

POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2

MYTAXISERVICE

Requirement Analysis and Specification Document

Author:
Andrea TIRINZONI

Author:
Matteo PAPINI

November 5, 2015

Contents

1	Introduction	5
1.1	Purpose	5
1.2	Scope	5
1.3	Definitions, acronyms and abbreviations	6
1.3.1	Definitions	6
1.3.2	Acronyms	7
1.3.3	Abbreviations	7
1.4	Goals	7
1.5	Domain assumptions	8
1.6	References	8
1.7	Overview	8
2	Overall description	9
2.1	Product perspective	9
2.1.1	System interfaces	9
2.1.2	User interfaces	9
2.1.3	Hardware interfaces	9
2.1.4	Software interfaces	9
2.2	Product functions	9
2.3	User characteristics	9
2.4	Constraints	10
2.4.1	Hardware limitations	10
2.4.2	Parallel operation	10
2.4.3	Safety and security considerations	10
2.5	Assumptions and dependencies	10
3	Specific requirements	11
3.1	External interface requirements	11
3.1.1	Passenger interface	11
3.1.2	Driver interface	15
3.2	Functional requirements	18
3.2.1	[G1] Every time a passenger calls a taxi, she gets one . . .	18
3.2.2	[G2] Passengers know the maximum waiting time for the taxi they have been assigned to	18
3.2.3	[G3] Passengers are able to identify the taxi they have been assigned to	18
3.2.4	[G4] Taxi drivers are able to recognize the passenger they have been assigned to	18
3.2.5	[G5] Taxi drivers are forwarded only calls in which the meeting point is in their current zone	19
3.2.6	[G6] The taxi driver who has been waiting for the longest time in a certain zone is the first to receive a request from that zone	19
3.2.7	[G7] Taxi drivers receive requests only when they are free	19

3.2.8	[G8] Passengers can reserve a taxi only for a time at least two hours from the reservation	20
3.2.9	[G9] External developers have the possibility to use the taxi call and reservation function of MyTaxiService in their applications	20
3.2.10	[G10] Each reserved taxi arrives at the meeting point at most two minutes late	20
3.2.11	[G11] Taxi drivers receive only and all the requests relative to the taxi they are currently driving	20
3.3	Nonfunctional requirements	21
3.3.1	Performance	21
3.3.2	Availability	21
3.3.3	Security	21
3.3.4	Portability	21
3.4	Scenarios	21
3.4.1	Taxi call	21
3.4.2	Taxi reservation	22
3.4.3	Taxi driver routine	22
3.4.4	External developer point of view	22
3.4.5	Registration	23
3.5	Use case diagram	25
3.6	Use case 1	26
3.6.1	Name	26
3.6.2	Actors	26
3.6.3	Entry condition	26
3.6.4	Flow of events	26
3.6.5	Exit condition	26
3.6.6	Exceptions	26
3.6.7	Special requirements	26
3.7	Use case 2	28
3.7.1	Name	28
3.7.2	Actors	28
3.7.3	Entry condition	28
3.7.4	Flow of events	28
3.7.5	Exit condition	28
3.7.6	Exceptions	28
3.7.7	Special requirements	28
3.8	Use case 3	30
3.8.1	Name	30
3.8.2	Actors	30
3.8.3	Entry condition	30
3.8.4	Flow of events	30
3.8.5	Exit condition	30
3.8.6	Exceptions	30
3.8.7	Special requirements	30
3.9	Use case 4	32

3.9.1	Name	32
3.9.2	Actors	32
3.9.3	Entry condition	32
3.9.4	Flow of events	32
3.9.5	Exit condition	32
3.9.6	Exceptions	32
3.9.7	Special requirements	32
3.10	Use case 5	34
3.10.1	Name	34
3.10.2	Actors	34
3.10.3	Entry condition	34
3.10.4	Flow of events	34
3.10.5	Exit condition	34
3.10.6	Exceptions	34
3.10.7	Special requirements	34
3.11	Use case 6	36
3.11.1	Name	36
3.11.2	Actors	36
3.11.3	Entry condition	36
3.11.4	Flow of events	36
3.11.5	Exit condition	36
3.11.6	Exceptions	36
3.11.7	Special requirements	36
3.12	Use case 7	38
3.12.1	Name	38
3.12.2	Actors	38
3.12.3	Entry condition	38
3.12.4	Flow of events	38
3.12.5	Exit condition	38
3.12.6	Exceptions	38
3.12.7	Special requirements	38
3.13	Use case 8	40
3.13.1	Name	40
3.13.2	Actors	40
3.13.3	Entry condition	40
3.13.4	Flow of events	40
3.13.5	Exit condition	40
3.13.6	Exceptions	40
3.13.7	Special requirements	40
3.14	Use case 9	42
3.14.1	Name	42
3.14.2	Actors	42
3.14.3	Entry condition	42
3.14.4	Flow of events	42
3.14.5	Exit condition	42
3.14.6	Exceptions	42

3.14.7	Special requirements	42
3.15	Use case 10	44
3.15.1	Name	44
3.15.2	Actors	44
3.15.3	Entry condition	44
3.15.4	Flow of events	44
3.15.5	Exit condition	44
3.15.6	Exceptions	44
3.15.7	Special requirements	44
3.16	Class diagram	46
4	Appendix	47
4.1	Alloy	47
4.1.1	Signatures	47
4.1.2	Cardinality Facts	50
4.1.3	Facts	51
4.1.4	Cardinality Assertions	53
4.1.5	Assertions	54
4.1.6	Assertion checks	54
4.1.7	Run	55
4.1.8	Generated worlds	56
4.2	Used software	59
4.3	Spent Hours	59

1 Introduction

1.1 Purpose

This is the Requirement Analysis and Specification Document (RASD) for the myTaxiService system. This document precisely describes the given problem, the customer's goals and the features of the system to be implemented. It is intended to be the starting point for developers and a common agreement with the customer.

1.2 Scope

The aim of MyTaxiService is to improve the public taxi service of Milan, both from an administration perspective and from the point of view of passengers. A brief description of the system to be implemented, as outlined by the customer, is now provided.

All services addressed to passengers are accessible via both a mobile app and a web application:

- Taxi call: the passenger requests a taxi for as soon as possible, by specifying the meeting point; as a confirmation, she receives the code of the incoming taxi and the maximum waiting time;
- Taxi reservation: the passenger requests a taxi for a precise time and place. She has to specify the meeting point, the destination and the time of the meeting with at least two hours in advance. As a confirmation, she receives the code of the incoming taxi approximately ten minutes before the meeting time.

Users are able to sign up to the service, by using either the mobile app or the web application and providing a phone number and a password. Users can check the system notifications from both interfaces.

Taxi drivers whose state is set to free receive requests through a mobile app. They also use the app to perform the following actions:

- Changing their state from free to busy or vice versa;
- Accepting requests;
- Refusing requests.

The system manages the requests in an efficient way:

The city is divided into zones having an area of approximately 2 km². Each zone is associated to a queue containing all the taxi drivers currently in that zone. The position of the taxis is provided by the drivers' app through GPS. When a taxi driver sets his state to free, he is automatically added to the queue of the zone he is currently in.

Each new request is forwarded to the taxi drivers queueing in the zone containing the meeting point, beginning with the driver that has been free for the most time, until one of them accepts it.

MyTaxiService also provides some APIs concerning the call and the reservation functionalities, which can be used by external developers in their projects.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

Precise definitions of some recurrent terms and expressions are provided:

- **User**: someone who uses the MyTaxiService mobile app for passengers or the MyTaxiService web application.
- **Guest**: a user who is not registered.
- **Passenger**: a registered user.
- **Taxi**: a vehicle used to provide cab service in the city, denoted by a unique code assigned by the Government.
- **Taxi driver**: a person who is granted access to the MyTaxiService mobile app for drivers by the Government. A logged in taxi driver, at any given time, is associated to exactly one taxi, so in the document we will refer to the taxi by addressing the driver and vice versa.
- **Taxi ID**: the code assigned to a certain taxi.
- **Meeting point**: the place in the city where the passenger wants to have a taxi sent to.
- **Zone**: an area of the city of approximately 2 km²;
- **Waiting time**: the amount of time the passenger has to wait from when she receives the confirmation of an incoming taxi to when the taxi reaches the meeting point.
- **Request (to a taxi driver)**: the demand to a taxi driver to reach a meeting point as soon as possible.
- **State (of the taxi driver)**: can be either:
 - Free: the taxi driver is ready to receive requests;
 - Busy: the taxi driver cannot or does not want to receive requests;
- **Taxi call**: the act of requesting a taxi for as soon as possible. Sometimes simply referred to as “call”.
- **Taxi reservation**: the act of requesting a taxi for a certain time in the future. Sometimes simply referred to as “reservation”.

- **Taxi allocation:** the operation performed by the central system to allocate a taxi either for a call or a reservation.
- **Central system:** the server part of the software.
- **Passenger application:** a term referring both to the mobile app for passengers and to the web application.
- **Driver app:** the mobile app used by taxi drivers.

1.3.2 Acronyms

The following acronyms will be used in this document:

- **RASD:** Requirement Analysis and Specification Document;
- **API:** Application Programming Interface;
- **GPS:** Global Positioning System;

1.3.3 Abbreviations

- **App:** Application;

1.4 Goals

The goals of myTaxiService are the following:

- **[G1]** Every time a passenger calls a taxi, she gets one;
- **[G2]** Passengers know the maximum waiting time for the taxi they have been assigned to;
- **[G3]** Passengers are able to identify the taxi they have been assigned to;
- **[G4]** Taxi drivers are able to recognize the passenger they have been assigned to;
- **[G5]** Taxi drivers are forwarded only calls in which the meeting point is in their current zone;
- **[G6]** The taxi driver who has been waiting for the longest time in a certain zone is the first to receive a request from that zone;
- **[G7]** Taxi drivers receive requests only when they are free;
- **[G8]** Passengers can reserve a taxi only for a time at least two hours from the reservation;
- **[G9]** External developers have the possibility to use the taxi call and reservation function of MyTaxiService in their applications;

- [G10] Each reserved taxi arrives at the meeting point at most two minutes late;
- [G11] Taxi drivers receive only and all the requests relative to the taxi they are currently driving.

1.5 Domain assumptions

The following domain assumptions will be considered:

- [D1] For each zone and at any time at least one taxi is available;
- [D2] When taxi drivers accept a call, they will get to the meeting point as soon as possible;
- [D3] Taxi drivers are able to get to any location in their current zone in at most 10 minutes;
- [D4] Taxi identifiers are clearly visible from the outside;
- [D5] Users provide correct personal data at registration;
- [D6] Each request is accepted by at least one taxi driver within 2 minutes;
- [D7] When free, a taxi driver does not change zone;
- [D8] Zones cover all the city territory and never intersect;
- [D9] Taxi GPS always work and provide the correct location;
- [D10] At login, taxi drivers provide the ID of the taxi they are actually driving;
- [D11] Taxi drivers redo login each time they change taxi.

1.6 References

- "Assignments 1 and 2.pdf"

1.7 Overview

The document is structured as follows:

- Part one gives a general description of the problem, states the customer's goals and specify the domain assumptions that will be considered;
- Part two describes the final product and the constraints to its development;
- Part three provides both functional and nonfunctional requirements and a description of the system in form of UML models.
- Appendix provides an Alloy model of the core part of the system.

