POLITECNICO MILANO 1863

myTaxiService

REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT

Paolo Antonini 858242
Andrea Corneo 849793

version 1.3 – 5th February 2016

Contents

1
Introduction

1.1 Purpose

Improving the public transport is crucial for a modernizing city and Milan, as such, has already taken some steps in this direction. However, until now the taxi service has been overlooked, if not neglected. Recently, though, the government of the city requested a comprehensive study about possible ways to improve its taxi service. This document will outline a computer system which will hopefully help the city council to achieve this goal.

As a result, the natural addressee of this document is the municipal government itself, which will evaluate the feasibility of the project we are proposing. Once the project is approved, this document, along with other ones that will follow, will constitute a solid, non-technical guide to developers' work.

1.2 Scope

To be more specific, the council wants the access to the service to be simplified, as well as to guarantee a fair management of taxi queues, which will benefit both passengers and taxi drivers.

These targets can be reached by implementing myTaxiService, an inclusive computer system which lets users make an immediate or delayed reservation for taxi rides, and manages the requests in order to minimise the waiting time for all passengers. Users will be able to access the new service through both a mobile application and a website; nevertheless, the current, phone-based system continues to be operative, in order to meet the needs of those people who are unfamiliar with the new service, and also of those who are not able to access it.

Moreover, taxi drivers will be supported in their activity with a mobile application as well, which will run on their smartphone, letting them receive the requests for taxi rides.

1.3 Definitions, acronyms, and abbreviations

In order to avoid ambiguity, we find it appropriate to explicitly define some terms which will recur throughout the document:

Customer the person who makes use of the reservation service through one of the available methods (mobile application, website, phone call); customers can register on the service, thus gaining access to some convenient additional functions; otherwise they are considered as guests; they may be referred to as users as well; typically, but not necessarily, the person who made the reservation is also the person who actually benefits from the ride.

Mobile application a program designed to work on mobile devices, namely smartphones and tablets; in the specific context of this document, we have two or them, since both the customers and the taxi drivers need one.

Website in the specific context of this document, the website is a particular web page on the Internet, reachable with a browser (e.g. Internet Explorer, Google Chrome, Mozilla Firefox, ...), by means of which a user can access the service.

System in the specific context of this document, the system is the actual, computer-based provider of the service; notice that the two mobile applications, the website and phone calls are ways to access it.

Taxi queue list of taxis assigned to a specific area of the city; the policy that regulates the queue is a "First In, First Out" one, which means that the first taxi that is enlisted in the queue, is the first who receives a reservation, as well; there may be some exceptions, which will be described farther in this document.

Area in order for the system to guarantee a fair management of the taxi queues, the city is divided in portions of approximately $2\,km^2$, whose borders are well-defined by road intersections.

Request / Reservation through the system, the customer can reserve a taxi ride; actually, his reservation may be either immediate (and we will call it request), meaning that a taxi cab is immediately allocated and will reach him as soon as possible, or delayed, which is to say that he books a ride by specifying its origin, destination and time.

1.4 References

Farther in this writing we will refer to some external documents, which are listed below:

- *"Assignments 1 and 2"*, section 2, parts I and II; 13th October 2015. This is to be regarded as the high level description of the problem. Available at: `https://beep.metid.polimi.it/documents/3343933/d5865f65-6d37-484e-b0fa-04fcfe42216d`.
- *"Legge quadro per il trasporto di persone mediante autoservizi pubblici non di linea"*, law number 21/1992; 15th January 1992. This is the

current law that regulates the taxi service in Italy. Available (only in Italian) at: `http://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:legge:1992-01-15;21`.

- *"Codice in materia di protezione dei dati personali"*, law number 196/2003; 30th June 2003. This is the current Italian law that regulates privacy issues. Available (only in Italian) at: `http://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:decreto.legislativo:2003-06-30;196`.

1.5 Overview of the document

This document develops as follows. In chapter 2 we give a general description of the factors that affect both myTaxiService system and its requirements. In particular, this section focuses on the functions which are provided by the system, the constraints we have to impose, the assumptions we make. Requirements are thoroughly analysed in chapter 3, which is the most technical one. Formal and informal graphical representations are given as a support. Eventually, appendix A contains the modelling in Alloy code.

2
Overall description

2.1 Product perspective

Before analysing in detail all the aspects of the system we are proposing, we find it appropriate to state its structure, and give mnemonic names to the different components.

The central system, on which the other components rely, is called *myTaxiService*; this is to be accessed by the office of the municipality in charge of the service, who will administer it (for example, by adding and removing taxi drivers). Customers can use a mobile application, *myTaxiApp*, or the website, *myTaxiWeb*, to exploit the system. As of the taxi drivers, their activity is supported by another mobile application, named *myTaxiAssist*.

When it is clear from the context, however, we will refer to the whole ecosystem as myTaxiService too, by extension of the name of the main system.

2.1.1 System interfaces

Here we state the system the system interfaces, that are, basically, all the compatibility constraints we have to impose.

myTaxiApp this application is intended to run on smartphones and tablets; to reach the widest portion of population, we need the application to be available for Apple's iOS, Android and Windows Phone devices. A connection to the Internet is needed.

myTaxiWeb given that a browser is always present in any operating system, the only requirement is that a connection to the Internet is available.

myTaxiAssist we need this application to integrate with the other technological devices which are already installed in the taxi cab, and above all with its GPS sensor; with this in mind, we decide to provide every taxi driver with an Android smartphone, which is the best platform to provide integrated services; as a result, we need this application to be compatible with only Android operating system. Like before, an Internet connection is needed.

myTaxiService we believe that a good trade-off between costs and performances is the installation of a server farm based on cloning[1]. Obviously, for the whole ecosystem to work correctly, an Internet connection is needed.

[1] Technically speaking, we are suggesting a Reliable Array of Cloned Services, or RACS. This allows both a load balancing (especially needed in peak times), and also a great masking in case of technical failures.

2.1.2 User interfaces

In this subsection we specify the logical characteristics of each interface between the software product and its users.

myTaxiApp since the application is designated for mobile handsets, limited screen size and resolution are design constraints to take into account. Moreover, navigability, effectiveness and ease of use are fundamental, so every function shall be fulfilled by the user within no more than two screens.

myTaxiWeb here we have less constraints on resolution than in myTaxiApp, since the website is to be accessible only via personal computers. However, we need it to be lightweight, so that on a 7 Mbit/s connection (which we assume to be available everywhere in Milan) the page can fully load in no more than 2 seconds. Moreover, the requirements regarding the ease of use still hold, so also here every function shall be fulfilled within no more than two screens.

myTaxiAssist this application is specifically designated for smartphones, so some resolution constraints hold. Furthermore, taxi drivers need to be extremely rapid while using it: after two hours of training, every taxi driver can exploit any function of the application in less than a minute. This is also possible thanks to the use of big (no more than four in a screen), self-explanatory icons. Again, every function shall be fulfilled within no more than two screens.

myTaxiService as was stated before, some employees of the municipality shall be able to perform some administrative functions on the system, such as the insertion of new taxi drivers; this can be done through a simple webpage. As a consequence, no resolution constraint holds.

