myTaxiService
Design **D**ocument
A.Y. 2015/2016
Politecnico di Milano
Version 1.0

Cattaneo Michela Gaia, matr. 791685
Barlocco Mattia, matr. 792735

December 4, 2015

Contents

1 Introduction

1.1 Purpose

This document represents the Design Document, which main goal is to describe the overall system architecture and to show the technical design decisions made, with the support of schema and diagrams.

It delineates how the software system will be structured in order to satisfy the requirements defined in the Requirement Analysis and Specification Document (RASD), each one translated into a representation of components, interfaces, and data necessary in the implementation phase.

This document is addressed to the project development teams, technical architects, database designers and testers, as long as it has useful guidelines and a specific description of the design implementation of the system.

1.2 Scope

The system aims at simplifying the access of the passengers and guaranteeing a fair management of taxi queues.

The server listens for the requests of the clients, which can be taxi drivers or users and access to their GPS position.

The user clients are able to request a taxi and, if logged into the system, they can also make reservations and choose the payment method they prefer. On the other hand, taxi driver clients can change their status from available to unavailable, accept or decline the requests of the users and see their position in the queue of the area they are in.

The server manages the notifications to forward to the clients, for example the arrival of a taxi for a user or the requests of the users for the taxi drivers.

1.3 Definitions, Acronyms, Abbreviations

- **Definitions**

  - **User**: a person who requests a service from the system. It can be a visitor or a passenger.

  - **Visitor**: a person who is not registered in the application.

  - **Passenger**: a person who is registered in the application.

  - **Taxi driver**: a taxi driver who access the application with a specific ID.

  - **Request**: the request of a taxi in a certain area and position in the city made by a user.

  - **Reservation**: the reservation of a taxi in a certain area, place and time that can be made only by passengers.

- **Acronyms and abbreviations**

- **RASD**: Requirement Analysis and Specification Document

- **DBMS**: Database Management System

- **JEE**: Java Enterprise Edition

- **API**: Application Programming Interface

- **UML**: Unified Modeling Language

- **HTML**: HyperText Markup Language

- **HTTP**: HyperText Transfer Protocol

- **MVC**: Model View Controller

1.4 Reference documents

- myTaxiService Requirement Analysis and Specification Document (RASD)

- ISO/IEC/IEEE 42010:2011, Systems and software engineering Architecture description

1.5 Document structure

This document is divided in six main parts:

- **Introduction**: this section describes the document in general and its purpose.

- **Architectural Design**: this section specifies the architectural design part, giving information about the components involved.

- **Algorithm Design**: this section provides a general description of the main algorithms used during implementation.

- **User Interface Design**: this section outlines an overview on how the user interfaces of the system will look like.

- **Requirements Traceability**: this section explains the connection between the requirements already defined in the RASD and the design elements introduced in this document.

- **Appendix**: this section provides information about the tools used to redact this document and how many hours of work each author has spent.

2 Architectural Design

2.1 Overview

2.2 High level components and their interaction

2.3 Component view

2.4 Deployment view

2.5 Runtime view

2.6 Component interfaces

2.7 Selected architectural styles and patterns

2.8 Other design decisions

3 Algorithm Design

The most relevant algorithm of the myTaxiService application is the one implementing the queue management. It is important to optimize its implementation as long as it is the most complex and the one that distinguishes the system.
Another significant algorithm is the one that manages the forwarding of the requests of the users to the taxi drivers.

- **Initialization.** The class which manages the areas distribution and the queues of the taxi driver also deals with the initialization of the queues of the taxis. It starts assigning to every area as many taxi as the average number of the requests expected in that area, based on the statistics. Once all the areas have been filled with their optimal number of taxis, the remaining taxi drivers are sent to the areas that are most needy.
  The "needyArea" function, in fact, returns the index of the most needy area at the moment, calculating the area that has the maximum difference between the maximum number of requests recorded and the average of requests.

  COMPLEXITY: O(T) with T = number of taxis.

Algorithm 1 Initialization

1: **procedure** INITQUEUES(TaxiDriver[] taxiDrivers, Area[] areas)
2:     $i \leftarrow 0$
3:     $j \leftarrow 0$
4:     **for** $i < areas.size()$ **do**
5:         $k \leftarrow 0$
6:         **for** $k < areas[i].getAverage()$ **do**
7:             $areas[i]$.addToQueue($taxiDrivers[j]$)
8:             $j \leftarrow j + 1.$
9:         **end for**
10:     **end for**
11:     **if** $j < taxiDrivers.size()$ **then**
12:         **for** $j < taxiDrivers.size()$ **do**
13:             $areas[$needyArea($areas$)$]$.addInQueue($taxiDrivers[j]$)
14:         **end for**
15:     **end if**
16: **end procedure**

- **Manage requests.** After the initialization, the taxi drivers are able to take requests. If they accept the request, the function sets their availability to false and stops monitoring their position. It is also possible that a taxi driver declines the request or is not able to give an answer to the server. In this case, the first taxi driver is moved to the last position of the queue and forwards the request to the second of the queue, who is now moved to the first position. This action is iterated in the queue of that area until a taxi accept the request. After accepting the call, the taxi driver is set unavailable and he is deleted from the queue, as it is likely that he will end up in another area.

