Politecnico di Milano

Software Engineering 2 Project
a.y. 2015-16

MyTaxiService
Design Document
Version 1.0

Casati Fabrizio, 853195
Castelli Valerio, 853992

Referent professor: Di Nitto Elisabetta

November 29, 2015

Contents

Chapter 1

Introduction

1.1 Purpose

This document represents the Software Design Description (SDD), also simply called Design Document, for myTaxiService.

The purpose of this document is to share with all the interested parties a more detailed description of how myTaxiService is designed and architected, with a particular emphasis on what design decisions the development team has made and the rationale behind them.

1.2 Scope

One of the key decisions a software engineer must make while designing a system is between which software components should be actually developed and implemented, and which can be taken for granted as already existing on the market.

This decision has an obvious impact on what a design document should describe and what components are instead left out of the analysis and simply considered ready to be used.

For this reason, throughout this document we will mainly focus on the peculiar characteristics of myTaxiService and provide explanations, diagrams and descriptions for those software components that are specific to this project. On the other hand, we will assume to be able to use a few commercial components for the most common tasks. This is done for a couple of practical reasons: firstly, it allows us to focus on the critical aspects of the project and thus to save budget; and secondly, it lets us leverage a set of well-tested, robust and scalable components that are specifically design and optimized to accomplish certain tasks, thus obtaining a result which is better than anything we could design from scratch to implement the same functionalities. We will clearly annotate which components will need to be developed as parts of our system and which ones will be obtained from a

third party.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

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

- ACID properties: set of properties of transactions in a relational database management system. Stands for Atomicity, Consistency, Isolation and Durability.

1.3.2 Acronyms

- SDD: Software Design Description.
- DD: Design Document. Used as a synonym of SDD.
- DMZ: Demilitarized Zone.
- DBMS: Database Management System.
- API: Application Programming Interface.
- RASD: Requirement Analysis and Specification Document.
- KPI: Key Performance Indicator.

1.3.3 Abbreviations

- Req. as for Requirement

1.4 Reference Documents

- Assignment document: Assignments 1 and 2 (RASD and DD).pdf
- Template for the Design Document (Structure of the design document.pdf)
- IEEE Systems and software engineering — Architecture description (ISO/IEC/IEEE 42010, first edition)
- IEEE Standard for Information Technology —Systems Design — Software Design Descriptions (IEEE Std 1016^{TM}-2009)
- Microsoft Application Architecture Guide, 2nd edition (published in 2009, ISBN: 9780735627109)

1.5 Document Structure

In order to let stakeholders easily navigate through the document, we will now briefly discuss its structure and give a short description of each section.

In the Architectural Design section, we will go in depth describing how the system is designed from different point of views. In particular, we will provide:

- A general overview of how the system is architected from a high level point of view
- A description of the main components that make up the system, their inner structure and how they interact with each other
- A view of how the components of the system are actually deployed on the physical infrastructure
- A detailed view of the normal runtime conditions in which the system operates, with sequence diagrams of the main tasks
- A list of the significant architectural styles and patterns that have been chosen to design the system

In the Algorithm Design section, we will focus on defining the most relevant and critical algorithms that drive the system operations. In particular, for each of them we will outline the key steps using a short pseudo-code representation.

In the User Interface Design, we will mainly reference the existing UI sketches that were already defined in the RASD and further refine them.

Finally, the Requirements Traceability section will explain how our software architecture fulfills the requirements that were identified in the requirement analysis phase and how those requirements have influenced our design decisions.

