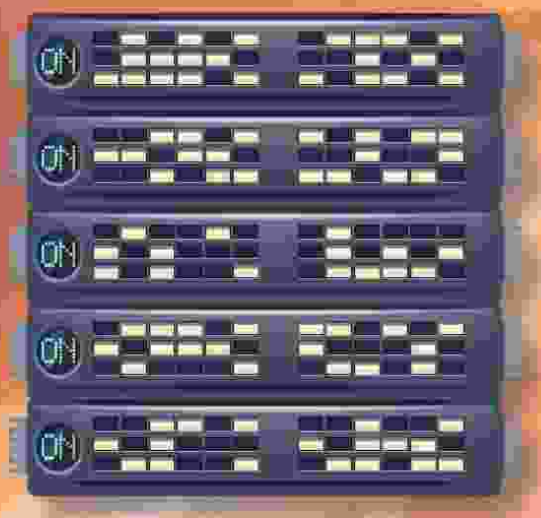
CF

GRADO SUPERIOR

CICLOS FORMATIVOS

R.D. 1538/2006

Desarrollo Web en Entorno Servidor



MARCOS LÓPEZ SANZ
JUAN MANUEL VARA MESA
JÉNIFER VERDE MARÍN
DIANA MARCELA SÁNCHEZ FÚQUENE
JESÚS JAVIER JIMÉNEZ HERNÁNDEZ
VALERIA DE CASTRO MARTÍNEZ

Ra-Ma®

www.ra-ma.es/cf

La ley prohíbe
Copiar o Imprimir este libro

DESARROLLO WEB EN ENTORNO SERVIDOR

ISBN de Edición en Papel: 978-84-9964-156-0

Editado por:
RA-MA, S.A. Editorial y Publicaciones
Calle Jarama, 33, Polígono Industrial IGARSA
28860 PARACUELLOS DE JARAMA, Madrid
Teléfono: 91 658 42 80
Fax: 91 662 81 39
Correo electrónico: editorial@ra-ma.com
Internet: www.ra-ma.es y www.ra-ma.com

Maquetación: Gustavo San Román Borrueco
Diseño Portada: Antonio García Tomé

ISBN: 978-84-9964-365-6

E-Book desarrollado en España en Septiembre de 2014

Desarrollo Web en Entorno Servidor

MARCOS LÓPEZ SANZ
JUAN MANUEL VARA MESA
JÉNIFER VERDE MARÍN
DIANA MARCELA SÁNCHEZ FÚQUENE
JESÚS JAVIER JIMÉNEZ HERNÁNDEZ
VALERIA DE CASTRO MARTÍNEZ

Ra-Ma®

Descarga de Material Adicional

Este E-book tiene disponible un material adicional que complementa el contenido del mismo.

Este material se encuentra disponible en nuestra página Web www.ra-ma.com.

Para descargarlo debe dirigirse a la ficha del libro de papel que se corresponde con el libro electrónico que Ud. ha adquirido. Para localizar la ficha del libro de papel puede utilizar el buscador de la Web.

Una vez en la ficha del libro encontrará un enlace con un texto similar a este:

"Descarga del material adicional del libro"

Pulsando sobre este enlace, el fichero comenzará a descargarse.

Una vez concluida la descarga dispondrá de un archivo comprimido. Debe utilizar un software descompresor adecuado para completar la operación. En el proceso de descompresión se le solicitará una contraseña, dicha contraseña coincide con los 13 dígitos del ISBN del libro de papel (incluidos los guiones).

Encontrará este dato en la misma ficha del libro donde descargó el material adicional.

Si tiene cualquier pregunta no dude en ponerse en contacto con nosotros en la siguiente dirección de correo: ebooks@ra-ma.com

A Cuca,

por la oportunidad y su constante apoyo.

Índice

Introducción

El esquema básico sobre el que se asientan los sistemas de información de Internet es la arquitectura cliente-servidor. En este contexto existe una gran variedad de tecnologías que el desarrollador tiene a su disposición para el desarrollo de aplicaciones flexibles y dinámicas y que, en definitiva, han favorecido el auge de la Web. Si bien la tendencia actual es hacia un equilibrio en el reparto de las responsabilidades entre el entorno cliente y el servidor, aún muchos de los sistemas web confían en la potencia de los grandes centros servidores para diseñar y desarrollar soluciones adecuadas que ofrezcan diferentes funcionalidades. En este sentido, un desarrollador web debe conocer y dominar no solo los lenguajes de codificación del entorno servidor, como PHP o ASP, sino también los estándares, tecnologías y plataformas que tiene a su disposición. De esta forma, un desarrollador web puede crear soluciones que procesen las peticiones de un cliente y devuelvan una respuesta adecuada a sus necesidades y requerimientos.

El desarrollo de aplicaciones del entorno servidor no solo se centra en la utilización de lenguajes que atiendan las peticiones del cliente, estática o dinámicamente, sino que también incluye la gestión del control de acceso a recursos de datos (bases de datos y ficheros) y la provisión de servicios por parte de terceros. Es necesario, por tanto, dominar los aspectos de creación de aplicaciones web dinámicas, los principios de conexión y acceso a datos, la gestión de datos multimedia y el desarrollo de servicios web basados en estándares.

Por todo ello, y ante el auge y explosión de Internet como herramienta de comunicación y trabajo y la creciente demanda de profesionales cualificados del sector, este libro surge con el propósito de acercar al lector los aspectos más importantes que encierra el desarrollo web en el entorno servidor. Con tal propósito, puede servir de apoyo también para estudiantes de los Ciclos Formativos de Grado Superior de Desarrollo de Aplicaciones Web y para las Ingenierías Técnicas.

Para todo aquel que use este libro en el entorno de la enseñanza (ciclos formativos o universidad) se ofrecen varias posibilidades: utilizar los conocimientos aquí expuestos para inculcar aspectos genéricos del desarrollo web o simplemente centrarse en preparar a fondo alguno de ellos. La extensión de los contenidos aquí incluidos hace imposible su desarrollo completo en la mayoría de los casos.

Ra-Ma pone a disposición de los profesores material adicional en la ficha del libro en la página web de la editorial para el desarrollo del tema que incluye las soluciones a los ejercicios expuestos en el texto, un glosario, bibliografía y diversos recursos para suplementar el aprendizaje de los conocimientos de este módulo. Puede solicitar también la Guía Didáctica que complementa al libro dirigiéndose a la editorial, acreditándose como docente y siempre que el libro sea utilizado como texto base para impartir las clases.

1 Selección de arquitecturas y herramientas de programación

OBJETIVOS DEL CAPÍTULO

- ✓ Caracterizar y diferenciar los modelos de ejecución de código en un entorno cliente/servidor.
- ✓ Conocer los mecanismos de ejecución de código en función de las diferentes tecnologías web.
- ✓ Reconocer las ventajas de la generación dinámica de páginas web.
- ✓ Identificar y caracterizar los principales lenguajes y tecnologías de programación en entorno servidor.
- ✓ Conocer las distintas herramientas de programación web para lenguajes de servidor.

En este primer capítulo presentamos los conceptos necesarios para comprender el contexto del desarrollo de aplicaciones y sistemas de información web. Para ello, introducimos los modelos de programación más comunes en la Web, así como los diferentes lenguajes y tecnologías de programación, del lado del servidor, aplicables en este tipo de entornos. Por último, haremos un recorrido por algunas de las herramientas que el desarrollador tiene a su alcance para sacarle el máximo partido a la programación web.

1.1 MODELOS DE PROGRAMACIÓN EN ENTORNOS CLIENTE/SERVIDOR

La *World Wide Web* (o *la Web*, como se conoce comúnmente) representa un universo de información accesible globalmente a través de Internet. Está formada por un conjunto de recursos interconectados que conforman el conocimiento humano actual. El funcionamiento de la Web es posible debido a la coexistencia de una serie de componentes software y hardware. Estos elementos abarcan desde los componentes físicos de Internet (*hubs*, repetidores, puentes, pasarelas, encaminadores, etc.) y los protocolos de comunicaciones (TCP, IP, HTTP, FTP, SMTP, etc.) hasta la utilización del sistema de nombres de dominio (DNS) para la búsqueda y recuperación de recursos o la utilización de software específico para proveer y consumir dichos recursos.

En este contexto, el desarrollo en entornos web debe tener en cuenta la distribución de los elementos y la función que tiene cada uno de ellos. La configuración arquitectónica más habitual se basa en el modelo denominado *Cliente/Servidor*, basado en la idea de servicio, en el que el *cliente* es un componente consumidor de servicios y el *servidor* es un proceso proveedor de servicios. Además, esta relación está robustamente cimentada en el intercambio de mensajes como el único elemento de acoplamiento entre ambos.

El agente que solicita la información se denomina *cliente*, mientras que el componente software que responde a esa solicitud es el que se conoce como *servidor*. En un proceso habitual el cliente es el que inicia el intercambio de información, solicitando datos al servidor, que responde enviando uno o más flujos de datos al cliente. Además de la transferencia de datos real, este intercambio puede requerir información adicional, como la autenticación del usuario o la identificación del archivo de datos que vayamos a transferir. La Figura 1.1 muestra este proceso genérico.

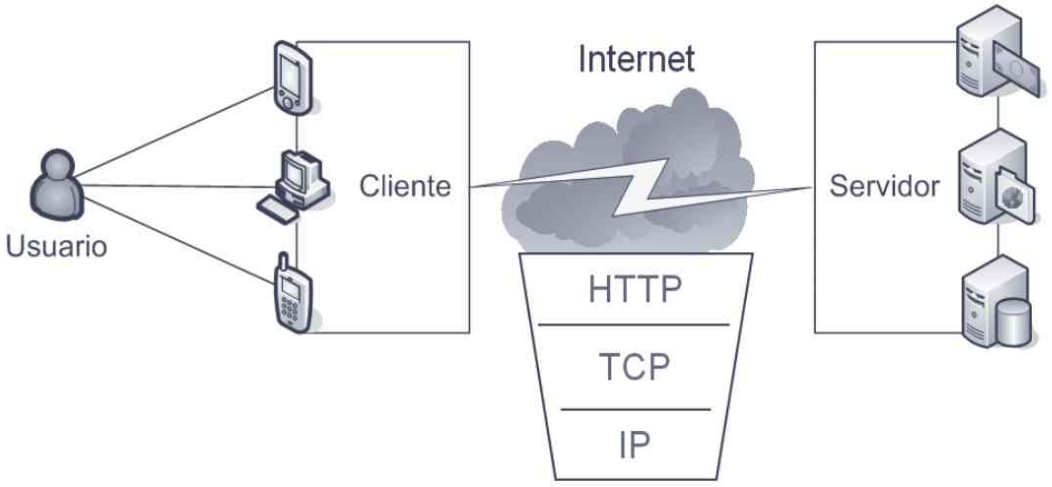


Figura 1.1. *Vista general del modelo cliente/servidor*

Las funcionalidades en los entornos cliente/servidor de la Web suelen estar agrupadas en diferentes *capas*, cada una centrada en la gestión de un aspecto determinado del sistema web. Si bien es posible encontrarse con divisiones más o menos especializadas, tradicionalmente se identifican tres tipos de capas fundamentales: *capa de presentación*, *capa de la lógica de negocio* y *capa de persistencia* (almacenamiento de datos). Los modelos arquitectónicos de programación se pueden clasificar en función de en qué lugar, si en el cliente o en el servidor, se sitúan cada una de estas capas. La decisión de dónde se sitúa cada una de estas capas dependerá del entorno de ejecución y, por tanto, de las tecnologías y lenguajes utilizados.

- **Capa de presentación**: esta capa es la que ve el usuario. Le presenta una interfaz gráfica del recurso solicitado y sirve para recoger su interacción. Suele estar situada en el cliente. La programación de esta capa se centra en el formateo de la información enviada por el servidor y la captura de las acciones realizadas por el cliente.
- **Capa de negocio**: es la capa que conoce y gestiona las funcionalidades que esperamos del sistema o aplicación web (lógica o reglas de negocio). Habitualmente es donde se reciben las peticiones del usuario y desde donde se envían las respuestas apropiadas tras el procesamiento de la información proporcionada por el cliente. Al contrario que la capa de presentación, la lógica de negocio puede ser programada tanto en el entorno cliente como en el entorno servidor.
- **Capa de persistencia o de datos**: es la capa donde residen los datos y la encargada de acceder a los mismos. Normalmente, está formada por uno o más gestores de bases de datos que realizan todo el proceso de administración de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

El diseño de cada una de estas capas influye a la hora de seleccionar uno u otro modelo de programación en la Web. Dichos modelos de programación cliente/servidor pueden clasificarse de este modo atendiendo a diferentes criterios. Así, es posible clasificarlos dependiendo del *tamaño* de los componentes, por la *naturaleza* del servicio ofrecido o según el *reparto de funciones* entre cliente y servidor:

- **Según el tamaño de los componentes**: esta primera clasificación hace referencia a qué elemento de la arquitectura web debe soportar más o menos carga de procesamiento. Se habla de configuraciones *Fat Client* (*thin Server*), donde el mayor peso de la aplicación se ejecuta en el cliente relegando al servidor a un mero administrador de datos; o *Fat Server* (*thin client*), donde la funcionalidad asociada al cliente está limitada a la presentación de la información enviada por el servidor.
- **Según la naturaleza del servicio ofrecido**: también es posible clasificar los entornos cliente/servidor en función de las capacidades ofrecidas por el servidor. De esta forma, podemos encontrar *servidores de ficheros*, donde el objetivo del cliente es el acceso a datos contenidos en ficheros; *servidores de bases de datos*, que se centran en la provisión y administración de sistemas gestores de bases de datos; *servidores de transacciones*, centrados en el concepto de transacción con el objetivo de que los flujos de información con el cliente se realicen en un solo mensaje solicitud/respuesta; *servidores de objetos*, cuya principal característica es la utilización de objetos intercomunicados, tanto en el cliente como en el servidor; o *servidores web*, que conforman la base del modelo *World Wide Web* y que está fundamentado en la existencia de clientes simples que se comunican con servidores web utilizando HTTP como protocolo de comunicación.
- **Reparto de funciones entre cliente y servidor**: las diferentes tecnologías web existentes permiten gestionar y distribuir las responsabilidades de cada una de las prestaciones funcionales entre el cliente y el servidor. Lo más habitual es tener una configuración cliente/servidor de dos o tres capas, dependiendo de si las capas de negocio y datos se agrupan (*modelo en dos capas*) o si se separan (*modelo en tres capas*). La separación en dos o tres capas la podemos ver, además, tanto desde el punto de vista del software como del hardware.

1.2 GENERACIÓN DINÁMICA DE PÁGINAS WEB

El contenido que visualiza un cliente se obtiene debido a su interacción con una aplicación que es gestionada por el servidor. Las aplicaciones web pueden ser clasificadas de acuerdo a múltiples criterios: dependiendo de la *configuración* o modelo arquitectónico seleccionado, según la *tecnología* o *lenguaje utilizado* e, incluso, desde un punto de vista más *funcional*, es decir, observando la capacidad de comunicación con los usuarios en respuesta a sus interacciones con la interfaz que se les presenta. Son estos criterios los que dan lugar a la clasificación de las aplicaciones web en *estáticas*, *dinámicas* e *interactivas*.

- **Aplicaciones web estáticas**: hacen referencia a aquellas aplicaciones web en las que el usuario recibe una página web cuya interacción no conlleva ningún tipo de acción, ni en la propia página, ni genera respuesta alguna por parte del servidor. Este tipo de aplicaciones suele realizarse utilizando el lenguaje HTML exclusivamente para la organización visual de la información.

- **Aplicaciones web dinámicas**: la programación de estas aplicaciones suele conocerse con el nombre de HTML dinámico (DHTML) y se refiere a aquellas aplicaciones en las que la interacción del cliente con el recurso recibido por parte del servidor (página web) produce algún tipo de cambio en la visualización del mismo (cambios de formato, ocultación de partes del documento, creación de elementos nuevos, etc.). Los lenguajes involucrados en este tipo de aplicaciones incluyen, entre otros, HTML, CSS o las múltiples variaciones de JavaScript (VBScript, JScript, Flash, etc.).

- **Aplicaciones web interactivas**: al contrario que en los tipos de aplicaciones anteriores, donde la interacción del usuario produce un cambio en el recurso recibido, las aplicaciones web interactivas se basan en que dicha interacción hace que se genere un diálogo entre el cliente y el servidor. Desde el punto de vista del modelo de programación, la lógica asociada al inicio y gestión de dicho diálogo puede ser ejecutada tanto en el cliente como en el servidor (e incluso en ambos).

El tipo de aplicaciones web interactivas es el que más se utiliza actualmente en Internet y donde el número de tecnologías disponibles ha evolucionado más en los últimos años. Las tecnologías implicadas en este tipo de aplicaciones varían mucho en función de si son ejecutadas en el lado del cliente o en el lado del servidor.

En el lado del cliente encontramos lenguajes y tecnologías para la creación de aplicaciones interactivas como HTML (por ejemplo, en la forma de formularios que rellena el usuario y envía al servidor), controles ActiveX (ejecución de comportamientos asociados a certificados), objetos embebidos de tipo Flash (animaciones interactivas visualmente atractivas), *applets* o AJAX (carga dinámica de contenidos), entre otros.

En el lado del servidor es posible utilizar lenguajes embebidos en código HTML (como PHP, ASP o JSP), enlaces a ejecutables (tipo CGI o SSI), objetos (*Servlets*) e incluso lenguajes que separan la presentación de la lógica de negocio (ASP.Net de Microsoft).

1.3 LENGUAJES DE PROGRAMACIÓN EN ENTORNO SERVIDOR

Se entiende por lenguaje de programación correspondiente a un entorno servidor a aquel cuyo código, bien sea como objeto precompilado o bien como código interpretado, es ejecutado por un software específico en el componente que actúa como servidor.

Existen múltiples alternativas a la hora de ejecutar código en el servidor. Una de ellas es utilizar lenguajes de *scripting* (SSI, LiveWire, ASP, PHP, etc.), cuya característica principal es que el código es interpretado y que se intercala con una plantilla de código HTML con la estructura básica de la página que se envía al cliente. Otra alternativa es el uso de enlaces a programas y componentes ejecutables (CGI, JSP, EJB, etc.), de tal forma que el código ejecutado por el servidor está almacenado en unidades precompiladas y ejecutadas de forma independiente, que generan las páginas enviadas al cliente. Otras técnicas de programación en el lado del servidor incluyen el uso de estrategias "híbridas" basadas en técnicas de respaldo (denominadas *code-behind*), como ASP.Net de Microsoft, en el que algunos elementos de código se intercalan con la lógica de presentación mientras que se mantiene la funcionalidad de dichos elementos en ficheros o librerías independientes en el servidor.

1.3.1 LENGUAJES DE SCRIPTING

Se entiende por lenguaje de *scripting* (o embebido) aquel que se intercala con el código HTML que forma una aplicación web. La idea básica es crear una serie de *plantillas HTML* en las que se insertan instrucciones de programación en diferentes lenguajes que son ejecutadas por un servidor para determinar las partes dinámicas de las páginas web. Para que estas porciones de código sean ejecutadas, el agente que actúa de servidor (*servidor web*) debe tener instalado un módulo (*scripting engine*) que sea capaz de reconocer e interpretar el lenguaje en el que se programa dicho código.

Existen diversos lenguajes de *scripting* que pueden utilizarse para la creación de aplicaciones web dinámicas. Entre ellos destacan PHP, ASP, Perl, Python y JSP:

- **PHP** (*Hypertext Processor*): es uno de los lenguajes más extendidos actualmente. Cuenta con una comunidad de desarrolladores muy importante debido a sus características de gratuidad, código abierto, la posibilidad de ser portado y ejecutado en diferentes plataformas, etc. Se define como un lenguaje imperativo de tipos dinámicos, con la posibilidad de utilizar construcciones orientadas a objetos. Es soportado por la gran mayoría de servidores web actuales.
- **ASP** (*Active Server Pages*): se trata de una tecnología propietaria y de código cerrado para el lado del servidor de Microsoft. Su última versión es la 3.0 y data del año 2002, año a partir del cual se empezó a sustituir progresivamente por la versión ASP.Net. Aunque puede ser ejecutado en otros servidores web, ASP está diseñado especialmente para ser utilizado con IIS (*Internet Information Server*). El lenguaje utilizado con ASP es Visual Basic en su versión de *scripting* (VBScript), aunque también pueden usarse otros (JScript).
- **Perl**: fue inicialmente concebido como un lenguaje de manipulación de cadenas de caracteres basado en un estilo de programación por bloques, como C o AWK. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas actualmente orientado a la generación dinámica de páginas web. Fue uno de los primeros lenguajes en ser utilizados para la programación web (para crear aplicaciones CGI). Es de código abierto y cuenta con una comunidad bastante numerosa de seguidores.

- **Python**: es un lenguaje de *scripting* independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programas, desde aplicaciones Windows a servidores de red, o incluso páginas web. Es un lenguaje interpretado, lo que significa que no necesitamos compilar el código fuente para poder ejecutarlo. Esto ofrece ventajas, como la rapidez de desarrollo, e inconvenientes, como una menor velocidad. En los últimos años el lenguaje se ha hecho muy popular gracias a la cantidad de librerías, tipos de datos y funciones incorporadas en el propio lenguaje o la sencillez y velocidad con la que se crean los programas. Además, Python es un lenguaje gratuito, es decir, no necesita de ningún software de pago adicional para su ejecución incluso para propósitos empresariales.
- **JSP** (*Java Server Pages*): el funcionamiento de Java como lenguaje de *script* es similar al de PHP o ASP: se trata de porciones de código Java intercalado con código HTML estático. Sin embargo, la forma de interpretarlo es diferente. Una vez que el módulo encargado de ejecutar la página JSP llega a una porción de código Java, lo transforma a un *Servlet* (porción de código Java cargado en la memoria de un servidor web) y lo ejecuta obteniendo el código que ha de enviarse al cliente. La diferencia con otros lenguajes es que el *servlet* queda cargado en memoria por si llegara otra petición a la misma página JSP, con lo que el rendimiento se ve mejorado notablemente.

Existen muchos otros lenguajes de *scripting* en el lado del servidor enfocados a entornos más específicos y con características particulares, como ColdFusion Markup Language, Lua, Ruby, SMX, Lasso, WebDNA o Progress WebSpeed.

1.3.2 APLICACIONES CGI Y DERIVADOS

Una de las formas más simples de generar páginas dinámicas es delegar la creación de las mismas a un programa externo, que reciba ciertos parámetros de entrada y devuelva como resultado el contenido que debe visualizar el cliente. El estándar CGI (*Common Gateway Interface*), especificado en la RFC3875, define precisamente este comportamiento, estableciendo los vínculos que hay que establecer con una aplicación independiente o fichero ejecutable. Puesto que el programa externo no depende del código a generar, el lenguaje elegido puede ser cualquiera (como C o C++). La localización del *script* o fragmento de programa a ejecutar se indica en la URL que forma la petición HTTP del cliente.

Como hemos dicho, CGI es la forma más simple de publicar contenido dinámico en la Web; sin embargo, esta forma de crear aplicaciones web presenta ciertas desventajas, siendo la principal de ellas el escaso rendimiento a la hora de responder a múltiples peticiones CGI simultáneamente puesto que para cada una se tiene que crear un proceso nuevo en el servidor con los costes de memoria y uso de procesador que conlleva. Respecto a este problema han surgido algunas alternativas como son el uso de extensiones para cada uno de los lenguajes en los que se programe el CGI pero como parte integrante del servidor web (módulos NSAPI o ISAPI por ejemplo), o el uso de *FastCGI* como forma de crear un único proceso que atienda a todas las peticiones CGI.

Otra de las alternativas que se han propuesto como solución a los problemas de CGI es el uso de plataformas independientes de ejecución de código. El principio de estas plataformas es equivalente al uso de extensiones para servidores web. Es el caso de la programación con Java en el lado del servidor. En este contexto, la arquitectura de un servidor que es capaz de ejecutar programas Java (*Servlets*, *JavaBeans*, etc.) está formada por el propio servidor web en sí mismo y la Máquina Virtual de Java (JVM), que es capaz de ejecutar un programa especial Java (llamado *Servlet container*) encargado de gestionar datos de sesión y *Servlets*.

Otra de las alternativas que un desarrollador tiene a su disposición cuando programa con Java es usar los denominados *Enterprise Java Beans* (EJB), que son componentes programados en Java que implementan la lógica de negocio de la aplicación utilizando las características de la plataforma J2EE. Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJB. El uso de EJB está especialmente indicado en los casos en los que se requiera mantener la transaccionalidad de las peticiones hechas por un cliente en un entorno web.

1.3.3 APLICACIONES HÍBRIDAS DE CÓDIGO REPARTIDO

Como alternativa a los lenguajes de *scripting* (interpretados) y a las aplicaciones CGI y derivadas (ejecutadas por un programa o módulo independiente del servidor web), en los últimos años ha surgido una tecnología intermedia que podríamos denominar como "híbrida". La solución más representativa es la plataforma de Microsoft .Net Framework a través de ASP.Net.

ASP.Net es una tecnología totalmente orientada a objetos que puede ser escrita en cualquier lenguaje soportado por el entorno .Net Framework (VB.Net; C# y JScript.Net). Desde el punto de vista del rendimiento, la aplicación se precompila en una sola vez al lenguaje nativo y, luego, en cada petición tiene una compilación *Just in Time*, es decir, se compila desde el código nativo, lo que permite mucho mejor rendimiento.

Las páginas de ASP.Net, conocidas oficialmente como *Web Forms* (formularios web), son el principal medio de construcción para el desarrollo de aplicaciones web desde el punto de vista de Microsoft. Los formularios web están contenidos en archivos con una extensión ASPX que son los que el cliente solicita a través de una URL al servidor. Estos ficheros ASPX contienen código HTML o estático y también etiquetas propias de la plataforma .Net. Estas etiquetas definen *Controles Web* que se procesan del lado del servidor y *Controles de Usuario* donde los desarrolladores colocan todo el código estático y dinámico requerido por la página web.

Adicionalmente, el código dinámico que se ejecuta en el servidor puede ser colocado en una página dentro de un bloque "<% -- código dinámico -- %>" que es muy similar a otras tecnologías de *scripting* como PHP, JSP y ASP. Esta circunstancia es otra de las razones por las que se considera la programación con ASP.Net como "híbrida". En algunos entornos de alto rendimiento y seguridad esta práctica es, generalmente, desaconsejada excepto para propósitos de enlace de datos, pues requiere más llamadas cuando se genera la página.

1.4 INTEGRACIÓN CON LOS SERVIDORES WEB

Hasta ahora hemos comentado que para el desarrollo de aplicaciones web en entornos cliente/servidor pueden utilizarse diferentes modelos, tecnologías y lenguajes. En cualquier caso, en muchas ocasiones, la posibilidad de responder a una petición hecha por un cliente depende de las capacidades que tenga el servidor web y los módulos o extensiones que tenga instalados.

Empezando por la funcionalidad básica de un servidor web, su cometido principal es proveer de contenido estático a un cliente que ha realizado una petición a través de un navegador. Para ello, carga un archivo y lo sirve a través de la Red al navegador del solicitante. Este es el funcionamiento habitual cuando lo que se implementa en el servidor es una aplicación web estática (HTML) o dinámica (DHTML), es decir, que el servidor se convierte en un instrumento que proporciona un lugar para guardar y administrar los recursos HTML, que pueden ser accesibles por los usuarios de la Red a través de navegadores.

Para que el servidor pueda entender la petición del cliente, éste tiene que realizar una petición "formal". Es decir, la petición tiene que constar de unos elementos concretos y especificados en un orden determinado. Las direcciones de las peticiones suelen ser de tipo URL (*Localizador Uniforme de Recurso*), que contiene una dirección, la referencia a un cierto protocolo (HTTP, FTP, etc.) y la descripción de un recurso concreto en forma de ruta al objeto que queremos descargar. Opcionalmente podemos incluir un campo adicional (puerto) indicando el número de puerto por el que el servidor está escuchando las peticiones del cliente.

Con respecto a las peticiones de los clientes es importante destacar que existen distintos modos o métodos para intercambiar información entre cliente y servidor y que deberemos tenerlo en cuenta en el desarrollo de aplicaciones web en el entorno del servidor.

- **Método GET**: es un método de invocación en el que el cliente le solicita al servidor web que le devuelva la información identificada en la propia URL. Lo más común es que las peticiones se refieran a un documento HTML o a una imagen, aunque también se puede referir a un programa de base de datos. En tal caso, el servidor ejecuta ese programa y le devuelve al cliente el resultado generado tras esa petición.
- **Método POST**: mientras que el método GET se utiliza para recuperar información, el método POST se usa habitualmente para enviar información a un servidor web. Estos casos suelen darse al enviar el contenido de un formulario de autenticación, así como entradas de datos o especificar parámetros para algún tipo de componente ejecutado en el servidor.

Como ya se ha comentado anteriormente, la potencia de los servidores web aparece en el momento en que queremos desarrollar aplicaciones web interactivas y con una cierta complejidad. En ese caso, podemos utilizar varias tecnologías implementadas en módulos específicos en el servidor para aumentar su potencia más allá de su capacidad de entregar páginas HTML. Dichas tecnologías suelen instalarse como extensiones en el software del servidor e incluyen soporte para *scripts* CGI, proporcionan seguridad SSL, permiten la ejecución de *scripts*, interactúan con la máquina virtual de Java, etc.

1.5 HERRAMIENTAS DE PROGRAMACIÓN

La selección de un entorno de desarrollo para la programación web debe tener en cuenta los diferentes escenarios, modelos y configuraciones que intervienen en este contexto. Las aplicaciones web están formadas por un conjunto de páginas HTML, *scripts* de lenguajes web, programas escritos en diferentes lenguajes, bases de datos y documentos de diferentes formatos y pueden residir en varias ubicaciones o sitios web, es decir, en diferentes servidores.

El proceso de desarrollo, sin embargo, no tiene por qué realizarse en el mismo equipo en el que finalmente se despliegue y ejecute la aplicación web que se está desarrollando. El mismo razonamiento es aplicable al proceso de pruebas en el que se simulen las peticiones al servidor web desde un cliente determinado. Para el proceso de desarrollo deberemos tener en cuenta, en el caso del servidor, el conjunto de tecnologías y lenguajes seleccionados, el tipo de servidor, los módulos y extensiones que tiene configurado, los permisos de acceso y ejecución o la localización de los datos utilizados por la aplicación web. Desde el punto de vista del cliente habrá que tener en cuenta, entre otros, la versión y tipo de navegador, la resolución gráfica o el sistema de permisos del navegador en el equipo cliente.

A la hora de seleccionar las herramientas que un desarrollador tiene a su disposición es necesario hacer una primera clasificación de los instrumentos involucrados en función de sus capacidades:

- **Navegadores**. Son las aplicaciones que se ejecutan en el entorno del cliente y que permiten visualizar los documentos escritos en lenguaje HTML y capturar las interacciones del usuario.
- **Editores de documentos**. Este grupo está formado por editores de texto que permiten escribir código HTML directamente, sin ninguna ayuda ni facilidad adicional.
- **Entornos de programación**. Son entornos integrados que nos permiten editar, compilar y ejecutar los programas generados a partir de diferentes lenguajes usados en el desarrollo de las aplicaciones web.

- **Herramientas de tratamiento de imágenes**. La mayoría de las páginas web muestran contenido gráfico de una u otra manera, por ello, es necesario el uso de este tipo de herramientas para adecuar las características de las imágenes a su transmisión por la Red y su posterior visualización.
- **Herramientas para la creación y administración de bases de datos**. Se trata de herramientas para la carga de datos y el mantenimiento posterior de los datos almacenados.

En el ámbito del desarrollo de aplicaciones web las herramientas más importantes son los llamados *entornos de programación*, que son programas, aplicaciones o simples utilidades destinadas a la programación web, haciendo mucho más fácil para el programador su tarea. Desde el punto de vista de las herramientas que se pueden utilizar para la construcción de programas o *scripts*, podemos emplear una gran variedad de utilidades que van desde simples editores de texto hasta los entornos integrados que facilitan enormemente la construcción de aplicaciones web en lenguajes específicos.

1.5.1 MARCADORES DE TEXTO

Los marcadores de texto, o *text markers*, son simples editores de texto, aunque dirigidos al ámbito de la programación. Al escribir el programa, y dependiendo del lenguaje de programación utilizado, el editor de texto nos ayuda a identificar mejor la sintaxis del lenguaje, cambiando de color las etiquetas, realizando tabulaciones en el texto, etc. Algunas de ellas son:

- **Arachnophilia**. Es un robusto editor de texto, ideal para programar sitios web en diferentes lenguajes. Soporta HTML, JavaScript, C++, CGI, Perl, Java, entre otros. También incluye un cliente inteligente de FTP para subir automáticamente los archivos modificados.
- **Notepad++**. Es un editor gratuito de código fuente. Soporta varios lenguajes de programación y se ejecuta en Windows.
- **UltraEdit**. Completo editor de texto para programación. Soporta múltiples formatos con colores configurados para cada lenguaje. Es uno de los múltiples editores de pago que podemos encontrar en el mercado.

1.5.2 HERRAMIENTAS GENÉRICAS

Dentro de los entornos de programación existen herramientas que ofrecen funcionalidades avanzadas aparte del reconocimiento de la sintaxis del lenguaje. Estas aplicaciones están mucho más desarrolladas que los marcadores de texto y ofrecen capacidades tales como la sugerencia de estructuras o funciones predeterminadas o la posibilidad de validar la corrección del código escrito. A continuación describimos algunas de estas herramientas a modo de ejemplo:

- **Microsoft FrontPage**. Es una herramienta de construcción y edición de páginas web para el sistema operativo Windows. Si bien tuvo bastante éxito hace unos años, actualmente muchos desarrolladores consideran que el código HTML generado por esta aplicación es un poco descuidado y muchas veces reiterativo, especialmente en versiones antiguas. Forma parte del paquete ofimático Microsoft Office y es de pago.
- **Eclipse**. Se trata de un entorno de programación gratuito escrito en Java que permite desarrollar aplicaciones en varios lenguajes (C, Java, PHP...). Una de sus principales características es que permite la extensión de sus funcionalidades mediante la instalación de múltiples módulos para diferentes propósitos específicos.
- **Dreamweaver**. Es un editor de código HTML profesional de pago para el diseño visual y la administración de sitios y páginas web. También incluye numerosas herramientas y funciones de edición de código: referencias HTML, CSS y JavaScript, un depurador JavaScript y editores de código (la vista de código y el inspector de código) que permiten editar JavaScript, XML y otros documentos de texto directamente en Dreamweaver.

1.5.3 HERRAMIENTAS ESPECÍFICAS

En muchas ocasiones, la utilización de un lenguaje de programación determinado exige que el desarrollador tenga instalada una plataforma de desarrollo concreta. Tal es el caso de la programación en Java que requiere la instalación de la Máquina Virtual de Java (JVM) o de el entorno .Net Framework de Microsoft, necesaria para la programación con ASP.Net. Las herramientas que mostramos a continuación se encuentran dentro de la categoría de herramientas específicas para una tecnología concreta.

- **Microsoft Visual Studio**. Es el entorno de desarrollo más conocido para el diseño de aplicaciones en los sistemas operativos de Microsoft. Está destinado al desarrollo de aplicaciones web con lenguajes como Visual Basic o C++ y permite la publicación de la aplicación web que se está desarrollando directamente desde su interfaz.
- **NetBeans IDE**. Es una aplicación de código abierto diseñada para el desarrollo de aplicaciones fácilmente portables entre distintas plataformas haciendo uso de la tecnología Java. Se trata de un entorno de desarrollo gratuito optimizado para el desarrollo de aplicaciones con Java.

ACTIVIDADES 1.1

- Los modelos de programación por capas presentados en este capítulo se centran en la asignación de funcionalidades a nivel software. Proponemos que el alumno complemente estos modelos investigando las diferentes ventajas y desventajas que pueden existir en el caso de realizar una asignación de responsabilidades en diferentes nodos hardware.
- La arquitectura de un servidor web varía en función de la plataforma sobre la que se vaya a utilizar y las capacidades que se requieren. Algunos de los servidores web más conocidos son *Apache*, de la *Apache Foundation,* o *Internet Information Server de Microsoft.* Proponemos que el alumno busque la última versión de estos servidores y describa tanto su arquitectura básica como los mecanismos de extensión que ofrecen para el soporte de diferentes tecnologías de programación en entorno del servidor.
- Los lenguajes del entorno del servidor presentados han ido evolucionando históricamente incluyendo cada vez más funcionalidades. Proponemos que el alumno elija uno de esos lenguajes y realice una descripción detallada de su evolución, indicando cuáles son las influencias recibidas de otros lenguajes y sobre qué otros lenguajes ha influido.
- Al igual que los servidores cuentan con extensiones, las herramientas de programación web también pueden ser extendidas. Busque información sobre uno de los editores mencionados y amplíe su información indicando los métodos en los que pueden ser extendidos.

RESUMEN DEL CAPÍTULO

Para poder abordar la creación de aplicaciones web es necesario conocer antes el contexto en el que se desarrollan estas aplicaciones. Así, podemos ver que el modelo arquitectónico en el que se basan es el denominado "cliente/servidor", donde un cliente, a través de un navegador, realiza peticiones a un servidor, que será el encargado de gestionar su petición y devolver una respuesta adecuada. Las aplicaciones desarrolladas en este contexto pueden ser muy sencillas (aplicaciones web estáticas) o muy complejas (aplicaciones web dinámicas e interactivas). La selección de una u otra tecnología, o lenguaje, dependerá principalmente del modelo de programación escogido, es decir, del reparto de funcionalidades entre el cliente y el servidor.

Los lenguajes utilizados en el entorno del servidor se agrupan en lenguajes de *scripting* (interpretados como PHP, ASP, Perl o Python), lenguajes para aplicaciones ejecutadas de forma independiente del servidor web (CGI y derivados) o lenguajes que optan por una programación "híbrida" (ASP.Net). Finalmente, para el desarrollo óptimo de aplicaciones web se suelen utilizar herramientas específicas que ayudan al desarrollador en su labor.

TEST DE CONOCIMIENTOS

1 ¿Cuál de los siguientes lenguajes no puede utilizarse como lenguaje de *scripting*?
- a) PHP.
- b) Perl.
- c) HTML.
- d) JSP.

2 ¿Cuál de las siguientes estrategias representa una estrategia de programación válida en entornos cliente/servidor?
- a) *Fat Client/Thin Server*.
- b) Programación en dos capas.
- c) Uso de servidores de objetos.
- d) Todos los anteriores.

3 ¿Cuál de las siguientes herramientas no permite la validación de la sintaxis de código?
- a) UltraEdit.
- b) Microsoft Visual Studio.
- c) NetBeans IDE.
- d) Ninguno los anteriores.

4 Señale la respuesta correcta con respecto a CGI:
- a) Es un lenguaje de programación del lado del servidor.
- b) Es un estándar que indica la forma en la que un servidor debe ejecutar un programa externo.
- c) Puede programarse en HTML.
- d) Es una herramienta de carácter genérico.

5 Señale la respuesta correcta con respecto a la integración de código con servidores web:
- a) Las peticiones a una aplicación web pueden hacerse siguiendo un formato "informal" (en lenguaje natural).
- b) Las peticiones de tipo POST se utilizan para recuperar información del servidor.
- c) El código de una aplicación web estática no requiere la ejecución de ningún programa externo por parte del servidor.
- d) Ninguna de las anteriores.

2 Inserción de código en páginas web

OBJETIVOS DEL CAPÍTULO

- ✓ Reconocer la arquitectura de las aplicaciones web del lado del servidor dependiendo del lenguaje utilizado.
- ✓ Aprender a generar código de forma dinámica para ser mostrado por el cliente web.
- ✓ Conocer la sintaxis y las etiquetas propias de cada lenguaje de servidor que permite insertar código en páginas web ejecutadas en el servidor.
- ✓ Dominar la declaración de variables, tipos de datos simples, así como la conversión entre cada uno de ellos en función del lenguaje utilizado.
- ✓ Comprender la importancia de controlar el ámbito de declaración de cada variable y su influencia en el desarrollo de aplicaciones web.

El desarrollo de aplicaciones web del lado del servidor se fundamenta en la utilización de una serie de lenguajes y tecnologías que son ejecutadas por software especializado en uno o más servidores. En función del lenguaje que queramos utilizar, la distribución de la lógica de la aplicación o la forma de interactuar con la información que proviene del cliente, la gestión del flujo de trabajo de la aplicación y la configuración del entorno del servidor será diferente. En este segundo capítulo presentamos diferentes tipos de servidores web, introducimos las características principales de algunos lenguajes de servidor que permiten insertar código en páginas HTML y manejar el flujo de ejecución para tratar la información enviada por un cliente a través de una página web.

2.1 LENGUAJES Y TECNOLOGÍAS DE SERVIDOR

Tal y como vimos en el primer capítulo, las tecnologías de servidor se basan en la existencia de un software especial denominado *servidor web*. La configuración de este componente software vendrá determinada por la utilización de un lenguaje u otro, ya que cada lenguaje requiere de una configuración determinada y de unos componentes específicos para ser ejecutado. Si queremos dominar el desarrollo de aplicaciones web dinámicas basadas en código de servidor es necesario comprender bien el escenario en el que se produce la ejecución de una aplicación web dentro de una arquitectura cliente-servidor tradicional.

Lo primero es entender que un *servidor web* es un programa cuya misión última es servir datos en forma de documentos HTML (*HyperText Markup Language*) codificados en este lenguaje. El objetivo es proveer al cliente con textos complejos con enlaces, figuras, formularios, botones y objetos incrustados, tales como animaciones o reproductores de sonidos.

Para que los datos le lleguen al cliente es necesario tener en cuenta cómo se produce la comunicación entre un cliente y un servidor. El intercambio de datos entre ambos actores se hace por medio un protocolo determinado, generalmente mediante el protocolo HTTP. Por defecto, un servidor web se mantiene a la espera de peticiones HTTP realizadas por parte de un cliente (a través de un *navegador web*) "escuchando" un puerto de comunicaciones (normalmente el 80).

La secuencia de comunicación es la siguiente: en un primer paso, el navegador solicita, como cliente DNS, la traducción de una URL (por ejemplo *http://www.educacion.es*) a una IP. Una vez que ha recibido la traducción del servidor DNS se realiza la petición HTTP al servidor web que tenga la IP obtenida (si ponemos directamente la dirección IP en vez de la URL en el navegador el resultado sería equivalente). Puesto que HTTP es un protocolo sin estado, cada petición de un cliente a un servidor no está influenciada por las transacciones anteriores. Esto quiere decir que el servidor tratará cada petición como una operación totalmente independiente del resto. Una novedad de este protocolo a partir de su versión 1.1 es que se pueden habilitar conexiones persistentes, lo cual afecta directamente a la eficiencia de las transacciones puesto que se permite enviar más objetos con un número menor de conexiones. En el siguiente paso, el servidor procesa la solicitud realizada por el cliente y, tras ejecutar el código asociado al recurso solicitado en la URL, responde al cliente enviando el código HTML de la página. Finalmente, el navegador del cliente, cuando recibe el código, lo interpreta y lo muestra en pantalla.

¿SABÍAS QUE...?

Algunas definiciones básicas: *Protocolo*- Conjunto de reglas y estándares que gobiernan el intercambio de información entre entidades dentro de una red. Existen muchos tipos de protocolos, cada uno con un propósito específico: FTP, POP3, SMTP, ICMP, etc. *Protocolo HTTP*- Protocolo de la capa de aplicación usado en cada transacción web. Cada petición realizada mediante HTTP implica una conexión con el servidor que es liberada al término de la misma, es decir, no tiene estado. Por ejemplo, la solicitud de un archivo HTML con 5 imágenes requiere el establecimiento de 6 conexiones distintas (5 imágenes más la página HTML en sí).

