Capitulo 5. Desarrollo e Implementación

Este capítulo es la descripción final de esté sistema referente al código y la funcionalidad que se obtuvo. Es esencialmente la explicación de la implementación y funcionamiento total del sistema. El desarrollo consiste en aterrizar todo el diseño, llevarlo del papel a la realidad. Los problemas y limitaciones de implementación que surgieron se harán visibles en está parte.

Un objetivo primordial en este trabajo de tesis fue que dicho sistema logre demasiada transparencia a los usuarios finales, esto quiere decir, lograr un sistema que haga creer al usuario final que cualquier acción es un proceso muy simple ya que no son visibles las operaciones que realiza el sistema. En pocas palabras el sistema debe ser fácil de usar y comprender por el usuario.

5.1 Arquitectura del sistema

Para saber como se logra la transparencia debe estar clara la arquitectura del sistema (Fig. 5.1) en base a los tipos de usuarios, el como quedaran acomodadas todas las partes dentro del mismo. Estos usuarios se comunican con el sistema a través de un navegador Web. Dicho navegador se comunica directamente con el servidor Web el cual alberga al contenedor de *servlets* y *JSP`s*. En este sistema, el software *Tomcat*, un proyecto de la fundación de software *Apache* (http://www.apache.org), como se ah dicho anteriormente esté software se encargará de ser el servidor Web y contenedor de *servlets* y *JSP`s* al mismo tiempo.

Como se nota en la Fig. 5.1, los usuarios finales (administrador, alumno y profesor) nunca interactúan directamente con la base de datos o las “clases java”, todo se hace a través de un navegador Web, mediante el cual se logra la transparencia.

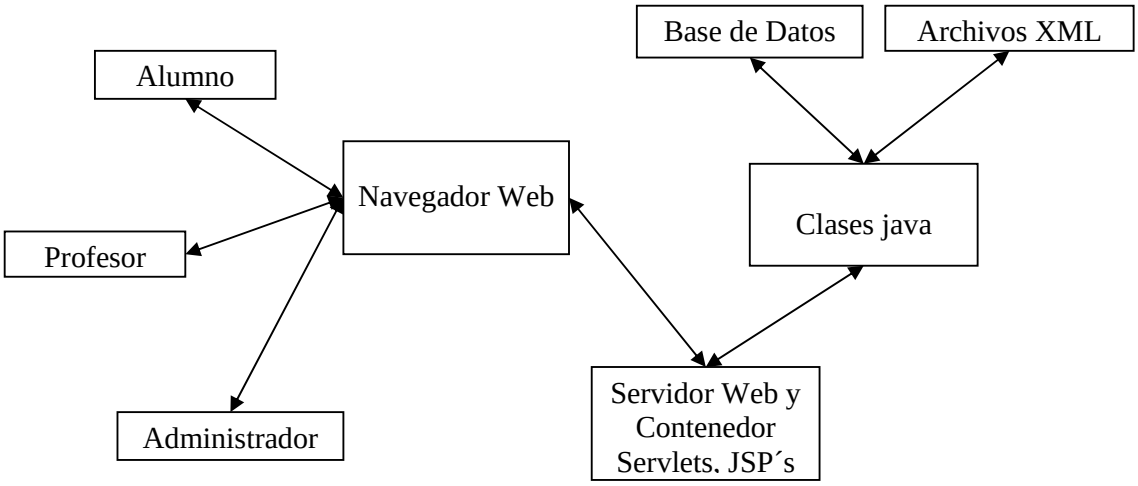

Fig.5.1 Arquitectura del Sistema

Hablando más técnicamente, como se había dicho anteriormente, esté sistema fue desarrollado bajo un patrón de diseño MVC(Modelo-Vista-Controlador) por lo cual la única parte con la que interactúa directamente con el usuario es la "vista", para este caso todas las vistas están representadas mediante JSP`s los cuales para cumplir su función de "vista" no pueden generar ningún tipo de información por si solos, es decir, los datos que ingrese cierto usuario en un JSP (vista) serán mandados a un servlet (controlador) que definirá el modelo (función, método o clase) donde se procesara la petición, una vez procesada dicha petición el modelo regresa sus resultados a dicho servlet el cual etiquetara los resultados de tal manera que en el siguiente JSP (vista), que se invoque, el usuario pueda comprender y/o entender la respuesta a su petición.

Este navegador sólo establece la comunicación entre los usuarios y el servidor donde se ejecutara la lógica aplicativa, que se encargarán de la persistencia de los datos dentro del sistema.

MVC: Implementación sistema de exámenes

De acuerdo a lo anterior para este trabajo de tesis la implementación del sistema para los tres tipos de usuarios se aplica un esquema MVC diferente pero bajo el mismo patrón, es decir:

Primero tenemos el nivel más genérico donde el controlador por decirlo así es el primer servlet que se invoca en el menú del tipo de usuario según el id y password del cliente, el modelo son las clases que permiten verificar los datos del usuario, y la vista son los JSPs que aparecen dependiendo el tipo de usuario.

El siguiente nivel usuario administrador el controlador es el servlet que dicho usuario llama para realizar una operación, el modelo son las clases que permiten que se lleven acabo las operaciones a las que tenga acceso el usuario, y la vista es la manera en que se le presentan operaciones que puede realizar el usuario mediante un JSP

En el nivel usuario alumno el controlador es el servlet que el cliente invoca para realizar un servicio, el modelo son las clases que permiten que se lleve acabo dicho servicio, y la vista por dar un ejemplo son los exámenes generados para el alumno a través de un JSP.

