Travlendar+ project Daverio Pietro,
Fiorillo Alessandro

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Daverio Pietro, Fiorillo Alessandro
Version:	1.0
Date:	26-November-2017
Download page:	https://github.com/FiorixF1/DaverioFiorillo
Copyright:	Copyright © 2018, Daverio, Fiorillo @ Politecnico di Milano

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this derivable is to provide an architectural design solution for Travlendar+ that fulfils all functional and non-functional requirements expressed in the linked Requirement Analysis and Specification Document.

The following document will also provide an overview of an implementation, integration and test plan solution, bounded with this derivable.

The target audiences of this paper are developers, testers teams and analysts involved in the project.

1.2 Scope

Travlendar+ is a calendar-based application thought to support user thorough his appointment scheduling and transport management.

The system will provide to user:

- the possibility to insert scheduled appointments
- the insertions, modification and organization of different types of preferences
- registration and login in order to access backup/sync options, beside a more advance preferences management
- different route choices to reach the location of appointments

(for further details look at RASD document).

The system will offer his services to user through a Web GUI and a mobile application supporting different operating systems and device format.

1.3 Definitions, Acronyms, Abbreviations

- *RASD* **R**equirement **A**nalysis and **S**oftware **S**pecification
- *DD* **D**esign **D**ocument
- *Appointment*
- *Route*
- *Path*
- *MA*
- *GUI* **G**raphic **U**ser **I**nterface
- *WebUI* **W**ew **U**ser **I**nterface

1.4 Revision History

1.5 Reference Document

- *Travlendar+ RASD document*
- *"Mandatory_Project_Assignment.pdf"* : assignment given of the project

1.6 Document Structure

1 - Introduction The first section offers an overview on the content of the following document, highlighting the purpose of this derivable and recalling a brief description of the problem itself. It also contains references to other documentation linked to this project.

2 - Architectural Design Firstly, he architectural design section presents an overview of a proposed system architecture to accomplish RASD specification.
Secondly, it steps into the system identifying behaviour of components, their interfaces and interoperability with other components, inside or outside the system.
Finally, it explains the thought behind design choices and it gives a list of patterns employed.

3 - Algorithm Design This section provides a list of the most significant algorithms, used by system, expressed in object oriented pseudo-code. They are given in order to specify some critical operation steps.

4 - User Interface Design This section offers a look upon the user interfaces in order to give a good representation of how the UI will look like to users by mean of interfaces mock-up. Furthermore, it' offered a deeper inquiry on the user interaction with the system through UX and BCE diagrams.

5 - Requirement Traceability Section implied in the traceability purpose of requirements, defined in RASD document, with components identified by current derivable in order to increase the observability of requirements fulfilment in following parts of system development and testing.

6 - Implementation, Integration and Test Plan It defines the strategy and provides a sequential plan for the implementation, integration and test processes, describing the sequence in which components are developed, integrated and tested together.

7 - Effort Spent Appendix showing the commitment required by the project to the team.

2 Architectural Design

2.1 Overview

2.2 Component view

2.3 Deplyment view

2.4 Runtime view

2.5 Component Interfaces

2.6 Selected architectural styles and patterns

2.7 Other design decisions

Here you can see how to include an image in your document.

Here is the command to refer to another element (section, figure, table, ...) in the document: *As discussed in Section* **??** *and as shown in Figure 1, ...* Here is how to introduce a bibliographic citation [1]. Bibliographic references should be included in a `.bib` file.

Table generation is a bit complicated in Latex. You will soon become proficient, but to start you can rely on tools or external services. See for instance this https://www.tablesgenerator.com.

