2003

Littlebee: User's online interactive design of baby's wear

Yan Hai

LITTLEBEE: USER'S ONLINE INTERACTIVE DESIGN OF BABY'S WEAR

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Yan Hai

June 2003

LITTLEBEE: USER'S ONLINE INTERACTIVE DESIGN OF BABY'S WEAR

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Yan Hai

June 2003

Approved by:

[redacted]

Ernesto Gomez, Chair, Computer Science 5/15/03 Date

[redacted]

George M. Georgiou

[redacted]

Kerstin Voigt

ABSTRACT

"Littlebee" is a server-side web application in order to perform interactive baby's wear design on the World Wide Web, based on the styles and appliqués pre-stored in the database. The clients can change the colors or appliqué positions of the clothes, using any standard graphic enabled web browser. This system is implemented in the Java programming language, which makes it portable to various computer platforms.

ACKNOWLEDGMENTS

Thanks to my advisor, Dr. Gomez, for his strong coaching and encouragement to enable me to complete this project.

Thanks to Dr. Georgiou and Dr. Voigt for their kind assistance to me throughout this project.

Thanks to all faculty and staff of Computer Science Department at California State University, San Bernardino, for the support.

Thanks to the Associated Students, Inc. for their financial support.

Yan Hai

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

SOFTWARE REQUIREMENTS SPECIFICATION

1.1 Introduction

"Littlebee" is a server-side web application by which clients can perform interactive baby's wear design on the World Wide Web, based on the styles and appliqués already stored in the database. Clients can change the colors or appliqué positions using any standard graphic enabled web browser. Chapter one presents an overview of the project. The context of the problem is discussed followed by the purpose and assumptions. Then the limitations of the project are reviewed. Finally, the definitions of the terms used in this paper are presented.

1.2 Context of the Project

There are many online baby's wear sellers, but most of them only show their catalogues, product pictures and prices on their websites. Few of them, if any, have any interactive interfaces with their customers. If the function of an online design of the clothes can be added to the websites, buyers may be much more interested in buying since they can design the clothes by themselves and then buy it. "Littlebee" is intended to implement this function.

1.3 Purpose of the Project

The purpose of the project is to develop a web-based application that allows customers to do online design of baby's wear. The online design includes combining appliqués with clothes styles and changing colors, using images pre-stored in the database. Figure 1 "Use Case Diagram" illustrates the functions of "Littlebee" project.

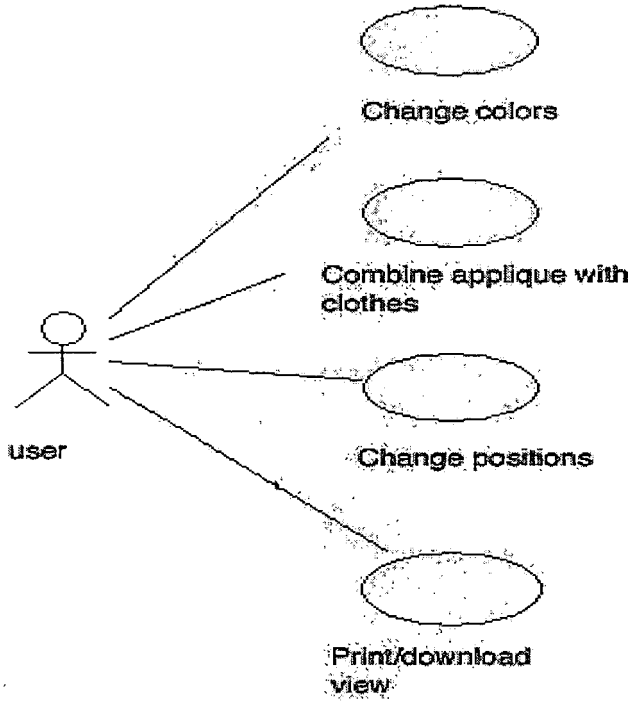


Figure 1. Use Case Diagram

1.4 Assumptions

The project is based on the following assumptions:

1. Client side has a minimum of 32 Megabytes of RAM

to support image processing functions.

2. Client side browsers are capable of instantiating Java Virtual Machine that supports the Java version1.3.1.
3. All images are in GIF format and with white (255,255,255) background.

1.5 Limitations

During the development of the project, a number of limitations were noted. These limitations are presented here.

1. User can choose no more than one appliqué to add to different clothes styles.
2. When changing the colors of appliqué or clothes, if two colors are changed into one same color, they will always be the same color after that, till user chooses images again from the multi-choice forms.
3. Only image names and images are stored in database, there are no prices, stock information and delivery information in database.
4. The application focuses on online design only, including no online order-taking functions.

1.6 Definition of Terms

The following terms are defined as they apply to the project.

Littlebee: The name of the project discussed in this thesis.

JDK: Java Development Kit, standard edition

JSP: Java Server Pages, which is a set of java instructions that will generate servlet programs.

Java Servlet: Java API which generates dynamic web pages.

JVM: Java Virtual Machine, a computing machine that has java byte-codes as input. It makes java a platform independent language.

JDBC: Java Database Connectivity, a standard interface provided by Sun Microsystems to connect different databases.

Oracle8: Powerful database software developed by Oracle, Inc.

Tomcat: A free, open-source implementation of java servlets and java server pages technologies developed under Jakarta project at the Apache software foundation.

Appliqué: Embroidery patterns on clothes.

1.7 Organization of the Thesis

The thesis portion of the Littlebee Project is divided into six chapters. Chapter One provides the software requirements specifications, an introduction to the context of the problem, purpose of the project, the limitations, and the definitions of terms. Chapter Two describes the design of the software. Chapter Three documents the steps used in testing the project. Chapter Four presents the maintenance required for the project. Chapter Five presents the users manual of the project. Chapter Six summarizes the conclusions drawn from the development of the project. The appendices containing the figures and project source codes follow Chapter Six. Finally, the references for the project are listed at the end of the thesis.

CHAPTER TWO

SOFTWARE DESIGN

2.1 Introduction

Chapter Two discusses the software design, more specifically, the software environment, the system design, the image storage design and the user interface design.

2.2 Software Environment

The software environment used for Littlebee Project is as following:

Operating System: Windows 2000 on disk D.

Tools: J2SDK 1.3; Apache 1.3.24 as the web server; Tomcat 3.3.1 as the adds-on servlet container; Oracle 8.1.5 as the database server. Acme package from ACME laboratories as the GIF encoder.

Windows 2000 and Oracle database are used for their popularity in the industry and their availability on my machine. All other tools are freely available on the Internet and also extensively used in the industry. Please note the software can be changed into software with similar functions. The application is portable since it is written in Java.

The configurations of above environment are done as following:

J2SDK 1.3: Set path=%path%; D:\jdk1.3\bin

Apache: Add following block at the end of httpd.conf:

include %TOMCAT_HOME%\conf\auto\mod_jk.conf

Then, download mod_jk.dll in binary form from apache website and put it in %APACHE_HOME%\modules\

Tomcat:

1. Set JAVA_HOME=D:\jdk1.3 before starting Tomcat server.

   Create a directory for the application.

   Create web.xml and put it in the WEB-INF folder.

   Create apps.xml and put it in the conf folder.

   Unzip %ORACLE_HOME%\jdbc\lib\classes111.zip (JDBC driver) into WEB-INF\classes\.

2. Start Tomcat with bin\startup jkconf option to generate mod_jk.conf file. After that, Tomcat stops automatically.

3. In mod_jk.conf file, modify the following block

   JkMount / ajp13

JkMount /* ajp13 as

JkMount /servlet/* ajp13

JkMount /*.jsp ajp13 to let Apache handle static pages and Tomcat deal with servlets and JSPs.

4. Always start Tomcat before Apache.

During deployment, one should pay careful attention on where to put the files.

1. All servlets and java classes must reside in %TOMCAT_HOME%\webapps\myapp\WEB-INF\classes
2. All JSP files must reside in %TOMCAT_HOME\webapps\myapp\jsp\
3. All static HTML files must reside in %APACHE_HOME\htdocs\littlbee\

2.3 System Design

In the project, the graphical user interface is presented by html files and java server pages, which includes no business logic, while the java servlets and java classes provide all business logic to facilitate the system to finish its task.

The whole system is a multi-tiered architecture.

1. The client tier: the client tier interacts with the

user using html files and displays requested information from the system to clients.

2. The web tier: the web tier accepts the user's requests and generates presentation logic. This tier is implemented by JSP and java servlet. In order to make the system easily maintained and extended, the requests and responses are handled in two subsystems: a controller servlet handles all the incoming requests, while the JSP files deal with all the presentation issues. This design promotes the encapsulation of the object-oriented programming, making it easy to add, remove and modify functionalities without affecting any existing clients.
3. The business logic: this tier deals with the business logic of the application using servlets and java classes.

See Appendix A for a general view of the system architecture.

2.4 Image Storage Design

YHAI database is used to store business information of a company, in this case, the style and appliqué images. It

is a simple database right now with two tables storing style images and appliqué images respectively.

All images used in this project are in GIF (Graphics Interchange Format) type. In Oracle database system, images can be stored as LONG RAW type or LOB type. A LONG RAW type record cannot be retrieved if its value is longer than 32K, which is too small for images used. So this type is discarded. For Large Object types, there are also two choices: internal large objects as BLOB and external large objects as BFile.

Table 1 contrasts the features of the two image types.

BFile is used in Littlebee Project, because as a web application, the clients can read but not write the images for security reason, and as fashion trends always change, the image data need to be updated easily at server side. BFile can achieve these purposes.

To set up tables with Bfile fields, a DIRECTORY object has to be created first. The DIRECTORY specifies a logical alias name for a physical directory on the server's file system under which the file to be accessed is located.

