

ENGG*4450 : Large Scale Architecture

Design Document

Adam Erb
Gregor Scott



CONTENTS

1	Purpose	1
2	General Priorities	2
3	Outline of the Design	2
4	Major Design Issues	2
4.1	Storing and arranging images	2
4.2	Dealing with duplicate entries	3
4.3	Keeping low coupling between packages	3
5	Design Details	3
5.1	Chosen Digital Library Architecture	3
5.2	Architecture Specifics	3
5.3	Interface	4
5.4	Features Implemented	4
5.4.1	Creating and Opening new Libraries	4
5.4.2	Saving Libraries	5
5.4.3	Adding & Removing Images	5
5.4.4	Adding & Removing Tags	5
5.4.5	Arranging by Tag	5
5.4.6	Viewing Images	5
5.4.7	Re-Saving Images in New Formats	5
5.4.8	Scale Operation	5
5.4.9	Crop Operation	5
5.4.10	Contrast & Brightness Adjustment	6
5.4.11	Error Checking	6
6	References	6

1 PURPOSE

This purpose of this design document is to outline the implementation of a personal Digital Image Library (DIL) for the purpose of our ENGG*4450 term project. This document will outline all details of the final design, and design process in producing this project. The final product has included all of the features stated in the Software Requirements Specification (SRS) document, and in any areas which differ between the two documents, a detailed justification will be outlined.

2 GENERAL PRIORITIES

To arrive at the design solution we did we considered several options by for arriving at the final design. We measured these design options by means of the following criteria:

- 1) Functionality
- 2) Performance
- 3) Reusability
- 4) Degree of Coupling
- 5) Overall Cohesion

3 OUTLINE OF THE DESIGN

The chosen design, as stated in the SRS document, functions inside of the Java Runtime Environment (JRE). In the final design, the project was spilt up into 3 distinct packages, being:

- 1) the DIL
- 2) the Graphical User Interface (GUI)
- 3) the open-source image processing library JHLabs (*see References*)

The final design attempted to reduce coupling between the three packages as much as possible. An overview of the updated package interrelation from the SRS can be seen below in figure 1.

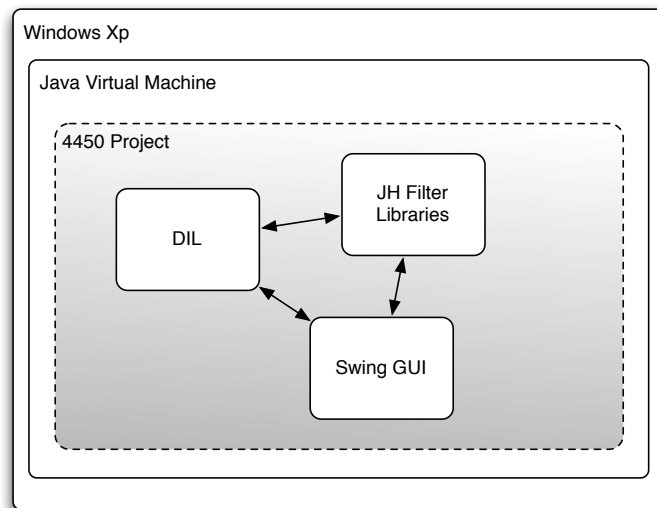


Fig. 1: Class Diagram of System.

4 MAJOR DESIGN ISSUES

4.1 Storing and arranging images

Of course the immediate problem encountered by the SRS is: how do we arrange the library objects within the digital library?

Opt 1. String Based Lists

The first design we considered was using simple string based lists inside the library, to keep track of tags and images. The advantage to this was simplicity, compatibility with the GUI! (GUI!), since this is ultimately the format the GUI needs to be in, and easy organization. The drawback however is that any logic defining a link between an image and a tag has to be created independent of these classes, and in this way increases coupling.

Opt 1. Object Based List

This design alternative was what was ultimately chosen. It supported Image and Tag objects which are pointed to by the library, but point to one another respectively. The advantage to

this option is the relationships between the each object can easily be defined in the object classes, minimizing the work of calculating dependancies. The drawback is determining out to index and compare these objects, which is outlined in the next section.