Chapter 2

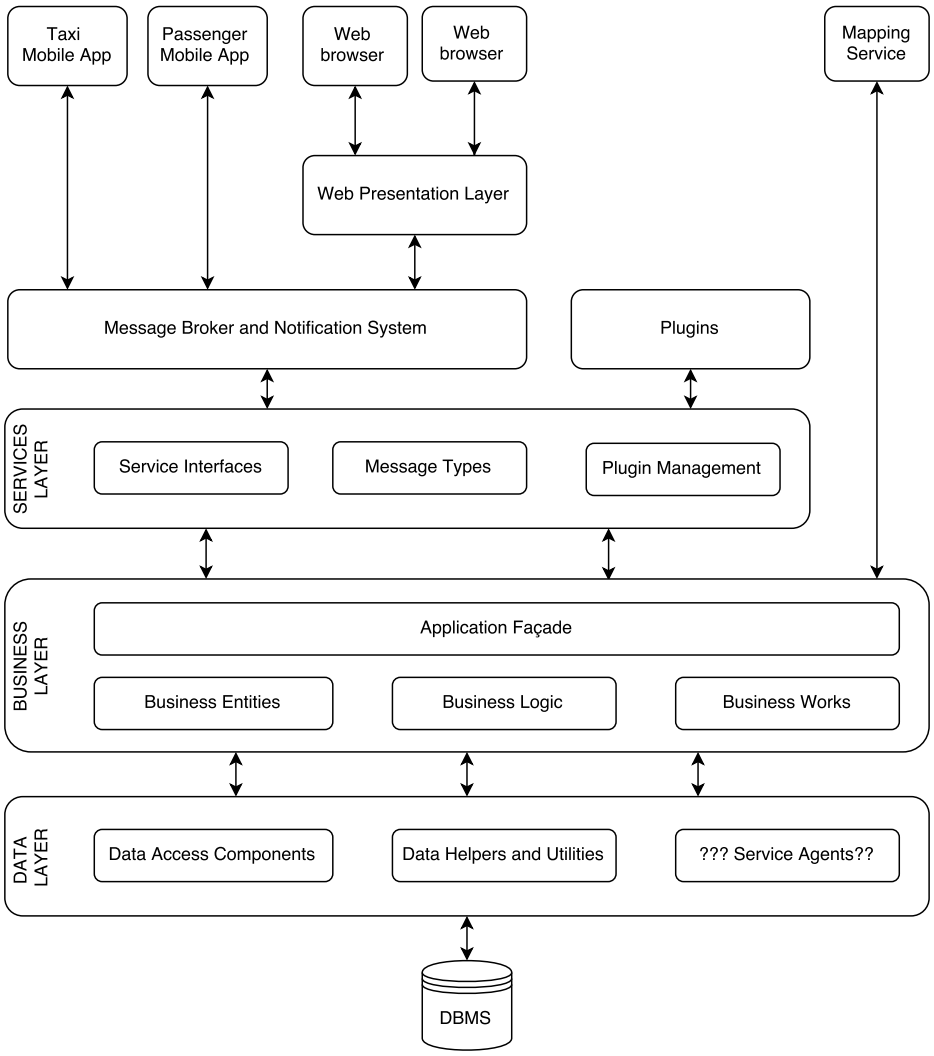
Architectural Design

2.1 Overview

In the following paragraphs we'll present a general overview of how the system is architected, with specific focus on the distinction between logically separated layers.

Specifically, myTaxiService has been envisioned from the ground up to be fully scalable and easily deployable on any number of servers. This characteristic is not only desirable, but actually fundamental for the system to fully accomplish the tasks it's been designed for. In order to guarantee this level of flexibility and modularity, we have settled for a system architecture that inherently enables a wide range of hardware configurations for all kinds of workloads.

The system architecture is thus logically organized in a set of interplaying software layers, whose detailed description is hereby presented.



The lowest layer of the architecture is composed by the relational **Database Management System (DBMS)** that supports all data storage operations. This component fully supports ACID distributed transactions and is meant to be run inside a Demilitarized Zone (DMZ) which is separated from the rest of the system for security reasons. (N.B. questo va messo qui?) In order to provide a higher level of abstraction to all components that need access to data and to be as platform agnostic as possible, the DBMS interface is not directly exposed to the classes that implements the business logic of the system. Instead, an intermediate **Data Layer** is responsible of performing queries on the DBMS while exposing a more flexible, customized interface to the upper layers.

The **Business Layer** implements all core functionalities of the system.

In particular, all operations related to handling taxi requests and reservations, taxi availability and zone management are performed by components of this layer. Data is stored and retrieved using the APIs exposed by the Data Layer. Core functionalities are exposed to clients through a unified Application Façade that allows fine-grained tuning of which operations can be invoked from outside the central system.

The Business Layer also depends on an external **Mapping Service** for the implementation of reverse geocoding and waiting time estimation operations. This external service is directly invoked by classes of the Business Layer by using a public API provided by the Mapping Service itself.

However, not every functionality exposed by the Application Façade may actually be made publicly available to every client. In principle, several levels of permissions could be offered to third parties while maintaining control over private APIs which should only be used by "official clients". Read-only reporting functionalities, for example, could be made available to any requesting party, while more critical operations could be offered only to selected developers after having verified certain requirements are satisfied. Furthermore, specific sets of APIs could be made available only to approved plugins and not be offered to remote services. For this reason, the **Services Layer** provides a comprehensive interface to all kinds of third party services and plugins by carefully defining which methods are available for remote and local invocation, what protocols should be followed for invoking them and what kind of messages can be exchanged between a remote component and the central system.

While locally invokable APIs are made available only to plugins, remotely invokable APIs are also offered to components living outside the perimeter of the central system. The **Message Broker and Notification System** is precisely concerned with guaranteeing an efficient and reliable communication channel with these remote entities by supporting message queues, publish/subscribe communications and dynamic, asynchronous event notifications.

The **Web Presentation Layer** is responsible for the implementation of the web application. It generates the dynamic web pages, offers them to the client via a web server, accepts requests and forwards them to the business layer by means of the communication layer and of the services layer.

