My Taxi Service

RASD Document
(Requirements and Specifications Document)

Politecnico di Milano **A.A. 2015-2016**
Software Engineering 2 Raffaela Mirandola

Salvatore Perillo 853349 Claudio Sesto 841623

SUMMARY

1.Introduction

1.1 Purpose

This document is written following the standard IEEE 830-1998 (RASD) and helps all those who are in charge to develop the software and eventually those programmers who will have to make updates.
This document contains the requirements for the development of the software and some invented scenario as example of the functionalities included in the system.
This document can also be used during the discussion between customer and developer before or during the actual implementation to avoid misunderstadings.

1.2 Scope

The main software that will be developed is “myTaxiService”; the aim of this software is to allow people to use better the taxi service already existing in the city both via web application or via smartphone.
The application grants the client to request a taxi or make a reservation 2 hours before the meeting time and the system must ensure a fair management of the taxis in order to avoid some of them wasting time by do nothing. The client doesn’t need to register to request a taxi.
The taxi driver, to be inserted in the queue of the available taxis, must register by inserting name, surname, e-mail address, the number of the license, username and a password; after this, he can login with a username and a password and start working. Eventually, in case of need, the taxi driver can set his state to “Busy” so that the system doesn’t forward requests to him.
In order to improve service and help for future changes, the software allow technicians to develop additional services via programmatic interfaces.

1.3 Definitions,acronyms and abbrevations

- Client: whoever needs the taxi service
- Registered User: someone who has registered to the service
- Taxi driver: a person whose job is to take people in a car to the place they want to go to in return for money
- Technicians: those people who add additional service
- System: the set of software and hardware used

1.4 References

- Specification Documents: MyTaxiServices
- IEEE std 830-1998

1.5 Overview

This document is composed by 4 main part:

- Part 1 : Intoduction - In this part you can find a initial presentation of this document and all introductory information about the software.

- Part 2 : Overall Description - the second part contains more accurate information about the software.All about constraints,assumptions,interfaces and also information about the users
- Part 3 : Specific requirements - here there are all the information concerning the requirement of the project
- Part 4 : Appendixes - all the supporting information are included in this part

2.Overall Description

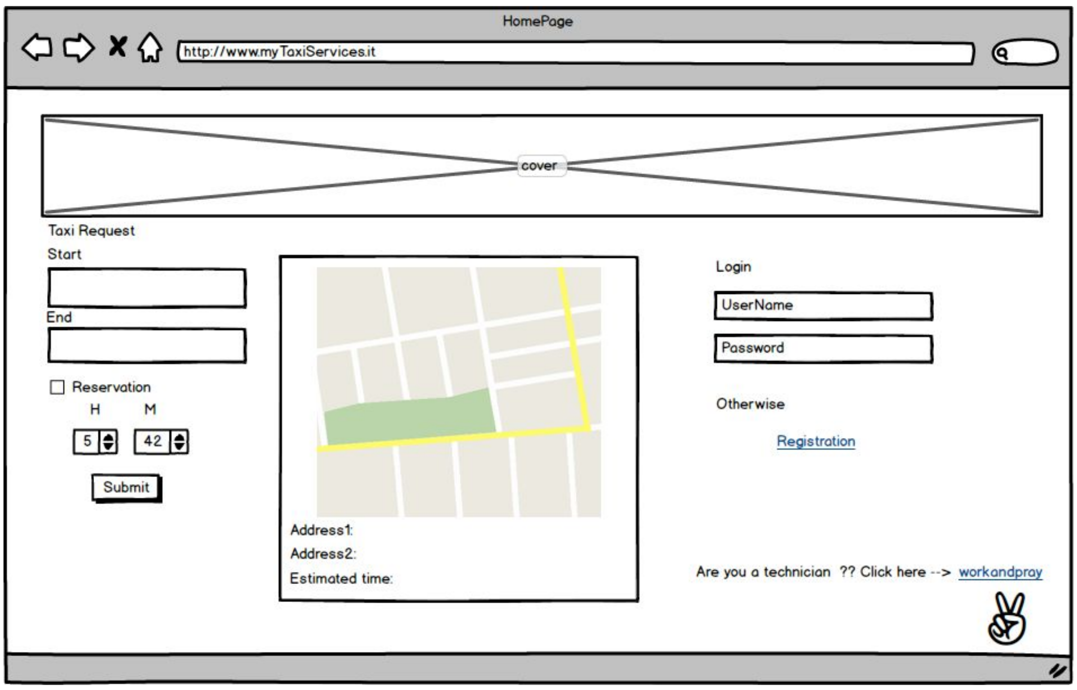
2.1 Product perspective

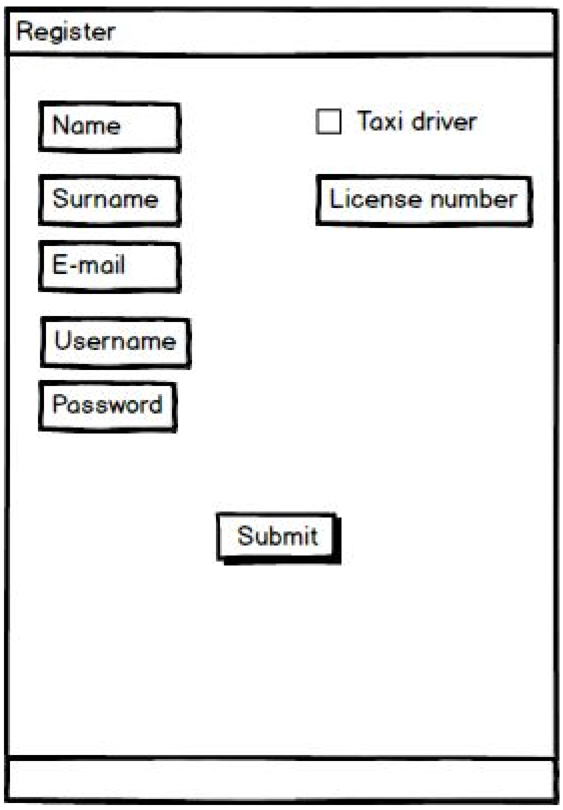
The system consist in a web page and a smartphone app, so it must be available for any browser and an application that works on Android, iOS and Windows phone operative system; this also implies that every taxi driver must have an appropriate smartphone.
The system is developed without having any kind of support/interaction with other systems, so is totally independent. Due the fact that the application can be easily updated with the programmatic interface, we don't exclude an interaction in the future with other systems.

2.1.1 Interfaces

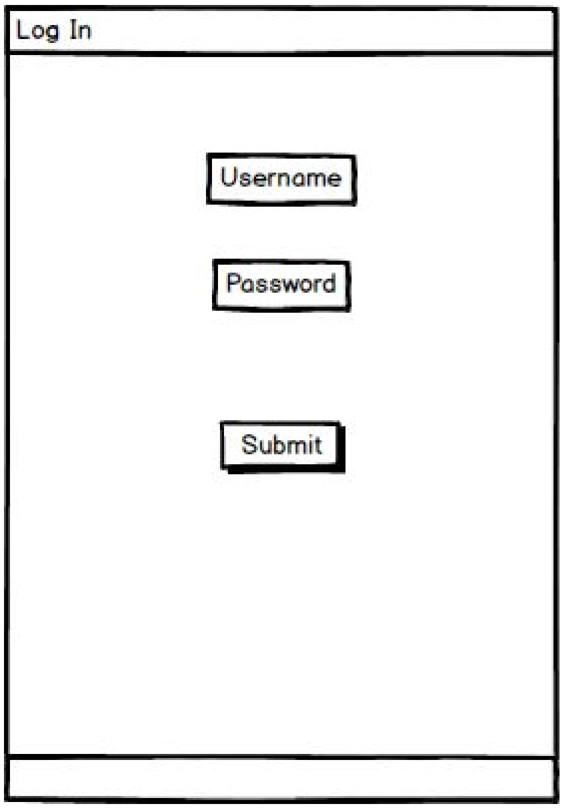
Homepage

This is the Homepage of the web application.Here a user can request a ride (normal or reservation)There is the input form for the login and,if you want you there is a link for registration and another one for future implementation of the programmatic interfaces(now you can see only few news about coming idea)

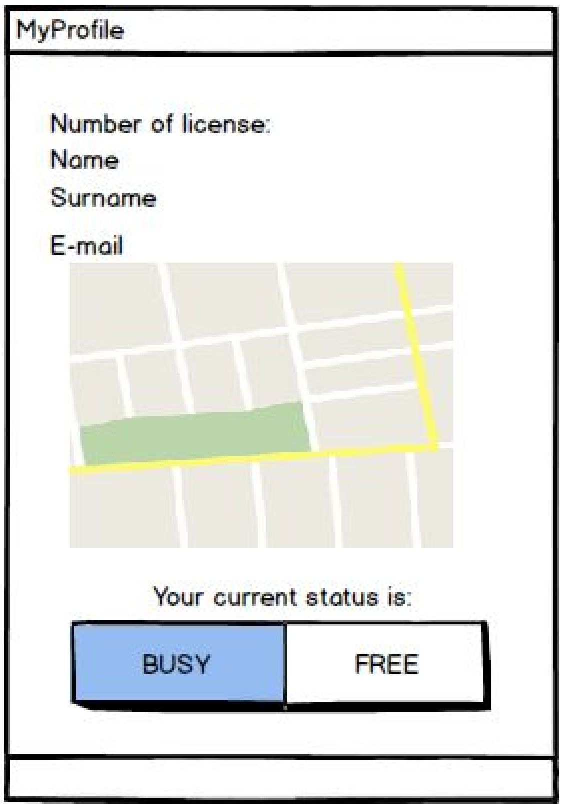




Registration Page
Here there is a example of the registration page (only for the mobile app) This page is equal both for customer and taxi driver.Only taxi driver can fill the field about the license(there will be a control about the validity of the field).The registration is not mandatory to request a ride ,is only additional feature for further funcionalities.



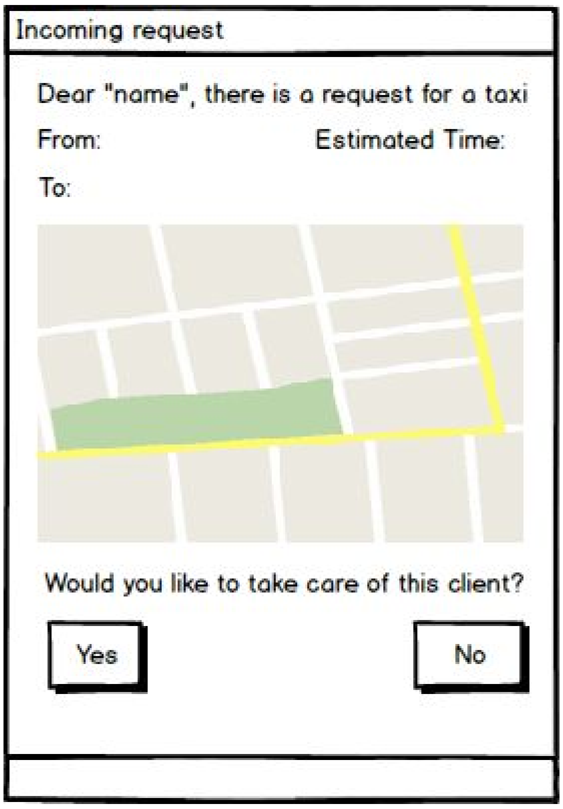
Log In:
This is a standard page for the log in that request username and password to let a registered user to view it’s own profile; this page is the same for user and taxi driver



2.1.1.1 Taxi driver interface

My Profile

this page contains all the informations about the driver. The map indicates the current position of the taxi and the buttons below set the state of the taxi and are mutually exclusive (of course can't be busy and free at the same time).


Incoming request
Dear "name", there is a request for a taxi
From:
Estimated Time:
To:
Would you like to take care of this client?
Yes
No

Incoming Request

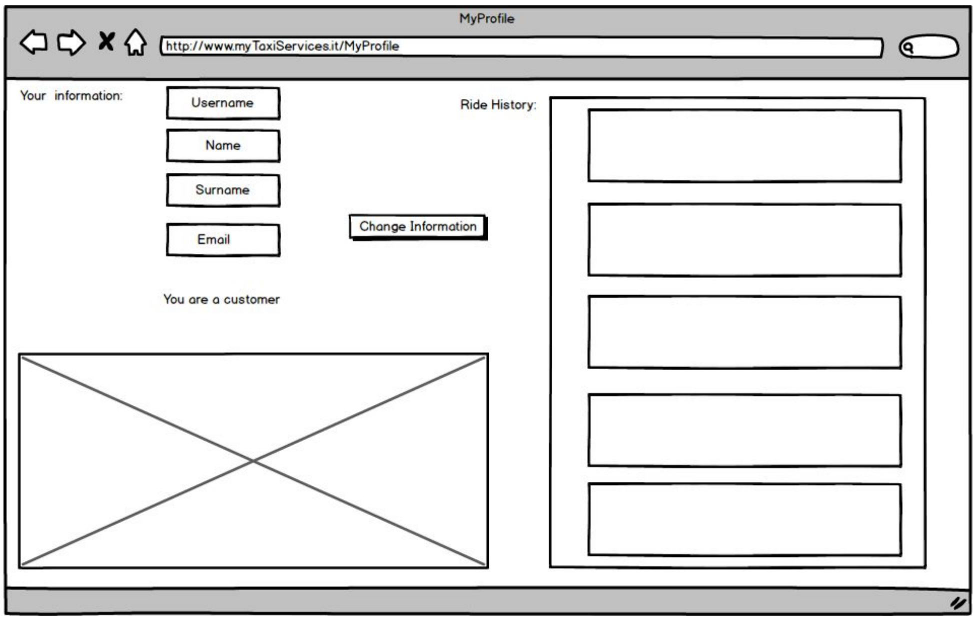
Whenever the system assign a ride to the taxi, both for a simple ride or a reservation, the app shows automatically this window showing the informations about the trip: the taxi driver must send a response within 2 minutes otherwise the window disappear, like he pressed "no".

If the taxi driver accept the request, the window disappear and goes automatically to "MyProfile" with the status set on "busy"

2.1.1.2 User Interface

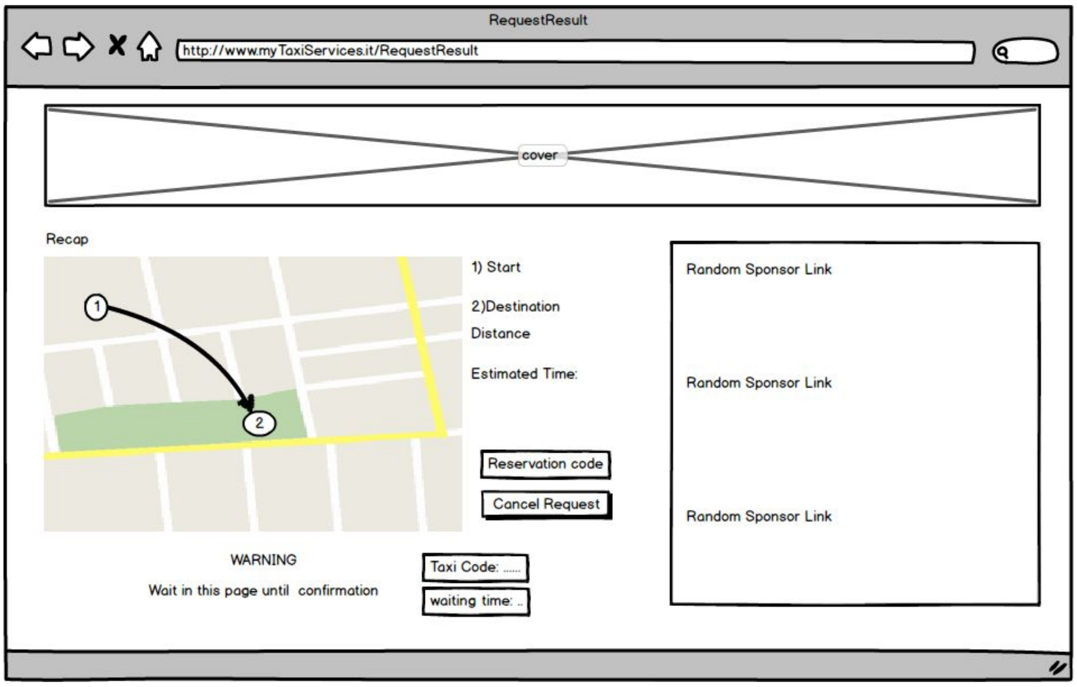
MyProfile

Below is represented the profile page of a customer profile (only web app) which contains all the information.There is also a ride history.In this page a user can also change his information.



