POLITECNICO DI MILANO
Computer Science and Engineering
Software Engineering 2 Project

MyTaxiService

Integration Test Plan Document

Authors: Greta Ghiotti
Raffaele Malvermi
Mirco Mutti

Version 1.0

Date 18/01/2016

Reference Professor: Mirandola Raffaela

Index

1. Introduction

1.1 Revision History

Version 1.0

Version released on 18/01/2016

1.2 Purpose and Scope

Purpose

This Integration Test Plan Document (ITPD) aims to support other documents related to this project (in particular the Design Document), and guide system tests during the development phase and after it.

In the following paragraphs we describe the approach and the sequence of component integration that we propose, according to the guidelines for the system architecture and its partition among sub-systems and components, given in the DD. Also, we try to list some possible test cases in order to simplify the testing phase and make it more efficient.

Scope

First of all, this document is addressed to the development team or anyone will be appointed to do integration testing on this system, according to its purpose.

Also, it's for the committee of the project, because it could be a useful starting point to estimate the amount of testing costs.

1.3 List of Definitions and Abbreviations

According to definitions used in previous documents, in this paragraph we list some definitions in order to avoid ambiguity.

SynchronousCommunication: it's the name used to define the server component that calls Client methods

AsynchronousCommunication: with this term we mean the server component used to receive incoming requests

Synchronous: It's the Client-side component with the same functionalities of SynchronousCommunication

Asynchronous: it's the Client-side component with the same functionalities of AsynchronousCommuniation

CommunicationClient: with this term we mean both the asynchronous and synchronous communication modules, seen from the Client point of view (only the inner dependencies)

CommunicationServer: with this term we mean both the asynchronous and synchronous communication modules, seen from the Server point of view (only the inner dependencies)

Usability testing: "tests the ease with which the user interfaces can be used. It tests that whether the application or the product built is user-friendly or not.**"**

from http://istqbexamcertification.com/

Compatibility testing: "It tests whether the application or the software product built is compatible with the hardware, operating system, database or other system software or not."

from http://istqbexamcertification.com/

Endurance testing: "involves testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use**"**

from http://istqbexamcertification.com/

1.4 List of Reference Documents

All the concepts and the definition underlying this document are referred to

- Assignment 1 and 2, for MyTaxiService project description;
- RASD, for requirements and goals;
- DD, for component dependencies and function definitions;
- User guides concerning Mockito, Junit, Arquillian and JMeter tools;

