Requirements Analysis and Specifications Document

Figure 1: Politecnico di Milano

Version 0.1

- Claudio Cardinale (mat. 849760)
- Gilles Dejaegere (mat. **MAT**)
- Massimo Dragano (mat. 775392)

Contents

Introduction

Description of the given problem

We will project and implement myTaxiService, which is a service based on mobile application and web application, with two different targets of people:

- taxi driver
- clients

The system allows clients to reserve taxi via mobile or web app, using GPS position to identify client's zone (but the client can insert it manually) and find taxi in the same zone.
On the other side the mobile app allows taxi driver to accept or reject a ride request and to communicate automatically his position (so the zone).

The system includes extra services and functionalities such as taxi sharing

Actual system

Until now the taxi company has a system where the clients have to call a call center communicating its position via voice (so it can be not correct), the call center's operator inserts the request into an internal information system and the taxi driver can accept or reject it via a dedicated hardware device.

The system sends automatically an SMS to the client with the estimated arrival time and the taxi name.

This system stores taxi information into a Mysql database.

Goals

The main goal of the system is to be more efficient and reliable than the existing one in order to decrease costs of the taxi management and offer a better service to the users.

Taxi drivers:

- [G1] Allows taxi drivers to sign up into the system.
- [G2] Allows taxi drivers to log in the system.
- [G3] Allows taxi drivers to precise to the system if they are available or not.
- Taxi drivers should receive a notification for incoming request. **is it a goal ?**

- Taxi drivers should receive a notification if they have to take care of another user (during a shared ride). **is it a goal ?**
- [G4] Allows taxi drivers to accept or decline incoming requests for an immediate ride
- [G5] Allows taxi drivers to accept or decline incoming request for a later reservation.

Users:

- [G6] Allows users to request for an immediate taxi ride.
- [G7] Allows users to request for the reservation of a taxi at least two hours in advance.
- Users should receive a notification with the code of the taxi that takes care of the user's request. **is it a goal ?**
- Users should be notified if no taxi driver is able to perform the users request. **Is this a goal ?**
- [G8] Allows users to require to share the taxi.

Domain properties

We suppose that these properties hold in the analyzed world :

- Actual drivers are already registered on the previous system
- A taxi driver accepting a ride of reservation will actually take care of the request.
- A user requiring a taxi will actually take it.
- All the GPS always give the right position.
- The GPS of the taxi drivers can not be switched off.
- Taxi drivers answer all types of demands in less than 5 minutes.
- The user pays the taxi driver directly for each commission.
- A taxi can be in only one zone at the same time and this is the real zone.
- Users make a reservation two hours before the ride **Here as domain or do we have to do the requirements into G7?**
- When a new taxi driver joins in the taxi company the taxi company registers him in the information system. Analogously when a taxi driver exits from the company, the company deletes him from the information system.
- The taxi arrives at start point with max 30 minutes of delay
- Start zone is different from end zone

Glossary

- User: he is a client of the service. He should insert each time he performs a request the following information
  - Name
  - Phone number
  - Position, it can be taken automatically from GPS (either via APP or Web browser)
- Taxi driver: he is a taxi driver registered on the taxi company, which grants to taxi driver the access to this information system
- Queue: it is the taxi queue, when more than one taxi is in the same zone, there is a FIFO queue. So in this way when there is a new client the oldest taxi can take it. There is a queue for each zone.
- Ride: it starts when the taxi receives the request and ends when it leaves the last client of the ride. The simple ride is specified by start ride, user and taxi; but other ride types (like reservation or taxi sharing) have other parameters.
- Taxi sharing: it is the possibility that if different people (it's not required that they know each other) of the same start zone go to the same direction, even if the end is not the same, to use the same taxi and to have a unique group fee. A sharing ride is identified by users that use it and for each user the start and end point
- Reservation: it is the ability to reserve a taxi until two hours before time of ride, so when a reservation is done the system makes a normal taxi request 10 minutes before the ride. The reservation is identified by start point, end point, user and time.
- Taxi request: it is the request the system sends (automatically or after a user request) to taxi to specify a ride, specifying start point, user and other elements if they are available.
- User request: it is the request for a taxi drive as soon as possible, it contains the user data and the start point that can be get by GPS (current position) or inserting manually
- Zone: it is a zone of approximately 2 km^2, the city is split into these zones. From taxi position the system gets his zone and inserts the taxi into the zone queue. So the system guarantees a fair management of taxi queues
- Task: a task is an action done automatically by the server, for example "send request 10 minutes before ride" is a task
- Taxi: it is a means of transport that can bring only 4 passengers.
- System: it is the new system we will create with the database of the old system.

