CAPÍTULO 4
Clientes y Servidores en Sistemas Distribuidos

Patricia Bazán

Como hemos mencionado en el Capítulo 2, la arquitectura Cliente/Servidor o arquitectura de 2 niveles, identifica dos componentes que trabajan coordinadamente para dar la visión de un sistema único. Se diferencia de otras variantes de sistemas distribuidos en los siguientes conceptos:

Servicio - La relación cliente/servidor es la relación primaria entre dos procesos que se ejecutan en computadoras separadas. El servidor provee servicios y está a la escucha de requerimientos, mientras que el cliente solicita y consume servicios.

Recursos Compartidos - Los recursos compartidos en un sistema distribuido son administrados por los servidores que regulan el acceso a los mismos por parte de los clientes.

Protocolos Asimétricos - Hay una relación muchos a uno entre clientes y servidores. Por su parte la relación es asimétrica dado que los clientes siempre inician el diálogo y los servidores esperan los requerimientos.

Transparencia de la Ubicación - El servidor es un proceso que puede residir en la misma computadora del cliente o en otra computadora de la red, siendo transparente para el cliente dicha ubicación.

Comunicación basada en pregunta-respuesta - Clientes y servidores son procesos poco acoplados que interactúan a través de mensajes de tipo pregunta-respuesta, pudiendo ser estos sincrónicos o asincrónicos.

Escalabilidad - Los sistemas Cliente/Servidor pueden crecer moderadamente de manera horizontal (agregando clientes) o de manera vertical (potenciando al servidor).

Estas características aportan una distribución inteligente de funcionalidades a través de la red y otorgan un marco para el desarrollo de aplicaciones poco acopladas.

Por otra parte, para poder crear la visión de sistema único que requieren los sistemas distribuidos, es preciso contar con una infraestructura tecnológica que soporte cada una de las funciones.

La infraestructura red y comunicaciones ha crecido vertiginosamente en los últimos 20 años permitiendo un grado de interconexión global con alto ancho de banda y medios de transporte seguros y confiables.

El desafío de la industria informática es como agregar por encima de esta infraestructura física la arquitectura Cliente/Servidor que permitan unir las piezas del rompecabezas. Este desafío obliga a proveer:

- Protocolos de transporte que soporten el intercambio entre redes de computadores en forma confiable.
- Sistemas operativos de red que garanticen seguridad y privacidad y faciliten a los programas la tarea de encontrar los servicios que necesitan consumir
- Bases de datos para almacenar, recuperar y organizar grandes volúmenes de información incluso multimedial.
- Sistemas GroupWare que permitan el intercambio persona a persona y conferencias grupales.
- Agentes inteligentes que ayuden a los humanos a organizar sus actividades en el *ciberespacio*
- Plataformas de gerenciamiento de sistemas distribuidos que permitan mantener todo unido día a día.

El capítulo se organiza de la siguiente manera: en la Sección 1 se describen los roles y funciones de un componente servidor y sus variantes. En la Sección 2 se caracteriza el componente cliente y se describen algunas tecnologías asociadas. En la Sección 3 se describe la tecnología Cliente/Servidor con Java, detallando en la Sección 4 la tecnología JEE en entornos distribuidos. En la Sección 5 se presenta el patrón de diseño MVC a la luz de la tecnología Web. En la Sección 6 se arriba a algunas conclusiones.

4.1. Roles y Funciones del Servidor

El rol de un programa servidor es servir a múltiples clientes que requieren acceder a recursos compartidos que son propiedad del servidor. Veamos en detalle las funciones principales de un programa servidor:

- **Espera por requerimientos iniciados por el cliente -** Un programa servidor gasta gran parte de su tiempo esperando pasivamente requerimientos de los clientes que llegan generalmente en forma de mensajes arribando sobre una sesión de comunicación. Algunos servidores asignan sesiones dedicadas a cada cliente y otros crean un pool dinámico de sesiones reusables. Otros utilizan una combinación de ambos métodos. Lo cierto es que el servidor debe garantizar pronta respuesta a los requerimientos del cliente independientemente del mecanismo.
- Ejecuta muchos requerimientos a la vez - Un programa servidor que no soporta multitarea corre el riesgo de tener “en espera” a un cliente monopolizando recursos. El servidor debe ser capaz de servir concurrentemente a múltiples clientes protegiendo la integridad de los recursos compartidos.