En función de cómo se procesen las peticiones realizadas por un cliente web podemos distinguir distintos tipos de servidores web. Las diferentes estrategias de optimización de la ejecución y de diseño del servidor dan lugar a diferentes tipos de servidores web:

- **Servidores basados en procesos**. Se puede considerar como la estrategia predecesora de todas las demás. Su funcionamiento se basa en la obtención de un paralelismo de ejecución mediante la duplicación del proceso de ejecución. Aunque existen diversas formas de gestionar los procesos, el más simple es aquel en el que el proceso principal espera la llegada de una nueva conexión (petición HTTP) y, en ese momento, se duplica creando una copia exacta (*fork*) que atenderá esta conexión, de tal forma que un proceso se dedica al tratamiento de la petición mientras que otro puede seguir escuchando nuevas peticiones. Esta estrategia admite múltiples optimizaciones como la técnica *pre-fork*, incluida por el servidor Apache desde su versión 1.3.

¿SABÍAS QUE...?

Técnica *pre-fork*. La idea es que en la memoria del servidor web se lanzan varias copias del proceso que escucha peticiones. Estas copias de proceso se mantienen hasta que se necesitan. Entre las ventajas de esta técnica destaca la facilidad para ser implementada y la seguridad para aislar el procesamiento de una petición frente a otras. Por el contrario, el mayor inconveniente de esta estrategia está en un mayor consumo de recursos de memoria (dedicada a cada proceso) y un menor rendimiento debido a que la gestión de los procesos es una tarea del sistema operativo.

- **Servidores basados en hilos**. Frente a los servidores basados en procesos existe una alternativa más económica en cuanto consumo de recursos. En los servidores basados en hilos de ejecución, en lo que se refiere al procesamiento de las peticiones, el funcionamiento es básicamente idéntico al de los servidores basados en procesos. La mayor diferencia se encuentra en la distinción entre el concepto de hilo (*thread*) y el de proceso. La creación de un hilo por parte de un proceso no es tan costosa como la duplicación de un proceso completo. Los hilos de ejecución creados por un proceso comparten el mismo espacio de memoria reduciendo el consumo de memoria del servidor web de forma drástica. Este aspecto, si bien mejora el rendimiento del servidor, puede constituir un problema de seguridad a la hora de acceder al espacio de memoria. Puesto que, generalmente, todos los hilos creados por un servidor comparten la misma zona de memoria, si un hilo modifica una variable, el resto de hilos del mismo proceso verían esa variable con el valor modificado.

- **Servidores dirigidos por eventos** (basados en *sockets* no bloqueantes). La novedad de este tipo de servidores es la utilización de *sockets* (espacios de memoria para la comunicación entre dos aplicaciones que permiten que un proceso intercambie información con otro proceso estando los dos en distintas máquinas, en este caso cliente y servidor). Las lecturas y escrituras sobre *sockets* son realizadas de forma asíncrona y bidireccional. De esta forma, aunque un cliente realizará peticiones de forma independiente (sin estado), el servidor podrá acceder a la información intercambiada con dicho cliente con solo acceder al *socket* creado para tal cliente. La ventaja de este diseño radica principalmente en su velocidad. Por el contrario, el mayor inconveniente se encuentra en que la concurrencia de procesamiento es simulada; es decir, existe un solo proceso y un solo hilo, desde el cual se atienden todas las conexiones gestionadas por un conjunto de *sockets*.
- **Servidores implementados en el núcleo del sistema** (kernel). La principal idea de esta estrategia de diseño es situar el procesamiento de cada una de las ejecuciones del servidor web en un espacio de trabajo perteneciente al sistema operativo (*kernel*) y no en un nivel de usuario (sobre el sistema operativo). Si bien es cierto que aunque esta alternativa permite acelerar el procesamiento de las peticiones al servidor, conlleva bastantes peligros. Cualquier incidencia (de seguridad o rendimiento) en este tipo de servidores implica un problema a nivel del sistema operativo que lo puede llegar a dejarlo completamente inoperativo.

Las diferentes estrategias de diseño descritas anteriormente han sido implementadas en diferente grado por algunos de los servidores web más utilizados actualmente. Entre ellos destacan los siguientes:

- **Apache Server**. Se trata de un servidor HTTP diseñado para ser utilizado en múltiples plataformas y sistemas operativos. Entre sus características destacan el hecho de ser un servidor robusto, que implementa los últimos estándares y protocolos de la Red y que cuenta con gran capacidad de personalización y modularización. Además, al ser gratuito y de código abierto, cuenta con una comunidad de desarrolladores bastante amplia que hacen que exista gran documentación al respecto de su uso y configuración para diferentes propósitos. Puede configurarse tanto como servidor basado en hilos como basado en procesos.

  Su diseño altamente flexible y configurable permite a los administradores de sitios web elegir qué características van a ser soportadas por el servidor, de tal forma que se pueden seleccionar qué módulos van a estar disponibles cuando se compile o ejecute el servidor web. Tiene módulos que ofrecen soporte para acceso a bases de datos, establecer páginas protegidas por contraseña, personalizar las páginas de error devueltas por el servidor, generar registros de actividad en múltiples formatos, etc.

- **Microsoft IIS**. Es el servidor web de Microsoft©. Según sus propias fuentes, IIS (*Internet Information Server*) es un motor de páginas web flexible, seguro y fácil de gestionar que permite alojar cualquier tipo de contenido *on line*. Destaca principalmente por dar soporte nativo a ASP (*Active Server Pages*) y a las diferentes tecnologías de la plataforma .Net. Además, permite añadir ciertos módulos para la ejecución de otros lenguajes como PHP. Este tipo de servidor lo llevan solo los sistemas basados en Windows, y suele instalarse como un complemento adicional al sistema operativo, accesible mediante permisos de administrador. En los sistemas Windows XP (versión *Professional*), por ejemplo, IIS no viene instalado por defecto.

  Al igual que el servidor Apache, permite configurar extensiones para reescritura de direcciones URL (para que varias URL sean respondidas por el mismo servidor), servicios de despliegue de aplicaciones multimedia, acceso a bases de datos o administración remota del servidor.

- **Sun Java System Web Server**. Se trata de un servidor web de alto rendimiento, de escalabilidad masiva y seguro que ofrece contenido dinámico y estático. Actualmente forma parte del software que Oracle tiene disponible para su descarga de forma gratuita (aunque el soporte técnico es de pago). Destaca por sus características de virtualización de dominios, versatilidad de configuración y seguridad robusta. Está optimizado para la integración y ejecución de aplicaciones Java (JSP, *Servlets*, NSAPI y CGI).

Como servidor web puede instalarse en prácticamente cualquier sistema operativo. Al igual que cualquier servidor de propósito general, admite la configuración de módulos y extensiones para la ejecución de código PHP, Ruby on Rails, Perl, Python y más lenguajes de servidor.

- **Ngnix**. Es un servidor HTTP que ha ganado cuota de mercado en los últimos años (de < 1% en 2007 a casi el 10% a finales de 2011). Es de código abierto y cuenta con una arquitectura modular de alto rendimiento, además de permitir funcionar como servidor *proxy* para otros protocolos de Internet como IMAP o POP3. Nginx es un servidor basado en procesos conocido por: su estabilidad, contener un amplio conjunto de filtros, ser fácilmente configurable y capaz de ser ejecutado en múltiples plataformas consumiendo relativamente pocos recursos. Tiene soporte para transacciones seguras basadas en SSL y TLS.
- **Lighttp**. Es un servidor web especializado para entornos en los que se requieren respuestas rápidas. También conocido como *Lighty*, soporta la mayoría de los estándares de comunicaciones de Internet y puede descargarse como software libre y de código abierto. Se distribuye bajo licencia BSD. Su consumo de memoria es bastante reducido (comparado con otros servidores web como Apache) así como lo es su carga de procesamiento en la CPU. Está especialmente pensado para soportar un balanceo de cargas elevado sin perder prestaciones, utilizando poca RAM y poca CPU. Otra de las características de este servidor es que utiliza un solo proceso con varios hilos de ejecución pero sin tener la capacidad de crear nuevos hilos de ejecución en función de la demanda.

¿SABÍAS QUE...?

Se puede consultar una lista actualizada de la utilización en el mercado de cada servidor en *http://news.netcraft.com/*.

La anterior lista incluye la descripción de los servidores más utilizados en la actualidad. La gran mayoría de ellos soportan la ejecución de diferentes lenguajes de programación del lado del servidor, algunos de forma nativa (como el soporte para ASP ofrecido por IIS) y otros mediante módulos y extensiones específicas (como el soporte para PHP de Apache). Estas extensiones hacen que el componente software que conforma cualquier servidor web sea capaz de responder adecuadamente a las peticiones que un cliente envía por HTTP a través de la Red. Por otro lado, en el capítulo anterior vimos cómo existían lenguajes que eran interpretados (como PHP o ASP) y otros que eran precompilados y que necesitaban de un substrato específico de ejecución para ser ejecutados (como ASP.Net o las aplicaciones basadas en Java). El soporte para uno u otro tipo de lenguajes vendrá determinado por los módulos y extensiones del servidor escogido. La configuración adecuada, por tanto, del núcleo del servidor web y de las correspondientes extensiones permitirá que el código desarrollado responda a los requisitos que de él se esperan.

La siguiente sección se centra en los principios básicos que un programador debe conocer para poder desarrollar una aplicación web ejecutada en cualquiera de los servidores mencionados con anterioridad. De esta forma, será posible adecuar la respuesta del servidor a las peticiones de un cliente web.

2.2 OBTENCIÓN DEL CÓDIGO ENVIADO AL CLIENTE

Los usuarios de páginas web normalmente se comunican utilizando un navegador web con un servidor con el fin de obtener una funcionalidad determinada. Esta funcionalidad puede abarcar desde la visualización de información (textual o multimedia) hasta la administración de un sitio web pasando por el almacenamiento de información, actualización de contenidos u otras operaciones más complejas. Con el fin de dar soporte a todas estas potenciales funcionalidades es absolutamente necesario ser capaces de ofrecer al cliente la información que necesita. La ejecución de código escrito en un lenguaje concreto por parte del servidor será responsable de generar dicha información.

En este capítulo nos centraremos en analizar las características fundamentales de PHP como ejemplo de lenguaje de programación web del lado del servidor. PHP (*Hypertext Preprocessor*) es un lenguaje de *scripting* de propósito general y de código abierto que ha sido especialmente diseñado para el desarrollo de aplicaciones web y que puede ser embebido (intercalado) en código HTML. En capítulos sucesivos mostraremos los ejemplos de código correspondientes a PHP, ASP y JSP.

PHP nació en 1994 a partir de un conjunto de ficheros binarios que se utilizaban para recoger datos de tráfico de datos en sitios web. En 1997 el intérprete (*parser*) de este lenguaje fue reescrito con el fin de dar soporte a la generación dinámica de contenido web y de crear aplicaciones del lado del servidor. Desde su versión 4, publicada en el año 2000, hasta su versión más actual PHP se ha convertido en un lenguaje de gran popularidad entre la comunidad de desarrolladores web. Entre las razones de su éxito destaca una relativa facilidad para ser aprendido.

PHP sirve como ejemplo paradigmático para comprender la arquitectura de un servidor web e identificar los componentes implicados durante el procesamiento de una aplicación web en respuesta a una solicitud del cliente. Las capas de una arquitectura genérica de un servidor que soporte la ejecución de PHP pueden verse en la Figura 2.1.

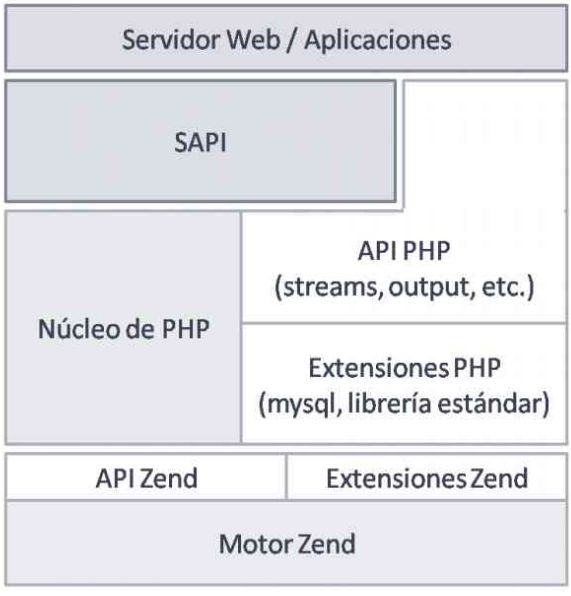


Figura 2.1. *Arquitectura genérica de PHP con Zend*

La capa más externa de esta arquitectura la conforma el propio servidor web, que es quien recibe las peticiones que el cliente hace a una dirección determinada. La capa inmediatamente inferior es la capa SAPI (*Server Abstraction API*) donde PHP interactúa con el servidor u otras aplicaciones. Esta capa SAPI gestiona, parcialmente, la inicialización y cierre de PHP como parte integrante de un servidor, contiene diferentes mecanismos para acceder a datos del cliente tales como las *cookies* o la información enviada por el método POST.

Por debajo de la capa SAPI se encuentra el motor de PHP propiamente dicho. El *núcleo de PHP* maneja la configuración del entorno de ejecución (asignando valores a variables globales y estableciendo las directivas de inicio del fichero de configuración `php.ini`) ofreciendo *interfaces* de programación (API PHP) tales como la interfaz de entrada/salida estándar, transformación de datos (*data parsing*) y, quizás lo más importante, ofreciendo un interfaz para cargar extensiones, tanto aquellas compiladas estáticamente como las cargadas dinámicamente.

En el corazón de la arquitectura de PHP se encuentra el motor Zend, que es el componente encargado del análisis y ejecución de las porciones de código PHP (*scripts*). El motor Zend, al igual que el de PHP, ha sido diseñado con el fin de admitir extensiones que permiten modificar completamente su funcionalidad (compilación, ejecución y manejo de errores) o parte de ella (cambiando las operaciones que se ejecutarán en caso de encontrarse un error).

Los pasos que se dan dentro del motor Zend cuando se ejecuta un *script* PHP son los siguientes:

1. El *script* es analizado por un analizador léxico (*lexer*) que transforma el código escrito por el desarrollador en un conjunto de piezas (*tokens*) entendibles por la máquina. Estos *tokens* son pasados después al analizador sintáctico (*parser*).

2. En el siguiente paso, el *parser* toma el conjunto de *tokens* y genera un conjunto de instrucciones (o código intermedio) que es ejecutado por el motor Zend. Este motor Zend actúa como máquina virtual que recibe como entrada el código intermedio de instrucciones y las ejecuta. Muchos *parsers* generan un árbol sintáctico que puede ser manipulado u optimizado antes de ser enviado al generador de código. El *parser* del motor Zend combina todos estos pasos en uno y genera el código intermedio directamente a partir de los *tokens* enviados por el analizador léxico. Este proceso se conoce con el nombre de *compilación*.

3. Después de que el código intermedio ha sido generado, la última fase es la *ejecución* donde interviene el componente ejecutor de Zend. Su función es ejecutar una por una las instrucciones indicadas en el código intermedio. Las fases de compilación y ejecución en el motor Zend se implementan internamente como funciones que son accesibles desde el exterior por lo que es posible crear extensiones que modifiquen el comportamiento básico de cada una de estas fases.

¿SABÍAS QUE...?

El motor Zend funciona como una *máquina virtual*, es decir, se trata de un programa que simula una computadora física. El motor Zend implementa alrededor de 150 instrucciones específicas y optimizadas para la ejecución de código PHP (como inclusión de librerías o imprimir un *string*).

2.3 ETIQUETAS PARA INSERCIÓN DE CÓDIGO

Como ya hemos comentado, la programación en el entorno del servidor, en el caso de los lenguajes embebidos, se centra en la inserción de código propio en páginas HTML. Cada uno de los lenguajes que se pueden utilizar para insertar código dentro de una página web utiliza una serie de etiquetas para delimitar los fragmentos de código que han de ser procesados por el servidor web. Independientemente del lenguaje utilizado (sea PHP, ASP, JSP u otros lenguajes de *scripting* de servidor) el componente del servidor encargado de procesar el código ignorará el código HTML que se encuentra fuera de dichas etiquetas.

El siguiente código muestra esta circunstancia:

```
<html>
  <head>
    <title>Primer ejemplo</title>
  </head>
  <body>
    <h1>Primer ejemplo: <br/></h1>
    <?php
      echo "Hola mundo.";
    ?>
  </body>
</html>
```

El ejemplo anterior muestra el primer fragmento de código PHP de este capítulo, el archiconocido "Hola mundo". Cuando el navegador web solicita este recurso, el fragmento de código PHP se ejecuta en el servidor y el código HTML resultante es enviado al cliente. El resultado es el que podemos ver en la siguiente pantalla:

Figura 2.2. *Renderizado en el navegador cliente del código PHP enviado*

- Obviamente, el *script* PHP mostrado en el ejemplo anterior es bastante sencillo y podríamos haber conseguido el mismo resultado editando el código HTML para incluir ese saludo directamente. Como podemos comprobar, el uso de PHP para insertar cadenas de caracteres estáticas en una página web es muy sencillo con PHP. Sin embargo, este sencillo ejemplo nos sirve para introducir algunas de las características básicas de los lenguajes de *scripting* de código embebido:

- Todo *script* en comienza y termina con una etiqueta de inicio y otra de fin. En PHP, estas etiquetas son `<?php` y `?>` respectivamente; en ASP se utilizan `<%` y `%>` y en JSP `<%=` y `%>`. En el caso de PHP también podemos usar `<?` y `?>` (denominadas *short tags*).
- Podemos configurar otros estilos de etiquetas en estos lenguajes. Por ejemplo, el estilo que utilizamos en HTML para JavaScript: `<script language="PHP">` y `</script>`.
- Los espacios en blanco que escribamos dentro del código embebido no tienen ningún efecto salvo para mejorar la legibilidad del código escrito. Cualquier combinación de espacios, tabuladores, retornos de carro y demás elementos usados para separar sentencias está permitida.
- El código de servidor embebido en páginas HTML está formado por un conjunto de sentencias que deben estar claramente separadas. En PHP y JSP esta separación se realiza con un punto y coma (`;`) mientras que en ASP esta separación se hace mediante retornos de carro.
- Los *scripts* embebidos pueden situarse en cualquier parte del recurso web ejecutado y puede ser intercalado en cualquier fragmento de HTML. El número de *scripts* que podemos tener dentro de un fichero HTML es indefinido.
- Cuando se ejecuta un código embebido, el *script* entero se sustituye por el resultado de dicha ejecución, incluidas las etiquetas de inicio y fin.

Para ilustrar estas características, podemos observar el siguiente fragmento de código que intercala código PHP en diferentes partes de un fichero HTML. En este caso podemos ver cómo hay tres fragmentos de código. El primero define el contenido de una variable (`$salida`) y los otros dos lo que hacen es escribir el valor de dicha variable entre el código HTML y en la posición en la que se encuentran los *scripts* PHP. Podemos ver cómo el valor asignado a la variable la utilizamos para indicar tanto el título de la página como el contenido de la misma.

```
<html>
  <?php
    $salida = "Contenido PHP";
  ?>
  <head>
    <title>
      <?php
        echo $salida;
      ?>
    </title>
  </head>
  <body>
    <h1>Segundo ejemplo: <br /></h1>
    <?php
      echo $salida;
    ?>
  </body>
</html>
```

Si bien es cierto que una de las múltiples ventajas del código embebido es precisamente la posibilidad de situar los *scripts* en cualquier parte del fichero HTML, esta circunstancia también puede resultar en un mantenimiento más difícil de la aplicación web desarrollada. Con el fin de disminuir los efectos de la dispersión del código de PHP podemos utilizar directivas de inclusión de código que veremos en capítulos posteriores.

2.3.1 COMENTARIOS

Otra forma de mejorar la legibilidad del código y que se recomienda siempre como una buena práctica a la hora de programar, es la inclusión de comentarios explicativos intercalados en el código.

En PHP, podemos incluir los comentarios utilizando un estilo fácilmente reconocible en otros lenguajes de alto nivel. Entre ellos destacamos los siguientes:

- ✓ Comentarios de una línea:

```
// esto es un comentario de una línea
# esto es otra forma de comentario de una línea
```

- ✓ Comentarios de múltiples líneas:

```
/* esta es la forma de
   escribir comentarios
   de varias líneas */
```

¿SABÍAS QUE...?

Comentar el código que escribimos se considera una buena práctica en el mundo de la programación. De esta forma, podemos hacer descripciones básicas del código escrito que puede llevar a confusión. Además, hacemos que el código desarrollado sea más fácil de mantener y compartir.

2.3.2 INCLUSIÓN DE CÓDIGO EN PÁGINAS HTML

Como hemos visto en los primeros ejemplos, una de las sentencias más habituales a la hora de mostrar el contenido en una página HTML es la instrucción "`echo`", que sirve tanto para cadenas de caracteres como para imprimir variables. Sin embargo, no es la única alternativa ya que también podemos utilizar la sentencia "`print`" como vemos en los siguientes ejemplos:

```
echo "ejemplo de impresión de cadena de caracteres";
// el resultado de usar print sería el mismo
print "ejemplo de impresión de cadena de caracteres";
// también se pueden imprimir números directamente
echo 215062;
// y mostrar el contenido de una variable
echo $variableResultado;
print $variableResultado;
```

La diferencia entre `echo` y `print` es que la sentencia `echo` puede imprimir más de un argumento, de la siguiente manera:

```
echo "imprimir primer argumento", " y el segundo";
```

Otra forma de mostrar el contenido de una variable es utilizando el símbolo igual junto a la etiqueta de inicio sin necesidad de usar `echo` o `print`:

```
<?=$variableResultado; ?>
```

Las sentencias `echo` y `print` son funciones que pertenecen a la librería estándar de PHP y, por ello, también podemos encontrárnoslas con los parámetros entre paréntesis:

```
echo "mensaje";
// es lo mismo que
echo ("mensaje");
```

Cuando utilizamos paréntesis con `echo` y `print` es necesario tener en cuenta el número máximo de parámetros admitidos por cada una de estas sentencias. En el caso de `echo`, el máximo es 1.

De cualquier forma, ambas sentencias pueden mostrar como resultado la combinación de cadenas de caracteres estáticas, números, vectores (*arrays*) y otros tipos de variables que comentaremos más adelante en este capítulo. Con respecto a las cadenas de caracteres debemos hacer una mención especial al manejo que PHP hace de las comillas simples y dobles cuando queremos que se muestren (o no) como resultado en el HTML enviado al cliente.

Tanto las comillas simples (`) como dobles (") se pueden usar para crear cadenas de caracteres.

```
echo 'Este texto se mostrará…';
echo " exactamente igual que éste.";
```

Si queremos mostrar comillas simples (o dobles) como parte de la salida, lo que debemos hacer es utilizar comillas dobles (o simples) como delimitadores de la cadena de caracteres de la siguiente forma:

```
// Ejemplo de cadenas de caracteres con comillas
echo "This string has a ': a single quote!";
echo 'This string has a ": a double quote!';
```

Otra forma de evitar que las comillas se consideren delimitadores es "escaparlas", es decir, indicarle al intérprete de PHP que el carácter a continuación de la barra invertida (\) es un carácter que debe imprimirse sin interpretarlo.

```
// Ejemplo de caracteres "escapados"
echo "Este texto contiene una \" (una comilla doble)";
echo 'Este texto contiene una \' (una comilla simple)';
```

Si queremos incluir otros caracteres especiales dentro de una cadena de caracteres delimitada por dobles comillas que vamos a insertar en el HTML que se le enviará al cliente, utilizaremos también la barra invertida. De esta forma, si queremos insertar una barra invertida en el texto de salida escribiremos `\\` y lo mismo sucedería con el símbolo `$` (`\$`).

```
echo "Este texto contiene una barra invertida: \\";
echo " y este un símbolo del dólar: \$";
```

Los ejemplos anteriores sirven para ilustrar la forma básica de insertar contenido estático o dinámico obtenido por la ejecución de cierto código en el lado del servidor utilizando PHP como lenguaje de *scripting*. Otros lenguajes de servidor como ASP o JSP tienen sentencias y funciones similares que permiten obtener el mismo resultado.

2.4 VARIABLES

Los datos manejados por el código de cualquier lenguaje de programación web se almacenan en variables. Estas variables son almacenes temporales de datos que permiten gestionar los datos utilizados por la aplicación web durante el flujo de ejecución de una página determinada. Como hemos podido comprobar en la sección anterior, toda variable tendrá asociado un tipo de datos.

2.4.1 DEFINICIÓN Y USO

Las variables en PHP se identifican por el símbolo del dólar (`$`) seguido del nombre de una variable. En PHP, las variables no necesitan ser declaradas explícitamente y, además, no tienen un tipo definido hasta que no se les asigna un valor. Si observamos el siguiente fragmento de código, le asignamos el valor 15 (de tipo `integer`) a la variable `$var`. A partir de ese momento, `$var` será de tipo `integer`:

```
$var = 15;
```

Puesto que la variable del ejemplo anterior la usamos para asignarle un valor, es en ese mismo instante cuando se declara implícitamente la variable (se reserva espacio de memoria para almacenar dicho valor). Esta circunstancia es común a todas las variables de PHP: la primera vez que se usan, su tipo es definido (o redefinido) y la variable es declarada. Como vimos en la sección anterior, el tipo de esta variable puede cambiar a lo largo de su vida:

```
$var = 15;
$var = "cambio de tipo";
```

Lo que sucede durante la ejecución de las sentencias anteriores es que el tipo de la variable `$var` cambia de `integer` a `string` conforme cambia el contenido que almacenan. Esta característica de PHP es una de sus mayores fortalezas y es lo que lo convierte en un lenguaje denominado *loosely typed*, es decir, que no requiere de una definición explícita de las variables ni de su tipo. Esta circunstancia hace que el uso de PHP sea más flexible pero también conlleva ciertos riesgos.

El nombre de una variable en PHP tiene que cumplir una serie de reglas:

- El nombre debe comenzar con una letra o con un guión bajo ("_").
- El nombre únicamente puede contener caracteres alfanuméricos y guiones bajos (a-z, A-Z, 0-9 y _).
- El nombre de una variable no debe contener espacios en blanco. Si queremos formar el nombre de una variable con más de una palabra lo que se suele hacer es utilizar un guión bajo entre ellas (`$mi_variable`) o poner la primera letra de cada palabra en mayúsculas (`$miVariable`).

¿SABÍAS QUE...?

Existen técnicas de "ofuscación" de código cuyo objetivo es realizar un cambio no destructivo que complique la lectura o comprensión del mismo por motivos diversos (por ejemplo dificultar la ingeniería inversa). Una forma básica de ofuscación es utilizar nombres de variables sin sentido o creados a partir del nombre de una palabra reservada levemente.

Un aspecto importante del uso de variables en PHP es que es *case-sensitive*. Esto quiere decir que las variables `$Variable`, `$variable`, `$Variable` y `$VARIABLE` son variables completamente diferentes.

Cometer un error con el nombre de una variable puede tener consecuencias graves tales como bucles que nunca terminan. En el contexto de una página web que tuviera este tipo de problemas, sería que el navegador web devolvería un error de *timeout* porque el servidor no sería capaz de terminar la ejecución de la página solicitada. Podemos ver un ejemplo en el siguiente fragmento de código:

```
for ($contador=0; $contador<10; $Contador++)
  miFuncion( );
```

La variable `$contador` nunca se incrementa. Al contrario, lo que sucede es que otra variable, `$Contador`, es incrementada cada vez que se ejecuta el bucle. De esta forma, `$contador` siempre es menor que 10 con lo que nunca se sale del bucle. Entre los errores más habituales que podemos encontrarnos están cambios en una letra del nombre de una variable, que letras estén en mayúsculas o minúsculas, omitir guiones bajos o simples errores tipográficos.

2.4.2 TIPOS DE DATOS Y VARIABLES

Como ya vimos en secciones anteriores, podemos incluir una variable como parte de una cadena de caracteres. PHP transformará el contenido existente entre unas dobles comillas por texto y reemplazará el nombre de las variables con el valor que tengan asignado en el momento de ejecutar dicha sentencia. Podemos verlo en el siguiente ejemplo:

```
$hora = 12;
$minutos = 15;
$ciudad = "Sevilla";
$texto = "El tren destino $ciudad sale a las $hora:$minutos";
echo $texto
// la salida será El tren destino Sevilla sale a las 12:15
```

En PHP podemos diferenciar tipos escalares (`boolean`, `integer`, `float` y `string`), tipos compuestos (`array` y `object`), especiales (`NULL` y `resource`) y pseudo-tipos (`mixed`, `number` y `callback`). Las variables que son de tipo escalar pueden contener un único valor cada vez. Las variables de tipo compuesto, como `arrays` y `objects`, están formadas por múltiples valores escalares u otros valores compuestos. Cuando una función puede admitir (o devolver) parámetros de diferente tipo, podemos utilizar el pseudo-tipo `mixed` para indicarlo. A continuación describiremos más en detalle las características de estos tipos de datos.

Las variables de tipo `boolean` son las más sencillas: únicamente pueden contener los valores `true` (cierto) y `false` (falso), tal y como vemos en los siguientes ejemplos:

```
$variable = false;
$test = true;
```

Una variable que almacena un tipo `integer`, contiene un valor entero (positivo o negativo) mientras que una variable de tipo `float` contendrá números reales con una parte entera y una parte decimal (separado por un punto). Veamos algunos ejemplos:

```
// Esto es un integer
$var1 = 7;
// Esto es un float
$var2 = 5.12;
```

Una variable que contenga un tipo `float` también puede representarse utilizando una notación exponencial:

```
// Esto es un float que representa al número 12340
$var3 = 1.234e4;
// Esta variable contiene un valor equivalente a 0.04
$var4 = 4e-2
```

Las cadenas de caracteres (tipo `string`) ya las vimos cuando introdujimos las funciones `echo` y `print`. Veamos dos ejemplos más:

```
$variable = "Esto es una cadena de caracteres";
$test = 'Esto también es un string';
```

PHP también permite la definición de valores constantes cuyo contenido no varía durante la ejecución del código. La forma de definir constantes en PHP es mediante la función `define()` que admite dos argumentos obligatorios: un nombre y un valor que se asociará a partir de ese momento a ese nombre. El siguiente ejemplo ilustra esta circunstancia:

```
define("PI", 3.14159);
echo PI;
// La salida sería 3.14159
```

Podemos observar cómo las constantes no llevan el símbolo `$` delante del nombre. Además, su valor no puede cambiarse una vez han sido definidas, pueden ser accedidas desde cualquier fragmento del fichero PHP en ejecución y la única restricción acerca de su valor es que únicamente pueden contener valores escalares (ni `arrays` ni `objects`).

Hasta ahora, hemos visto ejemplos muy sencillos de asignación de valores a variables utilizando el signo "=". La mayoría de las asignaciones numéricas y expresiones que se dan en otros lenguajes de alto nivel también funcionan en PHP. Veamos algunos ejemplos:

```
// Sumar enteros para producir un entero
$var = 2 + 2;

// Resta, multiplicación y división
// que dan como resultado un entero o un valor
// dependiendo del valor inicial de la variable
$var = (($var - 3) * 4) / 2;

// Varias formas de añadir 1 a la variable $var
$var = $var + 1;
$var += 1;
$var++;

// Lo mismo pero con la resta
$var = $var - 1;
$var -= 1;
$var--;

// Duplicar un valor
$var = $var * 2;
$var *= 2;
```

```
// Dividirlo por 2
$var = $var / 2;
$var /= 2;
```

¿SABÍAS QUE...?

Hay muchas funciones matemáticas, estadísticas y de manejo de variables numéricas disponibles en la librería matemática estándar de PHP.

Con respecto a la asignación y manejo de cadenas de caracteres, PHP se comporta de manera igualmente sencilla:

```
// Asignar un valor de texto a una variable
$var = "test string";

// Concatenar 2 cadenas
// para producir un resultado unificado
$var = "cadena" . " unida";

// Añadir un texto al final de una cadena de caracteres
$var = "cadena";
$var = $var . " unida";

// El operador de concatenación
// también tiene su versión corta
$var .= " test";
```

2.4.3 CONVERSIONES ENTRE TIPOS DE DATOS

En la sección anterior hemos visto cómo se codifica el contenido que finalmente visualizaremos en el navegador del cliente. La ejecución de una sentencia `echo` o `print` siempre dará, como resultado, una cadena de caracteres. Esto se debe a que el contenido de un documento HTML siempre se codifica en modo texto. Sin embargo, cuando se trabaja con cualquier lenguaje de programación de alto nivel, es habitual que los lenguajes permitan la identificación y utilización de diferentes tipos de datos, además de las consabidas cadenas de caracteres.

La conversión entre los diferentes tipos de datos es, además, algo habitual cuando se trabaja en entornos web. Por ejemplo, a la hora de enviar la información al cliente, el resultado finalmente mandado no será el mismo si lo que se manda es una fecha que si es un número real. Además, las operaciones más habituales con datos requieren que los operadores sean del mismo tipo, de ahí la importancia de conocer los procesos de conversión de datos.

PHP ofrece diferentes mecanismos que permite transformar las variables de un tipo en otro. Algunas de ellas son:

```
string strval(mixed variable)
integer intval(mixed variable)
float floatval(mixed variable)
```

Además, existen funciones específicas de gran utilidad que permiten establecer el tipo de una variable. Es el caso de la función `settype(mixed variable, string type)` donde el parámetro "`variable`" admite valores de tipo `array`, `boolean`, `float`, `integer`, `object` o `string` y el parámetro "`type`" es una cadena de caracteres que indica el tipo al que queremos transformar el parámetro "`variable`".

PHP soporta también el cambio de tipo de datos de forma similar a cómo se hace en lenguaje C. Para ello, podemos escribir entre paréntesis el tipo deseado justo delante del nombre de una variable, de tal forma que cuando se ejecute esa sentencia, el tipo resultante de la variable será el indicado entre paréntesis. La siguiente tabla muestra un resumen de este tipo de conversiones.

Tabla 2.1 Conversiones implícitas de tipo de datos en PHP

Sentencia	Resultado
`(int) $var` `(integer) $var`	Conversión a tipo `integer`.
`(bool) $var` `(boolean) $var`	Conversión a tipo `boolean`.

`(float) $var`
`(double) $var`
`(real) $var`
	Conversión a tipo `float`.

`(string) $var`	Conversión a tipo `string`.
`(array) $var`	Conversión a tipo `array`.
`(object) $var`	Conversión a tipo `object`.

Las reglas de conversión de datos (*cast rules*) son en su mayoría de sentido común, sin embargo, hay algunas conversiones que no son tan intuitivas. La siguiente tabla muestra cómo diferentes valores asignados a una variable denominada `$var` son convertidos utilizando los operadores de conversión `(int)`, `(bool)`, `(string)` y `(float)`.

Tabla 2.2 Ejemplos de conversión de datos en PHP

Valor de $var	(int) $var	(bool) $var	(string) $var	(float) $var
null	0	false	""	0
true	1	true	"1"	1
false	0	false	""	0
0	0	false	"0"	0
3.8	3	true	"3.8"	3.8
"0"	0	false	"0"	0
"10"	10	true	"10"	10
"6 metros"	6	true	"6 metros"	6
"hola"	0	true	"hola"	0

Otra peculiaridad de la conversión de tipos de datos en PHP se da cuando combinamos en una misma expresión dos variables que inicialmente tienen tipos diferentes o cuando pasamos una variable como argumento a una función que espera un tipo de dato diferente. En estos casos en los que una variable de un tipo se utiliza como si fuera de otro tipo, PHP convierte automáticamente la variable a un valor del tipo requerido (obteniéndose un resultado como el mostrado en la Tabla 2.2).

Veamos algunos ejemplos que muestran qué ocurre cuando sumamos enteros (`integer`) o reales (`float`) a cadenas de caracteres o cuando estos son concatenados a cadenas de caracteres:

```
// $var se convierte a tipo integer con valor 35
$var = "20" + 15;

// $var se convierte a tipo float con valor = 35.1
$var = "20" + 15.1;

// $var se convierte a string con valor = "20 años"
$var = 20 . " años";
```

No todas las conversiones de datos son tan obvias y en ocasiones son el motivo de numerosos errores en el código difíciles de detectar:

```
// $var se convierte a tipo integer con valor = 20
// Obsérvese la diferencia de usar "." ó "+"
$var = 20 + " años";

// $var se convierte a tipo integer con valor = 42
$var = 40 + "2 razones";

// $var se convierte a tipo float...
// ...pero no tiene ningún sentido el primer operando
$var = "prueba" * 4 + 3.14159;
```

La conversión automática de tipos debido a la utilización de ciertos operadores puede cambiar el tipo de una variable. Es lo que ocurre en los siguientes ejemplos:

```
$var = "1"; // $var es un string con valor = "1"
$var += 2;  // $var ahora es un integer con valor = 3
$var /= 2;  // $var ahora es un float con valor = 1.5
$var *= 2;  // $var sigue siendo un float con valor = 3
```

Debemos tener un cuidado especial cuando manejemos valores no `boolean` como si fueran de ese tipo. Esta situación se da en muchas ocasiones porque algunas de las funciones de la librería estándar devuelven un valor "`false`" (de tipo `boolean`) cuando el valor resultado no puede devolverse. Es importante que tengamos en cuenta que un valor devuelto como `0`, `0.0`, "`0`", un *string* vacío, `null` o un *array* vacío se interpreta siempre como "`false`" cuando se utilizan como valores `boolean`.

La solución a este tipo de problemas pasa por preguntar, siempre, por el tipo de la variable devuelta como resultado de la invocación de una función. Para ello, PHP ofrece una serie de funciones que permiten comprobar cuál es el tipo de una variable:

```
boolean is_int(mixed variable)
boolean is_float(mixed variable)
boolean is_bool(mixed variable)
boolean is_string(mixed variable)
boolean is_array(mixed variable)
boolean is_object(mixed variable)
```

Como podemos ver, todas las funciones anteriores devuelven un valor booleano "cierto" (`true`) o "falso" (`false`) para la variable pasada como argumento dependiendo de si dicha variable coincide o no con el nombre de la función que se está invocando. Por ejemplo, el siguiente fragmento de código imprime el valor 1 ya que la variable `$prueba` pasada como argumento contiene un valor de tipo `float`.

```
$prueba = 13.0;
echo is_float($prueba); // imprime 1 (significa true)
```

Cuando estamos desarrollando aplicaciones web con PHP, en muchas ocasiones nos interesa ver por pantalla cuál es el tipo de una variable determinada. Para ello, PHP nos ofrece las funciones `print_r()` y `var_dump( )`, que imprimen el tipo de una variable y su valor asignado en ese momento.

```
$variable = 15;
var_dump($variable);
```

La salida será: `int 15`.

A modo de resumen, no debemos olvidar que la información manejada por los lenguajes de programación web del lado del servidor puede ser de naturaleza muy diversa. Al igual que en cualquier lenguaje de programación, la distinción de diferentes tipos de datos es un aspecto crucial a la hora de desarrollar aplicaciones.

2.4.4 PRECEDENCIA DE OPERADOR

La precedencia de operador en una expresión es similar a la precedencia definida en cualquier otro lenguaje: la multiplicación y la división se realizan antes que la resta y la suma, etc. Sin embargo, depender de la precedencia de operador en ocasiones nos lleva a escribir un código farragoso y difícil de entender que a menudo resulta en un aumento del número de errores. Por ello, lo que se recomienda normalmente es utilizar paréntesis para romper la precedencia de operador y clarificar qué operaciones se realizarán antes que otras. Esto se debe a que los paréntesis tienen la máxima precedencia.

Tomemos por ejemplo la siguiente expresión:

```
$variable = 2 + 5 * 6;
```

A la variable `$operacion` le asignamos el valor 32 porque la multiplicación tiene más prioridad que la suma. Sin embargo, el resultado se ve mucho más claro si se utilizan paréntesis:

```
$variable = 2 + (5 * 6);
```

¿SABÍAS QUE...?

La forma de indicar el operador de negación difiere de unos lenguajes a otros. En PHP y JSP se indica con una "`!`" mientras que ASP se utiliza la palabra reservada `not`.

2.4.5 ESTADO DE UNA VARIABLE

Durante la ejecución de un *script* PHP, una variable puede estar en un estado indeterminado (no tener un valor asignado) e incluso puede no haber sido definida. Para comprobar el estado de una variable, PHP ofrece las funciones `isset()` y `empty()`.

```
boolean isset(mixed var)
boolean empty(mixed var)
```

La función `isset()` comprueba si a una variable se le ha asignado un valor no nulo, mientras que la función `empty()` comprueba si esa variable tiene un valor. A pesar de que las diferencias son sutiles, es importante que conozcamos su uso para poder dominar la gestión de variables. Veamos unos ejemplos:

```
$variable = "prueba";

// Comprobar si la variable está definida:
// imprimirá el texto "variable está definida"
if (isset($var)) echo "variable está definida";

// Comprobar si la variable está vacía:
// no imprimirá nada
if (empty($var)) echo "variable está vacía";
```

Una variable puede ser "destruida" utilizando la función `unset()`:

```
unset(mixed var [, mixed var [, ...]])
```

Después de invocar la función `unset()` en el siguiente ejemplo, la variable "`$var`" estará en un estado "no definida":

```
$var = "algo";
unset($var);
// La siguiente sentencia no imprime nada
if (isset($var)) echo "var está definida";
```

Otra forma de comprobar si una variable está vacía es forzar a que su tipo sea `boolean` (utilizando el operador `(bool)` indicado con anterioridad) y comprobar si se interpreta a `true` o a `false`. El ejemplo interpreta la variable `$var` como de tipo `boolean`, que es equivalente a comprobar si `$var` no está vacía con `!empty($var)`:

```
$var = "foo";
// Ambas sentencias imprimirán lo mismo
if ((bool)$var)    echo "var no está vacía";
if (!empty($var))  echo "var no está vacía";
```

La Tabla 2.3 muestra los valores de retorno para la ejecución de las funciones `isset()` y `empty()` y la aplicación del operador de conversión `(bool)`. Podemos observar como PHP devuelve algunos resultados extraños como que `empty()` devuelve `true` cuando le asignamos el valor "0".

Tabla 2.3 Comprobación del estado de una variable

Contenido de $var	isset($var)	empty($var)	(bool) $var
`$var = null;`	false	true	false
`$var = 0;`	true	true	false
`$var = true`	true	false	true
`$var = false`	true	true	false
`$var = "0";`	true	true	false
`$var = "";`	true	true	false
`$var = "foo";`	true	false	true
`$var = array( );`	true	true	false
`unset ($var);`	false	true	false

2.4.6 ÁMBITO DE LAS VARIABLES

Un aspecto importante a la hora de manejar variables en las aplicaciones web de servidor es su ámbito. Se entiende por ámbito al contexto dentro del que la variable está definida, es decir, la zona del programa en la que puede ser accedida. De esta forma, es posible hablar de ámbito local o ámbito global. Además, existe el concepto de variable de sesión que veremos en un capítulo más avanzado.

Las variables utilizadas dentro de una función son diferentes de aquellas que se utilizan fuera de ella. De tal forma que las variables internas a una función única y exclusivamente pueden ser utilizadas dentro de dicha función. El siguiente ejemplo ilustra esta circunstancia:

```
function duplicar($var){
  $temp = $var * 2;
}
$variable = 5;
duplicar($variable);
echo "El valor de la variable \$temp es: $temp";
```

Este ejemplo devolvería la siguiente salida…

```
El valor de la variable $temp es:
```

… y ningún valor para `$temp`. En este caso se dice que el ámbito de la variable `$temp` es local a la función `duplicar()` y se descarta (se destruye) al salir de dicha función.

Como podemos observar, el motor de PHP no devuelve ningún mensaje de error asociado a no encontrar la variable `$temp`. Simplemente asume que la variable está vacía. Sin embargo, una de las características configurables de las extensiones para PHP de los servidores web es que podemos indicar al motor de PHP que muestre un aviso acerca de aquellas variables no declaradas.

Si lo que queremos es utilizar un valor que es local para una función en cualquier otra parte del código, la forma más fácil que tenemos de hacerlo es devolviendo el valor de la variable. Nuestro ejemplo anterior quedaría de la siguiente forma:

```
function duplicar($var){
$resultado=$var*2;
      return ($resultado);
}
$variable = 5;
$temp = duplicar($variable);
echo "El valor de la variable \$temp es: $temp";
```

Este ejemplo devolvería la siguiente salida…

```
El valor de la variable $temp es: 10
```

Podríamos haber utilizado el nombre de la variable `$temp` dentro de la función `duplicar()`. Sin embargo, la variable `$temp` interna a la función sería diferente de la que utilizaríamos para recoger el valor devuelto por la función. La regla general nos dice que las variables utilizadas exclusivamente dentro de las funciones son locales a esas funciones, independientemente de si existe o se usa una variable que tiene un nombre idéntico fuera de esas variables. Como veremos en capítulos más adelante, existe una excepción a esta regla: las variables definidas "por referencia" y aquellas variables declaradas como "globales" en la función no son locales a dicha función.

