Java Struts Framework

Juan F. Rodríguez Hervella
Univ. Carlos III de Madrid
Av. Universidad, 30, Edif. Torres Quevedo. E-28911 Leganés (Madrid)
Tel: (+34) 91-624-8859
E-mail: jrh@it.uc3m.es

Resumen

Struts es un proyecto de la "Apache Software Foundation" [6] que tiene como objetivo proporcionar un entorno específico para la construcción de grandes aplicaciones web. Se trata de un entorno basado en el concepto de "Model 2", que es una variación del paradigma de diseño clásico conocido como "Model View Controller" (MVC), donde se separan claramente los roles de los diseñadores web, los programadores java y los administradores de bases de datos. El núcleo del entorno está constituido por un conjunto de tecnologías estándar, como son los Servlets, JavaBeans, ResourceBundles y XML. En el presente trabajo se describe completamente el entorno de diseño y se aplica a un desarrollo que tiene como objetivo proporcionar servicios de "Tunnel-Broker" (intermediario) para la configuración automática de túneles entre proveedores de servicio. Los túneles se utilizan para proporcionar servicios de multihoming en redes IPv6.

1. Introducción

Los Servlets de Java han demostrado su superioridad frente a los CGIs, debido a su portabilidad y extensibilidad. Un servlet no es más que una clase Java usada para extender las capacidades de los servidores que albergan aplicaciones accedidas mediante un modelo de programación basado en peticiones/respuestas. El problema de los Servlets es que escribir las respuestas mediante "println()" y en formato HTML resulta tedioso. No obstante, la solución a este problema aparece con los JSPs, lo que permite trasladar la respuesta al usuario y embeberla dentro de una página HTML. Hoy en día los desarrolladores pueden mezclar fácilmente HTML y código Java, además de poder utilizar los Servlets a discreción.

El problema del modelo anteriormente descrito es que las aplicaciones web han pasado a centrarse en desarrollos basados en JSPs ("JSP-centrics"). Los desarrolladores y analistas funcionales necesitan otro tipo de herramientas para poder controlar el flujo de la aplicación.

Si pensamos detenidamente sobre las herramientas disponibles, vemos que lo que realmente se necesita es un paradigma de diseño nuevo. Mientras que los servlets pueden utilizarse para el control de flujo, los JSPs pueden enfocarse para realizar el trabajo sucio de construir HTML. Esta forma de utilizar ambas herramientas se conoce como Modelo-2, mientras que la utilización de JSPs única y exclusivamente se conoce como Modelo-1. La figura 1 representa los tres componentes que conforman la arquitectura MVC.

El artículo está estructurado en dos partes bien diferenciadas. La primera parte está constituida por la introducción al paradigma MVC/Model-2, que se describe en el capítulo 2. La segunda parte de este artículo describe un mecanismo que permite la configuración automática de túneles para propósitos relacionados con el soporte de multihoming en IPv6. Este mecanismo puede ser implementado dentro del entorno de desarrollo proporcionado por Java Struts. Por consiguiente, en el capítulo 3 se introduce la idea de "multihoming tunnel broker" y en el capítulo 4 se analiza una posible implementación basada en Struts. El artículo finaliza en el capítulo 5 con unas breves conclusiones y trabajos futuros.

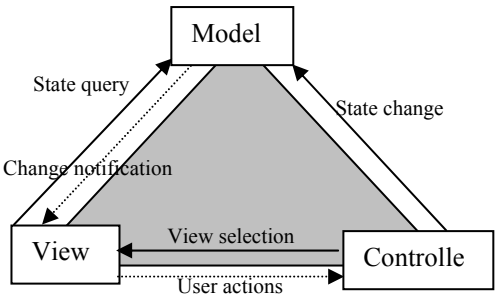

Figura 1: Model-View-Controller

2. Patrón Modelo-Vista-Controlador

En el modelo MVC, el flujo de la aplicación se controla mediante un agente central. El controlador reenvía las peticiones (HTTP-requests) hacia un manejador apropiado. El "Modelo" representa la lógica de negocio de la aplicación web. El "Control" pasa del manejador a la vista apropiada a través del controlador y finalmente se presenta el resultado de la transacción al usuario. El reenvío del controlador a la vista apropiada puede determinarse estáticamente mediante una serie de asociaciones a partir de un archivo de configuración. Este paso intermedio permite desacoplar la "vista" del "modelo", muy en línea con los conceptos de facilidad de manejo y extensibilidad de la aplicación que se persigue con el modelado bajo el

entorno de Struts. La figura 2 muestra un esquema a alto nivel de los componentes principales que constituyen la arquitectura de Struts.

El "modelo" dentro del sistema MVC se subdivide en dos subsistemas, la parte que representa el estado interno del sistema y las acciones que pueden realizarse para modificar dicho estado.

La mayoría de las aplicaciones representan el estado interno del sistema como un conjunto de JavaBeans. Las propiedades del bean representan el estado del sistema y los métodos representan las operaciones relacionadas con la lógica de negocio. Por ejemplo, una cesta de la compra que almacena los artículos seleccionados por el usuario hasta la fecha, puede representarse mediante un JavaBean.

La "vista" del modelo MVC se construye utilizando JSPs. Los JSPs pueden contener texto estático (HTML) y además pueden presentar contenido dinámico basado en la interpretación en tiempo de solicitud de la página de determinadas etiquetas especiales. El entorno de JSPs también proporciona un conjunto de acciones estándar que suelen ser muy comunes en la programación con JSPs y que pueden reutilizarse en distintos proyectos. Además de estas acciones predefinidas, existe una forma de definir etiquetas propias, lo que se conoce con el nombre de "custom tag libraries".