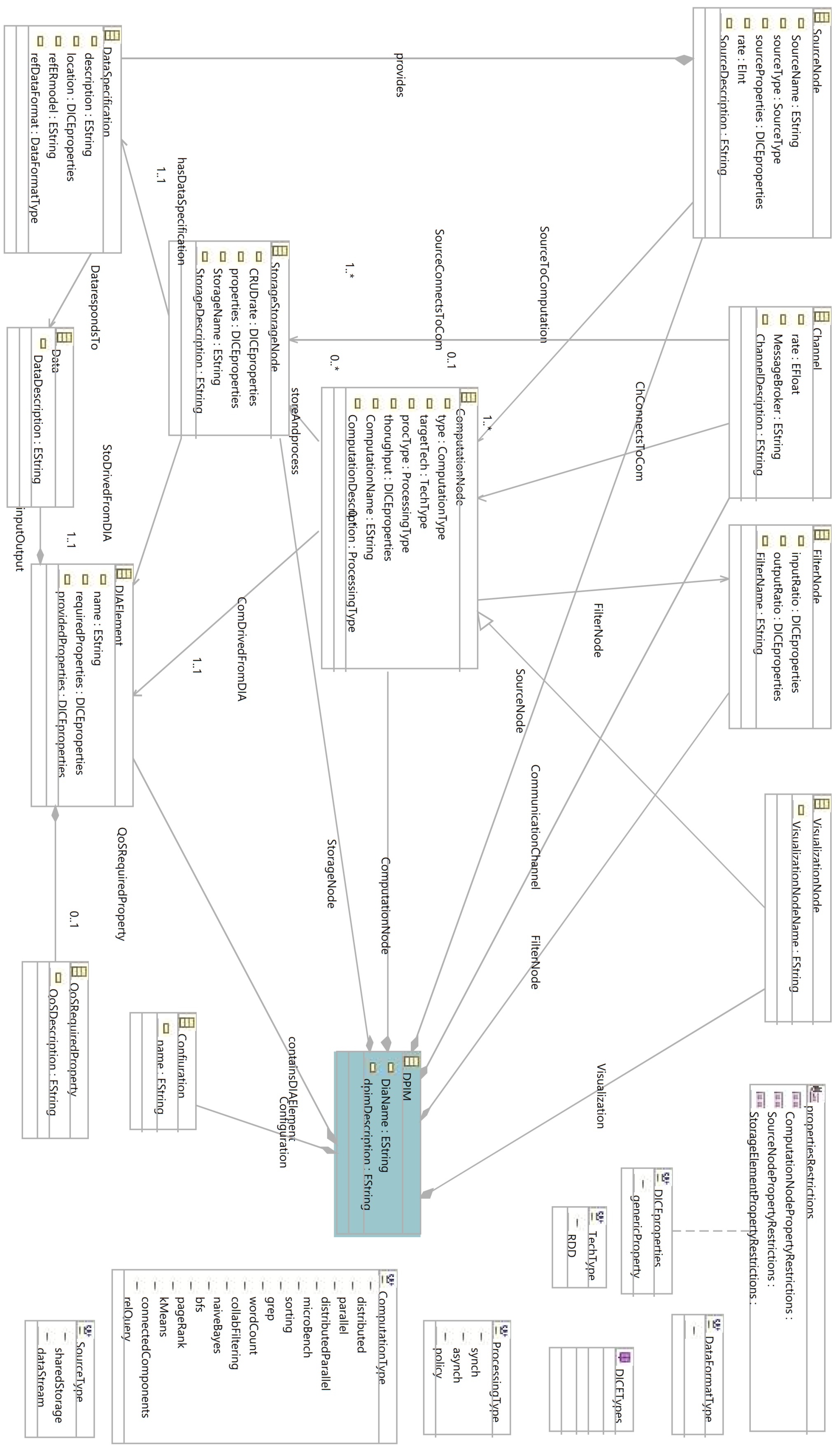

Figure 1: DICE DPIM metamodel.

