SDD: Project Platypus

Hidstrand, Fredrik
Keinestam, Johannes
Sjöqvist, Magnus
Thander, Fredrik

TDA367
Chalmers University of Technology
10 April 2011

Table of Contents

System design document for Project Platypus

Version: 2.0
Date: 2011-05-24

This version overrides all previous versions

1 Introduction

This section provides a brief overview of the project.

1.1 Design goals

The design must have loose couplings between the different parts of the application so that mainly the GUI is easily replaceable. The application must also feature testable parts (modules, classes). It should be easy to write new filters and incorporate them into the application.

1.2 Definitions, acronyms and abbreviations

- GUI, graphical user interface.
- MVC, a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI code, application code and data spread all over.
- Plugin, an optional extension to the program that adds additional functionality.
- Filter, a plugin with a set of adjustments to the chosen images, for instance increase contrast.
- Preset, a collection of predefined filters.

1.3 References

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

2 Proposed system architecture

In this section we propose a high level architecture.

2.1 Overview

The application will use a standard MVC model with an anemic model. This will make the controller handle nearly all the logic, with the model acting just as data. Testing of our application will therefore be centralized to the controller.

2.1.1 Filters

The application will be very extendable, in that it is possible to add custom-made filters. The filter will contain its own panel for the filter settings. The filters will be stored in the users home folder.

2.1.2 Event handling

The application will have a central location for the communication between the modules. This communication will be located in a class named *ComBus*.

2.2 Software decomposition

2.2.1 General

The application is composed of the following modules, see figure 2:

- Main, application entry and build class.
- view, top level package for all the GUI-classes.
- view.gui, GUI classes with little logic and mainly auto-generated code.
- view.splash, logic concerning splash screen.
- ctrl, controller classes containing all logic.
- model, the model containing data.
- util, general utilities.

2.2.2 Tiers

The application will always function as a standalone client on the user's computer.

2.2.3 Communication

N/A

2.2.4 Decomposition into subsystems

N/A

2.2.5 Layering

See chapter 2.2.1 and figure 1.

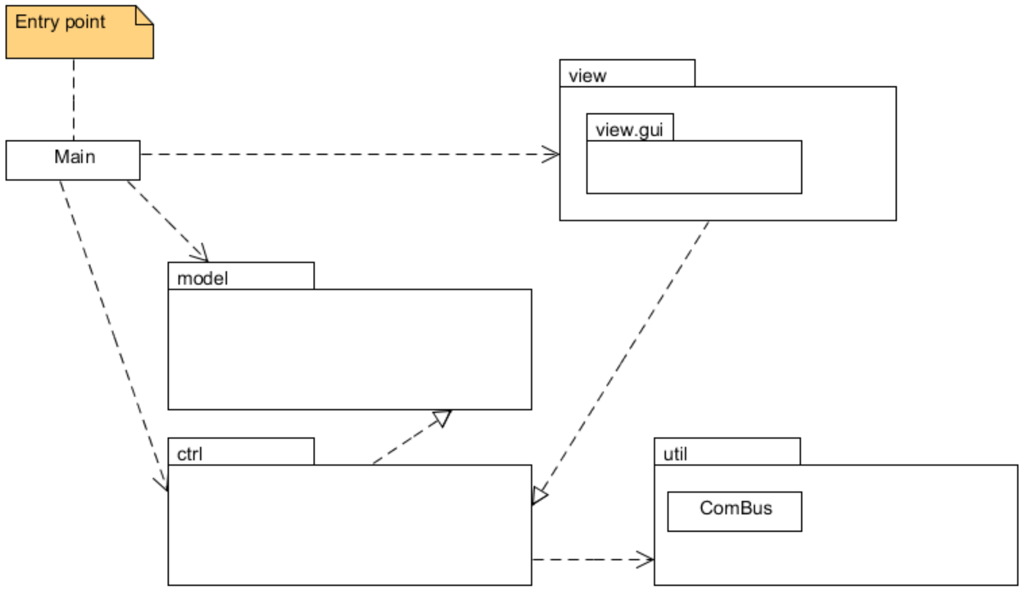

Figure 1: Packages and dependencies

2.2.6 Dependency analysis

See figure 1. No circular dependencies, except between packages *view* and *view.gui*. Since *view.gui* is a subpackage to *view* this does not constitute a problem, especially since they are so closely related logic-wise.

2.3 Concurrency issues

The process of applying the filters to the batch of images will be threaded to decrease waiting time. Some filters may be prone to bugs, which we handle with an option for the user to abort the operation exporting of images. If aborted, the current saving and applying of filters will stop, and the images already processed are saved.

2.3.1. Performance

Because PlatyPix handles performance heavy duties, it has to run in a JVM that can allocate more memory than usual. The startup script starts the program in a Java machine with 1GB RAM. The memory usage can be very high during the last step, when the full-size images are rendered and saved. A performance analysis was done in a Java profiler, see figure 2.

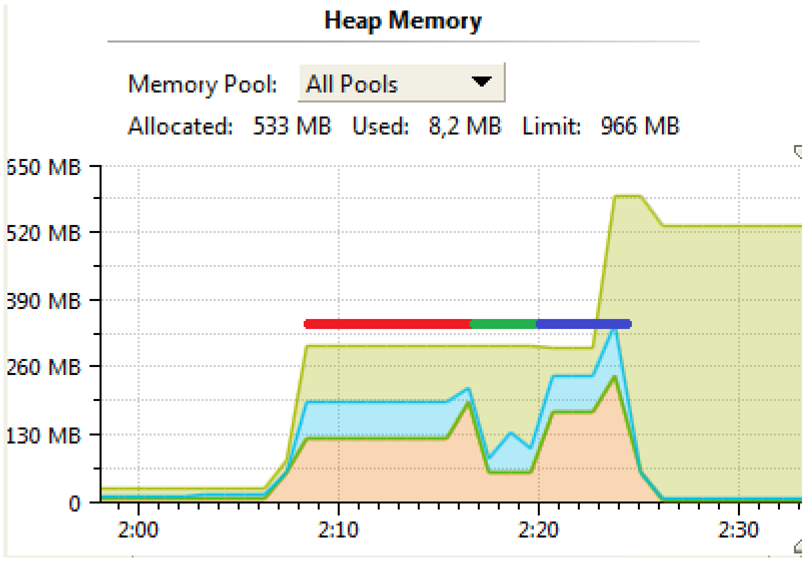

Figure 2: Screenshot from YourKit Profiler

The graph is of memory usage during the save operation of a 10 Megapixel photo, with three filters being applied (two of which are performance heavy). The green part of the graph represents the memory allocated by the JVM, while the blue part is the total of used memory. The marking colors shows the following time lines: red is Blur filter, green is RGB channels filter and blue is Sharpening filter. Blur is very time consuming, but not as performance consuming as the sharpening filter. Memory usage above these levels are not to be expected, but possible.

2.4 Persistent data management

The persistent data will be the presets that the user may choose to save. These presets will be saved in separate data files at a specific location in the users home folder.

2.5 Access control and security

By giving a lot of freedom to the contents of a filter we’re running the risk of them containing malicious code. It has been discussed and the outcome is that the application will not enforce a security system; instead it is the responsibility of the user.

2.6 Boundary conditions

Since the program only has one tier, opening and closing the application will function as it normally would.

2.7 References

N/A