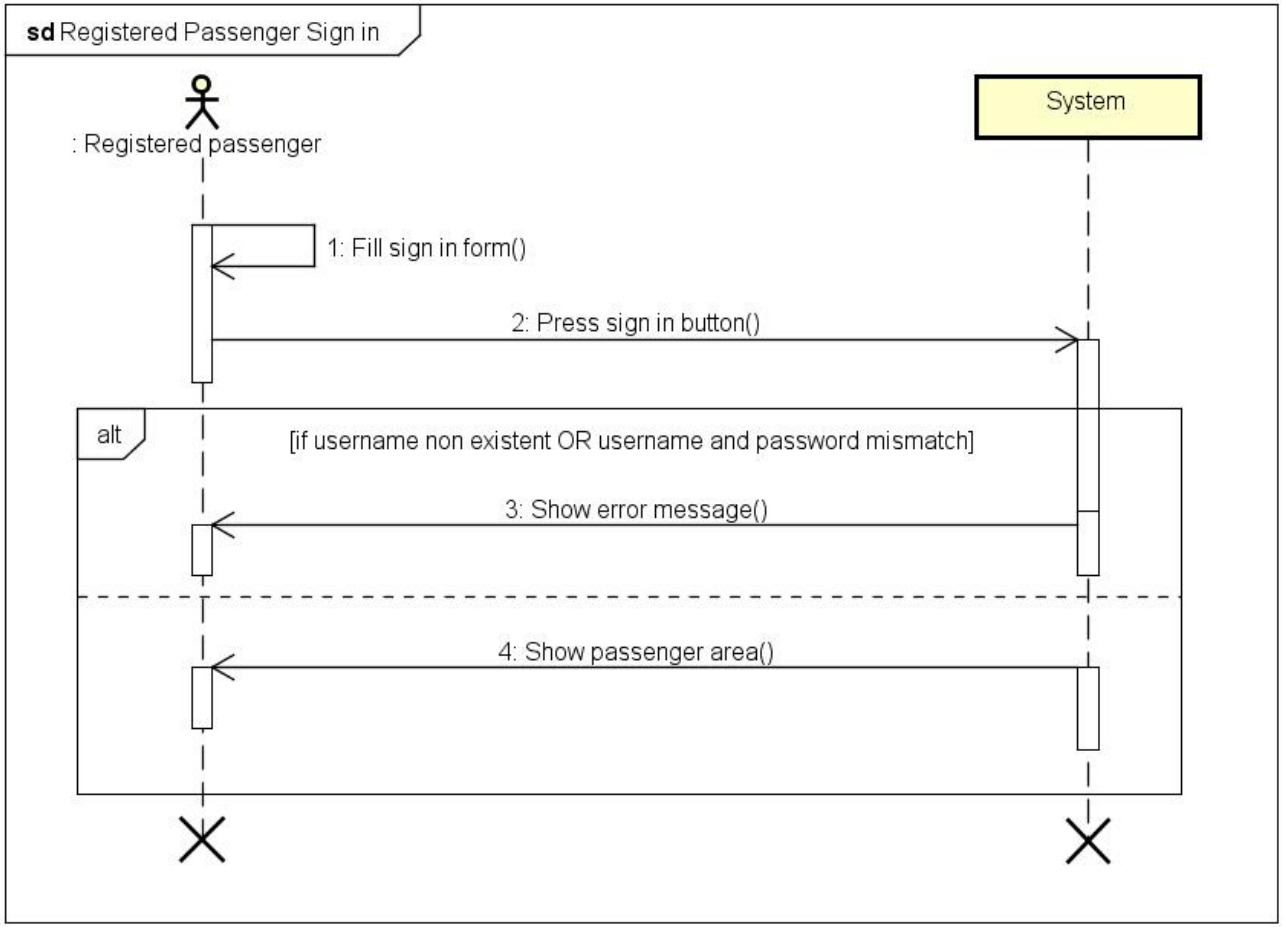

Figure 13: Sequence Diagram - Registered Passenger Sign In

3.4.2.3 Guest Taxi Request

Actor	Guest
Preconditions	Nothing
Execution Flow	1. The guest fills the request form, which requires to insert full name and telephone number and to give position consensus 2. The guest presses the request button 3. The system searches and finds an available taxi 4. The system loads the confirmation request page
Postconditions	The Guest has the code of the taxi he requested
Exceptions	GPS position not available; the request fails

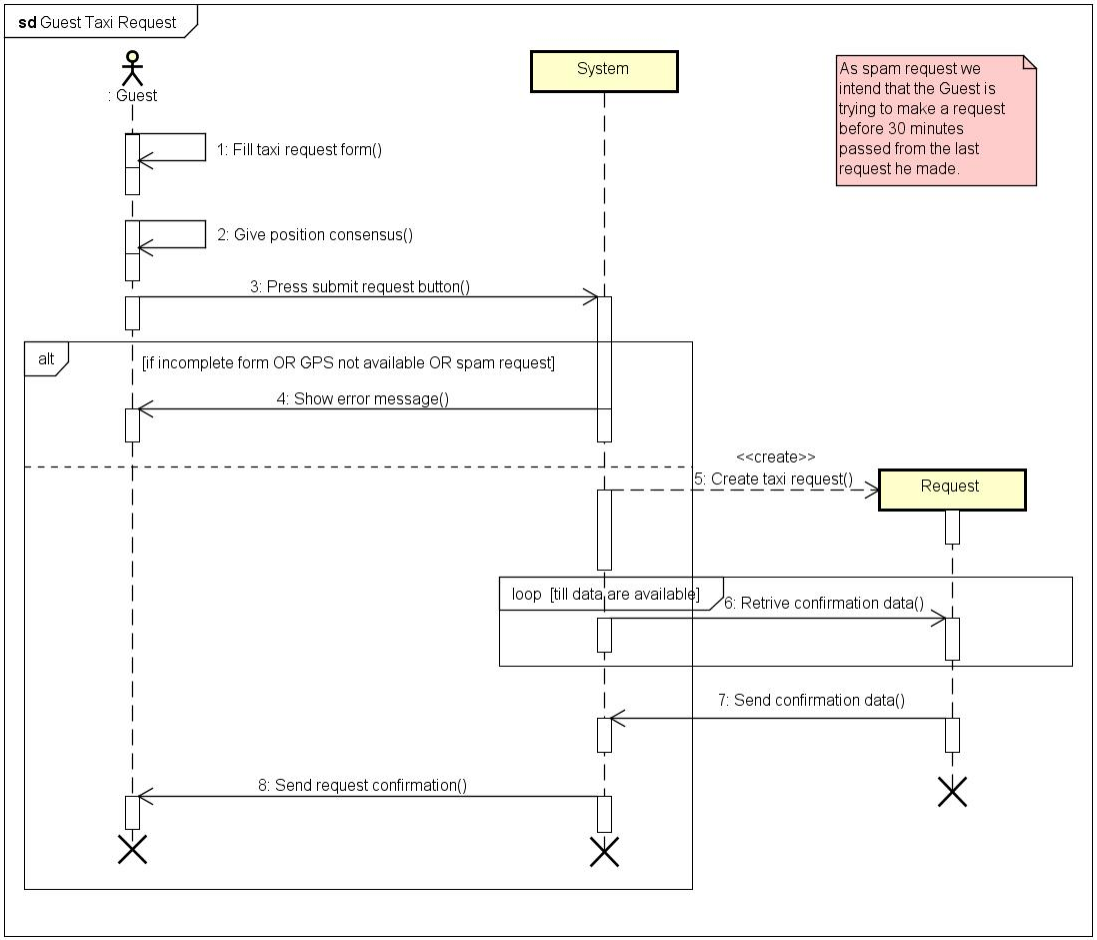

Figure 14: Sequence Diagram - Guest Taxi Request

3.4.2.4 Registered Passenger Taxi Request

Actor	Registered Passenger
Preconditions	The registered passenger is registered to the service and is logged in
Execution Flow	1. The registered passenger accesses the Passenger Area page 2. The registered passenger gives position consensus and presses the request button 3. The system searches and finds an available taxi 4. The system loads the confirmation request page
Postconditions	The registered passenger has the code of the taxi he requested
Exceptions	GPS position not available; the request fails