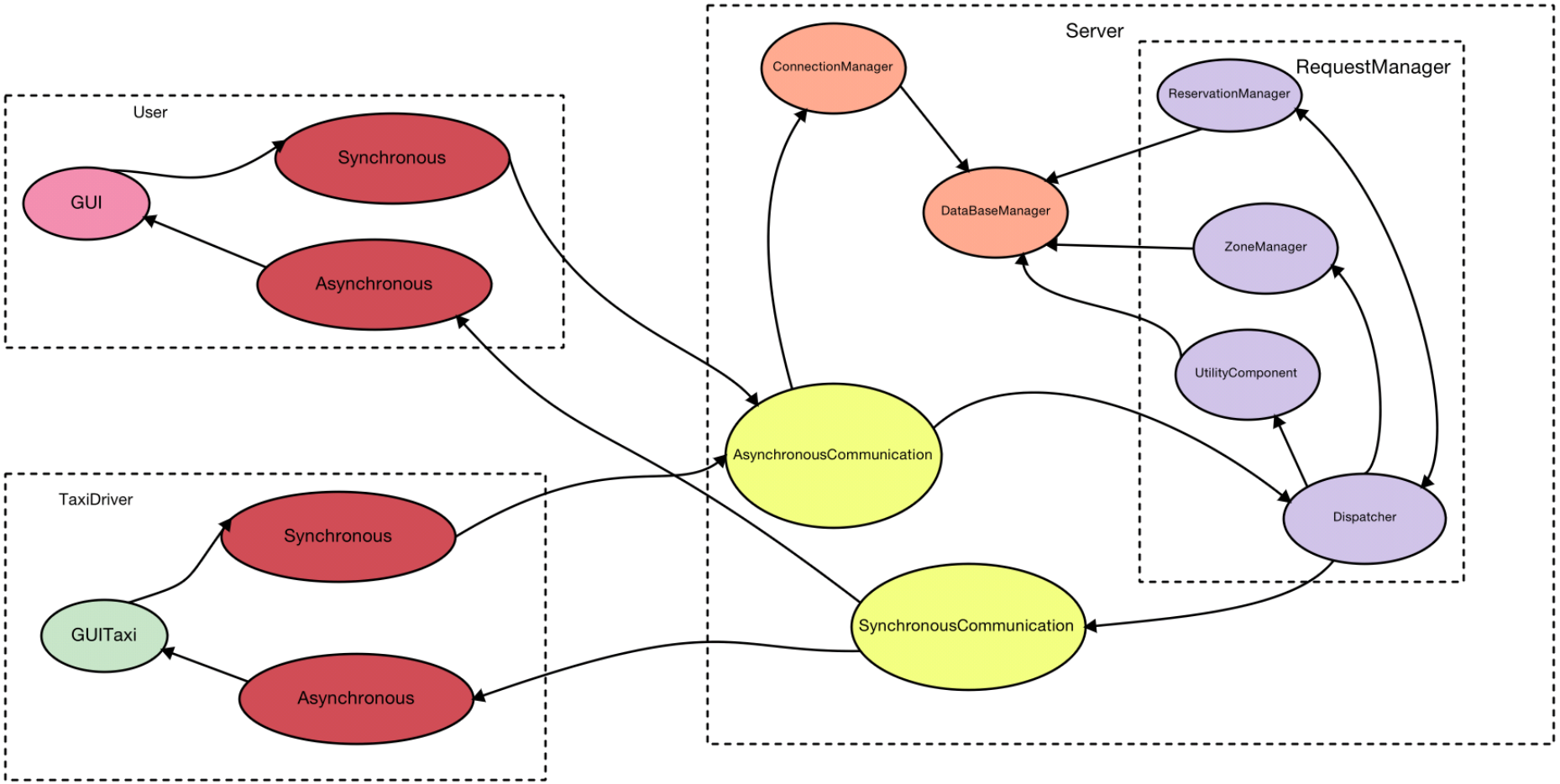
2. Integration Strategy

2.1 Entry Criteria

In the list below we report the necessary pre-conditions that should be met, in order to guarantee the validity of the integration test described in the following chapters.

