**POLITECNICO DI MILANO**

Department of Electronics, Information and Bioengineering

Master Degree course in Computer Science Engineering

# MY TAXY SERVICE

## Requirement Analysis and Specification Document

## (RASD)


Instructor: Prof. Elisabetta Di Nitto

Authors:

Luca Luciano Costanzo matr. 789038

Simone Disabato matr. 852863

Released on 2015/11/06

# Contents


**1 Introduction** **1**

1.1 Purpose . . . . . 1
1.2 Scope . . . . . 1
1.3 Stakeholders . . . . . 2
1.4 Glossary . . . . . 3

**2 Overall Description** **5**

2.1 User Characteristics . . . . . 5
2.2 Domain Assumption . . . . . 5
2.3 Constraints and General Assumption . . . . . 6
2.4 Actors and Functionalities . . . . . 7
2.5 Non Functional Requirements . . . . . 9

**3 Functional Requirements and Scenarios** **12**

3.1 Scenarios Natural Language Description . . . . . 12
3.1.1 System Registration . . . . . 12
3.1.2 Log in . . . . . 13
3.1.3 Manage personal information . . . . . 13
3.1.4 Management of work shifts . . . . . 13
3.1.5 Start waiting time . . . . . 14
3.1.6 Ask for a zerotime ride . . . . . 14
3.1.7 End of a ride . . . . . 15
3.1.8 Book a future ride . . . . . 15
3.1.9 Check the reservations . . . . . 16

3.1.10 Read the alerts . . . 17
3.2 Functional Requirements Scheme Description . . . 17
3.2.1 Use Case Diagram . . . 17
3.2.2 UML Class Diagram . . . 19
3.2.3 Registration . . . 20
3.2.4 Login . . . 22
3.2.5 Personal Information Management . . . 24
3.2.6 Zerotime reservation (MA version) . . . 26
3.2.7 Zerotime reservation (WS version) . . . 28
3.2.8 Future reservation . . . 31
3.2.9 Start waiting time . . . 33
3.2.10 Work shifts management . . . 35
3.2.11 Check the reservations . . . 37
3.2.12 Read the alerts . . . 39
3.3 Entities Behaviour . . . 40
3.3.1 Zerotime Ride Class . . . 40
3.3.2 Future Ride Class . . . 41

**4 Alloy Modeling** **42**
4.1 Alloy Code . . . 42
4.2 Alloy Response . . . 48
4.2.1 Counterexamples . . . 49
4.2.2 Alloy World . . . 52

**5 Other info** **55**
5.1 Working hours . . . 56
5.2 Tools . . . 56
5.3 Revisions . . . 57

**Abstract**

The main task of this document is to give a specification of the requirements that our system has to fulfil adopting the IEEE-STD-830-1993 standard for RASD documentation . It also introduces the functional and non-functional requirements via UML diagrams and a high level specification of the system. In the last part of this document it presents the formal model of the specification using Alloy analysis.

The information in this document are intended for the stakeholders and the developers of the project. For the stakeholders this document presents a description useful to understand the project development, meanwhile for the developers it's an useful way to show the matching between the stakeholders' requests and the developed solution.

# Chapter 1

# Introduction

This chapter will show the main purposes of this project, a general description of the system, the stakeholders involved and the glossary, to interpret correctly each term that will be used in this document.

## 1.1 Purpose

The system is a software called myTaxiService. It's composed by both a website and a mobile application and it aims to give the opportunities to join the city taxi service easily and fastly.

Besides the systems manages the taxi drivers' working hours and the assignment of each ride to the corresponding customer.

## 1.2 Scope

Users, once registered, are able to ask for an immediate ride or to book one of them.

The system provides the user with a complete map of the city and its suburbs within the taxi service is available. The current position of the user is obtained by localization services of the user's smartphone if it's possible,

otherwise the user notifies his position directly on the map with a marker or by a searching box. The destination is also chosen either graphically or by a research. The user can view the suggested path and then he must confirm the request.

When a user asks for a ride, the system checks the availability of a taxi driver near the current position, by splitting the city in several areas and using a FIFO (First In First Out) policy to manage the assignment of the ride's driver. The selected driver can accept or decline the ride. In the former case the system informs the user about waiting time, estimated travelling time, prices and cab car-code.

The system gives also the possibility to book a ride with at least two hours in advance. As the user does when he asks for a ride, he selects the desired starting venue and the destination. Afterwards, the system gives a calendar where the customer can choose the date (at most 30 days in advance) and the starting hour. Ten minutes before the meeting time the system starts all the operations described before in order to assign a taxi-driver.

A reservation from the app or the website can be undone until the system confirmation of the availability of a taxi, while a booking can be cancelled at most fifteen minutes before the meeting hour.

After those deadlines the ride is considered bought by the customers and an eventual absence on the established venue forbids other possibilities to book or to take a ride.

## 1.3 Stakeholders

The stakeholders are two: the city's government where the service will be given to the populations and the software house to which the city's government assigns the project. Their requests are the same with reference to

the documentations about the project and its development, while the implementation (in this project will not be done) must be complete and easy to use, due to maximize the number of people who will use it.
Hence, the purpose of the city is the improvement of the current taxi service and the possibility to have an alternative easy way to access the service. At the same time, the purpose of the company is to realize the application respecting all the characteristics, in order to receive the established remuneration by the city.

## 1.4 Glossary

**Alert**
: a system internal message to inform about, ask (a driver) and organize a single ride.

**Delay-time**
: the maximum time of waiting ensured by the system to each user. It is calculated from the ride's established time.

**Driver**
: an employer of the taxi's company who drives the reserved taxi.

**Future**
: a temporal period that in our project is between 1 hour and 30 days from the current day.

**GPS**
: a system able to calculate the correct position of each driver.

**MA**
: acronym for the mobile application.

**Map**
: an informatics representation of the taxi company's city, that can be used either by the user and by the system (also with GPS).

**Passenger**

a person who is using the service in the current moment.

**Reservation or booking**

the action that allows an user to take a ride in zerotime or in the future.

**Ride**

word referred to any single service given by the taxi's company. The service is a single-passenger travel with a driver between two precise venues.

**User**

a person who is logged in the system.

**Venue**

(departure or destination venue): a valid address that identifies a precise point on the map (and in the real city).

**Visitor**

a person who accesses to the WS or opens the MA, but it isn't enrolled or logged in the system (usually it is his first visit).

**WS**

acronym for the website of the system.

**Zerotime**

a neologism (due to precision it is a composed word) that indicates an action that the system tries to provide immediately.

# Chapter 2

# Overall Description

This chapter defines: a summary of the major functions provided by the system, the user characteristics, the constraints and the assumptions over the domain, the actors involved and a list of non-functional requirements.

## 2.1 User Characteristics

The application doesn't specify user target, so everyone with a basic web knowledge of WS or of mobile applications in general (for MA) will be able to access the service and its benefits. The user has to be registered to allow the system to recognize all the passengers and provide their with the best service. For instance, with recognition each user has to comply his contract rules and he has to use the requested services. This precludes the company to send a taxi where no-one will arrive, so it can manage better the resources.

## 2.2 Domain Assumption

In the assignments there are some ambiguities. In order to solve them some hypothesis has been added:

- *User identifications*. The uniqueness of the user is the main issue. The property is guaranteed by the fiscal-code (or the equivalent ID-code in the country where the service is activated), which is unique by definition and forbids to an user multiple registrations on the service.
- *Ubiquity*. An user cannot have overlapped rides. Hence, if an user books a ride (even in zero-time) it cannot reserve or ask for another ride until the ride isn't finished. Besides, only one driver can be assigned to each ride.
- *Drivers' work times*. A driver is alerted for a ride only when he's at work (in his work shifts).
- *GPS correctness*. The GPS's data received are correct with a precision of two meters, so that the real current position of a driver/user is into a circle with center in the position given by the GPS and radius equal to two.
- *Map unicity*. The map of the city is unique both for WS and MA. As consequence, all users access the same version of the map.
- *Deadlines integrity*. A zero-time ride can be undone until the system confirmation, while a reservation can be cancelled at most 15 minutes before the leaving hour. After that, the aborts are invalid.
- *Driver Availability*. A taxi driver who is working (so he is in his work hours) replies to a system request in a maximum time. It is a strong assumption, but when a driver is waiting for a ride we suppose he is on his cab and can immediately sees a noisy notification on his smartphone.
- *Taxi Availability*. We assume that can occur, with an estimated probability of 0.01, that there isn't no available drivers in the selected area.

## 2.3 Constraints and General Assumption

The main constraints in the assigned project is the interaction of the system with a DBMS (DataBase Management Service) and, for a better implementation of registration's functionalities, the interaction with a SMPT

server (email services).

The other constraints are the following:

- *Policies and laws*. The developed system, both WS and MA, must follow all the laws concerning the taxi service and the websites. For example, the WS has to inform all users and visitors about the cookie use.
- *Hardware limitation*. There is no specific limitation on hardware requirements. Due to allow a greater number of people the use of the service, it is strictly recommended to develop a MA that can be installed on all the three mobile platforms (the requirement is at least the two majors).
- *Interface's implementation*. The system must be developed with interfaces extendable to other possible functions. With refer to external interfaces is required only the correct integration with the GPS.
- *Privacy of users' data*. No user is able to search and find any data about another user.
- *Delay time*. The systems has to ensure a maximum waiting time, called delay time, to each user, that may be different in zero-time and future reservation.
- *Area dimension*. The dimension of each area is estimated in 2 squared kilometers.
- *Taxi Availability*. In the particular cases of taxi's absence in the current area the system has to enter a special mode where it is be able to asks for a ride in the areas closest to the first one.

## 2.4 Actors and Functionalities

The actors of myTaxiService are:

- **A1 Visitor**: a visitor is a person who is not logged in the system. The only two functionalities that a visitor is allowed to do are the registration (only the first time) and the login, both for the WS and

the MA.

- **A2 User**: an user is a person who is logged in the system. He is able to:
  - Ask for a zerotime ride.
  - Book a future ride.
  - Manage his personal information.
  - Check his reservations (also with the historical ones).
  - Modify or cancel his reservations.
  - Read his alerts.
- **A3 Driver**: a driver is a special case of user (even if it is a very strange situation, the system doesn't forbid a driver to book a ride with another driver, so all the functions of a normal user are also available for a driver), with reserved special functions:
  - Management (with specific global rules) of work shifts.
  - Start waiting time:
    this function means that the driver notifies the system that he's waiting for a ride and about his current position (with GPS).
  - Accept or deny a ride.
- **A4 Cab company**: The taxi corporation is a public company, directly managed by the city's government. In the system it is a special user that has only a few special functionalities:
  - Employees' management:
    the taxi corporation can add or remove a driver (this action correspond to an assumption or a sacking of the driver) and check their work shifts.
  - Control the current situation of the service and, in emergency states, it can ask some drivers a transfer to another area.
  - Alert all users about service informations, for instance dates of worker's strikes, city's areas unreachable and so on.
- **A5 Ride's allocator**: the ride's allocator isn't a physical person, but an actor internal to the system that handles the assignments of the rides.
  Hence, it is called when a ride needs to be carry out and its functions

are:
  - Management of the taxi's queues.
  - Check the availability of the designed driver.
  - Sending of the alerts to inform both users and drivers.
- **A6 GPS**: the GPS isn't a physical actor, it gives the correct position of a driver or an user (if it uses the MA on a smartphone that support this functionalities).

## 2.5 Non Functional Requirements

In this paragraph are shown some non functional requirements that describe some qualities of the system and the correctness of the sequential execution of each operation:

- *User friendliness*. The system should be as simple as possible. Hence, each user that has never seen the WS or the MA is able to use it easily. In particular, no training should be needed to use the application.
- *Portability*. As we said above under the hardware limitations' constraint, the system has to be compatible with at least two of the three mobile platforms for the MA. Referring to the WS, it should be accessible and correctly shown on all common browsers.
- *Performance*. To supply suitable service, the system has to be reactive and able to answer to a high number of also simultaneous requests. Because of this the interaction between the client and the server has to be reduced to a minimum, in order to no overload the net.
- *Requests processing and performance*. When an user asks for a taxi it is supposed that he want to use the service immediately with the minimum waiting time possible. Therefore, the system must respect the maximum delay time (see above under the homonymous lemma in paragraph 2.3) and it is be able to process a number of contemporary actions estimated in 0.01% of the city's population.
  An important observation on this requirement is the following: the management of more than one contemporary requests into the same

area forces the developer to implement specific parallel techniques of programming due to not overlap them.

