

# Content-Based Image Retrieval System

Submitted to:  
Dr. Raghavan

Submitted by:  
Chao Wang ([cxw3681@cacs.louisiana.edu](mailto:cxw3681@cacs.louisiana.edu))

*URL:* <http://cypress.cacs.louisiana.edu:8080/cxw3681/jsp/IRMain.jsp>

# 1. Introduction

This project is the continued work of Biren N. Shah's master thesis project "Generic and Efficient Online Content-based Image Retrieval". The primary purpose of the new work is to add a web-based interface to the existing CBIR engine.

# 2. Method

1) **Tools:** Jakarta Tomcat Version 3.2.1 (Web Server)

- **Configurations of Tomcat:**

Download tomcat web server from <http://jakarta.apache.org/tomcat/> and install and configure the web server using instructions from [http://jakarta.apache.org/tomcat/tomcat-3.2-doc/uguide/tomcat\\_ug.html](http://jakarta.apache.org/tomcat/tomcat-3.2-doc/uguide/tomcat_ug.html)

Important changes to the configuration file “*server.xml*”:

1. Change port number when necessary: change to 8080 or any other port that you want to use.

```
<!-- ===== Connectors ===== -->

<!-- Normal HTTP -->
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
<Parameter name="handler"
value="org.apache.tomcat.service.http.HttpConnectionHandler"/>
<Parameter name="port" value="8080"/>
</Connector>
```

2. Add a new context: (After adding a new context, you would have a relative independent directory to work with: \$TOMCAT\_HOME/webapps/cxw3681

```
<Context path="/cxw3681"
docBase="webapps/cxw3681"
crossContext="false"
debug="0"
reloadable="true" >
</Context>
```

- **Notes of using Tomcat:**

1. All servlet and other class files should reside in:  
\$TOMCAT\_HOME/webapps/cxw3681/WEB-INF/classes/
2. All jsp files should reside in: \$TOMCAT\_HOME/webapps/cxw3681/jsp/
3. All java bean files should belong to a specific package, and a directory by the same name as the package has to be created in  
\$TOMCAT\_HOME/webapps/cxw3681/WEB-INF/classes/, and the class files of beans should reside in that newly created directory.
4. All the images used in the jsp files should use relative path instead of absolute path, whose location should be stated relative to  
\$TOMCAT\_HOME/webapps/cxw3681/jsp/
5. All files using by servlet (input or output) should use absolute path instead of relative path, otherwise it's hard to find the files.

2) **Languages:** JSP (Java Server Page), Java Bean, Java Servlet, Java

### **Advantages of Servlets Over "Traditional" CGI**

- **Efficient.** With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are  $N$  simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory  $N$  times. With servlets, however, there are  $N$  threads but only a single copy of the servlet class. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.
- **Convenient.** Hey, you already know Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.
- **Powerful.** Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.
- **Portable.** Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.
- **Inexpensive.** There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

### **Advantages of JSP**

- **vs. Active Server Pages (ASP).** ASP is a similar technology from Microsoft. The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS-specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

- **vs. Pure Servlets.** JSP doesn't give you anything that you couldn't in principle do with a servlet. But it is more convenient to write (and to modify!) regular HTML than to have a zillion `println` statements that generate the HTML. Plus, by separating the look from the content you can put different people on different tasks: your Web page design experts can build the HTML, leaving places for your servlet programmers to insert the dynamic content.
- **vs. Server-Side Includes (SSI).** SSI is a widely-supported technology for including externally-defined pieces into a static Web page. JSP is better because it lets you use servlets instead of a separate program to generate that dynamic part. Besides, SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- **vs. JavaScript.** JavaScript can generate HTML dynamically on the client. This is a useful capability, but only handles situations where the dynamic information is based on the client's environment. With the exception of cookies, HTTP and form submission data is not available to JavaScript. And, since it runs on the client, JavaScript can't access server-side resources like databases, catalogs, pricing information, and the like.
- **vs. Static HTML.** Regular HTML, of course, cannot contain dynamic information. JSP is so easy and convenient that it is quite feasible to augment HTML pages that only benefit marginally by the insertion of small amounts of dynamic data. Previously, the cost of using dynamic data would preclude its use in all but the most valuable instances.