Por último tenemos el nivel usuario profesor donde el controlador es el servlet que se invoca para realizar un servicio de los que tiene acceso el cliente, el modelo son las clases que permiten que se lleven acabo dichos servicios, y la vista es la manera en que se despliegan los resultados a través de un JSP´s.

5.2 Manejo de sesión.

El usuario entra al sistema mediante un navegador Web ingresando el siguiente URL "http://<ipservidor>:8080/examenes/index.htm" donde mediante un formulario el usuario ingresa su login y password, para que establecer y/o iniciar una sesión.

El manejo de sesión es importante en este sistema, por dos razones, la primera es para poder mantener la consistencia de los datos en cada petición del usuario a través de los servlets y JSP´s debido a que por si solo el protocolo HTTP no tiene memoria. La segunda razón es que para este sistema tenemos que llevar un control de los usuarios en segundo plano por las siguientes razones:

- Para un usuario de tipo alumno es importante llevar control de la sesion para que no esté tentado a realizar algún tipo de trampa. La sesión se controla con una tabla en la base de datos. Cuando ingresa el alumno sus datos en la forma contenida en “index.hml”, está hace un llamado al servlet “inicioSesion” que hace una conexión con la base, luego valida los datos en la tabla “login” (Login y Password) y si son válidos verifica que dicho alumno no esté activo (es decir, si puede estar registrado, pero no activo) de lo contrario no permite la entrada al sistema. Si el usuario tiene acceso al sistema, accederá a “alumno.jsp” donde se le mostraran las operaciones que este puede realizar.

- Para el caso de un usuario profesor la finalidad de la sesión es para que nadie pueda modificar el control que lleva sobre sus alumnos y exámenes que desee aplicar. Cuando ingresa sus datos en la forma contenida en “index.hml”, está hace un llamado al servlet “inicioSesion” que hace una conexión con la base, luego valida los datos (Login y Password) y si son válidos permite el acceso al sistema mediante “profesor.jsp” donde se le mostraran las operaciones que éste puede realizar sobre el sistema.

Finalmente y retomando el caso de que el protocolo HTTP no tiene memoria, para lograr la comunicación y persistencia de los datos a través de los servlets y JSP´s, se utiliza una solución a este problema a través del API `HttpSession`. Esté API es utilizado dentro del sistema de la siguiente manera:

```
HttpSession sesion = request.getSession( true );
```

Está línea de código realiza dos funciones, la primera es crear un objeto `HttpSession` llamado "sesion" el cual servirá para realizar operaciones o lo que aplique sobre los datos de la sesión, la segunda función es retomar la sesión con el servidor o si aún no existe la establece con "true". Por tal razón está línea es indispensable tanto en servlets como en JSP´s para mantener el hilo (*thread*) entre el usuario y el servidor. Una vez que se tiene establecida la sesión, con la siguiente línea se asigna y/o modifica el atributo "id" a la sesión mediante el metodo "setAttribute".

```
sesion.setAttribute("id", request.getAttribute("aidi"));
```

Una vez definido el atributo "id" es utilizado para realizar operaciones u obtener información del usuario en la base de datos en la siguiente línea observamos como podemos hacer un llamado a este atributo con "getAttribute" para procesar alguna petición del usuario dentro de una forma HTML.

```
<input type="hidden" name="aidi" value="<%out.print(sesion.getAttribute("id"));%>">
```

Ya que se tiene establecida la sesión entre el usuario y sistema se procede a la descripción de las funciones que pueden realizar los usuarios.

5.3 Usuario Profesor

Como ya se estableció en el diseño, el profesor puede realizar 3 operaciones principales (activar examen, subir examen, ver resultados alumnos), las cuales a su vez tienen distintas modalidades:

- **Activar examen**.- está operación consiste en dos modalidades, activar y desactivar un examen.
- **Subir examen**.- está operación consiste en tres modalidades, crear examen, modificar preguntas y subir examen desde un archivo
- **Ver resultados**.- está operación consiste en dos modalidades, ver resultados por examen y permitir a un alumno volver a tomar algún examen.

5.3.1 Activar examen

Mediante la forma contenida en “profesor.jsp” el usuario hace la petición de activar examen, primero buscando los cursos que este tiene disponibles con el servlet “busactivarExam” y los muestra al usuario con “activaExam.jsp” para que este elija con una forma el curso y parcial, del examen que desee activar/desactivar, finalmente utiliza el servlet “activarExam” para conectar con la base de datos y actualizar la tabla de exámenes activos.

Como último comentario acerca de está opción el usuario además de poder activar exámenes en base al parcial que desee aplicar también puede activar “exámenes finales” los cuales serán generados a partir de las preguntas de todos los parciales asociados a dicho curso, pero el procedimiento para generar estos exámenes se explicara a detalle más adelante.

5.3.2 Subir examen

Antes de que el usuario pueda ver las diferentes modalidades para subir un examen al sistema, se le muestra al profesor los cursos que tiene asignados en la Base de Datos mediante el servlet “cargarExam” utilizando la clase “buscacursosProf2.class”, después se le mostraran dichos cursos con “ListCreaExam.jsp” de los cuales elegirá uno junto con el parcial al que estará asociado dicho examen.

