Politecnico di Milano

Software Engineering 2 Project
a.y. 2015-16

MyTaxiService
Requirements Analysis and Specifications Document
Version 1.0

Casati Fabrizio, 853195
Castelli Valerio, 853992

Referent professor: Di Nitto Elisabetta

November 6, 2015

Contents

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Stakeholders . . . 2
- 1.4 Definitions, acronyms, and abbreviations . . . 3
  - 1.4.1 Definitions . . . 3
  - 1.4.2 Acronyms . . . 4
  - 1.4.3 Abbreviations . . . 5
- 1.5 References . . . 5
- 1.6 Overview . . . 5

2 Overall description **6**
- 2.1 Product perspective . . . 6
  - 2.1.1 System interfaces . . . 6
  - 2.1.2 User interfaces . . . 6
  - 2.1.3 Software interfaces . . . 7
- 2.2 Product functions . . . 7
- 2.3 User characteristics . . . 8
  - 2.3.1 Actors identifying . . . 8
- 2.4 Constraints . . . 9
  - 2.4.1 Regulatory policies . . . 9
  - 2.4.2 Hardware limitations . . . 9
  - 2.4.3 Interfaces to other applications . . . 9
  - 2.4.4 Parallel operation . . . 9
  - 2.4.5 Criticality of the application . . . 9
  - 2.4.6 Safety and security considerations . . . 10
- 2.5 Domain properties and assumptions . . . 10

3 Specific requirements **13**
- 3.1 External Interface Requirements . . . 13
  - 3.1.1 User interface sketches . . . 13
- 3.2 Functional Requirements . . . 21
  - 3.2.1 Goal 1 . . . 21

3.2.2 Goal 2 . . . 21
3.2.3 Goal 3 . . . 22
3.2.4 Goal 4 . . . 22
3.2.5 Goal 5 . . . 23
3.2.6 Goal 6 . . . 24
3.2.7 Goal 7 . . . 24
3.2.8 Goal 8 . . . 25
3.2.9 Goal 9 . . . 25
3.2.10 Goal 10 . . . 26
3.2.11 Goal 11 . . . 27
3.2.12 Goal 12 . . . 28
3.3 Performance Requirements . . . 28
3.4 Design Constraints . . . 28
3.4.1 Standard compliance . . . 28
3.5 Software System Attributes . . . 29
3.5.1 Reliability . . . 29
3.5.2 Availability . . . 29
3.5.3 Security . . . 29
3.5.4 Portability . . . 30
3.6 Scenarios . . . 30
3.6.1 Scenario 1 . . . 30
3.6.2 Scenario 2 . . . 30
3.6.3 Scenario 3 . . . 31
3.6.4 Scenario 4 . . . 31
3.6.5 Scenario 5 . . . 32
3.6.6 Scenario 6 . . . 32
3.6.7 Scenario 7 . . . 33

Appendix A Use cases **34**
A.1 Request a taxi . . . 34
A.2 Reserve a taxi via the web application . . . 35
A.3 Allocate a reservation to a taxi . . . 36
A.4 End a ride . . . 37
A.5 Communicate taxi unavailability . . . 37
A.6 Enter inside city . . . 38

Appendix B Alloy **40**

Appendix C UML diagrams **55**
C.1 Use case diagram . . . 55
C.2 Class diagram . . . 57
C.3 Activity diagrams . . . 59
C.3.1 Request a taxi . . . 59
C.3.2 Place a taxi reservation . . . 60

C.3.3 Reservation allocation 61
C.3.4 Confirm a ride has ended 62
C.3.5 Set a drivers status to unavailable 63
C.4 Sequence diagram 64
C.5 State chart diagram 66

Appendix D Work hours **67**

Chapter 1

Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD). The main goal of this document is to completely describe functional and non-functional requirements of the system, clearly state the peculiar characteristics of the application domain, analyze the needs of the customer, specify the constraints and the limits of the software and identify how the system will typically be operated after its completion (use cases). The audience of this document comprises the members of the city council who will validate the system functionalities, the developers that will have to implement them and any other party potentially interested in understanding the precise specifications upon which the system will be developed. This document can also be used as a reference for further legal agreements between the city council and the development company.

1.2 Scope

The aim of the project is to create a new taxi management system to optimize an existing taxi service provided by the government of a city. The primary goal of this system, which from now on will be referred to as myTaxiService, is to provide passengers simplified access to taxi reservations and offer an appropriate management of taxi queues. In particular, a passenger may request a taxi either through a web application or a mobile application; in both cases, the passenger should receive a notification containing an identifier of the incoming taxi and an estimate of the time he has to wait for. Taxi drivers should be able to notify the system about their availability and to receive, accept and refuse incoming calls by means of a single mobile application. The city is divided into taxi zones that are uniquely associated with corresponding taxi queues. The system should be able to collect GPS location data from each taxi of the fleet and use this information to

automatically compute the distribution of taxis among the various zones. Every time a taxi driver notifies the system he's free to accept rides, the system should store the taxi identifier in the queue of taxis associated with the zone the taxi is in at that moment. Ride requests are catalogued by the zone they're coming from; in particular, each request should be forwarded to the first taxi queueing in the same zone. In case of acceptance, the system should send the appropriate confirmation notification to the passenger, containing both the identifier of the taxi and an estimate of the waiting time. Otherwise, the system should forward the request to the second taxi in the queue and should also move the first taxi in the last position in the queue, until a taxi positively answers to the call. In case no taxi can be found in the queue of the selected zone or every taxi drops the request, the system will not be able to provide the service. Passengers should be able to both ask for a taxi when they need it and reserve it in advance by specifying the meeting time, the origin and the destination of the ride. In this case, the reservation has to occur at least two hours before the ride. The system will confirm the reservation to the passenger and will proceed to allocate a taxi only 10 minutes before the specified meeting time. Finally, the system should provide programmatic interfaces to allow additional services to be developed on top of the basic one.

1.3 Stakeholders

We have identified a list of potential stakeholders for our system. Every stakeholder has a specific interest, either positive or negative, for the system.

- Passengers: all the people that want to require a taxi and that will use the system to accomplish this.
- Taxi drivers: all the people that drive a taxi. They are supposed to employ the system to answer passengers' requests and drive them to their desired location.
- City government (or city administration): it is the main stakeholder of the project, since it is the one who has commissioned it and that will pay for it.
- Taxi drivers' union: as the system will have an impact on the way taxi drivers work, it's necessary to consider that their labor union could have something to say on this matter.
- Mobile phone producers: they have an interest in providing the mobile terminals that the city government will provide to the taxi drivers in order to operate the service.

1.4 Definitions, acronyms, and abbreviations

1.4.1 Definitions

Here we present a list of significant, context-specific terms used in the document.

- System: the system that has to be designed.
- Passenger: a person that uses the system to request a ride.
- User: synonym for passenger, unless otherwise specified.
- Guest passenger: a person that is using the passenger application, but that has not performed a login.
- Logged in passenger: a person that is using the passenger application and that has performed a login.
- Guest taxi driver: a person that is using the taxi driver application, but that has not performed a login.
- Logged in taxi driver: a person that is using the taxi driver application and that has performed a login.
- Administrative personnel: the city administration staff that is responsible for the administrative operations related to the configuration of the system.
- City administration: the political organ that governs the city.
- City government: synonym for city administration.
- City council: synonym for city administration.
- Taxi code: the unique identifier associated to each taxi.
- Secret code: the random identifier associated to each request to certify that the passenger that is getting on a taxi is effectively the one that has requested it.
- Security code: synonym for secret code.
- Available taxi: a taxi is considered available if its driver can answer to a passenger's call.
- Unavailable taxi: a taxi is considered unavailable if it's not in service.
- Currently riding taxi: a taxi is considered to be currently riding if it's driving a passenger somewhere.

- Outside city taxi: a taxi which is outside the city boundaries.
- Taxi zone: a geographical area of the city which is logically associated to a single list of available taxis.
- Availability list: the list where the taxi manager stores references to the taxis which are available in the whole city.
- Unavailability list: the list where the taxi manager stores references to the taxis which are not in service.
- Currently riding list: the list where the taxi manager stores references to the taxis which are performing a ride.
- Out of city list: the list where the taxi manager stores references to the taxis which are outside the boundaries of the city.
- Zone queue: the FIFO list associated to a certain zone that contains the references to the available taxis which are considered for requests coming from locations inside that zone.
- Location data: the latitude and longitude information of a geographical location.
- Location reverse-geocoding: the operation that maps a couple of latitude, longitude coordinates into an address.
- Exceptional circumstances: extraordinary situations which are not considered to be part of the normal operating conditions. This includes, but is not limited to, strikes, severe weather conditions, service suspension decided by the city council and specific holidays.

1.4.2 Acronyms

- RASD: Requirements Analysis and Specification Document.
- SRS: Software Requirements Specification. Synonym of RASD.
- DB: Database.
- DBMS: Database Management System.
- API: Application Programming Interface. Synonym for "programming interface".
- ETA: Estimated Time of Arrival.
- GPS: Global Positioning System.
- RAM: Random Access Memory.

- DDoS: Distributed Denial of Service.
- FIFO: First In, First Out.

1.4.3 Abbreviations

No abbreviations other than the acronyms are used in this document.

1.5 References

- Assignment document: Assignments 1 and 2 (RASD and DD).pdf
- IEEE Recommended Practice for Software Requirements Specifications (Std 830-1998)

1.6 Overview

In the following part of this SRS document, we provide a general description of the system, with a particular focus on the assumptions it is based on and the required functionalities (Section 2).

Next, we go deeper in the specification of system functionalities, by providing a formal definition of all system requirements and an UML model (Section 3).

In the final appendix section, we will provide the skeleton of a possible Alloy formalization of the system (Appendix A).

Chapter 2

Overall description

2.1 Product perspective

The system that will eventually be released is composed of four major components:

- A mobile application for passengers;
- A web application for passengers;
- A mobile application for taxi drivers;
- The backend infrastructure to manage the service.

In particular, we expect the new system to fully replace any mechanism that the city administration had previously in place for the management of taxi calls.

2.1.1 System interfaces

The system has to be designed to be expandable, in order to let other developers implement new functionalities in the future should they be needed. This implies that the system will have to provide a proper set of APIs to expose its core functionalities to external plugins.

2.1.2 User interfaces

The mobile application should natively support both smartphones and tablets by offering a uniquely optimized user interface tailored to the specific kind of device being used.

Furthermore, the mobile application should be resolution independent; in particular, it should support all display sizes comprised between 3" and 6" (when used on a mobile phone) and between 7" and 12" (when used on a tablet).

The web application should properly address dynamic web page resizing by proportionally scaling header, footer and side bars to the amount of available space without stretching individual components and forms. It should also be offered both in a desktop version and in a mobile-friendly version.

In addition, both the mobile application and the web application should be designed to be accessible by blind people, that means that it should be possible for a screen reader software to correctly interpret them in a meaningful way.

2.1.3 Software interfaces

The system will have to communicate with an external mapping service in order to use the functionalities of GPS location reverse-geocoding and waiting time estimation.

The system will also interface with a commercial DBMS in order to store and retrieve data and perform queries over it.

Finally, in order to offer the web application functionalities, the system will have to interface with an external web server.

2.2 Product functions

In order to be considered functional, the system has to provide a set of key functionalities. These functionalities are either related to the administrative side of the taxi service or to the end users (which include taxi drivers and passengers).

In particular, the system is expected to be able to:

- Let the city administration insert and update the information about the taxis operating in the city.
- Let the city administration define the division of the city in taxi zones, specify their boundaries and update them if needed.
- Let passengers request a taxi.
- Let passengers register to the service.
- Let logged in passengers reserve a taxi for a specific date and time.
- Keep track of the zone in which every taxi is waiting.
- Keep track of the availability of each taxi.
- Allocate requests coming from a specific zone to taxis waiting in that zone following a FIFO policy.

2.3 User characteristics

In general, we do not expect the users of our system to be particularly tech-savvy. The main audience for which our system is intended comprises, in fact, customers coming from all levels of education.

Taxi drivers are very expert of the dynamics of a traditional, non-technological taxi service and they expect the new system to fully comply with those dynamics without having to change their habits. They know, however, how to use a smartphone with reasonable proficiency.

The city administration personnel has no specific technological background, apart from a generic introduction to the administrative tools they have to use.

Passengers are considered to know enough technology to be able to use a smartphone and a web browser, but no further assumption on their level of expertise is made.

2.3.1 Actors identifying

- Guest passenger: a passenger that has not logged in to the service yet. He can only decide to request a taxi for his current location, or to register in order to fully use the platform.
- Logged in passenger: a passenger that has registered to the service and has successfully logged in. Apart from the possibility to request a taxi, he can also reserve a taxi for a specific date and time and manage his personal information.
- Guest taxi driver: a taxi driver that has not yet logged in. This user can only perform the login operation, which is required in order to access the full set of functionalities reserved to taxi drivers.
- Logged in taxi driver: a taxi driver that has logged in. This user can mark himself as available or unavailable to receive requests; if available, he can receive, accept and refuse requests. He can also see what his current zone is and his position in the corresponding taxi queue. Finally, he can review all his served requests.
- Administrative personnel: the staff that configures the system with the taxi driver data and the zone division.
- Mapping service: the linked system from which our platform will retrieve information about the estimated time of arrival of taxis from their location to the required pickup location.

2.4 Constraints

There are a few constraints on the way the system has to be designed. In the following paragraphs, we will analyze them with further detail.

2.4.1 Regulatory policies

The system will be subject to the regulatory policies that discipline the operations of a taxi service. These policies come from the city government, the state law and the taxi drivers' union.

2.4.2 Hardware limitations

The system should support smartphones and tablets as end-user terminals by offering mobile applications both for taxi drivers and passengers.

In order to maximize the accessibility of the service, the system should offer a native application for the three major mobile platforms (iOS, Android, Windows Phone). The mobile applications destined to taxi drivers and passengers should be designed taking into account the fact that only a fraction of our potential customers use a latest-generation smartphone. As such, the mobile application should use no more than 64MB of RAM to execute and should be able to run correctly on smartphones with single core processors clocked at 800Mhz or more.

The central system should be designed to run on standard x86 server machines.

2.4.3 Interfaces to other applications

The system has to interface with an external mapping provider in order to calculate the ETA and to reverse-geocode location data into addresses.

2.4.4 Parallel operation

The system has to support multiple concurrent transactions, both in terms of ride requests and in terms of managing the transition of taxis among the different zones.