- Atiende primero a los cliente VIP - El programa servidor debe ser capaz de proveer distintos niveles de prioridad en los servicios a sus clientes. Claramente un servicio batch tendrá prioridad más baja que un servicio de tipo interactivo.

- Inicia y corre tareas en segundo plano (background) - El programa servidor debe ser capaz de lanzar tareas que corren en background y que no están relacionadas con la tarea principal que se está ejecutando.
- Mantiene la corrida - Un programa servidor es una aplicación de misión crítica por lo cual debe garantizar un mantenimiento de su corrida y es tolerante a fallas.
- Crece a lo ancho y a lo largo - Un programa servidor tiene un apetito insaciable de memoria y capacidad de procesamiento por lo tanto debe ser escalable y modular.

Variantes de Servidores

Los sistemas Cliente/Servidor se clasifican en función del tipo de servicio que proveen los servidores, es decir, cuál es el recurso que administran. Estos recursos pueden ser archivos, datos, objetos, elementos de comunicación entre personas o transacciones.

- Servidores de Archivos - El recurso administrado es un archivo, concebido como conjunto de registro. El cliente solicita un archivo y el servidor responde enviando, a través de la red, bloques de registros. Es el mecanismo más primitivo para acceder

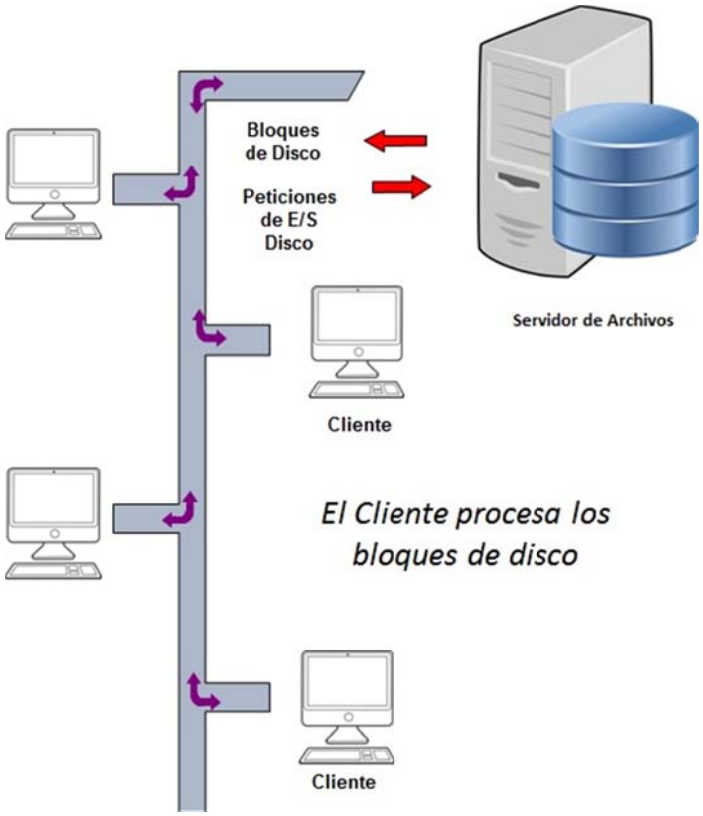

Fig. 32. Funcionamiento de un Servidor de Archivos

los datos dentro de un sistema de información. La Figura 32 muestra la comunicación de este tipo de servidores con sus clientes.

- Servidores de Base de Datos - El cliente envía requerimientos SQL como mensajes a un servidor de base de datos. El resultado de cada comando SQL es retornado a través de la red. La diferencia con el esquema anterior es que es el servidor el que procesa el mensaje y retorna los datos requeridos en lugar de retornar registro por registro como en un esquema servidor de archivos. La Figura 33 muestra la comunicación de este tipo de servidores con sus clientes.

