Requirement Analisys and Specification Document

Filippo Agalbato
850481

Andrea Cannizzo
790469

November 29, 2015

Contents

Chapter 1

Introduction

1.1 Purpose

This is the Requirement Analysis and Specification Document for the project software My Taxi Service. The goal of this document is to describe the system in terms of goals, requirements, assumptions and design, to define the different classes of users that will interact with it, and to analyze its most typical and most critical use cases.

This document is targeted to the customer's project managers, to evaluate the project specification from a high-level point of view, and to the intended designers, developers, programmers, analysts and testers, to build the actual software and to maintain, integrate and expand it in the future.

1.2 Scope

The aim of this project is to build a software system able to manage and optimize taxi services in a large city, targeted both to the city public transportation management council and to the citizens as customers of the taxi service. The system will allow passengers to use the service in a smart and accessible way, and at the same time will offer a better service thanks to an enhanced management of taxi resources and allocation.

In particular, passengers and taxi drivers will be registered and remembered by the system. Passengers will be able to request a taxi by using the application and taxi drivers will be notified of these requests by the system, that will divide the city in several taxi zones each with its own queue of taxis. Taxi drivers notified of passengers requests can then accept them.

1.3 Definitions, acronyms and abbreviations

- Passenger: A citizen recognized by the system as a registered user of My Taxi Service, using the system as a customer does, to call taxis and be served;
- Logged passenger: A passenger that is currently and correctly logged into the system;

- Taxi driver: A citizen recognized by the system as a registered user of My Taxi Service, using the system as a taxi driver, recognized by the parent company and of verified identity;
- Logged taxi driver: A taxi driver that is currently and correctly logged into the system;
- Request for service: A passenger uses the mobile app or web application to send the central management system a formal and well-formed request to be served by a taxi. This request includes a departure time, an itinerary starting point and ending point and the number of people to be served.
- Real time request: A request for service that is intended to serve a passenger immediately and is instantly relayed by the system to an available taxi driver;
- Reservation: A request for service that is not meant to serve a passenger immediately and as such is stored by the system and only relayed to an avaialble taxi driver once the correct time comes;

1.4 References

- Specification Document: My Taxi Service project specification for the Academic Year 2015-16 – "Assignments 1 and 2 (RASD and DD).pdf".
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

1.5 Overview

This document is structured as follows:

- Introduction: A declaration of the scope, intent and purpose of this document;
- Overall description: A high level description of the system, with focus on the goals it will have to meet and on the assumptions that have been held as true during this analisys;
- Specific requirements: A more detailed look on how the system will need to be implemented, while retaining a sufficiently abstract point of view;
- System model: A varied representation of the system via its main scenarios identification, an exhaustive listing of UML models, and a consistency analysis made in the Alloy language.

Chapter 2

Overall description

2.1 Product perspective

My Taxi Service as a software system will take the form of a web application running on a central server with access either through a standard browser or a specific mobile application (for passengers) or simply a mobile application (for taxi drivers) that will allow bidirectional communication (for which a standard HTTPS protocol will be used). Taxi drivers will need to own a smartphone with internet access. There is no software system already in place with which My Taxi Service will need to be integrated, as this is the case of a greenfield development to modernize a system where everything was previously done "by hand". A programmatic API will be part of the system, to allow for future expansion of its capabilities, such as taxi sharing management. No other specific or apparent constraints on existing hardware and software hold, as long as the system is able to support multiple connections and handle different requests for service at a time.

2.2 Product functions

My Taxi Service is required to meet the following project goals:

1. Taxis in the city will be divided in queues, each one of which will be fairly managed.
2. Passengers will be registered and remembered by the system;
3. Passengers will be able to log in to their personal account;
4. Passengers will be able to request a taxi at their current position or at a position of their choice;
5. Passengers will be able to reserve a taxi in advance;
6. Passengers will be able to access their reservation history;
7. Passengers will be able to modify or delete elements from their reservation history;

8. Passengers will be notified if their request for service is accepted and the estimated time of arrival of the incoming vehicle;
9. Passengers will be notified if their request for service cannot be met for whatever reason;
10. Taxi drivers will be registered and remembered by the system;
11. Taxi drivers will be able to log in to their personal account;
12. Taxi drivers will be able to inform the system of their availability;
13. Available taxi drivers will be notified when they are called to answer a request for service;
14. Taxi drivers will be able to confirm that they are going to take care of the assigned request for service;
15. Available taxi drivers will be notified that they need to move to a different city zone to ensure a total coverage of possible passengers requests.

2.3 User characteristics

My Taxi Service is intended for two different classes of users: passengers and taxi drivers. As such, it will also offer two different and distinct views and user interfaces.

Passengers will require no special knowledge of the application to use it, for the user interface must be designed to be as intuitive and easy-to-use as possible, in such a way that even a child could see it for the first time and understand how it works.

Taxi drivers will have access to different functions, but the core principle should well remain the same: there is no requirement for in-depth training to use the system as an end-user (there should be none at all), for the application will simply relay informations and orders to and from the taxi driver and the system. Being a web-based application, passengers will of couse need internet access to use these services; taxi drivers, on the other hand, are required to own a smartphone with readily available and abundant internet access and, of course, the application installed.

2.4 Constraints

2.4.1 Regulatory policies

My Taxi Service will need to meet any and all requirements imposed by the laws and policies of the city in which it will be deployed, with special regards to the rules concerning taxi services. These will need to be studied in depth, with possible help from the customer's part, and integrated as needed into the management system.

2.4.2 Parallel operation

My Taxi Service central system will need to constantly monitor the situation of taxis in the whole city in order to dispatch and manage them as best as possibile. At the same time, it will need to be sized appropriately to support the many different connections and operations from clients to the central server, both passengers and taxi drivers.

2.5 Assumptions and dependencies

1. The city will be divided in several zones;
2. The system will minimize the risk of not being able to answer to a request for service due to lack of available taxis;
3. Users profiles are private and hidden;
4. The mobile app will be available for each major mobile operating system and will forward data to the central system in a uniformed way;
5. All taxi drivers own a suitable smartphone;
6. All taxi drivers *must* own the My Taxi Service app and be registered users in order to work;
7. Taxis will be tracked by a GPS system;
8. The system will receive GPS readings from each taxi and will keep careful track of it;
9. The system knows the license plates of all taxi drivers' cars and their associations with the corresponding GPS tracker mounted on the vehicle.