2 Overall description

2.1 Product perspective

The application to be developed is self-contained and, thus, it is not part of an existing system. Its interfaces are listed below.

2.1.1 System interfaces

The application will not have any administration interface.

2.1.2 User interfaces

The application has two types of user interfaces: the one used by passengers to access the services (available both from mobile and web) and the one used by taxi drivers to communicate with the central system (available only on the mobile platform).

2.1.3 Hardware interfaces

The mobile user interfaces require only a cell phone with IOS, Android or Windows Phone. The web interface requires only a computer with a web browser installed. Both require Internet access. Taxi drivers' phones also require GPS. The central system requires a database to store its data.

2.1.4 Software interfaces

The application provides APIs to external developers, by which they can access the passenger services of the system from their projects.
The application uses Google Maps APIs.

2.2 Product functions

See part 3.

2.3 User characteristics

The application is intended to be easily accessible by anyone. Thus, the general user (i.e., a passenger) needs only to be familiar with smart phones and/or Internet browsing.

Each taxi driver will be trained to properly use the respective application. This last one will be very intuitive and user-friendly, thus no prior knowledge is required, nor is a hard training.

2.4 Constraints

2.4.1 Hardware limitations

The mobile application must be available on the most common phone operating system (IOS, Android, and Windows Phone).

2.4.2 Parallel operation

The central system must be able to handle simultaneous requests.

2.4.3 Safety and security considerations

User's privacy must be guaranteed (according to law).

2.5 Assumptions and dependencies

See part 1.

3 Specific requirements

3.1 External interface requirements

3.1.1 Passenger interface

As already mentioned, the passenger interface is available both from a web platform and a mobile app. Considering the only difference between the two is a matter of visualization, whereas the functions are the same, they will be treated as one. The term "screen" will be used to denote both a webpage and a mobile view. The provided functions are detailed below.

The first screen showed to a non-logged-in user allows her only to register or sign in. If the user chooses "register", the registration screen will be showed. Here the user must enter her cell phone number (auto-completed on the mobile platform), a password and other personal data (name and surname), and confirm by clicking "submit". If the user chooses "sign in", the login screen will be showed. Here the user must enter her cell phone number (again auto-completed on the mobile platform) and her password.

The screen showed to a logged-in user provides the following functionalities: taxi call, taxi reservation and notification view. In the taxi call screen, the user must specify the meeting point (by either selecting it manually on an embedded Google maps service or by getting the current location through GPS) and click "call now". In the taxi reservation screen, the user must specify the meeting point and the destination (using the same process described above), the meeting time and click "reserve now". In the notification screen, the messages from the central system are visualized.

Welcome screen



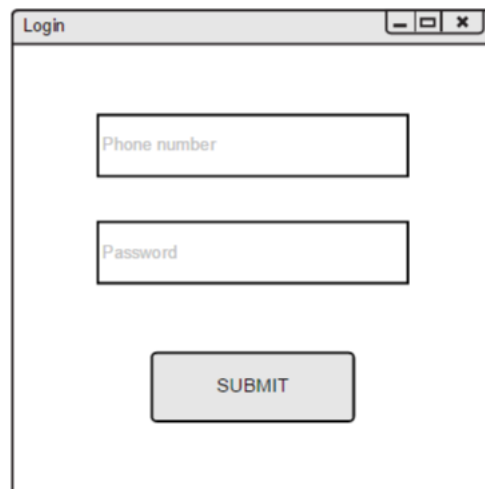
A window titled "Welcome" with standard window controls (minimize, maximize, close). The content area is white and contains two centered options. The first option is labeled "New user?" in a small, light gray font, followed by a gray rectangular button with the text "REGISTER" in black. The second option is labeled "Already registered?" in a small, light gray font, followed by a gray rectangular button with the text "SIGN IN" in black.

Registration screen



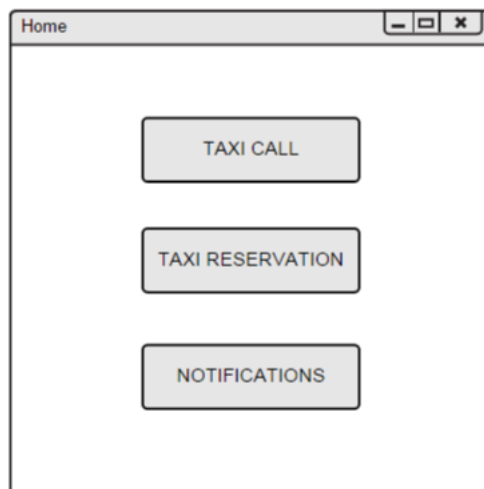
A window titled "Registration" with standard window controls (minimize, maximize, close). The content area is white and contains a registration form. The form has four labeled input fields: "First name", "Last name", "Phone number", and "Password", each followed by a white rectangular input box. Below these is a section labeled "Privacy stuff" with a light gray border and a white input box. Underneath the privacy section is a checkbox with the text "I agree". At the bottom center of the window is a gray rectangular button with the text "SUBMIT" in black.

Login screen



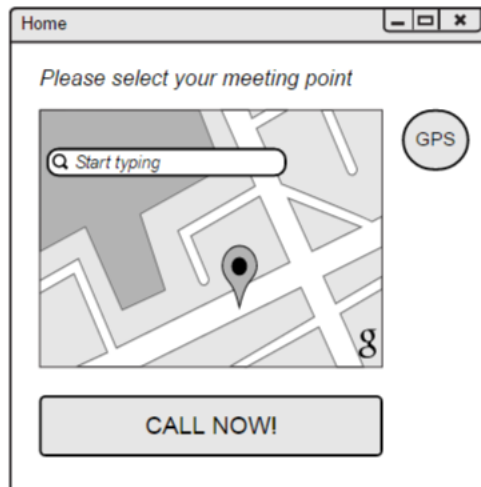
A window titled "Login" with a standard OS title bar (minimize, maximize, close buttons). The window contains two text input fields, one labeled "Phone number" and one labeled "Password", stacked vertically. Below the fields is a single "SUBMIT" button.

Main screen

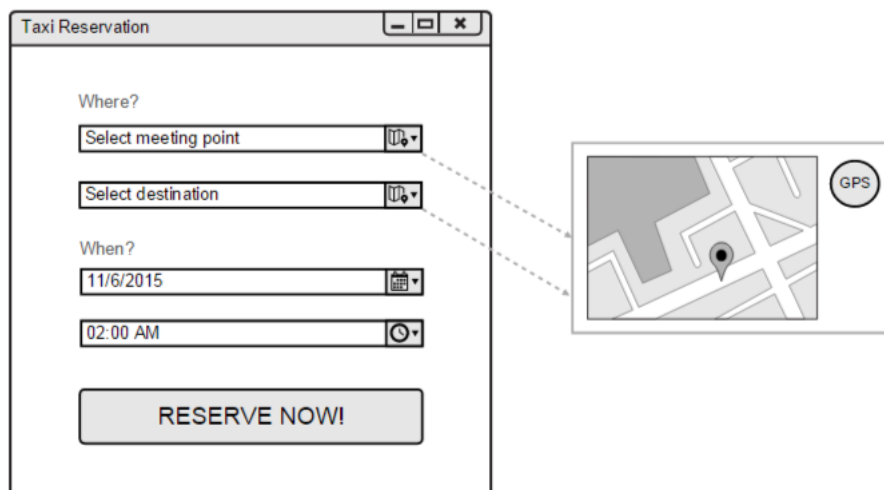


A window titled "Home" with a standard OS title bar (minimize, maximize, close buttons). The window contains three buttons stacked vertically: "TAXI CALL", "TAXI RESERVATION", and "NOTIFICATIONS".

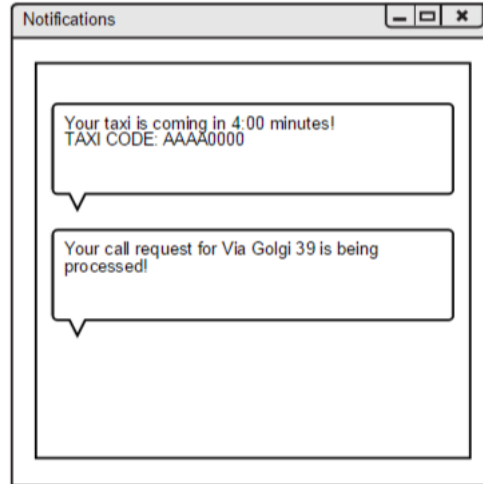
Call Taxi screen



Reservation screen



Notification screen



3.1.2 Driver interface

The driver interface is available only on the mobile platform. The first screen showed to a non-logged-in taxi driver allows her only to sign in. To do so, she must enter the password provided by the government and the ID of the taxi she is currently driving. Identification is based on mobile phone number, automatically retrieved by the application.

The screen showed to a logged-in driver allows her to change state and shows the queue zone and position (if free). When a request is received, the request screen automatically pops up and allows her to accept or refuse (while showing meeting point and passenger name). If the request is accepted, the main screen is substituted with a screen containing all the information about the request (meeting point, passenger's name and phone number). When the run is completed, the driver can close this screen by clicking "Run completed".

Driver login screen



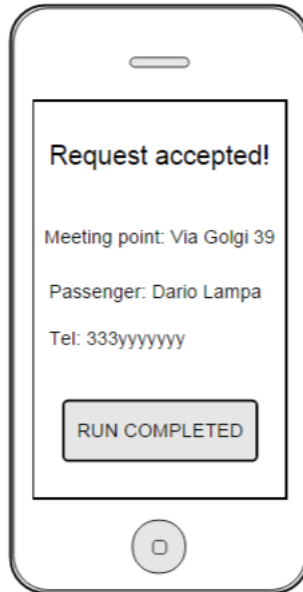
A mobile app mockup for a driver login screen. It features a white background with a rounded rectangle containing the following elements: the text "Welcome!" at the top; a text input field with the placeholder "333xxxxxxx"; a text input field with the placeholder "Password"; a text input field with the placeholder "Taxi ID"; and a grey button labeled "SIGN IN" at the bottom. The entire screen is framed by a grey border representing the phone's bezel, with a small grey circle at the bottom center representing the home button.

Driver login screen

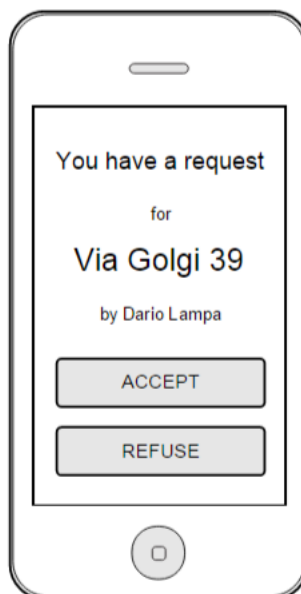


A mobile app mockup for a driver login screen. It features a white background with a rounded rectangle containing the following elements: the text "You are" at the top; the text "3rd" in a large font; the text "in queue for zone" in a smaller font; the text "Lambrate" in a large font; and a large grey circle at the bottom with the text "I'M BUSY!" inside. The entire screen is framed by a grey border representing the phone's bezel, with a small grey circle at the bottom center representing the home button.

