System design document for ResuMate

Contents

Version: ResuMate 1.0

Date: 26/5-2013

Author: Danny Lam, Sara Nadi, Patricia Paulsson & Laszlo Sall Vesselenyi

This version overrides all previous versions.

1 Introduction

1.1 Design goals

Our objective is to attain a robust and clean design which may be accomplished by the usage of the MVC-design pattern. The interaction between the different components of our program should be attained in a way such that no complications are included. By just a glance at our design a client should be able to comprehend how the connections are distributed in the system. For Usability, see RAD.

1.2 Definitions, acronyms and abbreviations

RM, ResuMate, our application's name.

GUI, graphical user interface.
Java, platform independent programming language.
JRE, the Java Run time Environment. Additional software needed to run an Java application.
MVC, model-view-controller
iText, the external library that handles PDF creation

2 System design

2.1 Overview

The MVC-design pattern is used since it is a refined way to interact between classes of logic and with ones of view involving minimal dependency. We have used a passive kind of MVC where the controllers delegate between model and view to reduce dependencies even more.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the following modules, see Figure #.

- main, holds application entry point
- views, GUI for application
- model, model part of the MVC model
- utils, includes utility classes used by multiple other packages
- controller, the control classes for the MVC model
- io, for file and PDF handling

2.2.2 Decomposition into subsystems

The only subsystem is the file handling in package io (not a unified subsystem, just classes handling io).

2.2.3 Layering

1. Model
2. IO
3. Controllers
4. Utils

5. Views

See figure below (figure 1).

2.2.4 Dependency analysis

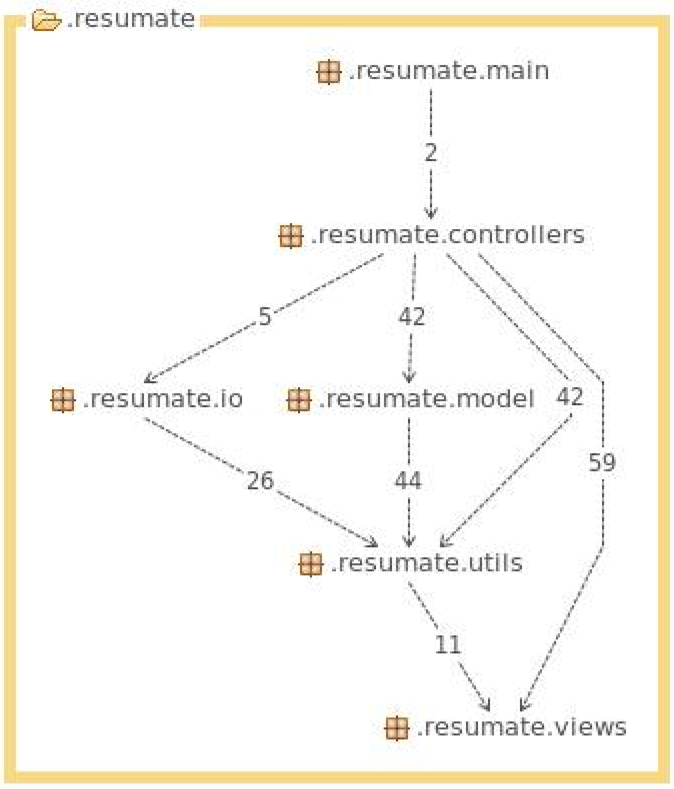

Figure 1

The program is shown above very dependant of the Utils package.

2.3 Concurrency issues

No, we have not used threads. Java Swing is single threaded and thus no concurrency issues may appear.

2.4 Persistent data management

Documents can either be saved as a project folder or exported as a PDF.

- Saved project folders can be reopened by the application and are only useful to the user for storing work.
- A PDF of a document cannot be opened by the application, but can on the other hand be used by the user to distribute the document to other parties for inspection.