Chapter 3

Specific requirements

3.1 Functional requirements

This section refers to system goals (specified in section 2.2). The goal referred to in the requirement description is the one it derives from.

1. The system must balance taxi drivers workload by assigning each one to a *first-in, first-out* queque in a city zone (Goal 1);
2. The system must provide a sign up function for passengers (Goal 2);
3. The system must store all required passenger information (Goal 2);
4. The system must provide a log in function to access all passenger features (Goal 3);
5. The system must provide a function that allows logged passengers to require a taxi (Goal 4);
6. The system must provide a form in which the logged passenger will be able to add trip information for a request for service (starting point, destination point, number of passengers) (Goal 4);
7. The system must retrieve GPS information from the logged passenger's mobile phone (Goal 4);
8. The system must provide a reservation function (Goal 5);
9. The system must provide a form in which the logged passenger will be able to add trip information for a reservation (starting point, destination point, number of passengers, leaving time) (Goal 5);
10. The system must finalise a reservation two hours before its requested time, making it unchangeable (Goal 5);
11. The system must store a reservation history for each passenger (Goal 6);
12. The system must provide a function which shows reservation history (Goal 6);
13. The system must provide a function to allow logged passengers to modify a reservation up to two hours before the requested time (Goal 7);

14. The system must provide a function to allow logged passengers to delete a reservation up to two hours before the requested time (Goal 7);
15. The system must analyse a newly-made real time request and send a confirmation with the estimated waiting time if it can be fulfilled (Goal 8);
16. The system must analyse a newly-made reservation request and send a confirmation if it can be fulfilled (Goal 8);
17. The system must notify passenger if his reservation cannot be met (Goal 9);
18. The system must delete all reservations in a passenger's history that cannot be met after sending him notice (Goal 9);
19. The system must provide a sign up function for taxi drivers (Goal 10);
20. The system must store all required taxi driver information (Goal 10);
21. The system must check that all taxi drivers trying to sign up own both a valid taxi driver's license and a proper driver's license (Goal 10);
22. The system must check that the information needed for a taxi driver sign up points to a specific person in the company that has not already registered before confirming (Goal 10);
23. The system has to provide a log in function to access all taxi drivers features (Goal 11);
24. The system must provide a function to allow logged taxi drivers to inform the system of their availability (Goal 12);
25. The system must notify taxi drivers of an incoming request for service (Goal 13);
26. The system must provide all necessary information for taxi drivers to carry out requests (Goal 13);
27. The system must provide a function to allow taxi drivers to confirm that they are going to take care of an assigned request for service (Goal 14);
28. The system must provide an explicit *decline* function to allow taxi drivers to notify that they are not going to take care of an assigned request for service (Goal 14);
29. The system must flag a taxi driver as having declined an assignment if no answer is received within 30 seconds (Goal 14);
30. The system must move a taxi driver that has declined an assignment at the end of their queue (Goal 14);
31. The system must analyse taxi locations and calculate their best possible distribution (Goal 15);
32. The system must choose which taxi drivers need to be moved to ensure total coverage of the city (Goal 15);
33. The system must notify taxi drivers in which city zone they have to move, as needed (Goal 15).

3.2 Non-functional requirements

3.2.1 Performance requirements

- The system must support a certain number of terminals for taxi drivers: this number will be at first estimated and then extracted from a statistical analysis of the system usage in its first months of life;
- The system must support a certain number of terminals for passengers: this number will be at first estimated and then extracted from a statistical analysis of the system usage in its first months of life;
- The system must support the simultaneous use of 50% of the registered passengers;
- The system must support the simultaneous use of 100% of the registered taxi drivers;
- Taxi drivers notifications must be received in less than 5 seconds.

3.2.2 Software system attributes

Reliability is an important factor, as that the system manages all taxis in the city and, were it to fall, the whole taxi circulation would be paralyzed. Portability must be considered top priority as the mobile application will need to run on all major mobile operating systems.

Security is less critical since no confidential client information are registered on or exchanged by the system.

Chapter 4

System model

4.1 Scenarios

4.1.1 Passenger sign up, real time request, taxi movement

Bob wants to go to the theater. He downloads the My Taxi Service app on his smartphone, he opens it and he finds the passenger sign up page. He inserts his name and surname, his address, and his personal citizen ID. After receiving confirmation of a successful operation, he logs in by tapping on the corresponding button. He finds three options: *call a taxi now*, *reserve a taxi for later*, *view reservation history*. He decides to tap the *call a taxi now* function. He finds a form where he inserts the destination address and the number of passengers. The system notifies him to turn his GPS on if he wants to call a taxi to his current location, otherwise he can manually insert a starting address. He taps *confirm* and the system answers by confirming back and sending him the estimated waiting time.

Charlie is a taxi driver already logged in the system, who has given his availability by tapping on the *driver available* button on his personal page in the app. He receives notification that he has been assigned to picking up Bob. He accepts by tapping on the big red *accept* button on his screen, he revs up and goes to the given starting address.

After arriving at destination, Charlie sends Bob on his way and notifies the system that he is available again. The system tells him in which zone of the city he will have to move to keep general widespread taxi coverage.

4.1.2 New reservation, previous reservation modification

Tom wants to go to the airport at 4.30 AM. Since he is a very anxious person, one week in advance he logs in to My Taxi Service, chooses the reservation function from his personal page and reserves a ride for the right day and time and destination and starting address. He is relieved in seeing that the system has confirmed his reservation. Three days later, his anxiety growing, he thinks he should change the reservation in order to be at the airport one hour before what he had planned. He logs again in My Taxi Service and he chooses the *reservation history* function: the system shows him all of his previous reservations, which in this case is just the one he had made three days earlier. He taps

