POLITECNICO
MILANO 1863

Travlendar+
Integration Test Plan Document

Bolshakova Liubov
Campagnoli Chiara
Lagni Luca

Software Engineering 2
November 24, 2017

Contents

1 Introduction **2**

1.1 Objective 2

1.2 Meanings 2

2 Integration Strategy **2**

2.1 Entry Point Classes 2

2.2 First Mixture 2

2.3 Core Application Classes 3

2.4 Second Mixture 3

2.5 Fine-Grained Classes 4

2.6 Final Mixture 5

2.7 Elements to be integrated 7

1 Introduction

1.1 Objective

The pourpose of the Integration Test Plan Document (ITPD) is to schedule and present to the team disposed for testing the application (and its infrastructure) the required and logic sequence of tests to be applied to different components and interfaces that compose our application.
We do this in order to have a confirm that the components and interfaces proposed in the DD behave in a correct and predictable manner an that the interactions between the components act as qwe expect.

1.2 Meanings

2 Integration Strategy

2.1 Entry Point Classes

This subsection is concerning the entry points of our application or rather , the first components that are involved in the interaction between the external environment and our application.
This is the first thing to do because , if we want to test properly our components and interfaces, we must be sure that the initial data ,that will be provided by the user, will be processed correctly and will feed other objects , related to these ones, whit correct and appropriate data.

These classes are:

1. Coordinate

- ExternalCompany
- Technician
- User
- UserDevice

We have decided to start from this classes because they're the one that must be exists and that it's not necessessary for them to be associated to other classes (except in the case of coordinate).
We wanto to specify that these classes, only by their own, are usless , but they're necessessary for the interaction between the application and the agents.
The Coordinate is the first one to be tested, because it's the only one which is used also by some others.
For the other classes the order is not important.

All the methods of these classes should be tested (also the getters, because they're used also inside other methods of the classes).

2.2 First Mixture

At this stage is required to test the Entry classes that are related in order to see if they cooperate correctly.

Caller class	Called class
Coordinate	UserDevice
Coordinate	User
User	UserDevice

2.3 Core Application Classes

This subsection is concerning the core of our application.
With that we mean all the classes that, related to the entry point classes , can compose a functional and finite (but too simplified and unprecise) application.
These classes are essential for our application but they can be obtained only by an interaction beetween the external environment and our entry points.

These classes are:

1. MeanOfTransport
2. Navigator
3. Trip
4. Meeting
5. Notification
6. NotificationManager
7. TechnicalProblem
8. SystemShared

In this case the order is important because all the classes are related.

2.4 Second Mixture

At this stage is required to test how core classes are related to each others and with entry classes in order to test if the basic behaviour of the application is correct.

Caller class	**Called class**
MeanOfTransport MeanOfTransport	Coordinate ExternalCompany
Navigator Navigator	MeanOfTransport Trip
Trip Trip	Coordinate Meeting
Meeting Meeting	Coordinate User
Notification Notification	ExternalCompany NotificationManager
NotificationManager NotificationManager NotificationManager	MeanOfTransport Navigator Trip
User TechnicalProblem	TechnicalProblem Technician
SystemShared SystemShared SystemShared SystemShared	User ExternalCompany UserDevice NotificationManager

2.5 Fine-Grained Classes

This subsection is concerning classes that are specifications of other classes and are useful to provide a better application from the point of view of features and behaviours
These classes are (most of them) subclasses that adapt a behaviour to a specifica situation.

These classes are:

- Visitor
- Client
- UserPreferences
- WindowsPhoneWrapper
- IOSWrapper
- AndroidWrapper
- PathRestriction
- OAMOTRestriction
- OtherTypeOfRestriction
- Break
- StrikeNotification
- WeatherNotification
- OtherNotification
- ExternalMeanOfTransportCompany
- Feet
- Bike
- Boat
- ONAMOT
- Ride
- Car
- Ship
- Tram
- Bus
- Train
- OAMOT
- AdditionalOAMOTData

In this case all the classes can be tested in a random order.

2.6 Final Mixture

Caller class	Called class
UserDevice	WindowsPhoneWrapper
UserDevice	IOSWrapper
UserDevice	AndroidWrapper
Visitor	TechnicalProblem
Visitor	Meeting
Visitor	UserPreferences
Visitor	UserDevice
Client	TechnicalProblem
Client	Meeting
Client	UserPreferences
Client	UserDevice
OAMOTRestriction	AdditionalOAMOTData
OAMOTRestriction	Trip
OtherTypeOfRestriction	Trip
Break	Trip
ExternalCompany	WeatherNotification
ExternalCompany	OtherNotification
ExternalMeanOfTransportCompany	StrikeNotification
ExternalMeanOfTransportCompany	OtherNotification
ExternalMeanOfTransportCompany	NotificationManager
ExternalMeanOfTransportCompany	Bike
ExternalMeanOfTransportCompany	ONAMOT
ExternalMeanOfTransportCompany	Boat
ExternalMeanOfTransportCompany	Car
ExternalMeanOfTransportCompany	Tram
ExternalMeanOfTransportCompany	Bus
ExternalMeanOfTransportCompany	Train
ExternalMeanOfTransportCompany	Taxi
ExternalMeanOfTransportCompany	OAMOT
AdditionalOAMOTData	Car
AdditionalOAMOTData	OAMOT
Feet	Coordinate
Feet	WeatherNotification
Feet	OtherNotification
Feet	NotificationManager
Bike	Coordinate
Bike	WeatherNotification
Bike	OtherNotification
Bike	NotificationManager
Navigator	Feet
Navigator	Bike
Navigator	Boat
Navigator	ONAMOT
Navigator	Car
Navigator	Tram
Navigator	Ship
Navigator	Bus
Navigator	Train
Navigator	Taxi
Navigator	OAMOT

