POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016–2017
Software Engineering 2: "PowerEnJoy"
Integration Test Plan Document

Pietro Ferretti, Nicole Gervasoni, Danilo Labanca

January 15, 2017

Contents

1 Introduction

1.1 Revision History

1.2 Purpose and Scope

The purpose of this document is to provide a complete description of the integration testing plans for PowerEnjoy. It will focus on testing the proper behavior of the software by checking the interoperability between its components. It is intended for developers and for any team member involved in the testing process.

The aim of this project is to specify in detail a new digital management software for PowerEnJoy, a car-sharing service that employs electric cars only.

PowerEnjoy will offer a very valuable service to its users, letting them borrow cars to drive around the city freely, as an alternative to their own vehicles and public transport. Among the advantages of using PowerEnJoy we can note being able to find available cars in any place that is served by our system and having dedicated spots to park in (namely, PowerEnJoy's power grid stations). Furthermore, thanks to the fact that all the cars that we provide are electrically powered, PowerEnJoy is also very environmentally friendly.

1.3 List of Definitions and Abbreviations

1.3.1 Definitions

- *Guest*: a person that is not registered to the system.
- *User*: a person that is registered to the system. Users can log in to the system with their email or username and their password. Their first name, last name, date of birth, driving license ID are stored in the database.
- *Safe area*: a location where the user can park and leave the car. Users can end their ride and park temporarily only in these locations. The set of safe areas is predefined by the system.
- *Power grid station*: a place where cars can be parked and plugged in. While a car is plugged in a power grid station its battery will be recharged. Power grid stations are by definition safe areas.
- *Available car*: a car that is currently not being used by any user, and has not been reserved either. Available cars are in good conditions (not dirty nor damaged) and don't have dead batteries.
- *Reservation*:

  - the operation of making a car reserved for a user, i.e. giving permission to unlock and use the car only for that user, forbidding reservations by other users.
  - the time period between the moment a reservation is requested and the moment the user unlocks the car, or the reservation is canceled.

- *Ride*: the time period from the moment a reserved car is unlocked to the moment the user notifies that he wants to stop using the car and closes all the doors. A ride doesn't stop when a car is temporarily parked, but continues until the user chooses to leave the car definitely.
- *Temporary parking*: the act of parking a car in a safe area and, after notifying the system, locking it and leaving it for a finite amount of time. The user that does this retains the right to use the car and can unlock it later to use it again.
- *Bill*: a record of the money owed by the user at the end of a ride.
- *Outstanding bill*: a bill that hasn't been paid yet.
- *Suspended user*: a user that cannot reserve or use cars. Usually users are suspended because they have outstanding bills.
- *Payment method*: a way to transfer money from the user to the system. Our system will only accept credit cards and online accounts like Paypal.
- *Payment API*: an interface to carry out money transactions, offered by the external provider associated to the payment method used (e.g. a bank).

1.3.2 Acronyms

- **DD**: Design Document
- **RASD**: Requirements Analysis and Specification Document
- **DB**: Database
- **DOB**: Date of birth
- **PGS**: Power Grid Station
- **GPS**: Global Positioning System

1.3.3 Abbreviations

- **[Gx]**: Goal
- **[RE.x]**: Functional Requirement
- **[UC.x]**: Use Case

1.4 List of Reference Documents

- Requirements analysis and specification document: "RASD.pdf"
- Design document: "DD.pdf"
- Project description document: "Assignments AA 2016-2017.pdf"

2 Integration Strategy

2.1 Entry Criteria

Before starting the integration testing phase specific conditions concerning the whole project development must be met. First of all, it is fundamental that the Requirements Analysis and Specification Document and the Design Document have been properly written and completed.

Regarding the code development it is only needed that the **INSERIRE PERCENTUALI** has been written. Furthermore, each component has to be successfully unit tested before being involved in the integration testing.

