Apuntes

de

J2EE

Tema 1: Introducción

Uploaded by

Ingteleco

http://ingteleco.webcindario.com
ingtelecoweb@hotmail.com

La dirección URL puede sufrir modificaciones en el futuro. Si no funciona contacta por email

1.- INTRODUCCIÓN A LA COMPUTACIÓN DISTRIBUIDA

1.1.- ¿QUÉ ES LA COMPUTACIÓN DISTRIBUIDA?

La computación distribuida es un término que hace referencia al reparto de la ejecución de procesos entre los nodos de una red. La computación distribuida implica la existencia de un conjunto de procesadores distribuidos a lo largo de una red que ejecutan procesos pertenecientes a una o varias aplicaciones.

Así pues, una aplicación distribuida será aquella cuyos procesos se encuentren repartidos a lo largo de una red de computadores, de tal forma que la ejecución de la aplicación se lleve a cabo de forma distribuida por varios de los computadores de la red.

Esta asignatura se centrará en el desarrollo de aplicaciones distribuidas y más específicamente en un tipo concreto de estas: las aplicaciones web. Estudiaremos la arquitectura Cliente-Servidor como arquitectura base para el desarrollo de este tipo de aplicaciones, así como algunas de las tecnologías más empleadas en el desarrollo de aplicaciones web.

1.2.- ARQUITECTURA CLIENTE-SERVIDOR

1.2.1.- Clientes y Servidores

Existen múltiples definiciones para hacer referencia al termino Cliente-Servidor, a continuación se presenta una de ellas:

- **Cliente-Servidor:** "Término que se usa para describir las relaciones entre procesos proveedores de servicios (servidores) y procesos consumidores de servicios (clientes)"

Es importante tener en cuenta que la definición dada hace referencia a un modelo de computación lógico, es decir, a procesos. Es también muy común pero ambiguo usar la misma terminología para hacer referencia a las máquinas donde residen los procesos cliente y servidor.

Las arquitecturas Cliente-Servidor constituyen la base dominante en los sistemas distribuidos hoy en día. Permiten que la funcionalidad de las aplicaciones se divida entre dos elementos claramente distinguibles:

- Los **clientes**, que llevan a cabo todo el proceso de interacción con el usuario, inician los procesos de comunicación con los servidores y realizan las operaciones adecuadas para solicitarles a éstos ciertos servicios presentando al usuario los resultados de la ejecución de los mismos. La mayor parte del código se encuentra dedicado a la visualización de la información al usuario.

- Los **servidores**, que escuchan solicitudes de conexión de clientes e implementan todos los servicios que son capaces de proporcionar a los mismos, devolviéndoles los

resultados de su ejecución. Generalmente no disponen de interfaz gráfica pero sí van acompañados de programas que facilitan su gestión, administración y que permiten llevar a cabo estudios sobre el rendimiento del servidor.

Como ya se ha comentado, muchas veces se denominan de forma ambigua servidores tanto a las aplicaciones que prestan estos servicios como a las máquinas dónde éstas residen.

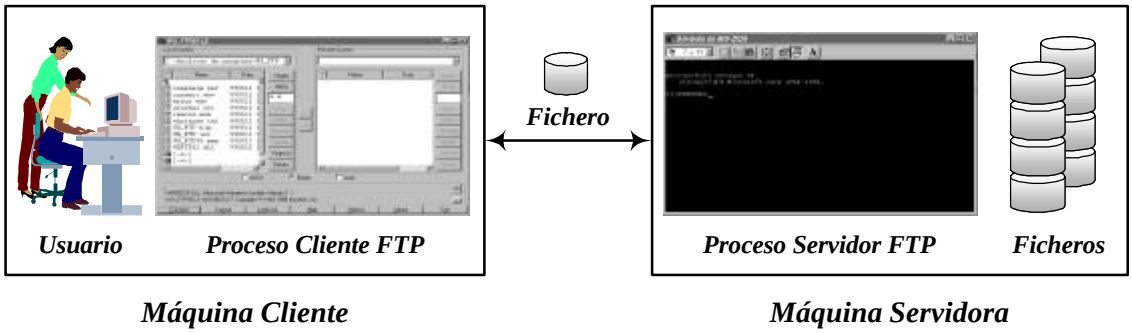


Figura 1.1: Clientes y Servidores involucrados en una transferencia FTP

La existencia o no de una separación física entre el cliente y el servidor es un hecho puramente circunstancial, de hecho, ésta distancia puede no existir al ejecutarse sobre la misma máquina ambos procesos.

El tercer elemento que queda por definir en la arquitectura Cliente-Servidor es el ***middleware***, que básicamente es el software de conexión que permite que los diferentes procesos interactúen en una red.

- **Middleware:** Conjunto de servicios independientes del negocio o dominio que permiten que las aplicaciones interactúen a través de una red. En esencia es la capa de software que se ubica entre la red y el software de aplicación, permitiendo la interoperabilidad a pesar de las diferencias entre protocolos de comunicaciones, sistemas operativos, bases de datos, etc.

De manera informal, se suele decir que el middleware es la "-" del término Cliente-Servidor.

1.2.2.- Bloques de la arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor aunque puede variar en ciertos aspectos en función de la naturaleza del problema sobre el que se aplique, siempre mantiene constantes tres elementos básicos: el cliente, el servidor y el middleware.

1.2.2.1.- El Cliente

Los clientes son responsables del aspecto de presentación de la información que proporciona el servidor. El contenido del cliente puede variar desde una interfaz de usuario sencilla hasta una aplicación completa. Sus características son las siguientes:

- Interactúa con un usuario a través de una interfaz.

- Realiza funciones pertenecientes a la aplicación, si ha lugar.
- Interactúa con el middleware cliente que le pone en contacto con el servidor.
- Recibe respuestas del middleware servidor y si es necesario, las visualiza a través de la interfaz de usuario.

Puesto que su única misión es permitir al usuario la comunicación con el servidor de la forma más cómoda y productiva posible, las distinciones entre los diferentes tipos de clientes se basan en el modo en que el usuario interactúa con ellos. De este modo es posible subdividirlos en:

1. **Clientes sin interfaz gráfica:** se caracterizan por una mínima interacción con el usuario como cajeros automáticos, teléfonos móviles, faxes o robots.
2. **Clientes con interfaz gráfica:** permiten realizar los procesos de comunicación y de visualización de resultados de forma más amigable y su manipulación es más sencilla para el usuario.
3. **Clientes con interfaz orientada a objetos:** utiliza técnicas de manipulación directa sobre objetos representados en forma icónica, para llevar a cabo las tareas deseadas. Por ejemplo, arrastrar un icono de una ventana a otra puede suponer la transferencia de la información representada mediante dicho icono entre cliente y servidor.

1.2.2.2.- El Servidor

La labor de los servidores es atender a múltiples clientes que desean acceder a los servicios proporcionados por el servidor. Las actividades que durante su vida un servidor lleva a cabo son:

- **Espera conexiones de clientes**, aguardando de forma pasiva a que dichas conexiones se produzcan.
- **Ejecuta muchas peticiones de forma simultánea**.
- **Ofrece servicios a los clientes.** Ofrece desde funciones simples como la fecha y la hora hasta sofisticadas funciones de aplicación como el procesamiento de pedidos o la transferencia de fondos.
- **Prioriza a los clientes según su importancia**, puesto que algunas peticiones de servicio son más importantes que otras según diversos factores como origen de la petición (quién la solicitó) o naturaleza de la misma (de qué se trata).
- **Se mantiene funcionando sin detenerse**. Un servidor debe ser robusto y no fallar nunca, puesto que la detención del servicio puede afectar a decenas o cientos de usuarios, que se verán impedidos de llevar a cabo sus actividades. Por esta razón, suelen existir servidores de respaldo mantienen una réplica de los datos y sustituyen a los principales en caso de que éstos fallen.

Dependiendo de la naturaleza de los servicios que ofrecen a los clientes, los servidores se pueden clasificar en los siguientes grupos: Servidores de Ficheros, Servidores de Bases de Datos,

Servidores de Transacciones, Servidores de Trabajo en Grupo, Servidores de Objetos, Servidores Web, etc.

1.2.2.3.- El Middleware

El middleware es la infraestructura que descansa en el espacio comprendido entre cliente y servidor. Parte de estos recursos se encuentran en el lado del cliente y parte en el lado del servidor y aunque no están directamente relacionados con los servicios o tareas a realizar (es decir, son independientes del dominio o *business unaware*), son imprescindibles para la consecución de los mismos puesto que proporcionan todas las facilidades necesarias para la correcta transmisión de la información entre el proceso cliente y el servidor.