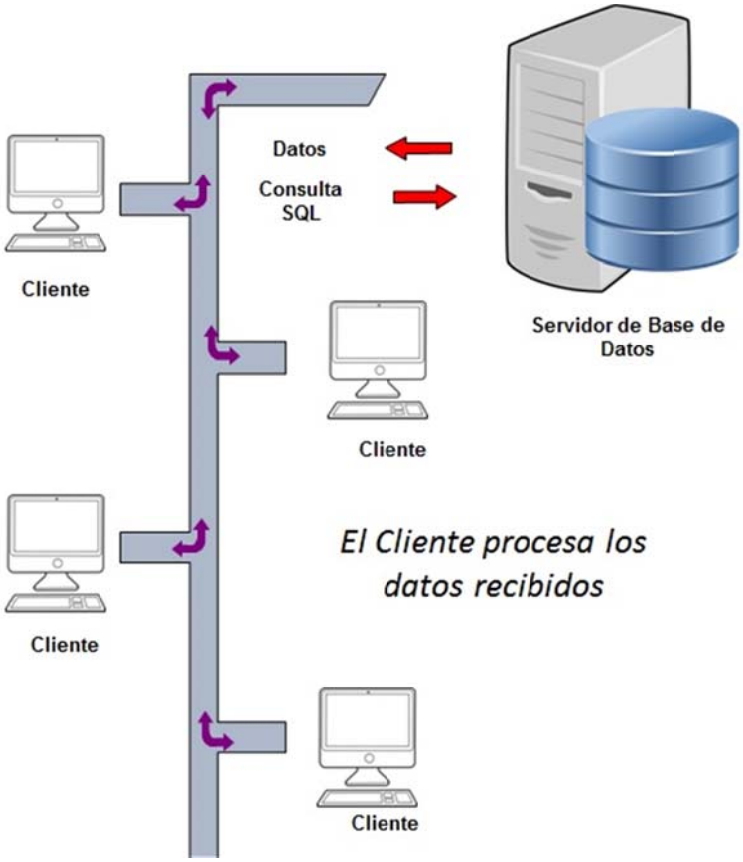

Fig. 33. Funcionamiento de un Servidor de Base de Datos

- Servidores de Transacciones - El cliente invoca un procedimiento remoto que reside en el servidor en un motor de base de datos. Este proceso remoto ejecuta un conjunto de sentencias SQL. El intercambio de red consiste en un único request/reply a diferencia del esquema anterior donde hay un mensaje request/reply por cada comando SQL. Este conjunto de sentencias SQL agrupadas se conocen con el nombre de transacción y constituyen una unidad atómica que tiene éxito o falla en su conjunto. Con este esquema se desarrollan aplicaciones donde el código se distribuye entre servidor (procesos de datos y reglas de negocios) y cliente (interface gráfica). El Capítulo 6 brinda detalles de este tipo de servidores. La Figura 34 muestra la comunicación de este tipo de servidores con sus clientes