- *Input error*. Whenever the GPS is available to identify the correct position of an user, the system should use it. In the other cases, to prevents misunderstandings on leaving address caused by input errors, the system should asks the user to confirm his position with a complete message (for instance, "are you in [address]?"). The same requirement is mandatory for the booking functions, the work shifts management and during the registration. In the last case it is requested to check the validity of all information before saving it on the DBMS.
- *Robustness*. The system should reacts to all the possible situations, in order to don't fall down or lose data. Hence, it should inhibit spam's attacks (this can be guaranteed by identification and denial of multiple requests by the same user) and any kind of data attacks (see also Data integrity, consistency and availability).
  After a failure the system must be able to restore all data and reactivate all the functionalities in less than 30 minutes.
- *Data integrity, consistency and availability*. In normal conditions, data have to be always accessible. To restore the system after an eventual fault and to prevent data loss, the data should be duplicated adopting special DBMS architectures.
- *Reliability*. In the ideal situation the system is active 24 hours per day and 7 days per week, so the maintenance should be allowed every day without compromising the functionalities.
  In particular cases, especially on critical function of the system, it is allowed to shut down all the system. This situation must be notified to all users in advance (about one week before) and the night hours are the most indicated to perform the needed operations.
- *Security*. To guarantee the privacy on all users' informations and data, the system has to implement specific security protocols and techniques. First of all, all accounts must be protected by an email and a password associated to the account. This one must be created only

with the personal tax code (or the equivalent code in the city's country) and confirmed with a link send by emails. Then, all inserted data must be controlled before the storing on the system to prevent undesired modifications (SQL injections and others).
At last, all sensible data must be transmitted on a secure connection, therefore the system must adopt the HTTPS protocol. Even if this protocol requests the adoption of special permits and certificates, it cannot be ignored and it must be implemented and tested since the first release.

# Chapter 3

# Functional Requirements and Scenarios

This chapter presents the functional requirements of the system, already introduced in the previous chapter functionalities. At the beginning they will be presented in natural language as scenarios, then from these scenarios a higher level description will be abstracted. This last section will present the use-cases in the form of UML and sequence diagrams.

## 3.1 Scenarios Natural Language Description

As an example, the story of the passenger Albert and the taxi driver Bob is shown.

### 3.1.1 System Registration

Albert is new in the system and wants to register. He clicks on the "Register" button and he's redirected on the registration page. He inserts his personal information, like name and surname, gender, birthday, city of residence, personal tax code, e-mail and passwords. Then he submits the request to the system. The data sent are checked by the system to prevent hacks and to ensure that Albert with that personal tax code is not already

registered. If the registration is successful a confirmation link is sent to the e-mail address specified by Albert and he's redirected to a login page, otherwise an error message is displayed.
Instead, Bob doesn't need to register itself to the system, because the cab company automatically inserts his on the system and then gives his the access keys. Besides, no confirmation is required for a taxi driver, so no messages are sent to Bob.

### 3.1.2 Log in

Albert is already registered and wants to log in. He clicks on the login button and he's asked to insert his own e-mail address and password. If the combination of data is correct but he hasn't clicked yet on the confirmation link sent to his e-mail address, the login is refused and a message is shown that reminds him to do it. If the combination of data is correct and Albert has already confirmed the e-mail address he can enter the system. Otherwise if the combination isn't correct an error message appears.

### 3.1.3 Manage personal information

Once Albert is logged into the system, he wants to manage his personal information, like changing the e-mail address (if he does it he has to confirm again the new address by clicking the link received in a new e-mail), city of residence, or adding other optional data, like his occupation, ...

### 3.1.4 Management of work shifts

Bob is already a registered driver in the system and wants to manage his work shifts. He clicks the corresponding button on his profile page, then a form is shown with all the week days and for each day an input field for the starting hour and an input field for the ending hour of his work shift. Bob has to select which days of the week he works and the hours. Finally he submits the information.

### 3.1.5 Start waiting time

Bob today starts to work now and has to notify the system that he is available. He clicks the corresponding button on his profile page to start the waiting time, then a map is shown in his MA with his current position given by the GPS and the current address is written and can be manually modified. If the GPS is not working Bob has to manually specify his position by inserting the address and has to submit the information.
Therefore the ride's allocator inserts Bob in the queue of availables driver for the city area where he is.

### 3.1.6 Ask for a zerotime ride

Now that Albert has entered the system he can have access to all the features available to a passenger. He wants to ask for a zerotime ride and clicks the corresponding button in his user profile page. A map is shown with the current position obtained by the GPS of the MA of Albert and the actual address is written and can be manually modified by the user. If the GPS is not working or Albert is accessing the service via the WS, a message is shown claiming that the current position cannot be found and Albert has to manually specify the address of his actual position.
Albert submits his actual position and then he has to specify also his destination by selecting a point in the next map that is shown or by writing the destination address in the corresponding text field.
The current position and the destination address are verified, if they are correct the travelling information are shown to Albert, like the estimated travelling time for the ride and the price. Albert must finally confirm the ride request. Instead if the starting or the destination addresses are invalid an error message is displayed and Albert has to correct them.

After Albert's final confirmation, the ride's allocator contacts the first available driver in the queue near Albert's position.
Bob is the first available driver near Albert's position and receives a notifi-

cation through the MA on his smartphone. He is asked to choose whether to accept the ride or to reject it. If he declines it another driver in the queue will be notified.
Bob accepts the ride.

After Bob's confirmation the ride's allocator remove Bob from the queue of available drivers, calculates the estimated time and sends a notification to Albert with the estimated waiting time, the driver's name and his cab car-code, through his MA and e-mail.

Today Bob is waiting for a ride in front of the railway station. Albert, who was arrived in the city by train a few minutes ago, sees the parked taxi and asks him a ride. Bob uses the same functionality shown above to notifies the system that he is no longer available for the rides.

### 3.1.7 End of a ride

Albert has reached his destination and pays Bob for the ride. Bob now has to notify the system that he is available again. He clicks the start waiting time button in his profile page, then a map is shown in his MA with his current position given by the GPS and the current address is written and can be manually modified. If the GPS is not working Bob has to manually specify his position by inserting the address and has to submit the information.
Therefore the ride allocator inserts again Bob in the queue of available drivers for the city area where he is now.

### 3.1.8 Book a future ride

Albert wants to book a future ride for a week later, he clicks the corresponding button in his user profile page. Albert is asked to select the date for the ride that can be only between 30 days later and the starting position by selecting a point in the map that is shown or by manually writing

the address in the corresponding text field. He submits the information and then he has to specify the destination position in the same way of the starting position and has to submit it.
The starting and destination position are verified, if they are correct the travelling information are shown to Albert, like the estimated travelling time for the ride and the price. Instead if the starting or the destination position are invalid an error message is displayed and Albert has to correct them. Albert must finally confirm the ride request. After the confirmation the ride reservation is saved and Albert can view it (with all the information), modify it or cancel it, by accessing to his reservations page by clicking the corresponding button in his profile page.
Albert might change idea and delete his ride booking during the next seven days until 15 minutes before the chosen ride time.

After seven days, 1 hour before the chosen ride time Albert receives a reminder of his booking via the MA and the e-mail.
Albert hasn't cancelled his reservation, so 10 minutes before the chosen time, the system starts all the operations described before to contact an available driver and assign him to the Albert's ride.

### 3.1.9 Check the reservations

Albert wants to check his reservations and clicks the corresponding button in his user profile page. A list of all the reservations (also the zero-time rides) ever made by Albert are shown, with all the information, divided per page and in descending order by date. The future rides not yet completed have a button beside to modify the date or to cancel them. If Albert clicks the modify button he's redirected to the same page of the future ride booking.

### 3.1.10 Read the alerts

Albert wants to read his received alerts and clicks the corresponding button in his user profile page. A list of all the alerts/notifications he has ever received is shown, with all the information, divided per page and in descending order by date.

## 3.2 Functional Requirements Scheme Description

In this paragraph are shown all the functionalities in a formal way, using Sequence Diagrams, UML modeling and flow sequences with related actors, preconditions, postconditions and exceptions.

An important note is the following: if an input error is modelled in a sequence diagram, it is usually represented as an alternative frame. Hence, when the input data are received and checked by the system, there are two alternatives: the data are correct, thus the operations follows the right sequence; the data are incorrect and an error message is shown. We made this choice due to simplify all the sequence diagrams: obviously any control on the inserted data is repeated in a loop until they are acceptable.

### 3.2.1 Use Case Diagram

The Figure 3.1 gives an overview on the main actors involved in the system and the functionalities that they are able to execute.

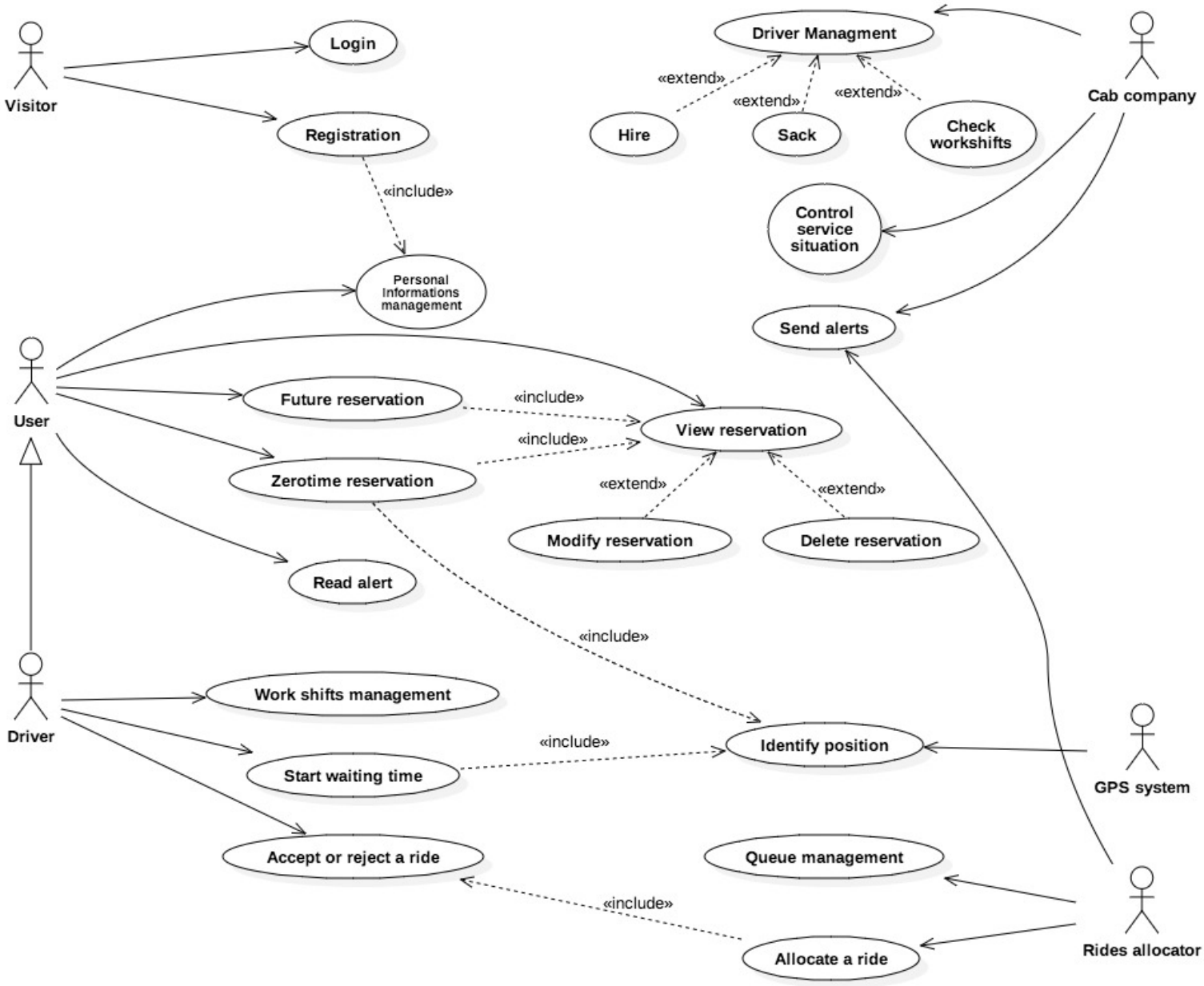


Figure 3.1: Use Case Diagram for myTaxiService

### 3.2.2 UML Class Diagram

The Figure 3.2 describes a first proposal of entities and their relation.