After the directory and the tables are created, the Bfiles are initialized using BfileName(), which associates

the existing operating system files with the relevant database records of a particular table. For SQL statements of creating directory and table, inserting image data,

Table 1. Features of BFile and Binary Large Object (BLOB)

Features	BFile	BLOB
Location	External-outside the database tablespaces	Internal-inside the database tablespaces
Participation of transactions	No	Yes
Maximum size	Operating system dependent, but no more than 4G	4G
Readability and writability	Read only	Both read and writable
Storage style	LOB locator is stored in table column. Data is stored as external file.	LOB locator is stored in table column. Data is stored in separate table space.
Advantages	Easy update	Efficient access

see attached CreateTables.sql, LoadAppliques.sql and LoadPatterns.sql.

2.5 Interface Design

All servlets and java classes are in littleBee_proj package. Besides that, there are seven JSP files.

JSP Files

DisplayPattern.jsp and DisplayApplique.jsp display the clothes style and appliqué images retrieved from the database; they will also allow users to change the color of the images by using the Red, Green, or Blue values.

ChangePatternColor.jsp, ChangeAppliqueColor.jsp will display the images after a color is changed. Colors can be changed repeatly.

ChangePosition.jsp will display user-chosen clothes style and the matching appliqué (the color of the appliqué may have been changed) and allow the user to click on the style at any position they want the appliqué to be.

SendComposition.jsp shows the clothes with the appliqué placed at clicked position.

Download.jsp displays the user-designed clothes to be stored onto a local disk or to be printed out.

Littlebee_proj Package

ControllerServlet is the entry point of all requests. It dispatches different requests with parameters to corresponding JSP files.

ConnectionServlet sets up the database connection and retrieves image data from the Oracle8 database. Although the OCI8 driver has better performance, it requires the

presence of the OCI libraries, Net8, and CORE libraries to be installed on the database client side besides the driver itself. Oracle JDBC thin driver is used for portability, the driver is all needed at client side. This servlet also cooperates with Oracle8 to handle Oracle BFile data and convert it to java data type.

Change_PCServlet allows the user to change colors on clothes or appliqués. Images are retrieved from the user session and colors get filtered by user-defined red, green, and blue values.

ChooseImageServlet chooses the appropriate image to display, either the original style/appliqué from the database, or the one modified by the customer.

CompositeServlet displays the clothes with the appliqué on it. The appliqué position is set by the user's mouse click. Java 2D BufferedImage is used to integrate the clothes and appliqué images.

WhiteBackgroundFilter class changes the image background from white (255,255,255) to transparency, so that when the user changes the image color, there won't be any effect on the image background.

ColorFilter class changes the image color from old red, green, blue values to new values.

ServletUtil class is a utility class used by all servlets to do miscellaneous works.

See Figure 2 for class diagram in littleBee_proj package.

Acme Package

To write out java images as Gif files, Littlebee project uses a freely available Gif encoder from ACME Laboratories. For details of Gif encoder and ACME, please see www.acme.com.

2.6 Summary

The software design of the project was presented in Chapter Two. The system environment and overview present a whole picture of the Littlebee project. The image storage design and the interface design give the detailed idea of the application.

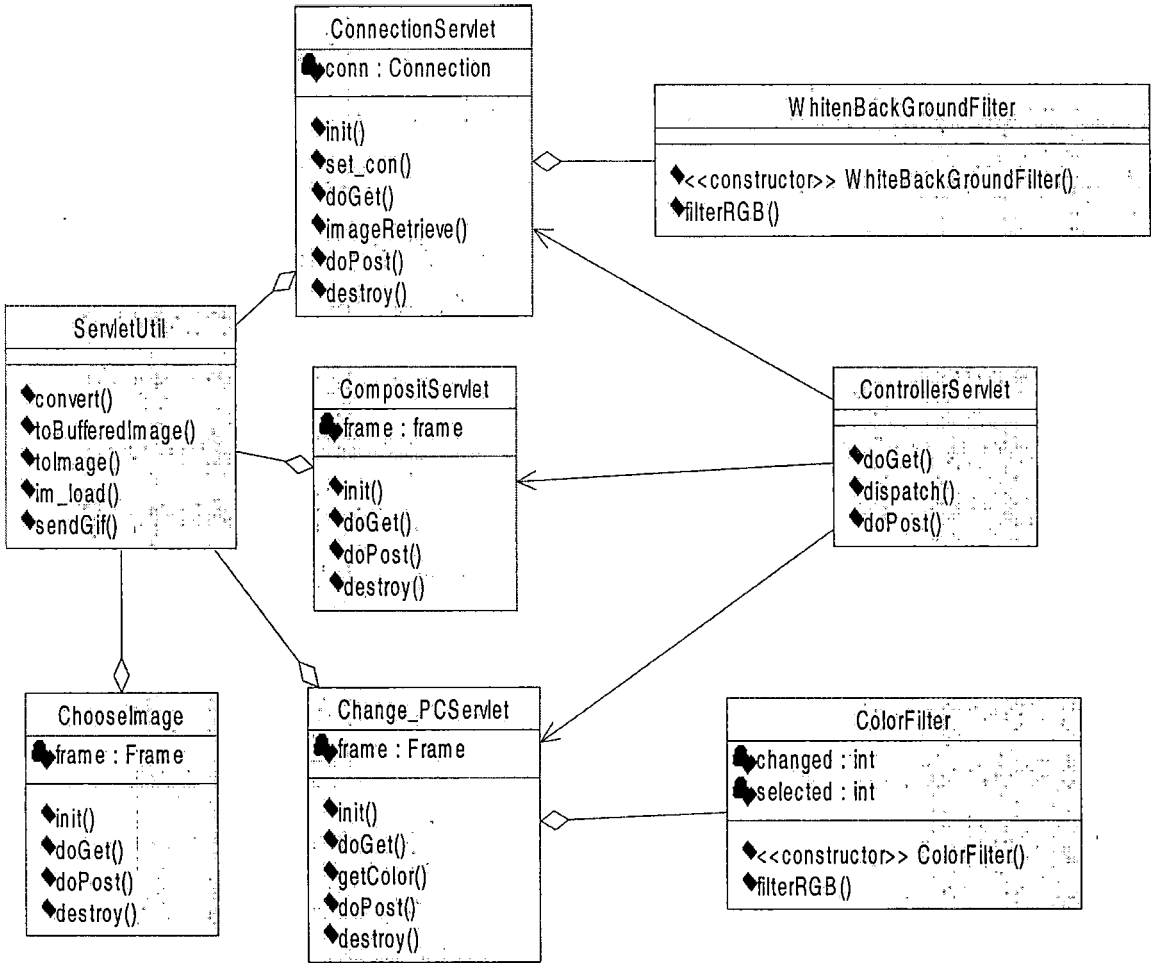


Figure 2. Class Diagram for littleBee_proj Package

CHAPTER THREE

SOFTWARE QUALITY ASSURANCE

3.1 Introduction

Chapter Three documents the software quality assurance. Specifically, the main ideas in unit test, integration test and system test are presented with some test cases.

3.2 Unit Test and Integration Test Plan

The unit test is to test particular functions or code modules of the application, while the integration test is to test the combined parts of an application to determine if they work together correctly. The following test cases were designed to achieve these purposes.

Database setup and configuration test: three sql scripts were loaded and run on the database server. One of them creates two tables including BFILE variables, which would contain appliqué and clothes images; the other two scripts loaded the tables with images. After the tables were created and the data were loaded, both true and false sql queries were performed to validate the information stored in the database. The tests were successful except the images couldn't be displayed in sql plus server. Please see Appendix C for the sql scripts.

Core java module — data retrieval test: this test was further divided into two steps. Firstly, the module tried to retrieve non-image information from the database, such as appliqué name and clothes pattern number, etc. Secondly, the Acme Gif encoder was added to retrieve and display images from the database. In this process, the java module that retrieved data from the database, the integration of java module with database and the integration of java module with Gif encoder were tested. During the development of this project, several problems were found and fixed, more exceptions were handled, tests and fixing were done until this subsystem worked well.

Core java module — image processing test: Firstly, different images from outside of the database were processed to test the codes. Image colors were filtered or changed. Two different images were combined and their positions changed. White background and colored background images were loaded to test programs constraints. Then, the byte codes retrieved from database were processed. It was found at this stage that once two colors were changed into one same color, they would be always the same during this session. This problem cannot be fixed because of the design of the project.

JSP file test: All JSP files were extensively tested independently with data within range and out of range. Then they were connected with the core java modules to test the integration. Some image display problems were found and fixed.

3.3 System Test

Based on the requirements of Littlebee Project, its system testing was performed with the black box testing method. All actual outputs were compared with the specified outputs. Many simple cases were tested to prove the basic correctness of the application. Both legal input values and illegal input values were used for the process to produce either usable values or meaningless outputs.

3.4 Other Tests

Littlebee Project is basically an internet system, so other tests related to website technologies were also performed during the test process. For example, link testing was performed to verify that all links lead to the right pages. Performance testing was considered, but because the internet communication latency, and the client side hardware and software are hard to specify, it is very difficult to test the performance of the system. The

primary factors in performance under control are the web server software and hardware, the database, and the web server connection capacity. The performance of the application has been proved to be fast when it was run on the Intranet.

CHAPTER FOUR

MAINTENANCE

4.1 Configuration

When the application is deployed, a particular class path and environment variables have to be set, and particular files have to be placed in the correct path. How these will be done depends on the type of server software and its version. If the software or its version is different from that stated in Chapter Two "Software Environment", the documentation of the software needs to be examined.

