Integration Test Plan

Matteo Maria Fusi, Matteo Locatelli

0.1 Document version

- 1.0 - 21/1/2016: first release

0.2 Time Spent

- Matteo M. Fusi: ~8h
- Matteo Locatelli: ~8h

0.3 Software used

- Astah Professional (used to design the graphs)
- Lyx (used to write this document)

Contents

1 Introduction

1.1 Purpose and scope

The purpose of this document is to present the plan for testing the interfaces between the several components that implements the mytaxi application proposed on referred RASD. The proposed solution is based on the Desing Document referenced. The aim of this solution is to test the integration in a fast way keeping the required testing scaffolding as short as possible using a thread integration strategy on the possible several devices.

1.2 List of definitions and abbreviations

Also see referenced RASD for definitions and abbreviations.

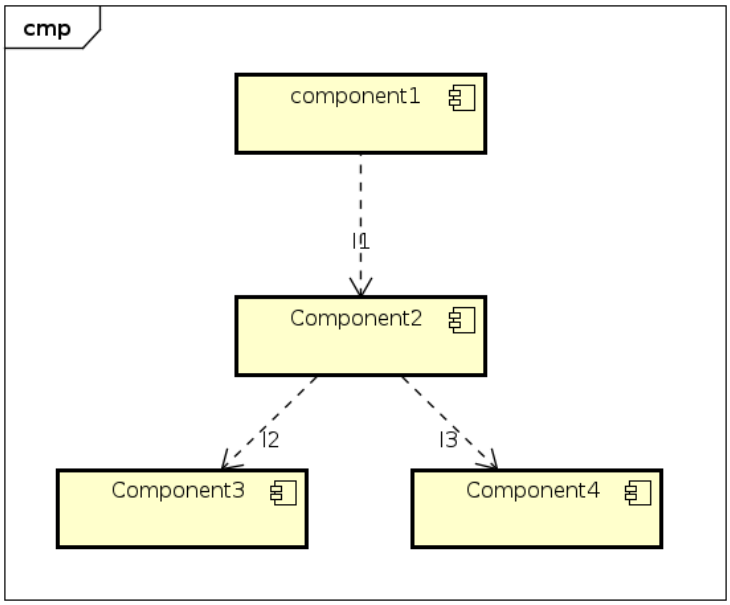
1.3 List of referenced documents

- RASD v 1.0.1 : link
- Design Document v 1.1.0 link

2 Integration Strategy

2.1 How to read diagrams

The arrows indicate an integration test identified by the notation I followed by a number (for example *I1*, *I2*...). The direction of the arrow indicate a dependency between integration tests; which means that if you want to execute an integration test you need to execute all the integration tests that start from the component at the head of the arrow. For example, reference to the below diagram. In order to test the integration between Component1 and Component2 (identified by the integration test I1) we must execute the integration tests I2 and I3 before I1.