El entorno Struts proporciona un conjunto de etiquetas personalizables ("custom tags") para facilitar la creación de interfaces de usuario. Además, dichas etiquetas interaccionan con los "ActionForm" beans, que son las herramientas que dentro del modelo Struts capturan y validan cualquier entrada proporcionada por el usuario.

El "controlador" del entorno de desarrollo Struts se centra en la recepción de solicitudes por parte del cliente. Este cliente suele ser un usuario ejecutando un navegador web. El controlador decide qué lógica de la aplicación debe ser ejecutada en función de la petición del cliente, y posteriormente delega la responsabilidad de producir la siguiente fase del interfaz de usuario a una determinada "vista". El componente principal del controlador es un servlet de la clase "ActionServlet". Este servlet se configura definiendo un conjunto de "ActionMappings". Una "ActionMapping" no es más que una ruta que se asocia a una determinada petición HTML y que típicamente especifica el nombre completo de una clase de tipo "Action". Todas las acciones son subclases de la clase "Action". Las acciones encapsulan llamadas a las clases que representan la lógica de negocio, interpretan los resultados y eventualmente redireccionan el control al componente de la "vista" apropiado para crear una respuesta para el usuario.

En los siguientes apartados se describe paso a paso los elementos que Struts utiliza para implementar el modelo MVC/Model-2.

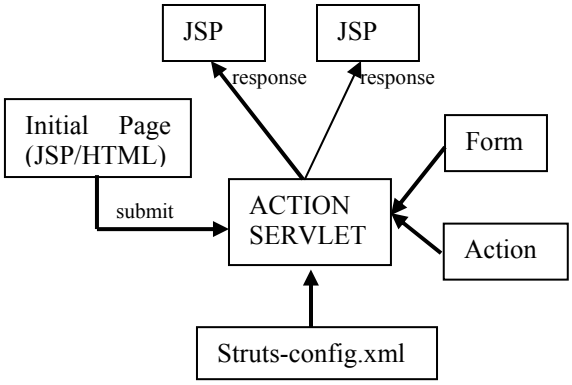

Figura 2: Struts a vista de pájaro

2.1. Componentes de la parte del modelo (MVC)

En general, el desarrollo de la parte del modelo de la arquitectura MVC se centra en la creación de JavaBeans que soportan todos los requerimientos funcionales de la aplicación. Estos beans son muy dependientes de la aplicación que se quiera desarrollar, pero generalmente se pueden clasificar en varias categorías.

El entorno Struts normalmente presupone la definición de al menos una clase "ActionForm" para la entrada de datos por parte del usuario. Los "ActionForm" beans poseen propiedades que se corresponden en su mayor parte con las propiedades de los beans que representan la lógica de negocio.

Otros tipos de beans son aquellos que representan el estado del sistema, normalmente referenciados en la literatura como "system state beans". El estado actual del sistema se representa por uno o varios de estos tipos de beans. Cuando la aplicación es grande, dichos beans pueden representar información que se almacena y recupera de medios persistentes (bases de datos) mediante "entity EJBs"[1].

Por último, dentro de la parte del modelo podemos considerar aquellos beans que encapsulan la lógica funcional de la aplicación dentro de métodos específicamente diseñados para dicho propósito, aunque estas llamadas a métodos pueden repartirse entre los anteriormente comentados tipos de beans, en la mayoría de los casos las aplicaciones desarrollan beans específicos que recubren la funcionalidad proporcionada por los elementos que representan la lógica de negocio (EJBs).

[1] Enterprise Java Beans, ver capítulo 4 más adelante en este mismo trabajo.

2.2. Componentes de la parte de la vista (MVC)

Struts define todas las etiquetas HTML como etiquetas JSPs. Esto permite utilizar JavaBeans para la creación de formularios y otros elementos relacionados con los interfaces que antiguamente se construían usando HTML.

También existen etiquetas JSP para creación de lógica dentro de las páginas, de entre las que cabe destacar:

- Iterate: repite el cuerpo de la etiqueta para cada elemento de una colección Java determinada.
- Present: ejecuta el cuerpo de la etiqueta cuando una determinada propiedad existe en la petición HTML. Es el equivalente a un "if" en programación estructurada.
- NotPresent: similar al tag anterior, pero solamente se ejecuta el cuerpo de la etiqueta cuando la propiedad especificada no existen en la petición HTML.
- Link: equivalente a la etiqueta HTML <a>
- Img: equivalente a la etiqueta HTML <img>
- Paramenter: recupera un determinado valor de la petición HTML y lo almacena como una propiedad del bean.

Struts proporciona facilidades para validar dentro de los beans los campos introducidos por los usuarios. Para ello basta con redefinir el método "validate" dentro de la clase "ActionForm" sobre la que se trabaja. El servlet controlador ejecuta este método después de que las propiedades del bean se han rellenado pero justo antes de que se llame al método "execute" de la clase "ActionForm".

Además de la utilización de las librerías "custom tags" anteriormente comentadas, el programador puede construirse sus propias etiquetas. Para finalizar, decir también que Struts proporciona un conjunto de clases estándar para construir aplicaciones con soporte de internacionalización. Para más información sobre internacionalización, se recomienda al lector consultar [4].

2.3. Componentes de la parte del controlador (MVC)

Struts incluye un servlet que implementa la función principal del entorno que consiste en asociar una URI con una determinada clase "Action". El servlet es un elemento de la clase "ActionServlet". De esta forma, las principales tareas del programador con respecto al controlador pueden resumirse en los siguientes pasos:

- Escribir una subclase de la clase "ActionForm" que interactúa entre el modelo y la vista.
- Escribir una subclase de la clase "Action" para cada petición lógica que puede recibir la aplicación.
- Configurar una "ActionMapping" (en XML) para cada petición lógica que puede ser enviada. El archivo de configuración que contiene dichas asociaciones se denomina "struts-config.xml".

