Requirements Analysis and Specifications Document

Figure 1: Politecnico di Milano

Version 1.1

- Claudio Cardinale (mat. 849760)
- Gilles Dejaegere (mat. 853950)
- 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 drivers
- clients

The system allows clients to reserve a 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 drivers to accept or reject a ride request and to communicate automatically their position (so the zone).
The clients are not registered since the company wants a quick system so if there is a registration a lot of clients won't use the app. So the clients must insert their name and phone number each time (this is faster than creating an account and logging each time).

The system includes extra services and functionalities such as taxi sharing.

The main purpose 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 clients.

Actual system

Until now the taxi company has a system where the clients have to call a call center communicating their 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.

When the taxi driver joins into the company he receives the login data that he cannot change.

This system stores taxi information into a Mysql database.

Goals

Taxi drivers:

- [G1] Allows taxi drivers to log in the system.

- [G2] Allows taxi drivers to precise to the system if they are available or not.
- [G3] Taxi drivers should receive a push notification for incoming request.
- [G4] Taxi drivers should receive a push notification if they have to take care of another client (during a shared ride).
- [G5] Allows taxi drivers to accept or decline incoming requests for an immediate ride
- [G6] Allows taxi drivers to accept or decline incoming request for a later reservation.
- [G7] Allows taxi to know the fee for each ride before it starts via the request notification (but after he has accepted).

Clients:

- [G8] Allows clients to request for an immediate taxi ride.
- [G9] Allows clients to request for the reservation of a taxi at least two hours in advance.
- [G10] Clients should receive an SMS notification with the ETA and code of the taxi that takes care of the client's request.
- [G11] Allows clients to require to share the taxi.
- [G12] Allows clients to identify themselves via phone number (and name) not via login, they are not registered into the system.
- [G13] Allows clients to specify the number of passengers.
- [G14] Allows clients to know the fee for the ride via SMS notification of taxi assigned see [G10]

Domain properties

We suppose that these properties hold in the analyzed world :

- Actual drivers are already registered in the previous system
- A taxi driver accepting a ride of reservation will actually take care of the request.
- A client requiring a taxi will actually take it.
- All the GPS always give the right position.
- The GPS of the taxi drivers cannot be switched off.
- Taxi drivers answer all types of demands in less than 5 minutes.
- The client 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.
- The client makes a reservation two hours before the ride
- 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 starting point with max 30 minutes of delay.
- The old system works properly without problems.
- If a queue is empty, a taxi joins in this queue in max 15 minutes.
- Taxi codes and phone numbers are unique.
- Each taxi belongs to one taxi driver.
- Each taxi driver has one taxi.
- The number of passengers is positive

Glossary

- Client: he is a client of the service. Each time he performs a request/reservation, he should insert the following information:
  - Name
  - Phone number
  - Position, it can be taken automatically from GPS (either via APP or Web browser)
  - Number of passengers
  - Sharing mode
  - Time (only for reservation)
- Taxi driver: he is a taxi driver registered in the taxi company which grants to taxi driver the access to this information system
- 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.
- Requests queue: when more than one request is in the same zone, there is a FIFO queue. So the first taxi available serves the first request.
- Request: it is the request of a taxi, it can be shared or not. When a user requests a taxi, the request is created and inserted in request queue. See client to field details.
- Merged request: it is a request associated to more than one user. When more that one request in the same queue respect the taxi sharing condition, a merge request is created (merging that requests). The merging is performed when a shared request is at the head of the queue.
- 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, client 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 they know each other) of the same starting zone go to the same direction, even if the end is not the same, to use the same taxi and to have a discount.

A sharing ride is identified by clients that use it and for each client the starting and ending point.

- Reservation: it is the ability to reserve a taxi until two hours before the time of the ride, so when a reservation is done the system makes a taxi request 10 minutes before the ride. The reservation is identified by starting point, ending point, client and time. It can be sharing or not. See client to field details.
- Taxi notification: it is the notification the system sends (automatically or after a client request) to taxi to specify a ride, specifying starting point, client and other elements if they are available.
- Client request: it is the request for a taxi drive as soon as possible, it contains the client data and the starting 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.
  - A zone is specified by a list of bounds.
  - It has more than 2 positions that compose its bounds.
  - Bounds must be composed by positions and not by none of its subclasses.
  - Bounds must be a set (it must not contain duplicates).
- 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 passengers.
- System: it is the new system we will create with the database of the old system.
- Matching itineraries: two itineraries (A and B) are matching if one of the two following conditions are fulfilled:
  1. B is included in A: the starting point and the ending point of the itinerary B are both close to the itinerary A and the starting point of B is closer to the starting point of A than the ending point of B.
  2. The beginning of B is the end of A: the starting point of B is close to the itinerary A and the ending point of A is close to the itinerary B.
  3. A is included in B: see condition 1.
  4. The beginning of A is the end of B: see condition 2.
- ETA: estimated time available; it is the time the taxi needs to arrive to client starting position.
- SMS: short message service; it is a notification sent to a mobile phone, we need an SMS gateway to use it.
- SMS gateway: it is a service which allows to send SMS via standard API.
- API: application programming interface; it is a common way to communicate with another system.

- Push notification: it is a notification sent to a smartphone using the mobile application, so it must be installed.
- Push service: it is a service that allows to send push notifications with own API
- Path: it's a structure that contains at least 2 positions
- Sharing discount percentage: discount percentage applied only if the sharing option is enabled and there is more than one request in the merged request

Text assumptions

- There is an old system as described above.
- We should develop a mobile application able to use its own position through a GPS or asking it to the client.
- Shared requests are taken into account until they don't get accepted by any driver.
- The clients are not registered in the system, because we only need few information (see Glossary). We made this choice because in real systems, clients often have no time to register, so we think that without user registration the system is more user friendly and quick to use.
- The system does not need client registration since it works like the old system (where every client must say identification data via call). The real applications of many cities run in this way.
- The registration/deletion by the company of a taxi driver is done on the old system, so we do not have to do this part. We have only to interface with the old database.
- We need information only about taxi driver, not about taxi vehicle. So we store information only about taxi driver.
- All taxi drivers of the city are regulated and use this system
- The client cannot cancel a request
- We assume that we need a request queue
- We assume that, since the clients are not registered, they have to check each time the sharing option (if they want to use it)
- We assume that we have to communicate fees even if the clients don't use the sharing option
- We assume that we have a fee for each passenger, the fee depends of the number of passengers
- We assume that we have a fixed sharing discount percentage
- We assume that we have a fixed price per kilometer per passenger

Constrains

Regulatory policies

The system must require to client/taxi driver the permission to get his position and he has to manage sensible data (position, phone number) respecting the privacy law. Furthermore the systems mustn't use notifications to send SPAM respecting the privacy law.

