

Requirements **A**nalysis & **S**pecification
Document
“MyTaxiService” Application
A.Y. 2015-1016

Ennio Visconti (mat. 790382)
Simone Zocchi (mat. 852910)
Khanh Huy Paolo Tran (mat. 852496)

November 6, 2015



POLITECNICO
MILANO 1863

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, Acronyms, Abbreviations:	3
1.3.1	Acronyms & Abbreviations	3
1.3.2	Definitions	3
1.4	Reference	4
1.5	Overview	4
2	Overall Description	4
2.1	Product perspective	4
2.2	Product functions	5
2.3	User characteristics	6
2.4	Constraints	6
2.4.1	Regulatory policies	6
2.4.2	Hardware limitation	6
2.4.3	Interfaces to other applications	6
2.4.4	Parallel operation	6
2.4.5	Reliability requirements	7
2.4.6	Criticality of the application	7
2.4.7	Safety and security considerations	7
2.5	Assumptions and dependencies	7
2.6	Goals	7
2.7	Apportioning of requirements	8
3	Scenarios	8
3.1	Scenario 1: Usage of the app	8
3.2	Scenario 2: Taxi reservation	8
3.3	Scenario 3: Shared ride, passenger's side	8
3.4	Scenario 4: Shared ride, driver's side	9
3.5	Scenario 5: Taxi driver notifications	9
4	Specific requirements	9
4.1	External interface requirements	9
4.1.1	User interfaces	9
4.1.2	Hardware interfaces	14
4.1.3	Communication interfaces	14
4.1.4	Software interfaces	14
4.2	Functional requirements	15
4.2.1	User 1 - Visitor	15
4.2.2	User 2 - Passenger	15
4.2.3	User 3 - Taxi driver	16
4.2.4	General management	17
4.3	Use Cases	17

4.3.1	Visitor registration	17
4.3.2	Login	18
4.3.3	Standard request	18
4.3.4	Rides Log	18
4.3.5	Taxi reservation	19
4.3.6	Taxi reservation (already booked)	19
4.3.7	Shared Ride request	20
4.3.8	Ride management and completion	20
4.3.9	Shared ride management	21
5	Appendix	27
5.1	Class Diagram	27
5.2	Statechart Diagram	28
5.3	Alloy	29
5.3.1	Purpose	29
5.3.2	Code	29
5.3.3	Example Worlds	36
6	Review	38

1 Introduction

1.1 Purpose

This document has the **goal** of describing all the **requirements** asked by the customer, showing the **constraints** and the possible **scenarios** involving the system. The intended **audience** of this document are both the **developers** of the system and the **customer**.

1.2 Scope

MyTaxiService is an application born to support an existing taxi company, especially to simplify passengers' access to the service and guarantee a fair management of taxi queues. In fact it will be GPS-based and it will be able to assign a taxi to a certain call, based on the location of the taxi and of the requesting passenger, in order to minimize the waiting time.

It will also provide an option to reserve a taxi in advance and to share the ride with other passengers to save money. All these features will be available via a freely downloadable mobile application or a web application.

It will also provide an interface for the taxi drivers, in order to communicate with them more easily and to make them able to automatically share their location over time.

1.3 Definitions, Acronyms, Abbreviations:

1.3.1 Acronyms & Abbreviations

- **ETA**: Estimated Time of Arrival
- **EWT**: Estimated Waiting Time
- **DB**: DataBase
- **DBMS**: DataBase Management System
- **PSP**: Payment Service Provider
- **UML**: Unified Modeling Language
- **API**: Application Programming Interface
- **GPS**: Global Positioning System

1.3.2 Definitions

- **Free**: a taxi is Free or Available when no passengers are in it, the driver is ready to receive new requests and no rides have been programmed for it.

- **Busy:** a taxi is Busy when is being used by a passenger doing a standard ride or there are no more places in a shared ride. It is therefore unable to receive any new request.
- **Shared:** a taxi is Shared if it is being used by a passenger doing a shared ride and so it can accept only new requests for shared rides.
- **User:** user is intended as one of the registered users (passenger or taxi driver).
- **Request:** a Request is the user using the application to book a taxi to go to some place.
- **Reserve:** a Reservation is a Request done by an user for a taxi in the future. Booking time must be at least 2 hour before the ride to start.

1.4 Reference

- Specification document: Software Engineering 2 Project, AA 2015-2016.
- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.

1.5 Overview

The document is organized into 5 parts:

- Section 1: **Introduction.** It gives a description of the document underlining his goals and gives basic information about the application.
- Section 2: **Overall description.** It gives general information about the software.
- Section 3: **Scenarios.** It contains a set of relevant scenarios of the usage of the application.
- Section 4: **Specific requirements.** It presents features and requirements of the system.
- Section 5: **Appendix.** It contains UML diagrams that support the description of the system's behaviours and an Alloy model of the world to check some restrictions.

2 Overall Description

2.1 Product perspective

The name of the application is *MyTaxiService*. The pre-existent taxi service system consists of databases containing taxis and drivers. The users interact

with the system from different operating systems both on desktop and mobile devices. Supported mobile devices are only limited to those having one of the following operating systems or browser:

- Mobile:
 - iOS v.7 to 9 and future versions
 - Android 4.2 to 5 and future versions
- Desktop
 - Browser Chrome (v.46+)
 - Browser Internet Explorer (v.11+ or Edge)
 - Browser Firefox (v.21+)

Dispatches in case of new taxi request were forwarded to drivers by radio, so the previous system didn't have any particular information system. The application will have access to the DBs so it will need an interface with the current DBMS. It will also have an interface for the users.

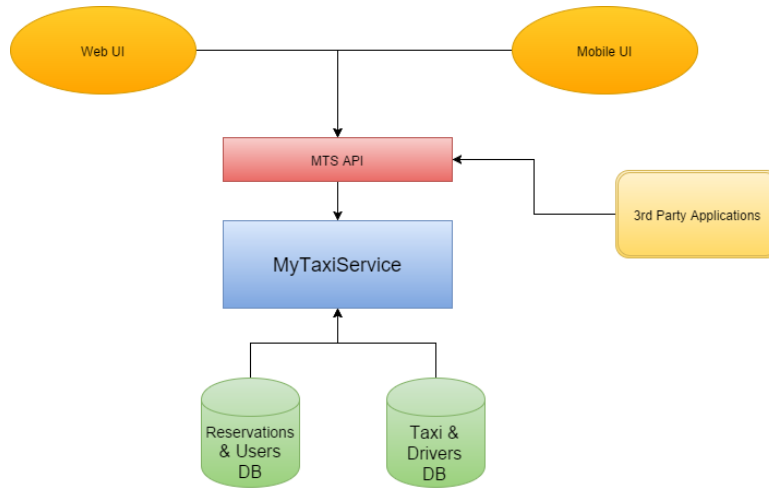


Figure 1: Overall view of *MyTaxiService* system

2.2 Product functions

The application will provide a set of functionalities for taxi reservations and also for saving money by sharing your ride. It will indeed be designed to improve the management of taxi queues in the various zones of the city and to decrease average customers' waiting time.

