

Design Document

Filippo Agalbato
850481

Andrea Cannizzo
790469

December 2, 2015

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Scope	2
1.3	Definitions, acronyms and abbreviations	2
1.4	References	3
1.5	Overview	3
2	Architectural design	4
2.1	Overview	4
2.2	High level components and their interactions	4
2.3	Lower level modules view	4
2.4	Deployment view	4
2.5	Runtime view	4
2.6	Component interfaces	4
2.7	Architectural styles and patterns	5
2.8	Other design decisions	5
3	User interface design	6
4	Algorithm design	15
4.1	Main controller	15
4.2	Reservations queues	17
4.3	Taxis as entities	18
4.4	Zones as entities	19
5	Requirements traceability	20
A	Document and work information	21
A.1	Revisions	21
A.2	Tools used	21
A.3	Overall time spent	21

Chapter 1

Introduction

1.1 Purpose

This is the Design Document for the project software MY TAXI SERVICE. The goal of this document is to describe the system in term of architectural choices and design decisions, to provide an overview of the interactions between its components and the users, and to present samples of algorithmic pseudocode for the core systems. This document is targeted to the customer's project managers, to evaluate the project architecture from a high-level point of view, and to the intended designers, developers, programmers, analysts and testers, to build the actual software and to maintain, integrate and expand it in the future.

1.2 Scope

The aim of this project is to build a software system able to manage and optimize taxi services in a large city, targeted both to the city public transportation management council and to the citizens as customers of the taxi service. The system will allow passengers to use the service in a smart and accessible way, and at the same time will offer a better service thanks to an enhanced management of taxi resources and allocation.

In particular, passengers and taxi drivers will be registered and remembered by the system. Passengers will be able to request a taxi by using the application and taxi drivers will be notified of these requests by the system, that will divide the city in several taxi zones each with its own queue of taxis. Taxi drivers notified of passengers requests can then accept them.

1.3 Definitions, acronyms and abbreviations

- Passenger: A citizen recognized by the system as a registered user of MY TAXI SERVICE, using the system as a customer does, to call taxis and be served;
- Logged passenger: A passenger that is currently and correctly logged into the system;

- Taxi driver: A citizen recognized by the system as a registered user of MY TAXI SERVICE, using the system as a taxi driver, recognized by the parent company and of verified identity;
- Logged taxi driver: A taxi driver that is currently and correctly logged into the system;
- Request for service: A passenger uses the mobile app or web application to send the central management system a formal and well-formed request to be served by a taxi. This request includes a departure time, an itinerary starting point and ending point and the number of people to be served.
- Real time request: A request for service that is intended to serve a passenger immediately and is instantly relayed by the system to an available taxi driver;
- Reservation: A request for service that is not meant to serve a passenger immediately and as such is stored by the system and only relayed to an available taxi driver once the correct time comes;

1.4 References

- Specification Document: My Taxi Service project specification for the Academic Year 2015-16 – *Assignments 1 and 2 (RASD and DD).pdf*.
- Agalbato, Cannizzo, *Requirement Analysis and Specification Document*.
- IEEE Std 1016-2009 Software Design Specifications.

1.5 Overview

This document is structured as follows:

- Introduction: A declaration of the scope, intent and purpose of this document;
- Architectural design: A high level view of the architecture of this system and a low level approach at its basic modules, with emphasis on their interaction, deployment, and lifecycle, including also a review of the architectural choices that were made;
- Algorithm design: A detailed look at some algorithmic design for some of the core systems;
- User interface design: An analysis of the needs of the user and how the system can meet them in terms of interface;
- Requirements traceability: An overview of how the functional requirements defined in the previous analysis map into the architectural choices.

Chapter 2

Architectural design

2.1 Overview

2.2 High level components and their interactions

2.3 Lower level modules view

2.4 Deployment view

Two artifacts are going to be deployed, each with a different target.

