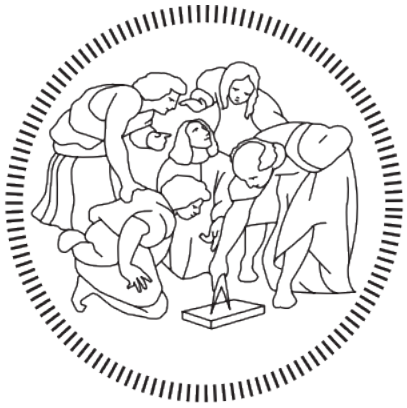
PowerEnJoy project: Integration Test Plan Document

Boriero Stefano 876106 Brunitti Simone 875039

January 15, 2017



POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Revision History

1.2 Purpose and Scope

The purpose of this ITPD (Integration Test Plan Document) is to describe how we intend to accomplish the integration testing of our PowerEnJoy platform. The components to be integrated are those identified in the DesignDocument that has been already written.

1.3 List of Definitions and Abbreviation

- System: the whole PowerEnJoy system
- Subsystem: a part of the system
- Module: a part of a subsystem
- Component: an atomic part of a module

1.4 List of Reference Documents

- Assignment
- RASD
- Desing Document
- iOS Test Tools
- Windows Phone Test Tools
- Android Test Tools

2 Integration Strategy

2.1 Entry Criteria

Before starting the integration phase, we have to meet certain constraints on different aspects of the project.

All components must have been unit tested The integration procedure should start after these percentages of completion are reached:

- **100%** of DatabaseController functionalities
- At least **90%** of Employee subsystem
- At least **80%** of Car subsystem
- At least **50%** of Customer subsystem

2.2 Elements to be Integrated

As it can be seen from the Design Document, the whole system can be divided into three major subsystem:

- **Customer subsystem**, offering functionalities to exploit the services provided by the car sharing company
- **Employee subsystem**, offering functionalities for company's employee to manage the system
- **Car subsystem**, offering functionalities to dialog with cars

Each of this subsystems is divided into submodules encapsulating the components related to a group of functionalities in this way:

- **Customer subsystem**:
  - *ReservationController*: offers the user functionalities to manage a reservation
  - *BalanceController*: offers the user functionalities to manage his balance
  - *AccountController*: offers the user functionalities to manage his account
  - *RegistrationController*: offers the user functionalities to register to the system
  - *ReportController*: offers the user functionalities to notify issues
- **Employee subsystem**:
  - *SafeAreaController*: offers the employee functionalities to manage safe areas

- *AssistanceController*: offers the employee functionalities to manage needy cars

- **Car subsystem**:
  - *CarController*: offers the system functionalities to send commands to the car
  - *RideController*: offers the CarApp the functionalities to manage a ride
  - *BillController*: offers functionalities to calculate bills
  - *FaultController*: offers the CarApp the functionalities to manage faults
  - *FleetController*: offer the CarApp the functionalities to join and detach from the fleet

The elements to be integrated are the components of these submodules

2.3 Integration Testing Strategy

We will follow a functional grouping approach, grouping components in the submodules presented above. We will first integrate functional groups of Employee subsystem, as these exploited also by the other subsystems. Then we will proceed integrating functional groups of Car subsystem and Customer subsystem.

For each functional group we will follow a bottom-up approach, and once integrated all submodules of a subsystem, we will integrate the subsystem with its dispatcher and web application. This will allow us to focus initially on binary integrations and to start testing complete functionalities as we proceed.

2.4 Sequence of Component/Function Integration

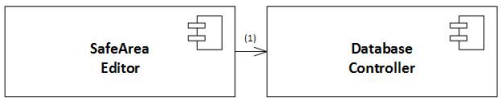
2.4.1 Software Integration Sequence

Employee Subsystem Strategy

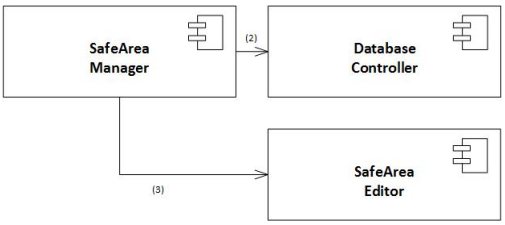
For testing the Employee System, we will use a bottom-up strategy. These are the separate modules we are going to test:

1. SafeAreaController
2. AssistanceController
3. EmployeeDispatcher

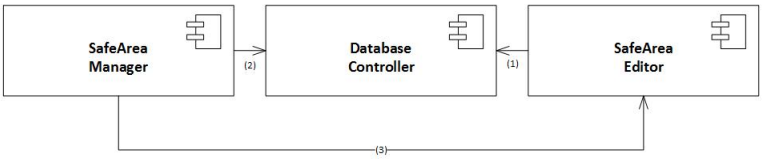
SafeAreaController
First we integrate the SafeArea Editor with the Database Controller



Then we integrate the SafeArea Manager with both the Database Controller and the SafeArea Editor

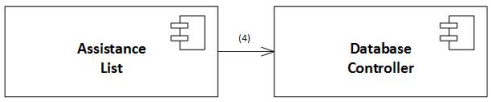


This is the final integrated subsystem

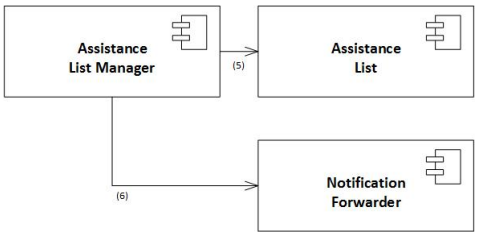


AssistanceController

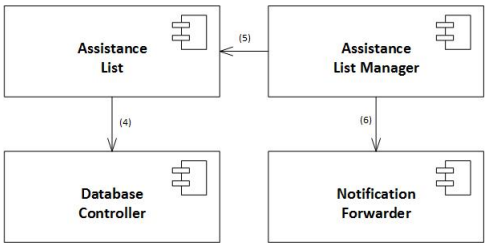
Following a similar procedure as the one used to integrate the SafeArea Controller, we start by testing the interaction between the Assistance List and the Database Controller



We then proceed to integrate the AssistanceList Manager with both the Assistance List and the Notification Forwarder

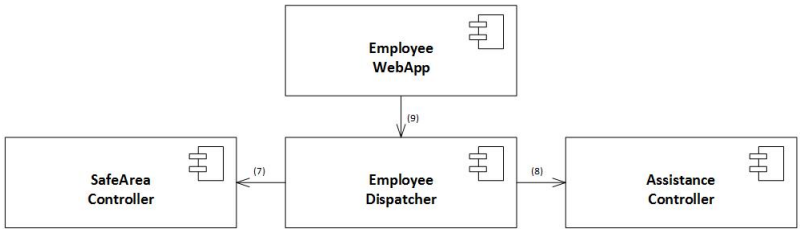


This is the final integrated AssistanceController subsystem



EmployeeDispatcher

Finally, we are able to test all the system together by integrating the EmployeeDispatcher with the two previously tested subsystems