In order to implement a fair management of taxi queues the application will

be based on taxis' locations. The city is divided in zones of approximately 2 km² each and a queue is associated to every single zone. The taxi availability is constantly communicated to the system through the driver mobile application. It can be in three different states: **free**, **shared** or **locked**. The taxi is set as *free* as soon as the taxi driver logs into the application and goes back to this state when a ride is terminated. When a taxi becomes available it is inserted in the right queue, depending on its GPS location. When a ride request arrives, it is associated to a zone, depending on the starting address of the request, and the request is forwarded to the first taxi in the assigned zone's queue. If no taxi is available in that zone, the research is expanded to the adjacent zones. If a taxi driver declines a request, his taxi will be put at the bottom of the queue.

2.3 User characteristics

- **Visitor:** a visitor is whoever opens the application without having been authenticated.
- **Passenger:** a passenger is an authenticated user who is capable of reserving a ride from a certain location to another. He's also able to share a ride with other passengers or to look for shared rides.
- **Taxi driver:** a taxi driver is an authenticated user who has been granted permissions to confirm or not a reservation when his taxi is *free*. In case of confirmation he should list as *busy* and will not receive any incoming reservations aside shared ones. In that case the taxi will become *shared*.

2.4 Constraints

2.4.1 Regulatory policies

MyTaxiService will be released on Google Play Store, Apple App Store and Windows Store and so it must respect their market policies.

2.4.2 Hardware limitation

MyTaxiService will support only mobile devices newer than 2012. The mobile application will be suitable for devices with screen sizes starting from 4.2'

2.4.3 Interfaces to other applications

MyTaxiService is strongly dependent on MySQL as the main DBMS. It also uses PayPal as the only PSP.

2.4.4 Parallel operation

MyTaxiService must support parallel operations on the same DB coming from both taxi drivers' and passengers' applications.

2.4.5 Reliability requirements

MyTaxiService does not have reliability requirements.

2.4.6 Criticality of the application

The application doesn't have a critical role inside the taxi system, since any of the actions done by *MyTaxiService*, can be accomplished by phone.

2.4.7 Safety and security considerations

Since the payments are processed externally and the data is handled by an external DBMS, there are no specific security issues.

2.5 Assumptions and dependencies

- **Taxi drivers must have a tablet** in order to use the application and to manage incoming calls.
- **A standard user is whoever needs a taxi service** and is using the new available technology. **He must have a smartphone or a device with an internet access** so he can use either the web or the mobile application.
- Web and mobile applications have the same functionalities aside the registration that is only available on the web.
- **Users have a GPS-enabled devices.**
- **At least a taxi is always available for a passenger.** It's not mandatory that it's available in the passenger request's area.
- **A taxi returns in the city if the end of his previous ride is outside the city.**
- **The division into zones is given from the city management.**

2.6 Goals

- [G1] Allow visitor to become a registered user (passenger).
- [G2] Allow taxi drivers and passengers to log in into the application.
- [G3] Allow passengers to request a taxi.
- [G4] Allow passengers to see the code and the ETA of the incoming taxi.
- [G5] Allow taxi drivers to accept or refuse a call.
- [G6] Allow passengers to reserve a taxi for a certain time and place (origin and destination).

- [G7] Send a confirm notification to the passenger and make a taxi *busy* at least 10 minutes before the reservation time.
- [G8] Allow a passenger to choose a shared ride.
- [G9] Manage the taxis in order to minimize the waiting time of the passengers and the inactivity of the taxi drivers.

2.7 Apportioning of requirements

List of possible future functions:

- Friends list and possibility to share rides specifically with them.
- Possibility to cancel or modify a reservation.

3 Scenarios

3.1 Scenario 1: Usage of the app

Timmy needs a taxi to go back to his hotel in Sesame Street. So he picks up his phone and opens the *MyTaxiService* app. He is asked to allow the application to use his position and after that, he's asked to pick a destination. He writes "Sesame Street" and then he is asked to wait a moment. After few seconds, a pop-up appears on the screen containing the taxi code, the EWT for the taxi, the ETA to home and the expected fare. He waits for the taxi to arrive and after he's back home, he's notified of the fare which is directly drawn from his credit card.

3.2 Scenario 2: Taxi reservation

Bob is abroad on holiday and is going to take a plane to go back to his city, but he realizes that no one can take him back home from the airport, so he decides to book a taxi ride. His smartphone is dead so he turns on his laptop, connects to the airport's Wi-Fi and opens the *mytaxiservice.com* website. He signs in with his credentials and opens the "New Ride" view. He selects the starting place(the airport), the destination ("21st avenue"), and chooses the departing time. During his flight he charges his mobile phone so when he lands he turns on his phone and receives all the information about his reservation. At the booked time the taxi arrives and brings him home. The taxi driver communicates him the ride fare and he pays in cash.

3.3 Scenario 3: Shared ride, passenger's side

Penny bought a ticket to go to the theater for a new musical. She decides to go with a taxi but she wants to save money because she has already spent a lot for the show ticket. So she opens the *myTaxiService* application and chooses

a shared ride for the theater. In the meanwhile Leonard realizes he has to go to a restaurant near the theater for a dinner with his colleagues but his car is broken. So he picks up his mobile phone and opens the *myTaxiService* app. He decides to choose a shared ride to save some money so he selected the option and confirms. Both Penny and Leonard receive the information of the same taxi for their ride. The taxi first stops at Leonard's home and the taxi driver tells him that his ride is shared and he will stop to take another passenger. The taxi stops after a few minutes at Penny's home and then it takes them to their destination. Once arrived the taxi driver tells them their respective fares and they pay.

3.4 Scenario 4: Shared ride, driver's side

Walter White, a taxi driver for the *myTaxiService* company, is driving along "Main Street" when he receives a notification from the taxi application for a shared ride. He accepts the request and goes at the given address in order to pick up the passenger. After driving for a few minutes he receives another request so he stops to pick up three other passengers and his car is now full. He drives to the first destination, selects the "Terminate Ride" button and tell the passenger the fare that will be charged on his PayPal account. Then he drives to the last stop and leaves the other passengers, he takes the payment in cash and terminate the current ride on the app.

3.5 Scenario 5: Taxi driver notifications

Jack Tahyer is a taxi driver and he has to put *myTaxiService* onto his mobile. He is bottled up in traffic due to a car accident. He receives a notification for a new passenger but it would take too much time to reach the customer so he decides to decline the request. After some time he finally has open road and he receives another request for a shared ride. He accepts the request and follows the GPS directions.

4 Specific requirements

In this section detailed descriptions of the functional and quality requirements will be presented together with the explicit features requested.

4.1 External interface requirements

In the paragraphs below an analysis of inputs and outputs of the system will be presented both from the user and software perspectives.

4.1.1 User interfaces

The first view a user should see is the "Sign In" page both from mobile and web apps. Then, if it's their first time within the application, the user should move

to the sign up page, in order to begin the registration process.