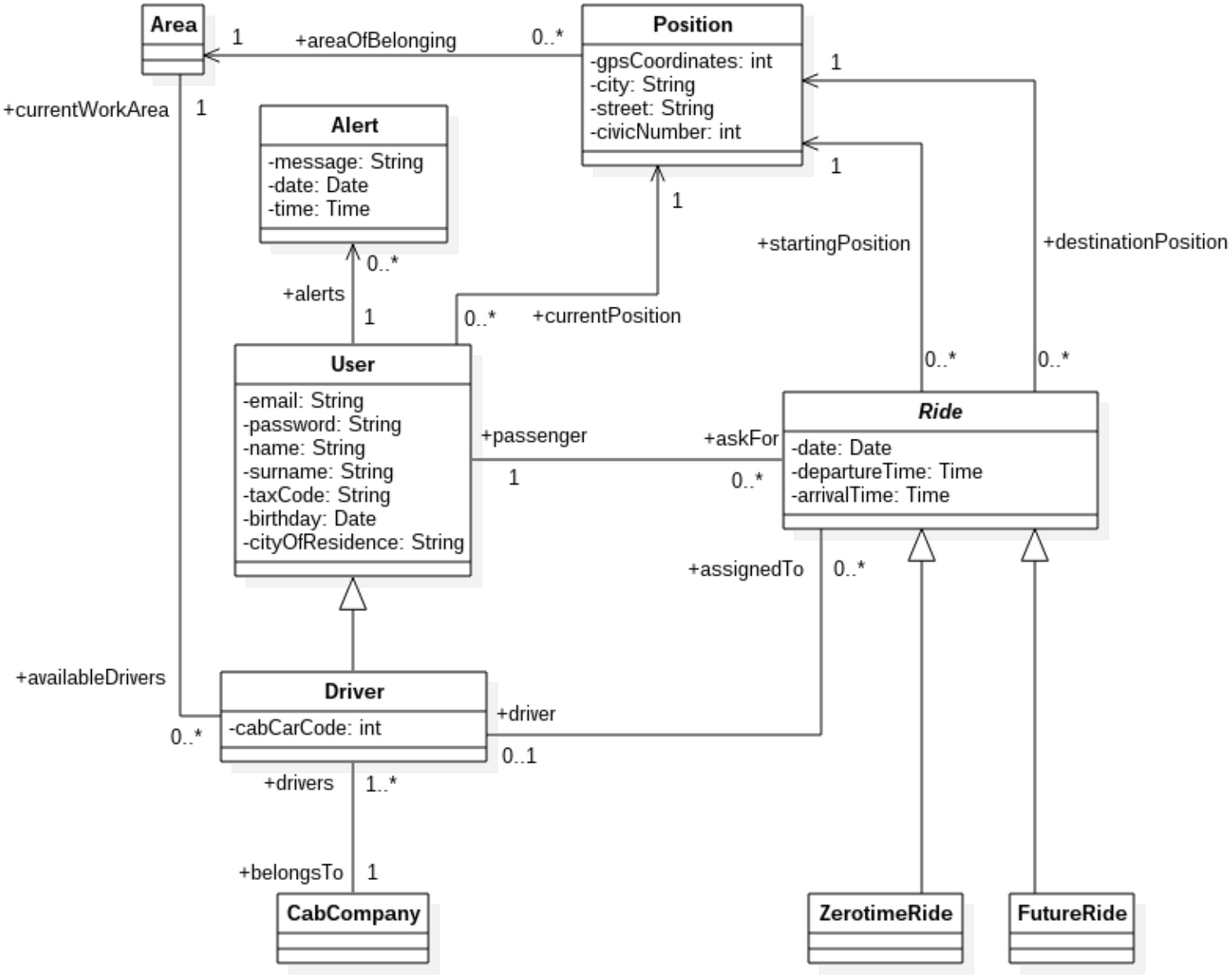


Figure 3.2: Class diagram for myTaxyService

### 3.2.3 Registration

| Actors | Visitor |
|---|---|
| Preconditions | The visitor has never registered before in the system. |
| Execution Flow | 1. The visitor accesses the registration page.<br>2. The system asks the visitor to insert personal informations: name, surname, gender, address, date of birth, email, tax code, and a password used to access the myTaxiService from now.<br>3. The system checks the data, the tax code's unicity and the correctness of the email address.<br>4. The system sends the confirmation link by email and notifies the correct registration.<br>5. The visitor clicks on the confirmation link.<br>6. The system redirects the visitor to the login page. |
| Postconditions | The Visitor is now registered on the system, but it isn't logged in yet. |
| Exceptions | The tax code already exists in the system (so the visitor is already registered), the inserted email is nonexistent or invalid, the visitor doesn't complete the registration clicking on the link sent by email. |

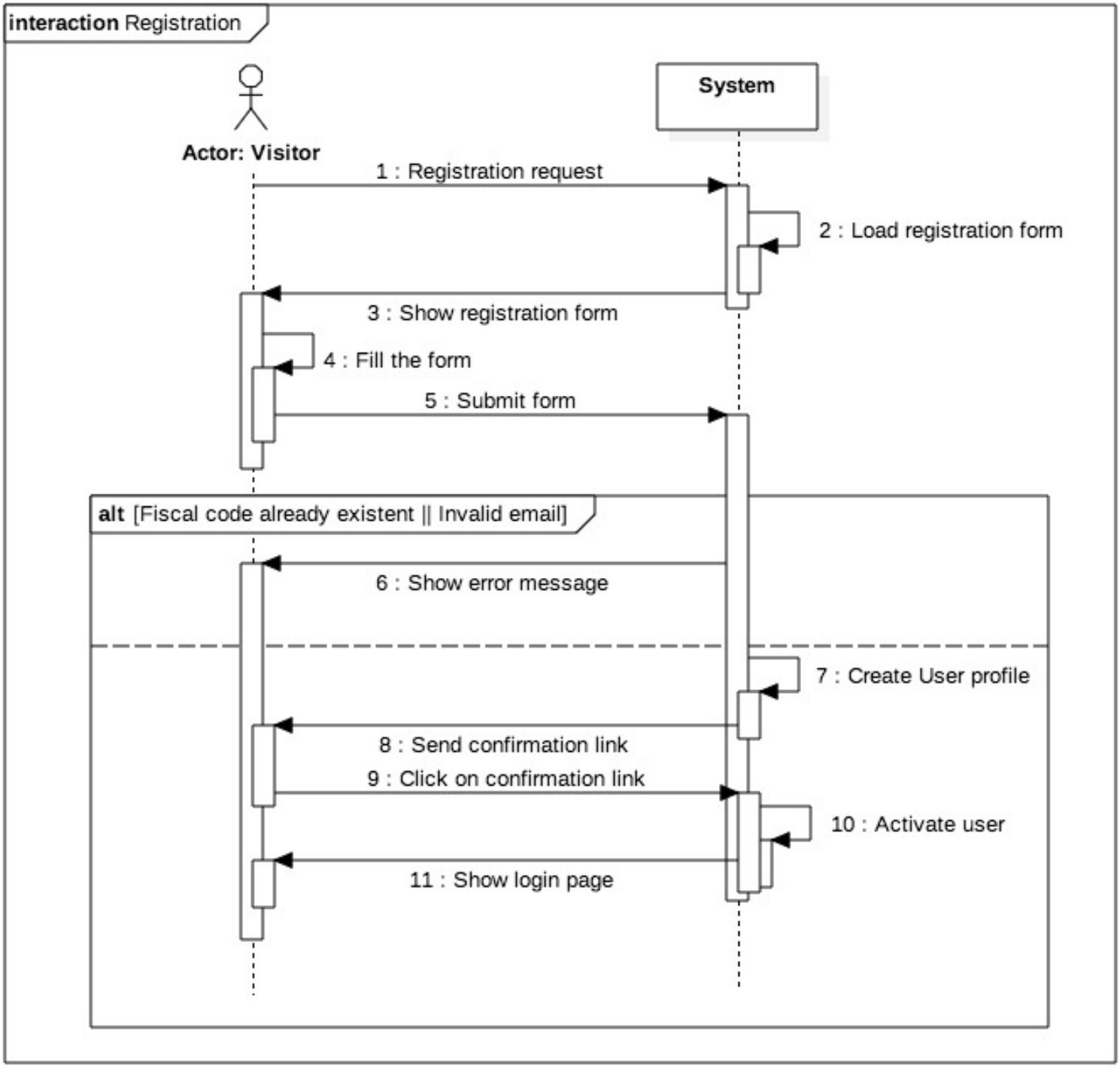


Figure 3.3: Sequence diagram for the registration

### 3.2.4 Login

| Actors | Visitor |
|---|---|
| Preconditions | The visitor is registered in the system. |
| Execution Flow | 1. The Visitor clicks on the login button.<br>2. He inserts his email and password.<br>3. The system checks inserted data.<br>4. The Visitor is redirected to his personal page and becomes an user or a driver. |
| Postconditions | The Visitor is now an user or a driver (depends on his keys), he is logged and his capable to use all his reserved functionalities. |
| Exceptions | The couple of email and login isn't incorrect. The inserted email doesn't refer to any user. In both case the login is refused. |

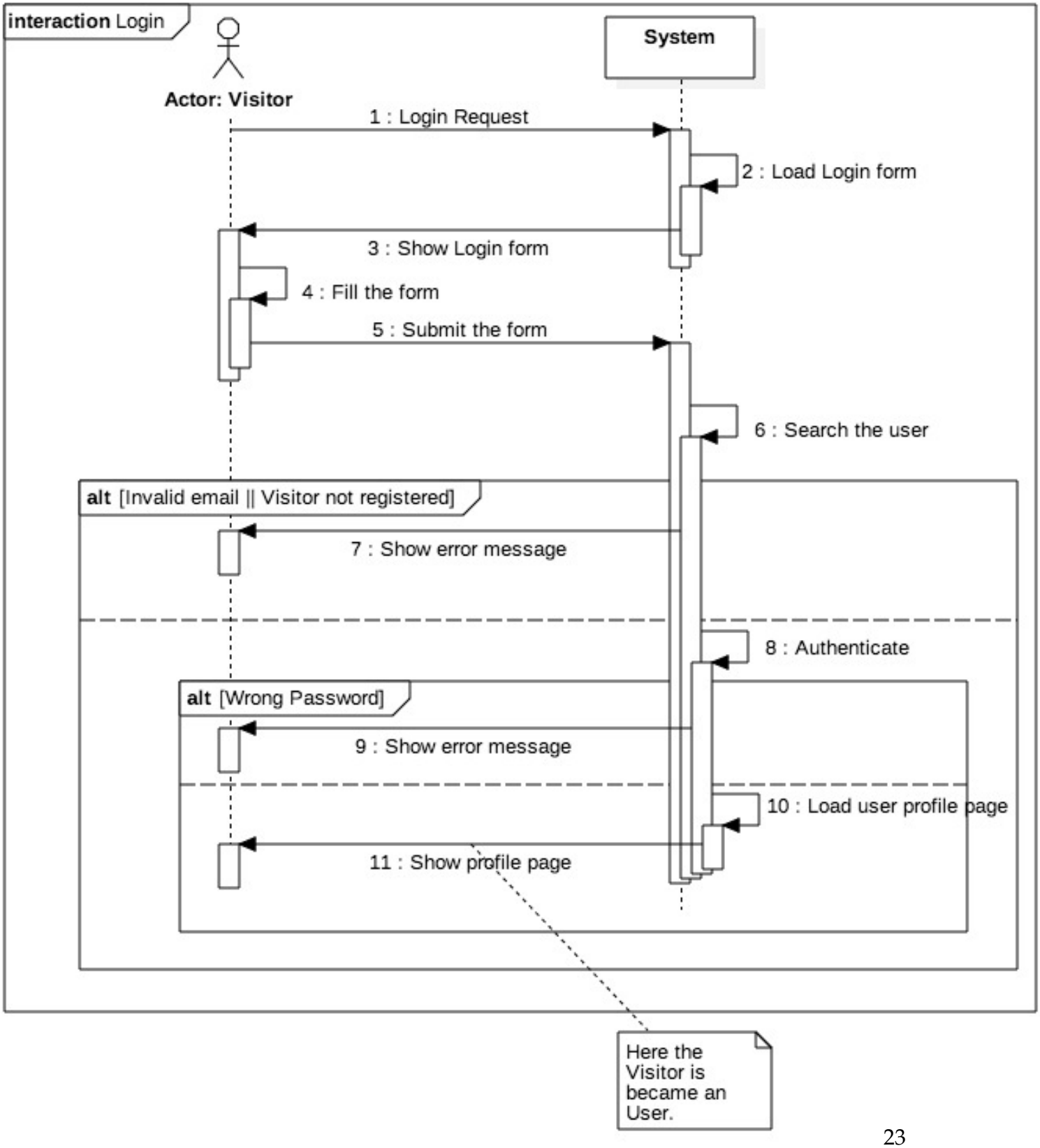




Figure 3.4: Sequence diagram for the login

### 3.2.5 Personal Information Management

| Actors | Users |
|---|---|
| Preconditions | The user is logged into the system. |
| Execution Flow | 1. The user selects the related options in his profile page.<br>2. The system shows the modification form to the user (name, surname and tax code are not modifiable).<br>3. The user modifies the desired information, then submits the form.<br>4. The system checks the information and modifies them in the system database. If the email has been modified, the system sends a confirmation link to user.<br>5. Only if the user has modified the email, he clicks on the confirmation link to save his new email. |
| Postconditions | The user's personal information are modified. |
| Exceptions | A driver tries to modify his personal information: the system refuses this operation (only the cab company can handle the driver's data). The modifications are invalid: the system rejects them. |

