Software Engineering 2 Project

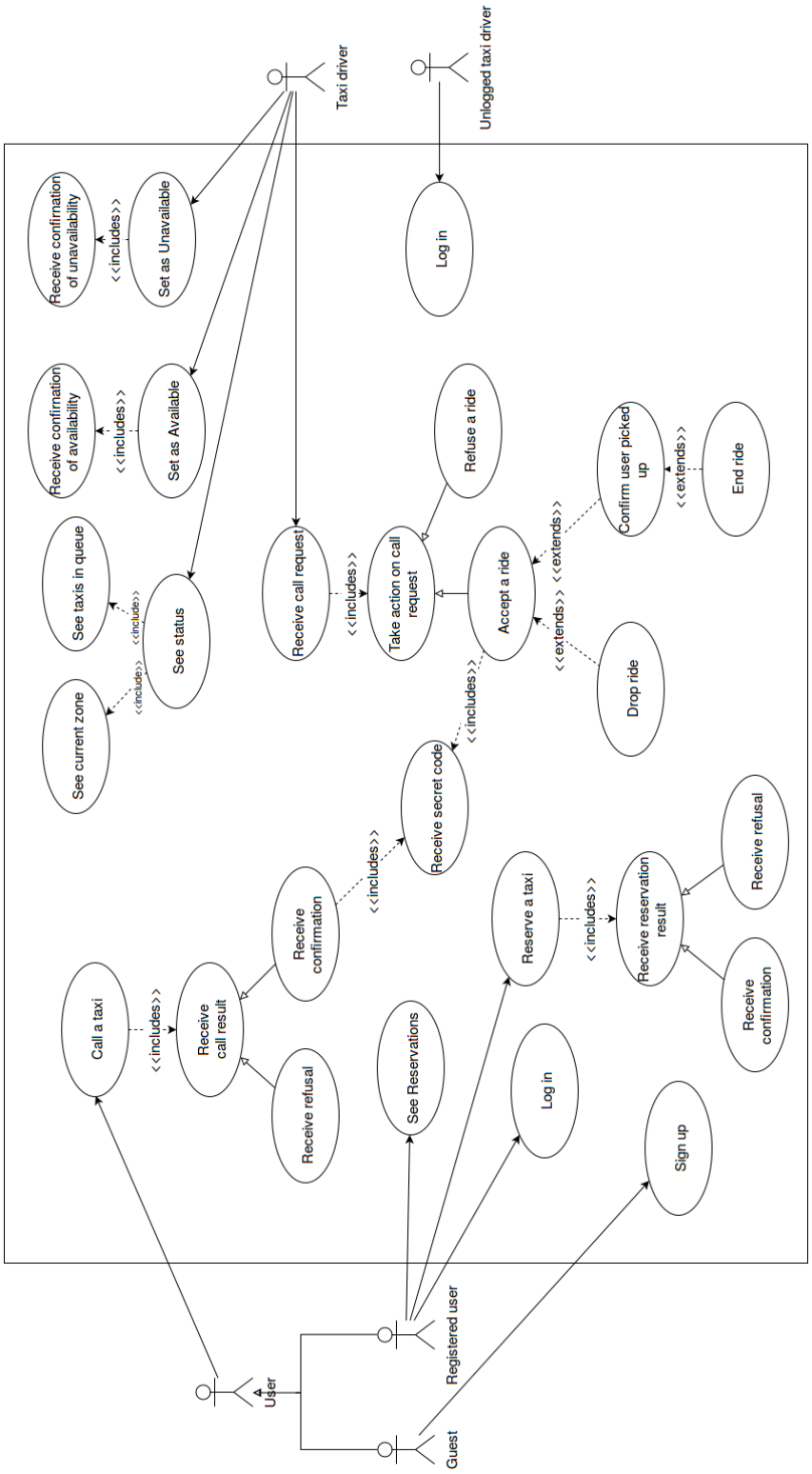
Casati Fabrizio, Castelli Valerio

November 5, 2015

Contents

Chapter 1

Introduction


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

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. Taxis should be able to accept or refuse a call. 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, 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 user 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 Definitions, acronyms, and abbreviations

1.3.1 Definition

Here we present a list of all terms used in the document.