2.2 Elements to be Integrated

As stated in the Design Document (paragraph 2.2), the system is based on the cooperation of two parts, the *Application Subsystem* and the *Resource Management Subsystem*. The first handles all the operations related to the user applications and interfaces, while the second takes care of keeping track of all the automatic updates from the sensors in the cars and the charging stations. The testing process will verify the correctness of the integration between these macro-components after checking the proper cooperation of smaller components inside each subsystem. Furthermore, it will be tested the higher integration, client/server level, between

- the Web Application and the Application Subsystem
- the Car Application and the Application Subsystem

Concerning the external APIs, we expect them to work properly so we will only check, by unit testing, that the components execute correctly the API call.

Application Subsystem Components to be tested:

1. Registration
2. Account Manager
3. Car Search
4. PGS Search
5. Safe Areas Search

6. Payment
7. Reservation Manager
8. Unlocking
9. Parking Ride Manager
10. Ride Manager
11. Report Manager
12. Request Dispatcher
13. Model

It will be tested that the Request Dispatcher interacts with each component from 1 to 10 in the proper way and that these components cooperate as expected with the Model one. In addition, it will be tested the integration between the Model component and the DBMS.

Resource Management Subsystem Components to be tested:

1. Car Manager
2. PGS Manager
3. Car Handler

It will be tested the cooperation between the Car Manager and the Car Handler components and between the PGS Manager and the DBMS.

2.3 Integration Testing Strategy

We believe that the most efficient way to test the integration between PowerEnjoy's software components is the *bottom up approach*. Using this strategy, we will be able to test the cooperation between different units as soon as they are fully developed without waiting for the whole subsystem to be ready. This method will also highlight all the possible integration issues while the software is still under development, allowing quicker smaller fixes. We will start the process from independent components which don't have any dependencies and we will built further integration test on the already tested software. As specified in the Design Document, several components of the Application Subsystem depend on the Resource Management Subsystem thus the testing phase will start in this subsystem.

2.4 Sequence of Component / Function Integration

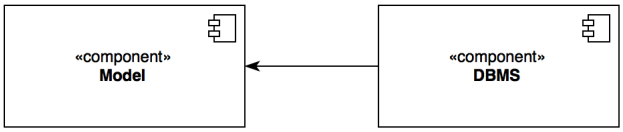
in this section we will describe in which order the system components will be integrated and tested diagrams will show the sequence more clearly graphs arrow = this component is a dependency of this other one this component is necessary for the other one to work

2.4.1 Software Integration Sequence

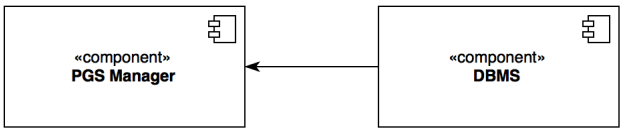
to start with the in

Model

why? because the model is an integral part of the application subsystem because we can't start integration testing any application subsystem component until the model is done immagine model-DBMS

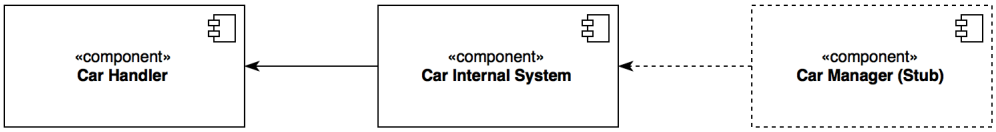


Power Grid Stations Manager why, a function that is already complete: updating the plugs' status immagine pgs manager-DBMS

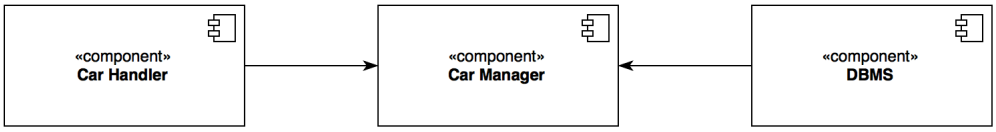


Car Handler the car handler keeps track of the cars in the system the car handler issues commands to the cars needs the cars car internal system tested and ready integrate the car handler with the car internal system bottom-up approach we need the car handler for the car manager, that is the foundation of many other functions we will use a stub to simulate the car manager interface in case it is needed

