# Integration test document - v1.0

Gianpaolo Branca Luca Butera Andrea Cini

POLITECNICO
MILANO 1863

## Contents

# 1 Introduction

## 1.1 Purpose

This document describe the strategy for testing the integration of components of the PowerEnjoy system, designed in the Design Document presented earlier. It describe the strategy and the tools necessary to fulfill the task.

## 1.2 Scope

After the design and the development of the components is now the time to put everything together and verify that everything works. This document will be presented to the testing team.

## 1.3 Definitions

- **Bottom-up:** a strategy that starts from the lower level to the higher level components.
- **Stub:** a temporary substitute component that simulate the behaviour of the one that is not already integrated. It is used to test other components that require interaction with this one.
- **Unit test:** a way to test a single function with assertions
- **Arquillian:** a framework for JEE for the integration tests

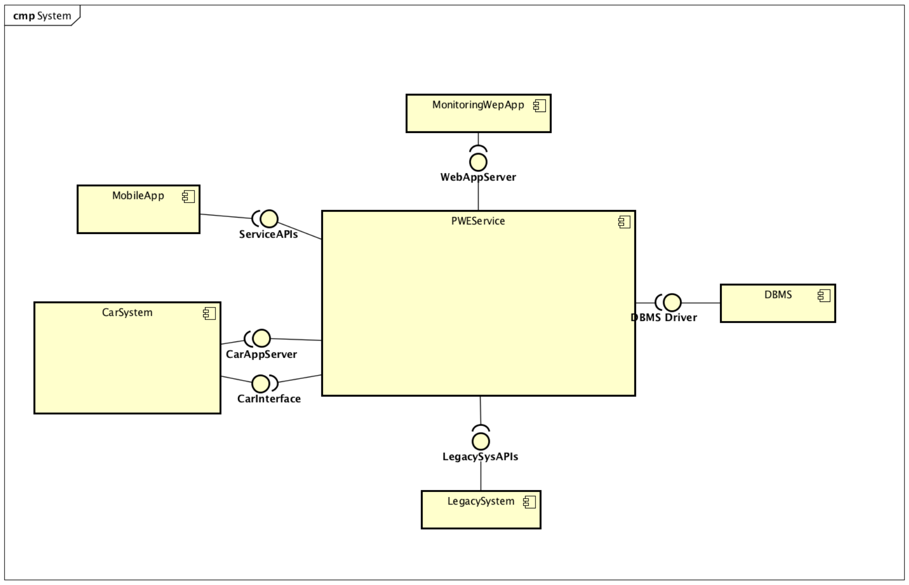
## 1.4 Abbreviations

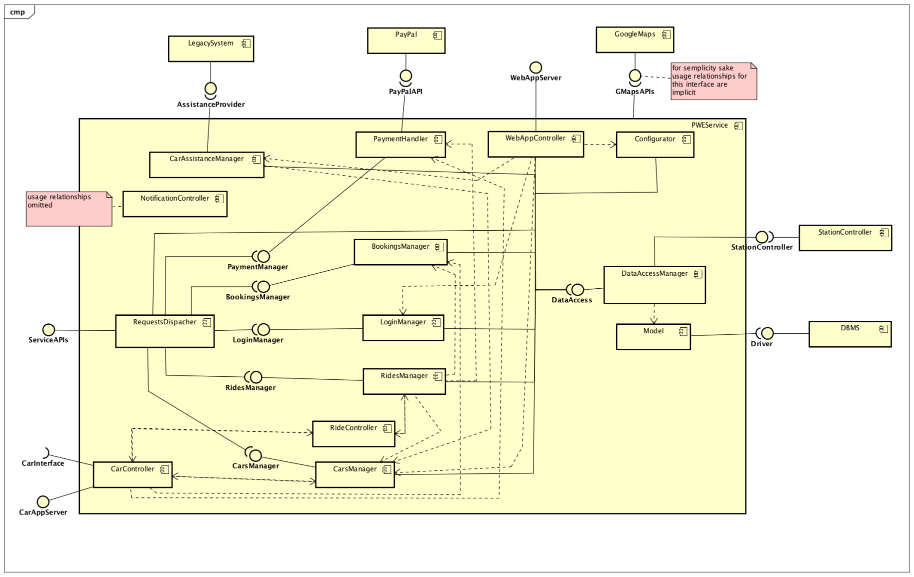
- **GUI:** Graphical unit interfaces
- **GPS:** Global position system
- **N/A:** Not applicable