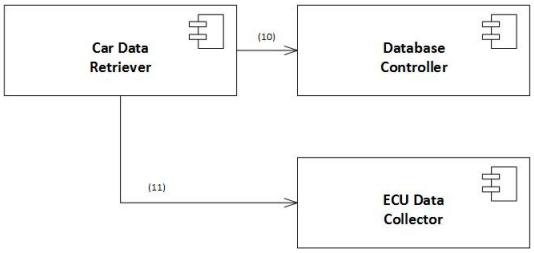
Car Subsystem Strategy

In this paragraph we are going to test the System that we will develop for managing the cars. As usual, we are going to follow a bottom-up approach by splitting our system in various subsystem, in order to make testing easier. These are the subsystems we want to test:

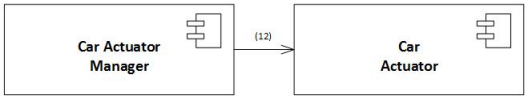
1. CarContoller
2. RideController
3. BillController
4. FaultController
5. FleetController
6. CarApplication

CarContoller

The strategy we will adopt in order to integrate this subsystem is the following: first, we test the Car Data Retriever with the Database Controller and the ECU Data Collector.

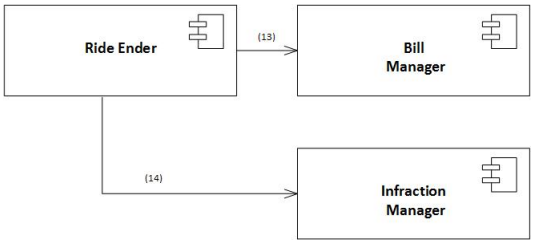


Then we will integrate the Car Actuator Manager with the Car Actuator.

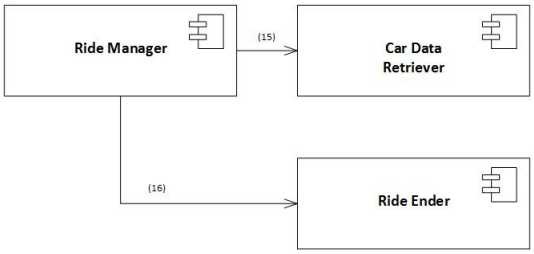


RideController

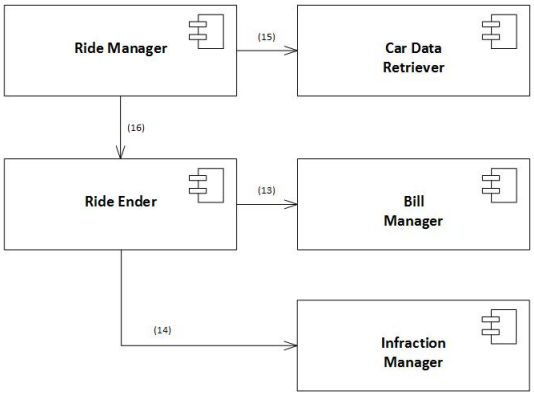
For the RideController, we first start integrating the Ride Ender with the Bill Manager and the Infraction Manager.



Subsequently, we integrate the Ride Manager with the Car Data Retriever and, finally, with the Ride Ender that we previously tested

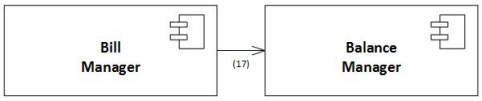


This is the final integrated RideController module



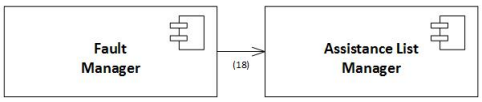
BillController

In testing the BillController, we must integrate the Bill Manager with the Balance Manager



FaultController

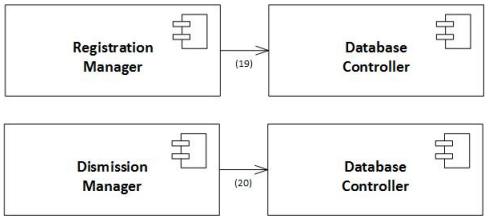
The integration sequence of the FaultController is straightforward, since we just need to test the interaction between the Fault Manager and the Assistance List Manager



FleetController

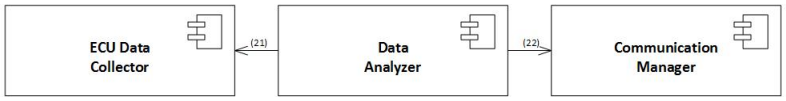
The FleetController simply needs to integrate its own components with the

Database Controller. We will first integrate the Registration Manager and then the Dismission Manager

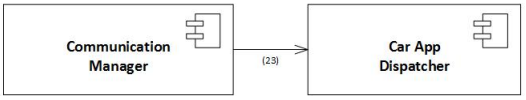


CarApplication

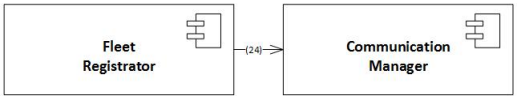
The CarApplication will be the last module we will test. To do this, we first integrate the Data Analyzer with the ECU Data Collector and the Communication Manager



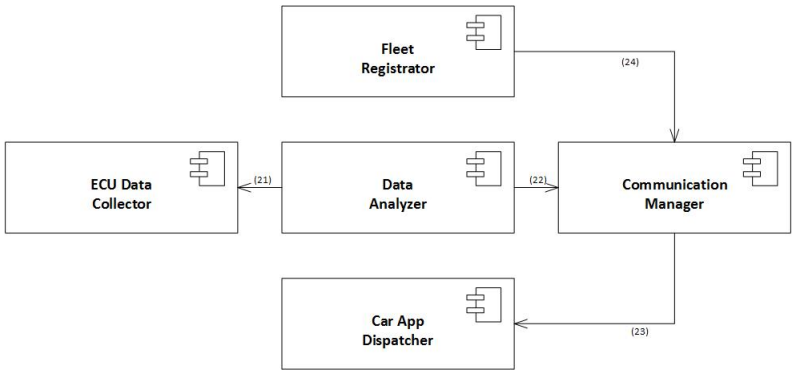
In the second step, we test the Communication Manager integration with the Car App Dispatcher



Finally, we test the interaction between the Fleet Registrator and the Communication Manager