Caller class	Called class
Navigator	WeatherNotification
Navigator	OtherNotification
Navigator	StrikeNotification
Boat	Coordinate
Boat	WeatherNotification
Boat	OtherNotification
Boat	StrikeNotification
Boat	NotificationManager
Boat	Ride
ONAMOT	Coordinate
ONAMOT	WeatherNotification
ONAMOT	OtherNotification
ONAMOT	StrikeNotification
ONAMOT	NotificationManager
ONAMOT	Ride
Car	Coordinate
Car	WeatherNotification
Car	OtherNotification
Car	StrikeNotification
Car	NotificationManager
Car	AdditionalAMOTData
Car	Ride
Ship	Coordinate
Ship	WeatherNotification
Ship	OtherNotification
Ship	StrikeNotification
Ship	NotificationManager
Ship	Ride
Tram	Coordinate
Tram	OtherNotification
Tram	StrikeNotification
Tram	NotificationManager
Tram	Ride
Train	Coordinate
Train	OtherNotification
Train	StrikeNotification
Train	NotificationManager
Train	Ride
Taxi	Coordinate
Taxi	OtherNotification
Taxi	StrikeNotification
Taxi	NotificationManager
Taxi	Ride
Bus	Coordinate
Bus	OtherNotification
Bus	StrikeNotification
Bus	NotificationManager
Bus	Ride

Caller class	Called class
OAMOT	Coordinate
OAMOT	WeatherNotification
OAMOT	OtherNotification
OAMOT	StrikeNotification
OAMOT	NotificationManager
OAMOT	AdditionalMOTData
OAMOT	Ride
SystemShared	Visitor
SystemShared	Client

2.7 Elements to be integrated

All the components that have been described in the component view section of the DD must be integrated.
We have avoided, when not essential , the representation of the design patterns classes done so far.

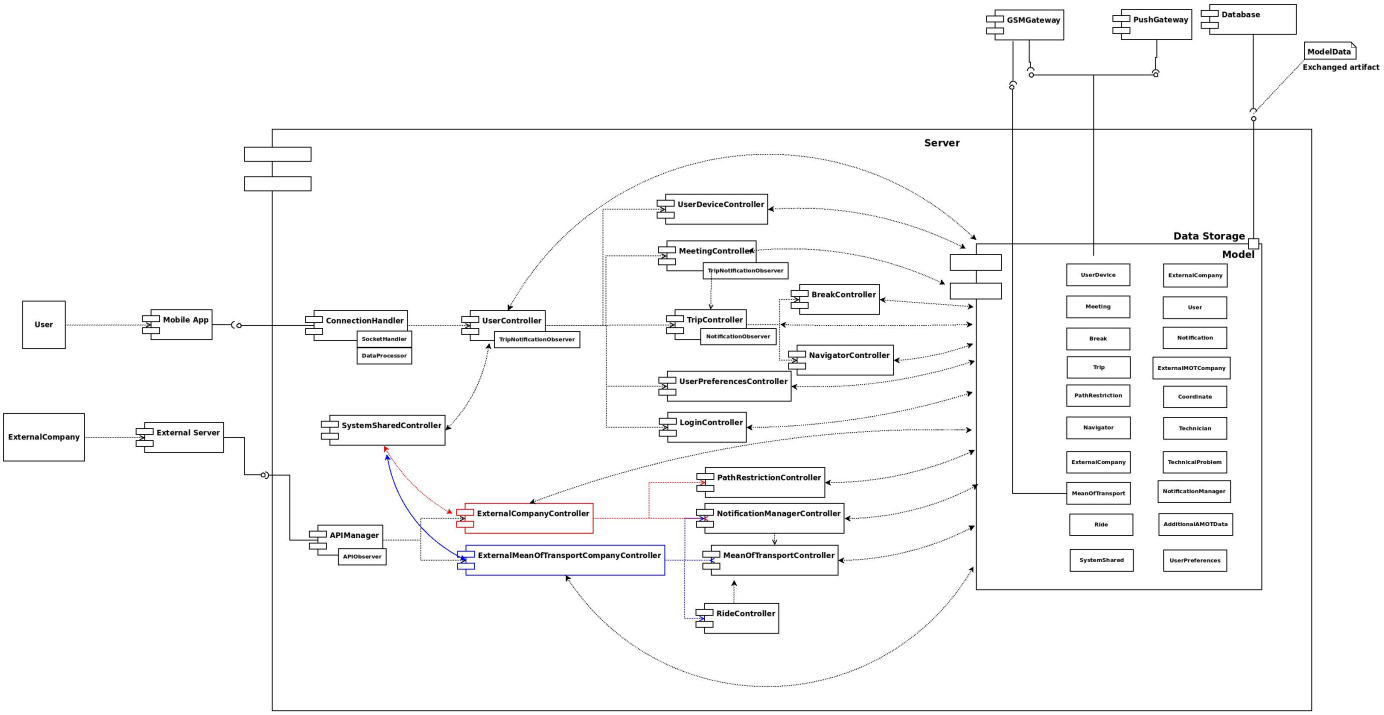

Figure 1: Component to be integrated

And, moreover, we haven't rapresented subclasses , because their behaviour is in a deeper level.
We have decided to do so because design patterns are already (by nature) well projected pieces of code and they're not strictly relative to our application but they're only adapted for it.
We can also see that middle level components, such as controllers, don't need the lower components to be fully implemented when tested.
This because each class forming a component needs only a subset of functionality provided by lower level classes.

2.8 Integration Testing Strategy