RequestResult

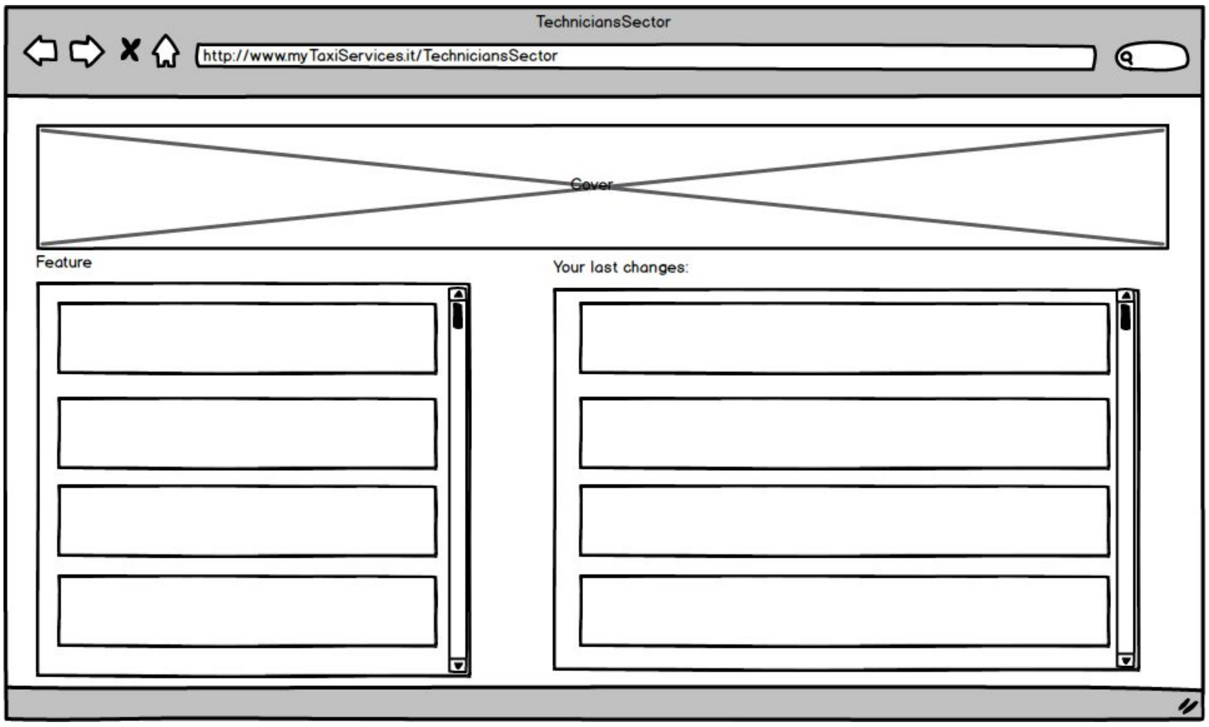
This page show all information concerning the just made request. After few minutes a customer can see the code of the taxi and the waiting time. If the request is a reservation ,the system only shows a confirmation message and a taxi will be allocated 10 minutes before the meeting time. If a user wants to cancel the reservation can press the cancel request button.



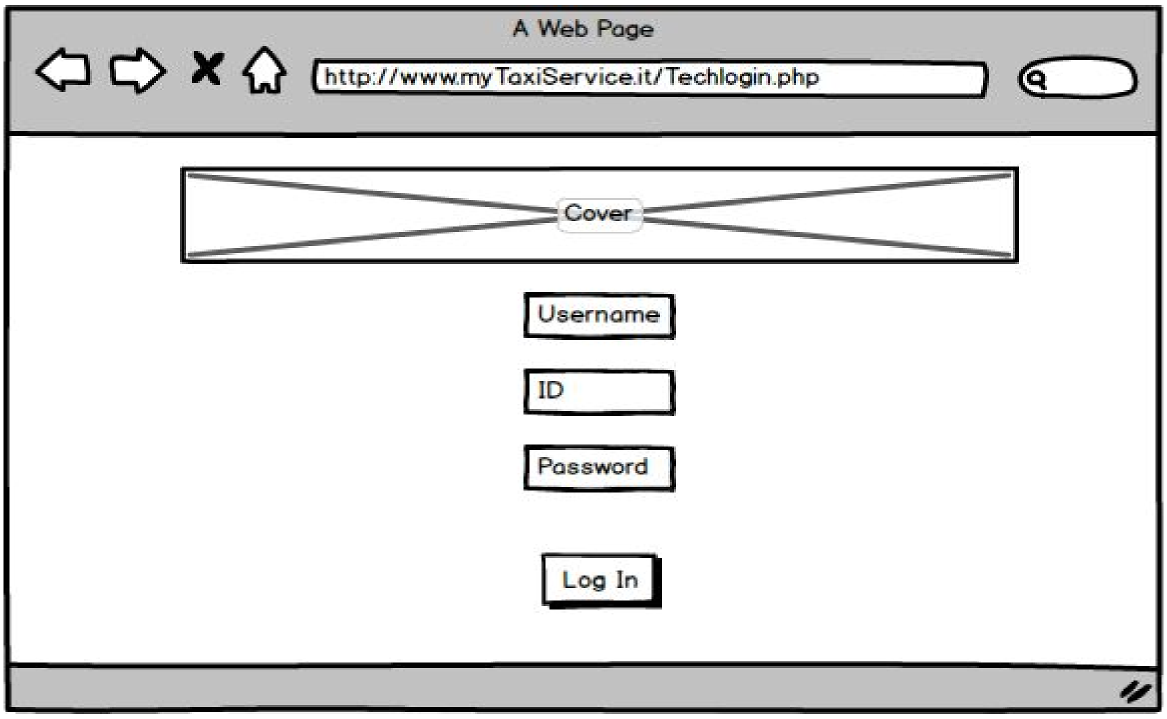
2.1.1.3 Technician Interface

Technicians Sector

This page is the one that a technician see whenever he logs in, there is a list of feature he is involved to work and by selecting one, on the right appear the history of the changes applied in the feature.
Near each feature there is also a link to access it and modify it.



Technician log in
This is the page that let a technician to log in; the technician must insert the ID assigned to him when he was hired: this ensure that only authorized people are allowed to enter and modify features



2.1.2 Hardware interfaces
there aren't any kind of support for hardware interfaces

2.1.3 API interfaces
We use a map indication API to show the route. In addition there are information about the start and destination addresses and concerning the estimated time of the trip.

2.1.4 Software interfaces
- Database Management System(DBMS)
  -Name: MySQL
  -Version: 5.7
  -Source: http://www.mysql.it/
- Application server:
  – Name: DigitalOcean
  – Version: 6.1.
  – Source:https://cloud.digitalocean.com

2.1.5 Communication interfaces

We use theTCP protocol for security reasons.

2.1.6 Memory

No memory constraints are required.

2.2 Goals

Below are listed the main goal of myTaxiService

1. Allow guest to register the application
2. Allow user/guest to make a reservation
3. allow user/guest to delete a reservation
4. Allow user/guest to take a ride
5. Allow a technicians to implement additional features
6. Allow a taxi driver to set its own state
7. Allow the system to allocate correctly a taxi based on the currently location
8. Allow user to change his information

2.3 User Characteristics

The audience of this application is anyone who needs a taxi service so the user part doesn't give any constraint; taxi drivers, the only thing they need to do is to have the license (otherwise they also cannot register to the website).
The technician instead must be a person with a high knowledge about computer science and programming languages used to develop apps for smartphone in general to avoid failure in the system and the service that doesn't works

2.4 Constraints

2.4.1 Regulatory Policies

myTaxiService doesn't meet any regulatory policies

2.4.2 Hardware Limitation

myTaxiService has only one kind of hardware limitations : a taxi driver must have a smatphone.
The system must use a database to store all informations about taxi drivers and registered users

2.4.3 Interfaces to other applications

myTaxiService doesn't meet any Interfaces to other applications

2.4.4 Parallel operation

myTaxiService must handle all the parallel requests from multiple users and doesn't lost any kind of data

2.5 Assumptions

1. A ride can end in any zone
2. each taxi can stay in one and only one zone
3. A client always receive a response
4. A taxi cannot have any kind of low functions
5. A taxi is never late for the meeting with the customer
6. A taxi can take care about only one request at a time
7. GPS signal is always on work, so we know each instant where each taxi is
8. At least one taxi in the all City is free
9. We already have a database with all taxis and the corresponding license
10. The company decided new features to be implemented and a team of technicians is chosen by giving them the permission to modify it.
11. The technicians are hired by the company: username, password and an ID are automatically assigned to them
12. The way the system assign GPS coordinates into a zone is already implemented
13. If a queue in the zone of a request is empty, the system forward the request to the queue of an adjacent zone
14. Zone are square shaped

3. Requirements

3.1 Functional Requirements

1. Allow guest to register the application in order to receive upcoming features
    - The system must have a database to store all registred users and provide a sign up functionality
    - A person who already registered, cannot register again unless previous unregistration

2. Allow user/guest to make a reservation
    - The system provide a page to make a reservation to any user
    - The guest have to insert the information about the trip: starting point, destination, meeting time
    - All these informations must be inserted 2 hours before the meeting time; the system send a message either the reservation is done or not.

3. Allow user/guest to delete a reservation
    - The system also have to provide the possibility to cancel an existing reservation from web or app; in this last case, no taxi will be allocated

4. Allow user/guest to take a ride
    - The system allow any user (from web site or smartphone app) to send a request for a taxi
    - After a request is make, the system have to send a message to the taxi at the top of the queue

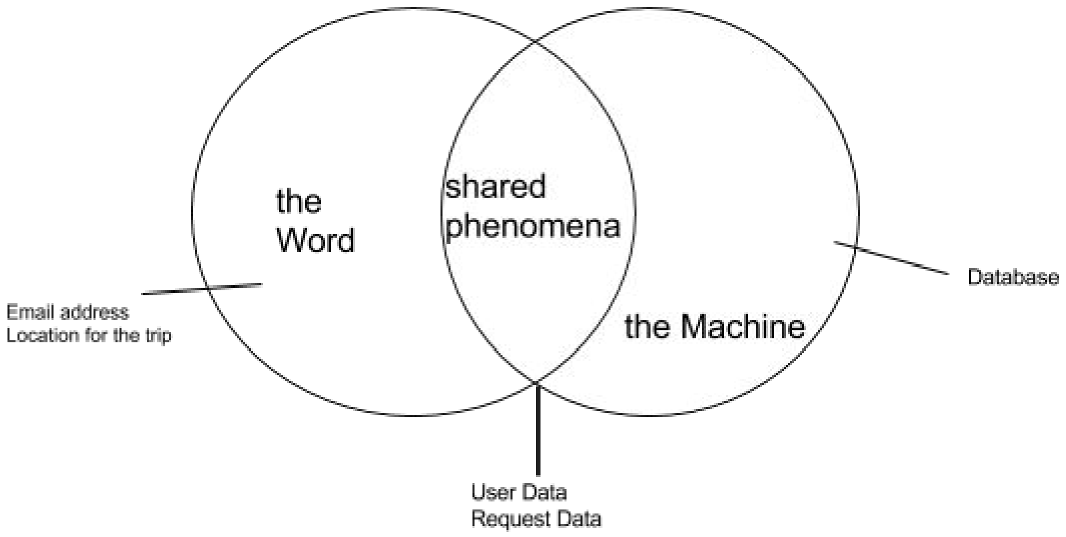
5. Allow a technicians to implement additional features
    - The technician must be logged in the correct section
    - After the login a technician can choose one of the feature that want to modify
    - If a technician isn't allowed to work on some features the system doesn't allow him to make any modification
    - When the new feature is completed, the system integrate it automatically.

6. Allow the system to allocate correctly a taxi based on the currently location
    - The system automatically gets GPS informations of all taxis and add them to the queue of the correct zone

7. Allow user to change his information
    - In the personal page there will be a link to change informations
    - Updated informations are stored in the database

3.2 The World and the Machine

We use "The World and The Machine" model by M. Jackson & P. Zave to understand the different kind of environments.In the World section we have all the "real" events and entities that interact with the application.The Machine domain rappresents the entities to be developed .In the intersection are rappresented the shared phenomenal,all world informations that are managed directly by the application.



3.3 Scenario

SCENARIO 1
Sean is a business man that needs to go to an important meeting but he is on late so need to call a taxi
The system answer the client that a taxi will come but Sean is nervous and is continuously sending requests that are ignored by the system; after 5 minutes a taxi arrives and brings Sean to the meeting.

SCENARIO 2
it's 17:00 Andrew have to be to the cinema in 1 hour and half and decide to get a reservation for a taxi; when he try to get the reservation for the 18:00, the system sends an error saying that a reservation must be done 2 hours before the meeting time.

SCENARIO 3
Jennifer request a taxi in the morning to join a party of her best friend that takes in the evening: the reservation goes through the system that add this reservation in the database. As requirement, 10 minutes before the meeting time, the system search for a free taxi according to the policy specified in the requirements: the taxi driver Joshua confirms the availability and its taxi is allocated by the system for the reservation.

SCENARIO 4
Sharon usually uses a taxi for moving around the city. After she discovers this new service, she decide to register in order to see the news of the upcoming features (taxi sharing,etc).
She fills the fields with the requested information (email,payment methods,etc) and at the end she sees the confirmation message.

SCENARIO 5
Paul needs to go to a barbecue in the evening so decide to take a reservation in the morning but in the afternoon, before the taxi is allocated, it starts to rain so the barbecue is canceled and Paul doesn't need anymore a taxi and cancel its reservation; the system receive the request to cancel the reservation and the procedure commit.
In all of this time, no taxi is bothered.

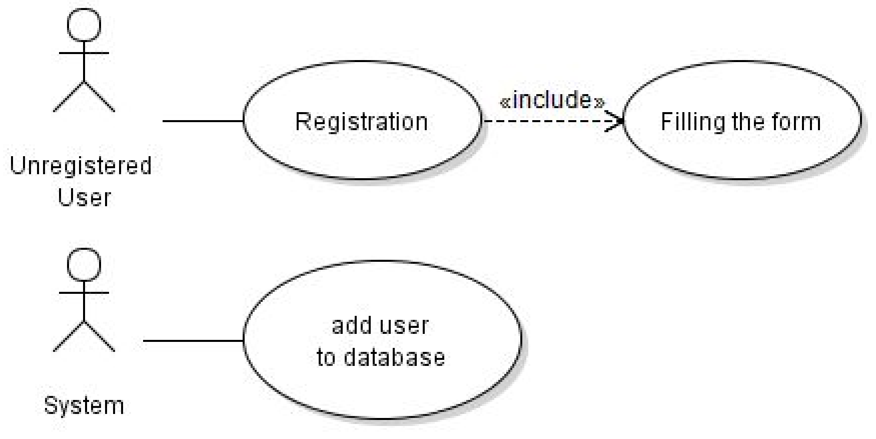
SCENARIO 6
Goffredo is a taxi driver and today worked a lot, so to take a break and get gas ,sets the status on "busy" and go to a petrol pump to relax a bit.
After 1 hour of rest he reset the status on "free" and after a while recieves the request for a ride.

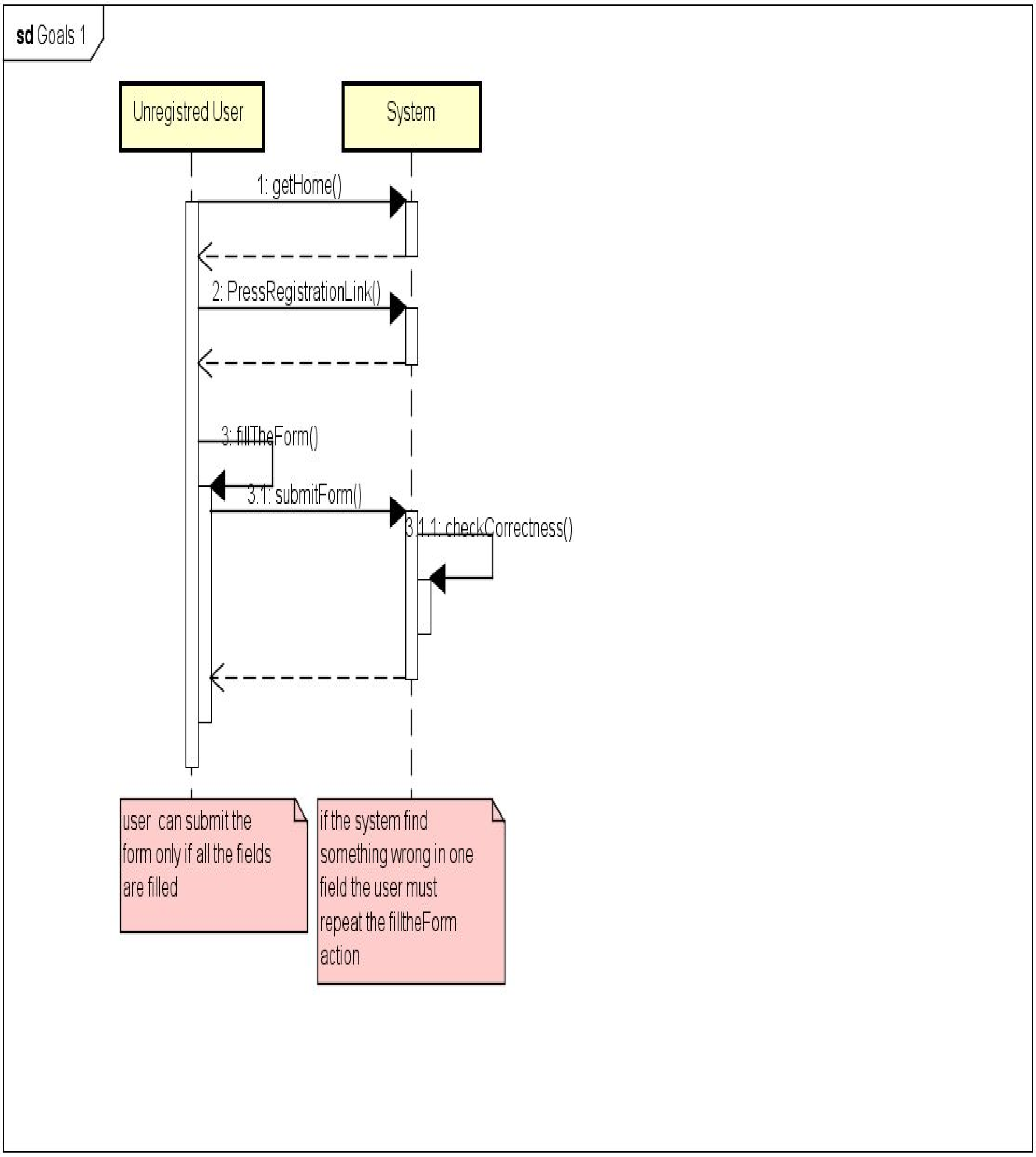
SCENARIO 7
Gennaro is a Computer Scientist and is hired by the company.
He has a lot of ideas and is assigned to a team working to develop a new feature; after this he enter in the web site, click on the link and start working.
At the end of the day, Gennaro is tired and committ his work so he and his team can continue the work later.
After some weeks, the work is completed and a new feature is implemented.