2.1.3 Hardware interfaces

Location services are of central importance for the operation of the system. However, the GPS coordinates are provided to the applications or the website by the operating system, whenever possible. In particular:

myTaxiApp the application receives them from the APIs of the specific operating system[2].

myTaxiWeb the website gets them through the localisation services provided by every reasonably modern web browser.

[2] An application programming interface (API) is a set of functions, protocols, and tools for building software applications.

myTaxiAssist the application obtains them from the GPS sensor placed in the taxi cab.

Nevertheless, since we cannot assume that a GPS sensor (or other technological solutions) are available and reliable in customers' devices, the possibility to manually specify the address is given. On the contrary, it is reasonable to take for granted the accuracy of the GPS device on board of taxi cabs.

Similarly, it is the operating system which provides an Internet connection both to the applications and to the web browser, so there is no need of direct integration with hardware components.

2.1.4 Software interfaces

Due to the great variety of operating systems and browsers, each of which bearing a potentially different configuration, the website myTaxiWeb shall not require either Java or Adobe Flash plugins; it will be developed using HTML 5 technology, so a suitable web browser is needed (as of Microsoft Internet Explorer, al least version 7 is needed; on the other browsers, compatibility is taken for granted).

The system itself has to communicate with a database, which contains both the personal details of the taxi drivers, along with the information about taxi cabs, and the record of all the reservations (pending, accomplished, cancelled). A relational database is the most suitable solution.

On the other hand, the two applications, from this point of view, are rather self-contained, so no particular software interface is needed.

2.1.5 Communications interfaces

The myTaxiService server is HTTP-based and it contains also an SMTP component to send emails. Applications, obviously, send to and receive data from the system over the same HTTP protocol.

Bluetooth is required by myTaxiAssist application, since this is the integration technology between taxi drivers' handsets and the devices on board of their taxi cabs. If the connection is lost, arguably because the taxi driver has left the cab, the system is notified, and the driver is considered "off duty" until the reconnection.

2.1.6 Memory constraints

Since this system is not aimed to data analysis, we have no need of considerable memory availability. As a result, this is not to be considered a major constraint.

2.1.7 Operations

The data processing features of the system are limited, since they are not needed. Every day at 3AM, however, backup operations are

performed.

2.2 Product functions

Before moving on with our analysis, it may be useful to provide a brief summary of the major functions that the software will perform.

Passengers can request a taxi either through a website (myTaxiWeb) or a mobile application (myTaxiApp). As soon as the request is received by the system (myTaxiService), it is immediately forwarded to the nearest waiting taxi driver. If he accepts, the system answers to the request by providing the passenger with the code of the incoming taxi and the waiting time. If the request is rejected, or ignored, the system looks for another taxi driver. In particular conditions, a user can also reserve a taxi for a later ride, by specifying its time, origin and destination.

Taxi drivers use a mobile application (myTaxiAssist) to inform the system about their availability and to confirm that they are going to take care of a certain call.

2.3 User characteristics

There are three types of users that interact with the system: customers, taxi drivers and administrators.

Since no educational level can be assumed for the users of myTaxiApp and myTaxiWeb, we need both to be as easy to use as possible, which means that every screen shall have a precise help page. Obviously a very basic web and technological knowledge is needed to benefit from the service.

The taxi drivers will follow a mandatory two-hour course, to gain the necessary expertise to use the application myTaxiAssist.

The administrators' functionalities are limited, but they also need a two-hour course to gain the necessary knowledge.

2.4 Constraints

The current Italian law that regulates the taxi service (see section 1.4 on page 6 for a link to the document) allows the implementation of all the improvements to the service that we are proposing.

However, since we process and store personal data such as names, phone numbers, and localisation data, we are subject to the Italian law concerning the protection of personal data (again, see section 1.4).

2.5 Assumptions and dependencies

We have to make some assumptions, since the high level description of the project given by the city council (see section 1.4 for a link to the document) is ambiguous and incomplete:

- the number and distribution of taxi cabs in service are always sufficient to serve 98 % requests in 10 minutes and 100 % of them within 30 minutes;
- the reservation for a delayed ride can be cancelled before its confirmation (that is, before the 10 minutes limit);
- to cross an area, a taxi needs at most 10 minutes;
- if the user reserves a taxi, he means to take it;
- a taxi driver who has accepted a request, cannot refuse to let the client in;
- the origin of the ride must be within the borders of the Metropolitan City of Milan;
- we have only taxi cabs for four people (plus the driver); if the group is more numerous, then a proper number of cabs is allocated (one every four people);
- a pending request is automatically forwarded to the next taxi driver after one minute;

2.6 Apportioning of requirements

The system, as it is outlined in this document, can be expanded in many ways. The city council explicitly mentions a taxi sharing functionality, which means that a user can share a taxi ride with others if possible, thus splitting the cost of the ride too.

We also suggest that the system, through myTaxiAssist application, can record payments. A relatively easy integration may be with PayPal payment system, which allows fast and seamless online money transactions without disclosing bank account details. PayPal registered users can be charged directly and can pay by using their own handset.

3
Specific requirements

3.1 Functions

To specify the functional requirements, that are all the fundamental actions that must take place in the we system we are proposing, along with the textual descriptions we provide some scenarios, sequence diagrams and use case diagrams, because we believe that they can be a valid support to our words.

3.1.1 Taxi request

Scenario 1 *Alexei, Anna's husband, is out of town for work. Thus, she decides to go and visit her lover, Count Vronsky. She recently came to know that a new taxi service was active in the city, so after a brief lookup in the Internet she finds the website, myTaxiWeb.it. She finds out that there is an app as well, but she is too eager to meet her lover to download it, so she quickly fills in the form with her data (name and surname, mobile phone number, and her origin and destination addresses) to request a taxi ride. Her request is sent by the system to the nearest taxi driver, who is named Kostya and drives the taxi number T1878, and he accepts without hesitation by clicking the proper button on his company smartphone. In a few seconds Anna receives an SMS confirmation that taxi T1878 is due in four minutes. Then she puts some lipstick, grabs her handbag and calmly leaves. Kostya is already waiting for her in the street, and in twenty minutes or so she is hugging her beloved. As soon as she gets off his taxi, Kostia confirms that he is available again to the system, by clicking the proper icon on his smartphone.*