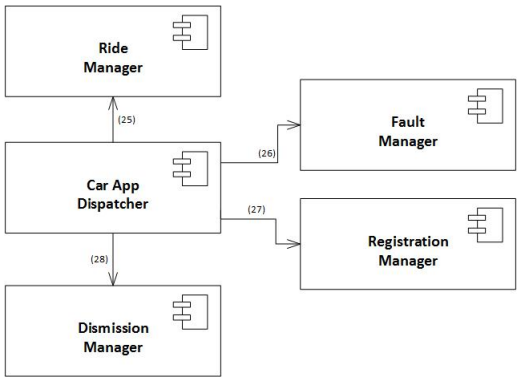


This is the whole integrated Car Application subsystem



Modules Integration

The final step of the Car System integration is to integrate all the modules in the following way



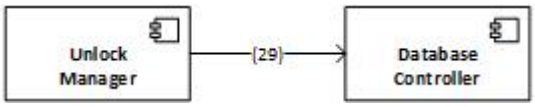
Customer Subsystem Strategy

As aforementioned, we will keep using a bottom-up approach in integating functional groups of the Customer subsystem. We will order them from the most critical module to the least critical. This is the ordering we are going to follow:

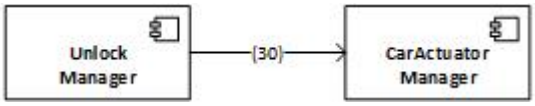
- ReservationController
- BalanceController
- AccountController
- RegistrationController
- ReportController

Reservation Controller

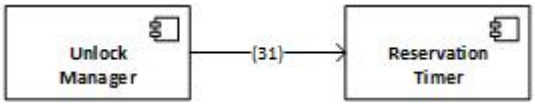
First we integrate the UnlockManger with the DatabaseController



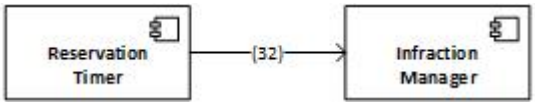
Then we integrate the UnlockManager with the CarActuatorDriver



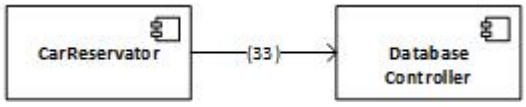
At last we integrate the UnlockManager with the ReservationTimer



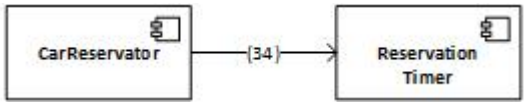
We proceed integrating the ReservationTimer with the InfractionManager-Driver



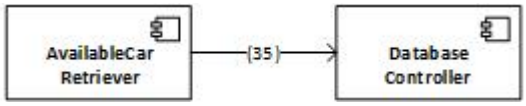
Now we integrate the CarReservator with the DatabaseController



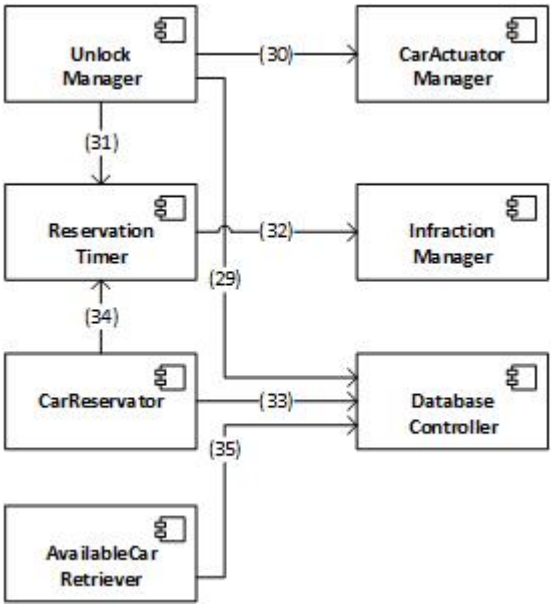
We integrate it also with the Reservation Timer



To complete the subsystem integration, we integrate AvailableCarRetriever with DatabaseController

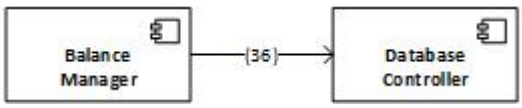


This is the final integrated ReservationController module

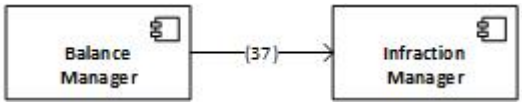


BalanceController

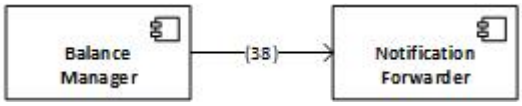
To integrate the components of this module we start from integrating BalanceManager with DatabaseController



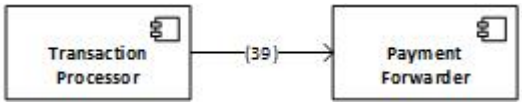
We integrate it also with the InfractionManager

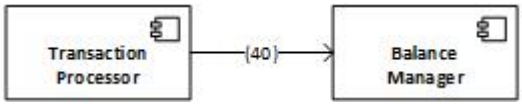


And with the NotificationForwarder

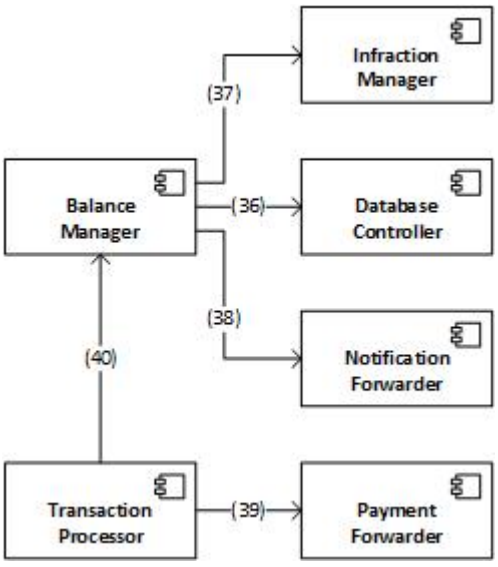


Then we integrate TransactionProcessor with both BalanceManager and PaymentForwarder