Assumptions

- There is an old system as described above.
- It exists a mobile application for users where users can make a reservation using the GPS position or by inserting their position
- The users are not registered in the system, because we need only their name and their position. **SEE REQUIREMENTS** A user is identified by his personal data: name and phone number
- There are only normal taxis for 4 passengers.
- The registration/deletion by company of a taxi driver is done in the same way of the old system, so we don't have to do this part.
- We need information only about taxi driver, not about taxi vehicle. So we store information only about taxi driver.
- The system doesn't need user registration, since it requires only identification data and position and since it works like the old system (where every user must say identification data via call). The real applications of many cities run in this way.
- We assume that if sharing option is selected it is not possible make a reservation for more than one person.
- All taxis of the city are regulated and use this system

Constrains

Regulatory policies

The system must require to user/taxi driver the permission to get his position and he has to manage sensible data (position, phone number) respecting the privacy law

Hardware limitations

- Mobile app
  - taxi driver:
    - 3G connection
    - GPS
    - Space for app package
  - user:
    - 3G connection
    - Space for app package
- Web app
  - Modern browser with AJAX support
  - ...

Interfaces to other applications

Interface with the old system. The new system will interface with the Mysql database of the old system.

Parallel operation

The server supports parallel operations from different users and different taxi drivers.

Reference documents

...

Proposed system

We will implement a client-server architecture (Fig. 2) based on common REST API and MVC pattern, so with just one server application we manage both web application and mobile application, obviously we will have version for taxi driver and version for users.

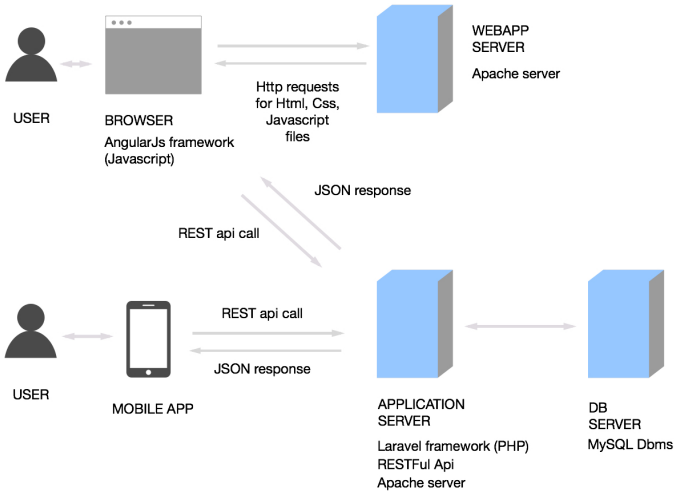


Figure 2: Architecture

WRITE MORE DETAILS

Identifying stakeholders HERE OR INTO 'Actor identifying'?

Do we have to write city or teachers?

Other considerations about the system

Actors identifying

The actors of our system are basically two:

- Taxi driver: it is a taxi driver registered automatically in the system by the taxi company
- User: he doesn't need to register himself to the system, since he uses the system only to call a taxi (so he have to insert only basic personal information and location)

Requirements

Functional requirements

Assuming that the domain properties stipulated in the paragraph [1.3] hold, and, in order to fulfill the goals listed in paragraph [1.2], the following requirements can be derived.

