UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Actualización del Sistema de Registro de Dominios de la Universidad del Valle de Guatemala

Trabajo de investigación presentado por
Zaida Dihnora Orellana M.
para optar al grado académico de
Ingeniero en Ciencias de la Computación

Guatemala
2006

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Actualización del Sistema de Registro de Dominios de la Universidad del Valle de Guatemala

Zaida Dihnora Orellana M.

Guatemala

2006

Actualización del Sistema de Registro de Dominios
de la Universidad del Valle de Guatemala

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Actualización del Sistema de Registro de Dominios de la Universidad del Valle de Guatemala

Trabajo de investigación presentado por
Zaida Dihnora Orellana M.
para optar al grado académico de
Ingeniero en Ciencias de la Computación

Guatemala
2006

Vo.Bo.:

(f) _____
Asesor: MSc. Luis Furlán

Tribunal:

(f) _____
MSc. Luis Furlán

(f) _____
MSc. Rolando Rodríguez

(f) _____
MSc. Sergio Izquierdo

Fecha de aprobación: 5/Dic/2006

PREFACIO

Este proyecto surgió ante la necesidad de actualizar el sistema de Registro de Dominios del nivel superior .GT, administrado en la oficina J-204 de la Universidad del Valle de Guatemala, debido a que el actual sistema de Registro se encuentra en funciones desde 1992 y su sitio en Internet fue creado cuando aún no se había desarrollado el comercio electrónico, ni la tecnología de servicio de páginas Web había alcanzado un nivel como el de hoy en día. Además, los administradores del Registro han adquirido un nuevo Sistema de Registros el cual necesita una interfaz como primer paso para ponerlo a funcionar.

Además, la cantidad de dominios registrados ha crecido y se espera que crezca aún más debido al auge del comercio electrónico en Guatemala. Esto hace que muchas empresas guatemaltecas o extranjeras deseen adquirir un nombre de dominio y tener presencia en Internet desde donde podrán ofrecer servicios o productos de una región o país específico.

Para la realización de este proyecto se contó con la colaboración de los administradores del Registro de Dominios y del Coordinador de Recursos Interactivos del Departamento de la Tecnología Interactivas por lo cual agradezco su colaboración e información.

ÍNDICE

PREFACIO v
LISTA DE CUADROS vii
LISTA DE GRÁFICOS viii
RESUMEN ix

Capítulos

I. INTRODUCCIÓN 1
II. ANTECEDENTES 2
III. OBJETIVOS 3
IV. METODOLOGÍA 4
V. REGISTRO DE DOMINIO 5
VI. DESARROLLO DE LA APLICACIÓN 8
VII. CONCLUSIONES Y RECOMENDACIONES 36
VIII. BIBLIOGRAFÍA 39
IX. APÉNDICE 42
X. GLOSARIO 54

LISTA DE CUADROS

Cuadro Página

1. Configuración de la variable de ambiente PATH 11
2. Configuración de la variable de ambiente JAVA_HOME 11
3. Configuración de la variable de ambiente CATALINA_HOME 12
4. Configuración de la variable de ambiente CLASSPATH 13
5. Estructura de directorios 19
6. Elementos del archivo web.xml 21
7. doPost llama a doGet 22
8. Expresión regular para Nombres propios 24
9. División de cadenas de caracteres 25
10. setAttribute y getAttribute 27
11. Carga de Beans y propiedad *scope* 27
12. Acceso a propiedades de los Beans 29

LISTA DE GRÁFICOS

Ilustración Página

1. Marco izquierdo de la aplicación 16
2. Copia de datos de contacto administrativo 17
3. Front-end del nuevo Sistema de Registro 18
4. Estructura de la aplicación 19
5. Arquitectura II de Java Server Pages 28

RESUMEN

Este documento describe una primera parte de la interfaz que se desarrolló para ser implantada a un nuevo Registro de Dominios en línea, el cual se encuentra actualmente en la dirección de Internet www.gt.

Para esta implementación se utilizó la tecnología Java, específicamente JSP/Servlets integrado con JavaBeans, con el objetivo de obtener ventajas del uso de esta tecnología y de los servidores de aplicaciones que se encuentran disponibles y que están basados en este lenguaje, también se desea que esta nueva aplicación utilice páginas dinámicas para el despliegue de resultados. Y los administradores del sistema de Registro de Dominios, desean que el ingreso de los datos y la actualización de los mismos en el Registro se realicen de forma más rápida y efectiva. Y que algunos procesos del Registro se realicen sin la intervención de ellos.

Esta nueva implementación permite al usuario ingresar al Sistema de Registro por medio de un código y una contraseña, crear y actualizar la información asociada a un nombre de dominio, realizar el cambio de contraseña de los usuarios, consultar la base de datos, obtener los datos del pago en línea realizado por el cliente y guardar esta información en una base de datos diseñada específicamente para esta aplicación.

I. INTRODUCCIÓN

El Registro de Dominio para Guatemala, administrado actualmente en la oficina J-204 de la Universidad del Valle de Guatemala creó desde 1998 un sitio en Internet (www.gt) con el fin de dar a conocer a los cibernautas, los reglamentos y requisitos necesarios para registrar nombres de dominio con la terminación **.gt**.

En este sitio también se puede registrar nombres de dominio y realizar el pago de primera vez del mismo. Sin embargo, los administradores *del Registro* desean, desde hace algún tiempo, desarrollar una nueva interfaz e implantarla a un nuevo Sistema de Registro comprado para tal efecto. La interfaz a desarrollar debe hacer uso de otro lenguaje de programación y también de nueva tecnología en servicio de páginas Web, para estar más de acuerdo a los cambios tecnológicos de hoy en día. Además, se desea agregar nuevos módulos que permitan que los procesos de registro se agilicen.

Por tanto, este documento describe la puesta en marcha de la primera parte de la aplicación la cual modifica la interfaz del Registro de Dominios del sitio anteriormente indicado, haciendo uso del lenguaje y tecnología que ofrece Java para páginas Web y de nueva tecnología en servicio de páginas y además se agrega un nuevo módulo, quizás el más importante para este sitio, el cual permite actualizar la información de un dominio anteriormente registrado. De esta forma se inicia el proceso de actualización de la interfaz solicitada por los administradores del Registro de Dominios .gt.

II. ANTECEDENTES

El sistema actual de *Registro de Dominios .gt*, que se encuentra en la dirección Web anteriormente indicada, contiene la información de políticas, procedimientos, contratos, solución de controversias y tarifas del *Dominio .gt*. Además, se puede registrar nombres de dominio y realizar consultas de estos para obtener la información respectiva, esto se realiza a través de WHOIS, que es una base de datos que guarda información de los nombres de dominio registrados. *(Ver Apéndice E. Pantalla I y IX, p. 45 y 49 respectivamente)*

El proceso actual de Registro de un nombre de dominio se inicia al ingresar un nombre de dominio. A continuación, se presenta al cliente una lista de todos los sufijos que pueden agregarse al nombre de dominio escogido, si éste se encuentra disponible. Estos sufijos son: com.gt, net.gt, org.gt, edu.gt, mil.gt, gob.gt e ind.gt. Existe una excepción y es que como requisito para utilizar los sufijos edu.gt, gob.gt y mil.gt, el cliente debe presentar documentos que permitan validar su solicitud. *(Ver Apéndice E. Pantalla II y III, pp. 45,46)*

Como siguiente paso, se despliega al usuario una página compuesta de varios formularios donde el *registrante*, que es la entidad o persona solicitante del nombre de dominio, debe proporcionar los datos de la empresa, datos del contacto técnico, contacto administrativo, contacto para cobro y la información de dos servidores de nombres con su respectiva dirección IP. *(Ver Apéndice E. Pantalla IV,V,VI, pp. 46,47)* Esta página al igual que todo el sitio, fue desarrollado utilizando el lenguaje HTML (Hypertext Markup Language, en inglés).

Finalmente, el usuario envía el formulario y por medio de un mensaje a su correo electrónico se le confirma su solicitud; además, el usuario tiene la opción de realizar el pago *en línea* del dominio que acaba de registrar, si así lo desea. *(Ver Apéndice E. Pantalla VII, VIII, p. 48)*

III. OBJETIVOS

El objetivo principal del proyecto que se describe a continuación, es actualizar el módulo de *Registro de Dominios .gt,* utilizando nueva tecnología en cuanto a operación de sitios en Internet se refiere. Y agregar al Registro un nuevo módulo que permita la actualización de la información de un dominio. Todo lo anterior con el fin de poder agregar la aplicación desarrollada a un nuevo Sistema de Registro.

Específicamente se usarán páginas dinámicas utilizando para ello la tecnología que ofrece Java para el desarrollo de páginas Web permitiendo así que el proceso de Registro de Nombre de Dominio se realice con mayor eficacia y facilidad que actualmente. Otro objetivo es que, el desarrollo de la aplicación sea de bajo costo. El uso de Java y otras herramientas que no requieren pagar para hacer uso del software, permitirán que este objetivo se cumpla.

Además, los administradores del sitio desean que el Registro de Dominio no dependa tanto de ellos pues aún existen varios procesos dentro del Sistema que podrían realizarse de forma automática, y se realizan de forma manual. Este es el caso de la actualización de la información de los dominios, que aún se realiza manualmente. Es en este punto donde el nuevo módulo, que permite actualizar la información de un dominio, es parte importante de este proyecto.

La nueva aplicación debe contar con un módulo para el registro de un nombre de dominio, un módulo de actualización de la información del dominio haciendo uso de un código de acceso y contraseña en ambos casos, la consulta de un dominio a través de WHOIS, y también debe contar con un mecanismo para almacenar los datos del pago en línea enviados por el cliente. Toda información que se obtenga del cliente por medio de esta aplicación, debe almacenarse en una base de datos diseñada para uso exclusivo de esta aplicación.

IV. METODOLOGÍA

Se investigó sobre la tecnología de JSP/Servlets y Java Beans para la elaboración de páginas dinámicas en Internet; sobre servidores de aplicaciones Web que están basados en especificaciones de Java y algunas herramientas de software que facilitaran el desarrollo de la aplicación. Con la información obtenida se procedió a elaborar la aplicación para el Registro de Dominios de la Universidad del Valle de Guatemala.

Se utilizó Java porque este lenguaje es muy potente, orientado a objetos, multiplataforma y muy utilizado internacionalmente. Además, el software se puede bajar del sitio de Sun de forma gratuita.

Se efectuaron pruebas continuas de cada parte de la aplicación que se iba realizando para detectar problemas que se pudieran ir presentando. Se dedujo que no había error cuando la información obtenida de la base de datos era igual a la que se almacenó en ella.

V. REGISTRO DE DOMINIO

A. Nombre de dominio

Un nombre de dominio se utiliza para identificar un espacio de forma única en Internet. Este espacio puede ser accedido por medio de la Internet utilizando el nombre de dominio registrado en las transacciones realizadas como correo electrónico, navegación de páginas Web, etc.

Existen dos tipos de nombres de dominio de nivel superior TLD (Top Level Domain, en inglés) estos son: dominios geográficos o de código de país, y dominios genéricos. El dominio de código de país de nivel superior (ccTLD, por sus siglas en inglés) es utilizado generalmente por empresas u organizaciones que desean que su sitio en la Red sea asociado a una región o país. Estos dominios son administrados por alguna entidad u organización de ese mismo país, que haya obtenido la autorización de la IANA, que es la entidad reguladora del Sistema de Nombres de Dominio. El código de dominio *gt*, son las siglas especificadas por el ISO 3166-1, las cuales identifican a Guatemala, y la UVG (Universidad del Valle de Guatemala) es la entidad delegada para administrarlo.

Los dominios de tipo "genérico", fueron creados en los Estados Unidos, con el objetivo de proporcionar una categoría a las organizaciones o empresas que adquieren este tipo de dominio. Los dominios genéricos más conocidos son: **.com**, que es orientado a servicios comerciales; **.net** orientado a servicios en la red; **.org** que son para organizaciones sin ánimo de lucro o que no encajan en ningún otro dominio; **.edu** que es utilizado para instituciones educativas, y otros más como .gov, .mil, que quedaron sólo para uso de los Estados Unidos. Los dominios genéricos indicados anteriormente, son utilizados bajo el dominio .gt con algunas variantes, y además se utiliza el dominio, .ind que fue creado por los administradores del dominio .GT y es específicamente para personas individuales, no para entidades.

Toda computadora conectada al Internet tiene un número único de identificación *ej.* 199.23.188.2. Este número es llamado dirección IP, pero es más fácil de recordar un nombre *ej.* *www.miempresa.com.gt*, que un número. Por este motivo se creó el sistema de nombres dominio (Domain Name System, en inglés). Un nombre es asociado a un número, donde el número identifica una computadora dentro de la Red; y el nombre permite recordar más fácilmente esa computadora.

El agente registrador, asesora a los clientes, tramita sus solicitudes, procesan los registros de nombres y envían al *registro* la información de DNS (Domain Name Service, en inglés).

Por último, el *registro*, son las empresas que administran los registros de nombres de dominio sobre los que tiene autoridad. Mantienen bases de datos públicas del nombre de dominio TLD que le corresponda, sirven consultas que realicen otros servidores, delegan en agentes registradores los dominios y mantienen la información asociada a éstos. Además, establecen condiciones de acceso a sus bases de datos realizadas por los agentes registradores y las condiciones a cumplir.