3.4 UML models

Goal 1

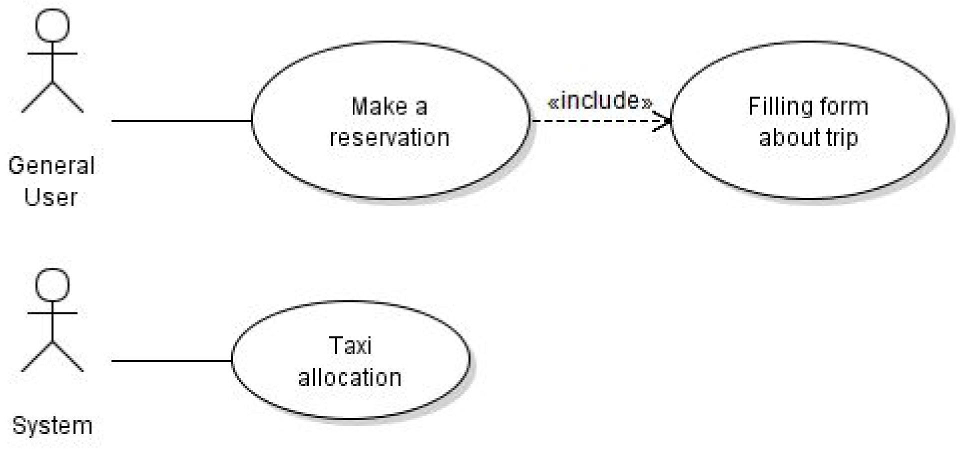
Actor	Unregistered user
Goal	Allow guest to register the application in order to receive upcoming features
Input condition	NULL
event flow	1) The user enter in the page for registration; 2) The user fills the form and press "submit"
output condition	All information are correctly inserted
Exceptions	If the form is not filled correctly, send an error message and start from point 2

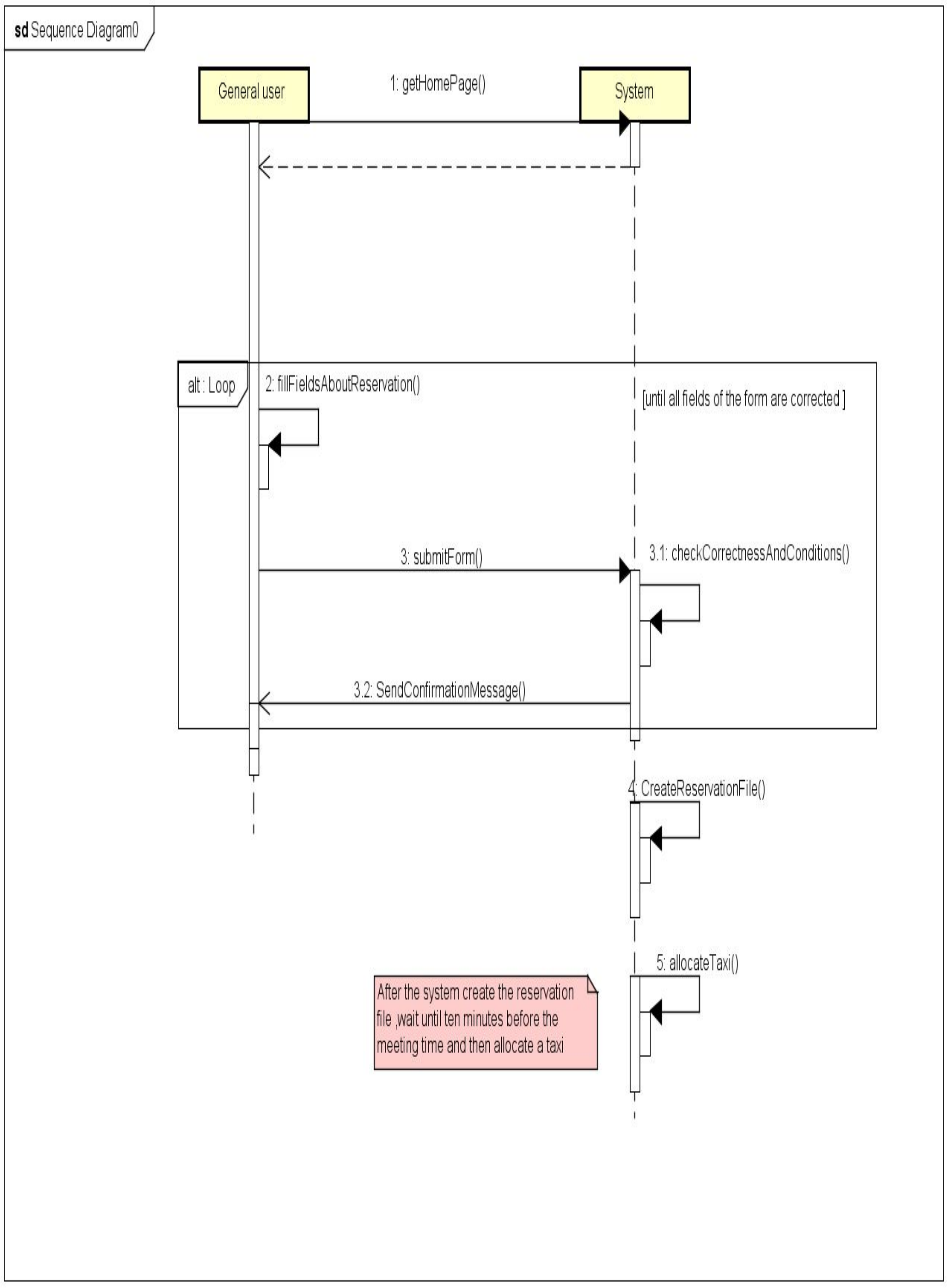



sd Goals 1
Unregistred User
System
1: getHome()
2: PressRegistrationLink()
3: fillTheForm()
3.1: submitForm()
3.1.1: checkCorrectness()
user can submit the form only if all the fields are filled
if the system find something wrong in one field the user must repeat the filltheForm action

Goal 2

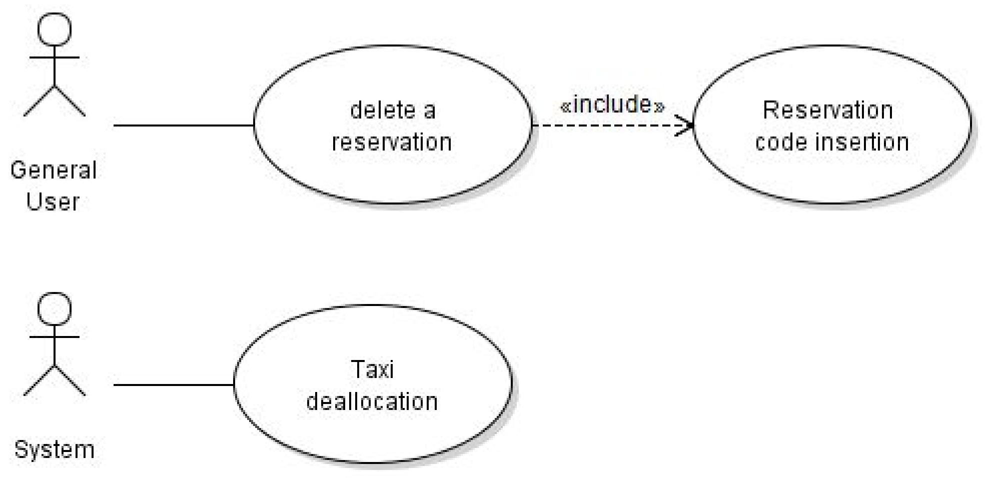
Actor	General user
Goal	Allow user/guest to make a reservation
Input condition	NULL
event flow	1) The user enters in the web site or application 2) The user insert departure point, departure schedule and destination 3) The user send request
output condition	All information are correctly inserted and the user receive the number of the reservation.
Exceptions	1) If the form is not filled correctly, send an error message and start from point 2. 2) If the form is not filled 2 hours before the departure schedule, send an error message.

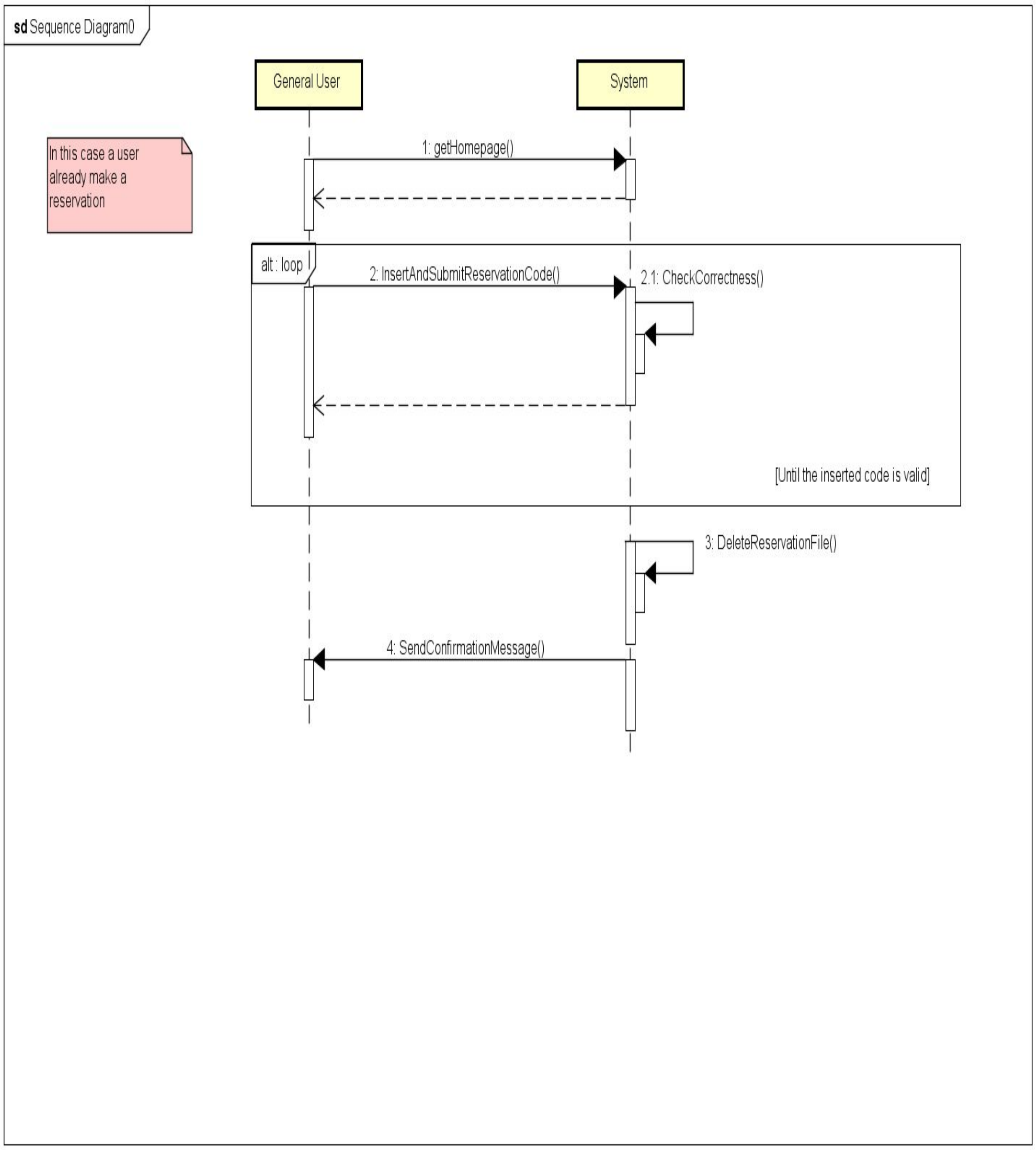



sd Sequence Diagram0
General user
System
1: getHomePage()
alt : Loop
2: fillFieldsAboutReservation()
[until all fields of the form are corrected]
3: submitForm()
3.1: checkCorrectnessAndConditions()
3.2: SendConfirmationMessage()
4: CreateReservationFile()
5: allocateTaxi()
After the system create the reservation file ,wait until ten minutes before the meeting time and then allocate a taxi

Goal3

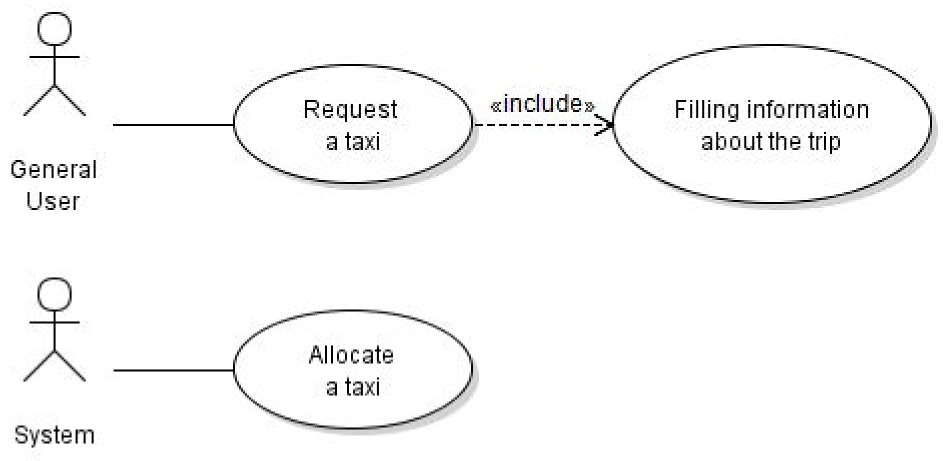
Actor	General user
Goal	Allow user/guest to delete a reservation
Input condition	The user has made a reservation
event flow	1) The user access the page to delete reservation 2) The user insert the code of reservation 3) The user press the “Confirm” button
output condition	The system remove the reservation and doesn’t allocate a taxi
Exceptions	code reservation is wrong, restart from point 2

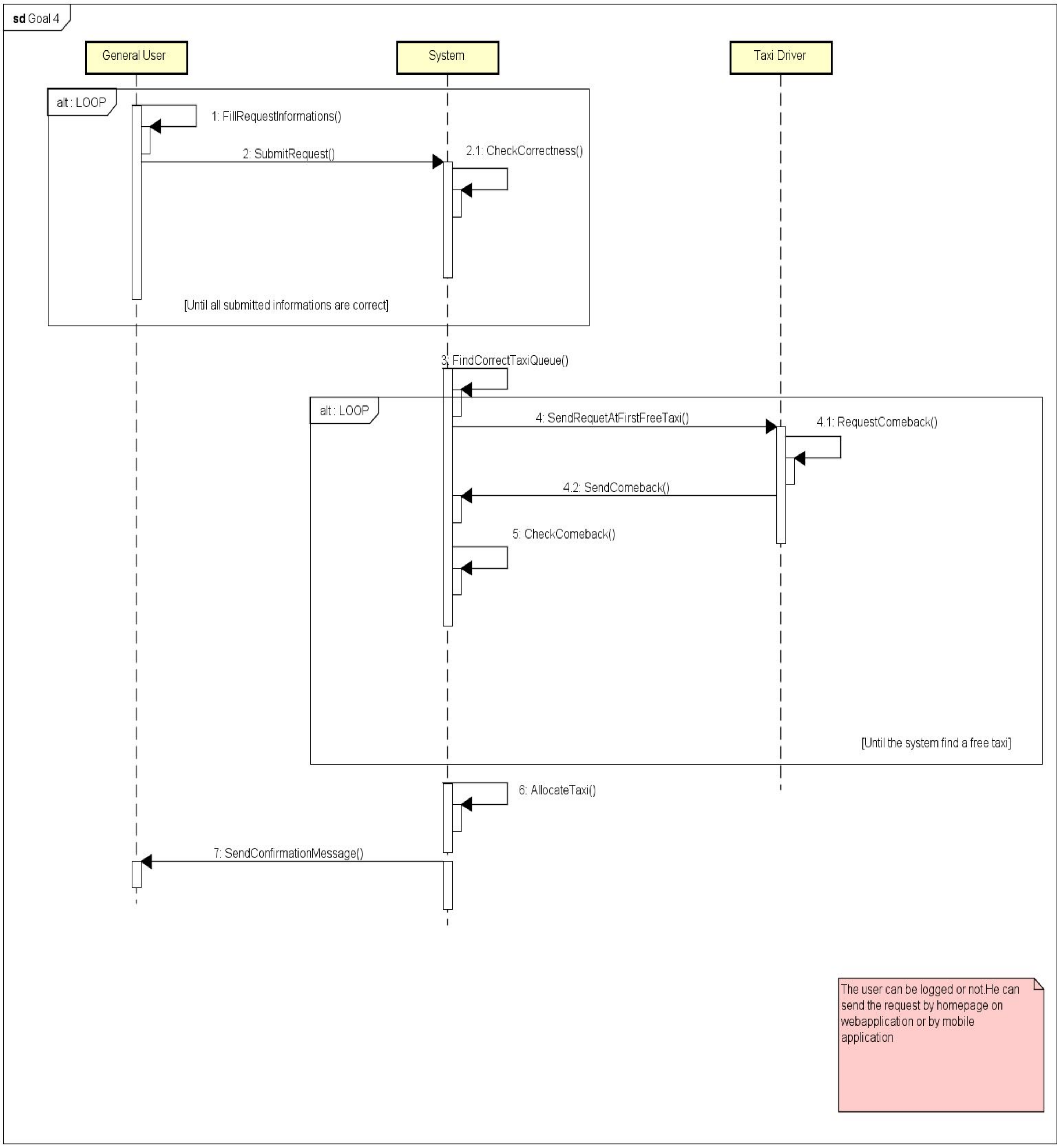



sd Sequence Diagram0
General User
System
In this case a user already make a reservation
1: getHomepage()
alt : loop
2: InsertAndSubmitReservationCode()
2.1: CheckCorrectness()
[Until the inserted code is valid]
3: DeleteReservationFile()
4: SendConfirmationMessage()

Goal 4

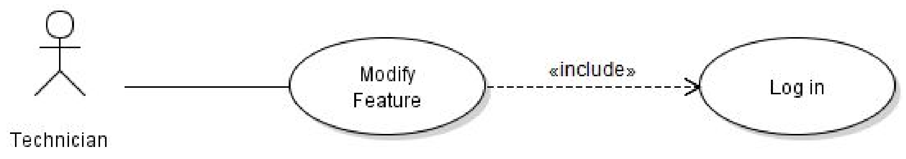
Actor	General user
Goal	Allow user/guest to take a ride
Input condition	NULL
event flow	1) The user access the application 2) The user insert departure and destination points 3) The user press “Confirm”
output condition	The system allocate a taxi and send a confirmation message to the user
Exceptions	If the user insert a wrong address both for departure or destination, the system send an error message.