Request accepted screen



Request screen



3.2 Functional requirements

3.2.1 [G1] Every time a passenger calls a taxi, she gets one

Required domain assumptions: D1,D2,D3,D6,D8.

Functional requirements:

- **[R1]** Every taxi call to the central system is immediately forwarded to the queue associated to the zone containing the required meeting point.
- **[R2]** For each reservation, a request is sent to the queue associated to the zone containing the required meeting point ten minutes before the meeting time.
- **[R3]** A request from the central system to taxi drivers always includes the meeting point, the passenger's name and her phone number.

3.2.2 [G2] Passengers know the maximum waiting time for the taxi they have been assigned to

Required domain assumptions: D2,D3,D9.

Functional requirements:

- **[R4]** For each taxi allocation, the central system calculates the maximum waiting time based on the taxi GPS location and the meeting point.
- **[R5]** Each time a taxi is allocated, the passenger is notified with the maximum waiting time.
- **[R15]** A request is automatically refused if the taxi driver doesn't accept within 30 seconds.

3.2.3 [G3] Passengers are able to identify the taxi they have been assigned to

Required domain assumptions: D4.

Functional requirements:

- **[R6]** Each time a taxi is allocated, the passenger is notified with the incoming taxi ID.

3.2.4 [G4] Taxi drivers are able to recognize the passenger they have been assigned to

Required domain assumptions: D5.

Functional requirements:

- **[R3]** A request from the central system to taxi drivers always includes the meeting point, the passenger's name and her phone number.

3.2.5 [G5] Taxi drivers are forwarded only calls in which the meeting point is in their current zone

Required domain assumptions: D7,D8.

Functional requirements:

- **[R1]** Every taxi call to the central system is immediately forwarded to the queue associated to the zone containing the required meeting point.

3.2.6 [G6] The taxi driver who has been waiting for the longest time in a certain zone is the first to receive a request from that zone

Required domain assumptions: D7,D8.

Functional requirements:

- **[R7]** When a taxi driver sets her state to free, her taxi ID is inserted in the last position of the queue corresponding to her current location.
- **[R8]** When a request arrives at a queue, it is forwarded in order to the drivers of the taxis in the queue, beginning from the first position, until someone accepts.
- **[R9]** When a taxi driver accepts a request, her state is set to busy.
- **[R10]** When the state of a taxi driver turns to busy, her taxi is deleted from the queue corresponding to her current location.
- **[R11]** When a taxi driver refuses a request, her taxi is moved in the last position of the queue corresponding to her current location.

3.2.7 [G7] Taxi drivers receive requests only when they are free

Required domain assumptions: \emptyset .

Functional requirements:

- **[R1]** Every taxi call to the central system is immediately forwarded to the queue associated to the zone containing the required meeting point.
- **[R10]** When the state of a taxi driver turns to busy, her taxi is deleted from the queue corresponding to her current location.
- **[R12]** When a taxi driver logs out from the central system, she is set to busy.

- **[R16]** A taxi driver, whose application has not responded to the system solicitations for more than two minutes, is automatically logged out from the central system.

3.2.8 [G8] Passengers can reserve a taxi only for a time at least two hours from the reservation

Required domain assumptions: \emptyset .

Functional requirements:

- **[R13]** Each reservation request is accepted only if the submit time is at least two hours before the meeting time.

3.2.9 [G9] External developers have the possibility to use the taxi call and reservation function of MyTaxiService in their applications

Required domain assumptions: \emptyset .

Functional requirements:

- **[R14]** The system provides public APIs for the call and reservation functions.

3.2.10 [G10] Each reserved taxi arrives at the meeting point at most two minutes late

Required domain assumptions: D1,D2,D3,D6

Functional requirements:

- **[R2]** For each reservation, a request is sent to the queue associated to the zone containing the required meeting point ten minutes before the meeting time.
- **[R3]** A request from the central system to taxi drivers always includes the meeting point, the passenger's name and her phone number.

3.2.11 [G11] Taxi drivers receive only and all the requests relative to the taxi they are currently driving

Required domain assumptions: D10,D11

Functional requirements:

- **[R17]** Every time a driver logs in, the application stores an association between her and the taxi she has specified.

- **[R18]** Every time a driver logs out, the association with her taxi is deleted.
- **[R19]** The application rejects all driver login attempts in which the specified taxi is already associated to another driver.

3.3 Nonfunctional requirements

3.3.1 Performance

Taken into account that all exchanges between the central system and the terminals are very short, the following performance requirements must be satisfied:

- The time occurring between the reception of a user request and the forwarding to the queue (for a taxi call) or its memorization (for a reservation) must be less than one second for at least 90% of the requests;
- The system must be able to handle at least 1000 requests/responses at the same time.

3.3.2 Availability

The system must be available 99.99% of the time (downtime should be at most 10 seconds per day).

3.3.3 Security

The system requires all users to have a password of at least eight alphanumeric characters. Only the passwords' hashes are stored in the system. Communications are encrypted (e.g. using https) to guarantee the privacy of the users.

3.3.4 Portability

The central system must be implemented with a non-host-dependent language. The mobile applications must be available on the main mobile operating systems: Android, Windows Phone and iOS.

3.4 Scenarios

3.4.1 Taxi call

Mr. Goodman, a lawyer, is having his car washed at A1A Carwash, when he receives a call from a client who has just been arrested. Goodman must reach the central police station as soon as possible, but his car will not be available for at least 20 minutes and there are no bus stops nearby. He, therefore, decides to call a taxi.

Goodman has the MyTaxiService app already installed on his mobile and has already gone through registration, so he's automatically logged in as he opens the app. Taking his privacy in high regard, the lawyer has never given the app

the authorization to use his position (as retrieved by GPS), so he manually selects the meeting point.

Not far from there, in a parking lot, a taxi driver is waiting for calls. He has been waiting for quite a lot, so he is happy to hear the distinctive MyTaxiService notification coming from his mobile. After reading the place of the meeting and the name of the customer, he accepts the request and immediately starts driving. One minute after sending the request, Goodman receives the waiting time (5 minutes) and the code of the incoming taxi, so he approaches the roadside and waits impatiently.

After only three minutes, the lawyer spots a taxi and starts waiving. The taxi driver pulls over, rolls down the window and asks “Saul Goodman?”.

3.4.2 Taxi reservation

Mr. White has been invited to an important conference, in which, as a guest of honor, he will give a lecture about crystallography”. The conference will be held on 25th December at Politecnico di Milano.

The day before, Mr. White decides to reserve a taxi for the conference. He starts the web application on his computer, he logs in correctly (by entering his phone number and his password) and he clicks on “Reservation”. He enters the precise date and hour (25th December at 14.00), the meeting point (Piazza Duomo) and the destination (Piazza Piola). After submitting his data, he receives a confirmation message by the application. The next day at 13.50, Jesse, a taxi driver who is waiting in his cab near Piazza Duomo, receives the request on his phone and accepts it.

At the same time, Mr. White, who is wandering in Piazza Duomo, receives a notification on his phone, containing the code of the incoming taxi and the waiting time (5 minutes). After 4 minutes, the taxi arrives and brings him to the specified location.

3.4.3 Taxi driver routine

Achille, a taxi driver, has just completed his last run to Politecnico di Milano. After stopping a couple minutes for a coffee, he decides to go back to work, thus signaling that he is free to the central system. He starts his mobile application and clicks “I’m Free!”. The application confirms his request and shows him that he is already the first in the queue for that zone. After a few minutes, he receives a request from Piazza Piola through the application. He decides to accept it, thus he clicks on “Accept request”. He immediately goes to the meeting point, but no one shows up. After waiting for the passenger for about 5 minutes, he decides to set his state again to free.

3.4.4 External developer point of view

Bruce K. is a very lazy programmer with an ambitious idea. He wants to develop the ultimate mobile application for public transport in his city. The app,

called BringMeThere, would work as following: the user selects a destination and a search profile, such as “in a hurry” or “short of money”. By using GPS and all available information about the current state of means of transport, the app provides the best solution and a series of alternatives, with explanatory details, e.g. “Walking will take you just five minutes, but if you are tired you should wait for the tram coming in 4 minutes, that will take 2 minutes to reach your destination, for a total of 6 minutes”.

Taking a taxi is also an option, but actually calling one by phone, or even switching to another app, could be annoying for a lazy user (it would certainly be for Bruce), so the programmer decides to integrate the calling process in his app.

With very little effort, Bruce uses a MyTaxiService API to implement this functionality. A user of BringMeThere will just have to press a button to have a taxi sent to his current location.

Bruce is already figuring all the money he will get by filling his app with advertisements.

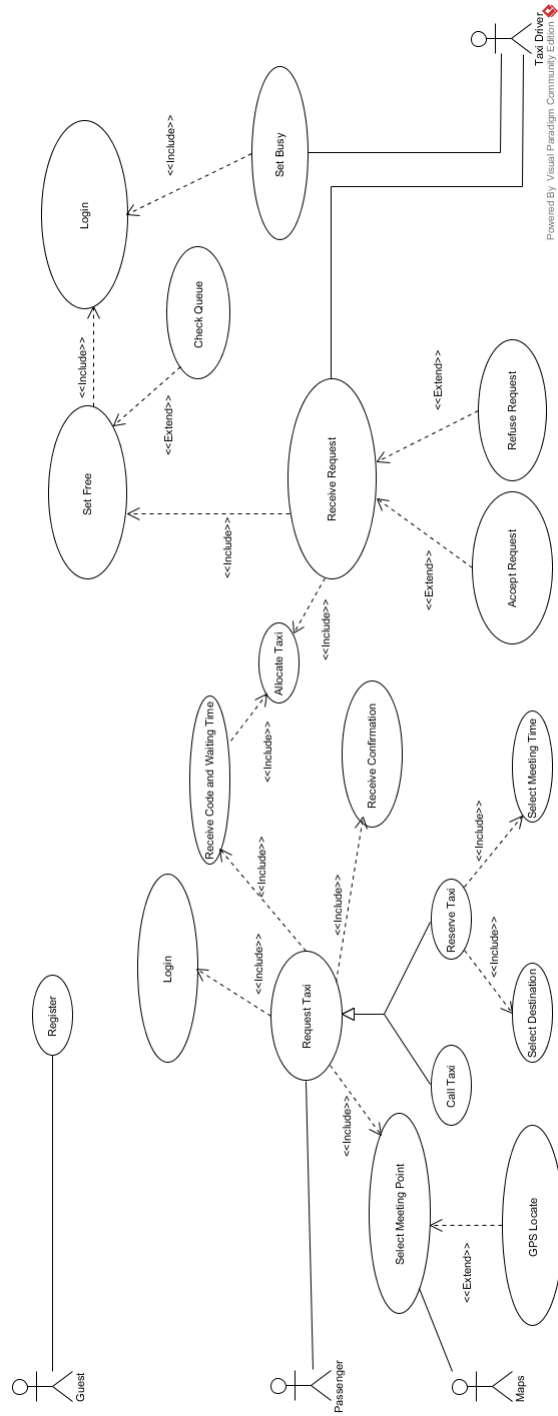
3.4.5 Registration

Dario Lampa is on a plane toward Milan. Knowing that he will probably need a taxi, he decides to search for some services on Google. Of course, the first result is MyTaxiService.