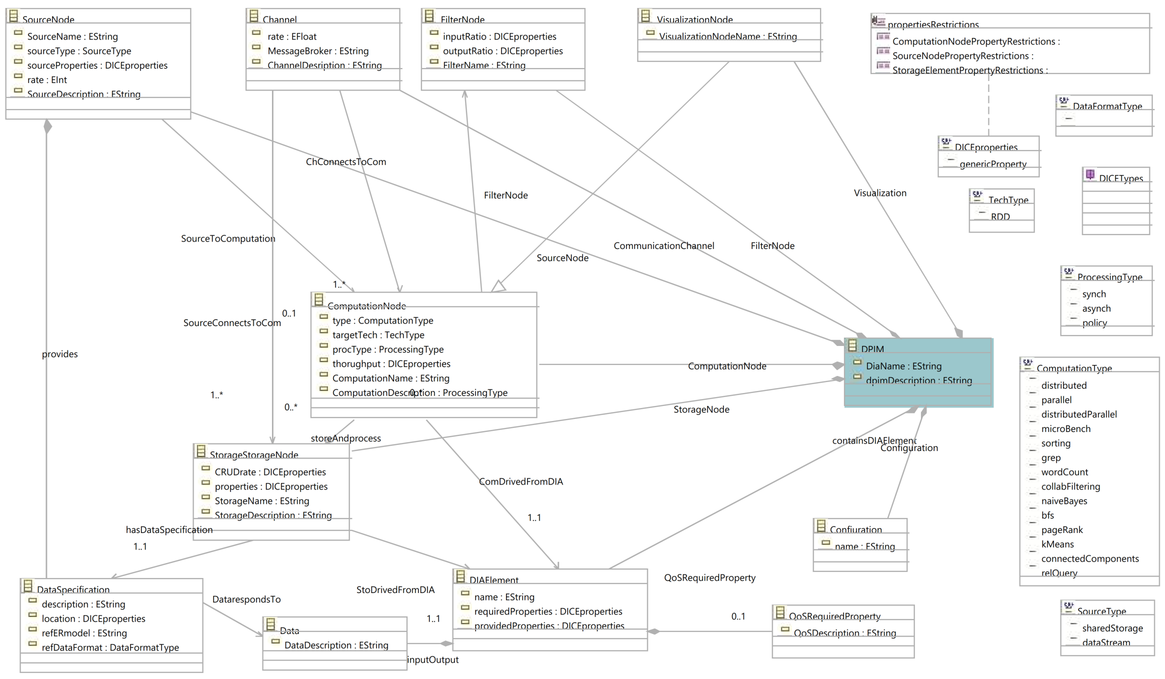

Figure 2: DICE DPIM metamodel in portrait form.

3 Algorithm Design

Organize this section according to the rules defined in the project description.

4 User Interface Design

Organize this section according to the rules defined in the project description.

5 Requirements Traceability

The purpose of this design project is to build up a system able to fulfil all requirements proposed in the bounded RASD document.

In the following section, it is reported the list of project's goals, and for each of them is given the set of components involved in related operations.

G1 Allow the user to add an appointment

[G1.1] The user can add the date of the appointment through a calendar

- Web Application
- Mobile Application

[G1.2] The user can select the location through a map

- Web Application
- Mobile Application
- SessionManager
- CalendarManager
- Map Services

[G1.3] The appointment must be processed by the system

- SessionManager
- CalendarManager
- Web Application
- Mobile Application

G2 - Provide a route to the user for reaching the appointments

[G2.1] The user must reach on time his/her appointments

- CalendarManager
- RouteManager
- Routing Services

[G2.2] The user can choose the starting location and time of the route

- Web Application
- Mobile Application
- SessionManager
- CalendarManager
- RouteManager

[G2.3] Generate routes according to the preferences of the user

- RouteManager
- PreferenceManager

- Database

[G2.4] The application provides a route for each objective, minimizing each of these attributes: length, duration, number of changes, carbon footprint

- RouteManager
- PreferenceManager

[G2.5] Always provide a route

- RouteManager
- CalendarManager
- NotificationManager
- Push Gateway

[G2.6] During strike days, public transportation must not be available

- CalendarManager
- ReportManager

[G2.7] Report unfavourable weather conditions

- NotificationManager
- Push Gateway

[G2.8] Update unfavourable weather conditions

- NotificationManager
- Weather Services

[G2.9] The user can save one route among the shown ones

- Web Application
- Mobile Application
- SessionManager
- CalendarManager
- Database

G3 Allow the user to sign up into the application

[G3.1] The registration must allow the univocal recognition of the user

- Web Application
- Mobile Application
- SessionManager
- Database

G4 Allow the user to log in

[G4.1] The system allow the login through e-mail and password

- Web Application
- Mobile Application
- SessionManager
- Database

[G4.2] The application allow the login through telephone number

- Web Application
- Mobile Application
- SessionManager
- Database

G5 Allow the user to add his own preferences

[G5.1] The user must be logged

- Web Application
- Mobile Application
- SessionManager
- Database

[G5.2] The preferences are synchronized between the application and the database

- SessionManager
- PreferenceManager
- Database

[G5.3] The user can choose the available transport means

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

[G5.4] The user can add a priority to the means of transport

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

[G5.5] The user can add the maximum length of routes on foot or by bicycle

- Web Application
- Mobile Application
- SessionManager

- PreferenceManager
- Database

[G5.6] For each vehicle the user can choose a time slot of validity

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

[G5.7] The user can set Flexible Lunch

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

[G5.8] The user can add breaks for fixed moments of the day

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

[G5.9] The user can add a private car or bicycle

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

[G5.10] The user can organize his preferences in “Preferences Profiles"

- Web Application
- Mobile Application
- SessionManager
- PreferenceManager
- Database

G6 Allow the user to manage his account

[G6.1] The user must be able to remove appointments and routes

- Web Application

- Mobile Application
- SessionManager
- CalendarManager
- Database

[G6.2] The user must be able to modify appointments and routes

- Web Application
- Mobile Application
- SessionManager
- CalendarManager
- RouteManager
- Database