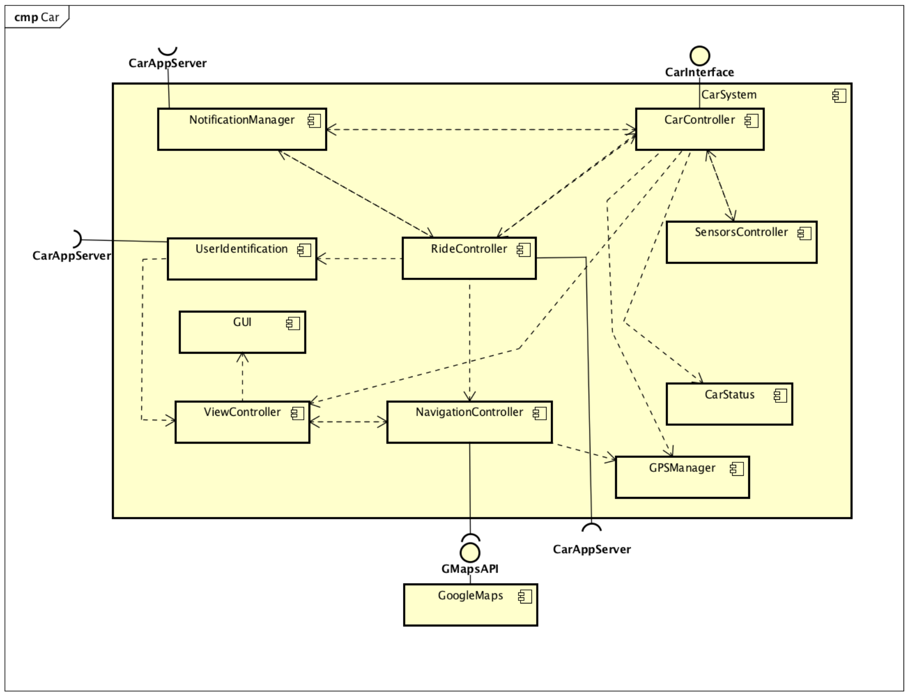
## 1.5 Reference documents

- Specification document
- RASD
- Design document
- Integration test document sample on Beep

## 2 Integration strategy







## 2.1 Entry criteria

Before starting the integration test activity a solid version of both the RASD and the DD must have been provided so that the interactions between the components are reasonably clear and well defined. In our belief, for the integration tests to be effective and meaningful, the followings points have to be reached:

- The data access layer must have been fully developed (**DataAccessManager**, **Model**, **StationController**) and the DBMS integrated.
- At least 70% of the **PWEService** and **CarSystem** components functionalities must be implemented.
- The **MobileApp** and the **MonitoringWebApp** components development is not critical for integration purpose (they offer presentation functionalities), but it must have reached a point in which they provide a way to call every service provided by the central node through its interfaces.
- Agreements with the external services providers must have been reached and the services must be available.
- A reliable stub for the LegacySystem must have been produced.

Every single component must be unit tested with at least an 80% branch coverage and its main functionalities fully developed before going into the integration test phase.

## 2.2 Elements to be integrated

Starting from the High level component view of our system, the subsystems to be considered for integration testing are integrated are:

- The **PWEService** the **CarSystem** are the most critical components to be tested.
- The **MonitoringWebApp** and the **MobileApp**, as already mentioned, are not essential for the integration test progression as they can be easily substituted by simple drivers until the system integration test.
- Assuming that the **LegacySystem** works correctly a stub seems rather good for our purposes.
- The integration of the **DBMS** should be straightforward with the usage of the JPA framework.

## 2.3 Integration test strategy

For our integration test we will use a mixed approach because, while going always bottom-up or always top-down will give us the benefit of a simpler integration plan, a more dynamic approach based on the specific group of components under consideration will allow us to test in more effective and meaningful way. At the start we will focus on the integration of the subcomponents of the two main subsystem that we have identified that will be performed in parallel:

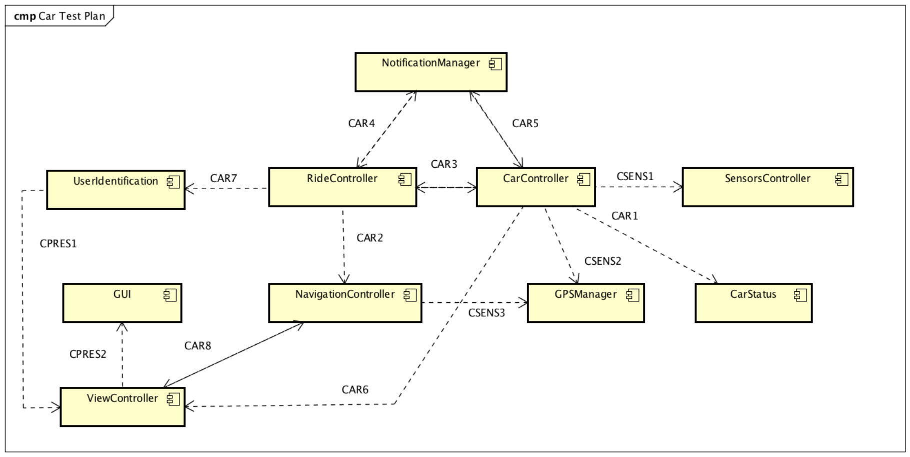
- **CarSystem**: At first the tests will be carried out on a virtual machine to simulate the car environment with a stub and a driver for the **SensorsController**, later on, after the unit test of the sensor controller in a real vehicle, the whole Car Application will be deployed on a car and the integration with the **SensorsController** properly tested.

## 2.4 Sequence of component integration

### 2.4.1 Software integration sequence

In this section we are going to analyze the tests that are needed for the subcomponents of each subsystem. To describe the sequence in which the components will be integrated, we will use diagrams of components linked with labeled and oriented arrows. The label will identify the flow of tests and the order within the flow.

### Car System



The **CarSystem** sub-components integration will happen in two steps and for both of the two steps a stub and a driver for the **PWEService** will be needed:

- Step 1, performed in a virtual machine environment: We will follow two parallel flows of test:
  - CAR testes: using stubs for the **SensorsController** and **GPSManager** we will test the critical components of the subsystem in the order highlighted in the diagram(arrows with CAR labels). These components are the most domain specific ones and their integration has to be considered with particular care.
  - CPRES testes: tests for the integration on the components of the presentation layer of the car application, they can be carried out in parallel from the CAR testes.
- Step 2 :
  - After the tests of the step 1 has been performed the application will be deployed in a real vehicles and the integration of the **SensorsController** and **GPSManager** will be tested with the possibility to manipulate the car to simulate mechanical problems(CSENS testes).

### 2.4.2 Subsystem integration sequence

we have 4 subsystem in our system:

- The server
- The Car system
- The mobile application
- The operators web application

After the integration test of the server components, the system will be integrated with the operators web application, the car system, and lastly with the web application. This is done because integrating firstly the web app can help discovering problems in the next integrations, since it is itself a control application. The mobile application is the last component because is the one we are going to deploy on smartphone stores. To test its functionality everything in the back end have to be already integrated.

## 3 Individual steps and test description

---

**Test Case C1**

**Test case identifier:** C1 **Test items:** CarController -> Sensors **Environmental Needs:** N/A

## 4 Tools and test equipment required

- Arquillian for automatic tests
- Manual testing for the mobile application, to check if it is responsive and easy to use. The app will be tested on many different phone with different android versions and display diagonal

## 5 Program stubs and test data required

### 5.1 Stubs

**Station Controller**

**Usages:**
**Description:** this stub is used to test the informations retrieved by the Recharging areas, because using a real net of recharging station would be very expensive and would slow down the test.

**Car Service**

**Usages:**
**Description:** this stub is used to test the server independently from the the cars, so that the testing for the systems can proceed in parallel.

**Car App server**

**Usages:**
**Description:** this stub is used to test the car system independently from the the server, so that the testing for the car can proceed in parallel.

**Legacy System**

**Usages:**
**Description:** this stub is used for simulate the forwarding of a request to the legacy server, because sending many fake requests can interfere with the other tasks of the company.

**PayPal**

**Usages:**
**Description:** this stub simulate the interaction with PayPal API, avoiding the usage of real money transfers for testing.

**Sensor Controller**

**Usages:**
**Description:** this stub simulate the interaction with the car sensors, avoiding to break cars each time we want to test if an assistance request is properly sent.

**GPS Manager**

**Usages:**
**Description:** this stub simulate the interaction with the GPS antenna and return a fake positions, simulating rides. Obviously driving a real car for integration testing is very expensive in terms of money and time.

## 5.2 Test Data

We will populate the data base with fake users, cars, and safe areas. they will be generated in an automatic way with Arquillian

## Effort spent