Cuando está definido el curso y parcial al que se subirá preguntas de examen, el usuario debe decidir una de las tres modalidades mostradas en “cargaExam.jsp” para agregarlas a la Base de Datos. Estás modalidades son:

1.- Crear examen - Básicamente consiste en permitir que el usuario profesor introduzca manualmente las preguntas que desee que sean parte de cierto examen. Antes de proceder a introducir dichas preguntas se debe indicar el número de preguntas a capturar, después se introducen dichas preguntas mediante “capturaPreguntas.jsp” con el formato de la figura 4.1.1, una vez capturadas todas las preguntas y respuestas se llama al servlet “guarPregXML” el cual se encarga de crear un archivo en formato XML con las preguntas y respuestas capturadas, este archivo se creara utilizando los métodos de JDOM y Xerxes (ProcessingInstruction, XMLOutputter, output, Document, getcontent(), etc.), una vez creado dicho archivo el sistema mostrara al usuario con “previewExam.jsp” dicho archivo con las preguntas y respuestas que capturo, y se le presentaran 3 opciones, la primera será volver a editar dicho archivo en caso de que contenga algún error, la segunda opción permitirá al usuario guardar dicho archivo con formato XML en su propia PC por si desea volver a subir dichas preguntas para otro curso según le convenga al usuario, y la tercera y última opción es guardar dichas preguntas en la Base de Datos, lo cual se realiza mediante el servlet “guarPregBD”.

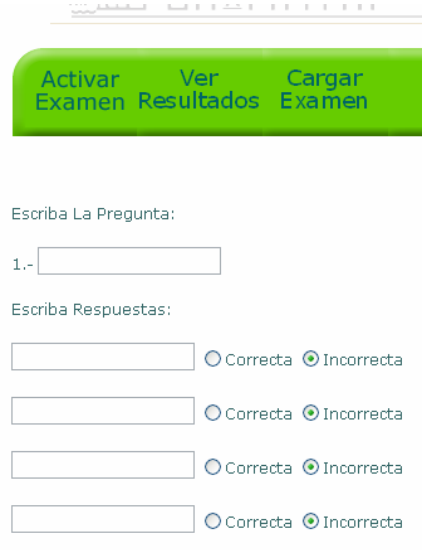

Fig.5.2.2 Forma generada a través de "capturaPreguntas.jsp"

2.- Modificar preguntas - En está modalidad el usuario tiene la opción de modificar las preguntas existentes en la base de datos, una vez que el usuario elije está opción el sistema las muestra todas las preguntas asociadas al curso y parcial en un forma similar a la Fig.5.2.2 pero con los campos llenos. Una vez realizada la modificación a las preguntas y/o respuestas, el sistema genera un archivo XML que se mostrara en "previewExam.jsp" donde se presentaran las opciones de guardar preguntas, guardar archivo y editar, explicadas en la modalidad anterior. A lo anterior surge la cuestión ¿porqué un profesor desearía modificar las preguntas almacenadas previamente?, a lo que se puede contestar que esto le permitirá corregir errores en la gramática, y/o simplemente evitar tener siempre el mismo planteamiento de las preguntas o respuestas.

3.- Subir examen desde archivo – Está última modalidad el usuario sube las preguntas mediante un archivo generado y guardado previamente en su PC por el sistema. Está funcionalidad es realizada mediante el servlet "uploadfileXML" el cual a través del paquete *Commons FileUpload* (desarrollado por la fundación de software *Apache*) logra subir el archivo al servidor para ser visualizado por el usuario a través de "previewExam.jsp" donde se presentaran las opciones de guardar las preguntas mostradas en la base de datos y editar el archivo de preguntas antes de proceder a subirlas en la base de datos.

5.3.3 Ver resultados

Está opción permite al usuario profesor dos modalidades, la primera es visualizar los resultados obtenidos en los exámenes ya sea por curso, sección, en general o por alumno. La segunda modalidad permite a ciertos usuarios alumnos volver a tomar dichos exámenes según el criterio del profesor. Como se muestra en la Fig 5.2.3 el profesor puede dar otra oportunidad al usuario alumno para que tome de nuevo cierto examen llenando la casilla de "si?" en la columna "Otra oportunidad" para que el sistema reactive el examen para cierto alumno, a través del servlet "newoportu" el cual se encarga de modificar la tabla de resultados para que el alumno pueda tomar de nuevo el examen.

Fig.5.2.3 Consulta de Resultados

5.4 Usuario alumno

Como ya se ah establecido previamente un alumno tiene dos opciones principales en el sistema (tomar examen y ver resultados).

- **Tomar examen**.- Está operación consiste primero en verificar los exámenes disponibles y elegir cual quiere tomar para contestar.
- **Ver resultados**.- Está operación consiste ver resultados obtenidos por examen.

5.4.1 Tomar examen

Aunque parezca una operación muy simple al usuario es una operación que implica varias acciones al sistema. Antes de continuar con la descripción de está situación se debe dejar claro que el sistema puede generar 2 tipos de exámenes, parcial y final. Después que el usuario ha entrado en la opción de “Tomar examen” el sistema mediante el servlet “verExamenes” mostrara la lista de exámenes disponibles y activados para dicho usuario ya sean parciales o finales, dicha lista se desplegara en “tomaExam.jsp”. El usuario alumno debe elegir examen que quiera aplicar. Si el examen es parcial la heurística del servlet “tomarExam” generara dicho examen sólo en base a las preguntas asociadas al curso, cierta sección y parcial. Por otro lado si el examen es final este será generado sólo teniendo como referencia el curso y sección sin importar el parcial, con lo cual se puede obtener un examen dinámico que obtenga sus preguntas de distintos parciales.