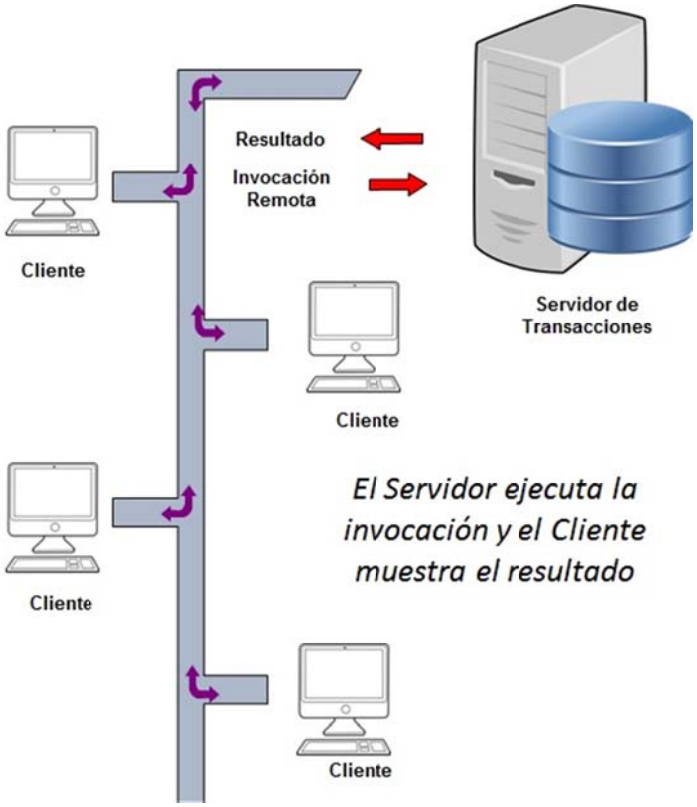

Fig. 34. Funcionamiento de un Servidor de Transacciones

- Servidores para Grupos de Trabajo - Son sistemas que permiten manejar información semi-estructurada como texto, imagen, mail, boletines y flujo de tareas. La Figura 35 muestra la comunicación de este tipo de servidores con sus clientes.

Fig. 35. Funcionamiento de un Servidor de Grupo de Trabajo

- Servidores de Objetos - Con este esquema las aplicaciones Cliente/Servidor se escribe como un conjunto de objetos que se comunican. Los objetos clientes se comunican con los servidores usando Object Request Broker (ORB).ORB localiza una instancia de la clase del objeto servidor, invoca el método requerido y retorna los resultados a la instancia del objeto cliente. La Figura 36 muestra la comunicación de este tipo de servidores con sus clientes.

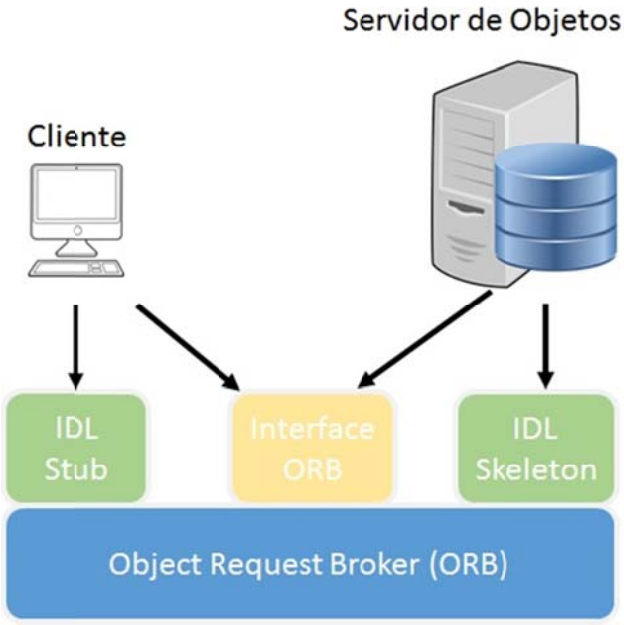

Fig. 36. Funcionamiento de un Servidor de Objetos

4.2. Roles y Funciones del Cliente

El rol del Cliente es el de iniciar el diálogo con el servidor solicitando acceso a los recursos compartidos que éste posee. Una vez obtenidos dichos recursos, es responsabilidad del cliente mostrar los resultados al usuario, para lo cual es muy probables que deba ejecutar código.

En este sentido, en una arquitectura Cliente/Servidor clásica en 2 niveles, el cliente resolverá la lógica de presentación al usuario (GUI - Graphic User Interface) y también lógica de negocio. La variedad de clientes estará dada generalmente por el tipo de lenguaje de programación o plataforma de ejecución que posee.

Los denominados lenguajes de programación de cuarta generación o lenguajes visuales, fueron los facilitadores para la construcción de clientes con la funcionalidad mencionada anteriormente.

La Figura 37 muestra algunas variantes tecnológicas para construir componentes clientes que cumplan con el rol definido en un entorno Cliente/Servidor clásico.

Fig. 37. Lenguajes de programación de cuarta generación.

4.3. Cliente/Servidor en la Web con Tecnología JAVA

En el Capítulo 2 en el que se analizó la evolución de los sistemas distribuidos, se mencionó la tecnología Web en 3 capas y se introdujo el concepto de Web Server *dinámico*.

El dinamismo que se busca en un Web Server consiste en lograr que dicho servidor que podría verse como un simple servidor de archivos, sea capaz de desencadenar algún tipo de procesamiento computacional, es decir, ejecutar código de programas.

En ese sentido, esta extensión de funcionalidad que se busca obtener, puede lograrse de modo nativo o a través de la programación de interfaces (tal como sucede siempre que se requiere extender la funcionalidad de un software).

