Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: “myTaxiService” Requirement Analysis and Specification Document

Massimo Schiavo, Marco Edoardo Cittar

November 6, 2015

Contents

1 Introduction

1.1 Description of the problem

We will design and implement myTaxiService, a new web and mobile application to optimize the taxi service in big cities. It should simplify the access of passengers to the service and guarantee a fair management of taxi queues.

The users will have to register and login in order to use the application, then they can request a taxi and be informed about the code of the incoming taxi and the waiting time.

We suppose the city is divided in zones and every one of them has a queue of available taxis present in the zone, whose position is calculated according to GPS. After a request arrives, the system informs the first taxi in the queue of the zone from which the request came. If the taxi accept the request, the system sends the confirmation to the user. If not, the system will put it at the end of the queue and forward the request to the next available taxi.

To accept ride requests, taxi drivers will have to login through the mobile app like normal users. Then they can set themselves as available and receive ride requests.

A user can also reserve a taxi by inserting origin, destination and time of the ride. The request must be submitted at least two hours before. The system will allocate a taxi and notify the user 10 minutes before the ride.

1.2 Actors

- Guest: the guests are users who are not registered yet. They must sign themselves up into the system in order to use the features available to registered users.
- Registered user: this type of user, after successful login, has access to all the features of the application as a customer. They can request rides, be them simple or detailed, and receive notifications after a ride has been confirmed.
- Driver: they have the functionalities of both customer and worker, so they can set themselves as available and so can be notified when a new ride request arrives, but can also use the application as a registered user when they are not working.

1.3 Goals

myTaxiService should have these features:

- Guests should be able to:
  - [G1] Register themselves into the system
  - [G2] Log themselves into the system

- Users should be able to:
  - [G3] See number of available taxis of the zone they're in
  - [G4] Make a request for a simple ride
  - [G5] Make a request for a detailed ride
- Drivers should be able to:
  - [G6] Set themselves as available
  - [G7] Read and accept ride requests
- The system should:
  - [G8] Notify passengers after the confirmation of a normal request
  - [G9] Notify passengers 10 minutes before the ride reserved through a detailed request
  - [G10] Forward requests to the first taxi in queue
  - [G11] After 30 seconds, forward the request to the second taxi in queue and put the first at the end

1.4 Definitions, acronyms, abbreviations

1.4.1 Definitions

- Simple ride: it's a ride request made by specifying only where the user must be picked up. This kind of ride will allocate a taxi to the request as soon as there's one available.
- Detailed ride: it's a ride request made by specifying origin, destination, date and time. The request must be done at least 2 hours before the ride.

1.4.2 Acronyms

- RASD: Requirement Analysis and Specification Document.
- DD: Design Document.

1.4.3 Abbreviations

- [Gn]: n^{th} goal.
- [Rn]: n^{th} requirement.
- [Dn]: n^{th} domain.

1.5 Reference documents

- Specification document: Assignments 1 and 2 (RASD and DD).pdf
- IEEE Standard For Requirement Specification.pdf

1.6 Document overview

This document is essentially structured in four part:

- Section 1: Introduction, it gives a description of document and some basic information about software.
- Section 2: Overall Description, gives general information about the software product with more focus about constraints and assumptions.
- Section 3: Specific Requirements, this part list requirements, typical scenarios and use cases. To give an easy way to understand all functionality of this software, this section is filled with UML diagrams.
- Section 4: Appendix, this part contains some information about the attached .als file and some described screenshot of software used to generate it.

2 Overall Description

2.1 Product perspective

The application we will project is both a web application and a mobile application. It will not interact with any other existing application or system. It will be user based and we will not provide any internal interface for administration.

2.2 User characteristics

People that will use our application are the ones interested in using the taxi service in the city. They will be able to request a taxi for a ride without any voice call but just with few clicks.

2.3 Assumptions

- When the app or the web page is closed the user automatically logs out.
- If a driver sees a ride request, he always accepts it.
- Users can't cancel ride requests.
- If the system sends a notification the user will always receive it.
- Users can make an unlimited number of daily ride requests.
- In case of empty queue 10 minutes before a detailed ride, the application will always manage to find one within the time of the ride.
- Users insert existing addresses when making requests.
- The system knows if the credentials inserted during login belong to a user or a driver, so it will display the correct personal page afterwards.

3 Specific Requirements

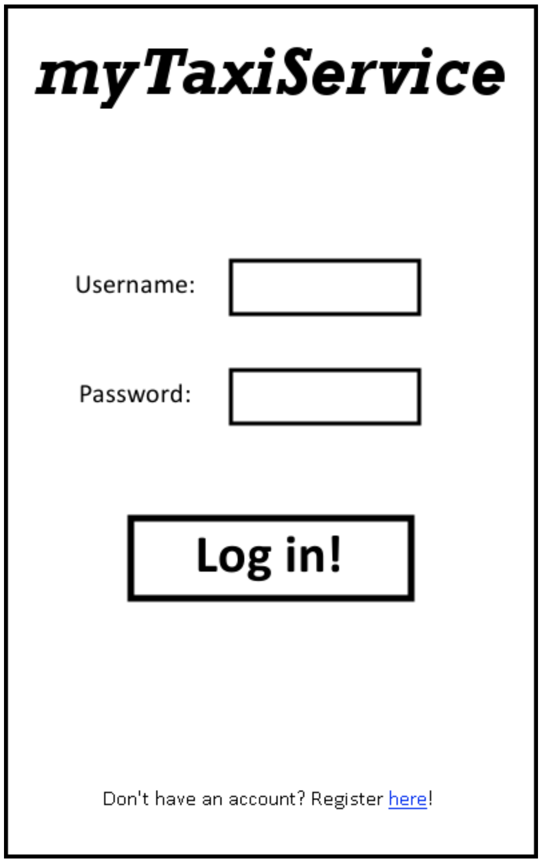
3.1 External interface requirements

3.1.1 User interfaces

Here are presented some drafts which represent how the application should look like. We decided to represent only the mobile application but the web one will be the same.

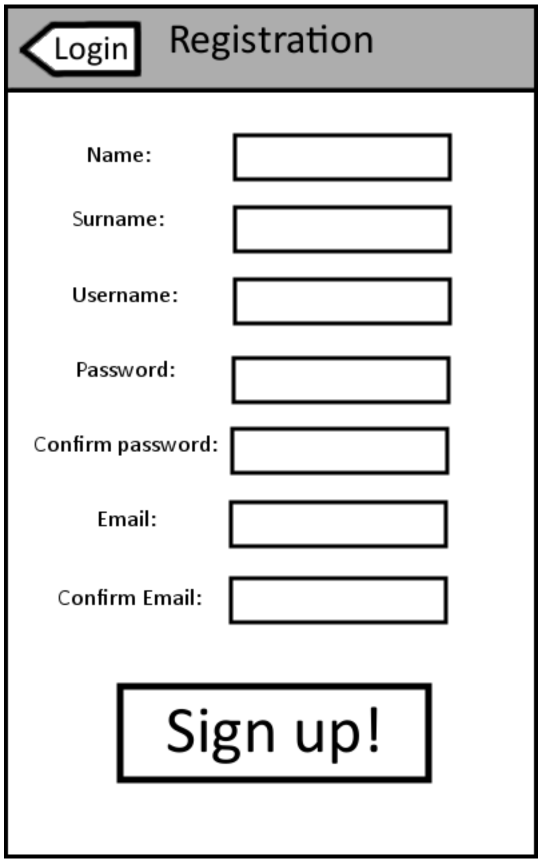
3.1.1.1 Login page

This is the page that shows up when the application is started. It allows the guests to log themselves into the system or register themselves if they don't have an account yet.



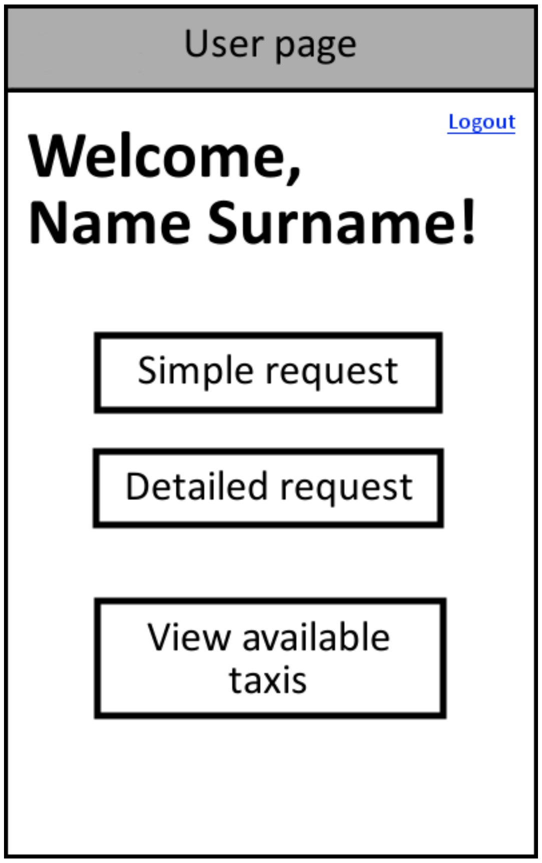
3.1.1.2 Registration page

In this page the guests will input all of their information and register into the system. If there's any error when they click the "Sign up!" button, an error message will appear right above the button, explaining the problem.