The first will be the *server* artifact, which will operate on a cluster of servers and will be composed of all components from the, as mentioned, Business and Web tiers. These will form the server side of the whole system, and will interface with their respected database components from the Data tier.

The second will be the *client* artifact, in the form of the MY TAXI SERVICE mobile app, which will run on smartphones and interface with the *server* artifact with the discussed methodologies, and will be made up of all previously highlighted components belonging to the Client tier.

2.5 Runtime view

2.6 Component interfaces

An Event-based style being used, components have little to no public interfaces to show to the outside world. Every information exchange taking place in the system, any order being given and received, everything is passed along as a *message* event, to which each component, upon inspecting its contents, will react according to what it knows it needs done. As such, every representation of such interfaces is by definition barebone, but a diagram is included for the sake of clarity.

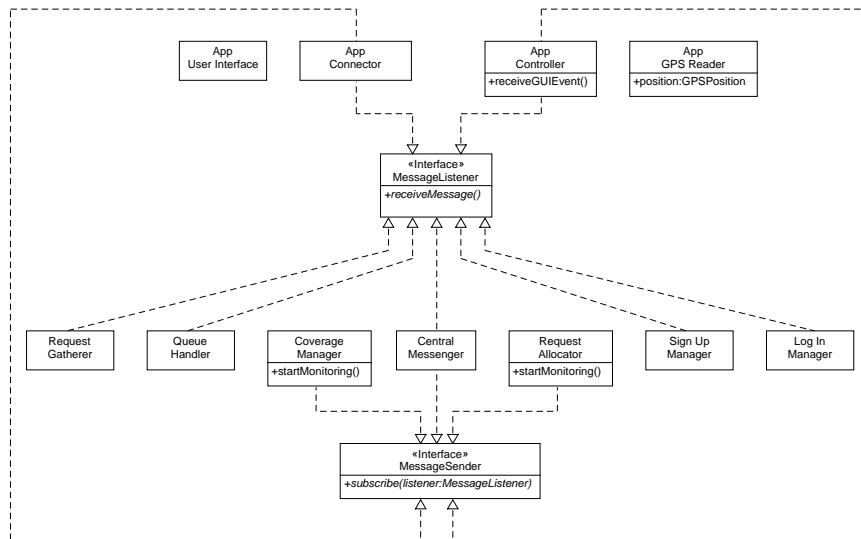


Figure 2.1: Component interfaces

2.7 Architectural styles and patterns

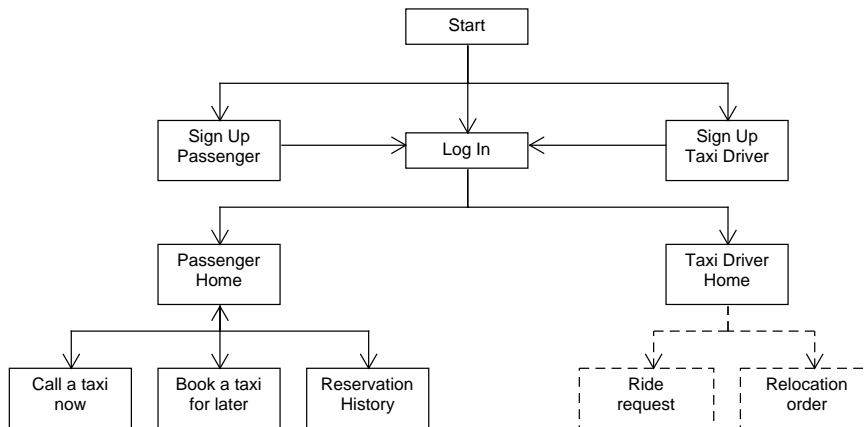
2.8 Other design decisions

Chapter 3

User interface design

The following are simple mockups of how the MY TAXI SERVICE app should appear. In the case of the webapp, the required design is very much the same.