- El modo nativo consiste en modificar el núcleo del software en cuestión dotándolo de las prestaciones que se buscan dentro mismo de su entorno de ejecución.
- El modo por programación de interfaces, consiste en contar con un componente de software que brinde las prestaciones que se buscan lograr (en este caso que el servidor no solo devuelva archivos sino que invoque rutinas para ejecutar código) y escribir las interfaces necesarias para conectar ambos componentes.

La tecnología Java para Web o Tecnología JEE [4.1][4.2] aporta una solución en modo nativo, mientras que la tecnología CGI (que será abordada en el Capítulo 5), aporta una solución por programación de interfaces.

La Figura 38, muestra las distintas variantes para obtener un Web Server “dinámico”.

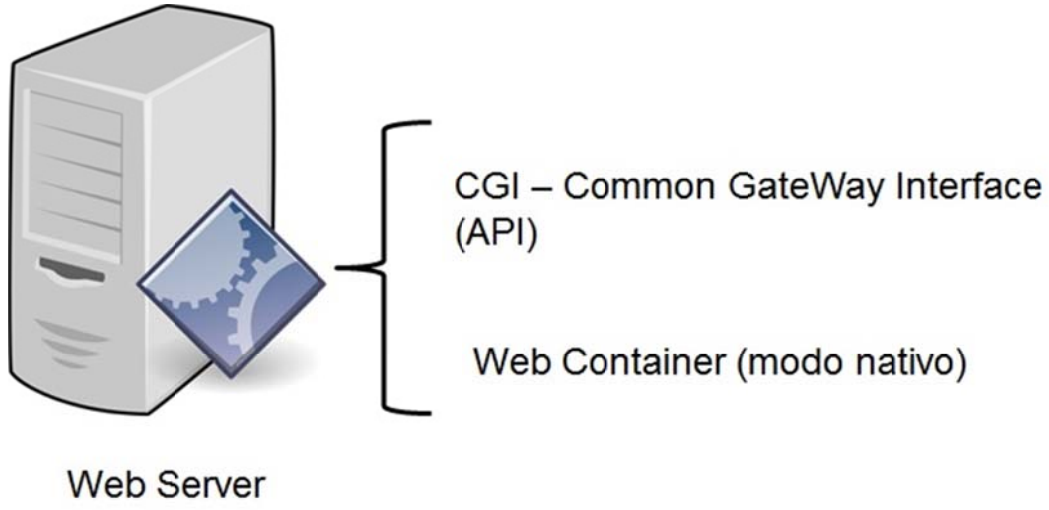

Fig. 38. Variantes de Web Server dinámico

La tecnología basada en Web modo nativo hace uso del concepto de páginas HTML dinámicas.

Una página dinámica se ejecuta dentro del Web Server pero dentro de un componente de la misma que interpreta el código de programación de manera separada del código HTML.

Se considera de modo nativo porque el código no requiere invocaciones a programas externos y no usa una API.

Ejemplos de páginas dinámicas lo constituyen el lenguaje PHP, JSP (basado en JAVA) o ASP (páginas dinámicas en tecnología Microsoft).

En el caso particular de páginas dinámicas en tecnología Java, la arquitectura del sistema se presenta en la Figura 39.