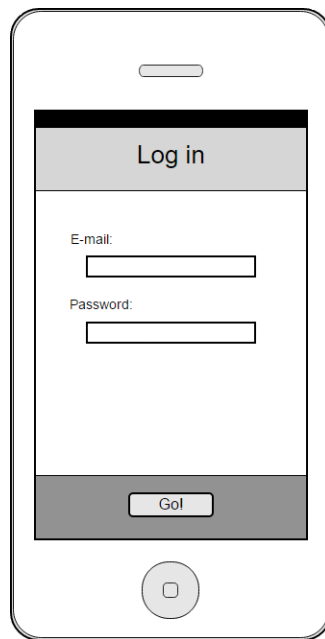


Figure 2: Login page for mobile application

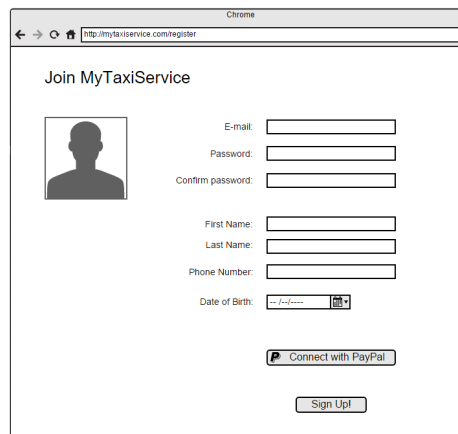
A web registration page titled "Join MyTaxiService". On the left, there is a placeholder for a profile picture. To the right of the profile picture, there are several input fields: "E-mail:", "Password:", "Confirm password:", "First Name:", "Last Name:", "Phone Number:", and "Date of Birth:". Below these fields, there is a button labeled "Connect with PayPal" and a button labeled "Sign Up!". The page is displayed in a Chrome browser window with the URL "http://mytaxiservice.com/register" in the address bar.

Figure 3: Web registration page

If however they already have an account, after the sign in process:

- If the user is a **passenger**, they will see the *Home* page which contains a brief log of his rides and an option to begin a *new ride*.

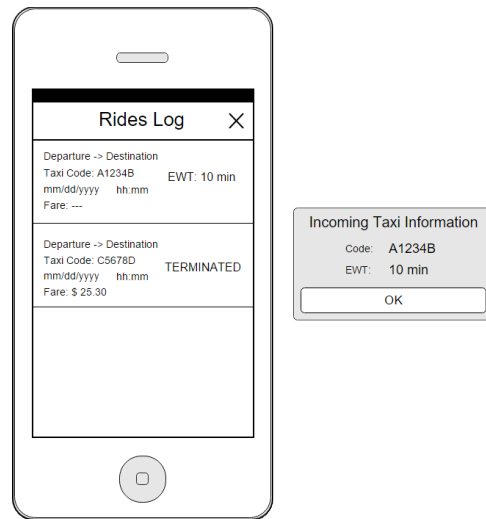


Figure 4: Rides informations

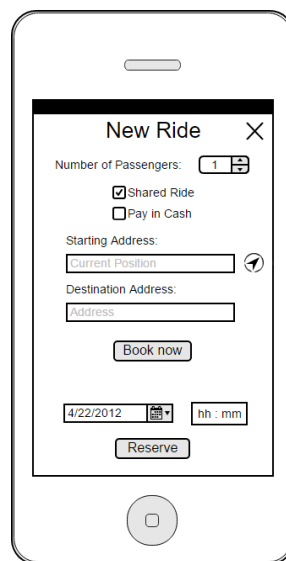


Figure 5: New Ride page

- If instead the user is a **taxi driver** then they will directly see the *Driver* page which shows recent notifications and a map with the route to the next destination (*Figure 6*). If a shared ride is accepted the interface changes (*Figure 7*) showing all the passengers that have booked the shared ride and the possibility to terminate the ride for a single user. When a ride is terminated, for both normal and shared rides, a notification pops up (*Figure 8*) showing the fare that should be payed and the payment method chosen by the passenger.

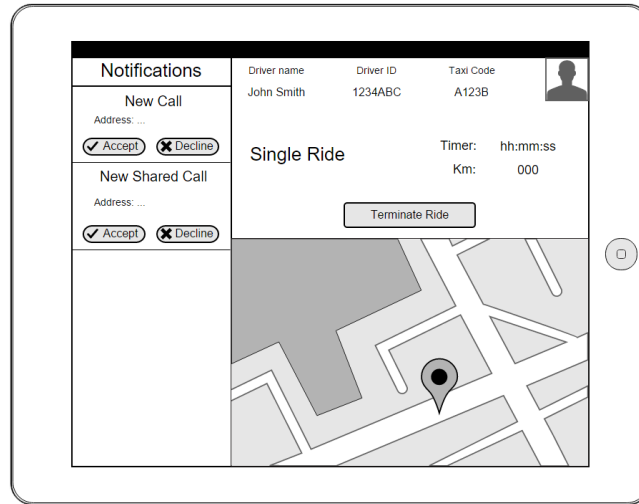


Figure 6: Taxi Driver interface

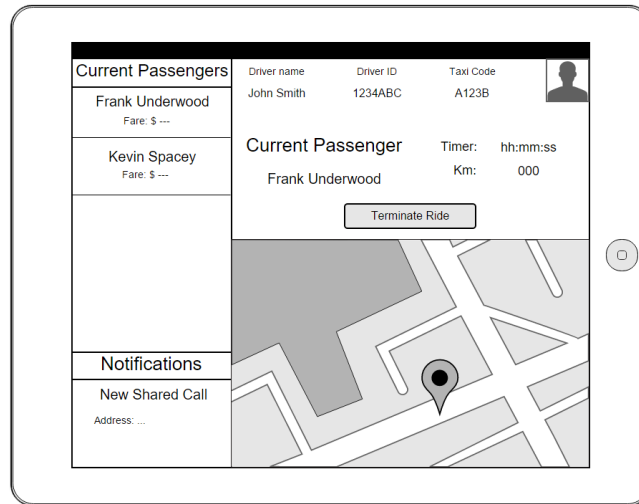


Figure 7: Taxi Driver interface for shared ride

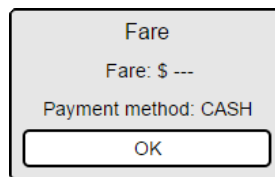


Figure 8: Notification for terminated ride

4.1.2 Hardware interfaces

Since the application is compliant to modern standards, there is no specific limitation on the hardware itself. The only limitation to use the app is that it must be used from a GPS enabled device.

4.1.3 Communication interfaces

Since both the front-end and the back-end are being developed from scratch, there are no specific restrictions on the communication between system components except from the underlying data layer which will be discussed later.

4.1.4 Software interfaces

The system has to interact with a preexisting database containing taxis and drivers. The application also provides public API for further and additional integration.

4.2 Functional requirements

4.2.1 User 1 - Visitor

1. ***Registering*** - [G1]

A visitor can register into the application by compiling the registration form.

Requirements:

- (a) Visitor must enter a valid email.
- (b) Visitor must enter a valid phone number.
- (c) Visitor must be adult in order to register.

4.2.2 User 2 - Passenger

1. ***Connect to PayPal***

A passenger can connect his account to PayPal. The application will ask permissions to use passenger's PayPal account.

2. ***Pay with PayPal***

A passenger can pay a ride with PayPal.

Requirements:

- (a) Passenger must have connected his account with PayPal.

3. ***Login*** - [G2]

A passenger has to login in order to access to all the other passenger functions.

4. ***Request a new taxi ride*** - [G3, G6, G8]

A passenger can book a taxi. He will have to fill the following input information: starting point (automatically filled if GPS is enabled) and destination.

Requirements:

- (a) A passenger must have not requested another taxi at the same time.

Domain properties:

- (a) Starting place must be in the city in which the service is being deployed.
- (b) Destination place must be in the country.

There are some particular cases:

- Shared taxi ride: request for a new shared ride.
- Reservation: passenger will have to add a booking date and the starting point if auto-fill is disabled.

Additional domain properties:

- (a) The ride time must be in the future within a month and at least 2 hours from current time.