Si queremos utilizar la misma variable en cualquier parte del código, incluido dentro de las funciones, podemos servirnos de la sentencia "`global`". Esta sentencia declara que una variable dentro de una función es la misma que la variable que hemos utilizado (o utilizaremos) fuera de esa función. Veámoslo en un ejemplo:

```
function duplicar(  ){
  global $temp;
  $temp = $temp * 2;
}
$temp = 5;
duplicar ();
echo "El valor de la variable \$temp es: $temp";
```

Puesto que `$temp` ha sido declarada dentro de la función como variable global, dicha variable se entenderá como una variable global que puede ser accedida desde fuera de la función `duplicar()`. Como consecuencia de declarar la variable `$temp` como global, la salida del *script* anterior será la siguiente:

```
El valor de la variable $temp es: 10
```

Debemos indicar, no obstante, que la utilización de variables globales sin control puede resultar en un código difícil de mantener y propenso a dar errores. Por otro lado, existen diferencias en cómo debemos declarar una variable global en PHP y en otros lenguajes de *scripting* de servidor. En PHP la calificación de global de una variable se realiza dentro de la función donde la utilizaremos. En ASP y JSP es al contrario, la calidad de global se indica en el programa principal o fuera del ámbito de la función.

En ocasiones, el uso de variables globales se ha extendido por un programa como una forma fácil de devolver varios valores como resultado de la ejecución de una función. Si lo que necesitamos es devolver diferentes valores, podemos optar por alternativas más elegantes y menos peligrosas como definir los argumentos de la función como "por referencia" en vez de "por valor" o bien hacer que la función devuelva un *array* (tipo de datos compuesto).

¿SABÍAS QUE...?

Existe el concepto de "superglobalidad", más genérico que el de variable global, que permite compartir una variable entre diferentes páginas. Dos ejemplos de variables superglobales son `$_GET` y `$_POST`, utilizadas para acceder a la información enviada por un cliente.

ACTIVIDADES 2.1

- Se propone descargar e instalar el servidor Apache para que se pueda ejecutar PHP. Para ello, se puede descargar el servidor desde su página *http://httpd.apache.org/* y el módulo de PHP desde *http://www.php.net/downloads.php*. De esta forma, el alumno podrá probar todos los ejercicios que se indican en el capítulo.
- En este capítulo se ha descrito la arquitectura del motor de ejecución de PHP sobre Zend. Se propone al alumno que estudie tanto los componentes del servidor IIS que permiten ejecutar código ASP como los componentes necesarios para la ejecución de páginas desarrolladas en JSP.
- Los ejemplos mostrados en este capítulo se han codificado con PHP. Se propone que el alumno trabaje las características estudiadas (variables, comentarios, tipos de de datos, etc.) en los lenguajes ASP y JSP.

RESUMEN DEL CAPÍTULO

El desarrollo de aplicaciones del lado del servidor exige la existencia de un componente software especial denominado *servidor web*. La misión de este componente es la de escuchar las peticiones realizadas por un cliente y devolver un documento HTML como respuesta. Para conseguir el código que debe enviarse al cliente, el servidor deberá ser capaz de ejecutar el código asociado al recurso solicitado por el cliente.

Los servidores web pueden ser de diferentes tipos en función de la estrategia de gestión de procesos e hilos que implementen. De esta forma, podemos encontrar servidores basados en procesos, en hilos, implementados sobre el núcleo del sistema operativo, etc. Algunos de los servidores web más extendidos son: Apache Server, Microsoft Internet Information Server o Sun Java System Web Server. Todos ellos contienen módulos específicos para poder ejecutar código implementado en diferentes lenguajes, tales como PHP, ASP o JSP.

PHP es un lenguaje de *scripting* de gran aceptación entre la comunidad de desarrolladores de aplicaciones web ejecutable en prácticamente cualquier tipo de servidor. A la hora de programar aplicaciones web utilizando lenguajes de *scripting* es necesario tener en cuenta que el código puede intercalarse entre el código HTML con el fin de poder personalizar y adecuar la respuesta del servidor web al cliente.

Conocer la sintaxis básica de un lenguaje como PHP incluye dominar la definición de variables, los tipos de datos permitidos y su conversión o la precedencia de operador. Como en cualquier lenguaje de programación, se recomienda el uso de comentarios para poder facilitar el mantenimiento del código.

EJERCICIOS PROPUESTOS

1. Cree un fichero que contenga las etiquetas HTML mínimas para mostrar un mensaje de texto (`<html>`, `<head>`, `<title>` y `<body>`) y guárdelo con extensión PHP. Posiciónelo en el directorio correspondiente del servidor web. A continuación escriba una sentencia `echo` o `print` que permita mostrar por pantalla el mensaje "Este es el resultado correcto del primer ejercicio". Abra un navegador y compruebe el resultado. Si obtiene una página en blanco, verifique que ha utilizado las etiquetas de apertura y cierre de PHP correctamente y que la sentencia que ha escrito termina en "`;`".

2. Cree un fichero PHP que muestre por pantalla el mensaje "Segundo ejercicio: visualización del contenido de variables". A continuación, defina dos variables `$nombre` y `$edad` y asígnales un valor correcto. Después, cree una sentencia que muestre un mensaje que contenga dichas variables, similar a "Mi nombre es: `valor_de_la_variable_$nombre` y mi edad es `valor_de_la_variable_$edad`". Inserte un comentario encima de cada variable explicando el significado del valor que almacenará cada variable.

3. Cree un fichero PHP que permita comprobar las capacidades aritméticas de PHP. Para ello, cree dos variables `$operador1` y `$operador2`. Asígnele los valores 13 y 4, respectivamente. Defina una tercera variable `$resultado` y escriba un código que permita hacer las siguientes operaciones:

- 13 – 4
- 13 + 4
- 13 * 4
- 13 / 4
- 13 % 4

4. Cree un fichero de PHP que permita conocer toda la información de una variable (utiliza la función `var_dump()`), de tal forma que pueda obtener una salida por pantalla similar a la siguiente:

```
Información de la variable "nombre":
string (4) "Juan"
Contenido de la variable: Juan
Después de asignarle un valor nulo: NULL
```

5. Cree un fichero PHP en el que se asignen los siguientes valores, de forma consecutiva, a una variable`$temporal`: "Juan", 3.14, `false`, 3, `null`. Muestre por pantalla el tipo que se le asigna a la variable utilizando la función `gettype()` después de cada asignación.

TEST DE CONOCIMIENTOS

1 Señale la respuesta correcta respecto a los servidores basados en procesos:

a) Se basa en la obtención de un paralelismo de ejecución mediante la duplicación del proceso de ejecución.
b) Todos los procesos creados por un servidor comparten la misma zona de memoria.
c) Se implementan en el núcleo del sistema operativo.
d) Ninguna de las anteriores.

2 ¿Cuál de las siguientes afirmaciones es cierta con respecto a la sintaxis de PHP?

a) Los espacios en blanco que se escriban dentro del código embebido afectan al significado del código ejecutado.
b) No se permite la utilización de tabulaciones como elementos para separar sentencias en PHP.
c) El nombre de las variables distingue entre mayúsculas y minúsculas.
d) El nombre de una variable puede empezar por números o letras.

3 ¿Cuál de las siguientes combinaciones de caracteres sirve para definir un comentario de una línea en PHP?

a) `--`
b) `<!--`
c) `//`
d) `**`

4 De acuerdo a la precedencia de operadores en PHP, ¿cuál sería el resultado de ejecutar la siguiente operación? "10.5" + 4.5*2 / 3

a) 23.5
b) 13.5
c) 10
d) `null`

5 Indique el tipo de la variable `$resultado` en la siguiente operación: `$resultado = "100" + 10.5;`

a) `String`.
b) `Entero`.
c) `Float`.
d) `Boolean`.

3 Programación basada en lenguajes de marcas con código embebido

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer y utilizar los mecanismos disponibles de decisión para la creación de bloques de sentencias en diferentes lenguajes del servidor.
- ✓ Aprender a utilizar bucles. Conocer su sintaxis en diferentes lenguajes y verificar su comportamiento.
- ✓ Estudiar la forma de almacenar y recuperar conjuntos de datos utilizando variables compuestas (*arrays*).
- ✓ Aprender a utilizar y definir funciones. Conocer las más habituales en diferentes lenguajes.
- ✓ Comprender los diferentes métodos de recuperación de información enviada por un cliente web.
- ✓ Aprender a procesar la información enviada por el cliente a través de formularios.

El capítulo anterior tenía como objetivo mostrar la sintaxis básica de los lenguajes de programación del entorno del servidor. Sin embargo, para poder dominar la programación web, necesitamos conocer estructuras de control que nos permitan dotar de flexibilidad a las aplicaciones creadas. Para facilitar el desarrollo de aplicaciones web, necesitamos dominar, además, diferentes técnicas de subprogramación basadas en la definición y utilización de procedimientos y funciones. Con dichas funciones, bien sean creadas por el programador o bien de forma predeterminada por el lenguaje, una aplicación web debe ser capaz de gestionar las peticiones del cliente. Por último, un buen programador debe ser capaz de manejar con soltura estructuras de datos complejas, tales como los *arrays* o matrices. Además, en este capítulo estudiaremos los métodos básicos de comunicación con el cliente web y la forma de recuperar dicha información.

3.1 SENTENCIAS CONDICIONALES

Las sentencias condicionales son estructuras de control que permiten decidir el flujo de ejecución de un programa, es decir, el orden en el que las instrucciones de un programa se van ejecutar. Al introducir sentencias condicionales en nuestro código, este deja de ejecutar las instrucciones de manera secuencial, una detrás de otra, para poder definir caminos alternativos dependiendo de si se cumplen las condiciones establecidas por el programador. Más tarde, definiremos la sintaxis de las distintas sentencias condicionales que existen en los lenguajes ASP, PHP y JSP, como son las sentencias `if` y `switch`.

3.1.1 SENTENCIAS IF

Este tipo de estructuras de control condicionales definen dos flujos de ejecución dependiendo de si se cumple o no la condición establecida por el programador. Si se cumple la condición se ejecutará una instrucción o un grupo de instrucciones. Si de lo contrario no se cumple la condición se ejecutarán otras instrucciones distintas.

Para el caso en el que solo queremos ejecutar ciertas instrucciones si se cumple la condición la sintaxis es la siguiente:

- ✓ **PHP y JSP**:

```
if (condición){
   instrucciones;
}
```

- ✓ **ASP**:

```
if (condición) then
  instrucciones
end if
```

Como podemos ver existen ciertas diferencias a la hora de definir la sentencia `if` en los distintos lenguajes. Por ello tenemos que tener en cuenta que tanto en JSP como en PHP el uso de las llaves es opcional cuando un bloque de sentencias contiene únicamente una instrucción. También necesitamos saber que para separar unas instrucciones de otras se utiliza el símbolo "`;`". Por otro lado, en ASP las instrucciones no se encuentran entre llaves si no entre las palabras reservadas `then` y `end if` para marcar el final del bloque `if`. Además, utiliza el retorno de carro como separador de instrucciones.

Puede darse el caso de que necesitemos extender la funcionalidad del bloque `if` para controlar no solo qué instrucciones queremos que se ejecuten cuando se cumpla la condición, sino que además necesitamos especificar qué instrucciones queremos que se ejecuten cuando no se cumpla dicha condición. Esto se consigue haciendo uso del bloque `else`.

Figura 3.1. *Diagrama de control de flujo de la sentencia* `if-else`

✓ **PHP y JSP**:

```
if (condición){
  instrucciones1;
}else{
  instrucciones2;
}
```

✓ **ASP**:

```
if (condición) then
  instrucciones1
else
  instrucciones2
end if
```

Otra de las posibilidades que nos ofrece la sentencia `if` es definir varios `if` anidados que nos permitan evaluar varias condiciones previas antes de ejecutar las instrucciones correspondientes.

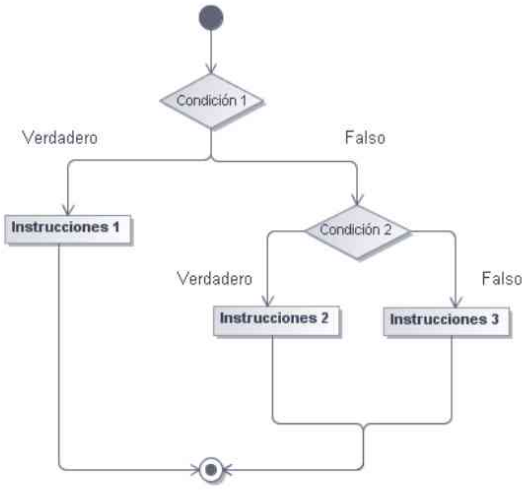


Figura 3.2. *Diagrama de control de flujo de la sentencia* `if-elseif`

- **PHP**:
  - **Forma 1**:

```
if (condición1){
  instrucciones1;
}elseif (condición2){
  instrucciones2;
}else{
  instrucciones3;
}
```

  - **Forma 2 (Válido también para JSP)**:

```
if (condición1){
  instrucciones1;
}else if (condición2){
  instrucciones2;
}else{
  instrucciones3;
}
```

- **ASP**:
  - **Forma 1**:

```
if (condición1) then
  instrucciones1
elseif (condición2) then
  instrucciones2
else
  instrucciones3
end if
```

- **Forma 2**:

```
if (condición1) then
  instrucciones1
else
  if (condición2) then
    instrucciones2
  else
    instrucciones3
  end if
end if
```

Con respecto a la forma de definir `if` anidados, destacar que PHP y ASP permiten tanto el uso de `elseif` como `else if` mientras que en JSP solo se puede escribir separado.

ACTIVIDADES 3.1

Pruebe el siguiente código en PHP y realice los cambios necesarios para que aparezcan por pantalla los distintos mensajes posibles. Además pruebe a hacerlo en JSP y ASP.

```
<html>
  <head>
    <title> Actividad 1 para PHP </title>
  </head>
  <body>
    <?php
      $i=5;

      if ($i<5){
      echo("i es menor a 5");
    }else if($i>5){
      echo("i es mayor a 5");
    }else{
       echo("i es igual a 5");
       }

     ?>
   </body>
  </html>
```

3.1.2 SENTENCIAS SWITCH O SELECT CASE

Estas sentencias se usan cuando dependiendo del valor que toma una variable o expresión, necesitamos que se ejecute un conjunto de instrucciones distintas para cada uno de los valores que pueda tomar.

Su funcionamiento es sencillo, primero se calcula el valor de la expresión y se compara dicho valor con cada uno de los casos. Y una vez que se encuentra el caso con el que coincide se ejecutan las instrucciones contenidas dentro del

caso correspondiente. Si al evaluar todos los casos no coincide con ninguno se ejecutan las instrucciones definidas en el bloque por defecto si es que está definido, pues la declaración de este caso es opcional.

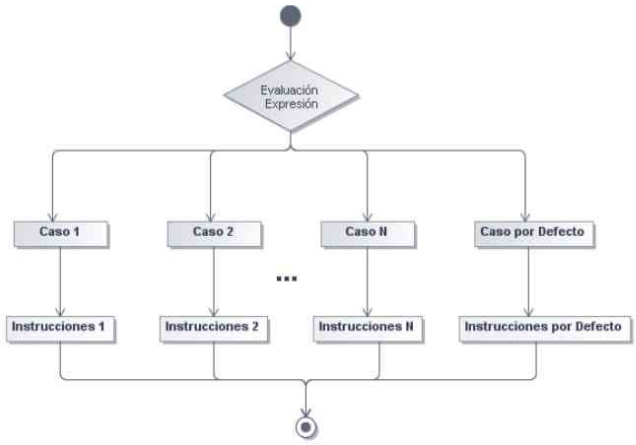


Figura 3.3. *Diagrama de control de flujo de la sentencia* `switch/select case`

✓ **PHP y JSP**:

```
switch (expresión){
  case valor1: instrucciones1; break;
  case valor2: instrucciones2; break;
  ...
  case valorN: instruccionesN; break;
  [default: instruccionesN+1;]
}
```

✓ **ASP**:

```
select case (expresión)
  case valor1
    instrucciones1
  case valor2
    instrucciones2
  ...
  case valorN
    instruccionesN
  [case else
    instruccionesN+1]
end select
```

En ciertas ocasiones podemos utilizar los `switchs` para sustituir a los `if` anidados, lo que es altamente recomendable debido a que es más eficiente computacionalmente el uso de estas sentencias, ya que ejecutan las instrucciones secuencialmente y evalúan la condición una sola vez. Además este tipo de sentencias presentan la ventaja de facilitar la lectura del código.

Con respecto a la sintaxis de estas sentencias señalar una de las diferencias más importantes que podemos apreciar entre los distintos lenguajes, que es el uso de `break` en JSP y PHP al final de la definición de cada caso. El uso de la sentencia `break` en JSP y PHP provoca que se interrumpa la evaluación de los casos siguientes una vez que se ha encontrado el caso que coincide con el valor resultante de evaluar la expresión. Otra de las diferencias que podemos encontrar es que mientras que para JSP y PHP el nombre que recibe el caso por defecto es `default`, para ASP es `case else`. Por último decir que en ASP el valor de los casos no va seguido de ":" y se indica el fin de la sentencia `select case` con la palabra reservada `end select`.

En JSP la expresión del `switch` no puede ser de tipo *string*.

ACTIVIDADES 3.2

Pruebe el siguiente código en JSP y realice los cambios necesarios para que aparezcan por pantalla los distintos mensajes posibles. Además pruebe a hacerlo en PHP y ASP.

```
<html>
  <head>
    <title> Actividad 2 para JSP </title>
  </head>
  <body>
    <%
      int codigo=1;

    switch (codigo){
        case 0:
          out.println("Has seleccionado el 0");
          break;
        case 1:
          out.println("Has seleccionado el 1");

          break;
        case 2:
        out.println("Has seleccionado el 2");

          break;
        case 3:
        out.println("Has seleccionado el 3");

          break;

        default:
          out.println("No sé cuál has seleccionado");
      }
    %>
  </body>
</html>
```

3.2 BUCLES

Este tipo de sentencias se utilizan para ejecutar de forma reiterativa una instrucción o grupo de instrucciones. Estas instrucciones se pueden ejecutar un número determinado o indeterminado de veces dependiendo del bucle que utilicemos para ello. En este apartado vamos a definir la sintaxis de cada uno de los bucles que existen para los lenguajes PHP, JSP y ASP, así como su comportamiento y su utilidad a la hora de programar. Los bucles que vamos a ver son los siguientes: `while`, `do-while`, `do until…loop`, `do…loop until`, `for` y `foreach`.

Al igual que en las sentencias condicionales, la condición puede ser simple o compuesta por varias expresiones lógicas y/o aritméticas cuyo valor resultante de evaluarlas es un valor booleano.

3.2.1 BUCLE WHILE O DO WHILE...LOOP

Este tipo de estructuras permiten ejecutar un número indeterminado de veces una instrucción o grupo de instrucciones, mientras se cumpla la condición. Como muestra la Figura 3.4 en cada iteración del bucle se evalúa la condición y si esta es verdadera pasan a ejecutarse las instrucciones contenidas en el cuerpo del bucle. Finalmente el bucle termina cuando el resultado de evaluar la condición es falso, es decir, cuando la condición ha dejado de cumplirse.

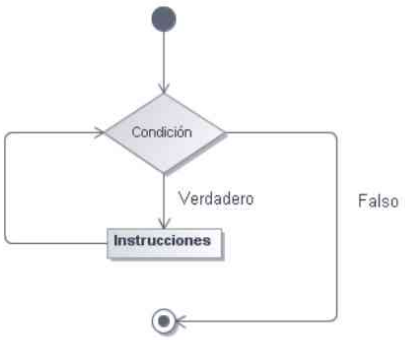


Figura 3.4. *Diagrama de control de flujo del bucle* `while`

A continuación mostramos su sintaxis:

- ✓ **PHP y JSP**:

```
while (condición){
  instrucciones;
}
```

- ✓ **ASP**:
  - **Forma 1**:

```
do while (condición)
  instrucciones
loop
```

- **Forma 2**:

```
while (condición)
  instrucciones
wend
```

Como podemos observar, la única diferencia entre la sintaxis utilizada por JSP y PHP, y la utilizada por ASP es el nombre que recibe el bucle para cada uno de ellos. Mientras que para JSP y PHP es `while`, para ASP es `do while... loop`.

Aunque ASP ofrece dos sintaxis distintas para definir este tipo de bucles, la más utilizada es la primera, ya que esta se refiere a la forma nueva de definir este bucle y la segunda es un vestigio de los inicios de Basic. Actualmente los intérpretes soportan también la segunda forma de definir este bucle, para los programadores reticentes a utilizar la nueva forma, pero es posible que en el futuro los intérpretes dejen de contemplarlo.

¿SABÍAS QUE...?

Todo bucle `while` puede expresarse como un bucle `for`, el cual estudiaremos más adelante.

3.2.2 BUCLE DO-WHILE O DO...LOOP WHILE

Este bucle es muy parecido al anterior con la salvedad de que siempre se ejecuta al menos una vez se cumpla o no la condición, debido a que la evaluación de la condición se realiza al final de cada iteración y no al principio. Como muestra la Figura 3.5, este bucle se ejecuta un número indeterminado de veces hasta que el resultado de evaluar la condición es falsa.

Figura 3.5. *Diagrama de control de flujo del bucle* `do-while`

La sintaxis para este bucle es la siguiente:

✓ **PHP y JSP**:

```
do{
  instrucciones;
}while (condición);
```

✓ **ASP**:

```
do
  instrucciones
loop while (condición)
```

La sintaxis para este bucle es idéntica para PHP y JSP, mientras que para ASP lo único que cambia es el nombre del bucle.

3.2.3 BUCLE DO UNTIL...LOOP

De los tres lenguajes que estamos estudiando solo el lenguaje ASP presenta este tipo de bucles, ya que tanto PHP como JSP pueden simular el mismo comportamiento haciendo uso del bucle `while`. Con la ayuda de la Figura 3.6 podemos analizar el comportamiento de este bucle, que como podemos ver la condición se evalúa al principio de cada iteración. Si la condición no se cumple se ejecuta las instrucciones que contiene y si el resultado de evaluar la condición es verdadero el bucle finaliza.

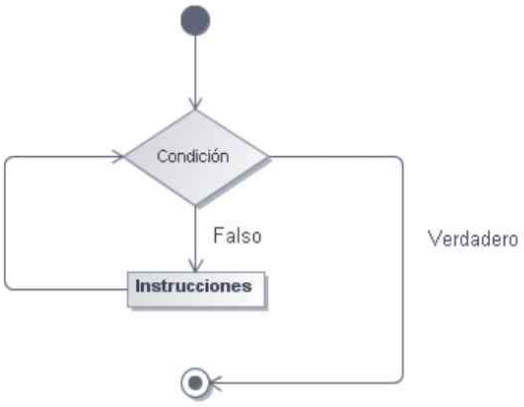


Figura 3.6. *Diagrama de control de flujo del bucle do* `until...loop`

La sintaxis de este bucle es la siguiente:

✓ **ASP**:

```
do until (condición)
  instrucciones
loop
```

3.2.4 BUCLE DO...LOOP UNTIL

Este bucle al igual que el anterior solo está presente en ASP, mientras que PHP y JSP pueden imitar su comportamiento haciendo uso del bucle `do-while`. El comportamiento de este bucle recuerda a los bucles `repeat` de otros lenguajes. Al igual que los bucles `repeat` se ejecuta las instrucciones hasta que la condición se cumple. Además la condición se evalúa al final de cada iteración del bucle, lo que significa que este bucle se ejecuta al menos una vez.

Figura 3.7. *Diagrama de control de flujo del bucle* `do...loop until`

Aquí podemos ver la sintaxis de este bucle:

✓ **ASP**:

```
do
  instrucciones
loop until (condición)
```

3.2.5 BUCLE FOR O FOR...NEXT

A diferencia del resto de bucles vistos hasta ahora este tipo de bucles se ejecutan un número determinado de veces. Al igual que el resto de bucles permite ejecutar un conjunto de instrucciones de forma repetitiva.

La Figura 3.8 muestra el comportamiento del bucle `for`. Primero se inicializa el contador, que es el que va a controlar el número de veces que se ejecuta el bucle. Seguido de esto se evalúa la condición y si es verdadera se ejecuta el contenido del bucle y se actualiza el contador. En el momento en el que el resultado de evaluar la condición resulta falsa, el bucle finaliza siguiendo con la ejecución de las instrucciones siguientes al bucle.

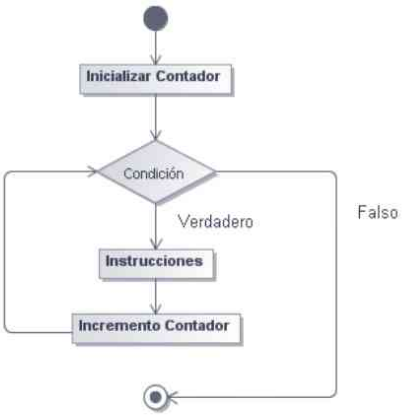


Figura 3.8. *Diagrama de control de flujo del bucle* `for`

✓ **PHP y JSP**:

```
for([contador=valorInicial];[condición];[incremento]){
  instrucciones;
}
```

✓ **ASP**:

```
for contador=valorInicial to valorFinal [step incremento]
  instrucciones
next
```

Como podemos observar, la sintaxis del bucle `for` de PHP y JSP cambia un poco con respecto al `for…next` de ASP. Para ello primero vamos a explicar cómo se define este bucle para PHP y JSP y posteriormente para ASP.

En el primer caso, para definir un bucle `for` necesitamos crear un contador y asignarle un valor inicial. También definiremos la condición que se evaluará al principio de cada iteración del bucle para comprobar si se debe seguir ejecutando o no. Y por último indicaremos que incremento se le va a aplicar al contador en cada ejecución del bucle expresado en forma de expresión aritmética. El bucle se ejecutará mientras se cumpla la condición. Dicha condición suele ser una comparación en el que está implicado el contador, y es cuando llega este a cierto valor cuando se sale del bucle y sigue ejecutando las instrucciones siguientes. Otra de las cosas que tenemos que tener en cuenta es que la declaración del contador es opcional, ya que lo podemos crear e inicializar fuera del bucle antes de este. Lo mismo pasa con el incremento y la condición que no es necesario definirla en el bucle si la definimos posteriormente dentro del mismo.

Con respecto al comportamiento y la definición del bucle `for…next` en ASP, la primera diferencia es que no necesita definir ninguna condición, si no que en su lugar se crea un contador y se le da un valor inicial. Además de indicar el valor inicial que toma también se especifica el valor que puede tomar como límite, cuando el contador toma el valor definido como límite se ejecuta por última vez el bucle y se sale de este para seguir con la ejecución secuencial de las siguientes instrucciones. De forma opcional podemos definir el valor con el que vamos a incrementar el contador en cada ejecución del bucle, pero si no lo definimos por defecto se incrementara el contador en uno.

3.2.6 BUCLE FOREACH

Este tipo de bucle lo encontramos en el leguaje de programación PHP y ASP y proporciona una forma sencilla de iterar sobre los elementos de un *array*, concepto que veremos más adelante. El bucle `foreach` funciona solamente con *arrays* y matrices, si intentamos utilizarlo con otro tipo de datos diferente o con una variable no inicializada notificará el error al usuario o programador.

Para definirlo podemos utilizar distintas sintaxis:

- **PHP**:
  - **Forma 1**:

```
foreach(variableArray as variableValor){
  instrucciones;
}
```

  - **Forma 2**:

```
foreach(variableArray as variableIndice => variableValor){
  instrucciones;
}
```

- **ASP**:

```
for each variableValor in variableArray
  instrucciones
next
```

El bucle `foreach` en ASP y la primera forma proporcionada en PHP se comportan de la misma forma. En ambos se recorre la variable de tipo *array* y en cada iteración, el valor del elemento actual se le asigna a la variable `variableValor`. Y además se avanza en uno el puntero interno del *array* de manera que en la siguiente iteración, se busca el siguiente elemento.

La segunda forma proporcionada en PHP se comporta exactamente igual que la primera, solo que además, se le asigna el índice del elemento actual del *array* a la variable `variableIndice` en cada iteración.

3.2.7 SENTENCIA BREAK

A veces necesitamos forzar la interrupción de la ejecución de un bucle, como por ejemplo ocurre cuando utilizamos los `switchs`. En el caso de las sentencias `switchs` no tiene sentido que sigamos evaluando el resto de casos si ya hemos encontrado el caso que coincide con el resultado de evaluar la expresión, pues estaríamos ejecutando instrucciones innecesarias pudiendo utilizar el procesador para ejecutar instrucciones útiles. Ya que finalmente lo que buscamos es programar de manera eficiente para que el procesador pase la mayor parte del tiempo ejecutando instrucciones provechosas. Este ejemplo es válido para PHP y JSP, ya que como hemos visto en apartados anteriores las estructuras equivalentes a los `switch` de estos lenguajes en ASP no llevan este tipo de sentencias para provocar la salida del bucle pero es necesaria en otros casos.

En el caso de ASP la sintaxis es la siguiente dependiendo del bucle al que se le obligue salir:

```
For…Next: exit for

Do While…Loop, Do Until…Loop, Do…Loop While o Do…Loop Until:
exit do

While…Wend: exit while
```

En el caso de PHP y JSP la sintaxis es más simple y es independiente del tipo de bucle:

- **PHP**: `break [número];`
- **JSP**: `break [etiqueta];`

Otro de los detalles que debemos saber para la utilización de estas sentencias es que pueden llevar un parámetro opcional. En el caso de PHP acepta un parámetro numérico adicional que indica cuántas estructuras o bucles hay que saltar. En cambio en JSP podemos definir etiquetas antes de los bucles y el parámetro que le podemos añadir a esta sentencia es el nombre de la etiqueta para indicar donde debe continuar la ejecución del programa.

Podemos observar a continuación la sintaxis para la utilización de la sentencia `break` con el parámetro de la etiqueta:

- **JSP**:

```
etiqueta1:
while(condición1){
  instrucciones;
  if(condición2){
    break etiqueta1;
  }
}
```

No debemos olvidar que esta sentencia se puede utilizar cuando tenemos uno o más bucles.

3.2.8 SENTENCIA CONTINUE

Esta sentencia se utiliza para saltar el resto de la iteración de un bucle y continuar con la evaluación de la condición y la siguiente iteración.

La sentencia `continue` solo existe en PHP y JSP, pero no en ASP donde se simula su comportamiento haciendo uso de otras estructuras de control.

La sintaxis es prácticamente igual a la de la sentencia `break` como podemos ver:

- **PHP**: `continue [número];`
- **JSP**: `continue [etiqueta];`

ACTIVIDADES 3.3

- Escriba el bucle `while` proporcionado en ASP para PHP y JSP:

```
<html>
   <head>
     <title> Actividad 3 para ASP </title>
   </head>
   <body>
     <%
      dim i
    i=0

    do while (i<5)
      response.write "Hola mundo<br>"
      i=i+1
    loop
     %>
  </body>
</html>
```

- Convierta el bucle del primer punto de este apartado en un bucle `do-while` para PHP y JSP y en un bucle `do while…loop` en el caso de ASP.
- Convierta el bucle del primer punto de este apartado en un bucle `do until…loop` para ASP.
- Convierta el bucle del primer punto de este apartado en un bucle `do…loop until` para ASP.
- Convierta el bucle del primer punto de este apartado en un bucle `for` para PHP y JSP y en un bucle `for…next` en el caso de ASP.

3.3 TIPOS DE DATOS COMPLEJOS

En este apartado vamos a aprender a definir, inicializar y acceder a los valores de un *array* o matriz, que no es más que un *array* multidimensional. Además explicaremos algunos algoritmos de ordenación, búsqueda e inserción de valores dentro de un *array*.

3.3.1 DEFINICIÓN Y ACCESO

Los *arrays* o matrices son estructuras que permiten el almacenamiento de un conjunto de datos; son una construcción tradicional de los lenguajes de programación. Podemos definir un *array* o matriz como un conjunto ordenado de elementos identificados por un índice (la posición del elemento dentro de esta colección ordenada), de modo que en cada posición marcada por un índice el *array* contiene un valor. Podemos construir tantos índices como queramos, aunque el uso habitual de los *arrays* es en forma de matriz unidimensional. La longitud del *array* se modifica de forma dinámica siempre que añadimos un nuevo elemento.

A continuación mostramos la sintaxis para definir un *array* en los distintos lenguajes:

- **PHP**: `nombreVariable=array(clave => valor,…);`
- **ASP**: `Dim nombreVariable(tamaño)`
- **JSP**:
  - **Forma 1**:

```
tipo[] nombreVariable  = new tipo [tamaño];
```

  - **Forma 2**:

```
tipo[] nombreVariable  = {valor1,valor2,…,valorN};
```

Una vez vista la sintaxis para declarar un *array*, vamos a ver un mismo ejemplo en todos los lenguajes que nos ayude a comprender cómo se declara e inicializa un *array*:

- **PHP**: `$miarray=array(0=>2,1=>4);`
- **ASP**:

```
Dim miarray(1)
miarray(0)=2
miarray(1)=4
```

- **JSP**:
  - **Forma 1**:

```
int[] miarray= new int [2];
miarray[0]=2;
miarray[1]=4;
```

  - **Forma 2**:

```
int[] miarray= {2,4};
```

Por otro lado para definir una matriz la sintaxis es la siguiente:

- **PHP**: `array(clave => array(),…);`
- **ASP**: `Dim nombreVariable(n° filas,n° columnas)`
- **JSP**:
  - **Forma 1**:

```
tipo[][] nombreVariable = new tipo [n°filas][n°columnas];
```

  - **Forma 2**:

```
tipo [][] nombreVariable = {valor1.1, valor1.2, …,
  valor1.N},…,{valorN.1,valorN.2,…,valorN.N}};
```

Al igual que con los *arrays*, vamos a ver con un mismo ejemplo para los distintos lenguajes como se define y se inicializa una matriz. Sobre estos ejemplos explicaremos la diferencia entre las distintas formas que existen para definirlos:

- **PHP**: `mimatriz(0 =>array(0=>2,1=>4),1 =>array(0=>1,1=>3));`
- **ASP**:

```
Dim mimatriz(1,1)
mimatriz(0)(0)=2
mimatriz(0)(1)=4
mimatriz(1)(0)=1
mimatriz(1)(1)=3
```

- **JSP**:
  - **Forma 1**:

```
int[][] mimatriz = new int [2][2];
mimatriz[0][0]=2;
mimatriz[0][1]=4;
mimatriz[1][0]=1;
mimatriz[1][1]=3;
```

  - **Forma 2**:

```
int [][] mimatriz = {{2,4},{1,3}};
```

Como podemos ver en los ejemplos aunque la sintaxis cambia el comportamiento es el mismo.

En PHP la definición de un *array* se define por el par clave-valor, en el que la clave hace referencia al índice y es opcional y el valor se refiere a lo que se va almacenar en esa posición del *array*. Mientras que la clave solo puede ser de tipo entero o `string` (cadena de caracteres), el valor puede ser de cualquier tipo de dato.

Para definir un *array* en ASP se utiliza la palabra reservada `dim` seguido por el nombre de la variable y el tamaño que tendrá entre paréntesis. Con respecto al tamaño tenemos que tener en cuenta que si por ejemplo lo definimos con tamaño 3, será uno más el número de elementos que podremos almacenar, en este caso serían 4. Esto es debido a que el *array* empieza en 0 y toma como límite el número que le hemos pasado como tamaño.

En cambio en JSP, existen varias formas de definir e inicializar un *array*, la primera de ellas lo que hace es crear una variable de tipo *array*. Con la sentencia `new` reserva memoria, guarda la referencia en la variable y posteriormente lo inicializa, accediendo al *array* a través de los índices contenidos entre corchetes y no entre paréntesis como lo hace ASP. La segunda forma en cambio declara e inicializa el *array* en una misma línea.

El acceso de los valores de un *array* se consigue gracias a la utilización de los índices. A continuación mostramos como acceder a la posición 0 del *array* definido en el ejemplo anterior.

- **PHP** y **JSP**: `miarray[0]`
- **ASP**: `miarray(0)`

Si queremos acceder al valor de cada una de las posiciones de un *array* necesitamos utilizar un bucle `for` o `foreach` con el que poder iterar sobre los índices del *array* y acceder al valor almacenado en cada posición. Para ello vamos a realizar un ejemplo en cada uno de los lenguajes en el que mostramos como recorrer un *array* e imprimir el valor contenido en cada una de las posiciones:

✓ **PHP**:

```
foreach($miarray as $valor){
  echo $valor;
}
```

✓ **ASP**:

```
for each valor in miarray
  response.write(valor)
next
```

✓ **JSP**:

```
for(i=0;i<miarray.length;i++){
  out.println(miarray[i]);
}
```

Tanto en el ejemplo de ASP como en el de PHP hemos utilizado el bucle `for each`, ya que estos bucles han sido creados para este tipo de tareas, aunque también podríamos haber utilizado el bucle `for` que hemos explicado anteriormente. También podemos apreciar que para imprimir valores por pantalla hemos utilizado la función `out.println` en JSP, `echo` en PHP y `response.write` en ASP.

Por último añadir una serie de funcionalidades que presenta PHP para manejar el manejo de *arrays*.

- Borrar un *array* entero:

```
unset($miarray);
```

- Borrar el elemento contenido en una posición concreta:

```
unset($miarray[0]);
```

- Añadir un nuevo elemento a la posición siguiente del *array*:

```
$miarray[]=8;
```

ACTIVIDADES 3.4

Pruebe este código en PHP para practicar con los *arrays* e intente recrearlo en JSP y ASP:

```
<html>
  <head>
    <title> Actividad 8 para PHP </title>
  </head>
  <body>
    <?php
      $a=array(0=>8,1=>9);

    foreach($a as $valor){
      echo ($valor."<br>");
    }
    ?>
  </body>
 </html>
```

✓ **PHP**:

```
foreach($miarray as $valor){
  echo $valor;
}
```

✓ **ASP**:

```
for each valor in miarray
  response.write(valor)
next
```

✓ **JSP**:

```
for(i=0;i<miarray.length;i++){
  out.println(miarray[i]);
}
```

Tanto en el ejemplo de ASP como en el de PHP hemos utilizado el bucle `for each`, ya que estos bucles han sido creados para este tipo de tareas, aunque también podríamos haber utilizado el bucle `for` que hemos explicado anteriormente. También podemos apreciar que para imprimir valores por pantalla hemos utilizado la función `out.println` en JSP, `echo` en PHP y `response.write` en ASP.

Por último añadir una serie de funcionalidades que presenta PHP para manejar el manejo de *arrays*.

- Borrar un *array* entero:

```
unset($miarray);
```

- Borrar el elemento contenido en una posición concreta:

```
unset($miarray[0]);
```

- Añadir un nuevo elemento a la posición siguiente del *array*:

```
$miarray[]=8;
```

ACTIVIDADES 3.4

Pruebe este código en PHP para practicar con los *arrays* e intente recrearlo en JSP y ASP:

```
<html>
  <head>
    <title> Actividad 8 para PHP </title>
  </head>
  <body>
    <?php
      $a=array(0=>8,1=>9);

    foreach($a as $valor){
      echo ($valor."<br>");
    }
    ?>
  </body>
 </html>
```

3.3.2 ALGORITMOS ASOCIADOS

En los siguientes apartados vamos a explicar los distintos algoritmos de inserción, ordenación y búsquedas utilizados en *arrays*, al igual que las funciones predefinidas por los lenguajes que realizan dichas tareas.

Algoritmos de búsqueda

Los dos algoritmos más importantes para realizar búsquedas dentro de un *array* son: la búsqueda secuencial y la búsqueda binaria.

La búsqueda secuencial es la más sencilla pero a la vez menos eficiente. Por otro lado la búsqueda binaria es algo más compleja pero muy eficiente y se utiliza sobre *arrays* previamente ordenados.

La búsqueda secuencial consiste en ir comparando cada elemento del *array* con el elemento que pretendemos buscar. Este algoritmo termina cuando encontramos el elemento que buscábamos o de lo contrario cuando llegamos al final del *array*. Si se encuentra el elemento se devuelve la posición del *array* y en el caso contrario se devuelve un código de error que indica que no ha sido encontrado dicho valor dentro del *array*.

Por otro lado como podemos ver en la Figura 3.9 el algoritmo de búsqueda binaria consiste en comparar el elemento que divide el *array* en dos mitades, es decir el del centro, con el elemento que buscamos, si no coinciden se determina en que mitad del *array* se encuentra para buscar dentro de ella, descartando la otra mitad. Nuevamente se repite el mismo procedimiento sobre la mitad del *array* elegida y así sucesivamente hasta que encuentre el elemento o por el contrario nos quedemos con una sublista de 1 elemento que no se pueda dividir y dicho número no sea el número que buscamos.

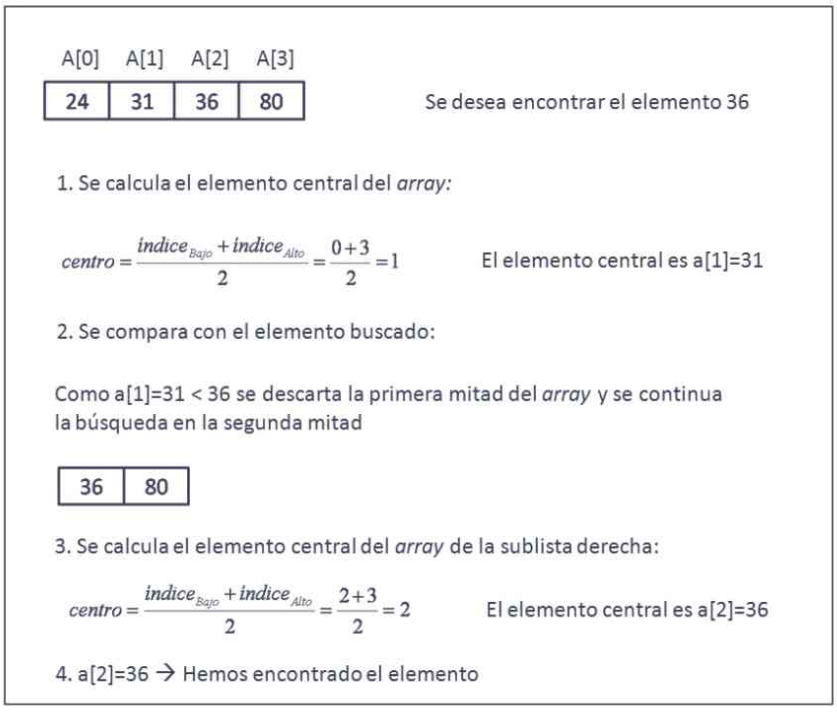


Figura 3.9. *Búsqueda binaria*[1]

[1] Nota: Las imágenes referentes a los algoritmos de ordenación y búsqueda en *arrays* están basadas en Joyanes & Zahonero, 2004.

Afortunadamente hay un gran número de funciones que nos ahorran el tener que programar estos algoritmos. A continuación presentamos las funciones que nos permiten realizar búsquedas dentro de un *array* en los distintos lenguajes:

✓ **PHP**:

```
array_search(variable,valor);

$a=array(0=> 'blue', 1=> 'green');
$b=array_search($a,'green');
```

✓ **JSP**:

```
binarySearch(variable,valor);

String[] a={'blue','green'};
int b=binarySearch(a,'green');
```

Como podemos ver en los ejemplos estas funciones buscan dentro de un *array* un valor y si lo encuentran devuelven el índice para indicar la posición en la que se encuentra dicho valor. En ASP no existe ninguna función que dado un *array* y un valor busque dicho valor dentro del *array*.

Algoritmos de ordenación

Estos algoritmos se encargan de ordenar el contenido de un *array*. Existen distintos tipos de algoritmos de ordenación: los directos (burbuja, inserción directa, selección directa e intercambio) y los avanzados (*QuickSort*, *MergeSort* y *HeapSort*).