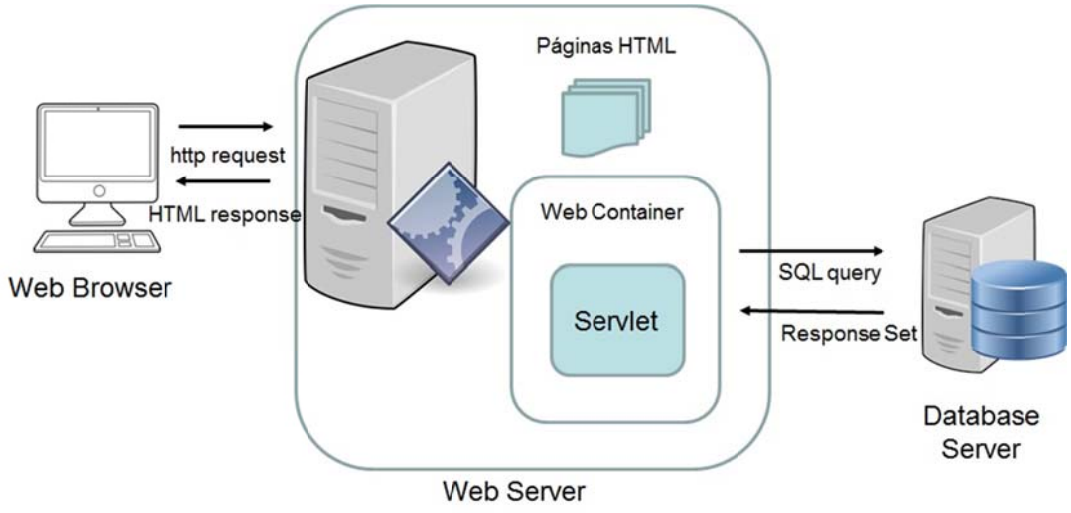

Fig. 38. Tecnología Web modo nativo en JAVA

4.4. Tecnología JEE en el Contexto de Ambientes Distribuidos

Un servlet Java es un programa Java que corre del lado del servidor y se ejecuta en el contexto de un componente del Web Server denominado Web Container.

El Web Container es una máquina virtual Java (JVM) que brinda una implementación de la interface de aplicación del servlet (API). Las instancias de los servlets son componentes que utiliza el Web Container para responder a los requerimientos HTTP.

El componente Web Container puede atender requerimientos que se ejecutan en hilos separados, logrando un mayor rendimiento y escalabilidad. Además proporciona servicios adicionales para seguridad y manejo de errores.

Los Servlets, por su parte, aprovechan las prestaciones de su entorno de ejecución y se comportan como programas servidores robustos y orientados a objetos, que por estar escritos en Java son independientes de la plataforma.

En definitiva, un Servlet es un programa Java que debe generar código HTML dentro de su respuesta, debido a que se ejecuta en el contexto de un Web Server.

Esta característica, obliga a que se mezcle la lógica de la presentación y la lógica de negocio dentro del mismo programa, por esa razón surgen las páginas dinámicas Java o JSP (Java Server Page) que son páginas HTML con etiquetas particulares para incrustar código Java.

El Web Container convierte las páginas JSP en instancias de Servlets.

El siguiente es un ejemplo de código Java que implementa un Servlet para "Hola Mundo"

```
public void generarRespuesta   (HttpServletRequest request, HttpServletResponse response)
Throws IOException {
// Determina el nombre especificado
String name = request.getParameter ("name");
If ( ( name == null ) ||  (name.length() == 0 ) ) {
        name = DEFAULT_NAME;
}
// Especifica que el tipo de contenido es HTML
response.setContentType ("text/html");
PrinterWriter out = response.getWriter () ;
// Generar respuesta HTML
out.println ("<HTML>");
out.println ("<HEAD>");
out.println ("<TITLE>Hola Mundo</TITLE>");
out.println ("</HEAD>");
out.println ("<BODY BGCOLOR = 'white'>");
out.println ("<B>Hello, " + name + "</B>");
out.println ("</BODY>");
out.println ("</HTML>");
out.close();
}
```

Este mismo ejemplo pero construido con páginas JSP sería:

```
<%! private static final String DEFAULT_NAME = "Mundo" ; %>
<HTML>
<HEAD>
<TITLE> Hola Mundo en JSP </TITLE>
</HEAD>
<% -- Determinar el nombre especificado -- %>
<%
        String name = request.getParameter ("name");
        if ( ( name == null ) || (name.length () == 0 ) ) {
                name = DEFAULT_NAME;
        } %>
<BODY BGCOLOR = "white">
<B> Hello, <% =name %> </B>
</BODY>
</HTML>
```

La primera vez que una página JSP es requerida, el Web Container convierte el archivo JSP en un servlet que puede responder a los requerimientos HTTP.

El procesamiento de una página JSP por parte del Web Container realiza tres pasos:

1- En el primer paso el Web Container traduce el archivo JSP en código fuente Java que contiene la definición de una clase Servlet.

2- En el segundo paso, el Web Container compila el código fuente Java en un archivo de clase Java.