Scenario 2 *Jay is quite a wealthy man, and as it happens, he has not always lived a perfect life. Today, like every other month, he has an appointment with a man of his past, some Meyer, in a rough area of the town. So he opens myTaxiApp on his smartphone, to which he registered recently, and looks for his destination in the small set of saved addresses. Once he found it, he lets the GPS locate him, then confirms the request for a taxi ride. myTaxiService system looks for a driver in Jay's area, but there is none; fortunately, in an adjacent one there is George, on Taxi T1925, who has been idle for some time. However, as George sees the destination of the request, he immediately understands that he had better not go there, so he rejects the request. myTaxiService registers his refusal, but since the nearest taxi is about half an hour far from Jay, it asks him whether he wants to withdraw his reservation, which he does. He cannot be late at his appointment.*

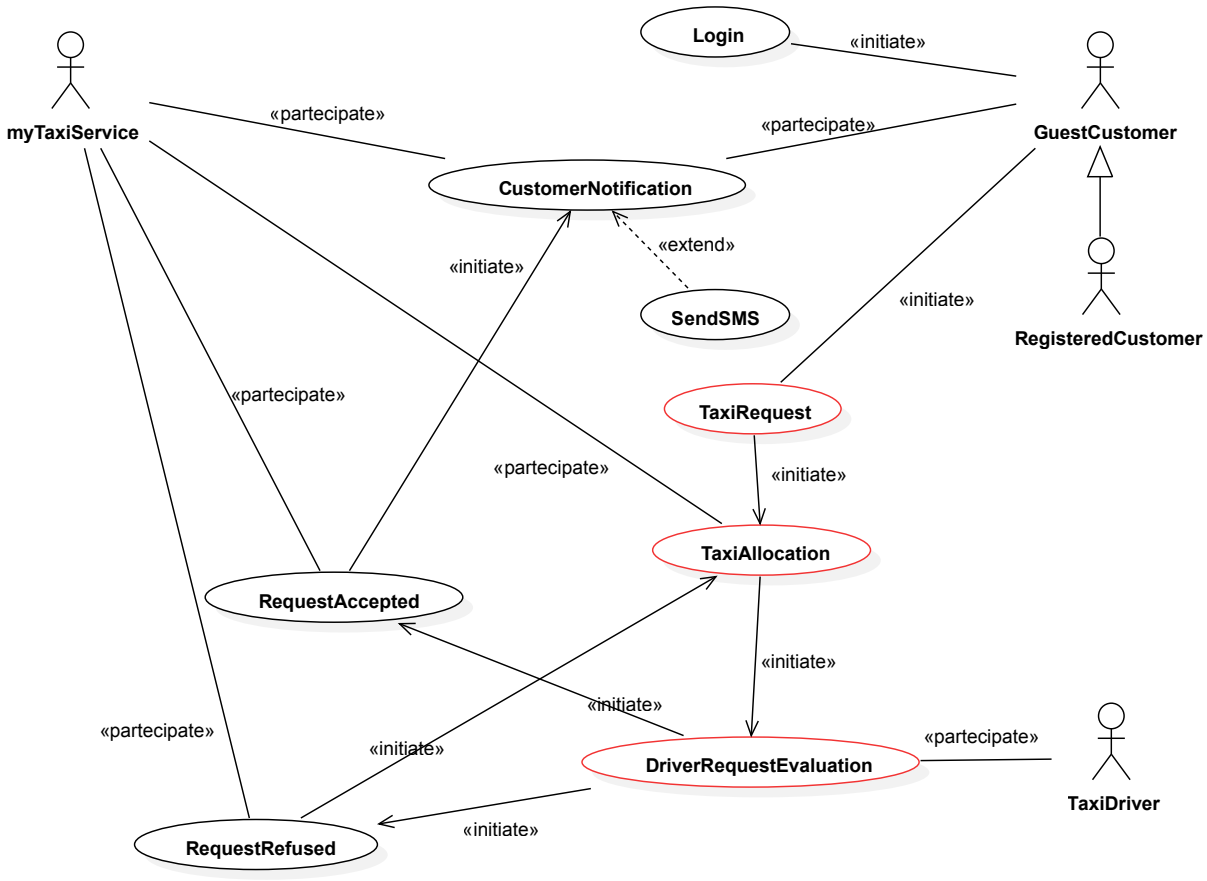


Figure 3.1: Comprehensive use case diagram for the taxi request. It will be further expanded in the following with more detailed diagrams.