### 3) **Interface Architecture:** see Fig.1

#### **Jsp files:**

There are 5 jsp files. `IRMain.jsp` is the main page. `IRMethod.jsp` is the method selection which is common to all three options. `IRImageTable.jsp`, `IRPalette.jsp`, and `IRChoose.jsp` are files specific to each option respectively. When the jsp file is first loaded, it will create beans, and then get information from beans, so it could display page dynamically, which is also true when servlets update beans and redirect back to the jsp files when they could get the updated information. In jsp files, it creates forms, which are the same as in the traditional html pages. So the jsp files pass all the responsibilities of handling user's requests to servlets.

#### **Beans :**

There are two beans used in this program: *RandomPicsBean*, *ColorBinBean*.

*RandomPicsBean* is used to randomly select pictures to display when is used in the first option, which gives the user the choice to randomly display a set of images in the database. When user selects an image as an query image in the first option or gives a query request in the second or third option, this bean is also used to store query results from servlets handling the request.

*ColorBinBean* is used to find out and display all the color bins used in the CBIR engine in the second option, and also used to store user defined color amounts.

Both beans also have variables related to validating user's inputs, such as uploading the right image type, entering a valid number, etc. The request handling servlet will check the user's input and set the beans' variables if anything turns wrong. Then the jsp file will display appropriate warnings accordingly.

### **Servlets:**

All the user's searching requests are handling by servlets. (First option by IRChoose, second option by IRPalette, and third option by IRUpload) They parse the user's requests, get the needed information from beans and interact with CBIR engine by calling the right functions to find the results, then store the results in beans, set variables when there is something wrong with the requests and redirect back to the jsp files where the user is able to see the results or proper warnings.

### **4) Code Flow Chart:** see Fig. 2

All the files and their functions residing in solid rectangular are newly created in this project; those residing in dashed rectangular are files and functions already existing in the previous project.

### **5) Additional notes on implementing the project:**

#### **Database population:**

First method: Only populate initial set, other images could be added online.

Main and Similarity methods: Should repeat the population phase for the initial set and each clustering, but now I only implement the initial set. That is all images in the database are treated as initial set images.

Input (add file) image file should be of ppm format (extension is ppm), pjpeg utility is used to convert jpeg images to ppm format. Quantized version of the file is also of ppm format; Histograms created are stored in <filename>.ppm.hst type in the hstfiles directory.

Images are not stored into the database directly. The high level feature vector of each image is found during the database population phase and is stored in the database. There is mapping from each vector to the corresponding image. The vectors are retrieved from the database during retrieval. Once the vectors are retrieved, you could use the mapping to display the image which is stored somewhere in the directory.

When doing database population, perform each step and then check whether the modifications desired after each step are actually taking place. For instance, after performing step 1, make sure that histograms data is actually created, .ppm.hst is not empty.

When populating database using a large number of images, when going through "ppmquantall 64 + all file names", it will give the error: "pnmcat: Too many open files, ppmquant: EOF / read error reading magic number", which will lead to not executing this

command at all, so the intermediate result will be too large to keep. To deal with that, type “limit descriptors 1024” at the unix prompt.

In “CreateHashTable\_IncludeAllBins.java”, I do the normalization into a scale 0-100,000 when reading in from the .hst files to create .lst files instead of storing the frequency count values as it is. In this way, I could use it for the second option and also make it extensible to any other image size.

There is some Biren’s code, which will create the intermediate files, such as “miscdata.lst”, that is the same for the three methods, it will not work when the three methods have different size of initial set. I change them to include method name in the file name, so it will work in that case.

#### **Online addition:**

Only first method support online addition, main and similarity methods don't support it.