2.5 Access control and security
Not applicable.

2.6 Boundary conditions
Our application will be launched as normal desktop application meaning using script (jar).

3 References

1. MVC, see http://en.wikipedia.org/wiki/Model-view-controller

APPENDIX

Package diagram

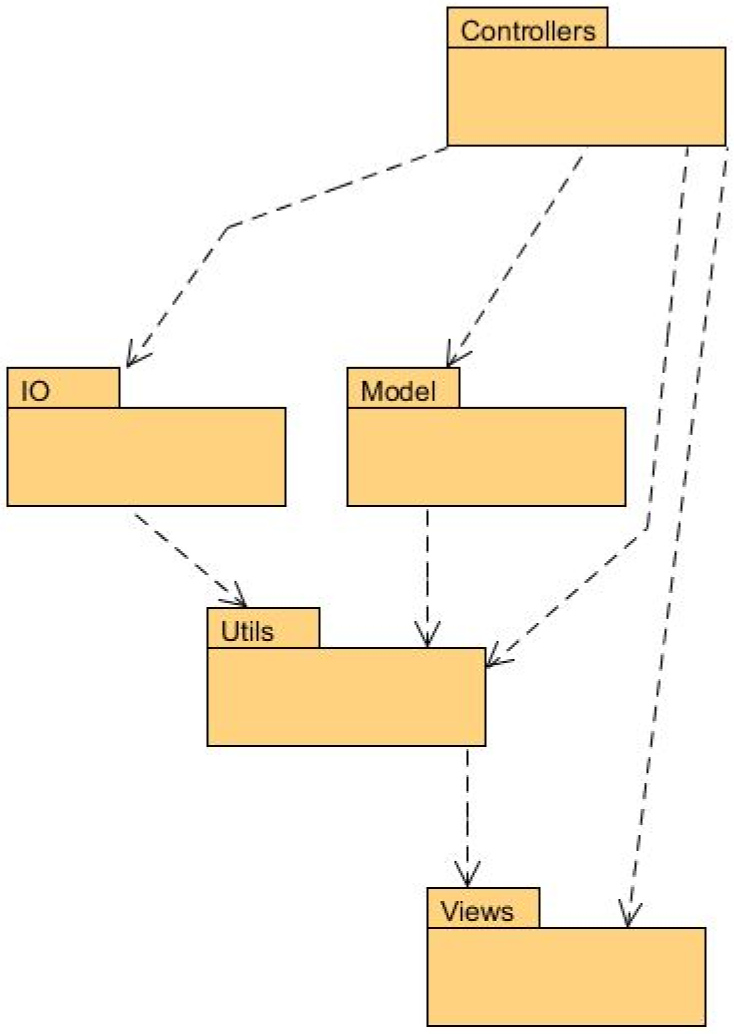

Figure 2

Class diagram

Orange: model package
Blue: controllers package
Green: io package
Pink: views package