2.4.5 Criticality of the application

The system is not considered to be mission-critical, meaning that an unexpected failure doesn't imply risks for human lives. However, it should be considered that the taxi service of a large city is rather critical and should, at least theoretically, be always available. As such, the system is still considered of critical importance in order to avoid paralysis of the taxi infrastructure, which would lead to not-negligible negative consequences for the city mobility.

2.4.6 Safety and security considerations

Given its relative criticality, it is important to ensure that the system is adequately protected against attacks and misusage. A list of safety requirements will be analyzed more in detail in the system attributes section of this document.

2.5 Domain properties and assumptions

- Zones do not overlap.
- A taxi driver drives only a single taxi.
- A taxi can be driven only by a single taxi driver.
- Passengers can only request rides using either the web application or the mobile application provided by the system.
- Passengers do not request rides using their personal acquaintance with taxi drivers.
- Taxi drivers will only make rides by accepting a request from the system.
- Contact information of taxi drivers are not disclosed by the city administration.
- All the data already stored in the system-as-is is actually transferred to the system-to-be when it is put into operation.
- Uses don't have the need to choose a particular taxi driver among those that are available for their ride.
- Passengers do not have to confirm the choice of the selected taxi.
- The city council has divided the city in taxi zones.
- The taxi service is operated 24h/day, unless exceptional circumstances occur.
- In case any exceptional circumstance occur for which the taxi service cannot be operated, the system is not required to satisfy incoming and reserved requests.
- For any given GPS location inside the boundaries of the city, it is always possible to reverse-geocode it into a valid city address.
- A taxi driver will take a passenger onboard if and only if both the taxi code and the secret security code sent to the passenger are equal to his own.

- Every taxi driver is provided with a business mobile phone by the city council.
- The business phone of every taxi driver has always an active data plan.
- Every taxi is uniquely identified by an alphanumeric code that is assigned by the city administration when he becomes part of the taxi service.
- Passengers are not allowed to place reservations more than 15 days in advance.
- The city council may decide to update the taxi zone division in the future.
- The only taxis eligible for fulfilling a reservation are the ones that are present in the queue of the zone associated with the reservation source address 10 minutes before the scheduled meeting time.
- In order to receive ride requests, a taxi driver must explicitly mark himself as available.
- A taxi driver will not be able to receive ride requests while he's unavailable.
- Taxis that are considered to be out-of-city will not be able to receive calls.
- A taxi driver who is currently on a ride will not be able to receive calls.
- A taxi driver must always notify the system when he terminates a ride.
- A passenger is not required to be registered in order to request a ride.
- A passenger must be registered and logged in to the system in order to reserve a ride.
- After a taxi driver has been associated to a call, he must confirm or refuse the request within two minutes. After that period of time, the call is considered refused.
- Traffic information and route data can be obtained from a third party mapping service.
- A passenger that has already requested a taxi will not request another one before having completed his ride.

- If no taxis are available to fulfill a reservation 10 minutes before the scheduled meeting time, new attempts of rescheduling are to be made at intervals of 2 minutes for at most 20 times.
- A taxi driver that has received a request will not change zone before accepting (or rejecting) it.

Chapter 3

Specific requirements

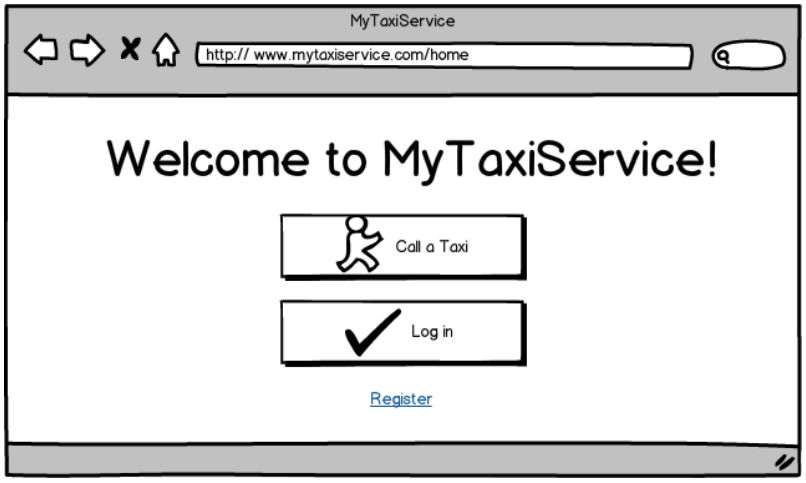
3.1 External Interface Requirements

3.1.1 User interface sketches

Here follows a sequence of sketches for possible implementations of the user interfaces.

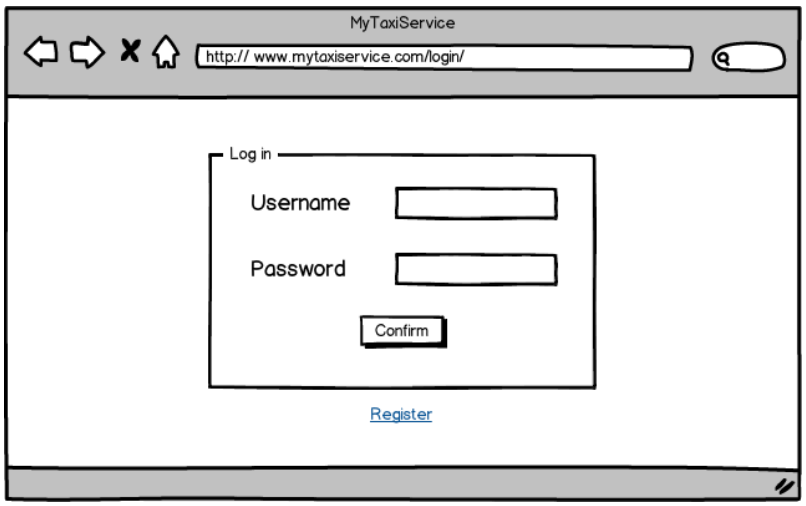
Sketch 1

This sketch shows a possible implementation of the main page of the web application. This page is shown to all guest passengers, and it allows a guest to either login or request a taxi.



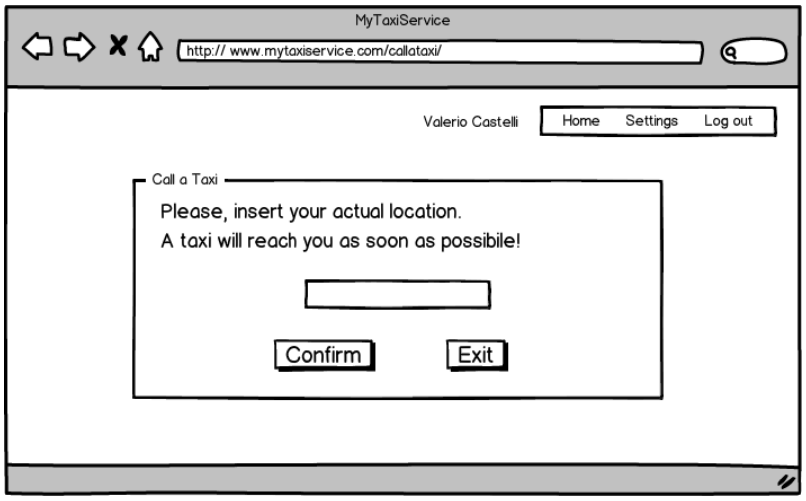
Sketch 2

This sketch shows a possible implementation of the login form of the web application.



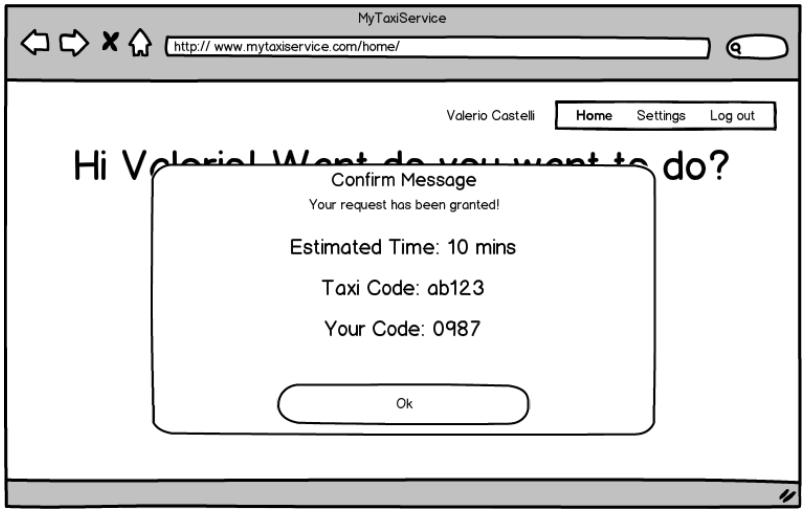
Sketch 3

This sketch shows a possible implementation of the "request a taxi" page of the web application. This page is shown when a passenger clicks on the "request a taxi" button on the main page. The user is asked to insert the address where he wants the taxi to be dispatched.



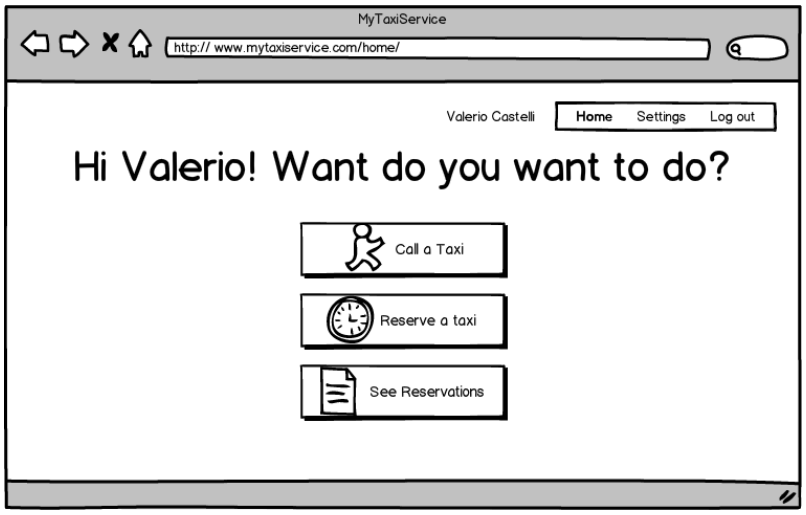
Sketch 4

This sketch shows the popup notification message that is shown to the passenger when his taxi request has been processed by the system. It shows the estimated time of arrival, the code of the incoming taxi and the security code.



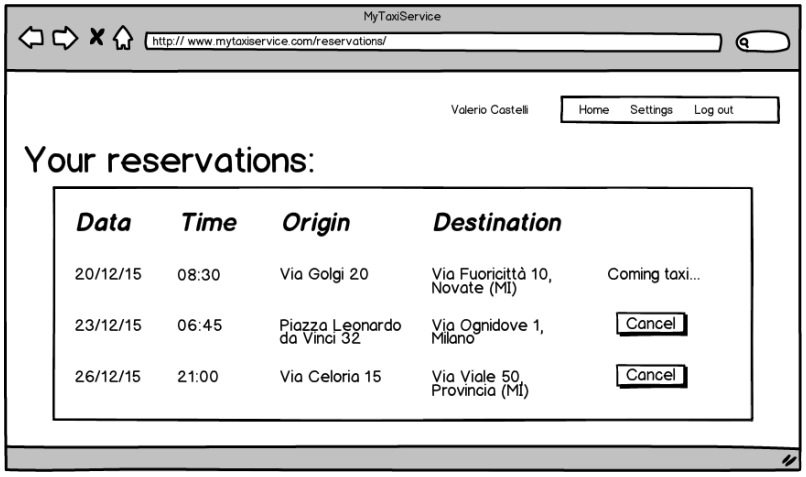
Sketch 5

This sketch shows a possible implementation of the personal webpage of a logged-in passenger in the web application. From this page, a logged-in passenger can request a taxi, place a reservation for a taxi or see the reservations he has already placed.



Sketch 6

This sketch shows a possible implementation of the "show reservations" feature for logged-in passengers.



Sketch 7

This sketch shows a possible implementation of the popup notification that is shown to a taxi driver when he accepts a ride. It shows the meeting address and the secret code.

Sketch 8

This sketch shows a possible implementation of the main screen of the taxi driver mobile application. In this screen, a taxi driver can see his current

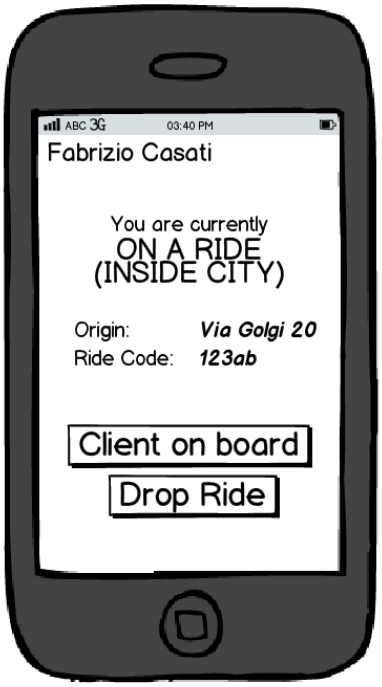
status, check the zone he's in and the number of taxis ahead of him in queue. He can also change his status to unavailable.

Sketch 9

This sketch shows a possible implementation of the login screen of the taxi driver mobile application.

Sketch 10

This sketch shows a possible implementation of the screen that is shown to the taxi driver when he is on a ride. It shows his status, the meeting address and the secret code. It also allows the taxi driver to confirm that the passenger has got onboard or drop the request if the passenger doesn't show up.



Sketch 11

This sketch shows a possible implementation of the popup notification that is shown to a taxi driver when the system assigns him a request. In particular, the meeting address is shown, and the taxi driver has the ability to either accept or refuse the ride.

Sketch 12

This sketch shows a possible implementation of the main screen of the taxi driver mobile application when the taxi driver has marked himself as unavailable. The only action that a taxi driver can do here is change his status to available.

3.2 Functional Requirements

3.2.1 Goal 1

The city administration must have the possibility to enter the data about the taxi drivers and about the taxi zones.

Requirements:

- The system must provide a data entry form to collect data about the taxi drivers. In particular, the following information is required for each taxi driver: name, surname, mobile phone number, taxi code, taxi driver license, taxi license plate.
- The system must provide a way to specify the taxi zone division of the city.
- The system must provide a way for the operator to confirm that the initial data entry operation is complete.
- The system will not allow any operation to take place until data entry is marked as completed.