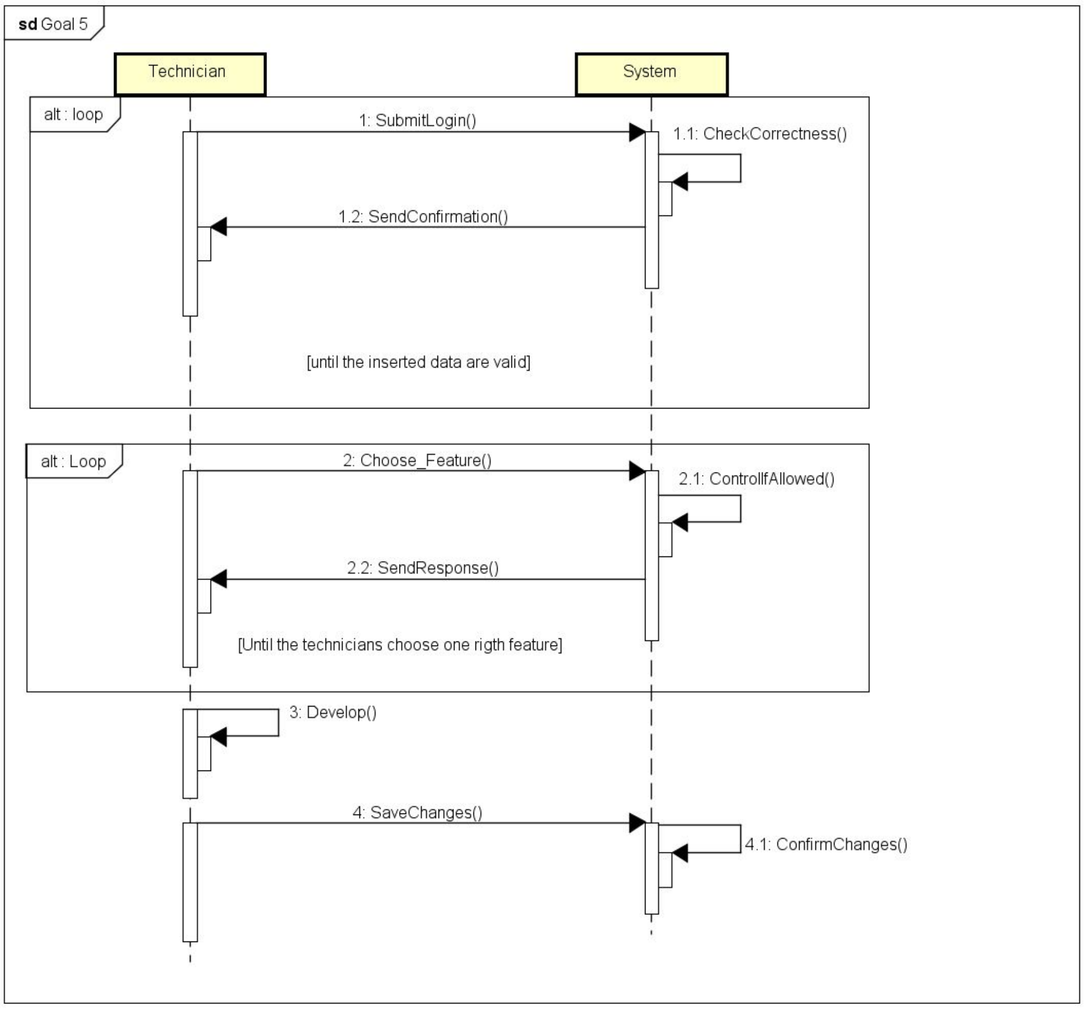



sd Goal 4
General User
System
Taxi Driver
alt : LOOP
1: FillRequestInformations()
2: SubmitRequest()
2.1: CheckCorrectness()
[Until all submitted informations are correct]
3: FindCorrectTaxiQueue()
alt : LOOP
4: SendRequetAtFirstFreeTaxi()
4.1: RequestComeback()
4.2: SendComeback()
5: CheckComeback()
[Until the system find a free taxi]
6: AllocateTaxi()
7: SendConfirmationMessage()
The user can be logged or not.He can send the request by homepage on webapplication or by mobile application

Goal 5

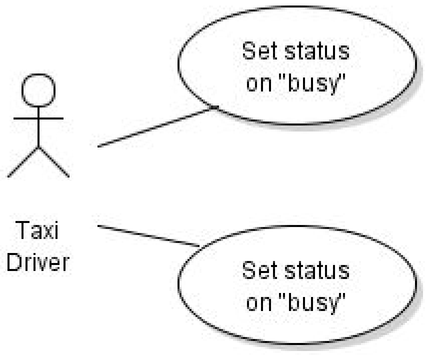
Actor	Technician
Goal	Allow a technicians to implement additional features
Input condition	new feature have to be implemented
event flow	1) The techncian logs in 2) The technician works on the feature and save changes
output condition	All the modifications are committed
Exceptions	If the technician insert a wrong password and Username, an error message is sent

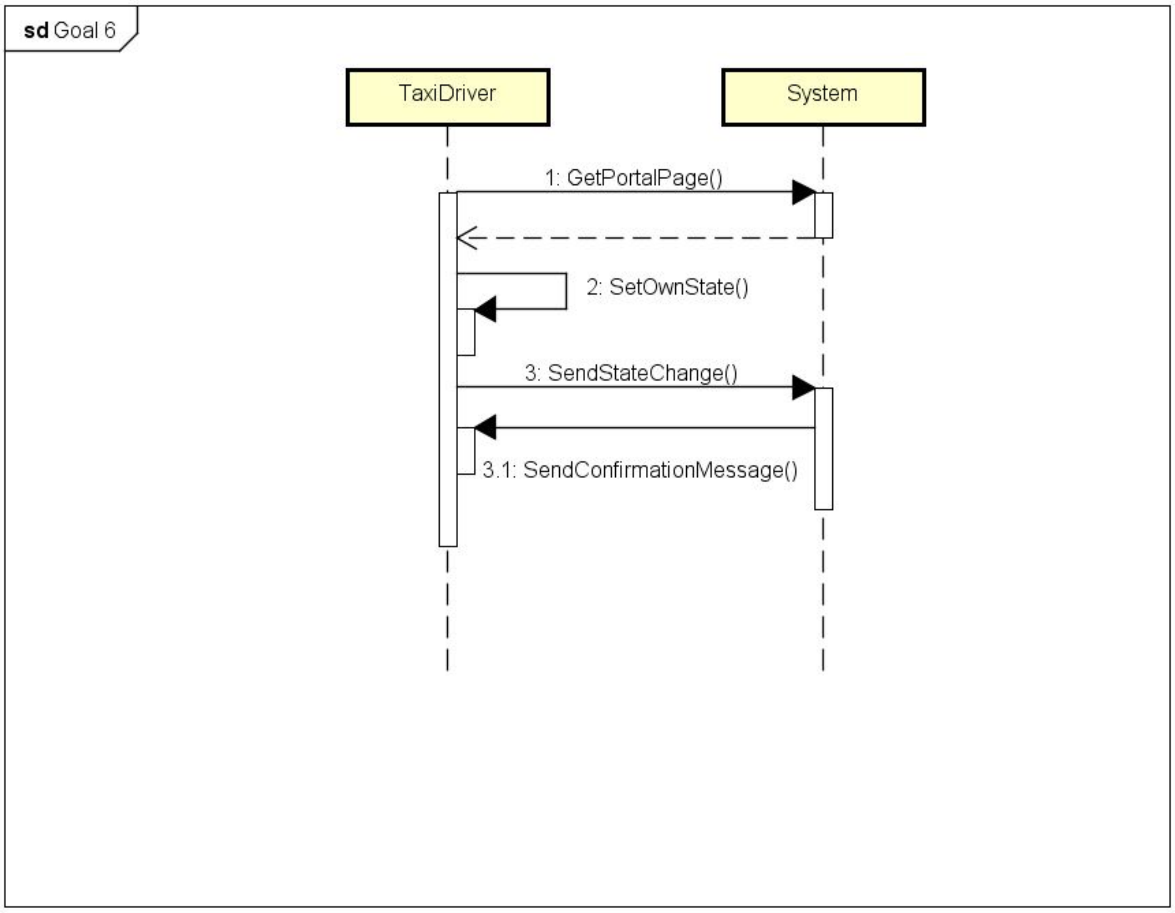




Goal 6

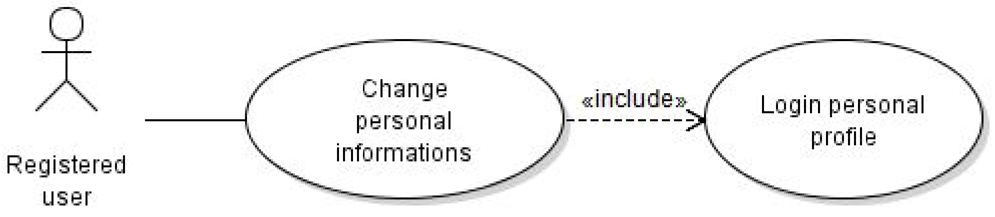
Actor	General user
Goal	Allow taxi driver to set his own state
Input condition	NULL
event flow	1) Taxi driver access his personal page 2) Taxi driver set his state
output condition	The status is changed
Exceptions	NULL

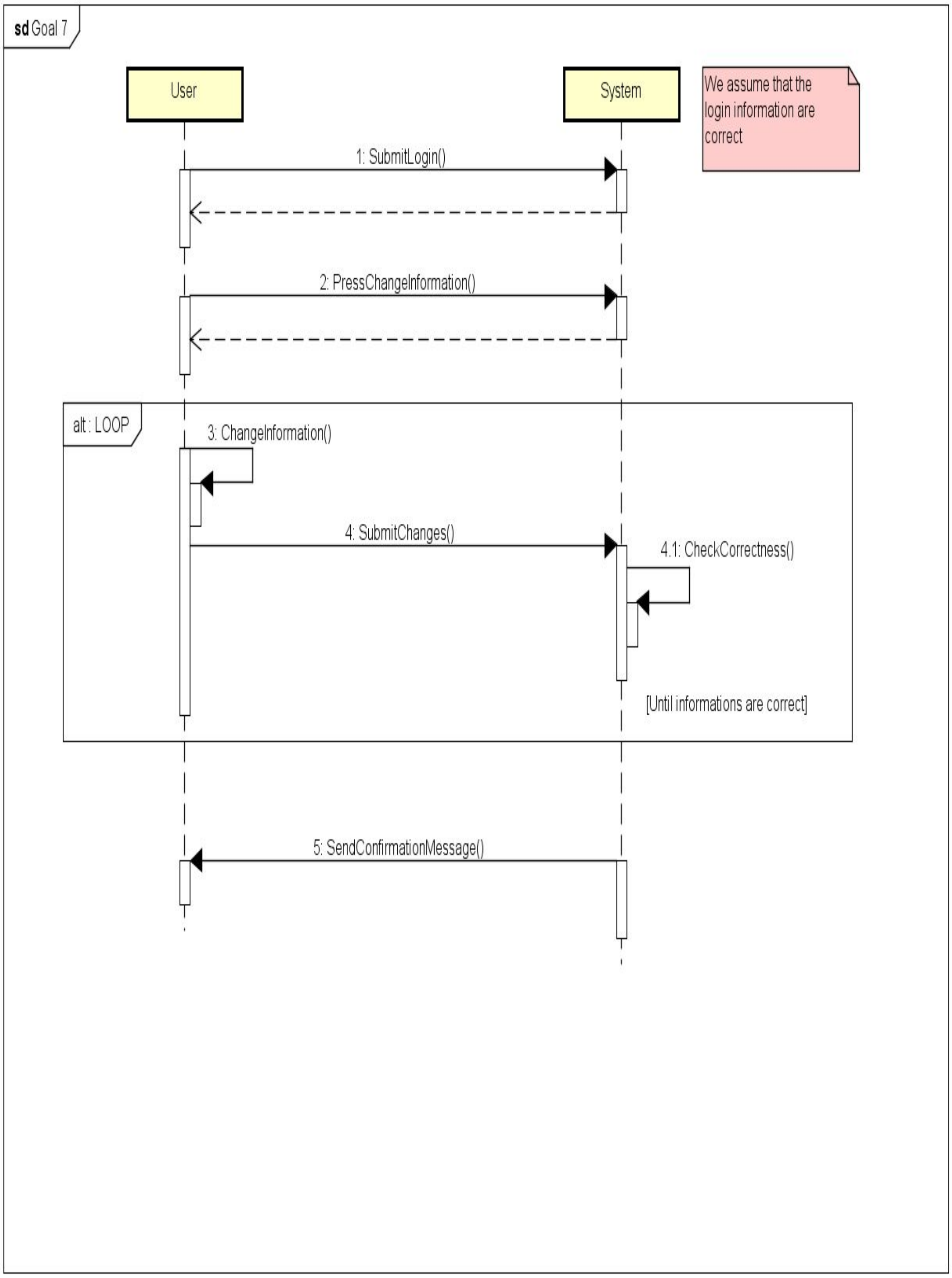




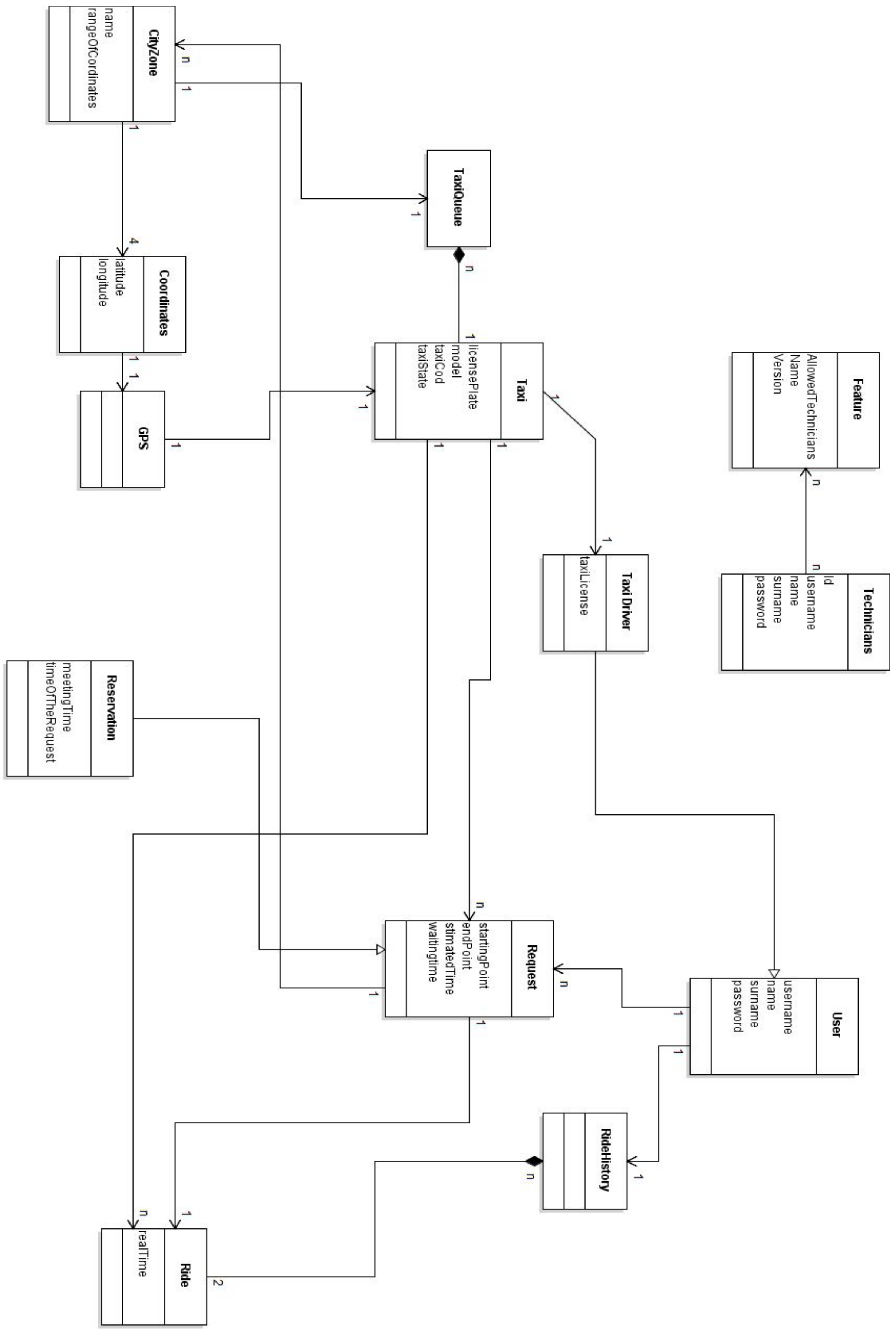
Goal 7

Actor	Registered User
Goal	Allow user to change his information
Input condition	NULL
event flow	1. The user log in the application 2. The user press the “change information” button 3. user fills the fields that want to change 4. user confirm the changes
output condition	user press the save changes button
Exceptions	1. if user isn’t registred must make a registration first 2. if user don’t fill correctly the fields must repeat the third point until all is corrected




sd Goal 7
User
System
We assume that the login information are correct
1: SubmitLogin()
2: PressChangeInformation()
alt : LOOP
3: ChangeInformation()
4: SubmitChanges()
4.1: CheckCorrectness()
[Until informations are correct]
5: SendConfirmationMessage()

3.4.1 Class Diagram (in this part we take care only about registered user)



3.5 Non Functional Requirements

3.5.1 Performance Requirements

There aren’t any kind of constraints concerning the performance.We assume the system is powerful enough to handle all the requests.