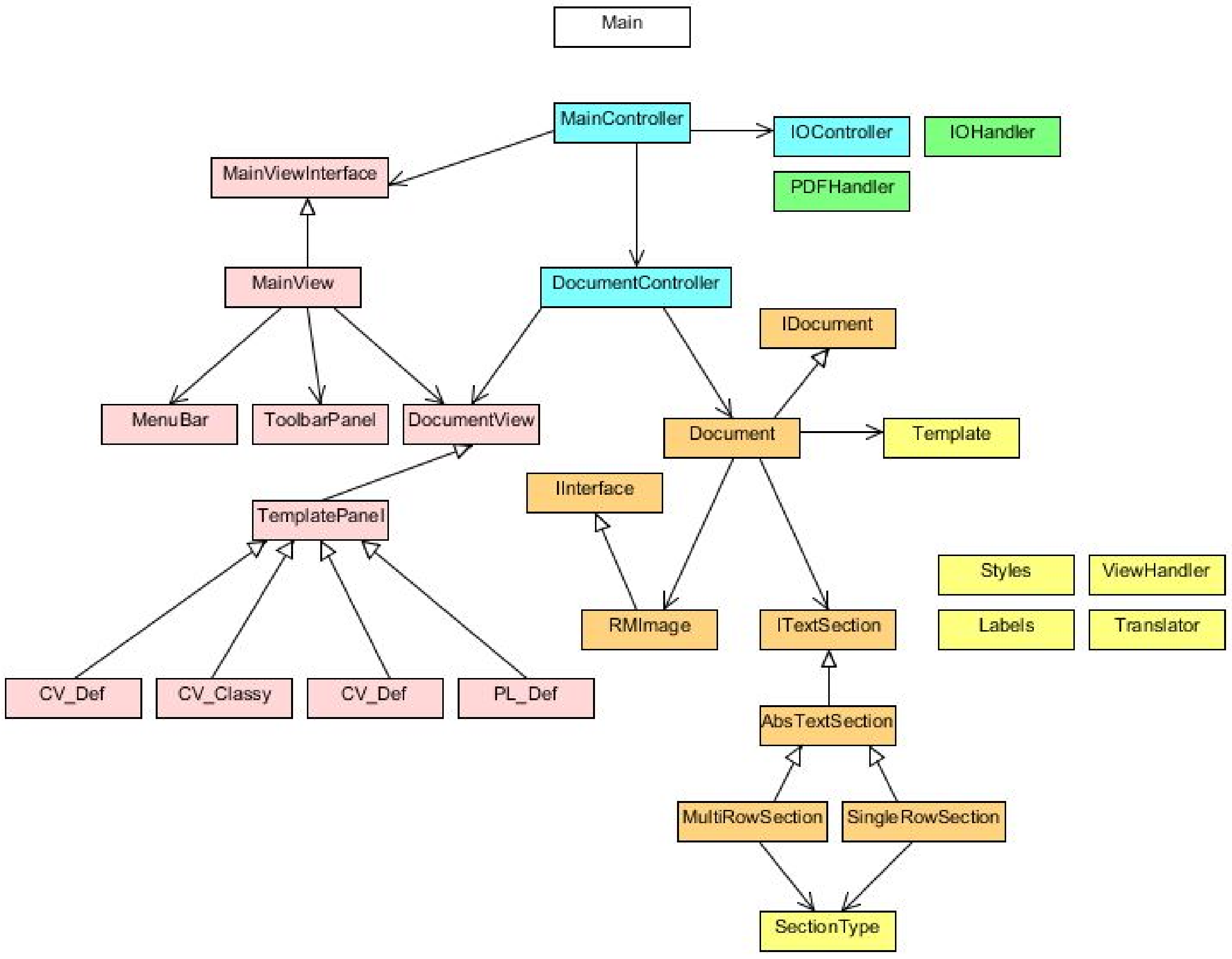

Figure 3

Package diagram

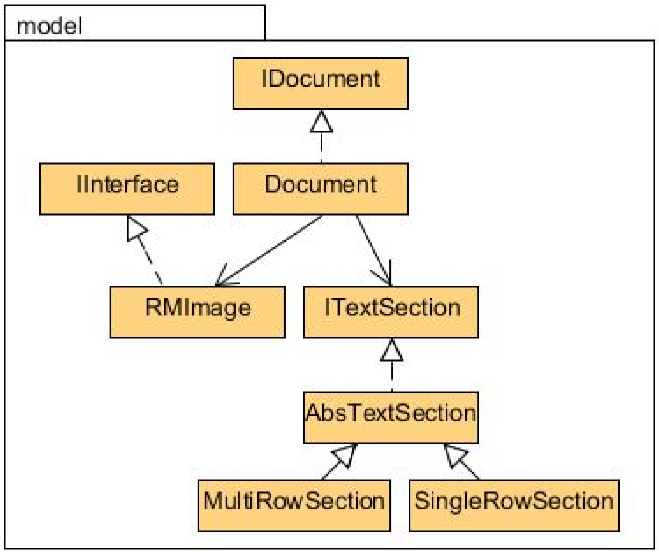

Figure 4

Class diagram

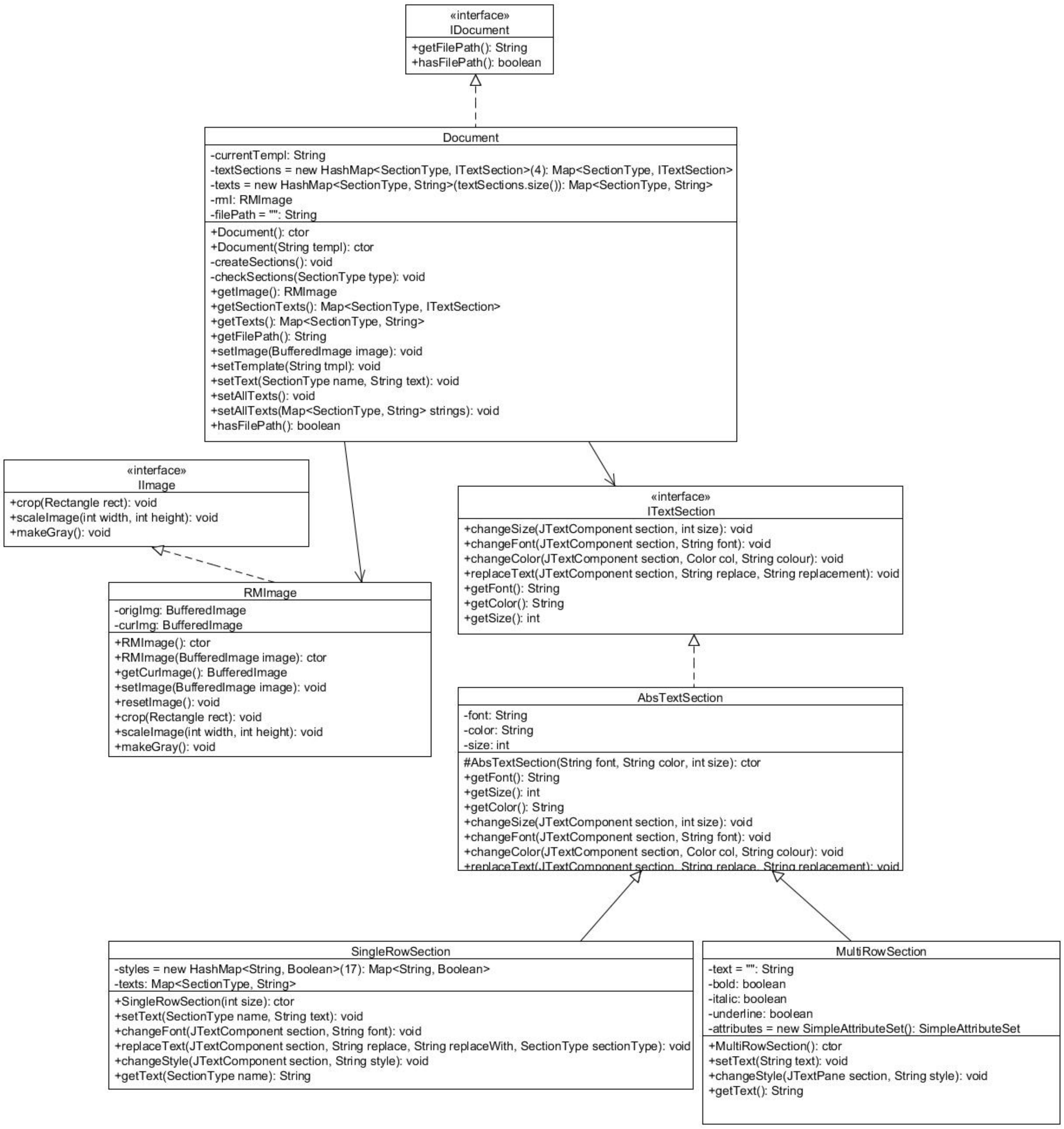
«interface»
IDocument
+getFilePath(): String
+hasFilePath(): boolean
Document
-currentTempl: String
-textSections = new HashMap<SectionType, ITextSection>(4): Map<SectionType, ITextSection>
-texts = new HashMap<SectionType, String>(textSections.size()): Map<SectionType, String>
-rml: RMImage
-filePath = "": String
+Document(): ctor
+Document(String templ): ctor
-createSections(): void
-checkSections(SectionType type): void
+getImage(): RMImage
+getSectionTexts(): Map<SectionType, ITextSection>
+getTexts(): Map<SectionType, String>
+getFilePath(): String
+setImage(BufferedImage image): void
+setTemplate(String tmpl): void
+setText(SectionType name, String text): void
+setAllTexts(): void
+setAllTexts(Map<SectionType, String> strings): void