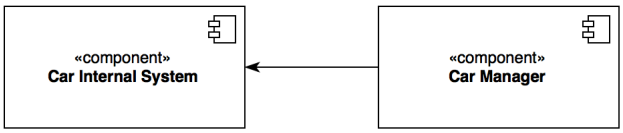
immagine car handler-Car Internal System



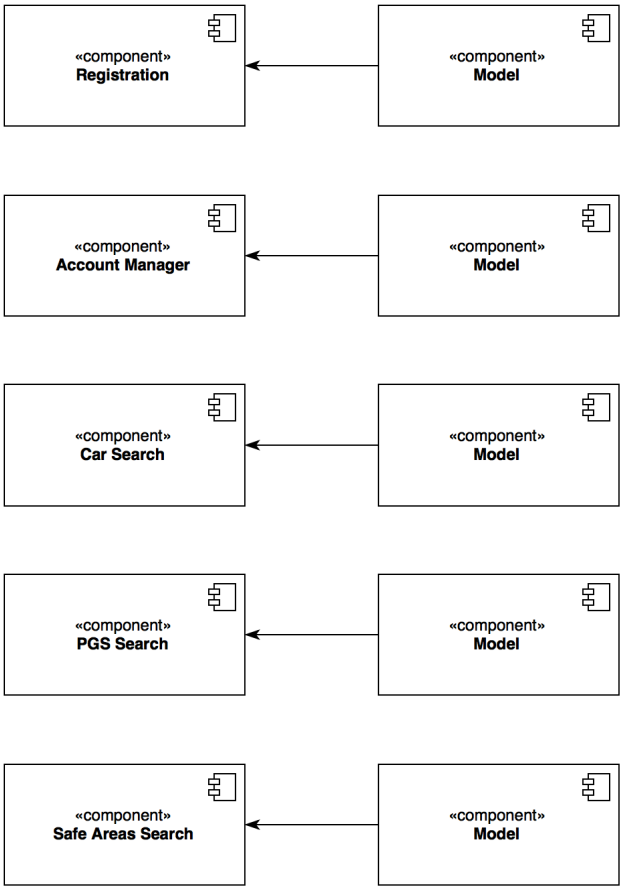
Car Manager we have the car handler, we have the dbms we can integrate the car manager, bulk of the resource management subsystem immagine car manager-car handler-dbms