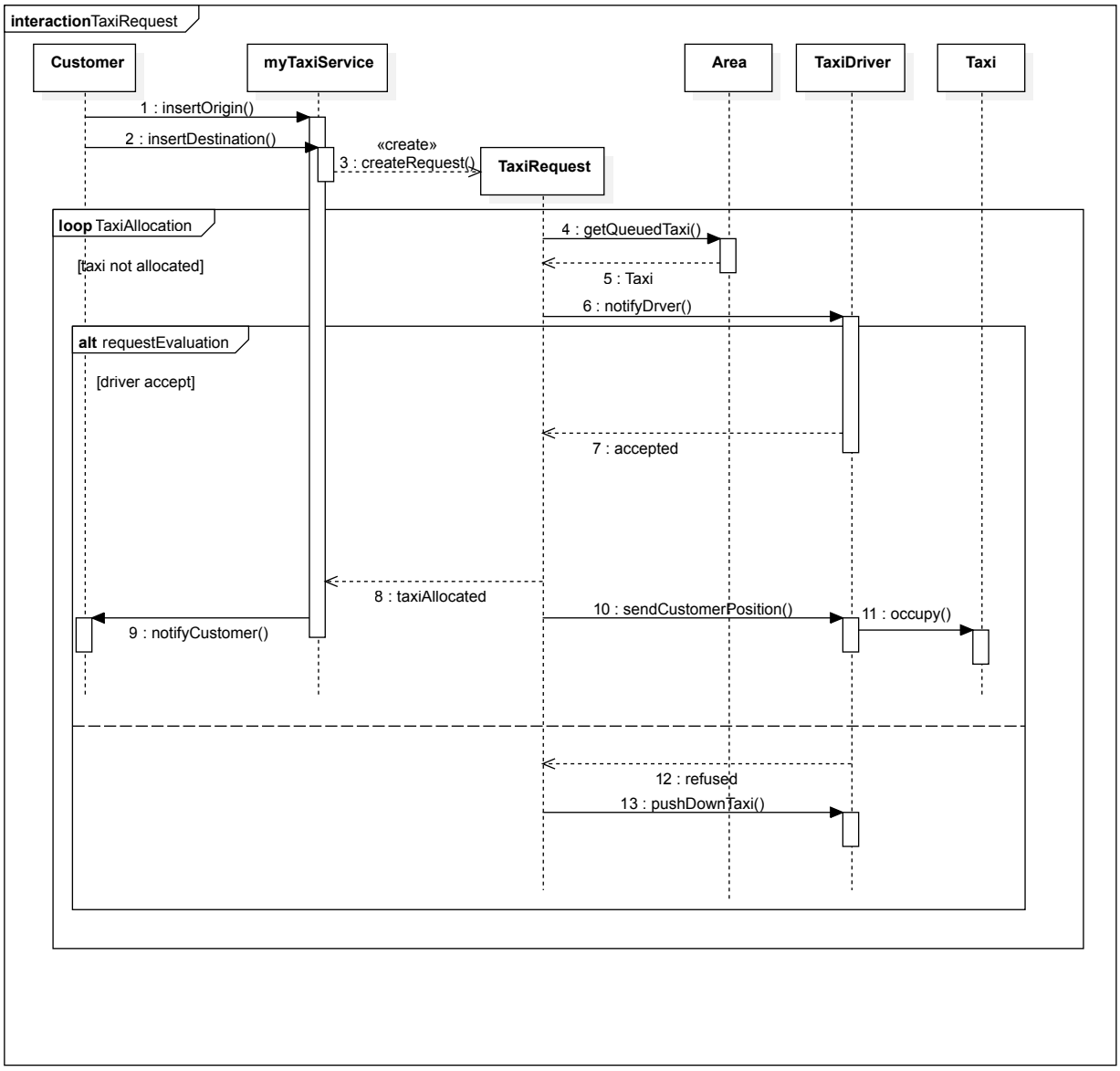


Figure 3.2: Sequence diagram for the taxi request analysis.

name	TaxiRequest
actors	GuestCustomer, RegisteredCustomer
entry conditions	a customer opens myTaxiApp/myTaxiWeb to request a taxi
flow of events	1. RegisteredCustomer can log in, GuestCustomer can sign in; 2. GuestCustomer inserts his personal data, RegisteredCustomer views his personal data; 3. GuestCustomer sends his GPS position or inserts the address, RegisteredCustomer can also select an address from the set of saved ones; 4. Customer sends the request.
exit conditions	all provided data are correct
exceptions	• invalid address: an error message is shown and a correct address is requested

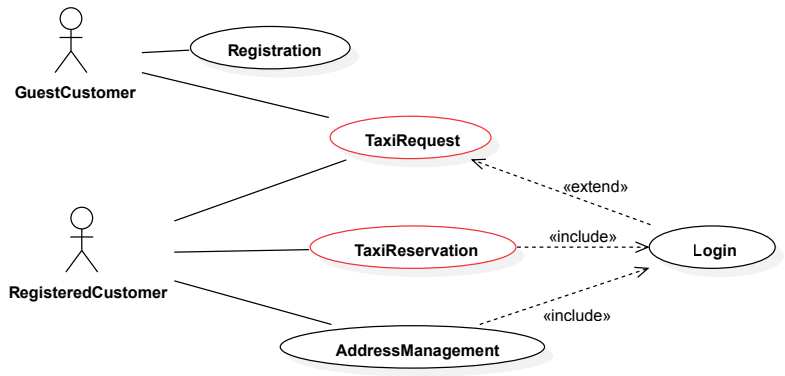


Figure 3.3: The customer can be either registered or guest. A guest can register and request a taxi (see figure 3.4). A registered customer can make a reservation (see figure 3.11) and manage his favourite addresses.

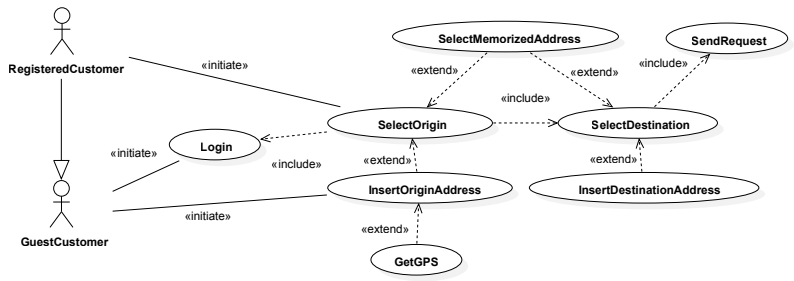


Figure 3.4: Detailed use case for the taxi ride request.

name	TaxiAllocation
actors	myTaxiService
entry conditions	request received
flow of events	1. the system determines the pertinence area; 2. a request is sent to the first taxi driver in the taxi queue; 3. if accepted, the address is sent to the driver, and the customer is notified with the time of arrival; otherwise the system proceeds to allocate the next taxi in the queue.
exit conditions	driver has accepted the request
exceptions	• if no taxis are available in the area, the system would recursively allocate the first taxi in the longest queue of the adjacent areas; • if no taxis are available at all, the system notifies the customer of the situation and sends an estimated waiting time. The customer can cancel the request.

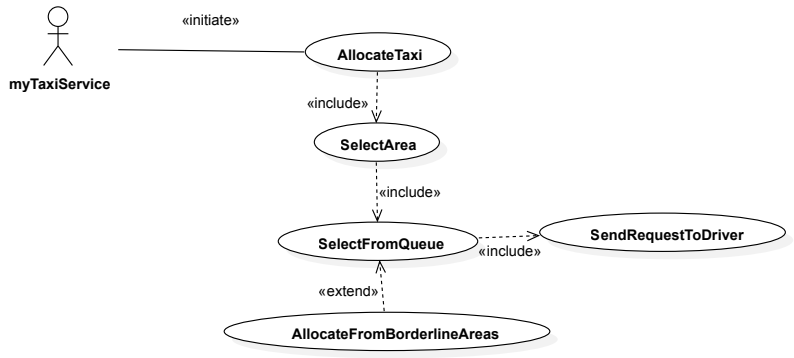


Figure 3.5: Taxi allocation, from the point of view of myTaxiService.

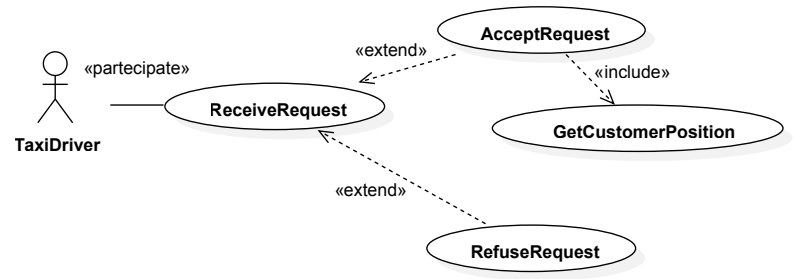


Figure 3.6: Taxi allocation, from the point of view of the taxi driver: he can either accept or reject the request.