B. Proceso de Registro de Dominio en la Universidad del Valle de Guatemala

Desde 1992 la IANA (Internet Assigned Numbers Authority), dio autorización a la Universidad del Valle de Guatemala para administrar el nombre de dominio de nivel superior *gt*.

Actualmente existen dos formas para realizar el proceso de registro de un dominio:

Una de ellas es presentarse a la oficina J-204, de la Universidad del Valle, y llenar allí un formulario con los datos requeridos de la empresa, contactos administrativo, técnico y de cobro y la información de los servidores de nombres.

Si el dominio que el cliente solicita está disponible, deberá pagar en el banco la cantidad de US$70.00, si es de nacionalidad guatemalteca, o US$80.00 si es extranjero. Este pago abarca dos años de administración del dominio y la inscripción.

Luego debe presentar en la oficina de *Administración del Dominio gt* el recibo de pago, si toda la información solicitada al cliente es correcta, deberá esperar alrededor de unas 4 horas hábiles para poder hacer uso del dominio.

La otra forma de realizar este proceso es llenar el formulario que se encuentra en el sitio del Sistema de Nombres de Dominio .gt y enviarlo. La información que debe proporcionar el cliente es igual a la solicitada que cuando éste se presenta en la oficina de administración.

Por medio del correo electrónico se envía un aviso de aceptación de la solicitud. Luego el cliente/usuario tiene 30 días para realizar el pago respectivo del dominio y si no lo realiza en este plazo, el dominio vuelve a quedar disponible. También el cliente puede pagar en línea si lo desea.

En ambos casos, si el cliente desea actualizar la información sobre los contactos o servidores de nombre, debe llamar a la oficina J-204 o envía un correo electrónico solicitando el cambio de la información.

VI. DESARROLLO DE LA APLICACIÓN

A. Herramientas de desarrollo

Para el desarrollo del nuevo módulo del Registro de Dominios, se utilizó Java, específicamente Servlets y JSP (Java Server Pages, en inglés), integrado con JavaBeans. Se utilizó servlets porque, según Cuilla (1998:31), estos se han convertido en la herramienta para crear aplicaciones Web dinámicas, con un alto rendimiento e independencia de plataforma. Sobre esta afirmación hay muchas opiniones al respecto, pero esta tecnología ofrece nuevas posibilidades para los desarrolladores de aplicaciones Web, las cuales vale la pena considerar cuando se desarrollan aplicaciones para la Internet.

Una de las mejores razones para utilizar Servlets es que están escritos en Java y ofrecen una buena plataforma para servicios de red porque los programadores no manejan directamente la memoria. Esto evita problemas de seguridad, sobre escritura de memoria, y los mecanismos de excepción y multi-hilos hacen que el desarrollo sea más confiable ofreciendo una rápida respuesta de los servicios de red. Debido a su arquitectura, los servlets se cargan una vez y permanecen cargados para futuras invocaciones.

La eficiencia en el uso de servlets es evidente cuando se debe procesar mucha información, pero al responder a los clientes utilizando código en HTML (Hypertext Markup Language, en inglés) se vuelve ineficiente a largo plazo porque a este código se dificulta proporcionarle mantenimiento. Por esto, se recomienda utilizar JSP. Este lenguaje, del lado del cliente, fue creado específicamente para presentar el contenido dinámicamente generado a partir de servlets.

JSP presenta los resultados de forma genérica, pero si se desea que los resultados varíen de acuerdo a los datos que se reciban, se debe utilizar JavaBeans.

> ‹‹Para aplicaciones complejas, donde se necesita presentar contenido diferente, un servlet podría manejar la petición inicial, procesar de manera parcial los datos, establecer los beans y reenviar los resultados a una de varias páginas JSP distintas, de acuerdo a las circunstancias.›› (Hall, 2001:354).

Los servlets son programas que corren dentro de servidores de aplicaciones basados en Java. Para este proyecto se utilizó como servidor de aplicaciones ***Tomcat***. Este servidor fue desarrollado por la fundación *Apache Software Foundation*, y está basado en la especificación de Java Servlets y Java Server Pages, ambas especificaciones desarrolladas por Sun.

Se utilizó Tomcat porque este servidor está basado en Java, el software puede ser bajado del sitio de *Apache* de forma gratuita y existen versiones tanto para sistemas Windows como Linux.

El manejador de base de datos relacional utilizado para esta aplicación fue MySQL. El software de este manejador puede bajarse de forma gratuita e instalarse en varias plataformas, actualmente es muy utilizado para guardar de forma permanente la información de las aplicaciones Web; es bastante robusto y estable, además Java cuenta con un controlador para esta base de datos, lo cual facilita el uso de esta base de datos en la aplicación.

El *Registro de Nombre de Dominio .gt*, utiliza actualmente PostgreSQL, que es otro manejador de base de datos relacional cuyo software es gratuito y corre en muchas otras plataformas. No se utilizó este manejador para la aplicación porque es más complejo que MySQL, al cumplir completamente con el estándar SQL92. Además, este manejador aún no ha sido muy probado para el desarrollo de aplicaciones Web.

B. Instalación del servidor de aplicaciones

La aplicación que se implementó fue colocada en una máquina cuyo sistema operativo es Linux, específicamente, el sistema operativo Fedora Core 2. Se utilizó este software porque la red interna de la UVG (Universidad del Valle de Guatemala) está basada en un sistema operativo Linux, Fedora Core 4.

Para desarrollar la aplicación, se utilizaron dos computadoras diferentes. Una en la que se desarrolló y probó toda la aplicación y la segunda, que pertenece a la UVG que fue utilizada específicamente para colocar la aplicación y en la que los administradores del Registro de Dominio pudieran ver el avance de la misma. Ambas computadoras debían tener el mismo software aunque en algunos casos varió la versión de software utilizada. Para acceder a estas máquinas se creó un usuario con contraseña, permitiendo así el ingreso al sistema para desarrollar la aplicación.

En la computadora perteneciente a la UVG no se realizó ninguna instalación de software, pues ya contaba con el software requerido, y en el único caso en que se necesitó instalar un software, el JDK (Java Development Kit, en inglés) de Sun, el administrador de este sistema lo hizo. En cambio, en la otra computadora si se instaló el software necesario para esta aplicación.

1. *Instalación del JDK.* Lo primero que se hizo fue instalar el JDK de Sun, la versión 1.5 antes de instalar el servidor de aplicaciones *Tomcat.* Esto se debe a que el servidor hace uso de bibliotecas de Java. El JDK fue instalado bajo el directorio */usr/local/jdk*.

A continuación se configuró la variable de ambiente $PATH, que indica la ruta donde se encuentran los directorios que permiten ejecutar programas al usuario. En esta variable se agregó la ruta del directorio */bin* del JDK recién instalado, el cual indica la ubicación del compilador de Java llamado *javac.*

La variable *$PATH* (en sistemas Linux), se configuró en el archivo *.bash_profile* del usuario de esta aplicación. (*Ver Apéndice A*) Al colocar la variable en este archivo, los cambios que se realicen son vigentes sólo para ese usuario. La sintaxis utilizada para agregar esta variable en el archivo indicado se hizo de la siguiente forma.

Cuadro # 1 Configuración de la variable de ambiente PATH

```
PATH=$PATH:/usr/local/jdk/jdk1.5.0_08/bin
export PATH
```

En el **Apéndice A** de este documento se encuentra completo el archivo .bash_profile.

2. *Servidor Tomcat.* A continuación, se instaló la versión más actualizada del software para Tomcat, la 5.5.15, bajo el directorio */usr/local/tomcat*. Esta versión está basada en la especificación de Java Servlets 2.4 y la especificación de Java Server Pages 2.0.

Se configuró la variable de entorno JAVA_HOME, la cual le indica al servidor donde se encuentra ubicado Java. Se debe colocar esta variable al ambiente de desarrollo de lo contrario, el servidor *Tomcat* no podrá compilar páginas JSP o servlets. Esta variable también se agregó al archivo *.bash_profile.* La sintaxis fue la siguiente:

Cuadro # 2 Configuración de la variable de ambiente JAVA_HOME

```
JAVA_HOME=/usr/local/jdk/jdk1.5.0_08
export JAVA_HOME
```

Otra variable que se configuró fue CATALINA_HOME. Esta no es necesaria para el servidor Tomcat pero se creó porque es útil para encontrar la raíz del

directorio de instalación de Tomcat rápidamente, no importando el directorio donde se encuentre el usuario en ese momento. Al igual que las variables anteriores, esta también fue colocada en el archivo *.bash_profile.* Su sintaxis es:

Cuadro # 3 Configuración de la variable de ambiente CATALINA_HOME

```
CATALINA_HOME="/sakai-demo"
export CATALINA_HOME
```

3. *Estableciendo el ambiente de desarrollo.* Se creó en el directorio */home* del usuario de desarrollo, un directorio en el cual se colocaron el código fuente de los servlets que se fueron creando.

Debido a que para esta aplicación se utilizaron *paquetes*, que es un método de agrupar clases relacionadas entre sí, se creó un subdirectorio dentro del directorio de desarrollo. El uso de paquetes, permitió mantener las clases agrupadas, lo que ayuda al mantenimiento de la aplicación. Este subdirectorio debe llamarse igual que el nombre del paquete en cuestión. Al paquete se le dio el nombre *dominio* así que dentro del directorio de desarrollo, se encuentra el subdirectorio *dominio* que contiene los servlets desarrollados para esta aplicación.

Seguidamente, se agregó a la variable CLASSPATH la ruta de dos archivos *servlet-api.jar* y *jsp-api.jar* que se encuentran en el directorio */common/lib*, bajo el directorio de instalación de Tomcat. Por medio de esta variable, se identifica la ubicación de las clases definidas por desarrolladores o terceras personas.

Se debe asignar un valor a esta variable porque los servlets y JSP no son parte de la plataforma de Java, y el compilador debe poder identificar las clases servlet para compilar sin errores. Si esto no se hiciera, cualquier código que se

quiera compilar y use clases del API de servlets y JSP dará un error, indicando que no se encuentran las clases.

A esta variable se le debe agregar también, la ruta del directorio actual que en sistemas Linux se representa con el carácter (.) también se debe agregar el directorio raíz del directorio de desarrollo. Estos dos últimos valores, se colocan cuando se usan paquetes, de lo contrario, no es necesario agregarlos.

Cuadro # 4 Configuración de la variable de ambiente CLASSPATH

```
export CLASSPATH=$CATALINA_HOME/common/lib/servlet-
api.jar:$CATALINA_HOME/common/lib/jsp-
api.jar:/home/dorellana/developer/:.:$CATALINA_HOME/common/lib/
mysql-connector-java-5.0.3-
bin.jar:$CATALINA_HOME/webapps/Domain/WEB-INF/lib/commons-
id-0.1-dev.jar
```

C. Interfaz de la aplicación

1. *Desarrollo de la interfaz.* Para el desarrollo de la interfaz, se utilizaron los siguientes lenguajes: HTML con etiquetas JSP, CSS (Cascading Style Sheet, en inglés) y JavaScript. A continuación se describe el uso de cada uno de estos lenguajes para el desarrollo de la interfaz y el diseño de la misma.

a. *HTML con JSP.* Todas las páginas creadas, fueron codificadas utilizando HTML y etiquetas JSP que son las que permiten el dinamismo a las páginas Web. Las páginas creadas tienen una extensión *jsp* .

Cuando un servidor Web envía un archivo HTML sucede lo siguiente:

> ‹‹El servidor recibe una petición de un URL; este consulta sus archivos de configuración para determinar el archivo correspondiente y envía ese archivo al navegador byte por byte. […] El servidor puede o no puede enviar el archivo basado en la dirección IP del usuario o contraseña, pero si el servidor lo envía, el

archivo será el mismo para todos los usuarios en todos los contextos. No puede responder a los datos ingresados del usuario o al entorno del servidor y no puede incluir datos de otras fuentes. En lenguaje Web común, esto es llamado HTML estático.›› (Dowling, 2001:39)

Este es el problema principal del uso de HTML para páginas Web, que utilizando sólo este lenguaje, no se puede adaptar el resultado al usuario y su entorno. Para solucionar este problema, surgen los servidores de aplicación que utilizan inicialmente CGI (Common Gateway Interface, en inglés), donde este, según Deep y Holfelder (1996:61), es un mecanismo que permite a los clientes Web ejecutar programas en un servidor Web y recibir la salida de estos.

El mayor problema que presenta la tecnología CGI, es que según Frank Hayes (1999:74), se crea un nuevo programa cada vez que un usuario envía un formulario. Esto ocasiona que si hay muchas peticiones de usuario corriendo en el servidor, se reduce el rendimiento y la escalabilidad del servidor. El lenguaje de programación mayormente asociado al desarrollo de páginas Web utilizando CGI, es Perl.

La solución a los problemas que presenta CGI, es que los archivos y programas que deben ejecutarse, corran dentro del servidor. De esta forma éste no tendrá que cargar y descargar de la memoria, cada uno de ellos por cada petición.

Tomcat, es un servidor en el cual corren todos los programas necesarios, y donde se encuentran los archivos que se necesiten para las aplicaciones. Servlets, JSP y Java Beans son las herramientas que ya se encuentran dentro del servidor y son utilizadas para generar páginas Web dinámicas.