He connects to the homepage and, considering it is heavily user-friendly, he decides to give it a try, thus clicking "Register". He enters his cell phone number, a password, his name and surname, and submits the data. The application asks him to enter the validation code, which in turn has been sent to his phone. Unfortunately, his phone is currently in airplane-mode, and it will be so until the end of the flight. After a few minutes, the page shows an error message, saying that his request has been refused.

Though this little inconvenient really disappointed Dario, he decides to give MyTaxiService a second chance. After he has landed, he repeats the registration procedure. This time, the validation code is successfully received. After he enters it, the registration is finally complete. Lampa Dario is now a registered user of MyTaxiService!

3.5 Use case diagram



3.6 Use case 1

3.6.1 Name

Register

3.6.2 Actors

Guest

3.6.3 Entry condition

The guest clicks “Register” in the first screen of the passenger interface.

3.6.4 Flow of events

1. The guest enters the required data and clicks submit.
2. The application sends a verification code via SMS to the specified cell phone number and asks the user to enter that code.
3. The guest receive and enters the code.
4. The application store the registration data into the database.

3.6.5 Exit condition

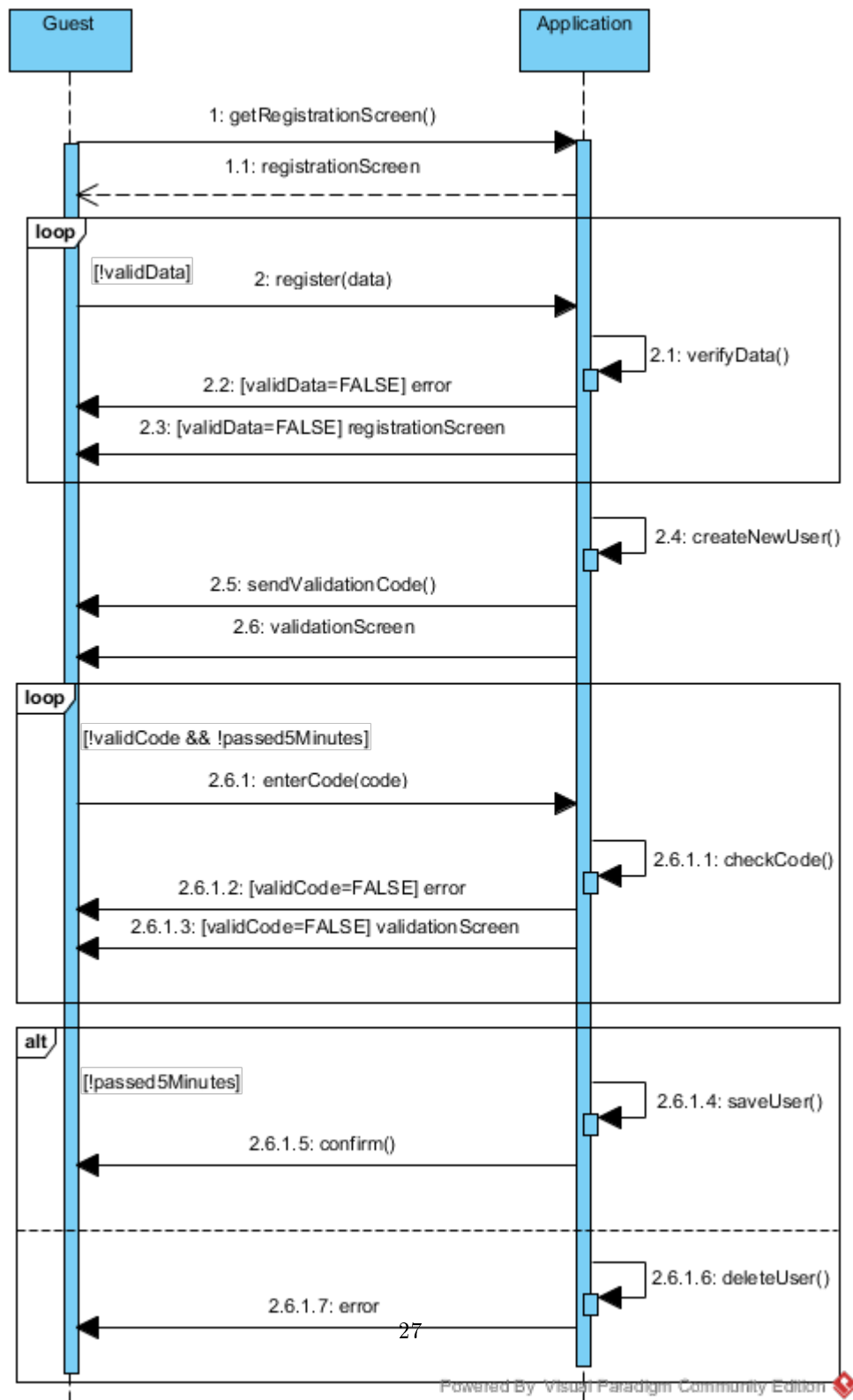
The registration is successfully stored in the database. From now on the guest is considered a user and she can login into the application.

3.6.6 Exceptions

1. Some registration data is wrong or missing: the application displays an error and goes back to data request.
2. The validation code is wrong or missing: the application displays an error and goes back to code request.
3. The validation code is not entered within the allowed time: the application closes the registration request and deletes all temporary data.

3.6.7 Special requirements

The validation code must be entered within 5 minutes after it has been sent.



3.7 Use case 2

3.7.1 Name

ReserveTaxi

3.7.2 Actors

Passenger

3.7.3 Entry condition

The passenger clicks “Reserve a taxi” in the main screen of the passenger interface. The passenger is already logged-in (otherwise he won’t be able to see the screen).

3.7.4 Flow of events

1. The passenger enters the required data in the reservation screen: she selects a meeting point and a destination, and she types a meeting time.
2. The passenger clicks “Reserve now”.
3. The system stores the reservation into the database and sends a confirmation message.

3.7.5 Exit condition

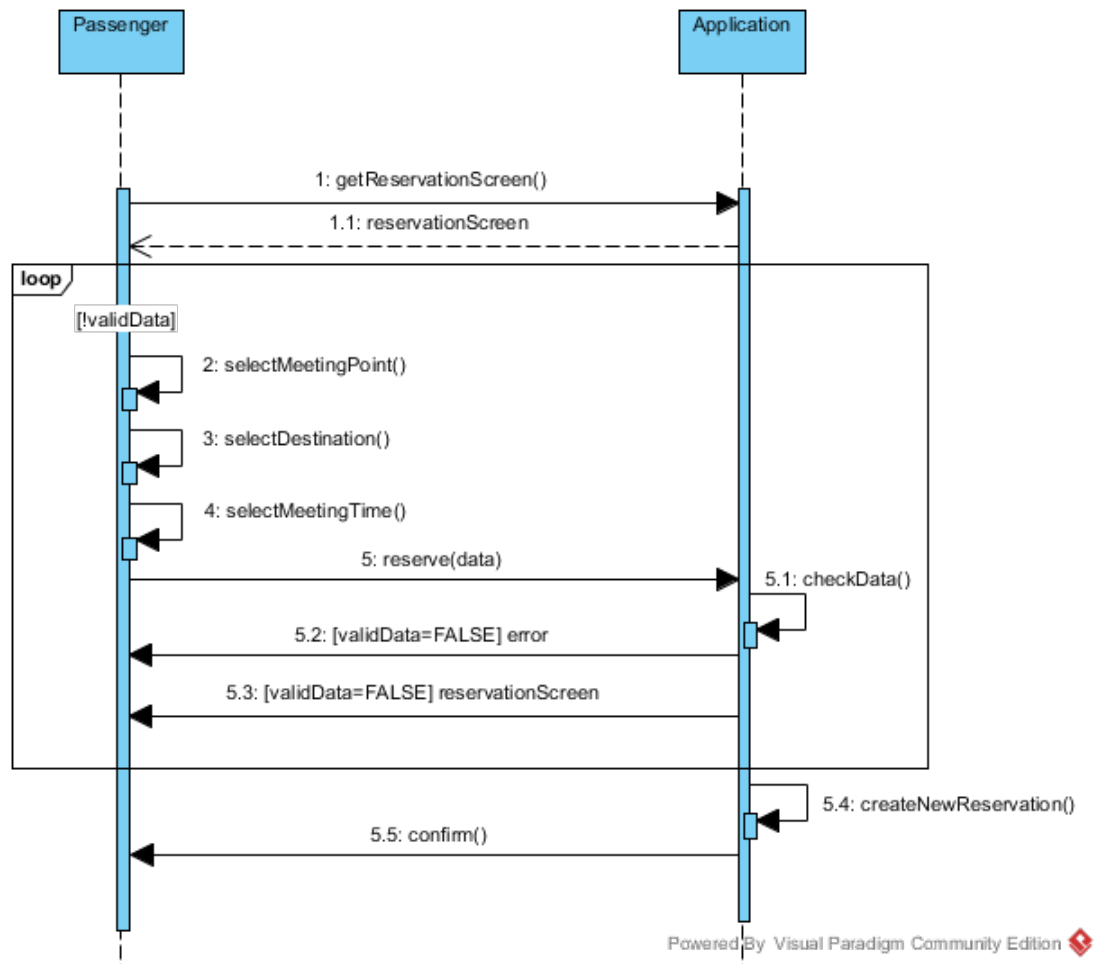
The reservation is successfully stored into the database and the passenger receives a confirmation.

3.7.6 Exceptions

1. The meeting time is less than two hours from the reservation time: the application displays an error and shows again the reservation screen.
2. Meeting point and destination are distant less than the minimum distance: the application displays an error and shows again the reservation screen.
3. The meeting point is not valid: the application displays an error and shows again the reservation screen.

3.7.7 Special requirements

The minimum distance between meeting point and the destination is 100 meters.



Powered By Visual Paradigm Community Edition

3.8 Use case 3

3.8.1 Name

LoginTaxi

3.8.2 Actors

Taxi driver

3.8.3 Entry condition

The taxi driver clicks “Sign in” in the first screen of the driver interface.