1.2.3.- El peso de los clientes y los servidores

En general, se suele hablar de clientes ligeros y servidores pesados o de clientes pesados y servidores ligeros, para expresar la cantidad de código invertida en cada uno de los extremos de una arquitectura Cliente-Servidor.

En un principio la mayor carga de código se encontraba en la parte cliente, en la que las aplicaciones llevaban codificadas todas las operaciones necesarias para que el usuario llevase a cabo su tarea, siendo los servidores bastantes simples y capaces de realizar procesos sencillos de forma repetitiva. De este modo, el buscar cierta información de entre un conjunto de datos era tarea del programa cliente, que solicitaba al servidor dicho conjunto de datos y una vez recibido, procedía a su análisis, mostrando los resultados obtenidos.

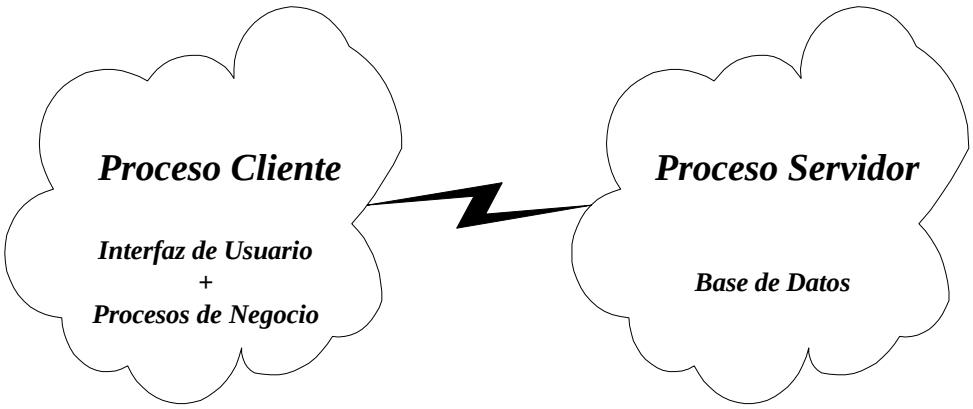


Figura1.2: Configuración de cliente pesado y servidor ligero

Evidentemente este enfoque supone un problema a la hora de llevar a cabo alguna modificación, puesto que al encontrarse en la parte cliente, es necesario sustituir por completo o modificar toda la aplicación cliente para ajustarla a los cambios, y ya que ésta se encontrará distribuida a lo largo de toda una organización (que puede extenderse por varios edificios), un pequeño cambio supone una incidencia en decenas o cientos de estaciones de trabajo, cuyos usuarios se ven afectados en su trabajo.

El modelo más fácil de gestionar es el de clientes ligeros y servidores pesados. Con este enfoque toda la carga de proceso descansa del lado del servidor, que es capaz de llevar a cabo operaciones

con un grado de refinamiento muy elevado, muy parametrizables y flexibles para poder acomodarse a un gran abanico de usuarios distintos.

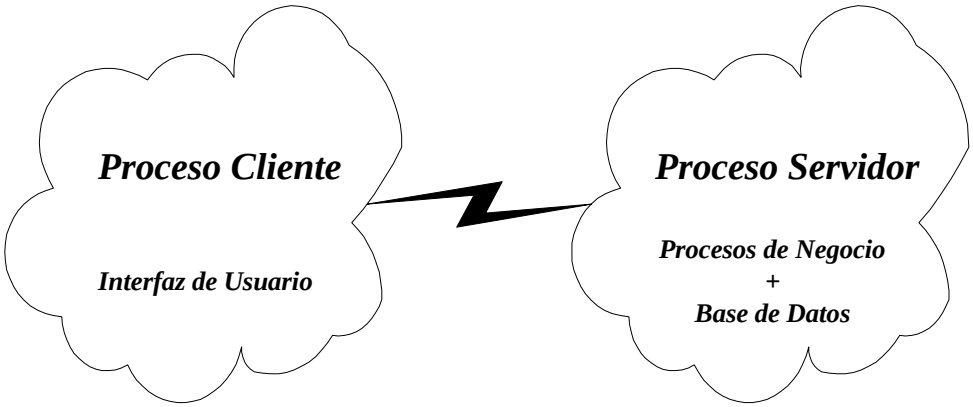


Figura 1.3: Configuración de cliente ligero y servidor pesado

Si es necesario algún tipo de modificación o ampliación en algún servicio, el mantenimiento se lleva a cabo en el servidor, una sola vez, sin afectar las estaciones de trabajo de los usuarios que lo utilizan, lo que redunda en un ahorro de tiempo y una mayor limpieza de los cambios. Por ejemplo, una aplicación web en la que al introducir una frase en un cuadro de texto cuente las letras pero no los signos de puntuación puede fácilmente modificarse para que acepte estos últimos, sin que por ello los usuarios tengan que cambiar el navegador.

1.3.- ARQUITECTURA CLIENTE-SERVIDOR DE 3 CAPAS

Una vez comentados los elementos que constituyen un entorno Cliente-Servidor, un concepto clave que conviene tener en cuenta es el de *Arquitectura de Aplicación*.

- **Arquitectura de Aplicación:** estructura de los componentes en que se divide la aplicación, sus interrelaciones y los principios y directrices que gobiernan su diseño y evolución en el tiempo.

La idea básica es que se debe dividir una aplicación en *unidades funcionales o capas lógicas* que se puedan asignar posteriormente a la máquina cliente y/o a una o varias máquinas servidoras. De esta forma, tendremos una nueva forma de clasificar las arquitecturas Cliente-Servidor que será en función del número de capas de que consta la arquitectura.

La arquitectura Cliente-Servidor básica (la vista hasta este momento) es una **arquitectura Cliente-Servidor de dos capas**. Las aplicaciones basadas en esta arquitectura quedan divididas en dos capas lógicas (proceso cliente y proceso servidor), con tres alternativas posibles:

- Cliente pesados que acceden a bases de datos en las que no se lleva a cabo ningún proceso, recayendo la responsabilidad de los mismos en el cliente.

- Cliente ligeros que invocan procedimientos almacenados en la base de datos, que llevaban a cabo procesos relacionados con la lógica de negocio de la organización.

- Clientes pesados en los que se lleva a cabo parte de los procesos y bases de datos con procedimientos almacenados en las que se lleva a cabo otra parte de los mismos.

Sin embargo, en la actualidad la arquitectura en dos capas esta comenzando a quedarse obsoleta y cada vez más se opta por usar arquitecturas Cliente-Servidor con un mayor número de capas lógicas. La arquitectura Cliente-Servidor más común en la actualidad es la **arquitectura de tres capas** que consiste en dividir la aplicación en tres capas o unidades funcionales. Las tres unidades funcionales más comunes, también llamadas *niveles lógicos de aplicación*, son:

1. La capa de Presentación o de interfaz de usuario.
2. La capa de Lógica de Negocio o de Aplicación.
3. La capa de Datos.

Con el modelo de tres capas, toda la lógica de negocio reside en una capa intermedia entre las aplicaciones de usuario y los almacenes de datos, es decir, entre la interfaz de usuario y la base de datos.