Cuando el sistema ha obtenido las preguntas y respuestas para que el alumno tome el examen, estás se guardan como un dato tipo List de tal manera que en cada posición contiene otro dato tipo List el cual en su primera posición almacena el texto de la pregunta, en la segunda posición una respuestá, la tercera contiene el valor de está respuesta correcta/incorrecta (0 ó 1), y si existen mas respuestas asociadas a la pregunta deben estar enlistadas de la misma manera. Este dato List se pasa a “genera.jsp” donde dicha Lista se despliega mediante el uso del método Iterator, pero recordemos que este dato contiene más datos tipo List por lo que para cada posición se debe utilizar el mismo método para poder imprimir el texto de dichas preguntas y respuestas como se muestra en el código de la Fig. 5.4.1, curiosamente si sólo declaramos un Iterator para imprimir los datos del primer List, únicamente se imprimen los datos del List contenido en la primera posición del primer List, es decir, genera una forma donde imprime la primera pregunta con sus respectivas respuestas, y para las demás preguntas no genera nada ni las imprime.

```
List resultados2=(ArrayList)request.getAttribute("preguntas");

String r = null;
int i=1;
String v = null;
int j=0;
Iterator it = resultados2.iterator();

    while(it.hasNext())
            {
                    List examenes =(List)it.next();
                    int z=1;
                    Iterator ite = examenes.iterator();

               out.print("<BR> "+i+".- "+ite.next());//imprime pregunta
               while(ite.hasNext())
                        {
                            r=(String)ite.next();
                            v=(String)ite.next();
                            if(v.equals("1"))
                                {
                                    j++;
                                }

                            out.print("<BR>");
                            %>
                            <input type="checkbox" name="pregunta<%out.print(i);%>" value="<%out.print(v);
                            <%out.print(" "+r); //imprime respuesta
```

Fig. 5.4.1 Código para imprimir examen en base a un List de List.

Nótese que para pasar de posición en posición en una lista se debe utilizar .next( ) sobre el Iterator. Como último comentario acerca de este código el tipo de dato que se almacena en cierta posición dentro del List tiene que ser igual o compatible con el que imprime los valores de dicho List ya que sino el sistema manda una excepcion y no puede imprimir nada. Por decir algo en este sistema se almacenan las preguntas y respuestas en la base de datos como tipo text, y el valor de cada respuesta (correcta/incorrecta) como int, pero cuando se requiere generar un examen estos datos son tomados como String al llenar el List por lo cual cuando de leen dichos datos para imprimir en pantalla (Fig.5.4.1.1) se hace el *casting* a String de la posición en el List.

Por otro lado, el sistema además de encargarse de imprimir las preguntas y respuestas, también brinda cierto nivel de seguridad para el usuario profesor. Debido a que además imprime en distinto orden de preguntas y respuestas, entre los exámenes de los alumnos que lo esten tomando al mismo tiempo, para evitar que se copien entre ellos, también el JSP(genera.jsp) tiene incrustado un código *javascript* que evita el plagio digital de las preguntas generadas dinámicamente por el sistema. Es decir, con el código incrustado se puede llegar a evitar que los alumnos se distribuyan las preguntas y respuestas para obtener calificaciones altas pero poco éticas. Aunque este código no es 100% seguro de violar para obtener dichas preguntas, se cuenta con otro *código javascript* encargado de evitar que el alumno pueda ver el código fuente del examen

generado y obtener a través de esté la información bloqueada. La justificación que se tiene para utilizar *javascript* es porque el examen no se puede bloquear desde un servlet o JSP debido a que las páginas en el navegador Web a pesar de ser generadas remotamente desde un servidor estas quedan alojadas temporalmente en la terminal (PC) desde la cual se hace la petición, es decir, el servidor sólo se encarga de generar la vista al usuario pero no tiene ni el poder o privilegio para bloquear alguna función del navegador utilizado en dicha terminal, sin en cambio un *javascript* va dentro del código de la página y se puede decir que realiza sus operaciones a través del navegador utilizando recursos de la terminal que aloja la pagina temporalmente.

El primer código *javascript* (ver apéndice B.2) incrustado evita que el usuario alumno seleccione el texto para evitar que sea copiado, es decir, si el alumno pulsa "CTRL + E" nada será seleccionado en el navegador (imágenes, texto, etc.) por lo cual nada podrá ser copiado aunque lo haga directamente en la barra de herramientas.

El segundo *javascript* (ver apéndice B.2)se encarga evitar que el alumno llegue fácilmente al código fuente de la página para poder obtener las preguntas y respuestas ya que cuando el usuario alumno presiona el botón derecho del "*Mouse*" aparecerá un menú distinto (Fig.5.4.1.2) al que aparece habitualmente y por consiguiente no tendrá la opción de ver código fuente ya que desde el ingreso del usuario al sistema, la barra de herramientas del navegador fue eliminada o no es visible al usuario, mediante otro *javascript* (ver apéndice B.2).

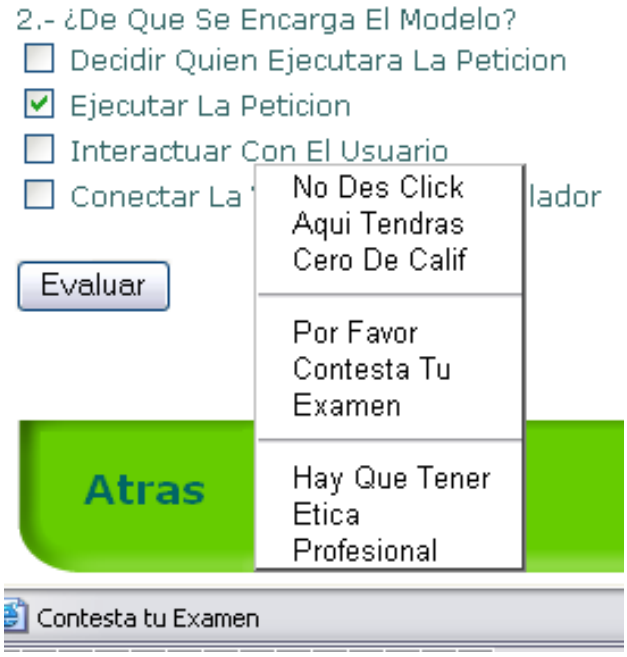

Fig.5.4.1.2 Menú botón derecho "*Mouse*"