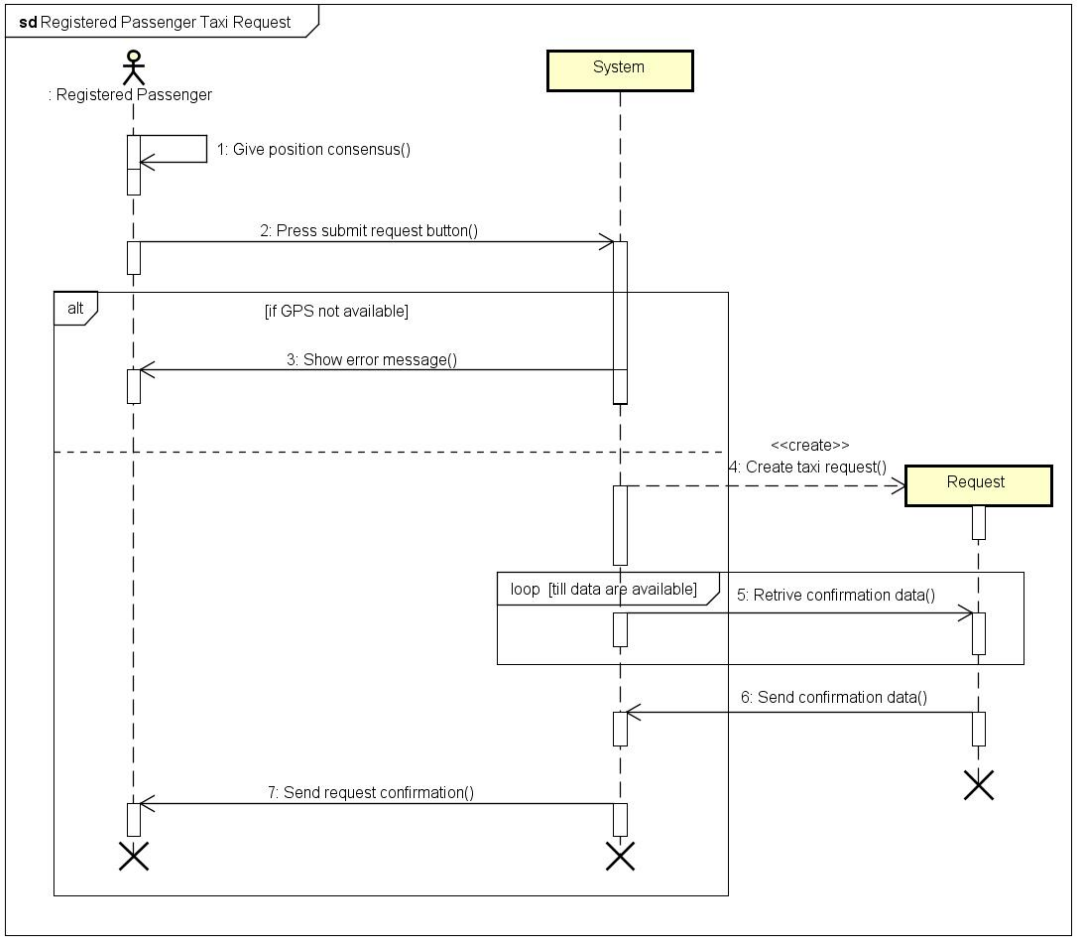

Figure 15: Sequence Diagram - Registered Passenger Taxi Request

3.4.2.5 Registered Passenger Make A Reservation

Actor	Registered Passenger
Preconditions	The registered passenger is registered to the service and is logged in
Execution Flow	1. The registered passenger accesses the Passenger Area page 2. The registered passenger fills the reservation form inserting origin and destination of the ride and the day and time when the ride has to happen 3. The system confirms the reservation
Postconditions	The registered passenger has a new reservation in his reservations list
Exceptions	The form isn't filled correctly or the time constraints aren't satisfied; the reservation fails