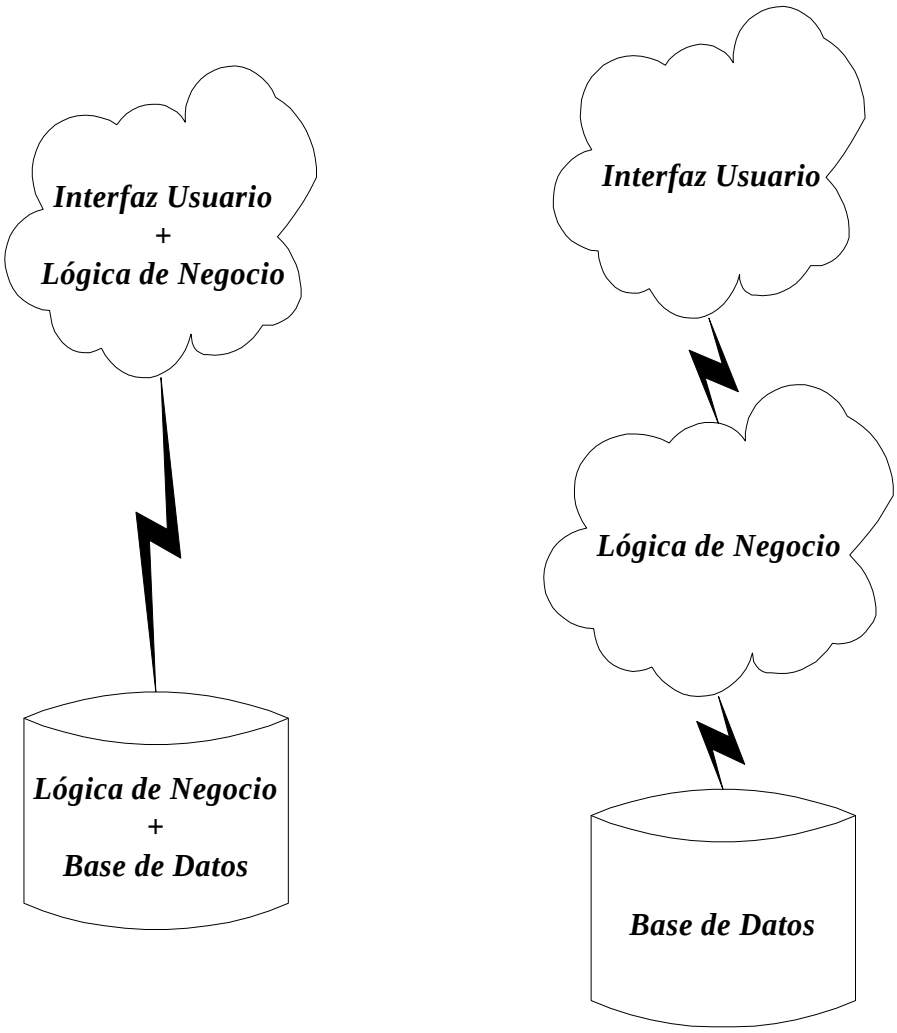


Figura 1.4: Arquitecturas Cliente-Servidor de 2 y 3 capas

Los tres elementos, interfaz, lógica de negocio y bases de datos son independientes entre sí y pueden alterarse sin afectar al resto. Por tanto, forman arquitecturas más robustas, flexibles y escalables mucho más fáciles de mantener y con más posibilidades de distribución. En cualquier caso es posible que el número de capas considerado como ideal vaya incrementándose con el paso del tiempo y dentro de unos años pase a ser 4 o 5.

Dependiendo de la asignación de dichas **unidades funcionales o capas lógicas** a diferentes **configuraciones o capas físicas**, se obtienen distintos niveles de distribución, que da lugar a **arquitecturas de 1, 2, 3 o n capas físicas**.

1.3.1.- Arquitectura de 1 capa física

Las tres unidades funcionales se ejecutan en una única máquina. Esta configuración es válida por ejemplo cuando tenemos un ordenador personal con una pequeña aplicación.

1.3.2.- Arquitectura de 2 capas físicas

Se basa en la asignación de las tres unidades funcionales a dos máquinas: la máquina cliente y la máquina servidor. Esta configuración da lugar a distintas posibilidades:

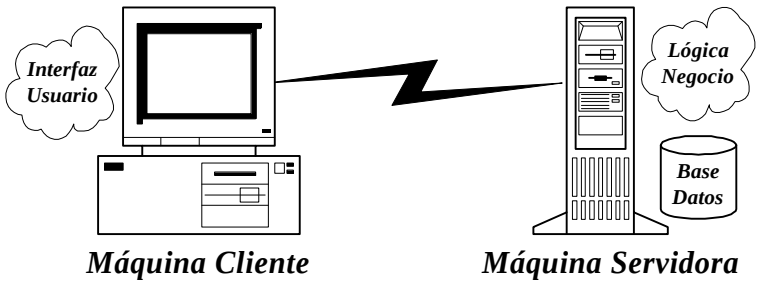


Figura 1.5: Configuración de cliente ligero y servidor pesado.

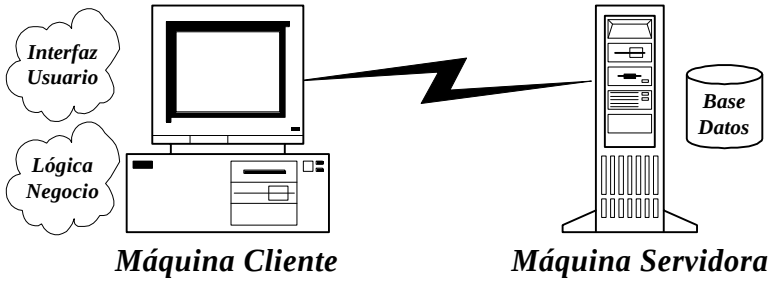


Figura 1.6: Configuración de cliente pesado y servidor ligero.

Como puede observarse la cantidad de código a ejecutarse en cada uno de los extremos de la arquitectura Cliente-Servidor (máquina cliente y máquina servidor) es variable. Esto da lugar, una vez más, a que hablemos de *clientes ligeros y servidores pesados* o de *clientes pesados y servidores ligeros*, pero esta vez desde el punto de vista de las máquinas donde se ejecutan los procesos.

1.3.3.- Arquitectura de 3 capas físicas

Es la arquitectura en expansión en la actualidad ya que cumple los requisitos de las aplicaciones Internet a gran escala. Se basa en la asignación de las tres unidades funcionales a una capa física distinta. Esto da lugar a una arquitectura escalable, robusta y flexible que hace que las aplicaciones sean más fáciles de gestionar y distribuir al estar basadas en clientes ligeros.

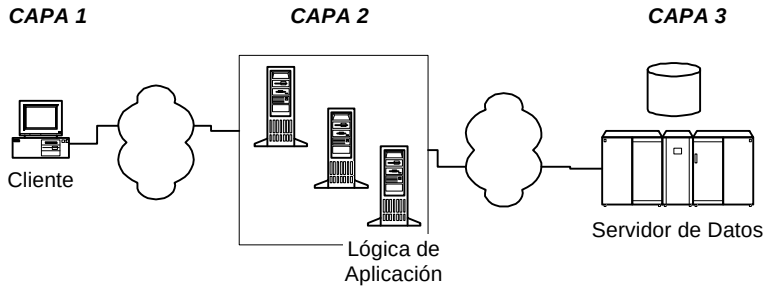


Figura 1.7 Arquitectura Cliente-Servidor de 3 o N capas.

La *capa de lógica de aplicación* en la mayoría de los casos no se implementa como una aplicación monolítica, sino como una colección de *componentes*, cada uno de los cuales automatiza una función de negocio relativamente pequeña. Así, los clientes consiguen realizar un proceso de negocio mediante la combinación de las funciones de estos componentes de la capa intermedia. Un componente puede llamar a otro para cumplir con una petición e incluso puede comunicarse con aplicaciones heredadas que se ejecutan en un mainframe. Por tanto, en la mayor parte de los casos la capa de lógica de aplicación se divide a su vez en más capas dando lugar a arquitecturas de N capas (en la Figura 1.7 se puede observar como la capa 2 se encuentra distribuida en varios servidores).

1.4.- ARQUITECTURA DE LAS APLICACIONES WEB

1.4.1.- Arquitectura básica de servidor web

Con el auge de Internet, la arquitectura Cliente-Servidor ha adquirido una mayor relevancia, ya que ésta ya no sólo es la base de la mayor parte de las aplicaciones que se están desarrollando actualmente sino que también es el principio básico de funcionamiento de la *World Wide Web* y de todas las aplicaciones web.

El funcionamiento tradicional de la web consiste en un cliente web (browser) que, a través del protocolo HTTP, realiza peticiones de paginas HTML a un servidor web que se las sirve. La labor del cliente web será visualizar el contenido de las páginas HTML servidas por el servidor web. El siguiente gráfico muestra este funcionamiento:

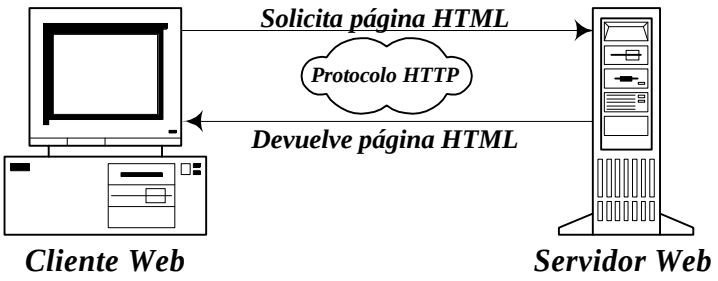


Figura 1.8: Funcionamiento tradicional de la web.

Se puede apreciar, por tanto, que el funcionamiento tradicional de la Web se basa en una arquitectura Cliente-Servidor donde destacan dos elementos: el navegador y el servidor web.