Two issues need particular attention during the configuration to improve the performance of the application: Firstly, in order to let the apache web server handle static pages and Tomcat servlets and Jsps, a pipe module must be installed between the web server and Tomcat to filter out the static pages. Secondly, Tomcat must always start before Apache in case of crash.

4.2 New Requirements

Another important maintenance job is adding more web pages and functions to the Littlebee Project. The architecture design of the application makes this work

easier. Each of the three components of the project, the Database, the business logic, and the user interface, is responsible for a separate function. If more business information is needed in the database, scripts in the database layer can add it. More java modules can be added in the business logic to process that new information or to add new functions to the existing information. Finally, the web pages or jsp can be placed in the user interface component.

CHAPTER FIVE

USERS MANUAL

Users of Littlebee application need to simply follow the links or instructions on web pages to finish their interactive design work. Here are the steps to do the job.

1. Figure 3 shows the page where the user selects a clothes style and presses the "Render" button.

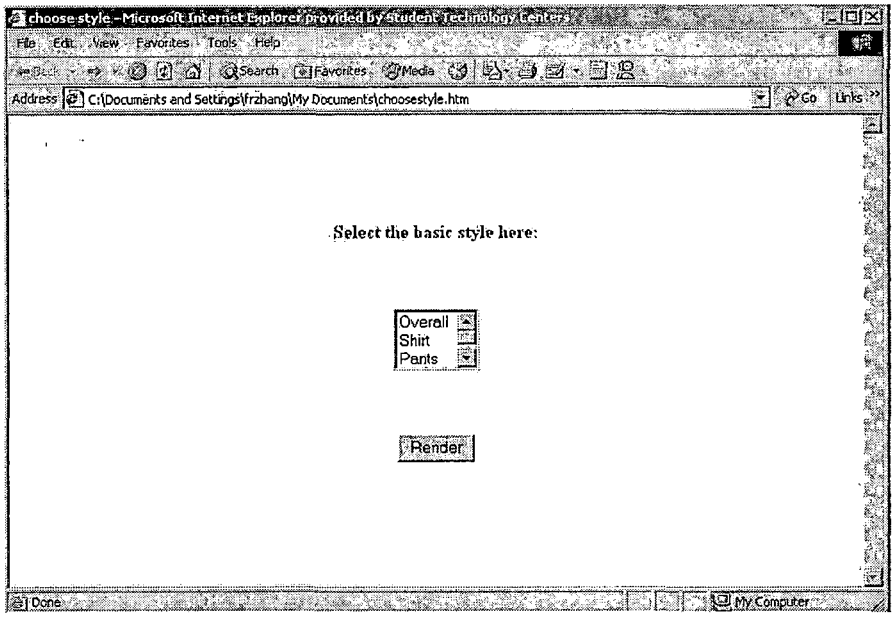


Figure 3. Screen Shot 1

2. In Figure 4, the clothes selected shows up, the user can further assign the red, green, or blue value to change the color at the place the user has clicked. If the user doesn't want to change colors, he/she can go directly

to choose an appliqué, which will be matched with the clothes already selected.

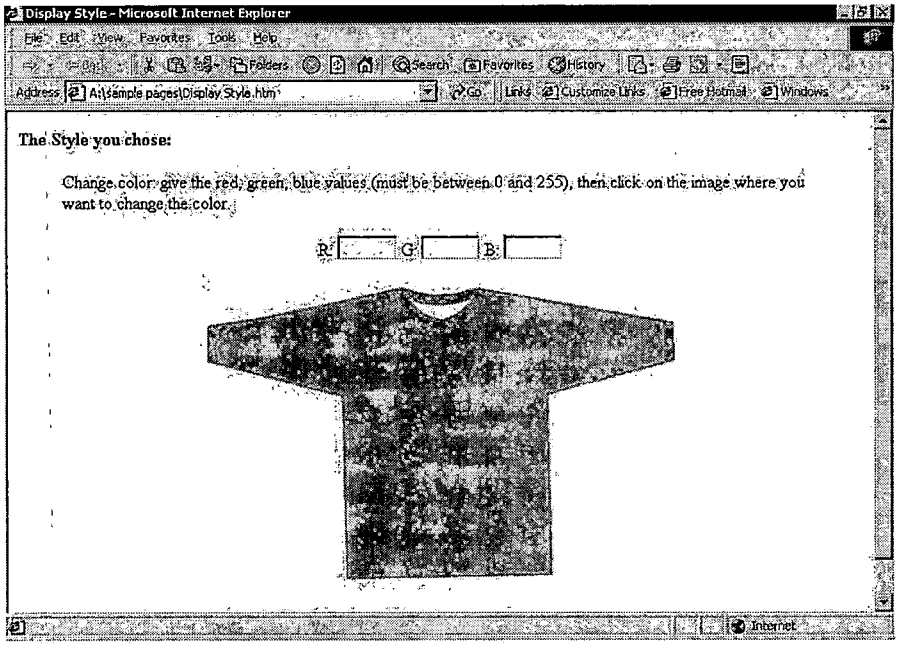


Figure 4. Screen Shot 2

3. In Figure 5, the image with the new color will be sent back to the user after the user has input red, green, blue values to change the color of the clothes. The user can change more colors, go to choose appliqués to match the clothes, or go to choose another clothes style.

4. In Figure 6, user selects an appliqué to match the clothes.

5. Similar to what the user can do with the color of the clothes, he/she can also change the colors on the

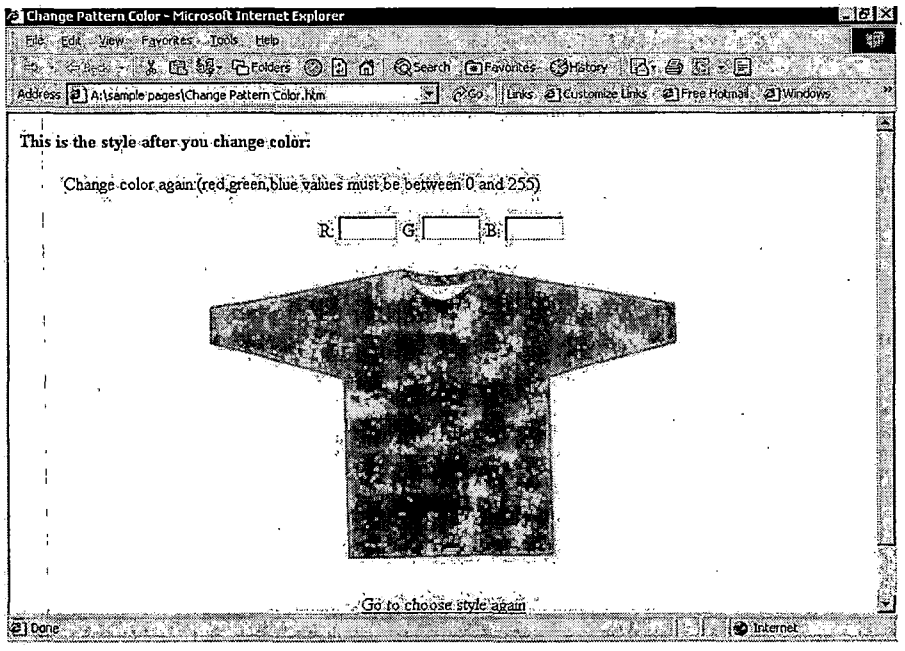


Figure 5. Screen Shot 3

appliqué, or go directly to place this appliqué onto the clothes in Figure 7.

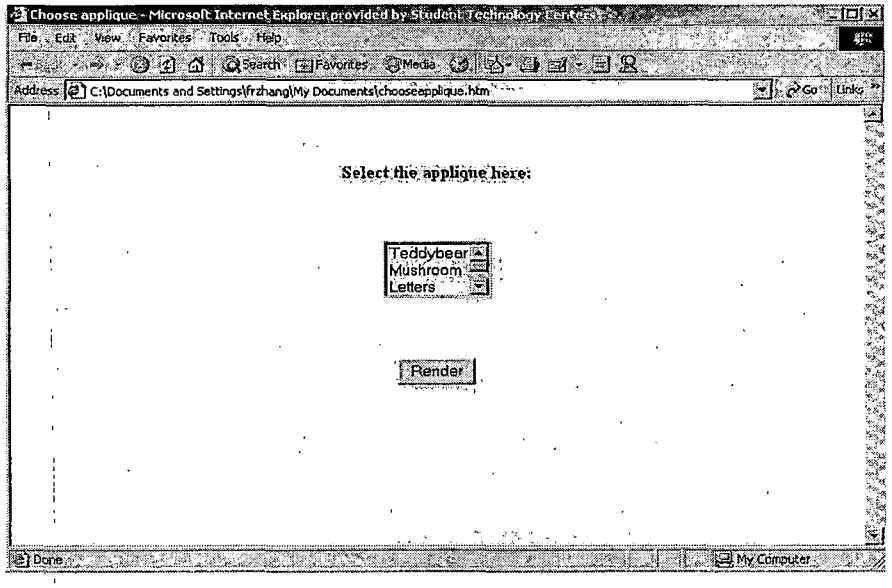


Figure 6. Screen Shot 4

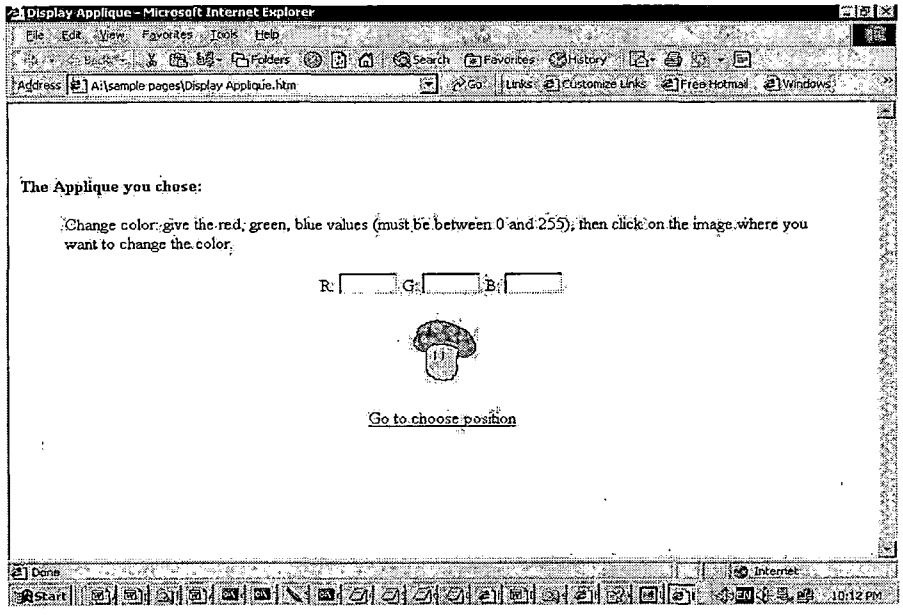


Figure 7. Screen Shot 5

6. After having changed the appliqué color, the user can change more colors, proceed to position the appliqué on the clothes, or choose another appliqué in Figure 8.