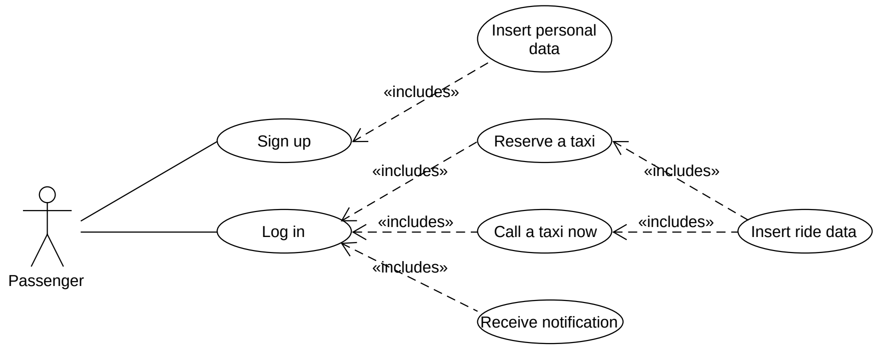

Figure 4.1: Overview of passenger use cases

the reservation entry and it pops out the option to *modify* or *delete* it. He carefully taps on *modify* (heavens forbid he deletes it) and he is prompted to insert again all necessary information. He sighs in relief when the system confirms again that everything went well. Unfortunately, two hours before the scheduled arrival of his taxi (Tom is well awake, least he forgets something for the journey), the system sends him notification that his request cannot be fulfilled due to unexpected events arising outside the sphere of influence of the taxi company. Tom is somewhat taken aback, but, after all, he expected nothing less.

4.1.3 Driver sign up and availability, assignment rejection, taxi queque handling

Eve is a taxi driver. She downloads the My Taxi Service app and signs up as a taxi driver as per company policy. She is prompted to insert her name and surname, her address, her personal citizen ID, her license number and her car's plate number. The system checks that she is indeed a real, active taxi driver and that she is on the company payroll. After receiving confirmation of a successful operation, she logs in by tapping on the corresponding button.

After going to work one morning, she logs and signals her availability by tapping on the corresponding button on her personal page. Since it's the first time in the day, the app firstly interfaces with her car's GPS tracker, and goes then in waiting mode. After a while, her app comes to life and notifies her that she has ten minutes to pick up a passenger at a certain place. She is busy having a fat cream *bombolone* for breakfast, so she declines.

The system receives Eve's reject and moves her at the end of her queue. At the same time, it notifies the next-in-line.

4.2 Use cases and UML diagrams

The following section gathers a collection of use cases and useful UML diagrams to better model the system in terms of main functionalities, flow of events, components, actors, and events.

Name	• Sign Up
Actors	• Guest
Entry conditions	• The guest isn't registered; • The guest has opened the mobile app or the web site home page.
Flow of events	• The guest clicks on the *Sign Up* button; • The system shows him the form to be filled; • The guest adds all requested information; • The guest clicks on the *Confirm* button; • The system verifies the information, confirms the registration and stores new user's information.
Exit conditions	• The guest is correctly registered.
Exceptions	• If the guest is already registered, the system notifies the user with a pop-up message; • If guest information are incomplete or incorrect, the system notifies the guest and asks him to rewrite them.

Table 4.1: Sign up use case

Name	• Log In
Actors	• Guest
Entry conditions	• The guest is registered; • The guest has opened the mobile app or the web site home page.
Flow of events	• The guest clicks on the *Log in* button; • The system shows him the form to be filled; • The guest writes user name and password: • The guest clicks on the *Confirm* button; • The system verifies the information and shows user's home page.
Exit conditions	• The user is correctly logged in.
Exceptions	• If validation process doesn't terminate well, the system ask the user to rewrite username and password.

Table 4.2: Log in use case

Name	• Request for service
Actors	• Passenger • Taxi driver
Entry conditions	• Passenger is logged into the system.
Flow of events	• The passenger taps the *Call a taxi now* button; • The passenger inserts a destination address, the number of passengers, and if needed the starting address; • The passenger confirms the request; • The system receives the request; • The system checks the taxi queue of the appropriate city zone and alerts the first taxi driver; • The alerted taxi driver accepts the request and drives to the given address; • The system informs the passenger that all went well and gives him the estimated time of arrival.
Exit conditions	• A taxi driver accepts the request for service.
Exceptions	• If the notified taxi driver declines, the system chooses the following in queue; • If the request cannot be met for whatever reason, the system notifies the passenger.

Table 4.3: Request for service use case

Name	• New reservation
Actors	• Passenger
Entry conditions	• The passenger is logged in; • The system is showing user's home page.
Flow of events	• The passenger clicks on the *New reservation* button; • The system shows him a form to be filled with all trip information (starting address, destination address, leaving date and time, number of passengers); • The user adds all information and clicks on the *Confirm* button; • The system checks the information, analyses the request and confirms with a message; • The system generates a real time request ten minutes before the meeting time.
Exit conditions	• The reservation is correctly stored.
Exceptions	• If the information is wrong the system asks to rewrite it; • If the reservation can't be satisfied the system shows an error message.

Table 4.4: New reservation use case

Name	• Modify reservation
Actors	• Passenger
Entry conditions	• The passenger is logged in; • The system is showing user's home page.
Flow of events	• The passenger clicks on the *Reservation history* button; • The system shows him all his reservations; • The user clicks on the reservation he wants to modify; • The system shows him a pop up window with two options: *Modify* or *Delete*; • The user clicks on the *Modify* button; • The system shows him a form to be filled with all trip information (starting address, destination address, leaving date and time, number of passengers); • The user adds all information and clicks on the *Confirm* button; • The system checks the information, analyses the request and confirms with a message; • The system generates a real time request at the right moment to satisfy user reservation.
Exit conditions	• The reservation is correctly modified and stored.
Exceptions	• If there is no reservation in the history, the system shows an error message; • If it is too late to modify the reservation (less then two hours before the ride), the system shows an error message; • If the information is wrong the system asks to rewrite it; • If the reservation can't be satisfied the system shows an error message.

Table 4.5: Modify reservation use case

Name	• Delete reservation
Actors	• Passenger
Entry conditions	• The passenger is logged in; • The system is showing user's home page.
Flow of events	• The passenger clicks on the *Reservation history* button; • The system shows him all his reservations; • The user clicks on the reservation he wants to delete; • The system shows him a pop up window with two options: *Modify* or *Delete*; • The user clicks on the *Delete* button; • The system deletes the reservation and shows reservation history.
Exit conditions	• The reservation is correctly deleted.
Exceptions	• If there is no reservation in the history, the system shows an error message.