1.4.1.1.- Navegador o Browser

El navegador o browser es el cliente web por excelencia. Tradicionalmente soporta dos funciones básicas:

- Utilizar el protocolo HTTP para acceder al servidor web y solicitar la página HTML o recurso deseado.
- Visualizar la página HTML o recurso al usuario.

Atendiendo a estas funciones se puede apreciar como el navegador es el ejemplo claro de cliente ligero ya que su única función es visualizar el contenido de las páginas web que le sirve el servidor, estando liberado de realizar cualquier proceso de negocio.

Los navegadores más populares son el Microsoft Internet Explorer y el Netscape Navigator. Ambos tienen una funcionalidad muy similar y son completamente gratuitos.

1.4.1.2.- Servidor web

La función tradicional del servidor web es servir las páginas HTML que le solicita el navegador, comportándose en este sentido como un mero servidor de ficheros HTML. Sin embargo, el servidor web no permite acceder al cliente indiscriminadamente a todos sus ficheros HTML sino únicamente a determinados directorios y documentos previamente establecidos por el administrador de dicho servidor.

Actualmente y debido a las necesidades crecientes de las aplicaciones web, los servidores web soportan funciones adicionales a la de servir ficheros HTML como, por ejemplo, la ejecución de procesos como se verá más adelante.

1.4.2.- Tendencias actuales para las aplicaciones web

Como ya se ha comentado, en su concepción tradicional los servidores web se limitaban a enviar una página HTML cuando el navegador la solicitaba. Esto permita que podamos diseñar un gran

número de paginas HTML que proporcionen información a los usuarios y a las cuales puedan acceder de forma universal, esto es, desde cualquier sitio y en cualquier momento.

Sin embargo, este enfoque tiene una limitación insalvable: **su naturaleza estática**. Una página HTML no varia desde que el diseñador la maqueta y la deja en la web, a menos que el propio diseñador la actualice manualmente cada cierto tiempo. De la misma forma, la interacción con el usuario es mínima ya que éste puede solicitar páginas estáticas al servidor pero no enviar datos al servidor para que sean almacenados ni recibir una respuesta personalizada de éste en forma de página HTML.

Ejemplo: supongamos que una tienda de deportes permite que sus clientes puedan consultar los datos de los productos a través de la web. Si las páginas fuesen HTML estático, deberían modificarse cada vez que los datos de un producto varían o cada vez que se añadan nuevos producto. Ello implica muchas horas de desarrollo de los encargados del mantenimiento de las páginas web.

Para solucionar la naturaleza estática de este enfoque, existen dos alternativas posibles:

- Añadir más inteligencia al cliente
- Añadir más inteligencia al servidor

La forma más extendida de **añadir inteligencia a los clientes** (a las páginas HTML) son **Javascript** y los **applets de Java**. Javascript es un lenguaje relativamente sencillo, interpretado, cuyo código fuente se introduce en la página HTML por medio de las etiquetas <SCRIPT>...</SCRIPT>; su nombre deriva de una cierta similitud sintáctica con Java. Los applets de Java tienen mucha más capacidad de añadir inteligencia a las páginas HTML que se visualizan en el browser ya que son verdaderas clases de Java (ficheros *.class) que se cargan y se ejecutan en el cliente.

Para **añadir inteligencia al servidor** existen distintas tecnologías parecidas entre si: por una parte **CGI/Servlets** y por otra **ASP/JSP**. La característica común de todas ellas es que son procesos que residen y se ejecutan en el servidor web y que son capaces de procesar la información que se envía al servidor y en función de ella generar las páginas HTML que correspondan. Con este nuevo enfoque, en realidad las páginas web estáticas no existen, sino que existen los procesos que las crean dinámicamente. Estas tecnologías serán estudiadas con más detalle más adelante.

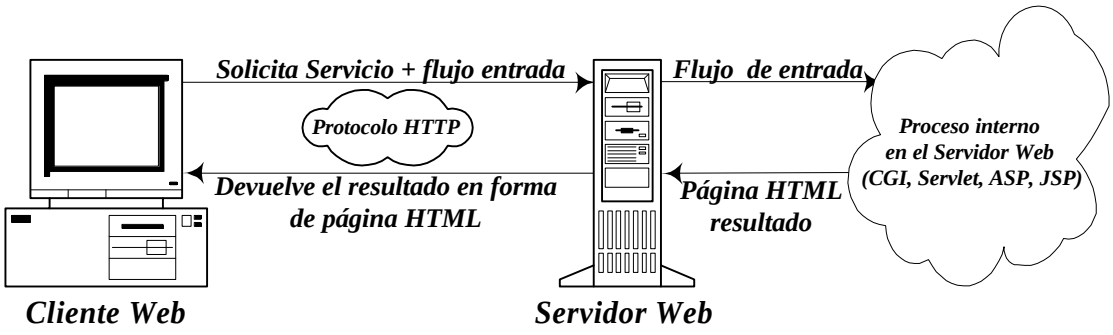


Figura 1.9: Funcionamiento interactivo de la web.

Ejemplo: Una solución al ejemplo anterior consistiría en disponer en el servidor de ciertos procesos capaces de acceder a la base de datos de los productos, extraer de allí la información y generar la página HTML con dicha información cada vez que el usuario la solicita. De la misma forma, el cliente podría solicitar mediante un formulario web que se le visualizasen únicamente aquellos productos que sean "zapatillas deportivas", de forma que el proceso del servidor reciba la información del usuario, acceda a la base de datos a por los datos de las "zapatillas deportivas" y genere una página HTML con esa información. Todo esto resultaba imposible de hacer con el enfoque tradicional de páginas HTML estáticas. La diferencia radica en que ahora las páginas HTML que se le han presentado al usuario no existen originalmente sino que un proceso que se ejecuta en el servidor las ha generado dinámicamente.

De este modo se pueden crear las denominadas **aplicaciones web**: no un mero repositorio de páginas estáticas enlazadas unas con otras, sino procesos dinámicos capaces de procesar información y generar respuestas dinámicamente.

De cara a esta asignatura, tiene mayor importancia las soluciones adoptadas para añadir más inteligencia al servidor.

1.4.3.- Arquitectura web actual: Arquitectura de N capas

La arquitectura de las aplicaciones web actuales ha supuesto una revolución dentro de las arquitecturas Cliente-Servidor dando lugar a la aparición de arquitecturas de N capas.

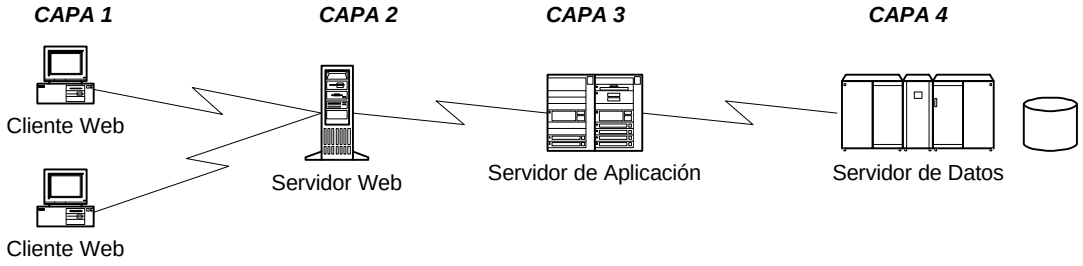


Figura 1.10: Arquitectura Web de 4 capas.

La figura 1.10 muestra una arquitectura Cliente-Servidor de 4 capas adaptada a un entorno web. Las aplicaciones web de este tipo cumplen una serie de características comunes:

- **Acceso universal a la información**, esto es, el acceso a la aplicación se puede realizar desde cualquier lugar y en cualquier momento gracias al servicio web de Internet.
- **Dan servicio a millones de clientes potenciales**; todos aquellos que puedan acceder a la web.
- **Se ejecutan en múltiples servidores heterogéneos**, cumpliendo cada uno de ellos un papel distinto.
- **Admiten interacciones servidor a servidor**.

- **Distribución geográfica mundial**.

1.4.4.- Arquitectura de la Plataforma J2EE

La plataforma J2EE (*Plataforma Java 2 SDK Enterprise Edition*) proporciona un estándar para el desarrollo de aplicaciones distribuidas multicapa (*Enterprise Applications*), dando soporte tanto a la implementación como al despliegue de las mismas. La plataforma J2EE ha sido diseñada de tal forma que permita satisfacer una gran parte de las necesidades que tienen las aplicaciones distribuidas multicapa, tanto en la parte cliente como en la parte servidora.