El objetivo de una clase "Action" es procesar una petición, a través del método "execute", y devolver un objeto de tipo "ActionForward" que identifica donde debería redirigirse el control para devolver una respuesta adecuada.

2.4. Configuración mediante "ActionMappings"

Para que el entorno Struts funcione correctamente, el servlet controlador del modelo MVC necesita conocer varias cosas acerca de cómo cada petición URI debe ser asociada. Este conocimiento se encapsula en una clase java denominada "ActionMapping". Esta clase se crea dinámicamente a partir de un fichero XML. Por consiguiente una de las responsabilidades del desarrollador del sistema consiste en la creación de dicho fichero, denominado "struts-config.xml". Este fichero se debe almacenar dentro del directorio WEB-INF de la aplicación web.

El elemento XML más externo es <struts-config>. Dentro de este elemento, existen tres elementos principales: <form-beans>, <global-forwards> y <action-mappings>. El elemento <forms-beans> contiene las definiciones de los beans utilizados. Estos beans normalmente son "ActionForms". La etiqueta <global-forwards> contiene definiciones de reenvío globales. Estas definiciones son instancias de la clase "ActionForward" . Recordemos que al final del método "execute" de un objeto de la clase "Action", se suele devolver un objeto de tipo "ActionForward". Esta etiqueta permite asociar nombre lógicos a recursos específicos, como por ejemplo JSPs. Esto resulta ser muy flexible porque permite cambiar de recurso (de página JSP) sin necesidad de cambiar todas las clases donde se utiliza una referencia a dicho recurso. Por último, la etiqueta <action-mappings> define el conjunto de acciones que realiza la aplicación web basada en Struts. Se trata de una asociación entre peticiones web y objetos/páginas JSP.

A modo de ejemplo se presenta en la figura 3 el contenido del archivo "struts-config.xml", obtenida de la "Apache Software Foundation" [6]. Lo primero que se observa en la existencia de un bean llamado "logonForm". Dicho bean extiende la clase "ActionForm" y se encarga de recoger los valores introducidos por el usuario en el formulario. Internamente el framework de Struts accede al bean mediante el nombre especificado en el atributo "name". También existe un "ActionForward" llamado "logong", y que presenta al usuario la página "logon.jsp". El atributo "redirect" indica que la respuesta que se envía al cliente es simplemente reenviada y se mantiene el contexto de la petición

http. Si “redirect” se pone a “trae”, se notifica al cliente web que debe realizar una nueva petición http de tal forma que los valores originales de la petición se pierden. A continuación existe una acción definida. Dicha acción, identificada por el atributo “path” como “logon”, hace uso del bean “logonForm” definido con anterioridad e identificado en la acción mediante el atributo “name”. Se utiliza validación (“validate=true”) y además se trata de una acción que se utiliza por defecto para responder a todas aquellas peticiones que no se correspondan con ninguna otra acción (“unknown=true”). El atributo “input” se refiere a la página que se muestra en caso de que la validación de los datos sea incorrecta. El resto de los atributos son autoexplicativos.

3. Multihoming-Tunnel Broker.

El concepto de multihoming surge para mejorar la conectividad y fiabilidad frente a fallos en las grandes corporaciones o ISPs (Internet Service Providers). Dado que la conectividad global ha llegado a ser un recurso crítico en algunas organizaciones, muchas de ellas han implantado más de una conexión con el exterior. No obstante, para obtener un aprovechamiento máximo de múltiples conexiones con el exterior, se deben desarrollar herramientas tanto en las máquinas finales como en los routers a fin de poder manejar dinámica y transparentemente los distintos caminos alternativos.

```
<struts-config>
   <form-beans>
      <form-bean
         name="logonForm"
         type="org.apache.struts.webapp.example.LogonForm" />
    </form-beans>
   <global-forwards
      type="org.apache.struts.action.ActionForward">
      <forward
         name="logon"
         path="/logon.jsp"
         redirect="false" />
   </global-forwards>
   <action-mappings>
      <action
         path="/logon"
         type="org.apache.struts.webapp.example.LogonAction"
         name="logonForm"
         scope="request"
         input="/logon.jsp"
         unknown="true"
         validate="true" />
   </action-mappings>
</struts-config>
```

Figura 3: struts-config.xml

En el mundo IPv4, se han desarrollado varias soluciones para soportar multihoming. La solución más generalizada consiste en inyectar información de alcance en el sistema de routing, lo que contribuye al crecimiento exponencial de las tablas de rutas de los encaminadotes del núcleo de internet. Al menos una solución que presenta mejores perspectivas de escalabilidad respecto al protocolo de encaminamiento ha sido desarrollada para el nuevo protocolo de internet IPv6. No obstante, la solución descrita en la RFC 3178 [1] presenta problemas de cara a su implantación, debido a que se requieren muchas configuraciones manuales del sistema y un alto grado conocimiento de administración de redes. Existe un grupo de trabajo del Internet Engineering Task Force (IETF) que se encarga de desarrollar soluciones de multihoming para IPv6. En el momento de escribir este trabajo existen varias propuestas, pero no se acaba de encontrar una solución adecuada que satisfaga todos los posibles escenarios. En lo que respecta a la solución propuesta en [1], su ámbito de aplicación no se encuentra claramente definido pero parece que se enfoca a soluciones para grandes corporaciones o ISPs.

En [2] se presenta una forma de implantación de la RFC 3178 basada en el concepto de “Tunnel-Broker”. Un “Tunnel-Broker” es un agente que interactúa con el usuario final y actúa como su representante para realizar una serie de tareas en segundo plano. Normalmente los “tunnel-brokers” se implementan como aplicaciones o servicios web.