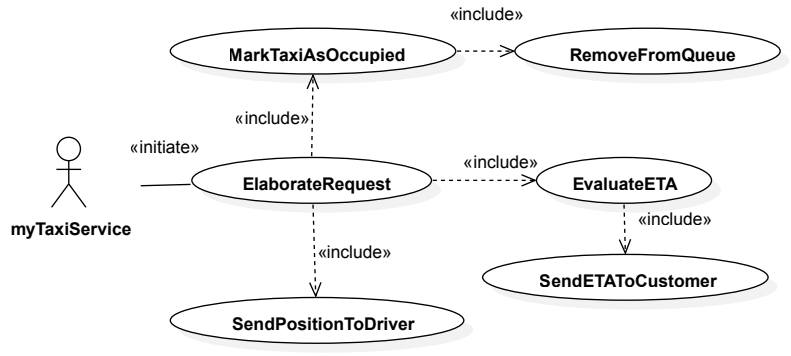


Figure 3.7: Request accepted use case, initiated by the acceptance of the request by the taxi driver.

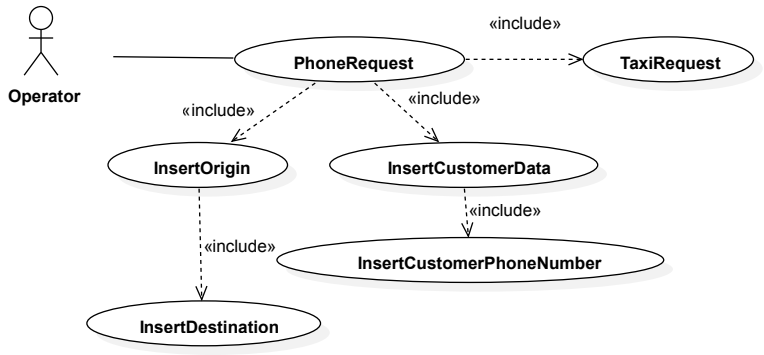


Figure 3.8: A customer may make a phone request. This is the relating use case diagram.

3.1.2 *Taxi reservation*

Scenario *Cookie has been in prison for a year, now. Her husband Lucious, notwithstanding the great weight of running a newly born entertainment company, growing three children all alone, and some sense of guilt for not being able to help her, wants their children to visit her. So tomorrow afternoon he is going to the jail by taxi with them. This morning, at about 9:30 AM, he tried to reserve a taxi, to guarantee a ride for themselves. However, his reservation was rejected, because he was making it more than* 24 *hours before the ride. But now, at 5:45PM, he tries again, and he is successful: his reservation is registered, and Lucious is notified that* 10 *minutes before the ride, he will receive a confirmation, with the number of the taxi too.*

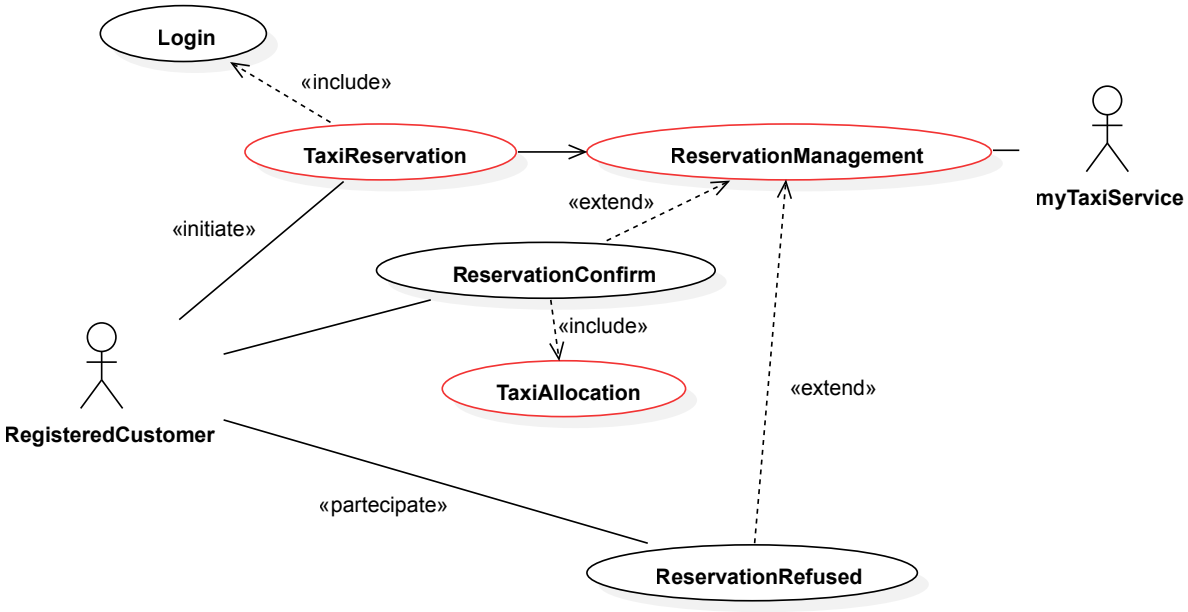


Figure 3.9: Comprehensive use case diagram for the taxi reservation. It will be further expanded in the following with more detailed diagrams.