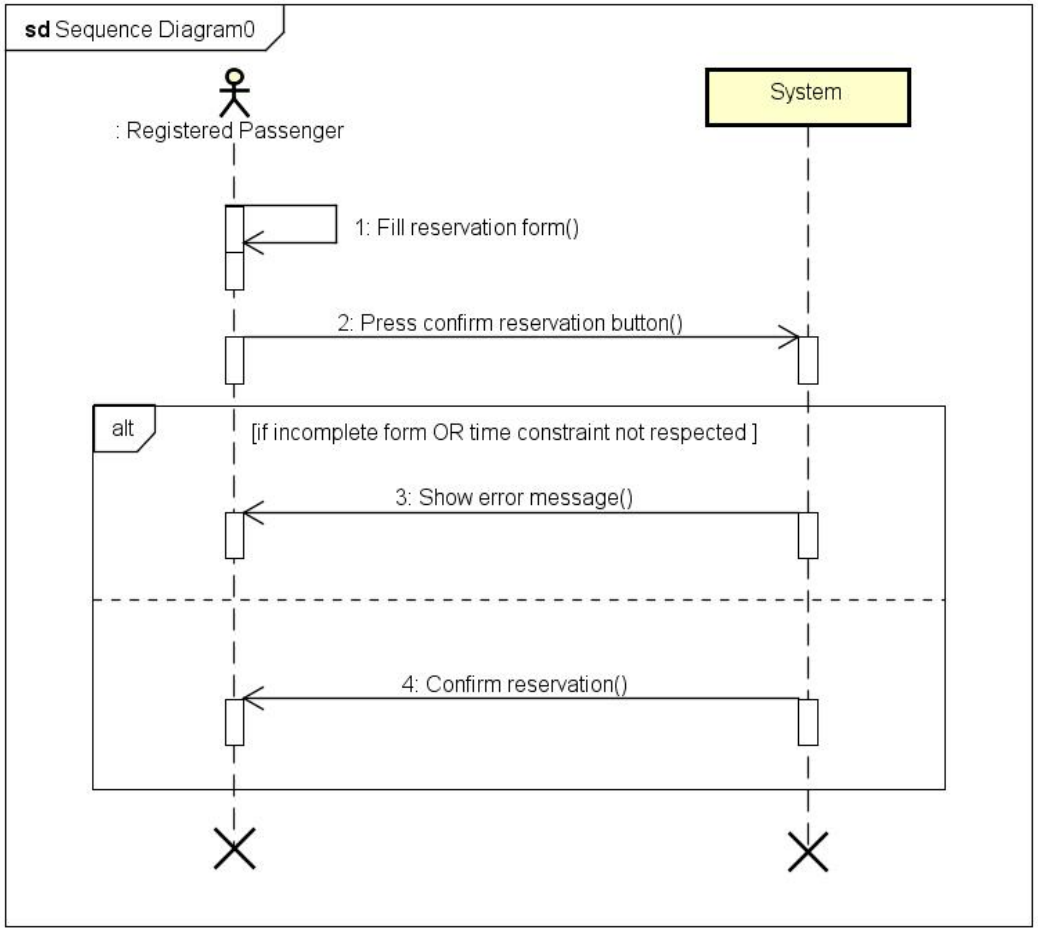

Figure 16: Sequence Diagram - Registered Passenger Make a reservation

3.4.2.6 Registered Passenger Cancel A Reservation

Actors	Registered passenger
Preconditions	The user is logged into the system as registered passenger and has already made a reservation
Execution Flow	1. The user accesses his personal page with the list of reservations 2. The user selects the reservation that wants to delete and presses the apposite button 3. The system cancels his reservation and reloads the Passenger Area page
Postconditions	The reservation is cancelled and no requests will be forwarded
Exceptions	It's too late to cancel the reservation (after 10 minutes before the time established): the reservation is not cancelled and the user is notified.

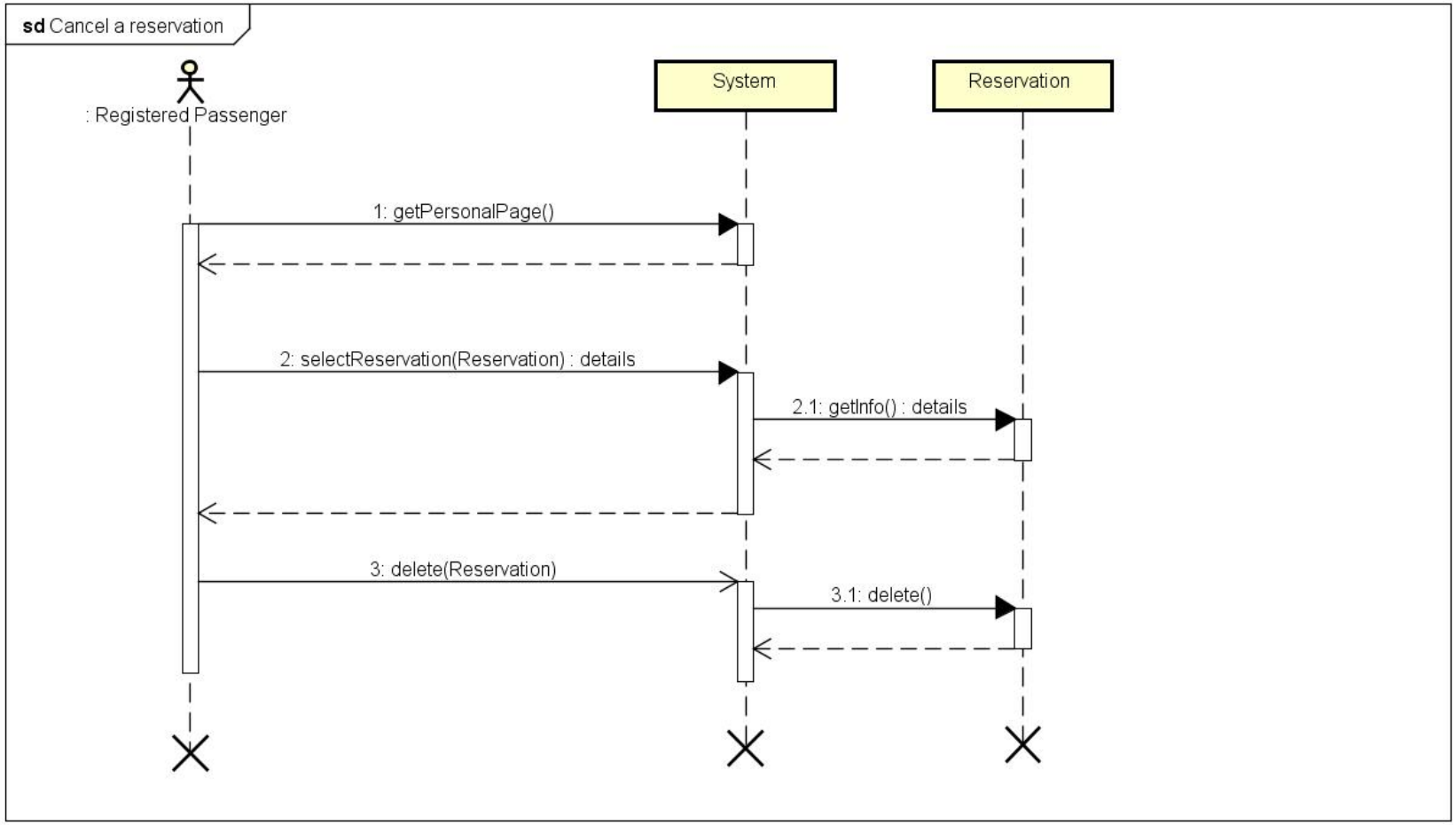

Figure 17: Sequence Diagram - Registered Passenger Cancel a reservation

3.4.2.7 Taxi Driver Give availability

Actors	Taxi driver
Preconditions	The user is logged into the system as taxi driver and is using the mobile application
Execution Flow	1. The taxi driver accesses his personal area 2. The taxi driver presses the apposite button 3. The system sets the taxi driver status as available 4. The system inserts the taxi driver in his area queue
Postconditions	The taxi driver is now available and will receive notifications of taxi requests.
Exceptions	The taxi driver is already available: the system will not react anyway at the button pressure.