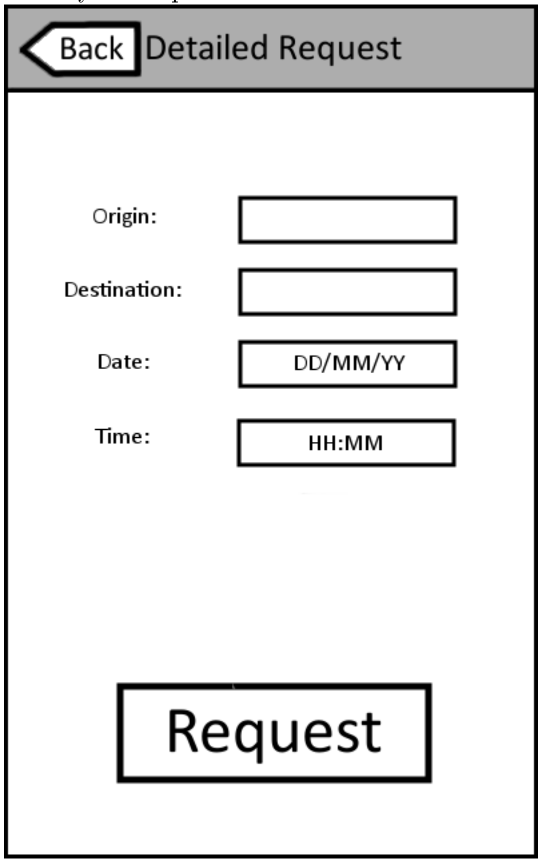
3.1.1.3 User page

After login, this is what users will see. The graphic is minimal but really user friendly, so that users can easily access to all of their functions.



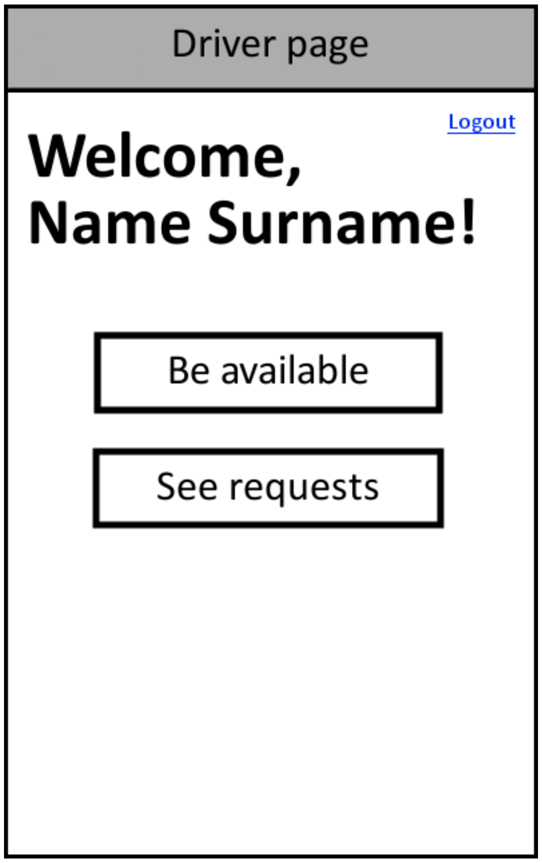
3.1.1.4 Detailed request

By clicking on the "Detailed request" button, users are directed to this page where they can input the information about the ride.



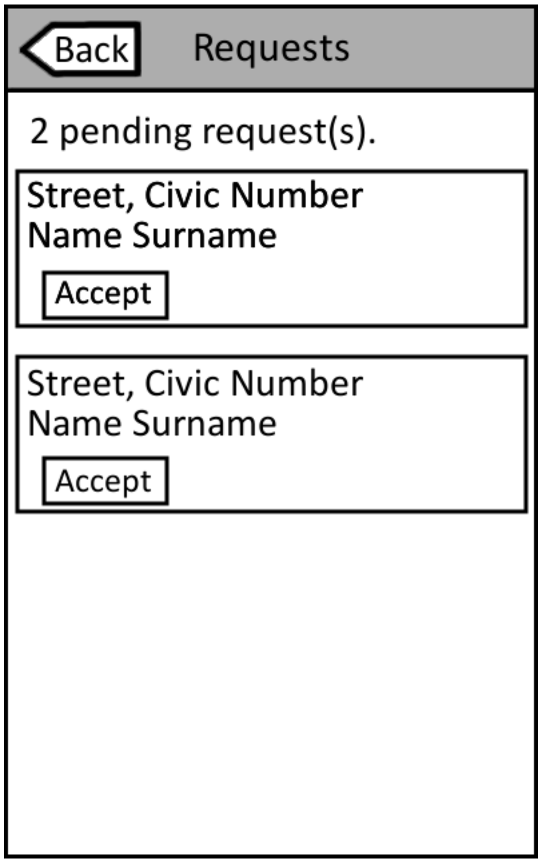
3.1.1.5 Driver page

After login, this is the page drivers will see. It allows them to set themselves as available and start receiving requests and access to the page where the requests are.



3.1.1.6 Requests page

This is the page where drivers can see the requests and accept them.



3.2 Functional Requirements

3.2.1 [G1] Register themselves into the system

- [R1] Guests can only see the login page.
- [R2] Guests can only access to the registration form.
- [D1] All the fields must be completed in a formal correct way.

3.2.2 [G2] Log into the system

- [R1] Guests have to provide valid username a password in order to log themselves into the system.
- [R2] The wrong insertion of one's own credentials will not allow to log himself into the system.

3.2.3 [G3] See number of available taxis of the zone he's in

- [R1] The system has to visualize on the personal page the number of taxi available in the zone.

3.2.4 [G4] Make a request for a simple ride

- [R1] Users have to insert the correct departure's address.
- [D1] The inserted address has to be formed by a street and a civic number.

3.2.5 [G5] Make a request for a detailed ride

- [R1] Users have to insert the correct departure's address.
- [R2] Users have to insert the correct destination's address.
- [R3] Users have to insert feasible date and time for the reservation.
- [D1] The inserted address has to be formed by a street and a civic number.
- [D2] Date must look like this: DD/MM/YY, and time like this: HH:MM.

3.2.6 [G6] Set themselves as available

- [R1] The system has to provide a function through which drivers can inform it of their availability.
- [R2] Once they're available, the system has to insert them into the Taxi Queue of their own zone.

3.2.7 [G7] Read and accept ride requests

- [R1] The system has to provide a function to read and accept request

3.2.8 [G8] Notify passengers after the confirmation of a simple request

- [R1] The system has to notify the user who made a simple request as soon as possible, this means as a taxi is available, informing him about the waiting time.

3.2.9 [G9] Notify passengers 10 minutes before the ride reserved through a detailed request

- [R1] The system has to notify the user who reserved a taxi within ten minutes before the time of the ride.

3.2.10 [G10] Forward requests to the first taxi in queue

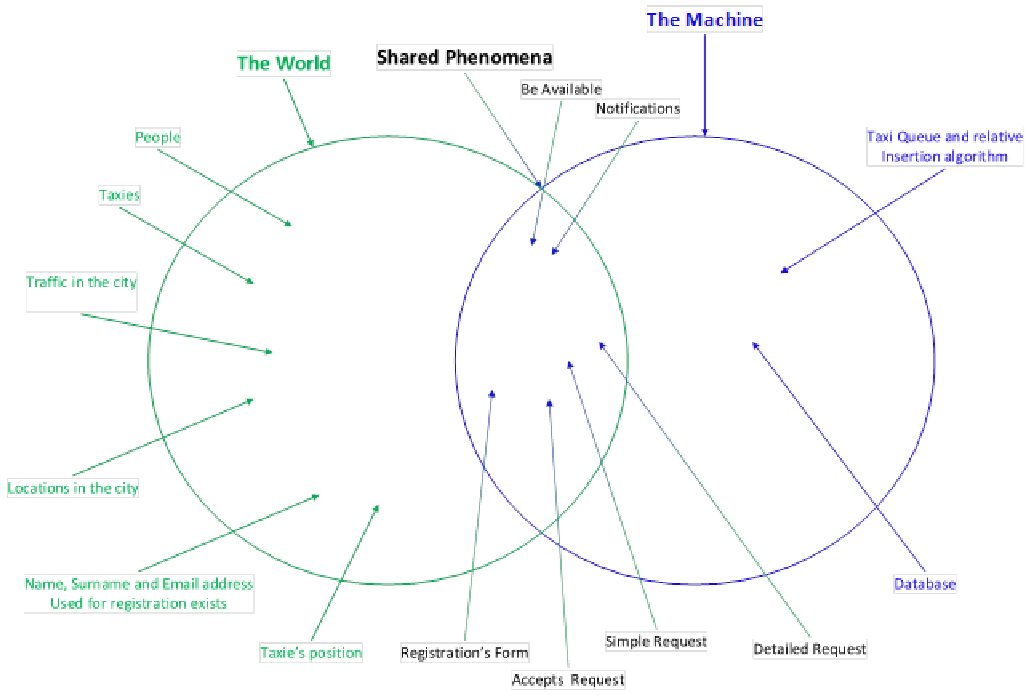
- [R1] The system has to forward requests to the first taxi in queue allowing him to accept the request.
- [R2] After a confirmation by a driver the system has to move the driver at the end of the queue and set him as not available.

3.2.11 [G11] After 30 seconds, forward the request to the second taxi in queue and put the first at the end

- [R1] After a missed confirmation the system has to move the first driver in queue to the last position and forward the request to the second until one driver confirm the request.