5. ***Look rides log*** - [G4]

A passenger can look at all his rides. They have a taxi ID (besides future rides), a fare (besides current and future rides), a starting and an ending place.

Requirements:

- (a) Passenger must have booked at least one ride.

4.2.3 User 3 - Taxi driver

1. ***Manage a ride notification*** - [G5]

A taxi driver can accept a ride notification. The application will show taxi information to the passenger and it will prevent the driver to receive any new standard ride notification (i.e. *busy*).

Requirements:

- (a) The taxi related to the driver is not *busy* or *shared*.
- (b) The taxi driver must answer the call otherwise after 15 seconds the call is automatically refused.

Domain property:

- (a) Maximum number of passengers is 4.

There are some particular cases:

- Accept a shared ride notification: the taxi is *shared* instead of *busy*.
Additional requirements:
 - (a) No passenger must be in the car. Besides the first shared ride notification, the others are automatically accepted until the end of the ride.
- Decline a ride notification: the taxi driver can decline a ride request.
Additional requirements:
 - (a) Taxi must not be *shared*.

2. ***Start a ride***

A taxi driver can start a ride for a passenger.

Requirements:

- (a) There must be a pending request from the passenger.

3. ***Finish a ride***

A taxi driver can finish a ride. The application will show the final fare and distance and will *free* the taxi.

Requirements:

- (a) The taxi related to the taxi driver is *busy* or *shared*.
- (b) The ride must have been started.

4.2.4 General management

1. **Queue management** - [G9] The application must manage a fair queue management through two kinds of zones.

Static zones are given from the city management and they have a taxi queue. Depending on the statistical request average, the queue is longer or shorter.

Dynamic zones are made on a ride's creation and based on the request's location.

More details about queue management algorithms are shown on the next document about design.

Requirements:

- (a) A taxi is stored in one and only one queue.
- (b) A taxi has one and only one status.

Domain properties:

- (a) A queue is associated to one and only one zone

4.3 Use Cases

In this sections a UML Use Case Diagram will be presented, giving a general idea of the application features and then some features will be analyzed more in detail by UML Sequence Diagrams.

4.3.1 Visitor registration

Actor	Visitor
Goal	[G1]
Entry Condition	NULL
Event Flow	<ol style="list-style-type: none"> 1. Visitor opens the registration page. 2. Visitor fills all the fields. 3. Visitor confirms the registration clicking on "Sign Up!". 4. The data is checked. 5. The data is registered.
Output Condition	Visitor successfully registers and becomes a passenger. From now on he can log into the application.
Exceptions	<ul style="list-style-type: none"> • One or more fields are empty. • Email is already associated to another account. • Password doesn't respect the constraints. <p>All the exceptions are handled by notifying to the visitor the occurred problem and resetting the form.</p>

4.3.2 Login

Actor	User, Visitor
Goal	[G2]
Entry Condition	The user is already registered.
Event Flow	<ol style="list-style-type: none">1. The visitor opens the <i>login</i> page.2. The visitor fills the email and password fields.3. The visitor confirms clicking on “Go!”.4. The user is redirected at the personal page.
Output Condition	The registered user is now logged in and has access to his personal features.
Exceptions	<ul style="list-style-type: none">• One or both email and password are wrong or empty. The exception is handled by notifying to the user the occurred problem and resetting the form.

4.3.3 Standard request

Actor	Passenger
Goal	[G3]
Entry Condition	The passenger is logged in.
Event Flow	<ol style="list-style-type: none">1. Passenger opens the “New Ride” page.2. Passenger fills the addresses fields, selects the number of passengers and unchecks the <i>Shared Ride</i> option.3. Passenger confirms clicking on “Book Now”.
Output Condition	The request is sent and the passenger waits for the taxi information.
Exceptions	<ul style="list-style-type: none">• One or more fields are empty or invalid. The passenger is notified of the occurred problem and the form is reset. <ul style="list-style-type: none">• No taxi driver is found in the area. The system looks for a taxi in the nearest areas around the interested one.

4.3.4 Rides Log

Actor	Passenger
Goal	[G4]
Entry Condition	The passenger is logged in. The passenger has booked or reserved a ride.
Event Flow	<ol style="list-style-type: none">1. The passenger opens the <i>Rides Log</i> page.
Output Condition	The passenger has access to the incoming taxi information and to all of his previous rides.
Exceptions	NULL

4.3.5 Taxi reservation

Actor	Passenger
Goal	[G6]
Entry Condition	The passenger is logged in.
Event Flow	<ol style="list-style-type: none">1. The passenger opens the “New Ride” page.2. The passenger fills the addresses fields, selects the number of passengers, unchecks the <i>Shared Ride</i> option and selects date and time for the reservation.3. The passenger confirms clicking on “Reserve”.
Output Condition	The request is sent and it will be processed 10 minutes before the reservation time.
Exceptions	<ul style="list-style-type: none">• One or more fields are empty.• The ride time is less then two hours from the current time. <p>The exception is handled by notifying to the user the occurred error and resetting the form.</p>

4.3.6 Taxi reservation (already booked)

Actor	Passenger, Taxi driver
Goal	[G5], [G7]
Entry Condition	The passenger is logged in. The taxi driver is logged in. There is a pending taxi request from the passenger for the next 10 minutes.
Event Flow	<ol style="list-style-type: none">1. The system searches a taxi in the requested location area.2. A notification is sent to the taxi driver found.3. The taxi driver accepts the call.4. The EWT is calculated.5. The taxi ID code and the EWT are sent to the passenger.
Output Condition	The request is sent and the passenger receives the taxi information.
Exceptions	<ul style="list-style-type: none">• No taxi driver is found in the area. <p>The system looks for a taxi in the nearest areas around the interested one.</p> <ul style="list-style-type: none">• The taxi driver declines the call. <p>The exception is handled searching for another taxi in the area and restarting from the event flow’s point 2.</p>

4.3.7 Shared Ride request

Actor	Passenger
Goal	[G8]
Entry Condition	The passenger is logged in.
Event Flow	<ol style="list-style-type: none">1. The passenger opens the “New Ride” page.2. The passenger fills the addresses fields, selects the number of passengers, checks the <i>Shared Ride</i> option and selects date and time for the reservation.3. The passenger confirms clicking on “Reserve”.
Output Condition	The request is sent and the passenger waits for the taxi information.
Exceptions	<ul style="list-style-type: none">• One or more fields are empty or invalid. The passenger is notified of the occurred problem and the form is reset.• No taxi driver is found in the area. The system looks for a taxi in the nearest areas around the interested one.