+hasFilePath(): boolean
«interface»
IImage
+crop(Rectangle rect): void
+scaleImage(int width, int height): void
+makeGray(): void
RMImage
-origImg: BufferedImage
-curImg: BufferedImage
+RMImage(): ctor
+RMImage(BufferedImage image): ctor
+getCurImage(): BufferedImage
+setImage(BufferedImage image): void
+resetImage(): void
+crop(Rectangle rect): void
+scaleImage(int width, int height): void
+makeGray(): void
«interface»
ITextSection
+changeSize(JTextComponent section, int size): void
+changeFont(JTextComponent section, String font): void
+changeColor(JTextComponent section, Color col, String colour): void
+replaceText(JTextComponent section, String replace, String replacement): void
+getFont(): String
+getColor(): String
+getSize(): int
AbsTextSection
-font: String
-color: String
-size: int
#AbsTextSection(String font, String color, int size): ctor
+getFont(): String
+getSize(): int
+getColor(): String
+changeSize(JTextComponent section, int size): void
+changeFont(JTextComponent section, String font): void
+changeColor(JTextComponent section, Color col, String colour): void
+replaceText(JTextComponent section, String replace, String replacement): void
SingleRowSection
-styles = new HashMap<String, Boolean>(17): Map<String, Boolean>
-texts: Map<SectionType, String>
+SingleRowSection(int size): ctor
+setText(SectionType name, String text): void
+changeFont(JTextComponent section, String font): void
+replaceText(JTextComponent section, String replace, String replaceWith, SectionType sectionType): void
+changeStyle(JTextComponent section, String style): void
+getText(SectionType name): String
MultiRowSection
-text = "": String
-bold: boolean
-italic: boolean
-underline: boolean
-attributes = new SimpleAttributeSet(): SimpleAttributeSet
+MultiRowSection(): ctor
+setText(String text): void
+changeStyle(JTextPane section, String style): void
+getText(): String