3.3 The world and the machine

We use "The World and Machine" model by M.Jackson & P.Zave to describe our analysis. This model allowed us to identify all phenomena/things not observable by machine ("The World), all shared phenomena/things (Shared Phenomena), composed by domain-controlled and machine-controlled, and all phenomena/things not observable by domain composed by all the operations performed by the system that people can't see ("The Machine).



3.4 Scenarios

3.4.1 Scenario 1

Bob has planned to go to a workshop that will take place on the other side of Milan. He woke up early because he knows that there's always so much traffic in the city. Bob is ready to go with his car when the unexpected happens: the car seems broken. Without wasting any time he opens myTaxi app on his phone. He logs into the system and request a Taxi for a simple ride. John, already logged into the system as driver, sees the simple request on his phone and accepts it. The system immediately notifies Bob telling him John's taxi code and the relative waiting time. Bob has to attend just a few minutes because John's Taxi is only one kilometer away from him. Bob is happy because he will arrive on time despite the unexpected discovery of having his car broken.

3.4.2 Scenario 2

Bob, during the lunch break at the workshop, realizes that at the end of the event he doesn't have his car because he arrived in taxi. He also realizes that the end of the workshop is scheduled at 6p.m, critical time for traffic in the city with several waiting time for taxis. In order to this consideration, Bob opens myTaxi app and reserves a taxi for 6.15 p.m. to be sure that a taxi will be there at that moment. At 6.05 p.m. Bob's phone rings. A notification with the code of taxi that will arrive to him has just been received from his phone.

3.4.3 Scenario 3

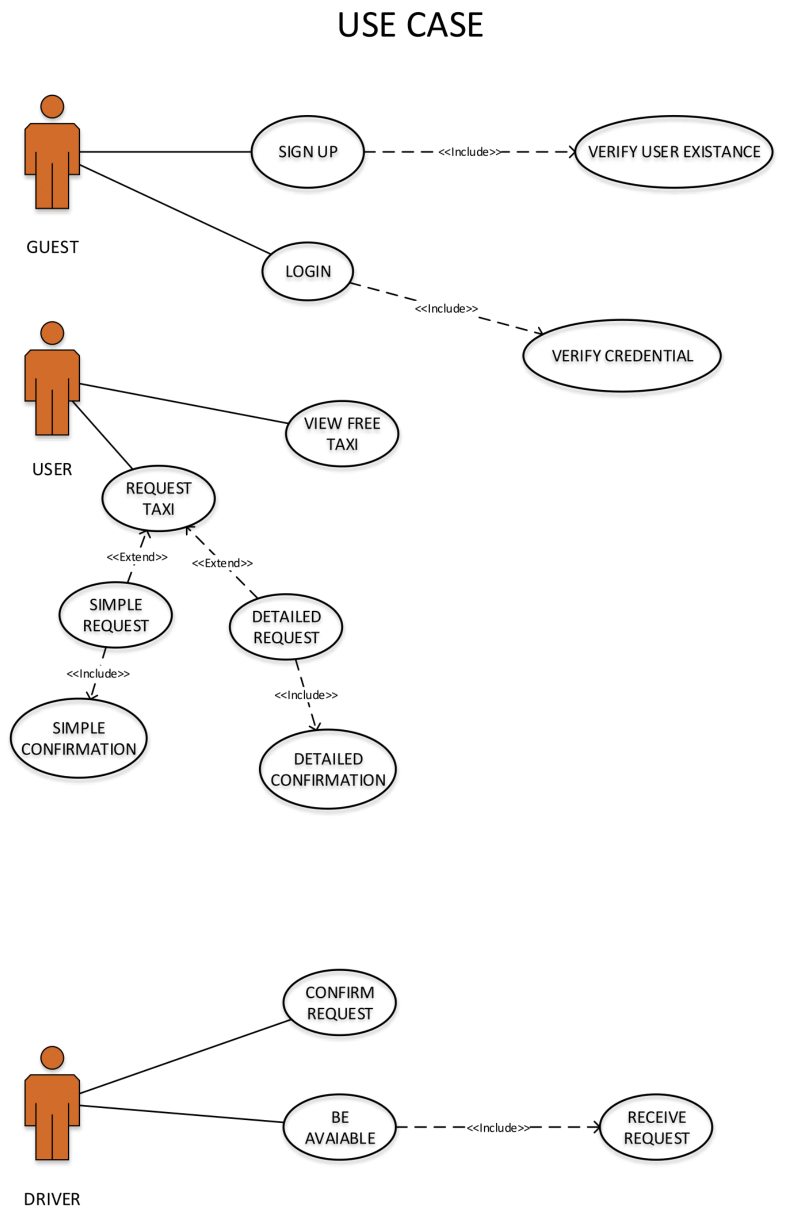
It's 12a.m. and John, a taxi driver, decides to have a coffee break but he also decides to not log out from the app because he is the third in the Taxi Queue. He thinks he can drink a coffee and smoke a cigarette before he becomes the first of the Taxi Queue... but he is wrong. The other two driver that were in list before him log out from the system and John becomes the first in the Taxi Queue. A request arrives but Bob doesn't see it because he is still drinking his coffee. After 30 seconds John hasn't accepted yet and the system puts him into the last position forwarding the request to the second in list.

3.4.4 Scenario 4

Giuly decides to go to EXPO in the evening. She knows that the queue to enter is very long so she decides to go there at 5 pm in order to be in a good position when the gate will open. Giuly is scared to take the train to go there because she would be going back home alone. She decides to go there with a taxi and simply she opens myTaxi app and requests a taxi for a simple ride. Unfortunately, the system has detected that there is no one taxi available in her zone at the moment. Giuly immediately receives a notification that informs it of the problem inviting her to attend for the first available taxi.

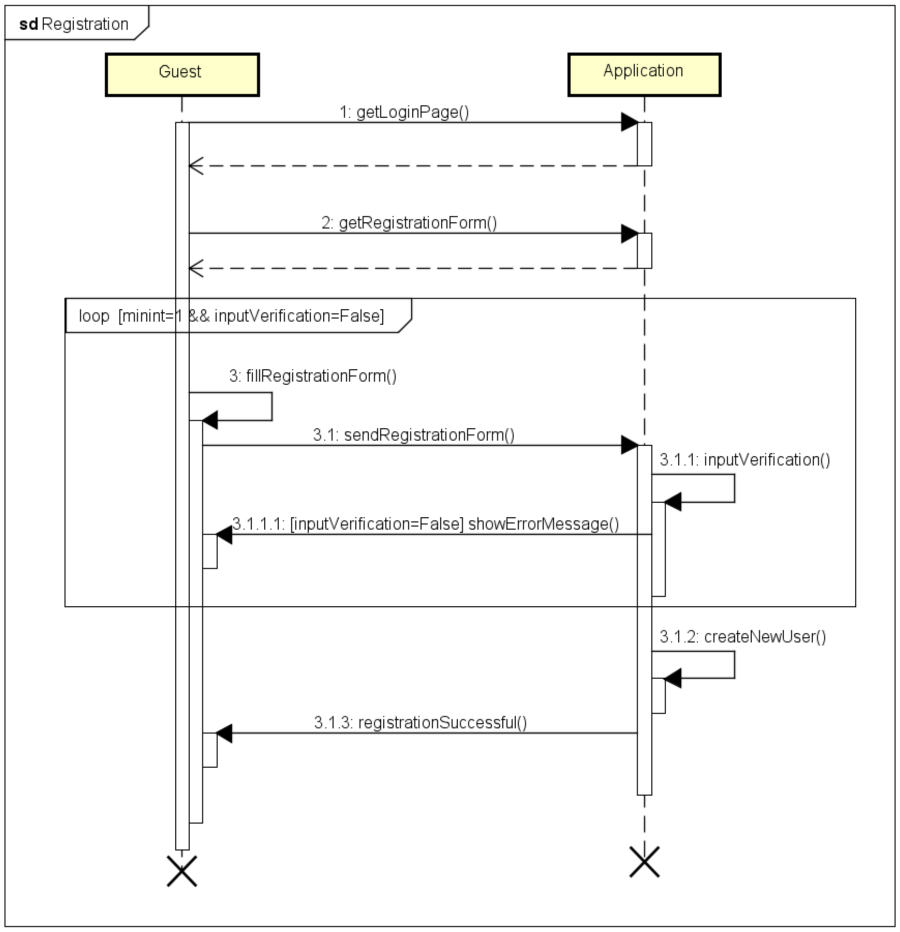
3.5 UML models

3.5.1 Use case diagram



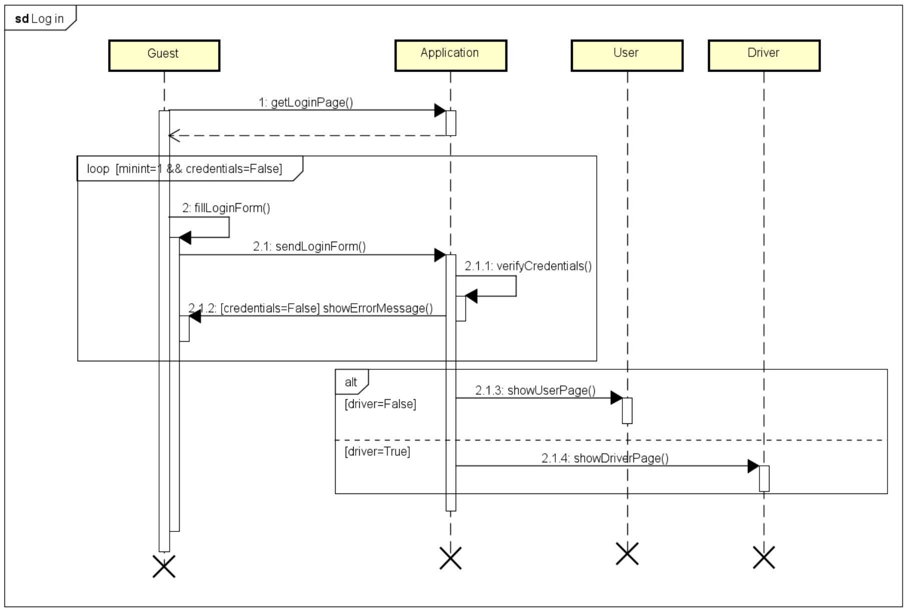
3.5.1.1 Sign up

Actor	Guest
Goal	[G1]
Entry conditions	NULL
Flow of events	1. Guest on the login page clicks on the “Register” button to start the registration process. 2. Guest fills in at least all the mandatory fields. 3. Guest clicks on “Sign up” button.
Exit conditions	Guest successfully ends the registration process. From now on he will be able to log in with the credentials submitted before and start using the application as a User.
Exceptions	1. Some mandatory field is not filled. 2. Chosen username is already taken. 3. Email is already in use. The exceptions are handled by showing an error and making the guest repeat the registration process from point 2 of the event flow.