4.3.8 Ride management and completion

Actor	Taxi driver
Goal	NULL
Entry Condition	The passenger has requested a ride. The pointed out taxi in the notification arrives at the established location.
Event Flow	<ol style="list-style-type: none">1. The driver starts the ride by clicking on “Start Ride” when the passenger gets in the taxi .2. The driver takes the passenger to the destination.3. The driver clicks on “Terminate Ride”.4. A notification appears showing the total fare that has to be payed and the payment method chosen by the passenger during the reservation phase.
Output Condition	The taxi is now available in his current area and the taxi driver can receive notifications for new rides.
Exceptions	NULL

4.3.9 Shared ride management

Actor	Taxi driver
Goal	NULL
Entry Condition	A passenger booked a shared ride and he's already on board.
Event Flow	<ol style="list-style-type: none">1. The driver receives a new shared ride call.2. The new passenger is added to the <i>Current Passengers</i> of the taxi.3. The driver gets to the new address to take the new passenger.4. The second ride is started when the new passenger gets in the taxi.5. The <i>Current Passenger</i> is automatically set to the first passenger that will leave the taxi6. The driver takes the first passenger to the destination and terminates his ride.7. A notification appears showing the total fare that has to be payed and the payment method chosen by the passenger during the reservation phase.
Output Condition	The taxi is still shared with other passenger on board.
Exceptions	NULL