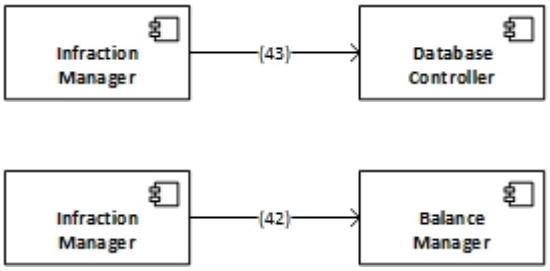


This is the final integrated BalanceController module

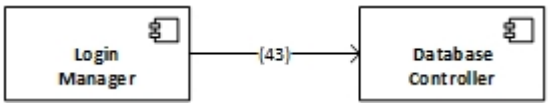


AccountController

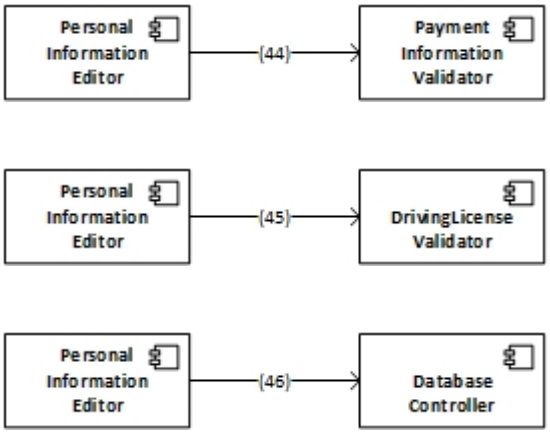
We start integrating InfractionManager with DatabaseController and BalanceManagerStub



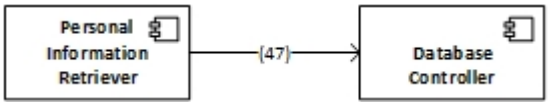
The second step is integrating LoginManager with DatabaseController



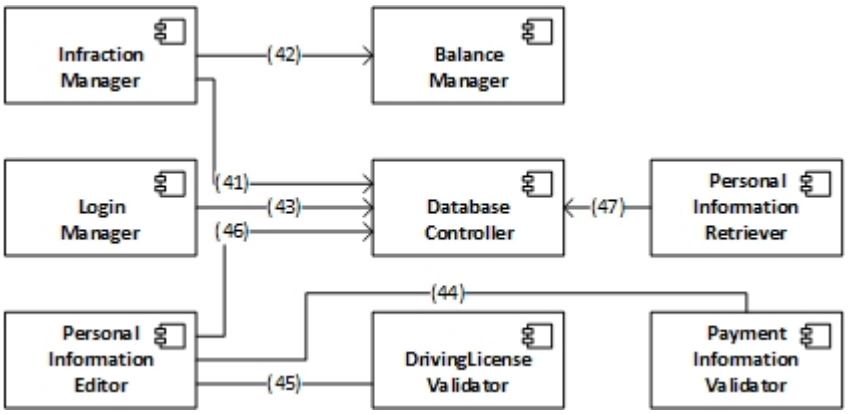
After that we integrate PersonalInformationEditor with PaymentInformationValidator, DrivingLicenseValidator and DatabaseController



The last integration is the one between PersonalInformationRetriever and DatabaseController

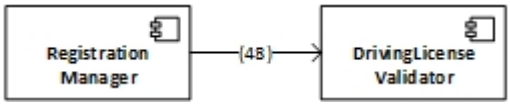


This is the final integrated AccountController module

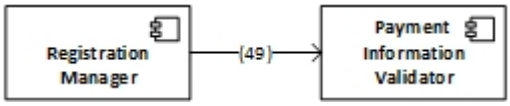


RegistrationController

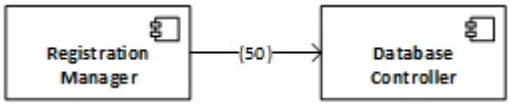
We start the integration procedure from the integration between the RegistrationManager and the DrivingLicenseValidator



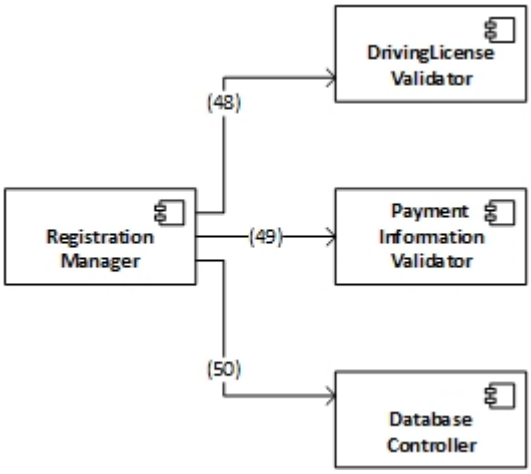
As second step, we integrate the RegistrationManager with the PaymentInformationValidator



In the end we integrate RegistrationManager with DatabaseController

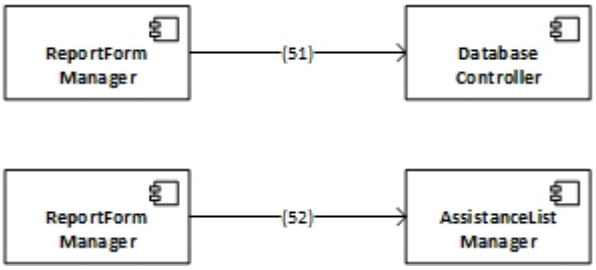


This is the final integrated RegistrationController module

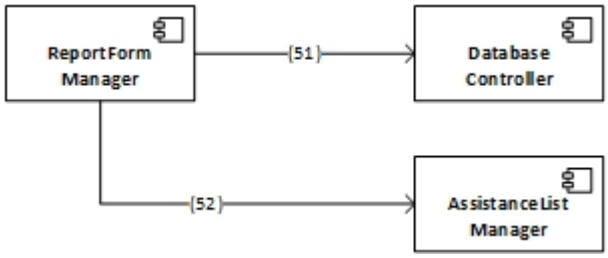


ReportController

For this module we integrate the ReportFormManager and both the DatabaseC-ontroller and the AssistanceListManagerStub