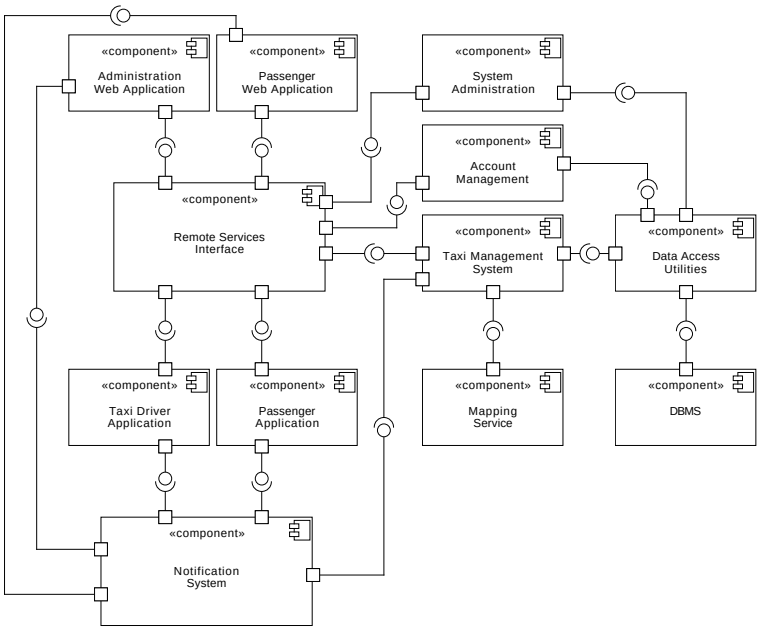
Finally, the **Mobile Applications** let users access the functionalities offered by the system on their smartphones and tablets in a native way. As for the web application, they interact with the central system using the communication layer and the services layer.

2.2 High level components and their interactions

The purpose of this section is to present the main components in which myTaxiService is divided and the relations between them.

In particular, it should be noted that the components discussed here are very abstract clusters of functionalities which do not directly map to any specific module in the final system. Instead, they are intended as an useful representation to show how the main functionalities of the system are grouped together and interact with each other. Further details on how these components are realized in the actual system will be discussed with greater depth in the Component view subsection.

A schematic representation of the component structure of the system is the following.



The **Account Management** component is responsible for all operations related to user accounts. More specifically:

- It implements the passenger registration process
- It implements the login procedure for all users
- It supports operations on existing accounts, including settings management and password retrieval operations

The **Taxi Management System** is the single most important component in the system. It is responsible for:

- Maintaining the availability status of each taxi updated
- Managing the taxi queue associated with each zone in the city
- Accepting and managing taxi reservations
- Fulfilling taxi requests by selecting the first available taxi in the corresponding taxi zone

The **System Administration** component offers system configuration and monitoring functionalities. It enables the insertion, update and deletion of taxis and taxi drivers and the definition of the boundaries of the zones in which the city is divided. It also lets administrators perform queries to obtain system statistics including uptime, number of served requests per day and other key performance indicators. The user interface for interacting with this component is provided by a web application.

The **Database Management System (DBMS)** is the component responsible for storing and retrieving data in a persistent, reliable way. It should be noted that this component will not be implemented from scratch; instead, a commercial solution will be used.

The **Data Access Utilities** component provides an abstraction layer to all those components that need to store data into the DBMS or retrieve data from it.

The **Mapping Service** component is provided by a third party and is accessed via a publicly available API. It is used to perform reverse geocoding and to compute the ETA of the taxi assigned to a certain request.

The **Remote Services Interfaces** component provides external applications and clients a way to invoke the services offered by the central system. Specifically, it implements a platform-independent, SOAP-based web service which exposes the full set of remotely invokable methods defined by the public API specification of myTaxiService.

The **Notification System** component implements an event-based mechanism to notify remote applications and clients of specific changes in the status of the central system. In particular, this is employed to let users known when the system has assigned a taxi to their reservation.

As for how the system can be accessed by its users, the **Taxi Driver Application** and the **Passenger Application** provide a way for taxi drivers and passenger respectively to interact with the system through their smartphones, while the **Passenger Web Application** allows passengers perform their operations through a web browser. Finally, administrators can access the system using the aforementioned **Administration Web Application**.

2.3 Component view

In this section, we'll provide a more detailed description of the most significant components that must be developed as part of myTaxiService.

It should be noted that, when multiple operations or tasks are managed by a certain sub-component, not all of them will be explicitly mentioned in this section. This is because, at this level of detail, we want to put the focus on the primary goal of every sub-component; an exhaustive description of the functionalities implemented by each sub-component will be presented in the Requirements Traceability section of this document.

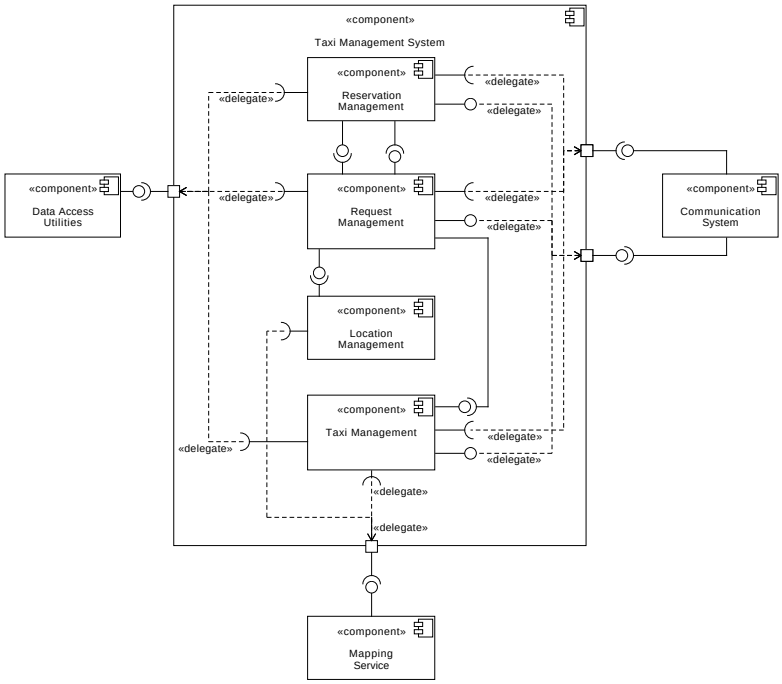
The first component that we'll examine is the **Taxi Management System**. As we already mentioned, this component is primarily responsible for all operations related to taxi management. Although it may appear to be an atomic component, it is in fact composed of three different sub-components, each of them dedicated to handling a specific subset of operations:

- The **Reservation Management** sub-component performs all the activities related to reservation handling. In particular, it is able to receive a request for a new reservation, store it into an internal queue and periodically look for reservations that have come to expiration. Once such a reservation is found, it creates a request for it and transfers the computation to the Request Manager.

- The **Request Management** sub-component performs all the activities related to request handling. In particular, it is able to receive a new taxi request (either from an end user or from the reservation manager), retrieve the associated geographical coordinates through invocation of the Mapping Service API if the request only contains the meeting address, and use them to discover the zone from which the request is coming by invocation of an appropriate method of the Location Management sub-component. It then forwards the necessary information to the Taxi Management sub-component.

- The **Location Management** sub-component is essentially responsible for checking if the geographical coordinates of a certain place are inside the city and, if they are, it computes the zone they belong to.

- The **Taxi Management** sub-component implements the methods to allocate a suitable taxi to a given request, keep the taxis' statuses updated and manage the zones' queues accordingly to the requirements which have been defined in the RASD.



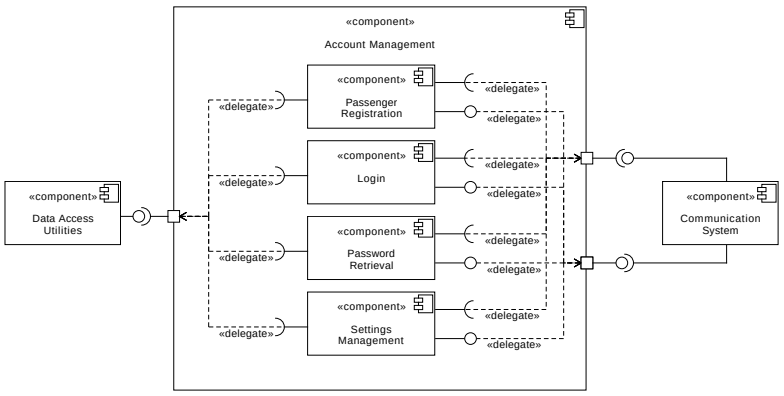
The second component that we'll examine is the **Account Management** component. As we already mentioned, this component is primarily responsible for all operations related to user accounts handling. More specifically, this component is divided into four sub-components, each related to a different kind of operation:

- The **Passenger Registration** sub-component enables passengers to register to the taxi service and create their own account. In particular, it validates the required user data for formal consistency (i.e. checks that the date of birth, email address, mobile phone number and password are in valid format) and creates a temporary user account. It then sends a verification message containing a validation link to the specified email address and, upon user's confirmation, it enables the user account for full usage.

- The **Login** sub-component performs all necessary operations to let registered users log into the system. The user can be either a registered passenger, a taxi driver or a system administrator; depending on the user type, different login data may be required and a different level of privileges will be granted.

- The **Password Retrieval** sub-components implements the password retrieval procedure for all registered users. Depending on the kind of user, a different recovery procedure may be followed.

- The **Settings Management** sub-component contains all the logic that is related to manipulation of an existing account by its owner. Depending on the kind of user, different options may be allowed.



The third component that we'll examine is the **System Administration** component. As we already mentioned, this component is primarily responsible for all operations related to system configuration and monitoring functionalities. More specifically, this component is divided into five sub-components, each related to a different kind of operation:

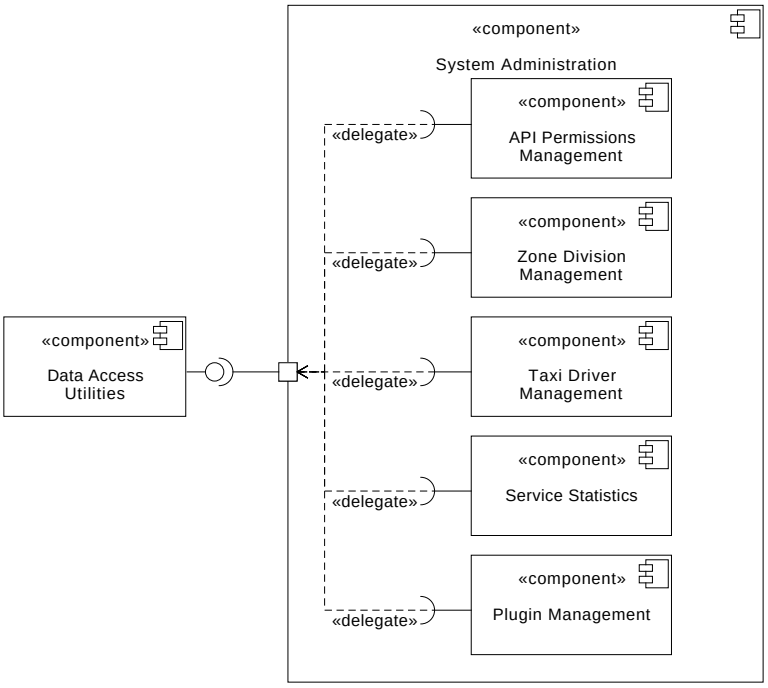
- The **API Permissions Management** sub-component give administrators fine-grained control over which APIs can be publicly accessed and what level of permissions is required to invoke them. More specifically, for each method exposed by the Remote Services Interface component it defines a visibility level that can be either "public" or "restricted". If the method is marked as "restricted", it also allows specifying a specific private key that must be held by the invoking application in order for the invocation to be successful.