Los servlets procesan los datos enviados por el cliente, JSP permite separar la lógica del programa, de la presentación y por medio del uso de Java Beans se provee el dinamismo a las aplicaciones Web.

‹‹Los JavaBeans son componentes de software que esconden funcionalidad compleja detrás de una simple interfaz.›› (Jepson, 2001:04)

b. CSS.

‹‹CSS provee un método alternativo para el control global para características de documentos.›› (Novak y Pete Markicwicz, 1998: 134)

Por medio de este lenguaje, se dio formato a elementos tales como el tipo, tamaño y color de texto, el color y forma de las tablas y en algunos casos, se definió la posición de algunos elementos dentro de la página que los contenía. Todas estas características fueron colocadas en un archivo llamado *info.css*, que se colocó en el mismo directorio donde fueron colocados los archivos *jsp.* Se decidió colocar todas las características antes mencionadas, en un solo archivo para poder hacer cambios en ellas sin tener que revisar y modificar cada página del sitio. El uso de este archivo dentro de la aplicación se vio como una ventaja para desarrolladores Web que quizá más adelante tengan que realizar cambios de estilo en la aplicación.

c. JavaScript.

‹‹Este lenguaje del lado del cliente implementa un subconjunto de instrucciones Java pero es más simple de programar e implementar.›› (Novak y Pete Markicwicz, 1998: 143)

Este lenguaje está enfocado al desarrollo de páginas Web del lado del cliente. Se utilizó este lenguaje para proveer dinamismo a las páginas del sitio, principalmente para colocar el cursor en el primer campo de los formularios al cargar las páginas. También se usó para mover el cursor de un campo al siguiente, dentro de los campos que forman el número de tarjeta de crédito, y de la dirección IP, que se encuentra en el formulario de Servidores de Nombres.

Uno de los usos más comunes de JavaScript en páginas Web, es validar el ingreso de datos del usuario en los formularios pero se decidió no utilizarlo para

estos chequeos prefiriendo realizar todo tipo de validaciones desde el servidor, y evitar así posibles fallos del navegador. Algunos navegadores permiten deshabilitar las funciones de Javascript, lo que ocasionaría problemas de validación de los datos si toda verificación se realiza del lado del cliente.

2. *Características de la interfaz.* Todas las páginas creadas para esta aplicación tienen el formato siguiente: un *marco* (o frame, en inglés) horizontal y dos *marcos* verticales. Un marco es una división en una página Web, en la cual se puede desplegar otro documento HTML.

Se utilizaron tres marcos porque es así como se han diseñado todas las páginas pertenecientes a la UVG. *(Ver Apéndice F. Pantalla I-IX, pp. 50-54)* En el marco horizontal, se colocó el nombre del servicio que se ofrece al público, es decir, el registro de nombres de dominio. Y en el marco vertical izquierdo, se colocaron las opciones que ofrece esta aplicación al usuario.

Fig. 1 Marco izquierdo de la aplicación

Para capturar los datos del usuario se creó un formulario para cada uno de los tipos de la información solicitada, es decir, datos de la empresa registrante, datos de contactos administrativo, técnico, de cobro y la información de dos servidores de nombres. Se realizó en pasos el ingreso de los datos por parte del

usuario, para evitarle equivocaciones y agotamiento. Pues anteriormente era evidente el agotamiento del usuario al llegar al final de la página.

Además, dividir en pequeños formularios la información permitió manipular más los datos y proporcionarle ciertas ventajas al usuario. Por ejemplo, esta aplicación cuenta con la posibilidad de "copiar", los datos del contacto administrativo al contacto técnico y al de cobro, si el usuario lo desea. *(Ver Apéndice F. Pantalla IV, V, VI, VII, pp. 51-53)* Esto se debe a que en muchas empresas, el contacto administrativo es el mismo que el contacto técnico y/o de cobro, así que se debería ingresar la misma información que se tiene para el contacto administrativo a los otros dos. En la aplicación realizada, esto se hace por medio de dos casillas de verificación colocadas en el formulario de contacto administrativo y sólo una casilla en el formulario del contacto técnico.

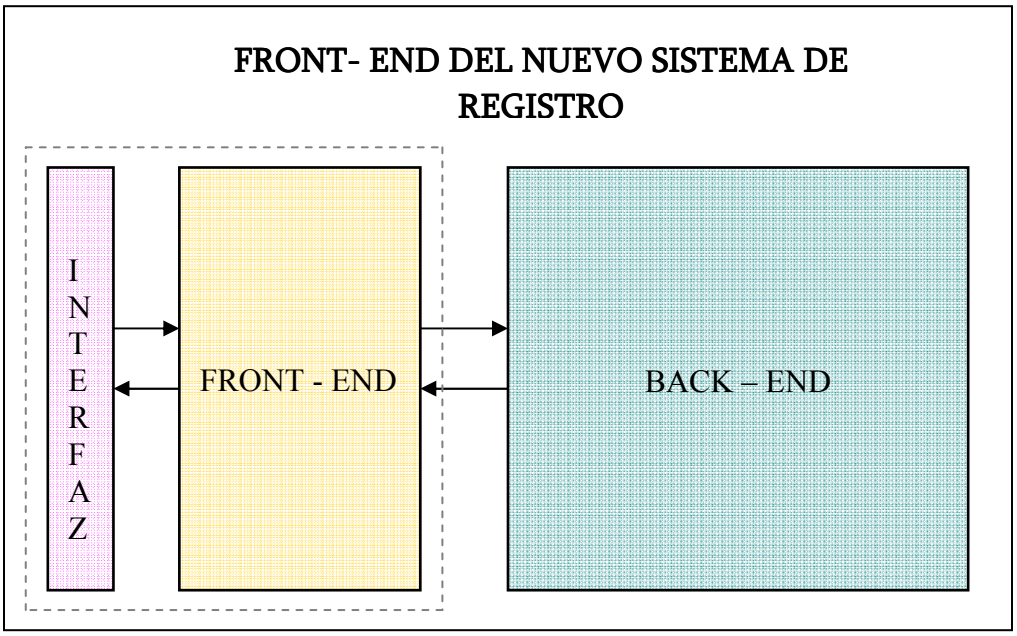
El usuario marca una o ambas casillas indicando de esta forma, que todos los datos que ingresó en el formulario deben ser iguales para los contactos indicados.

Fig. 2 Copia de datos de contacto administrativo

D. Procesamiento de la información

Este proyecto consta de la interfaz anteriormente explicada y a continuación, se describe el funcionamiento y la interacción entre todos los componentes de esta aplicación, los cuales hacen que la aplicación devuelva los resultados requeridos por el cliente. Para comprender qué elementos se desarrollaron y a cómo se unen al nuevo sistema de Registro se creó el siguiente diagrama.

Fig. 3 Front-end del nuevo sistema de Registro



Los elementos que se encuentran dentro del cuadrado con línea punteada son los que se desarrollaron para este proyecto.

1. *Estructura de directorios.* Para correr la aplicación primero se organiza la aplicación Web dentro del servidor. A partir de la especificación de Serlvets 2.2 una aplicación se define como: *una jerarquía de directorios y archivos en un diseño estándar.* Esta jerarquía puede estar "empacada" o no estarlo. La forma empacada de la aplicación es llamada WAR (Web Archive, en inglés) y es utilizada, generalmente, cuando se distribuye la aplicación, y la forma "desempacada" es utilizada para el desarrollo de la aplicación.

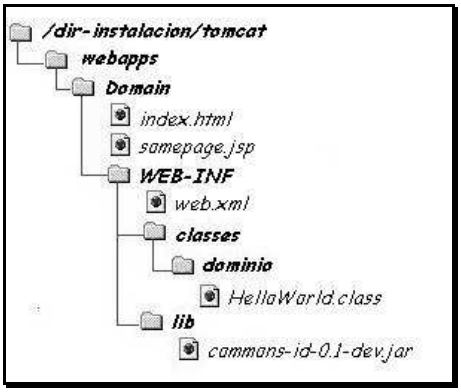
Para facilitar la creación de archivos WAR, se recomienda colocar los archivos, en una estructura similar a la utilizada por este tipo de archivo. Para esto se colocaron los directorios y archivos siguientes en el directorio */webapps*, del directorio de instalación del servidor.

Cuadro # 5 Estructura de directorios

Directorio	Contenido
Domain	Directorio raíz de la aplicación realizada para el Registro de dominios. En él se colocaron de forma jerárquica los subdirectorios requeridos y los archivos estáticos, tales como *.html y *.jsp.
Domain/WEB-INF	En éste se colocó el archivo llamado web.xml que permite configurar la aplicación. Y se crearon los siguientes directorios, *classes* y *lib*.
Domain/WEB-INF/classes/dominio	En el directorio *classes* se colocan todas las clases correspondientes a la aplicación. Pero como la aplicación fue agrupada en paquetes, se creó el directorio *dominio,* nombre del paquete, dentro de este directorio, y fue en este subdirectorio donde se colocaron las clases Java, necesarias para la aplicación.
Domain/WEB-INF/lib	En éste se colocó una biblioteca en formato *.jar, que permitió utilizar identificadores numéricos y alfanuméricos de forma secuencial y aleatoria dentro de la aplicación.

A continuación se muestra un esquema de cómo quedó la estructura de directorios que se describió anteriormente, dentro del servidor Tomcat.

Fig. 4 Estructura de la aplicación



a. *Archivo del contexto.* Este archivo es usado en el servidor Tomcat para especificar opciones de configuración. Este archivo debe tener un formato XML (eXtensible Markup Language, en inglés), y debe tener el mismo nombre que se le dio a la aplicación bajo */webapps*.

El archivo para esta aplicación se llamó *Domain.xml* y se colocó bajo */conf/Catalina/localhost* del directorio de instalación del servidor. En el **Apéndice B** de este documento, se muestra el contenido de este archivo, tal y como se usó para esta aplicación.

Entre las propiedades configuradas para esta aplicación se agregó la propiedad "reloadable" con valor *true*. En la documentación, se recomienda utilizar esta propiedad sólo en la fase de desarrollo y al finalizar esta etapa, regresar el valor a *false*, que es su valor por defecto.

La propiedad indicada anteriormente, con valor *verdadero,* permite realizar de forma automática el despliegue de las clases mientras el servidor está corriendo, sin necesidad de pararlo, desplazar las clases y reiniciarlo como es su acción por defecto.

b. *Archivo de configuración de la aplicación.* Este archivo, llamado *web.xml* contiene configuración básica para la aplicación Web. En este archivo se determina: el URL (Uniform Resource Locator, en inglés) de los servlets que componen la aplicación; algún método de autenticación, si lo hubiera; y definiciones de filtros si se utilizan.

Este archivo, como se indicó anteriormente, debe ser colocado en la estructura de la aplicación que se encuentra en */webapps* y debe seguir una sintaxis XML. En el **Apéndice C** de este documento y en la sección correspondiente se encuentra el archivo utilizado para la aplicación. A continuación se presenta un ejemplo de los elementos que se colocaron en este archivo.

Cuadro # 6 Elementos del archivo web.xml

```
<web-app>
<servlet>
      <servlet-name>Form1</servlet-name>
      <servlet-class>dominio.Form1</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>Form1</servlet-name>
      <url-pattern>/dominio.Form1</url-pattern>
</servlet-mapping>
</web-app>
```

En el elemento *servlet* se asocia una clase servlet a un nombre. Y en el elemento *servlet-mapping* se asocia una dirección dentro de la estructura de la aplicación, a un nombre de clase.

En el archivo de configuración de esta aplicación, se realizó un cambio a estos elementos, y fue que tanto para la propiedad *url-mapping* como para la propiedad *serlvet-class* se colocó también el nombre del paquete antes del nombre y ruta de la clase. Según la documentación, ésta es la forma de colocar estas propiedades cuando la aplicación se ha agrupado en paquetes.

Luego de crear toda la estructura anterior y los archivos necesarios para el funcionamiento de la aplicación, se puede iniciar el proceso de desarrollo y "desplazamiento" de las clases a la estructura de la aplicación.

Para poder ver la aplicación y probarla, se debe verificar que el servidor Tomcat esté corriendo. Sino, se producirá un error informando que ***no hay conexión al servidor***. Luego de esta verificación, se accede a la aplicación utilizando la siguiente ruta: http://localhost:8080/Domain/index.jsp. Este URL indica el nombre de la máquina anfitrión que está utilizando el puerto 8080 para comunicarse con el servidor Tomcat. Este puerto es configurable, pero para esta aplicación se utilizó la configuración por defecto. Luego se indica la raíz de la

aplicación, en este caso, *Domain* y por último se pone la ruta a partir del directorio raíz de la estructura.

2. *Servlets de Java.* Las clases e interfaz del API de Servlets, se encuentran en los paquetes *javax.servlet* y *javax.servlet.http.* Para recibir peticiones HTTP haciendo uso del método *service*, se debe extender la clase HttpServlet.

> «… Se genera una instancia de un servlet con cada petición del usuario que resulta en un nuevo subproceso que se pasa a doGet o doPost, según corresponda.» (Hall, 2001:34)

El método *init()* establece el código de configuración que se ejecuta una sola vez. Además, por cada petición de usuario se llama al método *service* y este a su vez, ejecuta *doGet* o *doPost*, de acuerdo con el tipo de la petición HTTP que se reciba.