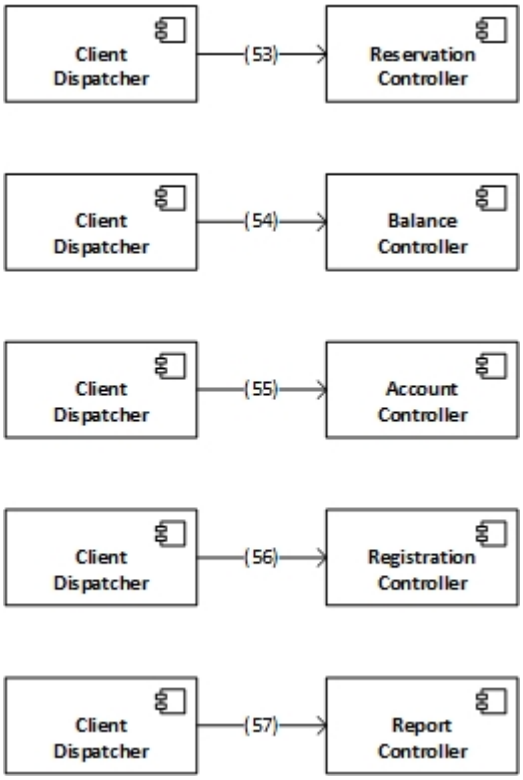


This is the final integrated ReportController module

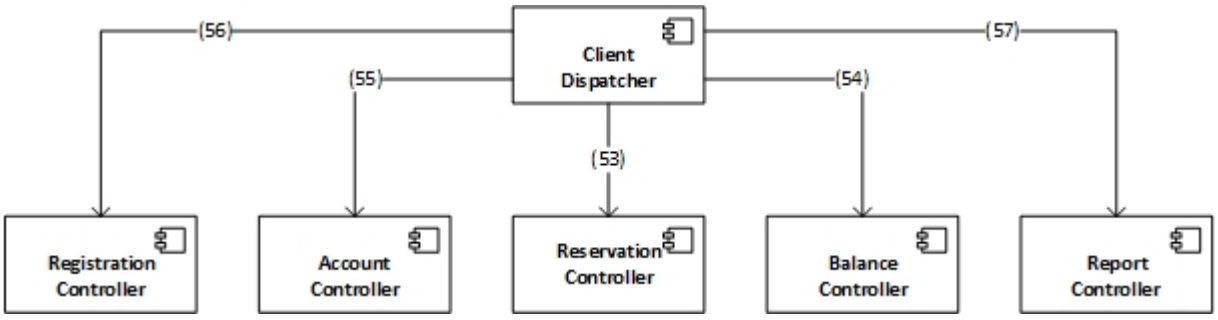


ClientDispatcher

We will integrate the ClientDispatcher with each one of the previously integrated modules

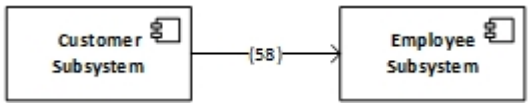


This is the final integrated module

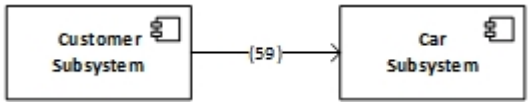


2.4.2 Subsystem Integration Sequence

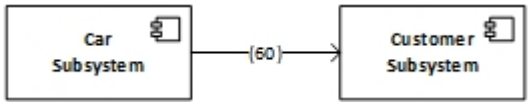
To integrate the tested subsystem, we start from integrating the Customer subsystem with the Employee Subsystem



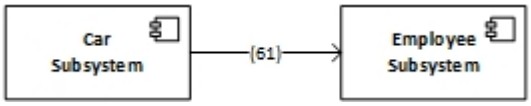
Then we proceed integrating the Customer subsystem with the car subsystem



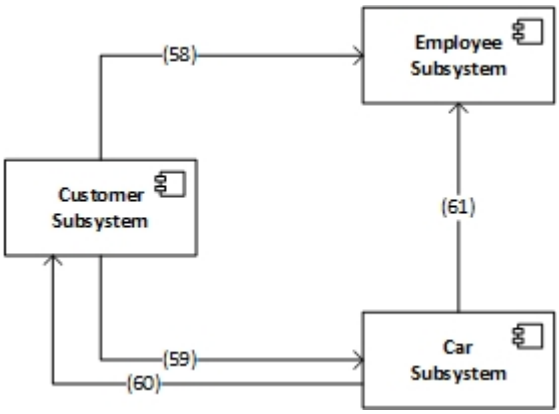
And the other way around



To end the process, we integrate the Car subsystem with the employee subsystem



This is the final integrated system



3 Individual Steps and Test Description

3.1 Employee Subsystem

3.1.1 I1 Safe Area Editor - Database Controller

create-safe-area(boundaries)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Empty list	InvalidArgumentValueException is raised
List containing some Null values	NullArgumentException is raised
List containing some out-of-bound boundaries (i.e. inexistent coordinates)	OutOfBoundException is raised
List containing some coordinates which are already in an existing safe area	Overlap Exception is raised
List containing all valid boundaries	The database is updated to include the new safe area

3.1.2 I2 Safe Area Manager - Database Controller

get-safe-areas()	
Input	**Effect**
Nothing	Return a list of all the safe areas

3.1.3 I4 Assitance List - Database Controller

add(vehicle-id)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle already present in the assistance list	AlreadyNeedyException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidVehicleException is raised
Valid vehicle	Add the vehicle to the assistance list and sets it as unavailable

solved(vehicle-id)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle that is not in the assistance list	NotNeedyVehicleException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidVehicleException is raised
Valid vehicle	Delete the vehicle from the assistance list and sets it as available

get-needy-vehicles()	
Input	**Effect**
Nothing	Return a list of all the vehicles that need assistance

take-in-charge(vehicle-id,employee-id)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet), Any	InvalidVehicleException is raised
Any, Invalid employee-id	InvalidEmployeeException is raised
Vehicle that is not in the assistance list (e.g. it has already been taken in charge by another employee), Any	NotNeedyVehicleException is raised
Valid vehicle, valid employee-id	Remove the vehicle from the assistance list and assigns it to that employee

3.1.4 I5 Assistance List Manager - Assistance List

addVehicle(vehicle-id)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle already present in the assistance list	AlreadyNeedyException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidArgumentException is raised
Valid vehicle	Add the vehicle to the assistance list and sets it as unavailable

3.1.5 I6 Assitance List Manager - Notification Forwarder

notify(user,info)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
User cannot be reached, Any	FailedCommunicationException is raised
Valid user, Valid info	A notification containing the specified info is sent to the specified user

3.1.6 I7 Employee Dispatcher - Safe Area Controller

create-safe-area(boundaries)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Empty list	InvalidArgumentValueException is raised
List containing some Null values	NullArgumentException is raised
List containing some out-of-bound boundaries (i.e. inexistent coordinates)	OutOfBoundException is raised
List containing some coordinates which are already in an existing safe area	Overlap Exception is raised
List containing all valid boundaries	The database is updated to include the new safe area

get-safe-areas()	
Input	**Effect**
Nothing	Return a list of all the safe areas

3.1.7 I8 Employee Dispatcher - Assistance Controller

solved(vehicle-id)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle that is not in the assistance list	NotNeedyVehicleException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidVehicleException is raised
Valid vehicle	Delete the vehicle from the assistance list and sets it as available

get-needy-vehicles()	
Input	**Effect**
Nothing	Return a list of all the vehicles that need assistance

take-in-charge(vehicle-id,employee-id)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet), Any	InvalidVehicleException is raised
Any, Invalid employee-id	InvalidEmployeeException is raised
Vehicle that is not in the assistance list (e.g. it has already been taken in charge by another employee), Any	NotNeedyVehicleException is raised
Valid vehicle, valid employee-id	Remove the vehicle from the assistance list and assigns it to that employee