- **Inserción Directa**. Este método se basa en la inserción de los elementos de manera ordenada en la posición que les corresponde.

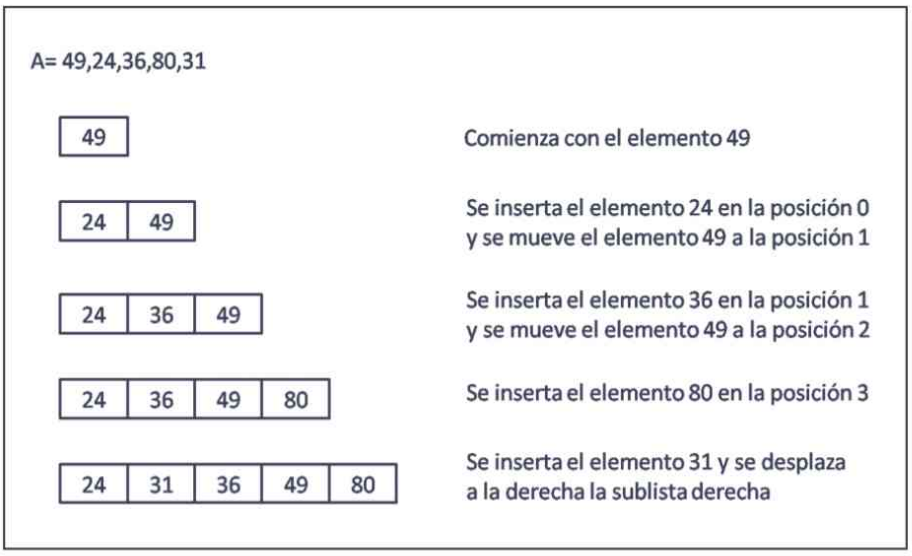


Figura 3.10. *Método de ordenación de inserción directa*

- **Selección Directa**. Este método selecciona el elemento cuyo valor es el más pequeño de todo el *array* y lo intercambia con el primer elemento. A continuación busca el segundo elemento cuyo valor es el más pequeño del *array*, para ello realiza una búsqueda entre el segundo y el último elemento. Una vez encontrado se intercambia con el segundo elemento, y así sucesivamente hasta que quede solo un elemento y, por tanto, la lista esté ordenada.

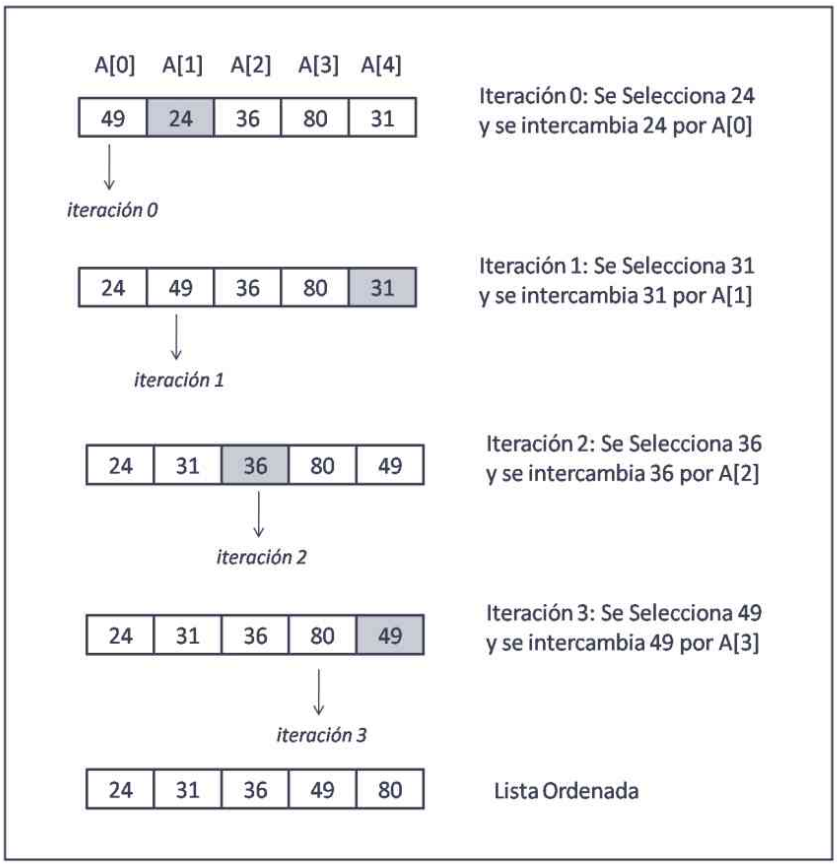


Figura 3.11. *Método de ordenación de selección directa*

- **Intercambio**. Este algoritmo permite ordenar un *array* de manera ascendente. Se basa en la lectura sucesiva del *array* comparando el elemento de la posición 0 de la lista con el resto, si el elemento de la posición 0 del *array* es mayor que el otro elemento con el que se le compara se intercambian. Una vez que hemos comparado el primer elemento con todos, se hace lo mismo con el resto de posiciones hasta conseguir ordenar el *array*.

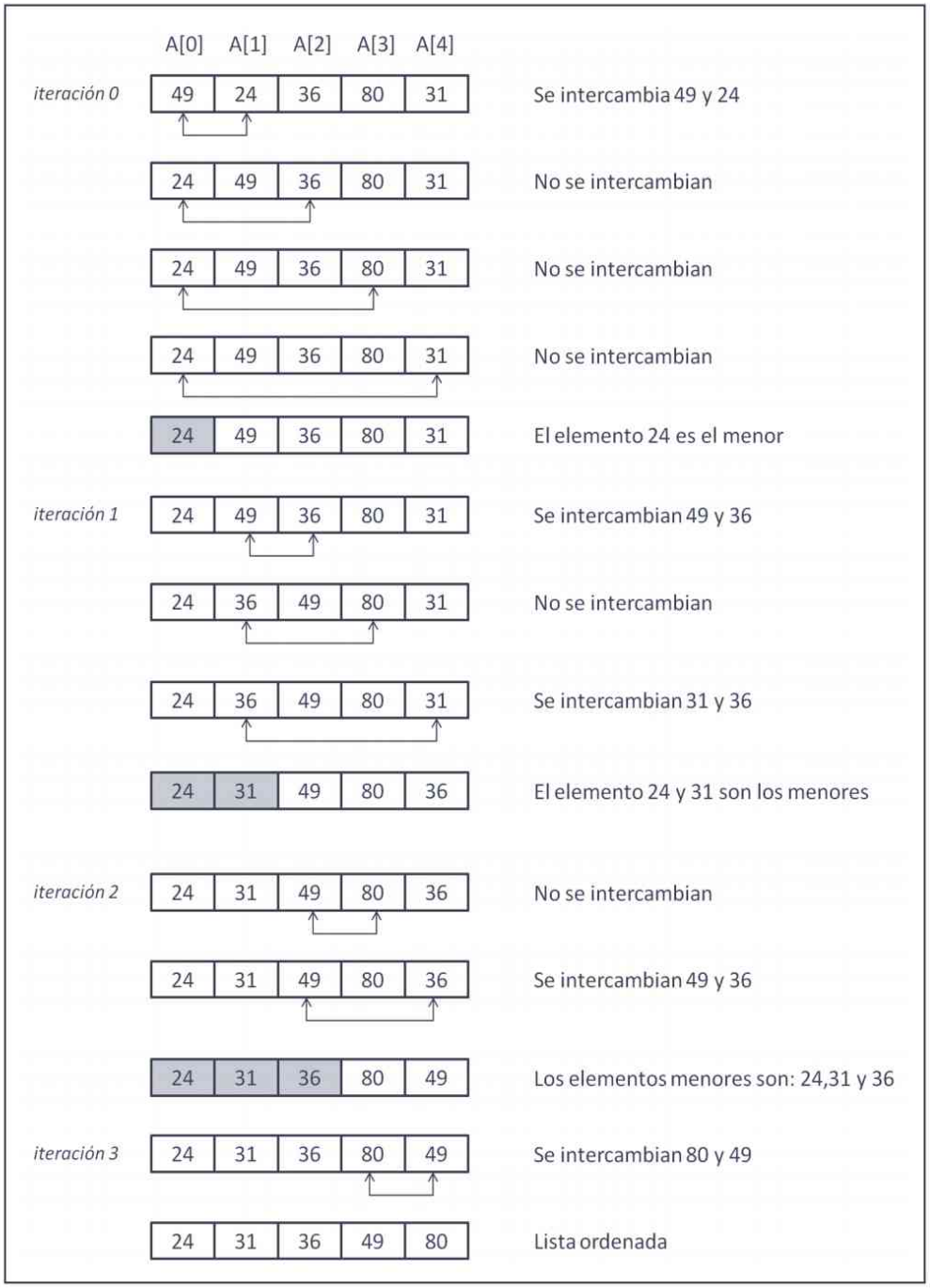


Figura 3.12. *Método ordenación intercambio*

- **Burbuja**. Este método consiste en el intercambio entre pares de elementos adyacentes. Si un elemento no está ordenado respecto al siguiente se intercambian la posición. Esto se realiza en sucesivas iteraciones hasta que el *array* finalmente queda ordenado.

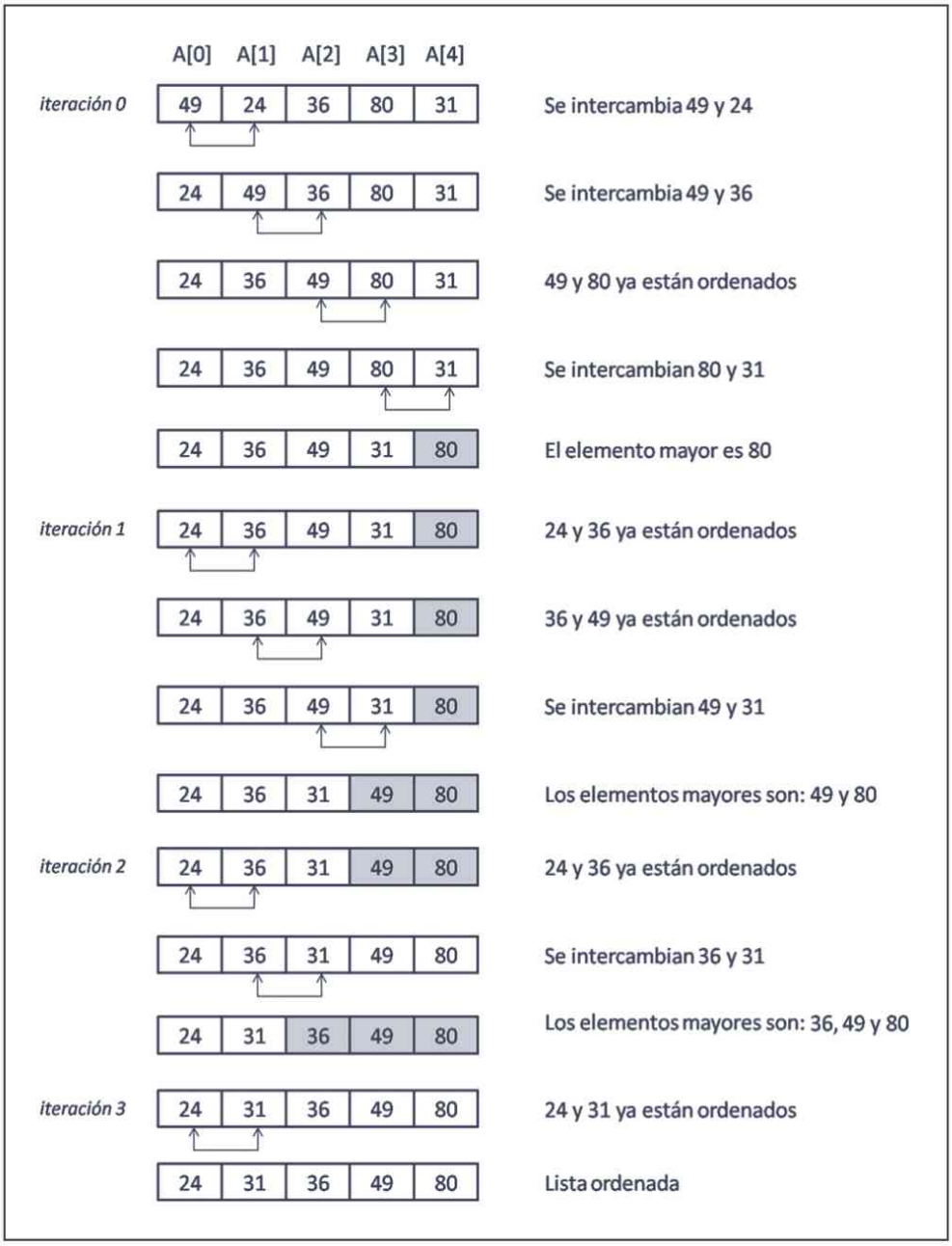


Figura 3.13. *Método de ordenación de la burbuja*

- **QuickSort u ordenación rápida**. Este método consiste en dividir el *array* en dos partes separadas por elemento central, llamado *pivote*. Primero tenemos que elegir el pivote que puede ser un elemento cualquiera, como por ejemplo el primer elemento del *array*. Posteriormente necesitamos dividir el *array* de tal manera que los elementos menores que el pivote formen parte de la sublista izquierda y los que sean mayores se encuentren en la sublista derecha. Una vez que tenemos dividido el *array* en las diferentes sublistas estas se ordenan de forma independiente aplicando este mismo algoritmo.

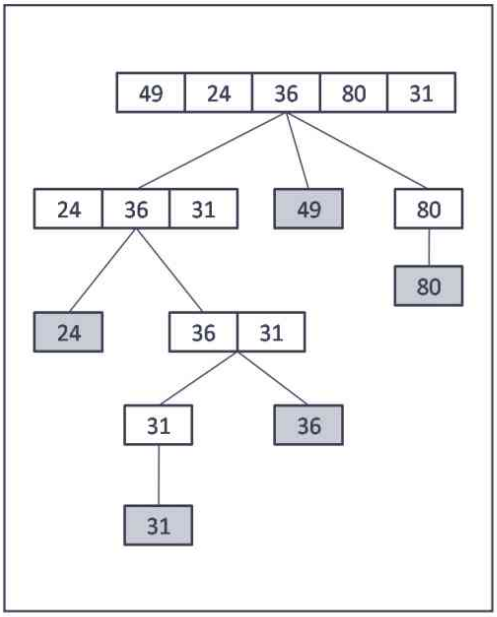


Figura 3.14. *Método de ordenación quicksort*

- **MergeSort**. Este algoritmo parte de dos *array* previamente ordenados y lo que pretendemos al aplicarlo es obtener un *array* ordenado que contenga la información de ambos. Para ello se utiliza un *array* nuevo que tenga como tamaño la suma del tamaño de los dos *arrays*. Después se van comparando los elementos de cada *array* y se inserta en el nuevo *array* el menor de los dos. A medida que se van recorriendo los *array* se lleva el control de la posición por la que se encuentra, haciendo uso de un índice para cada *array* que almacenan la posición actual. Cuando se inserta un elemento en el nuevo *array* se actualiza el índice correspondiente al *array* del que proviene dicho elemento.

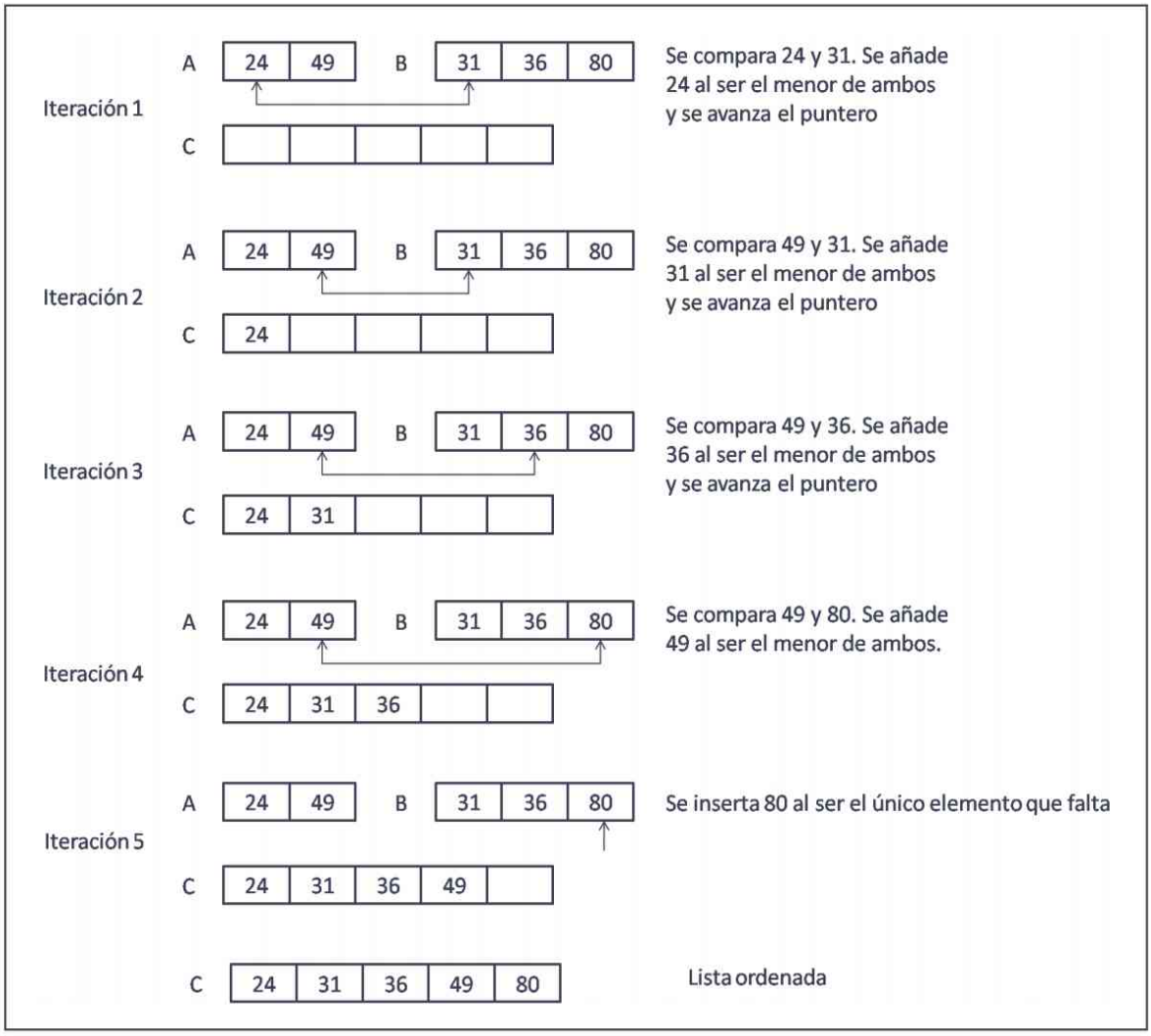


Figura 3.15. *Método de ordenación mergesort*

- **HeapSort**. Es un algoritmo basado en árboles binarios. Para ordenar un *array* de forma ascendente debemos cumplir la condición de que un nodo padre no puede ser menor a un nodo hijo.

Una vez que ya hemos visto los algoritmos más conocidos para ordenar un *array*, hay que tener en cuenta que los algoritmos de ordenación directos son mucho más sencillos de implementar y más ineficientes que los métodos avanzados. Los algoritmos de ordenación preferidos por excelencia suelen ser la selección e inserción directa.

Es bueno conocer el comportamiento de los distintos algoritmos de ordenación, pero hoy en día los lenguajes proporcionan funciones con algoritmos de ordenación ya implementados. A continuación podemos ver las funciones de ordenación que existen en los lenguajes PHP y JSP:

✓ **PHP**:

```
rsort(variable);
sort(variable);

$a=array(0=> 21,1=> 8,2=> 9);
rsort($a);
sort($a);
```

✓ **JSP**:

```
sort(variable);

String[] a={21,8,9};
sort(a);
```

Como podemos observar PHP presenta dos funciones para ordenar un *array*, uno para ordenarlo de menor a mayor (`sort`), que existe también en JSP. Y otro que los ordena de manera inversa (`rsort`). En cambio en el lenguaje de programación ASP no existe ninguna función que permita ordenar un *array*.

Algoritmos de inserción

Este tipo de algoritmos se utilizan para añadir nuevos elementos dentro de un *array*. Si el *array* no está ordenado basta con añadir el elemento al final del *array*, si no habría que recorrer el *array* para buscar la posición exacta en la que debemos insertar el elemento y desplazar los elementos mayores que estén a la derecha.

3.4 PRINCIPIOS DE SUBPROGRAMACIÓN

En este apartado vamos a estudiar cómo podemos definir y usar pequeñas funciones y/o procedimientos definidos por el usuario para incentivar la reutilización de código y la programación modular. También estudiaremos algunas de las funciones predefinidas que presentan los distintos lenguajes que estamos viendo.

3.4.1 DEFINICIÓN Y USO

Un subprograma es un fragmento de código que tiene una funcionalidad específica. Este permite que el código sea modular y lo podamos reutilizar.

El nacimiento de los subprogramas nace de la idea de que un problema difícil resulta más fácil solucionarlo si se divide en otros más pequeños y su vez más sencillos.

Existen dos tipos de subprogramas: las funciones y los procedimientos. Las funciones son aquellos subprogramas que devuelven un valor como resultado de su ejecución. Por otro lado los procedimientos son aquellos que se ejecutan sin devolver ningún tipo de valor.

Una vez que ya sabemos qué son los subprogramas, qué tipos nos podemos encontrar y para qué sirven, vamos a pasar a ver su sintaxis en los distintos lenguajes.

Primero vamos a mostrar cómo se define una función:

- **PHP**:

```
function nombre($arg1,$arg2,…){
  instrucciones;
  return $valorDevuelto;
}
```

- **ASP**:

```
function nombre (arg1,arg2,…)
  instrucciones
  nombre=valordevuelto
end function
```

- **JSP**:

```
tipoDevuelto nombre (tipo1 arg1,tipo2 arg2,…){
  instrucciones;
  return valorDevuelto;
}
```

Para entender mejor cómo se define una función mostramos un ejemplo a continuación:

- **PHP**:

```
function sumar ($sumando1,$sumando2){
  return $sumando1+$sumando2;
}
```

- **ASP**:

```
function sumar (sumando1,sumando2)
  sumar=sumando1+sumando2
end function
```

- **JSP**:

```
double sumar (double sumando1, double sumando2){
  return sumando1+sumando2;
}
```

Entre los distintos lenguajes podemos apreciar ciertas diferencias como por ejemplo que tanto PHP como JSP utilizan la sentencia `return` para devolver el valor resultante, mientras que ASP le asigna dicho valor al nombre de la función. Otra de las diferencias notorias es que solo en JSP se indica el tipo de dato de cada argumento que pasamos como parámetro en la función y se especifica en la cabecera de dicha función el tipo de dato del valor devuelto.

Para finalizar esta sección vamos a ver cómo se definen los procedimientos en los distintos lenguajes como hemos hecho con las funciones.

- **PHP**:

```
function nombre($arg1,$arg2,…){
  instrucciones;
}
```

- **ASP**:

```
sub nombre (arg1,arg2,…)
  instrucciones
end sub
```

- **JSP**:

```
void nombre (tipo1 arg1,tipo2 arg2,…){
  instrucciones;
}
```

Aquí podemos ver un ejemplo de cómo definir un procedimiento:

- **PHP**:

```
function mensaje ($texto){
  echo $texto;
}
```

- **ASP**:

```
sub mensaje (texto)
  response.write(texto)
end sub
```

- **JSP**:

```
void mensaje (String texto){
  out.println(texto);
}
```

Salvo ASP que tiene una sintaxis específica para definir procedimientos PHP y JSP utilizan la misma sintaxis que para definir las funciones salvo que no utilizan la sentencia `return` para devolver un valor. En el caso de JSP en la cabecera de la función se especifica que el tipo del valor devuelto es vacío (`void`).

Las funciones y procedimientos se suelen definir en la cabecera del documento entre las etiquetas `<head></head>` para asegurarnos que estén cargadas en memoria previamente antes de usarlas. Para usar las funciones y procedimientos solamente necesitamos llamarlos por su nombre y pasarle los parámetros necesarios para que se puedan ejecutar. En el caso de las funciones podemos asignar a otra variable el valor que esta devuelve para utilizarlo posteriormente. A continuación mostramos la sintaxis y un ejemplo de la llamada a un procedimiento o función.

- **PHP**:

```
nombre($arg1,$arg2,…);

$texto = "Esto es una prueba.";
mensaje ($texto);
```

- **ASP**:
  - **Forma 1**:

```
nombre (arg1,arg2,…)

texto = "Esto es una prueba."
mensaje (texto)
```

  - **Forma 2** (solo procedimientos):

```
call nombre (arg1,arg2,…)

texto = "Esto es una prueba."
call mensaje (texto)
```

- **JSP**:

```
nombre (arg1,arg2,…);

texto = "Esto es una prueba.";
mensaje (texto);
```

La segunda forma definida en ASP para llamar a un procedimiento haciendo uso de la sentencia `call` no la podemos utilizar para llamar a una función, mientras que la primera la podemos utilizar para ambas cosas.

¿SABÍAS QUE...?

En ASP también podemos provocar la salida de un procedimiento o una función para ello se utilizarán las sentencias: `exit sub` y `exit function`.

ACTIVIDADES 3.5

- Implemente la función que esbozamos como ejemplo en este apartado que suma dos números que se le pasan como parámetro y muestra el resultado de llamarla. Hágalo para los tres lenguajes que hemos visto.
- Implemente el procedimiento utilizado como ejemplo en este apartado que imprime el mensaje que se le pasa como parámetro y llámelo para que se ejecute. Hágalo para los tres lenguajes que hemos visto.

3.4.2 FUNCIONES PREDEFINIDAS DEL LENGUAJE

Las funciones predefinidas son aquellas que están integradas dentro de un lenguaje por defecto.

A continuación mostramos dos tablas con las funciones predefinidas en el leguaje ASP para *strings* y *arrays* que son unas de las funciones que más se utilizan. Existen muchas funciones más que podemos consultar en distintos manuales de ASP.

Tabla 3.1 Funciones para *strings* en ASP

Método	Descripción	Sintaxis

`InStr()`	Devuelve la posición de la primera aparición de la `cadena2` en la `cadena1`. La búsqueda comienza en el primer carácter. El cuarto parámetro indica el tipo de comparación (0-binaria, 1-textual), pero es opcional.	`InStr([posición_comienzo,] cadena1,cadena2[,tipo_ comparación])`
		
		`InStr(1,"ala","a")➔1`

`InStrRev()`	Devuelve la posición de la primera aparición de la `cadena2` en la `cadena1`. La búsqueda se realiza en sentido inverso, empezando por el último carácter. El cuarto parámetro indica el tipo de comparación (0-binaria, 1-textual), pero es opcional.	`InStrRev(cadena1,cadena2[,posici ón_comienzo][,tipo_comparación])`
		
		`InStr("ala","a")➔3`

`LCase()`	Convierte una cadena en minúsculas.	`LCase(cadena)` `LCase("HOLA")➔"hola"`
`Left()`	Devuelve un número especificado de caracteres desde el lado izquierdo de una cadena.	`Left(cadena, n°caracteres)` `Left("jueves",3)➔"jue"`
`Len()`	Devuelve el tamaño de la cadena.	`Len(cadena)` `Len("jueves")➔6`
`LTrim()`	Elimina los espacios en el lado izquierdo de una cadena.	`LTrim(cadena)` `LTrim(" hola ")➔"hola "`
`RTrim()`	Elimina los espacios en el lado derecho de una cadena.	`RTrim(cadena)` `RTrim(" hola ")➔" hola"`
`Trim()`	Elimina los espacios a la izquierda y a la derecha de una cadena.	`Trim(cadena)` `Trim(" hola ")➔"hola"`
`Mid()`	Devuelve los caracteres que se encuentran en la posición que hemos especificado de la cadena.	`Mid(cadena,índice_ comienzo[,tamaño])` `Mid("jueves",2,3)➔"uev"`
`Replace()`	Sustituye a una parte específica de una cadena con otra cadena un número determinado de veces.	`Replace(cadena,cadena_ antigua,cadena_nueva [,índice_ comienzo][, n°sustituciones][, tipo_comparación])` `Replace("Es martes","martes","jueves")➔"Es jueves"`
`Rigth()`	Devuelve un número específico de caracteres desde el lado derecho de una cadena.	`Right(cadena, n°caracteres)` `Right("jueves",2)➔"es"`

`Space()`	Devuelve una cadena que consta de un número determinado de espacios.	`Space(n°espacios)`
		`Dim cad`
		`cad=Space(2)➔cad= "  "`

`StrComp()`	Compara dos cadenas y devuelve un valor que representa el resultado de la comparación. El tipo de comparación pude ser: 0-binaria y 1-textual. Por defecto realiza la comparación binaria. Devuelve 0 si son iguales y -1 si no lo son.	`StrComp(cadena1, cadena2, tipo_comparación)` `StrComp("hola","HOLa")➔-1`
`String()`	Devuelve una cadena que contiene un carácter repetido tantas veces como la longitud especificada.	`String(n°repeticiones, carácter)` `String(2,"#")➔"##"`
StrReverse()	Invierte una cadena.	`StrReverse(cadena)` `StrReverse("hola")➔"aloh"`
`Ucase()`	Convierte una cadena a mayúsculas.	`Ucase(cadena)` `Ucase("hola")➔"HOLA"`

Tabla 3.2 Funciones para *arrays* en ASP

Método	Descripción	Sintaxis
`Array()`	Devuelve un *array*.	`Array(argumentos)` `Array(2,8,19,35)`

`Filter()`	Devuelve un *array* que contiene un subconjunto del *array* pasado como argumento que satisfacen los criterios de filtración.	`Filter(nombre_array, filtro[, booleano] [,tipo de comparación])`
		`a=Array("Lunes", "Martes", "Miercoles")`
		`b=Filter(a,"M")➔ b=("Martes", "Miercoles")`

`IsArray()`	Devuelve un booleano que indica si una variable es un *array*.	`IsArray(nombre_array)`
		`a=Array("Lunes", "Martes", "Miercoles")`
		`IsArray(a)➔True`

`Join()`	Devuelve una cadena con los elementos que contiene un *array*. Puede emplear delimitadores de forma opcional para separar los elementos.	`Join(nombre_array [,delimitador])`
		`a=Array("Lunes", "Martes)`
		`Join(a,"#")➔ "Lunes # Martes"`
		`Join(a)➔ "Lunes Martes"`

`LBound()`	Devuelve el índice más pequeño de un *array*. Se puede indicar la dimensión a la que nos referimos en el caso de matrices.	`LBound(nombre_array,dimension)`
		`a=Array("Lunes", "Martes)`
		`LBound(a)➔0`

`Split()`	Parte una cadena utilizando el delimitador y lo devuelve en forma de *array*. El delimitador por defecto es el espacio en blanco.	`Split(expresión,[delimitador] [,n°cadenas][,tipo_comparación])` `a=Split("hola mundo") ➔ a=("hola","mundo")`

`UBound()`	Devuelve el índice más grade de un *array*. Se puede indicar la dimensión a la que nos referimos en caso de matrices.	`UBound(nombre_array,dimension)`
		`a=Array("Lunes", "Martes)`
		`UBound(a)➔1`

También vamos a listar las funciones más importantes y utilizadas para *strings* y *arrays* para PHP:

Tabla 3.3 Funciones para *strings* en PHP

Método	Descripción	Sintaxis
`echo()`	Imprime una cadena.	`echo(string)` `echo "hola"; ➜ "hola"`

`explode()`	Rompe un *string* en trozos, utilizando el delimitador y lo guarda en un *array*.	`explode(delimitador,string[,limite])`
		`explode(",","Hola mundo");`
		`➜Array([0]=>Hola, [1]=> mundo)`

`implode()`	Convierte un *array* en un *string*.	`implode(delimitador,nombre_array)`
		`$a = array('lunes', 'martes');`
		`$b=implode(" ",$a); ➜ b= " lunes martes"`

`ltrim()`	Elimina los espacios u otros caracteres predefinidos en el lado izquierdo de una cadena.	`ltrim(string[,caracteres predefinidos])` `ltrim(" hola "); ➜ "hola "`
`rtrim()`	Elimina los espacios u otros caracteres predefinidos en el lado derecho de una cadena.	`rtrim(string[,caracteres predefinidos])` `rtrim(" hola "); ➜ " hola"`
`str_repeat()`	Repite un *string* un número específico de veces.	`str_repeat(string, nºrepeticiones)` `str_repeat("#",2);➜ "##"`
`str_replace()`	Reemplaza una parte de un *string* por otra cadena.	`str_replace(cadena_vieja,cadena_nueva,string,[count])` `str_replace("hola","adios", "hola a todos");➜ "adiós a todos"`
`strcmp()`	Compara dos *string*. Devuelve 0 si son iguales y distinto de 0 si son distintos.	`strcmp(string1,string2)` `strcmp("hola", "hola"); ➜0`
`strlen()`	Devuelve la longitud de un *string*.	`strlen(string)` `strlen("hola");➜4`
`strrev()`	Devuelve un *string* invertido.	`strrev(string)` `strrev("hola");➜"aloh"`

`strstr()`	Busca la primera ocurrencia de un *string* en otro.	`strstr(string1,string2)`
		`strstr("hola a todos", "todos");`
		`➜ "todos"`

`strtolower()`	Convierte a minúsculas un *string*.	`strtolower(string)` `strtolower("HOLA");➜ "hola"`
`strtoupper()`	Convierte a mayúsculas un *string*.	`strtoupper(string)` `strtoupper("hola");➜ "HOLA"`
`trim()`	Elimina los espacios en blanco y otros caracteres predefinidos a un lado y a otro del *string*.	`trim(string[,caracteres predefinidos])` `trim(" hola "); ➜"hola"`

Tabla 3.4 Funciones para *arrays* en PHP

Método	Descripción	Sintaxis
`array()`	Crea un *array*.	`array(clave => valor)` `$a=array("a"=>"lunes","b"=>"martes");`

`array_merge()`	Combina uno o más *arrays* en otro *array*.	`array_merge(array1[,array2]…)`
		`$a1=array("a"=>"lunes","b"=>"martes");`
		`$a2=array("c"=>"miercoles");`
		`array_merge($a1,$a2);➔array("a"=>"lunes","b"=>"martes","c"=>"miercoles")`

`array_pop()`	Elimina el último elemento del *array*.	`array_pop(nombre_array)`
		`$a=array("a","b","c");`
		`array_pop($a);➔ array("a","b")`

`array_push()`	Añade uno o más elementos a un *array* por el final.	`array_push(nombre_array, valor1, valor2…)`
		`$a=array("a","b","c");`
		`array_push($a, "d");`
		`➔array("a","b","c","d")`

`array_values()`	Devuelve el valor de todos los elementos.	`array_values(nombre_array)`
		`$a1=array("a"=>"lunes","b"=>"martes");`
		`array_values($a1);➔ array([0]=>"lunes", [1]=>"martes")`

`count()`	Devuelve el nº de elementos que hay en un *array*.	`count(nombre_array)`
		`$a=array("a","b","c");`
		`count($a);➔3`

`in_array()`	Informa de si un elemento está dentro del *array*. Devuelve un valor booleano	`in_array(valor,nombre_array[,tipo_búsqueda])`
		`$a=array("a","b","c");`
		`in_array("a",$a); ➔ true`

Por último vamos a mostrar unas tablas con algunas funciones de JSP para *strings* y *arrays*, ya que hay muchísimas que podemos encontrar en la API de Java.

Tabla 3.5 Funciones para *strings* en JSP

Método	Descripción	Sintaxis

`compareTo()`	Compara dos *strings*. Devuelve 0 si son iguales y distinto de 0 si no lo son.	`Cadena1.compareTo(cadena2)`
		`String c1 = "hola";`
		`c1.compareTo("hola");➔ 0`

`concat()`	Concatena dos *strings*.	`Cadena1.concat(cadena2)`
		`String c1 = "hola";`
		`c1.concat(" maria"); ➔ c1="hola maria"`

`contains()`	Devuelve un booleano que indica si un *string* está contenido dentro de otro *string*.	`Cadena1.contains(cadena2)`
		`String c1 = "hola";`
		`c1.contains("ol"); ➔ true`

`copyValueOf()`	Convierte un *array* en un string.	`String.copyValueOf(nombre_array)`
		`Char[] a=new char []`
		`{'h','o','l','a'};`
		`String c2=String.`
		`copyValueOf(a);➔ c2="hola"`

`endsWith()`	Evalúa si un *string* termina con el *string* pasado como argumento. Devuelve un valor booleano.	`Cadena1.endsWith(cadena2)`
		`String c1 = "hola mundo";`
		`c1.endsWith("mundo"); ➔true`

`length()`	Devuelve la longitud de un *string*.	`Cadena.length()`
		`String c1="hola";`
		`c1.length();➔4`

`replaceAll()`	Reemplaza en un *string*, una parte del *string* por otro.	`Cadena1.replaceAll(cadena_`
		`vieja,cadena_nueva)`
		`String c1="hola a todos";`
		`c1. replaceAll("todos",`
		`"Maria");➔ c1="hola a Maria"`

`startsWith()`	Comprueba si un *string* comienza por el *string* pasado como argumento. Devuelve un valor booleano.	`Cadena1.startsWith(cadena2)`
		`String c1 = "hola a todos";`
		`c1. startsWith("hola"); ➔ true`

`substring()`	Devuelve una subcadena del *string*.	`Cadena.substring(indice_`
		`inicio,indice_fin)`
		`String c1 = "hola";`
		`c1.substring(1,3);➔`
		`"ol"`

`toCharArray()`	Convierte un *string* en un *array* de caracteres.	`Cadena.toCharArray()`
		
		`String c1 = "hola";`
		`char[] c2 = c1.toCharArray();➔`
		`c2= {'h','o','l','a'}`

`toLowerCase()`	Convierte en minúsculas un *string*.	`Cadena.toLowerCase()`
		`String c1 = "HOLA";`
		`c1. toUpperCase(); ➔ c1="hola"`

`toUpperCase()`	Convierte en mayúsculas un *string*.	`Cadena.toUpperCase()`
		`String c1 = "hola";`
		`c1. toUpperCase();➔ c1="HOLA"`

`trim()`	Devuelve una copia del *string*, donde se eliminan los espacios en blanco del inicio y final de la cadena.	`Cadena.trim()`
		
		`String c1 = " hola ";`
		`c1.trim();➔ c1="hola"`

Tabla 3.6 Funciones para *arrays* en JSP

Método	Descripción	Sintaxis

`binarySearch()`	Realiza una búsqueda binaria dentro de un *array*. Devuelve el índice dentro del *array* en el que se encuentra el elemento.	`Arrays.binarySearch(nombre_array,elemento)`
		`int[]a={8,3,2};`
		`Arrays.binarySearch(a,8);➜0`

`equals()`	Informa de si dos *arrays* son iguales. Devuelve un valor booleano.	`Arrays.equals(array1, array2)`
		`int[]a={8,3,2};`
		`int[]b={8,3,2};`
		`Arrays.equals(a,b); ➜ true`

`fill()`	Asigna un valor a cada posición del *array*.	`Arrays.fill(nombre_array,valor)`
		`int a[] = new int[3];`
		`Arrays.fill(a,-1); ➜`
		`a={-1,-1,-1}`

`length()`	Devuelve la longitud de un *array*.	`nombre_array.length()`
		`int[]a={1,2,3};`
		`a.length();➜3`

`sort()`	Ordena un *array* de menor a mayor.	`Arrays.sort(nombre_array)`
		`int[]a={8,3,2};`
		`Arrays.sort(a); ➜ a={2,3,8}`

`toString()`	Convierte un *array* a *string*.	`Arrays.toString(nombre_array)`
		`int[]a={1,2,3};`
		`Arrays.toString(a); ➜ "123"`

ACTIVIDADES 3.6

Consulte otras funciones predefinidas del lenguaje en las siguientes URL:

a. PHP: *http://php.net/manual/es/index.php*

b. ASP: *http://www.w3schools.com/vbscript/vbscript_ref_functions.asp*

c. JSP: *http://docs.oracle.com/javase/7/docs/api/*

3.5 ACCESO A LA INFORMACIÓN DEL CLIENTE WEB

En este apartado vamos a aprender los distintos tipos de paso de parámetros y sus diferencias, cómo definir un formulario y como recuperar los datos enviados por el método *POST* y *GET*.

3.5.1 MÉTODOS GET Y POST

Tanto el método *GET* como el método *POST* no son más que métodos del protocolo HTTP para el intercambio de información entre el cliente y el servidor.

GET es el método más usado actualmente. Este método le "pide" al servidor web que le devuelva al cliente la información identificada en la petición URI. Lo más común es que las peticiones URI se refieran a un documento HTML o a una imagen, aunque también se puede referir a una consulta de una base de datos. En tal caso, el servidor procesa la petición y le devuelve al cliente el resultado generado. Ejemplos típicos son las páginas web personalizadas que son actualizadas dinámicamente.

Mientras que el método *GET* lo utilizamos para recuperar información, el método *POST* se usa para enviar información a un servidor web. Estos casos de "posting" pueden ser utilizados para completar un formulario de autenticación, así como entradas de datos o especificar parámetros para algún tipo de software del servidor. La función realizada por el método *POST* depende de la petición URI.

La principal diferencia entre método *GET* y el método *POST* radica en cómo codifican la información. El método *GET* envía las variables dentro de la URL de la página.

Para entender cómo se envían las variables en el método *GET* es necesario que entendamos las partes de una URL. La URL está formada por:

- **Protocolo**. Especifica el protocolo de comunicación que se utiliza para el intercambio de la información.
- **Nombre de dominio**. Nombre del servido donde se aloja la información.
- **Directorios**. Secuencia de directorios separados por "/" que indica la ruta en la que se encuentra el recurso.
- **Fichero**. Nombre del recurso o fichero al que queremos acceder.

En la Figura 3.16 podemos ver las partes que forman una URL:

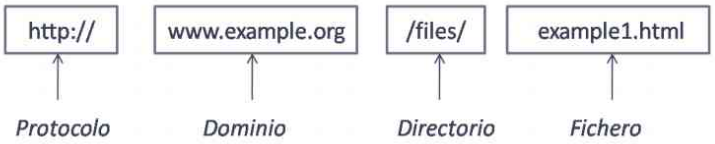


Figura 3.16. *Formato URL*

Una vez que sabemos cómo es la estructura de una URL es necesario conocer que al final de esta podemos concatenar el símbolo de cierre de interrogación "?". Este símbolo se utiliza para indicar que se van a definir a continuación variables con el valor que toman (variable=valor). Las distintas variables se separan entre ellas con el símbolo "&". A continuación vemos un ejemplo de cómo viajan las variables en las peticiones realizadas con el método *GET*:

```
http://www.example.org/file/example1.php?v1=0&v2=3
```

Por otro lado, en el método *POST* la información va codificada en el cuerpo de la petición HTTP y, por tanto, viaja oculta. Por este motivo hay que utilizar este método cuando queremos ingresar datos en un formulario, ya sea para realizar la autenticación de un usuario o la inserción y/o actualización en una base de datos. Si utilizáramos el método *GET* para realizar este tipo de acciones cualquier persona con mínimos conocimientos de informática podría manipular la URL, o pasarle a otra persona la URL para que acceda a una página para la que no debería tener permiso. Esto no quiere decir que el método *GET* sea más inseguro que el método *POST*, ya que este puede ser vulnerado con ataques de tipo *HTML injection* que es un ataque que permite ejecutar código *scripting* debido a las vulnerabilidades del sistema de validación del método *POST*. Estos ataques afectan al servidor que contiene la información.

Como conclusión decir que no hay un método más seguro que otro. Como podemos ver ambos tienen sus pros y sus contras, pero resulta recomendable que cada uno de ellos los utilicemos para la labor para la que fueron creados que son recuperación (*GET*) y el envío (*POST*) de información.