- "system": the system that has to be designed and main argument of this document
- "user": a person that use the system to request a ride
- "code" or "taxi code": the unique identifier associated to each taxi
- "available": a taxi driver is considered available if he can answer to a user call.
- "unavailable taxi": if not
- "taxi zone": zone in which the city is divided in.
- "unavailability list", UL: the list where codes of the taxi drivers actually unavailable are stored;
- "zone queue", ZQ: the FIFO list, one for each zone, where the codes of available taxi driveres are stored;
- "out of city list", OCL: the list where the code of taxi drivers actually out of city are put in;
- "reservations register", RR: the data structure that stores all the reservation of users;

1.3.2 Acronyms

No acronym is used in this document.

1.3.3 Abbreviations

No acronym is be used in this document.

1.4 References

- Assignments 1 and 2 (RASD and DD).pdf, data???, on beep
- IEEE Standard

1.5 Overview

In the following part of this SRS document, we provide a general description of the system, with a particular focus on the assumtpions it is based on and the required functionalies (Section 2). Next, we go deeper in the specification of system functionalities, by providing a formal definitions of all system requirements and UML model (Section 3).

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 driver;
- 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 Hardware interfaces

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

2.1.4 Software interfaces

The system will have to communicate with an external mapping service in order to use the functionalities of GPS location decoding (note: must be specified what we mean) 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.

2.2 Product functions

2.3 User characteristics

Stakeholders: (non va qui per!!!) Users: men/women that want to require a taxi and that will use the system to accomplish this.

Taxi drivers: the men/women that drive the taxi; they are supposes to answers to users' request (by means of the system) and drive them to the desired location.

City government: it is the main stakolder of the project, since it is the one who committed it and that will pay for it.

Mobile phone producer: the company with whom the city government needs to manage an agreement to provide mobile phone to all the taxi drivers.

Taxi drivers union: as the system will have an impact on the way taxi drivers work, it's necessary to consider some possibile hostile action from their unions.

training and user support staff???

2.4 Constraints

2.5 Assumptions and dependencies

2.5.1 Domain properties

- Zones do not overlap.
- A taxi driver only drives a taxi.
- A taxi can be driven only by a single taxi driver.
- User can only request rides using either the web application or the mobile application provided by the system. - [OR] Users 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. [related req.: how can we transfer this data?]
- Uses don't have the need to choose a particular taxi driver among those that are available for their ride.
- Users 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-decode it into a valid city address.
- A taxi driver will take a passenger onboard if and only if both the taxi code and the security number sent to the passenger are equal to his own.

*** exceptional circumstance: strikes, severe weather conditions, service suspension decided by the city council, particular holidays [terminology]

- 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.

2.5.2 Assumptions

- Users 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 hes unavailable.
- Taxis that are considered to be out-of-city will not be able to receive calls. (redundant?)
- 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 user is not required to be registered in order to request a ride.
- A user 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 user that has already requested a taxi will not request another one before having completed his ride.

Chapter 3

Specific requirements

3.1 External Interface Requirements

3.2 Functional Requirements

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software System Attributes

3.5.1 Reliability

3.5.2 Availability

3.5.3 Security

3.5.4 Maintainability

3.5.5 Portability

3.6 Other Requirements

3.7 Goals and requirements

3.7.1 Goal 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, code/identifier, 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.

3.7.2 Goal 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.

3.7.3 Goal 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.

3.7.4 Goal 4

Goal 4: The taxi driver must be able to communicate his availability status.

Requirements:

- The 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 its 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 removed from the queue of the corresponding zone.
- When a taxi driver marks himself as unavailable, his internal status should be updated accordingly.
- When a taxi driver is set as unavailable, he should receive a notification warning him that hes no longer available to accept requests.