The requirements are grouped under each goal from which it is derived. The goals are grouped following under the users concerned.

Taxi drivers:

- [G1] Allows taxi drivers to sign up into the system:
  - The system must be able to check if it is an official taxi driver.
  - The system only allows official taxi drivers to register. **SEE DOMAIN, probably we should remove this**
- [G2] Allows taxi drivers to log in the system:
  - The system must be able to check if the password provided is correct.
  - The system must only let the taxi drivers log in if the provided password is correct.
- [G3] Allows taxi drivers to precise to the system if they are available or not:
  - The system must put the taxi driver in the appropriate queue when the taxi user becomes available.
  - The system must remove the taxi driver from the appropriate queue if he becomes unavailable.
- Taxi drivers should receive a notification for incoming request. **is it a goal ?**
- Taxi drivers should receive a notification if they have to take care of another user (during a shared ride). **is it a goal ?**
- [G4] Allows taxi drivers to accept or decline incoming requests for an immediate ride:
  - The system must ask to the taxi driver if he accepts to perform the ride.
  - The system must replace a taxi driver at the end of the queue if he declines the ride.
  - The system must ask to the next taxi driver if the former one declined the ride.
  - The system must notify the user with the code of the taxi driver who has accepted the ride.

  - The system must notify the user if no taxi driver in the queue accepts the ride.

- [G5] Allows taxi drivers to accept or decline incoming request for a later reservation:
  - The system must ask to the taxi driver if he accepts to perform the reservation.
  - The system must ask to the next taxi driver if the former one declined the reservation.
  - The system must notify the user with the code of the taxi driver who has accepted the reservation.
  - The system must notify the user if no taxi driver in the queue accepts the reservation.
  -

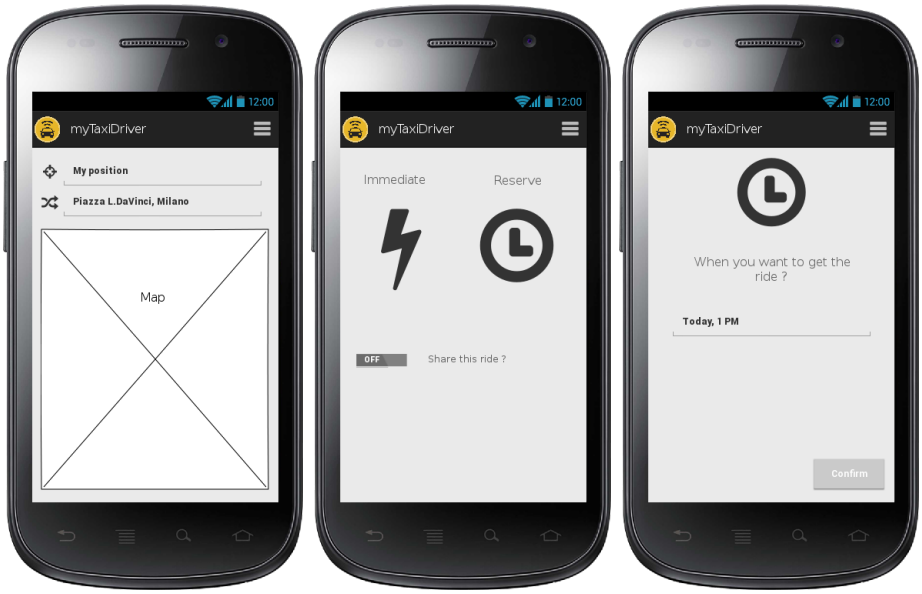
Users:

- [G6] Allows users to request for an immediate taxi ride :
  - The system must be able to check the position of the user.
  - The system must not accept requests of users outside the area of the city.
  - The system must transfer the request to the appropriate taxi driver.
- [G7] Allows users to request for the reservation of a taxi at least two hours in advance.
  - The system must be able to check the origin and the destination of reservation.
  - The system must not accept reservations with an origin outside the area of the city.
  - The system must transfer the reservation to the appropriate taxi driver.
- Users should receive a notification with the code of the taxi that takes care of the user's request. **is it a goal ?**
- Users should be notified if no taxi driver is able to perform the users request. **Is this a goal ?**
- [G8] Allows users to require to share the taxi.
  - The system must be able to find if there are reservations or requests for the same time period and having corresponding journeys.