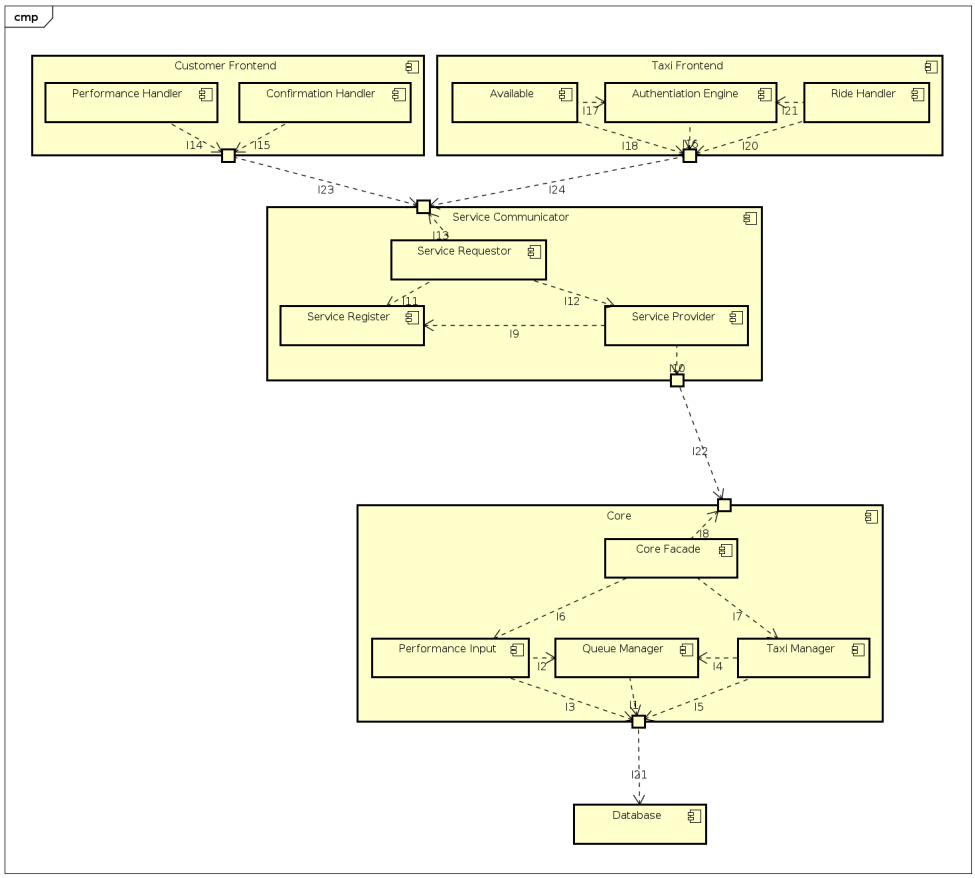


2.2 Entry criteria

We assume that every component is working properly: this means that it has been tested individually.
Furthermore we assume that the integration between the GoogleMapsService and the components of our system is working because of the fact that it is an external component that we chose to use.

2.3 Elements to be integrated

In the following pictures there are all the system components that have to be integrated and the arrows represent the order in which the components have to be integrated.

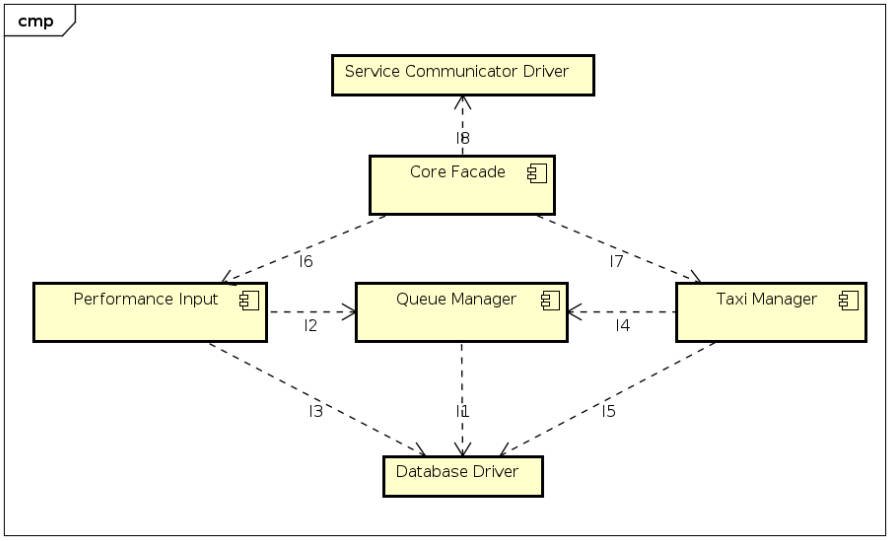


2.4 Integration testing strategy

We decided to use a **bottom up** approach to integrate all the components of the system, because the architecture is highly modular and in this way we can guarantee an easier and faster integration procedure: the system is split in three subsystems (front end, service communicator and core) so the integration of these three subsystems can be done simultaneously. In order to test also the communication between this systems, we use drviers that can simulate all the needed functionalities of the components that the analyzed component is connected to.

2.5 Sequence of integration

2.5.1 Core



Test Case Identifier	I1
Test Item(s)	Queue Manager → Database Driver
Input Specification	Query that has to be sent to the database to retrieve needed information.
Output Specification	Data needed by the Queue Manager.
Environmental Needs	Database Driver

Test Case Identifier	I2
Test Item(s)	Performance Input → Queue Manager
Input Specification	Request to the Queue Manager to manage the taxi queues to answer the customer's request.
Output Specification	Answer to the customer about his request.
Environmental Needs	I1 succeeded

Test Case Identifier	I3
Test Item(s)	Performance Input → Database Driver
Input Specification	Query that has to be sent to the database to retrieve needed information.
Output Specification	Data needed by the Performance Input.
Environmental Needs	Database Driver

Test Case Identifier	I4
Test Item(s)	Taxi Manager → Queue Manager
Input Specification	Provide the Queue Manager component with the needed information about taxis.
Output Specification	Proper queue management.
Environmental Needs	I1 succeeded