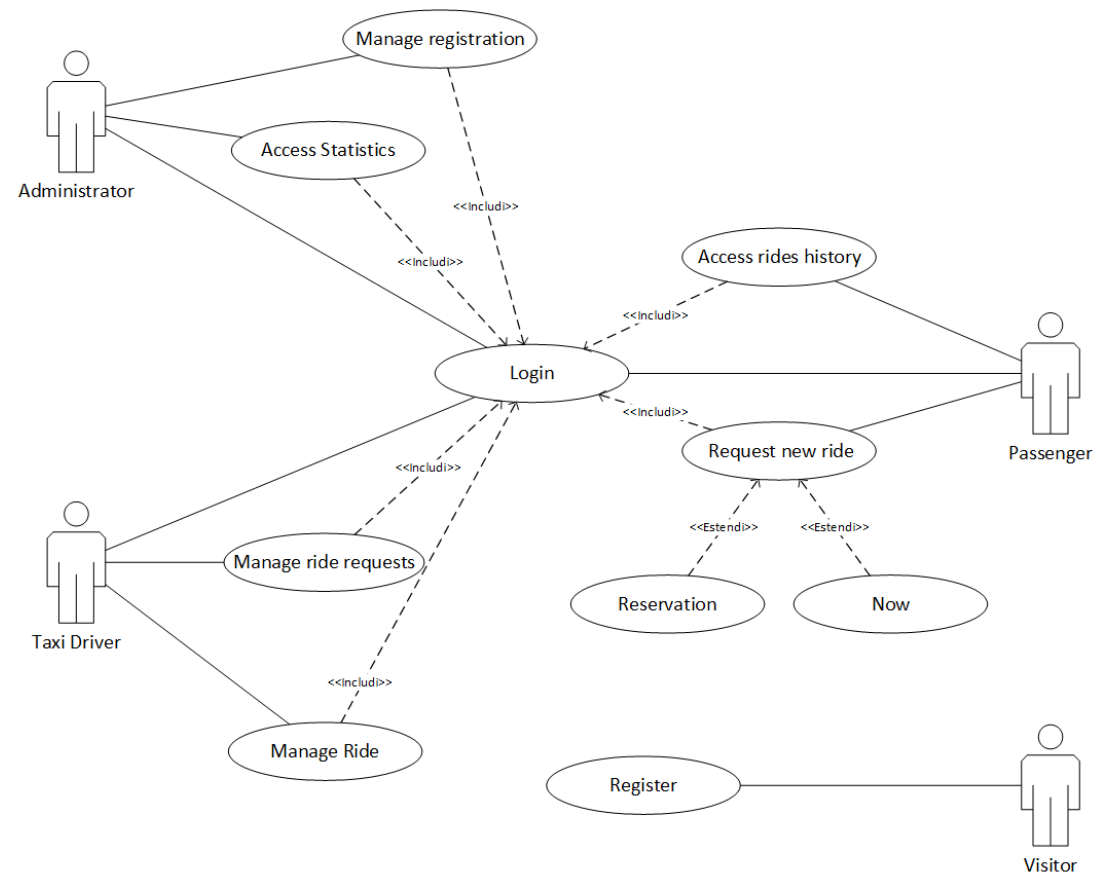


Figure 9: Use Case Diagram of the whole application usage

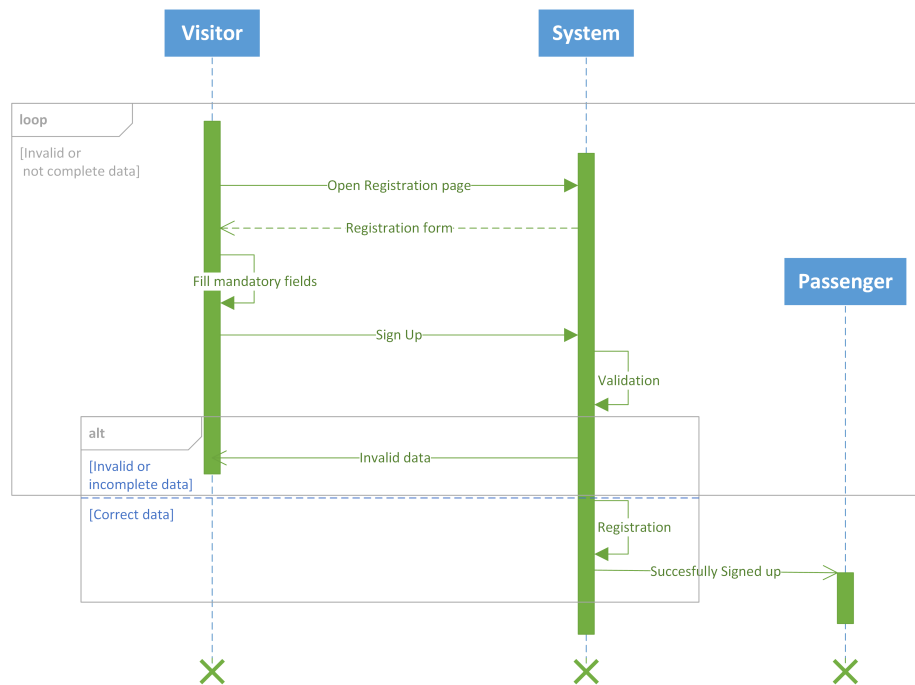


Figure 10: Registration process - Use case 4.3.1

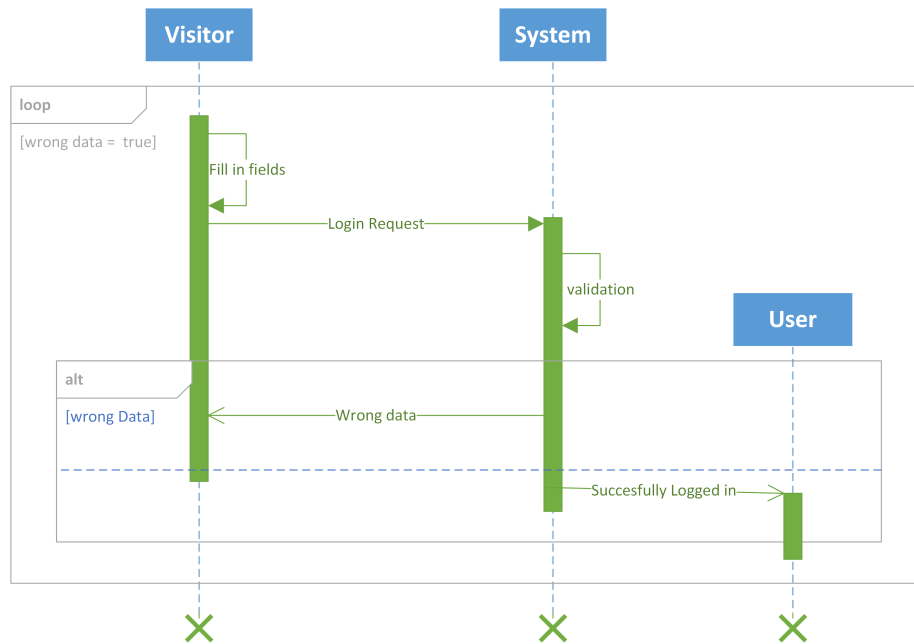


Figure 11: Login process - Use case 4.3.2

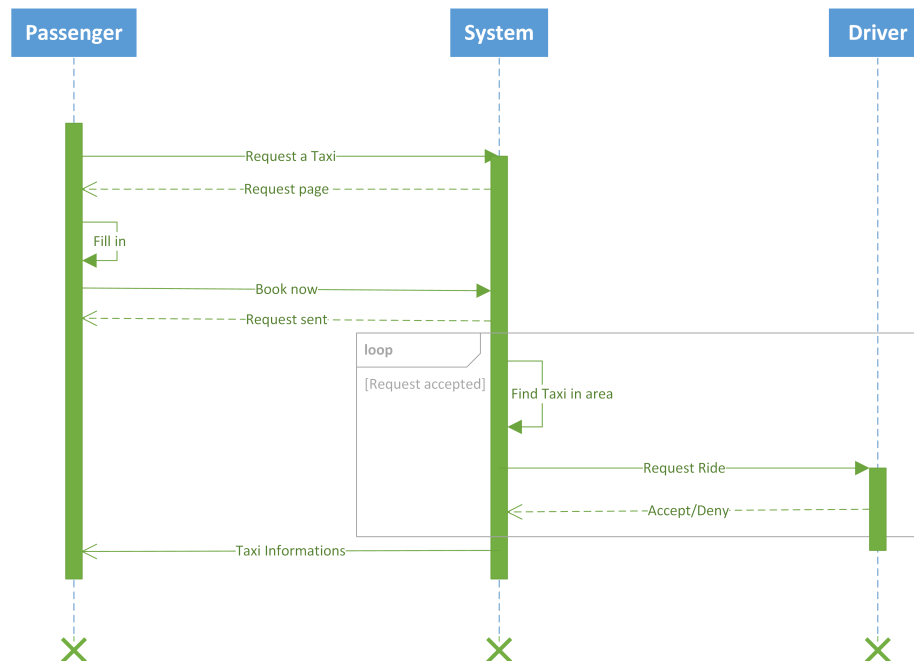


Figure 12: Taxi request process - Use case 4.3.3/5 and 4.3.6

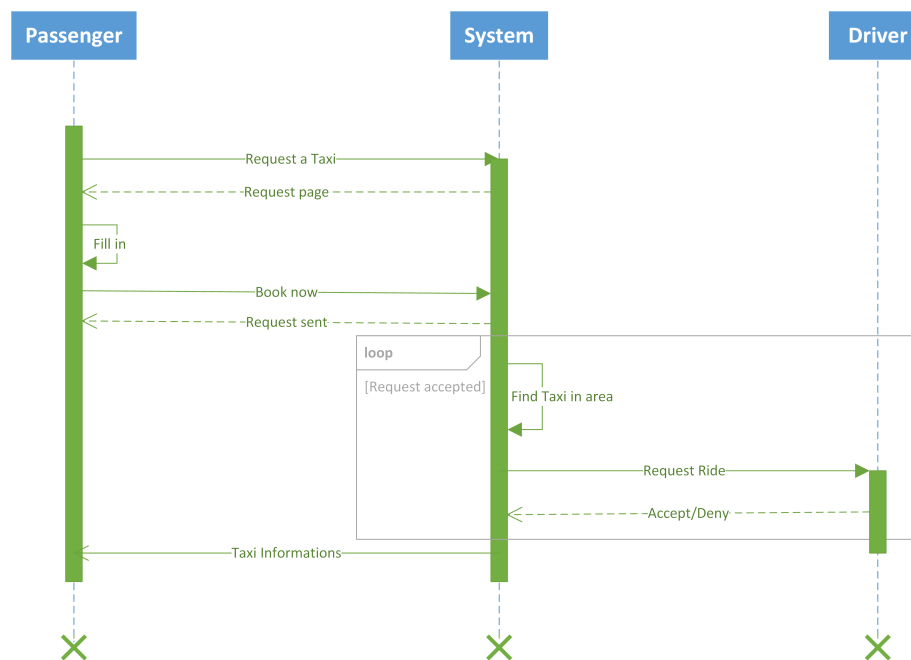


Figure 13: Shared taxi request process - Use case 4.3.7

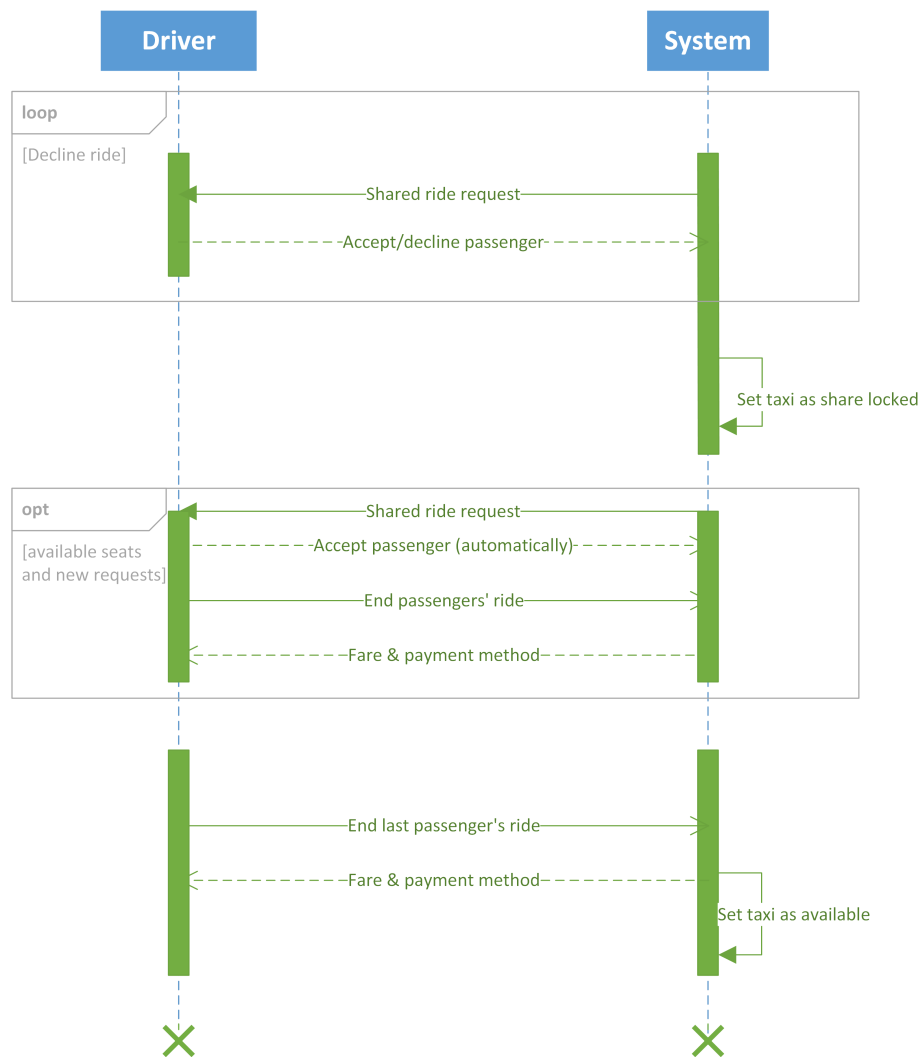


Figure 14: Ride management process - Use case 4.3.8/9

5 Appendix

5.1 Class Diagram

Z

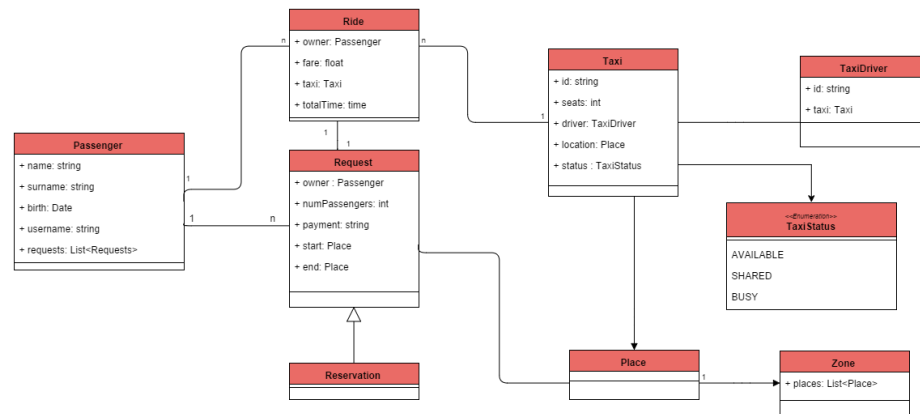
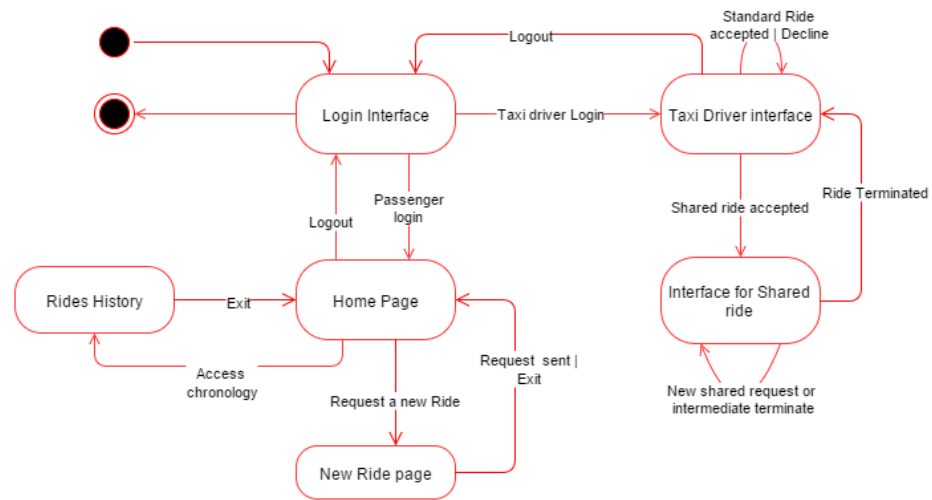


Figure 15: Simplified Class Diagram of main entities

5.2 Statechart Diagram



5.3 Alloy

5.3.1 Purpose

The Alloy language has been used for its original purpose, which is *model finding*. For this sake, in an iterative way we started from a basic class diagram and basic goals and requirements we could think at the beginning and we improved and extended them based on alloy worlds results.

5.3.2 Code

```

module mts/timing

/*
 * Creates a single line of and provides some definitions of time intervals and instants
 * together with basic operations among them.
 */
/* author: Ennio Visconti
 * author: Simone Zocchi
 * author: Khanh Huy Paolo Tran
 */
|
sig TimeInterval{
    start: one Int,
    end: one Int
}

sig TimeInstant extends TimeInterval{}
/** Every TimeInterval which has same starting and ending is a TimeInstant and nothing else is a TimeInstant */
fact { all t: TimeInterval | t.start = t.end <=> t in TimeInstant}

fact { all t1,t2: TimeInterval | (t1.start = t2.start && t1.end = t2.end) <=> t1 = t2 }

one sig Now in TimeInterval {}
fact { Now.start = 0 && Now.end = 0 }

/** Every TimeInterval must have a positive length */
fact PositiveLength {
    no t: TimeInterval | t.end < t.start
}

/** Returns true iff intervals overlap */
pred AreOverlapping[i1: TimeInterval, i2: TimeInterval] {
    i1.start <= i2.end && i2.start <= i1.end
}

/** Returns true iff intervals start at the same time */
pred StartTogether[i1: TimeInterval, i2: TimeInterval] {
    i1.start = i2.start
}