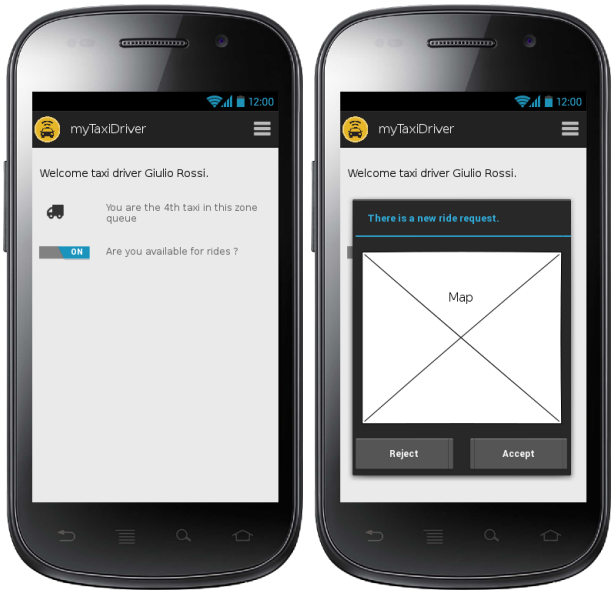
Non-functional requirements

User interface

Client mobile interface



Taxi mobile interface



Documentation

We will draft these documents to well-organize our work in the way to do in a fewer time the best work as possible:

- RASD: Requirement Analysis and Specification Document, to well- understand the given problem and to analyze in a detailed way which our goals are and how to reach them defining requirements and specification.
- DD: Design Document, to define the real structure of our web application and its tiers.
- Testing Document: a report of our testing experience of another myTaxy-Service project.

Architectural consideration

We will use the following technologies:

- Apache with php (with laravel framework) as API server and task service
- Mysql as sql server to store data persistently, it is the same of the old system
- Apache server for static documents
- RESTFull and JSON for API communication over HTTP(S)
- Javascript (with angularJs framework), CSS and HTM to create responsive site that communicate to server using REST API. These files are got via HTTP(S)
- Modern browser with javascript and ajax support
- Java and swing respectively for android and iOS apps, using original SDK
- Internet connection for communication of data

Scenario identifying

Here some possible scenarios of usage of this application

Scenario 1

John wants to go home as soon as possible after a day of work and he wants to do that as soon as possible. So in the morning (so he respects the constrains of two hours before) he reserves a taxi at the same time of the end of his job for a ride that starts from his office and ends at his home.
When he goes out from office he finds the taxi on the street that brings him to his home.

Scenario 2

Some friends live in the same zone and want to go to the airport for a trip together, they want a cheap solution. So they choose the taxi sharing option to go to the airport. The morning of the trip's day all friends request a taxi with sharing option.
Since they are 6 and a taxi can bring only 4 passengers they need at least 2 taxis, so 4 friends are in the same taxi while two others are in other two taxis, each of them filled by other people that have chosen the taxi sharing option and start from the same zone and have to go in the same direction.

Scenario 3

John wants to visit the Duomo tomorrow. He decides to reserve a taxi. Therefore, he opens the MyTaxi **adapt name** website on his laptop and click on the "reserve a taxi button". He is the redirected on a form where he has to fill some information about his ride. After submitting his form, he is redirecting on a waiting page. A few minutes later, John is notified by the application of the confirmation of his reservation and of the code of the taxi taking care of the ride.