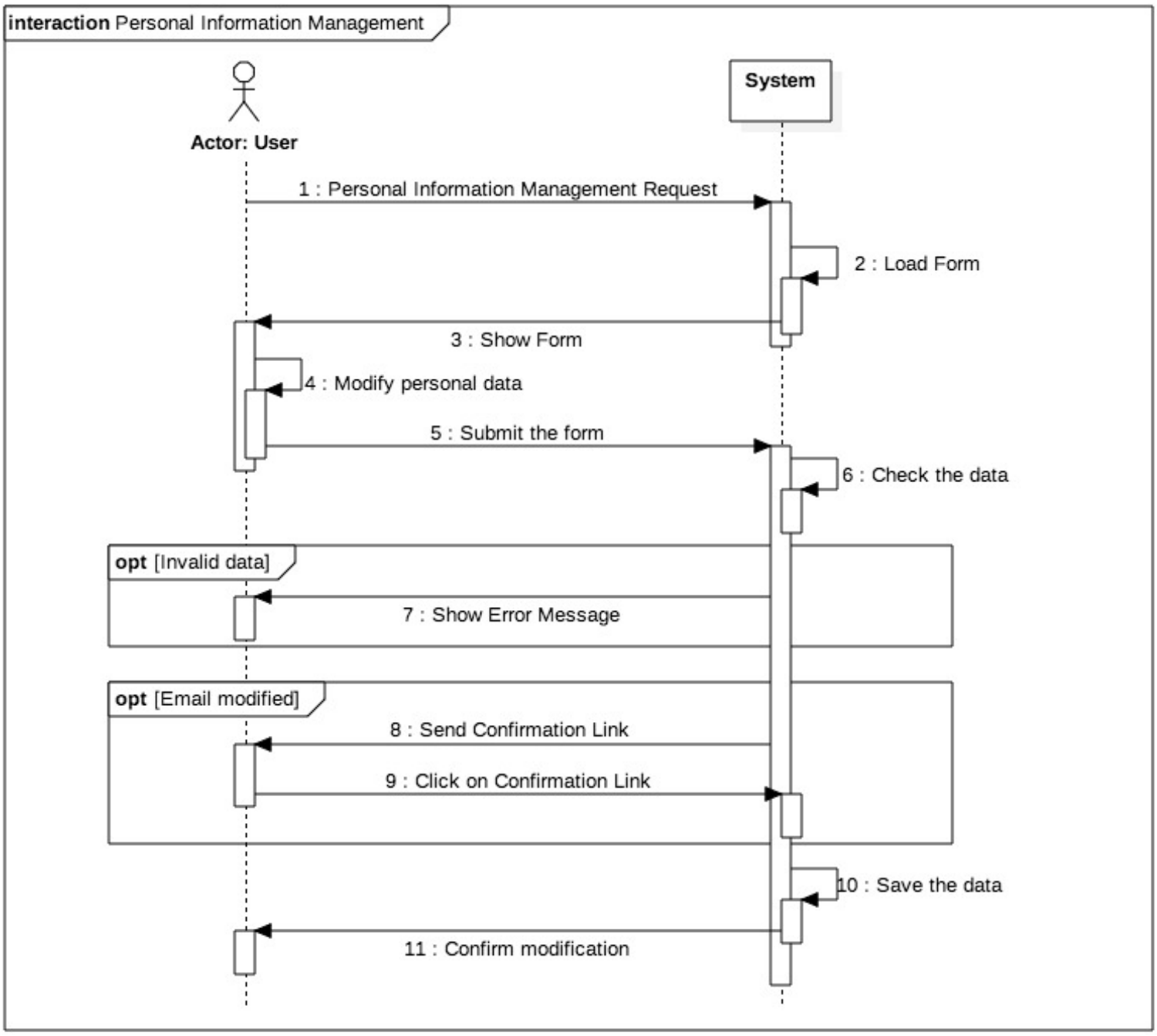


Figure 3.5: Sequence diagram for the personal information management

### 3.2.6 Zerotime reservation (MA version)

| Actors | User (or Driver) |
|---|---|
| Preconditions | The visitor is logged into the system. The driver that will be chosen is into a taxi queue. |
| Execution Flow | 1. The user clicks on the related button.<br>2. The system calls GPS system and the latter one calculate the user's position. Then the former one shows a map with the position to the user.<br>3. The user confirms his position by clicking on the shown popup and then submits the destination by clicking on the map or by inserting the address manually.<br>4. The system verifies the destination, then confirms it to the user and informs him about travelling time and price.<br>5. The ride allocator is involved by the system and it contacts the first driver in the queue. It asks him the availability for the ride. The driver accepts, so the ride allocator ends its work informing the user about the cab carcode, the driver name and the waiting time.<br>6. The system saves the ride's information. |

| | |
|---|---|
| Postconditions | The ride is assigned to the user, a taxi is moving to the position of the user with a driver different from the user. The driver selected for the ride isn't in any taxi queue. The ride is saved on user's history. |
| Exceptions | The GPS doesn't work (in this case see the paragraph 3.2.7). The destination is missing: the ride is not sent to the system.<br>The driver rejects the ride: the system puts him at the end of the queue and contacts the following one. The queue is empty: the system contacts the near areas (special mode).<br>The user aborts the ride before system confirmations: the system aborts all operations executed to assign the ride. |

### 3.2.7 Zerotime reservation (WS version)

| Actors | User (or Driver) |
|---|---|
| Preconditions | The visitor is logged into the system. The driver that will be chosen is into a taxi queue. |
| Execution Flow | 1. The user clicks on the related button.<br>2. The user insert the starting and the destination address directly on the map or in the form writing them.<br>3. The system checks the correctness of the addresses.<br>4. The system confirms the received data to the user and informs him about travelling time and price.<br>5. The ride allocator is involved by the system and it contacts the first driver in the queue. It asks him the availability for the ride. The driver accepts, so the ride allocator ends its work informing the user about the cab car-code, the driver name and the waiting time.<br>6. The system saves the ride's information. |

| | |
|---|---|
| Postconditions | The ride is assigned to the user, a taxi is moving to the position of the user with a driver different from the user. The driver selected for the course it isn't in any taxi queue. The ride is saved on user's history. |
| Exceptions | The destination or the leaving position is missing: the ride is not sent to the system.<br>The inserted data are invalid: the system ignores the request.<br>The driver rejects the ride: the system puts him at the end of the queue and contacts the following one. The queue is empty: the system contacts the near areas (special mode).<br>The user aborts the ride before system confirmations: the system aborts all operations executed to assign the ride. |

In the following page there is a sequence diagram for both zerotime reservation cases.

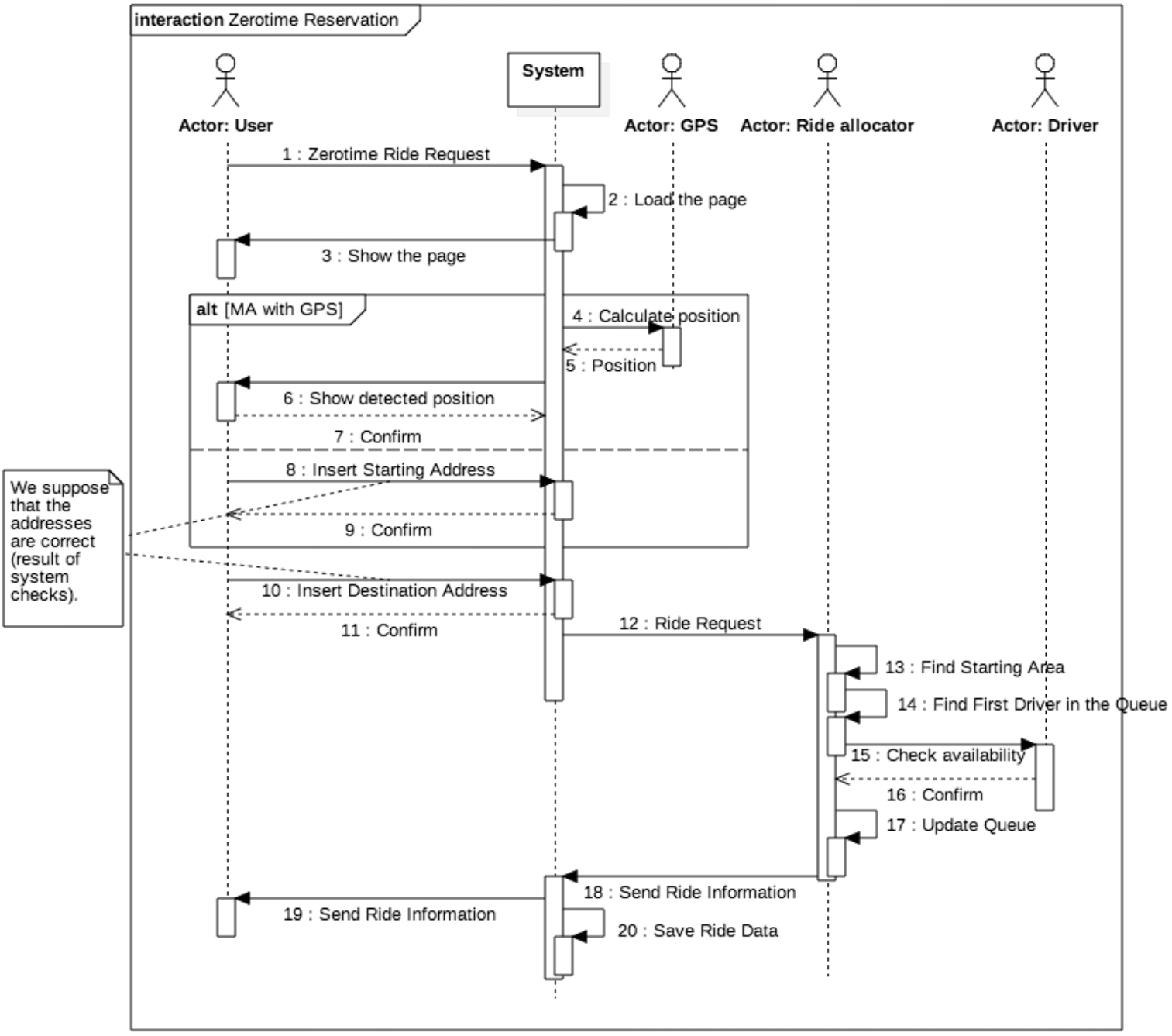


Figure 3.6: Sequence diagram for the zero time reservation

### 3.2.8 Future reservation

| Actors | User (or Driver) |
|---|---|
| Preconditions | The visitor is logged into the system. |
| Execution Flow | 1. The user clicks on future reservation.<br>2. The system shows him a form.<br>3. The user fulfills the starting and the destination addresses and the departure time. Afterwards he submits the form.<br>4. The system checks the correctness of the received data, then it confirms the data to the user and informs it about travelling time and price.<br>5. The system saves the ride's information. |
| Postconditions | The ride is saved on user's history and it is saved on the system. The system is able to perform the ride request ten minutes before the departure time. |
| Exceptions | The destination or the starting position or the departure time is missing: the ride is not sent to the system.<br>The starting or the destination address is invalid. The departure time is not between 1 hour and 30 days from the current time. In both cases the system rejects the ride. |