3.2.2 Goal 2

The city administration must have the possibility to update taxi driver data.

Requirements:

- The system should provide a way to update existing taxi driver data.
- The system should provide a way to remove a taxi driver from the list.
- The system should provide a way to insert a new taxi driver.

3.2.3 Goal 3

The city administration must have the possibility to update the taxi zone division.

Requirements:

- The system must provide a way to update the boundaries of an existing taxi zone.
- The system must provide a way to insert a new taxi zone.
- The system must provide a way to remove an existing taxi zone.

3.2.4 Goal 4

Taxi drivers must be able to communicate their availability status.

Requirements:

- The taxi driver mobile application should implement a toggle to allow drivers to switch their availability status.
- A taxi driver that has marked himself as available should only be able to mark himself as unavailable.
- A taxi driver that has marked himself as unavailable should only be able to mark himself as available.
- A taxi driver should be able to mark himself as available only if it's inside the city.
- A taxi driver who is currently on a ride should not be able to mark himself as unavailable.
- When a taxi driver marks himself as available, the mobile application should send his GPS coordinates to the central system.
- When a taxi driver marks himself as available, the system should use the retrieved GPS coordinates to compute the taxi zone to which the taxi belongs and move its code into the last position of the corresponding queue.
- If the system successfully registers a taxi driver as available, it should send him a positive notification on the mobile application. Otherwise, it should send him a negative notification and ask him to retry later.

- When a taxi driver marks himself as unavailable while being inside the city, his taxi code should be moved from the queue of the corresponding zone to the unavailability list.
- When a taxi driver marks himself as unavailable, the status of his taxi should be updated accordingly.
- When a taxi driver marks himself as unavailable, he should receive a notification warning him that he's no longer available to accept requests.

3.2.5 Goal 5

The allocation and distribution of taxis should be managed fairly and consistently.

Requirements:

- When the system starts, it should consider all drivers to be unavailable by default.
- The unavailability status of a taxi driver should not be affected by him leaving or entering the city.
- Each taxi zone must be internally associated with a queue containing the codes of the available taxis.
- The system must be able to distinguish the different states in which a taxi could be: that is available, unavailable, currently on a ride, out of the city.
- If a taxi which has marked himself as available changes zone while he's not on a ride, the system should move his code in the queue of the new zone.
- If a taxi which is currently on a ride changes zone, the system should not add his code in the queue of the new zone and his status should remain "currently riding".
- If a taxi which has marked himself as available exits the city while he's not on a ride, the system should update his status accordingly.
- If a taxi which is currently on a ride exits the city, the system should keep his status as "currently riding".
- If a taxi whose status is set as "out of the city" enters the city and the taxi was available before leaving the city and he has not marked himself as unavailable, the system should assign it to the queue of the zone in which he has entered.

- If a taxi whose status is set as "out of the city" enters the city and the taxi has signaled himself as unavailable while he was out, the system should set its status as "unavailable" and move its code to the unavailability list..
- If a taxi leaves the city while he's unavailable, the system should prevent him from signaling he's available until he is again inside the perimeter of the city.
- The mobile application of each taxi driver should periodically send his GPS coordinates to the main system.

3.2.6 Goal 6

The taxi driver should be able to receive, accept and refuse ride requests.

Requirements:

- The taxi driver that has been selected for a ride should receive a notification containing details on the meeting location and be allowed to accept or refuse the request.
- The system should be able to reverse-geocode the GPS coordinates of the meeting location into a valid city address.
- The system should provide an appropriate interface to let the driver accept or refuse the ride request.
- If a taxi driver accepts a request for a ride, the system should update its taxi status to "currently riding" and move its code to the currently riding list.
- If a taxi driver refuses a request for a ride, his taxi code should be moved to the last position of the queue of the zone he's currently in.
- If the selected taxi driver does not answer within two minutes from the assignment, the system should consider the request refused.
- If it's not possible to establish an internet connection to the central system, the mobile application should notify the taxi driver that it's unable to operate correctly.
- The mobile application should include the map of the city.

3.2.7 Goal 7

The taxi driver should be able to drop a request if the passenger doesn't show up.

Requirements:

- If a passenger who has requested a ride doesn't show up at the meeting location, the system should allow the taxi driver to drop the request.
- In order for the drop request functionality to be enabled, the system should check that the taxi driver has effectively reached the meeting point.
- If a taxi driver uses the drop request functionality, the system should notify the passenger that his request has been dropped.

3.2.8 Goal 8

The taxi driver should be able to communicate he has terminated a ride.

Requirements:

- A taxi driver who has terminated a ride should be able to notify the system that the ride has ended.
- A taxi driver should be able to notify that he's terminated a ride only after the system has checked that the meeting point has actually been reached within a given level of precision.
- When the system receives the notification that a driver has terminated a ride and the driver is still outside the city, the system should set its status as "out of the city" and move its code to the out of city list.
- When the system receives the notification that a driver has terminated a ride and the driver is inside the city, the system should mark its taxi as available and assign it to the queue of the zone he is in.

3.2.9 Goal 9

Passengers should be able to request rides.

Requirements:

- The system should not allow for requests to have an origin outside the city.
- The system should provide a way for passengers to request a ride.
  - The mobile application should be able to detect the passenger's GPS location and use it as the source point of a ride request.
  - The mobile application should allow the passenger to enter an address as the source point of a ride request.
  - The web application should allow the passenger to enter an address as the source point of a ride request.

- If it's not possible to retrieve the GPS location of the passenger (either because there is no satellite coverage or because the authorization to use location services has been refused), the system should notify the passenger that it's unable to automatically locate his position.
- If it's not possible to establish an internet connection to the central system, the mobile application should notify the passenger that it's unable to operate correctly.
- In case the internet connection goes down at any point during the request process, the mobile application should abort the process and notify the passenger that the request could not be completed.
- When a passenger requests a ride, the system should use the source field of the request to compute the corresponding taxi zone.
- After a request's taxi zone has been computed, the system should assign the request to the first taxi in that zone queue.
- If a taxi driver accepts a request for a ride, the system should send a notification to the requesting passenger containing the code of the incoming taxi and an estimation of the waiting time (ETA).
- If the selected taxi driver refuses the request, the system should move his taxi code to the last position of the queue and forward the request to the following taxi in the queue.
- If the system is not able to find any available taxi in the queue of the specified zone, it should notify the passenger that the request could not be fulfilled.
- The system should keep track of the taxi drivers who have already refused a request.
- If all taxi drivers refuse a certain request, the system should notify the passenger that the request could not be fulfilled.

3.2.10 Goal 10

Passengers should be able to register.

Requirements:

- The system should provide a registration form offering the following mandatory fields: name, surname, email, mobile phone number.

3.2.11 Goal 11

Logged in passengers should be able to reserve rides in advance.

Requirements:

- The system should keep an association between a passenger profile and his reservations.
- The system should provide a way for logged in passengers to reserve a ride for a specific date and time.
  - The mobile application should allow the passenger to enter the source and destination addresses for the ride and the meeting time.
  - The web application should allow the passenger to enter the source and destination addresses for the ride and the meeting time.
- If it's not possible to establish an internet connection to the central system, the mobile application should notify the passenger that it's unable to operate correctly.
- In case the internet connection goes down at any point during the reservation process, the mobile application should abort the process and notify the passenger that the request could not be completed.
- The system should provide a way for logged in passengers to review their active reservations.
- The system should provide a way for logged in passengers to delete a reservation.
- The system should store reservations in a reservation list.
- For every stored reservation, the system should proceed with selecting a taxi only 10 minutes before the scheduled meeting time.
- For every stored reservation, the system should only select taxis from the queue of the zone associated with the source address.
- The system should not allow a reservation to be scheduled if there aren't at least two hours between the desired meeting time and the moment when the reservation request is submitted.
- The system should not allow a reservation to be scheduled more than 15 days in advance.
- Reservations can't be deleted after a taxi has already been associated to them.

- If no taxis are available to fulfill a reservation 10 minutes before the scheduled meeting time, the system should attempt to reschedule it at intervals of 2 minutes for at most 20 times.
- If a reservation can't be fulfilled after 20 reschedules, the system should notify the passenger that his reservation will be dropped.

3.2.12 Goal 12

The taxi driver associated with a ride request or reservation can only take onboard the passenger who actually requested the ride.

Requirements:

- For each ride request, the system should generate a random 4 digit security number.
- The security number must be sent to the taxi driver upon his confirmation that he will take care of the ride.
- The security number must be included into the incoming taxi notification sent to the passenger who requested the ride.

3.3 Performance Requirements

Since the system is expected to operate in a medium-large sized city, it has to offer a certain level of performance in order to be considered usable. Since no actual data is available for our particular city, the following figures have been estimated by considering the structure of existing taxi services in major cities in the world.

In particular, the following requirements have to be met:

- The system should support at least 10.000 overall taxi drivers, with the pessimistic assumption that all of them might be in service at the same time.
- The system should support at least a million registered passengers.
- The system should support at least 1000 concurrent taxi requests.
- The response time should be less than 1s for 95% of all transactions.

3.4 Design Constraints

3.4.1 Standard compliance

- The system has to store taxi license plates in a format that is compliant with the national laws governing license plates issuing.

- The system has to store taxi licenses and driving licenses accordingly to the national law.
- The system has to comply to the regulatory policies regarding taxi services.

3.5 Software System Attributes

3.5.1 Reliability

In order to keep maintenance costs low, the system should offer a level of reliability such that no key component of the system should fail more than once a year.

3.5.2 Availability

The system is required to have an availability of 99.99%. This means that it's considered acceptable to have a downtime of up to 1 hour every year due to malfunctioning of the system. This implies that, at the architectural level, the system has to include a certain level of replication of its hardware structures in order to overcome potential failures.

3.5.3 Security

The system should guarantee that only authorized taxi drivers are able to access and operate the mobile taxi driver application. The system should guarantee that only authorized administrative personnel can have access to the service management panel. Before confirming an update of the taxi zone division or of the taxi driver list, the system should require a password and perform a safety backup.

Moreover, the central infrastructure of the system must implement proper mechanisms to be resistant to external malicious attacks.

In particular, it should be robust with respect to DDoS, SQL injection and other common attack vectors; furthermore, only the connections necessary for the correct operations of the services should be allowed, with all the other ones being blocked by a physical firewall. For what concerns security of users' (taxi drivers and passengers) data, passwords will have to be encrypted.

The disk drives of the servers that are going to run the system will have to be encrypted too, in order to neutralize physical attacks to the infrastructure.

Adequate protection against fire, floods and other disasters should be taken into account.

3.5.4 Portability

The system should be designed to be easily portable. In particular, the mobile side of the system should be coded using a programming language and a set of frameworks that can run on the major mobile platforms (iOS, Android, Windows Phone) with the option of being easily portable on new ones, in order to anticipate for future market changes. No platform-dependent frameworks should be used while coding the mobile applications.

3.6 Scenarios

3.6.1 Scenario 1

Alice has just got out of her office after an intense day of work. Unfortunately she finished late and so she missed the last train back home.

She decides then to request a taxi. She opens the taxi application on her mobile phone and she clicks on the "request a taxi for this location" button.

The mobile application detects her GPS location, creates a taxi request and forwards it to the central system.

The system receives the taxi request, extracts the GPS coordinates and realizes that Alice is in zone 2. Thus, it forwards a request to the first taxi in the queue associated with zone 2, which happens to be Bob's taxi.

Bob receives a request notification on his mobile terminal, which employs the associated GPS information to point out Alice's location on the city map. Bob decides to accept the request and clicks on the "accept request" button.

The system receives Bob's confirmation along with his GPS coordinates and moves his taxi code from the zone 2 queue to the currently riding list; it then uses this information to retrieve from an external mapping service an estimate time of arrival (ETA) and notifies it to Alice along with the taxi code of Bob's taxi and a security code associated with the ride. The system also notifies Bob the security code associated with the ride; then Bob starts driving.

Alice receives the notification on her mobile phone and patiently waits for the taxi to arrive.

When Bob's taxi arrives, Alice tells Bob the security code and the taxi code she received from the system. Bob checks they correspond to his own and let Alice get on the car.

3.6.2 Scenario 2

Jessica and Davide are planning to celebrate their anniversary, which is happening in a week, by going out and having dinner at a very expensive restaurant.

Since they probably would end up being drunk after the night, they decide it would be better to avoid driving and they decide instead to reserve a taxi.

Davide opens the taxi reservation web page on his laptop and logins using his credentials (he's a proud taxi user!). He then clicks on the "reserve a taxi" button and fills in the required details for the reservation: the departing address, the destination address and the date and time of the meeting. He then confirms.

The system receives the request and stores it into the list of reservations, then sends a confirmation back to Davide's laptop.

Davide receives the confirmation and is assured that the system will handle his request.

3.6.3 Scenario 3

On the night of their anniversary, Jessica and Davide are getting ready to leave.

10 minutes before the scheduled meeting time, the system uses the specified departing address to compute the zone associated with Davide's apartment and realizes he lives in zone 4. It then forwards the request to the first taxi in the queue associated with zone 4, which happens to be Matteo's taxi.

Matteo receives a request notification on his mobile terminal, which employs the associated GPS information to point out Davide's apartment location on the city map. Matteo decides to accept the request and clicks on the "accept request" button.

The system receives Matteo's confirmation along with his GPS coordinates, updates his taxi status to "currently riding" and moves it from the zone 4 queue to the "currently riding" list; it then uses this information to retrieve from an external mapping service an estimate time of arrival (ETA) and notifies it to Davide along with the taxi code of Matteo's taxi and a security code associated with the ride. The system also notifies Matteo the security code associated with the ride, then Matteo starts driving.

Davide receives the notification on his mobile phone and patiently waits for the taxi to arrive.

When Matteo's taxi arrives, Davide tells Matteo the security code and the taxi code he received from the system. Matteo checks they correspond to his own and let Davide and Jessica get on the car.

3.6.4 Scenario 4

Matteo has just driven Davide and Jessica to their destination, which is a restaurant in the middle of the city. He clicks the "confirm ride ended"

button in the mobile application. The mobile application sends a notification message to the central system, which acknowledges that the ride has ended.

Since Matteo was previously available before the ride and has not switched the available/unavailable toggle, the system computes the zone associated with his current GPS coordinates and moves his taxi into the last position of the respective queue.

3.6.5 Scenario 5

Bob has just driven Alice back to her apartment, which is a small flat in the suburbs (but still belonging to the city). He clicks the "confirm ride ended" button in the mobile application. The mobile application sends a notification message to the central system, which acknowledges that the ride has ended.