Hardware limitations

- Mobile app
  - taxi driver:
    - 3G connection
    - GPS
    - Space for app package
  - client:
    - 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. We know the structure of the old db and we perform a polling to find changes.
- Interface with SMS gateway provider via standard SMS rest APIs, to send notifications to clients.
- Interface with the push service(s) via own APIs to send push notifications to taxi drivers. Often we need an interface for each platform (android, iOS, and so on).

Parallel operation

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

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 a version for taxi driver and a version for clients.

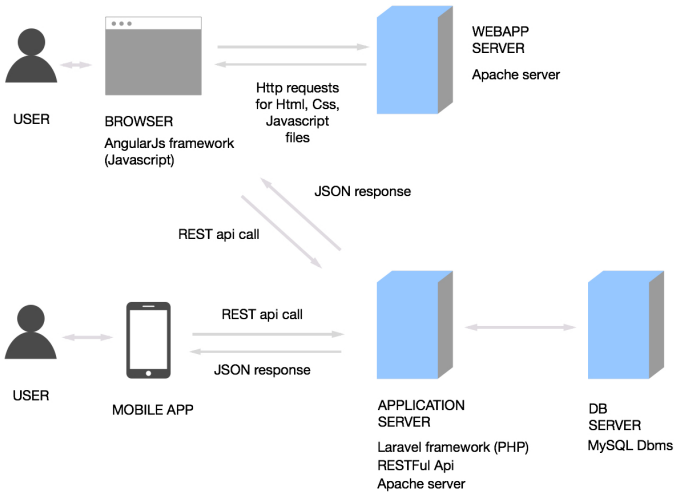


Figure 2: Architecture

Identifying stakeholders

We have only one main stakeholder: the government of the city who wants to improve the current taxi service in terms of usability, efficiency and cost.
However we can adapt this system to other cities (changing the interface with the old system)

Reference documents

- Specification Document: Assignments 1 and 2 (RASD and DD).pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- Examples documents:
  - MeteoCal_RASD_example2.pdf

– RASD Example SWIMv2.pdf
– RASD_meteocal-example1.pdf

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.
- Client: he doesn't need to register himself to the system, since he uses the system only to call a taxi (so he only has to insert name, phone number 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 clients concerned.

Taxi drivers:

- [G1] 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.
  - The system must interact with the old system to do that.
- [G2] Allows taxi drivers to precise to the system if they are available or not:
  - The system must be able to detect the taxi's location according to the taxi's GPS.
  - The system must be able to determine the appropriate queue for the taxi driver according to its position.
  - The system must put the taxi driver in the appropriate queue when the taxi client becomes available.
  - The system must remove the taxi driver from the appropriate queue if he becomes unavailable.
  - The system must communicate to the driver its position in the waiting queue.
- [G3] Taxi drivers should receive a notification for incoming request:
  - The system must be able to forward an incoming request to the appropriate taxi driver.
  - The system must be able to alert a taxi driver when a request is incoming.
- ~~[G4] Taxi drivers should receive a notification if they have to take care of another client (during a shared ride):~~
  - ~~The system must be able to match together matching request with the "shared ride" option activated.~~
  - ~~The system must be able to inform the taxi when a ride is shared.~~

  - ~~The system must be able to inform the taxi drivers of the different stops he has to do during his ride in order to take or drop the concerned passengers.~~

- [G5] 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 client with the code of the taxi driver who has accepted the ride.
  - The system must notify the client if no taxi driver in the queue accepts the ride.
- [G6] Allows taxi drivers to accept or decline incoming request for a later reservation:
  - The system must wait until ten minutes before the starting time of the reservation and manage it like an immediate ride.
- [G7] Allows taxi to know the fee for each ride before it starts via the request notification
  - The system must send a notification with the fees calculated for the ride, they are calculated using fixed fees for distance.
  - The system must send this notification after confirmation by taxi driver.
  - The system must estimate the distance using the distance between two zone's centers.
  - The system must use a fixed fee for each passenger, so the total fee is given by passenger fee multiplied for passengers number reducing the price of a sharing discount percentage.
  - The system must give the total fee for each client, if sharing option is enabled and used (there is more than one request in merged request).
  - The system must calculate the fee during the ride creation.

Clients:

- [G8] Allows clients to request for an immediate taxi ride:
  - The system must be able to check the position of the client.
  - The system must not accept requests of clients outside the area of the city.
  - The system must transfer the request to the appropriate taxi driver.

  - The system must be able to determine the zone where the client is located according to the client's GPS position.

- [G9] Allows clients 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.

- [G10] Clients should receive a notification with the code of the taxi that takes care of the client's request:
  - The system must be able to send an sms to the client with the code of the incoming taxi.

- [G11] Allows clients to require to share the taxi:
  - The system must be able to find if there are shared requests for the same time period and having matching start zone and direction.
  - The system must be able to merge together the requests found above.

- [G12] Allows clients to identify themselves via phone number (and name) not via login, they are not registered into the system:
  - The system must allow the clients to furnish their personal information to the system before making a request.

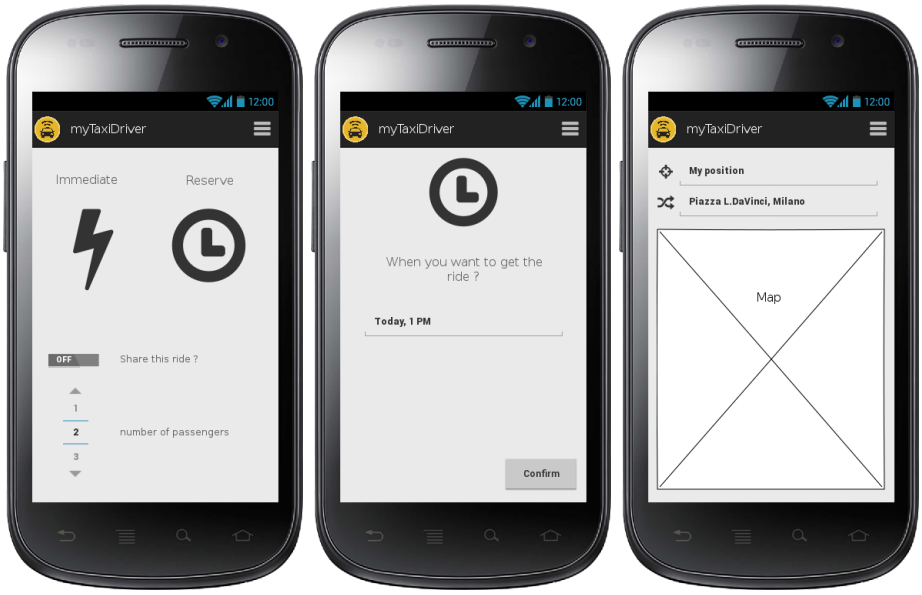
- [G13] Allows clients to specify the number of passengers:
  - The system must allow the client to specify the number of passengers during the request or reservation of the ride.