- The **Zone Division Management** sub-component implements all the methods for inserting and updating the zone division of the city. In particular, it allows insertion of new taxi zones and deletion or modification of existing ones. It also performs all the necessary checks to ensure that zone consistency is preserved: in particular, this means that overlapping zones won't be accepted. For security reasons, this component verifies that its methods are only invoked by a user with a sufficient level of privileges.

- The **Taxi Driver Management** sub-component implements all the methods for inserting and updating information about taxi drivers and the related taxis in the system. In particular, it allows insertion of new taxi drivers and taxis and deletion or modification of existing ones. It

also performs all the necessary checks to ensure that information consistency is preserved. in particular, this includes validation of taxi driver licenses and taxi plates and enforces the one-to-one correspondence between a taxi driver and its taxi. For security reasons, this component verifies that its methods are only invoked by a user with a sufficient level of privileges.

- The **Service Statistics** sub-components is focused on offering a set of Key Performance Indicators (KPI) about the system operational status to all interested parties with sufficient privileges. This information can include uptime, number of served requests per day, average waiting time and other indicators.
- The **Plugin Management** sub-component let administrators install, enable, disable and remove plugins. It also gives administrators a list of all permissions required by each plugin to work and allows them to selectively grant and revoke them.



2.4 Deployment view

The main purpose of this section is to show how the various components of the system are actually deployed on the hardware infrastructure.

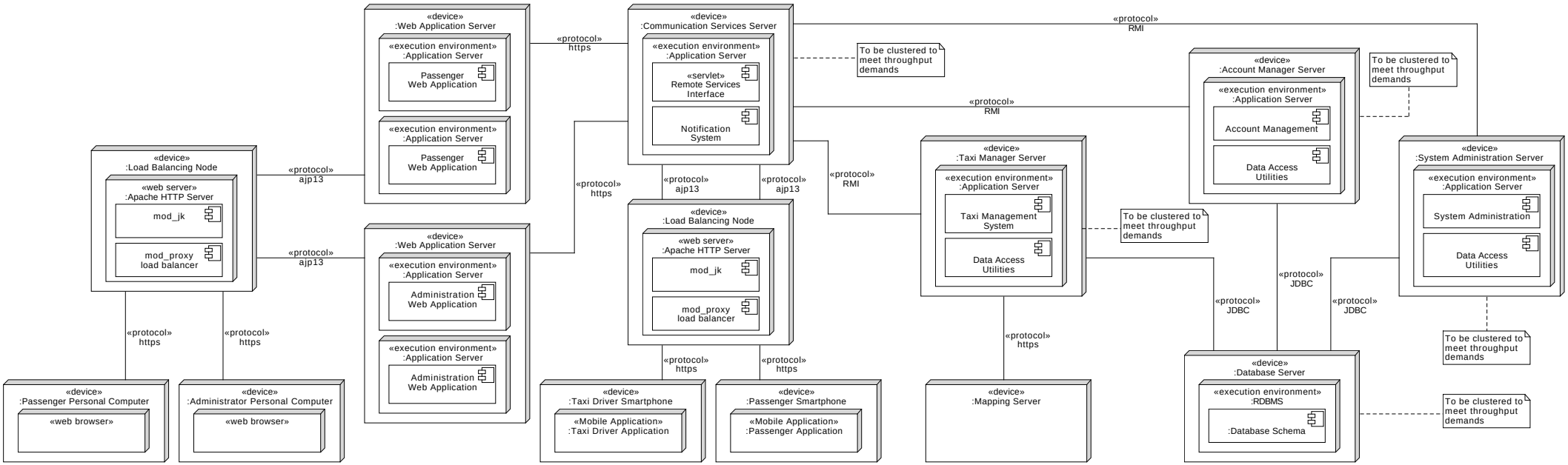
The design of the hardware architecture was mostly influenced by these concerns:

- It had to be sufficiently reliable, available and scalable with respect to the non-functional requirements stated in the RASD
- It had to be compatible with the limited financial resources of a city administration
- It had to be manageable by the city administration staff

For these reasons we have decided to avoid implementing a custom server farm specifically for this project. Instead, we have chosen to deploy myTaxiService on a third-party cloud infrastructure offered as a Platform as a Service.

The following diagram describes which are the most important devices and how software components are deployed on them.

It should be noted that, for readability reasons, we have explicitly depicted only a single instance of each hardware component despite the fact that they are actually arranged in replicated clusters. A load balancer is used to redistribute the workload across the instances.