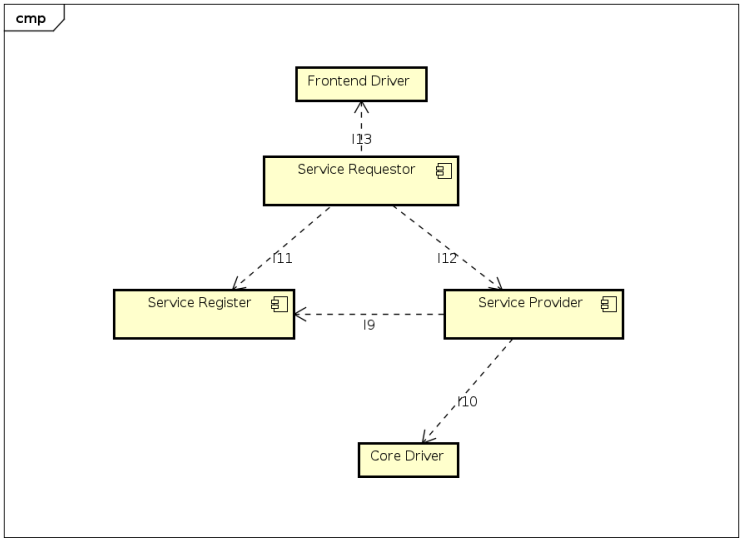
Test Case Identifier	I5
Test Item(s)	Taxi Manager → Database Driver
Input Specification	Query that has to be sent to the database to retrieve needed information.
Output Specification	Data needed by the Taxi Manager.
Environmental Needs	Database Driver

Test Case Identifier	I6
Test Item(s)	Core Facade → Performance Input
Input Specification	Provide the Performance Input component with a interface used to communicate with external components.
Output Specification	The Performance Input component communicates properly with other components.
Environmental Needs	I2 and I3 succeeded

Test Case Identifier	I7
Test Item(s)	Core Facade → Taxi Manager
Input Specification	Provide the Taxi Manager component with a interface used to communicate with external components.
Output Specification	The Taxi Manager component communicates properly with other components.
Environmental Needs	I4 and I5 succeeded

Test Case Identifier	I8
Test Item(s)	Core Facade → Service Core Driver
Input Specification	Request by the Core Facade to communicate with components outside the Core.
Output Specification	The Core Facade communicated properly with other components.
Environmental Needs	Service Core Driver

2.5.2 Service Communicator



Test Case Identifier	I9
Test Item(s)	Service Provider → Service Register
Input Specification	Service Provider needs to access the services available.
Output Specification	Service Provider can access the services that can be used by customers or taxi drivers.
Environmental Needs	None

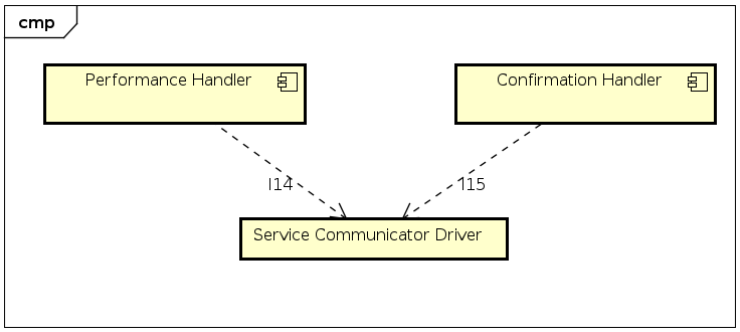
Test Case Identifier	I10
Test Item(s)	Service Provider → Core Driver
Input Specification	The Service Provider must communicate properly with the Core.
Output Specification	The Service Provider establishes a communication with the system Core.
Environmental Needs	Core Driver

Test Case Identifier	I11
Test Item(s)	Service Requestor → Service Register
Input Specification	The Service Requestor needs to access the services available to customers or taxi drivers.
Output Specification	The Service Requestor accesses all the needed services.
Environmental Needs	None

Test Case Identifier	I12
Test Item(s)	Service Requestor → Service Provider
Input Specification	The Service Requestor asks to the Service Provider to give the customer or the taxi driver client access to the needed service.
Output Specification	The customer or the taxi driver client can access the needed service.
Environmental Needs	I9 and I10 succeeded.