- [G14] Allows clients to know the fee for the ride via SMS notification of taxi assigned see [G10]
  - The system must insert the fee for the request in the SMS notification
  - The system must use fee calculated as specified in [G7].

Non-functional requirements

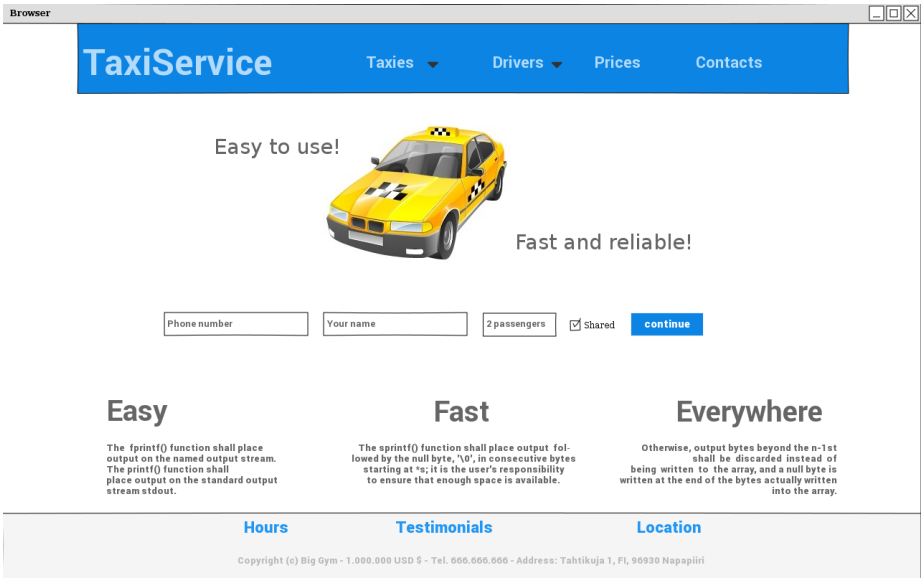
Client interface

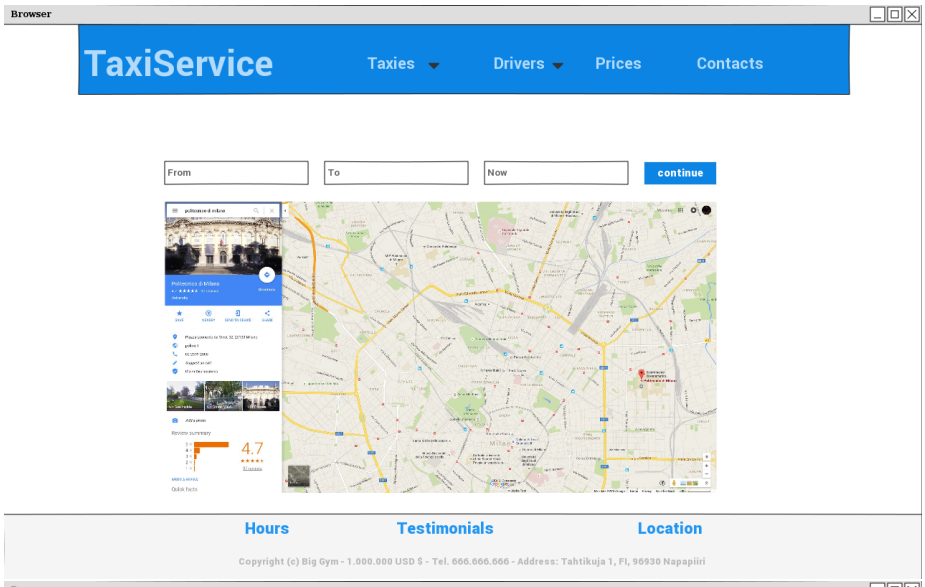
mobile

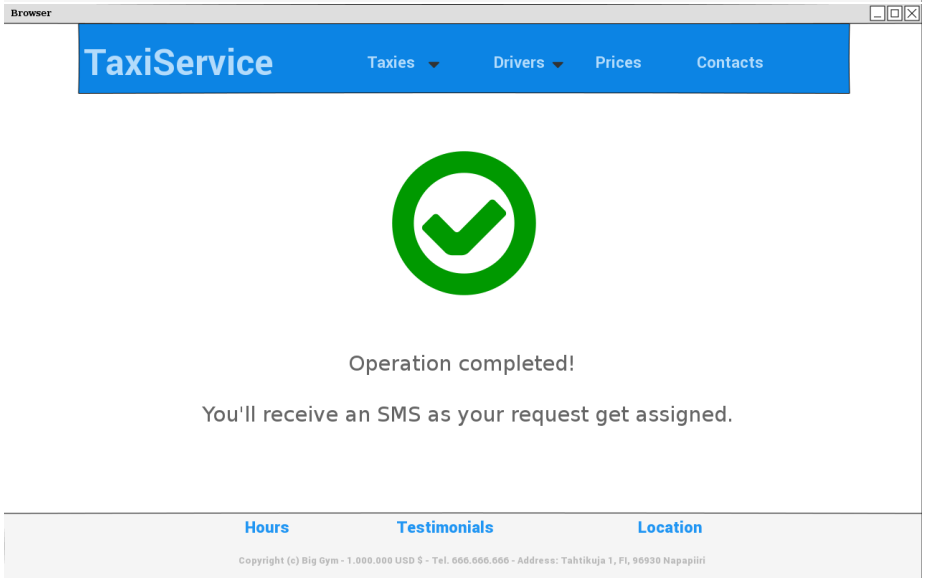


In android the phone number is taken by the app (Permissions.READ_PHONE_STATE)