«device»
:Load Balancing Node
«web server»
:Apache HTTP Server
mod_jk
mod_proxy
load balancer
«protocol»
ajp13
«protocol»
ajp13
«device»
:Web Application Server
«execution environment»
:Application Server
Passenger
Web Application
«execution environment»
:Application Server
Passenger
Web Application
«protocol»
https
«device»
:Communication Services Server
«execution environment»
:Application Server
«servlet»
Remote Services
Interface
Notification
System
To be clustered to
meet throughput
demands
«protocol»
RMI
«protocol»
RMI
«device»
:Account Manager Server
«execution environment»
:Application Server
Account Management
Data Access
Utilities
To be clustered to
meet throughput
demands
«protocol»
https
«protocol»
ajp13
«protocol»
ajp13
«protocol»
RMI
«device»
:Taxi Manager Server
«execution environment»
:Application Server
Taxi Management
System
Data Access
Utilities
To be clustered to
meet throughput
demands
«device»
:System Administration Server
«execution environment»
:Application Server
System Administration
Data Access
Utilities
«protocol»
https
«protocol»
https
«device»
:Web Application Server
«execution environment»
:Application Server
Administration
Web Application
«execution environment»
:Application Server
Administration
Web Application
«device»
:Load Balancing Node
«web server»
:Apache HTTP Server
mod_jk
mod_proxy
load balancer
«protocol»
JDBC
«protocol»
JDBC
«protocol»
JDBC
«protocol»
https
«protocol»
https
«protocol»
https
«protocol»
https
«device»
:Passenger Personal Computer
«web browser»
«device»
:Administrator Personal Computer
«web browser»
«device»
:Taxi Driver Smartphone
«Mobile Application»
:Taxi Driver Application
«device»
:Passenger Smartphone
«Mobile Application»
:Passenger Application
«device»
:Mapping Server
«device»
:Database Server
«execution environment»
:RDBMS
:Database Schema
To be clustered to
meet throughput
demands
To be clustered to
meet throughput
demands

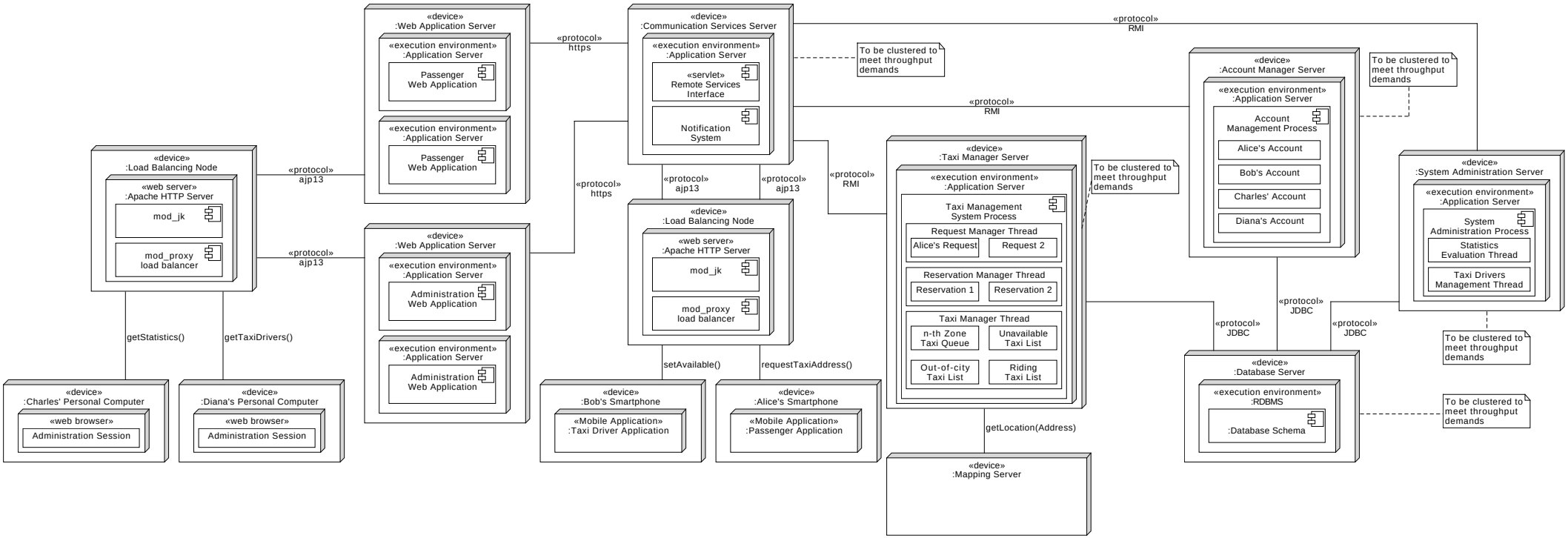
2.5 Runtime view

The main purpose of this section is to show how the various components of the system interact with each other in some real world scenarios.