Scenario 4

Julia is a taxi driver. She has just finished her last commission and has still plenty of time before the end of the day so she decides to make a new commission. She opens her MyTaxi **adapt Name** application and logs in her personal page. The she sets her availability to "Available". The application notifies her that she is the 3rd taxi on the waiting queue of her area. After waiting a small amount of time Julia receives a request for an immediate ride very close of her location. She immediately accepts it and heads to the request location.

Scenario 5

Scenario 6

Scenario 7

Uml models

Use case diagram

Use case description

In this paragraph some use cases will be described. These use cases can be derived from the scenarios and the use case diagram.

Taxi driver registration : ** Not sure we have to do this use case if the domain properties says that the taxi drivers are officially logged in Name : Taxi driver registration**

** Actors :** Guest
* Entry conditions :** The guest is a new taxi driver not yet registered to the system. **Flow of events :**

- The guest enters the website or mobile application.
- The guest clicks on the sign up icon on the home page of the website or mobile application. **TO ADAPT TO MOCKUPS**
- The guest is shown an input form where he has to fill some personal data (name, phone number, taxi driver license, password, etc.).
- The guest fills the form and clicks on the register button.
- The system redirects the guest on the home page.

Exit conditions : The guest is successfully registered as taxi driver.
* Exceptions :** The guest clicks on the registration button while he has not fulfilled every blank in the form or he has entered non valid data (non valid password, non valid mail, incorrect taxi driver licence, etc.). All of these errors will lead to the reloading of the form with indications concerning the errors.

Taxi driver log in

Name : Taxi driver log in
* Actors : Taxi driver**
* Entry conditions : ** The taxi driver has already successfully registered on the system or was registered on the previous system. **The condition is not necessary if all taxi drivers are automatically registered**
Flow of events :

- The taxi driver arrives on the homepage of the website or mobile application.

- The taxi driver inputs his taxi driver code and his password.
- The taxi driver clicks on the log in button.
- The system redirects the taxi driver to his personal page.

Exit conditions : The driver is successfully redirected to his personal page
* Exceptions : ** The code and password furnished by the taxi driver are not correct. In this case, the system does not redirect the taxi driver to his personal page but notifies him that an error has been made and allows to input his code and password again.

Taxi driver informs of his availability TO ADAPT TO MOCKUPS

Name : Taxi driver informs of his availability
* Actors :** Taxi driver
* Entry conditions : ** The taxi driver must be logged in.
Flow of events :

- The taxi driver clicks on the availability button on his personal page.
- The system redirect the taxi driver to a page with two buttons: "Available" and "Not available".
- The taxi driver clicks on one of the two buttons.

Exit conditions : The systems redirects the taxi driver to his personal page. If the taxi driver selected the available button, he can now see in which waiting queue he is and his position in this queue on his personal page.

** Exceptions :** There are no exceptions for this use case.

Taxi driver responds to a request of immediate ride

Name : Taxi driver responds to request of immediate ride
* Actors :** Taxi driver
* Entry conditions : **

- The taxi driver has to be available.
- The taxi driver must be the first of his queue.
- The system notifies the taxi driver of the request with an alert. **Here or in flow of events ?**

Flow of events :

- The systems shows the taxi driver the information concerning the request and shows him two buttons: "Accept request" and "Decline request". **TO ADAPT TO MOCKUPS**

- The taxi driver clicks on one of the two buttons.
- The system asks confirmation to the taxi driver concerning his choice.
- The taxi driver confirms.

Exit conditions :

- If the taxi driver has accepted the request, the taxi driver is removed from the waiting queue and the use case "notifying the user of incoming taxi" begins.
- If the taxi driver has declined the request, he is moved to the end of the queue and the use case "taxi driver respond to request of immediate ride" starts again.