Para lograr que un servlet manejara las peticiones GET y POST de igual forma, se implementó que el método *doPost* llamara al método *doGet* . De esta forma, no es necesario crear código específico si el tipo de peticiones varía. A continuación se muestra código fuente de cómo se realizó lo anterior.

Cuadro # 7 doPost llama a doGet

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
throws ServletException, IOException
{
        //código del servlet
}

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

La interfaz *HttpServletRequest* y *HttpServletResponse* que se muestra en el código anterior, proporciona al servlet una forma de recibir y enviar datos hacia y desde el servlet. El objeto *HttpServletRequest* proporciona al servlet los datos del formulario enviado por el cliente, estos datos pueden obtenerse utilizando métodos de esta interfaz. Y el objeto *HttpServletResponse* permite al servlet enviar los resultados al navegador.

3. *Validación de entradas del usuario.* Todo servlet de la aplicación realizada, primero captura los datos enviados por el cliente (navegador) y revisa que los campos del formulario tengan datos y que los datos ingresados son correctos dependiendo del tipo de dato de los campos.

Para controlar la existencia de campos vacíos en los formularios, se creó un método que realiza esta verificación, siendo la única excepción en todos los formularios, el campo llamado *extensión*. Este método se llamó en la aplicación *Empty* y en él se verifica que todo campo tengan algún valor, y si no lo tiene se genera un código de error con el respectivo nombre del campo donde éste se generó.

Luego, para validar el tipo de datos de los formularios se utilizó una biblioteca integrada al API de Java desde la versión 1.4.x. Esta biblioteca se llama *Regex*. Su nombre deriva de las palabras inglesas *regular expresión.*

a. *Regex.* Es una biblioteca de clases que permite la creación de expresiones regulares y manipulación de cadenas de caracteres. Una expresión regular, es una forma de representar cadenas de caracteres en un lenguaje específico. Estas expresiones permiten la identificación de un patrón, dentro de un texto o conjunto de caracteres.

Esta biblioteca fue utilizada para verificar patrones de caracteres y para dividir cadenas. Los patrones que fueron creados para esta aplicación, determinan si, por ejemplo, una cadena es un nombre de dominio válido, una

dirección de correo válida, si eran sólo números o sólo letras. A continuación se muestra uno de lo métodos utilizados para validar un nombre.

Cuadro # 8 Expresión regular para nombres propios

```
public boolean CheckName(String input)
{//Chequea nombre de personas, pais,ciudad,etc.

      String REGEX = "([\\w+ áéíóúñ\\.]+)";
            pattern = Pattern.compile(REGEX);
            matcher = pattern.matcher(input);

            boolean found = matcher.matches();
       return found;
}
```

La cadena *input* del código anterior, es el valor ingresado por el usuario; y la cadena *REGEX* es la expresión regular que se buscará en la cadena de entrada. Esta función, por medio del método *matches()* , regresa un valor booleano *true* o *false*, dependiendo de si el patrón fue encontrado o no dentro de la cadena.

Este API de expresiones regulares ya cuenta con conjuntos de caracteres predefinidos *i.e.* \d, \w, \W. La *d* indica los dígitos de 0-9; la *w*, indica las letras del alfabeto de la a-z y A-Z; y la *W*, indica que el patrón no debe contener las letras del alfabeto de la a-z y de la A-Z.

Esta biblioteca también se utilizó para dividir cadenas de caracteres. Por ejemplo, en la aplicación se presenta al usuario la opción de escoger otro nombre de dominio, si no se encuentra disponible el nombre de dominio que el usuario deseaba inicialmente. *(Ver Apéndice F. Pantalla II, p. 50)* Para hacer esto, se utilizó cadenas de petición (*query string*, en inglés), estas cadenas están formadas por parejas formadas de *nombre=valor* y separadas por el carácter *&* cuando hay más de una pareja. Estas, como se dijo anteriormente, forman patrones

nombre=valor, entonces se utilizó un método de la biblioteca *Regex*, llamado *split()*, el cual divide cadenas de caracteres al encontrar un patrón especifico.

Cuadro # 9 División de cadenas de caracteres

```
public String[] SplitQuery(String input)
{       String REGEX = "=";
        pattern = Pattern.compile(REGEX);
        String[] items = pattern.split(input);

        return items;
}
```

El código mostrado en el *Cuadro 9*, al correrlo deberá buscar en la cadena de entrada si ésta contiene el carácter = y si encuentra este patrón, el método regresa un arreglo de caracteres compuesto de todo el conjunto de caracteres que estén localizados antes y después de este carácter. Por ejemplo, se tiene la cadena *midominio=com.gt*, el método regresa un arreglo, que en el primer índice tiene el tiene los caracteres **midominio** y el índice siguiente tiene los caracteres que forman **com.gt**.

Si durante esta verificación sintáctica los datos no contienen estos patrones, se genera un código de error al cual se le agrega el nombre del campo donde éste se localizó. En la página se despliega un mensaje que indica el error y el nombre del campo donde ocurrió.

4. *Manejo de sesiones.* Una sesión puede considerarse como una serie de interacciones relacionadas entre un único cliente y el servidor dentro de un período de tiempo. Una sesión puede consistir de muchas peticiones a un único servlet o muchas peticiones a diferentes recursos en el mismo sitio Web.

> ‹‹Luego de que el servidor ha respondido la petición de los clientes, la conexión entre el cliente y el servidor se deja caer y se olvida. No existe “memoria” entre los clientes y la conexión.›› (Deep y Peter Holfelder, 1996:92).

Una de las características del protocolo HTTP es que no tiene estado. El servidor no sabe cómo asociar automáticamente una petición a una sesión en particular. Esto suele ser un problema si se quiere asociar varias peticiones a un mismo usuario. Por ejemplo, en aplicaciones Web se usa la analogía del *carrito de compras*, que no se podría llevar a cabo si varias peticiones no se asocian a un único cliente.

Para solucionar este problema, el API de Servlets tiene una interfaz llamada *HttpSession* que maneja sesiones. La implementación de sesiones se puede realizar por dos métodos: el uso de *Cookies* o por medio de la reescritura de URL. En el primer caso, el identificador de la sesión se almacena en una *cookie*; y en el segundo método, el identificador se agrega al URL de cada enlace a que "apunte" hacia el sitio donde se encuentra ubicada la página.

Usando los objetos asociados a la sesión, los servlets y las páginas JSP pueden buscar o actualizar información basándose en la petición del usuario. Una sesión existe en el servidor mientras no expire su tiempo o se invalide la sesión explícitamente.

Para almacenar información en una sesión, se utiliza el método *setAttribute()* que cuenta con dos parámetros: un nombre, y el objeto que se asocia a ese nombre. Al objeto asociado a la sesión se le dio el nombre de *dominiogt*. Para obtener la información almacenada en este objeto de sesión, se utiliza el método *getAttribute()* que sólo tiene un parámetro: el nombre del objeto asociado a la sesión.

Cuando el usuario termina el registro de un dominio o la actualización se invalida la sesión utilizando el método *invalidate()* de *HttpSession.* El código utilizado para obtener el objeto asociado a una sesión, se muestra a continuación.

Cuadro # 10 setAttribute y getAttribute

```
HttpSession sesion = request.getSession(true);
Handler handler =
     (Handler) session.getAttribute("dominiogt");

if(handler == null){ //aun no está en una sesión
    handler = new Handler();
    sesion.setAttribute("dominiogt",handler);
}
```

5. Uso de JavaBeans

> ‹‹JavaBean es un componente reusable de software que puede ser manipulado visualmente por una herramienta. Los desarrolladores querían que el bean fuera un proceso simple. Un bean se describe por medio de sus características. Una característica es una propiedad, un método o evento.›› (Daconta, Al Saganich y Erick Monk, 1999: 500).

Para que una clase JavaBean funcione adecuadamente, un objeto de esta clase debe seguir ciertas normas. Una clase Bean, debe contar con un constructor vacío; no deberá tener variables de instancia públicas; y se deberá acceder a los valores a través de los métodos getXxx y setXxx (estos métodos se explicará más adelante).

Para cargar un Bean en una página JSP se utiliza la acción *jsp:useBean*. Esta acción cuenta con opciones adicionales tales como la propiedad *scope* que hace que el Bean se asocie a otras páginas además de la que está en uso. En la *Tabla 11*, se presenta el código que se encuentra en toda página JSP creada para esta aplicación de Registro de dominio, que utiliza Beans para obtener información colocada en ellos por el servlet.

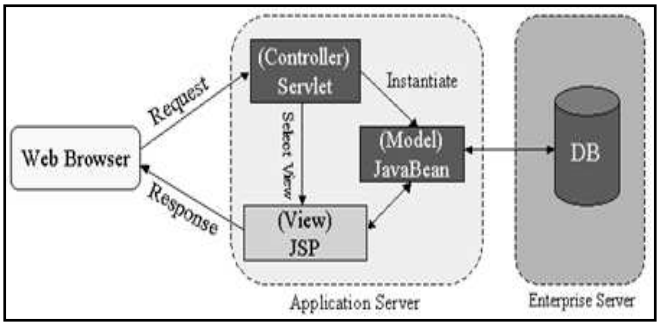
Cuadro # 11 Carga de Beans y propiedad *scope*

```
<jsp:useBean id="dominiogt" class="dominio.Handler" scope="session"/>
```

Este código es equivalente a decir: "cree una instancia del objeto de la clase Handler, que se encuentra en el paquete *dominio*; y asigne este objeto a la variable especificada con el nombre *dominiogt*". El valor que se muestra en la propiedad *scope,* significa que el objeto Bean se almacenó en un objeto *HttpSession*, asociado a la petición en curso, y que su valor puede ser recuperado por medio de *getAttribute()*.

La clase *Handler* que aparece en el código de el *Cuadro 11* es el objeto asociado a la sesión por medio del método *getAttribute()* de la interfaz *HttpSession*, y a su vez, es el Bean que se creó para esta aplicación. Por medio de este Bean, las páginas JSP obtienen la información colocada en él por los servlets.

Fig. 5 Arquitectura II de Java Server Pages



La aplicación del Registro de Dominio, desarrollada sigue el modelo que se representa en la figura anterior. Este modelo es llamado *Arquitectura II de Java Server Pages* o *MVC* (Model-View-Controller, en inglés). La secuencia de este modelo es así:

- El servlet, que es el *controlador*, recibe las peticiones del navegador y las procesa, enviando el control a otro servlet o página JSP, recuperando información de una base datos, etc.
- Luego coloca la repuesta solicitada en un Bean, que es el *modelo*, es decir, contiene las reglas de negocios.

- Y finalmente, las páginas JSP, que son la *vista*, presenta la información solicitada al usuario con cierto estilo.

Utilizar este modelo es recomendable puesto que provee una forma más eficiente de mantener la aplicación, pues divide la parte lógica de la parte de presentación de la información. El desarrollador Web, sólo se debe preocupar de la presentación de la información, y el desarrollador de código de Java se preocupa sólo de la parte lógica de la misma y su relación con las reglas de negocios.

Los JavaBeans son los componentes que proveen dinamismo a las páginas JSP. La siguiente sintaxis muestra cómo una página JSP carga un Bean, con el objetivo de obtener los atributos o propiedades de los Beans, cuyos valores fueron asignados por los servlets al procesar la información enviada por el usuario.

Cuadro # 12 Acceso a propiedades de los Beans

```
<jsp:getProperty name="dominiogt" property="nombreDominio"/>
```

Para acceder a las propiedades de un Bean se utiliza *jsp:getProperty*, que toma un nombre que debe ser igual al identificador asignado por medio de *jsp:useBean*. Y el atributo *property* indica el nombre de la propiedad del Bean que se desea acceder.

Anteriormente se indicó que la forma de acceder a los métodos de un Bean era por medio de los métodos *getXxx* y para agregar valores el método *setXxx.* En una clase JavaBean, los métodos a desarrollar para establecer los valores a las propiedades de los beans deben ser, por ejemplo, setContactAdminTable(); y getContactAdminTable(), para obtener los valores del bean. El método debe iniciar con mayúscula después de la palabra *set* o *get*. Pero cuando se llama un

método desde una página JSP, en el atributo *property*, el nombre debe iniciar con minúscula. Esta es la forma como funcionan los Beans, si no se sigue esta sintaxis, se genera un error en tiempo de corrida.

Las etiquetas para carga y acceso de propiedades de Beans, que se colocaron en las páginas JSP de esta aplicación, sólo acceden a la información que se encuentra en los Beans. En ningún momento se cambiaron los valores de las propiedades por medio del método *jsp:setProperty*. No es necesario que desde las páginas JSP se modifiquen las propiedades del Bean.

6. *Generación de identificadores únicos y contraseñas.* Una de las características de la aplicación realizada es la generación de identificadores únicos para los contactos, la empresa y servidores de nombre. *(Ver Apéndice F. Pantalla III, p. 51)* El código generado más importante es el de empresa, porque por medio de éste identificador se puede ingresar al sistema de Registros y crear o modificar los nombres de dominio pertenecientes a dicha empresa.