The first picture represents a snapshot of the system during its execution. In this scenario Alice is a passenger who has requested a taxi using the passenger mobile application installed on her smartphone, Bob is a taxi driver who has marked himself as available, Charles and Diana are two administrators that are performing queries on the system. The diagram clearly shows the set of processes and threads running on all the devices; in particular:

- The Taxi Management Server is running a single Taxi Management System Process that is responsible of different threads, one for every kind of taxi-related operation. For example, the Request Manager thread is shown controlling a set of taxi requests, including Alice's; the Reservation Manager thread is shown controlling a set of taxi reservations; and the Taxi Manager Thread is shown managing the taxi queues of the different zones and the list of unavailable, out-of-city and currently riding taxis.
- The Account Manager Server is running a single Account Management process that contains a list of all active user accounts. In particular, in the picture it is shown containing Alice's, Bob's, Charles' and Diana's accounts.
- The System Administration Server is running a single System Administration process that is responsible of different threads, one for every administrative transaction operated on the system. Specifically, the picture shows the Statistics Evaluation thread invoked to satisfy Charles' query and the Taxi Drivers Management thread invoked to satisfy Diana's request.
- The Communication Services Server is running an instance of the Remote Services Interface servlet to expose the key functionalities of the system to remote clients and an instance of the Notification System.
- The Web Application Servers are running instances of the Passenger Web Application presentation logic and of the Administration Web Application presentation logic.
- The Load Balancing nodes are running instances of the Apache Web Server to redirect HTTP requests to different instances of the business logic nodes.

The next pictures in this section depict the flow of events (in particular method invocations) for the most important operations in the form of sequence diagrams.


«device»
:Load Balancing Node
«web server»
:Apache HTTP Server
mod_jk
mod_proxy
load balancer
«protocol»
ajp13
«protocol»
ajp13
«device»
:Web Application Server
«execution environment»
:Application Server
Passenger
Web Application
«execution environment»
:Application Server
Passenger
Web Application
«device»
:Web Application Server
«execution environment»
:Application Server
Administration
Web Application
«execution environment»
:Application Server
Administration
Web Application
«protocol»
https
«protocol»
https
«device»
:Communication Services Server
«execution environment»
:Application Server
«servlet»
Remote Services
Interface
Notification
System
To be clustered to
meet throughput
demands
«protocol»
ajp13
«protocol»
ajp13
«device»
:Load Balancing Node
«web server»
:Apache HTTP Server
mod_jk
mod_proxy
load balancer
«protocol»
RMI
«protocol»
RMI
«device»
:Taxi Manager Server
«execution environment»
:Application Server
Taxi Management
System Process
Request Manager Thread
Alice's Request
Request 2
Reservation Manager Thread
Reservation 1
Reservation 2
Taxi Manager Thread
n-th Zone
Taxi Queue
Unavailable
Taxi List
Out-of-city
Taxi List
Riding
Taxi List
To be clustered to
meet throughput
demands
«protocol»
RMI
«device»
:Account Manager Server
«execution environment»
:Application Server
Account
Management Process
Alice's Account
Bob's Account
Charles' Account
Diana's Account
To be clustered to
meet throughput
demands
«device»
:System Administration Server
«execution environment»
:Application Server
System
Administration Process
Statistics
Evaluation Thread
Taxi Drivers
Management Thread
To be clustered to
meet throughput
demands
«protocol»
JDBC
«protocol»
JDBC
«protocol»
JDBC
«device»
:Database Server
«execution environment»
:RDBMS
:Database Schema
To be clustered to
meet throughput
demands
getStatistics()
getTaxiDrivers()
«device»
:Charles' Personal Computer
«web browser»
Administration Session
«device»
:Diana's Personal Computer
«web browser»
Administration Session
setAvailable()
requestTaxiAddress()
«device»
:Bob's Smartphone
«Mobile Application»
:Taxi Driver Application
«device»
:Alice's Smartphone
«Mobile Application»
:Passenger Application
getLocation(Address)
«device»
:Mapping Server

2.6 Component interfaces

2.7 Selected architectural styles and patterns

While designing the software architecture for myTaxiService we looked at all the possible options that enabled us both to satisfy current functional and non-functional requirements and to have enough flexibility for future needs and updates.

In this section we're going to highlight the main architectural styles and patterns that we've decided to use as part of our software architecture, giving a brief description for each of them and explaining why we chose them.

- Software Oriented Architecture (SOA). This architectural style allows us to design our system as a set of services which can be separately offered to third parties. The adoption of this architectural style provides several benefits. First, it allows us to easily implement the APIs that must be exposed by the core system, which can be seen as a set of services offered by myTaxiService to remote clients and applications. Second, this give us great flexibility when it comes to further expansions of the set of supported functionalities: by modeling our system as a SOA, it is possible to start offering a set of core services and then increase the number of public APIs as additional components and services are developed.

- Layered Architecture. This architectural style enables a great level of separation of concerns between the various components of the system, such that every component operates at a single logical layer of abstraction. Furthermore, each layer can be separately instantiated on a different machine in a completely orthogonal way, allowing great flexibility in terms of hardware configurations. Finally, this design choice allows us to obtain a greater level of clarity and flexibility: this simplifies the implementation phase, makes testing easier and more effective and dramatically improves maintainability and extendability.