[G6.3] The user must be able to delete his account

- Web Application
- Mobile Application
- SessionManager
- Database

G7 Allow the user to report events and disservices

[G7.1] The user can report strikes using the application

- Web Application
- Mobile Application
- SessionManager
- ReportManager
- Database

[G7.2] The user can report faults, malfunctions and suggestions

- Web Application
- Mobile Application
- SessionManager
- ReportManager
- Database

6 Implementation, Integration and Test Plan

6.1 Integration conditions

6.1.1 Entry Criteria

Integration and testing processes should be performed on each single unit as soon as the component has been fully developed. In addition, integration of different components should be perform only if all these criteria are satisfied:

- RASD and DD documents can be considered "stable", therefore it isn't expected any other modification of requirements or structure of the system, furthermore both deliverables ought to be distributed to all developer team involved in the project.
- There are same dependencies in integration and testing plan that have to be satisfied in order to guarantee the possibility to perform useful cross-components tests on functionalities, because of intrinsic bounds of the architecture.

    1. *Map Services* and *Route Services* for *RouteManager*
    2. *SMS Gateway* and *Push Gateway* for *NotificationManager*
    3. *Map Services* for *MA* and *WebUI*
    4. *DBMS* for *CalendarManager*, *ReportManager* and *PreferenceManager*

6.1.2 Elements to be integrated

It is possible to distinguish three different categories of components, basing the grouping on set of functionalities covered, their interaction and integration dependencies.

- **FrontEnd components**: set of units involved in the management of interactions between the system and user. Its modules are distributed among web and client tiers.

    1. *Mobile Application*
    2. *Web Application*
    3. *SessionManager*

- **BackEnd components**: units that provides the business logic for all system's features. Entirely located on application tier.

    1. *SessionManager*
    2. *CalendarManager*
    3. *RouteManager*
    4. *PreferenceManager*
    5. *NotificationManager*

- **Services components**: atomic components, without any dependency, each one provides a single basic functionality to the system. They are mainly located on the application tier.

    1. *DBMS*
    2. *Route Services*
    3. *Map Services*
    4. *WeatherServices*
    5. *Push Gateway*
    6. *SMS Gateway*

6.2 Implementation, Integration Strategy

Given the foretold component categorization, it is possible to easily set up a bottom up strategy for the integration of implemented components. This plan aims to reduce the whole testing effort needed to deploy complex drivers and stubs employed in the integration process.

The following integration plan will give higher priority to implementation, integration and unit test to components that doesn't rely on undeveloped ones. According to that scheme, the first developing iteration will focus on *services components*, followed by units with less unsatisfied dependencies.

It is also possible to parallelize large part of the development process working on parallel branches of the integration sequence diagram, smoothing the entire development process, without modifications to the integration and testing strategy.

Actually, the development of some component parts can be developed separately and asynchronously respect the unit itself. This practise is encouraged in order to offer the possibility to testing vertically some core functionalities of the system before the end of development. Further details on thread testing strategy in 6.2.2.

It is provided a list of functionalities and unit parts that can be implemented and integrated interdependently for testing purpose:

- ***Registration and Login***

  Web Application: UI with login/registration functionalities

  Mobile Application: UI with login/registration functionalities

  SessionManager: login/registration and client communication implemented

  DBMS: implemented

- ***Appointment management*** (add/remove/modify an appointment) without routes definition

  Web Application: UI with appointment management functionalities

  Mobile Application: UI with appointment management functionalities

  SessionManager: appointment management and client communication implemented

  CalendarManager: appointment management implemented

  DBMS: implemented

- ***Preference management*** (add/remove/modify preferences), requires *Registration and Login*

  Web Application: UI with preference management functionalities

  Mobile Application: UI with preference management functionalities

  SessionManager: preference management and implemented

  PreferenceManager: preference management implemented

  DBMS: implemented

6.2.1 Integration sequence

Directed arc from component-A to component-B is intended as a dependency of component-B on functionalities offered by component-A. The implementation, integration and testing scheme starts from the bottom with first to be developed units, and propagates to the top, stepping forward only if all dependencies are satisfied.