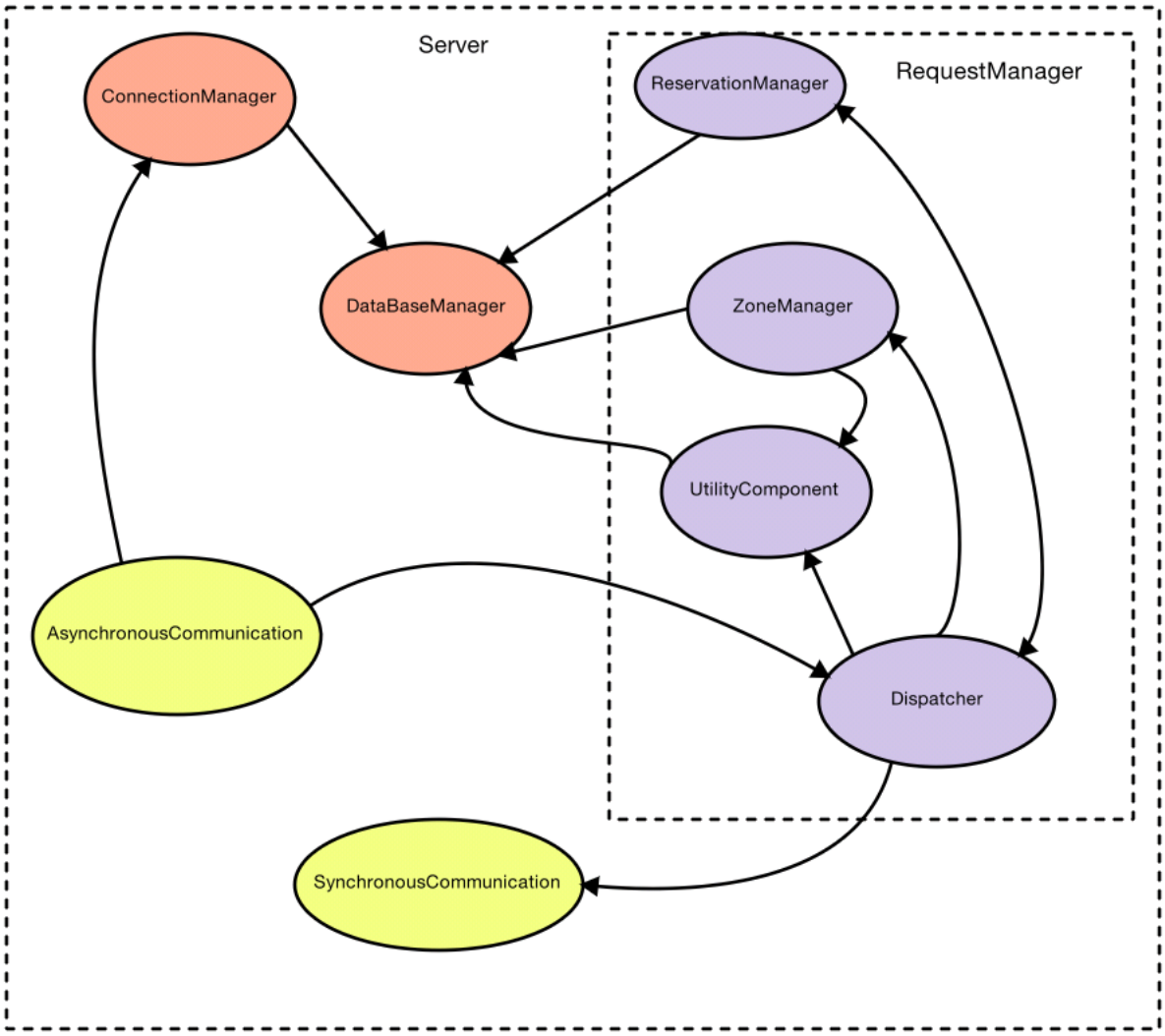
- First of all, the database has to be populated with, at least, a valid set of input points for the SetupZone Algorithm (view the section 3.1 of the Design Document for more details), a set of Users, MyTaxiDrivers and Reservations. The idea focuses on having a set of samples to make all the involved functionalities ready to be tested even if the server has just been set up or some required components haven't been developed yet.
- The DatabaseManager component has to be successfully unit tested, in order to guarantee the correct behavior of each functionality that other components may call during the following steps of the integration test.
- Also the Gui component, both for the UserClient and the TaxiDriverClient, has to be successfully checked with usability testing and each Gui functionality has to be safely accessible during the integration test.
- The access to the services not directly provided by the System, the External Database and the Google Maps Api, should be guaranteed.

2.2 Elements to be Integrated



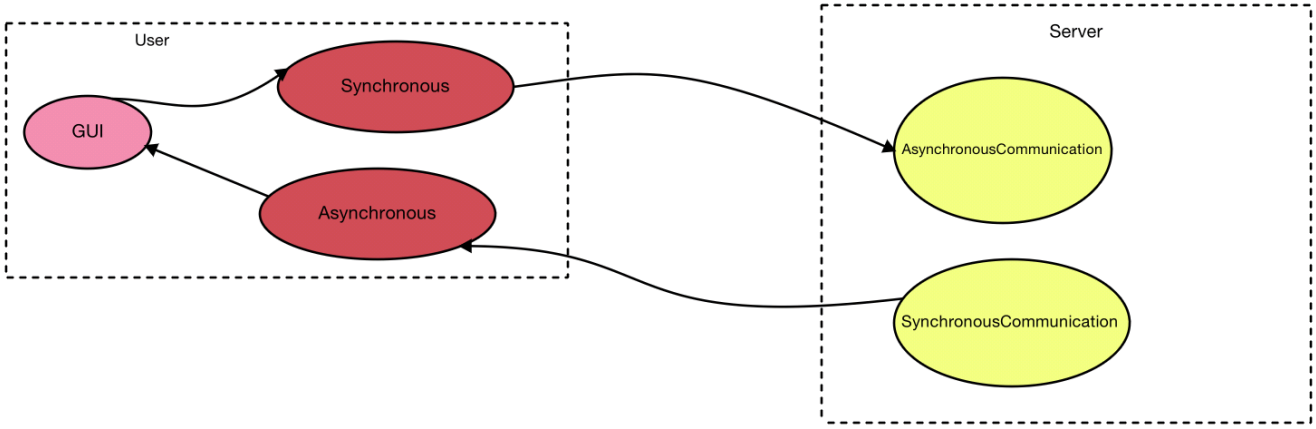
Looking at the dependency graph, describing all the dependencies among components, we can list which elements have to be integrated. One thing to be noticed is that each couple of elements listed in the following tables corresponds to a specific arc in the graph.