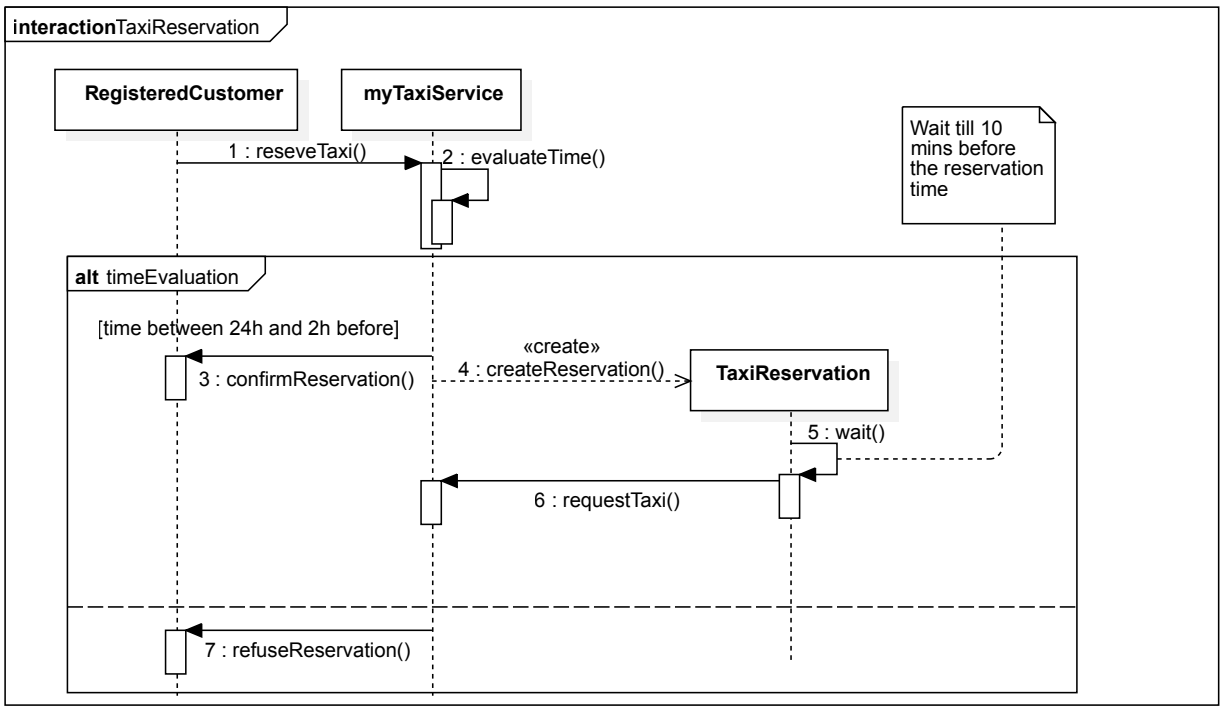


Figure 3.10: Sequence diagram for the taxi reservation analysis.

name	TaxiReservation
actors	RegisteredCustomer
entry conditions	RegisteredCustomer has successfully logged in
flow of events	1. RegisteredCustomer inserts the origin address, select it from the favourite list or send his GPS position; 2. RegisteredCustomer insert the destination address or select it from the favourite list; 3. RegisteredCustomer selects the time; 4. The reservation is sent
exit conditions	no exit condition
exceptions	• invalid address: an error message is shown and a correct addressed is requested

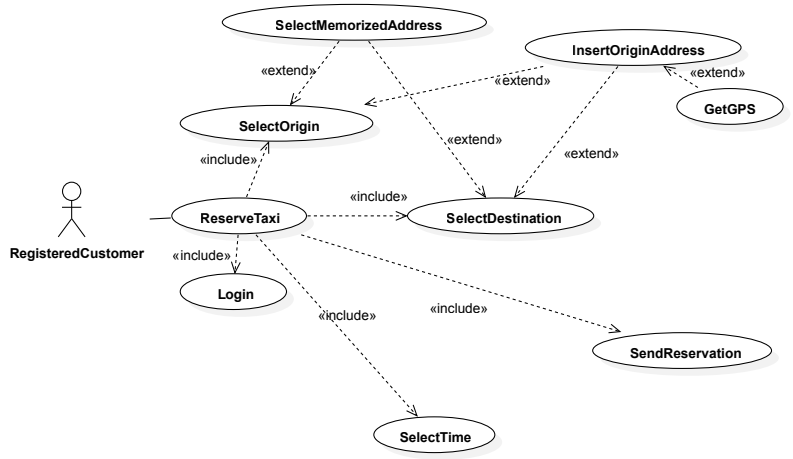
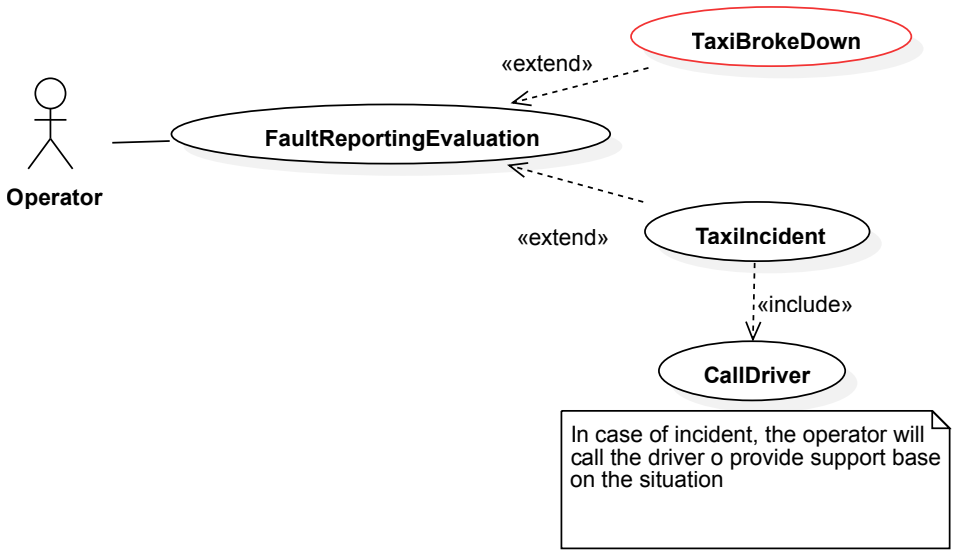


Figure 3.11: Detailed use case for the taxi ride reservation, from the point of view of the customer.

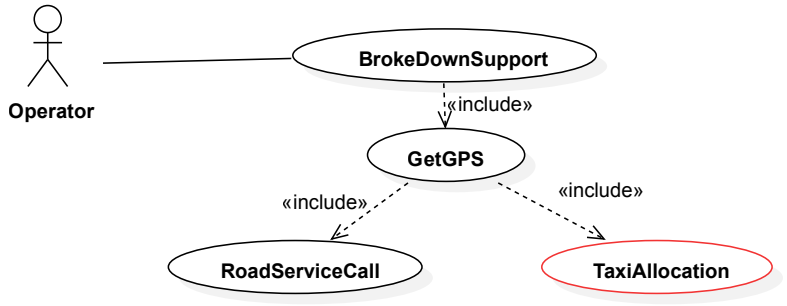
name	ReservationManagement
actors	myTaxiService
entry conditions	reservation received
flow of events	1. the system evaluates the time: if it is more than 24 hours or less than 2 hours before it rejects the reservation; 2. the system registers the reservation; 3. 10 minutes before the due time, a taxi is allocated as in TaxiAllocation use case (see section 3.1.1).
exit conditions	taxi successfully allocated
exceptions	• exceptions as in TaxiAllocation use case; • if the reservation is out of the time limits, a notification is sent to the customer;

3.1.3 *Fault report*

Scenario *While on duty, a taxi breaks down. The taxi driver reports the fault via the app and an operator provides assistance to him and allocate a new taxi.*



name	TaxiBrokeDown
actors	Operator
entry conditions	TaxiDriver reported a fault
flow of events	1. the operator gets the GPS position of the taxi; 2. if a customer is on board, the operator provides a new taxi, like in TaxiAllocation use case; if the taxi breaks down while reaching customer position, a new taxi is allocated directly to the customer; 3. the operator calls the breakdown service.
exit conditions	another taxi is allocated
exceptions	• no exception apply



3.1.4 *Taxi available*

Scenario *Whenever a taxi driver marks himself as "available" or, if already available, changes area, the system detects his new position by the on board GPS and queues him at the bottom of the queue in that specific area.*

3.2 *Functional requirements*

It is also useful to sum up all the functional requirements. It is important to point out that they all come as a consequence of what was stated in the previous section.