sd Registration
Guest
Application
1: getLoginPage()
2: getRegistrationForm()
loop [minint=1 && inputVerification=False]
3: fillRegistrationForm()
3.1: sendRegistrationForm()
3.1.1: inputVerification()
3.1.1.1: [inputVerification=False] showErrorMessage()
3.1.2: createNewUser()
3.1.3: registrationSuccessful()

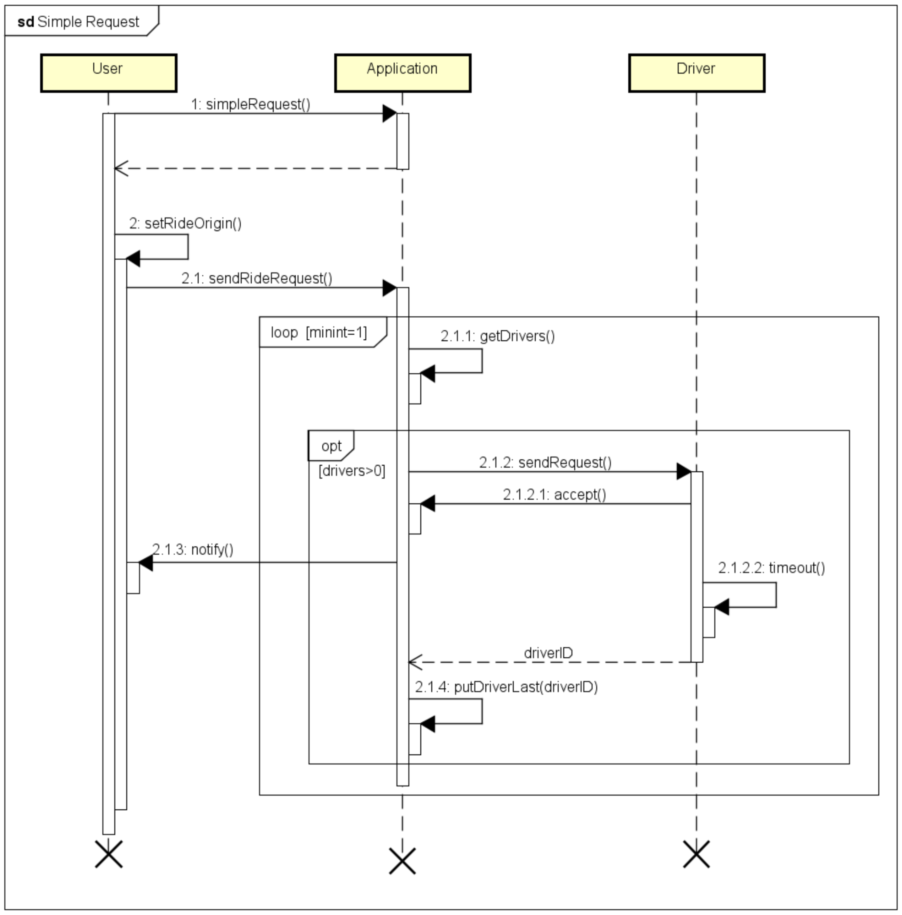
3.5.1.2 Log in

Actor	Guest
Goal	[G2]
Entry conditions	Guest must be already registered into the system.
Flow of events	1. Guest inserts username and password. 2. Guest clicks on the “Log in” button.
Exit conditions	The credentials are successfully validated, the application checks if the credentials belong to a user or a driver and shows their respective personal page.
Exceptions	An error is shown when the credentials inserted are not correct, making the user repeat the login process.



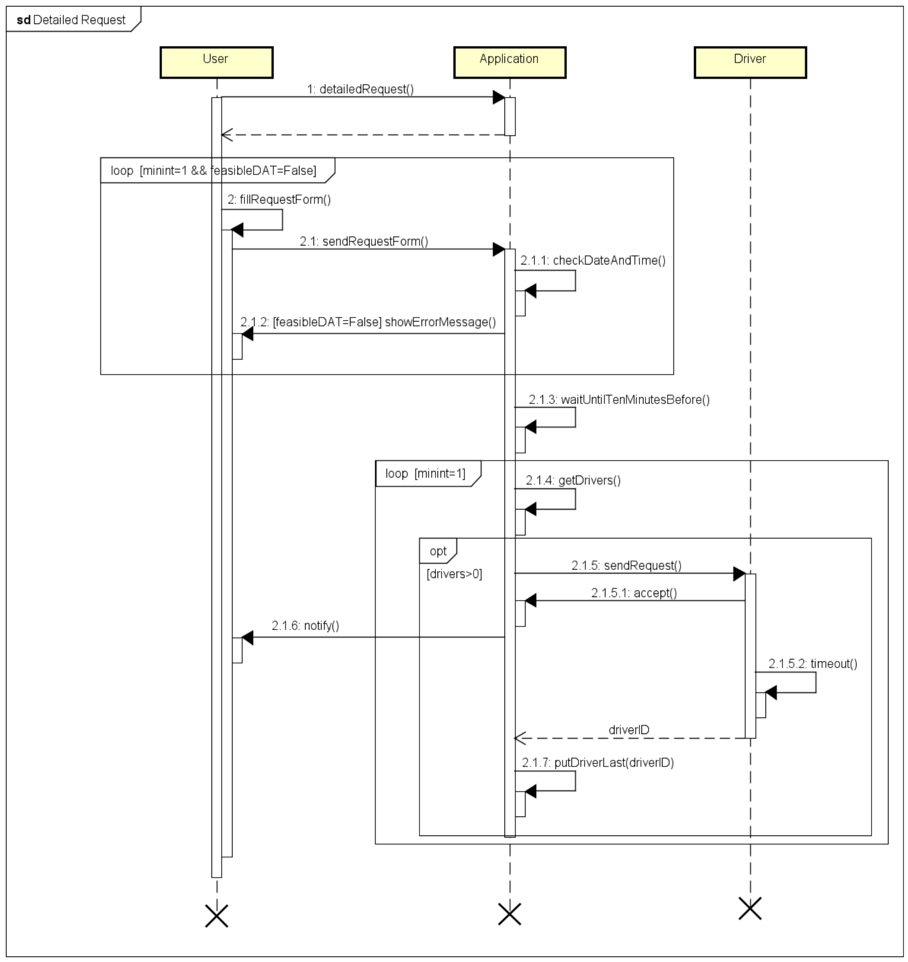
3.5.1.3 Simple request

Actor	User
Goal	[G4]
Entry conditions	User must be logged in.
Flow of events	1. User clicks on the "Simple request" button.
Exit conditions	User successfully makes a request which is still pending until a driver accepts it.
Exceptions	None. If there's no taxi, the system waits until one becomes available.



3.5.1.4 Detailed request

Actor	User
Goal	[G5]
Entry conditions	User must be logged in.
Flow of events	1. User clicks on the “Detailed request” button. 2. Fills the required fields, that are origin, destination, date and time of the ride. 3. Clicks on “Submit”.
Exit conditions	User successfully makes a request which is still pending until 10 minutes before the specified time, then the system allocates a taxi to the ride.
Exceptions	If the user inputs unfeasible date and time the application asks to input them again. Regarding the taxi queue there’s no exception, if there is no taxi in that zone at the moment, the application waits until one becomes available.


sd Detailed Request
User
Application
Driver
1: detailedRequest()
loop [minint=1 && feasibleDAT=False]
2: fillRequestForm()
2.1: sendRequestForm()
2.1.1: checkDateAndTime()
2.1.2: [feasibleDAT=False] showErrorMessage()
2.1.3: waitUntilTenMinutesBefore()
loop [minint=1]
2.1.4: getDrivers()
opt
[drivers>0]
2.1.5: sendRequest()
2.1.5.1: accept()
2.1.6: notify()
2.1.5.2: timeout()
driverID
2.1.7: putDriverLast(driverID)

3.5.1.5 View free taxis

Actor	User
Goal	[G3]
Entry conditions	User must be logged in.
Flow of events	1. User clicks on the "View available taxis" button.
Exit conditions	The application shows the number of available taxis in the zone the user is in, according to GPS.
Exceptions	None.