Las llamadas a través del método *GET* pueden ser cacheadas, indexadas por los buscadores e incluso se pueden almacenar en el navegador como favoritos. Esto a veces resulta útil, como por ejemplo cuando rellenamos un formulario para realizar algún tipo de búsqueda y nos devuelve la página con la información permitiéndonos volver atrás utilizando el botón del navegador, mientras que el método *POST* esto no lo permite.

3.5.2 DEFINICIÓN DE FORMULARIOS

Los formularios son métodos para recolectar información que debe ser rellenada por el usuario. Estos formularios pueden servir para permitir la autenticación de un usuario en una página web, recoger información que debemos insertar o actualizar en una base de datos, realizar búsquedas de información en el que debamos especificar un criterio de búsqueda, etc.

Para entenderlo bien vamos a ver dos ejemplos de formulario, uno en el que utilizamos el método *POST* y otro en el que utilizamos el método *GET* para el envío de información y sobre ellos explicaremos que elementos componen los formularios.

Ejemplo de formulario utilizando el método GET:

```
<html>
  <head>
    <title>Ejemplo Formularios</title>
  </head>
  <body>
    <h1>Ejemplo de procesado de formularios</h1>
    <FORM ACTION="ejemplo1.php" METHOD="GET">
      Introduzca su nombre:<INPUT TYPE="text"
        NAME="nombre"><BR>
      Introduzca sus apellidos:<INPUT TYPE="text"
        NAME="apellidos"><BR>
      <INPUT TYPE="submit" VALUE="Enviar">
    </FORM>
  </body>
</html>
```

Aquí podemos ver exactamente el mismo ejemplo, solo que en este formulario se utiliza el método POST:

```
<html>
  <head>
    <title>Ejemplo Formularios</title>
  </head>
  <body>
    <h1>Ejemplo de procesado de formularios</h1>
    <FORM ACTION="ejemplo2.php" METHOD="POST">
      Introduzca su nombre:<INPUT TYPE="text"
        NAME="nombre"><BR>
      Introduzca sus apellidos:<INPUT TYPE="text"
        NAME="apellidos"><BR>
      <INPUT TYPE="submit" VALUE="Enviar">
    </FORM>
  </body>
</html>
```

Como podemos ver en ambos ejemplos los formularios los definimos utilizando las etiquetas HTML y como todo documento HTML está dividido en dos partes: la cabecera (`<head></head>`) donde se establece el título de la página (`<title></title>`) y el cuerpo de la página (`<body></body>`) donde se define el contenido. Es en el cuerpo donde se añadirán los elementos que pueden forman parte del formulario dentro de las etiquetas `<form> </form>`, como los campos de texto donde introducir información (`<input type=text>`), los botones para enviar el formulario (`<input type=submit>`), y otro tipos de elementos que podemos añadir como los *combo box* que permiten elegir de una lista desplegable un elemento, *radio buttons*, *checkbox*, etc.

Una vez relleno el formulario la información introducida por el usuario se almacena en variables con el nombre que se le haya asignado a cada elemento con el atributo `name`. Al pulsar el botón se ejecuta la acción definida por el atributo `action` de la etiqueta `<form>`, en el caso del ejemplo se redirige a otra página donde se procesará y se hará uso de la información. Es también en la etiqueta `<form>` con el atributo `method` donde definimos como se va a enviar la información: a través de la URL (método *GET*) o en el cuerpo del mensaje (método *POST*).

Una vez que sabemos cómo realizar un formulario y que elementos lo forman vamos a ver en la Figura 3.17 como el navegador interpreta ese código y lo muestra al usuario.

Figura 3.17. *Ejemplo formulario*

ACTIVIDADES 3.7

- Copie los formularios de los ejemplos facilitados en este apartado, tanto el que utiliza el método *POST* como el que utiliza el método *GET*. Añádale un campo edad y observe la URL para poder apreciar cómo se produce el envío de los parámetros.

3.5.3 RECUPERACIÓN DE INFORMACIÓN CON GET

Una vez que hemos enviado el formulario al servidor, este tiene que recuperar la información para poder procesarla. En PHP quizás resulte un poco más fácil que en el resto de lenguajes que estamos viendo, porque en el caso del envío de información utilizando el método *GET* existe una variable especial `$_GET`, donde se almacenan todas las variables pasadas con este método. La forma en que lo almacena es sencilla, pues esta variable no es más que un *array* en el que el índice es el nombre asignado al elemento del formulario con el atributo `name` y dentro almacena el valor introducido por el usuario en el formulario.

En el ejemplo del formulario anterior si queremos acceder al nombre y apellidos que introduce el usuario solo es necesario realizar lo siguiente:

- **Forma 1**:

```
echo $_GET['nombre'];
echo $_GET['apellidos'];
```

- **Forma 2**:

```
print_r($_GET);
```

Como podemos ver la función `echo` muestra un elemento y `print_r` muestra el *array* completo.

Por otro lado en ASP para obtener la información enviada por el método *GET* es necesario utilizar `request.QueryString()`, pasándole como argumento el nombre del elemento del formulario del que queremos recuperar el valor. Utilizando el ejemplo del formulario anterior vamos ver cómo se mostraría el contenido de los elementos `nombre` y `apellidos`:

```
response.write(request.QueryString("nombre"))
response.write(request.QueryString("apellidos"))
```

Por último, vamos a explicar cómo se accede a la información enviada, ya sea por el método *GET* o *POST* para JSP:

```
out.print(request.getParameter("nombre"));
out.print(request.getParameter("apellidos"));
```

Como podemos ver solo es necesario utilizar el objeto `request` y llamar al método `getParamenter()` pasándole como argumento el nombre del elemento cuyo valor queremos que nos devuelva.

3.5.4 RECUPERACIÓN DE INFORMACIÓN CON POST

Al igual que en el envío de formularios utilizando el método *GET*, para recuperar la información enviada al servidor utilizando el método *POST* en PHP se utiliza la variable `$_POST`. Esta variable al igual que `$_GET` es un *array* y almacena la información de la misma forma, utilizando como índice el nombre del elemento del formulario y almacenando en su interior el valor que toma. Como hemos visto en apartados anteriores lo que cambia entre el método *GET* y *POST* es como se envía la información o más bien en que parte del mensaje HTTP que le envía el cliente a el servidor viaja.

Utilizando el ejemplo del formulario vamos a ver cómo se recuperaría la información enviada por el método *POST*:

```
echo $_POST['nombre'];
echo $_POST['apellidos'];

print_r($_POST);
```

Por otro lado cuando queremos recuperar la información enviada por el método *POST* en ASP se utiliza `request.Form()`, que se le pasa como argumento el nombre del elemento del formulario. A continuación vamos a mostrar cómo se recuperaría el nombre y apellidos del formulario definido en apartados anteriores si se envía la información utilizando el método *POST*:

```
response.write(request.Form("nombre"))
response.write(request.Form("apellidos"))
```

En JSP la forma de almacenar la información para enviarla es exactamente igual independientemente de con que método lo enviemos, por tanto, se accede a la información de la misma forma. Que se almacene la información de la misma forma tanto para un método como para otro no significa que se envíen de la misma manera, ya que como vimos en apartados anteriores la diferencia radica en que en el método *GET* la información viaja en la URL, mientras que utilizando el método *POST* viaja en el cuerpo del mensaje HTTP que el cliente envía al servidor.

En la sección 3.5.3 mostramos ejemplos de cómo acceder a la información enviada, ya sea utilizando el método *GET* o *POST* para JSP.

¿SABÍAS QUE...?

En PHP existe la variable `$_REQUEST` que contiene tanto el contenido de `$_GET` como el de `$_POST`. Esta variable también se puede utilizar para recuperar la información enviada utilizando ambos métodos.

RESUMEN DEL CAPÍTULO

En este capítulo se enseña a utilizar los distintos bucles y sentencias condicionales que presentan los lenguajes ASP, PHP y JSP. También se presenta cómo declarar, inicializar y recorrer un *array* para acceder a cada uno de los valores que almacena.

Por otro lado, se presentan de forma teórica los algoritmos de búsqueda, inserción y ordenación de *arrays* más utilizados y las funciones predefinidas de los lenguajes que realizan dichos algoritmos.

También se muestra cómo definir funciones y procedimientos y se nombran algunas de las que presentan los lenguajes de forma predefinida.

Para finalizar, se explica cómo crear un formulario y las diferencias entre los distintos métodos para el envío de información. También se estudia cómo recuperar la información en cada uno de los lenguajes dependiendo del método utilizado.

EJERCICIOS PROPUESTOS

- **1.** Utilice el bucle que crea oportuno para sumar los números naturales pares del 10 al 100 y muestre el resultado de la suma. Hágalo para los tres lenguajes vistos en este capítulo.
- **2.** Cree una función en los tres lenguajes que devuelva el producto de dos enteros pasados por parámetro y, en el cuerpo de la página HTML, llame a la función e imprima el valor devuelto.
- **3.** Cree un procedimiento en los tres lenguajes que imprima por pantalla "Hola mundo" y que en el cuerpo de la página HTML se llame a dicho procedimiento.
- **4.** Cree un formulario HTML que consista en pedir dos operandos y un *check box* (`<select>` en HTML) que permita elegir el operador (+,-,/,*) y utilice el método *POST* para el envío de información.
- **5.** Cree un formulario HTML que consista en pedir dos operandos y un *check box* (`<select>` en HTML) que permita elegir el operador (+,-,/,*) y utilice el método *GET* para el envío de información.
- **6.** Implemente el código en los distintos lenguajes que recupere la información del formulario enviado por el método *POST* del ejercicio propuesto 4. Tenga en cuenta que para transformar un *string* en un real en JSP hay que utilizar la función `Double.parseDouble(string)` y la función `Integer.ParseInt(string)` para transformar un *string* en un entero. En ASP la función que transforma un *string* en un real es `Cdbl(string)`.
- **7.** Implemente el código en los distintos lenguajes que recupere la información del formulario enviado por el método *GET* del ejercicio propuesto 5. Tenga en cuenta que para transformar un *string* en un real en JSP hay que utilizar la función `Double.parseDouble(string)` y la función `Integer.ParseInt(string)` para transformar un *string* en un entero. En ASP la función que transforma un *string* en un real es `Cdbl(string)`.

TEST DE CONOCIMIENTOS

1 En la petición "`example.php?nombre=Roberto`", ¿cómo se recupera el valor de la variable "`nombre`" de la página PHP "`example.php`"?
- a) `request.getParameter("nombre").`
- b) `$_POST['nombre'].`
- c) `$_GET['nombre'].`
- d) Ninguna de las anteriores.

2 En las sentencias `switch` o `select case` para ASP, ¿qué lenguaje no permite que la expresión que se evalúa sea un `string`?
- a) ASP.
- b) JSP.
- c) PHP.
- d) Todas las afirmaciones anteriores son falsas.

3 El método *GET* envía los datos del formulario:
- a) En el cuerpo de la petición HTTP.
- b) En la URL de la petición.
- c) Tanto en el cuerpo de la petición como en la URL.
- d) Ninguna de las anteriores.

4 La diferencia entre un procedimiento y una función es:
- a) Una función devuelve un valor mientras que el procedimiento no.
- b) Un procedimiento no puede recibir argumentos mientras que en una función es obligatorio.
- c) Un procedimiento devuelve un valor mientras que una función no.
- d) Ninguna de las anteriores.

5 ¿Qué estructura de control por excelencia se utiliza para recorrer un *array*?
- a) `While` o su equivalente en ASP `Do…While`.
- b) `If`.
- c) `For` o su equivalente en ASP `For…Next`.
- d) `Switch` o su equivalente en ASP `Select Case`.

6 ¿Cuántas veces se ejecutará el siguiente bucle escrito en JSP `while(i>0){i=i-1;}`, si `i=1` antes del bucle?
- a) 1.
- b) Ninguna.
- c) 2.
- d) Infinitas.

7 ¿Cuál de estos algoritmos no es un algoritmo de ordenación?
- a) Intercambio.
- b) Selección Directa.
- c) Burbuja.
- d) Binaria.

8 ¿Qué algoritmo de búsqueda es más eficiente?
- a) Búsqueda secuencial.
- b) Búsqueda binaria.
- c) Son igual de eficientes.
- d) Ninguna de las anteriores.

4 Generación dinámica de páginas web

OBJETIVOS DEL CAPÍTULO

- ✓ Entender cómo construir una aplicación web dinámica a través de mecanismos de separación entre la lógica de negocio y la presentación de la página.
- ✓ Analizar y distinguir qué significa una arquitectura en capas y una arquitectura en niveles.
- ✓ Conocer qué son los patrones de software y cómo estos pueden facilitar la construcción de un sitio web.
- ✓ Estudiar las herramientas que se pueden utilizar para la generación dinámica de interfaces web, tanto en el entorno cliente como en el entorno servidor.

La generación dinámica de páginas es uno de los grandes avances dentro del desarrollo de aplicaciones en entorno web. La gran ventaja de esta característica es permitir la independencia entre el diseño de páginas web y la lógica del negocio. La creación de páginas web ha evolucionado de manera paralela al desarrollo de la Web. Las primeras páginas web, escritas íntegramente en código HTML, coinciden con la denominada *Web 1.0* la cual está dominada por los contenidos *estáticos*. En esta primera etapa se utiliza el lenguaje de marcado HTML el cual mezcla el diseño y contenido a una página web a través de un conjunto de etiquetas. Las etiquetas permiten marcar tres tipos de elementos dentro de una página web:

- Características de formato de un elemento de la página (p. ej. `<strong>` para negrita en textos).
- Elementos estructurales de la página web (p. ej. `<head>` o `<body>`).
- Identificar elementos de hipervínculo. (p. ej. `<a href>`).

Esta primera etapa de la Web consolida una nueva labor dentro de la informática: los denominados *webmasters*. Los "maestros de la Web" son los encargados de la creación y mantenimiento de las nuevas Webs en los servidores web y son los poseedores del poder de publicación y actualización de contenidos en la Red.

El éxito de las páginas web trae consigo la introducción de los primeros elementos de interacción dentro de una página web: los *formularios*. Un formulario está constituido por cajas de texto, elementos de opción y de selección y botones que piden información al usuario desde el equipo cliente. A través de un *evento* (*POST* o *GET*) dicha información es enviada al servidor con el fin de realizar un conjunto de tareas que darán un resultado el cual es mostrado al usuario a través de una nueva página web. Ejemplos de este tipo de dinamismo en las páginas es el registro de un usuario en un servicio web (Hotmail, Gmail, etc.) a través de un formulario presentado por medio de una página web. La estrategia dinámica permite observar más claramente los dos componentes involucrados en la creación de páginas web: de un lado el servidor o proveedor de la página web y por otro el cliente o visualizador de la página web, Figura 4.1.

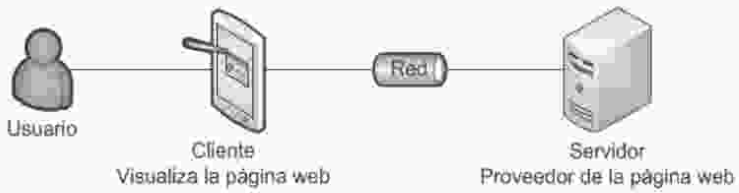


Figura 4.1. *Arquitectura física clásica para la construcción de páginas web*

En esta arquitectura la lógica del negocio está totalmente separada de la página web, sin embargo, el diseño sigue estando atado a etiquetas HTML, a código estático y al servidor que es quien provee la página. La siguiente evolución en la creación de páginas web dinámicas es el *código embebido* por el cual una página ya no solo contiene código HTML, sino también posee instrucciones de lenguajes como JSP o PHP que se embeben en el código dando inteligencia al lado del servidor. Este tipo de código se denomina embebido porque está sumergido en el código HTML, es decir, se mezcla con él. El éxito del código embebido es que este se ejecuta en el servidor y además, que dicha ejecución se realiza siempre que una página con código embebido es pedida. Esto quiere decir, que gracias al código embebido el servidor puede dar distintas respuestas con la misma página. El código embebido podrá leer cuál ha sido la selección en el momento en el que se ejecuto la página o las preferencias del usuario y de acuerdo a dichos parámetros arrojará resultados distintos y de hecho, la página web mostrará valores distintos. La última etapa en el dinamismo de las páginas web está dada por la inteligencia no solo en el lado del servidor sino también en el lado del cliente, por ejemplo, AJAX en el cual las páginas tienen la potestad de cambiar de acuerdo a la petición del usuario no solo desde el lado servidor sino también desde el lado cliente.

Todas estas evoluciones han permitido que la nueva Web sea más dinámica y que los contenidos no sean publicados únicamente por expertos *webmasters* sino por cualquier usuario que tenga mínimos conocimientos de informática y posea una interfaz adecuada de edición de código web. Además, se ha logrado la *independencia* entre el *diseño de la página* (menús, imágenes, colores de la página) y la *lógica del negocio*, es decir, los procesos que deben ejecutarse para lograr la respuesta deseada por el usuario final se ejecutan de manera independiente de la forma en la que se muestra la página. Esto ha permitido una especialización dentro de los profesionales involucrados en la creación de páginas web, ya que permite que los diseñadores se dediquen exclusivamente a cambiar el formato de los títulos y demás aspectos estructurales de la página web sin interferir en la labor de los desarrolladores de funcionalidad que se encargan de analizar, diseñar y construir los programas de acuerdo al tipo de servicio requerido. Esto permite que haya profesionales que se dedique a escribir los contenidos, otros al diseño de la página y otros a la inteligencia del negocio. Tres roles diferentes fusionados para crear un mismo elemento visual, una página web. Ejemplo de esta separación de roles puede ser la edición digital de un diario: los periodistas se encargan de los contenidos, los diseñadores de la página web se encargan de organizar la forma en la cual se presentarán las noticias y los informáticos se encargarán de que las funcionalidades de la página web como son los elementos interactivos, vídeos, RSS o similares se ejecuten correctamente. Uno de los tipos de herramientas que permiten administrar esta división de roles son los *gestores de contenido* o CMS, como por ejemplo, Joomla, donde se separa la imagen del contenido.

Pero, ¿cómo se puede reflejar esa independencia del diseño de página y la lógica del negocio a la hora de construir una aplicación o servicio web? La siguiente sección nos presenta las diferentes estrategias para dividir la lógica del negocio del diseño de las páginas web.

4.1 MECANISMOS DE SEPARACIÓN DE LA LÓGICA DE NEGOCIO

Como ya hemos dicho anteriormente, la creación de páginas web dinámicas permite la separación entre el diseño de la página y la lógica del negocio. Esta separación es una decisión de tipo *arquitectónico*, que tiene repercusiones tanto a nivel del modelo físico como del modelo lógico en la construcción de la aplicación web.

A nivel del modelo físico, los sistemas web actuales están guiados por las *arquitecturas multinivel* (del inglés, *multitier*). Este tipo de arquitectura propone distribuir la infraestructura, es decir los elementos de hardware en los que se ejecutarán los procesos y que constituyen el sistema, en niveles. Así, cada tipo de elemento que tenga un rol distinto representará una capa diferente. Este tipo de arquitecturas las explicaremos en la sección 4.1.1.

A nivel lógico, las aplicaciones web se desarrollan siguiendo un modelo de *arquitectura de capas* (del inglés, *multilayer*). Estas arquitecturas dividen el desarrollo de una aplicación en varias capas de tal manera que el trabajo de cada capa puede ser asignado a un equipo de trabajo diferente y solo basta con que el equipo conozca el modo de comunicarse con las otras capas que componen el sistema desarrollado. De manera concreta, estas arquitecturas dividen la forma en la que se organiza el código de de la aplicación sin tener en cuenta su distribución física, es decir podemos decidir tener toda la aplicación ejecutándose sobre una misma máquina o dividirlo en diferentes equipos sin afectar nuestra visión de la arquitectura. Las arquitecturas multicapas las explicaremos de forma exhaustiva en el apartado 4.1.2.

Es importante resaltar la diferencia entre nivel y capa, ya que muchas veces las dos palabras son utilizadas de manera indistinta. Sin embargo, veremos cómo un *nivel* corresponde a la forma física en la que se organiza una aplicación mientras que una *capa* hace referencia a las distintas partes en que se divide una aplicación.

4.1.1 MODELOS FÍSICOS DE SEPARACIÓN: ARQUITECTURAS MULTINIVEL

Cuando hablamos del modelo físico arquitectónico o de la arquitectura física, nos referimos a la forma en la cual se distribuye la infraestructura, es decir, los elementos de hardware en los que se ejecutarán los procesos y que constituyen el sistema. De manera genérica, los sistemas web actuales están guiados por las *arquitecturas multinivel* (del inglés, *multitier*). Este modelo de arquitectura deriva de la arquitectura Cliente-Servidor (expuesta en el capítulo 1) y proponen distribuir, los elementos arquitectónicos de infraestructura en niveles. Así, cada tipo de elemento que tenga un rol distinto representará una capa diferente. En el caso web, la mayoría de los sistemas presentan una arquitectura de dos o más niveles. Una arquitectura de dos niveles significa que las funciones estarán divididas en dos equipos de cómputo diferentes y equivale al modelo cliente-servidor tradicional. Ejemplo de una arquitectura de dos capas es un modelo en el cual el nivel uno representa al cliente quien a través de algún dispositivo de acceso a Internet (portátil, teléfono móvil, *tablet*, PDA) se comunica con el nivel dos en el cual se encuentra el equipo servidor que ejecutará los procesos y devolverá la respuesta al cliente. Este tipo de arquitectura es ilustrado en la Figura 4.2.

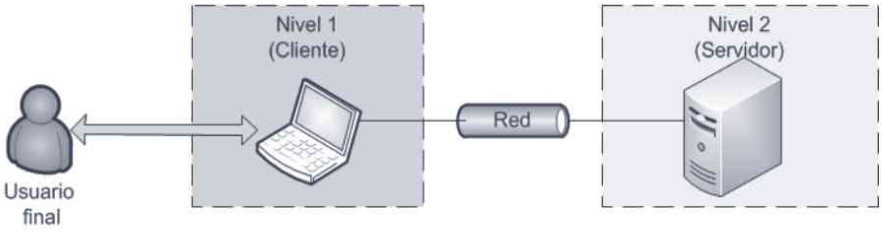


Figura 4.2. *Ejemplo de arquitectura de 2 niveles*

Si la arquitectura física del sistema es de tres niveles quiere decir que tenemos otro elemento hardware involucrado en el funcionamiento de nuestro sistema. Por lo general, este nuevo elemento desdobla las funciones del servidor es decir, el nuevo nivel casi siempre representa a un nuevo servidor que descarga funcionalidad y carga de trabajo al servidor único. La Figura 4.3 representa una arquitectura de tres niveles.

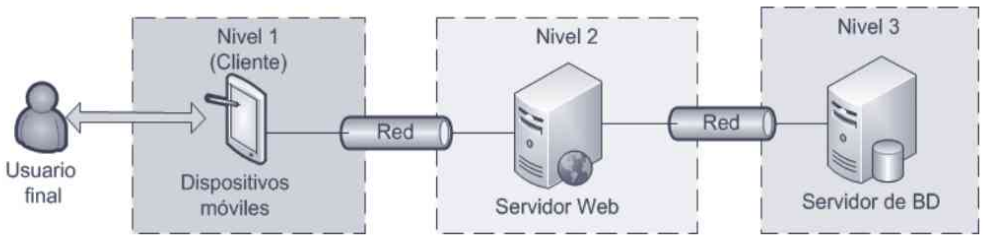


Figura 4.3. *Ejemplo de arquitectura a tres niveles*

En el caso de la Figura 4.3, el servidor único de la Figura 4.2 se desdobla en dos equipos: un servidor web (nivel 2) encargado de ofrecer las prestaciones para aceptar y contestar peticiones web y un servidor de base de datos (nivel 3) encargado de almacenar y gestionar el acceso a la información.

Las arquitecturas multinivel pueden tener tantos niveles como equipos involucrados en el manejo de la información y el procesamiento existan. La Figura 4.4 y la Figura 4.5 muestran dos ejemplos de arquitecturas de 4 niveles. Es importante anotar como aunque las dos tienen 4 niveles, los roles de cada nivel no son el mismo para cada ejemplo.

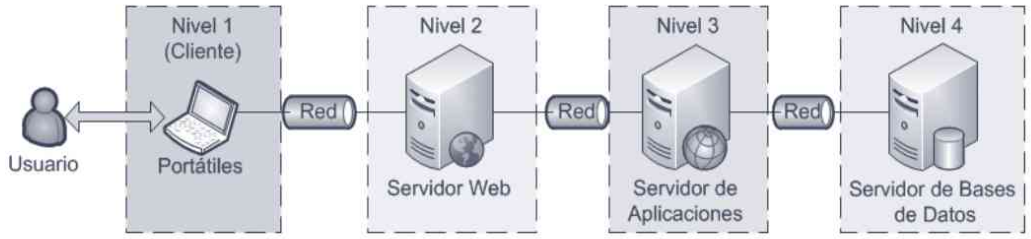


Figura 4.4. *Arquitectura estándar para la construcción de páginas web*

Para el caso de la Figura 4.4, existen dos equipos diferentes, el servidor web y el servidor de aplicaciones encargados de la gestión y el procesamiento de las peticiones recibidas de parte del cliente. Este caso es otro ejemplo de cómo la división en niveles corresponde a una división funcional, de manera que un equipo es destinado a una función particular y así crea un nuevo nivel en la arquitectura. Además, este ejemplo nos sirve para ilustrar la diferencia funcional entre un servidor web y un servidor de aplicaciones. Para aclarar los conceptos hagamos un poco de historia. Cuando se crearon los primeros servidores de páginas web (*web servers*), como Apache, su única misión era recuperar páginas web estáticas de su disco duro y enviárselas al cliente. Para cualquier otro tipo de información que debiera generarse de manera dinámica (respuestas a búsquedas, etc.) el servidor tenía que ceder el control a algún tipo de código externo mediante CGI el cual ejecutaba un *script* (programa). Con el paso del tiempo el uso de servidores web se generalizó y se hizo necesario incrementar los servicios ofrecidos ya que llamar a un interpretador para que ejecutara otro programa suponía una demanda muy fuerte sobre el equipo que mantenía el servidor de páginas (servidor web). Finalmente la evolución ha llevado a crear un nuevo término: servidor de aplicaciones (*Application server*). Hoy en día, se puede decir que todos los servidores web actuales son también servidores de aplicaciones, ya que incluyen alguna tecnología (CGI, PHP, JSP, etc.) que permite realizar aplicaciones que generan contenido dinámico o aplicaciones de servidor. Dependiendo de la funcionalidad se trae consigo complejidad al sistema, ya sea en la forma de requerimientos del sistema (memoria, procesadores), carga administrativa (configuración, tiempo de desarrollo) o alguna otra.

Teniendo en cuenta que la tendencia de que cualquier servidor web sea un servidor de aplicaciones, la Figura 4.5 muestra un ejemplo de arquitectura de 4 niveles con este rol. En este caso, mostramos la arquitectura de YouTube en sus inicios, la cual fue explicada por su jefe en el vídeo (Cuong, 2007).

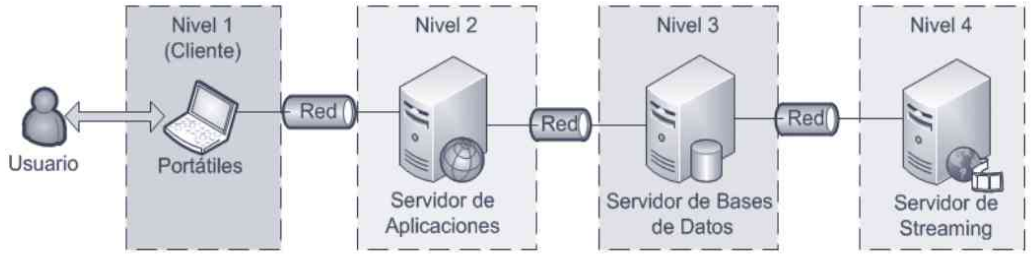


Figura 4.5. *Otro ejemplo de arquitectura a cuatro niveles*

Si estas interesado en otros ejemplos de construcción de sistemas web escalables como Flickr, puedes consultar el libro *Building Scalable Web Sites* (Henderson, 2006).

4.1.2 MODELOS DE SEPARACIÓN LÓGICOS

En contraste con los modelos físicos, cuando hablamos del modelo arquitectónico lógico o de la arquitectura lógica del sistema, nos referimos a la forma, en la cual elegimos dividir el software para obtener el mejor rendimiento del sistema. En los modelos lógicos se distribuyen los componentes software del sistema. Y así como pueden existir sistemas físicos distribuidos también pueden existir sistemas lógicos distribuidos.

Uno de los mecanismos más usado a la hora de construir páginas web es el uso de las *arquitecturas de capas* (del inglés, *multilayer*). Estas arquitecturas dividen el desarrollo de una aplicación en varios niveles lógicos. Estos niveles son formas de organizar el código. Hay que aclarar que de ninguna manera esta división en capas implica ejecutar cada una de las capas en diferentes equipos de cómputo. La construcción de software, y específicamente de páginas web, por capas trae como principal ventaja la organización y agrupamiento por funcionalidad. Esta agrupación ofrece reusabilidad, facilidad de mantenimiento y ciclos de desarrollo más cortos. Todas estas características redundan en una reducción de los costos de desarrollo y de mantenimiento de la aplicación. Otras ventajas del modelo son:

- **Desarrollos paralelos en cada capa**.
- **Aplicaciones más robustas debido al encapsulamiento**.
- **Mantenimiento y soporte más sencillo**. Es más fácil cambiar un componente que modificar una aplicación monolítica.
- **Mayor flexibilidad**. Podemos añadir nuevos módulos para dotar al sistema de nueva funcionalidad.
- **Alta escalabilidad**. Podemos manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más hardware. El crecimiento es casi lineal y no es necesario añadir más código para conseguir esta escalabilidad.

Las arquitecturas en capas son una de las formas de implementación de un patrón arquitectónico denominado *Modelo-Vista-Controlador* (MVC).

EL ESQUEMA MODELO-VISTA-CONTROLADOR (MCV)

Este es uno de los esquemas más seguidos e implementados a la hora de construir aplicaciones, no solo a nivel web, sino también aplicaciones *standalone*. Una arquitectura siguiendo el esquema Modelo-Vista-Controlador busca separar una aplicación en tres componentes principales: el modelo, la vista y el controlador.

- **Modelo**. Los modelos son las partes de la aplicación que implementan la lógica de la aplicación para un dominio específico. Esa es la representación de la información con la cual se opera. Usualmente, los modelos devuelven y almacenan el estado del modelo en una base de datos. En modelos pequeños el modelo es frecuentemente una separación conceptual más que una separación física.
- **Vista**. Las vistas son los componentes que despliegan la interfaz de usuario (UI – *User Interface*). Generalmente, esta interfaz es construida de acuerdo al modelo de datos y en el caso de las aplicaciones web está constituido por el conjunto de páginas web que muestran y recogen la información del usuario.
- **Controlador**. Los controladores son los componentes que manejan la interacción con el usuario, trabajan con el modelo y seleccionan cuál es la vista correcta a desplegar para mostrar la información.

El patrón MVC nos ayuda a crear aplicaciones que separan los diferentes aspectos de la aplicación, la lógica de entradas, la lógica de negocio y la lógica de interfaz, generando un acoplamiento pequeño entre cada una de estas partes. Así el patrón especifica que cada una de estas lógicas debe corresponder con un elemento. La lógica de interfaz pertenecerá a la vista, la lógica de entradas pertenecerá al controlador y la lógica del negocio estará reflejada en el modelo. Esta separación ayuda al creador de una aplicación web a manejar la complejidad del desarrollo ya que

le permite centrarse en solo un aspecto del sistema a la vez. El bajo acoplamiento de estos elementos promueve el desarrollo en paralelo. La Figura 4.6 muestra gráficamente este esquema en el entorno de una aplicación web.

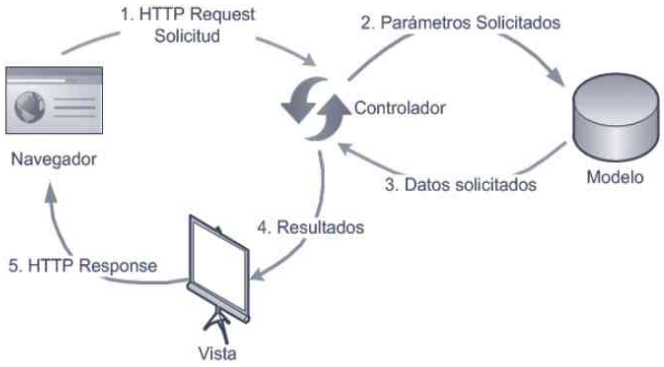


Figura 4.6. *Modelo-Vista-Controlador (MVC)*

Muchas de las soluciones que se ofrecen actualmente siguen este modelo o alguna de sus variaciones. De hecho, veremos cómo las arquitecturas en capas constituyen una derivación de este esquema.

Algunos marcos de trabajo para aplicaciones web siguiendo MVC son Microsoft's ASP .Net MVC Framework (*www.asp.net/mvc*), Oracle's Aplication Development Framework (ADF) (*http://www.oracle.com/technetwork/developer-tools/adf/ index.HTML*), PHP Faces (*http://code.google.com/p/php-faces/*) entre otros

LA ARQUITECTURA DE 3 CAPAS

Dentro de las arquitecturas multicapa, la *arquitectura de 3 capas* es una de las más utilizadas para el desarrollo de sitios web. Esta arquitectura divide la creación de una aplicación en tres niveles: *capa de presentación*, *capa de la lógica de negocio* y *capa de persistencia* (almacenamiento de datos) (Fowler, 2002). Está basado en el concepto de que todos los niveles de una aplicación son una colección de componentes que se proporcionan servicios entre sí o a otros niveles adyacentes. El modelo de tres capas está destinado a ayudar a construir componentes de software a partir de la división de los niveles lógicos. De esta manera, una de las decisiones de diseño que debemos tomar es que parte de la lógica de la aplicación se va a encapsular en cada uno de los componentes, así como que componente se va a encapsular en cada nivel. Un nivel puede estar conformado por varios componentes, por tanto, puede suplir varios servicios. La Figura 4.7 muestra un esquema de una arquitectura de tres capas. Los elementos que contiene cada una de las capas corresponden a componentes o funcionalidades manejados en dicha capa.

Figura 4.7. *Arquitectura de 3 capas en aplicaciones web desde el punto de vista funcional*

A continuación realizaremos una breve explicación de cada una de las capas de esta arquitectura.

Capa de presentación

Esta capa es la más cercana al usuario presentándole una interfaz gráfica del recurso solicitado y capturando la interacción entre el sistema y el usuario. También es conocida como capa de interfaz gráfica. Esta capa se comunica con la capa de negocio y suele estar dividida entre los niveles del cliente y el servidor.

En esta capa, los componentes son responsables de solicitar y recibir servicios de otros componentes de la misma capa o de la capa de negocio. Las tareas de esta capa se centran en la presentación y el formateo de la información enviada para que esta pueda ser visualizada correctamente por el usuario. En el caso de sistemas web, esta capa debe garantizar que la información sea accesible través de recursos web claros y amigables de tal manera que dicha información sea accesible a través de un navegador web (IExplorer, Chrome, Mozilla, Safari, etc.) en cualquier dispositivo de acceso (portátil, *tablet*, *smartphone*, etc.).

Algunas de las tareas específicas de la capa de presentación son: proveer las páginas web, garantizar que dichas páginas puedan ser visualizadas tanto en un ordenador como en un dispositivo *smartphone*, gestionar que todos los enlaces del sitio web sean válidos, garantizar que las páginas web tengan siempre la información actualizada, etc.

Capa de negocio

También denominada capa de lógica, capa de aplicación, capa media o capa de lógica de negocio. Es la capa que gestiona las funcionalidades del sistema o aplicación web ("lógica o reglas de negocio"). Habitualmente esta capa recibe las peticiones del usuario y desde ella se envían las respuestas apropiadas tras el procesamiento de la información proporcionada por el cliente. Se denomina capa de negocio porque es aquí donde se establecen todas las reglas que deben cumplirse para que las funciones de la aplicación web sean ejecutadas correctamente. La implementación de dichas reglas se hace a través de desarrollos software los cuales se ejecutan cuando llega la petición de un usuario. Al igual que la capa de presentación, la lógica de negocio puede ser programada tanto en el entorno cliente como en el entorno servidor. Algunas de las tareas concretas de los componentes de esta capa son: coordinar la aplicación web, procesar los comandos, realizar la toma de decisiones, realizar los cálculos respectivos. Además, esta capa es la encargada de mover y procesar los datos entre sus dos capas adyacentes. Típicamente, la capa del negocio está conformada por uno o más módulos que se ejecutan en un equipo de cómputo denominado servidor de aplicaciones.

Capa de persistencia

Esta capa es la encargada del acceso a los datos. Aquí se encuentra toda la codificación necesaria tanto para el acceso a los datos como para el manejo y almacenamiento de los mismos. Normalmente está formada por un conjunto de componentes que se encargan del acceso a los datos (p. ej. clases de conexión a la base de datos) los cuales se ejecutan en los servidores de aplicaciones y por uno o más sistemas gestores de bases de datos (SGBD) que se ejecutan en el nivel de datos (servidores de bases de datos) los cuales se encargan de todo el proceso de administración de datos, ejecutando las solicitudes de almacenamiento, modificación o recuperación de información dadas desde la capa de negocio. Los SGBD.

Una de las confusiones más corrientes es considerar que cada uno de los niveles deberá corresponder a una capa dentro de la arquitectura, es decir, hacer una correspondencia 1 a 1 entre niveles y capas. Esto no es necesariamente así. Como hemos explicado, muchos de los componentes de la capa de Presentación, por ejemplo, pueden estar distribuidos a nivel de cliente y otros a nivel de servidor. Así mismo, las utilidades del sistema, pueden están distribuidos en varios servidores, es decir, las funcionalidades de la capa de negocio y por ende cada servidor constituirá un nivel diferente. La Figura 4.8 muestra una comparación entre un modelo físico tradicional organizado en tres niveles y un modelo lógico de tres capas. En este caso, el nivel 1 del modelo físico se encarga de algunas funcionalidades de navegación y presentación a través del browser. El nivel dos, correspondiente al servidor de aplicaciones contiene todas las funcionalidades de la capa de negocio y algunas de la capa de presentación, como por ejemplo la validación o el control

de conexiones. Por último, el nivel 3 que contiene un servidor de bases de datos, contiene una base de datos que se encarga de las funciones de servicios de datos, almacenamiento y seguridad descritos para esta capa.

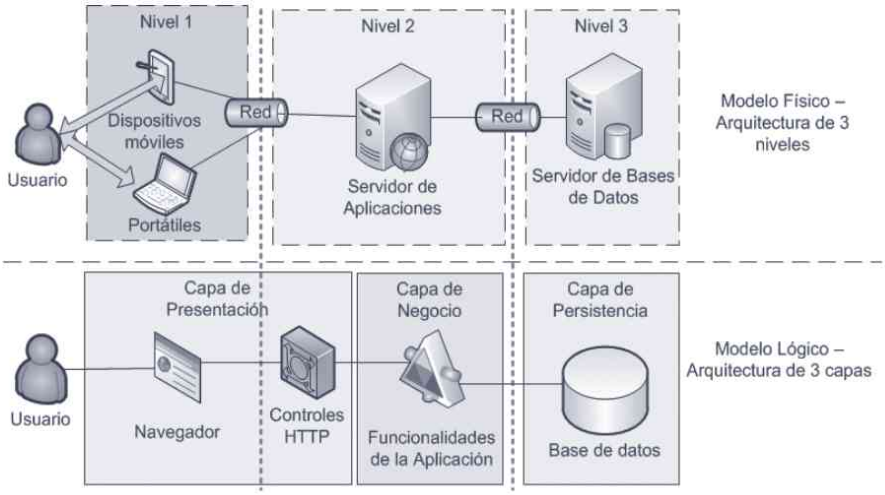


Figura 4.8. *Arquitectura de 3 capas en aplicaciones web*

ARQUITECTURAS MULTI-CAPA

Las nuevas tendencias en el desarrollo web así como el uso de la programación orientada a objetos crean nuevas formas de separación de la lógica del negocio.

Con el modelo de tres capas si se desea cambiar la interfaz gráfica de una aplicación web, este cambio solo afectará a los componentes de la capa de presentación sin afectar al resto de componentes de la aplicación. Sin embargo, en el momento de analizar el sistema y programarlo, los componentes de la capa lógica necesitan referenciar instancias de las clases de dominio (las que representan las entidades del negocio y que posteriormente se plasmarán en entidades en la base de datos) y los componentes de la capa de acceso a datos también tienen que referenciarlas para poder "rellenar" tales instancias con los datos que obtienen de las capas inferiores. Siguiendo la arquitectura de 3 capas, una posible solución puede ser la declaración de las entidades de BD en la capa de acceso a datos. Sin embargo, hay que tener en cuenta que con este esquema se incluiría en la capa de acceso a datos uno de los aspectos más importantes del desarrollo que es la definición del dominio de la aplicación con lo que los cambios en la capa de acceso a datos pueden impactar en las entidades. Esta solución puede funcionar para aplicaciones pequeñas donde los cambios pueden controlarse.

Teniendo en cuenta que el objetivo es no tocar la capa de lógica de negocio cuando haya un cambio en el nivel de datos, una mejor solución sería la creación de otra capa más, la capa de entidades que corresponde al dominio de la aplicación. En esta capa se encontraría el mapeo de la declaración de las entidades de la aplicación. Además este esquema permite una total independencia entre la lógica de negocio (modelo de negocio) y las entidades (modelo del dominio). Por otro lado, este esquema facilita la incorporación en la capa de persistencia de componentes tipo ORM (Mapeo objeto/relacional - *Object / Relational Mapping*) que permiten mapear objetos en un esquema relacional. La Figura 4.9 muestra esta nueva separación incluyendo la capa de entidades.

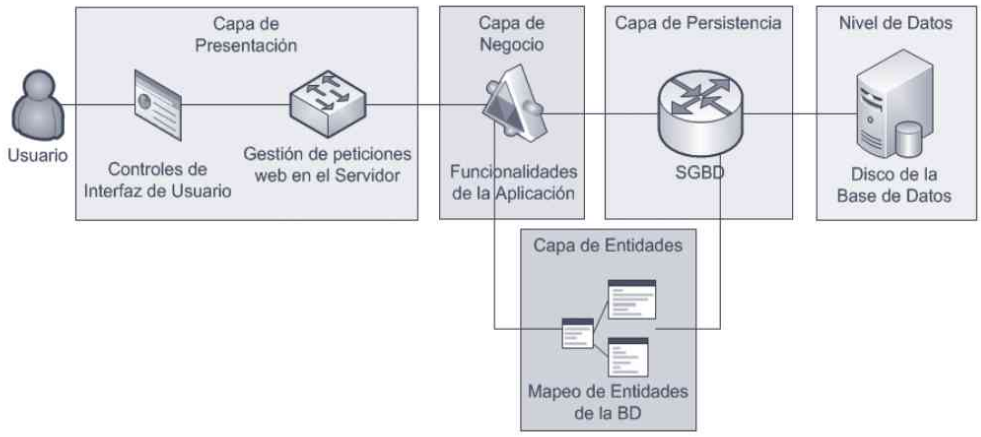


Figura 4.9. *Modelo de tres capas mejorado insertando una capa de entidades*

En los ejemplos presentados anteriormente, siempre la capa de persistencia está constituida por una base de datos. Sin embargo, esta capa de persistencia puede tener recursos de otro tipo, por ejemplo vídeos o información particular tratada de manera independiente. En estos casos, esta capa cambia su nombre a *capa de recursos*. En esta nueva capa, los recursos pueden ser ya establecidos como recursos de persistencia traducidos en sistemas de gestión de bases de datos o nuevos recursos que deberán ser implementados. Por ello, a la hora de diseñar la arquitectura del sistema habrá que tener en cuenta que ese nuevo recurso de la capa de recursos deberá cubrir funciones claves para garantizar que la capa del negocio se pueda conectar con él.