First, is a general overview of how the different screens are navigated by the user.



In the following pages, several mockups of the main screens.

My Taxi Service

Log In

Email Address

Password

Passenger
Sign Up

Taxi Driver
Sign Up

Figure 3.1: Sign Up / Log in view

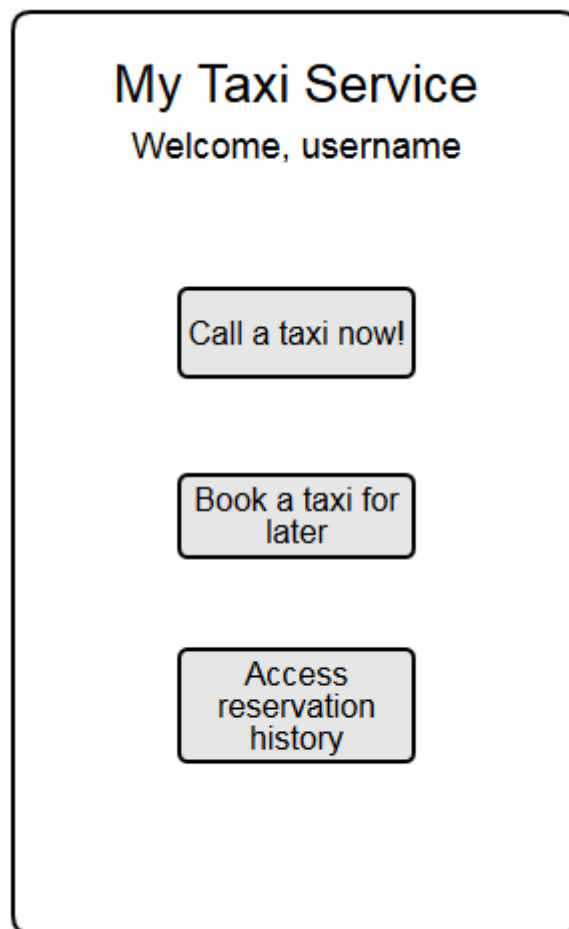


Figure 3.2: Passenger home

My Taxi Service

Call a taxi now

Destination address

Number of passengers

1

2

3

4

More

☒ Use my GPS

Submit

Figure 3.3: Call a taxi now function view

My Taxi Service

Book a taxi for later

Destination address

Pickup address

Number of passengers

1


2

3

4

More

Booking date

▼

Submit

Figure 3.4: Booking function view

My Taxi Service

Reservation history

Date

Destination address

Pickup Address

Number of people

Modify...