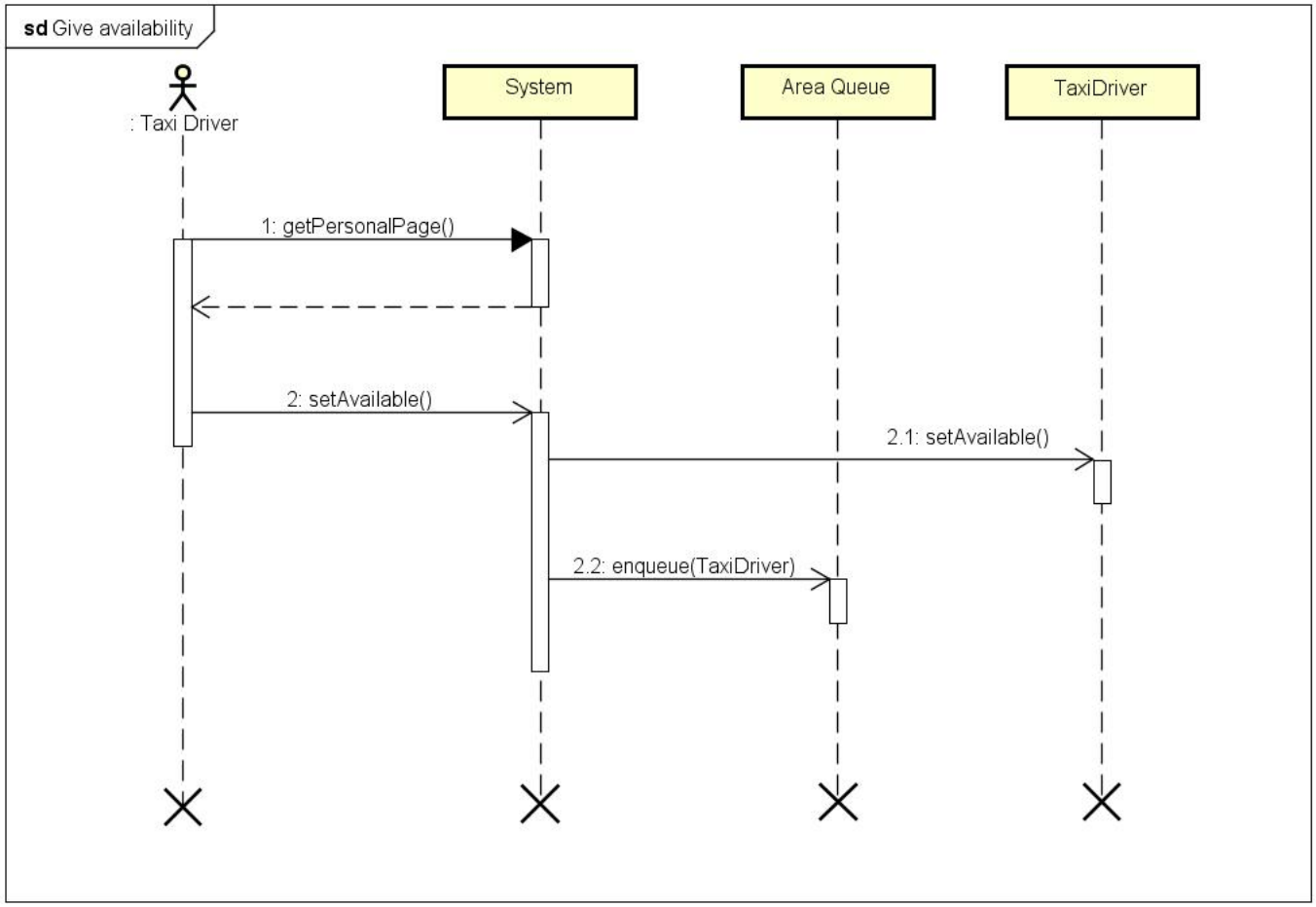

Figure 18: Sequence Diagram - Taxi Driver Give availability

3.4.2.8 Taxi Driver Revoke availability

Actors	Taxi driver
Preconditions	The user is logged into the system as taxi driver and is using the mobile application
Execution Flow	1. The taxi driver accesses his personal area 2. The taxi driver presses the apposite button 3. The system sets the taxi driver status as unavailable 4. The system removes the taxi driver from his area queue
Postconditions	The taxi driver is now unavailable and will not receive notifications of taxi requests.
Exceptions	The taxi driver is already unavailable: the system will not react anyway at the button pressure.

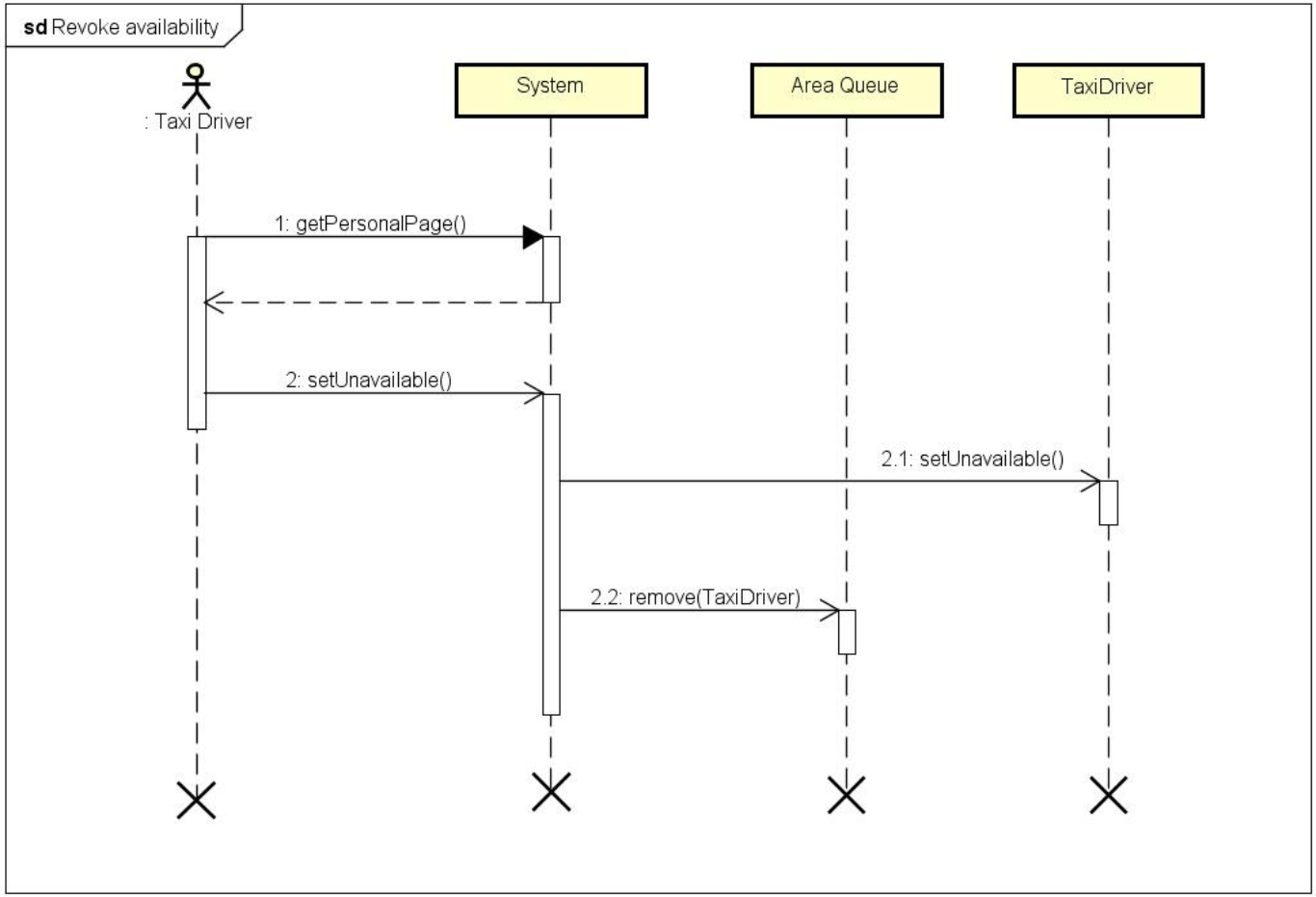

Figure 19: Sequence Diagram - Taxi Driver Revoke Availability

3.4.2.9 Taxi Driver Accept A Ride Request

Actors	Taxi driver
Preconditions	The user is logged into the system as taxi driver and is using the mobile application
Execution Flow	1. The taxi driver receives the notification of a request 2. The taxi driver accesses his personal page 3. The taxi driver presses the Accept button 4. The system notifies the user who asked for the taxi sending him information about the taxi and the arrival time 5. The system sets the taxi driver as unavailable
Postconditions	The taxi driver is not enqueued again and the user's request is considered satisfied
Exceptions	The taxi driver does not answer within 1 minute: the system automatically records a rejection.