Table 4.6: Delete reservation use case

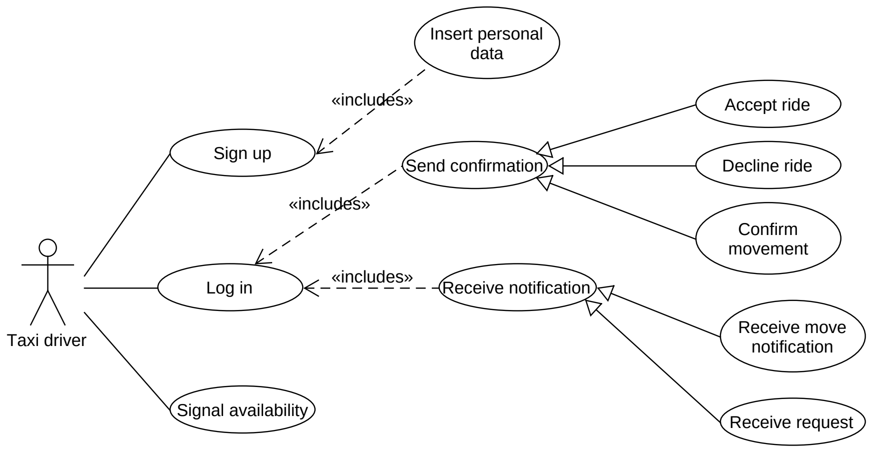

Figure 4.2: Overview of taxi driver use cases

Name	• Move taxi driver
Actors	• Taxi driver
Entry conditions	• Taxi driver is logged into the system and available.
Flow of events	• The system checks the total coverage of taxis in the city and decides a taxi driver has to move; • The system notifies the driver they have to move and where; • The taxi driver confirms they are going to move.
Exit conditions	• The taxi driver reaches the designated zone.
Exceptions	• The taxi driver does not move: the system either notifies them again or chooses another one.