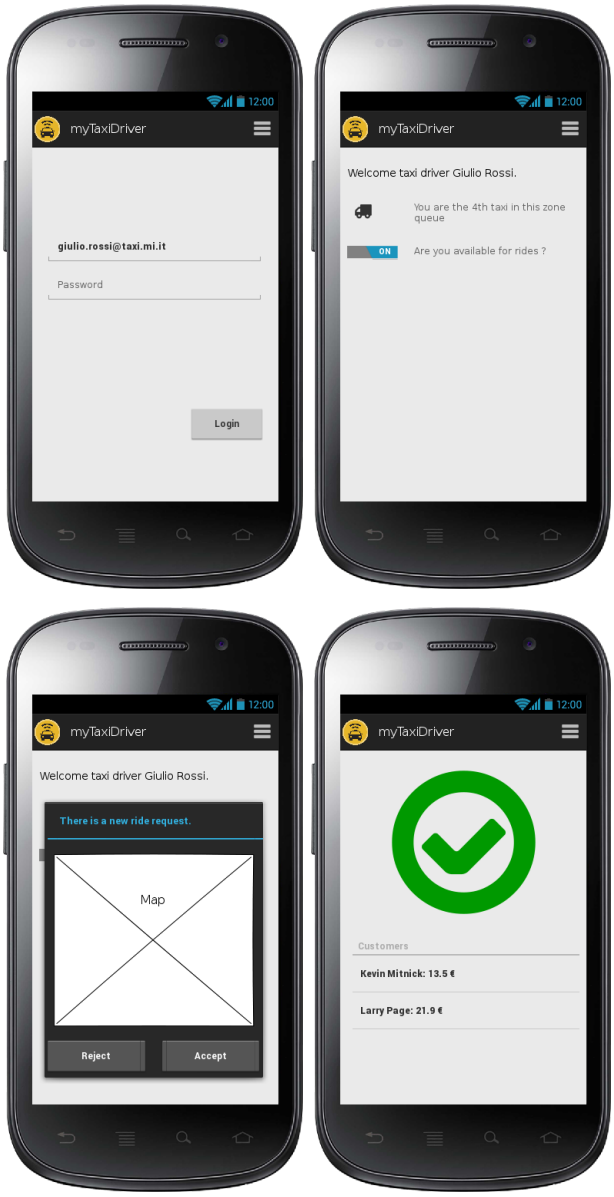
desktop




Browser
TaxiService
Taxies
Drivers
Prices
Contacts
From
To
Now
continue
Hours
Testimonials
Location
Copyright (c) Big Gym - 1.000.000 USD $ - Tel. 666.666.666 - Address: Tahtikuja 1, FI, 96930 Napapiiri


Browser
TaxiService
Taxies
Drivers
Prices
Contacts
Operation completed!
You'll receive an SMS as your request get assigned.
Hours
Testimonials
Location
Copyright (c) Big Gym - 1.000.000 USD $ - Tel. 666.666.666 - Address: Tahtikuja 1, FI, 96930 Napapiiri

Taxi driver 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. 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 swift respectively for android and iOS apps, using original SDK
- Internet connection for communication of data
- External rest APIs to send SMS

Scenario identifying

Here some possible scenarios of usage of this application.

Scenario 1

Tomorrow, John wants to come back home as soon as possible after his day of work. The day after, when John arrives at his office, and before beginning to work, John enters the MyTaxiDriver website (he is aware of the two hours delay necessary to perform a reservation). He reserves a taxi for the 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. Being the price of the flight very expensive, they want to reach the airport without having to spend too much money. Therefore they choose to use the taxi sharing option. The morning of the trip's day all friends request a taxi with sharing option.
Since they are 6 and a normal taxi can host only four passengers, they will need at least two taxis. Four friends will go in the same taxi while two others will go 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.

When they come back from their trip they want to go to a pub. Since they are together and go to the same location, they can order a unique ride indicating 6 as the passengers number.

At the end of the night the want to come back to their home, but since they are in different places (even if in the same zone) each of them have to do their own request (checking sharing option if they want).

Scenario 3

John wants to visit the Duomo tomorrow. He decides to reserve a taxi. Therefore, he opens the MyTaxiDriver website on his laptop and clicks on the "reserve a taxi button". He is redirected to a form where he has to fill some information about his ride. After submitting his form, he receives a confirmation message. His request has been taken into account, but this does not mean that a taxi driver has accepted to perform the ride. A few minutes later, John is notified by sms of the confirmation that his reservation will be performed 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 MyTaxiDriver application and logs in her personal page. Then 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 to her location. She immediately accepts it and heads to the request location.

Scenario 5

Bob wants to go and see the football match at the stadium tonight. He decides to reserve a taxi to bring him to the stadium and to bring him back home after the game. To reduce the cost of the rides he decide to try to share a taxi. To do so, he enters the MyTaxiDriver application on his smartphone, goes to the "Client home page", activates the slide button to enable the "sharing taxi" option and then clicks on the "Reserve" button. Then he has to fill a form to indicate all the information about the reservation. After doing so, he has to repeat the whole operation to reserve the return trip.

Scenario 6

Mark is a taxi driver. At the beggining of his workday, Marks opens his MyTaxiDriver application and logs in. Then he puts his status on "available" by operating the slide button. The application tells him that he is the 10th driver in his queue. That is far too much for Mark. He decides to go to another area to check if the queue is smaller. To do so, he first changes his status to "Not available", then rides toward the desired area and finally sets his status to "Available" again. He is now the 4th of his waiting queue.

Scenario 7

Bob tells his cousin Alice that he is going to the stadium tonight. Alice is also interested in football so she decides to go with Bob to see the match. However, Alice doesn't know how to go to the stadium. Bob tells her about the MyTaxiDriver application and convinces Alice to give it a try. Alice tries to reserve a trip to the stadium, but she does not respect the "reserve two hours before the ride" condition and the reservation fails. Then she tries to request for an immediate ride but no taxi driver accepts Alice's request. Alice will have to stay at home tonight.