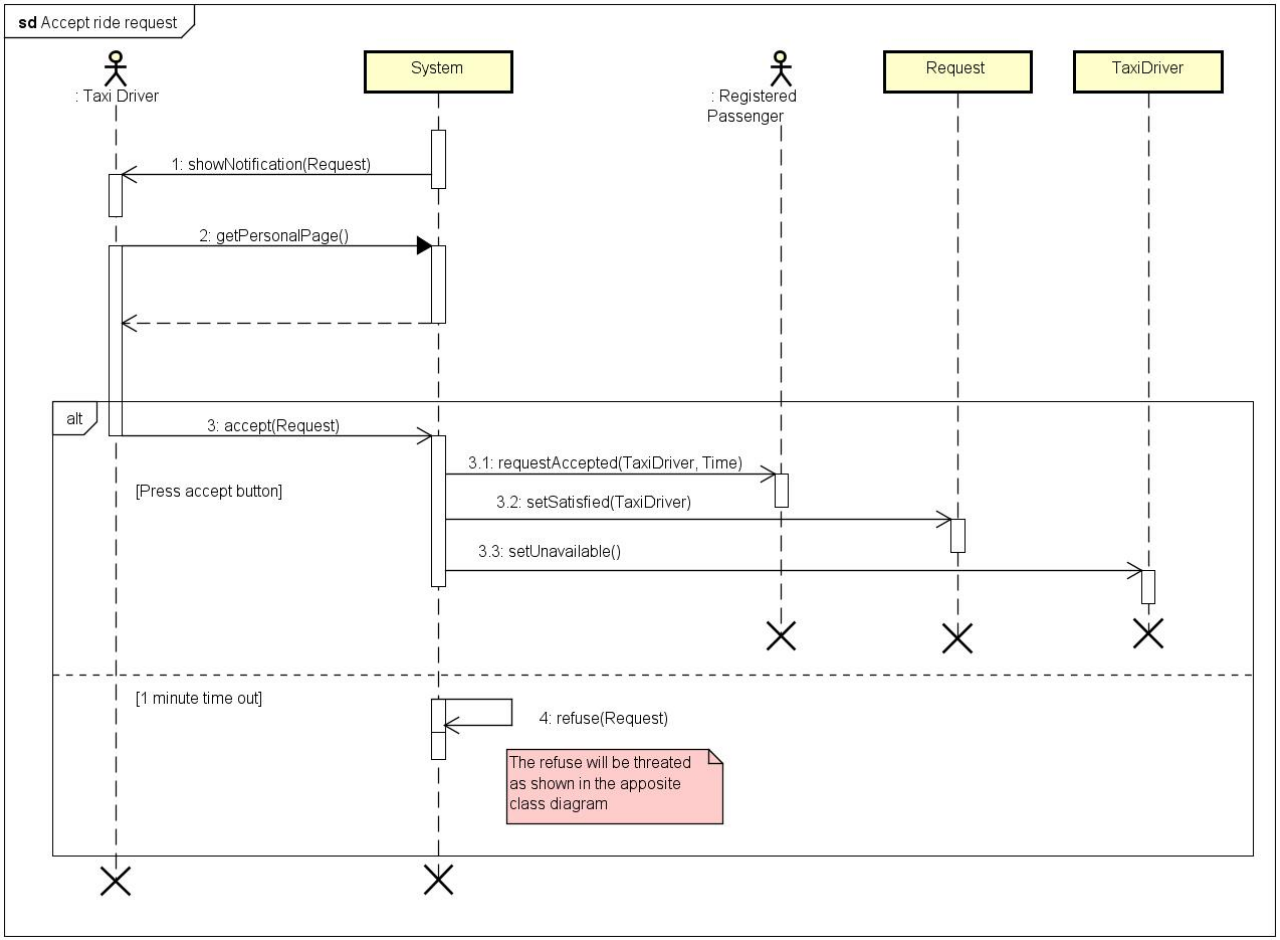

Figure 20: Sequence Diagram - Taxi Driver Accept A Ride Request

3.4.2.10 Taxi Driver Refuse A Ride Request

Actors	Taxi driver
Preconditions	The user is logged into the system as taxi driver and is using the mobile application
Execution Flow	1. The taxi driver receives the notification of a request 2. The taxi driver accesses his personal page 3. The taxi driver presses the Decline button 4. The system records the rejection and reinserts the taxi at the end of the area queue 5. The system sends the request notification to another taxi
Postconditions	The taxi driver is enqueued again and the user's request is considered unsatisfied
Exceptions	The taxi driver does not answer within 1 minute: the system automatically records a rejection. The taxi driver is at the third consecutive rejection: the system does not enqueue the taxi driver and sets his status as unavailable