4.2 Dealing with duplicate entries

One issue encountered in the project was deciding how to handle duplicate Images & Tags, and deciding by which metrics to define a duplication. In the instance of tags, this is fairly simple, tags should be considered duplicate when the tag name is the same. The problem which arises is all Object based lists in java index by Object pointer, so it would be easy to accidentally insert a new tag with the same name. To resolve this the **LibraryList<LibraryObject>** (an extension of **LinkedLists**) was created which accepts non-primitive data types, but still indexes based on the object string name. In this way the library can organize, and avoid duplication of entries, for either tags, or images. Originally, in the early design, two separate List types were defined for images and tags respectively. It was recognized early on that this was leading to duplicated code, and for this reason the two list types were combined.

4.3 Keeping low coupling between packages

Another issue that arose was how to pass data between packages without encountering stamp coupling, by passing Library data types to the GUI and filter library. To do this all arguments, and return types, were made available as primitive data-types; Mostly strings and boolean values. An alternative to this would be to simply pass the library images between packages. From a short term perspective, this would reduce complexity. Ultimately, the original, primitive data-type design was chosen because of improved scalability and easy debugging.

5 DESIGN DETAILS

5.1 Chosen Digital Library Architecture

The system was designed with a cohesive architecture in mind, that promoted modularity, while minimizing coupling between the units. This was done by splitting up the project into several distinct packages which were independent, short of a few binding classes. These packages were GUI package, the DIL class, and a third party image processing class, grouped for functional cohesion. To further promote reduced coupling these classes interacted primarily through primitive type messages: strings, integers, and boolean values.

An additional feature of the architecture was an emphasis on code reusability by means of generics and interfacing. The two objects defined in the library were **LibraryTag** and **LibraryImage**. Since both were organized and used within the library in similar ways, and both stored inside lists, an interface **LibraryObject** was created so to allow the two to be operated on in the same way. Additionally, a generic **LinkedList** extension **LibraryList<LibraryObject>** was created, which allowed for either of the objects to be stored in the same generic class. Based on **compareTo()** methods defined by the **LibraryObject** interface, either the images or tags could be compared and organized in the **LibraryList<LibraryObject>** based on the comparisons defined within the classes.

The design made use of a number of design patterns as well. The initializing class "GUI" implements the singleton pattern so that only one instance of the GUI can exist at any given time. The Library class also makes heavy use of the delegation pattern for **DigitalLibrary**, to **LibraryObject** traversing.

Based on this design the entire **Library** package, representing all of the logic behind the DIL consisted of only 5 minimal classes. A class diagram of this design can be seen below in figure 2. Additionally, **Javadocs** have been included with this submission outlining the interior methods of the classes which makeup the DIL package.

5.2 Architecture Specifics

To further detail the architecture of this project, a more detailed hierarchy of the objects is outlined below. The **DigitalLibrary** contains two objects of class **LibraryList<LibraryObject>** , which contain all images and tags in the library. **LibraryList<LibraryObject>** is a generic extension of **LinkedList**, which will only accept classes implementing **LibraryObject** , and organizes each **LibraryObject** in the list accordingly. Additionally, both **LibraryImage** & **LibraryTag** contain their own **LibraryList<LibraryObject>** which holds points to, either the tags pointing to the image, or the images being pointed to by the tag. In this way all elements of the **DigitalLibrary** are aware of their relationships.

1x frame :GUI()

1x digLib :DigitalLibrary()

1x allImages :LibraryList<DigitalImage>()

Nx imageN :DigitalImage() implements DigitalObject

1x tagsToImage :LibraryList<DigitalTag>()

1x allTags :LibraryList<DigitalTag>()

Nx tagN :DigitalTag() implements DigitalObject

1x imagesInTag :LibraryList<DigitalImage>()

Nx filterInstance :FilterOperation()

5.3 Interface

The system interface closely resembles what was proposed for the original SRS document. It is composed of a simple GUI through which the user can interact with the library via mouse clicks and some keyboard functionality. One key difference between the final and proposed product was a more limited keyboard usage in the final product than was originally described. Instead of waiting on a user keystroke to confirm a value entered we instead implemented pop-up windows asking for confirmation. This change was simply practical as the project was developed, and was recognized as simpler to build, and more effective from a User Interface (UI) perspective. An outline can be seen of the GUI below in figure 3. The user drop-down menu is also outlined below, with changes from proposed to implemented document **bolded** to indicate addition, and striked-out to indicate removal.