/** Returns true iff intervals start at the same time */
pred EndTogether[i1: TimeInterval, i2: TimeInterval] {
    i1.end = i2.end
}

pred show {}

run show for exactly 2 TimeInstant, 4 TimeInterval

```

Figure 16: The Alloy library “timing” built to support time requirements

```

module mts/entities

/*
 * Contains definitions for MyTaxiServer logic entities
 * and provides common operations among them.
 *
 * author: Ennio Visconti
 * author: Simone Zocchi
 * author: Khanh Huy Paolo Tran
 */

private open mts/timing

/***** TAXI DEFINITIONS *****/
abstract sig TaxiStatus {}
one sig Available, Shared, Busy extends TaxiStatus{}

some sig Taxi {
    passengers: some Passenger,
    driver: one TaxiDriver,
    status: one TaxiStatus,
    location: one Place
}

sig Ride {
    owner: one Passenger,
    means: one Taxi,
    from: one Place,
    to: one Place,
    duration: one TimeInterval,
    relatedRequest: one Request
}

fact NoConcurrentRidesPerUser {
    no r1, r2: Ride | AreOverlapping[r1.duration, r2.duration] ## r1.owner = r2.owner
}

fact DriverUnicity { all t: Taxi | t in t.driver.taxi }

fact MaximumTaxiSeats { no t: Taxi | #(t.passengers) > 4 }

let MIN_RESERVATION_OFFSET = 2 //Needed time between a reservation and the actual ride

```

Figure 17: The Alloy library “entities” built to define basic entities and domain constraints. Part 1


```

/*****
/**                                **/
REQUEST DEFINITIONS
*****/

sig Request {
  owner: one Passenger,
  users: set Passenger,
  origin: one Place,
  destination: one Place,
  time: one TimeInstant,
  relatedRide: lone Ride,
  relatedNotification: one Notification
}

sig Reservation extends Request {}

sig Notification {
  relatedRequest: one Request,
  driver: one TaxiDriver
}

fact RideRequestSameOwner {
  all ride: Ride, request: Request | (request in ride.relatedRequest
  && ride in request.relatedRide) <=> (request.owner = ride.owner)
}

fact TaxiNotificationRelation{
  all n: Notification | n.driver.taxi = n.relatedRequest.relatedRide.means
}

fact MinimumReservationTime {
  all r:Reservation | r.time.start >= Now.start + MIN_RESERVATION_OFFSET
}

fact SameIntervalSameRequest {
  all r1,r2: Request | r1.time = r2.time <=> r1 = r2
}