Table 4.7: Move taxi driver use case

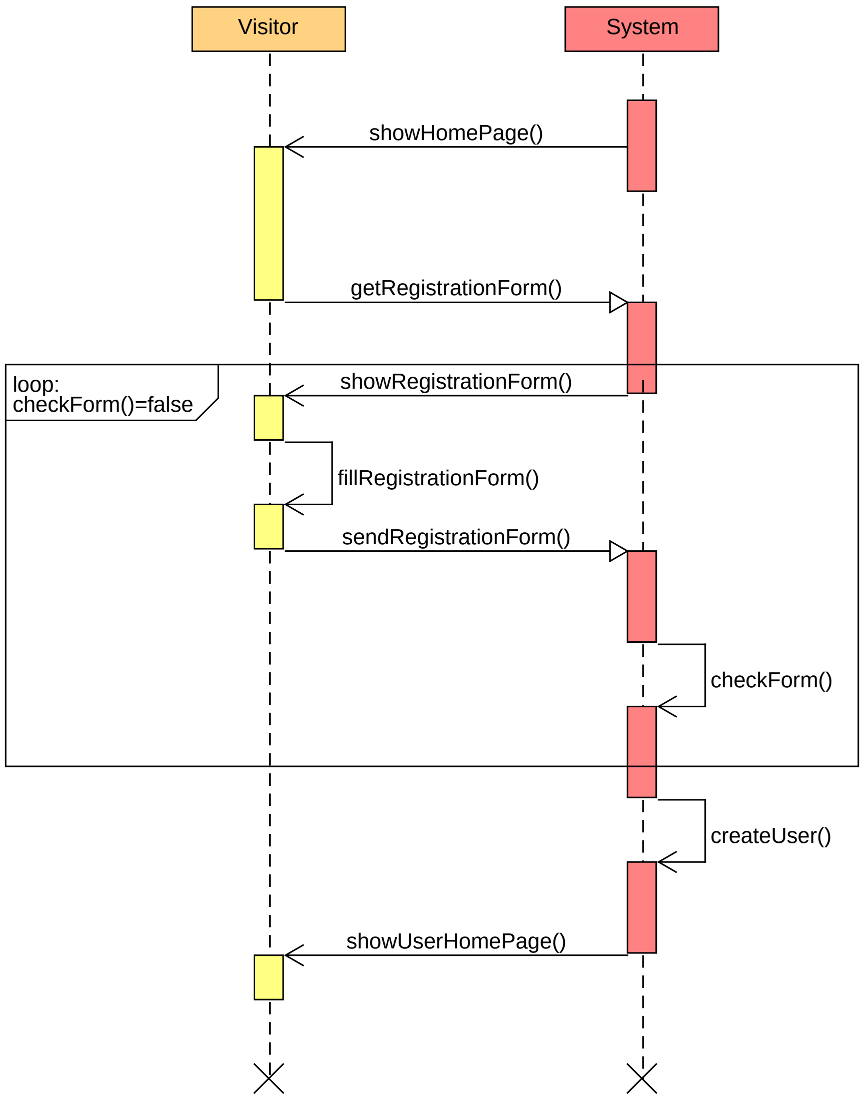

Figure 4.3: Sign up sequence diagram

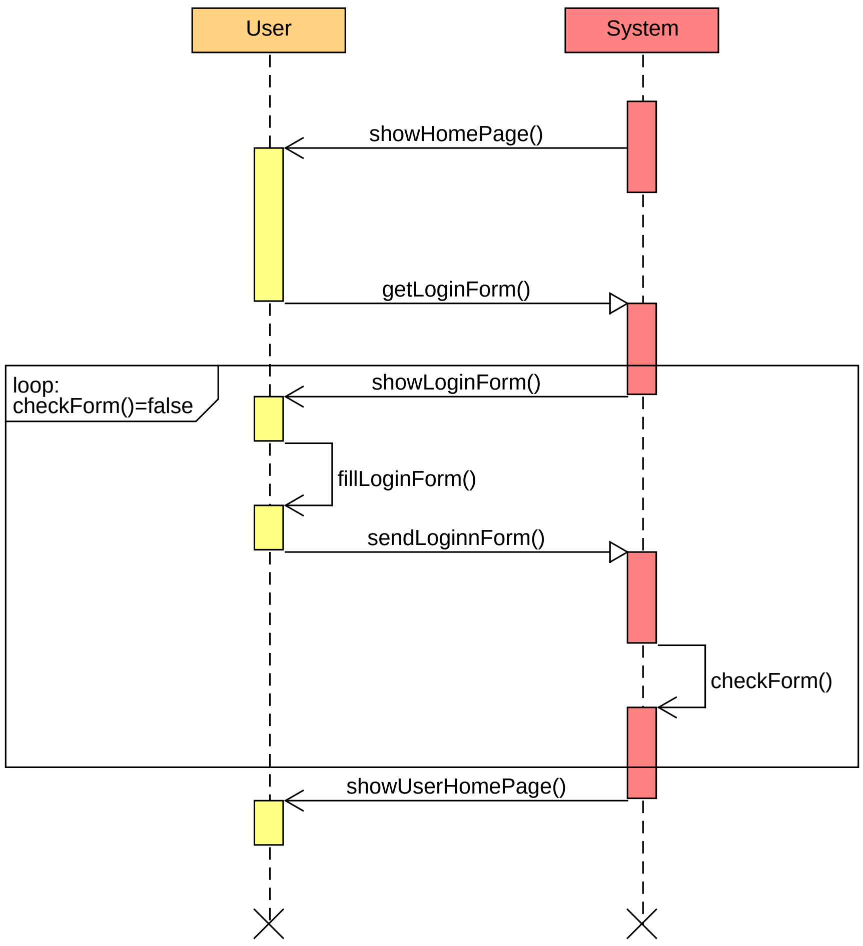

Figure 4.4: Log in sequence diagram

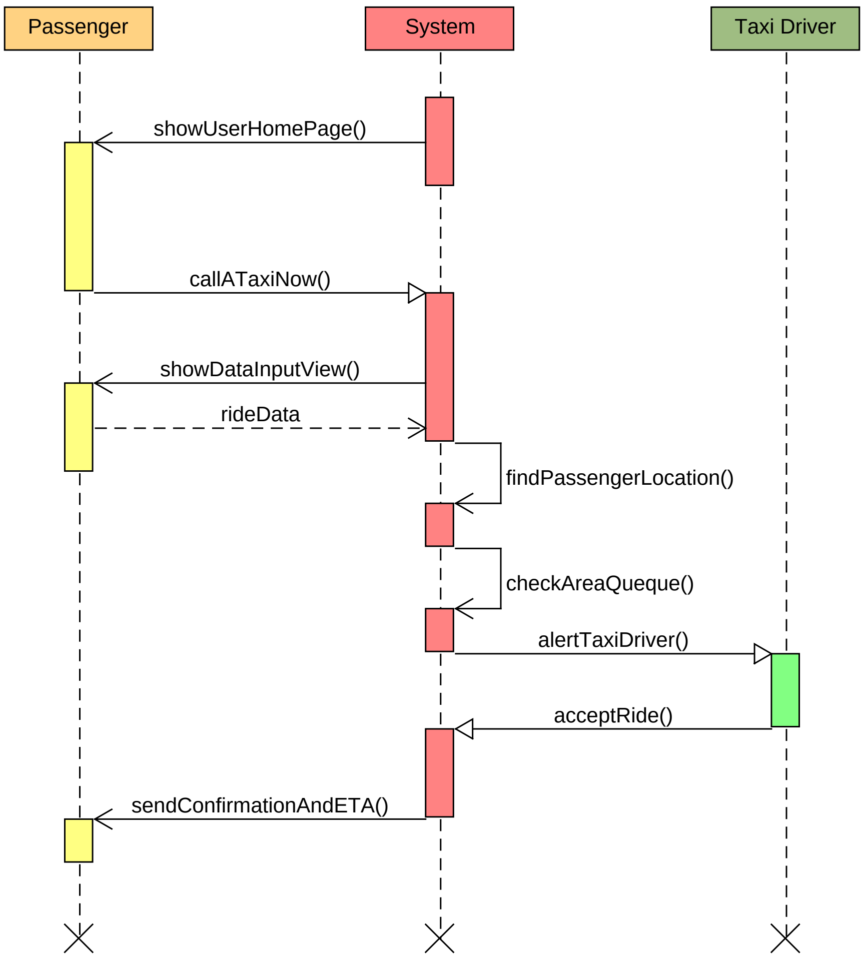

Figure 4.5: Request for service sequence diagram

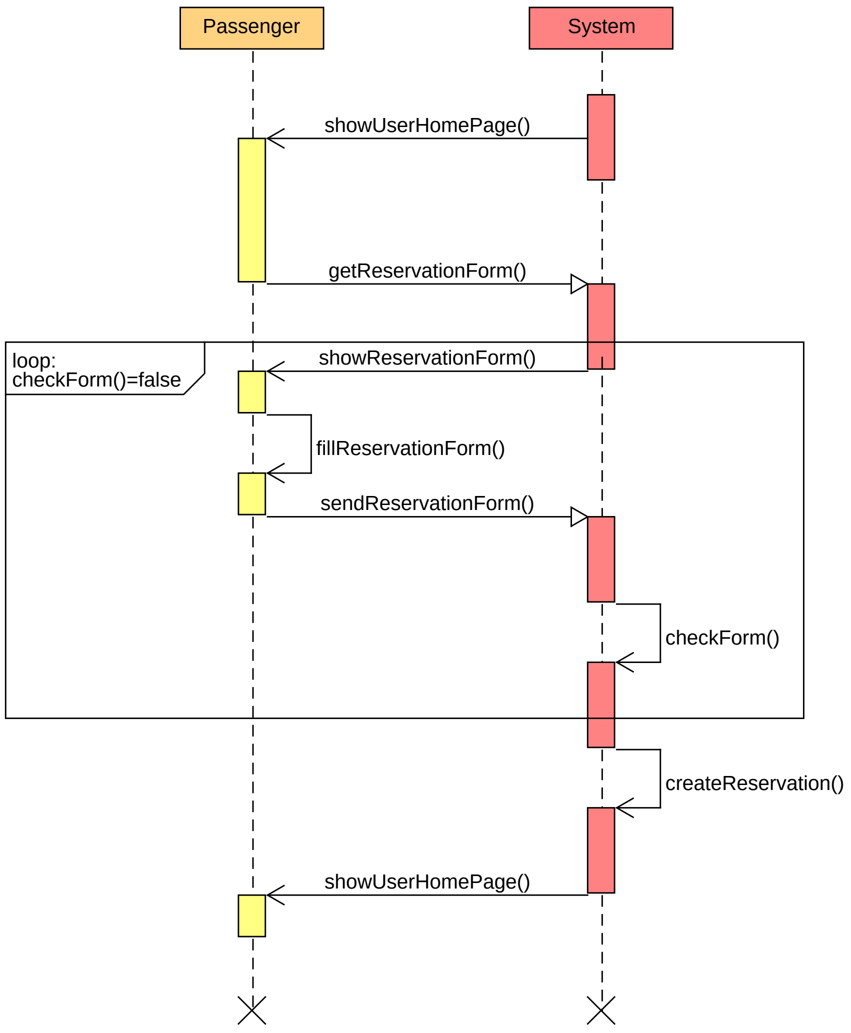

Figure 4.6: New reservation sequence diagram

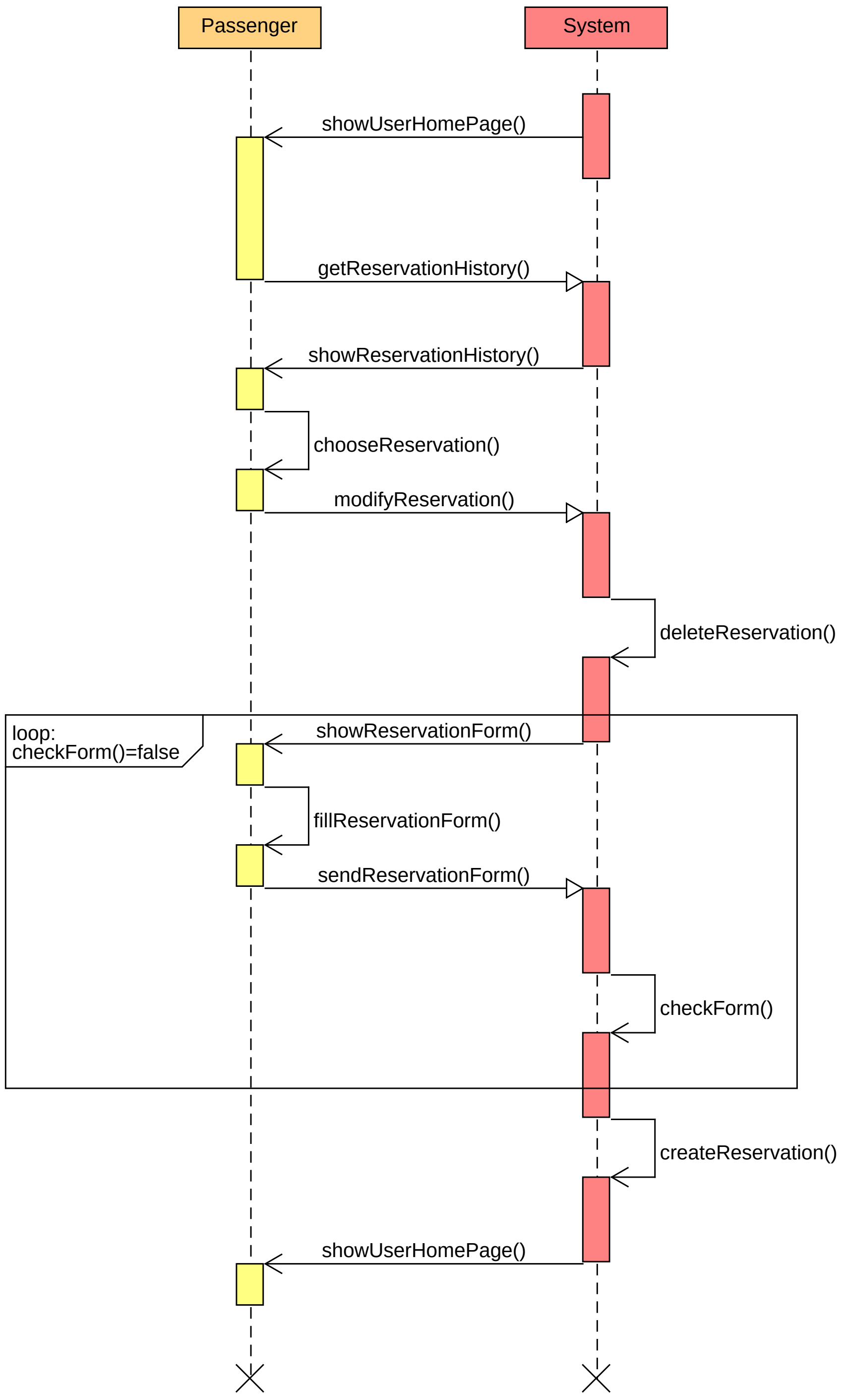

Figure 4.7: Modify reservation sequence diagram

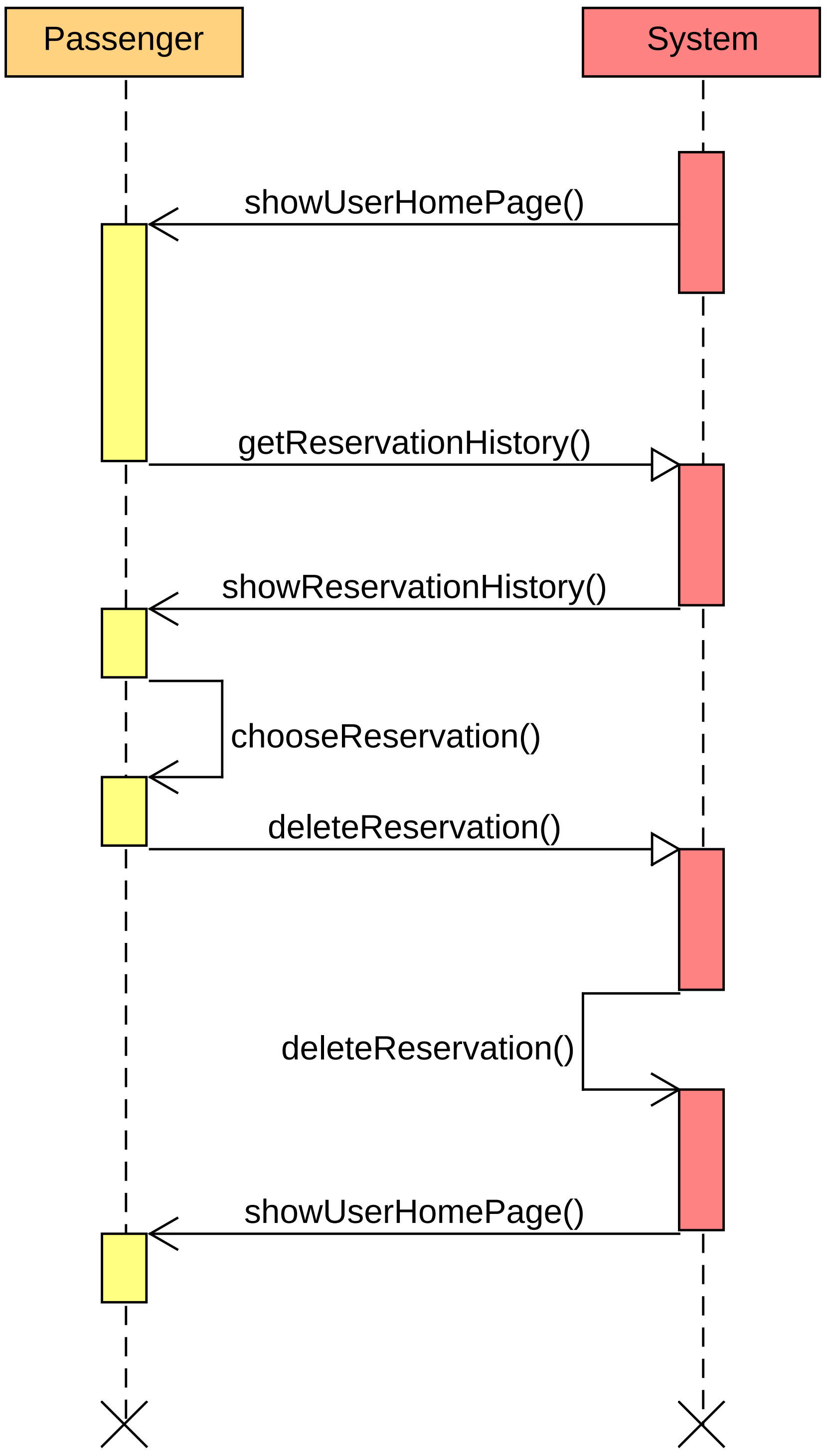

Figure 4.8: Delete reservation sequence diagram