3.5.2 Design Constraints

No design constraint are required.

3.5.3 Software System Attributes

3.5.3.1 Availability

User can access to the application any time they need. The system use a dedicated server with high level of performances to guarantee this attribute.

3.5.3.2 Maintainability

For the future implementation a full documentation will be avaible.The documentation will contain a full history of all version of the sofware.At this moment there are a plans for future implementations like taxi sharing and we let tachnicians to be updated about any kind of news.

3.5.3.3 Portability

The application can be used on any OS concerning the web application. For the mobile app a non-obsolete smartphone is needed both for taxi driver and customer.

3.5.4 Security

3.5.4.1 External Interface Side

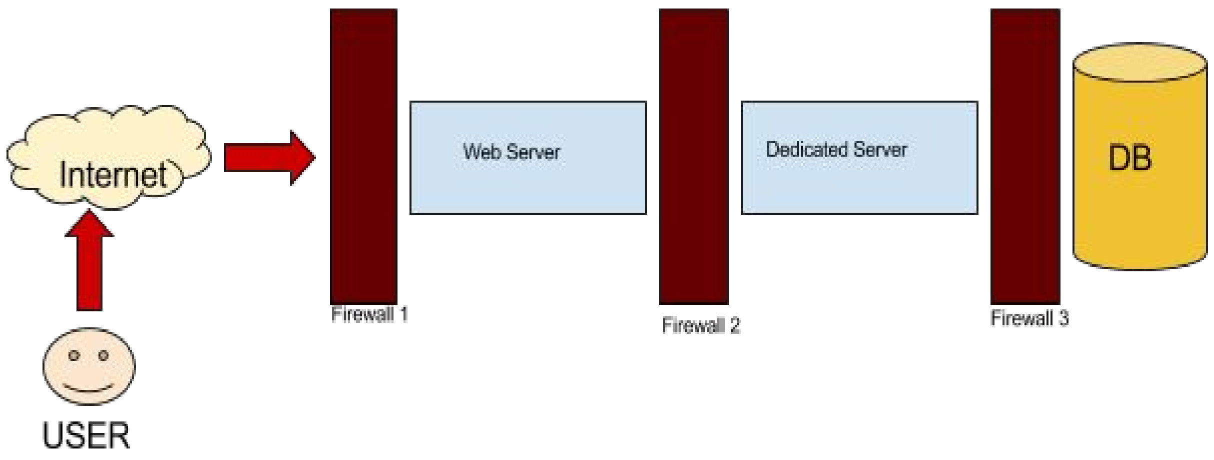
To ensure an availability of the service 24/7 er decide to introduce the basic forms of security: Username and password cannot contain some special characters (like parenthesis or <>); we will use a captcha code before sending a request to avoid DDoS attacks

3.5.4.2 Application Side

The taxi driver interface, to avoid MITM attacks, will be implemented with the SSL protocol

3.5.4.3 Server Side

A simple structure with firewalls with basic access rules between Web server, Application server, Database and the internet can be easily implemented; the Web Server will be placed in the Demilitarized Zone while the Database will be in the zone with the most strict access rules



4. Appendix

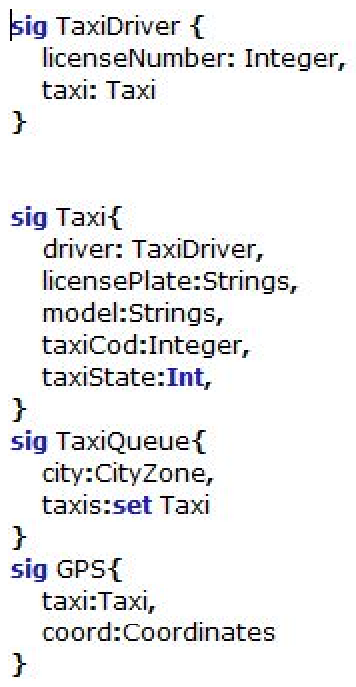
4.1 Alloy

The model we‘ve realized is based on the class diagram and the alloy file is available in the to be consulted for any reason (AlloyProjectPREDICATES.als).
For simplicity we have considered only the part concerning the registered user.

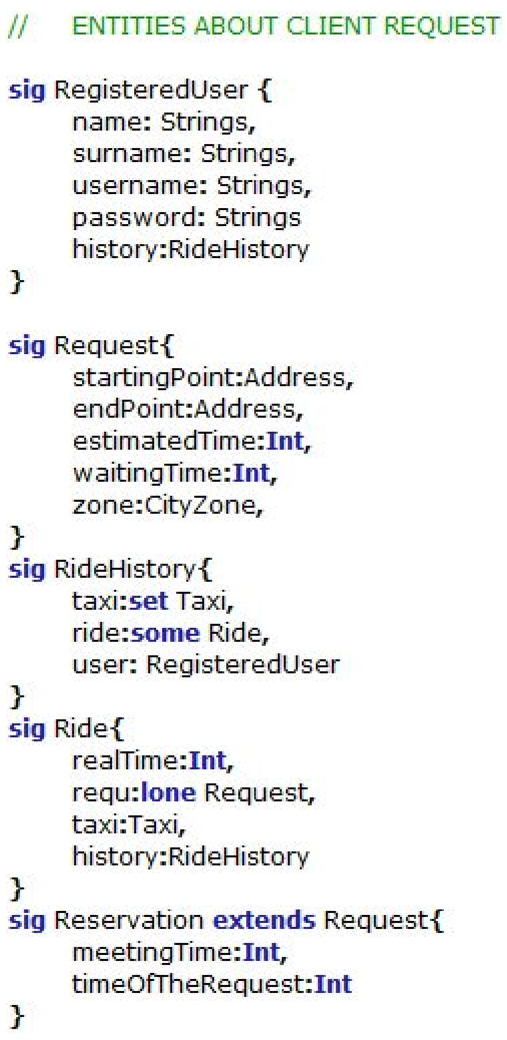
4.1.1 Signatures

This part contains all the signatures that represent the entities of our world:

These in particular are about the type of the entities,taxi and all other entities about a request of a ride.

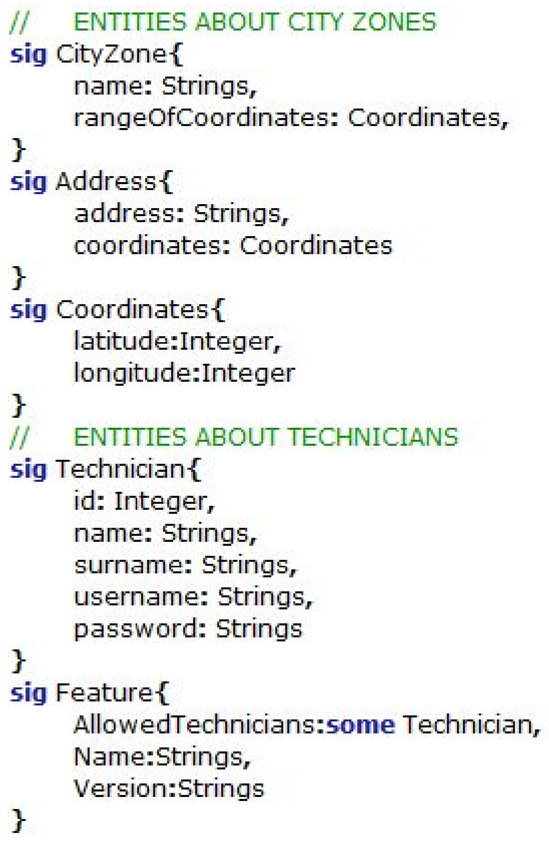

```
sig TaxiDriver {
   licenseNumber: Integer,
   taxi: Taxi
}


sig Taxi{
   driver: TaxiDriver,
   licensePlate:Strings,
   model:Strings,
   taxiCod:Integer,
   taxiState:Int,
}
sig TaxiQueue{
   city:CityZone,
   taxis:set Taxi
}
sig GPS{
   taxi:Taxi,
   coord:Coordinates
}
```

These entities describes the part of the registered users and all informations strictly correlated to them.



```
//     ENTITIES ABOUT CLIENT REQUEST

sig RegisteredUser {
        name: Strings,
        surname: Strings,
        username: Strings,
        password: Strings
        history:RideHistory
}

sig Request{
        startingPoint:Address,
        endPoint:Address,
        estimatedTime:Int,
        waitingTime:Int,
        zone:CityZone,
}
sig RideHistory{
        taxi:set Taxi,
        ride:some Ride,
        user: RegisteredUser
}
sig Ride{
        realTime:Int,
        requ:lone Request,
        taxi:Taxi,
        history:RideHistory
}
sig Reservation extends Request{
        meetingTime:Int,
        timeOfTheRequest:Int
}
```

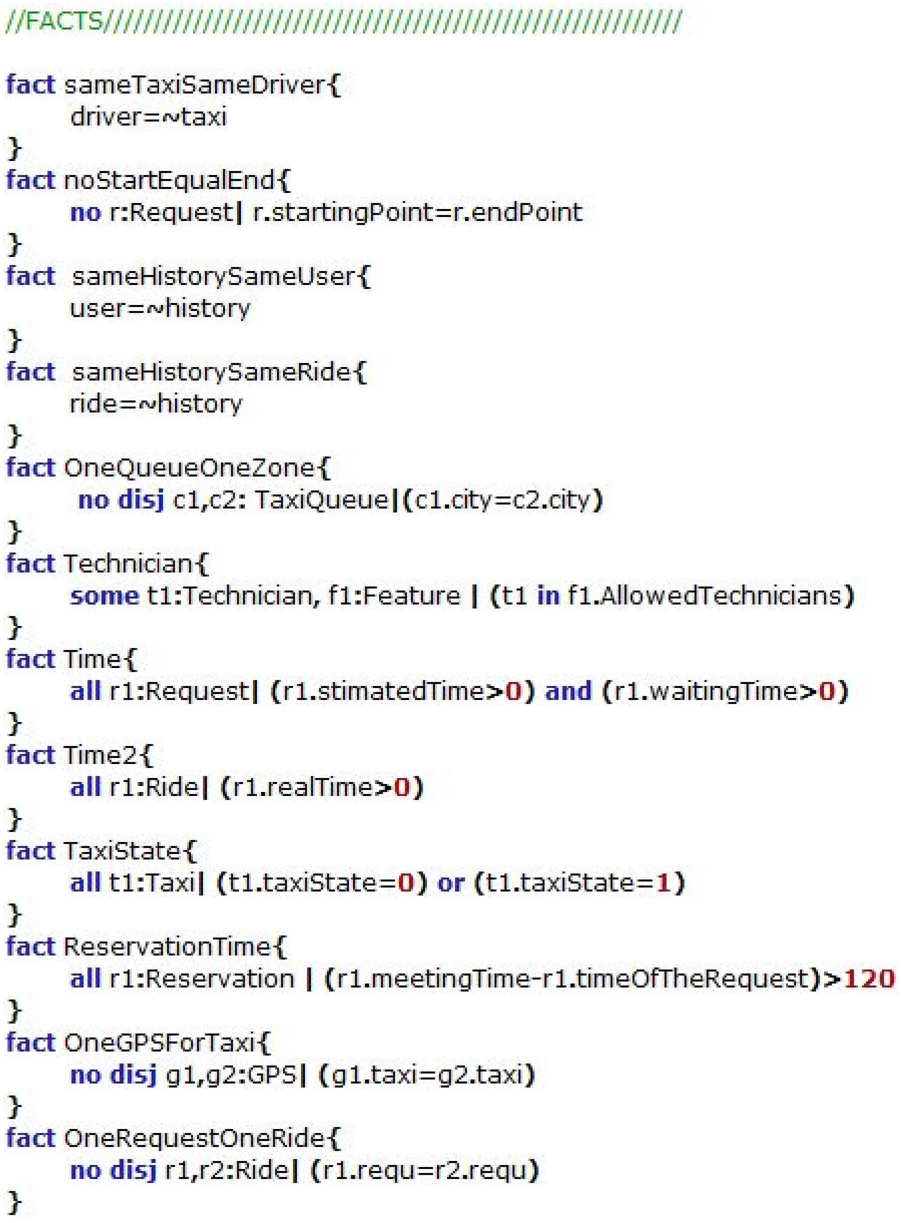
These last entities just concerne city zones and technicians



```
//      ENTITIES ABOUT CITY ZONES
sig CityZone{
        name: Strings,
        rangeOfCoordinates: Coordinates,
}
sig Address{
        address: Strings,
        coordinates: Coordinates
}
sig Coordinates{
        latitude:Integer,
        longitude:Integer
}
//      ENTITIES ABOUT TECHNICIANS
sig Technician{
        id: Integer,
        name: Strings,
        surname: Strings,
        username: Strings,
        password: Strings
}
sig Feature{
        AllowedTechnicians:some Technician,
        Name:Strings,
        Version:Strings
}
```

4.1.2 Facts

This part contains all the facts that are the constraint about relation between entities



```
//FACTS////////////////////////////////////////////////////////////

fact sameTaxiSameDriver{
	driver=~taxi
}
fact noStartEqualEnd{
	no r:Request| r.startingPoint=r.endPoint
}
fact  sameHistorySameUser{
	user=~history
}
fact  sameHistorySameRide{
	ride=~history
}
fact OneQueueOneZone{
	 no disj c1,c2: TaxiQueue|(c1.city=c2.city)
}
fact Technician{
	some t1:Technician, f1:Feature | (t1 in f1.AllowedTechnicians)
}
fact Time{
	all r1:Request| (r1.stimatedTime>0) and (r1.waitingTime>0)
}
fact Time2{
	all r1:Ride| (r1.realTime>0)
}
fact TaxiState{
	all t1:Taxi| (t1.taxiState=0) or (t1.taxiState=1)
}
fact ReservationTime{
	all r1:Reservation | (r1.meetingTime-r1.timeOfTheRequest)>120
}
fact OneGPSForTaxi{
	no disj g1,g2:GPS| (g1.taxi=g2.taxi)
}
fact OneRequestOneRide{
	no disj r1,r2:Ride| (r1.requ=r2.requ)
}
```

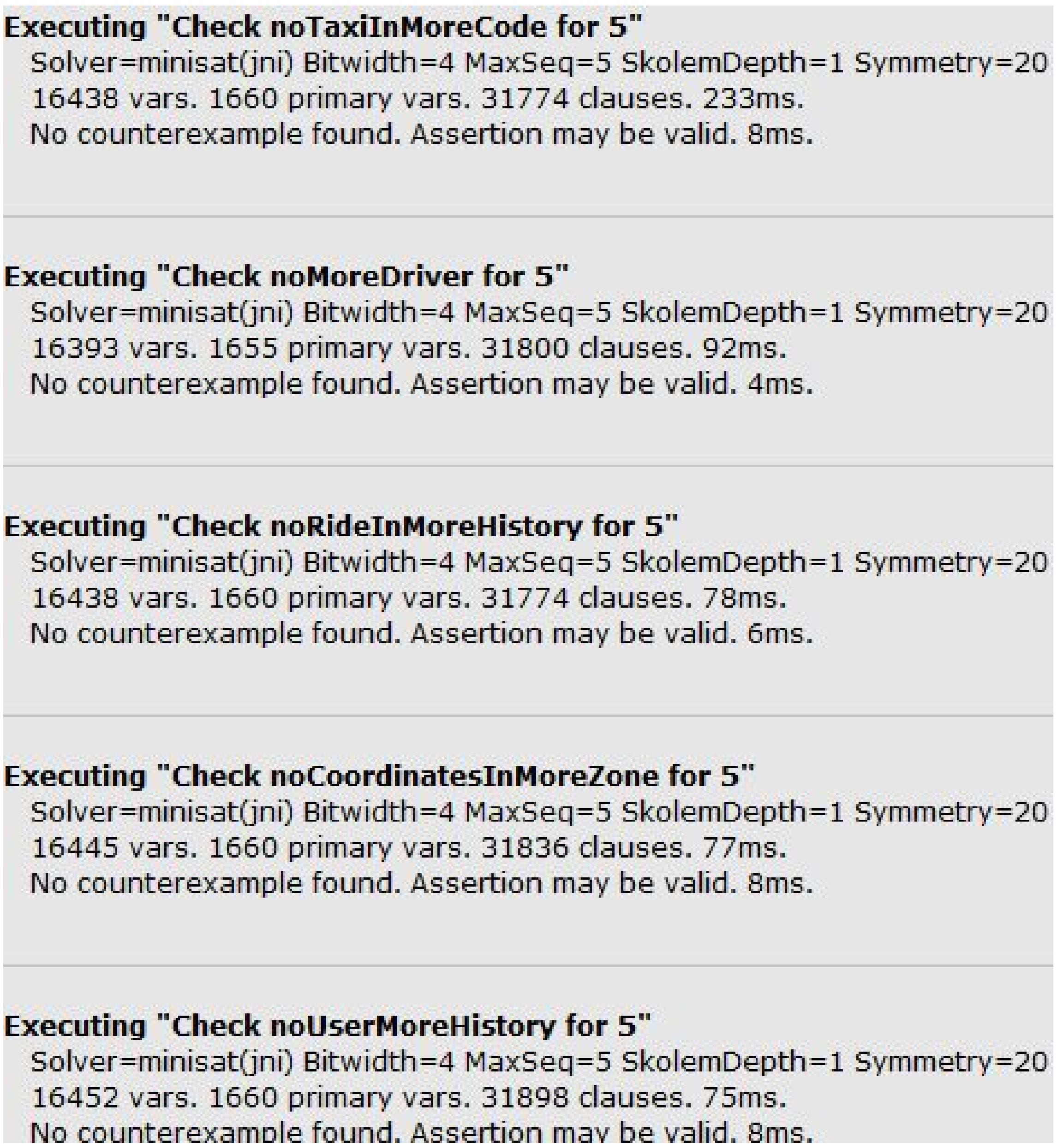
```
fact OneHistoryOneUser{
      no disj h1,h2:RideHistory| (h1.user=h2.user)
}
fact OneHistoryOneRide{
      no disj h1,h2:RideHistory| (h1.ride=h2.ride)
}
fact number{
      all r1:RideHistory|(#r1.taxi<=#r1.ride)
}
fact TaxiInQueue{
      all t1:Taxi|some c1:TaxiQueue | (t1 in c1.taxis)
}
fact TaxiInOneQueue{
      all q1,q2:TaxiQueue |all t1:Taxi |  ((t1 in q1.taxis)and (t1 in q2.taxis))implies(q1=q2)
}
fact oneCoordinateOneZone{
      all z1,z2:CityZone |all c1:Coordinates |  ((c1 = z1.rangeOfCoordinates)and (c1 in z2.rangeOfCoordinates))implies(z1=z2)
}
fact OneUserOneHistory{
      all u1,u2:RegisteredUser |all h1:RideHistory |  ((h1 = u1.history)and (h1 = u2.history))implies(u1=u2)
}
fact taxiMustHaveGPS{
      all t:Taxi|all g1,g2:GPS| ((t = g1.taxi)and (t = g2.taxi))implies(g1=g2)
}
```