3.8.4 Flow of events

1. The taxi driver enters her password and her current taxi ID.
2. The taxi driver clicks “Sign in”.

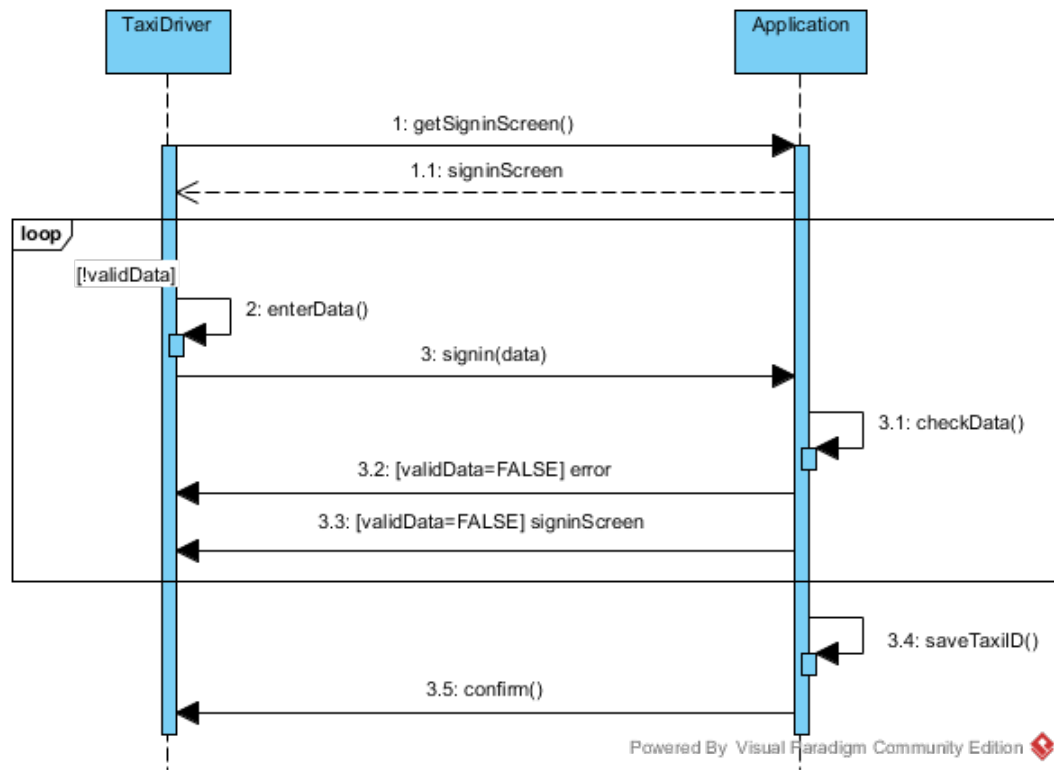
3.8.5 Exit condition

The taxi driver is successfully logged-in. An association between the driver and her taxi ID is stored in the database (meaning that that driver is currently driving that taxi).

3.8.6 Exceptions

1. The entered password is wrong (i.e., it is different from the one associated to the phone number of the driver): the application displays an error and shows again the login screen.
2. The entered taxi ID does not exist: the application displays an error and shows again the login screen.
3. The entered taxi ID corresponds to a taxi that is currently being driven by another driver: the application displays an error and shows again the login screen.

3.8.7 Special requirements



3.9 Use case 4

3.9.1 Name

ReceiveRequest

3.9.2 Actors

Taxi driver

3.9.3 Entry condition

The queue forwards a request to a taxi driver.

3.9.4 Flow of events

1. The taxi driver receives and checks the incoming request.
2. The taxi driver either accepts or refuses the request.

3.9.5 Exit condition

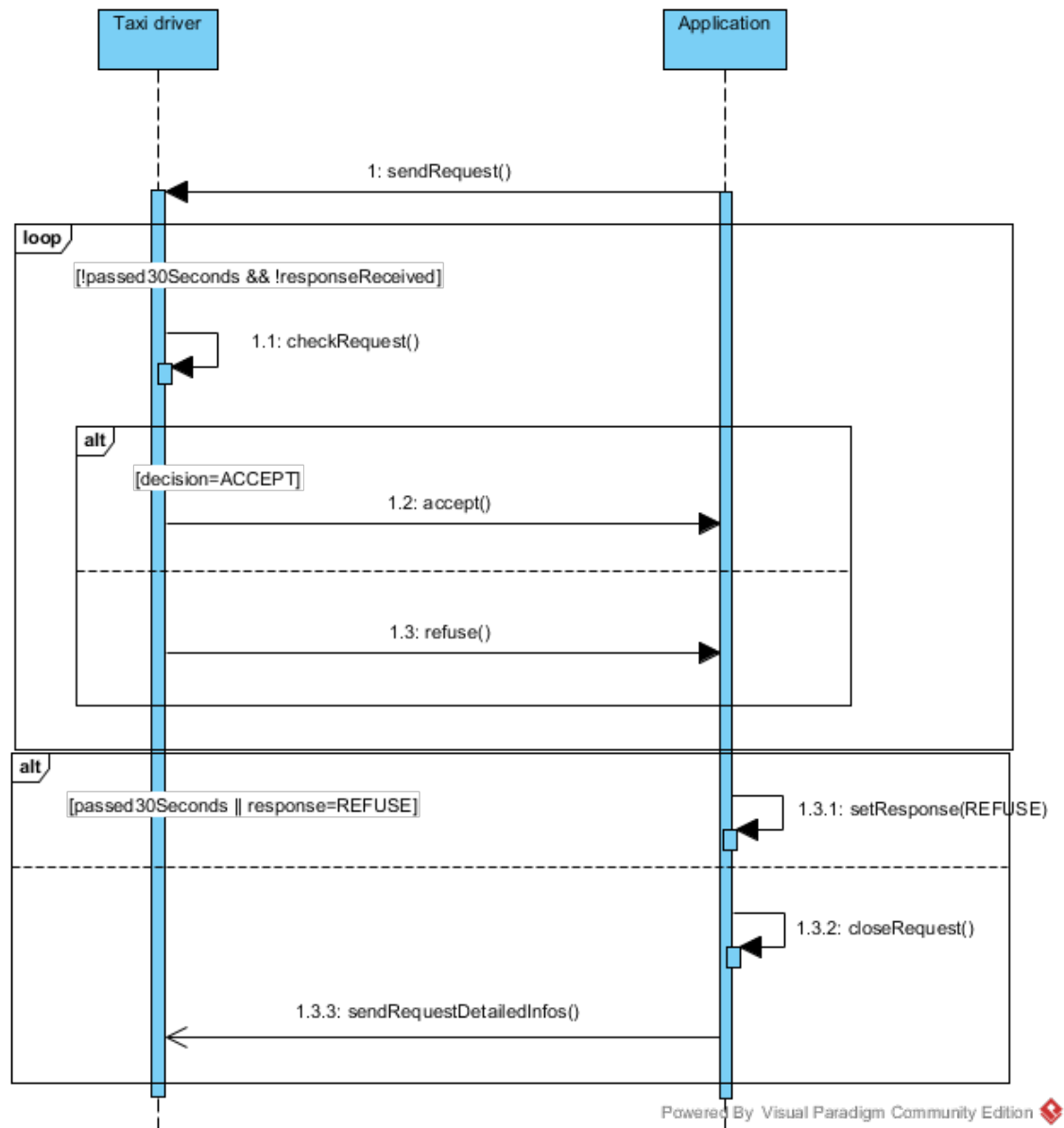
The queue receives the driver decision on the request.

3.9.6 Exceptions

1. A response from the driver does not arrive within the allowed time: the response is considered a refusal and the use case is terminated.

3.9.7 Special requirements

The allowed time for a driver to give a response on the request is one minute.



3.10 Use case 5

3.10.1 Name

AllocateTaxi

3.10.2 Actors

Taxi driver, passenger

3.10.3 Entry condition

Two possible entry conditions:

1. A taxi call has been received.
2. One of the stored reservations has meeting time ten minutes from now.

3.10.4 Flow of events

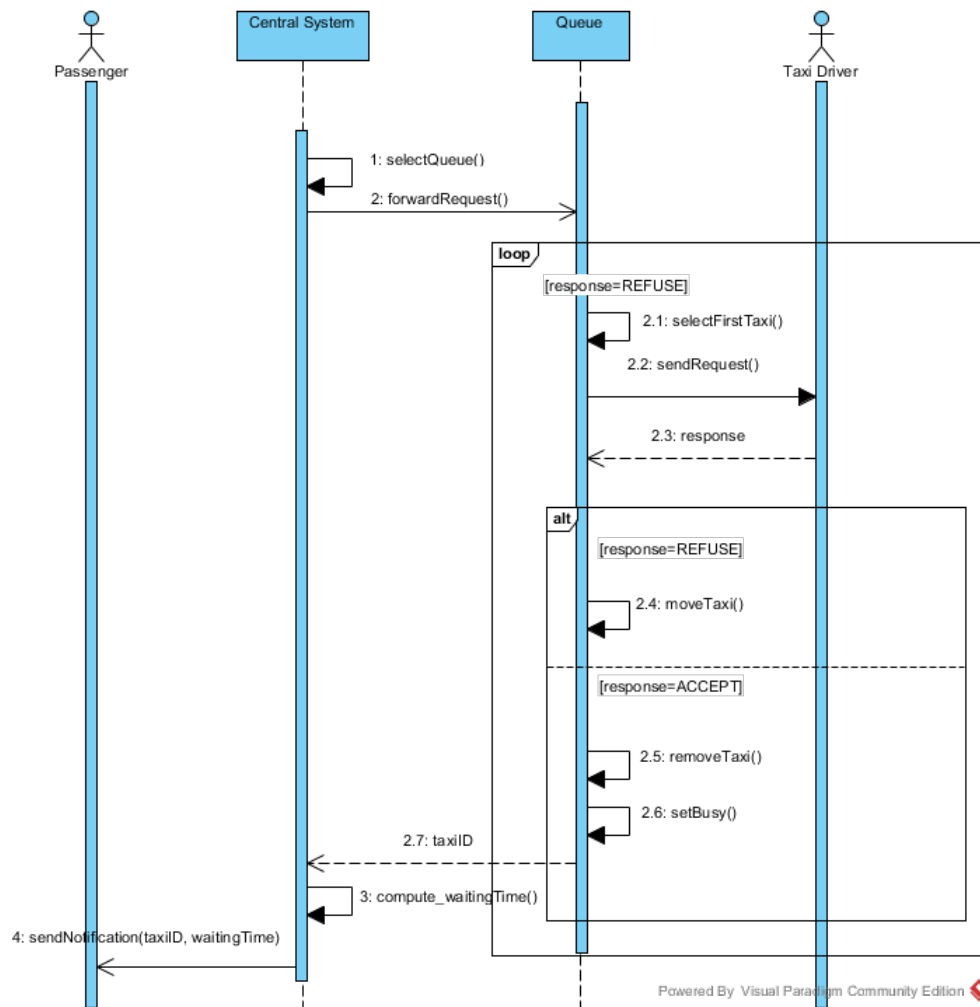
1. The system generates a request and sends it to the corresponding queue.
2. The queue forwards it to the first taxi in the queue.
3. The queue receives a response from the taxi driver.
4. If the response is an acceptance, the system computes the waiting time and sends a notification to the passenger (with waiting time and taxi ID). In this case, the taxi driver is set to busy and removed from the queue. Otherwise, the taxi is moved in the last position of the queue and point 2 is repeated.

3.10.5 Exit condition

The passenger receives a notification with waiting time and taxi ID.

3.10.6 Exceptions

3.10.7 Special requirements



Powered By Visual Paradigm Community Edition

3.11 Use case 6

3.11.1 Name

Login

3.11.2 Actors

Passenger

3.11.3 Entry condition

A non-logged passenger chooses “Sign In”.