También esta aplicación crea una contraseña que se asocia al identificador de empresa que se indicó anteriormente. Para crear el identificador para cada uno de los objetos de este Registro, se utilizó una biblioteca que permite crear identificadores numéricos secuenciales y aleatorios; alfanuméricos secuenciales y aleatorios e identificadores únicos por sesión, etc.

Esta biblioteca se llama *commons-id*, fue desarrollada por la fundación *Apache Software Foundation* y es parte del proyecto *commons* de esa fundación.

a. *Commons-id.* En el sitio de *Apache* (www.apache.org) y en el área *Commons* se encontró esta aplicación, pero sólo en código fuente. Se investigó cómo utilizar este código fuente dentro de esta aplicación, y se determinó que para crear un archivo que "empacara" todas las clases que componen esta

aplicación era necesario utilizar la herramienta *Ant*, que permite compilar clases de Java.

Antes de realizar la instalación de esta herramienta, se decidió buscar en Internet si ya estaba esta aplicación en algún sitio, lista para ser usada. Se encontró un sitio, donde el código fuente ya estaba empacado en un archivo *jar.*

Según Daconta *et al.* (1999:50) dicen:

> ‹‹Los archivos jar son usados para empacar y distribuir JavaBeans. Un archivo jar puede contener cualquier número de clases Java, gráficos y cualquier otro archivo.››

Para utilizar este archivo y sus clases dentro de una aplicación, la documentación indica que este archivo debe ser colocado en el directorio */lib* de la aplicación, y luego se debe agregar la ruta de este archivo en la variable CLASSPATH.

Así fue como se instaló esta aplicación y luego se procedió a usarla. Esta clase fue escrita utilizando el lenguaje Java motivo por el que no hubo ningún problema al integrar este generador de identificadores a la aplicación del Registro. Las clases de este generador fueron fáciles de utilizar.

b. *Algoritmos de cifrado.* Como se dijo anteriormente, con el generador de identificadores se creó una contraseña única para cada usuario permitiendo así que el cliente pueda ingresar nuevamente al sistema de Registro y actualizar los datos ya ingresados o crear más dominios. Esta contraseña se tenía que cifrar por seguridad.

Para realizar el cifrado, se usó el algoritmo SHA-1 de 160 bits, que se encuentra dentro del paquete JCA (Java Cryptography Architecture, en inglés) de

Java, el cual fue diseñado para permitir a los desarrolladores incorporar funciones de seguridad en los programas.

Se utilizó este algoritmo porque tiene la característica de crear una función de dispersión de una sola vía, y además este mecanismo garantiza que para dos valores diferentes, la función de dispersión es diferente. Otra ventaja del uso de este algoritmo es que usa 160 bits, que lo hace más seguro que MD5 que uso 128 bits. Este otro algoritmo se consideró como posible candidato para ser usado en esta aplicación, pero por lo anterior, se decidió no utilizarlo.

7. *Tarjeta de crédito.* La aplicación realizada para el Registro de Dominios de la UVG, permite hacer el pago *en línea*. Para ello, el usuario debe llenar el formulario específico. Una característica que se incorporó a esta aplicación, fue que el formulario no muestra todo el número de la tarjeta de crédito del usuario mientras éste está ingresando la información. *(Ver Apéndice F. Pantalla VII, p. 51)* Los cuatro primeros dígitos del número de tarjeta y los cuatro últimos se muestran al usuario, pero los ocho dígitos del medio no se muestran, sino que sólo se pueden ver asteriscos en lugar de los dígitos.

Se decidió hacer esto para proveer mayor seguridad al usuario si se encuentra en un lugar público y alguien puede ver lo que escribe. De esta forma, las personas que se encuentren a la par de él no podrán saber toda la secuencia de números de su tarjeta. Incluso ante algún fallo en la conexión, no se podrían leer los dígitos ingresados porque los campos que forman la tarjeta de crédito, fueron realizados en código HTML y tienen la propiedad de esconder la información por medio de asteriscos.

Luego que se realiza el pago del dominio *en línea*, en la base de datos se modifica un campo de fecha de pago y de fecha de expiración.

El campo que indica la *fecha de expiración*, que se encuentra en la tabla *orden* de la base de datos, se llama *fechaexp*. Para calcular el valor de este

campo, se agregan 30 días a la fecha del día corriente. Se creó este campo porque dentro del procedimiento del Registro de Dominios de la UVG, se indica que, si no se paga en 30 días, el nombre de dominio vuelve a estar disponible para cualquier otra entidad. Por esta razón, la fecha que se indique en el campo *fechaexp* será el último día en que el usuario puede pagar el nombre de dominio.

Ahora bien, si se realiza el pago *en línea*, el valor del campo de fecha de expiración es nulo (NULL, en inglés) y el campo *fecha* de la tabla *orden,* tiene el valor de la fecha en que se realizó el pago del dominio.

E. Base de datos

Se utilizó la base de datos MySQL y el controlador para esta base de datos de Java, el J/Connector.

1. *MySQL.* Se instaló MySQL versión 5.0.24, la licencia GPL (General Public License, en inglés) de este software. Este software se bajó del sitio *A:B MySQL* (http://www.mysql.org/). Como se dijo anteriormente, se decidió utilizar este manejador de base de datos, porque es bastante utilizado en aplicaciones Web, es muy estable y multi-plataforma.

Para la base de datos de esta aplicación, se utilizó el mismo diseño que tiene la base de datos de la aplicación actual. Sólo se le agregaron algunos campos, como el código y contraseña de la empresa y los códigos de contactos y servidores de nombre.

Se decidió no cambiar el diseño de la base de datos porque los campos que se debían agregar a la base de datos no modificaban significativamente el diseño y además para los administradores del sistema de Registro ya era un diseño conocido y con el cual se había venido trabajando desde hace tiempo. Las tablas

que se crearon para esta base de datos se encuentran en el **Apéndice D** de este documento.

Se decidió no utilizar el manejador *PostgreSQL*, también porque al revisar el software ya instalado en la computadora de la UVG, este software ya estaba instalado. Así que lo que se hizo fue solicitar al Coordinador de esta computadora que creara una base de datos con un usuario y contraseña respectivos.

Se ha investigado como se hará para integrar *PostgreSQL* a esta nueva aplicación cuando ya se instale definitivamente en algún servidor y comience a utilizarse para registrar los nombres de dominio. Se concluyó que esto es posible y relativamente fácil de integrar a la aplicación. Java ha desarrollado también un controlador para este manejador de base de datos y se debe instalar de acuerdo a la documentación que se encuentra en el sitio de Tomcat o de PostgreSQL. No habría ningún problema en la aplicación ya que se estaría utilizando el mismo API, lo único que varía es la base de datos.

También se investigaron las características de velocidad, estabilidad y seguridad de ambas bases de datos, y en cuanto a velocidad MySQL es más rápido debido sobre todo a la facilidad en el manejo de las conexiones y en la forma de realizar SELECT's. En estabilidad, JDBC para PostgreSQL no es muy eficiente pues necesita un manejo especial de excepciones y algunos códigos especiales para manejar el almacenamiento de valores NULL; y en cuanto a seguridad ambas bases de datos son similares, pero MySQL puede manejar más detalles.

2. *JDBC.* El controlador utilizado en esta aplicación, como se mencionó anteriormente, se llama *J/Connector* y se utilizó la versión 3.0. Se procedió a colocar este software en el directorio de desarrollo, se abrió y se colocó el archivo *mysql-connector-java-5.0.3-bin.jar* en el directorio */common/lib* del directorio

de instalación del servidor. Luego se colocó en la variable CLASSPATH, la ruta de este archivo.

Por último, se configuró el archivo de llamado Domain.xml , el cual se había mencionado que se encuentra bajo el directorio */conf/Catalina/localhost.* Básicamente, en este archivo se configuró la propiedad llamada *Resource*, la cual indica al servidor los recursos a los que las aplicaciones pueden acceder. En el **Apéndice B** de este documento se encuentra este archivo y en él se puede ver los parámetros utilizados para realizar esta configuración.

VII. CONCLUSIONES Y RECOMENDACIONES

A. Conclusiones

- La aplicación realizada actualiza y complementa el sistema de *Registro de Dominios .gt* actual mediante la creación de un dominio utilizando un código y contraseña que permite ingresar al sistema y además el cambio de la contraseña del mismo cuando al cliente le convenga. De esta forma, se ha obtenido una aplicación que complementa a la aplicación actual, pues esta sólo permite el registro del nombre de dominio, el pago en línea y la consulta de dominios a través de WHOIS.

- Se actualiza el sistema de Registro de Dominio .gt mediante el uso de un nuevo proceso de desarrollo de páginas Web que utiliza páginas dinámicas haciendo uso de la tecnología que ofrece Java.

- La nueva aplicación continúa haciendo uso de manejadores de bases de datos relacionales que permiten almacenar de forma permanente la información proporcionada por el cliente.

- Una característica de esta aplicación, es un campo de *tipo fecha* en la base de datos, el cual tiene un valor de 30 días a partir de la fecha de inscripción del dominio. Este campo se genera de forma automática. Los administradores de dominio .gt, al verificar este campo, pueden determinar qué nombre de dominio debe ponerse a disposición de otros clientes si no se realizó el pago en la fecha establecida.

- Esta aplicación proporciona al cliente/usuario del *Registro de Dominios .gt*, una mayor facilidad al llenar los formularios. Permite "copiar" los datos del contacto administrativo hacia el contacto técnico y/o de cobro, reduciendo así el

tiempo que al usuario le toma ingresar la información en los formularios, si los datos de cualquiera de los contactos, son iguales.

- Para que el usuario pueda llevar a cabo la actualización de la información del nombre de dominio sin la intervención de los administradores del dominio .gt, esta nueva aplicación, permite identificar de forma única a cada cliente, utilizando un código y una contraseña, las cuales se generan dentro del nuevo sistema.

- En esta nueva aplicación, la división de la información requerida en pequeño formularios, facilita al cliente el uso del Registro de Dominios .gt, que a diferencia de la aplicación actual, produce menos agotamiento al ingresar la información por partes.

- La tecnología Java, proveyó una forma fácil de manejar ciertas características que siempre en general, debería manejar una página Web. Por ejemplo, las sesiones y el manejo de peticiones GET y peticiones POST en un mismo servlet. Si el API de los Servlets no tuviera una forma de manejar estas situaciones, el desarrollador debería de invertir más tiempo para lograr esto.

B. Recomendaciones

- Esta aplicación tiene mucha capacidad para agregarle nuevos módulos. Estos módulos permitirían que el sitio del sistema de Registro de Dominios provea al usuario de nuevas opciones, todas relacionadas con las acciones que debe realizar un Registro, y que actualmente ya se encuentran disponibles en algunos otros sitios de Registro de Dominios. Por ejemplo, el borrado de los datos y la transferencia de un nombre de dominio a otro registrador.

- Para las personas interesadas en continuar con el desarrollo de esta aplicación, se recomienda utilizar editores de lenguajes de desarrollo de páginas

Web, pues de esta forma se utilizaría menos tiempo en esta etapa de desarrollo y de esta forma el desarrollador se enfoca más en la parte de Servlets y JavaBeans.

- Se recomienda más adelante, buscar un algoritmo de cifrado diferente del que se usó en esta aplicación. Esto con el fin, de proveer una mayor seguridad al usuario al cifrar su contraseña. El algoritmo utilizado en esta aplicación, es sólo de 160 bits, sin embargo, hay algunas otras bibliotecas que permiten el uso de otros algoritmos con un cifrado de más bits.

- Se recomienda que se realicen periódicamente cambios al color y tipo de letra a las páginas Web que componen esta aplicación con el objeto de ofrecer al usuario páginas estéticamente atractivas.

- Se recomienda evaluar el beneficio que representa para el cliente y para la administración del dominio el agregar más de dos servidores de nombres con el objeto de asegurar que siempre estará disponible en Internet, el sitio del dominio de cada cliente.

VIII. BIBLIOGRAFÍA

Cuilla, Chris. 1998. *Improved Development of Web Apps*. Revista Computing Canada. Vol. 24, Fascículo 39.

Daconta, Michael C. Al Saganich y Eric Monk. 1999. *Java 2 and JavaScript for C and C++ Programmers.* Estados Unidos. John Wiley & Sons, Inc. Pp. 868.

Deep, John y Peter Holfelder. 1996. *Developing CGI Applications with Perl.* Estados Unidos. John Wiley & Sons, Inc. pp. 299.

Dowling, Thomas. 2001. *Static Free Web Sites.* Revista netconnect. 39-41 pp.

Hall, Marty. 2001. *Servlets y Java Server Pages. Guía Práctica.* México. Pearson Educación. 608.

Hayes, Frank. 1999. *Common Gateway Interface.* Revista Computerworld. Vol. 33. Fascículo 29.

Jepson, Brian. 2001. *INTERNET Solutions for Web Designers and Builders.* Revista PC Magazine. IP04.

Novak, Jeannie y Pete Markicwicz. 1998. *Guide to maintaining and updating dynamic Web Sites.* Estados Unidos. John Wiley & Sons, Inc. pp 365.

A. Referencias de Internet

Hall, Marty. *A tutorial on Installing and Using Jakarta Tomcat 5.5 for Servlet and JSP Development.* Jakarta Apache. [En línea]. Disponible en Internet: < http://www.coreservlets.com/Apache-Tomcat-Tutorial/>.