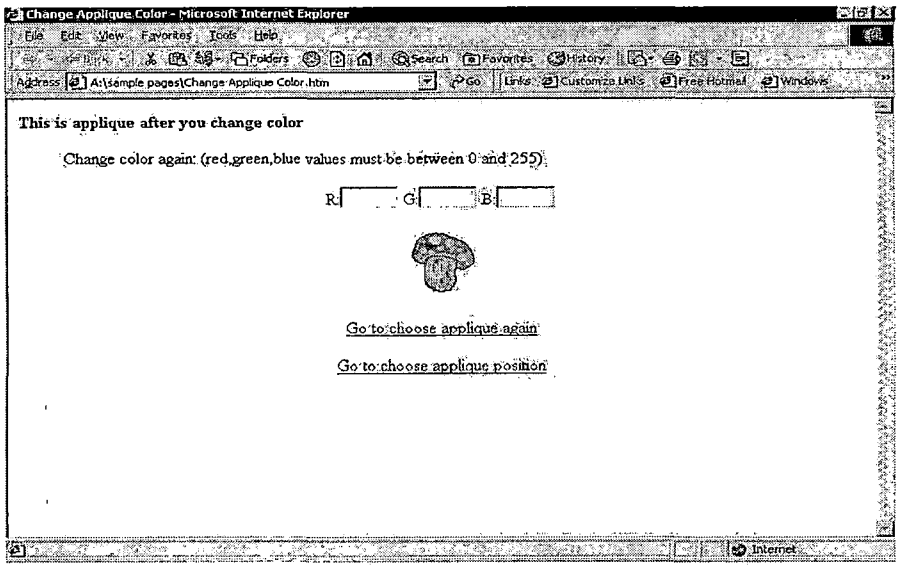


Figure 8. Screen Shot 6

7. In Figure 9, user simply clicks on the clothes where he/she wants to put the appliqué.

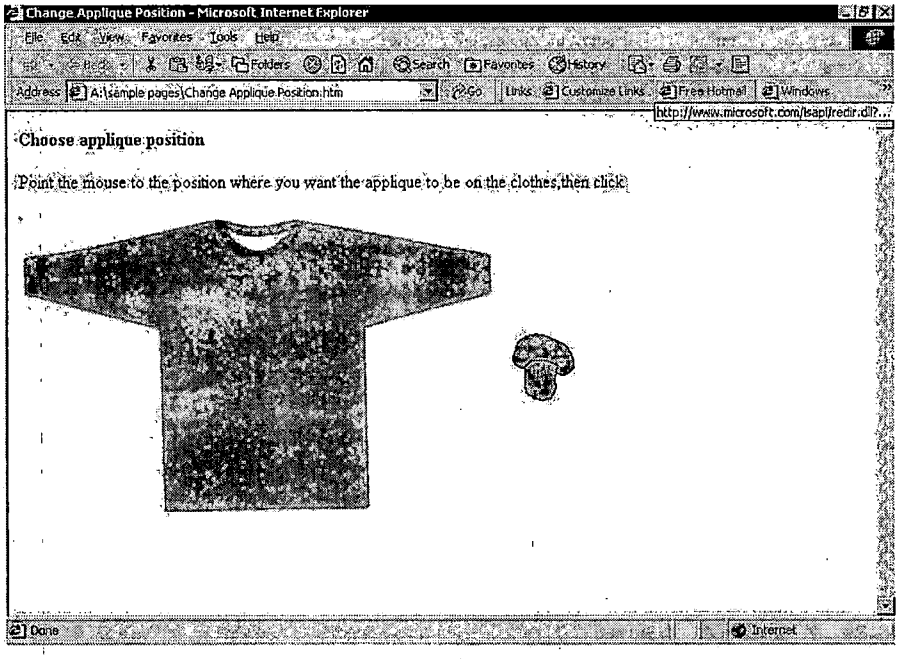


Figure 9. Screen Shot 7

8. In Figure 10, the appliqué will appear at the position where the user has clicked. Then the user may either proceed to adjust the position, or to try another clothes or appliqué, or to download and print the picture.