- 1) File
 - a) New Library
 - b) Open Library
 - c) Save As...
 - d) **Images**
 - i) **Add**
 - ii) **Remove**
 - e) ~~Import Photos~~
 - i) ~~Files~~
 - ii) ~~Folders~~
- 2) Tag
 - a) Add
 - b) Delete
- 3) Filters
 - a) Crop
 - b) Scale
 - c) Contrast Adjustment
- 4) Help
 - a) ~~Documentation~~
 - b) **About**

5.4 Features Implemented

5.4.1 *Creating and Opening new Libraries*

As stated in the SRS(3.1.1), the DIL is required to be able to create and open image libraries. This can be done so via the user drop down menus as defined SRS(2.2.1).

To create a new library click: File→New-Library, and using the “Save File” Dialog box, choose a location to save the new library in. Proper extensions are included so there is no need to specify an extension.

To open a new library click: File→Open-Library, and in the same way use the “Open File” Dialog box to find a previously save library file with the .diglib extension.

5.4.2 *Saving Libraries*

This functionality is built into the application and requires no user input. Once a .diglib file has been opened or created, the library will save itself every time a significant event occurs.

5.4.3 *Adding & Removing Images*

The adding and removing of images to this library is another supported feature defined by the SRS(3.1.2). Adding can be accomplished by clicking File→Import-Photo(s). Using the “Open File” dialog box multiple files can be selected for import.

To remove click on one or multiple images, in the File Display (see: figure 3), and then click File→Remove-Photo(s). All photos will be removed from the library.

5.4.4 *Adding & Removing Tags*

Adding and removing tags, to and from images, is another feature supported by the application from SRS(3.1.3). Adding a tag can be achieved by clicking on one or more images in the “File View” and then clicking Tags→Add. A new prompt window will pop-up prompting for input, and the user will enter the new or existing tag to add to the image.

Removing a tag from an images can likewise be accomplished by clicking on one or more image in the “File View”, and then clicking on Tag→Remove. Another prompt will pop-up and the user can enter the tag to be removed from the image.

5.4.5 *Arranging by Tag*

As defined by requirement 3.1.5 in the SRS, the user should be able to view and select image based on tags in the library. This can be done by clicking on various tags in the “Tag View” defined in the section 5.3. Clicking on any tag will populate the “File View” with images that have that tag. Alternatively, if the user clicks “All”, all images will be displayed in the “File View”.

5.4.6 *Viewing Images*

By clicking on any image in the “File View”, a scaled image will appear in the “Image View” defined in section 5.3.

SRS(3.1.5)

5.4.7 *Re-Saving Images in New Formats*

The system also supports re-saving images using different formats (see SRS(3.1.6)). This can be accomplished by clicking File→Save-As... A dialog box will then pop up and prompt the user for what new image format the image will be save in. As described by the SRS these digital formats are:

- .jpg
- .bmp
- .png
- .tiff

5.4.8 Scale Operation

The project also supports basic image filtering operations which consist of scale, crop, & contrast/brightness adjustment. The scale operation can be achieved by clicking on one or more images in the “File View” and then clicking Filters→Scale. A dialog box will then pop up can request a width and height factor by which to scale the image(s). In this operation the scale factor 1 is do nothing, > 1 is scale up by a factor of, and < 1 is scale down by a factor. SRS(3.2.2)

5.4.9 Crop Operation

The user can crop by clicking on one or more photos in the “File Display”, and then clicking Filters→Crop. By clicking and dragging the mouse across the image the user can define the area to crop. A box will appear showing the user the area selected, and prompting the user to either apply the crop or discard. SRS(3.2.1)

5.4.10 Contrast & Brightness Adjustment

The contrast adjustment can be applied by selecting one or more images in the “File Display” and clicking Filters→Adjust-Contrast. A input dialog box will then pop-up prompting the user for factors by which to adjust the brightness and contrast. For both adjustments, an input of 1 is do nothing, < 1 is a reduction in brightness/contrast and > 1 is an increase. SRS(3.2.3)

5.4.11 Error Checking

Additionally, to comply with the 3.4 of the SRS the project implements basic error checking for bad file types, and for bad user input.

6 REFERENCES

- [1] J. Huxtable, “Jh labs java image processing classes,” 2006, licensed under the Apache License, Version 2.0. [Online]. Available: <http://www.jhlab.com/ip/filters/download.html>
- [2] “Ieee recommended practice for software requirements specifications,” *IEEE Std 830-1998*, p. i, 1998.