La Figura 4.10 muestra la conexión de la capa de negocio con dos capas: una de persistencia y otra de recursos. En la primera capa representamos un SGBD con sus componentes incluidos dentro del mismo, mientras que la segunda capa representa un servicio no creado donde los componentes ya incluidos dentro de un sistema preestablecido, como un SGBD, deberán ser implementados. En realidad, no es que esas funciones no existan, la cuestión es que cuando hablamos de un SGBD estas funciones ya van inmersas en dicho sistema y por ello son transparentes a nosotros. Pero, si nos conectamos a un sistema que definimos nosotros mismos, estos componentes deberán ser explícitos.

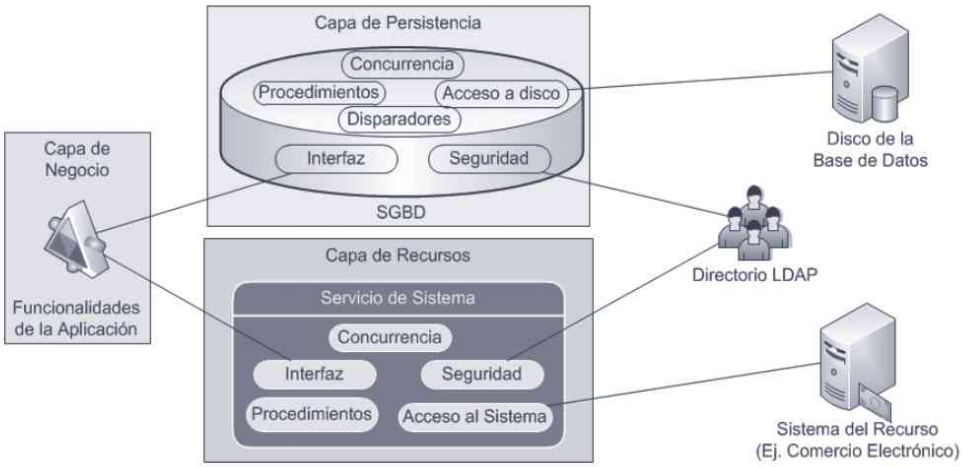


Figura 4.10. *Conexión a capas de persistencia y/o a capas de recursos*

Por último, es importante tener en cuenta que existen un conjunto de aspectos estructurales cuya funcionalidad no se implementen en una sola capa, sino que por el contrario, tienen que estar distribuidos por todas las capas, es decir, son aspectos transversales a toda la arquitectura. Aspectos como la comunicación, el rendimiento, la escalabilidad y la seguridad pertenecen a esta categoría. La Figura 4.11 muestra la arquitectura de tres capas de forma vertical y agrega los aspectos transversales antes comentados.

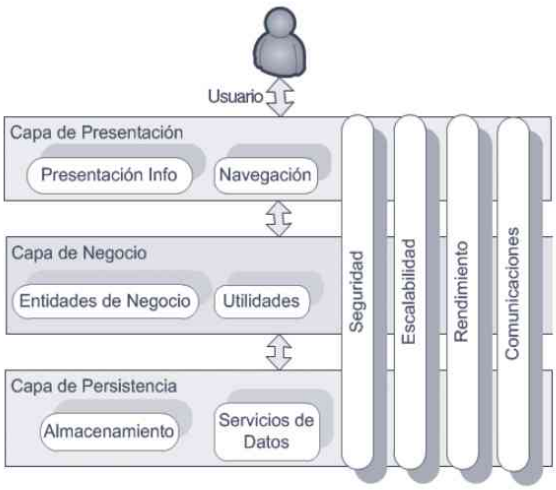


Figura 4.11. *Aspectos transversales a la división de la lógica del negocio en capas*

LA ARQUITECTURA ORIENTADA A SERVICIOS (SOA)

El nivel de cooperación que presentan las organizaciones requiere que las aplicaciones de software desarrolladas por dichas organizaciones interactúen unas con otras. El problema es que algunas de ellas no se ejecutan en la misma plataforma o están desarrolladas con marcos de trabajo diferentes. La solución fue presentada como la arquitectura orientada a servicios (SOA, *Service Oriented Architecture*), que brinda entre otras cosas una forma estándar de publicar y utilizar servicios, conocidos comúnmente como servicios web (*web services*). De esta manera una aplicación es vista como un conjunto de servicios. Los servicios web intercambian mensajes en formato XML utilizando protocolos de transporte como HTTP. Los servicios web, básicamente, establecen un lenguaje común mediante el cual distintos sistemas pueden comunicarse entre sí y, de esta forma, facilitan la construcción de sistemas distribuidos heterogéneos. De esta manera, una organización expone sus competencias (funcionalidades o capacidades) para que sean utilizados por la misma organización o por otras organizaciones.

SOA permite emular el comportamiento de los negocios en el mundo real. En una arquitectura orientada a servicios, los usuarios finales, mediante la utilización de hardware y/o software liviano, como un navegador web, pueden acceder a lo que se denomina el nivel de clientes o aplicaciones que básicamente está constituida por la capa de presentación y consumen los servicios publicados por una organización. Este tipo de servicios son en general sitios o portales en la Web. Las aplicaciones de otras organizaciones también pueden utilizar los servicios publicados por una organización concreta; esta acción requiere de acuerdos comerciales y credenciales para autenticar y autorizar a quienes consumen los servicios. De acuerdo a lo expuesto por DeCastro (Castro, 2010), el proceso de orientación a servicios puede dividirse en cuatro capas. Estas capas son expuestas en la Figura 4.12.

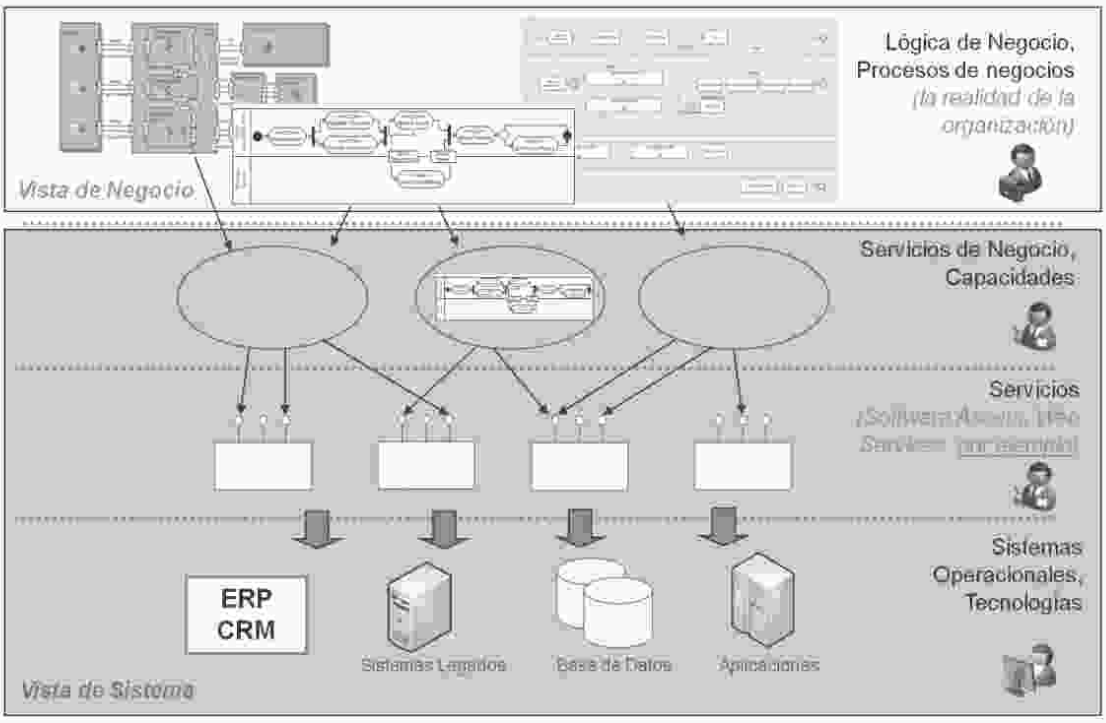


Figura 4.12. *Capas de la Arquitectura Orientada a Servicios. Extraído de DeCastro (Castro, 2010)*

Podemos observar como en la orientación a servicios, la lógica del negocio no es vista como una capa a la hora de realizar la aplicación sino como una capa de análisis en la que se determinan los *procesos de negocios*. Es decir, la lógica del negocio se refiere a la realidad de la organización, a los procesos que guían la manera en la que se hacen las cosas en el mundo real, a los procesos que especifican los requisitos y proponen soluciones al negocio. Esta labor es realizada por los analistas del negocio o los arquitectos del negocio. Las dos capas de servicios, servicios de negocio y servicios, se centran en comprender cuál es el objetivo del negocio y verifican qué servicios son requeridos para suplir los requerimientos. Además, en estas capas se encargan de crear las estrategias de integración y reutilización de los servicios. Esta labor es realizada por analistas de software o arquitectos de servicios.

Finalmente, la capa de los sistemas operacionales es implementada por analistas de sistemas o desarrolladores de servicios, los cuales entienden claramente cuáles son los pasos para el desarrollo e integración de servicios proponiendo soluciones técnicas óptimas que permitan la implementación tanto de los servicios como de la solución. Es en esta capa donde se desarrollan los servicios, que se denominan servicios web. Estos servicios están soportados bajo protocolos como SOAP (Protocolo de Acceso a Objetos Simples - *Simple Object Access Protocol*) que es el protocolo de comunicación entre aplicaciones y servicios web a través de mensajes por medio de Internet, UDDI (*Universal Discovery Description and Integration*) que es un modelo de directorios para servicios web y WSDL (Lenguaje de Descripción de Servicios Web – *Web Services Description Language*) el cual es un protocolo basado en XML que describe los accesos a un servicio web, es decir, describe las interfaces del servicio web y cómo utilizarlas. Una vez construidos los servicios web, estos podrán ser utilizados y llamados por las páginas web, por ello muchos de los lenguajes de construcción de páginas web dinámicas como PHP o ASP.Net soportan el llamado servicios web. La forma en la cual se pueden programar servicio web será explicada en el capítulo 7.

4.1.3 PATRONES DE SOFTWARE EN LA WEB

En los últimos años, los patrones se han convertido en uno de los *trending topics* en la ingeniería del software. Podemos definir los *patrones* como el esqueleto de las soluciones a problemas comunes en el desarrollo de software. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.

Para que una solución sea considerada como patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas en distintas circunstancias. Por ello, un patrón es una forma literaria de documentar las mejores prácticas y lecciones aprendidas en la resolución de un problema complejo dentro de un dominio de diseño concreto.

La documentación de un patrón es una tarea crucial, ya que de su buena definición y explicación dependerá de que la solución propuesta se entienda. Varios autores han propuesto formas de describir un patrón. Por su completitud, nos vamos a quedar con la propuesta de Brad Appleton (Appleton, 2000) acerca de los elementos que debería tener un patrón, sin embargo, enunciaremos en negrita cuales de estos elementos son los comunes a otras propuestas. Los elementos de un patrón enunciados por Appleton son:

- **Nombre**. Es un nombre descriptivo y único que ayuda a identificar y referenciar al patrón.
- **Problema**. Descripción resumida en una o dos frases que nos describe la intención del patrón, es decir, las metas y objetivos que queremos alcanzar.
- **Contexto**. Problema recurrente en el que es aplicable el patrón. Suelen usarse ejemplos del estado inicial del sistema antes de que el patrón sea aplicado.
- **Fuerzas**. Descripción de las fuerzas, los objetivos y restricciones relevantes para ese patrón, y de cómo éstas interaccionan entre ellas o con las metas que deseamos alcanzar. Además puede incluir un escenario concreto que sirva de motivación para el patrón. La noción de fuerza generaliza los tipos de criterios que justifican al patrón.
- **Solución**. Es el corazón del patrón. Está formado por un conjunto de instrucciones que describen cómo construir la solución del problema. Esta descripción puede ir acompañada de dibujos, diagramas o esquemas explicativos de dicho patrón.
- **Ejemplos**. Casos prácticos, pueden ser visuales, que ayudan al lector a entender el uso y la aplicabilidad del patrón.
- **Contexto resultante**. Indica el estado del sistema después de aplicar el patrón, incluyendo sus consecuencias (positivas y negativas).
- **Exposición razonada**. Expone cómo funciona el patrón y por qué es útil. Mientras que la solución muestra la estructura visible del patrón, la exposición explica sus mecanismos subyacentes.
- **Patrones relacionados**. Patrones que se pueden combinar con este, o es posible aplicar a partir del contexto resultante, o representan soluciones alternativas.

Figura 4.13. *Un patrón y sus características. De acuerdo al presentado por (Montero, Zarraonadía, Díaz, & Aedo, 2010)*

El éxito de los patrones ha generado que existan una gran cantidad de tipos. Entre las categorías tradicionales se encuentran:

- **Patrones de arquitectura**. Son aquellos que expresan un esquema organizativo o estructural fundamental para sistemas de software. Entre los patrones arquitectónicos se encuentran, MVC, las arquitecturas por niveles y las arquitecturas por capas descritas en los apartados anteriores.
- **Patrones de diseño**. Estos son los tipos de patrones más conocidos y son los que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software. Los patrones de diseño casi siempre resuelven problemas de diseño de código. A su vez, este tipo de patrones se divide en tres categorías:
  - **De Creación**. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
  - **Estructurales**. Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.
  - **De Comportamiento**. Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

  Uno de los catálogos más importantes sobre de patrones de diseño pertenece al Clan de los Cuatro (Gamma, 2006). Puedes consultarlo para ampliar más acerca de este tema.
- **Dialectos**. Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

Además de los patrones ya vistos actualmente existen otros patrones como los siguientes:

- **Patrones de interacción**. Son patrones que nos permiten el diseño de interfaces web.
- **Patrones de análisis**. Describen un conjunto de prácticas destinadas a elaborar modelos de los conceptos principales de la aplicación que se va a construir (Fowler, 1996). Estos patrones apoyan el trabajo de modelado, pues no siempre tienen experiencia al respecto y, en la mayoría de los casos, construyen sus modelos sin referencia alguna. Los patrones de análisis se diferencian de los de diseño en que se centran en aspectos sociales, de organización y económicos, los cuales son primordiales en el análisis de requisitos y la aceptación y usabilidad del sistema final.
- **Patrones de dominio**. Se caracterizan por dar soluciones a un dominio específico. Sirven como referencia conceptual del dominio del problema, ayudándonos a identificar cuáles son los requisitos o características más relevantes de ese dominio.

Pero, ¿qué otros patrones podemos usar dentro del desarrollo de una aplicación web? Montero y otros (Montero et al., 2010) presentan una excelente recopilación de patrones que pueden ser usados para la construcción de una página web enfocada a *e-learning*. Sin embargo, consideramos que dicha recopilación podría aplicar para cualquier página web.

La Tabla 4.1 recopila todos los patrones expuestos por estos autores y que están relacionados con la construcción de la página web.

Tabla 4.1 Selección de patrones web para aplicaciones web. Extraída de (Montero et al., 2010)

Navegación	
Nombre del patrón	**Descripción**
Index Navigation (Garzotto et al., 1999)	Proporcionar un acceso rápido a un conjunto de conceptos para que los usuarios que estén interesados en uno o varios de ellos puedan realizar su elección.
Guided Tour Navigation (Garzotto et al., 1999)	El usuario debe poder acceder de manera secuencial a un grupo de conceptos relacionados.
Organización	
Nombre del patrón	**Descripción**
Hierarchical Organization (van Duyne et al., 2002)	Organizar la información en una jerarquía de categorías puede ayudar al estudiante a encontrar las cosas (p. ej. siguiendo la estructura de un curso, capítulos, lecciones u organizarlo por contenidos).
Task-based Organization (van Duyne et al., 2002)	Completar un conjunto de tareas relacionadas de una manera rápida y sencilla enlazando dichas tareas según la secuencia en que se deben realizar.

Presentación	
Nombre del patrón	**Descripción**
Navigation Bar (van Duyne et al., 2002)	El usuario debe encontrar siempre visible y de manera consistente las herramientas de ayuda a la navegación.
Define and Run Presentation (Cybulski y Linden, 1999)	El usuario debe percibir los elementos multimedia como una composición estética, donde un conjunto de elementos son mostrados de manera secuencial en uno o varios canales.
Personalización	
Nombre del patrón	**Descripción**
Structure Personalization (Rossi et al., 2001)	El usuario debe acceder solo a la información que le interesa.
Content Personalization (Rossi et al., 2001)	El usuario debe recibir los contenidos de manera personalizada.
Usabilidad	
Nombre del patrón	**Descripción**
Sentido de localización	El usuario necesita saber su localización dentro del sitio web, indicándole dónde está, en qué contexto y la ruta de información seguida para llegar a ese punto.
Volver a un sitio seguro	Los usuarios pueden sentirse desorientados o perdidos cuando han realizado más de una parada en el sitio web. Es necesario proporcionar un modo para que los usuarios pongan marcas para volver cuando se sientan perdidos.
Logotipo del sitio arriba a la izquierda	Los usuarios necesitan saber cómo volver a un sitio seguro, p .ej. a la página principal. Según el estándar, el logotipo se debe situar arriba a la izquierda de cada página y al pinchar sobre él, siempre te lleva a la página principal (*homepage*).
Botón de vuelta atrás	Los usuarios suelen cometer errores, especialmente al realizar tareas en varios pasos. El usuario debería ser siempre capaz de volver al paso anterior, y deshacer las cosas que haya podido hacer.
Interacción	
Nombre del patrón	**Descripción**
Wizard	Para guiar al usuario en la realización de una tarea que necesita tomar varias decisiones, mostrarle cuáles son los pasos que existen y cuáles han sido realizados a la vez que se le guía a lo largo de la tarea.

Stepping	Permitir a los usuarios ir al paso anterior y siguiente de una tarea para realizar posibles modificaciones.
Input Error	Informar al usuario de entradas de datos incorrectas, dónde se han producido y cómo resolverlas.
Outgoing Links	Mostrar al usuario los enlaces que le llevarán fuera del actual sitio web marcándolos con un icono después de su etiqueta.
Seguridad	
Nombre del patrón	**Descripción**
Authorization	Describir los tipos de acceso a los recursos del sistema, distinguiendo entre entidades activas (roles de usuario o ejecución de un proceso) y entidades pasivas.
Role-Based Access Control	Asignar los tipos de acceso a los usuarios según sus roles en el sistema.
Multilevel Security	Tratar con diferentes niveles de seguridad.

En la Red podrás encontrar mucha información sobre la implementación de patrones en entornos de desarrollo para la creación de aplicaciones web. Aquí dejamos unos cuantos sitios web que pueden ser útiles de acuerdo a la plataforma o lo que busques para la Web.

Tabla 4.2 Recursos electrónicos sobre patrones para aplicaciones web

Patrón	URL
Patrones de J2EE	*http://Java.sun.com/blueprints/corej2eepatterns/Patterns/index.HTML*
Patrones de ASP.Net	*http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=20559)*
Patrones web de Yahoo	*http://developer.yahoo.com/ypatterns/*
Patrones para Android	*http://www.androidpatterns.com/*
Catálogo de patrones para la Web	*http://patterntap.com/*
Ruby on rails	*http://www.rubyonrails.org.es/*
Patrones en PHP	*http://www.ibm.com/developerworks/library/os-php-designptrns/*

ACTIVIDADES 4.1

- Crear una arquitectura para el análisis, diseño e implementación de una página web de un diario. Los roles que deben actuar en este caso son: los periodistas, que escribirán las noticias a través de ordenadores o dispositivos móviles; los diseñadores, que se encargarán de los aspectos de presentación de la página web; por último, los *webmasters*, que deberán garantizar que las funcionalidades de la página se ejecuten. Entre las funcionalidades a cubrir por la página están: acceso a distintos usuarios de acuerdo a privilegios, permitir compartir en Twitter o Facebook una noticia o permitir suscribirse a la página a través de RSS. Toda la información deberá ser almacenada en una base de datos. Crear la arquitectura pensando en capas como en niveles.
- Esta vez, proponemos crear una arquitectura en niveles y capas para un sistema de comercio electrónico gestionado a través de una página web. Las características del sistema son:
  a. La página web es para la venta de productos de tecnología. La página permite comprar los productos a través de medios de pago electrónicos como tarjetas de crédito o a través de Paypal. Para ello, las páginas deberán conectarse con la plataforma electrónica de un banco o de Paypal según el caso. Una vez realizado el pago, los productos son enviados a domicilio a los compradores.
  b. Los compradores podrán visualizar los productos así como las características de los mismos en la página web. Esta información es extraída de una base de datos de acuerdo a la petición de los usuarios. Además, algunos productos tienen un vídeo para mostrar su funcionamiento o las cualidades de dicho producto. El vídeo deberá ser reproducido.
- Como hemos mencionado, los catálogos de patrones son muy amplios. Invitamos a que se implemente el patrón observador (*observer*) especificado por Gamma (Gamma, 2006) en su capítulo 5 de patrones de comportamiento.

4.2 MECANISMOS DE GENERACIÓN DINÁMICA DE INTERFACES WEB

De manera general, podemos definir una página web dinámica como aquella que se crea en tiempo de ejecución y puede presentar información diferente para cada usuario que esté visualizando la página de acuerdo a los requerimientos y necesidades de cada usuario.

Una de las principales características y ventajas de las aplicaciones web es que centralizan el software y de esta manera facilitan las tareas de mantenimiento y actualización de grandes sistemas (Berzal, Cubero, & Cortijo, 2005). Esto evita tener copias de la aplicación en todos los clientes que usen dicha aplicación tal y como puede suceder con las aplicaciones de escritorio, como por ejemplo cualquier herramienta ofimática. En este modelo de distribución, uno de los grandes problemas es el acceso a las actualizaciones, ya que hay que garantizar que cada cliente tenga acceso a la actualización e instale dicha actualización en su equipo de cómputo. Este problema no ocurre cuando la aplicación es accesible a través de la Web. Así, si una aplicación web ha sido actualizada, proceso que se hace siempre en el servidor, a partir de ese mismo instante todos los usuarios podrán visualizar y utilizar la versión más reciente de la aplicación sin necesidad de realizar ninguna acción adicional. Centrándonos en la construcción de la interfaces web de manera dinámica, podemos observar como su desarrollo está ligado a uso de estándares abiertos, con licencias libres, algunos promovidos por el W3C (WWW Consortium) aunque también podemos encontrar otras propuestas propietarias.

Ya que las páginas web que pretendemos crear son dinámicas, hay que buscar mecanismos dinámicos para la creación de las mismas. Estos mecanismos procurarán que con un mismo código se puedan generar diferentes respuestas, de acuerdo a los parámetros seleccionados por el usuario. De manera práctica hay tres tipos de sitios web dinámicos: los primeros, donde el dinamismo se encuentra en el lado del cliente, los segundos donde el dinamismo está centrado en el nivel del servidor y los terceros en el cual la lógica se divide entre el cliente y el servidor. Hoy por hoy, casi todas las aplicaciones web reales utilizan tecnologías tanto del lado del cliente como del lado del servidor. Utilizar unas u otras es una decisión de diseño que deberá resolverse en función a lo que resulte más adecuado para satisfacer las necesidades particulares de cada aplicación.

4.2.1 CREACIÓN DE CONTENIDOS DINÁMICOS EN EL LADO CLIENTE

Como ya hemos visto en los modelos de separación de la lógica del negocio, toda aplicación web se puede generar de tal forma que sea el servidor quien ejecute la aplicación y sea el cliente quien solo se encargue de que la respuesta dada por el servidor sea visualizada por el usuario a través de un software liviano como por ejemplo un navegador web. Esta es una estrategia sencilla desde el punto de vista del programador, ya que solo se preocupa porque el resultado tenga un formato HTML, que es además restringido. Sin embargo, esta opción puede resultar no muy atractiva para los usuarios. El lenguaje HTML no había sido diseñado con el fin de permitir la construcción de interfaces de usuario dinámicas, por lo que sus limitaciones dieron pié a la aparición de nuevas tecnologías que permitieran ejecutar ciertas funcionalidades desde la máquina del cliente, generalmente dentro del propio navegador. Estas tecnologías son denominadas *lenguajes de script del lado cliente*. Dichos lenguajes ayudan a mejorar la el tiempo de respuesta en los usuarios, ya que pequeñas acciones que antes debían ser enviadas al servidor para responder a los usuarios, ahora son resueltas en el cliente y además, conllevan una descarga de trabajo en el servidor, con lo que éste también puede mejorar sus tiempos de respuesta. Este cambio también redunda en la satisfacción de los usuarios quienes valoran positivamente tener interfaces más depuradas, rápidas y fáciles de manejar.

Aparte de los lenguajes de *script*, existe otro conjunto de tecnologías que sirven para crear y visualizar contenidos dinámicos en las páginas web, sin que estos contenidos sean necesariamente ejecutados por la página web en sí misma. Ejemplos de estas tecnologías son los controles Active X, los *applets* o la creación de *plugins* específicos para visualizar otro tipo de contenidos dinámicos.

LENGUAJES DE SCRIPT EN EL LADO CLIENTE

HTML dinámico (DHTML) es la palabra que encierra todo el conjunto de herramientas y tecnologías que permiten dar propiedades de interactividad a las páginas web a través de la combinación de HTML, un *script* de lenguaje cliente y elementos de ayuda de imagen como las hojas de estilo o CSS.

Un *lenguaje de script de cliente* pretende modelar un documento HTML a través de objetos de forma que es posible acceder a cada uno de ellos de manera independiente y así acceder y cambiar propiedades de los distintos elementos que conforman una página web. Los *scripts* que realizan estas modificaciones están dentro de la misma página web y son interpretados por el navegador quien sabe la manera de ejecutar las acciones dichas por el *script*. De manera general, es conveniente separar el código que es *script* del código que es HTML. Para ello, existe la etiqueta `<script>` de HTML.

En DHTML, cada etiqueta HTML se convierte en un objeto con sus propiedades y eventos asociados. Los *scripts* representan las acciones a realizar si un evento X se presenta en un elemento Y de una página web. De esta manera, el *script* es ejecutado por el navegador cuando se produce algún evento asociado a alguno de los elementos de la página web. La Figura 4.14, representa el actual modelo de objetos de DHTML (DOM).

Figura 4.14. *Modelo de objetos de documentos DHTM (DOM)*

Este modelo de objetos es el que utilizamos cuando escribimos en un lenguaje de *script* de cliente tales como Javascript o VBScript (*VisualBasic Script*). JavaScript, originalmente llamado LiveScript, fue desarrollado por Netscape para sus productos relacionados con la Web. De hecho, JavaScript funciona tanto en navegadores web como en servidores HTTP tal como puede hacerlo la tecnología ASP de Microsoft. La sintaxis de JavaScript es orientada a objetos y muy similar en su sintaxis a los lenguajes Java y C, por lo que es un lenguaje relativamente fácil de aprender y de manejar para cualquier programador con algunos conocimientos básicos en dichos lenguajes. Como ejemplo mostramos el tradicional Hola mundo, siguiendo el modelo DOM expuesto:

```
<HTML>
  <head>
    <title>El primer script</title>
    <script type="text/Javascript">
      document.alert("Hola Mundo");
    </script>
  </head>
  <body>
  <p>Esta página contiene mi primer script</p>
  </body>
</HTML>
```

Inicialmente, solo el navegador de Netscape (ahora Mozilla Firefox) soportaba JavaScript, si bien Microsoft no tardó en incorporar una versión ligeramente modificada de JavaScript denominada JScript. JavaScript es independiente de Java y actualmente es un estándar abierto.

JavaScript es uno de los lenguajes más documentados en Internet donde es posible encontrar toda una serie de ejemplos que funcionan y que cubren aspectos muy demandados como menús desplegables, relojes, contadores o calendarios.

VBScript (*Visual Basic Script Edition*) fue la respuesta de Microsoft a JavaScript. VBScript es un lenguaje interpretado originalmente diseñado para el entorno cliente cuya sintaxis tiene su origen en el lenguaje de programación Visual Basic. La lucha inicial entre estos dos lenguajes ha sido ganada en el lado cliente por JavaScript. Sin embargo,

el gran avance de VBScript que se ha dado en los lenguajes de *script* del lado de servidor con el surgimiento de la tecnología de servidor ASP (*Active Server Pages*), ya que VBScript es parte fundamental en la ejecución de este entorno de desarrollo. Veamos la manera de realizar el ejemplo anterior del "hola mundo" en lenguaje VBScript:

```
<HTML>
  <head>
    <title>El primer script</title>
    <script type="text/vbscript">
      document.write("Hola Mundo en VBScript")
    </script>
  </head>
  <body>
    <p>Esta página contiene mi primer script</p>
  </body>
</HTML>
```

APPLETS

Los *applets* son pequeñas aplicaciones que realizan una tarea específica y que se ejecutan en un entorno determinado como extensión de un programa. En entornos web, se denomina *applet* a aplicaciones escritas en Java, que están incrustadas en una página web y se ejecutan en el navegador web. Un *applet* de Java es una aplicación completa compilada usando la máquina virtual de Java (*Java Virtual Machine*, JVM) el cual es un programa que sirve de intérprete del código intermedio que genera el compilador de Java. Así, los *applets* son incrustados en una página web y pueden ejecutarse en cualquier navegador que tenga instalada una máquina virtual Java.

Cuando se utiliza un *applet*, se descarga del servidor web el código intermedio del *applet* correspondiente a la máquina virtual Java; esto es lo que se denomina los *bytecodes* del *applet*. Al no tener que difundir el código fuente de la aplicación y disponer de una plataforma completa para el desarrollo de aplicaciones, se suele preferir los applets para las partes más complejas de una aplicación web mientras que se limita el uso de JavaScript a pequeñas mejoras estéticas de los formularios HTML. Además, los applets destacan por su seguridad ya que cada aplicación se ejecuta en un espacio independiente (*sandbox*) desde el cual el *applet* no puede acceder al hardware de la máquina del cliente a menos que este lo autorice de manera explícita.

La utilización de *applets* se ha visto limitada por algunos problemas de rendimiento ya que la ejecución de un *applet* es, en principio, más lenta que la de un control ActiveX equivalente, lo que les ha restado atractivo. Otro problema para el desarrollo de un *applet* es la existencia de distintos navegadores y de sus sutiles diferencias, las cuales tienen que ser tomadas en cuenta a la hora de desarrollar el *applet*. Estas diferencias entre los navegadores pueden hacer que lo que funcione bien en un navegador no tenga el mismo aspecto en otro. Sin embargo, los *applets* se siguen usando a la hora de desarrollar aplicaciones de visualización ya que permiten emplear un conjunto de elementos gráficos mucho más amplio que el de una página HTML.

CONTROLES ACTIVE X

Otra de las tecnologías que se puede utilizar para implementar parte de las aplicaciones web en el lado del cliente está basada en el uso de controles ActiveX. Los controles ActiveX fueron presentados en 1996 por Microsoft y están construidos sobre los modelos COM (*Component Object Model*) y OLE (*Object Linking and Embedding*), las cuales son las dos tecnologías para el desarrollo de objetos distribuidos desarrolladas por Microsoft, aunque dichas tecnologías no están atadas necesariamente a Microsoft. Los controles ActiveX son mini programas que han sido compilados de manera previa, lo que permite una ejecución más eficiente.

De manera oficial, los controles ActiveX funcionan solamente bajo el navegador Internet Explorer y bajo el sistema operativo Windows. Dado su fuerte acoplamiento con los productos de Microsoft, su utilización se suele limitar a las aplicaciones web para intranets. Las intranets constituyen un entorno más controlado que Internet al estar bajo el control de una única organización, por lo que los programadores pueden permitirse crear una aplicación web que no sea totalmente portable. En cierta medida, podemos decir que los controles ActiveX fueron la primera respuesta de Microsoft a los applets de Java. La segunda, mucho más ambiciosa, fue la creación de la plataforma .Net.

Los controles ActiveX puede ser desarrollados en cualquier lenguaje que soporte el desarrollo de componentes COM incluyendo lenguajes como C++, Visual Basic o lenguajes de la plataforma .Net como C# o VB.Net.

PLUGINS ESPECÍFICOS

Los navegadores web pueden extenderse con *plugins*. Los *plugins* son componentes que permiten alterar, mejorar o modificar la ejecución de la aplicación en la que se instala. Ejemplos de *plugins*, son los utilizados por los navegadores web para visualizar desde el navegador documentos en formato PDF, ejecutar presentaciones Flash, escuchar sonidos RealAudio, escuchar *podcast* a través de Windows Media o ejecutar applets escritos para la máquina virtual Java. Los *plugins*, usualmente, son gratuitos. Están disponibles a través de Internet, desde donde podemos descargar la versión adecuada de acuerdo al sistema operativo e instalarlo una única vez, tras lo cual quedará almacenado localmente y podremos utilizarlo cuantas veces necesitemos.

NPAPI (*Netscape Plugin Application Programming Interface*) (*https://wiki.mozilla.org/NPAPI*) es una arquitectura multiplataforma para el desarrollo de *plugins* que utilizan varios navegadores web. Fue desarrollado por primera vez para la familia de navegadores de Netscape, empezando por Netscape Navigator 2.0, y fue extendiéndose posteriormente a todos los navegadores, incluyendo los más utilizados hoy en día. Su éxito puede atribuirse en parte a su simplicidad. Un *plugin* declara que se encargará de tratar un tipo de contenido específico (por ejemplo, "audio/MP3"). Así, cuando el navegador encuentra el tipo de contenido cargará el *plugin* correspondiente para que este sea ejecutado. Por lo tanto, el *plugin* es el responsable de que los datos se muestren de forma conveniente, ya sea de manera visual, auditiva o de otro tipo. El *plugin* se ejecuta dentro de la página, a diferencia de los navegadores más antiguos que tuvieron que iniciar una aplicación externa para gestionar los tipos de contenido desconocido. El API exige que cada *plugin* exponga un número relativamente pequeño de funciones. En total pueden existir unas 15 funciones para la inicialización, creación, destrucción y posicionamiento de los *plugins*. Además, NPAPI admite secuencias de comandos *scripts*, acceso a impresión, acceso a la funcionalidad de pantalla completa, ventanas o contenido de vídeo.

Debido a algunos problemas en el modelo de seguridad de NPAPI, Google y en especial su división Chrome, centrada en el desarrollo de su navegador, ha lanzado un proyecto para crear una nueva versión de NPAPI denominada **PPAPI** (*Pepper Plugin API*) (*http://code.google.com/p/ppapi/*). Entre los objetivos de este nuevo proyecto están: 1) permitir el aislamiento del código que se ejecuta como *plugin* para que el mismo no impacte en el modelo de seguridad general del navegador que la implementa y así que una aplicación no ponga en riesgo a otras, ya que ven un navegador simple y sin tener idea del contexto en el que se está ejecutando, 2) una mejor integración de los *plugins* con el modelo de render estándar del navegador, que permita una mejor comunicación entre el HTML que el navegador está renderizando con la salida de un *plugin* y 3) generar un estándar nuevo que permita a la API estar disponible y ser utilizable por la nueva generación de dispositivos de hardware. Las versiones previas de PPAPI están disponibles actualmente para el navegador Google Chrome.

4.2.2 CREACIÓN DE CONTENIDOS DINÁMICOS EN EL LADO SERVIDOR

Como ya hemos expuesto, en la mayoría de las arquitecturas de desarrollo de aplicaciones web, es el servidor quien se encarga de ejecutar todas las funciones de la lógica del negocio. La ejecución de estas funciones implica la construcción de elementos de software que permitan dar respuesta satisfactoria a los usuarios. Por lo tanto, lo que se le exige a la mayoría de las tecnologías usadas del lado del servidor es que devuelvan a los clientes respuestas en

forma de código HTML. De esta manera no es necesario tener ningún *plugin* instalado en la máquina del cliente y, si queremos dar algo de dinamismo a las páginas bastará con utilizar algún recurso de HTML dinámico disponible en los navegadores web actuales.

Una parte importante de las aplicaciones del lado del servidor es poder recibir la información del cliente. En general, cualquier petición de un cliente se puede traducir en una petición de una página web. Sin embargo, muchas veces esa petición de página web puede ir acompañada de un conjunto de datos, como es el caso de una petición de página que responde a un formulario web. Los datos recogidos por un formulario HTML pueden enviarse codificados en la propia URL o enviarse en una cabecera del mensaje HTTP que se envía automáticamente cuando el usuario pulsa un botón del formulario. Otras veces la información traída del cliente será utilizada para ser almacenada en pequeñas cadenas de texto en el navegador web del cliente y permitirá realizar tareas como el mantenimiento de sesiones de usuario (las *cookies*, que son explicadas en el capítulo 5).

Una vez tomados los datos dados por el cliente, la aplicación web del lado del servidor deberá procesarlos de acuerdo al requerimiento del usuario. El cumplimiento de dicho requerimiento puede involucrar el acceso a bases de datos, el uso de archivos, el envío de mensajes a otras máquinas utilizando algún protocolo preestablecido o, incluso, el acceso a otros servidores o servicios web disponibles en la Red. Como resultado de la ejecución de la aplicación web, se ha de generar dinámicamente una respuesta para enviársela al cliente. Dicha respuesta suele ser un documento en formato HTML, si bien también podemos crear aplicaciones web que generen imágenes y documentos en cualquier otro formato en función de las necesidades del usuario.

Entre las ventajas más destacables de las aplicaciones web desarrolladas de esta forma destacan (Berzal et al., 2005):

- **Accesibilidad**: desde cualquier punto de Internet.
- **Fácil mantenimiento**: no hay que distribuir el código de las aplicaciones ni sus actualizaciones.
- **Seguridad**: el código no puede manipularlo el usuario, al que solo le llega la respuesta.
- **Escalabilidad**: si utilizamos arquitecturas multicapa y *clusters* de PC nos puede resultar relativamente sencillo ampliar en número de clientes a los que puede dar servicio la aplicación.

Al igual que en el lado cliente, en el lado del servidor se han dado dos tendencias: la primera que tiende a facilitar la creación de las páginas web a través de la utilización de lenguajes de *script* para servidor. En este caso, el código está dentro de la página web que se ejecutará en el servidor y se diferencia del código HTML por un grupo de tags que diferencian el código HTML del código *script* de servidor. El segundo tipo de tecnologías tiende a crear pequeños elementos software los cuales son ejecutados y dan como resultado una página HTML. A continuación presentamos las tecnologías más populares dentro de esta área ya que este es un campo en la que continuamente aparecen nuevas propuestas.

CGI, COMMON GATEWAY INTERFACE

Una de las primeras maneras que se creó para la creación de páginas web dinámicas es el uso de CGI, la *Interfaz de Pasarela Común* (*Common Gateway Interface*). En general, la tecnología CGI especifica un estándar para transferir datos entre el cliente y un programa residente en el servidor. Un CGI es una aplicación independiente que permite la transmisión de información del cliente hacia un compilador instalado en el servidor web, donde el CGI se ejecuta y envía la respuesta al cliente. Los CGIs permiten al servidor web interactuar con aplicaciones para realizar consultas a bases de datos, búsquedas de documentos, generadores de email automático, comercio electrónico y procesamiento de solicitudes, entre otras. Los CGIS suelen estar instalados en un suBDirectorio específico dentro del servidor web, por ejemplo el directorio `/cgi-bin`.

El funcionamiento de un CGI se puede resumir en los siguientes pasos:

1 El cliente acumula datos, por ejemplo en un formulario HTML y los prepara para su uso por la aplicación CGI, enviándolos al servidor a través de la URL usando el protocolo HTTP.

2 El servidor lee el URL que acompaña a la petición, determina cuál es la aplicación CGI que debe ejecutar y la activa.

3 El servidor pasa la información de la URL a la aplicación CGI.

4 El programa CGI procesa los datos de acuerdo a su funcionalidad y prepara la respuesta. Generalmente, la respuesta se formatea como un documento HTML, de acuerdo al protocolo HTTP.

5 El programa CGI pasa la respuesta al servidor quien la redirige hacia el cliente.

Los programas que maneja el CGI pueden estar compilados en diferentes lenguajes de programación. El más popular para el desarrollo de contenidos web es el lenguaje Perl de distribución gratuita, aunque también es posible escribir CGIs en C, C++, Java o Visual Basic.

Como ejemplo de Perl, el siguiente código muestra un pequeño programa que genera automáticamente la cabecera HTTP asociada a la respuesta y un documento HTML que muestra la dirección IP de la máquina que ha realizado la solicitud HTTP en primera instancia:

```
#!/usr/bin/perl
use CGI qw(:standard);
print header,
start_HTML,
h1("CGI de ejemplo"),
"Su dirección IP es: ", remote_host(),
end_HTML;
```

Si guardamos este código con el nombre `prueba.cgi` y lo ejecutamos de manera local, obtenemos lo siguiente:

```
...
Content-Type: text/HTML
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
  <HEAD><TITLE>Untitled Document</TITLE></HEAD>
  <BODY>
    <H1>CGI de ejemplo</H1>
    Su dirección IP es: localhost
  </BODY>
</HTML>
```

Para ejecutar la aplicación web, el usuario no tendría más que escribir la URL correspondiente en la barra de direcciones de su navegador:

```
http://localhost/cgi-bin/prueba.cgi
```

En este caso, el servidor web es el servidor `localhost`.

Dado que lanzar un proceso nuevo cada vez que recibimos una solicitud del usuario puede resultar bastante costoso para el rendimiento del servidor web varias empresas han desarrollado lo que se denominan *extensiones de servidor*. Las extensiones de servidor son API que extienden la funcionalidad básica del servidor web para manejar tareas dinámicas que de otra manera serían asignadas a aplicaciones CGI. En vez de usar CGI, las extensiones implementan librerías (DLL en el caso de Windows) las cuales se enlazan dinámicamente con el servidor web y se ejecutan en su espacio de direcciones. Ejemplos de extensiones son ISAPI (*Internet Server API*) de Microsoft y presente en Internet Information Server o los módulos NSAPI (*Netscape Server API*) del servidor HTTP de Netscape. Desafortunadamente, las extensiones son difíciles de desarrollar y mantener, y pueden hacer caer el servidor, por lo que los desarrolladores prefieren otro tipo de alternativas como las presentadas en los siguientes puntos.

SERVLETS

El término *servlet*, por analogía con el término *applet*, hace referencia a un programa escrito en Java que se ejecuta en el servidor y no en el navegador del cliente como los *applets*. Al igual que las tecnologías anteriores, el objetivo de un *servlet* es extender el comportamiento del servidor web (clases cargadas dinámicamente). Cada *servlet* se maneja a través de un proceso ligero o hilo dentro del servidor web por lo que el nivel de interacción es mayor que el conseguido por los CGI.

Un *servlet* tiene preestablecido dos interfaces una para la petición y otra para la respuesta, así los datos siempre son recibido y devueltos de la misma forma lo que facilita la creación de los mismos. Los *servlets* puede usar todo el API de Java (excepto AWT), además de comunicarse con otros componentes como RMI, CORBA, Java Beans, bases de datos, etc. Cada *servlet* queda en la memoria del servidor web como una única instancia, la cual se cargada la primera vez que llamamos al *servlet*. Si un *servlet* ya está inicializado, cada petición genera un hilo, lo cual reduce el uso de memoria en el servidor y mejora los tiempos de respuesta del cliente.

LENGUAJES DE SCRIPT DEL LADO SERVIDOR

Los lenguajes de *script* (*scripting languages*) del lado del servidor fueron creados para crear páginas web dinámicas introduciendo código que será interpretado en el interior de las páginas HTML. En otras palabras, en vez de crear programas que incluyan en su interior el código necesario para generar el documento HTML, creamos documentos HTML que incluyen el código de la aplicación en su interior. De esta manera, cuando una página es solicitada, el servidor ejecuta un único proceso (la página web) y además, se logra la independencia entre los diseñadores de páginas web y los programadores de la funcionalidad que tanto se ha deseado.

El desarrollo de aplicaciones web utilizando lenguajes de *script* consiste, en intercalar fragmentos de código dentro de los documentos HTML que sirven para crear las interfaces de usuario de las aplicaciones web. Los fragmentos de HTML proporcionan la parte estática de lo que ve el usuario mientras que los fragmentos de código generarán la parte dinámica.