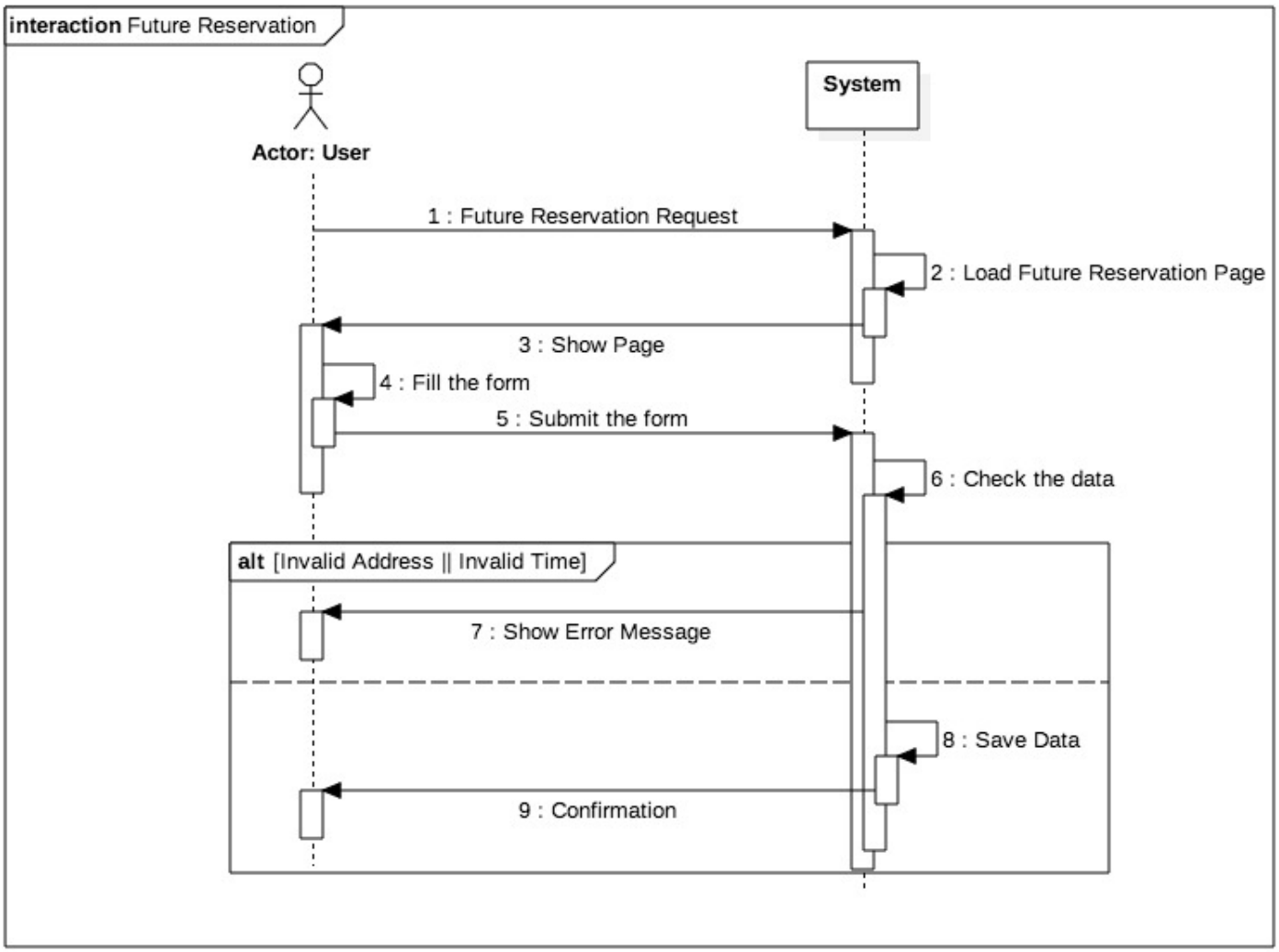


Figure 3.7: Sequence diagram for the future reservation

### 3.2.9 Start waiting time

Note: every time a driver ends a ride, he has to use this functionality.

| Actors | Driver |
|---|---|
| Preconditions | The driver is logged in the system. The driver is into his working hours, thus he has inserted a work shift that contains the current time. [If he is notifying the system that he is no longer available the driver is inserted in the queue related to his current position.] |
| Execution Flow | 1. The driver clicks the start waiting time button.<br>2. The system shows him a map and an input field for address.<br>3. The driver can click on a map area (the map was previously splitted in all the city areas and has a default selection on the position detected by GPS) or insert manually his address.<br>4. The system involves the ride allocator. The latter one adds [remove] the driver to the queue of driver's current area. Then the ride allocator informs the system about the result of the operations.<br>5. The system saves the data received by the ride allocator and confirms the operation to the driver. Only when the driver is added to a queue an additional information is given: the number of drivers before him. |
| Postconditions | The driver is put in the tail [put off] of the queue of the area where he is.<br>The driver isn't put in any other queue. |
| Exceptions | The GPS doesn't work: the system gives the driver the possibility of manually insert his address (and links it with the related area). |

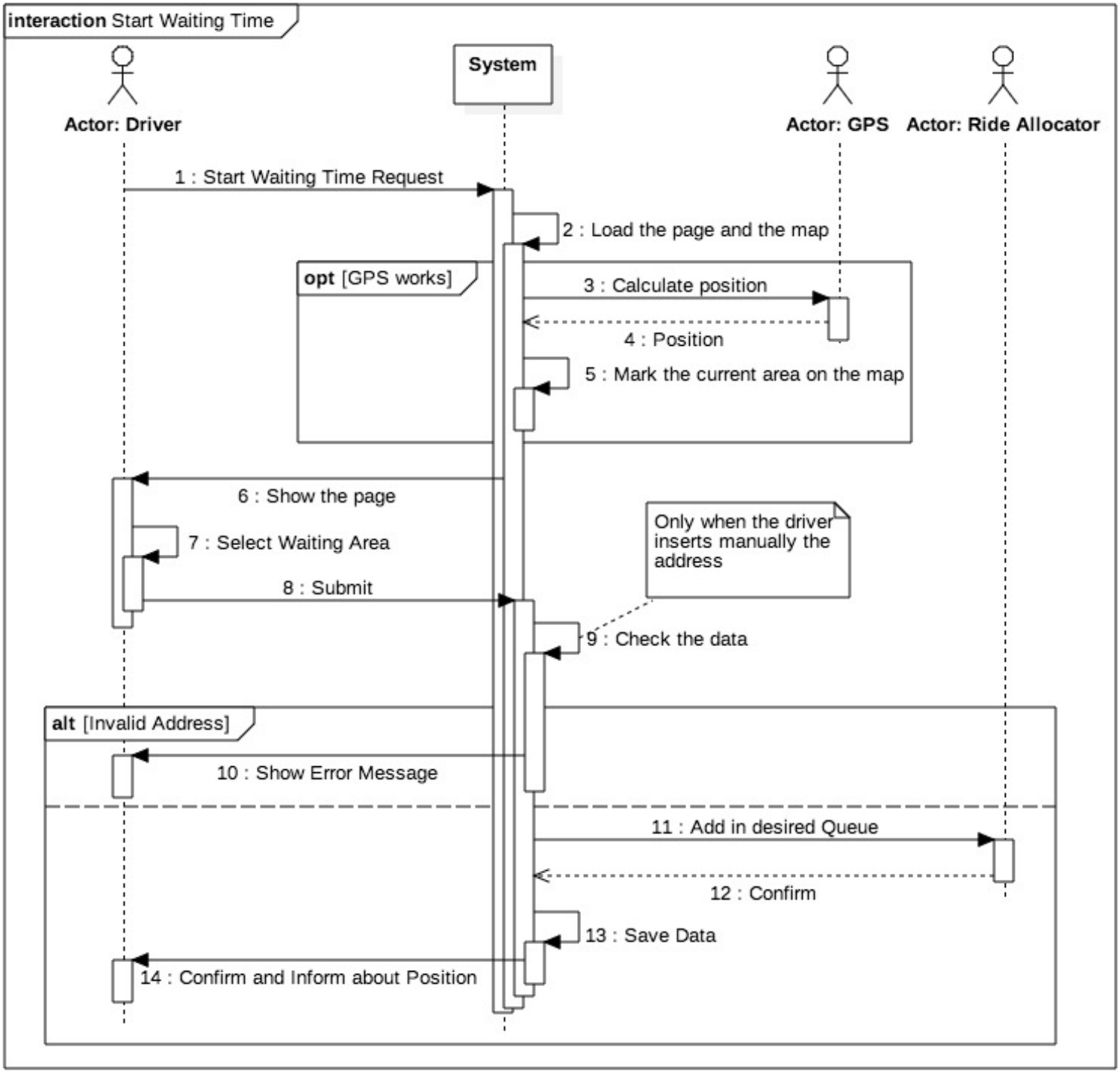


Figure 3.8: Sequence diagram for the start waiting time function. The case of not availability is not represented in this diagram, but the only change is on message 11. (Remove from the current Queue)

### 3.2.10 Work shifts management

| Actors | Driver |
|---|---|
| Preconditions | The driver is logged in the system. |
| Execution Flow | 1. The driver clicks on work shifts management button.<br>2. The system shows him the current month table (the rows represents the hours, the columns the days) where are visible all previous inserted shifts.<br>3. The driver fills the desired work hours by clicking on the cells, then submits the forms.<br>4. The system validates the data and notifies it the user. |
| Postconditions | The new shifts are saved in the system according to the laws and the work contract of the driver (e.g. maximum daily hours, number of work hours in a month and so on). No shifts are overlapped. |
| Exceptions | The driver inserts less hour than the minimum requested in a month: the system accepts the new shifts and notifies to driver that he has to insert more shifts. The driver insert illegal shifts (opposed to the laws or opposed to cab company rules or overlapped): the system refused them and notifies the driver. |

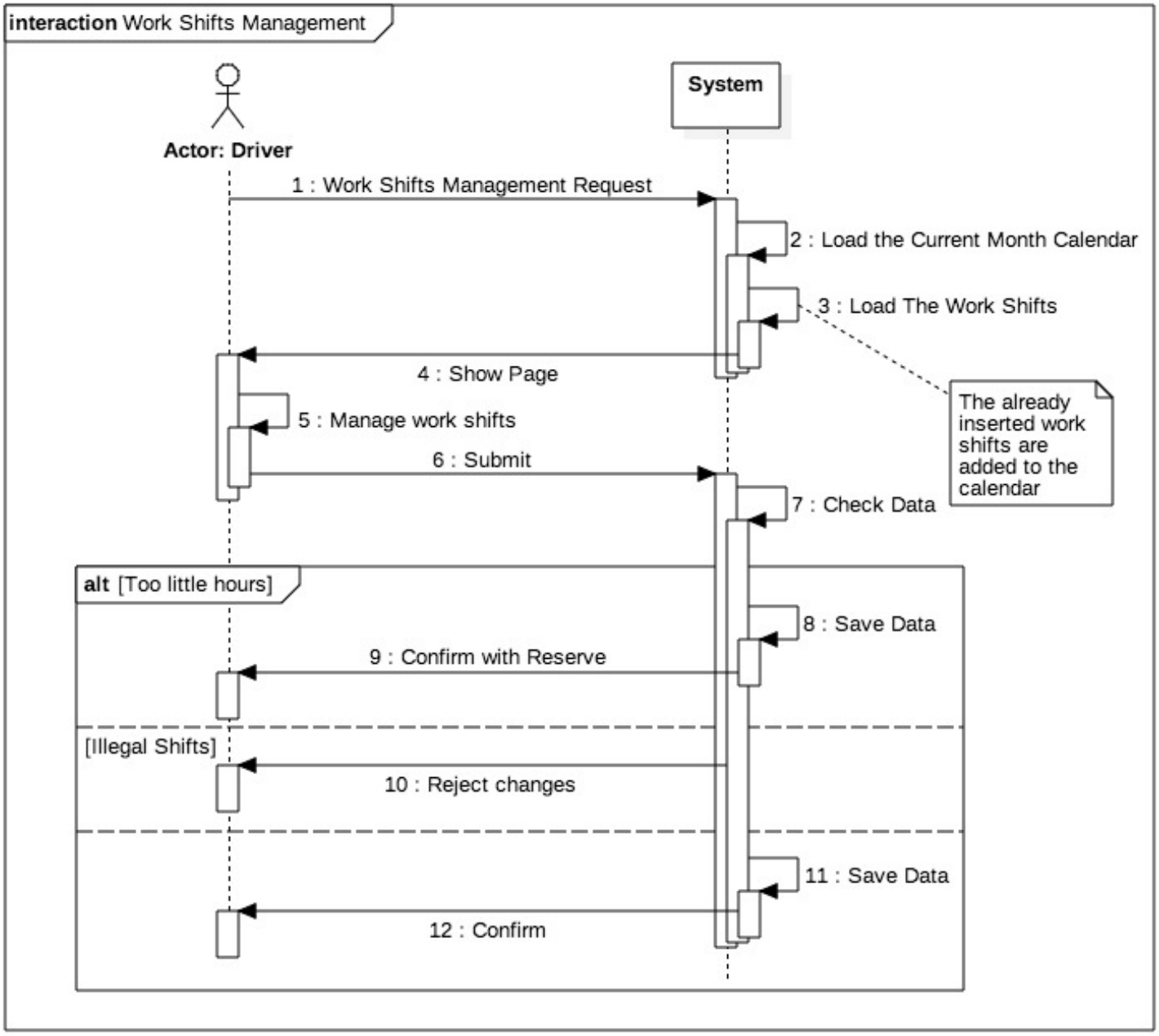


Figure 3.9: Sequence diagram for the work shifts management

### 3.2.11 Check the reservations

| Actors | User |
|---|---|
| Preconditions | The user is logged in the system. All the historical (already done) and the zerotime reservations can be showed only. |
| Execution Flow | 1. The user clicks on the Check reservations button.<br>2. The system shows the user all historical, zerotime and future reservations in descending order by date. In particular, for the latter ones the system allows to modify or cancel them by a button.<br>3. If the user wants to modify a reservation he clicks on related button and system redirects him to the future booking page (already filled in all boxes with current data).<br>4. If the user wants to cancel a booking he clicks on related button. Then the system shows him a confirmation message. |
| Postconditions | All the historical (or zerotime) reservations are saved again without changes. All future reservations are still in the system unless they are cancelled or modified by user. In the other cases the reservation are correctly modified or deleted. |
| Exceptions | The user tries to modify or cancel a future reservation less that 15 minutes before the departure time: the system rejects the operation and informs the user. |