3- En el tercer paso, el Web Container crea una instancia de la clase servlet y realiza los pasos del ciclo de vida del mismo invocando un método especial que es jspInit.

La Figura 40 muestra el procesamiento descripto.

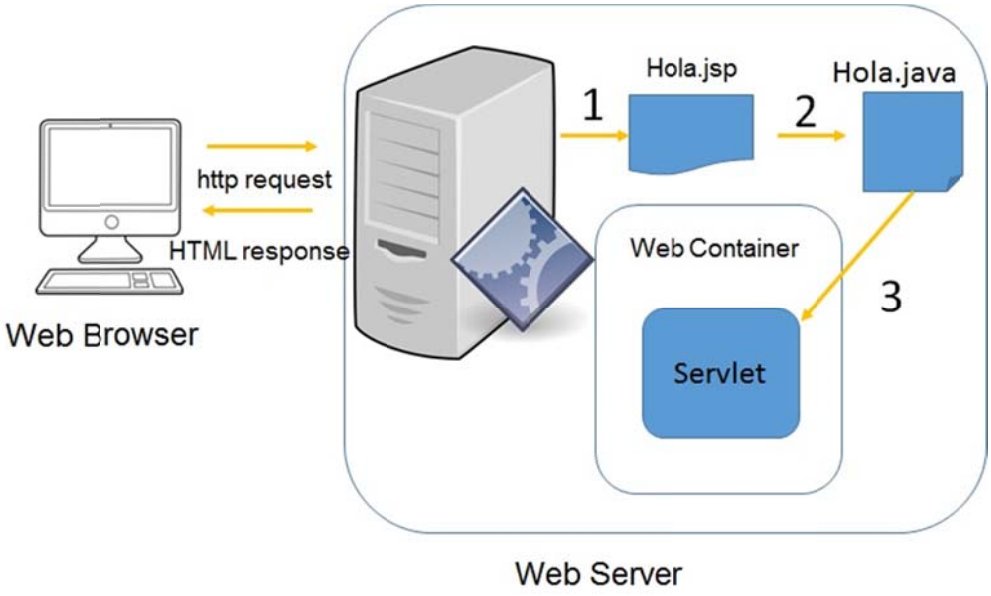

Fig. 40. Procesamiento de una página JSP

4.5. Patrón de Diseño MVC en un Entorno Distribuido

El patrón de diseño MVC - Model, View, Controler, tiene sus orígenes en la programación orientada a objetos y se basa en el concepto que un programa de software debe considerar en su diseño tres componentes:

- Model: los servicios de negocios y objetos del dominio de aplicación
- View: la *ventana* dentro de la aplicación que se presenta al usuario
- Controler: la lógica que acepta las acciones del usuario, realiza la operación y selecciona la próxima View para el usuario.

Este patrón de diseño fue adoptado por la tecnología Web y encontró la mejor forma de ser aplicado, en la tecnología Java.

- El Model es construido con clases Java o con componentes JavaBean.
- El View es construido con páginas JSP
- El Controller se implementa con servlets.

La Figura 41 muestra los tres componentes de MVC aplicado a la tecnología Web.

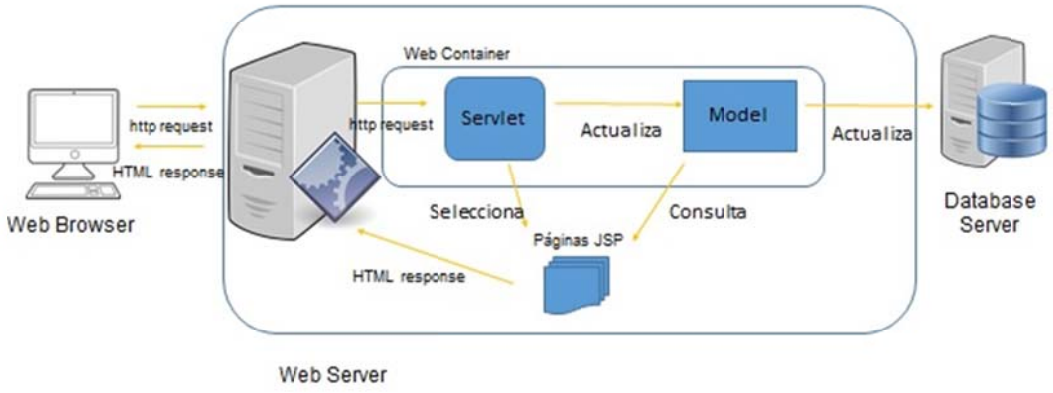

Fig. 41. Patrón MVC Web

Esta arquitectura de tecnología Web implica que la lógica de presentación, la lógica de negocios y la de acceso a los datos, si bien están conceptualmente divididas, son resueltas por el Web Server.