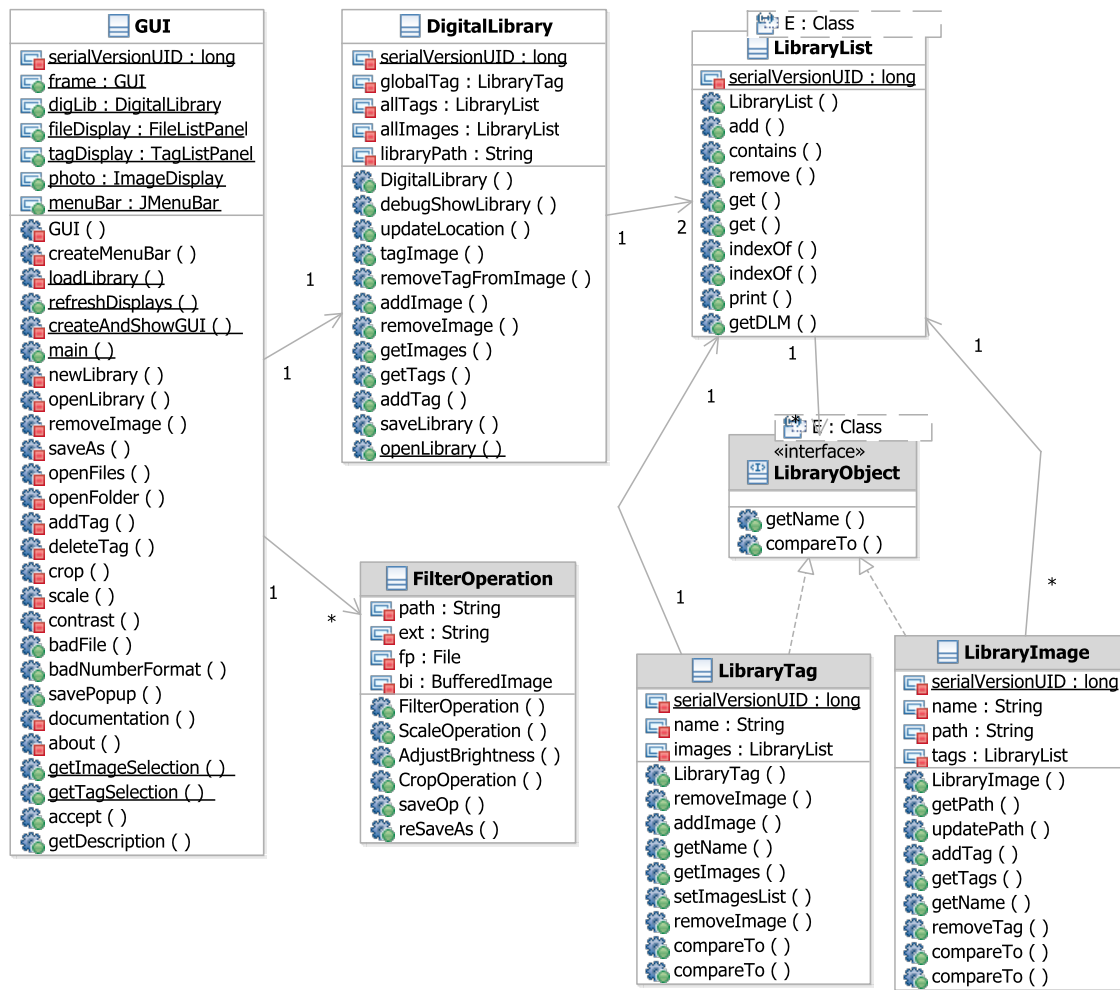


Fig. 2: Class Diagram of System.

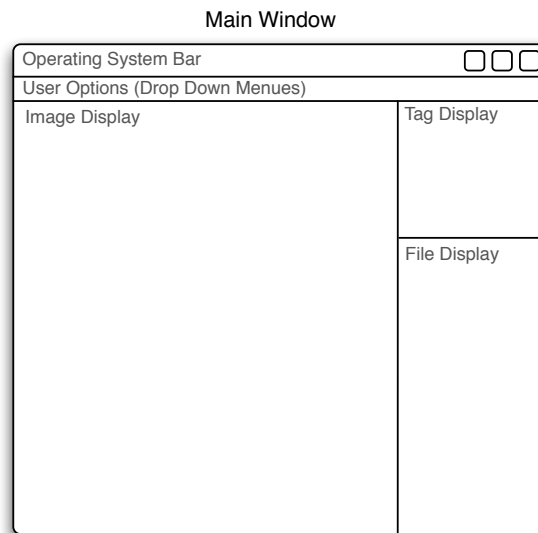


Fig. 3: System UI.