3.11.4 Flow of events

1. The application shows the login screen.
2. The passenger correctly inserts credentials.

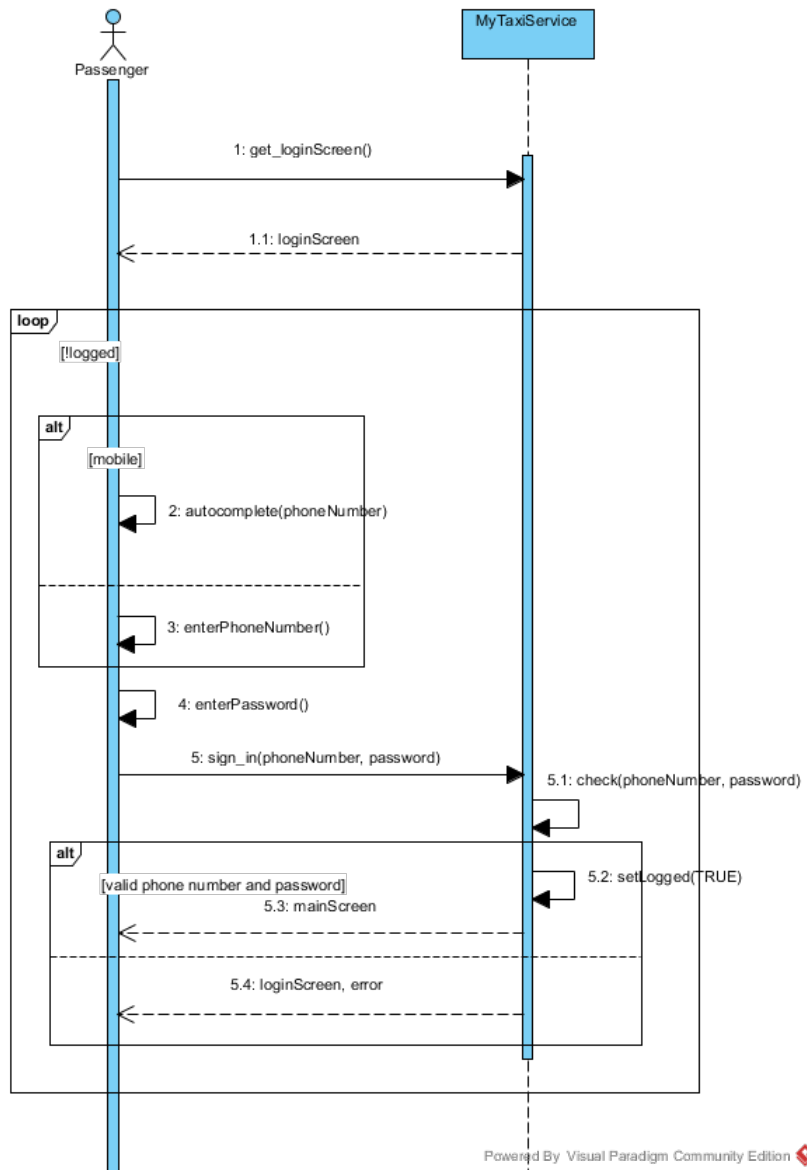
3.11.5 Exit condition

The application shows the main screen. The passenger is now logged.

3.11.6 Exceptions

If credentials are incorrect (inexistent phone number or wrong password), the application shows an error message on the login page.

3.11.7 Special requirements



Powered By Visual Paradigm Community Edition

3.12 Use case 7

3.12.1 Name

Call Taxi

3.12.2 Actors

Passenger

3.12.3 Entry condition

Logged passenger chooses “Taxi Call” on main screen.

3.12.4 Flow of events

1. The application shows the taxi call screen.
2. The passenger selects the meeting point.
3. The passenger presses “Call Now”.

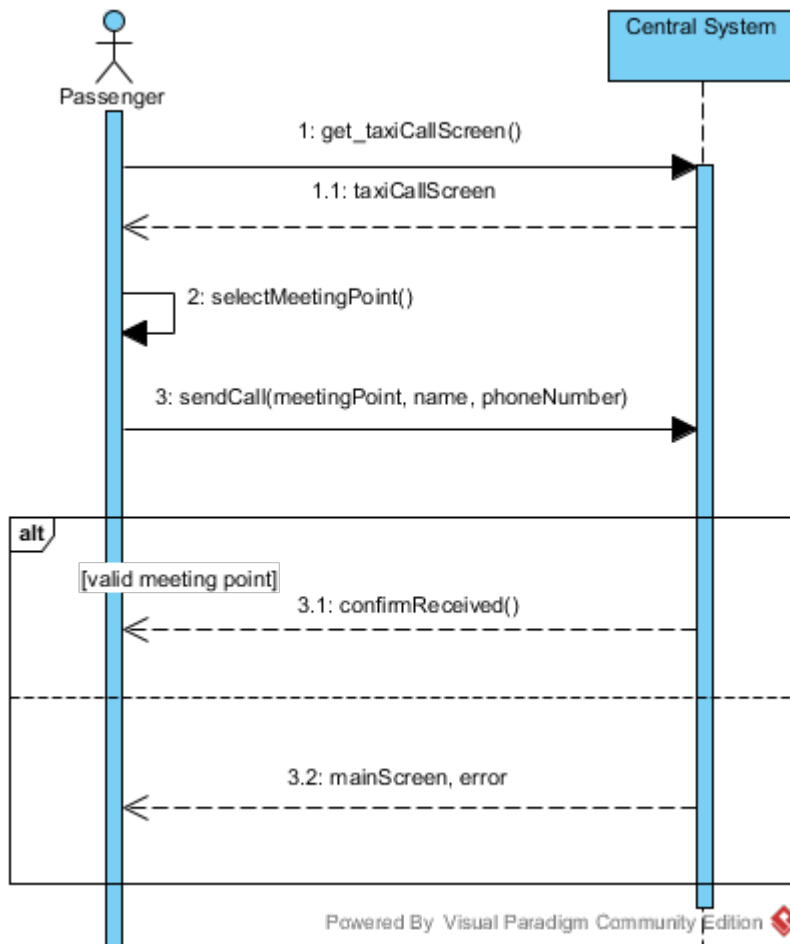
3.12.5 Exit condition

The passenger receives a confirmation, saying that her request is being processed.

3.12.6 Exceptions

If the meeting point is not valid (there is no zone containing the point), an error is showed and the applications returns to the main screen.

3.12.7 Special requirements



3.13 Use case 8

3.13.1 Name

Select Meeting Point

3.13.2 Actors

Passenger

3.13.3 Entry condition

This is a sub-use-case of Call Taxi and Reservation.

3.13.4 Flow of events

1. The passenger selects the meeting point either automatically (through GPS, only for mobile application) or manually (through Google Maps).

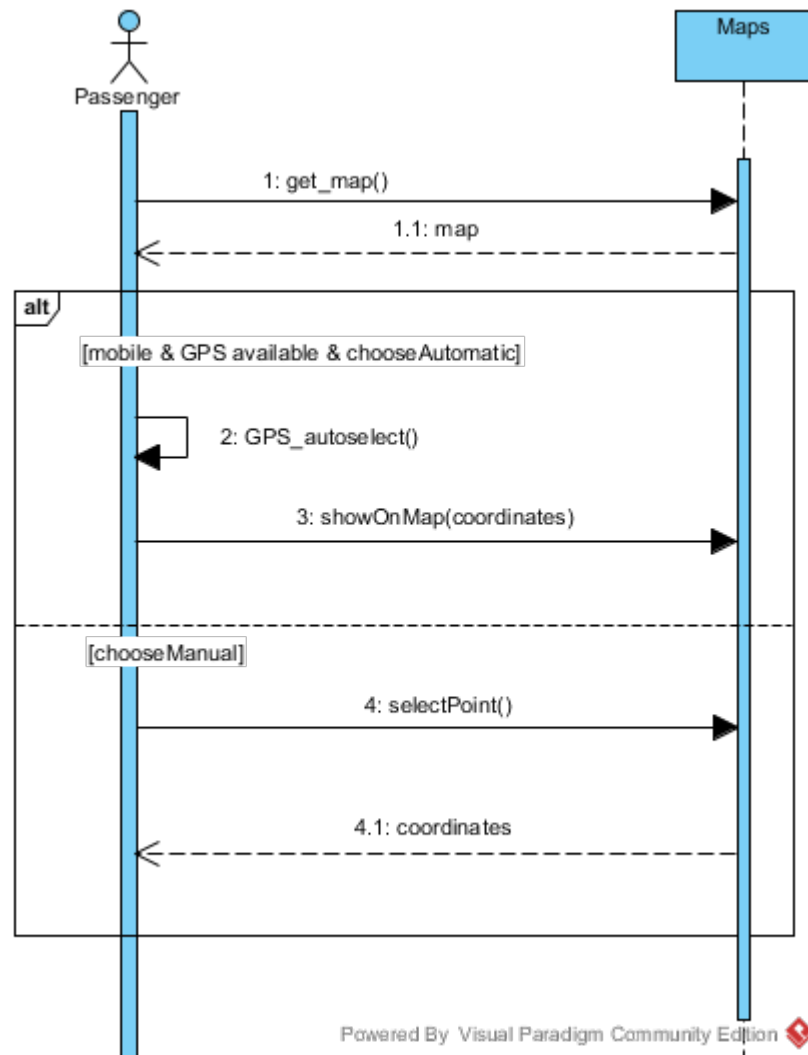
3.13.5 Exit condition

Meeting point is selected and ready to be communicated to the system.

3.13.6 Exceptions

If GPS is not available (not present, turned off, not authorized) the passenger can only select the meeting point manually.

3.13.7 Special requirements



3.14 Use case 9

3.14.1 Name

Set Free

3.14.2 Actors

Taxi Driver

3.14.3 Entry condition

Taxi driver is logged, busy and wants to switch to free.

3.14.4 Flow of events

1. The taxi driver press “I’m Free”.

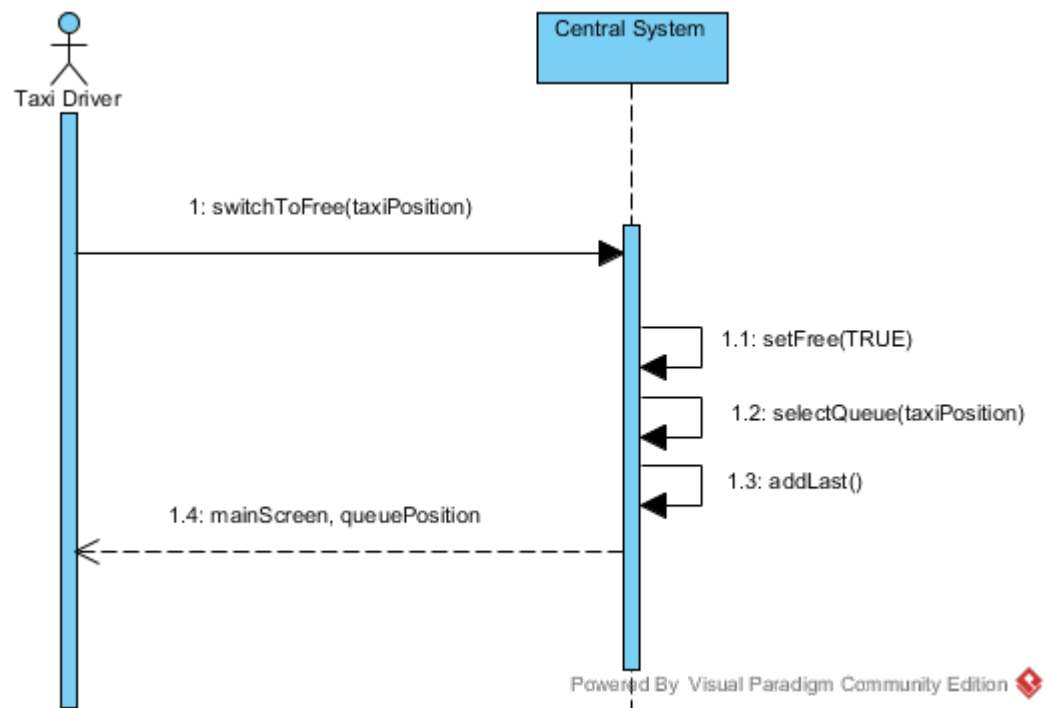
3.14.5 Exit condition

Taxi is added to the proper queue, “Free” is displayed.