3.1.8 I9 Employee WebApp - Employee Dispatcher

dispatch-request(request-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Invalid request	InvalidArgumentException is raised
Valid request	Forwards employee's request to the dedicated component

3.2 Car Subsystem

3.2.1 I10 Car Data Retriever - Database Controller

update-car-data(data-type, vehicle-id)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
Invalid data-type, Any	InvalidArgumentException is raised
Any, Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidArgumentException is raised
Valid data-type, Valid vehicle-id	Updates car data in the database

get-car-data(data-type, vehicle-id)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
Invalid data-type, Any	InvalidArgumentException is raised
Any, Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidArgumentException is raised
Valid data-type, Valid vehicle-id	Returns the requested data getting it from the database

3.2.2 I11 Car Data Retriever - ECU Data Collector

get-data(data-type)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Invalid data-type	InvalidArgumentException is raised
Valid data-type but cannot communicate with the vehicle	FailedComumunicationException is raised
Valid data-type	Dialogs with the on-board car application to get the specified data

3.2.3 I12 Car Actuator Manager - Car Actuator

unlock()	
Input	**Effect**
Nothing but the communication with the car fails	FailedCommunicationException is raised
Nothing	The car is unlocked

3.2.4 I13 Ride Ender - Bill Manager

calculate-bill(ride-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Invalid ride-info	InvalidArgumentException is raised
Vaild ride-info	Calculate the amount of the bill

3.2.5 I14 Ride Ender - Infraction Manager

signal-infraction(infraction-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Invalid infraction-info	InvalidArgumentException is raised
Vaild infraction-info	Change the user's status as signalled in the database

3.2.6 I15 Ride Manager - Car Data Retriever

get-car-data(data-type, vehicle-id)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
Invalid data-type, Any	InvalidArgumentException is raised
Any, Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidArgumentException is raised
Valid data-type, Valid vehicle-id	Returns the specified data and updates the database

3.2.7 I17 Bill Manager - Balance Manager

pay(amount,user)	
Input	**Effect**
Any of the two is a Null parameter	NullArgumentException is raised
Invalid amount(e.g. negative value), Any	InvalidArgumentException is raised
Any, User that is not in the database	InvalidArgumentException is raised
Valid amount, Valid user	The amount is subracted from the user balance

3.2.8 I18 Fault Manager - Assistant List Manager

add(vehicle-id)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle already present in the assistance list	AlreadyNeedyException is raised
Vehicle that is not in the database (i.e. it has not been registered in the fleet)	InvalidArgumentException is raised
Valid vehicle	Add the vehicle to the assistance list and sets it as unavailable

3.2.9 I19 Registration Manager - Database Controller

register-car(car-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle already present in the database	AlreadyRegisteredVehicleException is raised
Invalid car-info (e.g. vehicle plate too long)	InvalidArgumentException is raised
Valid car-info	Add the vehicle to the database and to the fleet

3.2.10 I20 Dismission Manager - Database Controller

dismiss-car(car-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle not present in the database	InexistantCarException is raised
Invalid car-info (e.g. vehicle plate too long)	InvalidArgumentException is raised
Valid car-info	Dismiss the vehicle from the fleet and delete it from the database

3.2.11 I21 Data Analyzer - ECU Data Collector

get-data(data-type)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Invalid data-type	InvalidArgumentException is raised
Valid data-type	Get the specified data from the ECU Data Collector

3.2.12 I22 Data Analyzer - Communication Manager

send-fault()	
Input	**Effect**
Nothing but the communication fails	FailedCommunicationException is raised
Nothing	A fault notification is sent

3.2.13 I23 Communication Manager - Car App Dispatcher

dispatch-request(request-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Invalid request	InvalidArgumentException is raised
Valid request	Forwards the car application request to the dedicated component

3.2.14 I24 Fleet Registrator - Communication Manager

send-registration-request(vehicle-info)	
Input	**Effect**
Invalid vehicle-info	InvalidArgumentException is raised
Valid vehicle-info but the communication fails	FailedCommunicationException is raised
Nothing	A registration request is sent

3.2.15 I25 Car App Dispatcher - Ride Manager

start-ride(vehicle-id)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Valid vehicle-id but communication fails	FailedCommunicationException is raised
Valid vehicle-id	The Ride Monitor is aware of the fact that the ride started

end-ride(ride-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
invalid ride-info (e.g. negative number of passengers)	InvalidArgumentException is raised
Valid ride-info but communication fails	FailedCommunicationException is raised
Valid ride-info	The Ride Monitor is aware of the fact that the ride ended, calculate the bill and forward it to the dedicated component

3.2.16 I26 Car App Dispatcher - Fault Manager

notify-fault(fault-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Valid fault-info but failed communication	FailedCommunicationException is raised
Valid fault-info	Forwards the fault-info to the employee system by means of a notification

3.2.17 I27 Car App Dispatcher - Registration Manager

register-car(car-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle already present in the database	AlreadyRegisteredVehicleException is raised
Invalid car-info (e.g. vehicle plate too long)	InvalidArgumentException is raised
Valid car-info	Add the vehicle to the database and to the fleet

3.2.18 I28 Car App Dispatcher - Dismission Manager

dismiss-car(car-info)	
Input	**Effect**
Null parameter	NullArgumentException is raised
Vehicle not present in the database	InexistantCarException is raised
Invalid car-info (e.g. vehicle plate too long)	InvalidArgumentException is raised
Valid car-info	Dismiss the vehicle from the fleet and delete it from the database

3.3 Customer Subsystem

3.3.1 I29 UnlockManager - DatabaseController

isUnlockable(vehicle-id, user-info)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing vehicle, Any	InvalidVehicleException is raised
Existing vehicle, user too far from the car	TooFarException is raised
Existing but unlocked vehicle, valid user-info	AlreadyUnlockedException is raised
Existing and locked vehicle	Returns true

setUnlocked(vehicle-id)	
Input	**Effect**
A Null parameter	NullParemeterException is raised
Non existing vehicle	InvalidVehicleException is raised
Existing vehicle	Database entry is updated to Unlocked

3.3.2 I30 UnlockManager - CarActuator

unlock(vehicle-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Unlockable vehicle-id	Right function is called

3.3.3 I31 UnlockManager - ReservationTimer

stop-timer(vehicle-id, user)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Valid vehicle-id, valid user	The timer for reservation is stopped

3.3.4 I32 ReservationTimer - InfractionManagerDriver

timeout(vehicle-id, user-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Any, Any	Right function is called from InfractionManager