Mahmoud, Qusay H. 2003. *Servlet and JSP Pages Best Practices.* Marzo, 2003. Sun Developer Network site. [En línea, citado el 5 de octubre 2006; 9:00 pm]. Disponible en Internet: <http://java.sun.com/developer/ technicalArticles/ javaserverpages/servlets_jsp/>

Postel, J. 1994. *Estructura y delegación de Nombres de Dominio. Rfc1591.* Marzo, 1994. [En línea, citado el 11 de octubre 2006]. Disponible en Internet: < http://www.rfc-es.org/rfc/rfc1591-es.txt>.

Reilly, David. 2006. *State and session tracking with Java Servlets Part 2: Securing Data.* Developer.com Gamelan. [En línea]. Disponible en Internet: < http://www.developer.com/ java/ent/article.php/616831>.

Shvarts, James. 2003. *Password encryption: rationale and Java example.* [En línea]. Disponible en Internet : <http://evolt.org/node/60122#main-page-content>

Telefónica. *Tutorial de nombres de dominios en Internet.* Versión 0.4. [En línea]. Disponible en Internet: <http://www.telefonicaonline.com/qx/ manual/Tutorial_dominios.pdf>

The Apache Software Foundation http://www.apache.org. 2006. *The Apache Tomcat 5.5 Servle/JSP Container. Documentation.* [En línea]. Disponible en Internet: < http://tomcat.apache.org/tomcat-5.5-doc/index.html>.

The Java Tutorials. *Java Beans Concepts.* [En línea]. Disponible en Internet: <http://java.sun.com/docs/books/tutorial/javabeans/whatis/beanDefinition.html>.

Web Technologies. 2006. *Introduction to Sessions.* [En línea]. Disponible en Internet: <http://support.sas.com/search/index.html>.

IX. APÉNDICE

A. Archivo de configuración del sistema *.bash_profile*

```
PATH=$PATH:/usr/local/jdk/jdk1.5.0_08/bin
export PATH

CATALINA_HOME="/sakai-demo"
export CATALINA_HOME

JAVA_HOME=/usr/local/jdk/jdk1.5.0_08
export JAVA_HOME

export CLASSPATH=$CATALINA_HOME/common/lib/servlet-
api.jar:$CATALINA_HOME/common/lib/jsp-
api.jar:/home/dorellana/developer/:.:$CATALINA_HOME/common/lib/my
sql-connector-java-5.0.3-bin.jar:$CATALINA_HOME/webapps/Domain/WEB-
INF/lib/commons-id-0.1-dev.jar
```

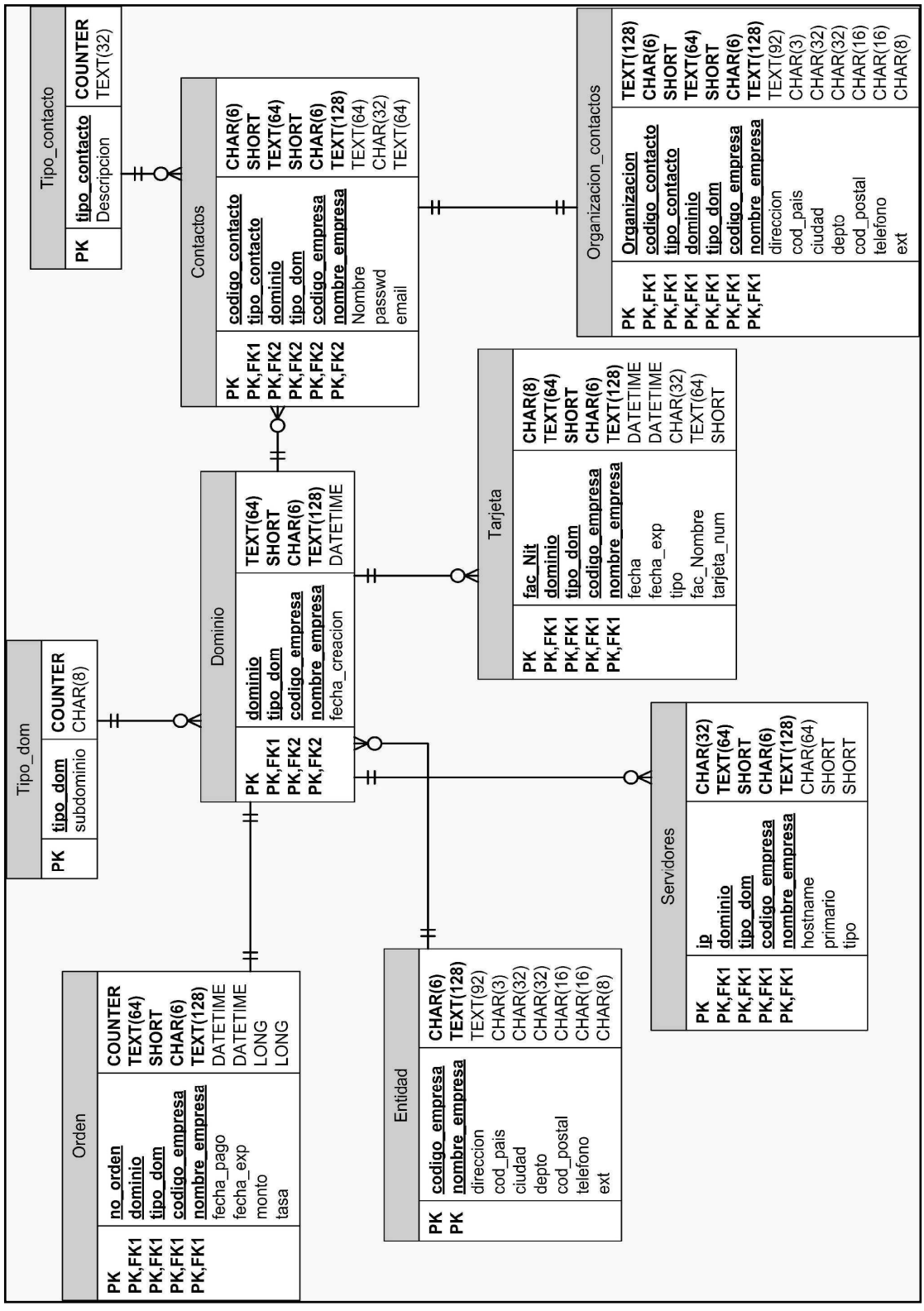
B. Archivo de descripción del contexto Domain.xml

```
<!-- Domain Context -->
<Context path="/Domain" docBase="Domain" debug="0" reloadable="true"
crossContext="true" >


 <Resource name="jdbc/MySQLDB" auth="Container" type="javax.sql.DataSource"
          maxActive="10" maxIdle="15" maxWait="10000"
          username="dorellana" password="orellana"
          driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/dorellana?autoReconnect=true"/>

</Context>
```

C. Parte del archivo de despliegue *web.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">

<web-app>

<servlet>
      <servlet-name>Init</servlet-name>
      <servlet-class>dominio.Init</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>Init</servlet-name>
      <url-pattern>/dominio.Init</url-pattern>
</servlet-mapping>


<servlet>
      <servlet-name>Form1</servlet-name>
      <servlet-class>dominio.Form1</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>Form1</servlet-name>
      <url-pattern>/dominio.Form1</url-pattern>
</servlet-mapping>

<servlet>
      <servlet-name>Form2</servlet-name>
      <servlet-class>dominio.Form2</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>Form2</servlet-name>
      <url-pattern>/dominio.Form2</url-pattern>
</servlet-mapping>

<servlet>
      <servlet-name>Form2a</servlet-name>
      <servlet-class>dominio.Form2a</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>Form2a</servlet-name>
      <url-pattern>/dominio.Form2a</url-pattern>
</servlet-mapping>