4.1.3 Assertions

The assertions contains conditions to verify the validity of the facts after some operations

```
//ASSERTIONS/////////////////////////////////////////////////////////////////////////////////////////////////////

assert noTaxiInMoreCode{
      all q1,q2:TaxiQueue |all t1:Taxi |  ((t1 in q1.taxis)and (t1 in q2.taxis))implies(q1=q2)

}
//check noTaxiInMoreCode for 5

assert noMoreDriver{
      no disj d1,d2:TaxiDriver|d1.taxi=d2.taxi
}
//check noMoreDriver for 5

assert noRideInMoreHistory{
      all h1,h2:RideHistory |all r1:Ride |  ((r1 in h1.ride)and (r1 in h2.ride))implies(h1=h2)
}
//check noRideInMoreHistory for 5
assert noCoordinatesInMoreZone{
      all z1,z2:CityZone |all c1:Coordinates |  ((c1 = z1.rangeOfCoordinates)and (c1 in z2.rangeOfCoordinates))implies(z1=z2)
}
//check noCoordinatesInMoreZone for 5

assert noUserMoreHistory{
      all u1,u2:RegisteredUser |all h1:RideHistory |  ((h1 = u1.history)and (h1 = u2.history))implies(u1=u2)
}
check noUserMoreHistory for 5
```

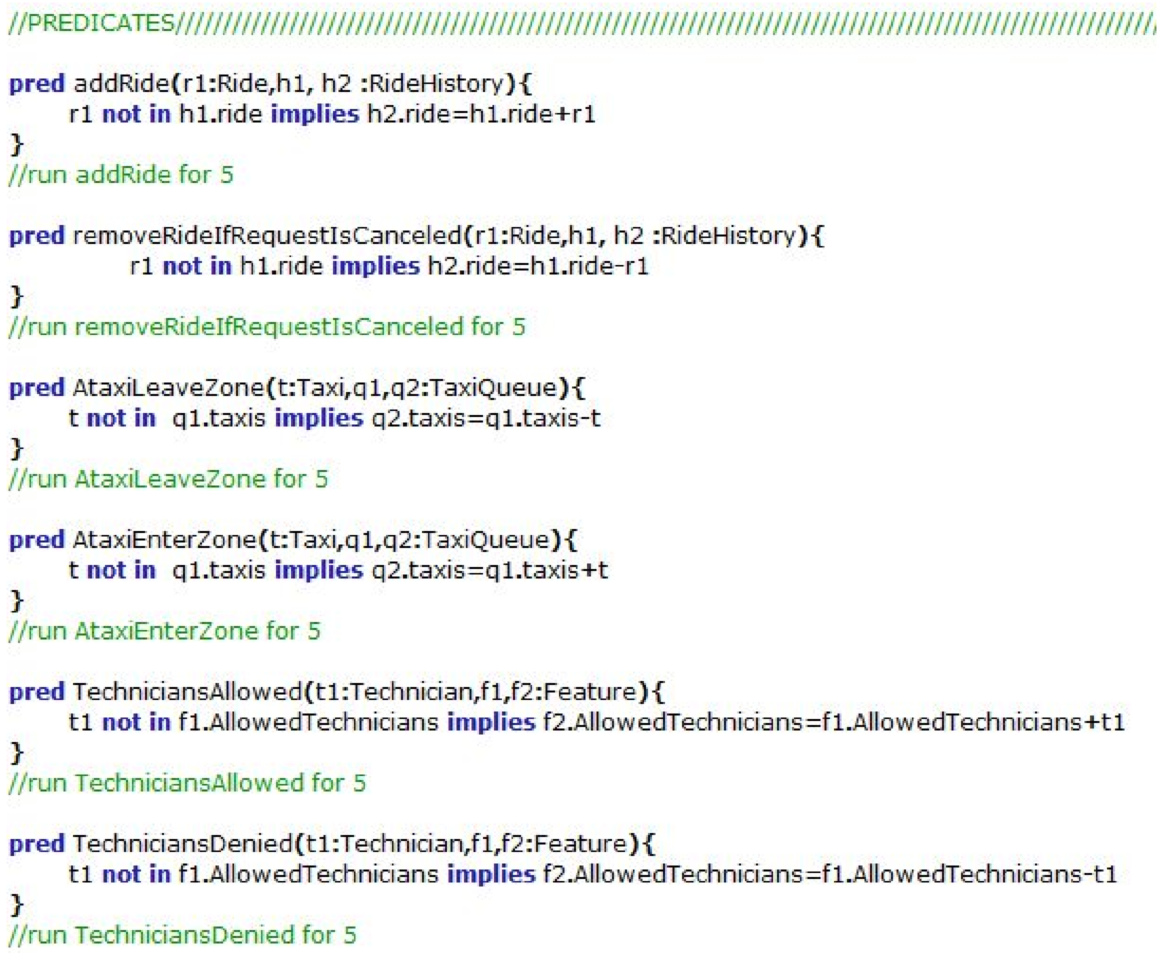
as you may notice, come checks are in comment: this is because the solver was not able to launch many checks at the same time



```
Executing "Check noTaxiInMoreCode for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16438 vars. 1660 primary vars. 31774 clauses. 233ms.
   No counterexample found. Assertion may be valid. 8ms.

Executing "Check noMoreDriver for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16393 vars. 1655 primary vars. 31800 clauses. 92ms.
   No counterexample found. Assertion may be valid. 4ms.

Executing "Check noRideInMoreHistory for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16438 vars. 1660 primary vars. 31774 clauses. 78ms.
   No counterexample found. Assertion may be valid. 6ms.

Executing "Check noCoordinatesInMoreZone for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16445 vars. 1660 primary vars. 31836 clauses. 77ms.
   No counterexample found. Assertion may be valid. 8ms.

Executing "Check noUserMoreHistory for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16452 vars. 1660 primary vars. 31898 clauses. 75ms.
   No counterexample found. Assertion may be valid. 8ms.
```

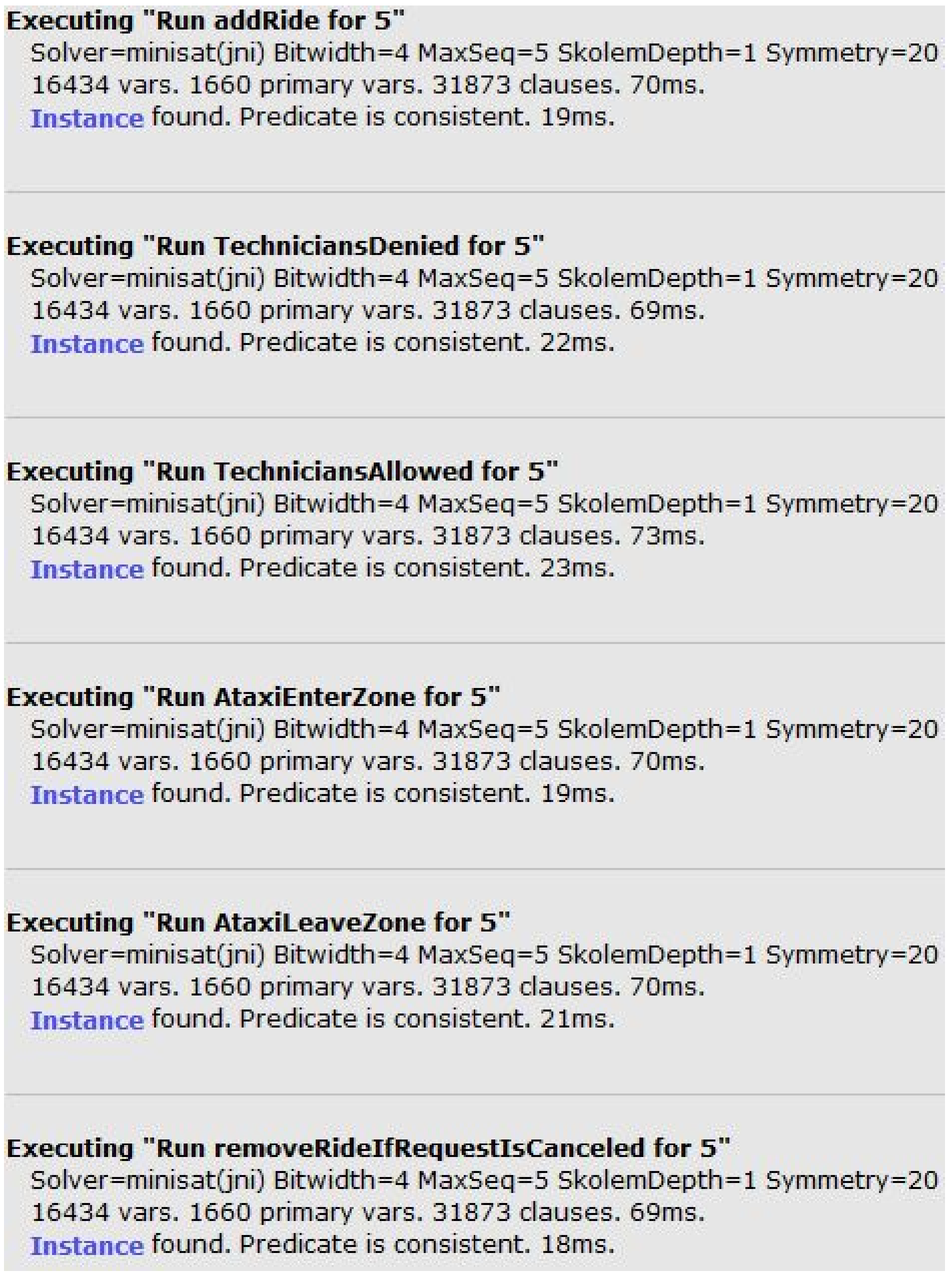
4.1.4 Predicates

This section contains all predicates to verify the model we have made



```
//PREDICATES/////////////////////////////////////////////////////////////////////////////////////////////////////

pred addRide(r1:Ride,h1, h2 :RideHistory){
	r1 not in h1.ride implies h2.ride=h1.ride+r1
}
//run addRide for 5

pred removeRideIfRequestIsCanceled(r1:Ride,h1, h2 :RideHistory){
		r1 not in h1.ride implies h2.ride=h1.ride-r1
}
//run removeRideIfRequestIsCanceled for 5

pred AtaxiLeaveZone(t:Taxi,q1,q2:TaxiQueue){
	t not in  q1.taxis implies q2.taxis=q1.taxis-t
}
//run AtaxiLeaveZone for 5

pred AtaxiEnterZone(t:Taxi,q1,q2:TaxiQueue){
	t not in  q1.taxis implies q2.taxis=q1.taxis+t
}
//run AtaxiEnterZone for 5

pred TechniciansAllowed(t1:Technician,f1,f2:Feature){
	t1 not in f1.AllowedTechnicians implies f2.AllowedTechnicians=f1.AllowedTechnicians+t1
}
//run TechniciansAllowed for 5

pred TechniciansDenied(t1:Technician,f1,f2:Feature){
	t1 not in f1.AllowedTechnicians implies f2.AllowedTechnicians=f1.AllowedTechnicians-t1
}
//run TechniciansDenied for 5
```

In the next page you can find the result of the execution of the predicates

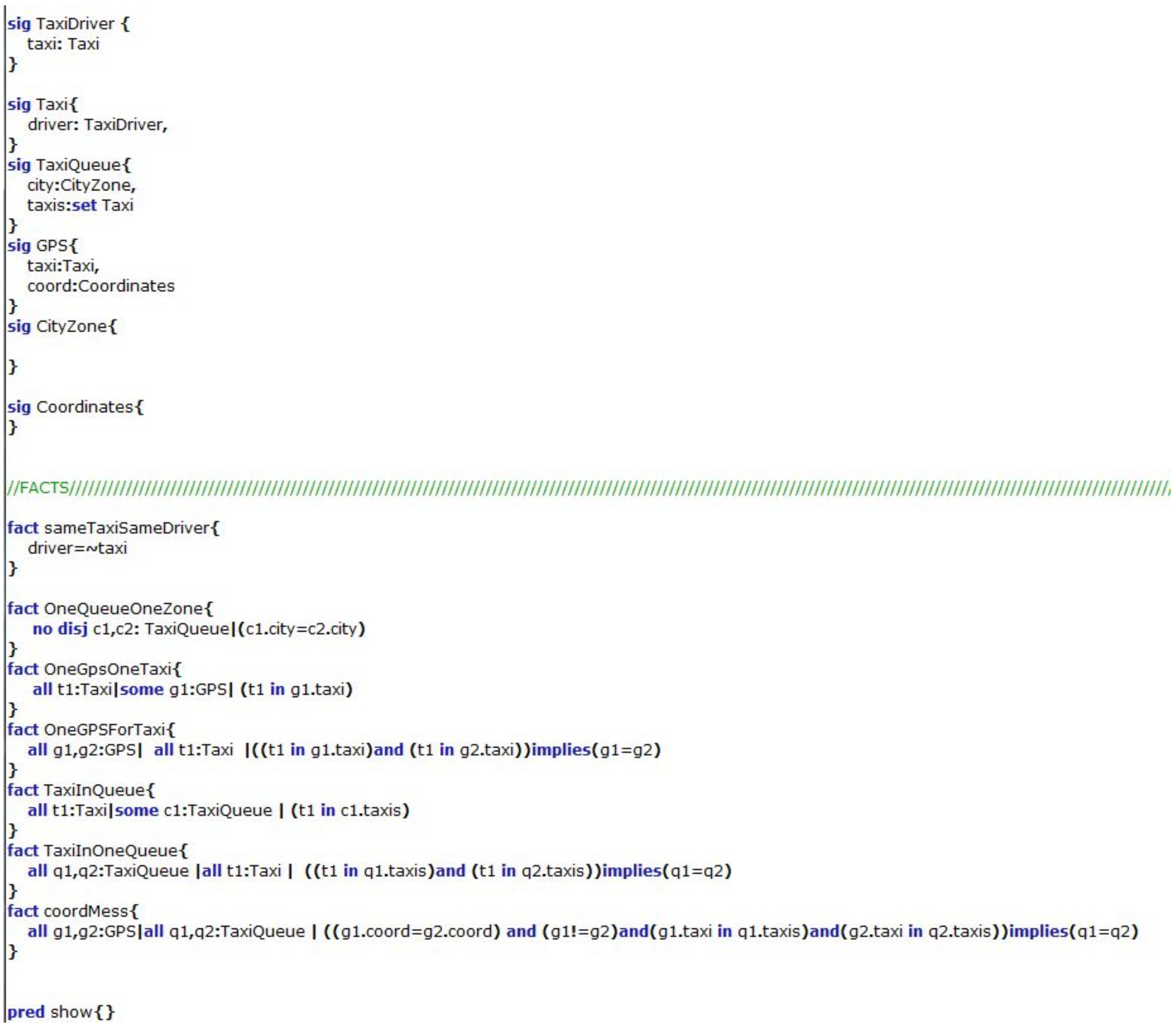


```
Executing "Run addRide for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16434 vars. 1660 primary vars. 31873 clauses. 70ms.
   Instance found. Predicate is consistent. 19ms.

Executing "Run TechniciansDenied for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16434 vars. 1660 primary vars. 31873 clauses. 69ms.
   Instance found. Predicate is consistent. 22ms.

Executing "Run TechniciansAllowed for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16434 vars. 1660 primary vars. 31873 clauses. 73ms.
   Instance found. Predicate is consistent. 23ms.

Executing "Run AtaxiEnterZone for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16434 vars. 1660 primary vars. 31873 clauses. 70ms.
   Instance found. Predicate is consistent. 19ms.

Executing "Run AtaxiLeaveZone for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16434 vars. 1660 primary vars. 31873 clauses. 70ms.
   Instance found. Predicate is consistent. 21ms.

Executing "Run removeRideIfRequestIsCanceled for 5"
   Solver=minisat(jni) Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16434 vars. 1660 primary vars. 31873 clauses. 69ms.
   Instance found. Predicate is consistent. 18ms.
```