  COMPLEXITY: O(A+Q) with A = number of areas and Q = number of taxi enqueued in that area.

Algorithm 2 Manage requests

1: **procedure** MANAGEREQUEST(Position pos, Area area)
2:     $answer \leftarrow$ "no"
3:     $timer \leftarrow$ new Timer(60)
4:     **while** $answer =$ "no" $||$ $answer =$ "timeOut" **do**
5:         sendRequest($area$.getFirstOfQueue()$, pos$)
6:         $answer \leftarrow$ waitAnswer($timer$)
7:         **if** $answer =$ "no" $||$ $answer =$ "timeOut" **then**
8:             $area$.moveToEndOfQueue()
9:         **end if**
10:     **end while**
11:     $area$.getFirstOfQueue()$.setUnavailable$()
12:     $area$.deleteFirstFromQueue()
13: **end procedure**

- **Manage queues.** The areas of the city are represented by a graph with an array of adjacencies, which is a useful representation in order to decide how to distribute and move the taxi drivers within the areas.

  In the first place the function search for the area where the taxi is in and sets the "startingArea" variable to the index of that area, then adds the taxi driver to the queue of the area if it has not reached the maximum number of taxi. If the area is completely full it is necessary to iterate on the areas, exploiting a breadth-first search algorithm.

  The function searches for adjacent areas with the number of taxi lower than the average of the requests and, if it does not find any, it looks for adjacent areas with the number of taxi lower than the maximum allowed for that area.

  If these two conditions are not supplied, the function assign this taxi driver to the first adjacent area visited and takes from that area the last taxi of the queue. This "rejectedTaxi" is moved to an adjacent needy area, repeating the iterations done before, searching needy areas and moving taxi drivers, until it is possible to reach an equilibrium state.

  It is always feasible to exit the while cycle because the number of taxi driver can not be the maximum for all the areas, as it is not cost efficient. The system only ensures that the minimum number of taxi drivers per area is guaranteed.

  COMPLEXITY: O(A) with A = number of areas.

Algorithm 3 Manage queues

1: **procedure** MANAGEQUEUE(Area[] area, TaxiDriver taxiDriver)
2:     **for** $i < area$.size() **do**
3:         **if** $taxiDriver.position$ is in $area[i]$ **then**
4:             $startingArea \leftarrow i$
5:         **end if**
6:     **end for**
7:     **if** $area[startingArea]$.getNumOfTaxi() $< max[i]$ **then**
8:         $area[startingArea]$.addToQueue($taxiDriver$)
9:     **else**
10:         **for** $i < area$.size() **do**
11:             $area[i].distance \leftarrow$ INFINITY
12:         **end for**
13:         $area[startingArea].distance \leftarrow 0$
14:         $unvisitedQueue$.enqueue($area[startingArea]$)
15:         $rejectedTaxi \leftarrow taxiDriver$
16:         **while** !$unvisitedQueue$.isEmpty() **do**
17:             $tmpArea \leftarrow unvisitedQueue$.dequeue()
18:             **for** $i < tmpArea.adjacencies$.size() **do**
19:                 **if** $tmpArea.adjacencies[i]$.getNumOfTaxi() $< avg[i]$ **then**
20:                     $tmpArea.adjacencies[i]$.addToQueue($rejectedTaxi$)
21:                     **return**
22:                 **end if**
23:             **end for**
24:             **for** $i < tmpArea.adjacencies$.size() **do**
25:                 **if** $tmpArea.adjacencies[i]$.getNumOfTaxi() $< max[i]$ **then**
26:                     $tmpArea.adjacencies[i]$.addToQueue($rejectedTaxi$)
27:                     **return**
28:                 **end if**
29:                 **if** $tmpArea.adjacencies[i].distance =$ INFINITY **then**
30:                     $tmpArea.adjacencies[i].distance \leftarrow tmpArea.distance + 1$
31:                     $unvisitedQueue$.enqueue($tmpArea.adjacencies[i]$)
32:                 **end if**
33:             **end for**
34:             $firstUnvisited \leftarrow unvisitedQueue$.getFirstOfQueue()
35:             $rejectedTaxi \leftarrow firstUnvisited$.addToFullQueue($rejectedTaxi$)
36:         **end while**
37:     **end if**
38: **end procedure**

---

4   User Interface Design

The user interface design has been already specified in the section 3.1.1 of the RASD, where all the user interfaces are described with the support of mockups.

5   Requirements Traceability

6 Appendix

6.1 Software and tools used

- TeXstudio (`http://www.texstudio.org/`): to redact and to format this document.

- draw.io (`https://www.draw.io/`):to create all the diagrams.

6.2 Hours of work

Time spent redacting this document:

- Cattaneo Michela Gaia: ~1 hours of work.

- Barlocco Mattia: ~1 hours of work.