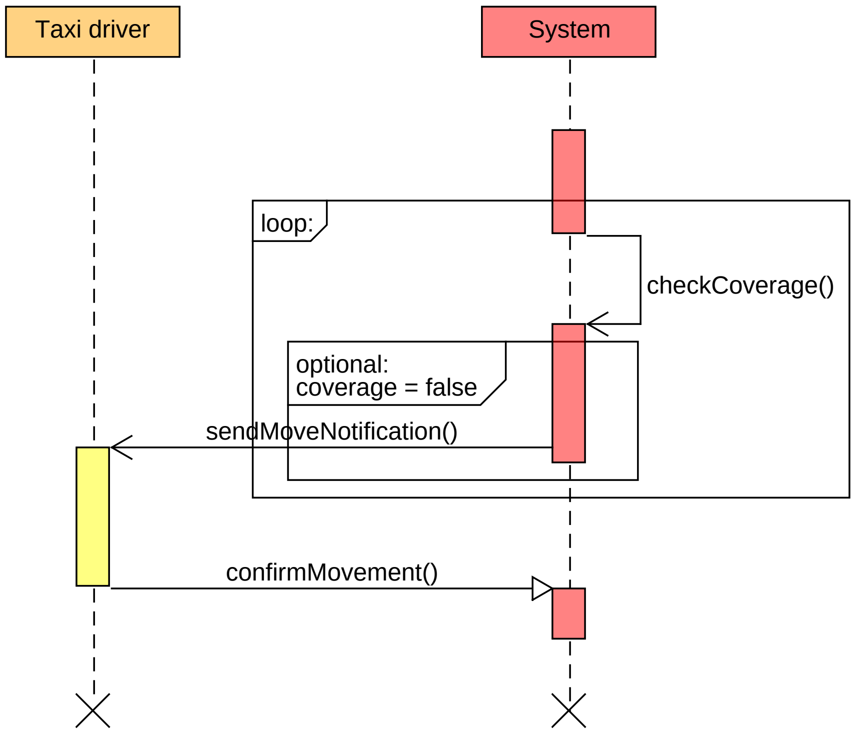

Figure 4.9: Move taxi driver sequence diagram

4.3 Alloy model

The following code is the Alloy modelization of the system that has been described in terms of requirements and goals in the previous sections. The *Alloy Analyzer* tool was used to verify these properties, starting from the class diagram. The tool has proven the consistency of this model. In fact, no counterexample has been found and every predicate is consistent. A representation of the world extracted from this model is included.

```