En los siguientes apartados se describe de forma breve el modelo de “tunnel-broker” descrito originalmente en [3], la solución de multihoming para IPv6 basada en [1] y el servicio de multihoming para IPv6 basado en los anteriores conceptos, descrito en [2].

3.1. El modelo “tunnel-broker”.

La RFC 3053 [3] presenta un modelo de diseño para ofrecer configuración automática de túneles IPv6 sobre IPv4. Este modelo pretende hacer más fácil la transición de IPv4 a IPv6, pues proporciona una forma automática de construir enlaces IPv6 a través de la infraestructura de encaminamiento proporcionada por IPv4.

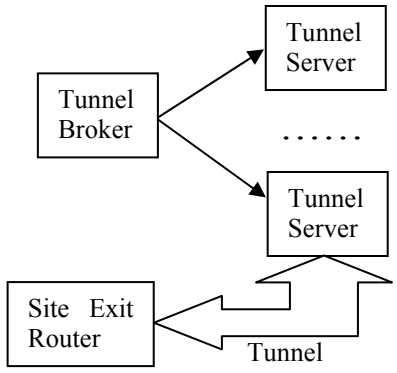

Figura 4: Entorno del tunnel-broker

Hasta que exista conectividad IPv6 nativa entre el cliente y el ISP, los modelos de transición basados en éste y otros mecanismos ya definidos serán predominantes. Resulta pues muy importante para

los proveedores de acceso el poder ofrecer servicios de valor añadido a sus clientes (IPv6 en este caso), aún cuando su red todavía no se haya reconvertido totalmente al nuevo protocolo. El modelo utilizado en la RFC 3053 [3] se puede observar en forma de esquema en la figura 4.

Este modelo posee los siguientes elementos:

- Tunnel-broker: es el elemento al que se conectan los usuarios para crear, modificar y borrar túneles. El tunnel-broker se comunica con uno o varios tunnel-servers, que son los encargados de la creación de dichos interfaces.
- Tunnel-server: constituye el extremo del túnel que se intenta configurar en base a solicitudes que provienen del tunnel-broker.
- Site exit router(s): representan a los routers de borde de la red del proveedor del servicio. El tunnel-server puede querer comunicarse con dichos routers para inyectar nuevas rutas y/o modificar el encaminamiento.

La interacción entre el usuario y el tunnel-broker no está especificada, aunque lo más sencillo consiste en utilizar el protocolo http. Aunque actualmente la mayoría de los servicios de tunnel-broker se ofrecen sin coste alguno para el usuario (tunnel-brokers gratuitos), se puede extender el modelo con elementos de facturación, autenticación y seguridad.

La seguridad es muy importante en cualquier servicio que se ofrezca a través de internet. En el caso específico del modelo tunnel-broker, se necesita securizar todas aquellas interacciones entre los distintos elementos funcionales de la arquitectura. Fundamentalmente aquellas interacciones entre el cliente y el tunnel-broker, las interacciones entre el tunnel-broker y el tunnel-server y posiblemente aquellas interacciones adicionales que se definan a posteriori, como por ejemplo con la infraestructura de DNS o con el sistema de base de datos. Para más información sobre los tunnel-brokers en general, se remite al lector a [3].

3.2. Soporte de Multihoming IPv6 en los routers frontera.

La RFC 3178 [1] describe una solución para proporcionar multihoming en IPv6. Esta solución se basa en la configuración de una serie de túneles entre diferentes ISPs y el sitio multihomed con el objetivo de proporcionar caminos alternativos en caso de que uno de los enlaces de salida quede temporalmente inutilizado. El esquema general de esta solución se observa en la figura 5. En dicha figura se observan dos proveedores de acceso a internet que se comunican a través de la infraestructura global de internet. Un sitio, denominado sitio multihomed, adquiere servicio de ambos proveedores para mejorar su disponibilidad y robustez frente a fallos imprevistos.

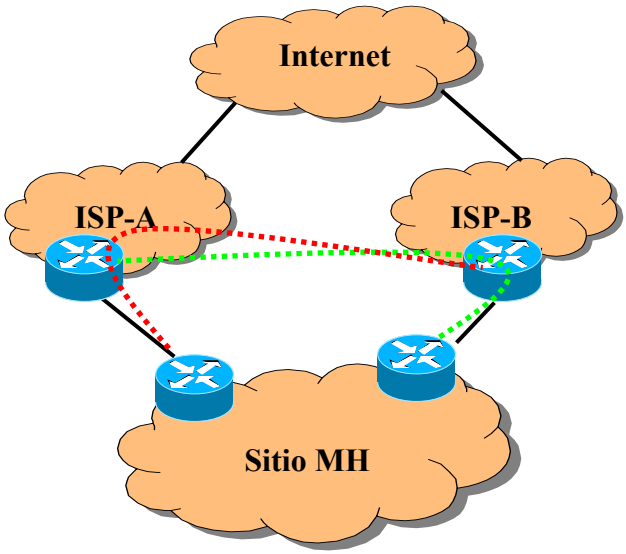

Figura 5: IPv6 Multihoming en RFC 3178

El sitio multihomed posee dos proveedores, ISP-A e ISP-B. Cada proveedor delega determinados prefijos al sitio. Para obtener los beneficios de tolerancia frente a fallos de los enlaces directos, [1] propone la creación de un túnel que constituye un camino virtual para conectar con el ISP, teniendo dicho camino la fundamental propiedad de atravesar el otro ISP. Esta propiedad se consigue en IPv6 de una forma sencilla utilizando como dirección local del túnel una dirección proporcionada por el otro proveedor.

Además de la configuración de los túneles, se debe adaptar el protocolo de routing para que cuando alguno de los enlaces directos falle, se encamine adecuadamente el tráfico utilizando el camino virtual que representa el túnel.