Since Bob was previously available before the ride and has not switched the available/unavailable toggle, the system computes the zone associated with his current GPS coordinates and moves his taxi in the respective code, in the last position.

At this point, however, Bob feels a little tired and decides to take a nap, so he switches the available/unavailable toggle to the unavailable position.

The mobile application sends a notification to the central system, which acknowledges Bob's decision, sets the status of its taxi to unavailable and proceeds to move it from the queue of his current zone to the unavailability queue. It then sends back a notification to Bob informing him that he's been correctly registered as unavailable and that he will not be able to receive further requests as long as he remain so.

Bob can now take some well deserved rest.

3.6.6 Scenario 6

Frank, a taxi driver, has just picked up Samuel from a very famous hotel in the city.

During their journey to the airport, which is Samuel's destination, Frank has to cross the city border: in fact, the airport is located just outside the city.

After leaving Samuel at the airport, Frank clicks the "confirm ride ended" button in the mobile application.

The mobile application sends a notification to the central system, which acknowledges that the ride has ended.

Since Frank is now located outside the city, the system changes the status of his taxi to "outside city" and moves it to the corresponding list.

3.6.7 Scenario 7

As soon as Frank crosses the city border the mobile application, which has been continually sending GPS location data to the central system for the whole journey, sends a notification to the central system containing the new location data.

The system determines that Frank's current location is now inside the boundaries of the city and computes the associated zone.

Since Frank had flagged himself as available before accepting Samuel's request and has not switched the available/unavailable toggle for the whole journey, the system moves his taxi into the last position of the computed zone queue.

Appendix A

Use cases

A.1 Request a taxi

Name	Request a taxi
Actors	Passenger, Taxi driver
Entry condition	The passenger presses the "Call Taxi" button in the mobile app.
Flow of events	The mobile app of the passenger detects the GPS location, creates a taxi request and forwards the request to the central system. The system receives the request and extracts the GPS location to compute the zone where the passenger is. The system selects the first available taxi driver from that zone queue and send him the request. The taxi driver receives the request and accepts it. The mobile app of the taxi driver sends a confirmation to the system. The system receives the confirmation and sends a notification to the passenger including the taxi code, a security code and an estimated time of arrival. The taxi driver checks that the security number and the taxi code sent to the passenger correspond to his own.
Exit condition	The taxi driver confirms that the passenger is on board.

Exceptions	- If the passenger mobile app cannot receive the GPS location, the mobile app asks the passenger to insert his position manually. - If the passenger cannot connect to Internet, the mobile app tells the passenger that he can't call a taxi at the moment. - If the taxi driver refuses the call (or doesn't answer in 2 minutes), the system moves the taxi driver in the last position of the zone queue and forwards the request to the next available taxi driver. - If the system cannot forward the request to the taxi driver because he is not connected to Internet, it moves the taxi driver in the last position of the zone queue and forwards the request to the next available taxi driver. - If all taxi drivers refuse the request, the system notifies the passenger that his request could not be fulfilled. - If the passenger doesn't show up at the meeting location, the taxi driver clicks on the "drop call" button. The system will then notify the user that his request has been dropped.

A.2 Reserve a taxi via the web application

Name	Reserve a taxi via the web application
Actors	Passenger
Entry condition	The passenger opens the main page of the web application.
Flow of events	The passenger logins, clicks on the "reserve a taxi" button and enters the departing address, the destination address and the date and time of the meeting. The passenger confirms the data and clicks on the "confirm reservation" button. The system receives the request, stores it into the list of reservations and sends a confirmation.
Exit condition	The passenger receives the confirmation.
Exceptions	- If one of the inserted fields in incorrect, the app requests the passenger to insert it again.

A.3 Allocate a reservation to a taxi

Name	Allocate a reservation to a taxi
Actors	Passenger, Taxi driver
Entry condition	The system checks if a reservation must be allocated, and discovers that there is one.
Flow of events	The system uses the specified departing address to compute the corresponding zone and forwards the request to the first taxi in the queue associated with that zone. The selected taxi driver receives a notification and clicks on the "accept request" button. The system receives the taxi driver's confirmation, sets his status to "currently riding" and moves his taxi code from the current taxi zone queue to the "currently riding" list. The system uses this information to retrieve from an external mapping service an estimate time of arrival (ETA) and sends it to the passenger in a notifications along with the security code associated with the ride and the code of the assigned taxi. The passenger receives the notification on his mobile phone and waits for the taxi. The taxi driver drives to the meeting location. After arriving to the meeting location, the taxi driver checks that the passenger's security number and taxi code matches his own.
Exit condition	The taxi driver take the passenger on board.

Exceptions	- If the taxi driver refuses the call (or do not answer in 2 minutes), the system moves the taxi driver in the last position of the zone queue and forwards the request to the next available taxi driver. - If the system cannot forward the request to the taxi driver because he is not connected to Internet, it moves the taxi driver in the last position of the zone queue and forwards the request to the next available taxi driver. - If all taxi drivers refuse the request, the system notifies the passenger that is request could not be fulfilled. - If the passenger doesn't show up at the meeting location, the taxi driver clicks on the "drop call" button. The system will then notify the user that his request has been dropped.

A.4 End a ride

Name	End a ride
Actors	Taxi driver
Entry condition	A taxi driver clicks the "confirm ride ended" button in the mobile application while being inside the city.
Flow of events	The mobile application sends a notification message to the central system. The system receives and computes the zone associated with the taxi driver's current GPS coordinates. The system moves the taxi into the zone queue and sets the taxi as available.
Exit condition	The taxi driver receives a confirmation that he's now considered available.
Exceptions	- If the taxi driver has concluded his ride outside the city, when he presses the "confirm ride ended" button the system will change his status to "outside city" and move his taxi code to the corresponding queue.

A.5 Communicate taxi unavailability

Name	Communicate taxi unavailability
Actors	Taxi driver
Entry condition	The taxi driver switches the availability toggle in the mobile application to the “unavailable” position.
Flow of events	The mobile application sends a notification message to the central system, The system receives the notification and sets the taxi driver state to “unavailable”, then proceeds to move his taxi code from the queue of his current zone to the unavailability queue. The system sends a notification to the taxi driver to confirm he’s been registered as unavailable.
Exit condition	The taxi driver receives the notification.
Exceptions	- If the taxi driver’s app cannot connect to internet, the app shows an error message saying that his current status is unchanged. - If the system cannot notify the taxi driver, it has to retry to forward the notification for a fixed number of times.

A.6 Enter inside city

Name	Enter inside city
Actors	Taxi driver
Entry condition	The taxi driver crosses the city border.
Flow of events	The mobile app of the taxi driver sends a notification to the central system containing the new location data. The system receives the notification and determines that the taxi driver’s current location is now inside the boundaries of the city, and computes the associated zone. The system sets the taxi status as available, moves the taxi code into the last position of the computed zones queue and send a notitication to the taxi driver.
Exit condition	The taxi driver receives the notification.

Exceptions	- If the system cannot notify the taxi driver, it has to retry to forward the notification for a fixed number of times. - If the taxi driver had set himself as unavailable while being outside the city, the system does not update his status and keeps his taxi code in the unavailability list.

Appendix B

Alloy

Here follows the skeleton of a possible Alloy formalization of the proposed system. This Alloy model depicts the key concepts and entities that make up the system and the relationships between them. However, it should be noted that not all the possible requirements could be expressed in terms of this Alloy model.

In particular, Alloy's lack of support for proper variables has forced us to avoid the inclusion of integrity constraints between the previous and the next state of an entity, as this would have violated the constraints on the uniqueness of the entities identifiers (which have been expressed as facts). As such, this Alloy model should not be interpreted as an exhaustive expression of all the rules that govern the operations of the system, and should be seen instead as an attempt to capture its essential features.

```
open util/integer as integer

/* Signatures */
sig float{}
sig string{}

sig Location{
    latitude: float,
    longitude: float
}

sig LocationUpdate{
    relativeTo: Location,
    sentBy: Taxi,
    sentTo: TaxiManager
}{
}
```

```
sig Taxi{
    taxiCode: Int,
    licensePlate: string,
    taxiStatus: TaxiStatus,
    serves: lone Request,
    isManagedBy: TaxiManager
}{
    taxiCode>=0
}

sig TaxiDriver{
    name: string,
    surname: string,
    taxiLicense: string,
    drivingLicense: string,
    mobilePhoneNumber: string,
    drives: Taxi,
    isNotifiedBy: TaxiManager
}

sig Zone{
    zoneId: Int,
    contains: set Taxi,
    boundaries: some Location
}
{
    // Each zone must contain at least 3 points
       in order to have a proper boundary
    #boundaries >= 3
    zoneId >= 0
}

enum TaxiStatus{
    available,
    unavailable,
    currentlyRiding,
    outsideCity
}

sig AccessManager{
    instance: AccessManager
}

sig SettingsManager{
```

```
    instance: SettingsManager
}

sig TaxiManager{
    instance: TaxiManager,
    handles: set Reservation,
    manages: set Request,
    unavailableTaxis: set Taxi,
    availableTaxis: set Taxi,
    currentlyRidingTaxis: set Taxi,
    outsideCityTaxis: set Taxi,
    servedRequests: set Request,
    pendingRequests: set Request,
    servedReservations: set Reservation,
    pendingReservations: set Reservation
}

sig DBManager{
    instance: DBManager
}

sig NotificationManager{
    instance: NotificationManager,
    isUsedBy: TaxiManager,
    sends: set Notification
}

sig ZoneManager{
    instance: ZoneManager,
    isUsedBy: TaxiManager,
    manages: set Zone
}

abstract sig User{
    userId: Int,
    name: string,
    surname: string,
    mobilePhoneNumber: string
}{
    userId >= 0
}

sig Guest extends User{}
```

```
sig RegisteredUser extends User{
    username: string,
    password: string,
    usesAccessManager: AccessManager,
    usesSettingsManager: SettingsManager
}

sig Request{
    requestId: Int,
    location: Location,
    sentBy: User
}{
    requestId >= 0
}

sig Reservation{
    reservationId: Int,
    origin: one Location,
    destination: one Location,
    madeBy: RegisteredUser
}{
    reservationId >= 0
}

abstract sig Notification{
    notificationId: Int,
    message: string
}{
    notificationId >= 0
}

sig IncomingTaxiNotification extends Notification
   {
    ETA: Int,
    taxiCode: Int,
    secretCode: Int,
    relativeTo: Request,
    receiver: User
}{
    ETA >= 0
    taxiCode >= 0
    secretCode >= 0
    // The taxi code must belong to a real taxi
    one t1:Taxi | taxiCode=t1.taxiCode
```

```
}

sig ReservationConfirmationNotification extends
   Notification{
    relativeTo: Reservation,
    receiver: User
}

sig NoTaxiAvailableNotification extends
   Notification{
    receiver: User
}

sig TaxiSecretCodeNotification extends
   Notification{
    secretCode: Int,
    taxi:Taxi,
    relativeTo: Request,
    receiver: TaxiDriver
}{
    secretCode >= 0
}

/* Facts */

fact RequestShouldBeEitherServedOrPending{
    no r:Request, tm:TaxiManager | (r in tm.
       pendingRequests) && (r in tm.
       servedRequests)
    all r:Request | one tm:TaxiManager | (r in tm
       .pendingRequests) || (r in tm.
       servedRequests)
}


fact ReservationShouldBeEitherServedOrPending{
    no r:Reservation, tm:TaxiManager | (r in tm.
       pendingReservations) && (r in tm.
       servedReservations)
    all r:Reservation | one tm:TaxiManager | (r
       in tm.pendingReservations) || (r in tm.
       servedReservations)
}
```

```
fact IncomingTaxiNotificationBehavior{
    /* Incoming taxi notifications are always
       sent exactly to the user that
    has made the request, and their taxi code
       matches */
    all n: IncomingTaxiNotification | one r:
       Request | (n.relativeTo = r)
    all n: IncomingTaxiNotification | one r:
       Request, t:Taxi, tm:TaxiManager | ((n.
       relativeTo = r) && (r.sentBy = n.receiver
    && t.serves=r && n.taxiCode = t.taxiCode && r
        in tm.pendingRequests))
}

fact TaxiSecretCodeNotificationBehavior{
    /* Incoming taxi notifications are always
       sent exactly to the user that
    has made the request, and their taxi code
       matches */
    all n: TaxiSecretCodeNotification | one r:
       Request | (n.relativeTo = r)
    all n: TaxiSecretCodeNotification | one r:
       Request, td:TaxiDriver, t:Taxi, tm:
       TaxiManager | ((n.relativeTo = r) && (td =
        n.receiver
    && t.serves=r && td.drives = t && r in tm.
       pendingRequests))
}

fact ReservationConfirmationNotificationBehavior{
    /* Incoming taxi notifications are always
       sent exactly to the user that
    has made the request */
    all n: ReservationConfirmationNotification |
       one r:Reservation | (n.relativeTo = r)
    all n: ReservationConfirmationNotification |
       one r:Reservation | (n.relativeTo = r)
       <=> r.madeBy = n.receiver
}

fact AllNotificationInNotificationManager{
    // All notifications are sent by the
       notification manager
    all n: Notification | one nm:
```

```
        NotificationManager | n in nm.sends
}

fact AllZonesAreManagedByZoneManager{
    // All zones are actually managed
    all z: Zone | one zm: ZoneManager | z in zm.
       manages
}


fact AllTaxisAreDrivenByASingleDriver{
    // All taxis are driven by exactly a driver
    all t: Taxi | one td: TaxiDriver | t in td.
       drives
}

fact TaxiStatusCoherence{
    // The taxi status should be coherent wrt the
        TaxiManager
    all t:Taxi | t.taxiStatus = available <=> (
       one tm: TaxiManager | t in tm.
       availableTaxis)
    all t:Taxi | t.taxiStatus = unavailable <=> (
       one tm: TaxiManager | t in tm.
       unavailableTaxis)
    all t:Taxi | t.taxiStatus = currentlyRiding
       <=> (one tm: TaxiManager | t in tm.
       currentlyRidingTaxis)
    all t:Taxi | t.taxiStatus = outsideCity <=> (
       one tm: TaxiManager | t in tm.
       outsideCityTaxis)
}


fact TaxiStatusCorrectlyUpdated {
    // If a taxi is available, there should be no
        request associated
    all t:Taxi| t.taxiStatus = available implies
       (t.serves = none)
    // If a taxi is currently on a ride, there
       should be a request which is currently
       served
    all t:Taxi| t.taxiStatus = currentlyRiding
       implies (one r:Request | r in t.serves)
```

```
    // If a taxi is outside the city, it cannot
       be associated with requests
    all t:Taxi| t.taxiStatus = outsideCity
       implies (t.serves = none)
    // If a taxis is unavailable, it cannot be
       associated with requests
    all t:Taxi| t.taxiStatus = unavailable
       implies (t.serves = none)

}

fact NoLocationInTwoZones{
    // No locations should simultaneously belong
       to two different zones
    all loc: Location | (no disj z1, z2:Zone | (
       loc in z1.boundaries && loc in z2.
       boundaries))
}

fact NoTwoIdenticalLocations{
    // Two locations cannot be identical
    no disj loc1,loc2: Location | (loc1.latitude
       = loc2.latitude && loc1.longitude = loc2.
       longitude)
}

fact UniquenessOfIdentifiers{
    // Uniqueness of identifiers
    // Two zones cannot have an identical zoneId
    no disj z1,z2: Zone | (z1.zoneId = z2.zoneId)
    // Two taxis cannot have the same taxi code,
       license plate
    no disj t1,t2: Taxi | (t1.taxiCode = t2.
       taxiCode || t1.licensePlate = t2.
       licensePlate)
    // Two users cannot have the same userId
    no disj u1,u2 :User | (u1.userId = u2.userId)
    // Two registered users cannot have the same
       username
    no disj u1,u2: RegisteredUser | (u1.username
       = u2.username)
    // Two requests cannot have the same
       identifier
    no disj r1,r2: Request | (r1.requestId = r2.
```

```
        requestId)
    // Two reservations cannot have the same
        identifier
    no disj r1,r2: Reservation | (r1.
        reservationId = r2.reservationId)
    // Two taxi drivers cannot have the same taxi
         license or driving license or phone
        number
    no disj td1,td2:TaxiDriver | (td1.taxiLicense
         = td2.taxiLicense || td1.drivingLicense =
         td2.drivingLicense || td1.
        mobilePhoneNumber = td2.mobilePhoneNumber)
}

fact NoTaxiInMultipleZones{
    // No taxi can be simultaneously in two zones
    all t:Taxi | no disj z,z':Zone | (t in z.
        contains && t in z'.contains)
}

fact SingletonClasses{
    // Singletons
    #AccessManager=1
    #SettingsManager=1
    #TaxiManager=1
    #DBManager=1
    #NotificationManager=1
    #ZoneManager=1
}

/* Predicates */
pred associateRequestToTaxi(t, t':Taxi, r:
    Request){
    // preconditions
    t.taxiStatus = available
    #t.serves = 0
    // postconditions
    t'.taxiStatus = currentlyRiding
    r in t'.serves
    t'.isManagedBy = t.isManagedBy
    // we have to remove the taxi from his zone
        queue
    one z,z':Zone | (t in z.contains) implies (z
        '.contains = z.contains - t)
```