** Exceptions : ** There are no exceptions.

Taxi driver responds to a request of later reservation

Name : Taxi driver responds to request of later reservation
* Actors :** Taxi driver
* Entry conditions : **

- The taxi driver has to be available.
- Either the taxi driver must be the first of his queue or the taxi drivers before him must have declined the request.
- The system notifies the taxi driver of the request with an alert. **Here or in flow of events ?**

Flow of events :

- The systems shows the taxi driver the information concerning the request and shows him two buttons : "Accept request" and "Decline request". **TO ADAPT TO MOCKUPS**
- The taxi driver clicks on one of the two buttons.
- The system asks confirmation to the taxi driver concerning his choice.
- The taxi driver confirms.

Exit conditions :

- If the taxi driver has accepted the request, the taxi driver is removed from the waiting queue and the use case "notifying the user of incoming taxi" begins.
- If the taxi driver has declined the request, the use case "taxi driver respond to request of immediate ride" starts again with the following taxi in the queue.

** Exceptions : ** There are no exceptions.

User requires a taxi for a ride

Name : User requires a taxi for a ride
* Actors :** User
* Entry conditions : ** The user clicks on the "require immediate ride button"
Flow of events :

- The systems redirects the user to a form where the user can choose if he wants to share the taxi or not. If the user chooses to share the taxi, he has to give some information like destination and number of passengers. **TO ADAPT TO MOCKUPS**
-

Exit conditions : The system forwards the request to the appropriate taxi and the use case "taxi driver respond to a request of immediate ride" begins.
* Exceptions :** The user furnish invalid data in the form (for example a negative or excessive number of passengers (see [1.3] Domain properties)). The request is not forwarded and user is not redirected until he enters valid data.

User requires a taxi for a later reservation

Name : User requires a taxi for a later reservation
* Actors :** User
* Entry conditions : ** The user clicks on the "require later reservation button"
Flow of events :

- The system redirects the user to a form where he has to furnish the following information :
  - Departure place;
  - Departure time;
  - Destination;
  - Number of passengers (only if the user has chosen to allow taxi-sharing).
- The user fills the form.
- The user clicks on the "Confirm" button. **TO ADAPT TO MOCKUPS**

** Exit conditions :** The system redirects the user to a waiting page and forwards the request for a reservation to the appropriate taxi driver. The use case "taxi driver respond to a request of later reservation" begins.
* Exceptions :** The user furnish invalid data in the form (for example a negative or excessive number of passengers or not valid departure time (see [1.3] Domain properties)). The request is not forwarded and user is not redirected until he enters valid data.

Notifying the user of incoming taxi

Name : Notifying the user of incoming taxi
* Actors :** User
* Entry conditions :** The user must have required an immediate ride or a later reservation.
Flow of events :

- The system sends an alert to the user.

Exit conditions : The system redirects the user on a page containing the information about the incoming taxi.
* Exceptions :** No taxi driver has accepted the request of the user. The user is notified and redirected to the home page of the platform.

Class diagram

Sequence diagram

State diagram

Alloy modeling

Model

```
open util/boolean

//sig

one sig TaxiCentral {
hasTaxiDrivers: some TaxiDriver,
hasClients: some Client
}

sig TaxiDriver {
hasTaxi: one Taxi,
available: Bool,
belongsToQueue: lone QueueElement //if 0 the taxi is not in the queue
}

sig Queue {
}

sig QueueElement {
belongsToQueue: one Queue,
position: Int
}
{
position > 0
}

sig Taxi {

}

sig Client {

}

abstract sig Ride {
belongsToTaxiDriver: one TaxiDriver,
startZone: one Zone,
finished: Bool
}

sig NormalRide extends Ride{
```

```
belongsToClient: one Client,
passengers: Int,
endZone: one Zone
}
{
passengers<=4
passengers>=1
}

sig SharingRide extends Ride{
belongsToClients: some  Client,
endZones: Client -> one Zone
}

sig Zone {
hasQueue: one Queue
}


//If #hasRide = 0 the request is not completed yet
abstract sig Request {
belongsToClient: one Client,
startZone: one Zone
}

sig SharingRequest extends Request {
hasRide: lone SharingRide,
endZone: one Zone
}

sig NormalRequest  extends Request {
hasRide: lone NormalRide,
passengers: Int,
endZone: lone Zone
}

//If #hasRide = 0 the request is not created yet
abstract sig Reservation {
time: one Date,
belongsToClient: one Client,
startZone: one Zone,
endZone: one Zone
}

sig SharingReservation  extends Reservation {
hasRequest: lone SharingRequest
```