Suponiendo que el encaminamiento reacciona con suficiente velocidad, este mecanismo proporciona características de protección frente a fallos de los enlaces “upstream” con los proveedores e incluso preserva las comunicaciones ya establecidas.

A pesar del beneficio potencial de esta solución para aquellos sitios multihomed que deseen obtener las características antes mencionadas, existe el problema de la gran complejidad de implantación de una solución de este tipo en proveedores de servicio comerciales, sobre todo cuando la solución se extiende a modelos donde se involucran más de dos ISPs.

3.3. El servicio de Multihoming IPv6 Tunnel-Broker

Primeramente, una diferencia importante respecto al funcionamiento del tunnel-broker que se describe en

[3], es que el tunnel-broker para multihoming establece túneles IPv6 sobre IPv6. El tunnel-broker recibe peticiones para crear/modificar/borrar túneles. Dichas peticiones deben tener al menos los siguientes campos:

- Identificación del cliente
- Dirección IPv6 del extremo del túnel cliente (perteneciente al espacio delegado por el otro ISP).
- Prefijo IPv6 para el que se crea una ruta de backup (perteneciente al espacio delegado por éste ISP).
- Información de autorización pertinente (login/password, certificados, etc).

El tunnel-broker se encarga de comprobar que toda la información suministrada es correcta (validación), y posteriormente se comunica con el tunnel-server transmitiendo la información necesaria para realizar la configuración del túnel solicitada. Una vez que el tunnel-server ha terminado su trabajo, el tunnel-broker genera una respuesta al cliente donde se le comunica la dirección IPv6 del extremo del túnel en las instalaciones del ISP, para que dicho cliente pueda, manualmente o por otros medios, configurar su extremo del túnel en sus instalaciones. Recordemos que el tunnel-broker trata de automatizar la configuración de túneles en un extremo, pero en [3] no se define ninguna forma para automatizar el proceso en las instalaciones del usuario del servicio. Recordemos que el servicio de tunnel-broker es un servicio basado en el modelo cliente-servidor.

El tunnel-server puede colocarse junto con algún router de borde, de forma que se simplifique la comunicación entre ambos elementos. Finalmente, en [2] se describe una extensión al modelo de multihoming en IPv6 haciendo uso de varios tunnel-brokers encadenados para proporcionar protección frente a fallos en varios saltos, aunque esta opción todavía está bajo estudio.

4. Implementación del Multihoming IPv6 Tunnel-broker usando Java Struts

El modelo del multihoming IPv6 tunnel-broker encaja perfectamente dentro del desarrollo web proporcionado por el entorno J2EE[2] y también dentro de la metodología de desarrollo proporcionada por el entorno Struts.

Existen muchas implementaciones basadas en software libre de tunnel-brokers para proporcionar conectividad IPv6 a través de infraestructura de red IPv4. Por poner un ejemplo, en la figura 6 se puede observar la página de acceso al servicio de tunnel broker proporcionado por la empresa anglosajona British Telecom (BT).

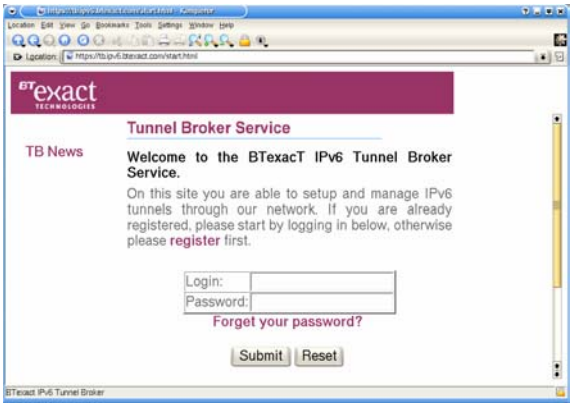

Figura 6: tunnel-broker de BT

Para el multihoming tunnel-broker la implementación es muy parecida a un tunnel-broker tradicional. Las acciones que se han identificado son las siguientes:

- Creación de túneles
- Liberación de túneles
- Consulta de túneles

Existen dos roles en la aplicación:

- Usuario
- Administrador

La asignación de los roles se realiza previa identificación del usuario que accede al sistema. Cada rol posee una serie de permisos sobre las acciones que pueden llevar a cabo. Resumidamente, se da permisos de alta/baja/modificación de cualesquiera túneles que se encuentren definidos al rol Administrador. El rol usuario posee derechos solamente sobre aquellos túneles creados bajo su auspicio.

Como en todo túnel broker, existe una primera fase de autenticación donde se asigna un determinado rol al usuario. Esta fase se suele programar en Struts usando una clase que extiende la clase "Action". Además se suele utilizar un "ActionForm" que el "ActionServlet" rellena con los datos introducidos por el usuario (login y password), de tal forma que cuando se ejecuta la acción configurada en el fichero "struts-config.xml", se pasa el objeto que contiene dichos datos. El código de la clase que almacena los datos del usuario se muestra en la figura 7. Podemos observar que se trata de la implementación típica de un objeto bean, donde se definen los métodos "get" y "set" necesarios para leer y modificar los atributos, respectivamente, pero con el añadido de que se extiende la clase "ActionForm".

En Struts 1.1 existen una nueva clase, denominada "DynaActionForm", que permite declarar las propiedades dentro del archivo de configuración de Struts en vez de tener que crear manualmente un objeto.