Image retrieval and online addition follow almost the same steps. The only difference is that the data created for an image which is to be online added has to be saved somewhere for future use whereas for query image the same data is not stored. All the retrieval data once the retrieval results are displayed is deleted.

#### **Retrieval:**

First method can retrieve an image not already existing in the database.

Main and Similarity methods only permit query image already in the database, not a new image. If using clustering approach, it first compute distance to the initial set, for the relevant ones, then go further into clusters to compute distances, then rank the whole results. Now it only implements to first level. It doesn’t work from Biren’s retrieval interface. Biren wrote some test programs to perform the experiments. Those test programs are not connected to the actual system... files with names CbirTest1, 2, 3 are the test programs, and the results in them are not direct as far as the main and similarity method retrieval is concerned. The results of those test programs are formatted and listed in a manner to report them in his thesis, and those results can be used to derive the main and similarity method retrieval results. In his “ImageRetrievalProcess.java”, there is some code for main method also along with first method, but it does not work. He tested, but may not have removed that part of the code from the program. For the first level retrieval, I rewrite the code in “ImageRetrievalProcessForMainAndNew.java”.

The real distance is the same for all methods. The estimated distance computation is same for all methods too. Only the high level feature vectors for the three methods are different.

#### **Others :**

To keep files organized, I add “package” statements in Biren’s code.

### **3. Results**

1) **New functionalities:**

- a. Makes the CBIR engine web-enabled, provides an interface that could be accessed on-line.
  - b. Provides retrieval function for main and similarity-preserving embedding methods (populate all images in the database as initial set)
- 2) **Testing:** compares the image retrieval results with the test results included in Biren's thesis (reside in his CD-ROM /results/experimentsondwr75/20/e12.txt)
- 3) **Future work:** populate the database using clustering approach.

#### **4. References**

"Generic and Efficient Online Content-based Image Retrieval", Biren N. Shah, Master Thesis, Spring 2001