Server sub-system



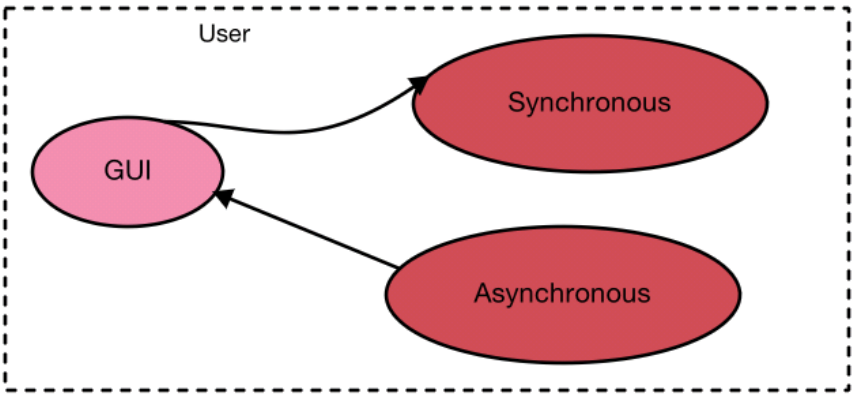
Caller Element → Called Element	Test Case
AsynchronousCommunication → ConnectionManager	ITC12
AsynchronousCommunication → Dispatcher	ITC13
ConnectionManager → DatabaseManager	ITC1
Dispatcher → UtilityComponent	ITC8
Dispatcher → ZoneManager	ITC9
Dispatcher → ReservationManager	ITC10
Dispatcher → SynchronousCommunication	ITC14
ReservationManager → Dispatcher	ITC11
ReservationManager → DatabaseManager	ITC2 , ITC3
UtilityComponent → DatabaseManager	ITC4
ZoneManager → DatabaseManager	ITC6 , ITC7
ZoneManager → UtilityComponent	ITC5

Communication



Caller Element → Called Element	Test Case
SynchronousCommunication → Asynchronous	ITC17
Synchronous → AsynchronousCommunication	ITC18