[2] Java 2 Enterprise Edition, http://java.sun.com/j2ee

```
Package tunnelbroker;
Import org.apache.struts.action.*;

Public class LoginForm extends ActionForm {

/** atributos de la clase
  */
 Protected String username;
 Protected String password;

/** metodos get y set
  */
 Public String getUsername() {return this.username;};
 Public String getPassword1() {return this.password1;};

 Public void setUsername (String username) {this.username =
username;};
 Public void setPassword (String password) {this.password =
password;};

}
```

Figura 7: LoginForm.java

La clase "LoginAction.java" es la clase que se encarga de ejecutar la lógica de negocio asociada a la operación de autenticación. Su código se muestra en la figura 8.

```
Package tunnelbroker;
Import org.apache.struts.action.*;
Import javax.servlet.http.*;
Import java.io.*;

Public class LoginAction extends Action {

 Public ActionForward perform (ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest req,
                               HttpServletResponse res) {

   LoginForm lf = (LoginForm) form;

   String username = lf.getUsername();
   String password = lf.getPassword();

   loginBean = new loginBean( username, password);

   If (loginBean != null){
       If( loginBean.isUser() ) {
           Return mapping.findForward("userPages");
       } else {
           Return mapping.findForward("adminPages");
       }
   } else {
       Return mapping.findForward("retryAgain");
   }
 }

}
```

Figura 8: LoginAction.java

El código del método "perform()" posee tres partes bien diferencias, en la primera parte se realiza un cast del formulario al objeto de la clase LoginForm.java y se recuperan los valores de sus atributos. Estos valores son los que el usuario ha introducido al rellenar el formulario. Merece la pena comentar que el método "perform()" ha sido etiquetado como obsoleto. La última versión de Struts, la versión 1.1 en el momento de escribir este trabajo, recomienda el uso del método "execute()" en su lugar.

Posteriormente entra en juego la lógica de negocio, en este caso se trata de comprobar que el usuario está dado de alta en la base de datos del ISP. Dicho proceso de registro en la base de datos se realiza offline o por otros medios ajenos a la aplicación multihoming tunnel-broker. Para comprobar que el usuario es válido se utiliza una clase auxiliar que implementa el patrón de diseño "facade"[3], y que interactúa con un conjunto de EJBs que son los encargados en última instancia de acceder al sistema de base de datos.

Por último, se redirige el control a la página JSP adecuada mediante el uso de objetos de tipo ActionForward.

El siguiente paso depende de si el usuario que ha accedido al sistema es un usuario normal o es un administrador. En las siguientes secciones se describen con más detalle distintos aspectos que conforman la aplicación multihoming tunnel-broker .

4.1. Acciones del rol usuario

Una vez autenticado, se presenta al usuario una ventana donde puede ver un listado resumido de todos los túneles que tiene operativos. Al lado de cada túnel se presentan tres botones, uno para eliminar el túnel, otro para modificar los parámetros asociados, y un checkbox para desactivar/activar el mismo.

Para cada una de las acciones asociadas se define una clase que extiende la clase Action, y que será llamada por el correspondiente objeto hijo de la clase ActionForm, que encapsula los datos relevantes del formulario.

4.2. Acciones del rol administrador

Una vez autenticado, se presenta al administrador una ventana donde puede ver un listado de todos los túneles actualmente configurados en el sistema. Este listado aparece resumido y agrupado por usuarios, y puede ordenarse según varios criterios. Se presentan los mismos botones que en el apartado anterior junto con un checkbox (como en el caso anterior) para activar/desactivar el túnel en cuestión.

[3] Patròn de diseño que actúa como un recubridor de la funcionalidad de otro objeto, permitiendo desacoplar el objeto cliente del objeto que proporciona los servicios añadiendo una capa intermedia, que puede además ampliar la funcionalidad ofrecida. Ver [5]

Resumiendo, podemos ver al rol administrador como un elemento que puede gestionar más información, pero en cualquier caso tanto el rol administrador como el rol usuario utilizan por debajo los mismos objetos.

En futuras versiones se prevé que el rol Administrador sea capaz de gestionar los usuarios, realizando operaciones de registro, baja y modificación de los mismos.

4.3. Base de datos

El sistema de base de datos utilizado en la aplicación multihoming tunnel-broker posee varias tablas. El sistema utilizado es MySQL-4.0. Por ejemplo, la tabla "usuario" mantiene información de usuario requerida para poder acceder al sistema. Esta información ha tenido que ser cargada previamente en la base de datos. Cada usuario de la tabla "usuario" posee una referencia a la tabla "túnel", donde se encuentra toda la información relacionada con los túneles que tiene actualmente configurados. Existen otras tablas que ayudan a la descomposición en forma normal de la información almacenada lo que permite una mayor eficiencia en la gestión de la información.

4.4. Lógica de negocio

Al tratarse de una aplicación web de grandes dimensiones, existen muchos elementos que deben interaccionar entre sí. Normalmente las aplicaciones web se descomponen en varias capas, como se puede observar en la figura 9.

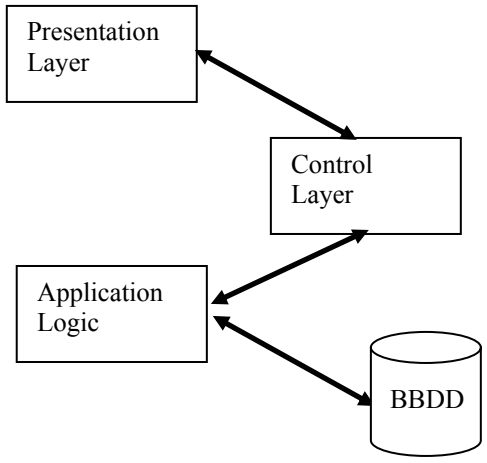

Figura 9: capas de una aplicación web típica

La capa de presentación se realiza mediante JSPs, la parte de control se realiza mediante "ActionServlets", "ActionForms", y extensiones de la clase abstracta "Action", como ya se ha explicado anteriormente.