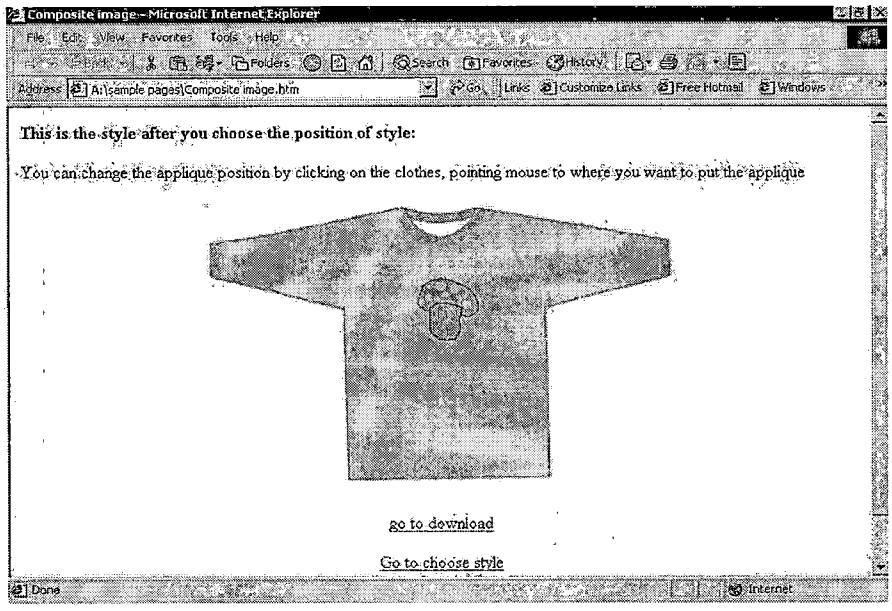


Figure 10. Screen Shot 8

9. Figure 11 shows the result after the user has once again changed the appliqué's position.

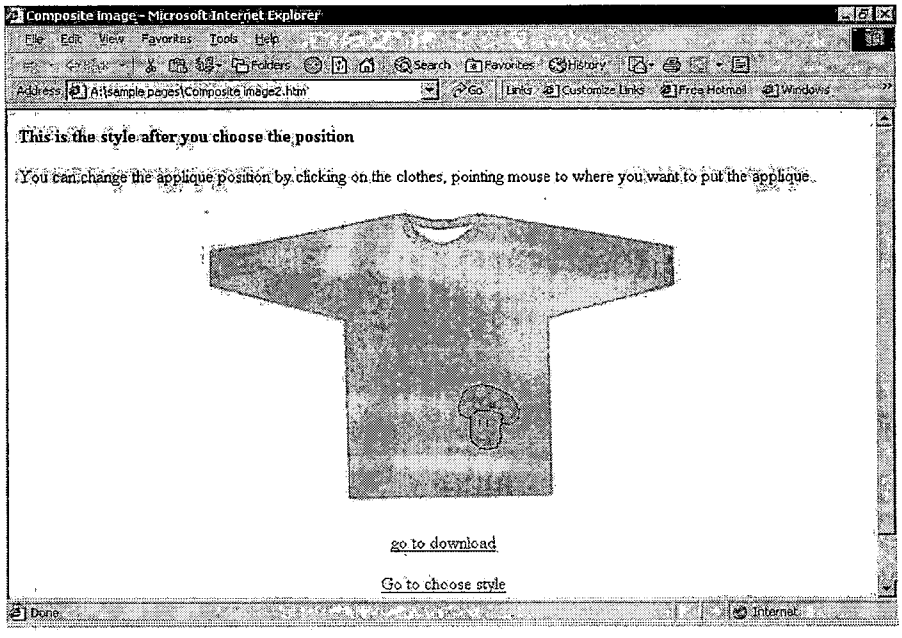


Figure 11. Screen Shot 9

10. At either Step 8 or Step 9, user can choose to preview the downloadable and printable picture.

CHAPTER SIX

CONCLUSION

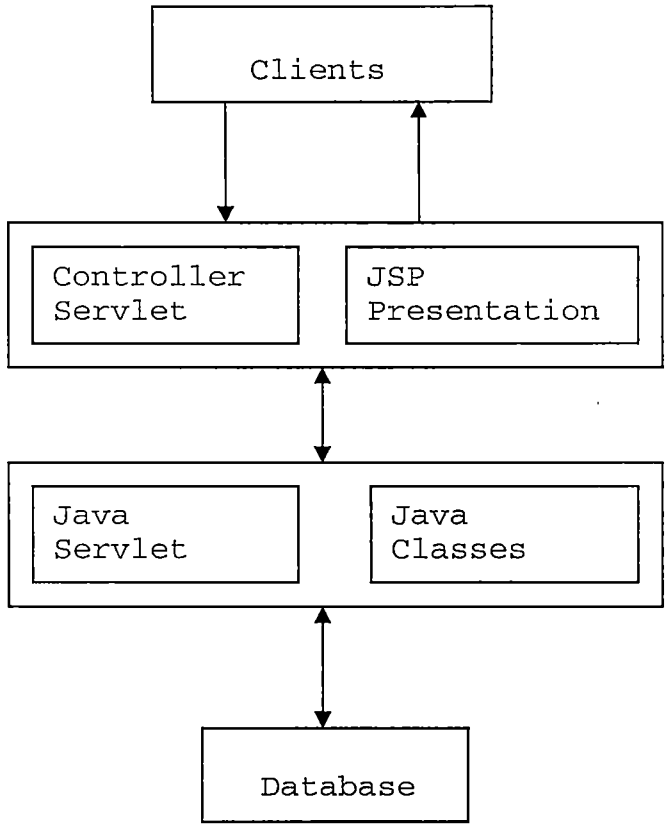
Littlebee Project provides an efficient way for users to interact with web servers. Once the servlets are loaded, a user can get quick response from the server. The server-side architecture also protects all the data, sources, class files, offering a better security.

The tradeoff of the above advantages is the heavy load at the server side. Future improvements could be made in this regard.

Other changes should be also considered if "Littlebee" is to be commercialized. More design functions should be added, such as more appliqués to be placed on the clothes. The backend database also needs to be enhanced to hold more information such as prices, delivery time and stock information, so that users can have the complete information actually place an order through the website.

APPENDIX A:

MULTI-TIERED SYSTEM ARCHITECTURE


Clients
Controller Servlet
JSP Presentation
Java Servlet
Java Classes
Database

APPENDIX B:

SOURCE CODE

ControllerServlet.java

```
package littleBee_proj;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;


/**************************************************************
************
 *
 * Yan Hai
 * 03/25/02
 *
 * This program is the entry point for all html forms
 * It dispatches control to Jsps after it gets the request
 * information
 *

**************************************************************
**********/


public class ControllerServlet extends HttpServlet {

public void doGet (HttpServletRequest req,
                   HttpServletResponse res)
            throws ServletException,IOException {

      //create a new user session
   HttpSession ses=req.getSession(true);

   try {
     String task=req.getParameter("task");

     if (task.equalsIgnoreCase("retrievePattern"))
     dispatch("/littlebee/jsp/DisplayPattern.jsp",req,res);

     else if (task.equalsIgnoreCase("retrieveApplique"))
     dispatch("/littlebee/jsp/DisplayApplique.jsp",req,res);

     else if(task.equalsIgnoreCase("changePatternColor"))
     dispatch("/littlebee/jsp/ChangePatternColor.jsp",
```

```
      req,res);

      else if(task.equalsIgnoreCase("changeAppliqueColor"))
      dispatch("/littlebee/jsp/ChangeAppliqueColor.jsp",
                      req,res);

      else
      dispatch("/littlebee/jsp/SendComposition.jsp",req,res);
          }

      catch (Exception e)
         {res.sendError(res.SC_INTERNAL_SERVER_ERROR,
                          "Server Error.");
          e.printStackTrace();
          return;
        }
   }//end of doGet()

   private void dispatch(String add,HttpServletRequest
           request,HttpServletResponse response)
           throws ServletException, IOException {

      RequestDispatcher dispatcher=
          getServletContext().getRequestDispatcher(add);
          dispatcher.forward(request,response);
    }//end of dispatch()

   public void doPost(HttpServletRequest req,
                      HttpServletResponse res)
          throws IOException, ServletException{
      doGet(req,res);
   }

} //end of ControllerServlet class
```

ConnectionServlet.java

```
package littleBee_proj;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.awt.*;
import java.awt.image.*;
```

```
import java.sql.*;

import oracle.jdbc.driver.*;
import oracle.sql.*;

/**************************************************************
************
 * Yan Hai
 * 03/25/02
 *
 * This program sets up database connection when it is
 * loaded, then retrieves image from Oracle8.1.5 database
 * and reads the image data
 * into byte stream. Oracle can have 10 processes by
 * default, and 10 Bfiles can be opened at the same time.
 * so no connection pool
 * is necessary for 5 users only.
 * Image white background is changed to transparency when
 * retrieved from database.
 *
 * @see WhiteBackgroundFilter
 *
**************************************************************
**********/



public class ConnectionServlet extends HttpServlet {
   protected Connection conn;

   public void init() {
     set_con();
   }

   private void set_con(){
     try {

      Class.forName("oracle.jdbc.driver.OracleDriver");

      String url=jdbc:oracle:thin:@127.0.0.1:1521:yhai";

      String user="Scott";
      String pw="tiger";
```

```
    conn=DriverManager.getConnection(url,user,pw);
    }
    catch(Exception ex) {
       System.out.println(ex.getMessage()+"\n");
       ex.printStackTrace();
    }

} //end of set_con()

public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
        Throws ServletException,IOException {

    String  table=req.getParameter("task");
    String  pattern_name=req.getParameter("selectpattern");
    String  aplique_name=
                  req.getParameter("selectapplique");


    HttpSession session=req.getSession();


  if(conn==null) {
     res.sendError(res.SC_INTERNAL_SERVER_ERROR,
           "Cannot set up database connection");

     return;
  }

  if(pattern_name!=null) {

    Image im= Toolkit.getDefaultToolkit().createImage
       (imageRetrieve(conn,pattern_name,table));

    Image new_im=Toolkit.getDefaultToolkit().createImage
      (new FilteredImageSource im.getSource(),new
       WhiteBackgroundFilter()));

    session.putValue("patternImage",new_im);

    ServletUtil.sendGif(new_im,res);

  }
 else  {
     Image im=Toolkit.getDefaultToolkit().createImage
```

```
            (imageRetrieve(conn,applique_name,table));
      Image new_im=
             Toolkit.getDefaultToolkit().createImage
             (new FilteredImageSource (im.getSource(), new
                    WhiteBackgroundFilter()));

      session.putValue("appliqueImage",new_im);

      ServletUtil.sendGif(new_im,res);
   }


}//end of doGet()

private byte[] imageRetrieve(Connection conn, String s,
                             String t) {

    BFILE bf_loc = null;//initialize BFILE locator
    InputStream in=null;
    PreparedStatement pstat=null;
    ResultSet rset = null;

      try {
         conn.setAutoCommit (false); //for efficiency

         byte[] imageData;
         int length=0;

        if(t.indexOf("Pattern")!=-1){
           pstat=
            conn.prepareStatement("SELECT Pattern FROM
            Patterns WHERE Pattern_Name like ?");
         pstat.setString(1,s);

         }
         else {
           pstat=
            conn.prepareStatement("SELECT Applique FROM
            Appliques WHERE Applique_Name like ?");
           pstat.setString(1,s);
         }

         rset=pstat.executeQuery();
```

```
    // cast Resultset to OracleResultSet
    if (rset.next()) {

        bf_loc = ((OracleResultSet)rset).getBFILE (1);
      }//get BFILE locator

      // Open the BFILE:
      bf_loc.openFile();
      // Get inputstream from the BFILE:
      in = bf_loc.getBinaryStream();
      length=(int)(bf_loc.length());

      // Get image bytes
      imageData=new byte[length];
      if(in!=null)
        in.read(imageData);

      conn.commit();
      return imageData;

   }
   catch (SQLException e) {
      System.out.println(e.getMessage()+"\n");
      e.printStackTrace();
      return null;
   }
   catch (IOException e){
      System.out.println(e.getMessage()+"\n");
      e.printStackTrace();
      return null;
   }
   finally {
      // Close the stream, BFILE, statement
     try {
      in.close();
      bf_loc.closeFile();
      pstat.close();
      rset.close();
     } catch (Exception e) {
      System.out.println(e.getMessage()+"\n");
      e.printStackTrace();
     }
    }
}//end of method imageRetrieve
```

```
   public void doPost(HttpServletRequest req,
                      HttpServletResponse res) throws
      IOException,ServletException {
         doGet(req,res);
      }

   public void destroy(){

      if(conn!=null){
         try {
           conn.close();
         }
         catch (Exception ex) {
           ex.printStackTrace();
         }
      } //end if
   }

}//end of class ConnectionServlet
```