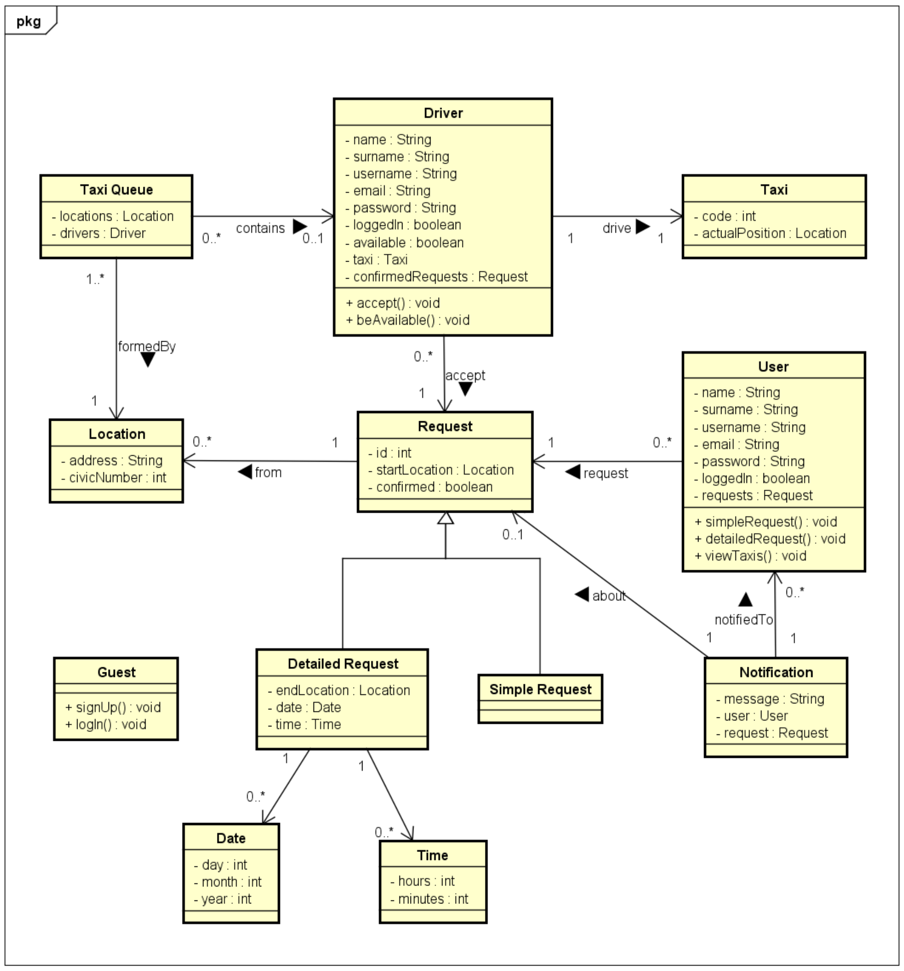
3.5.1.6 Be available

Actor	Driver
Goal	[G6]
Entry conditions	Driver must be logged in and not already available.
Flow of events	1. Driver clicks on the "Be available" button.
Exit conditions	The driver is set as available, put into the queue of taxi in his zone and can now receive ride requests.
Exceptions	If the driver is already available the application shows an error.

3.5.1.7 Confirm request

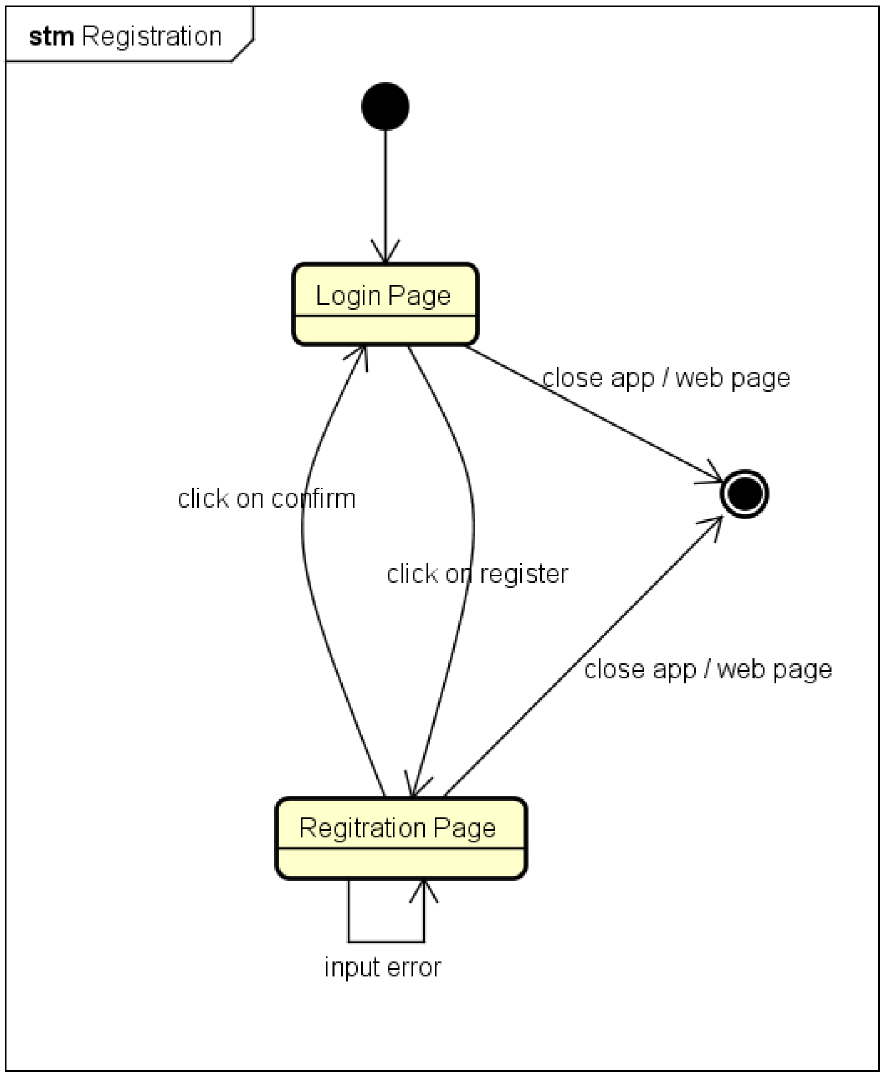
Actor	Driver
Goal	[G7]
Entry conditions	Driver must be logged in and available. Also he must have been notified by the system for a ride request.
Flow of events	1. Driver clicks on the “See requests” button. 2. Driver accepts one of the requests by clicking on the “Accept” button.
Exit conditions	The driver is now in charge of the ride and the system informs the user about the taxi details.
Exceptions	If the driver doesn’t accept within 30 seconds the request is forwarded to the next available driver and this one is put at the end of the queue of his zone.