//SIGNATURES

abstract sig User {}
sig UserList{
   list: set User
}

sig TaxiDriver extends User{
   located: one Location
}
```

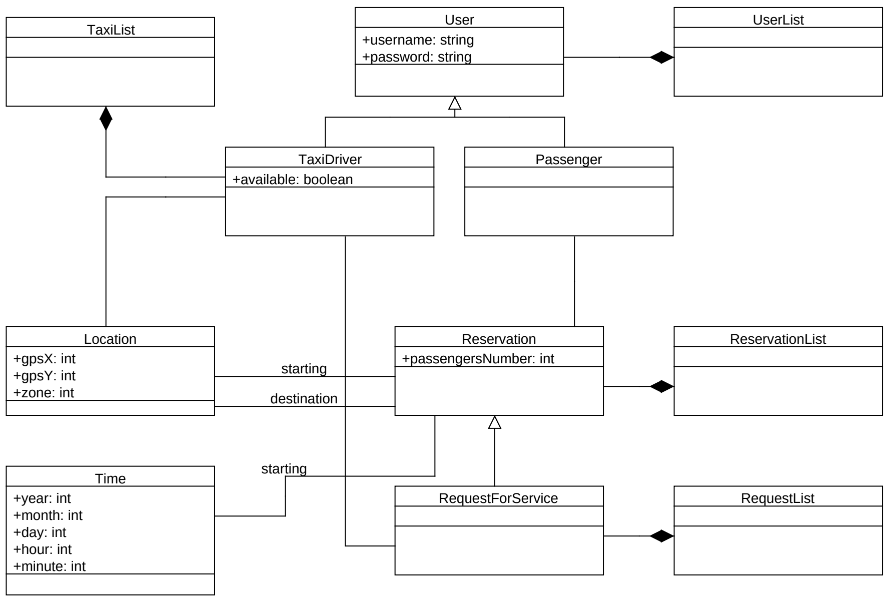

Figure 4.10: Main classes diagram

```
sig Passenger extends User{
}

sig Time{
}

sig Location{
}

sig Reservation {
    booker: one Passenger,
    startingTime: one Time,
    startingLocation: one Location,
    destination: one Location
}
sig ReservationList{
    list: set Reservation
}

sig RequestForService extends Reservation{
    server: one TaxiDriver
}



//FACTS
```