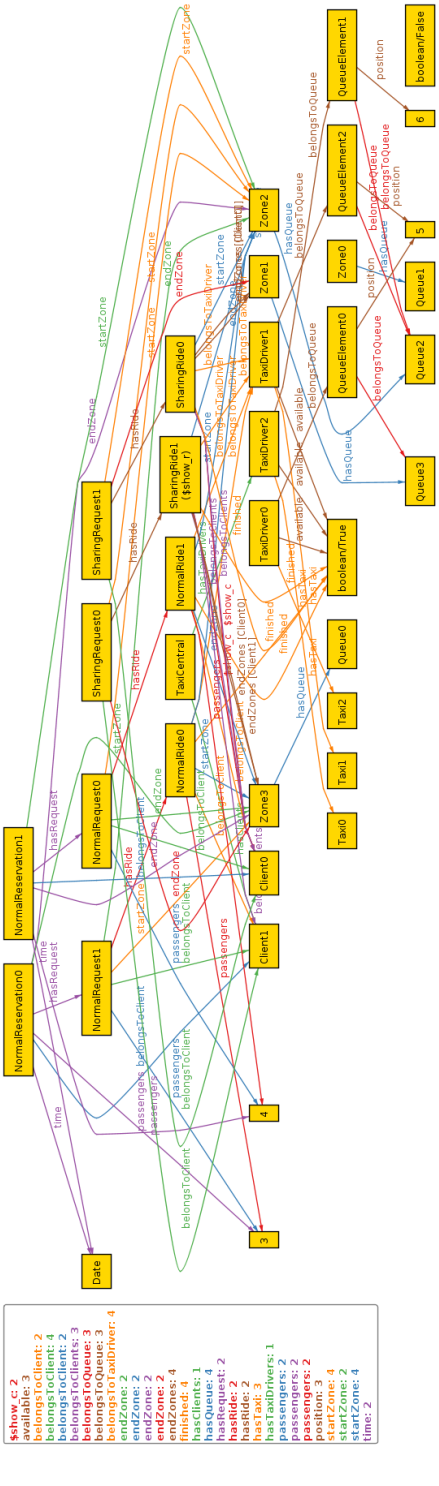
```
}

sig NormalReservation extends Reservation {
hasRequest: lone NormalRequest,
passengers: Int
}

sig Date {}

//fact

//A taxi can stay in the queue only if it is available
// and it doesn't run a ride and viceversa
fact taxiDriverConstrains {
all t: TaxiDriver | (#t.belongsToQueue = 1 <=> t.available.isTrue) and
(t.available.isTrue <=> no r: Ride | r.belongsToTaxiDriver = t and
 r.finished.isFalse) and
(t.available.isFalse <=> one r: Ride | r.belongsToTaxiDriver = t and
 r.finished.isFalse) and
//only one ride at the same time
(no r1, r2: Ride | r1 != r2 and r1.belongsToTaxiDriver = t and
r2.belongsToTaxiDriver = t and r1.finished.isFalse and r2.finished.isFalse)
}

fact TaxiCanBeAssociatedtoOnlyOneTaxiDriver {
no t: Taxi | some t1, t2:TaxiDriver |
t1!=t2 and (t in t1.hasTaxi) and
(t in t2.hasTaxi)
}

fact QueuesCanBeAssociatedtoOnlyOneZone {
no q: Queue | some z1, z2:Zone |
z1!=z2 and (q in z1.hasQueue) and
(q in z2.hasQueue)
}

fact SharingRequestRideDataCorrespondence {
all req: SharingRequest | req.belongsToClient in req.hasRide.belongsToClients and
req.hasRide.startZone = req.startZone and
req.hasRide.endZones[req.belongsToClient] = req.endZone
}

fact NormalRequestRideDataCorrespondence {
all req: NormalRequest | req.belongsToClient = req.hasRide.belongsToClient and
req.hasRide.startZone = req.startZone and
req.hasRide.endZone = req.endZone and
```

```
req.hasRide.passengers = req.passengers
}

fact SharingReservationRequestDataCorrespondence {
all res: SharingReservation | res.belongsToClient = res.hasRequest.belongsToClient and
res.hasRide.startZone = res.startZone and
res.hasRequest.endZone = res.endZone
}

fact NormalReservationRequestDataCorrespondence {
all res: NormalReservation | res.belongsToClient = res.hasRequest.belongsToClient and
res.hasRequest.startZone = res.startZone and
res.hasRequest.endZone = res.endZone and
res.hasRequest.passengers = res.passengers
}

fact NormalRequestCanBeAssociatedtoOnlyOneReservation {
no req: NormalRequest | some res1, res2: NormalReservation |
res1!=res2 and (req = res1.hasRequest) and
(req = res2.hasRequest)
}

fact SharingRequestCanBeAssociatedtoOnlyOneReservation {
no req: SharingRequest | some res1, res2: SharingReservation |