3.5.2 Class diagram

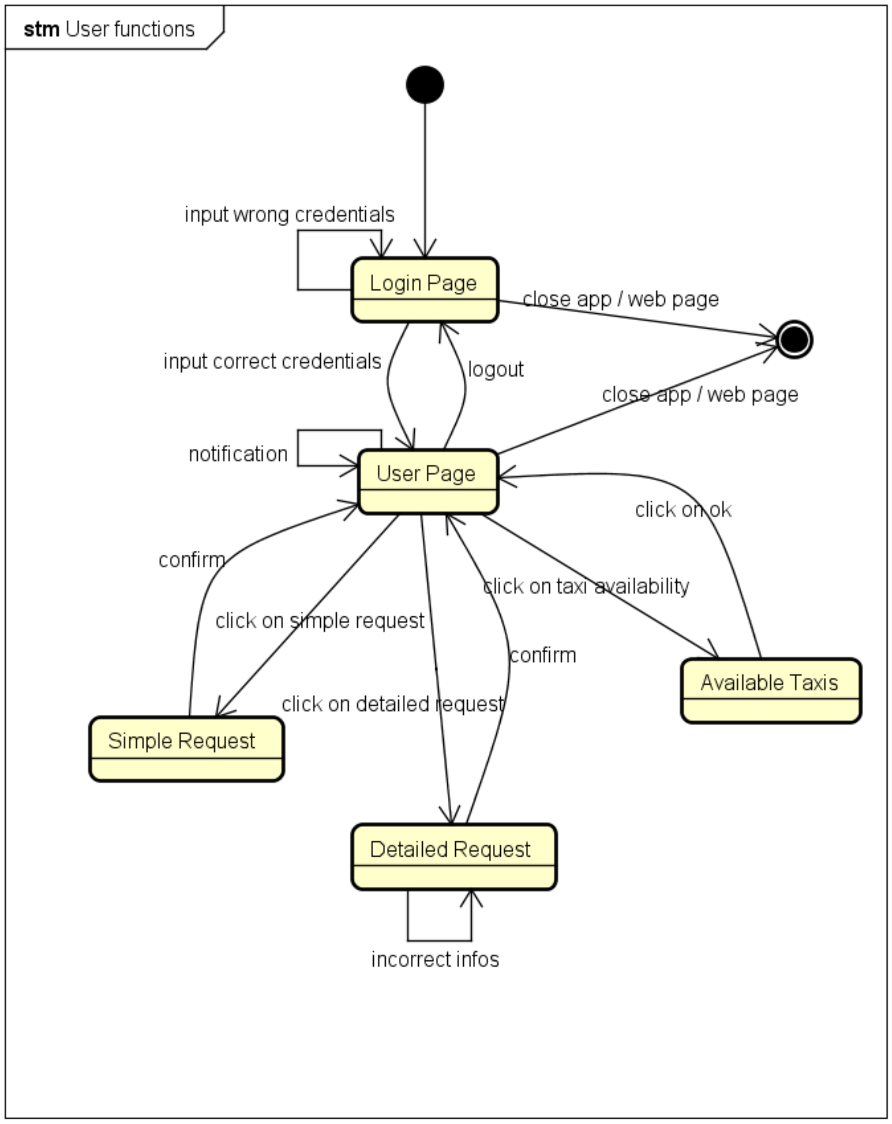


3.5.3 State machine diagrams

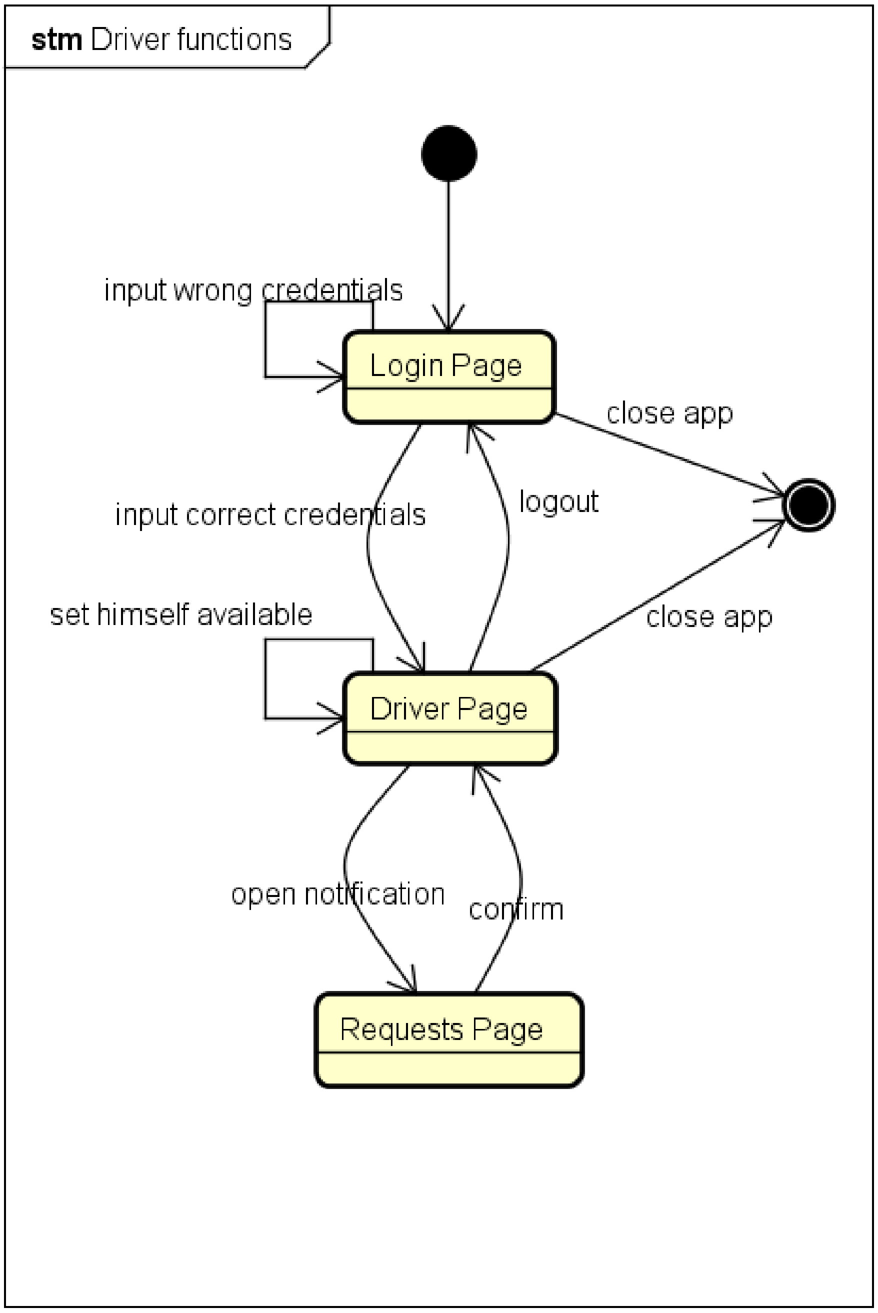
3.5.3.1 Registration



3.5.3.2 User functions



3.5.3.3 Driver functions



4 Appendix

4.1 Alloy

We have done this model referring to the Class Diagram. The file taxi.als can be found on our repository (https://github.com/MassimoSchiavo/MyTaxiService-Schiavo-Cittar/Deliveries). We have divided signatures, facts, assertions and predicates. In the last part there is the metamodel created with alloy analyzer and a world created with the predicate show.

4.1.1 Signatures

```
sig Integer{}
sig Strings{}
enum Boolean{YES,NO}

sig TaxiQueue{
        locations: set Location,
        drivers:set Driver}

sig Driver{
        available:one Boolean,
        taxi:one Taxi,
        confirmedRequest: set Request}

sig User{
        requests: set Request}

sig Time{
        hour:one Integer,
        minute:one Integer}

sig Date{
        day:one Integer,
        month:one Integer,
        year:one Integer}

sig Taxi{
        code:one Integer,
        actualPosition:one Location,
}

sig Location{
        address:one Strings,
        civicNumber:one Integer}
```

4.1.2 Abstract signature

```
abstract sig Request{
        id:one Integer,
        confirmed:one Boolean,
        startLocation:one Location}
```

4.1.3 Implementation of abstract signature

```
sig SimpleRequest extends Request{}

sig DetailedRequest extends Request{
        endLocation:one Location,
        date:one Date,
        time:one Time}
```

4.1.4 Facts

This is the fact part that defines the constraints of the class.

```
fact noEmptyDate{
        all d:Date|(#d.day=1)and(#d.month=1)and(#d.year=1)
}

fact noEmpyTime{
        all t:Time|(#t.hour=1) and (#t.minute=1)
}

fact noEmptyDriver{
        all d:Driver|(#d.taxi=1)
}

fact noEmptyTaxi{
        all t:Taxi|(#t.code=1) and (#t.actualPosition =1)
}

fact noEmptyRequest{
        all r:Request|(#r.id=1) and (#r.startLocation=1) and (#r.confirmed=1)
}

fact noEmptyDetailedRequest{
                all rd:DetailedRequest|(#rd.endLocation=1) and (#rd.date=1) and (#rd.time=1)
}

fact noEmptyLocation{
        all l:Location|(#l.address=1) and (#l.civicNumber=1)
}

fact noEmptyTaxiQueue{
        all tq:TaxiQueue|(#tq.locations>0)
}
```

```
//Every location is insert into one and only one TaxiQueue and a TaxiQueue is formed at least by one
location
fact noDuplicatedTaxiQueue{
        all l:Location|no disj tq1,tq2:TaxiQueue|(l in tq1.locations) and (l in tq2.locations)
        all l:Location{some tx:TaxiQueue|l in tx.locations}
}

//Every taxi has only one driver and any driver drives only one taxi
fact oneDriverOneTaxi{
        all d:Driver|no disj t1,t2:Taxi|(t1 =d.taxi) and (t2 = d.taxi)
        all t:Taxi|no disj d1,d2:Driver| (t =d1.taxi) and (t = d2.taxi)
}

//Every request is unique and exits only if a user has done it
fact noDuplicatedRequest{
        no disj r1,r2:Request|r1.id=r2.id
        all r:Request|no disj u1,u2:User|(r in u1.requests) and (r in u2.requests)
        all r:Request|one u:User| r in u.requests
}

//Every Taxi has different code between each other
fact noDuplicatedTaxi{
        no disj t1,t2:Taxi|t1.code=t2.code
}

//All drivers available are isert into a queue in corrispondance of his taxi's location
//Drivers not available are not into a Taxi Queue
fact avaiableToQueue{
        all d:Driver|d.available=YES => {one tq:TaxiQueue|(d in tq.drivers) and (d.taxi.actualPosition in
tq.locations)}
        all d:Driver|d.available=NO => {all tq:TaxiQueue|(d not in tq.drivers)}
}

//A notification exist only if the notified has done a request that has already been accepted
fact notify{
        all n:Notification|one u:User|(u = n.user) and{one r:Request|(r in n.request) and (r in u.requests)
and (r.confirmed=YES)}
        all r:Request|(r.confirmed=YES)=>one u:User|r in u.requests and{one n:Notification|(n.user=u)
and (n.request=r)}
}

//StartLocation and endLocation of a detailed request must be two different place
fact diffLocation{
        all dt:DetailedRequest|dt.startLocation!=dt.endLocation
}
```

```
//Two Taxi Queue with a shared driver must not exist
fact oneQueuePerDriver{
	all d:Driver|no disj tq1,tq2:TaxiQueue|(d in tq1.drivers) and (d in tq2.drivers)
}

//driver can accept request only if are in taxiqueue in which the startLocation is included
fact Acceptance{
	all r:Request|r.confirmed=YES <=> {one tq:TaxiQueue|(r.startLocation in tq.locations)and{one
d:Driver|(d in tq.drivers) and(r in d.confirmedRequest)}}
	all r:Request|r.confirmed=NO <=> {one tq:TaxiQueue|(r.startLocation in tq.locations)and{all
d:Driver|(d in tq.drivers) and (r not in d.confirmedRequest)}}
	all r:Request|no disj d1,d2:Driver|(r in d1.confirmedRequest) and (r in d2.confirmedRequest)
}
```