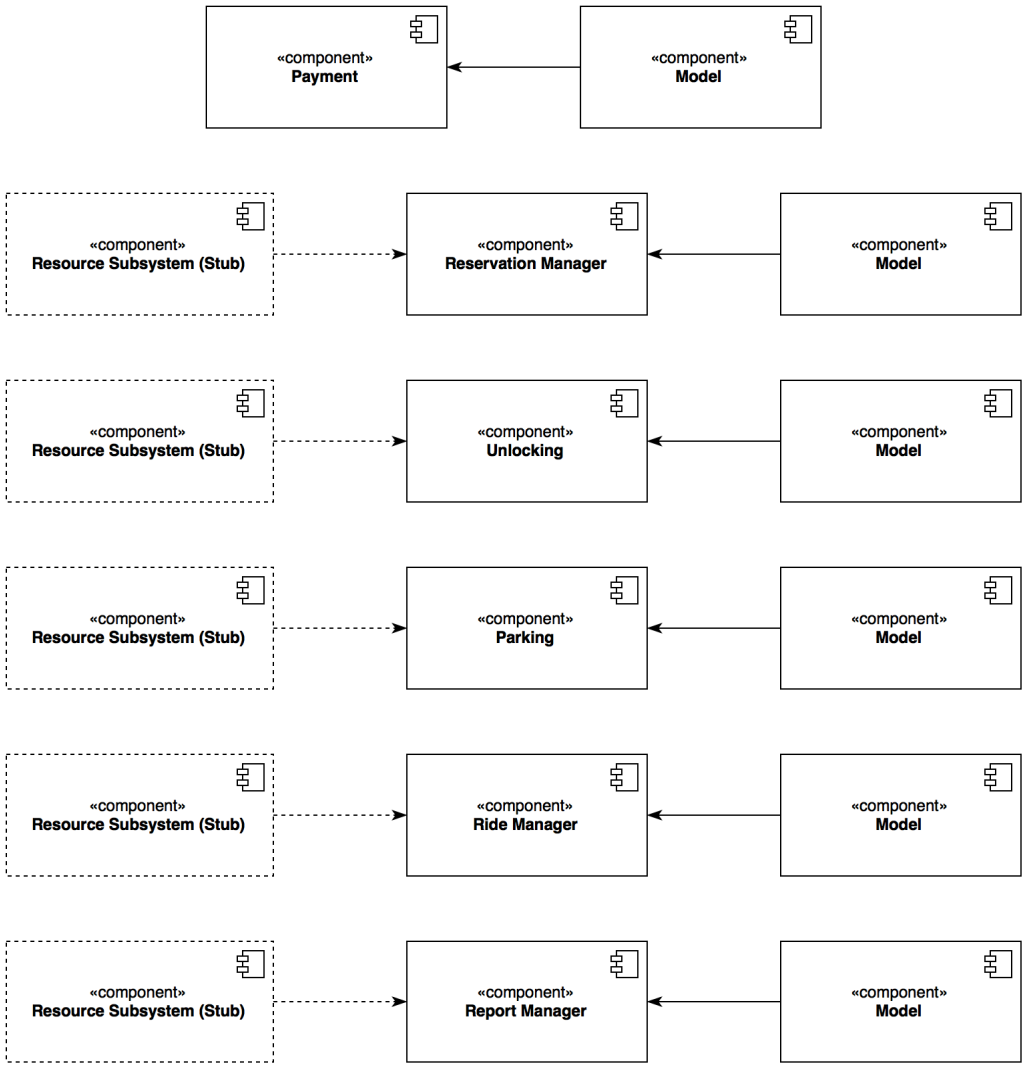


[poi car internal system-car manager] after the car manager has been integration tested, we can test the interaction between the car internal system and the car manager interface, this time for real immagine car-handler-car internal system -car manager



Application Subsystem Components with resource subsystem stub on the other side, done with the model, we can integrate all the single components of the application subsystem can be done concurrently, they are independent from each other some of them require interfacing to the resource subsystem, we will use a stub so these tests can be done at the same time as the ones in the resource subsystem immagine componenti-model(-resource stub)





Request Dispatcher "thread testing", "function something"? when testing for one of the intermediate components has been done, we can test the corresponding application functionality integrating the request dispatcher the request dispatcher is integrated with all the other components immagine request dispatcher-componenti

2.4.2 Subsystem Integration Sequence

done with the previous integrations, we can integrate the application subsystem and the resource subsystem

Application and Resource Subsystems Integration we put them together immagine application-resource-dbms (interfaces)

2.4.3 Client Applications Integration Sequence

after we're done with the central system, we can test if the clients work with it immagine clients-application-resource-dbms (car internal system?) oppure immagine clients-application immagine completa

3 Individual Steps and Test Description

3.1 Resource Management Subsystem

3.1.1 Component 1 - Component 2

method(arg1, arg2)	
Input	*Result*
input1	result1
input2	result2

3.2 Application Subsystem

3.3 Integration between subsystems

4 Tools and Test Equipment Required

5 Program Stubs and Test Data Required

6 Effort Spent

- Pietro Ferretti: hours of work
- Nicole Gervasoni: hours of work
- Danilo Labanca: hours of work

7 Revisions

7.1 Changelog

- ITDP v1.0, published on January 15, 2017