Finalmente después de que el usuario alumno contesta el examen o ya no puede contestar más preguntas, esté debe oprimir el botón “evaluar” ubicado al final del examen como se muestra en la Fig.5.4.1.2, con lo cual la forma será enviada al servlet “evaluaExam”. Inmediatamente se realizara una evaluación en base al número de preguntas y respuestas correctas. Es decir, si una pregunta tiene 2 respuestas correctas y el alumno sólo contesto una en su evaluación sólo se sumara medio punto y así sucesivamente hasta sumar todos los puntos posibles de cada pregunta, para que al final se divida entre el numero total de preguntas. Una vez obtenida la calificación se captura en la base de datos si es que el alumno no ah tomado el examen previamente. Por último, se le muestra al usuario la calificación obtenida y las preguntas que contesto ya sean bien o mal.

5.4.2 Ver resultados de alumno

Después de que el alumno ha presentado al menos un examen en el sistema este podrá verificar sus calificaciones obtenidas. Una vez que el alumno elija la opción “ver resultados” se ejecutara el servlet “buscaCurso” el cual obtendrá la lista de cursos a los que está inscrito el alumno y se los mostraran mediante “eligeCurso.jsp”, aquí el alumno podrá elegir el curso del cual necesita saber sus calificaciones, después de elegir un curso se enviaran los datos al servlet “resultadoAlumno” el cual hará una conexión con la Base de Datos y se encargara de regresar la lista de calificaciones obtenidas en los exámenes que ah tomado el usuario para cierto curso.

5.5 Uso de directivas

El uso de directivas es indispensable para este sistema ya que el uso de éstas puede afectar la estructura general de un servlet o JSP traducido mediante el sistema por lo cual puede afectar directamente el resultado de alguna petición por parte del usuario final.

5.5.1 Directiva page

Las directivas permiten definir uno o más atributos dentro de los servlets o JSP. Por decir algo para este sistema la directiva *page* es usada para poder mostrar los valores de un dato tipo List en el JSP "genera.jsp" de tal forma que para este caso sino importamos el paquete "java.util.*" mediante está directiva no se podrá mostrar el examen generado al usuario. La manera adecuada para importar dicho paquete es como se muestra en la línea siguiente:

```
<%@ page import="java.util.*" %>
```

Está línea por su parte debe estar ubicada antes de la declaración de un dato tipo List o en su defecto antes del uso de algún método que utilice este paquete. La directiva *page* puede definir más atributos pero estos sólo son referentes a la página generada .Entre algunos de estos atributos tenemos language="java" el cual sólo puede tener como valor valido "java" debido a que está directiva fue pensada para incorporar otros lenguajes de programación pero por el momento java es el único especificado. Por otro lado, encontramos el atributo session="true|false" en este atributo por defecto se sabe que la variable "session" es del tipo HttpSession. El valor de "true" indica que la variable predefinida "session" debería unirse a la sesión existente si existe una, si no existe se debería crear una nueva sesión para unirla, simplemente para no perder la sesión existente con el servidor. Por otro lado, el valor de "false" indica que al intentar de acceder a la variable "session" resultarán en errores en el momento en que la página JSP llame o sea invocada mediante un servlet. Como puede notarse este tipo de atributo no se maneja para esté sistema mediante está directiva (véase tema 5.2), pero es una opción valida. Por ultimo la sintaxis para utilizar la directiva *page* es la siguiente:

```
<%@ page <Atributo> ="<Valor>" %>
```

5.5.2 Directiva include JSP

Por último tenemos que está directiva que se usa para incluir cierto archivo dentro de un JSP al momento de generarse, es decir, se pueden añadir más JSP´s, código HTML, archivos de texto e incluso archivos con código Java, teniendo cuidado de que

los archivos que anexemos no incluyan las etiquetas <html>, </html>, <body>, o </body>, ya que podrían confundirse con las del JSP que los contenga, generando una mala impresión de la página generada.

La directiva *inlcude* JSP se puede utilizar de dos maneras o modos, estático o dinámico:

- Estático: Es un archivo que su contenido es anexado tal cual al JSP resultante sin generar algún dato o información adicional.

- Dinámico: La página JSP realiza una solicitud a la URL indicada que envía de vuelta un resultado que es anexado a la página JSP.

En el sistema se utilizan ambos tipos, para el caso estático se pega código de algunas funciones *javascript* y para el caso dinámico cuando pegamos algún código *javascript* que utilice las funciones previamente anexadas al JSP, otro caso dinámico es cuando se pega otro JSP dentro del que se ya se está traduciendo, y así para obtener información acerca de la sesión para el caso de la barra de navegación o menú (véase Fig. 4.4.1). Por último, la sintaxis de la directiva *include* dentro de un JSP es la siguiente:

```
<%@ include file="URL" %>
```

La ventaja de utilizar la directiva *include* es que por un lado nos ahorramos re-escribir código HTML para cada JSP, para el caso de los *javascript* y la barra de navegación o menú de cada usuario (véase Fig. 5.5.1), por está razón el mantenimiento del sistema se reduce ya que si surge algún error en estos archivos referenciados no se tiene que estar modificando y probando el código para cada JSP, sino solamente el archivo del cual se referencia al error.

```
<%@ page language="java" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>

<html>
    <head>
        <title>Examen</title>
        <%@ include file="Style&ScriptMenu.jsp" %>
        <%@ include file="blockeartexto.jsp" %>
        <%@ include file="Rollscrpt.html" %>
        <link href="StyleEtsamenes" rel="stylesheet" type="text/css">
    </head>


    <body text="#336666" class="etsamenes">
    Examen<BR>
    <%@ include file="MenuScript.jsp" %>
```

Fig.5.5.1 Uso de directivas

Como se puede ver en la Fig.5.5.1 en la mayoría del código del sistema se utiliza ambas directivas (*page, include*) para establecer una buena estructura en las JSP y evitar repetir el mismo código en cada JSP, para el caso especifico de la barra de navegación de cada usuario (véase apéndice B).