Client sub-system



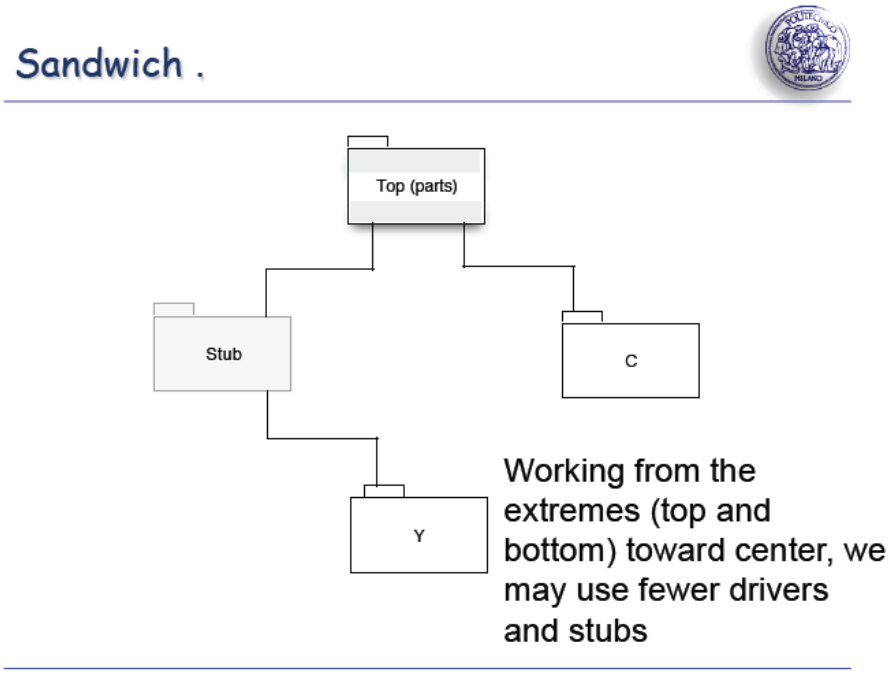
Caller Element → Called Element	Test Case
Asynchronous → Gui	ITC15
Gui → Synchronous	ITC16

2.3 Integration Testing Strategy

The approach that we propose for the integration testing of MyTaxiService system is the Sandwich Testing Strategy: even if the structure of our architecture (described in the DD) suggests a bottom-up point of view (in particular inside the server sub-system), we think that some features of the communication components (as instance, the connection stability and the correctness of the messages exchanged between the Clients and the Server) can be tested apart from other functionalities.

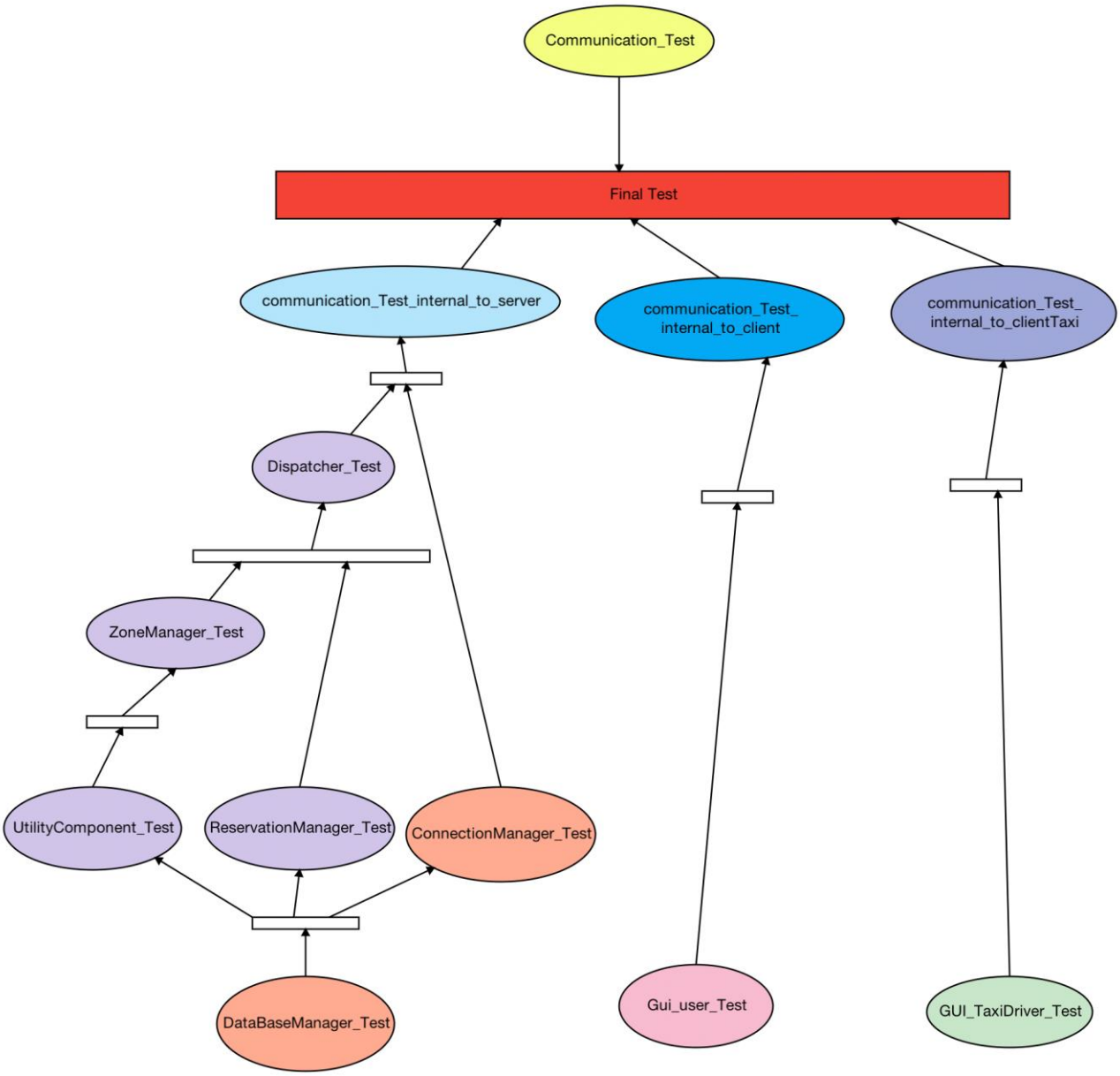
Therefore, the idea is to make simultaneous tests on communication (top-down approach) and the rest of the system, starting from modules which are testable irrespective of others (bottom-up approach).