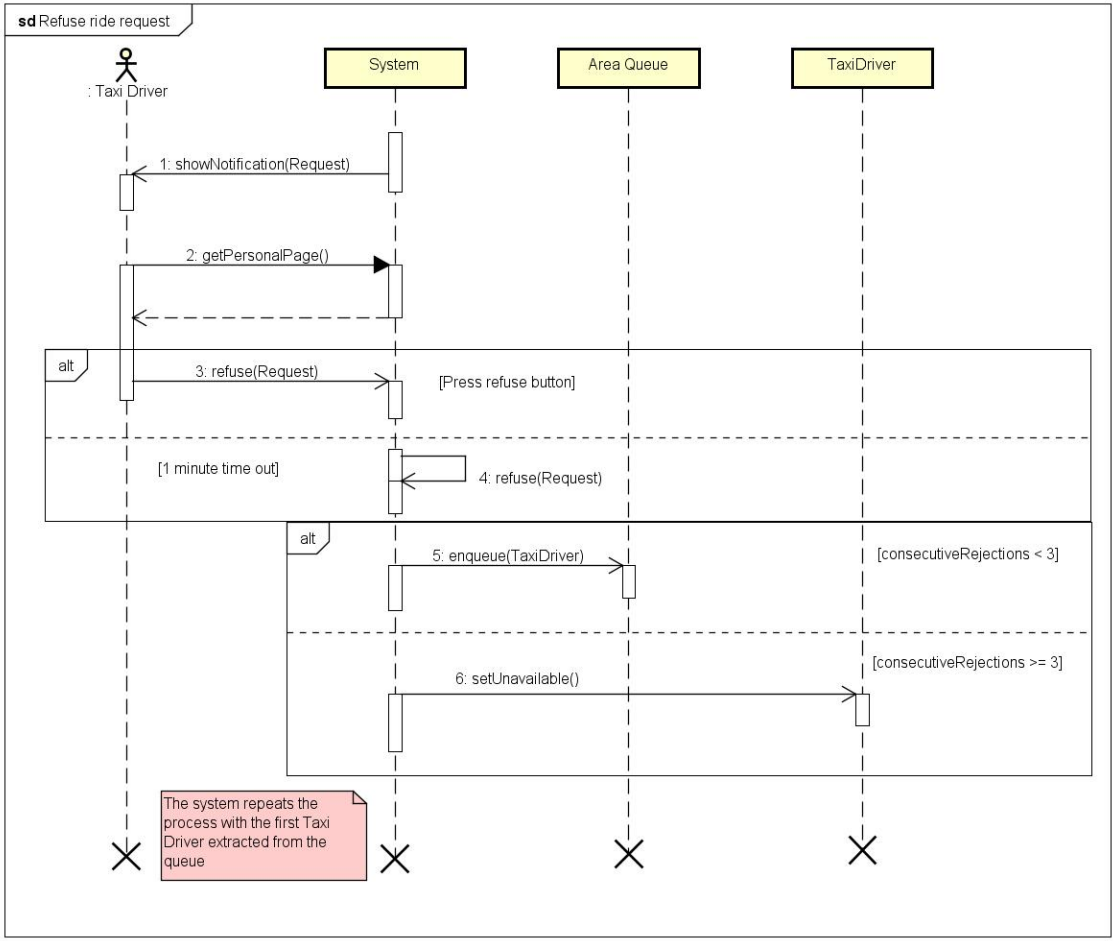

Figure 21: Sequence Diagram - Taxi Driver Refuse A Ride Request

3.5 State Charts

In this section the behaviour of some entities is exposed using UML state chart diagrams.

3.5.1 Request

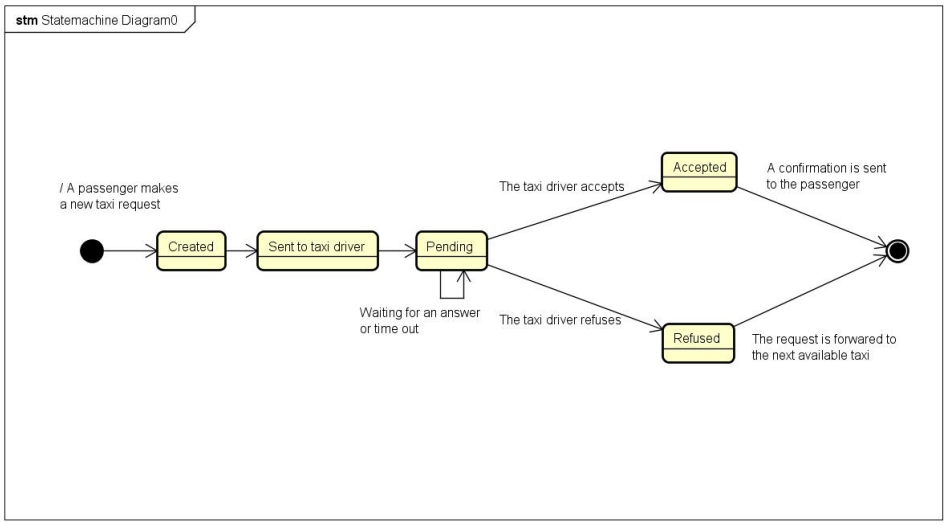

Figure 22: State Chart - Request

3.5.2 Reservation

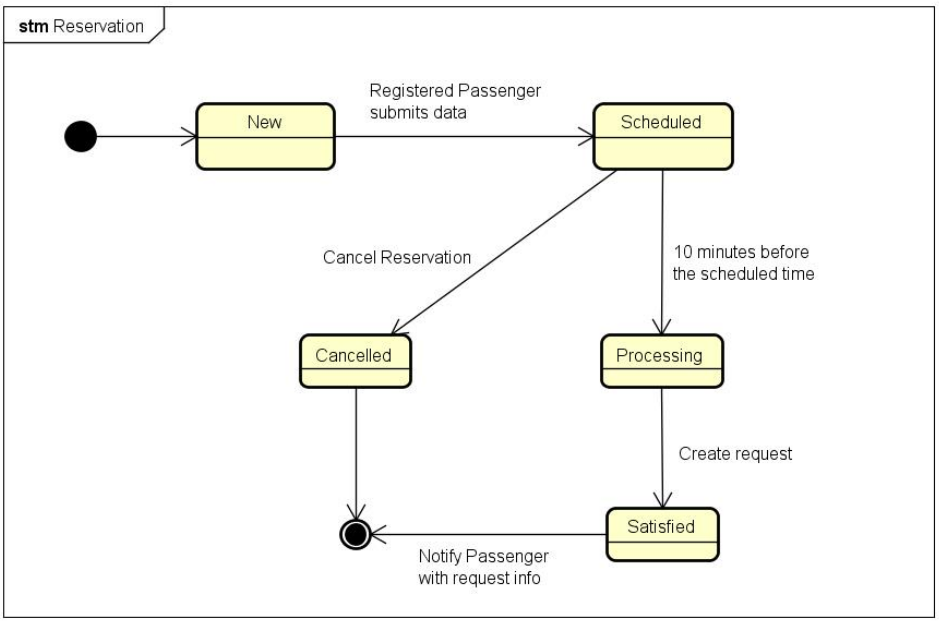

Figure 23: State Chart - Reservation

3.6 Non Funcional Requirements

Some non functional requirements have been found in order to fulfil some qualities and a correct operation flow that the system should provide.

3.6.1 User friendliness

The user interface has to be simple and intuitive. This will help any kind of user to easily access our service.

3.6.2 Portability

The client has to be compatible to all the major hardware and software platform on the market. While this is not a problem for the web application, for the mobile one refers to the Hardware Limitations section.

3.6.3 Performance

In order to have a good service, the system has to be reactive and able to answer a high number of simultaneous requests. Because of this the interaction between the client and the server has to be reduced to a minimum, in order to prevent overload.

3.6.4 Availability

The service should be available to the user the highest possible amount of time. We suppose to be able to grant a 4-nines availability, which means a 52 minutes per years.

3.6.5 Data integrity and consistency

Data have to be always accessible, so the system has to provide an access to them in normal condition. Data have to be duplicate in order to avoid losses in case of system fault. For instance the use of RAID, mirroring and backup could present a valid solution.

3.6.6 Security

To ensure that sensible data remain private, the system should offer different security measures. The accounts are protected by the couple username-password. Data should be filtered in order to avoid vicious or unwanted modification in the data base (e.g. SQL injection). Finally, in order to protect the on-line transmission of data an HTTPS protocol could be used. However this last proposal should be delayed to a future implementation because the HTTPS protocol needs the purchase of a certificate signed by recognized certification authority.

4 Alloy Modelling

In this chapter the consistency of the proposed Class Diagram will be tested via Alloy Analyzer. The report that follow is composed by the code used to describe the model and an example of world generated by our predicates.