```

Figure 18: The Alloy library “entities” built to define basic entities and domain constraints. Part 2

```

/*****
**                               **
**                               **
*****/
abstract sig User {}

sig TaxiDriver extends User {
    taxi: one Taxi
}

sig Passenger extends User {
    taxi: lone Taxi
}

fact LoneTaxiForAPassenger {
    no p:Passenger | (p not in p.taxi.passengers)
}

/*****
**                               **
**                               **
*****/
sig Zone {
    places: some Place
}

sig Place {}

```

Figure 19: The Alloy library “entities” built to define basic entities and domain constraints. Part 3

```

open mts/entities
open mts/timing

// No reservation can have the same Passenger in both owner and users
fact NoDuplicatePerson {
    no r:Request | r.owner in r.users
}

fact DifferentDestination {
    no r: Request | r.origin = r.destination
}

// No requests overlapping from the same user.
/**
 * If this assertion is verified, also RequestNewStandardRide and RequestNewSharedRide
 * are verified because they does not need further verifiable requirements
 */
assert NoConcurrentRequests {
    no disj r1, r2: Request | r1.owner = r2.owner
    no (r1.time = r2.time)
}

assert RequestNewStandardRide {}
assert RequestNewSharedRide {}

//Booking time must be in the future within a month and at least 2 hour from actual time.
assert BookNewRide {
    no r: Reservation | r.time = Now
}

check NoConcurrentRequests
check BookNewRide

```

Figure 20: The Alloy requirements assertions and checks based on the Passenger User Class

```

open mts/entities

fact NotificationToAvailable{
    all n: Notification, r: Request | n.relatedRequest = r <=>
        (r.relatedRide.means.status = Available)
}

assert AcceptRegularDrive {
    no t: Taxi, n:Notification, r:Request | (t.status = Busy or t.status = Shared)
    ## n.driver = t.driver ## n.relatedRequest = r
}

assert StartRide {
    no ride: Ride, request: Request | ride.owner not in request.owner
    ## ride.relatedRequest= request ## request.relatedRide=Ride
}

check AcceptRegularDrive
check StartRide

```

Figure 21: The Alloy requirements assertions and checks based on the Taxi Driver User Class

```

3 commands were executed. The results are:
#1: No counterexample found. NoConcurrentRequests may be valid.
#2: No counterexample found. BookNewRide may be valid.
#3: Instance found. show is consistent.

3 commands were executed. The results are:
#1: No counterexample found. AcceptRegularDrive may be valid.
#2: No counterexample found. StartRide may be valid.
#3: Instance found. show is consistent.

```

Figure 22: The Alloy run results

5.3.3 Example Worlds

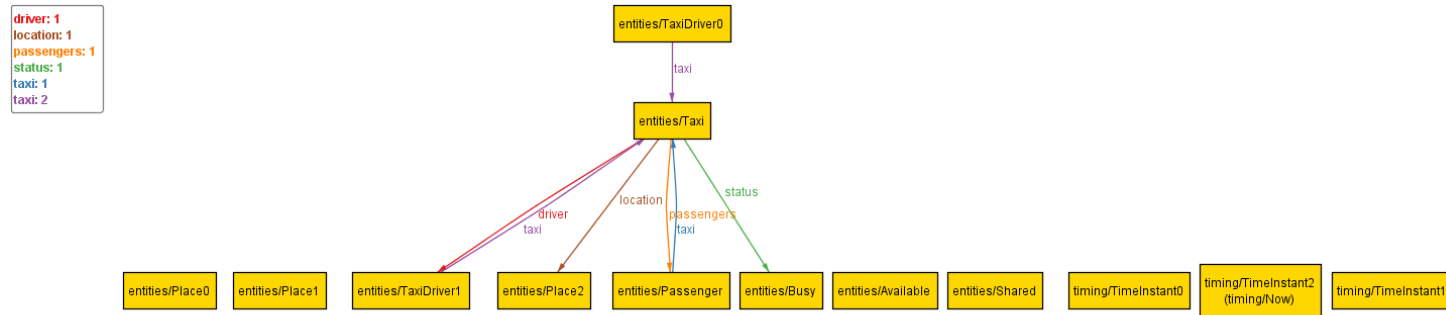


Figure 23: An Alloy passenger's world

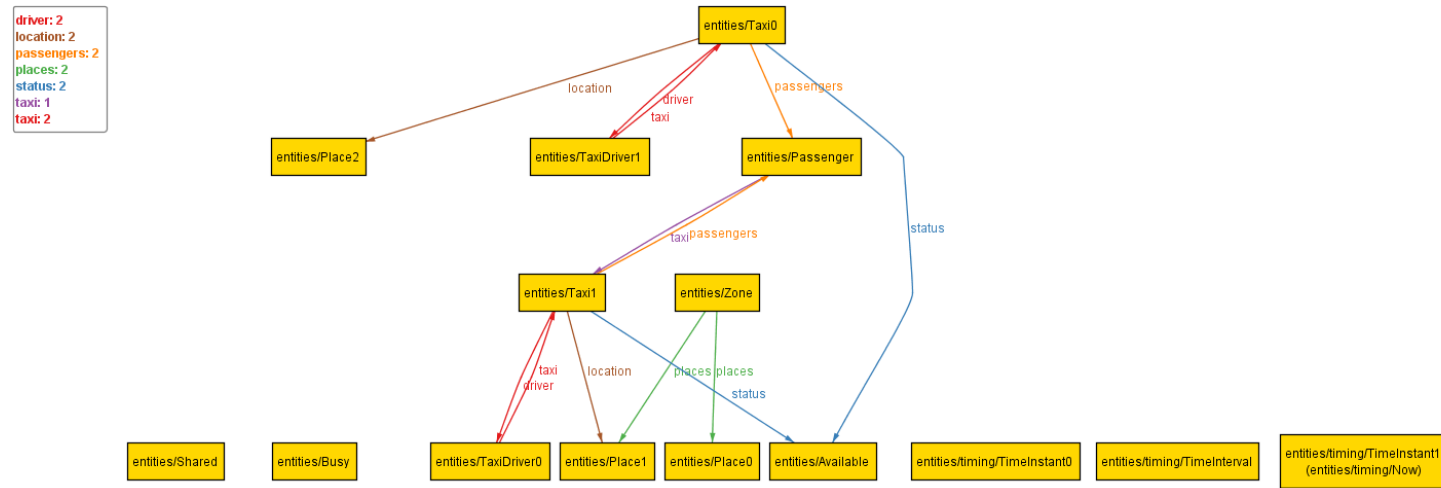


Figure 24: An Alloy taxi driver's world

6 Review

- [2.5] A taxi returns in the city if the end of his previous ride is outside the city.
- [2.5] The division into zones is given from the city management.
- [2.6] [G9] Manage the taxis in order to minimize the waiting time of the passengers and the inactivity of the taxi drivers.
- [4.2] ***Queue management - G9***
The application must manage a fair queue management through two kinds of zones.
Static zones are given from the city management and they have a taxi queue. Depending on the statistical request average, the queue is longer or shorter.
Dynamic zones are made on a ride's creation and based on the request's location.
More details about queue management algorithms are shown on the next document about design.

Requirements:

- (a) A taxi is stored in one and only one queue.
- (b) A taxi has one and only one status.

Domain properties:

- (a) A queue is associated to one and only one zone
- [4.2.3] Manage a ride notification...
Requirements:
 - (b) The taxi driver must answer the call otherwise after 15 seconds the call is automatically refused.