Delete

Date

Destination address

Pickup Address

Number of people

Figure 3.5: Reservation history view

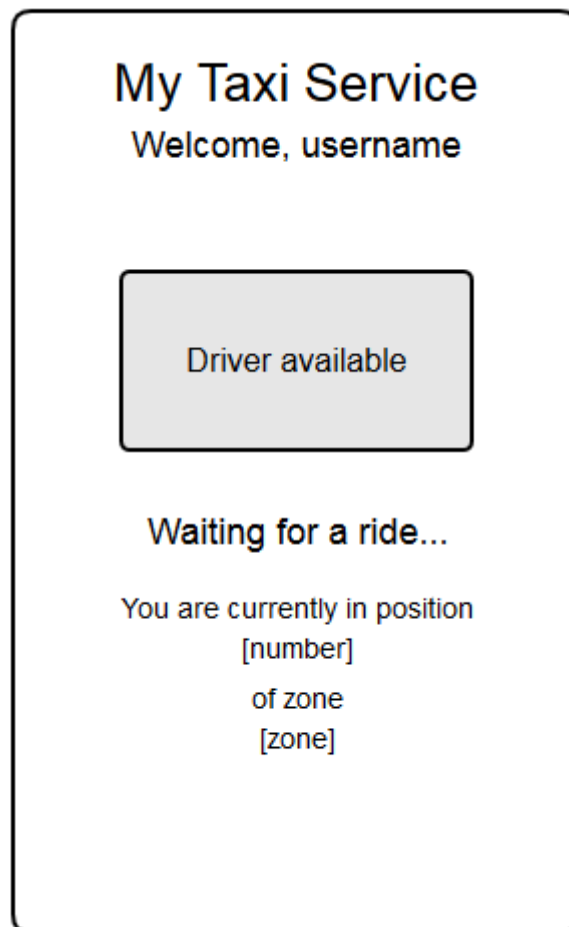


Figure 3.6: Taxi driver home

My Taxi Service

Welcome, username

Ride received

Pick up [number] people at [address].

You are currently in position
[number]
of zone
[zone]

Figure 3.7: Taxi driver receives a ride request

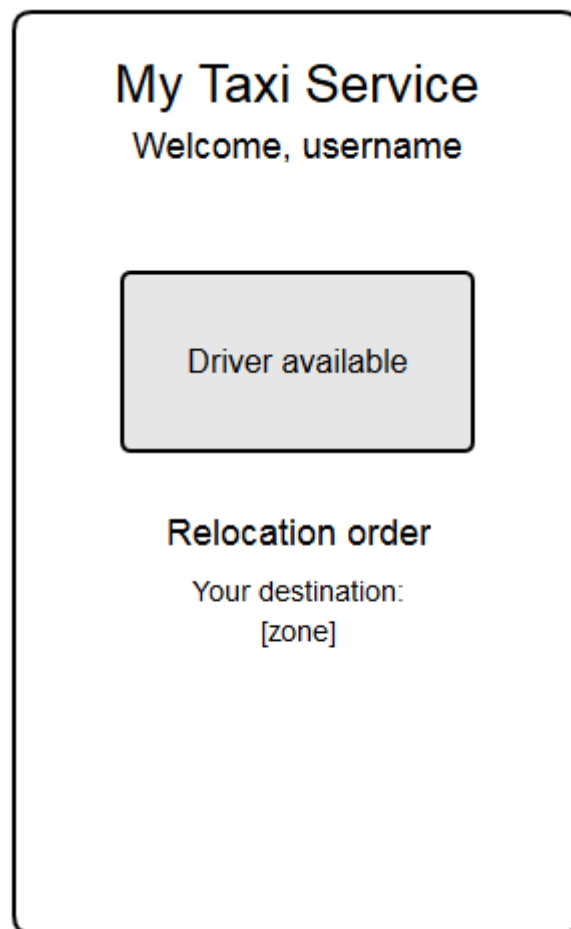


Figure 3.8: Taxi driver receives a relocation request

Chapter 4

Algorithm design

The algorithms provided here are written in a Java-like language, with classes and methods, to provide samples of the core functionalities.

4.1 Main controller

Core functionalities that bind together the system.

```
public class Controller {
    public static Zone[] map; // list of zones
    public static int zones; // number of zones
5    public Taxi[] toMove; // list of Taxi to be moved
        from a zone to an other
    public int toMovePointer; // pointer of toMove list

    public int contaDisponibili(){
10        //it counts every available taxi in the city and
            returns that number

        int a = 0;
        for(int i=0; i<zones; i++){
            a=a+map[i].contaTaxi();
15        }
        return a;
    }

20    public int totCoeff() {

        // it calculate the sum of the coefficients of
            every zone and returns that number

25        int a = 0;
```



```

        for (int i=0; i<zones; i++){
            a = a + map[i].coefficient;
        }
        return a;
30    }

    public int allocTaxi(int zone) {

35        // it calculate how many taxis should be available
           in the zone given as parameter

        int a = 0;
        a = (contaDisponibili() / totCoeff()) * map[zone].
            coefficient;
        return a;
40    }

    public int taxiToMove(int zone) {

        // it adds taxis that have to be moved from the
           zone given as parameter to the list "toMove"
           and return number of taxis added

45        int a = 0;
        if (allocTaxi(zone) < map[zone].contaTaxi()){
            a = map[zone].contaTaxi() - allocTaxi(zone);
        }

50        for (int i=0; i<a;){
            if (map[zone].list[i].available){
                toMove[toMovePointer] = map[zone].list[i];
                i++;
55                toMovePointer++;
            }
        }
        return a;
    }

60    public int reqZones(int zone) {

        // it allocates taxis to the new zone that
           requires them and return number of taxis
           allocated

65        int a = 0;
        if (allocTaxi(zone) > map[zone].contaTaxi()){
            a = allocTaxi(zone) - map[zone].contaTaxi();
        }
    }