Uml models

Use case diagram

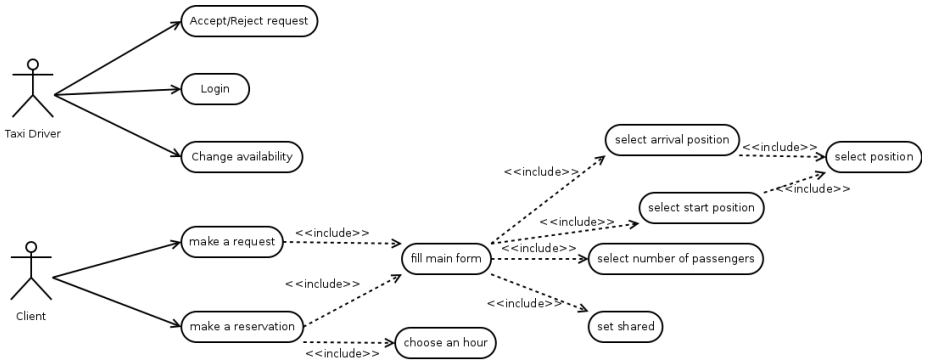


Figure 3: use case diagram

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 logs in

Name : Taxi driver logs in
Actors : Taxi driver
Entry conditions : There are no entry conditions.
Flow of events :

- The taxi driver arrives at the homeActivity of the mobile application.
- The taxi driver inputs his taxi driver code (his ID) and his password.
- The taxi driver clicks on the log in button.
- The system redirects the taxi driver to his personal activity.

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 activity but notifies him that an error has been made and allows to input his code and password again.

Taxi driver informs of his availability

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 activates the slide button on his personal activity.

Exit conditions : The system actualises the personal activity of the driver with the relevant information. If the taxi driver has selected the available button, he can now see in which waiting queue he is and his position.

Exceptions : There are no exceptions for this use case.

Taxi driver responds to a request

Name : Taxi driver responds to a request
Actors : Taxi driver
Entry conditions :

- The taxi driver has to be available.
- The taxi driver must be the first of his queue.

Flow of events :

- The system notifies the taxi driver of the request with an alert.
- The system shows the taxi driver the information concerning the request and shows him two buttons: “Accept” and “Reject”.
- 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 client 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 responds to request of immediate ride” starts again.

Exceptions : There are no exceptions.

Client requires a taxi for a ride

Name : Client requires a taxi for a ride
Actors : Client
Entry conditions : The client has to be on the "Client Homepage".
Flow of events :

- The client sets the "Share this ride" slide button to the desired value if he is using the mobile application, or sets the "Share this ride" box to the desired value if he is using the website.
- The client sets the "number of passengers" to the desired value.
- The client clicks on the "require immediate ride button".
- The system redirects the client to a form where the client has to give some information like the starting location of the ride.

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

Client requires a taxi for a later reservation

Name : Client requires a taxi for a later reservation
Actors : Client
Entry conditions : The client has to be on the "Client Homepage".
Flow of events :

- The client sets the "Share this ride" slide button or box to the desired value.
- The client clicks on the "require later reservation button".
- The client indicates the correct number of passengers.
- The system redirects the client to a form where he has to furnish the following information:
  - Departure place;
  - Departure time;
  - Destination;

- The client fills the form.
- The client clicks on the "Confirm" button.

Exit conditions : The system redirects the client to a waiting page. Ten minutes before the reservation time, the system forwards the request to the appropriate taxi driver. The use case "taxi driver responds to a request" begins.
Exceptions: The client furnishes 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 client is not redirected until he enters valid data.