Una de las características más llamativas de estos lenguajes es que muchos de ellos permiten que un bloque de instrucciones empiece en un *scriptlet*, o bloque de código *script*, a través del delimitador correspondiente (<%, <?) y terminar en otro *scriptlet* (%>, ?>), pero tenemos que asegurarnos de que todos los *scriptlets* se abran y se cierren correctamente. El siguiente código muestra un ejemplo de esta característica, para este caso, el delimitador de código es `<? Código ?>`. Además, las instrucciones `if`, de ejemplo, se encuentran escrita en pseudocódigo.

```
<HTML>
  <head>
    <meta charset="utf-8" />
    <title>Ejemplo de Lenguaje de Scripting en el Servidor
    </title>
  </head>
```

```
  <body>
    <?
      //Aquí empieza el código de script.
      Si (color = rojo)
      //Vamos a volver al HTML
    ?>
    <b> Color elegido es Rojo
    <?
      //Volvemos al script y continúa el Si
      Sino
    ?>
    <b> Color por defecto es negro
  </body>
</HTML>
```

Los lenguajes de *script* pueden ser clasificados en lenguajes interpretados como, por ejemplo, PHP (*PHP Hypertext Pre-processor* inicialmente, *Personal Home Page)*, o ASP (*Active Server Pages*) o lenguajes precompilados como, por ejemplo, JSP o ASP.Net.

PHP, Hypertext Pre-processor

PHP (*http://www.php.net/*) es un lenguaje de *script* interpretado en el lado del servidor utilizado para la generación de páginas web dinámicas, embebidas en páginas HTML y ejecutadas en el servidor. Creado en 1994 y de distribución libre, su implementación está dirigida por *The PHP Group* y se ha convertido en un estándar de facto para la creación de páginas web dinámicas. De hecho, actualmente es el lenguaje de creación de páginas web dinámicas con más auge. PHP no necesita ser compilado para ejecutarse. Para su correcto funcionamiento, es necesario que el servidor web (Apache o IIS) tenga instaladas las librerías PHP.

Uno de los aspectos llamativos para los desarrolladores es el parecido sintáctico de PHP con lenguajes como C, Java o Perl. Esto hace que los desarrolladores puedan reutilizar sus conocimientos en estos lenguajes a la hora de trabajar con PHP. Otra de las características destacables es su capacidad de conexión con la mayoría de servicios de bases de datos como MySQL, PostgreSQL, Oracle, SQL Server, entre otros.

Los archivos PHP cuentan con la extensión (`*.php`). A nivel de sintaxis, el código escrito en PHP siempre está entre los delimitadores `<?php` y `?>`. Un ejemplo es el siguiente:

```
<?php
  $mensaje = "Hola mundo ";
  echo $mensaje;
?>
```

Otra forma de insertar código PHP, es eliminando la palabra `php` del inicio de los *scripts*. El siguiente código muestra un ejemplo de este tipo y además el código PHP incrustado en el código HTML de la página:

```
<HTML>
  <head>
    <meta charset="utf-8" />
    <title>Ejemplo en PHP</title>
  </head>
  <body>
  <?
```

```
    $mensaje = "Hola mundo";
    echo $mensaje;
  ?>
  </body>
</HTML>
```

Sin embargo, para que esta opción funcione correctamente debemos comprobar si tenemos activada la opción de `short tags` en el archivo de configuración `php.ini`. El archivo `php.ini` contiene casi todos los detalles de configuración de PHP, además es el archivo desde donde se activan las diferentes librerías que podemos usar desde nuestras páginas PHP.

ASP, Active Server Pages

ASP es la tecnología creada por Microsoft para el desarrollar aplicaciones web dinámicas. Esta tecnología está ligada al servidor de aplicaciones Internet Information Server (IIS) perteneciente también a Microsoft.

Una página ASP no necesita ser compilada para ejecutarse y los archivos tienen la extensión `*.asp`. Existen varios lenguajes interpretados que se pueden utilizar para crear páginas ASP. El más utilizado es una variante de Visual Basic conocida como VBScript (*Visual Basic Script*) y que corresponde al visto en el apartado 4.2.1.1 de lenguajes de *script* de clientes. Las páginas ASP pueden programarse también en Perl o Jscript (la versión Microsoft de JavaScript).

A nivel de sintaxis, el código ASP se escribe entre los símbolos `<%` y `%>`. Un pequeño ejemplo de código en ASP es el siguiente:

```
<HTML>
  <head>
    <meta charset="utf-8" />
    <title>Ejemplo en ASP</title>
  </head>
  <body>
    <%
      Response.Write ("Hola Mundo")
    %>
  </body>
</HTML>
```

Sin embargo, el desarrollo de páginas ASP presentan varios inconvenientes. Por un lado, el código incluido en las páginas ASP resulta excesivamente extenso, poco legible y difícil de mantener al estar mezclado con el HTML de la interfaz gráfica ya que en ASP, el fragmento de código que emite una salida debe colocarse en el sitio donde queremos que la salida aparezca, lo que hace difícil la separación entre los detalles de interfaz y la lógica de la aplicación. Por otro lado, configurar una aplicación ASP grande puede ser una labor realmente compleja y presenta también problemas de eficiencia en cuanto al servidor web. Por ello, a partir de 2002, el desarrollo de páginas ASP con lenguaje interpretado se quedo estancado. Microsoft llevo a ASP hacia la plataforma .Net y creo ASP.Net, que aunque tiene el historial de su predecesor, es un lenguaje totalmente diferente.

JSP, Java Server Pages

JSP (*http://Java.sun.com/products/jsp/*) funciona de forma análoga a las páginas ASP o PHP, si bien utiliza el lenguaje de programación Java. Podemos tomar a JSP como una extensión de los *servlets* de Java orientados a la implementación de aplicaciones web de manera más fácil a través de la inserción de código dentro de las páginas. Por este motivo, se considera que las páginas JSP son más sencillas que los *servlets* o los CGI.

JSP permite generar documentos HTML o XML de forma dinámica combinando plantillas estáticas con el contenido dinámico que se obtiene como resultado de ejecutar fragmentos de código en Java. En JSP la generación del contenido dinámico se puede encargar a componentes instalados en el servidor, externos al código JSP, denominados *JavaBeans*. Para que los JavaBeans puedan ser ejecutados, se requiere tener instalado un servidor de aplicaciones, como Tomcat o cualquiera de los muchos servidores de aplicaciones existentes que se ajustan al estándar J2EE (*Java 2 Enterprise Edition*).

Para insertar un *scriptlet*, es decir un grupo de instrucciones Java, dentro de una página HTML, este código debe encontrarse entre los delimitadores o *tags* `<% conjunto de instrucciones %>`. En JSP las instrucciones simples deberán terminar con punto y coma (`;`). Veamos el programa "Hola Mundo" en JSP:

```
<HTML>
  <head>
    <meta charset="utf-8" />
    <title>Ejemplo en JSP</title>
  </head>
  <body>
    <%
      out.println("Hola mundo");
    %>
  </body>
</HTML>
```

JSP también permite otro conjunto de delimitadores de acuerdo al tipo de instrucción que queramos escribir. Algunos de esos delimitadores son:

- `<%- comentario -%>`: para insertar comentarios.
- `<%! Variables, métodos, etc %>`: para declaración de variables.
- `<%= instrucción %>`: para insertar una y solo una instrucción de Java. Recordar además que cuando se use `<%= una sola instrucción %>`, la instrucción no debe terminar con punto y coma.
- `<%@ include file="url" %>`: para incluir archivos. La URL debe ser relativa a la página web donde se ejecute.
- `<jsp:forward page="URL relativo"/>` : redirige o enlaza a la página web dada en el parámetro `page`.

En JSP, existen un conjunto de variables predefinidas tales como `request`, response, out, session, application, config y pageContext, las cuales describen algunos de los aspectos más importantes en el trabajo con aplicaciones web. Durante este libro iremos explicando cada una de ellas.

ASP.Net

Debido a los inconvenientes mencionados con el lenguaje ASP, Microsoft decidió dar un giro a su oferta de lenguajes para el desarrollo de sitios web dinámicos y así inserto ASP en su plataforma .Net La plataforma .Net pretende crear un marco de trabajo para el desarrollo rápido de aplicaciones. ASP.Net, es la nueva generación de ASP pero hay que aclarar que no es completamente compatible con ASP. Los archivos creados en ASP.Net tienen la extensión (`*.aspx`). A diferencia de ASP, si queremos crear páginas en ASP.Net podemos utilizar cualquiera de los lenguajes admitidos en la plataforma .Net.

Al igual que en el ASP clásico, la sintaxis ASP.Net permite utilizar los delimitadores `<% %>` para que podamos insertar el código que modificará dinámicamente la página web. Las páginas son compiladas y ejecutadas dinámicamente en el

servidor con el fin de generar una traducción para el navegador o dispositivo del cliente. Cuando un cliente hace una petición de un archivo `.aspx`, la rutina ASP.Net analiza y compila el archivo a una clase del entorno de trabajo .Net. Al igual que los *servlets*, un archivo `.aspx` se compila únicamente la primera vez que es accedido, y es la instancia compilada la que se reutiliza en las sucesivas peticiones.

Sin embargo, uno de los factores diferenciadores de ASP.Net con su predecesor es el uso de controles de servidor. Los *controles de servidor*, son objetos programables en el lado del servidor que suelen representar a un elemento de la interfaz gráfica de la página, como por ejemplo una caja de texto o una imagen. Los controles de servidor participan en la ejecución de la página y producen sus propias etiquetas para el navegador cliente. La principal ventaja de los controles de servidor es que permiten a los desarrolladores obtener comportamientos complejos a partir de componentes sencillos, reduciendo así la cantidad de código necesaria para la creación de una página web dinámica.

CONTROLES DE SERVIDOR

Como ya hemos dicho, los *controles de servidor* son propios de las aplicaciones ASP.Net y concretamente pueden ser utilizados en páginas web ASP.Net. Muchos de los controles de servidor son similares a controles HTML como los cuadros de texto o botones. Pero hay controles de servidor muchos más complejos que brindan numerosas funcionalidades: *grill*, calendarios, vista de árbol, menús o administrar conexiones de datos.

Los controles de servidor muestran propiedades que pueden ajustarse bien de forma declarativa, es decir, configurando la propiedad a través de la etiqueta en la que se inserta el control o bien de forma programada, es decir, a través de código explícito que personaliza la acción. Los controles de servidor también tienen eventos que los desarrolladores pueden personalizar para realizar acciones específicas durante la ejecución de la página, o en respuesta a una acción del lado del cliente que envíe la página al servidor, estos eventos son denominados *postback*. Los controles de servidor también simplifican el problema de mantener el estado durante diferentes idas y venidas del servidor, manteniendo sus valores de forma automática en sucesivos *postbacks*.

La declaración de un control de servidor, se debe realizar en un archivo `.aspx` a través de etiquetas propias o etiquetas HTML intrínsecas que contengan el atributo `runat="server"` y por medio de su ID nos permite manipularlos y programarlos en tiempo de ejecución. Los controles de servidor web se declaran con una etiqueta XML que hace referencia al espacio de nombres `asp`. El inicio y fin del *tag* está determinado por la sentencia `<asp />`.

El siguiente ejemplo, extraído de la página oficial de ASP, utiliza cuatro controles de servidor: `<form runat=server>`, <asp:textbox runat=server>, <asp:dropdownlist runat=server>, y `<asp:button runat=server>`. En tiempo de ejecución estos controles de servidor generan de forma automática el contenido HTML.

```
<HTML>
  <body>
   <form action="intro4_vb.aspx" method="post" runat=server>
      <h3> Name: <asp:textbox id="Name" runat="server"/>
      Category:
<asp:dropdownlist id="Category" runat=server>
        <asp:listitem>psychology</asp:listitem>
        <asp:listitem >business</asp:listitem>
        <asp:listitem >popular_comp</asp:listitem>
      </asp:dropdownlist>
      </h3>
     <asp:button text="Lookup" runat="server"/>
   </form>
  </body>
</HTML>
```

Para personalizar las acciones que realiza un control de servidor, lo que debemos hacer es programar sus eventos, es decir, las acciones a las cuales el objeto puede reaccionar. Por ejemplo, si tenemos un control de tipo botón, este control contiene un evento denominado `OnClick`. El evento `OnClick` se produce al presionar un botón y genera que se ejecute el código que hemos introducido para ese evento.

```
<asp:Button ID="btnAceptar" runat="server" Text="Aceptar"
  onclick="btnAceptar_Click"/>
```

En este caso, `runat="server"` nos indica que se trata de un control de servidor y `onclick="btnAceptar_Click"` nos indica que la función `btnAcepta_Click` contiene el código que se va a ejecutar cuando se produzca el evento al hacer clic en el botón con ID `btnAceptar`.

Un excelente manual de ASP.Net en la Red es *http://www.es-asp.net/*.

Aunque PHP no tiene controles de servidor, podemos encontrar iniciativas que pueden lograr resultados similares. En el caso de PHP, nos encontramos con *motores de plantillas* como Smarty (*http://smarty.net/*) o Templatelite (*http://www.templatelite.com/*).

Smarty es un motor de plantillas de código abierto para PHP. El objetivo de Smarty es ayudar a los desarrolladores a realizar aplicaciones web de calidad separando el código (PHP) de la presentación (HTML/CSS). El lenguaje PHP es completamente abierto, y no fuerza al desarrollo en una arquitectura concreta, por el contrario, un mismo archivo puede contener un millón de líneas con código PHP y código HTML intercalados, lo cual puede generar desventajas y problemas. Algunas de las ventajas del uso de Smarty en un proyecto PHP son:

- ✓ Plantillas limpias fáciles de usar por los diseñadores.
- ✓ Escalabilidad.
- ✓ Mantenimiento más sencillo.
- ✓ Depuración óptima del código, al tener ficheros pequeños únicamente con código PHP.
- ✓ Posibilidad de introducir comentarios dentro de las plantillas que no se enviarán al servidor. Ejemplo: `{* comentario smarty *}` en lugar de `<!-- comentario HTML -->`.
- ✓ Funciones integradas que facilitan el tratamiento de variables. Por ejemplo: `{foreach}{/foreach}`, `{if}{else}{/if}`.
- ✓ Funciones asistentes para generación de código HTML. Por ejemplo: `{HTML_image file="banner.jpg"}` generará el código HTML `<img src="banner.jpg" alt="" width="700" height="55" />`.
- ✓ Posibilidad de expansión a través de *plugins*.

La Web sigue evolucionando y nuevas funcionalidades y nuevos recursos siguen apareciendo para los desarrolladores a medida en que los usuarios demandan nuevos productos. Hay que seguir atentos, porque esta historia continuará.

ACTIVIDADES 4.2

- La Web es una de las herramientas más dinámicas en la actualidad y entender su evolución y sus retos es algo importante para desarrollar aplicaciones en este entorno. Esta vez os recomendamos la lectura del artículo "*Qué es Web 2.0. Patrones del diseño y modelos del negocio para la siguiente generación del software*".

  http://sociedadinformacion.fundacion.telefonica.com/DYC/SHI/seccion=1188&idioma=es_ES&id=2009100116300061&activo=4.do?elem=2146) de Tim O'Reilly.

- Una de las alternativas para el desarrollo de webs en ASP.Net es su sección especializada ASP.Net MVC, que actualmente está en la versión 4 (*http://msdn.microsoft.com/es-es/library/gg416514(VS.98).aspx*). Os animamos a que exploréis en su página web cuáles son las funcionalidades ofrecidas por esta herramienta para este patrón arquitectónico.

RESUMEN DEL CAPÍTULO

La generación de páginas web dinámicas conjuga dos aspectos muy importantes: por un lado, decisiones arquitectónicas y, por otro, la elección de herramientas que posibiliten dicho dinamismo.

En cuanto a las decisiones arquitectónicas, la creación de páginas dinámicas pasa por la separación entre la lógica del negocio y la presentación de una aplicación web. Teniendo en cuenta que una aplicación web está por defecto dividida en varios niveles, es decir, distribuida en varios elementos hardware, lo que se propone para la separación lógica es utilizar una arquitectura de capas, la cual divide funcionalmente los elementos de software. La arquitectura de tres capas es la más difundida. En esta arquitectura encontramos tres capas: capa de presentación, capa de negocio y capa de persistencia. Producto de esta arquitectura se han desarrollado las arquitecturas multicapa. Incluso, en estos momentos se prueban con nuevos modelos, como la Arquitectura Orientada a Servicios (SOA).

Otro de los aspectos que apoya y facilita la creación de aplicaciones dinámicas es la utilización de patrones de software. Los patrones pueden ser utilizados en las labores de análisis, diseño y construcción del sitio web.

Por otra parte, la creación de interfaces web dinámicas está apoyada en un conjunto variado de herramientas y lenguajes de programación. Estas herramientas se pueden dividir en: herramientas en el cliente y herramientas en el servidor, de acuerdo al lugar en el que se ejecute el código. Las herramientas en el cliente buscan crear páginas más interactivas. Ejemplos de herramientas de cliente son DHTML, *applets*, controles ActiveX o *plugins* para elementos específicos, como películas Flash o *podcast* en Real Audio. En cuanto a herramientas para la creación dinámica de páginas en el servidor, podemos ver dos tendencias. La primera, a través de la creación de elementos software, que dan como resultado una página HTML tales como CGI o *servlets*. La segunda tendencia es el uso de lenguajes de *script* para el servidor tales como PHP, JSP o ASP (ASP.Net). Este último ha introducido una funcionalidad denominada *controles de servidor*, que se puede homologar a los motores de plantillas promovidos por plataformas como PHP.

TEST DE CONOCIMIENTOS

1 Sobre los mecanismos de separación de la lógica del negocio, señale la respuesta correcta:
- a) La separación por niveles es más conveniente que la separación por capas.
- b) Busca la independencia entre el diseño de la página y la lógica del negocio.
- c) No se han especificado mecanismos concretos de separación, lo que tenemos son propuestas no enfocadas a este problema.
- d) Se centra en el análisis, diseño e implementación del sistema desde el punto de vista del usuario.

2 ¿Cuál de las siguientes afirmaciones no es cierta hablando de arquitecturas multinivel?
- a) Están basadas en el vocablo inglés, *multitier*.
- b) La estructura mínima de una aplicación web está compuesta por dos niveles.
- c) En ellas se puede observar claramente la separación entre lógica del negocio y la presentación de la página.
- d) Representan niveles de separación físicos dentro del diseño de la arquitectura de una aplicación web.

3 Hablando acerca de los modelos de separación lógicos, podemos afirmar que:
- a) Todos los modelos de separación lógicos son patrones de software.
- b) Las arquitecturas en capas fomentan la creación de aplicaciones monolíticas fáciles de mantener y escalar.
- c) Hay una relación directa entre el número de capas y el número de niveles, ya que son lo mismo.
- d) Ninguna de las anteriores es correcta.

4 ¿Qué significan las siglas MVC?
- a) Modelado-Visualización-Comprobación.
- b) Modelo-Vista-Controlador.
- c) Modelar-Verificar-Controlar.
- d) *Modeling-Virtual-Component*.

5 ¿Cuáles son las capas mínimas presentes en una arquitectura de 3 capas?
- a) Negocio, persistencia y recursos.
- b) Negocio, seguridad y almacenamiento.
- c) Integración, persistencia y presentación.
- d) Persistencia, presentación y negocio.

6 Sobre la arquitectura de capas, ¿cuál de estas afirmaciones es cierta?
- a) La capa de lógica del negocio se centra en la recepción de las peticiones por parte de los usuarios y enseñarles de manera visual la lógica del negocio.
- b) En toda arquitectura de tres capas, la capa de persistencia puede intercambiar su lugar con la capa de la lógica de negocio.
- c) La capa de recursos es una forma de ampliación de las capas para conectarse con varios sistemas.
- d) En la capa de integración se encuentra invariablemente el navegador web.

7 ¿Qué significa SOA?
- a) Arquitecturas Orientadas a Servicios.
- b) Servicios Orientados a Aplicaciones.
- c) Sistemas Orientados a Aplicaciones.
- d) Aplicaciones Orientadas a Servicios.

8 Hablando de patrones de software, ¿cuál de las siguientes afirmaciones es incorrecta?

a) Los patrones representan soluciones a problemas comunes en el desarrollo de software.
b) Una de las características de los patrones es que pueden ser aplicados a más de un caso práctico.
c) MVC es un patrón de tipo arquitectónico.
d) Los patrones que se aplican a la Web no funcionan en otros entornos de programación.

9 Sobre los lenguajes para la creación de interfaces web dinámicas, ¿cuál de las siguientes afirmaciones no es cierta?

a) Existen lenguajes de *script* tanto para el lado cliente como para el lado servidor.
b) Los *applets* y los *servlets* tienen la misma filosofía pero en entornos diferentes.
c) Los lenguajes de *script* embebidos necesitan de una pequeña aplicación que genere el código ejecutable.
d) Los *plugins* permiten ampliar las funcionalidades del navegador web.

10 Acerca de las herramientas para la creación de interfaces desde el lado del servidor, ¿qué afirmación es correcta?

a) El modelo DCOM es seguido por todos los lenguajes.
b) Los controles de servidor están disponibles para todas las plataformas de desarrollo de lado de servidor.
c) No importando la herramienta que se utilice, el resultado que se genera está en formato HTML o XML.
d) CGI es una de las primeras iniciativas del lado servidor.

5 Desarrollo de aplicaciones web utilizando código embebido

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer la importancia y los métodos para el establecer y controlar el mantenimiento del estado de un usuario en una aplicación web.
- ✓ Aprender a manejar el estado de un usuario a través del uso de sesiones y de *cookies*.
- ✓ Conocer las listas de control de acceso (ACL) como mecanismos de seguridad para el control de usuarios, perfiles y roles.
- ✓ Introducir al alumno en el conocimiento de nuevas técnicas de autenticación universal como OpenID u OAuth.
- ✓ Comprender el objetivo del uso de un servicio directorio LDAP (*Lightweight Directory Access Protocol*) y cómo usarlo a través de una página web.
- ✓ Presentar algunos de los conceptos de pruebas y depuración de aplicaciones web basado en lenguajes embebidos.

5.1 MANTENIMIENTO DEL ESTADO EN APLICACIONES WEB

Uno de los principales requerimientos a la hora de gestionar la interoperabilidad entre los usuarios y un sistema *on line*, como por ejemplo una compra o transferencia bancaria, es el mantenimiento del estado o de datos. Debido a que la Web se basa en un modelo *no conectado* (del inglés *stateless*, "sin estado"), no hay una manera fácil de controlar los estados operativos tanto de la aplicación como del usuario. En este modelo estático, el salto de una página web a otra implica la pérdida de datos y no preserva los estados de las mismas. La única forma de garantizar la permanencia de dichos estados es a través de un mecanismo de sesiones que nos permita registrar estos estados haciéndolos consistentes y permitiendo que estos puedan ser gestionados (Boronczyk, Naramore, & Gerner, 2009).

El manejo de sesiones nos permite controlar que se ejecuten de manera correcta y segura los recursos o servicios ofrecidos por la aplicación web. En general, el objetivo de una sesión es guardar información específica de cada usuario mientras este navega por una aplicación web. De esta forma cada usuario que entra en un sitio web, abre una sesión que es diferente e independiente de la sesión de los otros usuarios. La sesión evita que el usuario, por alguna u otra razón, pase de una página a otra y pierda la información o tenga el impedimento de continuar realizando alguna operación *on line*. Si esto sucede, se pueden presentar problemas de inconsistencia como que el usuario pierda el control de lo que está haciendo. Por otro lado, el servidor que ofrece la aplicación, no tendría control de lo que ocurre y podría estar otorgando un espacio de trabajo a un usuario no autorizado o podría estar restringiendo el uso de dicho recurso al usuario.

Para poder establecer una sesión entre el servidor de una aplicación web y un cliente, podemos desarrollar un mecanismo de control a través del envío de sesiones mediante al menos dos modalidades: a través de URL o mediante la creación de *cookies*. Por medio de la URL se pasan las variables o los datos de un punto a otro mediante dos técnicas *GET* o *POST*. Si se realiza en forma de *GET*, los datos se anexan a la dirección que apunta al recurso o si se utiliza *POST*, los datos se anexan en el encabezado de la solicitud de envío que es dirigida al recurso apuntado. Cuando un cliente se identifica en el servidor, el servidor envía hacia el cliente un código único de identificación y gestión de sesión llamado *SID*. El SID es un número que es asignado desde el servidor de forma aleatoria y es utilizado para gestionar la sesión. El SID será almacenado en el cliente temporalmente hasta que éste abandona el recurso cerrando la sesión en el servidor. El recurso físico del cliente (ordenador) solo preserva este número de identificación. Mientras que en el lado del servidor, el número es almacenado en conjunto a otros datos relacionados con las variables de la sesión y otros datos operativos de gestión interna del servidor.

En el caso del uso de *cookies* (o galletas de información), en el cliente se almacena un pequeño archivo que guarda el número SID y otros parámetros relacionados con la expiración, las páginas o recursos dirigidos, etc. Estos últimos parámetros resultan ser opcionales excepto el número SID. A través de este pequeño archivo, el servidor puede identificar al cliente y puede de esta forma controlar la sesión entre ambas garantizando así la funcionalidad operativa. Sin embargo, el uso de *cookies* podría estar condicionado debido a que muchos navegadores web no soportan *cookies*, no las utilizan o no permiten su uso. Para evitar que este inconveniente condicione la funcionalidad del sistema, algunos lenguajes embebidos como PHP utilizan un sistema que fuerza a utilizar ambos sistemas (basados en URL o en *cookies*). De esta forma, se garantiza que en caso de no utilizar *cookies*, el sistema siga operando utilizando URL por defecto como última alternativa. Esta opción se configura en el archivo de configuración de PHP, `php.ini`.

5.1.1 CONTROL DE SESIONES EN PHP

El lenguaje PHP cuenta con todo un catálogo de funciones relacionadas con el manejo de sesiones y de variables de sesión entre páginas de un mismo sitio web (*http://php.net/manual/en/ref.session.php*). Recordemos que las sesiones sirven para guardar información acerca de la visita de un usuario específico, tales como nombre, apellidos, correo electrónico, etc., de tal manera que dicha información pueda ser recuperada en cualquier momento mientras la sesión esté vigente.

Existen dos funciones que permiten controlar el inicio o la destrucción de las sesiones denominadas `session_start()` y `session_destroy()`. Para la creación de una sesión o para mantener una sesión ya existente y creada en una página anterior debemos encabezar la página con la función `session_start()`. Finalmente, cuando deseemos destruir la sesión sea porque el recurso no está vigente o porque ya no tiene sentido guardar los datos, utilizaremos `session_destroy()` para concluir el proceso. A continuación mostramos un ejemplo acerca de la creación de sesiones, este ejemplo ha sido extraído de la página oficial de PHP (*http://es.php.net/*).

Si en el archivo de configuración `php.ini` se ha definido la variable `session.auto_start = 1`, se inicializa automáticamente la sesión en cada página que visita un usuario, sin que se tenga que llamar al inicio de la página a la función `session_start()`.

```
<?php
  // page1.php
  session_start();
  echo 'Bienvenido a la página #1';

  // El formato para inicializar una variable de sesión es:
  //$_SESSION['nombre_var_sesión'] = 'Valor de la variable';
  $_SESSION['favcolor'] = 'verde';
  $_SESSION['animal'] = 'gato';
  $_SESSION['time'] = time();

  // Esta sesión opera si la cookie es aceptada
  echo '<br /><a href="page2.php">página 2</a>';

  // O puede ser que la sesión ID sea necesaria
  echo '<br /><a href="page2.php?' . SID . '">página 2</a>';
?>
```

El código mostrado nos introduce en el uso de variables de sesión. Al llamar a la función `session_start()`, PHP verifica si existe un identificador de sesión (SID) ya creado. En caso de existir, este devuelve el ID correspondiente de lo contrario crea un nuevo SID. Las variables de sesión se diferencian de las variables normales en que estás residen en el servidor. La forma de acceder a ellas es a través de *array* (vector) asociativo `$_SESSION`, donde se accede a cada variable a partir del nombre de cada variable de la siguiente manera: `$_SESSION["nombre_de_variable_de_sesion"]`. El ejemplo anterior muestra la inicialización de las variables `favcolor`, `animal` y `time`. La inicialización se hace a través del signo igual (=). Para recuperar el valor de una variable de sesión, tenemos que leer el mismo *array* con el nombre de la variable de sesión respectiva.

```
<?php
  echo $_SESSION["nombre_de_variable_de_sesion"];
?>
```

Para saber si una variable de sesión ya está definida utilizamos la función `isset`:

```
<?php
  if (isset($_SESION["nombre_de_variable_de_sesión"])
?>
```

```
<?php
  // Inicializar la sesión.
  session_start();

  // Quita todas las variables de las sesiones ahora.
  $_SESSION = array();

  // Si desea matar la sesión, también borre la sesión de la
    cookie.
  // Nota: ¡Esto destruirá la sesión y no los datos de la
    sesión!
  if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
      $params["path"], $params["domain"],
      $params["secure"], $params["httponly"]
    );
  }
  // Finalmente, destruye la sesión.
  session_destroy();
?>
```

Para realizar las funciones de registro de una sesión y la eliminación de una sesión se utilizan las funciones de control `session_register()` y `session_unregister()`. La función `session_register()`, además de crear o recuperar la sesión actual de una página, permite registrar más de una variable a la vez, es decir, crear más de un registro de sesión a la vez. Así mismo, para destruir o quitar los registros se utiliza la función `session_unregister()` respectivamente.

```
<?php
  // El uso de la session_register() está fuera de uso
  $barney = "Es un enorme dinosaurio púrpura.";
  session_register("barney");
  $_SESSION["zim"] = "Un invasor de otro planeta.";
?>
```

Un ejemplo tradicional de control de sesiones puede ser el siguiente. El usuario completa los datos de autenticación en un formulario web y luego envía una solicitud al servidor para que este lo identifique. A su vez, la página carga antes de procesar dicha solicitud un proceso de inicio de sesión, llamado comúnmente como *Login*. Si el usuario no puede ser identificado, se rechaza la solicitud de identificación en conjunto a la sesión temporal creada al inicio de este proceso. Ahora bien, si el usuario es aceptado el proceso continúa preservando el espacio de sesión creada antes

de proceder con dicha identificación. A partir de este momento, el usuario se encuentra identificado y con una sesión iniciada exitosa. Esto le permite al usuario operar dentro del recurso o página web solicitada de forma satisfactoria.

Sin embargo, hay un posible error que puede aparecer durante todo este proceso. El problema podría darse durante el proceso de creación de la sesión. Si la sesión no puede ser creada, por más que el usuario se haya identificado correctamente, la sesión se encuentra en un estado crítico. Para evitar potenciales problemas, se procede a cerrar la sesión si es posible y quitar la identificación del usuario. Esto normalmente se lo llama quitar sesión o *Log Out*. Así mismo, si el usuario decide abandonar la sesión, el proceso de cierre de sesión utilizará el mismo mecanismo de control. La idea es quitar la identificación y destruir la sesión creada anteriormente.

Para permitir que la sesión se preserve todo el tiempo, es necesario cargar la función `session_start()` en cada una de las páginas que se requiera operar con variables de sesiones llamadas `$_SESSION['varEjemplo']`. Estas mismas, ayudan a preservar los valores o datos de la sesión. Estos valores podrían ser los datos del usuario u otro tipo de información, como por ejemplo, el seguimiento de una sesión, de un carrito de compras, una operación determinada, etc. Todos ellos deberán permanentemente cargar, comparar, actualizar, etc., sus movimientos sin olvidar cargar la función `session_start()` previamente. La carga de dicha función debe colocarse en la primera línea de código embebido en una página dinámica PHP o estática HTML combinada, etc. Es decir, será la primera línea de proceso que debe ejecutarse antes que todo el cuerpo entero del archivo fuente. De esta forma el sistema funcionará correctamente. Si la función `session_start()` no encabeza la página PHP, la operación de las sesiones no podrá ser controlada correctamente y puede que ni siquiera funcione la sesión.

Para eliminar la sesión, se requiere de una operación sencilla. Dado que las páginas son estáticas, al pasar a otra página el estado se pierde. No obstante, las variables podrían aún tener valores residuales. Para evitar este inconveniente, se recomienda utilizar como primera línea la función `session_destroy()` que permitirá destruir la sesión creada y luego agregar las variables para proceder a vaciarlas. Por lo tanto, destruye la sesión y luego vacía las variables. Esto garantizará que la sesión sea quitada de forma satisfactoria. Suele recomendarse para garantizar el cierre de sesión, redirigir la aplicación a otra página con las instrucciones adecuadas.

5.1.2 CONTROL DE COOKIES EN PHP

Para manipular envío de *cookies* desde el servidor encontramos una función llamada `setcookie()`. Esta función crea un pequeño archivo plano que contiene el número SID y puede contener otros parámetros adicionales tales como el tiempo de expiración de la sesión, enlaces relacionados, etc. La sintaxis de la función `setcookie()` es la siguiente:

```
setcookie(Nombre, Valor, Tiempo_Vida, Path, Dominio, Seguro)
```

El primer parámetro es el nombre de la *cookie* y, por tanto, suele ser utilizado para poder construir el archivo *cookie*. El resto de los parámetros son opcionales y se utilizan en casos donde se requiere un comportamiento específico para las sesiones entre el cliente y el servidor. Un ejemplo habitual de comportamiento es el control del tiempo de la sesión creada.

```
<?php
  $value = "algo de algún lugar";

  //Una cookie sin parámetros
  setcookie("TestCookie");

  // Una cookie que expira en 1 hora
  setcookie("TestCookie", $value, time()+3600);
?>
```

En este trozo de código podemos observar dos formas tradicionales de escribir *cookies* utilizando la función `setcookie()`. Nótese que el nombre de la *cookie*, que es el primer parámetro es el único parámetro obligatorio. El segundo parámetro es el valor de la *cookie*, mientras que el tercer parámetro representa el tiempo de vida de la *cookie*. Hablando de sesiones, este parámetro puede representar el ciclo máximo de la sesión. Si se ajusta una sesión de 24 horas, la *cookie* hará funcionar el control del acceso por tan solo 24 horas. Pasado el tiempo, la *cookie* pierde vigencia y el servidor descarta el proceso de negociación. Esta puede ser una medida de seguridad operativa y para regular el consumo de ancho de banda de la Red.

El cuarto y quinto parámetros de la *cookie* son el *path* y el dominio, es decir, la ruta o directorio y dominio desde la cual la *cookie* puede ser recuperada. Si se especifica como valor del *path* "/", cualquier *script* en el servidor podrá recuperar su valor. Si especificamos un directorio, por ejemplo "/app/", la *cookie* solo podrá ser recuperada por *scripts* en ese directorio (o desde sus subdirectorios). El quinto parámetro, el dominio, permite limitar el acceso a los subdominios que tengamos. Por ejemplo, si especificamos este parámetro con el valor *miapp.net*, la *cookie* podrá ser recuperada tanto desde *http://miapp.net* como desde *www.miapp.net* o *dev.miapp.net*. El último parámetro establece si la *cookie* puede ser enviada si existe una conexión segura (HTTPS) o para cualquier conexión (HTTP). Este parámetro se indica mediante un 1 (solo conexiones seguras =`true`) o un 0 (cualquier tipo de conexión = `false`).

El siguiente código muestra la creación de una *cookie* denominada `visitas`, cuyo primer valor es 1 y cuya duración es de 24hs (86400) a partir del momento en que se crea.

```
<?php
  setcookie("visitas", "1", time() + 86400, "", "",1);
?>
```

Hay que tener en cuenta que la *cookie* no estará disponible hasta que el usuario no recargue la página.

Las *cookies* son recuperadas automáticamente por el *script* cada vez que se carga una página, ya que estas son enviadas como cabeceras HTTP por parte del servidor. Desde la versión 4.1 de PHP podemos recuperar las *cookies* a través de la variable `$_COOKIE`, el cual es un *array* accesible desde cualquier parte del *script* PHP. Para recuperar la *cookie* hay que indicar el nombre de la *cookie* que deseemos recuperar de esta forma: `$_COOKIE["nombre de la cookie"]`. Por ejemplo:

```
<?php
  //Mostramos el valor de la cookie llamada visitas
  echo $_COOKIE['visitas'];
?>
```

Para borrar una *cookie*, suele bastar con inicializar otra *cookie* con el mismo nombre pero sin ningún otro parámetro, por ejemplo:

```
<?php
  //Borra la cookie visitas
  setcookie ("visitas");
?>
```

Pero este sistema a veces falla (por ejemplo algunos navegadores no borran la *cookie* si no coincide el *path*). Por ello, la forma más segura de borrar una *cookie* es colocar otra nueva, con el mismo nombre, sin valor, mismo *path* y nombre de dominio (si se especificaron) y con un tiempo de vida negativo, por ejemplo `time() - 3600`.

Finalmente si deseamos acceder inmediatamente a la *cookie*, podemos recargar la página antes de que esta sea enviada al usuario, así:

```
<?php
```

```
    // Creamos una variable de control ($cookie_set) para
      verificar si la cookie ya ha sido creada o no.
    if(!isset($COOKIE_SET)){
      //Sino la variable no existe creamos la cookie
      setcookie("saludo", "hola");

      // Creamos la variable COOKIE_SET y le damos el valor 1
      header("Location: $PHP_SELF?COOKIE_SET=1");
      exit;
    }
?>
```

CONSIDERACIONES ESPECÍFICAS DE LAS COOKIES

El uso de las *cookies* puede presentar limitaciones. De manera general, se considera que una *cookie* no debería superar los 4 Kb. Sin embargo, una *cookie* de más de 2Kb puede acarrear inconvenientes también. Además de la limitación en tamaño, hay que contar con la cantidad de *cookies* posibles en el disco duro del cliente el cual no debería superar los 20 archivos. Esto se debe a que el propio sistema del cliente no puede memorizar tantos enlaces a la vez. Se trata de una limitación implementada por razones de seguridad funcional. Si se utiliza algún tipo de cifrado para un valor y lo almacena dentro de la *cookie*, también hay que tener mucho cuidado. La cantidad de caracteres que genera un cifrador (algoritmo de encriptación o similar) puede ser muy grande y superar la cuota de tamaño dentro de la *cookie*, lo cual inevitablemente, truncará la información. Por lo tanto, es recomendable evitar inflar demasiado las *cookies*.

Otro problema es el código o juego de caracteres. El uso de un juego apropiado para un determinado país puede ser perjudicial para otro país. Es por ello que es recomendable utilizar un juego de caracteres estandarizado, por ejemplo, la norma ANSI '94. Esta norma puede sacrificar ciertos tipos de caracteres, pero seguramente no fallará en las máquinas de los clientes.

Otro gran inconveniente es el bloqueo de las *cookies* por parte de los navegadores. El bloqueo se suele utilizar como medida preventiva o de seguridad. Este hecho implica varios inconvenientes. Obligar al usuario a habilitar las *cookies* para que nuestras *cookies* se puedan almacenar en el disco de la computadora cliente. Al no poder registrarse la *cookie* en la máquina cliente, resulta entonces, imposible gestionar las sesiones. Este problema directamente puede tener muchas lecturas negativas. Muchos usuarios podrían ignorarlas por falta de conocimiento o por miedo a los tan temidos virus informáticos y otras formas de fraude informático.

ACTIVIDADES 5.1

- Confeccione un formulario que solicite el nombre y apellidos de un usuario y almacénelo en dos variables de sesión denominadas *nombre_usuario* y *apellido_usuario.* Además, cree un hipervínculo a una segunda página que verifique si existen las variables de sesión. En caso de que existan, la página le dará la bienvenida al usuario. En caso contrario, muestre un mensaje indicando que no puede visitar la página.
- A partir de una página inicial que solicite el nombre del usuario en curso, cree una *cookie* llamada "usuario" que almacene el nombre del visitante del sitio. Una vez creada la *cookie*, realice una página PHP que verifique si la *cookie* "usuario" existe. En caso de encontrar la *cookie,* la página nos dará la bienvenida al sitio imprimiendo nuestro nombre en la página web.

5.2 SEGURIDAD: USUARIOS, PERFILES Y ROLES

Un aspecto complementario al manejo del estado en páginas web es el *control de usuarios*. Por lo general, las aplicaciones web están diseñadas para que usuarios con distintos niveles de acceso puedan realizar diversas operaciones, pero ¿cómo podemos organizar dichos niveles de acceso? Uno de los mecanismos más usuales es a través de *perfiles* o *roles*. Así, un usuario cualquiera que entre y utilice nuestra aplicación web. El rol es una de las características del usuario y es la propiedad que establece cuales son las tareas o procesos que el usuario puede ejecutar. De esta manera, se crea un control de acceso jerárquico en el cual un usuario tiene un rol, un rol contiene varias tareas y una tarea es ejecutada a través de varios procesos.

5.2.1 LISTA DE CONTROL DE ACCESO (ACL)

Una vez visto el uso de sesiones, podemos pensar que una de las formas de controlar el acceso de los usuarios es a través de una variable de sesión que nos muestre el rol del usuario en curso. Este sería un método manual el cual sería fácilmente controlable si hablamos de una aplicación pequeña y que sea diseñada por un solo desarrollador. Para aplicaciones de mediana y gran envergadura, existen las listas de control de acceso (ACL). Una lista de control de acceso o ACL (del inglés, *Access Control List*) es un concepto de seguridad informática usado para fomentar la separación de privilegios (Ballad & Ballad, 2008). Es una forma de determinar por medio de grupos y usuarios los permisos de acceso. En el caso de la Web, los permisos pueden determinar quien tiene privilegios de administrador para leer un documento, escribir en una base de datos, imprimir, etc.

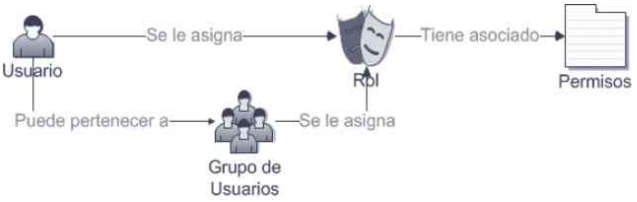


Figura 5.1. *Ejemplo de ACL para usuarios basado en roles o perfiles*

La implementación de un ACL para el control de acceso a un servicio (página web) para restringir el acceso a los usuarios a páginas específicas de acuerdo a los permisos dados a una cuenta particular, es una labor que requiere de dos pasos:

1 La creación de un ACL.

2 Su utilización en la página web.

Para la creación del ACL podemos apoyarnos en el uso de de una base de datos que nos permita especificar la forma en la cual los permisos y roles son modelados en nuestro sistema. Teniendo en cuenta el modelo de roles establecido en la Figura 5.1, presentamos un pequeño esquema de base de datos que podría servir de orientación para la creación de un ACL. De acuerdo a las necesidades particulares de cada aplicación, los campos de cada tabla variarán. La estructura del ACL que proponemos se muestra en la Figura 5.2.

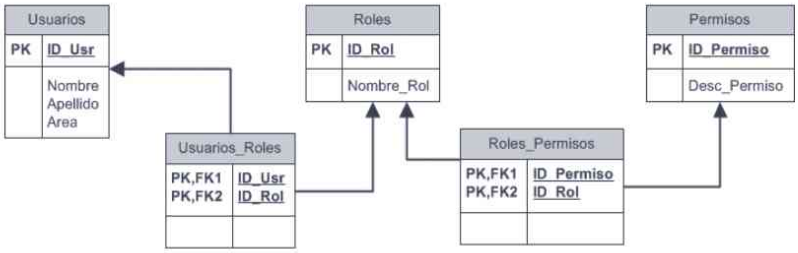


Figura 5.2. *Ejemplo de esquema ACL*

Una vez creado el ACL a través de una base de datos, el siguiente paso es la conexión de nuestras páginas web con dicho ACL. Como el ACL está implementado a través de una base de datos, este proceso se hará de la misma forma en la que se trataría cualquier aplicación de conexión a una base de datos. Así, de acuerdo al lenguaje de servidor que hayamos elegido para realizar la aplicación, crearemos el código PHP o ASP que permita conectarse a una base de datos y para nuestro caso, elegiremos la base de datos ACL_TEST. La forma de conectarse a una base de datos la especificamos en el capítulo 6 de este mismo libro. Hay que tener en cuenta que la implementación de un sistema completo de ACL cubre aspectos tanto de acceso y autenticación de la conexión de un cliente como de administración de los datos del ACL.