5.6 Usabilidad

Por último, como se ha venido comentando a lo largo de este documento se requirió que este sistema sea fácil de usar por el usuario, para lo cual se requirió aplicar algunas reglas de diseño de interfaces [Weinschenk 2000] de tal manera que realmente esté sea útil al usuario final. A continuación se enumeran dichas reglas con respecto al sistema desarrollado.

1. Ley. Control de usuario: “La interfaz debe permitir al usuario sentir que el tiene el control y también permitirle apropiarse del control”. Esta ley se refiere a que el usuario no debe sentirse controlado, por esa razón en el sistema implementado siempre hay una barra de navegación visible al usuario sin forzarlo a realizar alguna actividad que realmente no quiera realizar.

2. Ley. Limitaciones Humanas: "La interfaz no debe saturar los límites cognoscitivos, visuales, auditivos, táctiles o motoras del usuario". Tal como es citado en esta ley la interfaz no se debe ahogar al usuario en un mar de opciones o procesos para que éste pueda aprender a utilizar el sistema rápidamente. Por tal razón en el sistema

desarrollado se buscó la mayor simplicidad en la interfaz tanto en el diseño (uso de 3 colores, 2 estilos de letra, etc.) como en la cantidad de opciones para realizar cualquier acción.

3. Ley. Integridad Modal: “La interfaz se ajustara a las tareas individuales sin importar la modalidad que se esté utilizando”. Es decir, para este sistema la interfaz tiene que ser sólo de modo visual (unimodal) ya que manejan datos complejos e información que no es fácil de recordar (preguntas, respuestas, claves de cursos, id, etc.) y además tiene que ser accesible vía Internet, por lo que no cabe utilizar o agregar un modo auditivo a menos que se manejaran mensajes cortos, o la velocidad de respuesta fuera fundamental.

4. Ley. Comodidad: “La interfaz se ajusta a la manera en que cada grupo de usuarios trabaja y piensa”. Para cumplir adecuadamente esta regla se diseñaron interfaces similares pero no iguales para los usuarios. Es decir, como esté sistema no está enfocado a un sólo tipo de usuario, la interfaz sólo sugiere tareas que puede realizar cierto tipo de usuario.

5. Ley. Claridad lingüistica: “La interfaz se comunicara tan eficientemente como sea posible”. Esto se refiere a que la información debe estar establecida dentro de un contexto que permita al usuario comprenderla. Para el caso de este sistema se logro utilizando tablas cuando se requiere presentar datos, y agregando opciones simples y claras.

6. Ley. Integridad estética: “La interfaz debe tener un atractivo y apropiado diseño”. No sólo se tiene que maximizar el funcionamiento sino también la preferencia. Si tomamos en cuenta que el sistema esta enfocado a la comunidad estudiantil de UDLA la interfaz está basada en una combinación de los colores institucionales (verde, naranja) ya que a mi parecer estos nos identifican como comunidad sin importar de cual sea el lugar de procedencia a esta institución. Para esta interfaz se diseñaron una serie de botones y se definieron estilos de letra de acuerdo a la combinación de colores ya mencionada (Fig.5.6.1).

Fig.5.6.1 Uso de colores institucionales

7. Ley. Simplicidad: “La interfaz presentara elementos simples”. Como se ha venido comentando en reglas anteriores una de las prioridades de este sistema fue lograr dicha simplicidad. Por un lado, para que los usuarios puedan utilizar fácilmente este sistema no deben perder tiempo adivinando en que consiste cada elemento, y por el otro lado se tiene que al mantener un cierto grado de simplicidad para tener un buen diseño.

8. Ley. Previsibilidad: “La interfaz se comportará de una manera tal que los usuarios puedan predecir exactamente qué sucederá”. Esta regla se refiere a que cuando el usuario elija una opción está debe ser realmente lo que cite, por eso las opciones para cada tipo de usuario deben ser demasiado claras como se cita en la regla 5.

9. Ley. Interpretación: “La interfaz hará conjeturas razonables sobre lo que el usuario está intentando hacer”. Por ejemplo, cuando el alumno elije la opción de tomar examen el sistema debe mostrarle los exámenes disponibles para dicho usuario, ya que la aplicación debe guiar al usuario a concretar la acción que desee realizar.

10. Ley: Exactitud: “La interfaz debe estar libre de errores”. Esto es algo casi imposible para cualquier aplicación pero no significa que no se puedan minimizar el número de errores. Para el caso de este sistema sé trato de minimizar los errores en la presentación de la interfaz ya que en algunas ocasiones es inevitable obtener malas impresiones de los datos utilizando etiquetas HTML.