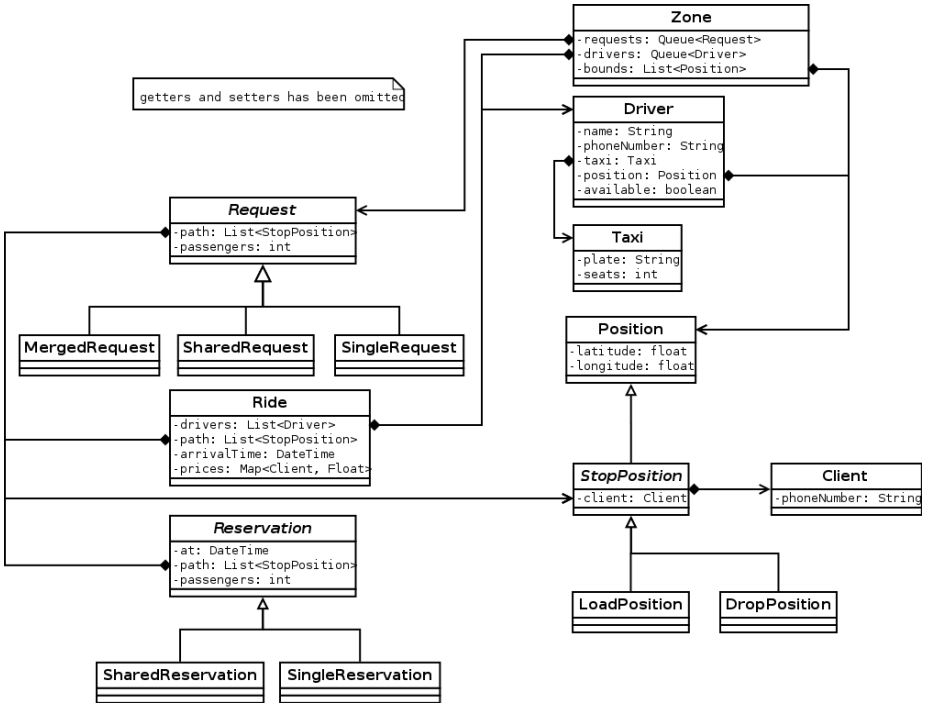
Notifying the client of incoming taxi

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

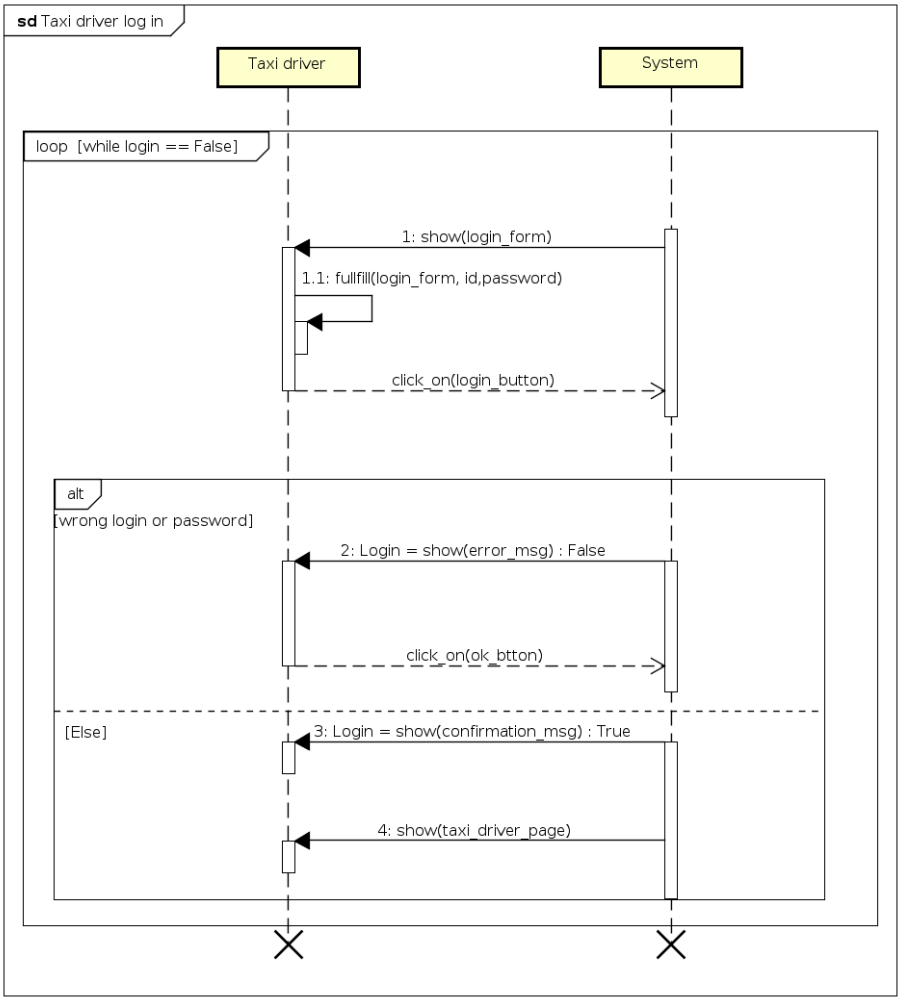
- The system sends an sms to the client containing the code of the taxi and the arrival time.

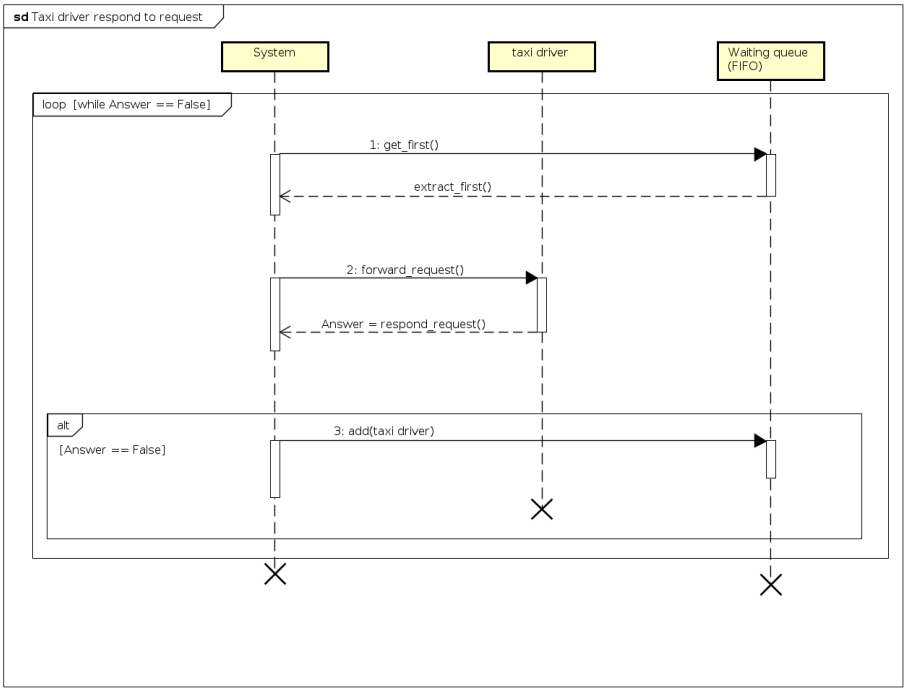
Exit conditions : The system redirects the client to a page containing the information about the incoming taxi.
Exceptions : No taxi driver has accepted the request of the client. The client is notified and redirected to the home page of the platform.