3.3.5 I33 CarReservator - DatabaseController

create-reservation(user-id, vehicle-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user-id, Any	InvalidArgumentException is raised
Any, Non existing vehicle-id	InvalidArgumentException is raised
Debtor user, Any	InvalidUserException is raised
Banned user, Any	InvalidUserException is raised
Regular user, Reserved vehicle	UnavailableCarException is raised
Regular user, Needy vehicle	UnavailableCarException is raised
Regular user, Available vehicle	Database entry is created

3.3.6 I34 CarReservator - ReservationTimer

start-timer(reservation)	
Input	**Effect**
A Null parameter	NullParameterException is raised
A reservation	Timer for the reservation is started

3.3.7 I35 AvailableCarRetriever - DatabaseController

get-near-vehicles(position, range)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Invalid position, Any	InvalidPositionException is raised
Valid position, Non-positive integer	InvalidArgmuentException is raised
Valid position, Positive integer	Returns the list of available cars in the input range from the input poition

3.3.8 I36 BalanceManager - DatabaseController

pay(user-id, amount)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, Non-positive value	InvalidArgumentException is raised
Existing user, More than deposited	Database entry is updated and DebtException is raised
Existing user, Positive value	Database entry is updated

deposit(user-id, amount)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, Non-positve amount	InvalidArgumentException is raised
Debtor user, More than debt	User balance and status entries are updated and user status is changed
Regular user, Positive value	User balance entry is updated

3.3.9 I37 BalanceManager - InfractionManager

signal-infraction(user-id, infraction-info)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, Infraction	Infraction is handled by InfractionManager

3.3.10 I38 BalanceManager - NotificationForwarder

notify(user)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user	InvalidUserException is raised
Existing user	A notification is sent to the user

3.3.11 I39 TransactionProcessor - PaymentForwarder

withdraw(payment-info, amount, user-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Any, Any, Non existing user	InvalidUserException is raised
Invalid info, Any, Existing user	TransactionExeption is raised
Valid info, Non-positive value, Existing user	InvalidArgumentException is raised
Valid info, Positive value more than affordable, Existing user	TransactionException is raised
Valid info, Positive value less than affordable, Existing user	Money is transfered from user payment provider to the company

3.3.12 I40 TransactionProcessor - BalanceManager

deposit(user-id, amount)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, Non-positve amount	InvalidArgumentException is raised
Debtor user, More than debt	User balance and status entries are updated and user status is changed
Regular user, Positive value	User balance entry is updated

3.3.13 I41 InfractionManager - DatabaseController

update-status(user-id, status)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, status	Database entry is updated

3.3.14 I42 InfractionManager - BalanceManager

pay(user-id, amount)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, Non-positive value	InvalidArgumentException is raised
Existing user, more than deposited	Database entry is updated and DebtException is raised
Existing user, Positive value	Database entry is updated and DebtException is raised

3.3.15 I43 LoginManager - DatabaseController

login(username, password)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidCredentialException is raised
Existing user, Wrong password	InvalidCredentialException is raised
Existing user, Correct password	Login is granted

3.3.16 I44 PersonalInformationEditor - PaymentInformationValidator

isValid(payment-info)	
Input	**Effect**
Null parameter	NullParameterException is raised
Invalid payment info	InvalidPaymentInfoException is raised
Valid payment information	Returns true

3.3.17 I45 PersonalInformationEditor - DrivingLicenseValidator

isValid(driving-license)	
Input	**Effect**
Null parameter	NullParameterException is raised
Invalid driving license	InvalidLicenseException is raised
Valid driving license	Returns true

3.3.18 I46 PersonalInformationEditor - DatabaseController

edit-personal-info(user-info, user-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
User-info, Non existing user	InvalidUserException is raised
User-info, Existing user	Database entry is updated

3.3.19 I47 PersonalInformationRetriever - DatabaseController

get-personal-info(user-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user	InvalidUserException is raised
Existing user	Returns user's information from the database

3.3.20 I48 RegistrationManager - DrivingLicenseValidator

isValid(driving-license)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Invalid license	InvalidLicenseException is raised
Valid license	Returns true

3.3.21 I49 RegistrationManager - PaymentInformationValidator

isValid(payment-info)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Invalid payment info	InvalidPaymentInfoException is raised
Valid payment info	Returns true

3.3.22 I50 RegistrationManager - DatabaseController

register(user-info)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Existing username	AlreadyExistingUsernameException is raised
New username	Database entry is created

3.3.23 I51 ReportFormManager - DatabaseController

get-vehicle-id(vehicle-plate)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing plate	InvalidArgumentException is raised
Existing plate	Returns vehicle-id from the database
insert-report(vehicle-id, report)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Valid id, Empty report	EmptyReportException is raised
Valid-id, Non-empty report	Database entry is created

3.3.24 I52 ReportFormManager - AssistanceListManager

add(vehicle-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Valid id	Vehicle is added to the AssistanceList

3.3.25 I53 ClientDispatcher - ReservationController

reserve(vehicle-id, user)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Any, Non existing user	InvalidArgumentException is raised
Non existing vehicle, Existing user	InvalidArgumentException is raised
Debtor user, Any	InvalidUserException is raised
Banned user, Any	InvalidUserException is raised
Regular user, Reserved vehicle	UnavailableCarException is raised
Regular user, Needy vehicle	UnavailableCarException is raised
Regular user, Available vehicle	Database entry is created

unlock(veicle-id, user)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing vehicle, Any	InvalidVehicleException is raised
Existing vehicle, user too far from the car	TooFarException is raised
Existing but unlocked vehicle, valid user-info	AlreadyUnlockedException is raised
Existing and locked vehicle, valid user-info	Database entry is updated, reservation timer is stopped and car is unlocked

get-near-vehicles(position)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Invalid position, Any	InvalidPositionException is raised
Valid position, Non-positive integer	InvalidArgmuentException is raised
Valid position, Positive integer	Returns the list of available cars in the input range from the input poition

3.3.26 I54 ClientDispatcher - BalanceController

deposit(payment-method, amount, user-id	
Input	**Effect**
A Null parameter	NullParameterException is raised
Any, Any, Non existing user	InvalidUserException is raised
Invalid info, Any, Existing user	TransactionExeption is raised
Valid info, Non-positive value, Existing user	InvalidArgumentException is raised
Valid info, Positive value more than aordable, Existing user	TransactionException is raised
Valid info, Positive value less than aordable, Existing user	Money is transfered from user payment provider to the company

3.3.27 I55 ClientDispatcher - AccountController

login(username, password)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidCredentialException is raised
Existing user, Wrong password	InvalidCredentialException is raised
Existing user, Correct password	Login is granted

get-personal-info(user-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user	InvalidUserException is raised
Existing user	Returns users information from the database

edit-personal-info(user-info, user-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Any, Non existing user	InvalidArgumentException is raised
Invalid payment info, Existing user	InvalidPaymentInfoException is raised
Invalid driving license, Existing user	InvalidLicenseException is raised
Valid info, Existing user	Database entries are updated