11. Ley. Claridad técnica: “La interfaz tendrá la mas alta fidelidad posible”. Esto es referente a varios puntos, por decir algunos las barras de navegación siempre están visibles en la interfaz además de que son fácil de ubicar ya que se presentan con un color que contrasta con el color dominante (blanco), por último esta ley también indica que la calidad de las imágenes debe ser de alta resolución, lo que talvez para este sistema no es tan bueno ya que esto puede provocar una navegación lenta por lo que se opto por optimizar la calidad de las imágenes de tal manera que la navegación entre pantallas sea rápida.

12. Ley. Flexibilidad: “La interfaz debe permitir al usuario ajustar el diseño para su propio uso”. A mi parecer está ley es muy valida pero para esta aplicación no se considero por lo explicado en la regla 6 es decir ya se tiene un diseño basado en una combinación de colores para dar un sentido de identificación con la institución (Fig.4.5.7).

13. Ley. Cumplimiento: “La interfaz debe dar una experiencia de satisfacción al usuario”. La manera de satisfacer esta regla es ofrecer al usuario final mejores maneras de realizar sus tareas que la actual, haciendo su trabajo eficiente y simple. Partiendo de que ésta fue la primicia para desarrollar esté sistema, podemos decir que se han simplificado algunas tareas, tal como la manera en que un profesor aplica exámenes (ya no tiene que emplear tiempo para imprimir, revisar y calificar), al igual para un alumno no tiene que esperar a que el profesor evalué manualmente los exámenes que tome.

14. Ley. Propiedad cultural: “La interfaz será de acuerdo a costumbres sociales y las expectativas del usuario”. Es decir, el lenguaje y los términos empleados en este sistema deben ser entendibles y aceptables por la comunidad que puede llegar a utilizar dicho sistema.

15. Ley. Tiempo conveniente: “La interfaz operara en el momento conveniente para el usuario”. Esta regla está enfocada sistemas de reconocimiento de voz que muchas veces sólo están a la espera de un evento por parte del usuario, para este caso es aplicable en el sentido de que el usuario puede entrar al sistema cuando quiera siempre y cuando cuente con una conexión a la red del servidor.

16. Ley. Consistencia: “La interfaz será consistente”. Esta regla se refiere a que el comportamiento y acciones del sistema deben estar de acuerdo entre si. Es decir, por un lado se debe mantener el uso de colores y estilos en todas las paginas generadas dinámicamente y por otro lado no se debe permitir que cualquier opción dentro del sistema sea inválida o genere falla por falta datos para realizar alguna petición.

17. Ley. Soporte de usuario: “La interfaz proveerá asistencia adicional cuando sea necesario o requerida”. Esta regla sólo aplica para la opción de crear examen, ya que si el usuario después de crear subir o modificar las preguntas de cierto examen, podrá visualizar dicho examen y si no queda conforme, podrá editarlo cuantas veces sea necesario. Esta regla también se refiere a guiar al usuario implícitamente mediante opciones claras a través del sistema, lo que se logra para este sistema aplicando preguntas simples.

18. Ley. Precisión: “La interfaz permitirá a los usuarios realizar una tarea exacta”. Es decir, la interfaz permitirá al usuario interactuar con el sistema de tal forma que la tarea requerida sea realizada tal como lo desea el usuario, tal es el caso de la creación de un examen, por esta razón el código debe encontrarse documentado.

19. Ley. Perdón: “La interfaz hará acciones recuperables”. Está regla se refiere a que debe existir una opción “UNDO”, es decir, poder deshacer acciones previas, aunque dicha opción no este presente como tal en el sistema se puede decir que aparece a mediante de los botones “Atrás” y “Adelante”, para el caso de activar examen aparece mediante el botón “Desactivar”, sólo es inexistente cuando un alumno contesta un examen y es evaluado, ya que el único usuario con el poder de borrar esa calificación es el profesor.

20. Ley. Sensibilidad: “La interfaz informara a los usuarios acerca de los resultados obtenidos por sus acciones y el estado de la interfaz”. Es decir, por cada acción que realiza el usuario se le notifica el éxito o fracaso de la tarea realizada. Por nombrar algún ejemplo cuando un alumno toma cierto examen después de ser evaluado se le notifica su calificación, al igual el profesor siempre recibe notificación cuando activa y/o crea algún examen.

Guía	Descripción
1	Proveer al usuario una opción para regresar a la página anterior.
2	Proveer al usuario una opción de salida.
3	Proveer un diálogo de alerta al salir del sistema.
4	Evitar dos líneas en blanco consecutivas en el texto.
5	Diferenciar los encabezados del contenido.
6	Utilizar sólo dos tamaños y tipos de letra a lo largo de la página.
7	Ajustar el texto de las ligas a una sola línea
8	Optimizar el tamaño de los gráficos.
9	Hacer uso de colores contrastantes.
10	Optimizar el uso de tablas.

Fig. 5.6 Guías usables para PDA o para móviles, [Aguilar 2005]

Para finalizar este capítulo y aunque este sistema no está desarrollado para dispositivos móviles varias de las “Guías usables para PDA o para móviles” (Fig.5.6) descritas en la tesis de licenciatura *“Generación de interfaces usables para dispositivos móviles”* desarrollada por *Eduardo Aguilar Castillo, en primavera del 2005* (disponible en http://biblio.udlap.mx/tesis), son aplicables a esté sistema, ya que si tenemos en cuenta que en un dispositivo Móvil no se tienen todas las herramientas que encontramos en un navegador convencional y suponiendo que el usuario final no sabe utilizar dichas herramientas entonces se puede llegar a encontrar en un escenario muy similar al que se encuentran los usuarios de dispositivos móviles, de tal modo que si se busca que el sistema sea fácil de utilizar entonces algunas de estas guías son validas para la interfaz de este sistema, debido a que el usuario se basara de los recursos que le presente el

sistema para poder lograr su cometido final. A continuación se describe como influenciaron estas guías en el desarrollo de la interfaz del sistema:

La primer guía sugiere “Proveer al usuario una opción para regresar a la página anterior”; para esté sistema en la parte inferior encontramos una barra donde se encuentran dos botones para realizar este cometido uno para ir atrás y otro para ir adelante si aplica la situación (Fig. 5.6.2.).