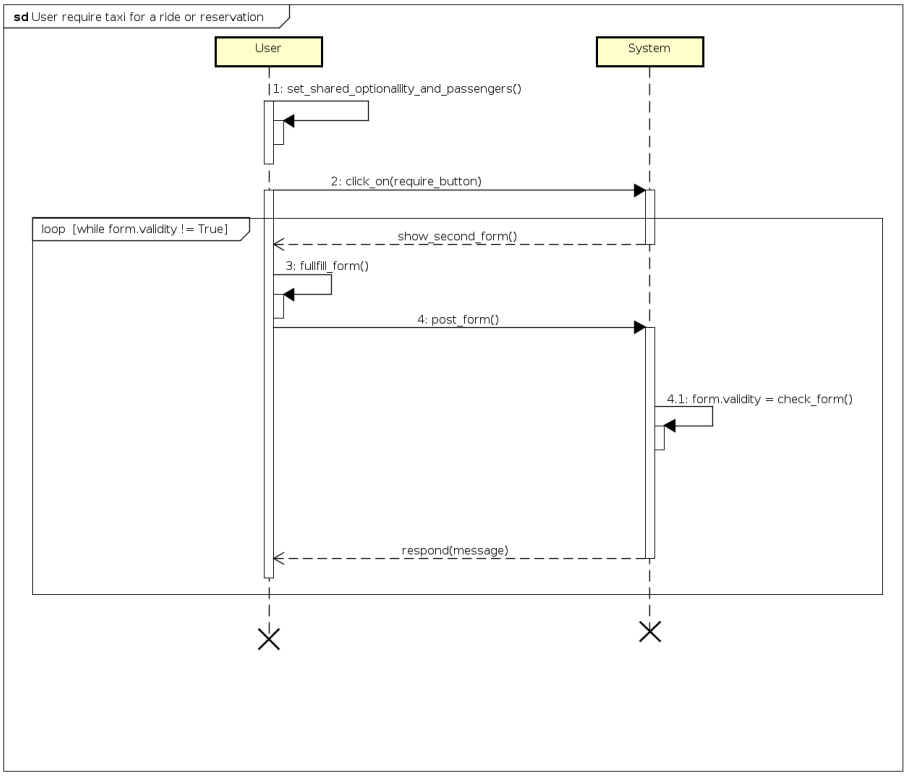
Class diagram



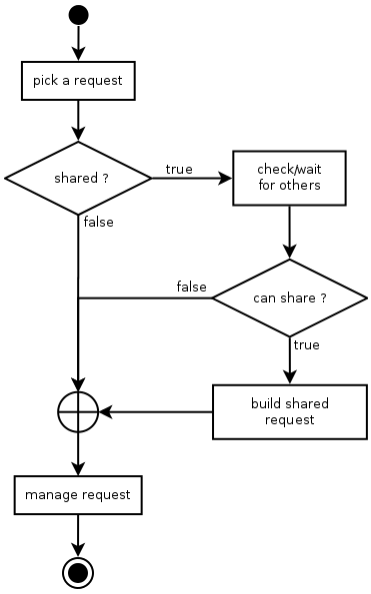
Sequence diagrams



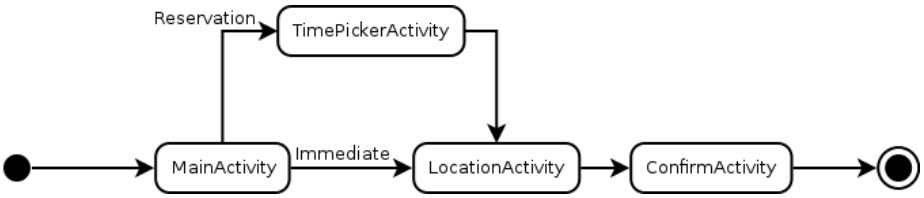

sd Taxi driver respond to request
System
taxi driver
Waiting queue (FIFO)
loop [while Answer == False]
1: get_first()
extract_first()
2: forward_request()
Answer = respond_request()
alt
[Answer == False]
3: add(taxi driver)


sd User require taxi for a ride or reservation
User
System
1: set_shared_optionality_and_passengers()
2: click_on(require_button)
loop [while form.validity != True]
show_second_form()
3: fullfill_form()
4: post_form()
4.1: form.validity = check_form()
respond(message)

Activity diagrams



State diagrams



Alloy modeling

Model

```
open util/boolean
sig Taxi {
code: Int,
seats: Int
} {
code > 0
seats > 0
seats <= 4
}

fact codeAreUnique {
all t1,t2: Taxi | (t1 != t2) => t1.code != t2.code
}


sig Driver {
taxi: one Taxi,
available: one Bool
}

fact allTaxiAreOwned {
Driver.taxi = Taxi
}

fact taxiHaveOnlyOneDriver {
all d1, d2:Driver | d1 != d2 => d1.taxi != d2.taxi
}

sig PhoneNumber { }

sig Client {
phoneNumber: PhoneNumber
}

fact phoneNumberAreUnique {
all c1,c2: Client | c1 != c2 => c1.phoneNumber != c2.phoneNumber
}


sig Position {
latitude: Int, // should be float
```

```
longitude: Int // should be float
}

abstract sig StopPosition extends Position {
client: one Client
}

sig LoadPosition extends StopPosition {
}

sig DropPosition extends StopPosition {
}

pred StopPosition.sameClient[other: StopPosition] {
this.client = other.client
}

pred StopPosition.complement[other: StopPosition] {
other != this
this.sameClient[other]
(this in DropPosition) <=> (other in LoadPosition)
}


sig Path {
positions: seq StopPosition
} {
#positions >= 2
}

fact pathPositionsAreUnique {
all p: Path | not p.positions.hasDups
}

fact pathClientHasALoadAndADrop {
all p: Path | all i1: p.positions.inds |
one i2: p.positions.inds | i2 != i1 and
let p1 = p.positions[i1] | let p2 = p.positions[i2] |
p1.complement[p2] and not ( one i3: p.positions.inds |
i3 != i2 and i3 != i1 and let p3=p.positions[i3] |
p3.complement[p1] or p3.complement[p2])
}

fact pathStartWithLoad {
all p: Path | p.positions.first in LoadPosition
}
```

```
fact pathEndWithDrop {
all p: Path | p.positions.last in DropPosition
}

assert pathPositionsAreEven {
all p:Path| rem[#p.positions, 2] = 0
}

assert clientGetInAndGetOut {
all p: Path| all i1: p.positions.inds | one i2: p.positions.inds |
i2 != i1 and let p0=p.positions[i1] | let p1=p.positions[i2] |
p0.complement[p1]
}

assert pathSameNumberOfLoadAndDrop {
all p: Path | #(p.positions.elems & DropPosition) = #(p.positions.elems & LoadPosition)
}
```

```
abstract sig Request {
path: Path,
passengers: Int
} {
passengers > 0
}

sig SingleRequest extends Request {
}

sig SharedRequest extends Request {
}

sig MergedRequest extends Request {
}


abstract sig Queue {
s: seq univ
}

sig DriverQueue extends Queue {
} {
```

```
s.elems in Driver
}

sig RequestQueue extends Queue {
} {
s.elems in Request
}

fact enqueuedDriversMustBeAvailable {
all d: DriverQueue.s.elems | d.available.isTrue
}

fact availableDriversMustBeEnqueued {
all d: Driver | d.available.isTrue <=> d in DriverQueue.s.elems
}

fact enqueuedElemntsMustBeUnique {
all q: Queue | not q.s.hasDups
}

fun peek[s: (seq/Int -> univ)]: univ {
s.first
}

fun peek[q: Queue]: univ {
q.s.peek
}

fun peek[q: DriverQueue]: Driver {
q.s.peek & Driver
}

fun peek[q: RequestQueue]: Request {
q.s.peek & Request
}

pred Queue.add[q: Queue, e: univ] {
q.s = this.s.insert[#this.s, e]
}


/**
* Precondition: ~ this.s.isEmpty
*/
pred Queue.get[q: Queue, e: univ] {
e in this.s.elems
```

```
q.s = this.s.delete[0]
not e in q.s.elems
}


assert getInverseOfAdd {
all q0,q1,q2: Queue, e: univ | q0.s.isEmpty and
q0.add[q1, e] and q1.get[q2, e] => q0.s = q2.s
}



sig Zone {
drivers: one DriverQueue,
requests: one RequestQueue,
bounds: seq Position
} {
#bounds > 2
all p: bounds.elems | not p in StopPosition
not bounds.hasDups
}

fact oneQueueForZone {
all z1, z2: Zone | z1 != z2 =>
z1.drivers != z2.drivers and z1.requests != z2.requests
}

fact zoneOwnAllDriverQueues {
Zone.drivers = DriverQueue
}

fact zoneOwnAllRequestQueues {
Zone.requests = RequestQueue
}


sig Ride {
drivers: some Driver,
path: Path,
prices: Client -> Int
}

fact RideContainsPricesForItsClients {
all r: Ride | r.prices.(Int) = r.path.positions.elems.(client)
}
```

```
one sig TaxiCentral {
drivers: some Driver,
clients: some Client,
zones: some Zone
}

fact ownAllDrivers {
TaxiCentral.drivers = Driver
}

fact ownAllClients {
TaxiCentral.clients = Client
}

fact ownAllZones {
TaxiCentral.zones = Zone
}

pred show() {
#TaxiCentral = 1
#Ride = 1
#SharedRequest = 1
#SingleRequest = 1
#MergedRequest = 1
}

run show for 6
check pathPositionsAreEven
check clientGetInAndGetOut
check pathSameNumberOfLoadAndDrop
check getInverseOfAdd
run add for 5
run get for 5
```