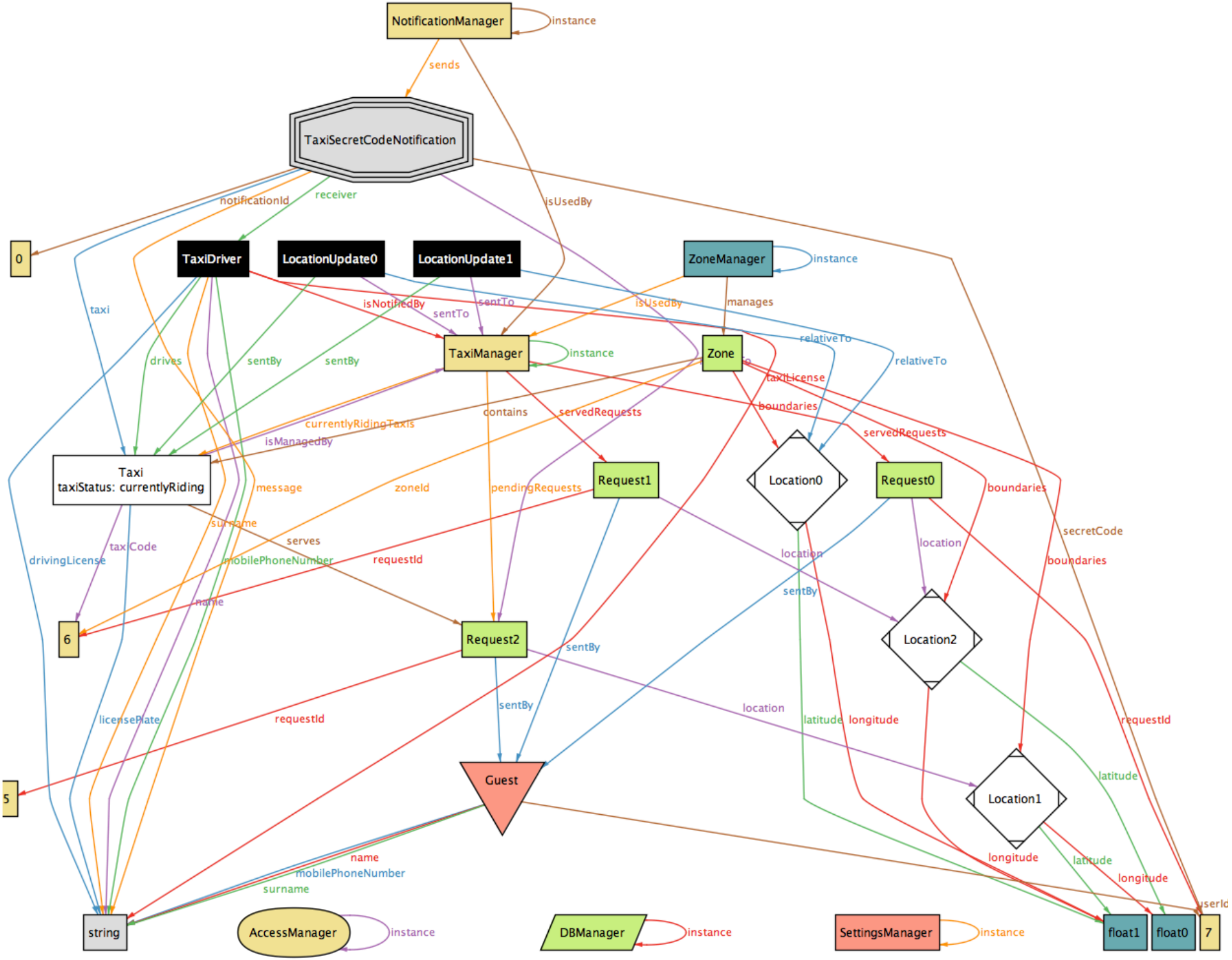
```
    // the new zone must be identical to the
       previous one
    one z,z'':Zone | (z''.contains = z.contains -
        t) implies (z.boundaries = z''.boundaries
       )
    /* the addition to the currentlyRidingTaxis
       list is guaranteed by the
    TaxiStatusCoherence fact */
    /* We can't express continuity of identifiers
        because it's in
    contrast with their uniqueness in the model (
       Alloy limitation) */
}

pred confirmRideHasEndedInZone(t, t':Taxi, r:
   Request, z:Zone){
    // preconditions
    t.taxiStatus = currentlyRiding
    #t.serves = 1
    r in t.serves
    // postconditions
    t'.taxiStatus = available
    #t'.serves = 0
    t'.isManagedBy = t.isManagedBy
    // we have to add this taxi to his current
       zone
    addTaxiToZoneQueue[t',z]
    /* the addition to the availableTaxis list is
        guaranteed by the
    TaxiStatusCoherence fact */
}

pred taxiAvailabilityToggle(t, t':Taxi, newStatus
   :TaxiStatus){
    // A taxi can become available only if it's
       not already so
    newStatus = available implies (t.taxiStatus!=
       available && t'.taxiStatus=available)
    // A taxi can become unavailable only if it's
        currently available (not on a ride, not
       outside the city)
    newStatus = unavailable implies (t.taxiStatus
       =available && t'.taxiStatus=unavailable)
}
```

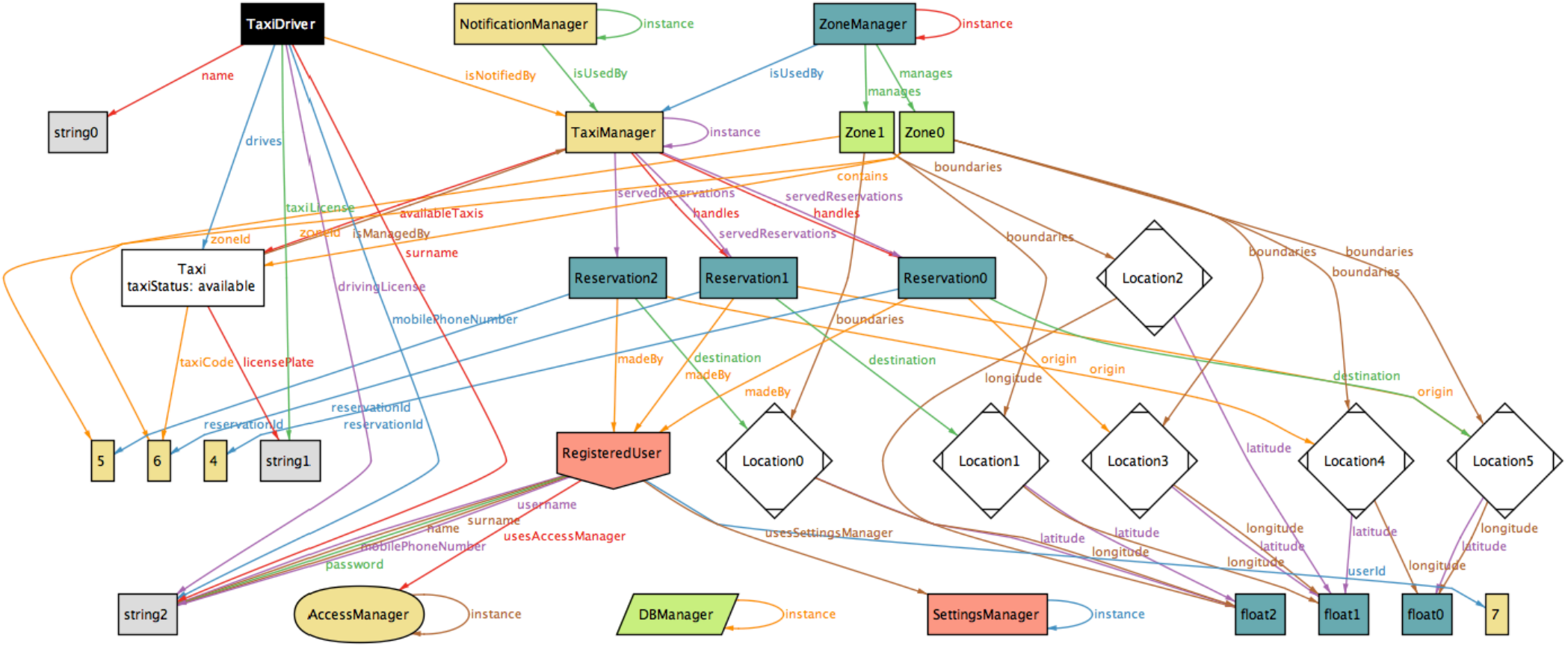
```
pred addTaxiToZoneQueue(t:Taxi, z:Zone){
    // postconditions
    one z':Zone | (z'.contains = z.contains + t)
    one z'':Zone | (z''.contains = z.contains + t
       ) implies (z''.boundaries = z.boundaries)
    /* We can't express continuity of identifier
       because it's in
    contrast with their uniqueness in the model (
       Alloy limitation) */
}

pred removeTaxiFromZoneQueue(t:Taxi, z,z':Zone){
    // Postcondition: the boundaries are the same
    one z':Zone | (t in z.contains) implies (z'.
       contains = z.contains - t) else (z'.
       contains = z.contains)
    one z'':Zone | (z''.contains = z.contains - t
       ) implies (z''.boundaries = z.boundaries)
    /* We can't express continuity of identifier
       because it's in
    contrast with their uniqueness in the model (
       Alloy limitation) */
}



pred show{
    #Taxi = 1
    #Zone = 2
    #User = 1
    #Request = 0
    #Notification = 1
}

run show for 3 but exactly 6 Location
run associateRequestToTaxi
run confirmRideHasEndedInZone
run taxiAvailabilityToggle
run addTaxiToZoneQueue
run removeTaxiFromZoneQueue
```