La arquitectura de este tipo de aplicaciones suele estar formada por una *capa cliente* que proporciona la interfaz de usuario, una o varias *capas intermedias* que proporcionan distintos servicios al cliente como la presentación y la lógica de negocio de la aplicación, y una *capa de datos* que proporciona la gestión y almacenamiento de la información.

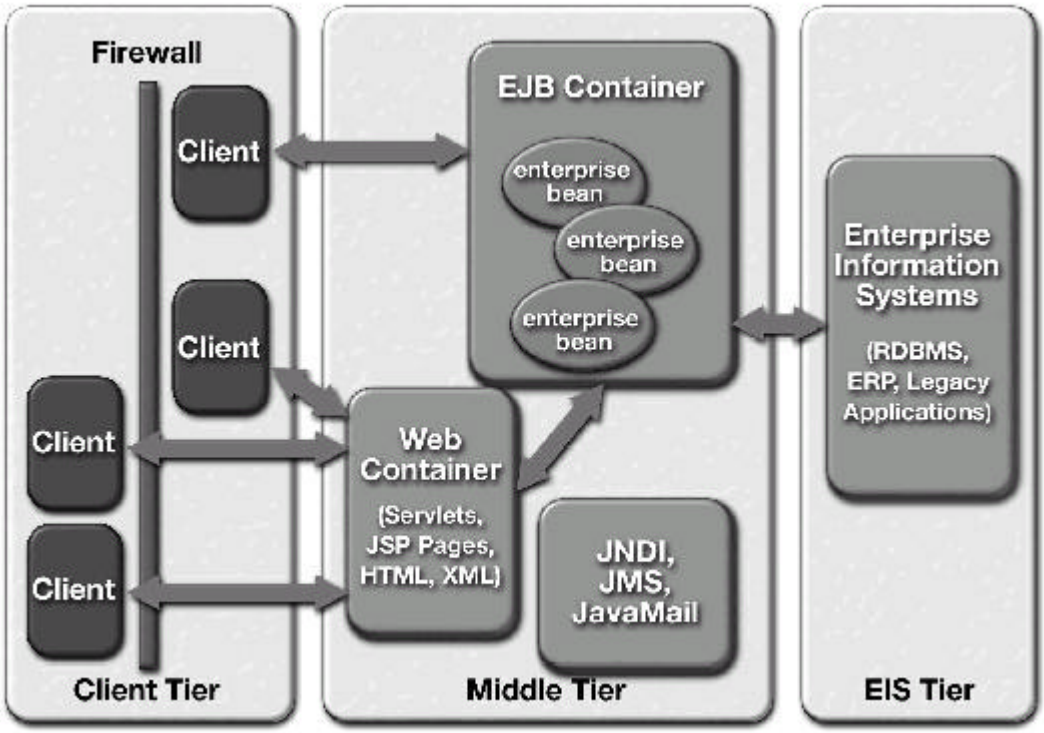


Figura 1.11: Arquitectura de la plataforma J2EE.

Como se muestra en la figura, la plataforma J2EE define un modelo de aplicación distribuida multicapa. Aunque en la figura se muestra una arquitectura en 3 capas, la plataforma J2EE define una capa intermedia que se divide a su vez en dos subcapas (*capa web* y *capa EJB*) con lo que nos encontramos ante una arquitectura Cliente-Servidor de 4 capas.

La ***capa cliente*** soporta clientes de distintos tipos:

- **Browsers:** se encargan de visualizar tanto las páginas HTML estáticas como las páginas HTML dinámicas generadas por los procesos de la capa web.

- **Applets de Java:** aplicaciones incrustadas dentro de páginas HTML que se descargan desde el servidor y se ejecutan en el cliente dentro de un navegador.

- **Aplicaciones:** como cliente de la plataforma J2EE también podemos tener aplicaciones normales cuya principal función será gestionar la interfaz de usuario. Lo habitual, teniendo en cuenta que es la plataforma J2EE es una plataforma Java, es que los clientes sean aplicaciones Java, sin embargo también es posible tener clientes que sean aplicaciones escritas en otro lenguaje distinto como Visual Basic.

Como se puede observar existen dos tipos de clientes claramente diferenciados: los clientes web (browsers y applets) y las aplicaciones normales. El acceso a la capa intermedia (sobre todo en el caso de los clientes web) se realizará por medio de protocolos web, como el protocolo HTTP.

La ***capa intermedia*** se divide a su vez en dos subcapas: *la capa web* que se encarga de todo lo relacionado con la generación dinámica de páginas web para el cliente y *la capa EJB* que se encarga de implementar la lógica de negocio de la aplicación.

La ***capa web*** está formada por componentes web. La plataforma J2EE especifica dos tipos de componentes web: ***Servlets*** y ***JSP***. Ambos tienen como función principal generar la interfaz de usuario para el cliente en forma de páginas web (por ello, también se denomina a la capa web, capa de presentación).

La ***capa EJB*** soporta la lógica de negocio de la aplicación implementada mediante componentes ***Enterprise JavaBeans***. La arquitectura *Enterprise JavaBeans* es una tecnología que se ubica en la parte servidora y que permite el desarrollo y despliegue de componentes que soporten la lógica de negocio de una aplicación distribuida multicapa.

La ***capa de datos*** la constituyen las bases de datos y los sistemas de información externos (aplicaciones heredadas) a los que vayamos a acceder.

1.4.4.1.- Escenarios posibles con la plataforma J2EE

En esta sección se mostrarán un conjunto de escenarios que reflejan las distintas arquitecturas que se pueden adoptar en el desarrollo de aplicaciones con la plataforma J2EE.

1.4.4.1.1.- Arquitectura de 4 capas

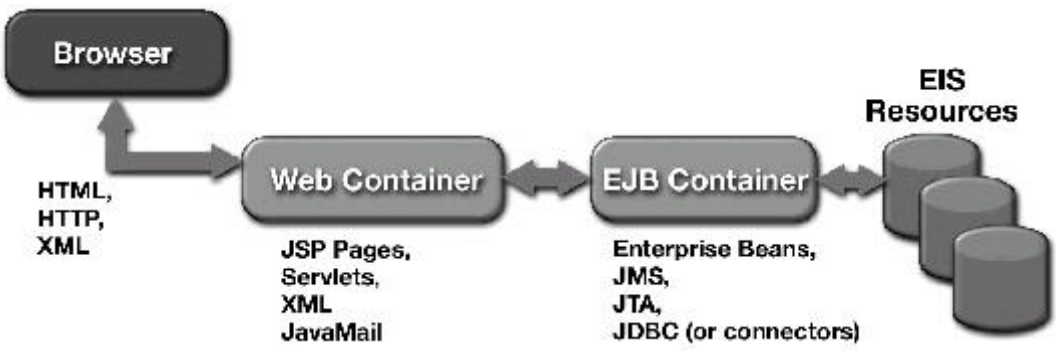


Figura 1.12: Arquitectura base para una aplicación de 4 capas con la plataforma J2EE

La figura 1.12 muestra un escenario en el cual la capa web soporta una serie de componentes web dedicados exclusivamente a la lógica de presentación de la aplicación. La generación dinámica de paginas web será, por tanto, responsabilidad de las paginas JSP (o bien de los Servlets). La capa EJB contiene los componentes EJB que por una parte responden a las peticiones de la capa web y por otra acceden a la capa de datos en búsqueda de información.

Las ventaja de esta arquitectura es que al estar compuesta por un gran número de capas, es fácil de distribuir, fácil de mantener y oculta perfectamente al usuario la implementación real de la aplicación.

La arquitectura propuesta en este escenario es la arquitectura base a la que se pretende llegar en esta asignatura.

1.4.4.1.2.- Arquitecturas con clientes en forma de aplicación

Como posibles escenarios en los que el cliente sea una aplicación podemos destacar tres:

- La aplicación cliente accede directamente a los componentes de la capa EJB y éstos acceden a la capa de datos a través de JDBC. La responsabilidad de la presentación de resultados y la gestión de la interfaz de usuario recaerá sobre la aplicación cliente, mientras que la lógica de negocio recae sobre la capa EJB.