</webapp>
```

D. Tablas de la base de datos para esta aplicación



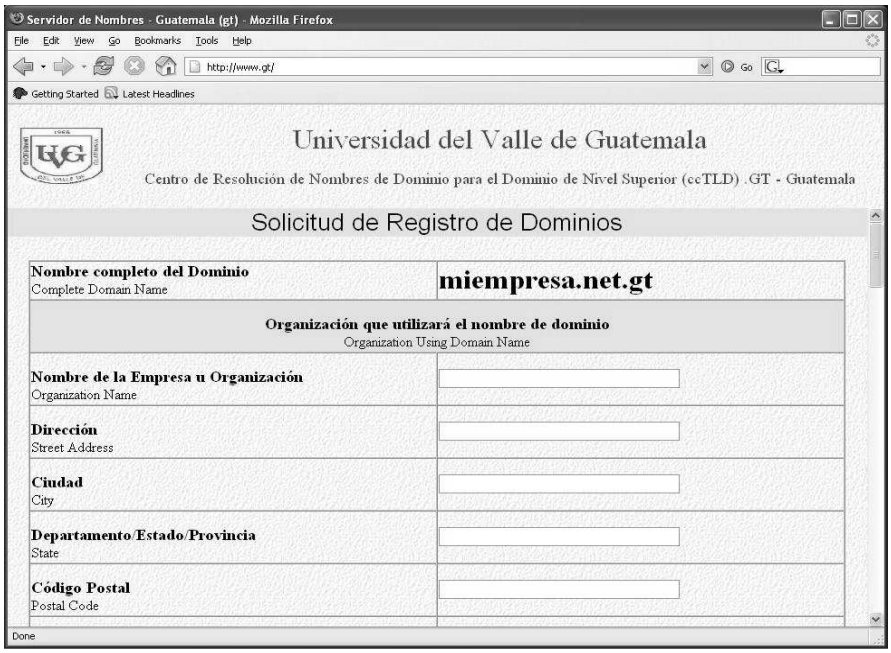
E. Pantallas de la aplicación utilizada actualmente

Pantalla I. *Pantalla inicial del sitio www.gt*

Pantalla II. *Pantalla inicial del Registro de Dominios*

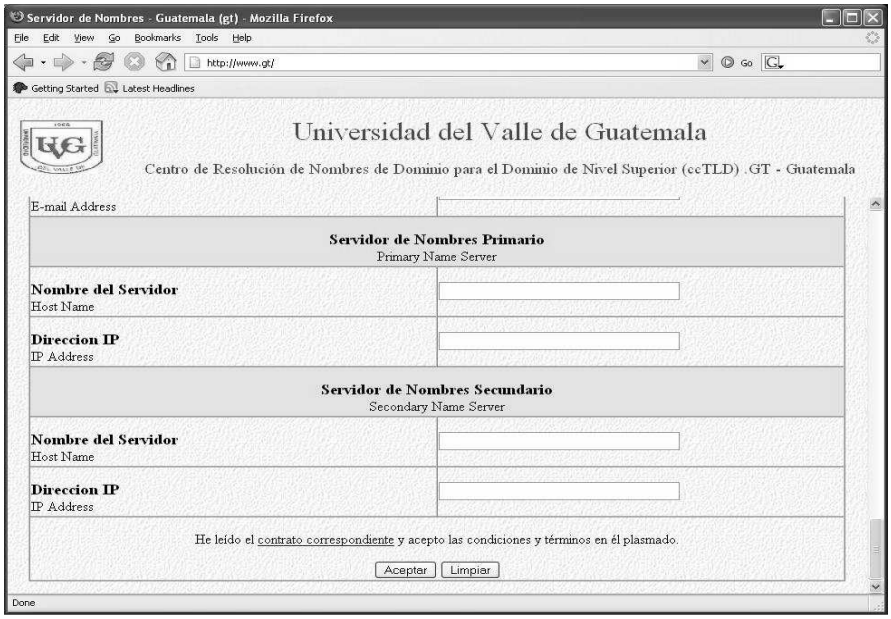
Pantalla III. *Listado de todos los subdominios disponibles que se ofrecen al cliente y el dominio que se ha reservado*

Pantalla IV. *Formulario de Solicitud de Registro de dominios (1) donde se muestra los campos a llenar con datos de la empresa*

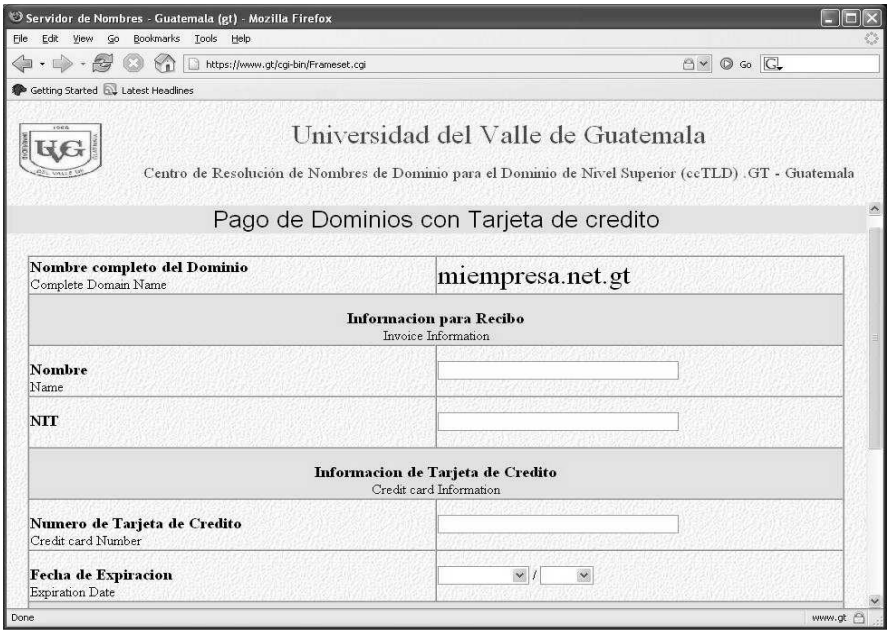


Pantalla V. *Formulario de Solicitud de Registro de dominios (2) donde se muestran los campos a llenar con datos del contacto administrativo*

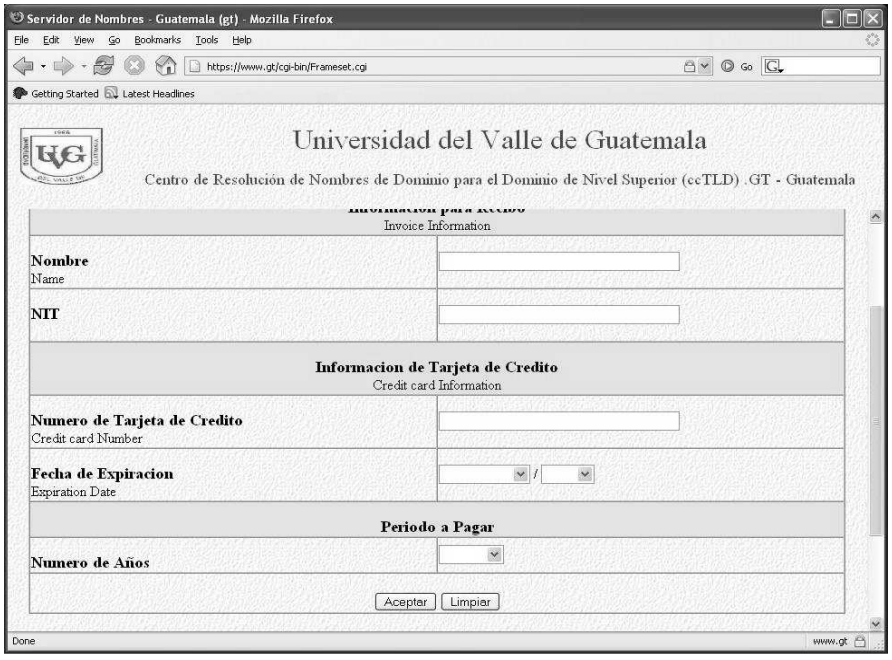
Pantalla VI. *Formulario de Solicitud de Registro de dominios (3) donde se muestran los campos a llenar con datos de dos servidores de nombres*



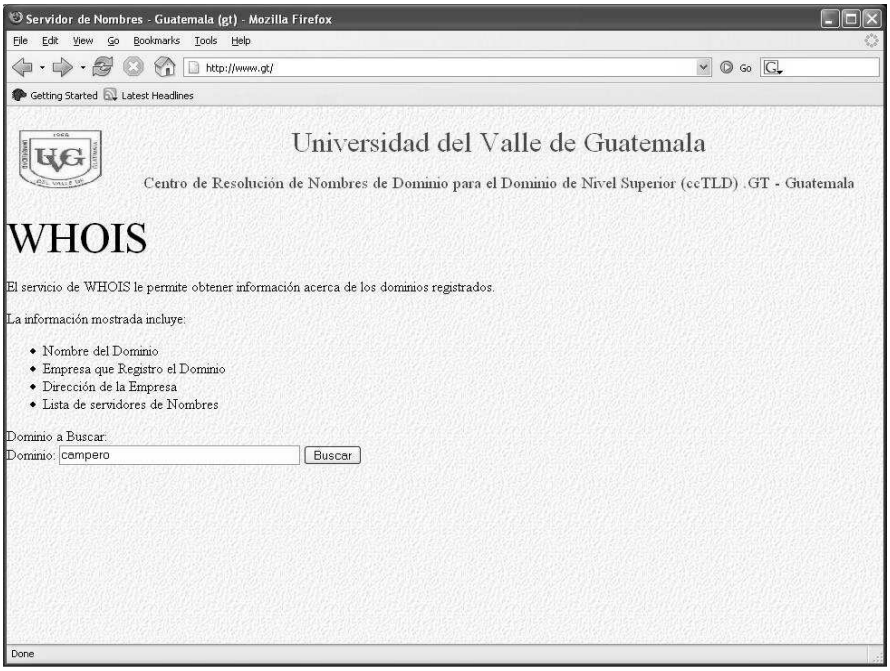
Pantalla VII. *Formulario de pago en línea (1)*



Pantalla VII. *Formulario de pago en línea (2)*

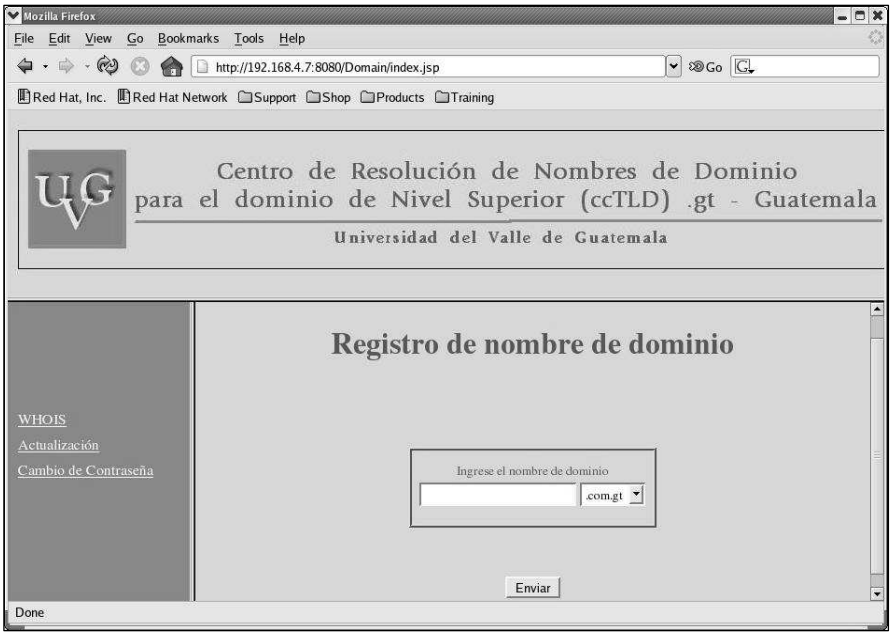


Pantalla IX. *Formulario de consulta utilizando WHOIS*

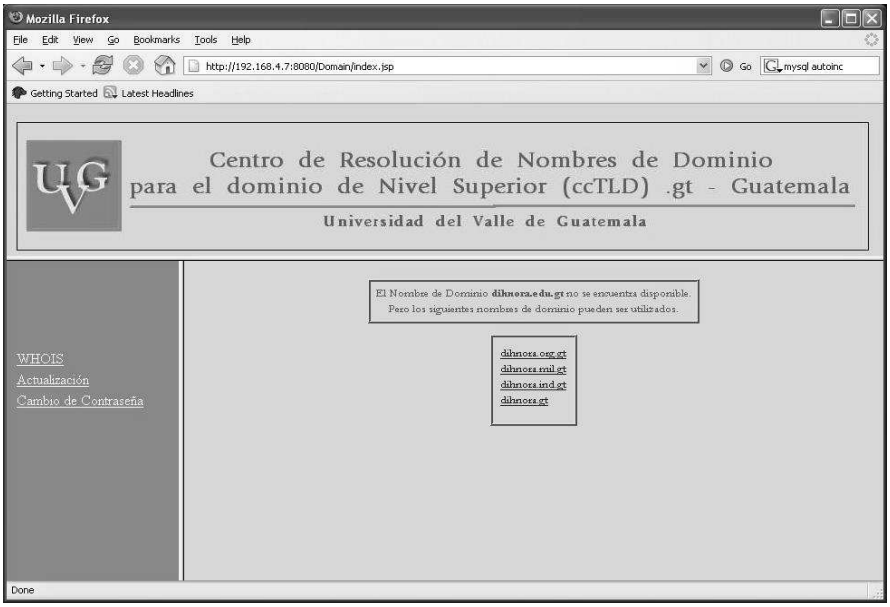


F. Pantallas de la nueva aplicación

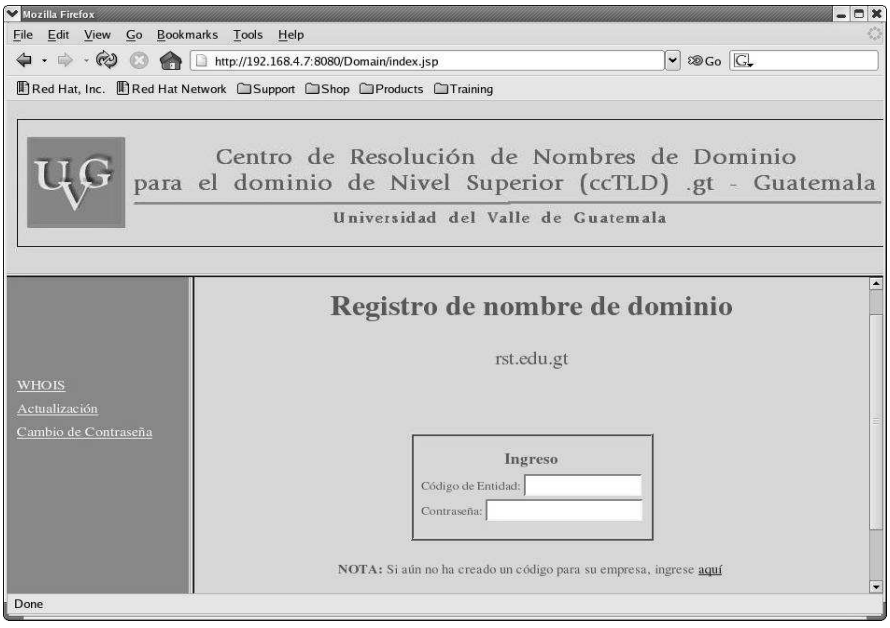
Pantalla I. *Pantalla inicial del Registro de Dominios*



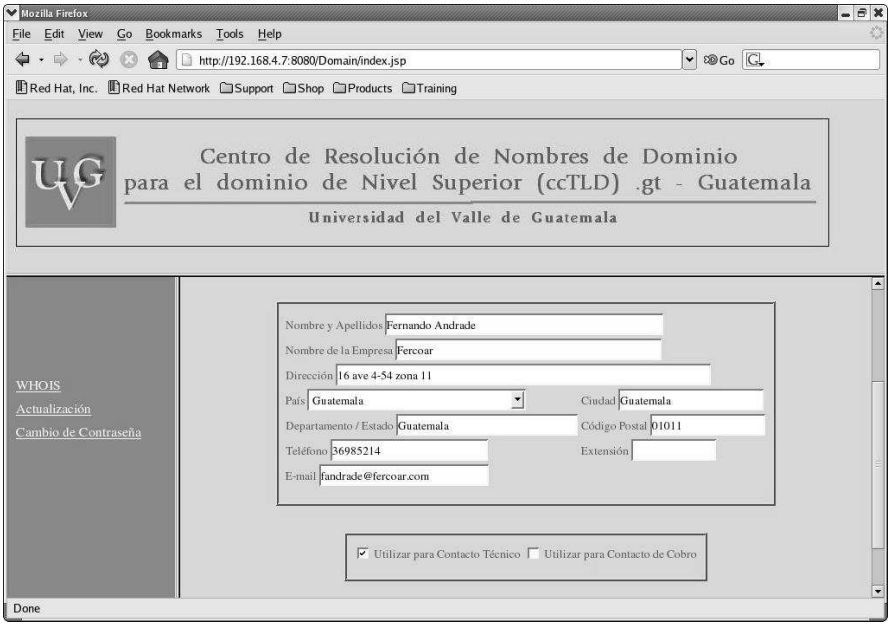
Pantalla II. *Listado de todos los subdominios disponibles que se ofrecen al cliente*



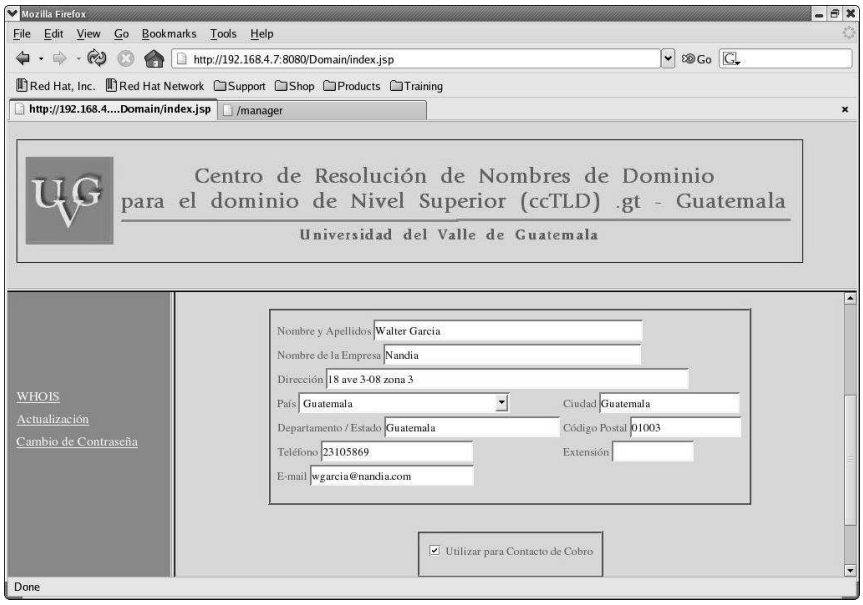
Pantalla III. *Pantalla de ingreso si ya se tiene contraseña y código de acceso. Si no, se le indica lo que debe hacer. Ambas permiten registrar dominios.*



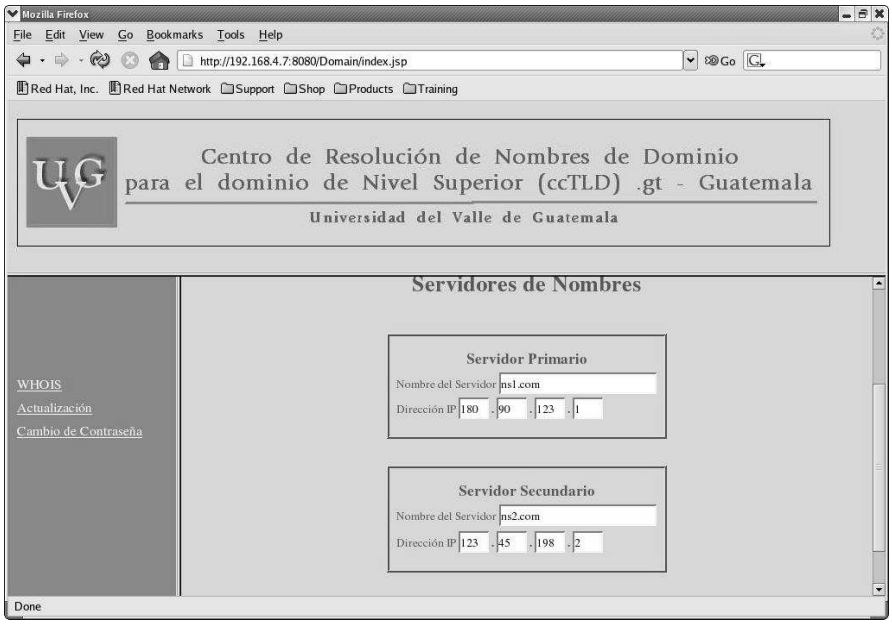
Pantalla IV. *Formulario de contacto administrativo. Permite "copiar" hacia contacto técnico o de cobro*



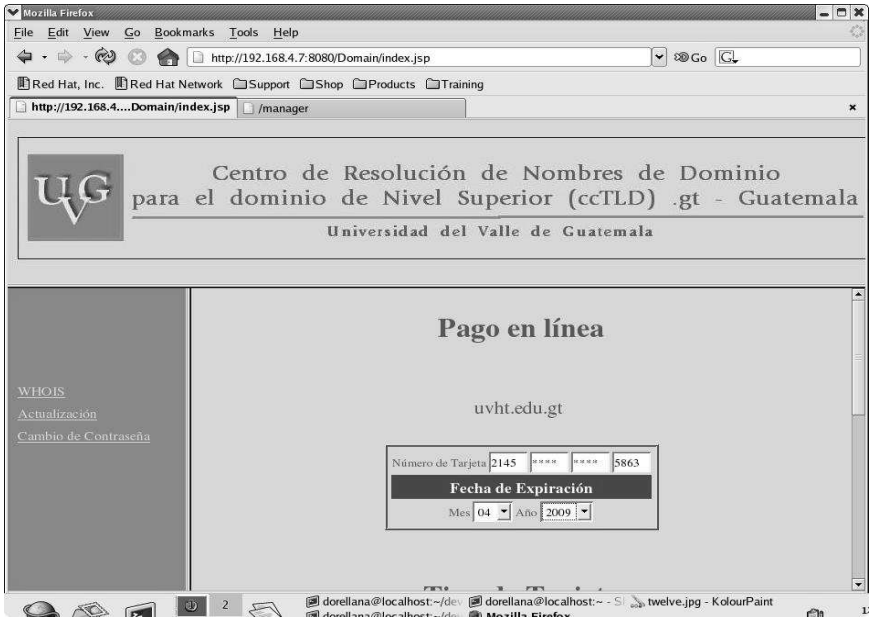
Pantalla V. *Formulario de contacto técnico. Permite "copiar" hacia contacto de cobro.*



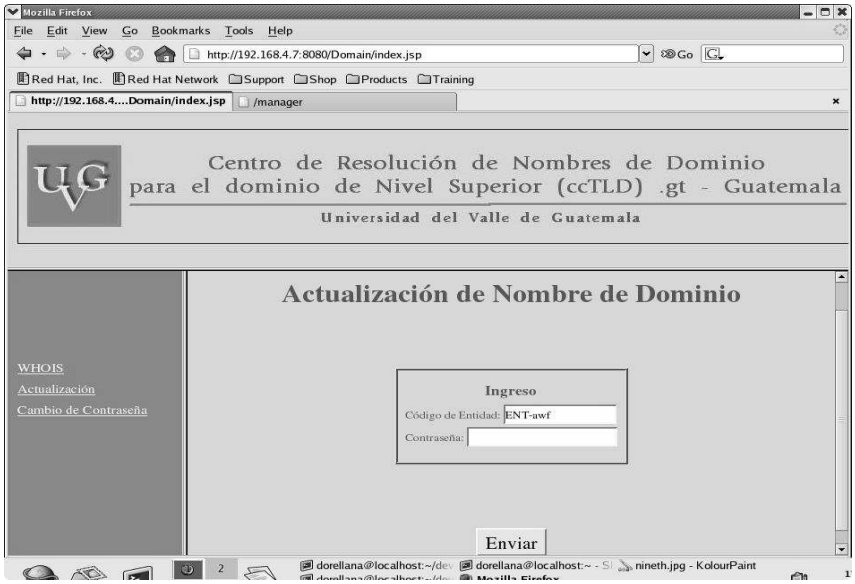
Pantalla VI. *Formulario de Servidores de Nombres*



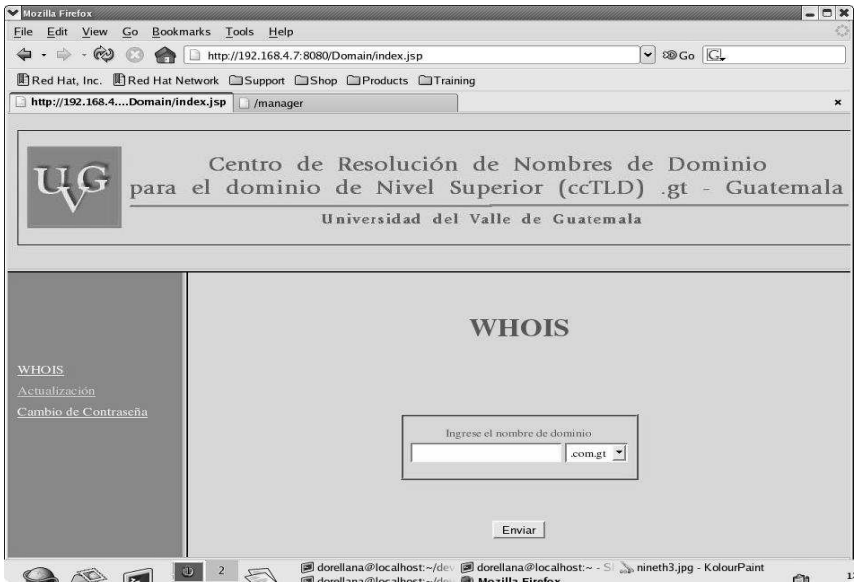
Pantalla VII. *Fragmento de formulario de pago en línea*



Pantalla VIII. *Pantalla para el ingreso al sistema y actualización de dominios*



Pantalla IX. *Pantalla para consulta utilizando WHOIS*



X. GLOSARIO

- **ANT** Es una herramienta utilizada para la compilación y creación de programas Java. Es considerado un *make* para Java.

- **API** *Application Programming Interface* Proporciona un conjunto de funciones o métodos de uso general. Provee un método para abstraer componentes de software.

- **CSS** *Cascading Style Sheet* Es un lenguaje formal, utilizado para definir la presentación de un documento escrito en HTML o XML.

- **ccTLD** *Country Code Top Level Domains* Es un tipo de dominio de nivel superior que es usado y reservado para países o regiones.

- **Dirección IP** Es un número que identifica de manera lógica y jerárquica a una interfaz de dispositivo (una computadora) dentro de una red que utilice el protocolo IP. La dirección IP no es fija, puede cambiar.

- **Expresiones regulares** Es una notación para especificar patrones, donde cada uno de estos patrones concuerda con una serie de cadenas.

- **GPL** *General Public License* Licencia creada por *Free Software Foundation* . Esta licencia está orientada a proteger la libre distribución, modificación y uso de software. Su propósito es proteger el software de intentos de apropiación.

- **HTML** *Hyper Text Markup Language* Es un lenguaje utilizado para crear páginas Web y contiene etiquetas, éstas etiquetas le dicen al navegador cómo desplegar la página. El archivo debe tener extensión htm o html.

- **HTTP** *Hypertext Transfer Protocol* Es un protocolo muy usado en la Red. Es un sistema, mediante el cual se envían peticiones para acceder a una página web y su respuesta. Es un protocolo "sin estado", es decir, no guarda información sobre conexiones anteriores.