Since in the Client-Server paradigm, communication between the two actors is one of the most important functionalities of the system, it could be useful and more efficient to verify the correct behavior of the connection at the beginning of the development process. Using this approach, it's possible to test some aspects of the communication before doing the final integration test. Detecting possible errors in the first phases of the development is clearly less costly.



Slide taken from the set "Verification: part II" Software Engineering 2

2.4 Sequence of Component/Function Integration



In the following paragraph we report a detailed description of the integration sequence rationale. To represent in a clearer way the integration steps structure we have included the integration graph above which suggests the order of the steps and when several test phases could be executed simultaneously. Each node in the graph represents a specific test phase, we have also included specific graphical elements (represented with white rectangular shapes) to underline when a test phase is ready to be executed (outcoming

arcs) and when the outcome of a test phase is needed by another one (precedence relations).

Top-down phases

The set of top-down phases is composed by Communication_Test (reported at the top of the graph) that represents an initial test of the basic communication features, such as the connection between Server and Clients and the soundness of the exchanged messages. This test phase can be done independently to the System logic by using a set of stubs to simulate the role of the different subsystems and doing so it could be executed at the beginning of the testing process.

Bottom-up phases

The set of phases is divided into three independent branches, one for each subsystem.

In the Server context the first phase that has to be executed is the DatabaseManager_Test which enables the integration test of UtilityComponent, ReservationManager and ConnectionManager with respect to their dependencies with the DatabaseManager. Then it's possible to go on with the integration between ZoneManager and UtilityComponent that finally enables (together with the previous phases) the test of the Dispatcher, completing in this way the RequestManager integration test. At this point we have the final phase of the Server integration: the dependencies between the CommunicationServer module and both the RequestManager and the ConnectionManager should be tested.

In the Clients context (both TaxiDriverClient and UserClient) the Gui_Test has to be executed as first step and then the integration test between the Gui and CommunicationClient.

Final phase

When both bottom-up and top-down phases terminate, the last test step that should be executed is the integration between the distinct subsystems: Server-TaxiDriverClient-UserClient. We have decided to manage this integration in a single phase to solve the interlaced dependencies between the subsystems: we have considered this way more efficient than the solution with three different integration (Server-TaxiDriverClient, Server-UserClient and then Server-TaxiDriverClient-UserClient) that needs the implementation of a driver and a stub for each Client.