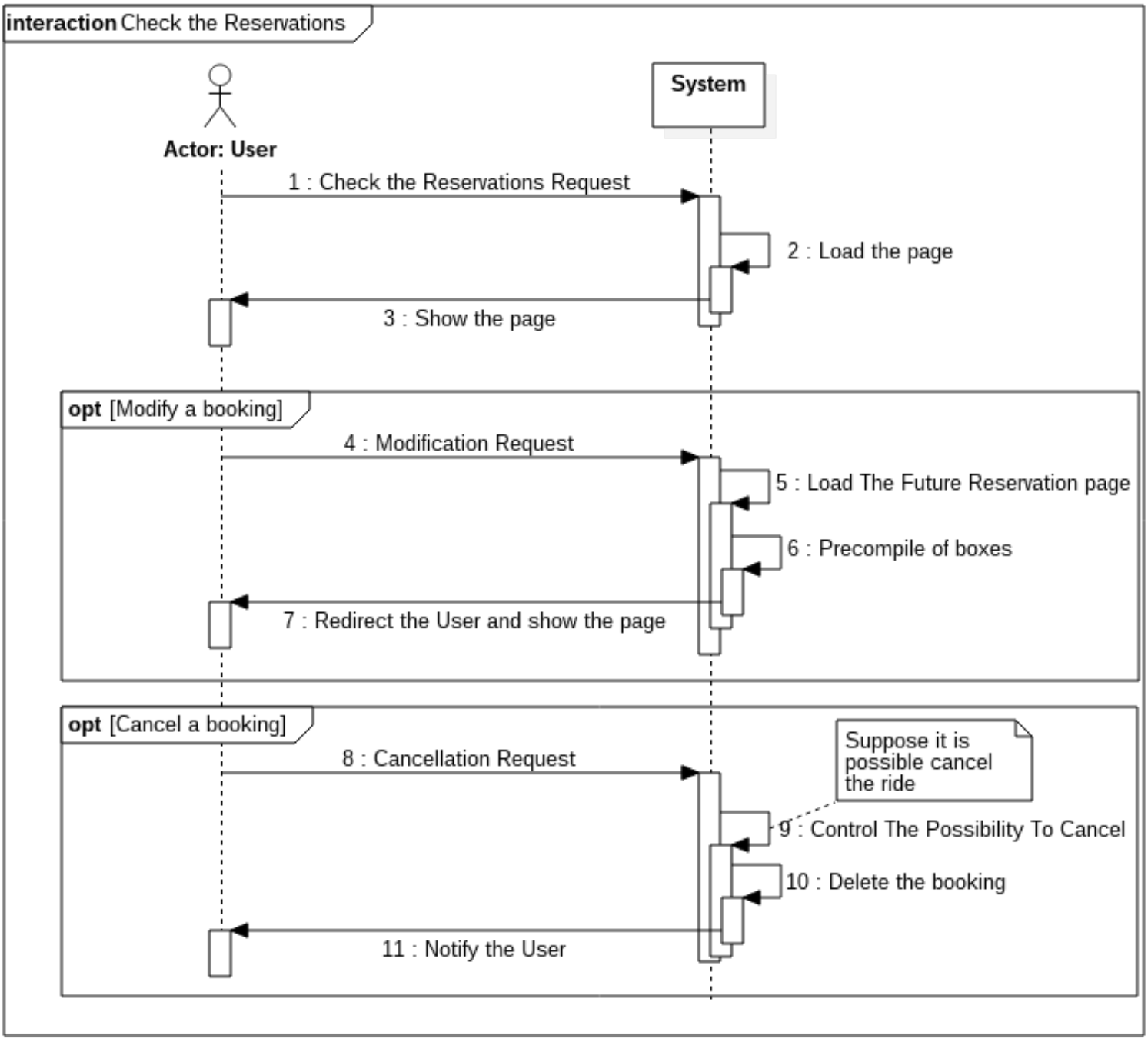


Figure 3.10: Sequence diagram for the check reservations function

### 3.2.12 Read the alerts

| | |
|---|---|
| Actors | User |
| Preconditions | The user is logged in the system. |
| Execution Flow | 1. The User clicks on Read the alerts button.<br>2. The system shows the user all his alerts in descending date order. |
| Postconditions | The alerts are not changed or removed by the system. |
| Exceptions | None. |

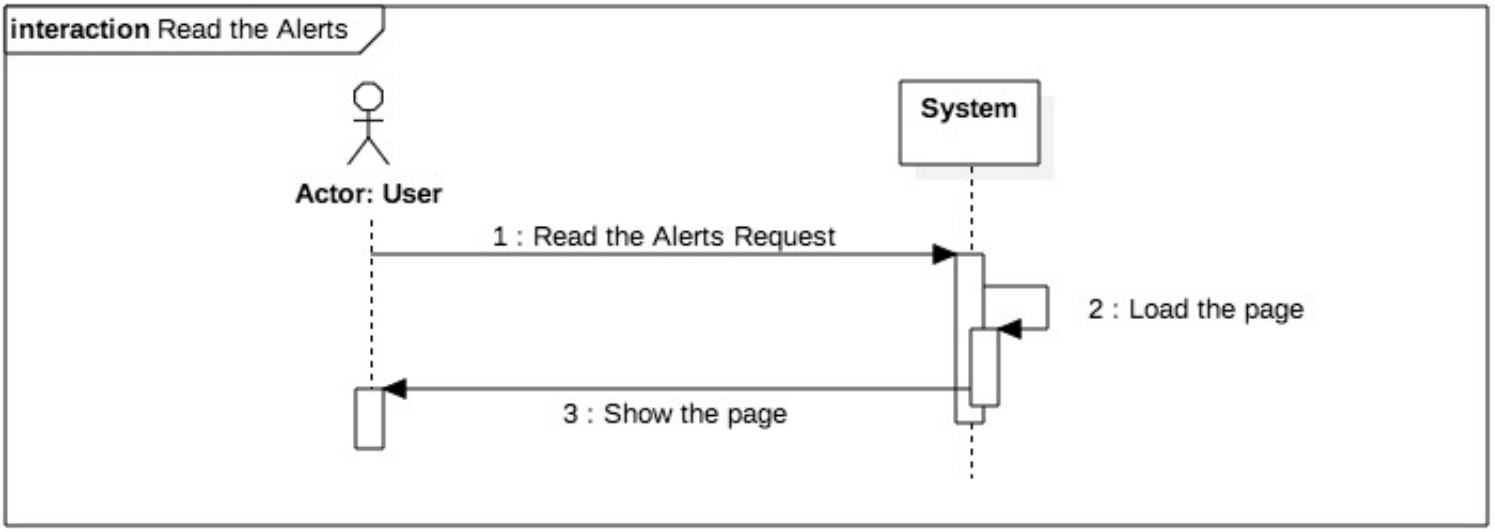


Figure 3.11: Sequence diagram for the read the alerts function

## 3.3 Entities Behaviour

In this section the behaviour of some entities presented in Figure 3.2 is exposed using UML state chart diagrams.

### 3.3.1 Zerotime Ride Class

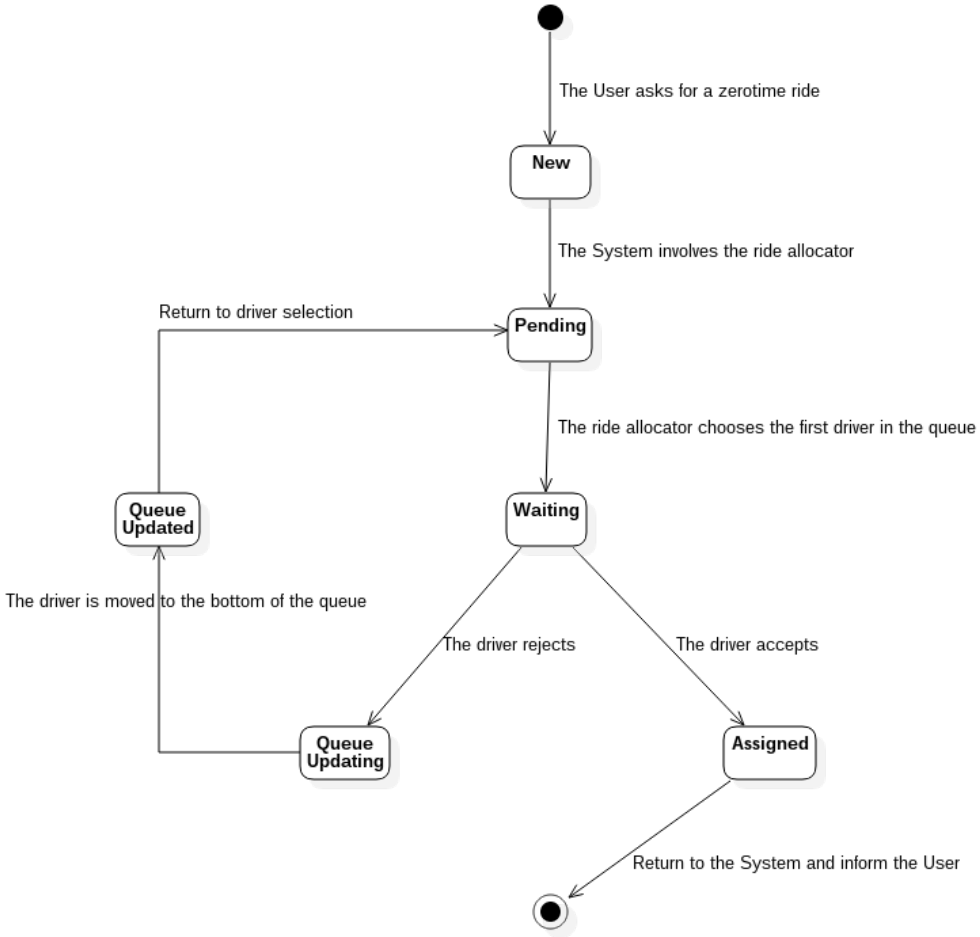


Figure 3.12: The behaviour of the Zerotime ride class presented via UML state diagram

### 3.3.2 Future Ride Class

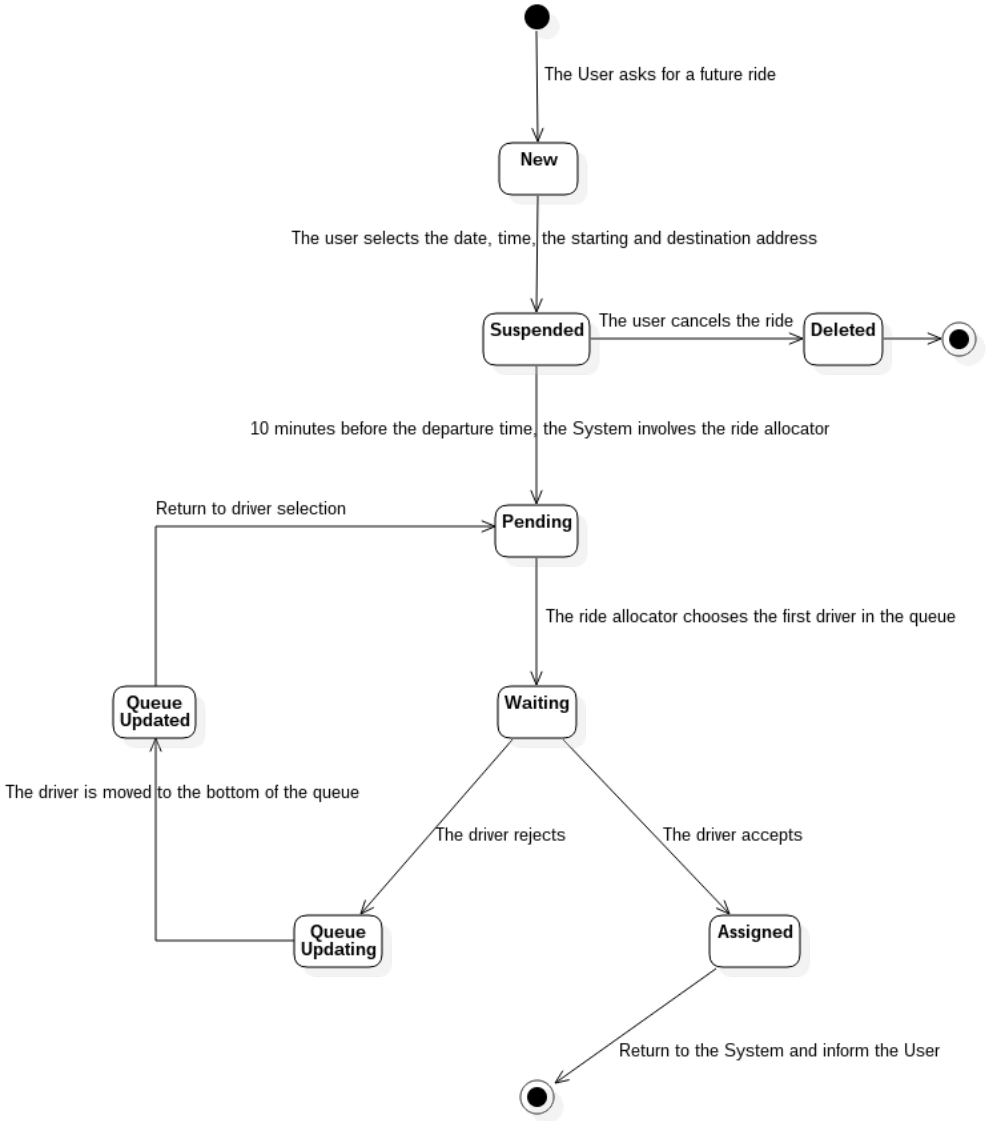


Figure 3.13: The behaviour of the Future ride class presented via UML state diagram

# Chapter 4

# Alloy Modeling

In this chapter the consistency of the UML class diagram will be tested using Alloy Analyzer. In the following paragraph there will be shown first the code used, then the response of Alloy, using assertions, predicates and an example of a possible world.