Esto incrementa la carga de trabajo en un único servidor y además hay que tener en cuenta que un Web Server es eficiente en la atención de requerimientos concurrentes, mientras que lógica de negocios se caracteriza por consumir mucha CPU.

Por otra parte, si el sistema ya posee desarrollos con clientes GUI standalone (que se ejecuten fuera del Web Browser), deberán duplicarse la lógica de negocios y acceso a datos, con el correspondiente manejo de transacciones.

Por tal motivo, la tecnología JEE considera un componente de ejecución más, EJB Server, que podemos clasificar como un Servidor de Objetos tal como lo describimos en el Capítulo 2.

EJB es el acrónimo de Enterprise Java Beans, siendo un bean un elemento de diseño que establece ciertas pautas para definir una clase Java, como por ejemplo:

- Las propiedades se definen con métodos para accederlas y modificarlas (llamadas comúnmente *seters* y *geters*
- Posee un método constructor sin argumentos que permite instanciar objetos de esa clase.
- Las variables de instancia no son públicas, es decir, sólo se acceden y modifican por los *seters* y *geters*.

La figura 42 muestra la arquitectura de un sistema distribuido con tecnología JEE.

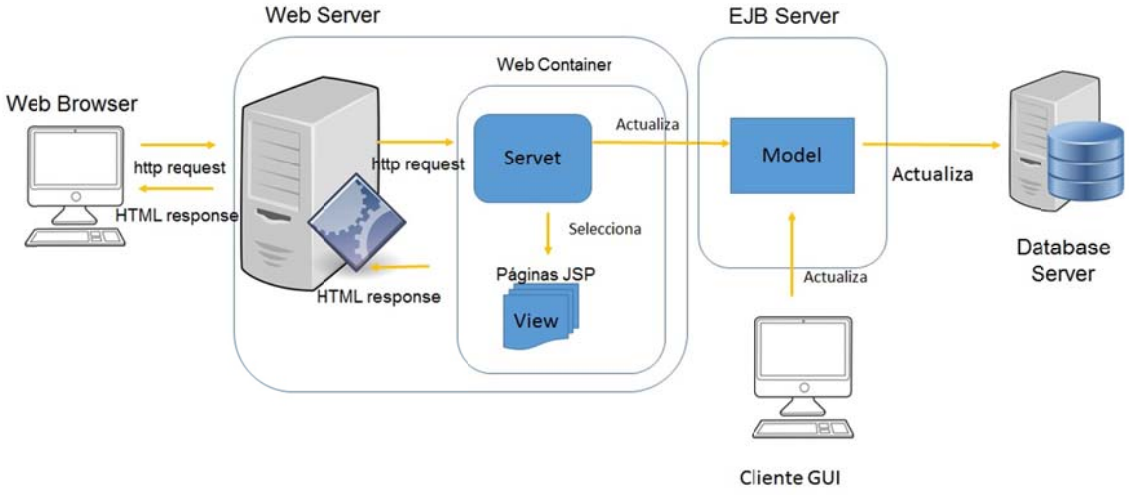

Fig. 42. Arquitectura Distribuida en 3 niveles con JEE

4.6. Conclusiones del Capítulo

En este capítulo se hace hincapié en los componentes servidor y cliente en los ambientes distribuidos, detallando sus principales características y funcionalidades.

En este sentido se analiza también la tecnología Web nativa como un caso de distribución Cliente/Servidor donde la capa media adquiere mayor relevancia y comportamiento.

Se detalla la tecnología Web en modo nativo, como un ejemplo de aplicación distribuida que también convive con Cliente/Servidor clásico y que expone los detalles de varios tipos de servidores (de archivos, de objetos y de base de datos).