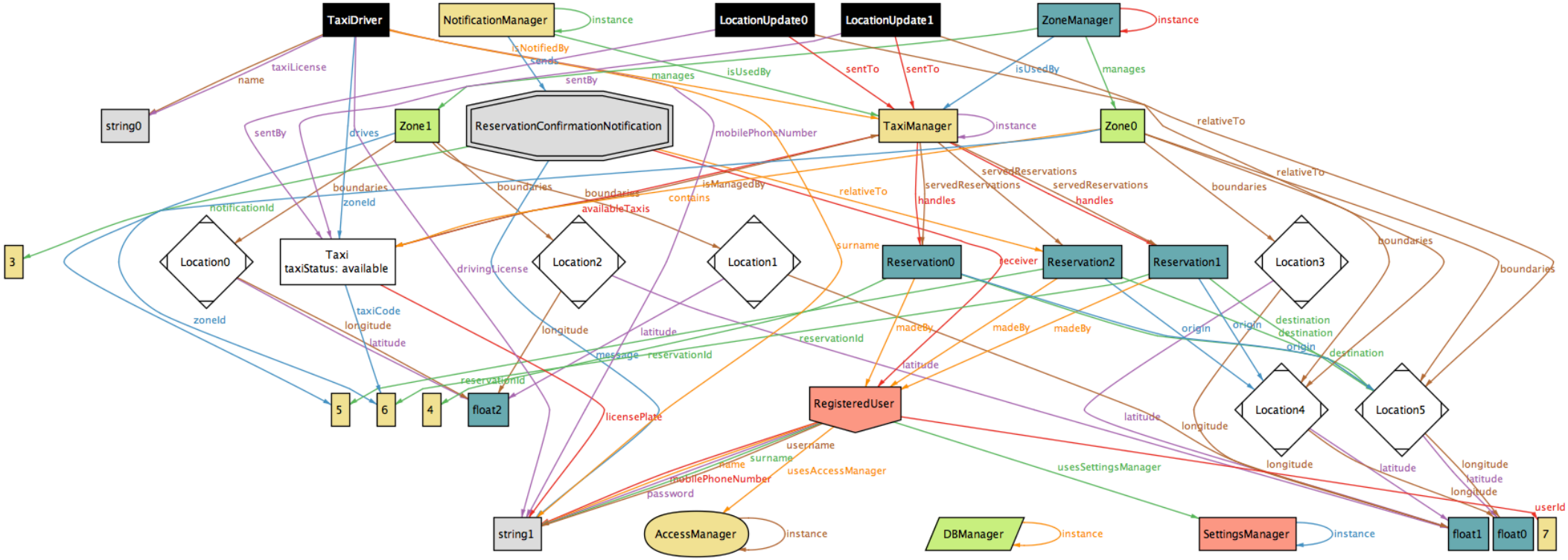
In the following pages, we will present the output of the execution of a few predicates in the models and a number of possible worlds generated by

them.

```
Executing "Run show for 3 but exactly 6 Location"
  Solver=sat4j Bitwidth=4 MaxSeq=3 SkolemDepth=1 Symmetry=20
  15432 vars. 1137 primary vars. 38671 clauses. 285ms.
  Instance found. Predicate is consistent. 149ms.

Executing "Run associateRequestToTaxi"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  14950 vars. 1086 primary vars. 37898 clauses. 299ms.
  Instance found. Predicate is consistent. 290ms.

Executing "Run confirmRideHasEndedInZone"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  14913 vars. 1089 primary vars. 37816 clauses. 229ms.
  Instance found. Predicate is consistent. 181ms.

Executing "Run taxiAvailabilityToggle"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  14713 vars. 1087 primary vars. 37288 clauses. 229ms.
  Instance found. Predicate is consistent. 145ms.

Executing "Run addTaxiToZoneQueue"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  14756 vars. 1083 primary vars. 37462 clauses. 271ms.
  Instance found. Predicate is consistent. 168ms.

Executing "Run removeTaxiFromZoneQueue"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  14816 vars. 1086 primary vars. 37620 clauses. 330ms.
  Instance found. Predicate is consistent. 282ms.
```


NotificationManager
instance
sends
TaxiSecretCodeNotification
notificationId
receiver
isUsedBy
0
TaxiDriver
LocationUpdate0
LocationUpdate1
ZoneManager
instance
taxi
isNotifiedBy
sentTo
sentTo
isUsedBy
manages
relativeTo
relativeTo
drives
sentBy
sentBy
TaxiManager
instance
Zone
taxiLicense
boundaries
currentlyRidingTaxis
contains
servedRequests
servedRequests
isManagedBy
Taxi
taxiStatus: currentlyRiding
message
zoneId
pendingRequests
Request1
Location0
Request0
boundaries
surname
serves
secretCode
taxiCode
drivingLicense
mobilePhoneNumber
requestId
location
location
boundaries
name
sentBy
6
Request2
Location2
sentBy
sentBy
location
licensePlate
requestId
latitude
longitude
requestId
Guest
latitude
5
Location1
name
mobilePhoneNumber
surname
longitude
latitude
longitude
userId
string
AccessManager
instance
DBManager
instance
SettingsManager
instance
float1
float0
7


TaxiDriver
NotificationManager
instance
ZoneManager
instance
name
isNotifiedBy
isUsedBy
isUsedBy
manages
manages
string0
drives
TaxiManager
instance
Zone1
Zone0
contains
boundaries
servedReservations
servedReservations
handles
handles
servedReservations
taxiLicense
availableTaxis
zoneId
zoneId
isManagedBy
surname
Taxi
taxiStatus: available
drivingLicense
Reservation2
Reservation1
Reservation0
Location2
boundaries
boundaries
boundaries
boundaries
mobilePhoneNumber
boundaries
taxiCode
licensePlate
madeBy
destination
madeBy
destination
origin
origin
destination
origin
madeBy
longitude
reservationId
reservationId
reservationId
5
6
4
string1
RegisteredUser
Location0
Location1
Location3
latitude
Location4
Location5
username
surname
name
mobilePhoneNumber
password
usesAccessManager
usesSettingsManager
latitude
latitude
longitude
longitude
latitude
longitude
latitude
longitude
longitude
latitude
longitude
userId
string2
AccessManager
instance
DBManager
instance
SettingsManager
instance
float2
float1
float0
7

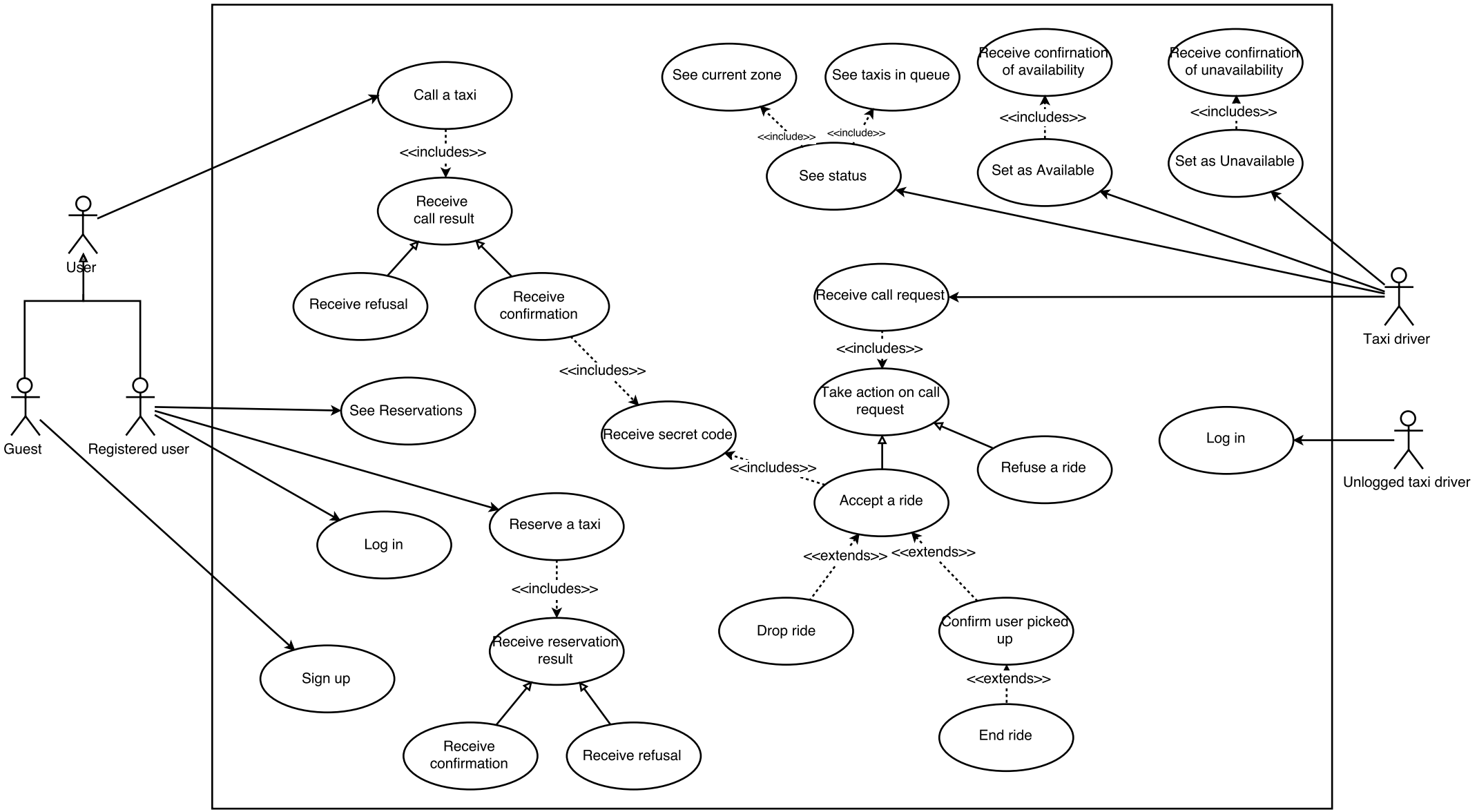

TaxiDriver
NotificationManager
instance
LocationUpdate0
LocationUpdate1
ZoneManager
instance
isNotifiedBy
sends
taxiLicense
name
sentBy
manages
isUsedBy
sentTo
sentTo
isUsedBy
manages
string0
Zone1
ReservationConfirmationNotification
mobilePhoneNumber
TaxiManager
instance
Zone0
relativeTo
sentBy
drives
boundaries
zoneId
boundaries
boundaries
availableTaxis
contains
isManagedBy
relativeTo
servedReservations
handles
servedReservations
handles
servedReservations
boundaries
relativeTo
boundaries
notificationId
3
Location0
Taxi
taxiStatus: available
drivingLicense
Location2
Location1
surname
Reservation0
receiver
Reservation2
Reservation1
Location3
boundaries
boundaries
zoneId
taxiCode
longitude
latitude
longitude
latitude
reservationId
madeBy
madeBy
madeBy
origin
origin
destination
destination
origin
destination
message
reservationId
reservationId
latitude
5
6
4
float2
licensePlate
RegisteredUser
latitude
longitude
Location4
Location5
longitude
latitude
longitude
latitude
longitude
userId
username
surname
name
mobilePhoneNumber
password
usesAccessManager
usesSettingsManager
string1
AccessManager
instance
DBManager
instance
SettingsManager
instance
float1
float0
7

Appendix C

UML diagrams

C.1 Use case diagram

Here we include a diagram which contains all the uses cases that are relevant for this system.

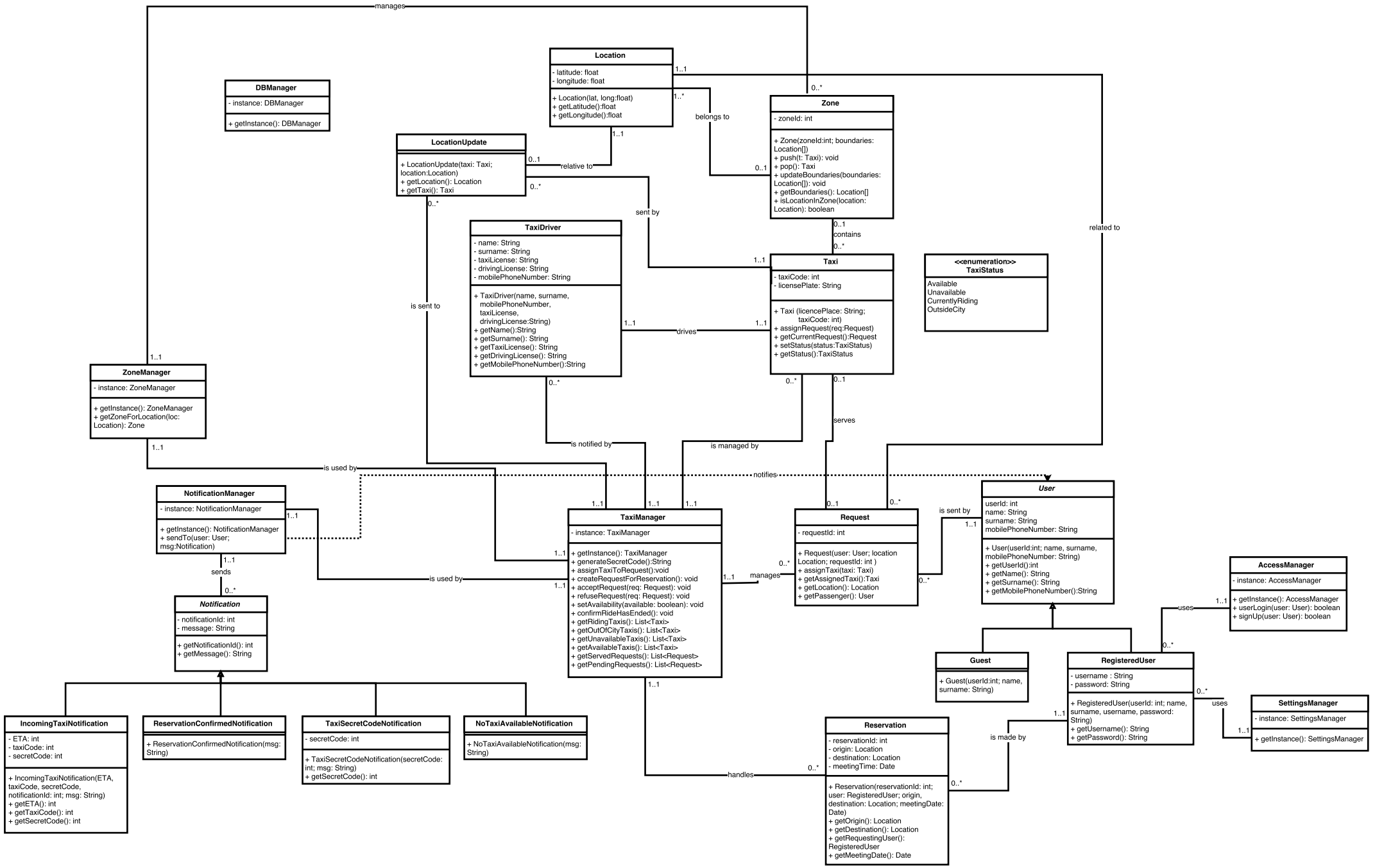

Call a taxi
<<includes>>
Receive call result
Receive refusal
Receive confirmation
<<includes>>
Receive secret code
<<includes>>
Reserve a taxi
<<includes>>
Receive reservation result
Receive confirmation
Receive refusal
See current zone
See taxis in queue
<<include>>
<<include>>
See status
Receive confirnation of availability
<<includes>>
Set as Available
Receive confirnation of unavailability
<<includes>>
Set as Unavailable
Receive call request
<<includes>>
Take action on call request
Refuse a ride
<<includes>>
Accept a ride
<<extends>>
<<extends>>
Drop ride
Confirm user picked up
<<extends>>
End ride
User
Guest
Registered user
See Reservations
Log in
Sign up
Taxi driver
Log in
Unlogged taxi driver

C.2 Class diagram

Here we include sketch of a possible class diagram for the system.