- **IANA** *Internet Assigned Numbers Authority* Es la entidad encargada de controlar números para protocolos, dominios de nivel superior de código de país y mantiene las direcciones de IP.

- **IP** *Internet Protocol* Protocolo usado para la comunicación de datos a través de una red. Los datos en este red, son enviados en bloques llamados "paquetes".

- **JAR** *Java Archives* Es un formato desarollado por SUN que permite agrupar clases para el lenguaje Java. Provee un compresión de archivos y reduce la carga administrativa en la distribución de clases. Es un formato muy utilizado en ambientes Java.

- **Javascript** Lenguaje interpretado, orientado a páginas web, con una sintaxis similar a la del lenguaje Java.

- **JCA** *Java Cryptography Architecture* Este API fue diseñado con el objetivo de que los desarrolladores provean a sus programas funciones de seguridad.

- **JDK** *Java Development Kit* Incluye el API de Java, la máquina virtual de Java, el compilador y otras funcionalidades.

- **J2EE** Es un estándar orientado al desarrollo de aplicaciones empresariales. Permite el acceso a bases de datos, utilización de directorios distribuidos, acceso a métodos remotos, funciones de correo electrónico, aplicaciones web y uso de Beans, entre otros.

- **J2SE** *Java 2 Platform Standard Edition* Permite el desarrollo y despliegue de aplicaciones Java en escritorios y servidores y ambientes de tiempo real. Incluye clases que soportan el desarrollo de servicios web de Java.

- **MD5** *Message-Digest Algorithm 5* Es un algoritmo de reducción criptográfico de 128 bits. Este algoritmo es muy utilizado. Fue diseñado por Ronald Rivest en 1991. En el año 2004 se divulgó ciertos problemas de seguridad de este algoritmo.

- **Microsoft** *Microcomputer Software* Empresa estadounidense creada por Bill Gates y Paul Allen en 1975. Producen sistemas operativos que se utilizan en la mayoría de computadoras hoy en día.

- **MVC** *Model-View-Controller* Es un modelo creado por la empresa Xerox en los años 80. Este modelo permite la separación de un problema en sus partes, permitiendo enfocarse en cada una de ellas específicamente.

- **Paquetes** Es una forma de organizar grupos de clases. Estos grupos pueden estar relacionados por ámbito, finalidad o herencia. Todas las clases que forma

un mismo paquete deben estar identificadas con la sentencia *package* seguida del nombre del paquete.

- **PostgreSQL** Es un servidor de bases de datos relacional de código libre bajo la licencia BSD.

- **SHA-1** *Secure Hash Algorithm 1* Es un sistema de funciones de dispersión. No se ha encontrado ningún ataque efectivo para este algoritmo.

- **SUN Microsystems** *Stanford University Network* Empresa informática creada en 1982 por Andreas von Bechtolsheim, alemán y Vinod Koshla, Bill Joy y Scott McNealy, estadounidenses. Han desarrollado la plataforma de programación Java.

- **TLD** *Top-level domain* Es la última parte de un nombre de dominio en Internet.

- **URL** *Uniform Resouce Locator* Es un formato estándar, que se usa para nombrar documentos, imágenes, etc. en Internet, que por medio de una secuencia de caracteres indica la ubicación del recurso indicado.

- **WAR** *Web Archive* Es una especificación desarrollada por SUN que permite agrupar un conjunto de clases y archivos que pertenecen a una aplicación web. Estos archivos son muy utilizados sobre todo por servidores de aplicación como Tomcat.

- **WWW** *World Wide Web* Es un sistema que funciona sobre Internet, se utiliza un navegador para extraer información de ella.

- **XML** *eXtensible Markup Language* Es un metalenguaje extendido de etiquetas permite definir la gramática de lenguajes específicos. Es una manera de definir lenguajes específicos.