3.14.6 Exceptions

Whenever the connection with the central system is lost, the Taxi Driver is considered busy by both the central system and the mobile application.

3.14.7 Special requirements



3.15 Use case 10

3.15.1 Name

Set Busy

3.15.2 Actors

Taxi Driver

3.15.3 Entry condition

Taxi Driver is logged, free and wants to switch to busy.

3.15.4 Flow of events

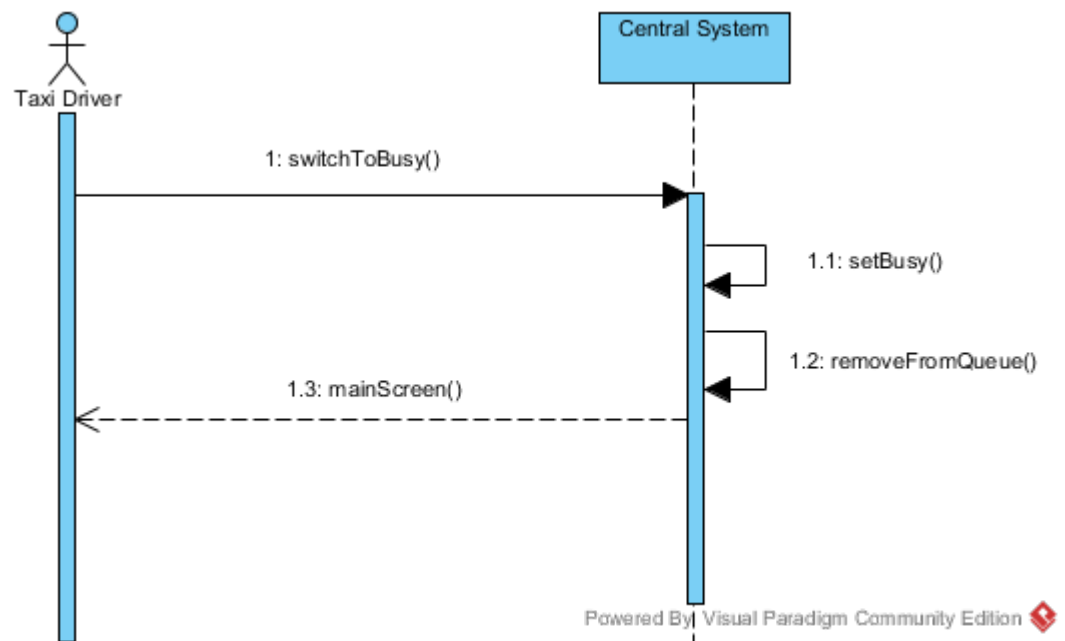
1. The taxi driver press “I’m Busy”.

3.15.5 Exit condition

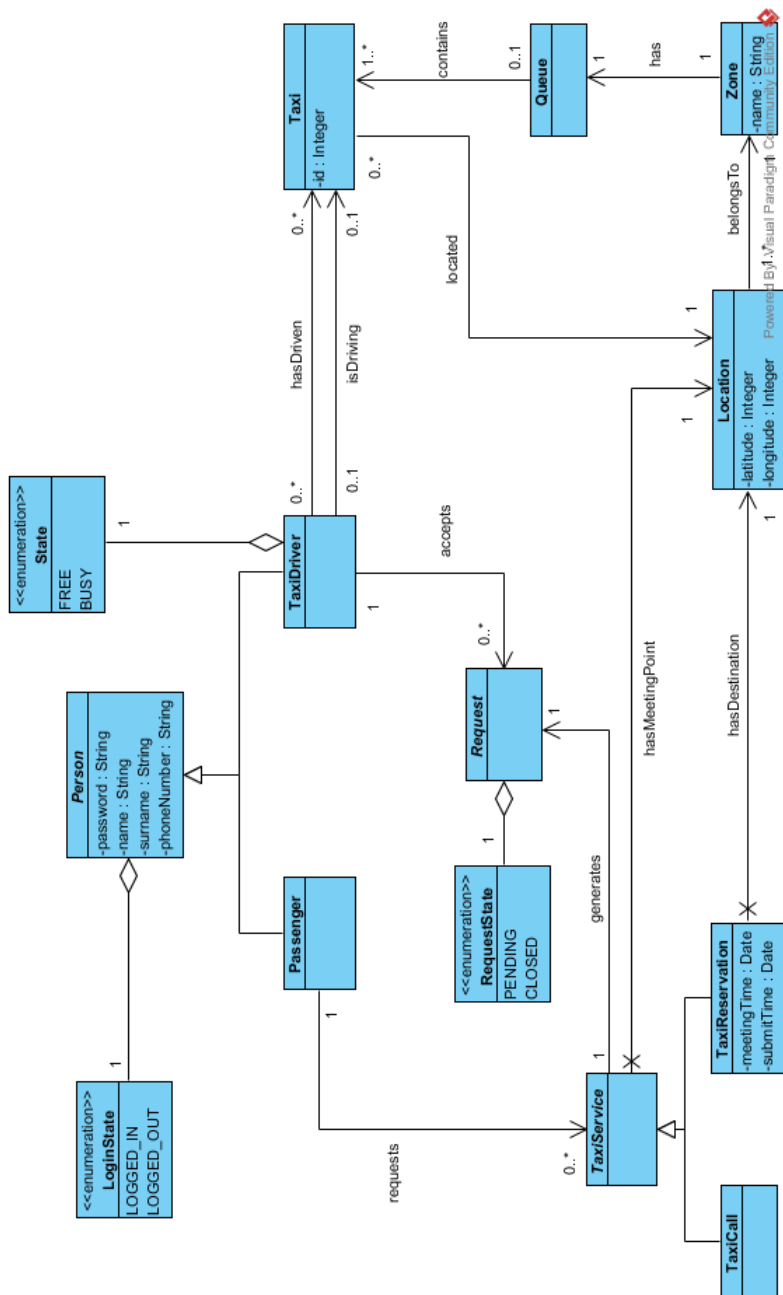
Taxi is removed from queue, “Busy” is displayed.

3.15.6 Exceptions

3.15.7 Special requirements



3.16 Class diagram



4 Appendix

4.1 Alloy

4.1.1 Signatures

```
module myTaxiService

//-----BASIC DATA TYPES-----
sig Date {

    isAtLeastTwoHoursBefore : set Date
}

fact antiReflexive {

    no x: Date | x in x.isAtLeastTwoHoursBefore
}

fact antiSymmetry {

    no x: Date, y:Date | y in x.isAtLeastTwoHoursBefore and x in
}

fact transitivity {

    isAtLeastTwoHoursBefore = ^isAtLeastTwoHoursBefore
}
//-----
```



```

//-----CLASS DEFINITION-----|-----
abstract sig Person {
    hasLoginState : one LoginState
}

sig Passenger extends Person {
    requests : set TaxiService
}

sig TaxiDriver extends Person {
    hasState : one State,
    isDriving : lone Taxi ,
    accepts : set Request
}

sig Location {
    belongsTo : one Zone
}

sig Zone {
    hasQueue : one Queue
}

sig Queue {
    contains : some Taxi
}

abstract sig TaxiService {
    hasMeetingPoint : one Location,
    generates : one Request
}

sig TaxiCall extends TaxiService {}

sig TaxiReservation extends TaxiService {
    hasDestination : one Location,
    hasMeetingTime : one Date,
    hasSubmitTime : one Date
}

sig Request {
    hasRequestState : one RequestState
}

//-----

```


4.1.2 Cardinality Facts

```
//-----CARDINALITY FACTS-----  
  
//No more than one driver for each taxi  
fact singleDriver {  
    isDriving in TaxiDriver lone -> Taxi  
}  
  
//Each zone contains at least one location  
fact zoneHasLocation {  
    belongsTo in Location some -> Zone  
}  
  
//Each queue has exactly one zone  
fact oneZone {  
    hasQueue in Zone one -> Queue  
}  
  
//Each taxi is contained in at most one queue  
fact taxiQueue {  
    contains in Queue lone -> Taxi  
}  
  
//Each request is accepted by exactly one taxi driver  
fact requestAccepted {  
    accepts in TaxiDriver lone -> Request  
}  
  
//Each TaxiService is requested by exactly one passenger  
fact singleAuthor {  
    requests in Passenger one -> TaxiService  
}  
  
//Each request is generated by exactly one TaxiService  
fact singleRequest {  
    generates in TaxiService one -> Request  
}  
  
//-----
```

4.1.3 Facts

```
//-----FACTS-----

//A taxi is in a queue if, an only if, it is being driven by a taxi driver, who is free.
fact taxiInAQueue {

    all t:Taxi | all q:Queue | t in q.contains implies (some d:TaxiDriver | t in d.isDriving and d.hasState in FREE)
    all t:Taxi | (some d:TaxiDriver | t in d.isDriving and d.hasState in FREE) implies (some q:Queue | t in q.contains)
    all t:Taxi | all q:Queue | t in q.contains implies (q in t.isLocated.belongsTo.hasQueue)
}

//If a driver is logged out her state is BUSY and no taxi is associated
fact driverLogOut {

    all d:TaxiDriver | d.hasLoginState in LOGGED_OUT implies (d.hasState in BUSY and d.isDriving = none)
}

//If a driver is logged in, she has a taxi associated
fact driverLogIn {

    all d:TaxiDriver | d.hasLoginState in LOGGED_IN implies (some d.isDriving)
}

//The submit time of a reservation is always at least 2 hours before the meeting time
fact reservationTime {

    all r:TaxiReservation | r.hasMeetingTime in r.hasSubmitTime.isAtLeastTwoHoursBefore
}

//A request is closed if, and only if, it has been accepted by a taxi driver
fact acceptedRequest {

    all r:Request | r.hasRequestState in CLOSED iff (some d:TaxiDriver | r in d.accepts)
}

//Meeting point and destination of a reservation never coincide
fact {

    all r:TaxiReservation | r.hasMeetingPoint not in r.hasDestination and r.hasDestination not in r.hasMeetingPoint
}
//-----
```