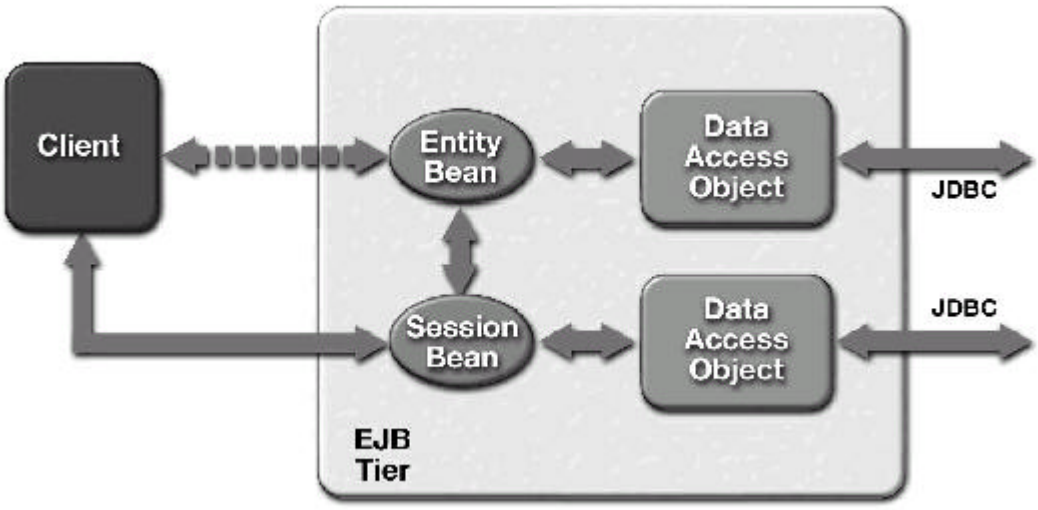


Figura 1.13: La aplicación cliente accede directamente a la capa EJB.

- La aplicación cliente se trata de una aplicación Java que accede directamente a la capa de datos usando JDBC. En este escenario, la lógica de presentación y la lógica de negocio recae sobre la aplicación cliente. La capas intermedias han desaparecido y se han trasladado hasta el cliente, es un caso típico de cliente pesado.

- La aplicación cliente se trata de una aplicación Visual Basic que recibe los contenidos dinámicos generados por la capa web (documentos XML principalmente) y se encarga de presentarlos al usuario. La lógica de presentación recae de nuevo sobre el cliente (ya que recibe XML y se tiene que encargar de presentarlo), mientras que la lógica de

negocio puede recaer sobre la capa web, en caso de que ésta acceda directamente a la capa de datos, o sobre la capa EJB en caso de que ésta exista.

1.4.4.1.3.- Arquitectura de aplicación centrada en la capa web

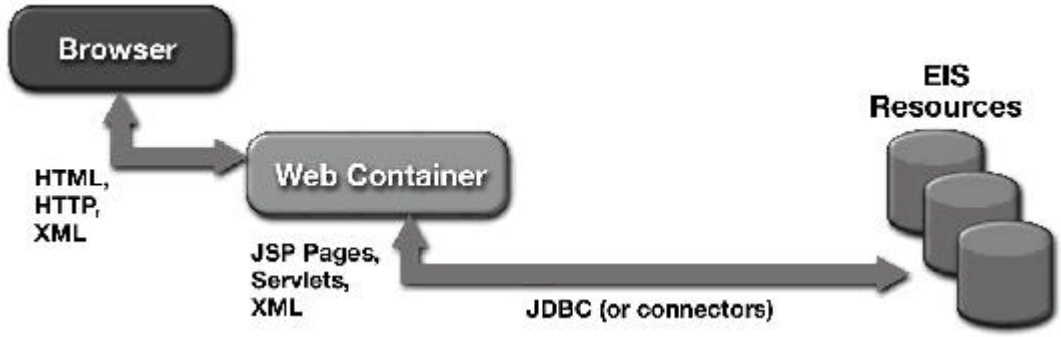


Figura 1.14: Arquitectura de una aplicación centrada en la capa web.

En este escenario la capa web recibe las peticiones de un browser, las procesa, accede a la capa de datos mediante JDBC, genera dinámicamente las páginas web y las devuelve al browser para que éste las visualice al usuario. En este caso, tanto la lógica de presentación como de negocio es soportada por la capa web.

1.5.- TECNOLOGÍAS WEB

1.5.1.- HTTP: Protocolo de Transferencia de HiperTexto.

El protocolo HTTP (*HyperText Transfer Protocol*) se diseñó para permitir a los navegadores la recuperación de documentos de hipertexto desde los servidores donde están almacenados. Puesto que un documento de hipertexto en ultima instancia no es más que un fichero normal y corriente, los navegadores terminan utilizando el protocolo HTTP para solicitar cualquier tipo de fichero al servidor donde se encuentra almacenado. El protocolo HTTP acaba siendo, por tanto, un protocolo para la transferencias de ficheros orientado a la web.

Tanto el nombre del documento, como la ruta de carpetas para acceder a él, como el nombre de la máquina es información contenida en la URL del documento a recuperar.

Ejemplo:

`http://www.eside.deusto.es/asignaturas/LSD/pagina.html`

En esta URL se indica que: a través del protocolo *http*, queremos recuperar un documento llamado *pagina.html* que se encuentra ubicado en una máquina llamada *www.eside.deusto.es* y dentro de esa máquina, en la ruta de carpetas */asignaturas/LSD/*

Los enlaces de hipertexto que aparecen en las páginas web no son más que un texto estático asociado a una URL que indica el documento a recuperar cuando el usuario selecciona ese enlace.

Las peticiones de un cliente HTTP siguen el siguiente formato:

GET *ruta versión* [CABECERA]	Solicita el documento especificado en la *ruta*, que debe empezar por "/" usando la *versión* de HTTP indicada (aunque este campo es opcional). En las líneas siguientes puede llegar información adicional para cursar la petición (como datos de autenticación de usuario). **Ejemplo:** `GET /asignaturas/LSD/pagina.html HTTP/1.0`.
HEAD *ruta versión* [CABECERA]	Solicita información sobre el recurso indicado en la *ruta*.
POST *ruta versión* [CABECERA] [CUERPO]	Solicita que se envíe al recurso especificado por *ruta*, los datos contenidos en la sección cuerpo. Generalmente el recurso destino será un programa que debe procesar los datos y que genera una página web como resultado del procesamiento de los mismos.

El servidor HTTP responde con una línea de estado con un código indicativo del resultado de la solicitud, varias líneas de cabecera con información sobre el recurso pedido y a continuación, una sección de cuerpo con todo el contenido del propio recurso solicitado.

Un navegador es un claro ejemplo de cliente HTTP. Cuando el usuario escribe la dirección de un recurso (URL) en el navegador, éste se pone en contacto con el servidor HTTP indicado en la URL y le realiza la petición del recurso siguiendo el formato del protocolo HTTP (el indicado anteriormente).

El número de puerto por defecto en el que reside el servidor HTTP es el 80.

1.5.2.- HTML: Lenguaje de Etiquetas de HiperTexto

HTML (*HyperText Markup Language*) es el lenguaje empleado para la creación de páginas web (documentos HiperTexto). Las páginas web contienen texto formateado según una serie de reglas. Estas reglas se definen mediante las etiquetas del lenguaje HTML que son una serie de marcas que indican cual es el formato y la estructura de la información contenida en la página HTML.

Las etiquetas HTML se escriben entre ángulos (< y >) y formatean el texto que se encuentra entre la etiqueta de apertura y la de cierre, que es una etiqueta con el mismo nombre que la de apertura pero precedida del símbolo "/".

Ejemplo:

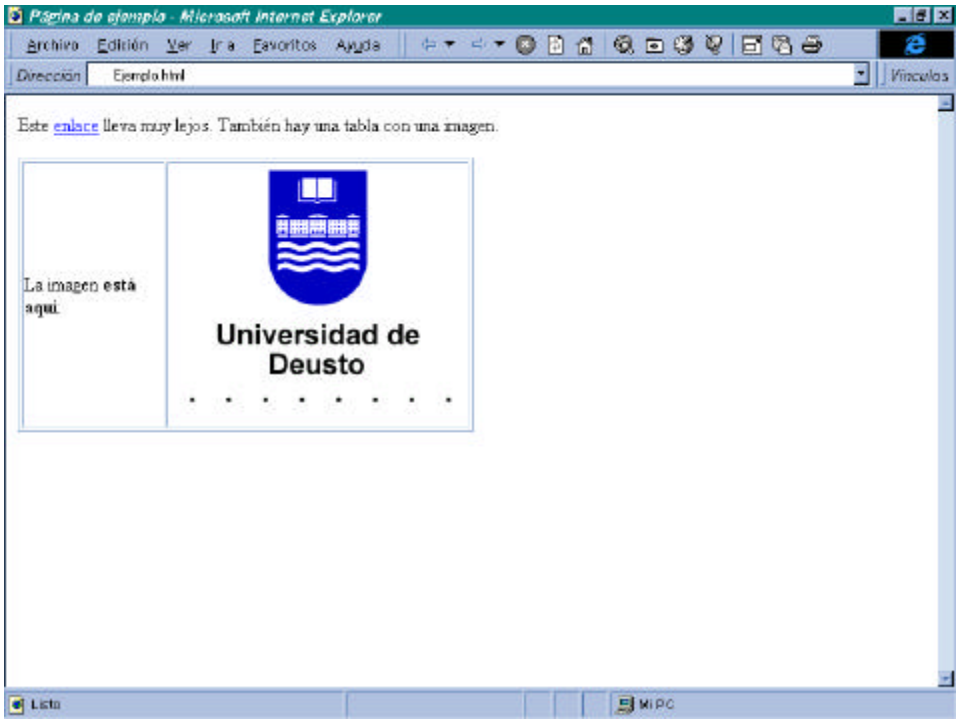
Texto HTML	Texto visualizado
`<I>Hola</I>`	*Hola*

De esta forma se pueden editar documentos HTML, basta con escribir el texto del documento y posteriormente escribir las etiquetas que dan formato y estructura a las distintas partes del texto.