3 Individual Steps and Test Description

All the Integration Test Cases (ITC) are listed in the table below. For every ITC we have specified an Identifier, the dependency tested in the case, the type of test, all the preconditions needed before testing (including the precedence between ITCs and required unit tests on components), the desired result (that states when the test is passed or not) and all the corresponding requirements described in the RASD (written in this way: **G***index_Of_The_Goal*.**R***index_Of_The_Requirement_In_The_Goal_List*).
One thing to be noticed is the high level of detail used to describe the desired outcomes: all the methods checked in the test cases have a list of post-conditions, described in a very detailed way in the Design Document. Testing a function has its mean in testing the satisfaction of all the post-conditions of that specific function.

ID	Dependency	Type	Precondition	Desired Output	Requirements
ITC1	ConnectionManager → DatabaseManager	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of log in and sign up functions	G1.R2, G2.R2, G7.R2, G12.R2
ITC2	ReservationManager → DatabaseManager	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of store reservations and delete reservations functions	
ITC3	ReservationManager → DatabaseManager	**STRESS TEST**	ITC2 successfully completed	Correct recovery of data after server restart	
ITC4	UtilityComponent → DatabaseManager	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of computeFees function	G6.R6 G13.R1 G14.R1

ITC5	ZoneManager → UtilityComponent	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of setupZones function	
ITC6	ZoneManager → DatabaseManager	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of setupZones function	
ITC7	ZoneManager → DatabaseManager	**STRESS TEST**	ITC6 successfully completed	Correct recovery of data after server restart	
ITC8	Dispatcher → UtilityComponent	**FUNCTIONAL TEST**	ITC4 successfully completed and Dispatcher unit tested	Correct behavior of all methods included in the UtilityInt interface provided by Utility Component	G3.R5 G6.R2 G6.R5 G6.R6 G13.R1 G14.R1
ITC9	Dispatcher → ZoneManager	**FUNCTIONAL TEST**	ITC5, ITC6 and ITC7 successfully completed and Dispatcher unit tested	Correct behavior of findZone function	
ITC10	Dispatcher → ReservationManager	**FUNCTIONAL TEST**	ITC2 successfully completed and Dispatcher unit tested	Correct behavior of AddReservation, CheckReservation, ListReservations and DeleteReservation functions	G4.[R2-R4] G5.R2
ITC11	ReservationManager → Dispatcher	**FUNCTIONAL TEST**	ITC2 successfully completed and Dispatcher unit tested	Correct behavior of ServeReservation and SendList functions	G4.R5
ITC12	Asynchronous Communication → ConnectionManager	**FUNCTIONAL TEST**	ITC1 successfully completed	Correct behavior of Log in and sign up functions of ConnectionInt interface provided by ConnectionManager	G1.R2, G2.R2, G7.R2, G12.R2

ITC13	Asynchronous Communication → Dispatcher	**FUNCTIONAL TEST**	ITC10, ITC11 successfully completed	Correct behavior of addRequest function	
ITC14	Dispatcher → Synchronous Communication	**FUNCTIONAL TEST**	ITC10, ITC11 successfully completed	Correct behavior of • RequestProposal • SharerAddition • SharingRideDetail • RouteMessage functions	G3.[R3-R7] G6.R5 G6.R6 G9.R2 G10.R2
ITC15	Asynchronous → Gui	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of methods included in AsynchronouseInter (in taxiDriver GUI) and AsyncInterface (in User Gui)	G6.R5 G6.R7 G9.[R1-R3]
ITC16	Gui → Synchronous	**FUNCTIONAL TEST**	Both components unit tested	Correct behavior of all methods provided by SyncInterface (of taxiDriver GUI and User GUI)	G1.R1 G2.R1 G3.[R3-R7] G4.R1 G5.R1 G6.R1 G7.R1 G8.R1 G9.R3 G12.R1
ITC17	Synchronous Communication → Asynchronous	**FUNCTIONAL TEST**		Correct exchange of messages (integrity of the content) and connection stability	
ITC18	Synchronous → Asynchronous Communication	**FUNCTIONAL TEST**		Correct exchange of messages (integrity of the content) and connection stability	