4.1.5 Assertions

These are the asserts used to verify the model.

```
//Every location can't be insert into two different taxis queues
assert LocationInoneTaxiQueue{
	all l:Location|no disj tq1,tq2:TaxiQueue|(l in tq1.locations) and (l in tq2.locations)
}
check LocationInoneTaxiQueue for 10

//Once a request is confirmed by a driver it cannot be confirmed by anyone anymore
assert RequestsConfirmedByOnlyOneDriver{
	all d1,d2:Driver|no r:Request|(d1!=d2) and (r in d1.confirmedRequest) and (r in
d2.confirmedRequest)
}
check  RequestsConfirmedByOnlyOneDriver for 10

//Every request is unique and belongs to only one user
assert oneOwner{
	all u1,u2:User|no r:Request|(u1!=u2) and (r in u1.requests) and (r in u2.requests)
}
check oneOwner for 10

//Verify acceptance fact.
assert Acceptance{
	all sr:SimpleRequest|lone d:Driver|(sr.confirmed=YES) and (sr in d.confirmedRequest)
	all r:Request|no d:Driver|(r.confirmed=NO) and (r in d.confirmedRequest)
	all r:Request|r.confirmed=YES implies one tq:TaxiQueue|(r.startLocation in tq.locations) and {one
d:Driver|(d in tq.drivers) and (r in d.confirmedRequest) and (d.taxi.actualPosition in tq.locations)}
}
check Acceptance for 5

//Verify Notification fact.
assert Notification{
	no n:Notification|n.request.confirmed=NO and n.user=none
}
check Notification for 6

//Verify available to Queue
assert availableToQueue{
	all tq:TaxiQueue|no d:Driver|(d in tq.drivers) and (d.available=NO)
	all tq:TaxiQueue|all d:Driver|(d in tq.drivers)=>(d.available=YES)
}
check availableToQueue for 6

//Verify DiffLocation fact.
assert DiffLocation{
	all dt:DetailedRequest|no l:Location|dt.startLocation=l and dt.endLocation=l
}
check DiffLocation for 6
```

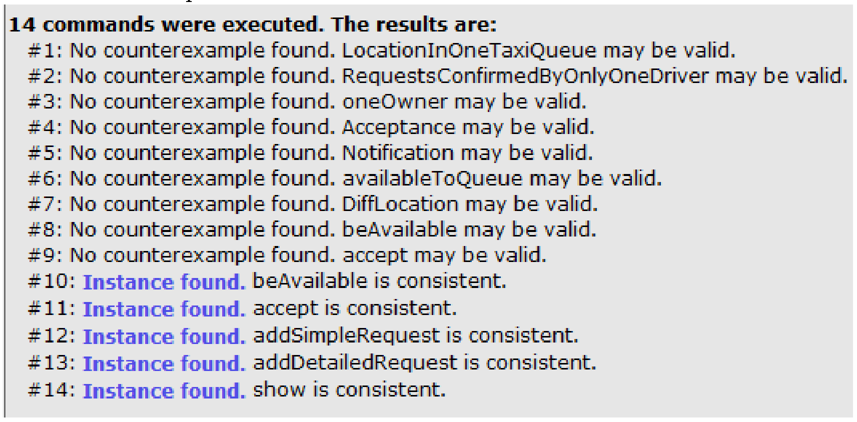
```
//Verify beAvailable pred.
assert beAvailable{
	all d:Driver|(d.available=NO) and beAvailable[d] implies (d.available=YES)
}
check beAvailable for 6

//Verify accept pred.
assert accept{
	all d1,d2:Driver|all r:Request| ((d1=d2) and(d1.available=YES) and (r.confirmed=NO) and
accept[d1,d2,r])implies((r in d2.confirmedRequest) and d2.available=NO and r.confirmed=YES)
}
check accept for 6
```

4.1.6 Predicates

```
pred beAvailable(d:Driver){
        (d.available=NO) implies (d.available=d.available - NO + YES)
}
run beAvailable for 6

pred accept(d1,d2:Driver,r:Request){
        (r not in d1.confirmedRequest and d1.available=YES and d1=d2) implies
(d2.confirmedRequest=d1.confirmedRequest+r and r.confirmed=YES and d2.available=d1.available-YES+NO
)
}
run accept for 4

pred addSimpleRequest(u1,u2:User,sr:SimpleRequest){
        (sr not in u1.requests) implies (u2.requests=u1.requests+sr)
}
run addSimpleRequest for 4

pred addDetailedRequest(u1,u2:User,dt:DetailedRequest){
        (dt not in u1.requests) implies (u2.requests=u1.requests+dt)
}
run addDetailedRequest for 4

pred show(){
        #TaxiQueue>1
        #User>1
        #Driver>1
        #Taxi>1
        #Request>1
        #Location>1
}
run show for 4
```

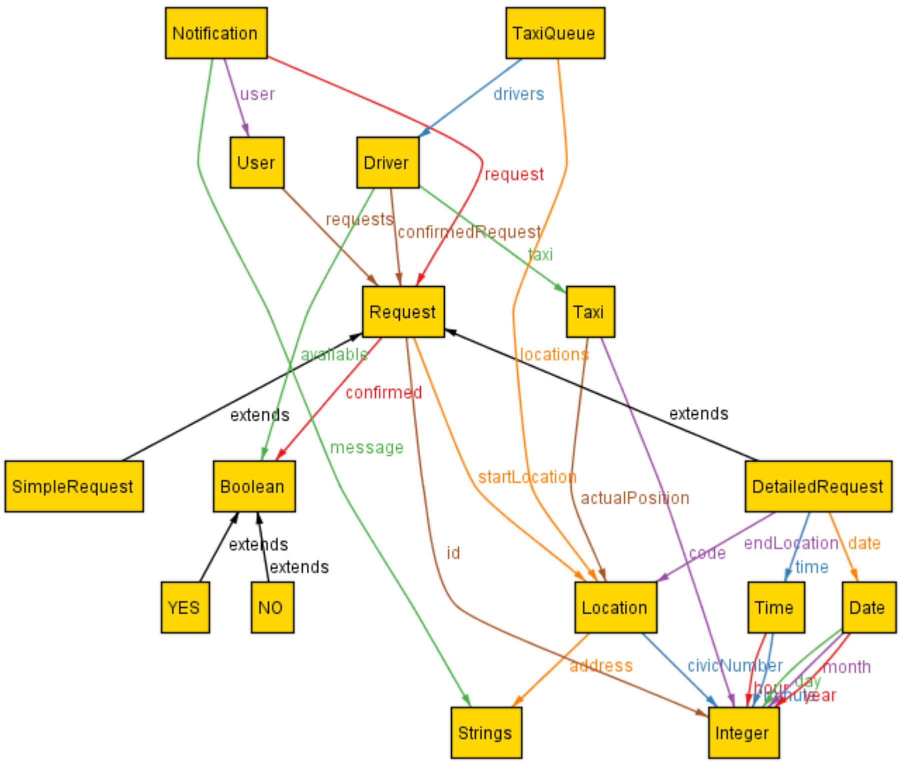
4.1.7 Results and metamodel

These are the results obtained with the alloy analyzer that show the consistency of the model in all parts.



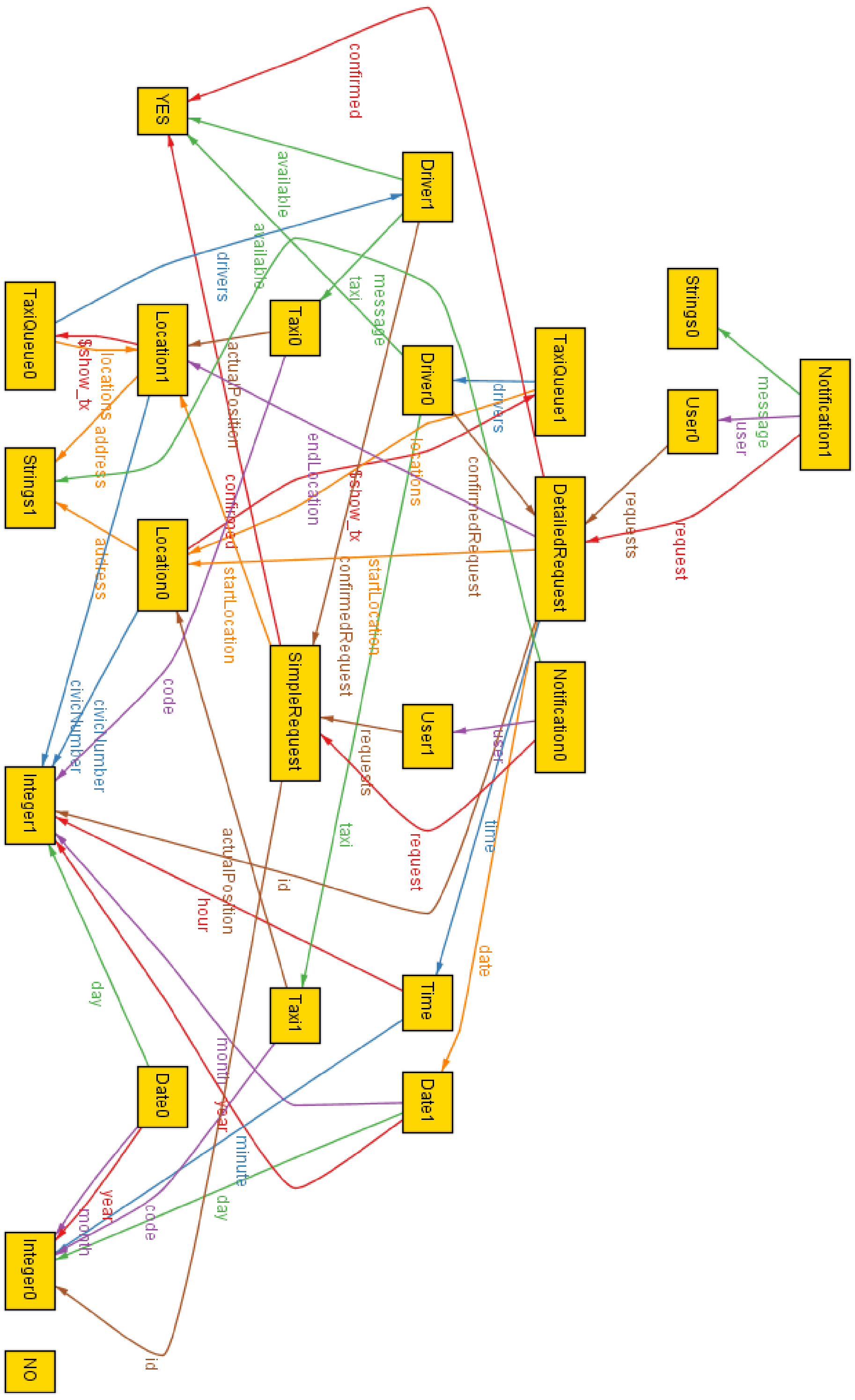
```
14 commands were executed. The results are:
  #1: No counterexample found. LocationInOneTaxiQueue may be valid.
  #2: No counterexample found. RequestsConfirmedByOnlyOneDriver may be valid.
  #3: No counterexample found. oneOwner may be valid.
  #4: No counterexample found. Acceptance may be valid.
  #5: No counterexample found. Notification may be valid.
  #6: No counterexample found. availableToQueue may be valid.
  #7: No counterexample found. DiffLocation may be valid.
  #8: No counterexample found. beAvailable may be valid.
  #9: No counterexample found. accept may be valid.
  #10: Instance found. beAvailable is consistent.
  #11: Instance found. accept is consistent.
  #12: Instance found. addSimpleRequest is consistent.
  #13: Instance found. addDetailedRequest is consistent.
  #14: Instance found. show is consistent.
```