3.7.5 Goal 5

Goal 5: The allocation and distribution of taxis should be managed fairly and consistently.

Requirements:

- The system should consider all drivers to be initially (?) 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 hes 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 hes 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.
- If a taxi leaves the city while hes unavailable, the system should prevent him from signaling hes available until he is again in the perimeter of the city.
- The mobile application of each taxi driver should periodically send his GPS coordinates to the main system.

3.7.6 Goal 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-decode 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.

- 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 hes currently in.
- If the selected taxi driver does not answer within two minutes from the assignment, the system should consider the request refused.
- If its not possible to establish an internet connection to the central system, the mobile application should notify the taxi driver that its unable to operate correctly.

3.7.7 Goal 7

Goal 7: The taxi driver should be able to drop a request if the passenger doesnt show up.

Requirements:

- If a passenger who has requested a ride doesnt 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.

3.7.8 Goal 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 hes 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.
- When the system receives the notification that a driver has terminated a ride and the driver is inside the city, the system should assign him to the queue of the zone in which he is.

3.7.9 Goal 9

Goal 9: Users 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 users to request a ride. - The mobile application should be able to detect the current GPS location and use it as the source point of a ride request. - The mobile application should allow the user to enter an address as the source point of a ride request. - The web application should allow the user to enter an address as the source point of a ride request.
- If its 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 user that its unable to automatically locate his position.
- If its not possible to establish an internet connection to the central system, the mobile application should notify the user that its 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 user that the request could not be completed.
- When a user requests a ride, the system should use the source field of the request to compute the corresponding taxi zone.
- After a requests taxi zone has been computed, the system should assign the request to the first taxi in that zones queue.
- If a taxi driver accepts a request for a ride, the system should send a notification to the requesting user 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.

3.7.10 Goal 10

Goal 10: Users should be able to register.
Requirements:

- The system should provide a registration form. (including which data?)

3.7.11 Goal 11

Goal 11: Registered users should be able to reserve rides in advance.
Requirements:

- The system should keep an association between a user profile and his reservations.
- The system should provide a way for registered users to reserve a ride for a specific date and time. - The mobile application should allow the user to enter the source and destination addresses for the ride and the meeting time. - The web application should allow the user to enter the source and destination addresses for the ride and the meeting time.
- If its not possible to establish an internet connection to the central system, the mobile application should notify the user that its 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 user that the request could not be completed.
- The system should provide a way for registered users to review their active reservations.
- The system should provide a way for registered users to delete a reservation.
- The system should store reservations in a reservation queue.
- 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 arent 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 cant 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 5 minutes for at most 10 times.
- If a reservation cant be fulfilled after 10 reschedules, the system should notify the passenger that his reservation will be dropped.

3.7.12 Goal 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 sent to the passenger who requested the ride (?).

3.8 Scenarios

SOME POSSIBLE SCENARIOS (with a template of event flow for the sequence diagram)

3.8.1 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 Bobs taxi.

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

The system receives Bobs 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 Bobs 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 Bobs 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.

3.8.2 Scenario 2

Jessica and Davide are planning to celebrate their anniversary, which is happening in a week, by going out and have 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 (hes 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 Davides laptop.

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

3.8.3 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 Davides apartment and realizes he lives in zone 4. It then forwards the request to the the first taxi in the queue associated with zone 4, which happens to be Matteos taxi.

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

The system receives Matteos confirmation along with his GPS coordinates and moves his taxi code 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 Matteos 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 her mobile phone and patiently waits for the taxi to arrive.

When Matteos 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.

3.8.4 Scenario 4

Matteo has just drove 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 code into the last position of the respective queue.

3.8.5 Scenario 5

Bob has just drove 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 code 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 Bobs decision and proceeds to move his taxi code from the queue of his current zone to the unavailability queue. It then sends back a notification to Bob informing him that hes 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.

3.8.6 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 Samuels 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 moves his taxi code into the out-of-city queue.