Figure 5

Sequence Diagrams

Use Case: Change Size

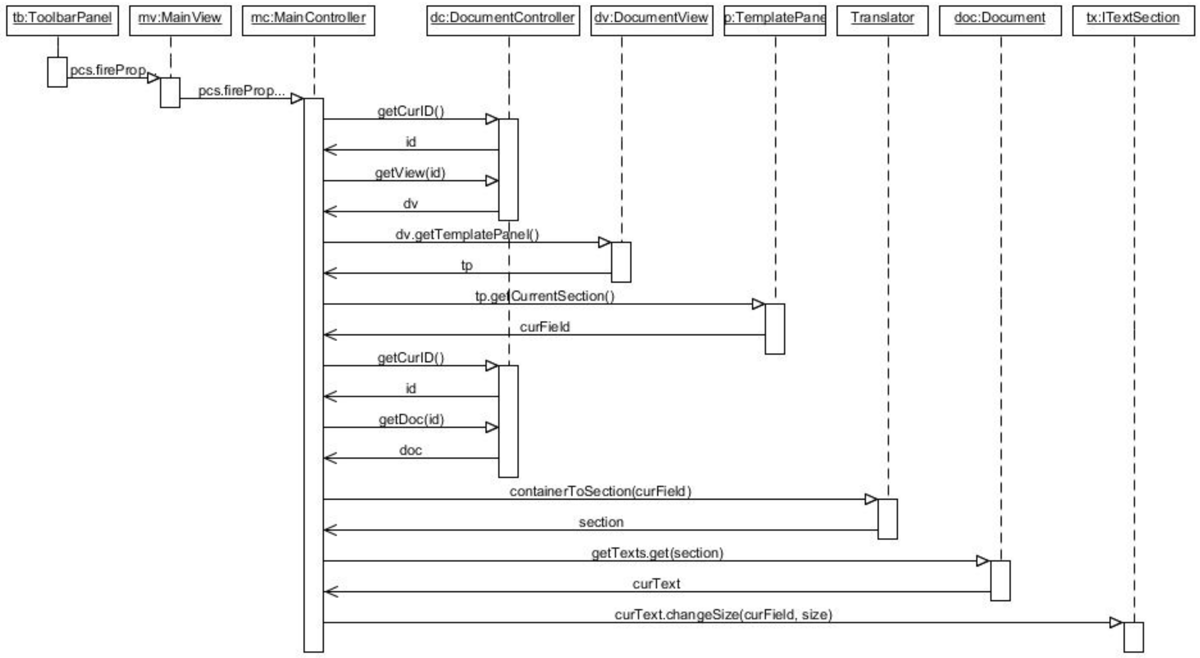

Figure 6

Use Case: Grayscale Image

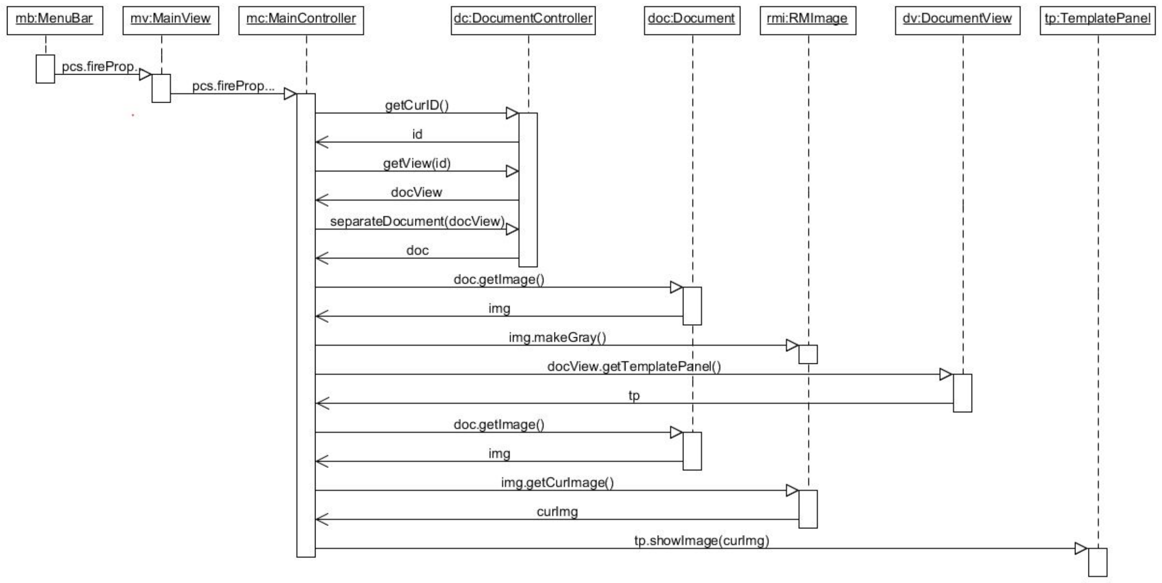

Figure 7