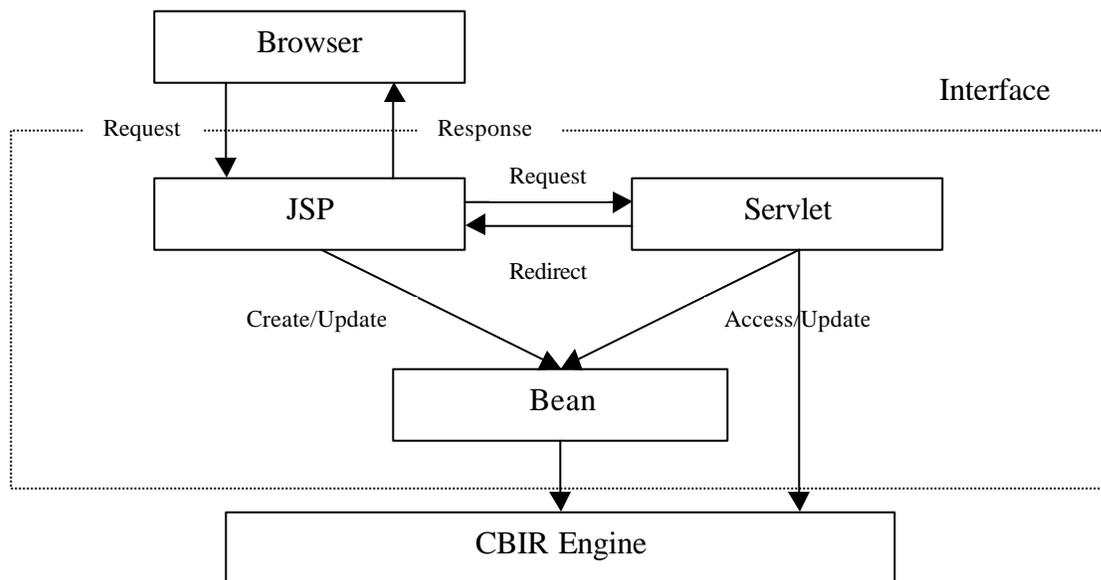


Figure 1: Interface Architecture

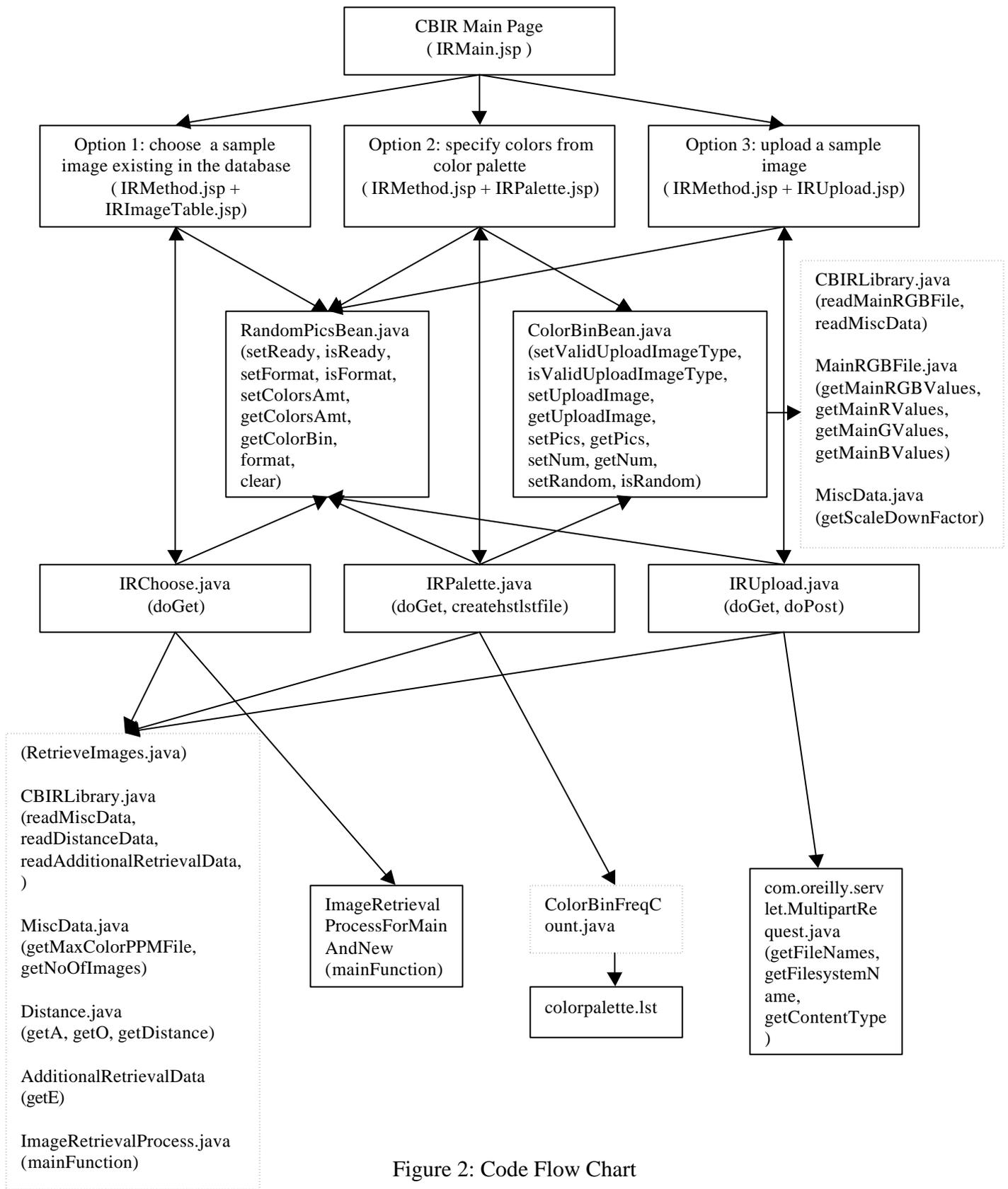


Figure 2: Code Flow Chart