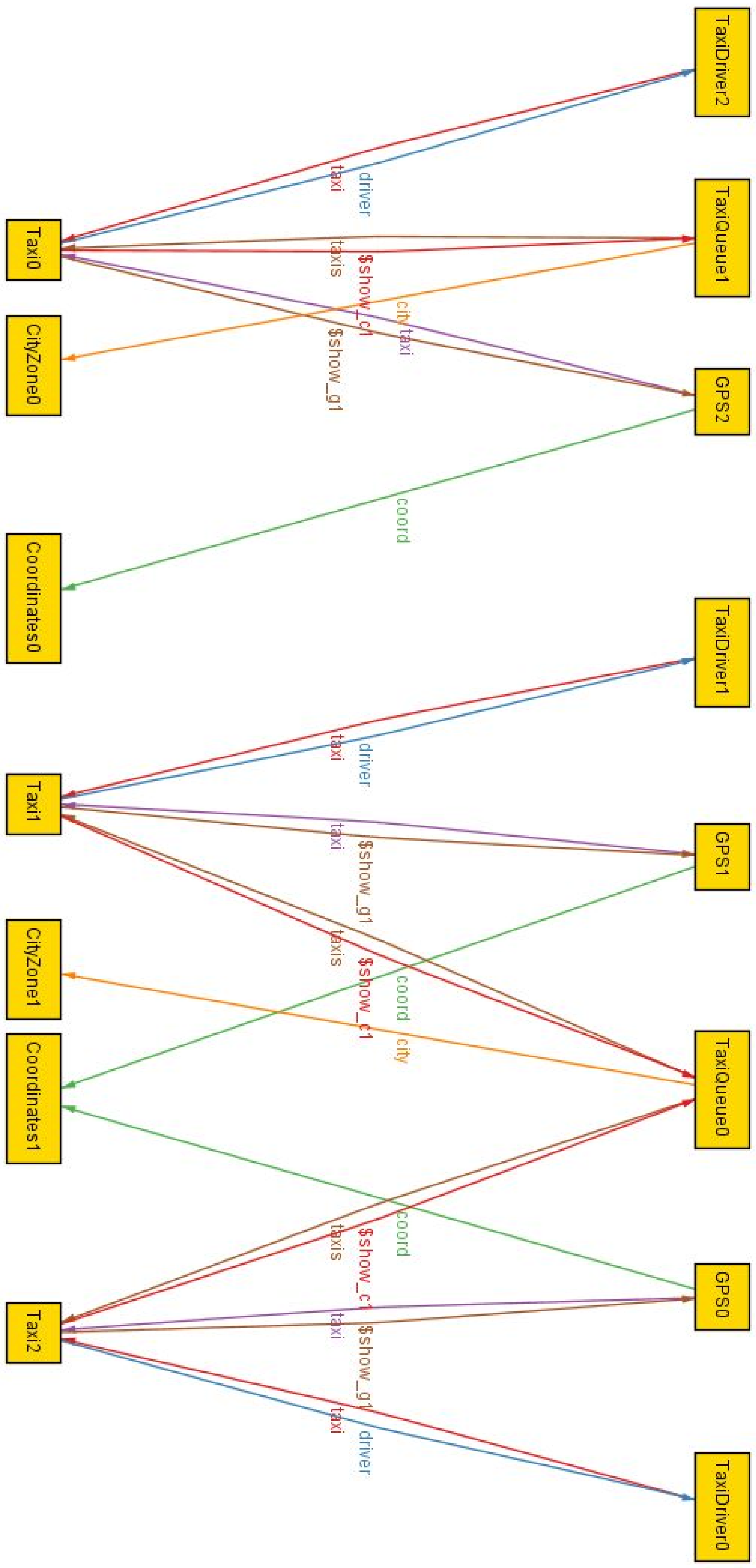
4.2 Generated Worlds

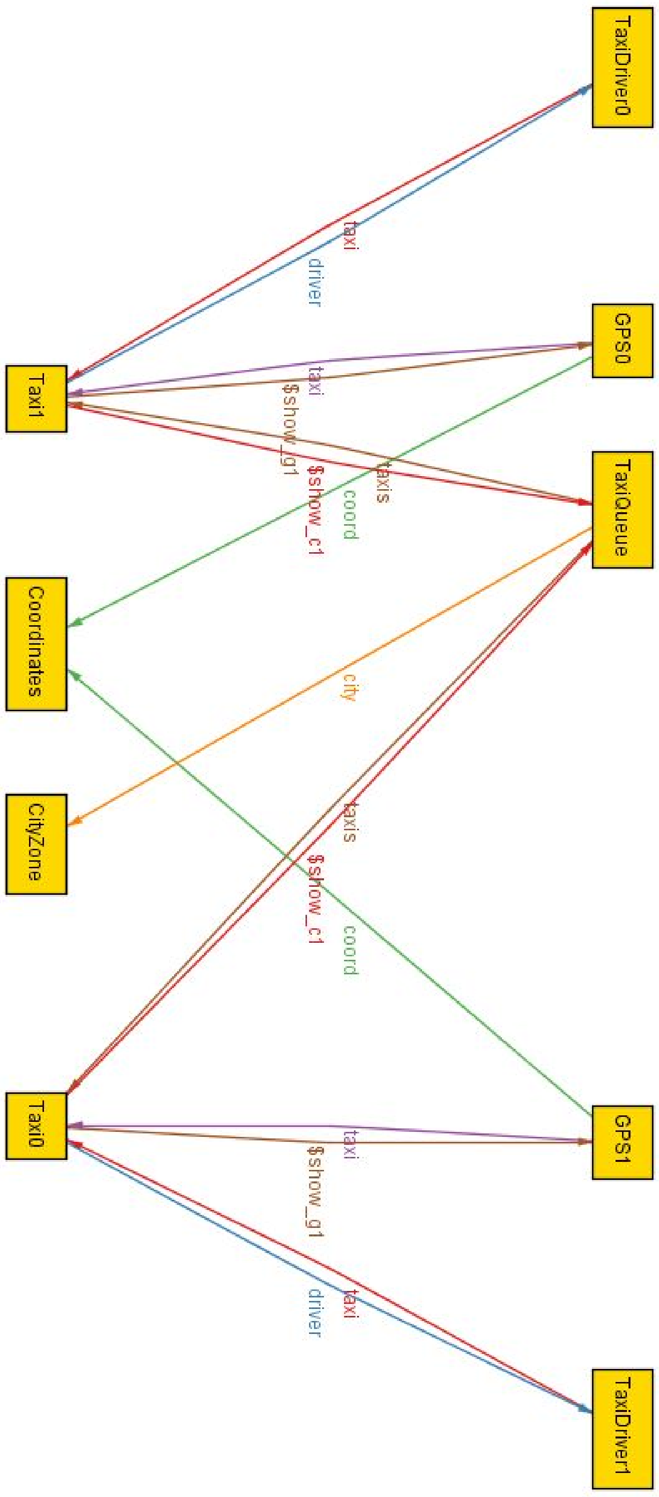
This section contains some generated world with alloy: due visibility reasons, we decide to split the whole world in sub-world according to the actor that are used and make some improvements.(example worlds of Taxi driver and technicians are different).This “sub-worlds” contain less information only for make the resulted diagrams more readable.

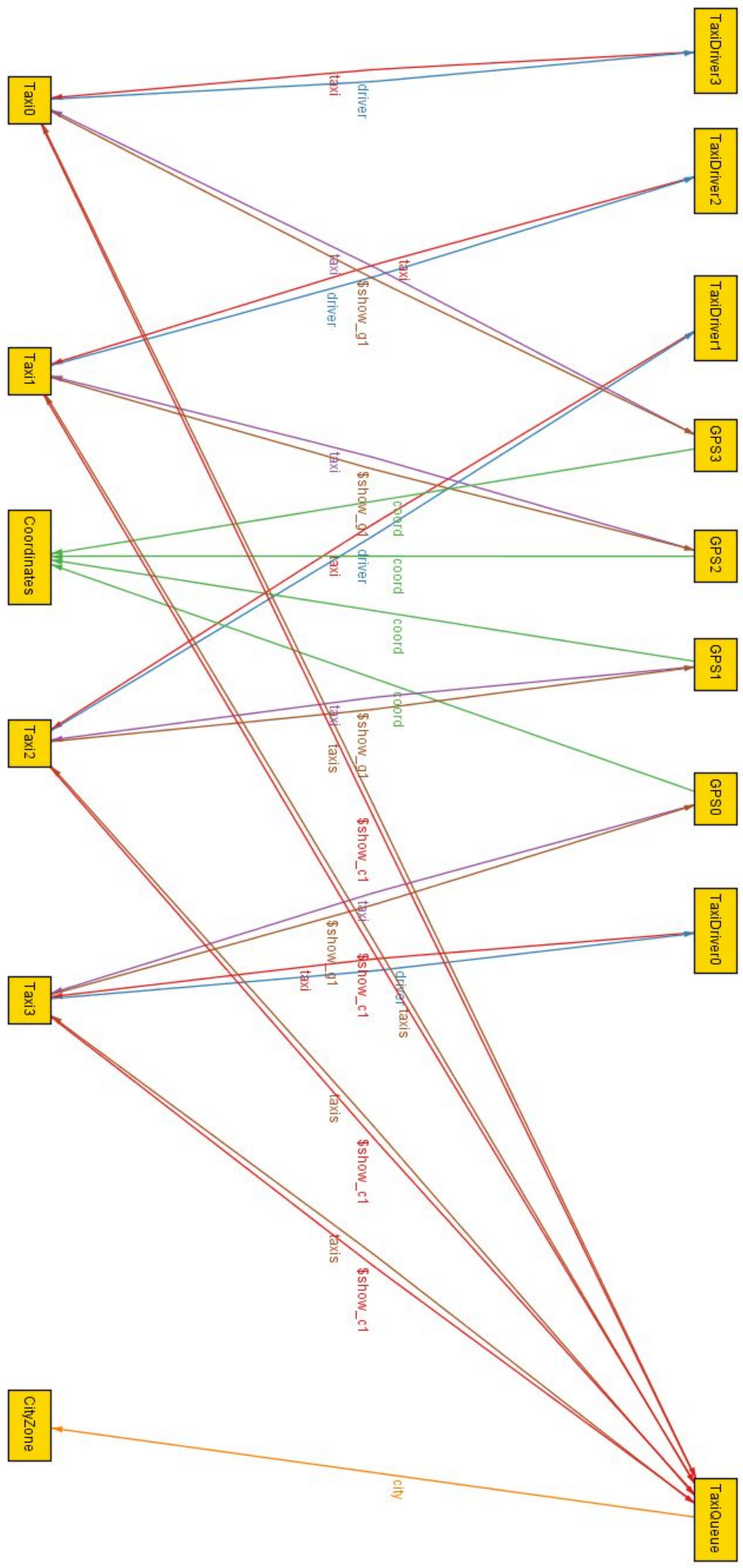
4.2.1 Taxi driver section



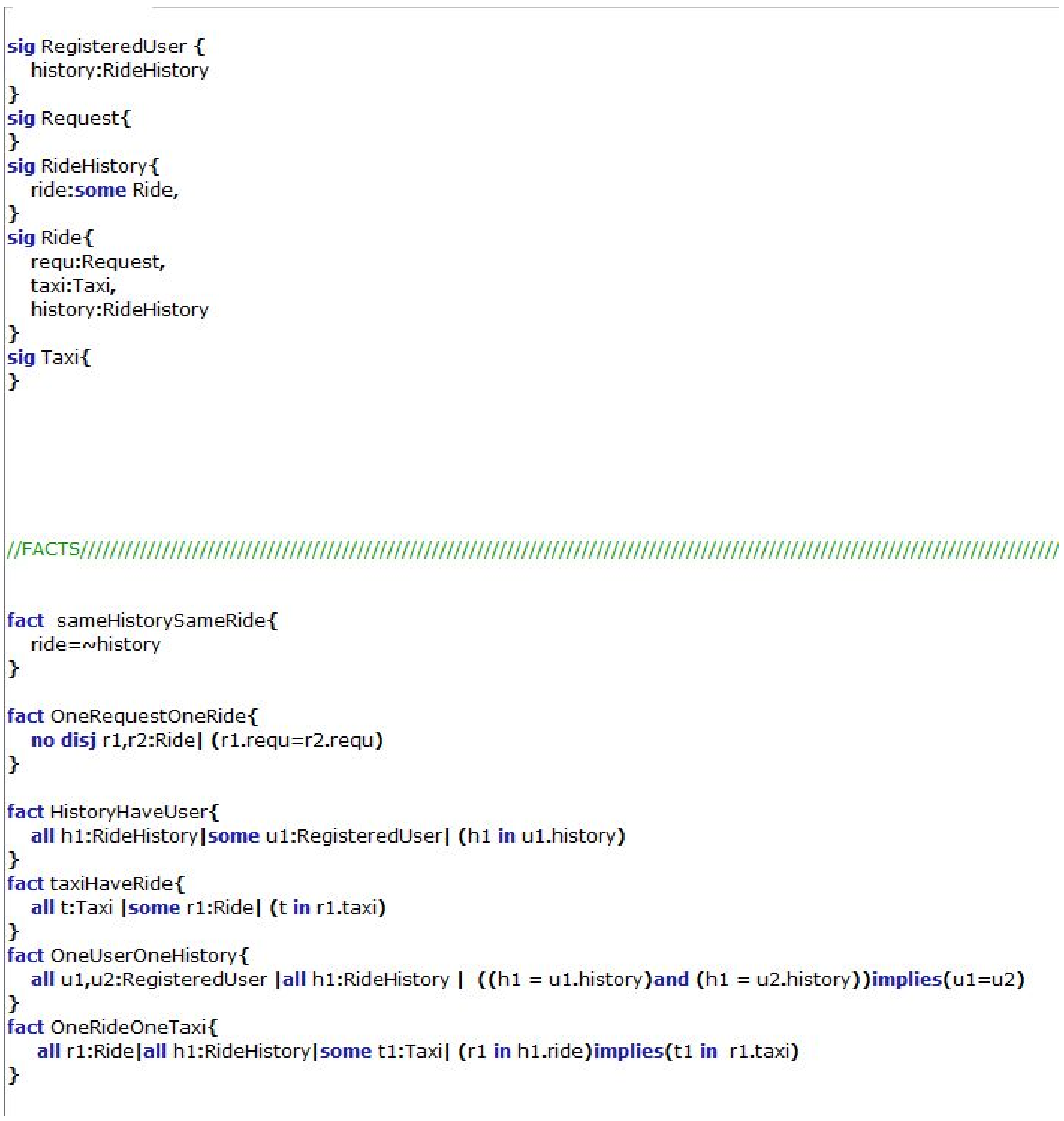
```
sig TaxiDriver {
   taxi: Taxi
}

sig Taxi{
   driver: TaxiDriver,
}
sig TaxiQueue{
   city:CityZone,
   taxis:set Taxi
}
sig GPS{
   taxi:Taxi,
   coord:Coordinates
}
sig CityZone{

}

sig Coordinates{
}


//FACTS/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fact sameTaxiSameDriver{
   driver=~taxi
}

fact OneQueueOneZone{
    no disj c1,c2: TaxiQueue|(c1.city=c2.city)
}
fact OneGpsOneTaxi{
    all t1:Taxi|some g1:GPS| (t1 in g1.taxi)
}
fact OneGPSForTaxi{
   all g1,g2:GPS|  all t1:Taxi  |((t1 in g1.taxi)and (t1 in g2.taxi))implies(g1=g2)
}
fact TaxiInQueue{
   all t1:Taxi|some c1:TaxiQueue | (t1 in c1.taxis)
}
fact TaxiInOneQueue{
   all q1,q2:TaxiQueue |all t1:Taxi |  ((t1 in q1.taxis)and (t1 in q2.taxis))implies(q1=q2)
}
fact coordMess{
   all g1,g2:GPS|all q1,q2:TaxiQueue | ((g1.coord=g2.coord) and (g1!=g2)and(g1.taxi in q1.taxis)and(g2.taxi in q2.taxis))implies(q1=q2)
}


pred show{}
```


TaxiDriver2
TaxiQueue1
GPS2
TaxiDriver1
GPS1
TaxiQueue0
GPS0
TaxiDriver0
Taxi0
CityZone0
Coordinates0
Taxi1
CityZone1
Coordinates1
Taxi2
driver
taxi
$show_c1
taxis
city
taxi
$show_g1
coord
driver
taxi
$show_g1
taxi
coord
$show_c1
taxis
city
coord
$show_c1
taxis
$show_g1
taxi
driver
taxi