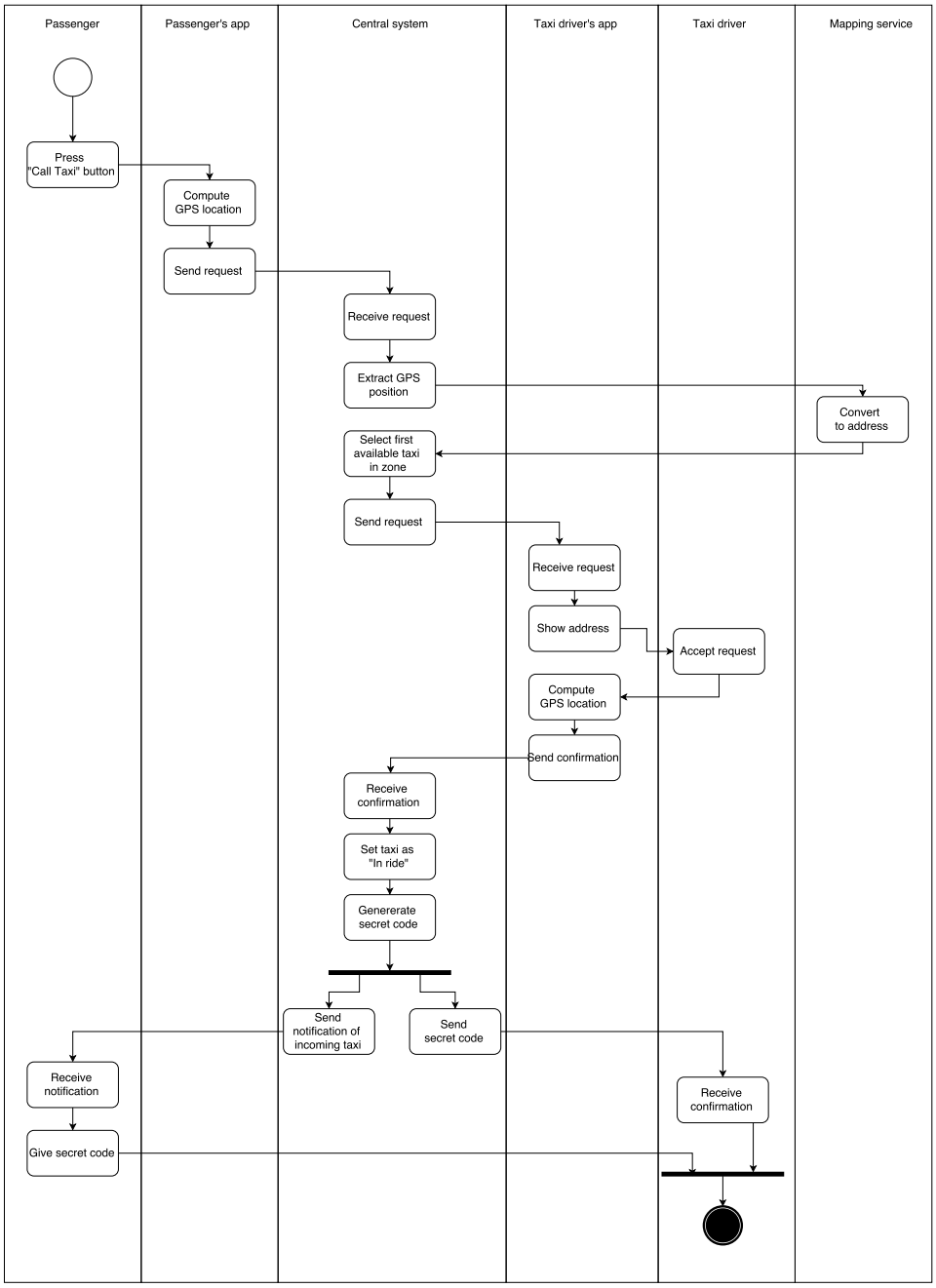
It should be noted that this diagram is intended to be used as a preliminary draft for further analysis in the design phase, and that it doesnt fully represent the final class infrastructure the shipped system will be based upon.

Furthermore, this diagram only covers the classes that are present in the central system and doesnt include the structure of the mobile applications and of the web applications.

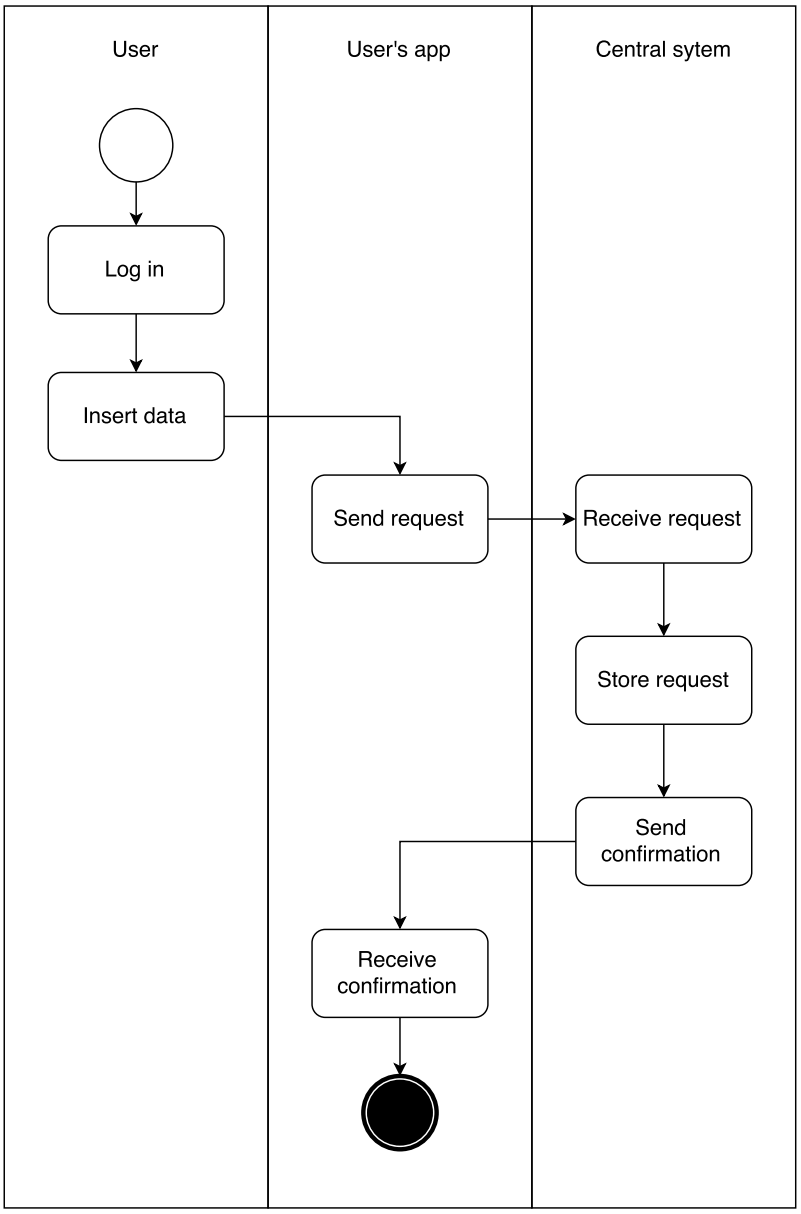

manages
DBManager
- instance: DBManager
+ getInstance(): DBManager
Location
- latitude: float
- longitude: float
+ Location(lat, long:float)
+ getLatitude():float
+ getLongitude():float
1..1
1..*
belongs to
0..*
Zone
- zoneId: int
+ Zone(zoneId:int; boundaries: Location[])
+ push(t: Taxi): void
+ pop(): Taxi
+ updateBoundaries(boundaries: Location[]): void
+ getBoundaries(): Location[]
+ isLocationInZone(location: Location): boolean
0..1
LocationUpdate
+ LocationUpdate(taxi: Taxi; location:Location)
+ getLocation(): Location
+ getTaxi(): Taxi
0..1
relative to
0..*
0..*
sent by
0..1
contains
0..*
TaxiDriver
- name: String
- surname: String
- taxiLicense: String
- drivingLicense: String
- mobilePhoneNumber: String
+ TaxiDriver(name, surname, mobilePhoneNumber, taxiLicense, drivingLicense:String)
+ getName():String
+ getSurname(): String
+ getTaxiLicense(): String
+ getDrivingLicense(): String
+ getMobilePhoneNumber():String
1..1
drives
1..1
Taxi
- taxiCode: int
- licensePlate: String
+ Taxi (licencePlace: String; taxiCode: int)
+ assignRequest(req:Request)
+ getCurrentRequest():Request
+ setStatus(status:TaxiStatus)
+ getStatus():TaxiStatus
<<enumeration>>
TaxiStatus
Available
Unavailable
CurrentlyRiding
OutsideCity
related to
is sent to
1..1
ZoneManager
- instance: ZoneManager
+ getInstance(): ZoneManager
+ getZoneForLocation(loc: Location): Zone
1..1
0..*
0..*
is notified by
is managed by
0..1
serves
is used by
notifies
NotificationManager
- instance: NotificationManager
+ getInstance(): NotificationManager
+ sendTo(user: User; msg:Notification)
1..1
1..1
sends
0..*
is used by
1..1
TaxiManager
- instance: TaxiManager
+ getInstance(): TaxiManager
+ generateSecretCode():String
+ assignTaxiToRequest():void
+ createRequestForReservation(): void
+ acceptRequest(req: Request): void
+ refuseRequest(req: Request): void
+ setAvailability(available: boolean): void
+ confirmRideHasEnded(): void
+ getRidingTaxis(): List<Taxi>
+ getOutOfCityTaxis(): List<Taxi>
+ getUnavailableTaxis(): List<Taxi>
+ getAvailableTaxis(): List<Taxi>
+ getServedRequests(): List<Request>
+ getPendingRequests(): List<Request>
1..1
manages
0..*
Request
- requestId: int
+ Request(user: User; location Location; requestId: int)
+ assignTaxi(taxi: Taxi)
+ getAssignedTaxi():Taxi
+ getLocation(): Location
+ getPassenger(): User
0..*
is sent by
1..1
User
userId: int
name: String
surname: String
mobilePhoneNumber: String
+ User(userId:int; name, surname, mobilePhoneNumber: String)
+ getUserId():int
+ getName(): String
+ getSurname(): String
+ getMobilePhoneNumber():String
Notification
- notificationId: int
- message: String
+ getNotificationId(): int
+ getMessage(): String
IncomingTaxiNotification
- ETA: int
- taxiCode: int
- secretCode: int
+ IncomingTaxiNotification(ETA, taxiCode, secretCode, notificationId: int; msg: String)
+ getETA(): int
+ getTaxiCode(): int
+ getSecretCode(): int
ReservationConfirmedNotification
+ ReservationConfirmedNotification(msg: String)
TaxiSecretCodeNotification
- secretCode: int
+ TaxiSecretCodeNotification(secretCode: int; msg: String)
+ getSecretCode(): int
NoTaxiAvailableNotification
+ NoTaxiAvailableNotification(msg: String)
handles
0..*
Reservation
- reservationId: int
- origin: Location
- destination: Location
- meetingTime: Date
+ Reservation(reservationId: int; user: RegisteredUser; origin, destination: Location; meetingDate: Date)
+ getOrigin(): Location
+ getDestination(): Location
+ getRequestingUser(): RegisteredUser
+ getMeetingDate(): Date
0..*
is made by
1..1
Guest
+ Guest(userId:int; name, surname: String)
RegisteredUser
- username : String
- password: String
+ RegisteredUser(userId: int; name, surname, username, password: String)
+ getUsername(): String
+ getPassword(): String
AccessManager
- instance: AccessManager
+ getInstance(): AccessManager
+ userLogin(user: User): boolean
+ signUp(user: User): boolean
uses
1..1
0..*
SettingsManager
- instance: SettingsManager
+ getInstance(): SettingsManager
uses
0..*
1..1