res1!=res2 and (req = res1.hasRequest) and
(req = res2.hasRequest)
}

fact NormalRideCanBeAssociatedtoOnlyOneRequest {
no ride: NormalRide | some req1, req2: NormalRequest |
req1!=req2 and (ride = req1.hasRide) and
(ride = req2.hasRide)
}

fact SharingAllRideMustBeAssociatedToARequest {
all r: SharingRide | one req: SharingRequest | req.hasRide = r
}

fact NormalAllRideMustBeAssociatedToARequest {
all r: NormalRide | one req: NormalRequest | req.hasRide = r
}

fact QueueElementCanBeAssociatedtoOnlyOneTaxiDriver{
no q: QueueElement | some t1, t2: TaxiDriver |
t1!=t2 and (q = t1.belongsToQueue) and
(q = t2.belongsToQueue)
```

```
}

fact AllQueueElementMustBeAssociatedToATaxiDriver {
all q: QueueElement | one t: TaxiDriver | t.belongsToQueue = q
}

fact SharingRideHasNoMoreThanFourClients {
all r: SharingRide | #r.belongsToClients <=4
//this implies no more than 4 requests
}

fact SharingStartDifferentFromEnd {
no r: SharingRide | all c: r.belongsToClients | r.startZone = r.endZones[c]
}

fact NormalStartDifferentFromEnd {
no r: NormalRide | r.startZone = r.endZone
}

fact OnlyOneRequestPerClientAtSameTime {
all c: Client |
(no r1, r2: SharingRequest | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and #r1.hasRide = #r2.hasRide and #r2.hasRide = 1) and
(no r1, r2: NormalRequest | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and #r1.hasRide = #r2.hasRide and #r2.hasRide = 1)
}

fact OnlyOneRidePerClientAtSameTime {
all c: Client |
(no r1, r2: SharingRide | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and r1.finished.isFalse and r2.finished.isFalse) and
(no r1, r2: NormalRide | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and r1.finished.isFalse and r2.finished.isFalse)
}


//pred

pred show(){
//at least one sharing ride with more than one clients
some r:SharingRide | #r.belongsToClients >1
#NormalRide>1
#TaxiDriver>1
#NormalReservation>=1
#SharingRequest>=1
}
```

```
//run

run show for 4
```

World generated



Used tools

The tools we used to create this RASD document are:

- DIA: for uml models
- Pencil: for mockup
- Gedit and ReText: to write MarkDown with spell check
- Pandoc: to create pdf
- Alloy Analizer 4.2: to prove the consistency of our model.