1. The service shall accept only valid input:
   (a) The service shall assure that each provided address exists.
   (b) The service can accept only requests with full personal data of the customer (name, surname and phone number).
2. The service shall guarantee the non-ubiquity of the actors:
   (a) A taxi driver must be registered once.
   (b) A registered customer can be registered only once.

(c) A taxi shall be registered only once.

(d) A taxi must have one and only one taxi driver, a taxi driver must have one and only one taxi.

3. The system has to ensure the security in the login process:

   (a) Only registered customer can login in myTaxiApp.

   (b) Only verified taxi driver can login in myTaxiAssist.

4. The service shall provide a method to verify the driver:

   (a) Only authorised personnel can add or remove a taxi driver and/or a taxi.

5. The service shall accept only valid reservations:

   (a) The service shall allow only registered customers to reserve a taxi.

   (b) The system shall accept reservations only if they are made between 24 hours and 2 hours before the request time.

6. The service shall guarantee a correct taxi allocation:

   (a) One taxi cab every four passengers must be allocated by the system.

   (b) The service must ensure that a taxi is allocated to only one request at the same time.

   (c) A non available taxi cab shall not be allocated to a request.

   (d) The service has to provide a taxi to a customer within 10 mins.

   (e) The service has to guarantee a fair management of the taxi queue.

   (f) When a taxi driver refuses a request, the system must push the taxi at the end of the queue.

   (g) While a taxi has a customer on board, it cannot be allocated.

   (h) If a request has been accepted by the system, the customer must be taken to its destination:

      i. If the taxi driver refuses the request, the request is forwarded to the first taxi in the queue.

      ii. If no taxis are available in the area, the first taxi in the queue of adjacent areas must be selected.

      iii. If the taxi breaks down, another taxi has to be allocated.

To provide an overview on the structure of the system, we show in figure 3.12 a class diagram of the whole system.

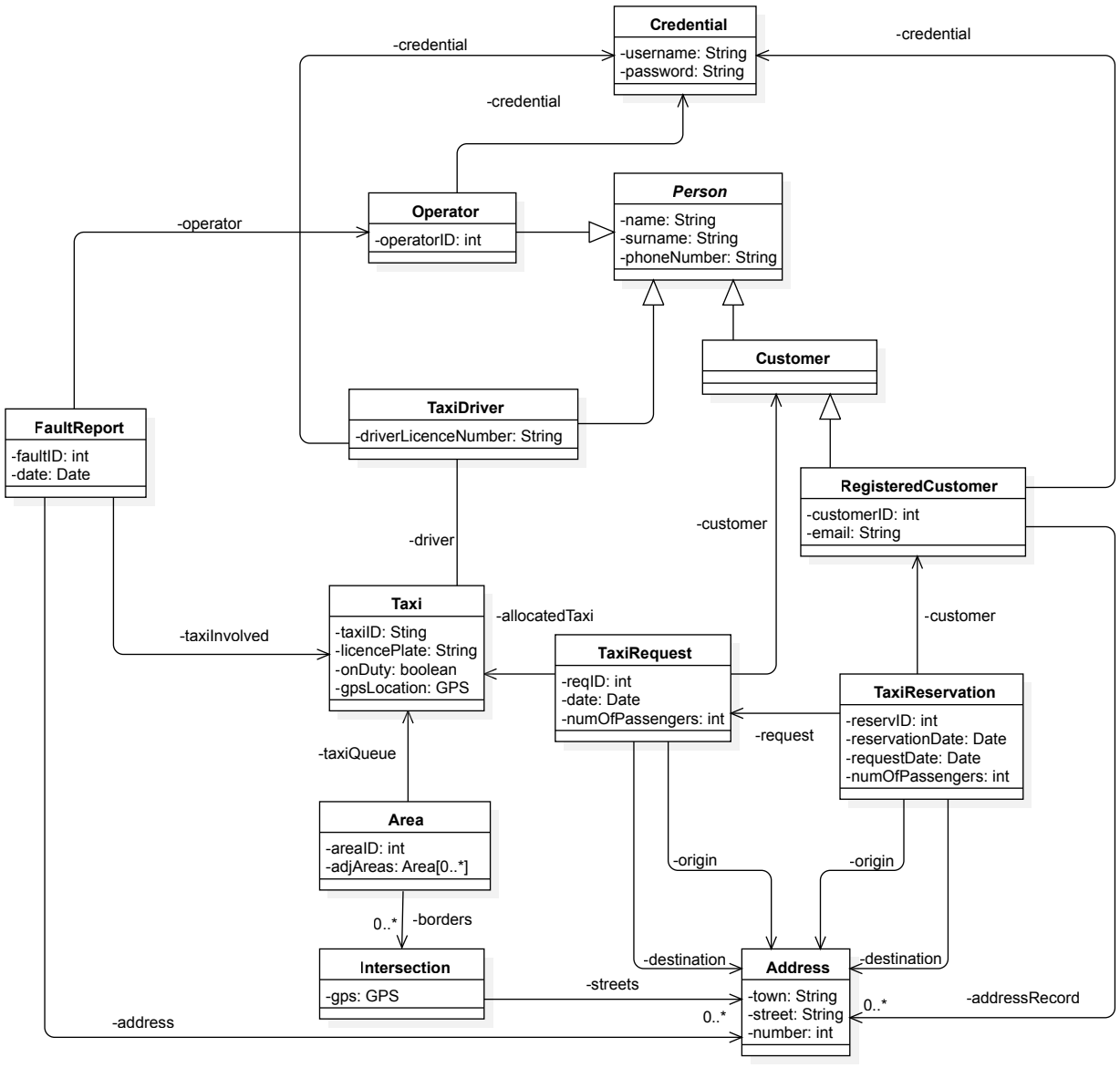


Figure 3.12: myTaxiService high level class diagram.

3.3 Performance requirements

The system has to support the simultaneous connection of all taxi drivers, who are estimated to be around 4000. We expect to have about 5 customers per taxi per hour during rush hours[1], so we need to support the simultaneous connection of at least 20 000 users, plus the users that connects to reserve a taxi for the day, that are estimated to be less numerous.

Each user exchange mainly small amounts of textual data, such as dates and addresses.

We can estimate that most users, during peak workload, are looking for a taxi in the immediate, so the system shall respond at the 90 % of the requests within 2 seconds and a taxi should reach the customer within 10 minutes for the 98 % of the requests. The reservation can be considered not a priority during rush hours, so the system shall respond to them within 5 seconds.

During normal workload, between 9AM and 4PM, we expect that most requests are going to be for reservations, which will be distributed during the day, so we expect a response time of less than 1 second for 99.9 % of all the incoming request.