Change_PCServlet.java

```
package littleBee_proj;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.awt.*;
import java.awt.image.*;
import java.util.*;

/**********************************************************
***********
 * Yan Hai
 * 03/25/02
 * This program accept parameters from clients, filter
 * image colors,output the new image, save new image to
 * user session use string array to process both patter and
 * appliqué images This servlet is for both pattern and
 * appliqué color change.
 *
 * @see ColorFilter
 * @see ServletUtil
```

```
 *
**********************************************************
*********/


public class Change_PCServlet extends HttpServlet {

   Frame frame=null;

   public void init(){
      frame=new Frame();
      frame.addNotify();
   }

   public void doGet(HttpServletRequest req,
HttpServletResponse res)
      throws ServletException, IOException {


      int x,y;
      int red, green, blue;
      String oper="oper";
      String[] sa=new String[4];

      String s="";


      Enumeration paraNames=req.getParameterNames();
      while(paraNames.hasMoreElements())
        s+=(String)paraNames.nextElement();

      if(s.indexOf("pattern")==-1){
           sa[0]="applique.x";
           sa[1]="applique.y";
           sa[2]="appliqueImage";
           sa[3]="updatedAppliqueImage";

      }
      else {
           sa[0]="pattern.x";
           sa[1]="pattern.y";
           sa[2]="patternImage";
           sa[3]="updatedPatternImage";
```

```
        }

        x=ServletUtil.convert(req.getParameter(sa[0]),req);
        y=ServletUtil.convert(req.getParameter(sa[1]),req);

        String r=req.getParameter("red");
        String g=req.getParameter("green");
        String b=req.getParameter("blue");


        red=ServletUtil.convert(r,req);
        green=ServletUtil.convert(g,req);
        blue=ServletUtil.convert(b,req);
        }

        oper=req.getParameter("operation");


        HttpSession session=req.getSession();


        Image old_im=null;


    if (oper==null)
        old_im=(Image)(session.getValue(sa[2]));

    else
        old_im=(Image)(session.getValue(sa[3]));


        ServletUtil.im_load(old_im,frame);


     if (old_im==null) {
       System.out.println("old_im is null");
       return;
    }

        int w=old_im.getWidth(frame);
        int h=old_im.getHeight(frame);

        if(w<=0||h<=0) {
          // res.sendError(res.SC_NOT_FOUND,"IMAGE NOT
LOADED");
```

```
        return;
      }

      int old=getColor(old_im,w,h, x, y);

      Image new_im=Toolkit.getDefaultToolkit().createImage
             (new  FilteredImageSource(old_im.getSource(),
                 new ColorFilter(old,red,green,blue)));


     session.putValue(sa[3], new_im );

      ServletUtil.sendGif(new_im,res);

   }

    public void doPost(HttpServletRequest req,
              HttpServletResponse res)
              throws ServletException, IOException {
      doGet(req,res);
   }

   public void destroy(){
    if(frame !=null) frame.removeNotify();
   }


  private int getColor(Image im, int w,int h,int x, int y)
{

     int[] pixels=new int[w*h];

   try {
     PixelGrabber pg=
             new PixelGrabber(im,0,0,w,h,pixels,0,w);
     pg.grabPixels();
    }
   catch (InterruptedException e){
     e.printStackTrace();//should return something
   }
     int pixel=((int[]) pixels)[y*w+x];

     return ColorModel.getRGBdefault().getRGB(pixel);
  }
```

```
}
```

ChooseImageServlet.java

```
package littleBee_proj;

import javax.servlet.*;
import javax.servlet.http.*;
import java.awt.*;
import java.awt.image.*;
import java.io.*;

/**************************************************************
************
 *
 * Yan Hai
 * 03/25/02
 *
 * This servlet chooses the appropriate style/appliqué to
 * display So user can make composition of the two images.
 *
 * @see ServletUtil
 *

**************************************************************
*********/

public class ChooseImageServlet extends HttpServlet{

    Frame frame=null;

    public void init(){
      frame=new Frame();
      frame.addNotify();
    }

    public void doGet(HttpServletRequest
                req,HttpServletResponse res)
      throws ServletException,IOException {
      int cas;
      cas=ServletUtil.convert(req.getParameter("case"),req);
```

```
    HttpSession se=req.getSession();

    Image imp = (Image) se.getValue("patternImage");
    Image ima = (Image) se.getValue("appliqueImage");
    Image imup =
           (Image)se.getValue("updatedPatternImage");
    Image imua =
           (Image)se.getValue("updatedAppliqueImage");

    ServletUtil.im_load(imp,frame);
    ServletUtil.im_load(ima,frame);
    ServletUtil.im_load(imup,frame);
    ServletUtil.im_load(imua,frame);

    if(cas==0) {
       ServletUtil.sendGif(imp,res);

    }
    else if (cas==1) {
       ServletUtil.sendGif(imup,res);
       se.putValue("updatedPattern","yes");
    }
    else if (cas==2) {
       ServletUtil.sendGif(ima,res);
    }

    else {
       ServletUtil.sendGif(imua,res);
       se.putValue("updatedApplique","yes");
    }

  if(frame != null) frame.removeNotify();
}

public void doPost(HttpServletRequest req,
            HttpServletResponse res)
    throws ServletException, IOException {
    doGet(req,res);
}

public void destroy() {
    if(frame!=null)  frame.removeNotify();
}
```

```
}  //  end class ChooseImageServlet
```

CompositeServlet.java

```
package littleBee_proj;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.awt.*;
import java.awt.image.*;

/**************************************************************
************
 * Yan Hai
 * 03/25/02
 *
 * This servlet composites two images, put appliqué at
 * mouse clicked position.
 *
 * @see ServletUtil
 *

**************************************************************
*********/
public class CompositeServlet extends HttpServlet {

   Frame frame=null;

   public void init() {
      frame=new Frame();
      frame.addNotify();
   }

   public void doGet(HttpServletRequest req,
                     HttpServletResponse res)
      throws ServletException, IOException {
      HttpSession session=req.getSession();
    //  ServletUtil su=new ServletUtil();

      String[] s=new String[2];
      if(session.getValue("updatedPattern")==null)
```

```
        s[0]="patternImage";
    else
        s[0]="updatedPatternImage";

    if (session.getValue("updatedApplique")==null)
        s[1]="appliqueImage";
    else
        s[1]="updatedAppliqueImage";

    Image p=(Image)session.getValue(s[0]);
    Image a=(Image)session.getValue(s[1]);

    int x=
      ServletUtil.convert(req.getParameter("p.x"),req);
    int y=
      ServletUtil.convert(req.getParameter("p.y"),req);
    ServletUtil.im_load(p,frame);
    ServletUtil.im_load(a,frame);

    int pw=p.getWidth(frame);
    int ph=p.getHeight(frame);
    int aw=a.getWidth(frame);
    int ah=a.getHeight(frame);
    BufferedImage pbi=
        ServletUtil.toBufferedImage(p,pw,ph);
    BufferedImage abi=
        ServletUtil.toBufferedImage(a,aw,ah);

    BufferedImage composite=
     new  BufferedImage(pw,ph,BufferedImage.TYPE_INT_ARGB);

    Graphics2D g=composite.createGraphics();

    g.drawImage(pbi,0,0,frame); //see class Graphics
//  g.drawImage(abi,x,y,frame);
    g.drawImage(abi,x-aw/2,y-ah/2,frame);

    Image im=ServletUtil.toImage(composite);


    session.putValue("compositeImage",im);
    ServletUtil.sendGif(im,res);

    if(g!=null) g.dispose();
}
```

```
    public void doPost(HttpServletRequest req,
                HttpServletResponse res)
        throws ServletException, IOException {
        doGet(req,res);
    }

    public void destroy() {
        if(frame!=null)  frame.removeNotify();

    }

}  // class CompositeServlet
```

ServletUtil.java

```
package littleBee_proj;

import java.awt.image.*;
import java.awt.*;
import java.io.*;
import javax.servlet.http.*;

import Acme.JPM.Encoders.GifEncoder;

/**********************************************************
***********
 * Yan Hai
 * 03/25/02
 *
 * This servlet serves as a utility class.

**********************************************************
**********/

public class ServletUtil {

    public static int convert(String s,HttpServletRequest
        req) {

        int i=0;
        try {
```

```
      int k=Integer.parseInt(s);
      System.out.println("k="+k);

       if((req.getParameter("task")!=null)&&
         (req.getParameter("task").indexOf("Color")!=-1))
         {
          if ((k<=255)&&(k>=0))
           i=k;
          else i=0;
         }

          else i=k;
   }
   catch (Exception e) {
      System.err.println(e.getMessage()+"\n");
      e.printStackTrace();
   }
   return i;
}

public static BufferedImage toBufferedImage(Image im,
     int w, int h) {

   BufferedImage bi=
      new  BufferedImage(w,h,BufferedImage.TYPE_INT_ARGB);

   Graphics g=bi.createGraphics();
   g.drawImage(im,0,0,w,h,null);
   g.dispose();
   return bi;
}

public static Image toImage(BufferedImage bi){

   Image im=
Toolkit.getDefaultToolkit().createImage(bi.getSource());
   return im;
}

public static void im_load(Image im,Frame frame) {
   MediaTracker mt=new MediaTracker(frame);
   mt.addImage(im,0);

   try {
      mt.waitForAll();
```

```
        }
        catch (InterruptedException e) {
           e.printStackTrace();
           return;
        }
     }

   public static void sendGif(Image im, HttpServletResponse
res) {

        res.setContentType("image/gif");
        res.setHeader("Cache-Control","no-cache");
        res.setHeader("Pragma","no-cache");
        res.setHeader("Cache-Control","no-store");
        res.setHeader("Expires","0");
        try {
          OutputStream out=res.getOutputStream();
          new GifEncoder(im,out).encode();

          out.flush();
          out.close();
        }
        catch(IOException e) {
          try {

res.sendError(res.SC_INTERNAL_SERVER_ERROR,e.getMessage());

          } catch (Exception ex){}
          e.printStackTrace();
          return;
        }
     }
}  //class ServletUtil
```

WhiteBackgroundFilter.java

```
package littleBee_proj;
import java.awt.image.*;
import java.awt.*;


/**********************************************************
***********
 * Yan Hai
```

```
 * 03/25/02
 *
 * This class filters white background of an image.
 * @see ConnectionServlet
 *
**********************************************************
*********/

class WhiteBackgroundFilter extends RGBImageFilter {

   public WhiteBackgroundFilter() {

      canFilterIndexColorModel=true;

      }
   public int filterRGB(int x, int y, int rgb) {

     DirectColorModel cm=
       (DirectColorModel)ColorModel.getRGBdefault();

    int alpha=cm.getAlpha(rgb);
    int red=cm.getRed(rgb);
    int green=cm.getGreen(rgb);
    int blue=cm.getBlue(rgb);

    if((red==255)&&(green==255)&&(blue==255))
      {
        alpha=0;
        Color c=new Color(red,green,blue,0);
        return c.getRGB();
      }
    else return rgb;

   }
}
```