C.3 Activity diagrams

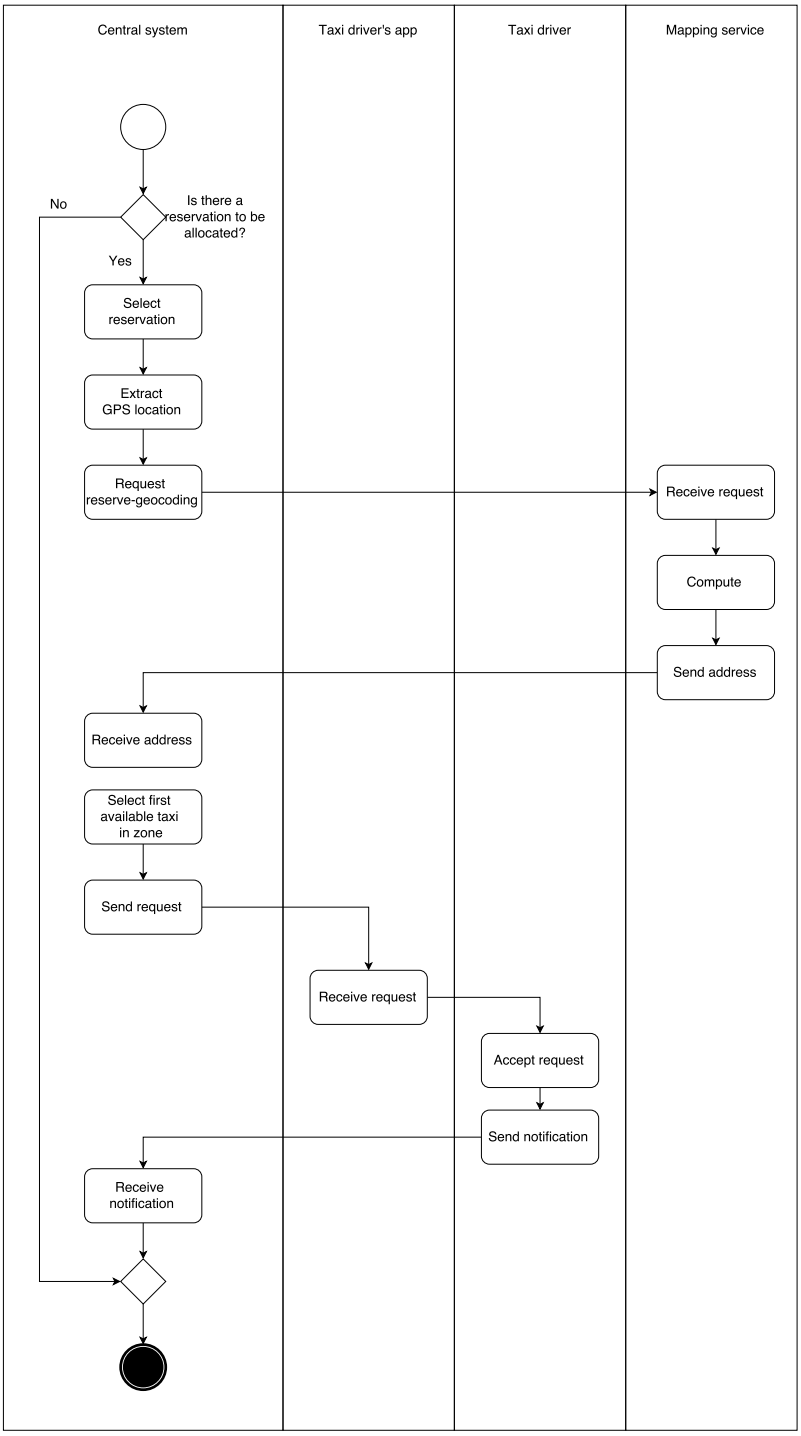
C.3.1 Request a taxi



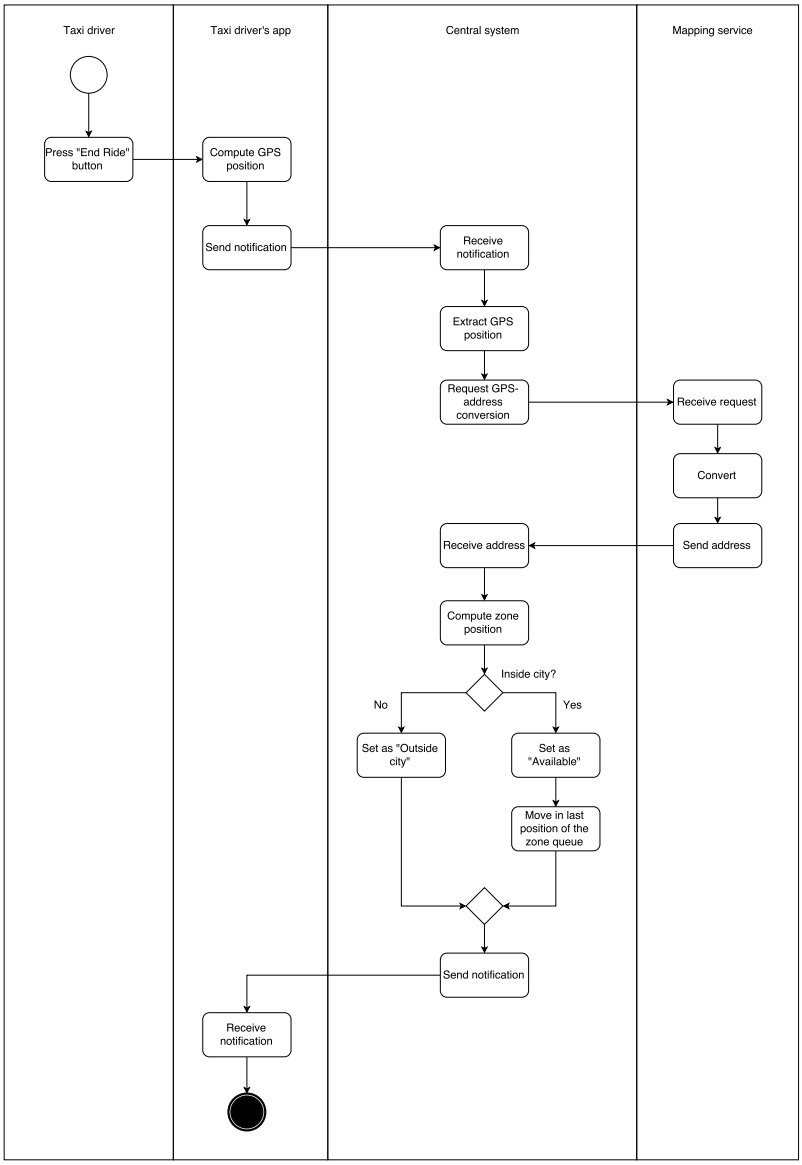
C.3.2 Place a taxi reservation



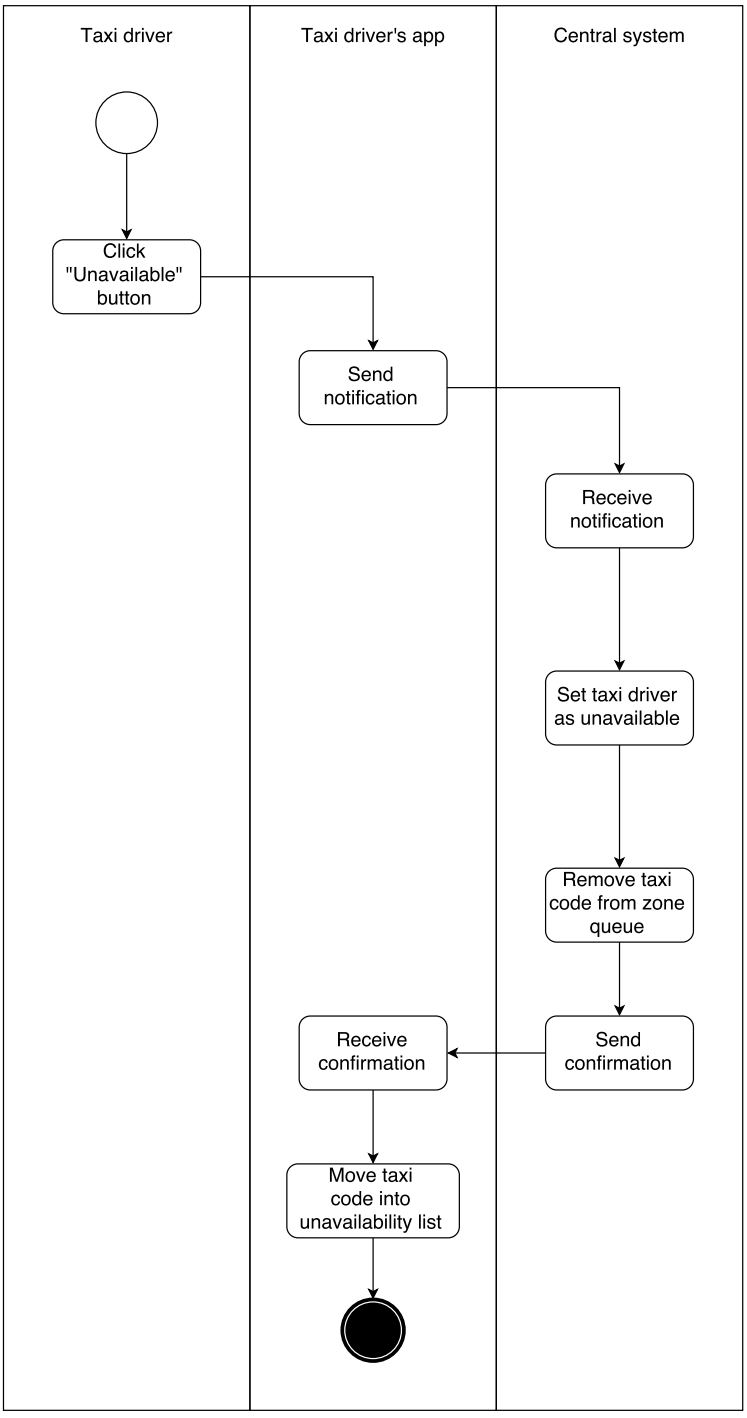
C.3.3 Reservation allocation



C.3.4 Confirm a ride has ended



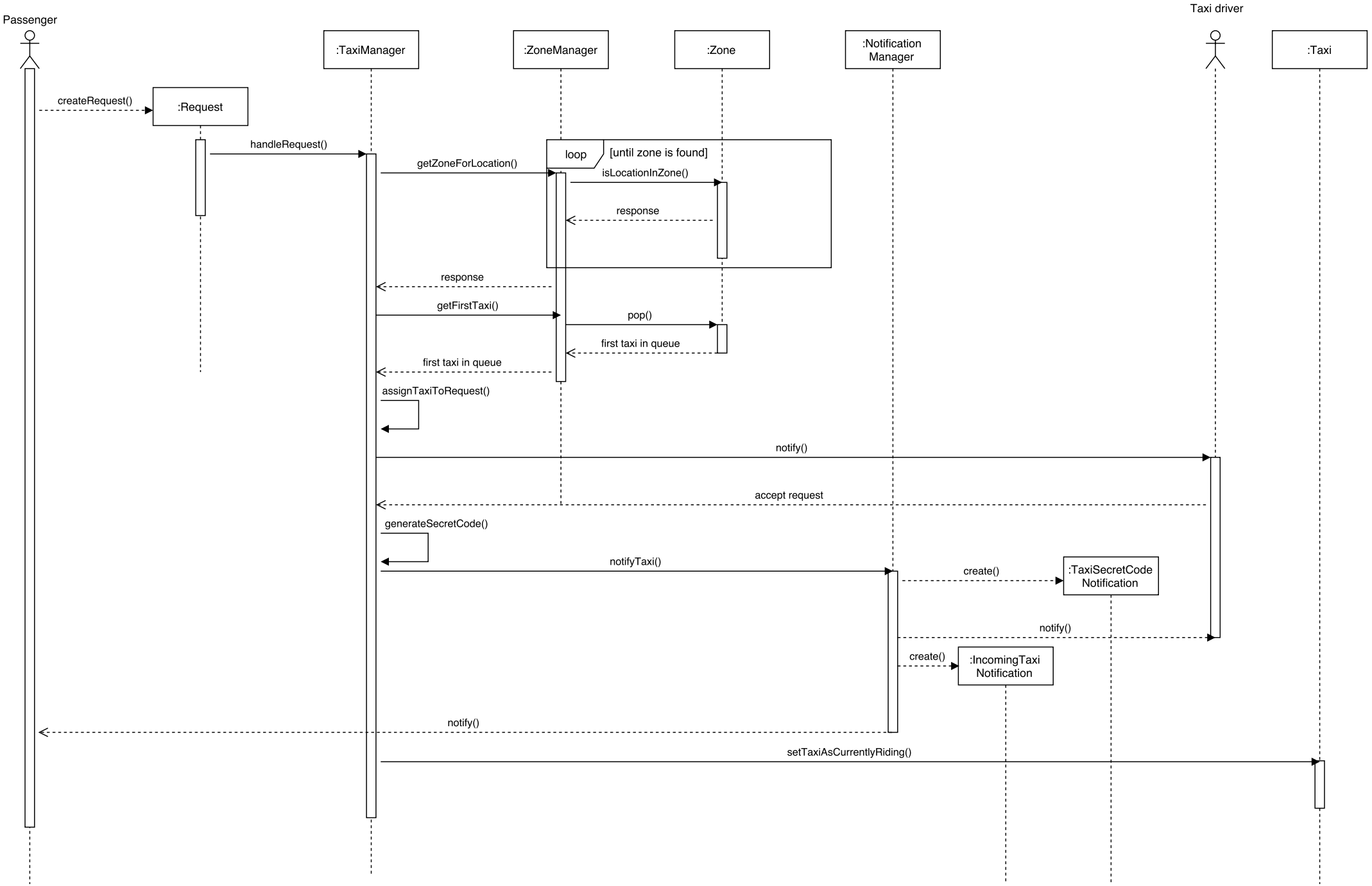
C.3.5 Set a drivers status to unavailable



C.4 Sequence diagram

Here we include a sequence diagram for the request a taxi operation.

For the sake of simplicity and readability, only the main flow of execution has been included in this diagram. Alternative flows of executions, such as those related to unexpected situations or exceptions, are not covered.


Passenger
:TaxiManager
:ZoneManager
:Zone
:Notification Manager
Taxi driver
:Taxi
createRequest()
:Request
handleRequest()
loop
[until zone is found]
getZoneForLocation()
isLocationInZone()
response
response
getFirstTaxi()
pop()
first taxi in queue
first taxi in queue
assignTaxiToRequest()
notify()
accept request
generateSecretCode()
notifyTaxi()
create()
:TaxiSecretCode Notification
notify()
create()
:IncomingTaxi Notification
notify()
setTaxiAsCurrentlyRiding()

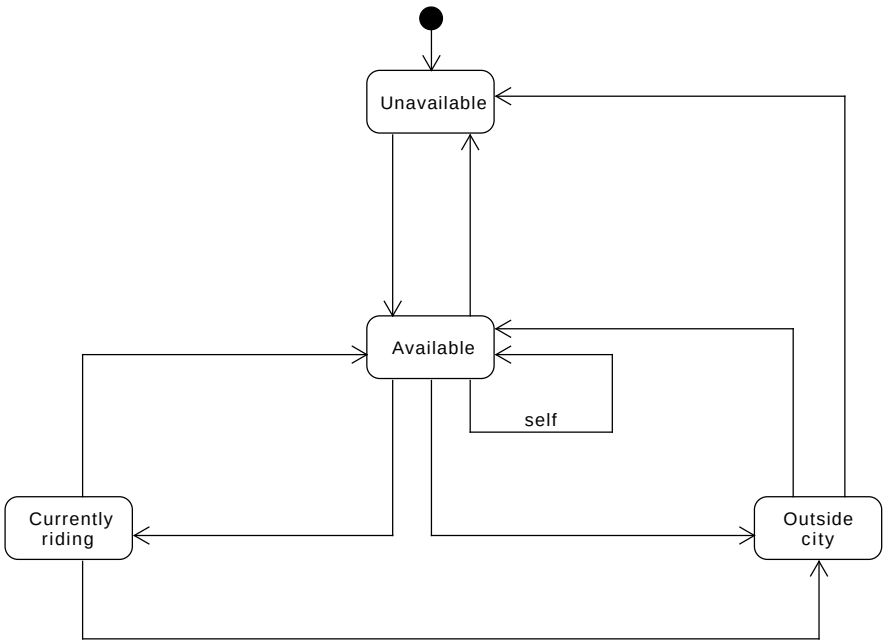
C.5 State chart diagram

This state chart models the possible states in which a taxi can be inside the system and the transitions between them.

It should be noted that the states the system recognizes are not mapped one-to-one to all the possible states a taxi can be in.

In particular, the taxi transition from the inside to the outside of the city and viceversa is not modeled if the taxi is performing a ride or if hes unavailable; that means that the status of the taxi will not change when crossing the border if the taxi is not in the available status.

This is not a limitation for the system: in fact, we dont need to know whether a taxi is inside or outside the city if hes performing a ride or if hes unavailable, because in either case he would not be able to receive requests anyway.



Appendix D

Work hours

To redact this document, we spent 40 hours per person.