[1] This estimation was calculated by dividing the mean time to cross an area (10 minutes) by one hour duration (60 minutes), and then subtracting 1.

3.4 Logical database requirements

We need the database to process at least 500 transactions per second.

As of the total number of registered users, there is no reasonable limit to state.

3.5 Design constraints

Actually, there is no major design constraint to take into account.

3.6 Software system attributes

There are a number of attributes of software that can serve as requirements, which we will explore in the following subsections.

3.6.1 Reliability

The system does not cover critical functions, so minor inconsistencies and faults in data integrity may be acceptable, however undesirable. The cloning method, already stated in section 2.1.1, may be of help to reach this target.

Anyway, the reliability of the system is related to the reliability of the server it runs on, so there is no point in stating strict reliability constraints.

3.6.2 *Availability*

We expect no more than a few hours of downtime during one year. To be more specific, we look for an availability of 99.9 % of uptime, considering possible technical faults and maintenance activities.

3.6.3 *Security*

All communications to and from the system shall be done over an HTTPS protocol, to guarantee the confidentiality of login credentials and personal data.

For legal purposes, a record of all taxi rides is kept. In future, it may become necessary to redefine the queue policy in area according to expected affluence, so these records have also statistic purposes.

The system shall ensure that personal data are accessible only in defined cases and for specific uses:

- only government can access full user's personal data;
- specific situations may require that the taxi driver calls the customer, so upon request an operator can provide customer's number to the driver; the request is registered;
- customer's personal data shall never be provided to other users;
- operators can access driver's phone number and taxi information; the access is registered.

3.6.4 *Maintainability*

As of the code, we want it to be well documented (namely, at least 90 % of the code should be covered by comments and documentation). This will facilitate further interventions, in future.

As best practice that should be followed in every object-oriented application (and that shall be followed for this project) is the adoption of the Model-View-Controller architectural pattern.

3.6.5 *Portability*

myTaxiService has no need of portability, since it will not be a distributed system itself. As of myTaxiApp and myTaxiAssist, please refer to section 2.1 for compatibility constraints.

A
Alloy model

To provide a validation of our model, we provide an Alloy code:

```
module myTaxiService


-----  SIGNATURES  ------------------------------------------------------------

abstract sig Person {
}

sig Customer extends Person {}

sig Operator extends Person {}

sig RegisteredCustomer extends Customer {
}

sig TaxiDriver extends Person {
}

sig Taxi {
    gpsLocation: one GPS,
    driver: one TaxiDriver,
}

sig GPS{}

sig Address {
    isIn: one Area,
}

sig Date {}

sig TaxiRequest {
    date: one Date,
    origin: one Address,
    destination: one Address,
    allocatedTaxi: some Taxi, -- there may be more than 4 passengers
    customer: one Customer,
} {
    origin != destination
}
```

```
sig TaxiReservation {
    reservationDate: one Date,
    requestDate: one Date,
    origin: one Address,
    destination: one Address,
    taxiRequest: one TaxiRequest,
    customer: one RegisteredCustomer,
} {
    requestDate = taxiRequest.date
    origin = taxiRequest.origin
    destination = taxiRequest.destination
    customer = taxiRequest.customer
    reservationDate != requestDate
}

sig Intersection {}

sig Area {
    adjAreas: some Area,
    taxiQueue: set Taxi,
    borders: some Intersection,
    contains: some Address,
}

sig FaultReport {
    operator: one Operator,
    taxiInvolved: one Taxi,
}


-----  FACTS  -------------------------------------------------------------

-- All TaxiDrivers have one and only one taxi.
fact oneTaxiPropriety {
    no disj t1, t2: Taxi | t1.driver = t2.driver
}

-- Adjacency is a symmetric, non reflexive property.
fact adjacentProprieties {
    all a, b: Area | (a in b.adjAreas) iff (b in a.adjAreas)  -- symmetry
    all a: Area | a not in a.adjAreas  -- non reflexivity
}

-- A taxi belongs to one and only one taxi queue.
fact noTaxiUbiquity {
    all disj a1, a2: Area | (a1.taxiQueue & a2.taxiQueue) = none
}

-- An address is located in a specific area.
fact noAddressesUbiquity {
    all disj a1, a2: Area | (a1.contains & a2.contains) = none
}

-- If an address is in an area, then that area contains that address.
fact inverseFunction1 {
     isIn = ~ contains
}
```

```
-- A customer can have only one active request.
fact oneActiveRequest {
    all disj r1, r2: TaxiRequest |
        r1.date = r2.date implies ((r1.customer & r2.customer) = none)
}

-- A taxi is assigned at most one request at the same time.
fact requestAssignation {
    all disj tr1, tr2: TaxiRequest |
        tr1.allocatedTaxi = tr2.allocatedTaxi implies ((tr1.date & tr2.date) = none)
}

-- A request can correspond to at most one reservation.
fact oneReservationOneRequest {
    no disj t1, t2: TaxiReservation | (t1.taxiRequest & t2.taxiRequest) != none
}


-----  ASSERTIONS  ------------------------------------------------------------

-- No multiple allocations of the same taxi to the same request.
assert noMultipleAllocations {
    all disj t1, t2: TaxiRequest |
        (t1.date = t2.date) implies ((t1.allocatedTaxi & t2.allocatedTaxi) = none)
}
check noMultipleAllocations

--A customer can have only one reservation at the same moment.
assert oneActiveReservation {
    all disj r1, r2: TaxiReservation |
        (r1.requestDate = r2.requestDate) implies ((r1.customer & r2.customer) = none)
}
check oneActiveReservation


-----  PREDICATES  ------------------------------------------------------------

pred showNoReservation() {
    #Customer > 0
    #RegisteredCustomer = 0
    #TaxiReservation = 0
    #FaultReport = 0
}
run showNoReservation

pred showFaultReport() {
    #Customer = 1
    #FaultReport = 1
}
run showFaultReport

pred show() {
    #Customer > 0
    #RegisteredCustomer > 0
    #TaxiRequest  > 0
    #Operator  = 0
```

```
    #FaultReport  = 0
}
run show
```

This is the result of the final computation:

```
5 commands were executed. The results are:
   #1: No counterexample found. noMultipleAllocations may be valid.
   #2: No counterexample found. oneActiveReservation may be valid.
   #3: Instance found. showNoReservation is consistent.
   #4: Instance found. showFaultReport is consistent.
   #5: Instance found. show is consistent.
```

Appendix

Working time

Paolo Antonini 32 hours.

Andrea Corneo 29 hours.

Version control

- **1.0**, 6th November 2015: first release;
- **1.1**, 10th December 2015: graphical fixes, tuning in section 3.4 on database requirements.
- **1.2**, 21st January 2016: graphical fixes.
- **1.3**, 5th February 2016: final release, with graphical fixes and overall checks.