4.1.4 Cardinality Assertions

```
//-----CARDINALITY ASSERTIONS-----
assert singleDriver {
    no disj d1,d2: TaxiDriver | some t: Taxi | t in d1.isDriving and t in d2.isDriving
}

check singleDriver

assert zoneHasLocation {
    no z: Zone | #z.(~belongsTo)=0
}

check zoneHasLocation

assert oneZone {
    no q: Queue | some disj z1,z2: Zone | q in z1.hasQueue and q in z2.hasQueue
    no q:Queue | no z:Zone | q in z.hasQueue
}

check oneZone

assert taxiQueue {
    no disj q1,q2: Queue | some t: Taxi | t in q1.contains and t in q2.contains
}

check taxiQueue

assert requestAccepted {
    all r:Request | lone d:TaxiDriver | r in d.accepts
}

check requestAccepted

assert singleAuthor {
    no t: TaxiService | some disj p1,p2: Passenger | t in p1.requests and t in p2.requests
    no t:TaxiService | no p:Passenger | t in p.requests
}

check singleAuthor

assert singleRequest {
    no r: Request | some disj t1,t2: TaxiService | r in t1.generates and r in t2.generates
    no r:Request | no t:TaxiService | r in t.generates
}

check singleRequest

//-----
```

4.1.5 Assertions

```
//-----ASSERTIONS-----
assert taxiInAQueue {
    no t:Taxi | some q:Queue | t in q.contains and (no d:TaxiDriver | t in d.isDriving)
    no t:Taxi | some q:Queue | some d:TaxiDriver | t in q.contains and t in d.isDriving and d.hasState not in FREE
}

check taxiInAQueue

assert reservationTime {
    no r:TaxiReservation | r.hasMeetingTime not in r.hasSubmitTime.isAtLeastTwoHoursBefore
}

check reservationTime

assert acceptedRequest {
    no r: Request | r.hasRequestState in CLOSED && (no t: TaxiDriver | r in t.accepts)
    no r: Request | r.hasRequestState in PENDING && (some t: TaxiDriver | r in t.accepts)
}

check acceptedRequest
//-----
```

4.1.6 Assertion checks

13 commands were executed. The results are:

- #1: No counterexample found. singleDriver may be valid.
- #2: No counterexample found. zoneHasLocation may be valid.
- #3: No counterexample found. oneZone may be valid.
- #4: No counterexample found. taxiQueue may be valid.
- #5: No counterexample found. requestAccepted may be valid.
- #6: No counterexample found. singleAuthor may be valid.
- #7: No counterexample found. singleRequest may be valid.
- #8: No counterexample found. taxiInAQueue may be valid.
- #9: No counterexample found. reservationTime may be valid.
- #10: No counterexample found. acceptedRequest may be valid.
- #11: **Instance found.** callScenario is consistent.
- #12: **Instance found.** reservationScenario is consistent.
- #13: **Instance found.** show is consistent.

4.1.7 Run

```
pred show{}

pred interestingWorld {
  some TaxiService
  all l: Location | #l.(~isLocated)<=1
  no l:Location | some t:Taxi, s:TaxiService | l in t.isLocated && l in s.hasMeetingPoint
  no p: Person | p.hasLoginState in LOGGED_OUT
}

pred callScenario {
  interestingWorld
  no TaxiReservation
  one TaxiCall
}

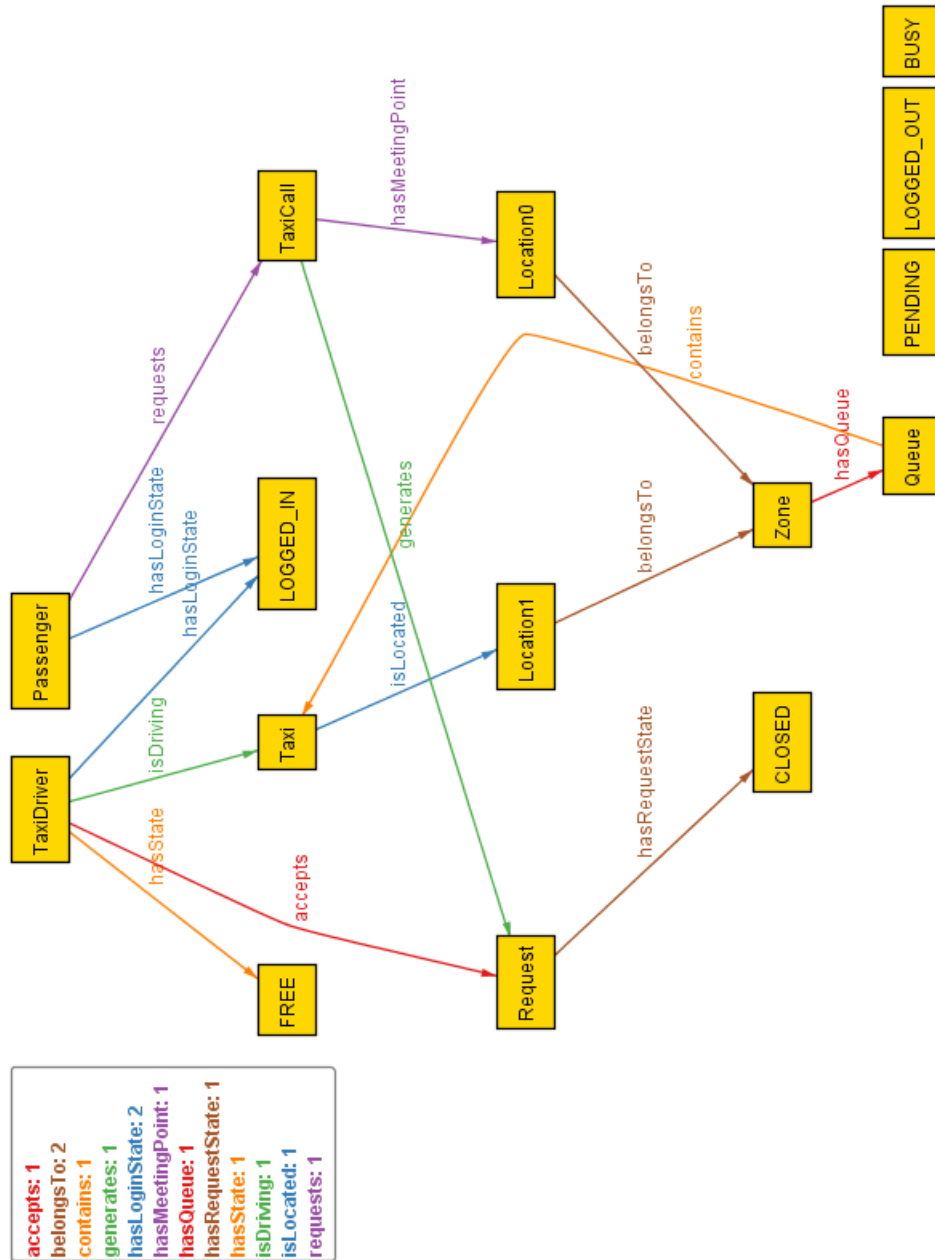
pred reservationScenario {
  interestingWorld
  one TaxiReservation
  no TaxiCall
}

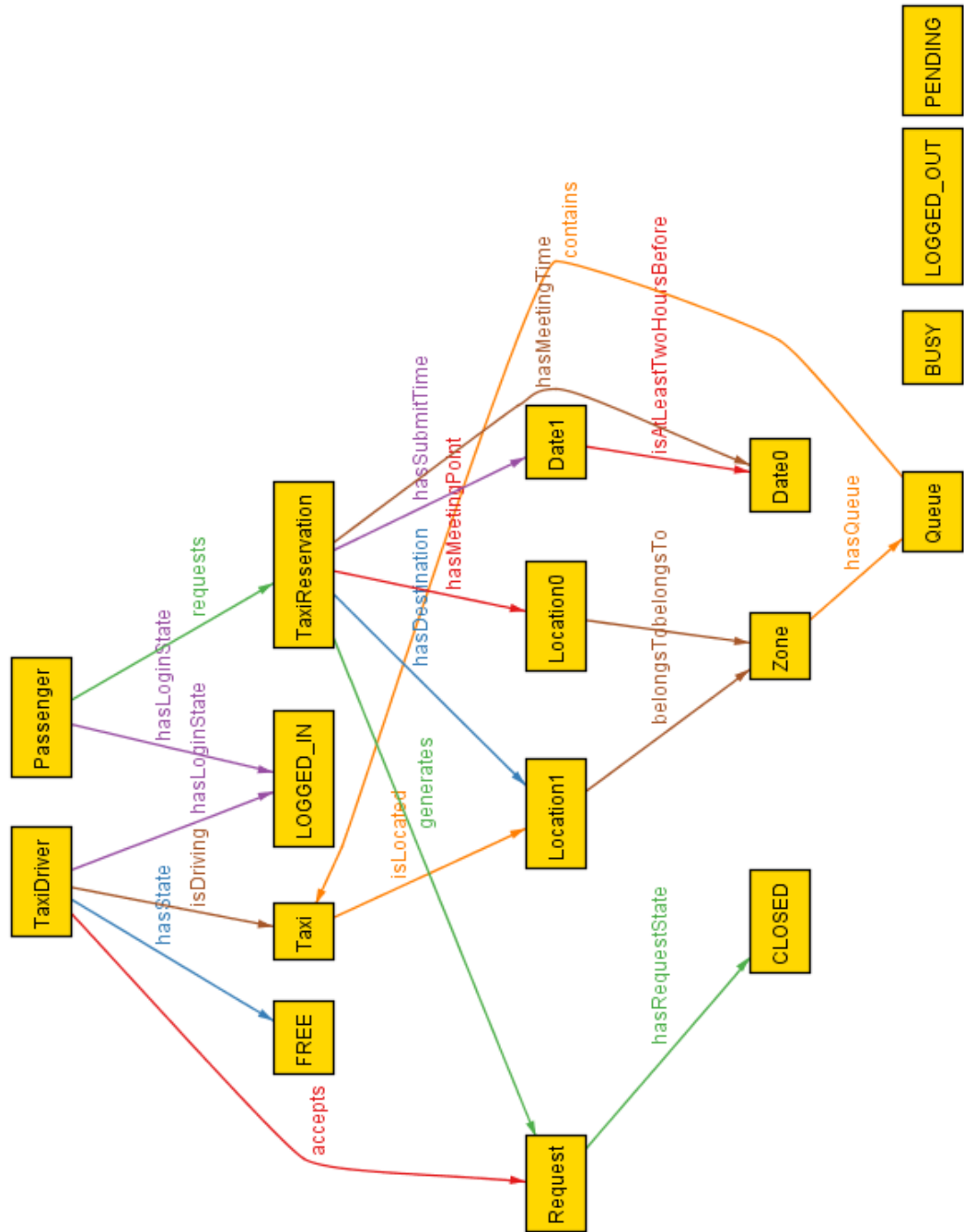
run callScenario for 2

run reservationScenario for 2

run show
```


4.1.8 Generated worlds







4.2 Used software

- **Microsoft Word and \LaTeX** : document redaction;
- **Visual Paradigm**: UML modeling;
- **Alloy**: system modeling and checking;
- **Moqups**: mockup creation;

4.3 Spent Hours

- Andrea Tirinzoni: $\sim 30\text{h}$
- Matteo Papini: $\sim 30\text{h}$