Ejemplo: Este sería un documento HTML sencillo.

```
<HTML>
	<HEAD>
		<TITLE>Página de ejemplo</TITLE>
	</HEAD>
	<BODY>
	<P>
	Este <A HREF="http://www.eside.deusto.es/asignaturas/LSD"> enlace </A>
	lleva muy lejos. También hay una tabla con una imagen.
	</P>
	<TABLE WIDTH="50%" BORDER="1">
	<TR>
		<TD>La imagen <B>está aqui</B>:</TD>
		<TD><IMG SRC="imagenes/ud.gif"></TD>
	</TR>
	</TABLE>
	</BODY>
</HTML>
```

Y se visualizaría de la siguiente forma:



1.5.3.- CGI (Common Gateway Interface)

A partir del estándar HTML 2.0 se incluyeron unas etiquetas que permiten insertar en las páginas HTML unos formularios con la intención de recoger datos del usuario. Estos formularios, una vez completados, se envían al servidor web que a su vez se los pasa a los procesos encargados de su interpretación y ejecución, generando una respuesta en forma de página HTML.

Un CGI (*Common Gateway Interface*) no es más que un programa escrito en cualquier lenguaje de programación que recibe datos a través del dispositivo de lectura estándar y que genera un resultado (generalmente una página HTML) hacia el dispositivo de escritura estándar.

Cuando el servidor web recibe una solicitud de ejecución de un CGI junto con los parámetros necesarios, éste inicia la ejecución del CGI como si de un proceso independiente se tratase, pasándole los parámetros al mismo a través del dispositivo estándar de entrada de datos, es decir, el CGI lee de dicho dispositivo los datos que necesita para realizar su tarea, que previamente habrán sido depositados allí por el servidor web.

A medida que el CGI se va ejecutando va generando los resultados del proceso (una página HTML) y los envía al dispositivo de escritura estándar, cuya salida estará capturada por el servidor web, que la redireccionará hacia el navegador del usuario que solicitó la ejecución del CGI para que éste la visualice.

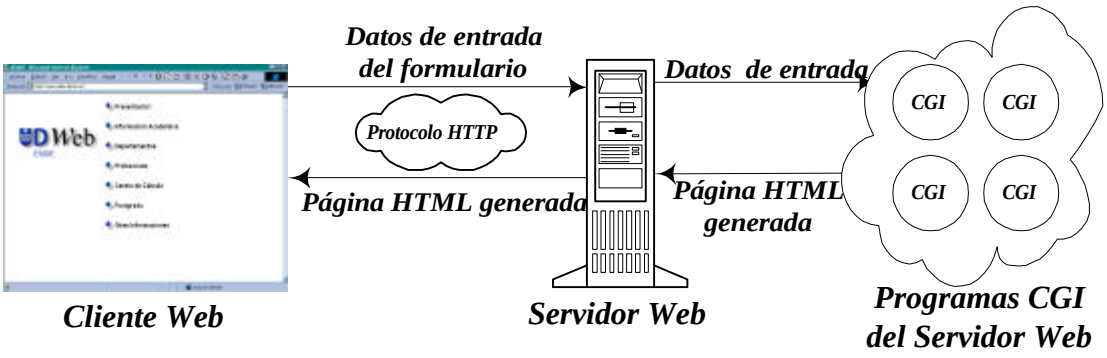


Figura 1.15: Esquema de funcionamiento de los CGI.

El principal problema de los CGI es que se crea un proceso independiente para atender a cada solicitud de los clientes. Esto causa numerosos problemas ya que al crearse procesos independientes es imposible conservar información del estado de una solicitud a otra, además de producirse gran sobrecarga en el caso de que el número de solicitudes sea grande.

1.5.4.- Servlets

La alternativa Java a los CGI son los *Servlets*. Poseen un funcionamiento similar a los CGI ya que su funcionamiento consiste también en recibir una petición del cliente, generar dinámicamente una página web y devolverla al cliente, pero en este caso pudiendo utilizar toda la potencia de la plataforma Java respecto a bibliotecas de clases, de tal modo, que se puede utilizar un *Servlet* para recoger los datos de un usuario en un formulario, y enviarlos por correo electrónico gracias a las clases de email que proporciona Java.

Los *Servlets* solucionan los problemas que planteaban los CGI ya que con ellos no se crea un proceso independiente para cada petición sino que se comparte un mismo *Servlet* entre todos los clientes con lo cual la sobrecarga desaparece y además es posible mantener el estado de una petición a otra.

Ejemplo: se presenta el código de un *Servlet* que recibe como parámetro un nombre y genera una página web con un mensaje de bienvenida personalizado.

```
public class ServletBienvenida extends HttpServlet {

// Procesar la solicitud POST de HTTP
```

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

    response.setContentType("text/html");
    PrintWriter out = new PrintWriter (response.getOutputStream());
    out.println("<html>");
    out.println("<head><title>ServletBienvenida</title></head>");
    out.println("<body>");

    // Recoger el valor del campo nombre
    String nombre =  request.getParameter("Nombre");

    // Mostrar el mensaje de bienvenida
    out.println("Bienvenido a nuestra web, " + nombre + "<BR>");
    out.println("</body></html>");
    out.close();
  }
}
```

1.5.5.- ASP (Active Server Pages)

ASP (*Active Server Pages*) es una tecnología desarrollada por *Microsoft* para diseñar e implementar procesos complejos, haciéndolos accesibles mediante web. La principal innovación de las ASP respecto a otras tecnologías como CGI consiste en que tanto el formato de la información en HTML como el código del proceso a ejecutar residen en la misma entidad: la página ASP.

Ya no es necesario disponer por una parte de páginas HTML estáticas y por otra parte de programas CGI o Servlets, sino que con sólo un elemento, la página ASP, es suficiente para contener ambas naturalezas y funciones.

Un ejemplo de página ASP sencilla podría ser el siguiente:

```
<HTML>
      <HEAD>
            <TITLE>Este es un ejemplo</TITLE>
      </HEAD>
      <BODY>
            Estos son los números del 1 al 5:
            <% For n = 1 To 5 %>
            Número <% Response.Write(n) %> <BR>
            <% Next n%>
      </BODY>
</HTML>
```

ejemplo.asp

Como se puede ver, un documento ASP es básicamente un documento HTML al que se le han añadido unas etiquetas especiales (<% y %>) que indican el principio y el fin del código ASP; código que se procesa en el servidor antes de servir la página al cliente que se la ha solicitado.

Esquemáticamente el proceso sería:

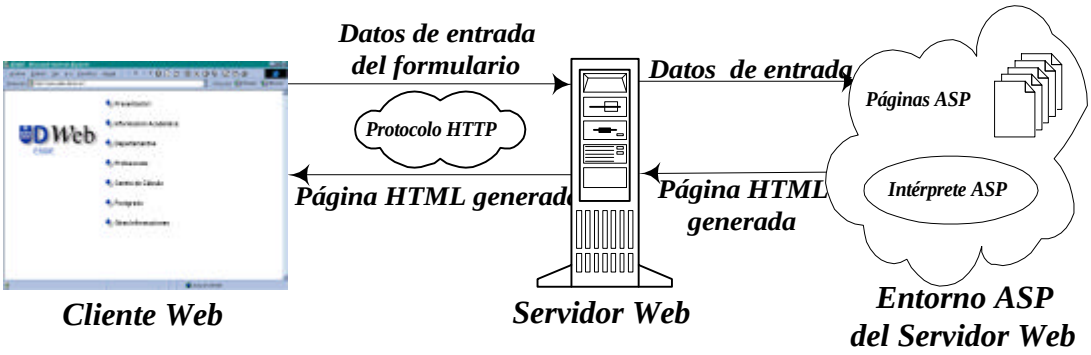


Figura 1.16: Esquema de funcionamiento de los ASP.

Cuando un cliente HTTP solicita esta página a un servidor con soporte ASP, éste la abre antes de servirla e interpreta cada línea de la página mediante las siguientes reglas:

1. Mientras no se encuentre la etiqueta <% el texto de la página se sirve tal cual. (código HTML estático)

2. Cuando se encuentra la etiqueta <% de inicio de código ASP, el intérprete de ASP ubicado en el servidor web empieza a ejecutar el código ASP, escrito en un lenguaje de programación como *VBScript* o *JScript*. Durante esta ejecución se puede generar más código HTML que será también servido al cliente (código HTML dinámico).

3. Cuando se encuentra la etiqueta %> de fin de código ASP, se vuelve al paso 1, hasta acabar de procesar toda la página.

Como es posible apreciar, el servidor web contiene un intérprete ASP que es el encargado de llevar a cabo la ejecución del código ASP. Actualmente sólo los servidores web de *Microsoft* (el *Personal Web Server* y el *Internet Information Server*) incorporan dicho intérprete, por lo que el uso de las ASP determina la elección del servidor web.

Como resultado del ejemplo, un cliente que hubiese solicitado la página `ejemplo.asp` al servidor, hubiese recibido como respuesta:

```
<HTML>
	<HEAD>
		<TITLE>Este es un ejemplo </TITLE>
	</HEAD>
	<BODY>
		Estos son los números del 1 al 5:

		Número 1<BR>
		Número 2<BR>
		Número 3<BR>
		Número 4<BR>
		Número 5<BR>
	</BODY>
</HTML>
```

Pagina HTML generada como resultado de la ejecución de la página ejemplo.asp

Al cliente siempre le llega el resultado de la ejecución de la página ASP en el servidor, y ese resultado siempre es íntegramente código HTML, de tal modo, que no es posible analizar la página recibida y determinar qué parte de la misma era HTML estático en su origen y qué parte se ha generado de forma dinámica. Por supuesto las páginas ASP también pueden recibir los datos que el usuario envía a través de un formulario.

La tecnología ASP es muy flexible y potente, permitiendo que las páginas ASP puedan acceder a bases de datos, recuperar ciertos valores de las mismas y generar dinámicamente documentos HTML que muestran el contenido de la base de datos. Asimismo, las páginas ASP se integran perfectamente con el resto de tecnologías *Microsoft*, permitiendo vincular el código ASP con el sistema de correo electrónico (para enviar e-mails como resultado del procesamiento de una página), con objetos COM que representan los procesos de negocio de una organización (para permitir que el código ASP pueda acceder, por ejemplo, al sistema de facturación de la empresa y poder facturar por web, obteniendo el número de tarjeta del cliente).

1.5.6.- JSP (Java Server Pages)

Básicamente la tecnología y el funcionamiento de las JSP es la misma que la de las ASP, la diferencia radica en que mientras la última ha sido desarrollada por *Microsoft* y se integra perfectamente con el resto de tecnologías de la corporación norteamericana, las JSP se han desarrollado con la vista fija en la plataforma Java, y se integra perfectamente con el resto de productos basados en Java.

La sintaxis de las JSP es similar a ASP, con las etiquetas <% y %>, pero el lenguaje de programación es de tipo *Script* basado en la sintaxis de Java. La vinculación de JSP con Java le permite interactuar con otros elementos de la plataforma como Enterprise JavaBeans (que son la alternativa Java a los objetos COM de *Microsoft*) o con bases de datos accesibles mediante JDBC (mecanismo para la conexión de aplicaciones Java con bases de datos).

Actualmente el único servidor web comercial con intérprete JSP es el *Java Web Server*, lo cual limita la elección del servidor web a uno sólo. Sin embargo, al ser Java una plataforma medianamente libre, se intenta promover que otros servidores web incorporen estas capacidades, y se espera que paulatinamente crezca el número de los que incluyen soporte JSP.

1.5.7.- Tecnologías de Cliente: Applets y JavaScript

Las tecnologías web mostradas anteriormente (CGI, Servlets, ASP y JSP) eran tecnologías encaminadas a añadir inteligencia en la parte servidora de las aplicaciones web ya que todas ellas implicaban la ejecución de procesos en el servidor.

A continuación se comentará muy brevemente dos tecnologías Java cuya finalidad es añadir inteligencia en el cliente: **Applets** y **JavaScript**.

Los **applet** son pequeños programas en Java que se incrustan en paginas HTML. Estos programas son descargados hasta el cliente cada vez que se solicita la página web en la que “viven” y ejecutados dentro de ella. Son, por tanto, programas que se ejecutan en el cliente y dentro del browser.

La tecnología **JavaScript** es similar a la tecnología ASP y JSP en el sentido de que es un lenguaje de *Script* que se incrusta entre las etiquetas de una pagina HTML, pero a diferencia de lo que ocurre con éstos, el código *JavaScript* es descargado íntegramente con la página HTML en la que esta incrustado y ejecutado en el cliente.

Como se puede comprobar *applets* y *JavaScript* son ambas tecnologías de cliente ya que implican la ejecución de procesos en el cliente y no en el servidor.

1.6.- MODELOS DE COMPONENTES DISTRIBUIDOS

En la actualidad existen tres modelos de componentes mundialmente reconocidos. A continuación se realizará una breve introducción de cada uno de ellos.

1.6.1.- CORBA

CORBA es una especificación desarrollada por OMG (Object Management Group), que es un consorcio que actualmente cuenta con más de 800 empresas que participan en las actualizaciones y ampliaciones de dicho estándar.

CORBA define un modelo de componentes estándar para el desarrollo de aplicaciones distribuidas orientadas a objetos. Este modelo permite que un conjunto de objetos distribuidos y heterogéneos se comuniquen, con independencia de donde estén localizados estos objetos, en que lenguaje estén implementados o sobre que plataforma se estén ejecutando.

Un componente CORBA va a ser, por tanto, una entidad binaria que puede residir en cualquier parte de la red y que ofrece una serie de servicios que el resto de objetos van a poder invocar sin necesidad de conocer la máquina donde reside el componente, el sistema operativo sobre el que se ejecuta, ni el lenguaje de programación en el que fue implementado. Todos estos detalles son totalmente transparentes para los clientes a la hora de invocar los servicios de un componente CORBA.

Basándonos en lo anterior, podemos entresacar como ventajas más importantes del modelo de componentes CORBA, las siguientes:

- Independencia del lenguaje de implementación de los componentes.
- Independencia de la plataforma de ejecución.
- Independencia de la localización física de los componentes.
- Modelo de componentes abierto.

1.6.2.- COM

Microsoft ha desarrollado su propio modelo de componentes de forma paralela al consorcio de empresas pertenecientes a OMG que crearon CORBA. El modelo de componentes COM reduce su

plataforma de ejecución a los sistemas operativos de la familia *Windows*, no siendo por tanto un modelo de componentes independiente de la plataforma como lo es CORBA.

COM (*Component Object Model*) constituye la apuesta del gigante informático en el campo de los componentes, siendo, por tanto, un modelo de componentes cerrado, explotado únicamente por *Microsoft* y preparado fundamentalmente para ser compatible únicamente con las tecnologías *Microsoft*.

Poco después de la tecnología COM surgió DCOM. La tecnología DCOM (*Distributed COM*) extiende las capacidades iniciales de COM para permitir la comunicación entre componentes remotos, con independencia de la localización física de la máquina donde residan, permitiendo de esta forma el desarrollo de aplicaciones distribuidas basadas en componentes COM.

1.6.3.- Enterprise JavaBeans

Los *Enterprise JavaBeans* (EJB) son la alternativa Java a los modelos de componentes CORBA y COM.

Como características básicas cabe destacar las siguientes:

- Es un modelo de componentes basado íntegramente en la plataforma Java, siendo, por tanto, un modelo independiente de la plataforma de ejecución.
- Es un modelo de componentes distribuido y como el resto de modelos permite trabajar con independencia de la localización física de los componentes.
- Los EJB incorporan un modelo de persistencia y de gestión de transacciones que se aplica a todos sus componentes.

Los *Enterprise JavaBeans* son el modelo de componentes que estudiaremos en esta asignatura y serán explicados con más profundidad en el tema 4.