3.3.28 I56 ClientDispatcher - RegistrationController

register(user-info)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Invalid payment info	InvalidPaymentInfoException is raised
Invalid driving license	InvalidLicenseException is raised
Existing username	AlreadyExistingUsernameException is raised
New username, Valid info	Database entry is created

3.3.29 I57 ClientDispatcher - ReportController

report(vehicle-plate, report)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing plate, Any	InvalidArgumentException is raised
Valid plate, Empty report	InvalidArgumentException is raised
Valid plate, Non-empty report	Database entry is created and vehicle is added to AssistanceList

3.4 Subsystem Integration

3.4.1 I58 Customer subsystem - Employee subsystem

add(vehicle-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Valid id	Vehicle is added to the AssistanceList

3.4.2 I59 Customer subsystem - Car subsystem

unlock(vehicle-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Unlockable vehicle-id	Right function is called

3.4.3 I60 Car subsystem - Customer subsystem

pay(user-id, amount)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Non existing user, Any	InvalidUserException is raised
Existing user, Non-positive value	InvalidArgumentException is raised
Existing user, more than deposited	Database entry is updated and DebtException is raised
Existing user, Positive value	Database entry is updated and DebtException is raised

3.4.4 I61 Car subsystem - Employee subsystem

add(vehicle-id)	
Input	**Effect**
A Null parameter	NullParameterException is raised
Valid id	Vehicle is added to the AssistanceList

4 Tools and Test Equipment Required

4.1 Test Tools

We plan to use a number of different tools in order to excute the test plan we described in this document.

4.1.1 Functional Testing

The first set of testing tools we are going to use is the one that will help us test our application in order to understand if it meets the functional requirements we defined in our RASD.

First, we are going to use **JUnit**, an unit testing framework for the Java language. We will use this framework mainly for testing the interaction between components, by checking the correct behaviour of functions when the input is represented by some particular data sets (that we defined in the previous chapter).

The second tool we want to use is **Arquillian**, a testing framework that can be easily embedded with JEE. Arquillian can be used to produce a great number of integration tests for Java applications, but we will mainy use it for testing our EJBs and the interaction of our application with the JPA (and therefore data access).

4.1.2 Non-Functional Tests

This section contains some of the testing tools we will use in order to test if our platform meets reasonable performance standards. Since perfomance varies depending on the platform the application is run on, we will need to use different tools for each platform.
First, we will test the performance of our central system using **JMeter**, that can be used to perform various stress tests on different components under different workloads.
On the other hand, we will also need to check the performances of the terminal devices, keeping under control memory usage, CPU usage and battery usage for mobile devices. To perform these tests, we will use:

- iOS: **Xcode** and other tools available for download on the Apple Developer website
- Android: **Android Monitor**
- Windows Phone: **Windows Phone Application Analysis** tool

4.2 Test Equipment

For executing the tests described in this document, we will need some specific equipment. First, we will need smartphones and tablets that we will use in order to test the application. These devices will need to have:

- An Internet connection
- A Web browser
- GPS sensor

There is no other requirement for these devices since we want to make the application widely available. For the PC web application, we will need desktop and notebook computers. These computers need to have access to an internet connection and to a web browser.

Next, we need at least a car to test the Car Application. This car must be equipped with an ECU consisting of the following components:

- Door Control Unit (DCU): actuators for opening and closing doors by remote control, sensors to capture actual state of doors
- Engine Control Unit (ECU): sensors to collect data and actuators to ensure optimal performances
- Seat Control Unit: sensors to detect the presence of passengers on the vehicle
- Telematic Control Unit (TCU): sensor to ensure vehicle tracking (GPS)
- Battery Managment System (BMS): sensors monitoring battery state

Moreover, the car must be equipped with an on-board android device with GSM, GPRS, LTE and Wi-Fi communication modules.

Finally, we will need the equipment to test the backend. These tests will be done on the server we are going to use for deploying the final system, so that we will be able to operate stress tests on these machines.

5 Program Stubs and Test Data Required

5.1 Program Drivers

As we already specified in the previous chapters, we will make use of a bottom-up approach for tesing our system. To achieve this type of testing, we will need drivers that will simulate function invocations. Here is a list of drivers we will use:

Driver	**Called Component**
SafeArea Editor Driver	Database Controller
SafeArea Manager Driver	Database Controller and SafeArea Editor
Assistance List Driver	Database Controller
Assistance List Manager Driver	Assistance List and Notification Forwarder
Employee Dispatcher Driver	SafeArea Controller and Assistance Controller
Employee WebApp Driver	Employee Dispatcher
Car Actuator Manager Driver	Car Actuator
Ride Ender Driver	Bill Manager and Infraction Manager
Ride Manager Driver	Car Data Retriever and Ride Ender
Bill Manager Driver	Balance Manager
Fault Manager Driver	Assistance List Manager
Registration Manager and Dismission Manager Drivers	Database Controller
Data Analyzer Driver	Data Collector and Communication Manager
Communication Manager Driver	Car App Dispatcher
Fleet Registrator Driver	Communication Manager
Car App Dispatcher Driver	Ride Manager, Fault Manager, Registration Manager and Dismission Manager
Unlock Manager Driver	Database Controller, Car Actuator Manager and Reservation Timer
Reservation Timer Driver	Infraction Manager
Car Reservator Driver	Database Controller and Reservation Timer
Available Car Retriever Driver	Database Controller
Balance Manager Driver	Database Controller, Infraction Manager and Notification Forwarder
Transaction Processor Driver	Balance Manager and Payment Forwarder

Infraction Manager Driver	Balance Manager and Database Controller
Login Manager Driver	Database Controller
Personal Information Editor Driver	Payment Information Validator, Driving License Validator and Database Controller
Personal Information Retriever Driver	Database Controller
Registration Manager Driver	Payment Information Validator and Database Controller
Report Form Manager Driver	Database Controller and Assistance List Manager
Client Dispatcher Driver	Reservation Controller, Balance Controller, Account Controller, Registration Controller and Report Controller

5.2 Test Data

The following list contains different sets of test data we want to try in order to check the correct behaviour of functions in some special situations.

Component	**Data**
SafeArea Controller	• Boundaries creating an invalid area (e.g an infinite area, a null area)
Fleet Controller	• Invalid car plate(e.g. invalid legnth)
Registration Controller	• invalid username • invalid email (e.g. non well-formed email) • invalid driving license
Balance Controller	• invalid payment information (e.g. invalid length)

All other sets of test data have already been discussed in chapter 3.

6 Effort Spent

Stefano Boriero: 20 hours
Simone Brunitti : 20 hours