- N-Tier Physical Architecture. This architectural style allows us to run the various components of the system on different devices, not necessarily located in the same server farm. This flexibility in terms of allocation of hardware resources provides a number of key advantages. First of all, it improves security and resilience with respect to attacks: in fact it clearly separates the hardware which runs processes directly connected to the outside world (the web servers) from the hardware running more sensitive and critical services, like the account manager, the administration services and the taxi management system. Furthermore, this gives us the opportunity to easily scale the number of machines of each tier to more efficiently adapt to variations of demand.

- Publish/Subscribe. This architectural style is primarily used in the implementation of the notification system. Even though one-to-one communications between the central system and remotely connected devices could be achieved without the usage of the publish/subscribe architectural style, this design choice provides us greater flexibility with respect to future expansions of the core functionalities. An example of situation in which this architectural style could be useful is the addition of taxi sharing.

2.8 Other design decisions

Chapter 3

Algorithm Design

Chapter 4

User Interface Design

Chapter 5

Requirements Traceability

In this section we'll specifically cover in detail how the requirements identified in the SRS document are satisfied by our myTaxiService architecture.

5.1 Functional Requirements

For sake of readability, we will use the notation **req. x.y** to make reference to the y-th requirement associated with the x-th goal.

- **Goal 1, 2 and 3** are essentially administration-oriented requirements; as such, they are satisfied by:
  - The **System Administration** component, which provides the backend business logic to handle the required data modification operations. In particular, operations related to taxi and taxi drivers (goal 1, 2) are handled by the Taxi Driver Management sub-component, while operations related to the definition of taxi zones (goal 1, 3) are handled by the Zone Division Management sub-component.
  - The **Administration Web Application** component, which provides administrators an appropriate user interface to interact with the backend.
- **Goal 4** describes the way in which the availability status of a taxi driver is updated. As such, several components are involved in its fulfillment:
  - The **Taxi Driver Application** component, which provides taxi drivers the ability to interact with the central system to notify a change in their availability status. In particular, this component is responsible for reqs. 4.1 to 4.6 and reqs. 4.8, 4.11.

  - The **Taxi Management System** component, which provides the backend business logic to handle the update operations. In particular:
    * Req. 4.6 and 4.7 is handled by the Location Management sub-component for the zone computation part and by the Taxi Management sub-component for the update operation on taxi queues
    * Reqs 4.9 and 4.10 are handled by the Taxi Management sub-component by moving the taxi to the unavailability list and setting its status to "unavailable"
    * Req. 4.8 and 4.11 are handled by the Taxi Management sub-component
  - The **Notification System** component, which provides to the central system the dispatch mechanism it needs for sending notifications to the taxi driver application. This component is essentially involved in satisfying reqs. 4.6, 4.8 and 4.11.

- **Goal 5** requires the allocation and distribution of taxis to be managed fairly and consistently. This is achieved by an interplay of components:
  - The **Taxi Management System** is the key component that concurs to the fulfillment of the goal. More specifically, reqs. 5.1 to 5.10 are handled by the Taxi Management sub-component, with the aid of the Location Management sub-component as for those requirements that involve checking whether the taxi driver is still inside the city or not.
  - The **Taxi Driver Application** component is crucial in fulfilling reqs 5.11 and 5.12 by providing an appropriate management of the UI to prevent incorrect availability assignments and by sending the GPS coordinates of the taxi driver to the central system with a given frequency.
  - The **Remote Services Interface** component, which enables the taxi driver application to request services to the central system and thus provides the necessary callbacks to satisfy req. 5.12.

- **Goal 6** is related to the ability of a taxi driver to receive, accept and refuse ride requests. This involves:
  - The **Taxi Management System** component, which is responsible of fulfilling reqs. 6.1, 6.2, and 6.4 to 6.6 by means of the Taxi Management sub-component.
  - The **Mapping Service** component, which provides the reverse-geocoding capability to fulfill req. 6.2 and the map data to fulfill req. 6.8.

  - The **Taxi Driver Application** component, which is responsible of fulfilling reqs. 6.1, 6.3, 6.7 and 6.8 by providing to the taxi driver an appropriate UI.
  - The **Notification System** component, which provides to the central system the dispatch mechanism it needs for sending notifications to the taxi driver application and fulfill req 6.1.
  - The **Remote Services Interface** component, which enables the taxi driver application to request services to the central system and thus provides the necessary callbacks to satisfy reqs. 6.4 and 6.5.

- **Goal 7** gives a taxi driver the ability to drop a request if the passenger doesn't show up. In order for this functionality to work correctly, several components should cooperate:

  - The **Taxi Driver Application** component, which provides a suitable UI to drop the request (req. 7.1) and retrieves the current GPS location fo the taxi driver in order to let the central system check if it matches the agreed meeting point (req. 7.2).
  - The **Taxi Management System** component, which is responsible of satisfying all the three requirements related to this goal, in particular my means of the Taxi Management and Request Management sub-components.
  - The **Passenger Application** component, which is responsible of implementing a suitable UI view to notify a user that his request has been dropped (req. 7.3).
  - The **Notification System** component, which provides to the central system the dispatch mechanism it needs for sending notifications to both the taxi driver application and the passenger application and fulfill req 7.3.
  - The **Remote Services Interface** component, which enables the taxi driver application to request services to the central system and thus provides the necessary callbacks to satisfy reqs. 7.1 and 7.2.

5.2 Quality Requirements

Chapter 6

References

Appendix A

Hours of work