```

```

70     for (int i=0; i<a;){
        toMove[toMovePointer].zoneToGo = zone;
        toMovePointer--;
    }

75     return a;
}

80     public void cover(){

        // it moves taxis to ensure the established
        coverage

        for(int i=0; i<zones; i++){
85             taxiToMove(i);
        }
        for(int i=0; i<zones; i++){
            reqZones(i);
        }
90     }

95     public static void move(Taxi t, int z1, int z2){

        // it moves a taxi from a queue to an other

        for (int i=0; i<map[z1].pointer; i++){
100             if (t == map[z1].list[i]){

                map[z2].list[map[z2].pointer] = map[z1].list[i];
                map[z2].pointer++;
                map[z1].list[i] = map[z1].list[map[z1].pointer];
                map[z1].pointer--;
105             }
        }
    }

110 }

```

4.2 Reservations queues

How taxis are allocated to requests.

```

public class ReservationQueue {
    public Reservation[] queue; // list of reservations
    public int pointer; // pointer of the queue

    public void fulfill(Time t){

        // it allocates a taxi to the reservation that has
        // to be fulfilled in that time

        for (int i=0; i<pointer; i++){
            if (queue[i].startingT == t){
                for (int j=0; j< Controller.map[queue[i].zone
                    ].pointer; j++){
                    if (Controller.map[queue[i].zone].list[j].
                        available){

                        queue[i].server = Controller.map[queue[i].
                            zone].list[j];
                        Controller.map[queue[i].zone].list[j].
                            toServe = queue[i];
                        Controller.map[queue[i].zone].list[j].
                            available = false;
                        queue[i] = queue [pointer];
                        pointer--;
                    }
                }
            }
        }
    }
}

```

4.3 Taxis as entities

How taxis are treated by the system.

```

public class Taxi {
    public int id; // identification number of the taxi
    public boolean available; // if the taxi is
        available
    public int myZone; // identification number of the
        zone in which the taxi is
    public int zoneToGo; //identification number of the
        zone in which the taxi has to move
}

```

```

        public Reservation toServe; // the reservation that
            he has to be fulfilled

10    public void move() {
        Controller.move(this, myZone, zoneToGo);
        this.myZone=zoneToGo;
    }
}

```

4.4 Zones as entities

How city zones are treated by the system.

```

public class Zone {
    public Taxi[] list; // list of taxis
    public int pointer; //pointer of taxis list
5    public int coefficient; // coefficient of density of
        the zone (to proportion the distribution)

    public int contaTaxi(){

        // it counts the number of available taxis in the
        zone and returns that number

10        int a = 0;
        for(int i=0; i<pointer; i++){
            if (list[i].available){
                a++;
15        }
        }
        return a;
    }
}

```

Chapter 5

Requirements traceability

Appendix A

Document and work information

A.1 Revisions

This is the first version of this document. There are currently no revisions.

A.2 Tools used

TeXworks editor With PDF^LATEX, for composing and editing this document;

UMLet For drawing all kinds of diagrams;

Moqups For drawing mockups.

A.3 Overall time spent

The authors spent about 30 hours of their time, equally divided among them, working on this document.