ITC19	Server subsystem	**LOAD TEST**	All tests on server components successfully completed	Huge amount of requests, given in a short period, successfully managed by the Server. (Requests should belong to all the possible types and the amount of samples depends on the dimensions of the city)	G3.R1
ITC20	Whole System	**ENDURANCE TEST**	The final integration test has been successfully completed	Correct behavior of the system after a long working period	G3.R1
ITC21	Whole System	**COMPATIBILITY TEST**	The final integration test has been successfully completed	The application works correctly on all the devices described in RASD and DD documents	
ITC22	Whole System	**PERFORMANCE TEST**	The final integration test has been successfully completed	The response time of all the system functionalities satisfy the non-functional requirements listed in the RASD document	G3.R8 G4.R6

4. Tools and Test Equipment Required

The purpose of the following paragraph is to report the set of tools and equipment that may guide or help the integration test that we have described in the previous chapters. In the Design Document we have never stated, in order to keep as high as possible the level of abstraction, which programming language has to be used in the following implementation phase. For this reason we can't explicitly report a list of tools that will be definitely used, but only a general description of the equipment needed features and a set of suggestions in case of JEE implementation (but, again, it won't be a constraint on the development process).

In order to satisfy the Entry Criteria, reported in the paragraph 2.2, it could be convenient to use unit testing tools to support the manual testing; for instance the combination of JUnit and Mockito (it's necessary to exploit mocks simulating the components related with the unit to-be-tested) could be useful to test the correct behavior of the DatabaseManager and the Gui.

To accomplish the main part of the integration test, the set of functional test cases (which are reported in the chapter above), it could be useful to have a tool supporting the integration, in order to reduce the manual implementation of stubs and drivers. We suggest a tool like Arquillian in case of JEE implementation and, again, JUnit to accomplish the unit testing required as test cases precondition.

To accomplish the test cases of type "performance" and "load", an application like JMeter could be used (only for JEE) or in general a tool that allows the developers to simulate heavy load on the Server, in order to discover the System upper bounds and also to analyze performance in critical conditions.

5. Program Stubs and Test Data Required

In the following table we report the set of drivers and stubs related to each integration step. In general: bottom-up test phases requires drivers, top-down requires stubs.

With the notation [*Component_name*]Driver we mean a driver that calls the function of the specified component. For each integration step the relative Driver needs to call only the function involved in the dependency between the components to-be-integrated; we might say that we use the driver to "explore" the dependency.

In the same way, we have denoted each stub as [*Component_name*]Stub. We have to use stubs, especially in the integration steps with communication involved, to simulate the return value of any calls related to a component not integrated yet.

In the first row we refer to the Communication_Test that we have already described in the chapter 2.4 (Sequence of component/Function Integration), the dependencies involved in this integration step are (SynchronousCommunication→Asynchronous) and (Synchronous→AsynchronousCommunication).

IntegrationStep	Drivers	Stubs
Communication_Test	Communication_Test_Driver	A set of stub simulating the behavior of Server, UserClient and TaxiDriverClient related to the basic communication.
ConnectionManager → DatabaseManager	ConnectionManagerDriver	
ReservationManager → DatabaseManager	ReservationManagerDriver	

UtilityComponent → DatabaseManager	UtilityComponentDriver	
ZoneManager → UtilityComponent	ZoneManagerDriver	
ZoneManager → DatabaseManager	ZoneManagerDriver	
ZoneManager → DatabaseManager	ZoneManagerDriver	
Dispatcher → UtilityComponent	DispatcherDriver	
Dispatcher → ZoneManager	DispatcherDriver	
Dispatcher → ReservationManager	DispatcherDriver	
ReservationManager → Dispatcher	ReservationManagerDriver	

AsynchronousCommunication → ConnectionManager	AsynchronousCommunication Driver	
AsynchronousCommunication → Dispatcher	AsynchronousCommunication Driver	
Dispatcher → SynchronousCommunication	DispatcherDriver	SynchronousCommunication Stub
Asynchronous → Gui	AsynchronousDriver	
Gui → Synchronous	GuiDriver	SynchronousStub

Number of hours

We have spent 18 hours per person to redact the version 1.0 of the ITPD.

Greta Ghiotti: 18 hours

Raffaele Malvermi: 18 hours

Mirco Mutti: 18 hours