## 4.1 Alloy Code

Here the code used is presented.

```
module myTaxiService

/**** Initial Comment at this model ****/
/*** The model is based on the UML class diagram shown in previous chapter and
    it tries to validate it. There is a little difference. The data and the time
    are modelled as integer numbers representing a timestamp from a certain
    initial date that are not relevant in this model. The reason for this choice
    is to use the available functionalities of Alloy. A second note on the model
    is the following: a driver can only wait for a ride or drive. Hence, the
    workshifts are not modelled   ***/
/*** Important Observation: the Integer defined by Alloy has the maximum value
    equal to 7, so this is a limit to our model ***/


/*** CLASS DECLARATION ***/

abstract sig Ride {
        /** Timestamp for departure and arrival time **/
        departure : one Int,
```

```
        arrival : lone Int,
        /** Relations**/
        startingPosition : one Position,
        destination : one Position,
        customer : one User,
        driver : lone Driver
} { departure > 0&&arrival > 0}

sig ZerotimeRide extends Ride {
}

sig FutureRide extends Ride {
}

/** By definition Zerotime and Future are a partition of abstract class Ride **/

sig CabCompany {}

sig User {
        email : one GenericEmail,
        -- password : one GenericText,
        -- name : one GenericText,
        -- surname : one GenericText,
        -- birthday : one Date,
        -- cityOfResidence : lone GenericText,
        taxCode : one GenericText,
        /** Relations**/
        currentPosition : lone Position,
        alert : set Alert
}

sig Driver extends User {
        cabCarCode: one GenericText,
        -- cabCompany : one cabCompany,
        /** Relations**/
        currentArea : lone Area
}

sig Position {
        -- GPSCoordinates : one Float
        -- city : lone GenericText,
        -- street : lone GenericText,
        -- civicNumber : lone Integer
        area : one Area
}

sig Area {
        -- name : one GenericText
        /** Relations**/
```

```
        availableDrivers: set Driver
}

sig Alert {
        -- receiver : one User,
        -- message : one GenericText,
        -- date : lone Date,
        -- time : lone Time
}

/** The current time **/
one sig CurrentTime {
        current : one Int
} { current > 0}

sig GenericText {}

sig GenericEmail {}

/** Definition of Boolean type **/
abstract sig Boolean {}

one sig True extends Boolean {}

one sig False extends Boolean {}

/** Note: The commented attributes are not relevant in this model **/

/*** DEFINITION OF THE CONSTRAINTS ***/

-- the tax code is unique
fact taxCodeUnicity {
        no disj u1 , u2 : User | u1.taxCode = u2.taxCode
}

-- the email is unique
fact emailUnicity {
        no disj u1 , u2 : User | u1.email = u2.email
}

-- the cabCarCode is unique
fact cabCarCodeUnique {
        no disj d1 , d2 : Driver | d1.cabCarCode = d2.cabCarCode
}

-- the driver exists for a zerotime ride
fact driverExistence {
        all ztr : ZerotimeRide | #ztr.driver = 1
}
```

```
-- the driver and the customer are two different people
fact differentDriverAndCustomer {
        no r : Ride | r.customer = r.driver
}

-- the cabman can drive at most one rides at the same time
fact taxiDriverUbiquity {
        no disj r1 , r2 : Ride | r1.departure >= r2.departure&&r1.departure =<
            r2.arrival&&r1.driver = r2.driver
}

-- an user cannot be on two rides at the same time
fact userUbiquity {
        no disj r1 , r2 : Ride | r1.departure >= r2.departure&&r1.departure =<
            r2.arrival&&(r1.customer = r2.customer || r1.customer = r2.driver ||
             r1.driver = r2.customer)
}

-- correctness of time : the departure is before the arrival (at least one unit
    of time)
fact noBackTime {
        no r : Ride | r.departure >= r.arrival
}

-- a zerotime ride is already departed
fact correctnessOfZeroTime {
        all ztr : ZerotimeRide | ztr.departure =< CurrentTime.current
}

-- a zerotime ride has always an arrival time
fact zerotimeRideAlwaysArrive {
        all ztr : ZerotimeRide | #ztr.arrival = 1
}

-- a future ride is after now
/** Note that in the statechart diagram, 10 minutes before the departure, a
    future ride becomes zerotime and it is managed **/
fact correctnessOfFuture {
        all fr : FutureRide | fr.departure > CurrentTime.current
}

-- no ride is cyclic (or null): the starting position is different from the
    destination
fact noSelfRide {
        no r : Ride | r.startingPosition = r.destination
}

-- if the cabman is driving or traveling he isn't in any queue
```

```
fact waitOrWork {
        all d : Driver | ( driverBusy [d] = True || driverTraveling[d] = True )
            implies no a : Area | d.currentArea = a
}

-- if the cabman is not driving or traveling he is in a queue
fact waitOrWork2 {
        all d : Driver | driverBusy[d] = False&&driverTraveling[d] = False
            implies one a : Area | d.currentArea = a
}

-- The driver is into the correct queue
fact driverInCorrectArea {
        all d : Driver , a : Area | d in a.availableDrivers <=> d.currentArea = a
}

/*** FUNCTIONS ***/

-- Return true if a ride is in progress
fun nowRide [r : Ride] : one Boolean {
        ( CurrentTime.current >= r.departure&&CurrentTime.current =< r.arrival
            ) implies {True} else {False}
}

-- Return true if a driver is working
fun driverBusy [d : Driver] : one Boolean {
        ( some r : Ride | nowRide [r] = True&&r.driver = d ) implies {True}
            else {False}
}

-- Return true if a driver is traveling as customer
fun driverTraveling [d : Driver] : one Boolean {
        ( some r : Ride | nowRide [r] = True&&r.customer = d ) implies {True}
            else {False}
}

/*** ASSERTIONS AND PREDICATES ***/

-- A driver is always into a queue. It is a false assertion and we want to proof
    it.
assert driversInTheQueue {
        all d : Driver | one a : Area | d in a.availableDrivers
}

check driversInTheQueue for 5 but exactly 3 Area , 2 FutureRide, exactly 3 Driver

-- A person that is on a zerotime ride can have booked a future ride (defined as
    the opposite assertion to find a counterexample)
assert multipleReservation {
```

```
        no u : User | some ztr : ZerotimeRide , fr : FutureRide | ztr.customer =
            u&&fr.customer = u
}

check multipleReservation for 5

-- A driver who is now working (driving) is not customer of any other ride
assert noUbiquity {
        all d : Driver | ( driverBusy [d] = True ) implies no r : Ride |
        ( nowRide[r] = True&&r.customer = d )
}

check noUbiquity for 5

pred noDriverAsCustomer () {
        no d : Driver | some r : Ride | r.customer = d
        #ZerotimeRide = 2
        #FutureRide = 1
        #Driver = 2
        #User = 5
        #Area = 3
        one a : Area | a.availableDrivers!=none
}

run noDriverAsCustomer for 5

pred show () {
        #ZerotimeRide = 2
        #FutureRide > 1
        #Driver = 2
        #User > 3
}

run show for 4
```

## 4.2 Alloy Response

Here the alloy response to the model is shown.

```
Executing "Check driversInTheQueue for 5 but exactly 3 Area, 2 Future
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   7123 vars. 454 primary vars. 15513 clauses. 369ms.
   Counterexample found. Assertion is invalid. 81ms.

Executing "Check multipleReservation for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8431 vars. 541 primary vars. 18192 clauses. 320ms.
   Counterexample found. Assertion is invalid. 66ms.

Executing "Check noUbiquity for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8976 vars. 536 primary vars. 19819 clauses. 242ms.
   No counterexample found. Assertion may be valid. 176ms.

Executing "Run noDriverAsCustomer for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8437 vars. 526 primary vars. 18365 clauses. 180ms.
   Instance found. Predicate is consistent. 89ms.

Executing "Run show for 4"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5807 vars. 376 primary vars. 12745 clauses. 134ms.
   Instance found. Predicate is consistent. 72ms.

5 commands were executed. The results are:
   #1: Counterexample found. driversInTheQueue is invalid.
   #2: Counterexample found. multipleReservation is invalid.
   #3: No counterexample found. noUbiquity may be valid.
   #4: Instance found. noDriverAsCustomer is consistent.
```

```
#5: Instance found. show is consistent.
```

### 4.2.1 Counterexamples

Two of the assertions we wrote are, in our idea, wrong assertions for the following reasons: the first one shows the correctness of a property of the model (a driver can be into a queue only when is waiting for a ride), the second one use the counterexample to proof the validity of another model property (an user that is traveling on a zerotime ride can book a future ride). In fact for the latter one there is no way to write the opposite assertion.

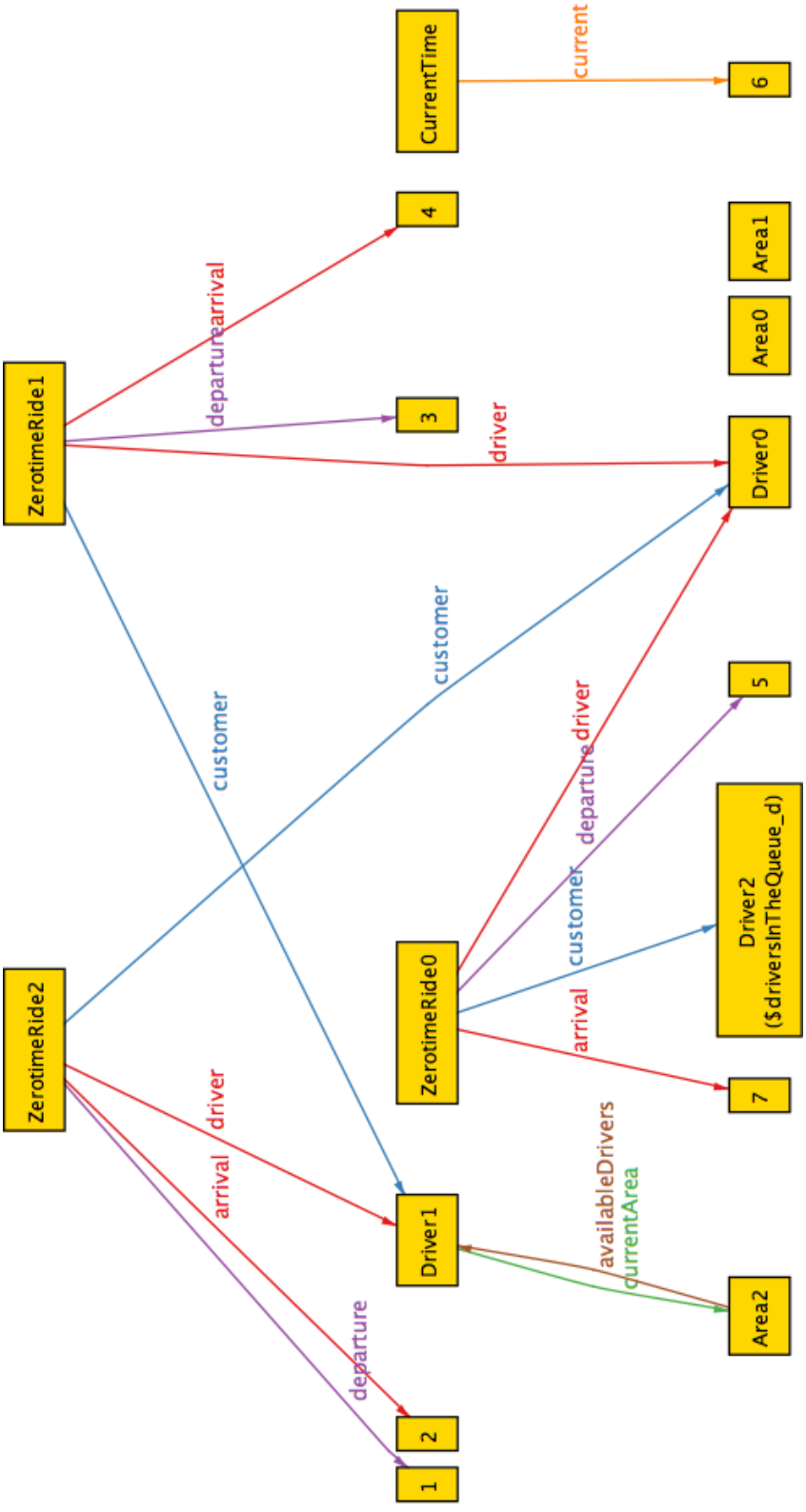


Figure 4.1: First Counterexample representation.