Fig.5.6.2 Botones de navegación

En la segunda guía nos encontramos con lo siguiente: “Proveer al usuario una opción de salida”, está oración no sólo se refiere a una salida del sistema sino a una salida alternativa antes de concretar cierta operación, por lo que la barra de navegación del sistema siempre está visible, pero si lo que quiere es salir del sistema en dicha barra también nos encontramos con la opción cerrar sesión (Fig.5.6.3.) que a su vez es útil debido al manejo de sesiones (ver tema 5.2).

Fig.5.6.3 Opción de salida

En la tercer guía se enumera que se debe “Proveer un diálogo de alerta al salir del sistema” Es decir que se le notifique al usuario que ha salido del sistema (Fig. 5.6.4).

La guía cuatro, “Evitar dos líneas en blanco consecutivas en el texto”, no es aplicable a este sistema ya que está situación es imperceptible a comparación de cómo se notaria en un dispositivo móvil, es decir, debido al tamaño de su presentación son más visibles y molestos este tipo detalles.

Para la guía cinco tenemos, “Diferenciar los encabezados del contenido”, está es una guía muy valida para cualquier tipo de interfaz ya que más que nada provee de buena presentación y buen entendimiento a la interfaz(Fig.5.6.4.).

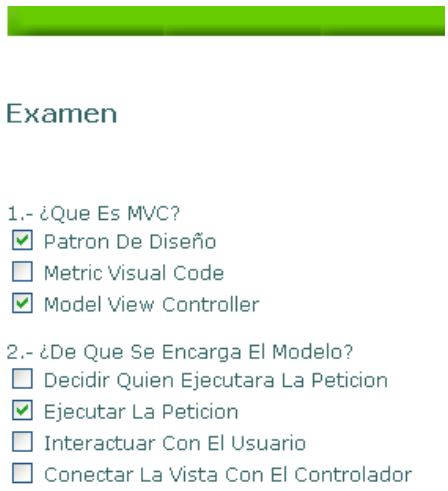

Fig.5.6.4 Diferenciar encabezados de contenido

Al igual que en la guía anterior para la numero seis, “Utilizar sólo dos tamaños y tipos de letra a lo largo de la página”, está guía es valida para varios tipos de interfaz ya que si el usuario se encuentra con una interfaz con el mismo estilo de letra o en su defecto mismo tamaño, podemos suponer que el usuario llegara a percibir que está enfrente de un texto plano y puede no comprender las respuestas a sus peticiones, haciéndolo creer que está interactuando con un sistema poco amistoso el cual no le indica o insinúa como navegar a través de él. Para el caso especifico de este sistema si cumple con la sugerencia de está guía (Fig.5.6.5.).

Fig.5.6.5 Manejo de dos estilos de letra

La guía número siete no es aplicable para este tipo de interfaz ya que en primer lugar en esté sistema no se hace uso de mas de una liga (aviso de sesión cerrada), y por otro lado, al igual que en la guía cuatro la percepción que se tiene en un navegador Web en un dispositivo móvil es muy distinta a la de un navegador de una Computadora convencional.

La guía numero ocho “Optimizar el tamaño de los gráficos”, hasta cierto punto es aplicable a está interfaz ya que como en la guía anterior el nivel de percepción es muy distinto, pero no por eso se deben imprimir en pantalla imágenes en un tamaño que deje sin visibilidad el resto de la página (Fig.5.6.6.).

Fig.5.6.6 Tamaño de las imágenes adecuado para no cubrir todo el texto

Por último ya que la guía díez tampoco aplica, tenemos a la nueve que sugiere “Hacer uso de colores contrastantes”, y tal como se ha referido en guías anteriores con este tipo de guía además de lograr cierto grado de usabilidad también permite dar

presentación y buen entendimiento al usuario, con lo que se logra que el sistema sea fácil de utilizar por el usuario final (Fig.5.6.7).

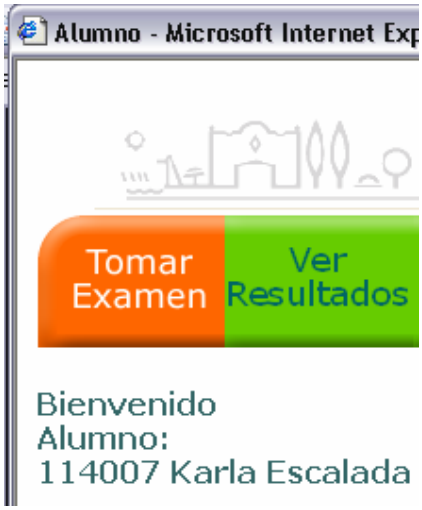

Fig.5.6.7 Uso de colores contrastantes tanto en el texto como en botones

5.7 Discusión

Las reglas y guías descritas anteriormente son de gran utilidad. Estas no garantizan el perfecto uso del sistema pero se puede decir que si elevan el grado de usabilidad del sistema desarrollado. Es decir, el considerar dichas reglas y guías hizo tomar en cuenta detalles que como desarrollador u programador nunca se toman en cuenta pero que son importantes y de gran trascendencia para el usuario final.