4.1 Alloy Code

```
module MyTaxiService

/*** Class declaration ***/
open util/boolean

sig GenericText{}

sig GenericData{
year: one Int,
month: one Int,
day: one Int,
hour: one Int,
minute: one Int
}{
year > 0 and
month > 0 and
day > 0 and
hour > 0 and
minute > 0
}

sig Guest {
ip: one GenericText
--name: one GenericText,
--surname: one GenericText,
--telephone: one GenericText
}

sig RegisteredPassenger extends Guest{
username: one GenericText,
email: one GenericText,
--password: one GenericText,
}

sig TaxiDriver{
taxiID: one GenericText,
availability: one Bool
--name: one GenericText,
--surname: one GenericText
}

sig Request{
area: one Area,
passenger: one Guest,
when: one GenericData
}

sig Reservation{
passenger: one RegisteredPassenger,
area: one Area,
when: one GenericData
```

```
}

sig Area {
where: one GenericText,
queue: set TaxiDriver
}

-- there are also other attributes but are not relevants for this
    representation

/*** DEFINITION OF THE CONSTRAINTS ***/

--each email has to be unique
fact emailUnicity{
no disj rp1, rp2: RegisteredPassenger | rp1.email = rp2.email
}

--each username has to be unique
fact usernameUnicity{
no disj rp1, rp2: RegisteredPassenger | rp1.username = rp2.username
}

--each taxiID has to be unique
fact taxiIDUnicity{
no disj t1, t2: TaxiDriver | t1.taxiID = t2.taxiID
}

--each Area has to be unique
fact areaUnicity{
no disj a1, a2: Area | a1.where = a2.where
}

--each taxi, if available, must be in only one queue
fact onlyOneTaxi{
∀ t: TaxiDriver | t.availability = True implies one a: Area | t in
    a.queue
}

--each taxi, if not available, must not be in any queue
fact noOneTaxi{
∀ t: TaxiDriver | t.availability = False implies ∀ a: Area | t not in
    a.queue
}

--a user can not have two requests without 30 minutes of distance
    betweeen them
fact requestTime{
∀ disj r1, r2: Request |
r1.when.year = r2.when.year and
r1.when.month= r2.when.month and
r1.when.day= r2.when.day and
r1.when.hour= r2.when.hour and
r1.when.minute > r2.when.minute and
r1.when.minute - r2.when.minute < 30 implies r1.passenger ≠
    r2.passenger
}

--the same passenger can't have two reservation with desired time
    within 30 minutes from one another
fact reservationTime{
∀ disj r1, r2: Reservation |
r1.when.year = r2.when.year and
```

```
r1.when.month= r2.when.month and
r1.when.day= r2.when.day and
r1.when.hour= r2.when.hour and
r1.when.minute > r2.when.minute and
r1.when.minute - r2.when.minute < 30 implies r1.passenger ≠
    r2.passenger
}

/*** FUNCTION ***/

fun passengerRequests [g: Guest]: set Request{
{r: Request | g = r.passenger}
}

fun passengerReservations [p: RegisteredPassenger]: set Reservation{
{r: Reservation | p = r.passenger}
}

fun getTaxiAreas [t: TaxiDriver]: set Area{
{a: Area | t in a.queue}
}

/*** ASSERTION ***/

--check that a taxi is in a sngle queue or is not in any
assert oneTaxiOneQueue{
∀ t: TaxiDriver | #getTaxiAreas[t] ≤ 1
}

check oneTaxiOneQueue for 3


--the same registered passenger doesn't have two reservations with
    desired time within 30 minutes from one another
assert noSpamReservation{
∀ p: RegisteredPassenger | no disj r1, r2: Reservation | r1 in
    passengerReservations [p] and r2 in passengerReservations [p] and
r1.when.year = r2.when.year and
r1.when.month= r2.when.month and
r1.when.day= r2.when.day and
r1.when.hour= r2.when.hour and
r1.when.minute < r2.when.minute and
r1.when.minute - r2.when.minute < 30
}

check noSpamReservation for 3

--the same passenger doesn't have two requests within 30 minutes from
    one another
assert noSpamRequest{
∀ g: Guest | no disj r1, r2: Request | r1 in passengerRequests[g] and
    r2 in passengerRequests[g] and
r1.when.year = r2.when.year and
r1.when.month= r2.when.month and
r1.when.day= r2.when.day and
r1.when.hour= r2.when.hour and
r1.when.minute > r2.when.minute and
r1.when.minute - r2.when.minute < 30
}

check noSpamRequest for 3
```

```
/*** PREDICATES ***/

pred hardSituation{
#Guest  > 10 and
#RegisteredPassenger > 10 and
#TaxiDriver > 5 and
#Request > 5 and
#Reservation > 2
}

run hardSituation for 20

pred casualSituation{
#Guest  > 1 and
#RegisteredPassenger > 1 and
#TaxiDriver > 1 and
#Request > 1
}

run casualSituation for 3

pred show{

}

run show for 2
```

4.2 Alloy Response

Here the alloy response for the model is shown.