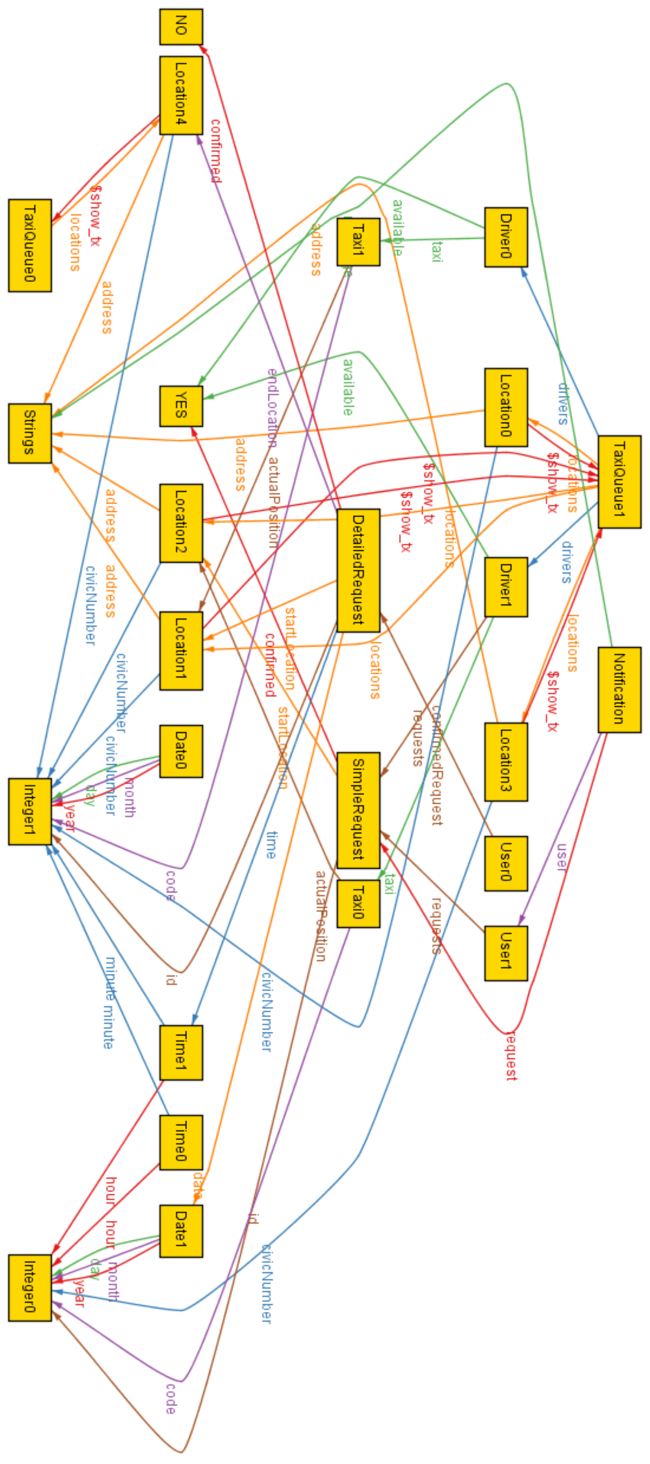
A screenshot that represents the metamodel created.

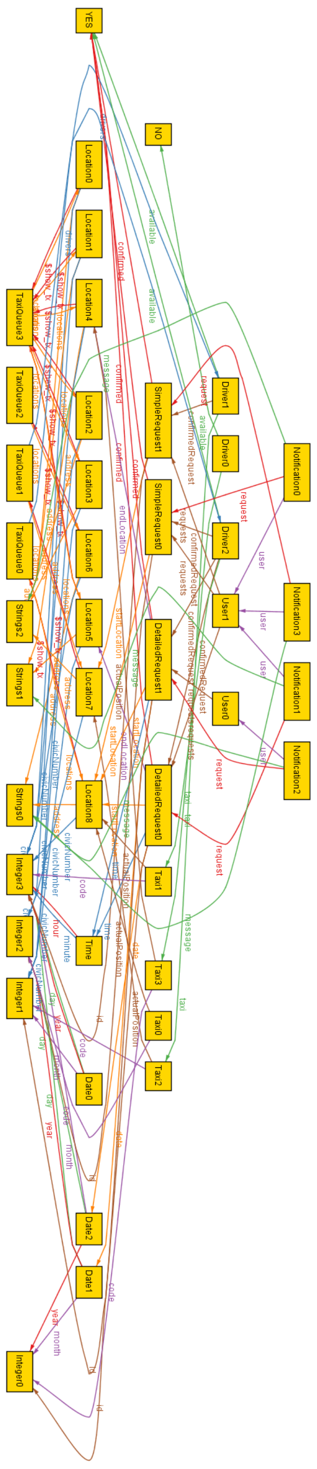


4.1.8 Generated world

The first screenshot represents the world generated with the alloy analyzer using the predicate show() for 2 cases. The second using the predicate show() for 2 cases with exactly five Location. The third instead using show for 4 cases with exactly two SimpleRequest and nine Location. With more cases the scheme would become impossible to read and understand because of the presence of too many arrows so we decided not to put more cases in this document.


YES
Driver1
Driver0
TaxiQueue0
TaxiQueue1
Location1
Location0
Strings1
Strings0
Taxi0
Taxi1
User0
User1
Notification1
Notification0
DetailedRequest
SimpleRequest
Integer1
Integer0
Time
Date1
Date0
NO
confirmed
available
available
drivers
drivers
taxi
taxi
message
message
user
user
request
request
requests
requests
locations
locations
$show_tx
$show_tx
actualPosition
actualPosition
endLocation
startLocation
startLocation
confirmedRequest
confirmedRequest
confirmed
address
address
civicNumber
civicNumber
code
code
id
id
hour
minute
time
date
day
day
month
month
year
year


NO
Location4
TaxiQueue0
Strings
YES
Location2
Location1
Date0
Integer1
Time1
Time0
Date1
Integer0
Taxi1
DetailedRequest
SimpleRequest
Taxi0
Driver0
Location0
Driver1
Location3
User0
User1
TaxiQueue1
Notification
confirmed
$show_tx
locations
address
available
taxi
drivers
endLocation
actualPosition
startLocation
requests
confirmedRequest
civicNumber
month
day
year
code
id
minute
hour
time
date
user
request


YES
NO
Location0
Location1
Location2
Location3
Location4
Location5
Location6
Location7
Location8
TaxiQueue0
TaxiQueue1
TaxiQueue2
TaxiQueue3
Strings0
Strings1
Strings2
Integer0
Integer1
Integer2
Integer3
Time
Date0
Date1
Date2
SimpleRequest0
SimpleRequest1
DetailedRequest0
DetailedRequest1
Taxi0
Taxi1
Taxi2
Taxi3
Driver0
Driver1
Driver2
User0
User1
Notification0
Notification1
Notification2
Notification3

4.2 Software and tool used

- LyX: to redact and format this document.
- Microsoft Visio: to make the Use Case Diagram.
- Astah Professional: to make Class, Sequence and State Machine Diagrams.
- GIMP: to draw the drafts of the application.
- Alloy Analyzer 4.0: to prove the consistency of our model.

4.3 Hours of work

This is the time spent to redact this document:

- Massimo Schiavo: 32 hours.
- Marco Edoardo Cittar: 32 hours.