Alloy result

The model is consistent:

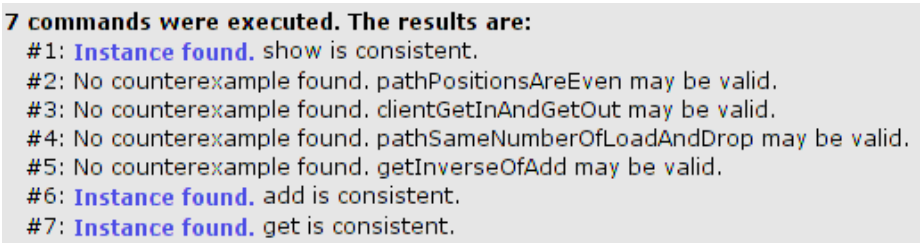
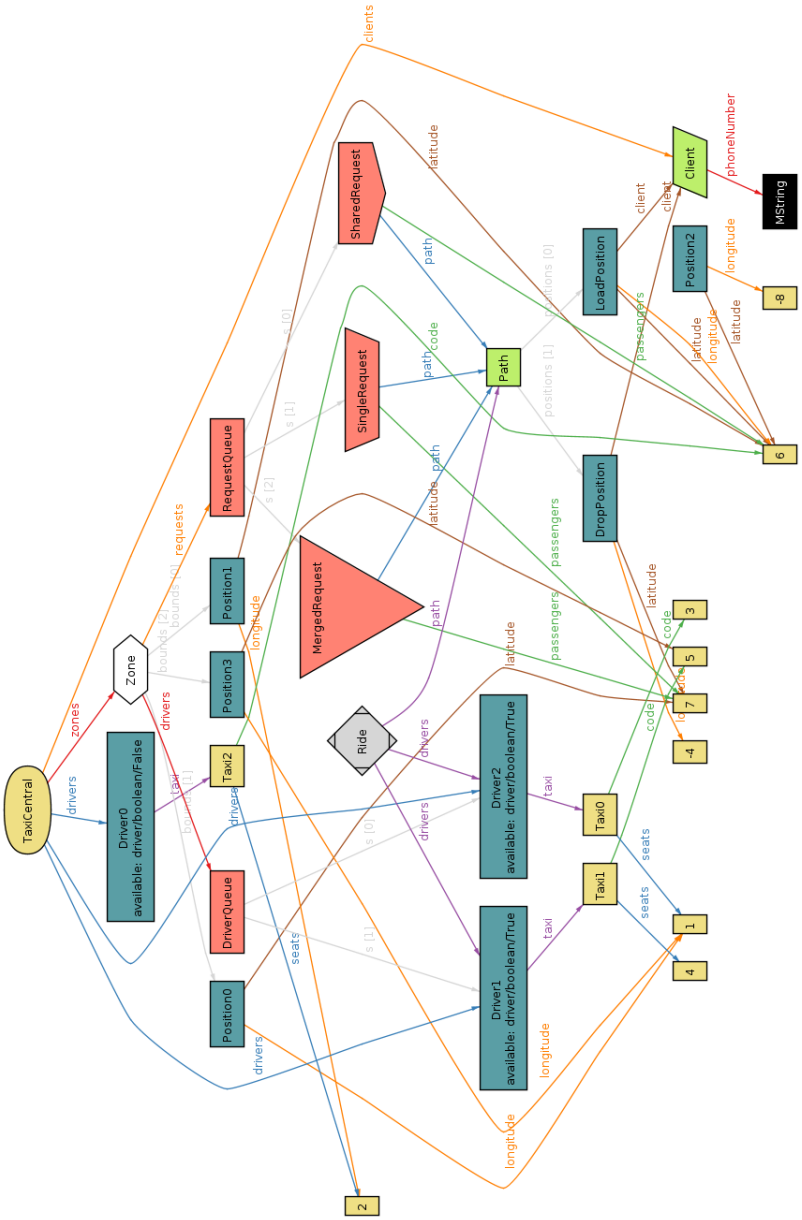


Figure 4: alloy result

World generated



Used tools

The tools we used to create this RASD document are:

- DIA: for uml models
- Github: for version controller
- 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.

Hours of work

Claudio Cardinale

- 21/10/15: 2h
- 23/10/15: 3h
- 24/10/15: 30m
- 25/10/15: 3h
- 26/10/15: 30m
- 27/10/15: 30m
- 29/10/15: 3h -> finding a solution for alloy syntax highlighting in pdf
- 30/10/15: 10h
- 31/10/15: 30m
- 01/11/15: 30m
- 03/11/15: 3h
- 04/11/15: 8h
- 05/11/15: 30m
- 06/11/15: 5h

Gilles Dejaegere

- 23/10/15: 3h
- 24/10/15: 1h30
- 27/10/15: 2h30
- 28/10/15: 2h
- 31/10/15: 2h
- 03/11/15: 3h30 morning + 2h of reunion
- 04/11/15: 2h
- 05/11/15: 2h
- 06/11/15: 4h

Massimo Dragano

- 26/10/15: 4h
- 27/10/15: 2h
- 29/10/15: 1h
- 30/10/15: 2h
- 31/10/15: 1h
- 01/11/15: 2h
- 02/11/15: 3h
- 03/11/15: 4h

- 04/11/15: 5h
- 05/11/15: 8h
- 06/11/15: 11h

Changelog

- v1.1
  - [G4] removed
  - Interface with old system fixed
  - [G7] requirement fixed