Test Case Identifier	I13
Test Item(s)	Service Requestor → Frontend Driver
Input Specification	The Service Requestor needs to communicate to the Frontend, that is the subsystem which asks for services.
Output Specification	The Service Requestor communicates properly with the Frontend and can give it access to the needed services.
Environmental Needs	Frontend Driver

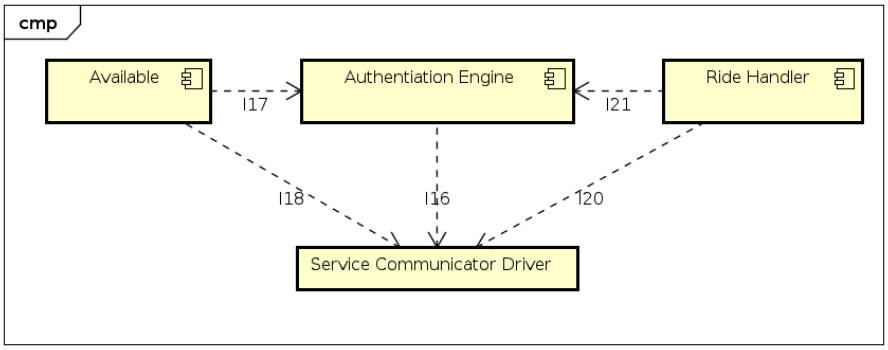
2.5.3 Customer Frontend



Test Case Identifier	I14
Test Item(s)	Performance Handler → Service Frontend Driver
Input Specification	The Performance Handler (which belongs to the Customer Frontend) needs to properly communicate with the Service Communicator to allow the customer to ask for performances.
Output Specification	The Customer Frontend allows the customer to ask for performances using the Performance Handler which communicates with the Service Communicator.
Environmental Needs	Service Frontend Driver

Test Case Identifier	I15
Test Item(s)	Confirmation Handler → Service Frontend Driver
Input Specification	The Confirmation Handler (which belongs to the Taxi Frontend) needs to properly communicate with the Service Communicator to allow the taxi driver to confirm his acceptance of a ride.
Output Specification	The Taxi Frontend allows the taxi driver to confirm his availability for a ride using the Confirmation Handler which communicates with the Service Communicator.
Environmental Needs	Service Frontend Driver

2.5.4 Taxi Frontend



Test Case Identifier	I16
Test Item(s)	Authentication Engine → Service Frontend Driver
Input Specification	The Authentication Engine needs to communicate with the Service Communicator to check if the authentication of the taxi driver is successfull.
Output Specification	The authentication of the taxi driver is confirmed or rejected.
Environmental Needs	Service Frontend Driver

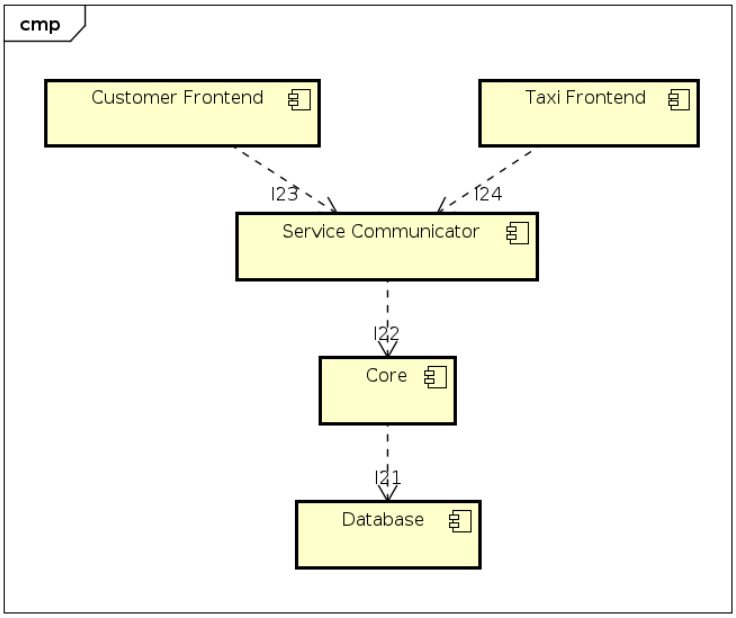
Test Case Identifier	I17
Test Item(s)	Available → Authentication Engine
Input Specification	The Available component needs to change the current status of the taxi driver (available or not).
Output Specification	To change the available status, the taxi driver must be logged into the system.
Environmental Needs	I16 succeeded

Test Case Identifier	I18
Test Item(s)	Available → Service Frontend Driver
Input Specification	The Available component needs to communicate with the Service Communicator to change the taxi driver's status.
Output Specification	The current status of the driver is properly changed.
Environmental Needs	Service Frontend Driver

Test Case Identifier	I20
Test Item(s)	Ride Handler→ Service Frontend Driver
Input Specification	The Ride Handler component needs to communicate with the Service Communicator to properly manage the ride.
Output Specification	The ride is managed correctly.
Environmental Needs	Service Frontend Driver