ColorFilter.java

```
package littleBee_proj;
import java.awt.*;
import java.awt.image.*;

/**********************************************************
************
```

```
 *
 * Yan Hai
 * 03/25/02
 *
 * This class changes color of an image.
 *
 * @see Change_PCServlet
 *

**********************************************************
*********/
public class ColorFilter extends RGBImageFilter{

   private int changed;
   private int selected;


   public ColorFilter(int old_color,int r,int g, int b)
     {
       canFilterIndexColorModel=true;

       Color c=new Color(r,g,b);
       changed=c.getRGB();

       selected=old_color;
     }

   public int filterRGB(int x, int y, int rgb) {
      DirectColorModel cm=
         (DirectColorModel)ColorModel.getRGBdefault();
      if ((rgb==selected)&&(cm.getAlpha(rgb)!=0))
          return changed;

      else
          return rgb;
   }
}
```

DisplayPattern.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">

<html><head><title>Display Style</title>
<meta http-equiv="Content-Type"
content="text/html;charset=iso-8859-1">
```

```
</head>

<body>
<% String a=request.getParameter("selectpattern");
   if(a==null)
      out.println("<I><B>You didn't choose any style,
please select one.</B></I>");
%>
<br></br><br>
<br><B>The Style you chose:  </B>
</br>
<body bgcolor="#FFFFFF" text="#000000">
<blockquote>

<% if (session.getValue("updatedPattern")!=null)
      session.removeValue("updatedPattern");
   if (session.getValue("pflag")!=null)
      session.removeValue("pflag");
%>
<FORM
Action="/myapp/servlet/littleBee_proj.ControllerServlet"
      method="get">
<INPUT TYPE="HIDDEN" NAME="task" VALUE="changePatternColor"
>

<p align="left"> Change color: give the red, green, blue
values (must be between 0 and 255), then click on the image
where you want to change the color.
</p>
            <p align="center">
               R: <input type="text" name="red" size="5"
value="">
               G: <input type="text" name="green" size="5"
value="">
               B: <input type="text" name="blue" size="5"
value="">
            </p>
<p align="center"> <INPUT TYPE="IMAGE" NAME="pattern"
SRC="/myapp/servlet/littleBee_proj.ConnectionServlet?task=r
etrievePattern&selectpattern=<%=request.getParameter("selec
tpattern") %>">

</FORM>
```

```
<p align="center"><A
href="/littlebee/webpages/chooseapplique.html">Go to choose
applique</A>

</blockquote>

<body>
</html>
```

DisplayApplique.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<html><head><title>Display Applique</title>
<meta http-equiv="Content-Type"
content="text/html;charset=iso-8859-1">
</head>

<body>
<body bgcolor="#FFFFFF" text="#000000">

<% String a=request.getParameter("selectapplique");
   if(a==null)
      out.println("<I><B>You didn't choose any applique,
please select one.</B></I>");
%>
<br><br>
<br><B>The Applique you chose:</B>
</br>
<blockquote>

<% if (session.getValue("updatedApplique")!=null)
      session.removeValue("updatedApplique");
   if (session.getValue("aflag")!=null)
      session.removeValue("aflag");
%>
<form
Action="/myapp/servlet/littleBee_proj.ControllerServlet"
      method="get">
<input type="hidden" name="task"
value="changeAppliqueColor" >

<p align="left"> Change color: give the red, green, blue
values (must be between 0 and 255), then
click on the image where you want to change the color. </p>
```

```
<p align="center">
R: <input type="text" name="red" size="5" value="">
G: <input type="text" name="green" size="5" value="">
B: <input type="text" name="blue" size="5" value="">
 </p>
<p align="center"> <INPUT TYPE="IMAGE" name="applique"
SRC="/myapp/servlet/littleBee_proj.ConnectionServlet?task=r
etrieveApplique&selectapplique=<%=
request.getParameter("selectapplique") %>">

</form>
<p align="center">
<A href="/myapp/littlebee/jsp/ChangePosition.jsp">Go to
choose position</A>
</blockquote>

<body>
</html>
```

ChangePatternColor.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML><HEAD><TITLE>Change Pattern Color</TITLE>
<META content="text/html; charset=iso-8859-1" http-
equiv=Content-Type>
</HEAD>

<BODY bgColor=#ffffff text=#000000>
<% String r,g,b;
   r=request.getParameter("red");
   g=request.getParameter("green");
   b=request.getParameter("blue");
  if((r.length()==0)||(g.length()==0)||(b.length()==0))
     out.println("<I><B>Please enter the red/green/blue
value");
%>

<P align=left><B>This is the style after you change
color:</B></P>

<BLOCKQUOTE>
<FORM
ACTION="/myapp/servlet/littleBee_proj.ControllerServlet"
```

```
      METHOD="get" >
<INPUT TYPE="HIDDEN" NAME="task"
VALUE="changePatternColor" >
<INPUT TYPE="HIDDEN" NAME="operation"
VALUE="ChangeMoreColor" >

<p> Change color again:(red,green,blue values must be
between 0 and 255)
<p align="center">
        R: <input type="text" name="red" size="5" value="">
        G: <input type="text" name="green" size="5"
value="">
        B: <input type="text" name="blue" size="5"
value="">
</p>

<% String sr;
   if (request.getParameter("operation")==null) {

sr="/myapp/servlet/littleBee_proj.Change_PCServlet?red="+
request.getParameter("red")+ "&green="+
        request.getParameter("green")+"&blue="+
request.getParameter("blue")+"&pattern.x="+
        request.getParameter("pattern.x")+"&pattern.y="+
        request.getParameter("pattern.y");
   }
   else  {

sr="/myapp/servlet/littleBee_proj.Change_PCServlet?red="+
request.getParameter("red")+"&green="+
    request.getParameter("green")+"&blue="+
    request.getParameter("blue")+"&pattern.x="+
    request.getParameter("pattern.x")+"&pattern.y="+
    request.getParameter("pattern.y")+
    "&operation="+request.getParameter("operation");
   }
%>

<p align="center"> <INPUT TYPE="IMAGE" NAME="pattern"
SRC="<%= sr %>" >

</FORM>
</BLOCKQUOTE>

<% session.putValue("pflag","yes"); %>
```

```
<BLOCKQUOTE>
  <P align="center"><A
href="/littlebee/webpages/choosestyle.html">
  Go to choose style again </A></P>
</BLOCKQUOTE>
<BLOCKQUOTE>
  <P align="center"><A
ref="/littlebee/webpages/chooseapplique.html">
  Go to choose applique </A></P>
</BLOCKQUOTE>

</BODY></HTML>
```

ChangeAppliqueColor.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML><HEAD><TITLE>Change Applique Color</TITLE>
<META content="text/html; charset=iso-8859-1" http-
equiv=Content-Type>
</HEAD>

<BODY bgColor=#ffffff text=#000000>

<% String r,g,b;
   r=request.getParameter("red");
   g=request.getParameter("green");
   b=request.getParameter("blue");

  if((r.length()==0)||(g.length()==0)||(b.length()==0))
     out.println("<I><B>Please enter the red/green/blue
value</I></B>");
%>
<P align=left><B>This is applique after you change color
</B></P>

<BLOCKQUOTE>
<FORM
ACTION="/myapp/servlet/littleBee_proj.ControllerServlet"
      METHOD="get" >
<INPUT TYPE="HIDDEN" NAME="task"
VALUE="changeAppliqueColor" >
```

```
<INPUT TYPE="HIDDEN" NAME="operation"
VALUE="ChangeMoreColor" >
<p> Change color again: (red,green,blue values must be
between 0 and 255)</p>
     <p align="center">
               R:<input type="text" name="red" size="5"
value="">
               G:<input type="text" name="green" size="5"
value="">
               B:<input type="text" name="blue" size="5"
value="">
     </p>

<% String sr;
   if (request.getParameter("operation")==null) {

sr="/myapp/servlet/littleBee_proj.Change_PCServlet?red="+re
quest.getParameter("red")+
      "&green="+request.getParameter("green")+"&blue="+
request.getParameter("blue")+
      "&applique.x="+ request.getParameter("applique.x")
+"&applique.y="+request.getParameter("applique.y");
   }
   else  {

sr="/myapp/servlet/littleBee_proj.Change_PCServlet?red="+re
quest.getParameter("red")+"&green="+
request.getParameter("green")+"&blue="+
request.getParameter("blue")+"&applique.x="+
request.getParameter("applique.x")+"&applique.y="+
request.getParameter("applique.y")+"&operation="+
request.getParameter("operation");
   }
%>

<p align="center"> <INPUT TYPE="IMAGE" NAME="applique"
SRC="<%= sr %>" >

</FORM>
</BLOCKQUOTE>


<% session.putValue("aflag","yes"); %>

<BLOCKQUOTE>
```

```
  <P align="center">
<A href="/littlebee/webpages/chooseapplique.html">Go to
choose applique again </A></P>
</BLOCKQUOTE>
<BLOCKQUOTE>
  <P align="center">
<A href="/myapp/littlebee/jsp/ChangePosition.jsp">Go to
choose applique position </A></P>
</BLOCKQUOTE>

</BODY></HTML>
```

ChangePosition.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<html><head><title>Change Applique Position</title>
<meta http-equiv="Content-Type"
content="text/html;charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">

<p><B>Choose applique position</B></p>

<p>Point the mouse to the position where you want the
applique to be on the clothes,then click </P>