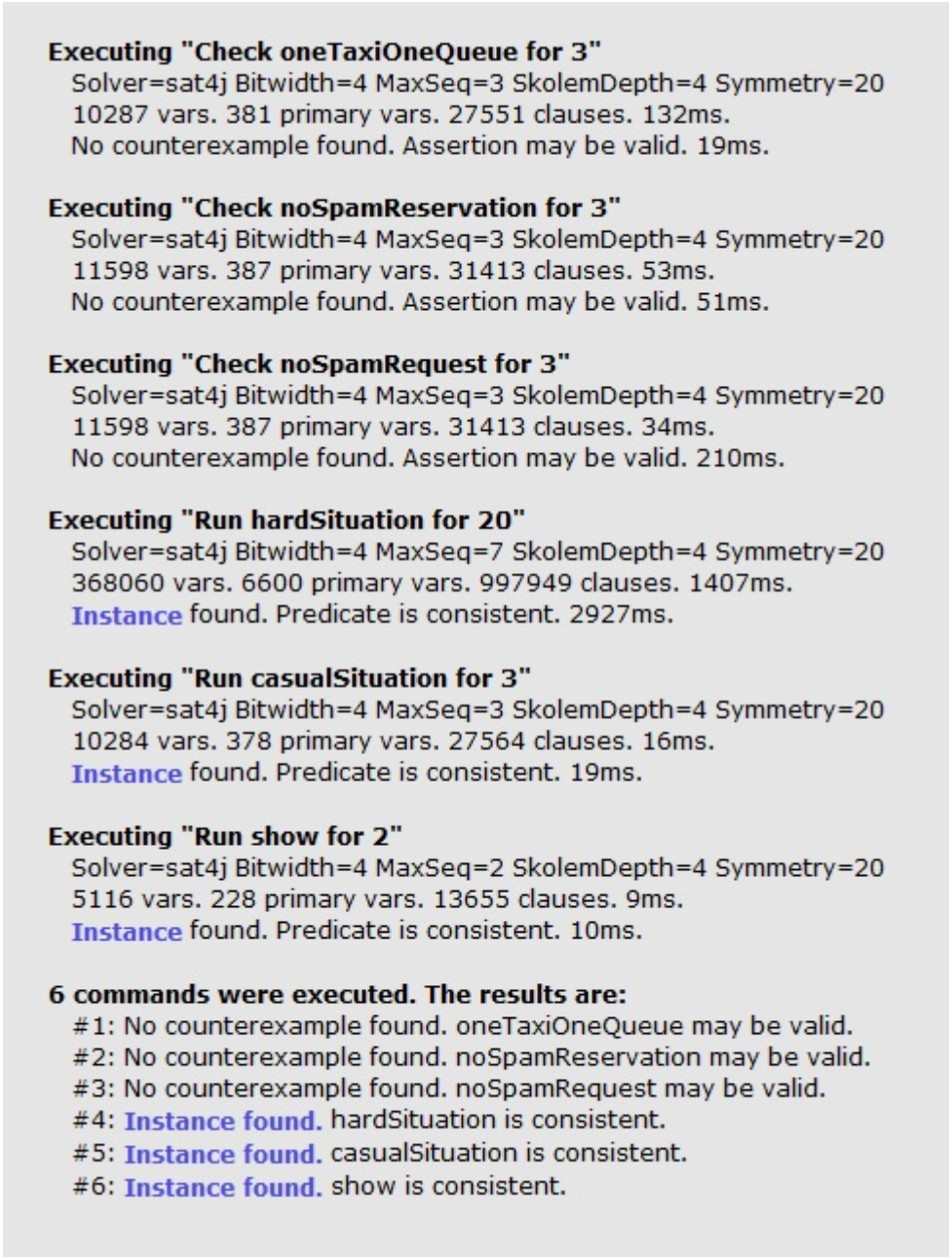

Figure 24: Alloy Answer

4.3 Alloy Worlds

4.3.1 Hard Situation

We do not insert the image for this test case because is huge and pretty incomprehensible. The test has been executed to ensure that the model keeps its consistency also with a significant number of entities.

4.3.2 Casual Situation

Here is shown the world generated by Alloy Analizer via the execution of the predicate "casualSituation". We have added basic constraints to this predicate, in order to ensure that everything has at least an instance (until a maximum of 3). The reservation is not considered because it's an advanced feature and has been tested deeply in the hard situation. However the system added an instance automatically.

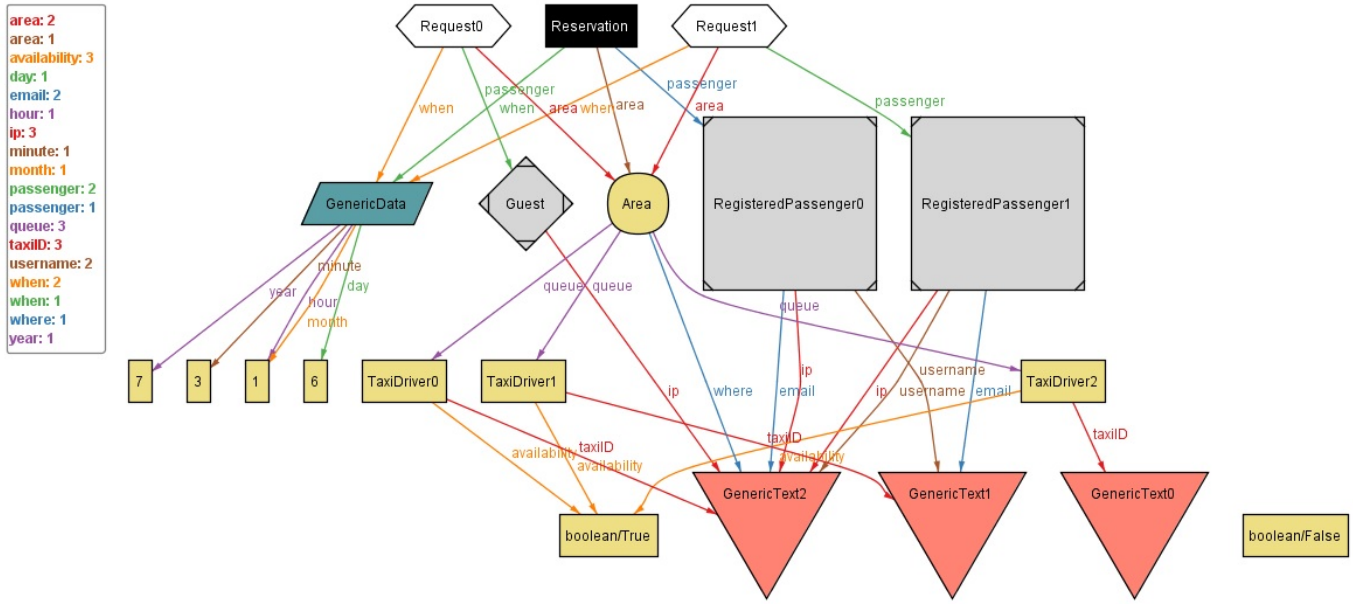

Figure 25: World Representation - Casual Situation

4.3.3 Show

Here is shown the world generated by Alloy Analizer via the execution of the predicate "show". In this predicate there are no constraints. The predicate generates a world that respects the given constraints where each entities has at maximum 2 instances (to keep the representation model little and readable).

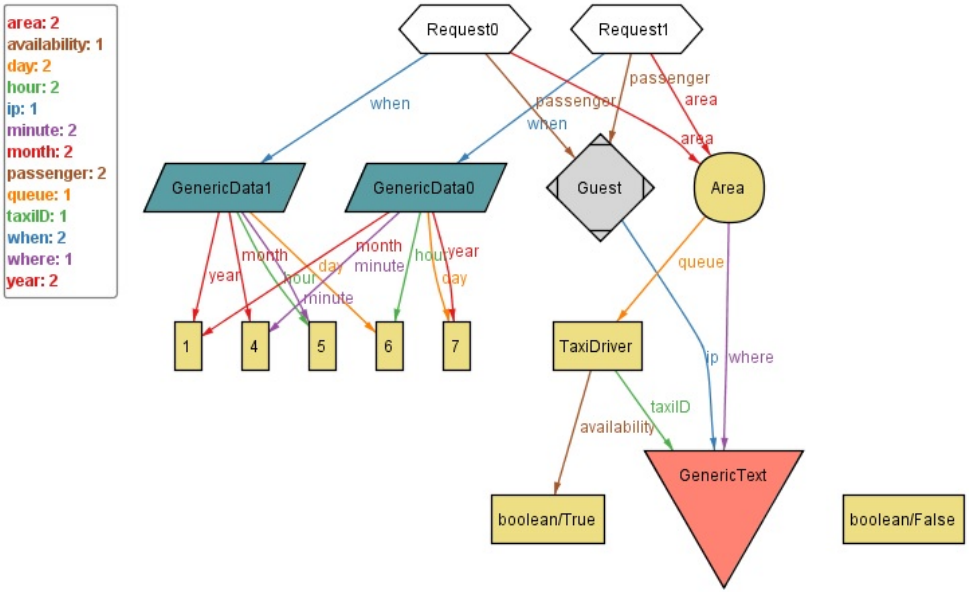

Figure 26: World Representation - Default Show

5 Other Info

This chapter contains information about the used tools and the hours of work by the members of the working group.

5.1 Working Hours

For each component follows an approximate indication of how much time was spent on the realization of this document

Component	Hours
Bernardis Cesare	29
Dagrada Mattia	31

5.2 Tools

We used various tools to develop this document:

- LaTeX 2_ε and TeXMaker editor 2.10.4
- Astah Professional 7.0.0
- Alloy Analyzer 4.2

5.3 Revision

This is the version 1.0 of the RASD document. Every future change or update will have to be documented and will cause a version upgrade.