Test Case Identifier	I21
Test Item(s)	Ride Handler→ Authentication Engine
Input Specification	The Ride Handler component needs to manage the ride.
Output Specification	To guarantee the management of the ride, the taxi driver must be logged into the system.
Environmental Needs	I16 succeeded

2.5.5 System Integration



Test Case Identifier	I21
Test Item(s)	Core → Database
Input Specification	The Core use a query to retrieve the needed information from the database.
Output Specification	The Core properly obtain the needed information.
Environmental Needs	None

Test Case Identifier	I22
Test Item(s)	Service Communicator→ Core
Input Specification	The Service Communicator needs to communicate with the Core to access the functionalities of the system.
Output Specification	The Service Communicator properly interacts with the Core.
Environmental Needs	I21 succeeded

Test Case Identifier	I23
Test Item(s)	Customer Frontend→ Service Communicator
Input Specification	The Customer Frontend needs to communicate with the Service Communicator to guarantee to customers all the functionalities of the system.
Output Specification	The Customer Frontend properly interacts with the Service Communicator allowing customers to use the system.
Environmental Needs	I22 succeeded

Test Case Identifier	I24
Test Item(s)	Taxi Frontend→ Service Communicator
Input Specification	The Taxi Frontend needs to communicate with the Service Communicator to guarantee to taxi drivers all the functionalities of the system.
Output Specification	The Taxi Frontend properly interacts with the Service Communicator allowing taxi drivers to use the system.
Environmental Needs	I22 succeeded

3 Individual Steps and Test Description

Test Procedure Identifier	TP1
Purpose	This test procedure verifies if the Core: • can properly communicate with the Service Communicator via the Core Facade • can receive a performance request as an input • can update the taxi status using the Taxi Manager component • can properly manage the queues when a customer asks for a performance and when a taxi driver moves from a zone to another one • can correctly access the database to retrieve information needed in order to guarantee the system functionalities
Procedure Steps	Execute I2 - I3 - I5, then I2 - I4, then I6 - I7 and finally I8

Test Procedure Identifier	TP2
Purpose	This test procedure verifies if the Service Communicator: • can properly communicate with the Core using the Service Provider component • allow the Service Provider and Service Requestor to access the Service Register to retrieve the services needed to guarantee the system correct functionality • allow the Service Requestor to ask for the access to services to the Service Provider • can properly communicate with the two frontends (customer's and taxi driver's) using the Service Requestor component
Procedure Steps	Execute I10, then I9 - I11 - I12 and finally I13

Test Procedure Identifier	TP3
Purpose	This test procedure verifies if the Customer Frontend: • allow the Performance Handler to properly communicate with the Service Communicator, in order to let the customer ask for performances • allow the Confirmation Handler to properly communicate with the Service Communicator, in order to let the customer confirm a ride if it's the case
Procedure Steps	Execute I14 - I15

Test Procedure Identifier	TP4
Purpose	This test procedure verifies if the Taxi Frontend: • allow the Available component to properly communicate with the Service Communicator, in order to let the taxi driver change his current state • allow the Ride Handler component to properly communicate with the Service Communicator, in order to let the taxi driver manages the ride • allow the Authentication Engine component to properly communicate with the Service Communicator, in order to let the taxi driver change log into the system • allow the taxi driver to perform the two previous operations only if he's logged into the system
Procedure Steps	Execute I16 - I18 - I20, then I17 - I21

Test Procedure Identifier	TP5
Purpose	This test procedure verifies if the whole system: • allow the Core to communicate with the database to retrieve the needed information • allow the Core, the Taxi Frontend and the Customer Frontend to communicate using the services provided by the Service Communicator
Procedure Steps	Execute I21, then I22 and finally I23 - I24

4 Tools and Test Equipment Required

In order to implement the integration tests explained in the previous section, we decided to use the combination Arquillian + JUnit: we use JUnit to generate a test unit for each test case explained in section 2, then we use Arquillian to simulate a component when needed. A test suite will be created for each test procedure. We decided to use Aquillian because our system structure is very modular and the feature of Arquillian to simulate components and to integrate them can be very useful in our case.

5 Program Stubs and Test Data Required

In order to properly execute the integration of the components of the system and referred to the system diagram in section 2.3, the following drivers are needed:

- Frontend driver: it simulates ServiceFrontendPort from the Service Communicator point of view.
- Service Core driver: it simulates CoreServicePort from the Core point of view.
- Service Frontend driver: it simulates FrontendServicePort from the Frontend point of view.
- Core driver: it simulates ServiceCorePort from the Core point of view.
- Database driver: it simulates CoreDatabasePort from the Core point of view.

Trivially every driver simulates the behaviour of the corresponding component.