3.8.7 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 Franks current location is now inside the boundaries of the city and computes the associated zone.

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

3.8.8 ???

A user requests a taxi: - A user access the app (mobile or web?) - The user request a taxi - The system receive the request - The system, by interaction with the mapping serve, compute in which zone the user is - The system checks if in this zone there is a taxi available; let's suppose there is one (or do we have to consider also the other case?) - The system sends to the taxi driver a notification (with which info???) - The taxi driver accepts - The system receive the confirmation - The system remove the taxi driver from the queue and put it on the UL - The system sends a notification to the user with taxi code and waiting time - The user receive the confirm

A user books/reserves a ride: - The user open the web app - The user log in - The user open the form to reserve a taxi - The user insert all the requested info - The user confirm - The system receive the request - The system store the request (???)

A taxi driver has ended a ride and wants to communicate his availability: - The taxi driver sends a notification of "ride endend" - The system receive it and delete the request of the user (?) - The taxi driver sends a message

of "available" - The system receive it - The system compute the taxi driver position (by the mean of the mapping service ?) - The system put the taxi code in the zone queue

A user withdraw / delete a book / ... (one of this will be enough ???)

A taxi driver withdraw / ... (one of this will be enough ???)

A taxi driver enter in city

A taxi driver changes zone / leaves the city

10 minutes before a meeting time, the system associates a taxi to the reservation

Appendixes

Alloy code.

```
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,
    receiver: User
}{
    notificationId ≥ 0
}

sig IncomingTaxiNotification extends Notification
   {
    ETA: Int,
    taxiCode: Int,
    secretCode: Int,
    relativeTo: Request
}{
    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
}

sig NoTaxiAvailableNotification extends
   Notification{
}

sig TaxiSecretCodeNotification extends
   Notification{
    secretCode: Int
```

```
}{
    secretCode ≥ 0
}

/* Facts */

fact RequestShouldBeEitherServedOrPending{
    no r:Request, tm:TaxiManager • (r in tm.
       pendingRequests) ∧ (r in tm.servedRequests
       )
    ∀ 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)
    ∀ 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 */
    ∀ 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 ReservationConfirmationNotificationBehavior{
    /* Incoming taxi notifications are always
       sent exactly to the user that
    has made the request */
    ∀ n: ReservationConfirmationNotification •
       one r:Reservation • (n.relativeTo = r) <=>
       r.madeBy = n.receiver
```

```
}

fact AllNotificationInNotificationManager{
    // All notifications are sent by the
       notification manager
    ∀ n: Notification • one nm:
       NotificationManager • n in nm.sends
}

fact AllZonesAreManagedByZoneManager{
    // All zones are actu∀y managed
    ∀ z: Zone • one zm: ZoneManager • z in zm.
       manages
}


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

fact TaxiStatusCoherence{
    // The taxi status should be coherent wrt the
        TaxiManager
    ∀ t:Taxi • t.taxiStatus = available ≤> (one
       tm: TaxiManager • t in tm.availableTaxis)
    ∀ t:Taxi • t.taxiStatus = unavailable ≤> (one
        tm: TaxiManager • t in tm.
       unavailableTaxis)
    ∀ t:Taxi • t.taxiStatus = currentlyRiding ≤>
       (one tm: TaxiManager • t in tm.
       currentlyRidingTaxis)
    ∀ 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
    ∀ 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
    ∀ 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
    ∀ t:Taxi• t.taxiStatus = outsideCity implies
       (t.serves = none)
    // If a taxis is unavailable, it cannot be
       associated with requests
    ∀ t:Taxi• t.taxiStatus = unavailable implies
       (t.serves = none)

}

fact NoLocationInTwoZones{
    // No locations should simultaneously belong
       to two different zones
    ∀ 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
    ∀ 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 = 1
    #Notification = 0
    #User = 2
}

run addTaxiToZoneQueue for 2 but 10 Location
```

Index

text