<form
action="/myapp/servlet/littleBee_proj.ControllerServlet"
      method="get" >
<input type="hidden" name="task" value="compositeImages" >
<% String psrc;

   if(session.getValue("pflag")==null)

psrc="/myapp/servlet/littleBee_proj.ChooseImageServlet?case
=0";
   else

psrc="/myapp/servlet/littleBee_proj.ChooseImageServlet?case
=1";
%>
```

```
<input type="Image" name="p" src="<%= psrc %>"
align="left">

<% String asrc;

   if (session.getValue("aflag")==null)

asrc="/myapp/servlet/littleBee_proj.ChooseImageServlet?case
=2";

   else

asrc="/myapp/servlet/littleBee_proj.ChooseImageServlet?case
=3";
%>

<Image src="<%= asrc %>"  align="center">
</form>

</body></html>
```

SendComposition.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">

<html><head><title>Composite image</title>

<meta http-equiv="Content-Type"
content="text/html;charset=iso-8859-1">
</head>

<body>
<body bgcolor="#FFFFFF" text="#000000">

<p> <B>This is the style after you choose the position of
style: </B></p>

<p> You can change the applique position by clicking on the
clothes, pointing mouse to where you want to put the
applique</p>

<FORM
action="/myapp/servlet/littleBee_proj.ControllerServlet"
```

```
      method="get" >

<input type="hidden" name="task" value="compositeImages" >

<% String s;
s="/myapp/servlet/littleBee_proj.CompositeServlet?"+"p.x="+
  request.getParameter("p.x")+"&p.y="+
  request.getParameter("p.y");
%>

<P align="center"><input type="Image" name="p" src="<%=s
%>"   >
</FORM>

<p align="center">

<ahref="/myapp/littlebee/jsp/Download.jsp" >

Go to download </p>

<p align="center"><a
href="/littlebee/webpages/choosestyle.html"> Go to choose
style</A> </p>
<p
align="center"><ahref="/littlebee/webpages/chooseapplique.h
tml" > Go to choose applique</A> </p>

</body></html>
```

Download.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<html><head><title>Download</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
</head>

<body>
<body bgcolor="#FFFFFF" text="#000000">

<%@ page import="java.awt.*,java.awt.image.*,java.io.*"  %>
<%@ page import="littleBee_proj.*" %>
<%@ page import="Acme.JPM.Encoders.GifEncoder" %>
```

```
<% Image im=(Image)session.getValue("compositeImage");
   Frame frame=new Frame();
   ServletUtil.im_load(im,frame);
   response.setContentType("image/gif");
   response.setHeader("Cache-Control", "public");
   try {
     OutputStream output=response.getOutputStream();
     new GifEncoder(im,output).encode();
     output.flush();
     output.close();
   }
   catch (IOException e){}

%>
</P>
</body></html>
```

index.html

```
<html>
<head>
<title>Introduction Page</title>
<meta http-equiv="Content-Type"
content="text/html;charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">
<p align="left"> </p>
<p align="left"> </p>
<p align="left"> </p>
<p align="center"><b>Littlebees: This is a website where
you can design baby's wear online.</b></p>
<p align="center"> </p>
<p align="center"> <a
href="/littlebee/webpages/choosestyle.html">You Design</a>
</p>
</body>
</html>
```

Choosestyle.html

```
<html>
<head>
<title>choose style </title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
```

```
</head>

<body bgcolor="#FFFFFF" text="#000000">

<p align="center"> </p>
<p align="center"><b>Select the basic style
here:</b></p>
<p align="center"> </p>
<p align="center">
<Form
Action="/myapp/servlet/littleBee_proj.ControllerServlet"
      method="get">
<input type="hidden" name="task"
value="retrievePattern">
  <select name="selectpattern"  size="3">
    <option value="Overall">Overall</option>
    <option value="Shirt">Shirt </option>
    <option value="Pants">Pants</option>
    <option value="Romper">Romper</option>

  </select>


</p>

<p align="center"> </p>

<p align="center">
   <input type="submit" name="Submit" value="Render">

</Form>

</p>
<p> </p>
<p align="center"> </p>
</body>
</html>
```

Chooseapplique.html

```
<html>
<head>
<title>Choose applique </title>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">

<p align="center"> </p>

<p align="center"><b>Select the applique here:</b></p>

<p align="center"> </p>
<p align="center">

<Form
Action="/myapp/servlet/littleBee_proj.ControllerServlet"
      method="get">

<input type="hidden" name="task"
value="retrieveApplique">
<p align=center style='text-align:center'>

<SELECT  NAME="selectapplique" SIZE="3">
<OPTION VALUE="Bear">Teddybear
<OPTION VALUE="Mushroom">Mushroom
<OPTION VALUE="Letters">Letters
<OPTION VALUE="Heart">Heart
<OPTION VALUE="Flower">Flower

</SELECT></p>


<p align=center style='text-align:center'> </p>

<p align=center style='text-align:center'><INPUT
TYPE="submit" VALUE="Render" NAME="Submit"></p>
</Form>

<p> </p>

<p align=center style='text-align:center'> </p>

</div>
</body>
</html>
```

apps.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<webapps>


   <Context path="/myapp"
            docBase="webapps/myapp"
            debug="0"
         reloadable="true" >

  </Context>

</webapps>
```

web.xml

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>

    <servlet>
      <servlet-name>controller</servlet-name>
      <description>
        This servlet plays the "controller" role in the MVC
        Architecture used in this application.
      </description>
      <servlet-
class>littleBee_proj.ControllerServlet</servlet-class>
      </servlet>

    <servlet>
      <servlet-name>database_connection</servlet-name>
      <description>
        This servlet produces GIF images that are
dynamically generated
        graphs, based on the input parameters included on
the request.
        It is generally mapped to a specific request URI
like "/graph".
```

```
      </description>
    <servlet-
class>littleBee_proj.ConnectionServlet</servlet-class>
    </servlet>

    <servlet>
    <servlet-name>change_color</servlet-name>
    <description>
       This servlet changes colors for styles and appliques
    </description>
    <servlet-class>littleBee_proj.Change_PCServlet
</servlet-class>
    </servlet>

    <servlet>
    <servlet-name>Choose_image </servlet-name>
    <description>
       This servlet choose the images to display
    </description>
    <servlet-class>littleBee_proj.ChooseImageServlet
</servlet-class>
    </servlet>

    <servlet>
    <servlet-name>composite </servlet-name>
    <description>
       This servlet composites two images
    </description>
    <servlet-
class>littleBee_proj.CompositeServlet</servlet-class>
    </servlet>

    <servlet>
    <servlet-name> utility </servlet-name>
    <description>
       This servlet functions as utility class of all
servlets
    </description>
    <servlet-class>littleBee_proj.ServletUtil </servlet-
class>
    </servlet>

   <session-config>
      <session-timeout>30</session-timeout>    <!-- 30
minutes -->
```

```
    </session-config>
</web-app>
```

APPENDIX C:

SQL FILES

CreateTables.sql

```
//create image tables script//
//create two tables: Patterns and Appliques in Scott
schema//
//create or replace//
clear buffer

connect sys/change_on_install;
grant create any directory to scott;
grant drop any directory to scott;
connect scott/tiger;

drop directory Pattern_Img;
create directory Pattern_Img as 'D:\MasterProject\patterns';
drop directory Applique_Img;
create directory Applique_Img as
'D:\MasterProject\appliques';

drop table Patterns;
create table Patterns(
     Pattern_No    NUMBER(3),
     Pattern       BFILE,
     Pattern_Name  VARCHAR2(20),
     CONSTRAINT Patterns_Pattern_No_pk PRIMARY KEY
(Pattern_No));

drop table Appliques;
create table Appliques(
     Applique_No   NUMBER(3),
     Applique      BFILE,
     Applique_Name VARCHAR2(20),
     CONSTRAINT Appliques_Applique_No_pk PRIMARY KEY
(Applique_No));

commit;
```

LoadAppliques.sql

```
//Load data to table Appliques//
//NOTE: DIRECTORY NAME SPECIFICATION//
clear buffer
insert into Appliques
```

```
     values (001,BFILENAME('APPLIQUE_IMG','bear.gif'),
'Bear');
insert into Appliques
     values (002,BFILENAME('APPLIQUE_IMG','flower.gif'),
'Flower');
insert into Appliques
     values
(003,BFILENAME('APPLIQUE_IMG','letters.gif'),'Letters');
insert into Appliques
     values (004,BFILENAME('APPLIQUE_IMG','heart.gif'),
'Heart');
insert into Appliques
     values (005,BFILENAME('APPLIQUE_IMG','mushroom.gif'),
'Mushroom');

commit;
```

LoadPatterns.sql

```
//Load data to table Pattern//
//Note: directory name spec: normal identifier is
interpreted//
//as uppercase,delimited identifer is interpreted as is//

clear buffer


insert into Patterns
     values (001,BFILENAME('PATTERN_IMG','overall.gif'),
'Overall');
insert into Patterns
     values (002,BFILENAME('PATTERN_IMG','shirt.gif'),
'Shirt');
insert into Patterns
     values (003,BFILENAME('PATTERN_IMG','romper.gif'),
'Romper');
insert into Patterns
     values (004,BFILENAME('PATTERN_IMG','pants.gif'),
'Pants');
commit;
```

REFERENCES

[1] Nich Efford, "Digital Image Processing, a Practical Introduction using Java", Addison-Wesley Pub Co., 2000

[2] David M. Geary, "Graphic Java, Mastering the JFC", Prentice Hall, 1999

[3] Marty Hall, "Core Servlets and Java Server Pages", Prentice Hall, 2000

[4] Lawrence H. Rodrigues, "Building Imaging Applications with Java", Addison-Wesley Pub Co., 2001