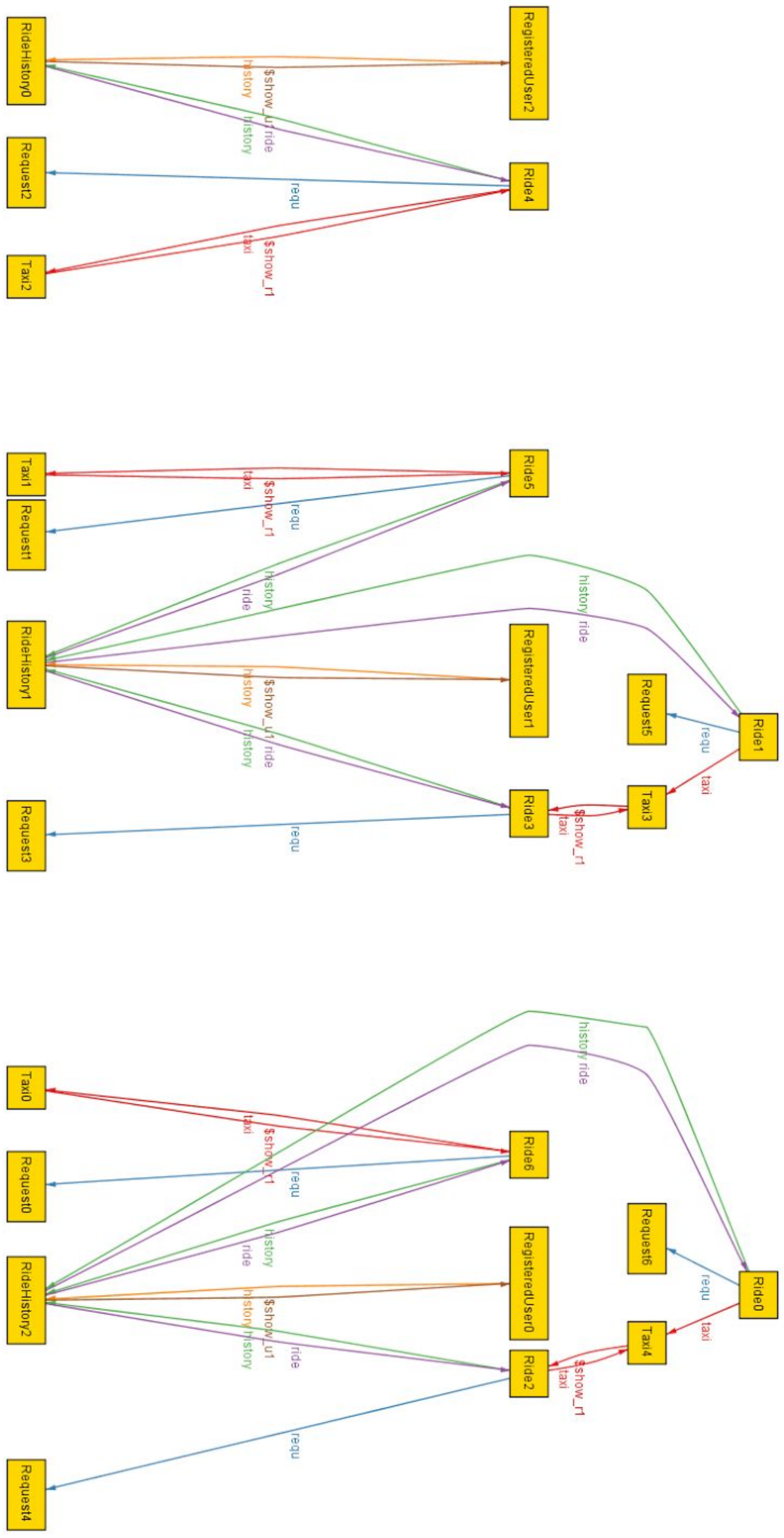

TaxiDriver0
GPS0
TaxiQueue
GPS1
TaxiDriver1
Taxi1
Coordinates
CityZone
Taxi0
taxi
driver
taxi
$show_g1
taxis
$show_c1
coord
city
taxis
$show_c1
coord
taxi
$show_g1
taxi
driver


TaxiDriver3
TaxiDriver2
TaxiDriver1
GPS3
GPS2
GPS1
GPS0
TaxiDriver0
TaxiQueue
Taxi0
Taxi1
Coordinates
Taxi2
Taxi3
CityZone
taxi
driver
$show_g1
coord
taxis
$show_c1
city

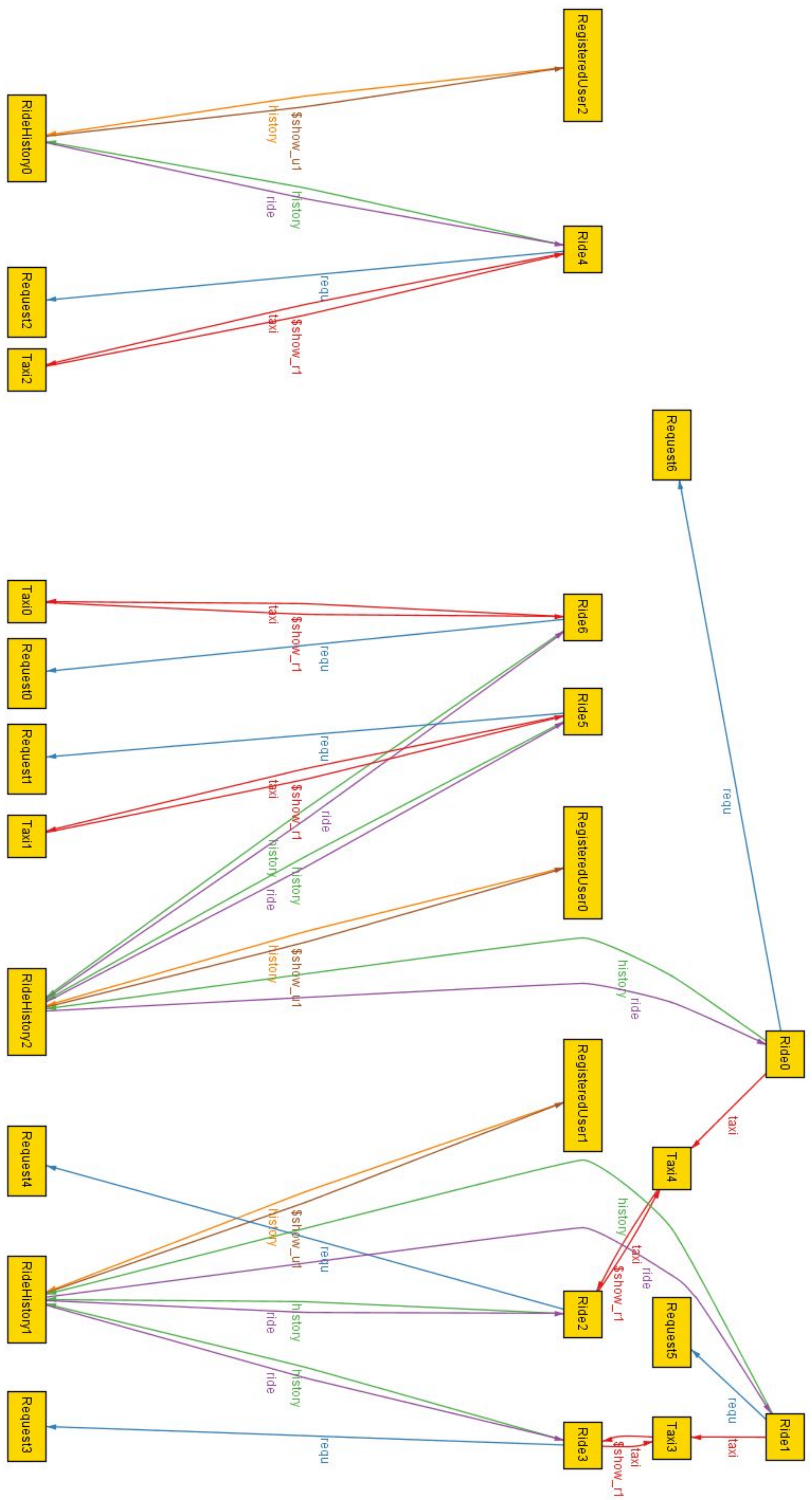
4.2.2 Client Section



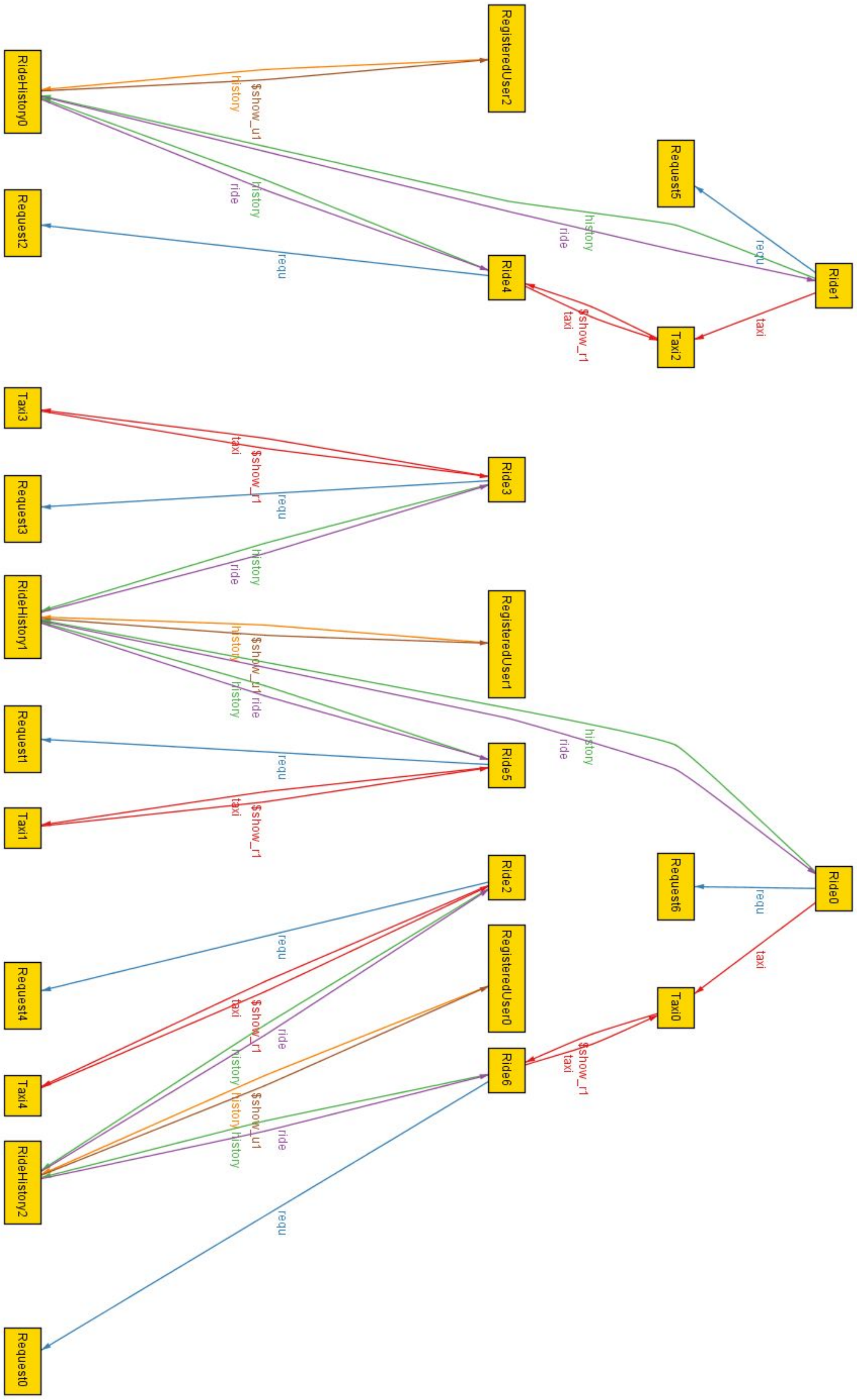
```
sig RegisteredUser {
   history:RideHistory
}
sig Request{
}
sig RideHistory{
   ride:some Ride,
}
sig Ride{
   requ:Request,
   taxi:Taxi,
   history:RideHistory
}
sig Taxi{
}




//FACTS//////////////////////////////////////////////////////////////////////////////////////////////////////////////


fact  sameHistorySameRide{
   ride=~history
}

fact OneRequestOneRide{
   no disj r1,r2:Ride| (r1.requ=r2.requ)
}

fact HistoryHaveUser{
   all h1:RideHistory|some u1:RegisteredUser| (h1 in u1.history)
}
fact taxiHaveRide{
   all t:Taxi |some r1:Ride| (t in r1.taxi)
}
fact OneUserOneHistory{
   all u1,u2:RegisteredUser |all h1:RideHistory |  ((h1 = u1.history)and (h1 = u2.history))implies(u1=u2)
}
fact OneRideOneTaxi{
    all r1:Ride|all h1:RideHistory|some t1:Taxi| (r1 in h1.ride)implies(t1 in  r1.taxi)
}
```



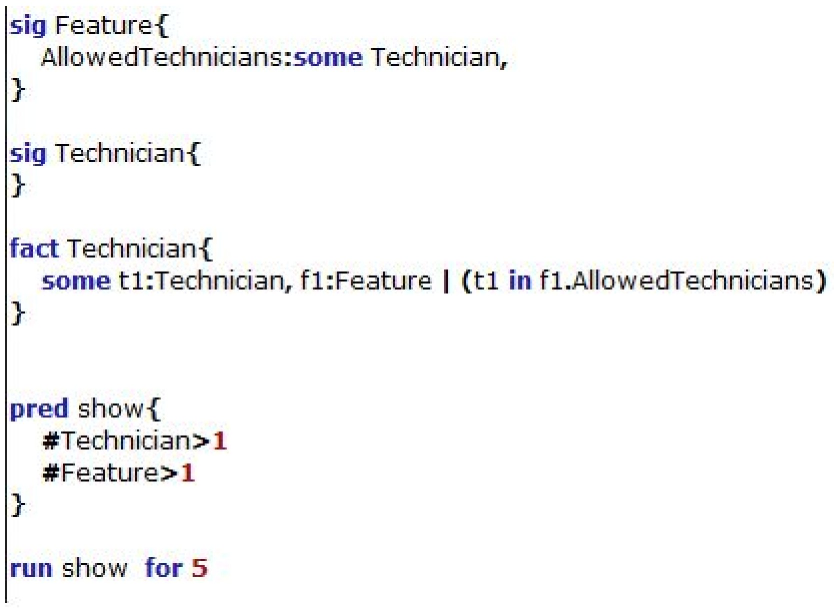
RegisteredUser2
RideHistory0
Request2
Taxi2
Ride4
$show_u
history
ride
history
requ
taxi
$show_r1
Taxi1
Request1
RideHistory1
Request3
Ride5
RegisteredUser1
Ride3
Request5
Taxi3
Ride1
taxi
$show_r1
requ
history
ride
history
ride
history
$show_u1
ride
history
requ
requ
taxi
$show_r1
taxi
Taxi0
Request0
RideHistory2
Request4
Ride6
RegisteredUser0
Ride2
Request6
Taxi4
Ride0
history
ride
taxi
$show_r1
requ
history
ride
history
$show_u1
history
ride
requ
$show_r1
taxi
requ
taxi



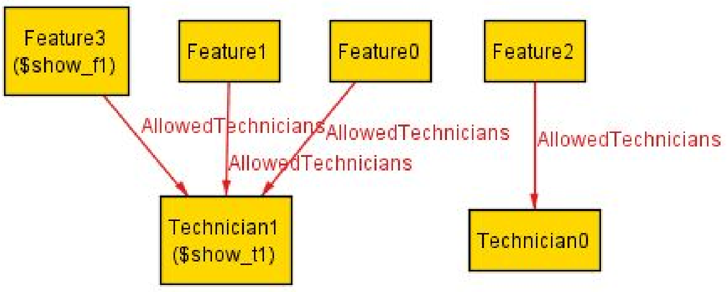
RegisteredUser2
RideHistory0
Ride4
Request2
Taxi2
Request6
Taxi0
Ride6
Request0
Ride5
Request1
Taxi1
RegisteredUser0
RideHistory2
Ride0
RegisteredUser1
Request4
Taxi4
RideHistory1
Ride2
Request5
Request3
Ride3
Taxi3
Ride1
history
ride
requ
taxi
$show_r1
$show_u1

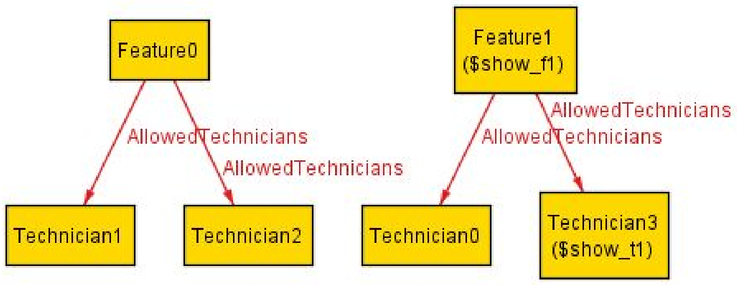

RideHistory0
RegisteredUser2
Request2
Request5
Ride4
Ride1
Taxi2
Taxi3
Ride3
Request3
RideHistory1
RegisteredUser1
Request1
Ride5
Taxi1
Ride2
Request6
Ride0
RegisteredUser0
Request4
Taxi0
Taxi4
Ride6
RideHistory2
Request0
history
ride
requ
taxi
$show_u1
$show_r1

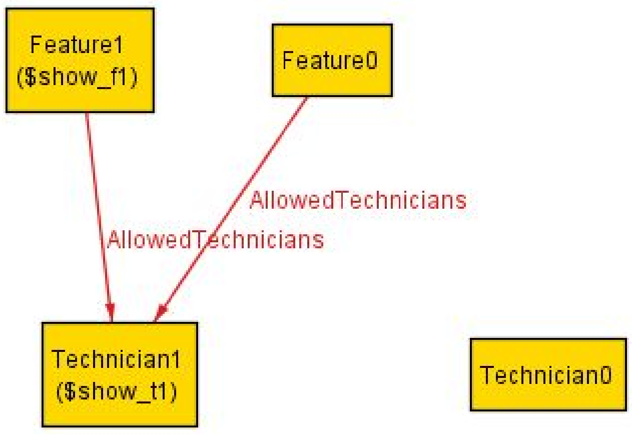
4.2.3 Technicians Section



```
sig Feature{
   AllowedTechnicians:some Technician,
}

sig Technician{
}

fact Technician{
   some t1:Technician, f1:Feature | (t1 in f1.AllowedTechnicians)
}


pred show{
   #Technician>1
   #Feature>1
}

run show  for 5
```


Feature3 ($show_f1)
Feature1
Feature0
Feature2
AllowedTechnicians
AllowedTechnicians
AllowedTechnicians
AllowedTechnicians
Technician1 ($show_t1)
Technician0


Feature0
Feature1 ($show_f1)
AllowedTechnicians
AllowedTechnicians
AllowedTechnicians
AllowedTechnicians
Technician1
Technician2
Technician0
Technician3 ($show_t1)



4.3 Software Used

- SourceTree-to work together with git
- Google Drive-to redact this document
- Violet-to create some uml diagrams
- Astah Professional -to create some uml diagrams
- Alloy Analizer-to check our models
- Balsamiq-to create the mock-ups

4.4 Hours of Work

- Salvatore Perillo   25h
- Claudio Sesto   25h