**Note**: It is immediately visible that Driver1 is the driver assigned to ZerotimeRide2 and he is inserted into the Area2 queue. This is possible because the ride has already finished, so according to the model the driver must be waiting for another ride.

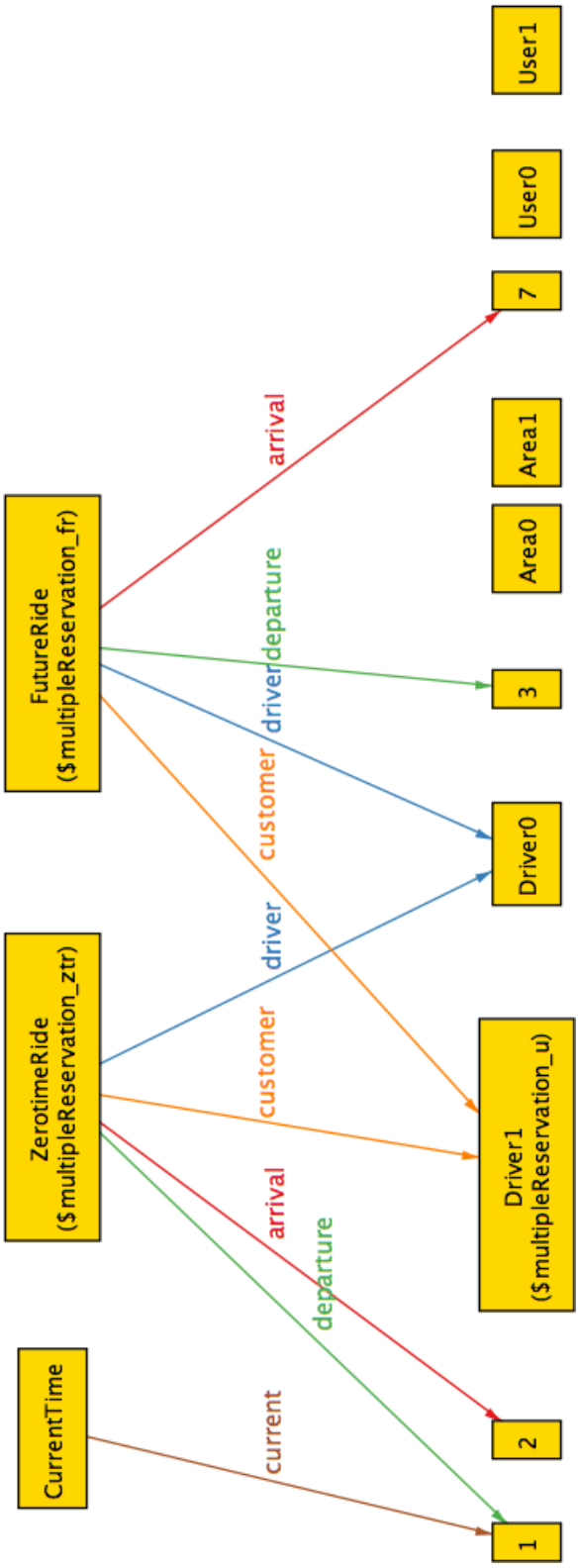


Figure 4.2: Second Counterexample representation

### 4.2.2 Alloy World

In this paragraph are showed two worlds.
The former one shows a "restricted" world where only the users can be a customer. In the latter one the strange case of a driver who is traveling as customer in an other ride is allowed. Besides, the positions are shown to give a more complete and precise idea of the world.

In all the images, the classes which are defined only in the Alloy model are hidden in order to have a clearer images. The unicity of email and taxCode of each User and the unicity of cab carCode are correctly defined into the model, but they are not as so important as the kinds of the rides and the management of the areas and the queues.

**World 1**

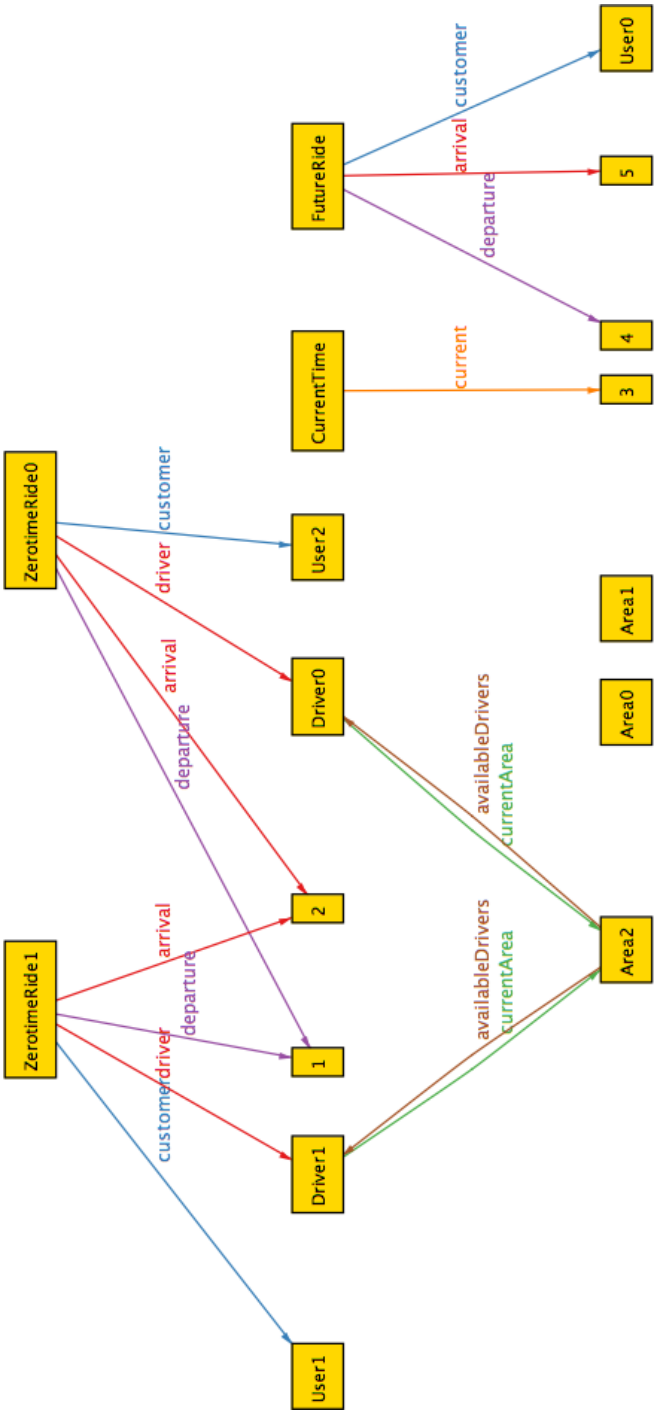


Figure 4.3: World created from the first predicate noDriverAsCustomer

**World 2**

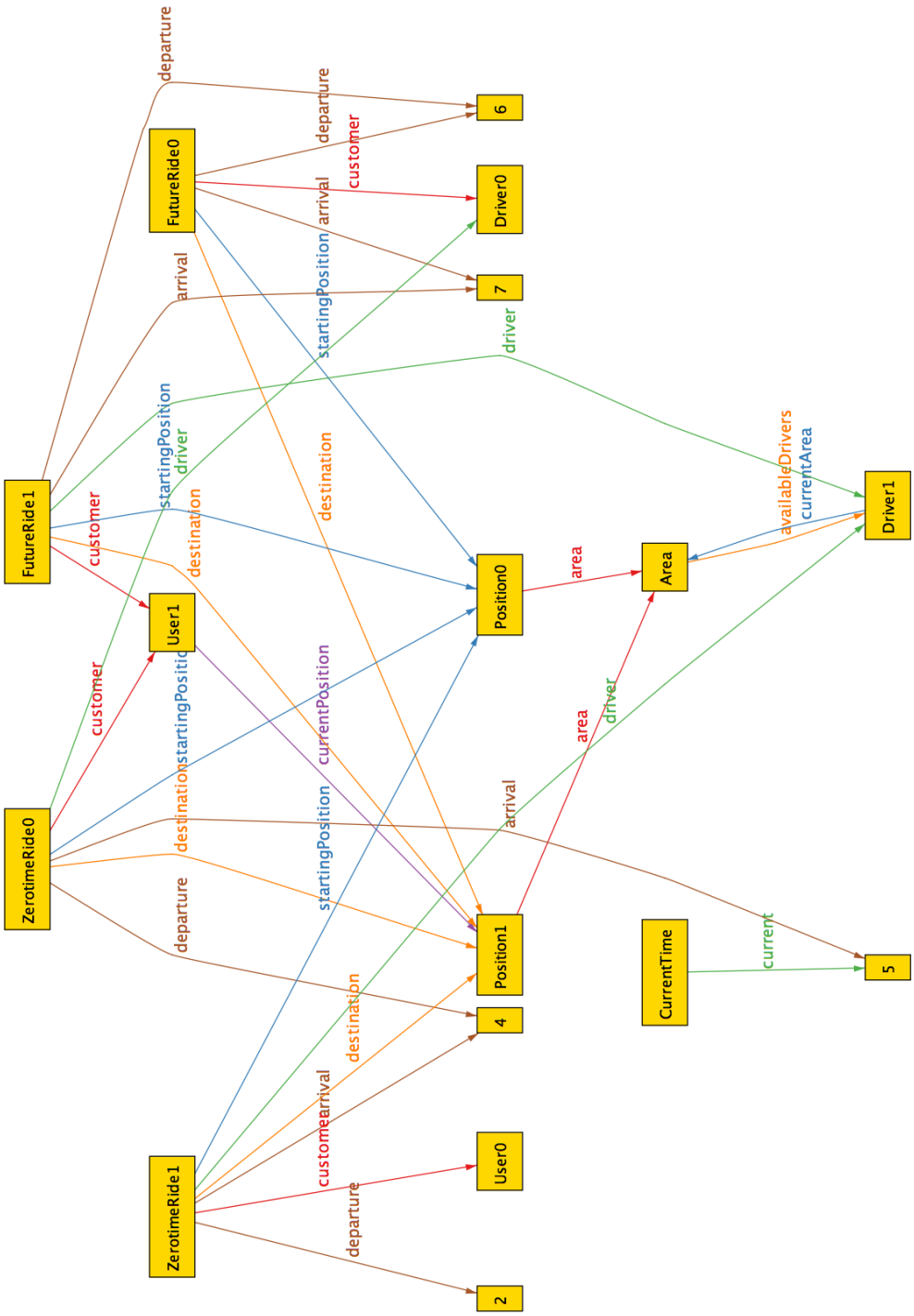


Figure 4.4: World created from the predicate show

# Chapter 5

# Other info

This chapter contains information about the used tools and the hours of work by the members of the working group.

## 5.1 Working hours

| Date | Costanzo's hours | Disabato's hours |
|---|---|---|
| 2015/10/16 | 1.30h | 3h |
| 2015/10/21 | - | 1h |
| 2015/10/22 | 2h | 1h |
| 2015/10/23 | 1.30h | 2.30h |
| 2015/10/26 | 1.30h | 2h |
| 2015/10/27 | 1h | 1.30h |
| 2015/10/28 | 1h | 2h |
| 2015/10/29 | 2h | 1h |
| 2015/10/30 | 5h | 3h |
| 2015/10/31 | 1.30h | - |
| 2015/11/01 | 1h | - |
| 2015/11/02 | 6h | 2h |
| 2015/11/03 | 2h | 2h |
| 2015/11/04 | 1h | 2h |
| 2015/11/05 | - | 4h |
| 2015/11/06 | 2h | 2h |
| **Total** | 29h | 29h |

## 5.2 Tools

In this first requirements study phase the following tools were used:

- Google Docs
- LaTeX and TeXMaker editor
- Alloy Analizer
- starUML

## 5.3 Revisions

- **version 1.2**: We have added a tool used that we had forgotten, the release date and we have corrected a few grammatical and lessical errors.