La lógica de negocio se suele implementar utilizando la tecnología de EJBs. Los EJBs son la tecnología central de J2EE para la creación de lógica y datos de negocio. Resumiendo, los EJBs poseen las siguientes características:

- Aplicaciones distribuidas: los EJBs se comunican utilizando RMI.
- Distintos tipos modelan distintas partes de la lógica de negocio:
  - EntityBeans: para modelar el acceso/modificación/almacenamiento de datos persistentes. Normalmente interactúan con la base de datos del sistema.
  - SessionBeans: modelan una operación de la lógica del negocio, suelen utilizar entityBeans.
  - Message-Driven Beans: modelan procesos ejecutados como respuesta a la recepción de un mensaje, y su llamada es asíncrona.
- Son accedidos por distintos tipos de clientes: otros EJBs, servlets, clientes de aplicación, etc.
- En tiempo de ejecución, residen en un contenedor, que proporciona servicios añadidos.

En la figura 10 se puede observar una plataforma J2EE en su totalidad. Esta figura es de interés porque el sistema multihoming tunnel-broker sigue una arquitectura parecida, distribuyendo el acceso a los distintos elementos que conforman la aplicación entre distintos servidores.

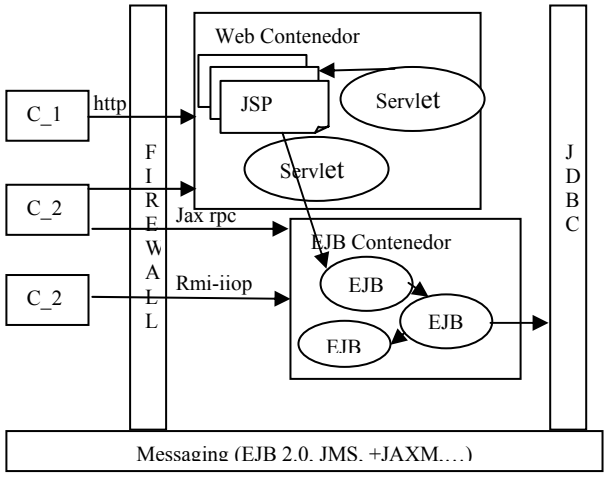

Figura 10: entorno típico distribuido

Concretamente, hasta el momento se han identificado los siguientes EJBs dentro de la plataforma multihoming tunnel-broker:

- LoginBean: bean de sesión responsable de mantener la información de login del usuario.
- UserBean: entity bean que representa la tabla de usuarios de la base de datos.

- TunelBean: entity bean que representa los datos persistentes almacenados en la tabla de túneles.
- AdminBean: session bean que representa el conjunto de operaciones que realiza el rol administrador.
- UserBean: sesion bean que representa las acciones permitidas a un usuario de rol Usuario.

No se descarta que se añadan nuevos EJBs según se avance en el ciclo de vida del desarrollo de aplicación.

4.5. Helpers

Se han desarrollado una serie de clases que son utilizadas por las acciones (del framework de Struts). Este paso intermedio sirve para descargar el framework de operaciones de acceso a EJBs. De esta forma, además de los típicos elementos contenedores de información del entorno Struts, existen una serie de clases denominadas "helpers", que básicamente actúan como "recubridores" del acceso a los elementos de la lógica de negocio.

4.6. Resumen

El estado de la implementación se encuentra en su desarrollo inicial. Se ha programado el acceso de los usuarios al sistema utilizando la infraestructura proporcionada por el entorno de Struts. La plataforma de desarrollo es un FreeBSD-4.10, con Java-1.4.2, TomCat-5 y MySQL-4.0. Se utiliza un programa específico para probar las funcionalidades de los EJBs, debido a que la versión disponible del J2EE para FreeBSD en el momento de escribir este trabajo es la 1.3. Dicho programa, que se inserta en la arquitectura de TomCat y a todos los efectos actúa como una API intermedia transparente para el programador de EJBs se puede descargar gratuitamente de http://www.openejb.org/.

El mero hecho de intentar atacar el problema del desarrollo de la aplicación multihoming tunnel-broker bajo la arquitectura de Struts ha proporcionado al autor una visión cercana de lo que supone programar aplicaciones web en entornos Java. La disponibilidad de entornos de desarrollo integrados que descarguen al programador de las tareas de configuración e implantación son fundamentales. En este sentido, la plataforma inicialmente elegida, FreeBSD-4.10, deja mucho que desear.

También se ha detectado que el tiempo de desarrollo se alarga con respecto al desarrollo de aplicaciones web basadas en otros lenguajes, como por ejemplo PHP. Aunque la mayoría de las implementaciones de tunnel-broker de software libre se encuentran programadas usando lenguajes de scripting, el autor considera que una implementación comercial de esta solución fiable y fácilmente extensible, se ajusta perfectamente al marco de trabajo de J2EE y en especial a la arquitectura MVC/Model-2 descrita en este trabajo.

5. Conclusiones y trabajos futuros.

Es este artículo se ha realizado una introducción al diseño modelado según el entorno Java Struts. Además, se ha aplicado el paradigma de diseño al desarrollo de una solución multihoming para IPv6 mediante el uso de tunnel-brokers [2].

A modo de resumen, la tabla 1 presenta las clases que constituyen el núcleo de Struts y su asociación con las responsabilidades clásicas de los componentes asociados al entorno modelo-vista-controlador:

Clase	Descripción
ActionForward	Selección de la vista
ActionForm	Datos para realizar un cambio de estado
ActionMapping	Evento de cambio de estado
ActionServlet	El controlador que recibe peticiones de usuario, cambios de estado y reenvía las vistas seleccionadas
Action	La parte del controlador que interacciona con el modelo para ejecutar los cambios de estado y comunica al ActionServlet la vista seleccionada.

Tabla 1: clases Struts y el modelo MVC.

Para que las diferentes vistas del diseño puedan hacer uso de los datos que constituyen la aplicación, el entorno Struts proporciona un conjunto de clases de ayuda en forma de etiquetas JSP, que se resumen en el cuadro 2.