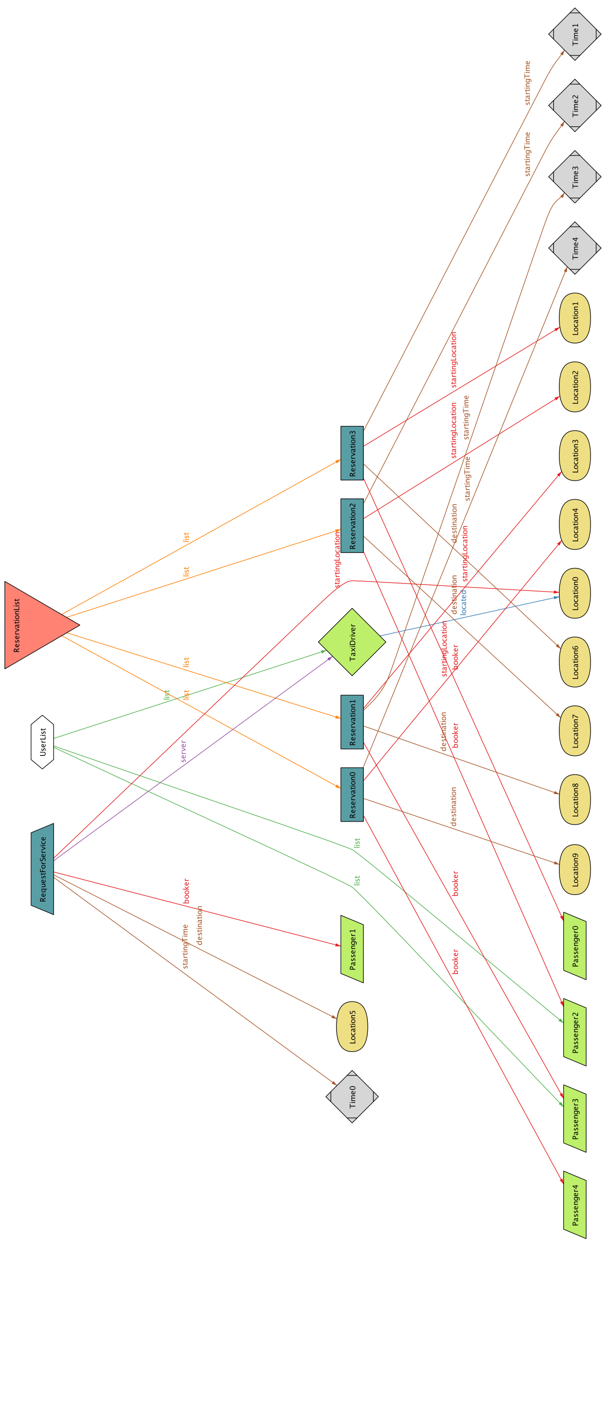

Figure 4.11: An alloy world representing this model

```

//two requests for service can't have the same
    passenger
fact{
    (no a1, a2: RequestForService, p: Passenger | a1
        != a2 and a1.booker = p and a2.booker = p)
}

//destination must be different form starting
    location
fact{
    (no a1, a2: Location, r: Reservation | a1=a2 and
        a1=r.startingLocation and a2=r.destination)
}

//two request for service can't have the same taxi
    driver
fact{
    (no a1, a2: RequestForService, t: TaxiDriver | a1
         != a2 and a1.server= t and a1.server=t)
}

//a passenger can't reserve two taxi at the same
    starting time
fact{
    (no a1, a2: Reservation, t: Time | a1 !=a2 and a1
        .startingTime=t and a2.startingTime=t and a1.
        booker=a2.booker)
}
//there is no location without reservation
fact{
    (all l : Location {one r: Reservation |l in r.
        startingLocation or l in r.destination})
}

//there is no passenger without reservation
fact{
    (all p : Passenger {one r: Reservation |p in r.
        booker})
}

//there is no time without reservation
fact{
    (all t : Time {one r: Reservation |t in r.
        startingTime})
}

//two taxi can't be in the same location
fact{
    (no a1, a2 : TaxiDriver, l: Location | a1 != a2
        and a1.located =l and a2.located = l)
}

//there is only one ReservationList
fact{
    (#ReservationList = 1)
}

//there is only one UserList
fact{
    (#UserList =1)
}


//ASSERTIONS


//checking the adding of a reservation to the list
assert addReservation{
    all r: Reservation, l1, l2: ReservationList | (r
        not in l1.list) and addReservation[r, l1, l2]
        implies (r in l2.list)
}
check addReservation for 5

//checking the removing of a reservation from the
    list
assert deleteReservation{
    all r: Reservation, l1, l2: ReservationList | (r
        in l1.list) and deleteReservation[r, l1, l2]
        implies (r not in l2.list)
}
check deleteReservation for 5



//PREDICATES


//adding a reservation to the list
pred addReservation ( r: Reservation, l1, l2 :
    ReservationList){
    r not in l1.list implies l2.list = l1.list+r
}
run addReservation for 7

//removing a reservation from the list
pred deleteReservation ( r: Reservation, l1, l2 :
    ReservationList){
    l2.list = l1.list - r
}
run deleteReservation for 2


pred show(){
}

run show for 10 but exactly 6 User
```

Appendix A

Document and work information

A.1 Revisions

This is the first version of this document. There are currently no revisions.

A.2 Tools used

TeXworks editor With PDFLaTeX, for composing and editing this document;

UMLet For drawing class diagrams, sequence diagrams, and use cases;

Alloy Analyzer For checking the consistency of the model.

A.3 Overall time spent

The authors spent about 30 hours of their time, equally divided among them, working on this document.