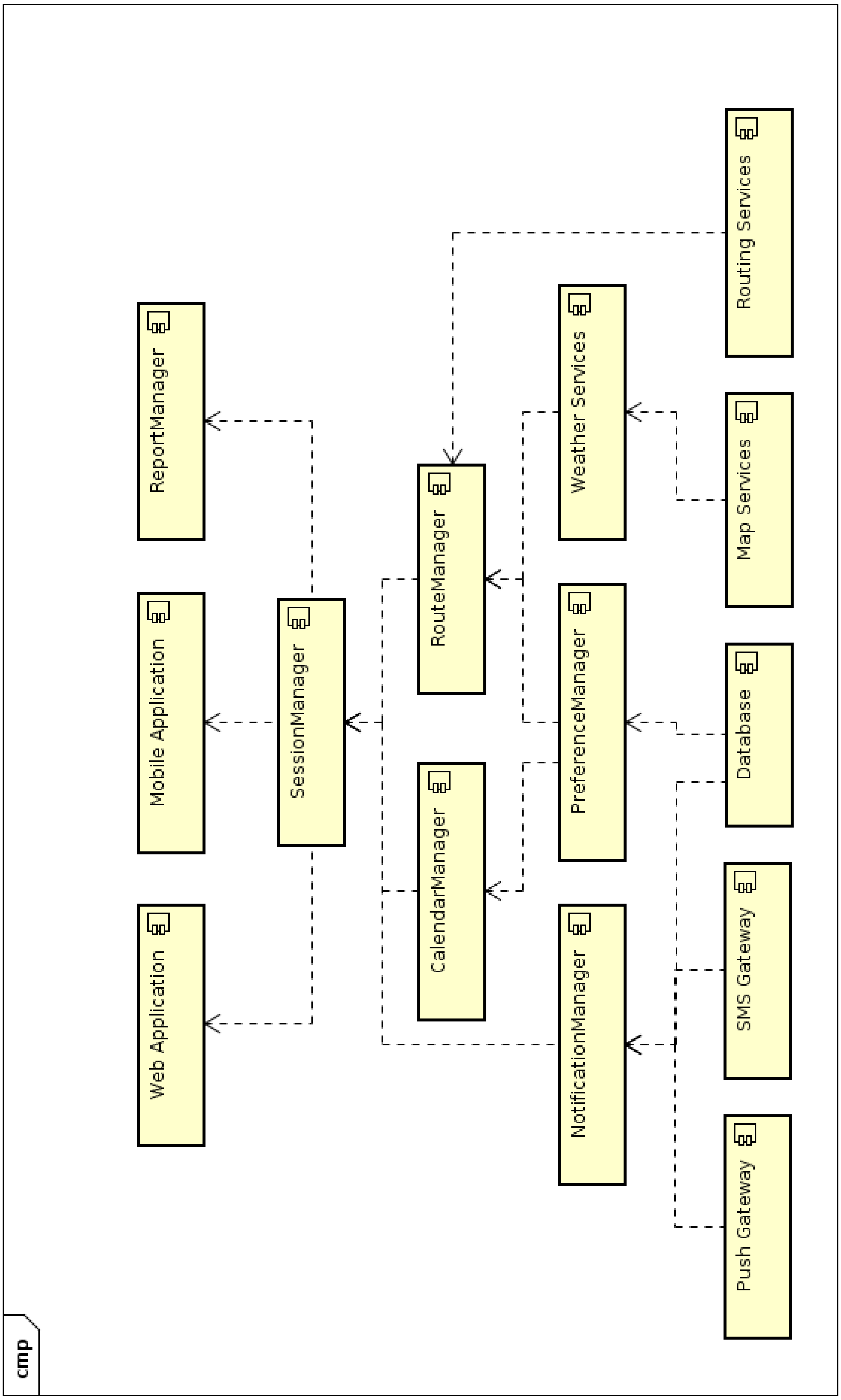

Figure 3: Bottom-Up integration and testing components diagram.

6.2.2 Test Strategy

While the system is suppose to be developed mainly through a bottom-up strategy, testing would be performed on each component as soon it is developed. Furthermore, it will also be verified the integration among component and all his dependency through specific integration tests.

For some functionalities it is suggested to make exception because, for these ones, it is possible:

- to split the deployment effort required by larger components, increasing atomicity of architecture unit in subcomponents;
- to anticipate the verification of some core features, that the system provides to user, far before the system test;
- to parallelize the development of an otherwise monolithic part of the implementation plan (particularly The *SessionManager* module)

Functionalities should be verified as soon as needed components are available (list of goals and involved components is provide in *Requirement Traceability* section 2.7).

7 Effort Spent

Daverio

20/11/2017	1h15m
21/11/2017	1h30m
22/11/2017	2h45m
24/11/2017	1h
25/11/2017	3h
26/11/2017	7h
Total	16h30m

Fiorillo

15/11/2017	1h30m
16/11/2017	1h30m
17/11/2017	1h30m
18/11/2017	3h
22/11/2017	1h
23/11/2017	1h30m
24/11/2017	2h50m
25/11/2017	6h
26/11/2017	1h30m
Total	20h20m

References

[1] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software and Systems Modeling*, 10(3):313–336, 2011.