Descriptor Tag Library	Propósito
Struts-html.tld	Extensiones JSP para formularios HTML
Struts-bean.tld	Extensiones JSP para el manejo de JavaBeans
Struts-logic.tld	Extensiones JSP para comprobar valores de propiedades.

Tabla 2: Clases de ayuda del entorno Struts

Finalmente, en la figura 11 se presenta a modo de resumen de todo lo comentado en este trabajo, el proceso de petición y respuesta típico dentro del marco de Struts de una forma gráfica detallada. La figura ha sido extraída de [4].

Hoy en día los desarrolladores necesitan construir herramientas que puedan ser fácilmente mantenidas a lo largo del tiempo. Entornos web como Struts hacen la vida más fácil al programador de tal forma que el desarrollador de aplicaciones puede dedicarse a desarrollar las funcionalidades únicas de la aplicación objetivo sin tener que molestarse por aspectos relacionados con la presentación o el control del flujo.

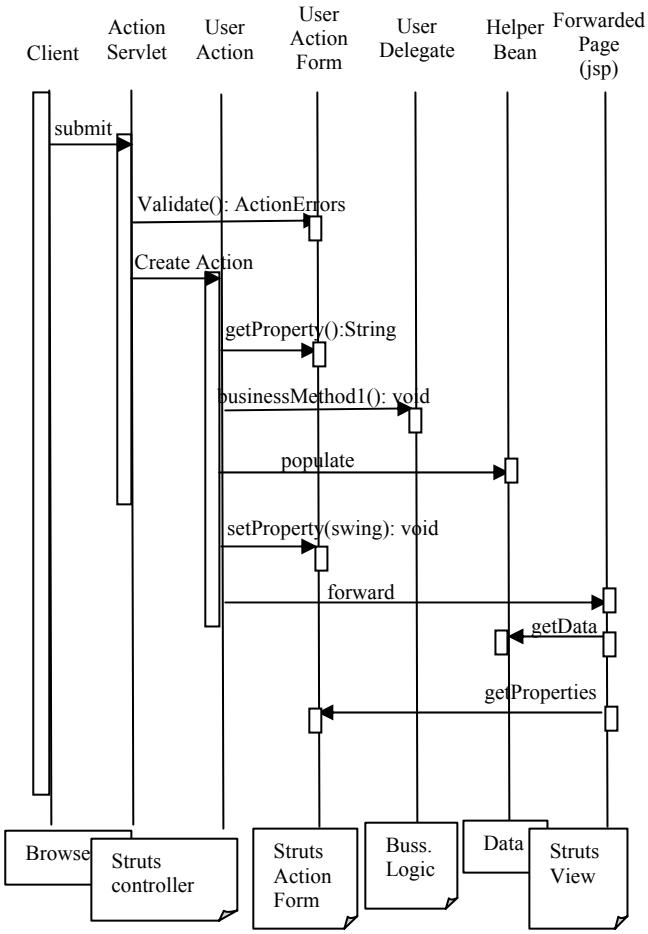

Figura X: Ejemplo de control del flujo

Struts se basa en los patrones de diseño definidos en [5] y en el paradigma MVC. Proporciona un desarrollo de aplicaciones basado en "capas", bajo un entorno más robusto y escalable que el método basado exclusivamente en JSPs. Struts se basa en tecnología Java y en estándares de la comunidad Sun para proporcionar estas características. No obstante, no hay nada nuevo bajo el sol, y como cualquier otro framework su utilización exitosa depende de la aplicación objetivo y del diseñador y analista funcionales encargados de llevar a cabo el proyecto.

Un aspecto fundamental del entorno Struts lo constituye la forma en que extiende el flujo petición-respuesta de una aplicación web convencional. El controlador del modelo Struts gestiona los caminos que la aplicación web recorre, ayuda en la recolección de los datos necesarios e incluso puede proporcionar información en varios idiomas (internacionalización).

No todo es un camino de rosas, por ejemplo, puede que la aplicación web objetivo no se adapte al modelo MVC, en cuyo caso el uso de Struts puede resultar inadecuado y costoso en cuanto a tiempo de desarrollo. Otros puntos débiles son que la elección del nombre de las clases es hasta cierto punto confuso, que una aplicación sólo puede utilizar una clase ActionServlet y que por ejemplo no se define la forma en que se debe acceder a la parte de datos persistentes del modelo, aunque el programador puede utilizar otras herramientas, como por ejemplo los EJBs.

A pesar de estos comentarios negativos, podemos afirmar para finalizar que el framework de desarrollo web proporcionado por Struts es uno de los entornos de desarrollo más popular y el más utilizado en la actualidad dentro de la comunidad Java.

Agradecimientos.

Mis más profundos agradecimientos van dirigidos a aquellas personas que han apoyado mi investigación durante todos estos años, en diferentes ámbitos y alcances. Ellos saben quienes son. Muchas gracias.

También merecen mención anónima mis compañeros de laboratorio, los administradores y todas aquellas personas que hacen que el día a día sea un poco más fácil de llevar.

Referencias

[1] J. Hagino and H. Zinder, "IPv6 Multihoming Support at Site Exit Routers", RFC 3178, October 2001.

[2] M. Bagnulo, J. F. Rodriguez, A. García, "Multihoming Tunnel Broker", EuroMicro, Rennes, France, September 2004.

[3] A. Durand, P. Fasano et al., "IPv6 Tunnel Broker", RFC 3053, January 2001.

[4] T. Husted, "Struts in Action", Manning Publications Co., 2003

[5] M. Floyd, "EJB design patterns: advanced patterns, processes and idioms", NY, John Wiley & Sons, 2002

[6] "Apache Software Foundation", Struts Framework version 1.1, http://jakarta.apache.org/struts/