Para el manejo del acceso y autenticación del cliente, debemos crear un servicio que sea invocado cada vez que un usuario se autentique en la página web y además, realice todas las operaciones pertinentes para dicho usuario. Es decir, una vez realizada la conexión con la base de datos que maneja el ACL, el siguiente paso es la creación y mapeo de una clase ACL. Esta clase será la clase base para la definición de todos los conceptos ACL como el usuario en curso, los roles que posee un usuario en curso y por último los permisos que posee el usuario en curso. Un ejemplo de cómo puede estar creada esta clase sobre un lenguaje embebido como PHP es el siguiente:

```
<?php
  class ACL{
    var $perms = array();//Guarda los permisos de un usuario
    var $userID = 0; //Es el ID del usuario actual
    var $userRoles = array();//Guarda los roles del usuario
      actual

    function __constructor($userID = '){
      if ($userID != ''){
        $this->userID = floatval($userID);
      }else{
        $this->userID = floatval($_SESSION['userID']);
      }
      $this->userRoles = $this->getUserRoles('ids');
      $this->buildACL();
    }

    function ACL($userID=''){
      $this->__constructor($userID);
    }
  }
%>
```

Si la autenticación es correcta, el siguiente paso es que vayamos a consultar los roles que posee el usuario y que para nuestra base de datos de ejemplo están registradas en la tabla `Usuarios_Roles`. En base a los roles encontrados para un usuario específico, deberemos construir un vector o cualquier otro mecanismo de programación para guardar los permisos de acuerdo a los roles que tenga un usuario. El objetivo de este vector de permisos es evitar repeticiones; así por ejemplo, si un usuario posee los roles de "Administrador" y "Modificador" del sistema, probablemente tendrá varios permisos repetidos dentro de la enumeración.

En cuanto a la administración del ACL, deberemos crear todas las interfaces necesarias para controlar todos los aspectos de administración. En cuanto al control de usuarios, la aplicación de administración del ACL debe permitir buscar a un usuario específico a través de alguna de las características registradas (nombre de usuario, nombre, email, etc.), permitir visualizar un listado con todos los usuarios registrados, donde sea posible visualizar los roles asignados a un usuario específico, visualizar los permisos de un usuario específico, editar los detalles de un usuario o editar los roles asignados a un usuario. De manera similar a la administración de usuarios, deberá existir una interfaz de manejo de roles que permita crear un nuevo rol y asignar los permisos a los que tiene acceso ese nuevo rol. Por último deberá existir una interfaz de administración de permisos donde de manera general se listen los permisos y se permita agregar, modificar o eliminar permisos.

ACTIVIDADES 5.2

- La Web nos ofrece muchos ejemplos de cómo implementar un sistema de autenticación con ACL. El ejemplo *A better login system* de Andrew Steenbuck (*http://net.tutsplus.com/tutorials/php/a-better-login-system/*) nos muestra en detalle la implementación de un sistema ACL tomando como base de datos para la implementación MySQL y, como lenguaje embebido, PHP. Proponemos implementar este ejemplo siguiendo uno a uno cada paso indicado. Verá que la página detalla tanto las funciones y clases PHP para la creación y el manejo del ACL como las páginas PHP para la administración de usuarios, roles y permisos.

5.3 AUTENTICACIÓN DE USUARIOS: OPENID Y OAUTH

Otra de las tendencias que podemos observar actualmente en el desarrollo de aplicaciones para Internet es la denominada *autenticación universal*. El consumo de servicios *on line* (Hotmail, Gmail, Facebook, Twitter, Yahoo, etc.) ha incrementado en los últimos años y con él ha aumentando la colección de nombres de usuario y contraseñas que cada usuario posee para gestionar cada uno de estos servicios. La falta de estandarización en los sistemas de autenticación ha generado que los usuarios empiecen a sufrir de la denominada "fatiga de contraseña" (Cardona, 2010). En general, cada vez que un usuario desea probar un nuevo servicio en la Web, el primer paso es registrarse en el sitio web de dicho servicio. Muchas veces el nombre de usuario que se utiliza usualmente no está disponible o el sistema de validación de contraseñas es más estricto de lo habitual con lo que los usuarios se ven forzados a inventar nuevos nombres de usuario que, desafortunadamente, tienden a olvidar. En algunos casos, los usuarios terminan tomando nota de dichos nombres de usuario y contraseñas en una libreta o en un archivo de texto, lo cual representa un claro problema de seguridad.

Con el fin de atacar esta problemática han surgido iniciativas como *OpenId* y *OAuth*. *OpenId* (*http://openid.net/*) es una iniciativa de identificación digital descentralizada lanzada en 2007 que permite al usuario entrar a una página web pudiendo ser verificado por cualquier otro servidor que soporte este protocolo. *OAuth* (*Open Authorization - http://oauth.net/*) es un protocolo abierto para disponer de una API segura de autorización a través de un método estándar y simple para aplicaciones web y de escritorio. Estas dos soluciones coinciden en que abordan la cuestión de

la autenticación universal, aunque *OAuth* parece ser más extensa, cubriendo también el problema de conceder accesos a recursos hospedados en servicios remotos.

El funcionamiento percibido por el usuario es simple tanto en el caso de *OpenId* como en el de *OAuth*. Así, si un usuario desea acceder a un servicio, en vez de crear tener que crear una cuenta específica para ese servicio puede elegir la opción de autenticarse mediante su cuenta de otro servicio ya existente p. ej. a través de su cuenta de Facebook, de Hotmail o de Twitter. En este caso, el servicio a consumir, en el que el usuario no se ha registrado, se comunica con el servicio validador, es decir, el servicio en el cual el usuario ya está registrado y al que se le solicita que realice la comprobación del usuario (Facebook, Hotmail, Twitter, etc.). Por lo general, el servicio validador envía al servicio a consumir una página segura donde introduciremos el nombre de usuario y contraseña. Una vez validado, el servicio validador comunica al servicio a consumir que el usuario es correcto, activa una sesión autenticada de usuario durante un período de tiempo limitado y nos devuelve a la página del servicio a consumir.

Entre las ventajas de este sistema podemos enumerar las siguientes:

- Con una única cuenta podemos manejar múltiples servicios.
- Limitamos la cantidad de información que nos vemos obligados a proporcionar según a qué proveedores de servicios, incrementando nuestra privacidad.

Estas herramientas de autenticación de usuarios cada vez están más extendidas, formando parte ya de muchos *frameworks* y CMS como Joomla, CodeIgniter, Drupal y WordPress. Por lo tanto, el número de páginas web que soportan este sistema es creciente.

5.3.1 EJEMPLO: UNA GUÍA DE IMPLEMENTACIÓN DE OAUTH

La implementación de *OAuth* no es compleja, sin embargo hay que tener varias cosas en cuenta. En el proceso de implementación de un módulo de autenticación basada en terceros con *OAuth* intervienen tres actores: el servicio al que el usuario quiere acceder usando una cuenta externa al cual se le llama consumidor, el servicio de autenticación externo al que accederá a través de *OAuth* al que se le llama proveedor de identidad y el usuario. De manera general, el procedimiento a implementar es el siguiente:

1 El consumidor debe registrarse como tal en el proveedor del servicio de autenticación. Por ejemplo, si queremos usar el servicio de autenticación de Twitter, el primer paso es conectarse con la cuenta de Twitter que queremos que sea la propietaria de la aplicación, es decir, entramos con la con la cuenta de Twitter del consumidor y registramos una nueva aplicación. En caso de Twitter podemos encontrar esta opción bajo el menú Configuración (Settings) → Conexiones (Connections) en el apartado para desarrolladores con un link "Registre una nueva aplicación" donde luego deberemos rellenar un formulario. El proveedor del servicio nos proporciona los datos necesarios para crear nuestra firma digital y las URL donde se encontrarán los servicios.

2 Cualquier comunicación mediante el protocolo *OAuth* se iniciará con una llamada denominada *Request Token* o petición de *token*. En este paso el consumidor y el proveedor del servicio se autentican mutuamente mediante una firma (*signature*) y acuerdan un *token* que utilizaremos para autenticar al usuario. Un *token* es un código (cadena) único de carácter temporal que sirve para identificar, normalmente, una sesión. El *token* no está asociado a ningún usuario hasta que el usuario lo valida. El *token* es enviado a través de HTTP o HTTPS Request mediante el método *POST* o *GET*. De acuerdo con el RFC5849 (Hammer-Lahav, 2010), la llamada `Request token` debe tener el siguiente formato:

- **oauth_consumer_key**. Es la clave dada por el proveedor del servicio al consumidor y que está asociada a la aplicación que se ha registrado. En general, es un parámetro dado cuando se registra la aplicación.
- **oauth_nonce**. *Nonce* significa "*number used once*" normalmente es una cadena alfanumérica aleatoria generada para cada petición.

- **oauth_signature_method**. Todos los *tokens* deben ser firmados por el consumidor y validados por el proveedor del servicio de autenticación, como mecanismos de control de seguridad. La firma codifica los parámetros del consumidor `consumer_secret` y el `token_secret`. Estos parámetros también deben haber sido dados por el proveedor del servicio al consumidor. El protocolo establece tres posibles métodos de codificación PLAINTEXT, HMAC-SHA1 y RSA-SHA1. Cada proveedor del servicio puede elegir el que desee y deberá ser el método en que el consumidor envíe su petición de *token*.
- **oauth_timestamp**. Es el sello de tiempo actual. Por lo general, el *timestamp* expresa el número de segundos desde las 00:00:00 horas del 1 de enero de 1979 (01/01/1970 00:00:00 GMT). Una de las principales fuentes de problemas es la hora del sistema. Si no está en la zona horaria que corresponde y con el reloj en sincronía, la llamada la petición de *token* fallará.
- **oauth_version**. Es un parámetro opcional. En caso de estar presente su valor debe ser "1.0". Probablemente con la aparición de *OAuth 2.0* se convertirá en un parámetro obligatorio.
- **oauth_signature**. En este parámetro se almacena la firma generada de codificar el parámetro `consumer_secret` de acuerdo al método definido en el parámetro `oauth_signature_method` (Hammer-Lahav, 2010).

3 Una vez lanzado el *Request Token*, el proveedor de identidad devuelve un `oauth_token` y un `auto_token_secret` al consumidor. Estos dos parámetros deben ser guardados para utilizarlos más adelante. Preferiblemente en la sesión del usuario o en una *cookie*.

4 El siguiente paso es redirigir al usuario a la página de autenticación del proveedor de identidad, pasando el *token* como parámetro.

5 Una vez en la página del proveedor del servicio, el usuario se autentica validando el `oauth_token`. Tras validar el `oauth_token` enviado por el consumidor y los datos que el usuario ha ingresado, el proveedor del servicio redirige nuevamente al usuario a la página base que se ha especificado en el momento de dar de alta la aplicación del consumidor en el proveedor del servicio o a la especificada en el parámetro opcional `oauth_callback`.

6 El consumidor recoge al usuario en la URL dada en el *callback* junto con el token de confirmación de identidad. El token con que el proveedor del servicio envía de vuelta al usuario a la aplicación del consumidor es el mismo que ya se tenía almacenado en la sesión de usuario. Comparando ambos *tokens*, se verifica si el usuario es quien dice ser y si esa así se le da acceso a la aplicación.

Si lo que se desea es que el usuario no tenga que introducir su contraseña cada vez que desee autenticarse en la aplicación del consumidor se puede cambiar el `request_token` (de carácter temporal) por un `Access_token` (de carácter permanente).

ACTIVIDADES 5.3

- En la Red podemos encontrar varios ejemplos de cómo implementar *OAuth*. Uno de ellos ha sido escrito por Txau y se denomina *Entendiendo OAuth a través de Twitter* (*http://www.becodemyfriend.com/2010/10/entendiendo-oauth-con-twitter-from-scratch/*), en la que se explica paso a paso cómo hacer la validación de usuarios desde sus cuentas de Twitter. Esta página muestra todo el código PHP que se necesitaría tanto para la creación de Request Token, como para la redirección y validación de la sesión en las páginas PHP del consumidor. Nuevamente, proponemos seguir paso a paso las instrucciones dadas en esta página web para el desarrollo de este ejercicio.

5.4 PROTOCOLO LIGERO DE ACCESO AL SERVICIO DE DIRECTORIOS: LDAP (LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL)

De manera general, un directorio es una guía en la que se lista un conjunto de objetos así como información diversa acerca de cada uno de dichos objetos. Los directorios permiten localizar información y para ello definen que información se almacenará así como la forma en la que ésta se organiza (Carter, 2003). A nivel informático, un servicio de directorio es el conjunto de componentes hardware y/o software que trabajan de forma cooperativa para prestar un servicio de búsqueda de información acerca de recursos electrónicos como usuarios, impresoras, archivos, etc.

LDAP (*Lightweight Directory Access Protocol* o Protocolo Ligero de Acceso al Servicio de Directorios) (Zeilenga, 2006) es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar información diversa en un entorno de red. Está basado en el estándar X.500 para compartir directorios, pero es menos complejo e intensivo en el uso de recursos. El protocolo X.500 propone organizar las entradas del directorio de manera jerárquica, generando directorios con capacidad para almacenar gran cantidad de datos y grandes capacidades de búsqueda que puedan ser fácilmente escalables. El protocolo origen, el X.500 especifica que la comunicación entre el cliente y el servidor de directorio debe emplear el protocolo de acceso a directorios (*Directory Access Protocol* - DAP). Pero DAP es un protocolo a nivel de aplicación, por lo que, tanto el cliente como el servidor deben implementar completamente la torre de protocolos OSI. LDAP surge como una alternativa a DAP y utiliza TCP/IP en lugar de los protocolos OSI con lo cual requiere menos recursos de red. Además, el modelo funcional de LDAP es más simple y ha eliminado opciones raramente utilizadas en X.500. LDAP es más fácil de comprender e implementar. Por esta razón, a veces se habla de LDAP como "X.500 Lite." Al igual que el protocolo X.500, LDAP organiza la información en un modo jerárquico usando directorios. Estos directorios pueden almacenar una gran variedad de información. Sin embargo, en la mayoría de los casos, LDAP se usa simplemente como un directorio telefónico virtual, permitiendo a los usuarios acceder fácilmente a la información de contacto de otros usuarios. Pero LDAP va mucho más lejos que un directorio telefónico tradicional, ya que es capaz de propagar su consulta a otros servidores LDAP por todo el mundo, proporcionando un repositorio de información ad-hoc global. Sin embargo, en este momento LDAP se usa más dentro de organizaciones individuales públicas o privadas.

LDAP es un sistema cliente/servidor. El servidor puede ser implementado usando una gran variedad de bases de datos para guardar el directorio, cada uno optimizado para operaciones de lectura rápidas y en gran volumen. Cuando una aplicación cliente LDAP se conecta a un servidor LDAP puede realizar funciones de consulta de un directorio o de modificación del mismo. Si se realiza una consulta, el servidor puede contestar localmente o dirigir la consulta a otro servidor LDAP que contenga la respuesta. Si la aplicación cliente está intentando modificar información en un directorio LDAP, el servidor verifica primero que el usuario tiene permiso para efectuar el cambio y después añade o actualiza la información.

La seguridad de la información almacenada en el directorio es uno de los aspectos más importantes a la hora de trabajar con LDAP. Algunos directorios deben permitir el acceso público, pero esto no implica que cualquier usuario pueda realizar todo tipo de operaciones. De manera general, se entiende que todos los usuarios puede buscar información sobre otros usuarios, pero solo el administrador o el usuario involucrado debe tener permiso para modificar la información. Es decir, el directorio creado debe permitir la implementación de una política de seguridad que defina los permisos de cada usuario. De acuerdo a la herramienta utilizada, el directorio puede o no incorporar estas capacidades. En caso de no incorporarlas debe poder integrarse con un servicio de red fiable que proporcione estos servicios de seguridad. Un método para gestionar estos roles puede ser a través de una ACL (*Access Control List*), explicado en el apartado 5.2.1.

5.4.1 ESTRUCTURA DE DIRECTORIO LDAP

Un directorio es un conjunto de objetos con atributos organizados de una manera lógica y jerárquica. Además de definir el protocolo de acceso al directorio, el estándar LDAP define cuatro modelos que permiten entender mejor el servicio de directorio.

- **Modelo de información**: describe la estructura de la información almacenada en el directorio LDAP.
- **Modelo de nombres o de nombrado**: describe cómo se organiza e identifica la información en el directorio LDAP.
- **Modelo funcional**: describe qué operaciones pueden ser realizadas sobre la información almacenada en el directorio LDAP.
- **Modelo de seguridad**: describe cómo puede protegerse la información contenida en el directorio LDAP frente a accesos no autorizados.

El modelo de información de LDAP define que la unidad básica de información almacenada en un directorio es la entrada (del inglés, *entry*). Las entradas son un conjunto de datos acerca de un objeto como por ejemplo una persona o una organización. Las entradas son organizadas jerárquicamente, en forma de árbol y es lo que se conoce como árbol de información de directorio (*Directory Information Tree*, DIT).

Una entrada consta de un conjunto de atributos, cada uno de los cuales describe una característica particular del objeto que describe la entrada. Cada atributo tiene un nombre, un tipo y uno o varios valores, es decir, los atributos pueden ser multivaluados. El tipo define el tipo de información que va a almacenar y los valores son la información en sí misma. Los atributos son definidos en un esquema y cada tipo de atributo tiene asociado una determinada sintaxis que describe los tipos de datos y como se van a almacenar los valores de dicho atributo. Para hacer más confiables e identificar de manera no ambigua las entradas un identificador único (UID) podría ser proporcionado en el conjunto de los atributos operacionales de la entrada. Otras restricciones pueden asociarse a los tipos de atributos, para limitar el número de valores o el tamaño total de un atributo, por ejemplo podemos limitar el tamaño del atributo foto, para evitar la utilización de espacio de almacenamiento más allá de los límites razonables. Por otra parte el atributo UID no debería de ser multivaluado.

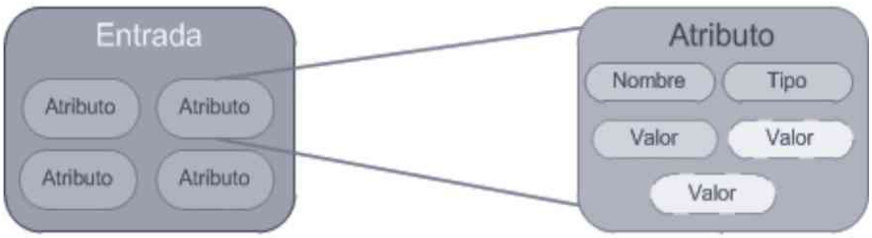


Figura 5.3. *Relación entrada-atributo*

Algunos ejemplos de atributos pueden ser:

Tabla 5.1 Ejemplos de atributos en un LDAP

Acrónimo	Descripción	Nombre
CN	Nombre Común	*Common Name*
O	Organización	*Organization*
OU	Unidad Organizativa - Departamento	*Organization Unit*
C	País	*Country*
DC	Componente de dominio	*Domain component*
Mail	Correo electrónico	*Email*
Phone	Teléfono	

Una entrada en un directorio posee un conjunto de atributos que son requeridos mientras que otros son permitidos u opcionales. Al conjunto de atributos permitidos y requeridos se le denomina esquema (*schema*). Los esquemas definen el tipo de objetos que se pueden almacenar en el directorio, que atributos debe contener, el formato de estos atributos y si son opcionales u obligatorios. El esquema también define las clases de objetos y en qué puntos del DIT pueden aparecer. Las clases de objeto representan el tipo de datos que los atributos de las entradas pueden tener. Una entrada puede tener asociados uno o más clases de objetos.

Por otro lado, el modelo de nombrado enuncia que las entradas en un directorio son organizadas como un árbol invertido. El modelo de nombrado de LDAP es importante porque permite dar un nombre único a cada una de las entradas del directorio. Para ello, cada entrada posee un identificador único o nombre distintivo (*Distinguished Name*, DN). Un DN está constituido de partes más pequeñas denominadas *Relative Distinguished Name* (RDN). Un RDN está constituido por uno o más pares nombre-valor. Cada nombre y valor son separados por un signo igual (=), por ejemplo para un RDN conformado por el par "`uid = pperez`", `uid` es el nombre y el valor es `juan`. Si el RDN está constituido por más de un par, estos se separan por un signo mas (+), por ejemplo un RDN podría ser "uid = pperez + cn = Pepe Pérez". Cada RDN se corresponde con una rama del DIT partiendo de la raíz hacia la entrada dentro del directorio.

Ilustraremos la diferencia entre RDN y DN con un ejemplo. Para ello nos basaremos en el árbol LDAP que presenta la Figura 5.4.

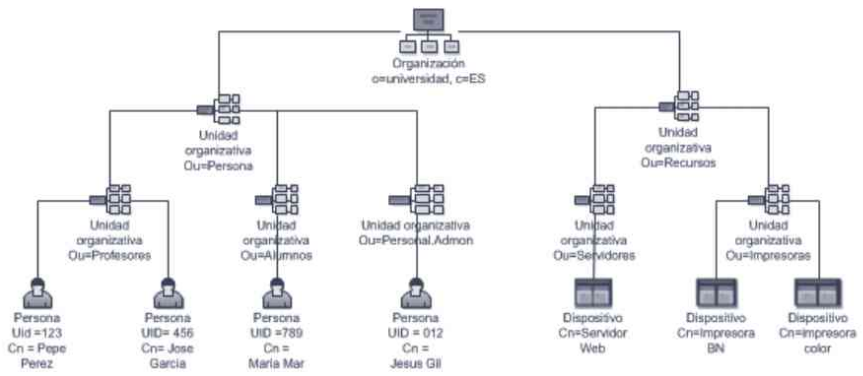


Figura 5.4. *Ejemplo de árbol LDAP*

En la cima del árbol se encuentra la organización, cuyo nombre distintivo es `o=universidad`. Dentro de esta entrada hay otras dos entradas que describen unidades organizativas, una que describe el personal y otra que describe los recursos de red que son `ou=Personas` y `ou=Recursos`. Bajo la entrada de `ou=Personas` se encuentran otras tres entradas que describen otras tres unidades organizativas: `ou=Alumnos`, `ou=Profesores` y `ou=Personal.Admon`. El último nivel está constituido por entradas que representan personas o dispositivos particulares. Así, por ejemplo encontramos una entrada que corresponde al `cn=Pepe Perez`, que está bajo la unidad organizativa Profesores, de la unidad organizativa Personas de la organización denominada Universidad. En este caso `uid= 123` es el RDN del objeto mientras que el DN es `uid=123456789`, ou =Alumnos, ou=Personas, o=urjc. De este esquema deducimos que no puede haber ninguna entrada suelta y solo la entrada raíz puede no tener entrada padre.

El nivel superior de un directorio LDAP es la base, conocido como el "DN base". Un DN base, generalmente, toma una de las siguientes tres formas:

- **o="nombre_organizacion", c=US** (*DN base en formato X.500*).
  En este ejemplo, o= "" nos referimos a la organización, que en este contexto debería ser tratada como un sinónimo del nombre de la empresa. `c=US` indica que el país base de la empresa está en los `US`.
- **o=nombre_organizacion.com** (*DN base derivado de la presencia en Internet de la empresa*). Este formato utiliza el nombre de dominio de la empresa como base.
- **dc=nombre_organización, dc=com** (*DN base derivado de los componentes de dominio DNS de la empresa*). Este formato utiliza el nombre de dominio DNS como su base. Este formato está separado en componentes de dominio: `nombre_de_organizacion.com`.

Habitualmente, un LDAP almacena la información de autenticación, usuario y contraseña, y es utilizado para validarse, aunque es posible almacenar otra información como por ejemplo, datos de contacto del usuario, ubicación de diversos recursos de la Red, permisos, certificados, etc. Un árbol de directorio LDAP puede reflejar diversos límites políticos, geográficos u organizacionales, dependiendo del modelo elegido. Los despliegues actuales de LDAP tienden a usar nombres del sistema de nombres de dominio (DNS) para estructurar los niveles más altos de la jerarquía. Conforme descendemos en el directorio pueden aparecer entradas que representan personas, unidades organizacionales, impresoras, documentos, grupos de personas o cualquier cosa que representa una entrada dada en el árbol (o múltiples entradas).

En cuanto al modelo funcional éste nos indica el conjunto operaciones que permiten controlar el acceso a los datos del directorio. El modelo de interacción con un servidor LDAP es el siguiente: un cliente inicia una sesión de LDAP conectándose a un servidor LDAP, por defecto en el puerto TCP 389. El cliente envía una petición de operación al servidor, y el servidor envía respuestas. Con algunas excepciones, el cliente no necesita esperar una respuesta antes de enviar la siguiente petición, y el servidor puede responder en cualquier orden. El cliente puede requerir las siguientes operaciones:

- **Bind**. Sirve para autenticarse y especificar una versión del protocolo LDAP.
- **Search**. Busca y obtiene entradas de directorio.
- **Compare**. Prueba si una entrada nombrada contiene un valor del atributo dado.
- **Add**. Agrega una nueva entrada al directorio.
- **Delete**. Elimina una entrada del directorio.
- **Modify**. Modifica una entrada del directorio.
- **Modify Distinguished Name (DN)**. Modifica o renombra una entrada.
- **Abandon**. Aborta una petición previa.
- **Extended Operation**. Es el comando para definir otras operaciones de manera genérica.
- **Unbind**. Cierra la conexión.
- **Start TLS** (desde LDAPv3). Activa la extensión *Transport Layer Security* (TLS) para realizar conexiones seguras.

Además de las funciones enunciadas, el servidor LDAP puede enviar "notificaciones no solicitadas" que no son respuestas a ninguna petición, por ejemplo antes de que se termine el tiempo de conexión.

Finalmente, el propósito del modelo de seguridad de LDAP es proteger la información del directorio contra accesos no autorizados. La función de autenticación de un cliente LDAP (`bind`) en un servidor es parte de la seguridad de un directorio. Otra parte del modelo de seguridad la conforman los controles de acceso, que son la forma en la cual se especifican los privilegios que poseen ciertos usuarios, después de ser autenticados exitosamente. Estos controles de acceso a un directorio aún no han sido estandarizados aunque se está trabajando en ello.

Active Directory es el nombre utilizado por Microsoft desde Windows 2000, como almacén centralizado de información de uno de sus dominios de administración. Bajo este nombre se encuentra realmente un esquema LDAP versión 3, lo cual permite integrar otros sistemas que soporten el protocolo. En este LDAP se almacena información de usuarios, recursos de la Red, políticas de seguridad, configuración, asignación de permisos, etc. Este servicio puede ser activado si disponemos de una versión servidora del sistema Windows como Windows Server.

ACTIVIDADES 5.4

La Red ofrece varios manuales sobre la forma de activar el servicio Active Directory, por ejemplo:

a. Instalar Microsoft Windows Server 2003: Instalación de Windows Server 2003 Enterprise Edition SP2 (*http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=394*)

b. Promocionar un equipo con Windows Server 2003 al controlador de dominio (activar Active Directory) *http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=347*

5.4.2 VALIDACIÓN WEB EN UN SERVIDOR LDAP

Lo primero que necesitamos para programar una página web que se conecte a algún servicio LDAP es activar el soporte de LDAP para el lenguaje embebido o de páginas de servidor que hayamos elegido. Por ejemplo, en caso de que poseamos un servidor de aplicaciones correspondiente (Apache, IIS, etc.) con PHP debemos activar este soporte a través del archivo de configuración de PHP, `php.ini`. Este es un archivo de texto, que puede ser editado con el bloc de notas (Notepad) y debemos buscar la línea:

```
;extension = php_ldap.dll
```

Y quitar el comentario, es decir, el ";" inicial. La línea quedará de la siguiente manera:

```
extension = php_ldap.dll
```

El quitar un comentario a una línea del `php.ini`, implica la activación de un servicio a través del uso de una librería. El nombre de la librería corresponde con la línea activada. En este caso, la librería se llama `php_ldap.dll` y, por tanto, también se debe verificar que la librería existe en la carpeta "ext" de la instalación del PHP. Para garantizar que los cambios sean tomados, es conveniente reiniciar el servidor de aplicaciones, sea el Apache o el IIS.

Una vez hemos configurado el servidor de aplicaciones para que soporte acceso a LDAP podemos usar algunas de las funciones del catalogo de funciones PHP para la conexión con LDAP. Una secuencia típica en una aplicación que accede un servidor LDAP es la siguiente:

```
ldap_connect(): Establece la conexión al servidor LDAP
ldap_bind(): Autentifica a un usuario, ya sea anonymous o un "login" provisto
Efectuar una búsqueda o actualización del directorio, desplegar resultados, etc.
ldap_close(): Cierra la conexión con servidor LDAP
```

La función `ldap_connect` nos permite conectarnos con un servidor LDAP. El formato de esta función es el siguiente:

```
resource ldap_connect ([ string $nombre_host [, int $puerto
 ]] )
```

Devuelve un identificador de conexión positivo en caso de éxito, ó falso si ocurre algún error. `ldap_connect()` establece una conexión con el servidor LDAP especificado en `nombre_host` y `puerto`. Ambos argumentos son opcionales. Si no se especifican, devuelve el identificador de la conexión LDAP actualmente abierta. Si solo es especificado `nombre_host`, el puerto tomado por defecto es el 389.

El siguiente código muestra un ejemplo de conexión a un servidor LDAP:

```
<?php
  //Nombre del servidor LDAP a que queremos conectarnos
  $servidor_ldap = "ldap.example.com";
  $puerto_ldap = 389; // No. del puerto al que se conectara

  // Estableciendo la conexión con el servidor LDAP
  $conexion_ldap = ldap_connect($servidor_ldap, $puerto_ldap)
  or die("No ha sido posible conectarse al servidor
    $servidor_ldap");
?>
```

Una vez nos hemos conectado con el servidor LDAP utilizamos la función `ldap_bind` para realizar la validación con el usuario y la contraseña indicados. La sintaxis de la función `ldap_bind` es la siguiente:

```
bool ldap_bind (resource id_de_conexion [, string
  rdn_del_usuario [, string contraseña]] )
```

Si no especificamos ningún dato de nombre de usuario o contraseña, quiere decir que nos estamos registrando con el usuario anonymous o el usuario invitado. Por lo general, este usuario tiene privilegios de solo lectura (*read-only*).

Otra de las funciones básicas trabajando con LDAP es la realización de búsquedas. En el caso de PHP, esta labor se realiza por medio de la función `ldap_search`.

```
resource ldap_search ( resource identificador_de_conexion,
  string dn_base, string filtro [, array atributos [, int
  solo_atributos [, int tamano_limite [, int tiempo_limite [,
  int deref]]]]] )
```

El ejemplo presentado a continuación muestra la búsqueda de una entrada en el directorio cuyo `uid` sea igual a "`dsanchez`", si el `uid` se encuentra se despliega el campo *email*.

```
<?php
  $ds = ldap_connect("localhost"); // debe ser un servidor
    LDAP
```

```
  // Si la conexión es exitosa hacer una autenticación
  if ($ds) {
    // Registrándonos como anonymous
    $r = ldap_bind($ds);
    echo "Resultado del bind(): ".$r."<p>";

    /* Una vez validados, realizamos un búsqueda en el LDAP.
    Buscamos los usuarios con uid = dsanchez y retornaremos
    su email */
    $sr = ldap_search($ds,"dc=Udec,dc=CL", "uid=dsanchez");

    /* Contamos el número de entradas retornado por la
    búsqueda */
    echo "Numero de entradas retornadas:
      ".ldap_count_entries($ds,$sr)."<p>";

    //Obtenemos las entradas retornadas
    $info = ldap_get_entries($ds, $sr);
    echo "Valor para: ".$info["count"]." itemes
      retornados:<p>";
    for ($i=0; $i<$info["count"]; $i++) {
      echo "uid es: ". $info[$i]["uid"] ."<br>";
      echo "Entrada email: ". $info[$i]["email"][0] ."<p>";
    }

   //Cerramos la conexión con el LDAP
   ldap_close($ds);
  }else {
    echo "<h4>No se puede conectar al servidor LDAP</h4>";
  }
?>
```

Si lo que deseamos es agregar una entrada al LDAP, la función que debemos utilizar es `ldap_add`. Un ejemplo de su utilización es el siguiente:

```
<?php
  $ds=ldap_connect("localhost");
  if ($ds) {
    /* Nos autenticamos con un usuario con privilegios de
      modificación en el LDAP */
    $r=ldap_bind($ds,"cn=root, o=My Company, c=ES",
      "secret");

    //Preparamos la información a insertar en el vector
      $info.
    $info["cn"]="John Jones";
    $info["sn"]="Jones";
    $info["mail"]="jonj@here.and.now";
    $info["objectclass"]="person";
```

```
    //Agregamos la información al directorio
    $r=ldap_add($ds, "cn=John Jones, o=My Company, c=US",
      $info);
    ldap_close($ds);
  } else {
    echo "No ha sido posible conectarse con el servidor
      LDAP";
  }
?>
```

Para comparar una información externa con otra que ya tenemos en el LDAP podemos utilizar la función `ldap_compare ()`.

```
ldap_compare (resource id_de_conexion, string dn, string
  atributo, string valor)
```

El siguiente código muestra un ejemplo de la utilización de esta función, teniendo en cuenta que tenemos una conexión LDAP y que la conexión la guardamos en una variable denominada `$ds`:

```
<?php
  …
  //Preparamos los datos a comparar
  $dn = "cn=Pedro Perez, ou=Mi Unidad, o=Mi Compania, c=ES";
  $valor = "contrasena_secreta";
  $atributo = "password";

  // comparamos los valores
  $r=ldap_compare($ds, $dn, $atributo, $valor);
  if ($r === -1) {
    echo "Error: " . ldap_error($ds);
  }elseif ($r === true) {
    echo "Contraseña correcta.";
  } elseif ($r === false) {
    echo "La contraseña proporcionada es incorrecta.";
  }
```

ACTIVIDADES 5.5

- El catálogo completo de funciones PHP para el manejo de LDAP se encuentra en la página oficial de PHP. Consúltenla.

5.5 PRUEBAS Y DEPURACIÓN

No importa el ciclo de vida de creación del software que utilicemos (cascada, ágil, proceso unificado, en espiral, etc.) una de las etapas que siempre aparece es la constitución de las pruebas. Las pruebas son una actividad en la cual un sistema o un componente de un sistema es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados, y se realiza una evaluación de algún aspecto del sistema o componente.

Las pruebas son esa etapa que garantiza la mínima calidad en el software. Esta calidad se refiere a que el componente desarrollado cumple con el objetivo para el que fue creado, es decir, hace lo que tiene que hacer. A nivel informático, podemos decir que una prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Las pruebas se realizan a través de la ejecución de casos de prueba. Un caso de prueba es la especificación formal de una prueba que será realizada a un software o programa específico. Un caso de prueba debe definir claramente cuál es el mecanismo para probar el software, cuáles son las entradas de prueba y definir cuáles son los posibles resultados esperados. Si los resultados esperados son los que se producen al ejecutar la prueba quiere decir que el software pasa la prueba; por el contrario, si el resultado producido por la prueba no coincide con el resultado esperado quiere decir que existe un error en el programa. Por ello, un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta ese momento. Ente los objetivos de la creación de pruebas están:

- Diseñar casos de pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.
- Encontrar y documentar los defectos que puedan afectar la calidad del software.
- Validar que el software funcione de acuerdo a como éste ha sido diseñado.
- Validar y probar los requisitos que debe cumplir el software realizado.
- Validar que todos los requisitos han sido implementados correctamente.
- Verificar que la interacción e integración entre los componentes es adecuada y correcta.
- Asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.

5.5.1 CLASIFICACIÓN DE PRUEBAS

Cuando comenzamos a trabajar en un software determinado, es necesario realizarles distintos tipos de pruebas, para lograr una buena calidad en el mismo y así lograr la menor cantidad posible de errores. Es por esto que las pruebas se clasifican en dos grandes grupos, ya que toda aplicación puede ser probada bajo los siguientes esquemas:

1. Demostrar que la operación interna de un módulo de programa se ajusta a lo especificado y que los componentes internos funcionan correctamente. Esta prueba se desarrolla en base a caminos lógicos y se denomina *pruebas de caja blanca*.
2. Conociendo la función del programa (servicio o página web), demostrar que esta función se realiza de manera correcta. En este caso las pruebas se realizan sobre las interfaces y se denominan *pruebas de caja negra*.

Las técnicas de pruebas de caja blanca permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa las estructuras de control del diseño procedimental para derivar casos de prueba que garanticen que:

- ✓ Que se ejecutan al menos de una vez todos los caminos independientes de cada módulo.
- ✓ Que se prueben todas las decisiones lógicas en sus ramas verdaderas y falsas.
- ✓ Ejecutar todos los bucles o ciclos con los límites que se les haya definido.
- ✓ Ejecutar las estructuras internas de datos para asegurar su validez.

Uno de los métodos de caja blanca más conocido es la complejidad ciclomática de McCabe. Este método se apoya en la realización de un grafo para detectar el número total de caminos en un código. Para una explicación más exhaustiva de esta técnica puedes consultar (Piattini, Calvo-Manzano, Cervera & Fernandez, 2007).

Por otra parte, las técnicas de caja negra son complementarias a las de caja blanca aunque en la práctica las pruebas de caja negra se realizan con mucha más frecuencia que las técnicas de caja blanca. Estas pruebas se llevan a cabo sobre la interfaz del software, y no se evalúa ni el comportamiento interno ni la estructura del programa.

Los casos de prueba de la caja negra pretenden demostrar que:

- ✓ Las funciones del software son operativas.
- ✓ Las entradas se aceptan de forma adecuada.
- ✓ Se produce una salida correcta.
- ✓ La integridad de la información externa se mantiene.

La realización de pruebas de software implica pruebas a diferentes niveles. Es necesario probar si cada componente independiente funciona, luego es necesario probar si los distintos componentes encajan entre sí y por último es necesario probar el sistema globalmente. Aunque este proceso puede resultar lógico, en muchos proyectos no se realizan todas las pruebas o no se les da el valor necesario.

De manera ideal, las pruebas deben ser llevadas a cabo por personas distintas a los programadores de los componentes, así se puede verificar además del correcto funcionamiento del programa, su correcta concepción e interpretación. Otras de las consideraciones que tenemos que tener en cuenta en la ejecución de pruebas de software son:

- ✓ La prueba puede ser usada para mostrar la presencia de errores, sin embargo, no se puede asegurar la ausencia de errores.
- ✓ La principal dificultad del proceso de prueba es decidir cuándo parar.
- ✓ Hay que evitar casos de pruebas no planificados, no reusables o triviales, a menos que el programa sea muy sencillo.
- ✓ Una parte necesaria de un caso de prueba es la definición del resultado esperado.
- ✓ Los casos de pruebas tienen que ser escritos no solo para condiciones de entrada válidas y esperadas sino también para condiciones no válidas e inesperadas.
- ✓ Los casos de pruebas tienen que ser escritos para generar las condiciones de salida deseadas.
- ✓ El número de errores sin descubrir es directamente proporcional al número de errores descubiertos.
- ✓ Las pruebas deberían empezar por “lo pequeño” y progresar hacia “lo grande”.
- ✓ Con la excepción de las pruebas de unidad e integración, un programa no debería ser probado por la persona u organización que lo desarrolló.
- ✓ Las pruebas deberían ser realizadas por un equipo independiente.

TIPOS DE PRUEBAS

No existe una división estándar acerca de los tipos de pruebas que pueden existir. Sin embargo y de acuerdo al ámbito y alcance de las pruebas, estas pueden ser divididas en pruebas de unidad, integración, seguridad, carga, volumen, de sistema y de regresión.

- **Pruebas de Unidad**. La prueba de unidad se centra en el módulo (clase, función, etc.). Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca.
- **Pruebas Funcionales**. Estas pruebas evalúan una funcionalidad completa, donde pueden estar implicadas una o varias clases, la propia interfaz de usuario y recursos locales en el cliente como *scripts* AJAX.
- **Pruebas de Integración**. El objetivo de las pruebas de integración es tomar todos los módulos probados en las pruebas de unidad y construir una estructura de programa que pruebe la integración de estos módulos de acuerdo a lo especificado en el diseño. Hay dos formas de integración:
  - Integración no incremental: se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.
  - Integración incremental: el programa se integra progresivamente y se va probando en pequeños segmentos.
- **Pruebas de Aceptación**. Son pruebas funcionales, pero vistas directamente desde el cliente. Son aquellas pruebas que demuestran al cliente que la funcionalidad está terminada y funciona correctamente.
- **Pruebas del Sistema**. Las pruebas de sistema verifican que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora.
- **Pruebas de Regresión**. Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad. El propósito de estas pruebas es asegurar que los defectos identificados en la ejecución anterior de la prueba se han corregido y que los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores.
- **Pruebas de Seguridad**. Las pruebas de seguridad intentan verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios. Por supuesto, la seguridad del sistema debe ser probada en su invulnerabilidad frente a un ataque frontal, pero también debe probarse en su invulnerabilidad a ataques por los flancos o por la retaguardia. Durante la prueba de seguridad, el responsable de la prueba desempeña el papel de un individuo que desea entrar en el sistema. Debe intentar conseguir las claves de acceso por cualquier medio, puede atacar al sistema con software a medida, diseñado para romper cualquier defensa que se haya construido, o buscar bloquear el sistema, negando así el servicio a otras personas, debe producir a propósito errores del sistema, intentando acceder durante la recuperación o debe curiosear en los datos sin protección, intentando encontrar la clave de acceso al sistema, etc.
- **Pruebas de Carga**. El objetivo de las pruebas de carga es determinar el rendimiento del sistema bajo condiciones de carga (peticiones a una página web, etc.) que se aproximan a la realidad esperada en producción. Da una idea de la escalabilidad del software y cuáles son los límites prácticos.
- **Pruebas de Volumen**. En estas pruebas se busca encontrar debilidades en el sistema al momento de manejar grandes volúmenes de datos durante prolongados períodos de tiempo. El objetivo principal es determinar si la plataforma de integración se degrada o deja de funcionar al manejar grandes volúmenes de datos.

5.5.2 EJECUCIÓN DE PRUEBAS

En general, existen dos formas de realizar pruebas en un software: las pruebas manuales y las pruebas automatizadas. Las pruebas manuales son aquellas que son realizadas de manera manual por el probador. De acuerdo al tipo de prueba el probador podrá ser desde el desarrollador en el caso de la realización de depuración del código o el usuario en el caso de la ejecución de pruebas de aceptación. En las pruebas manuales, el probador es quien establece el protocolo de pruebas y quien realiza las acciones para validar y verificar el software. Las pruebas manuales pueden servir para realizar tanto pruebas unitarias, como pruebas funcionales o pruebas de aceptación. Un ejemplo de realización de una prueba unitaria manual puede ser el probar una funcionalidad en una página web. El protocolo tácito de pruebas establecerá que el objetivo es ejecutar la página bajo uno o varios navegadores y verificar si la funcionalidad se ejecuta correctamente. Si el resultado mostrado en la página web es el esperado, se concluirá que la prueba es satisfactoria. En caso contrario, se detectarán las condiciones en las cuales la funcionalidad no respondió satisfactoriamente y se procederá a la toma de las acciones necesarias para corregir el problema. Muchas veces, estas medidas también son de tipo manual estableciendo estrategias de seguimiento como:

- Usar sentencias del tipo `echo`, print, print_r, var_dump para imprimir por pantalla los valores que toman determinadas variables y establecer si dichos valores son correctos o no.
- Comentar el código e irlo activando poco a poco hasta detectar el error.
- Probar la inclusión de librerías o rutinas, aun sin justificación, con la esperanza que alguna logre el resultado esperado.
- Usar un código encontrado en la Red, aun sin entender lo que hace.

Estos mecanismos manuales pueden ser útiles cuando los módulos a probar son pequeños y manejables, pero si trabajamos en entornos grandes y el volumen de líneas de código a probar es muy elevado puede ser engorroso y poco práctico.

Para los casos en los que el volumen de código es muy grande están las pruebas automatizadas**,** las cuales se apoyan en el uso de herramientas para la creación y ejecución de pruebas. Sobre el qué, en qué medida y dónde automatizar hay mucho escrito. Nosotros estudiaremos la propuesta de Crispin y Gregory (Crispin & Gregory, 2009), quienes clasifican los diferentes tipos de pruebas y la estrategia recomendada para las mismas, en cuadrantes. La Figura 5.5, muestra una adaptación de la propuesta de estos autores:

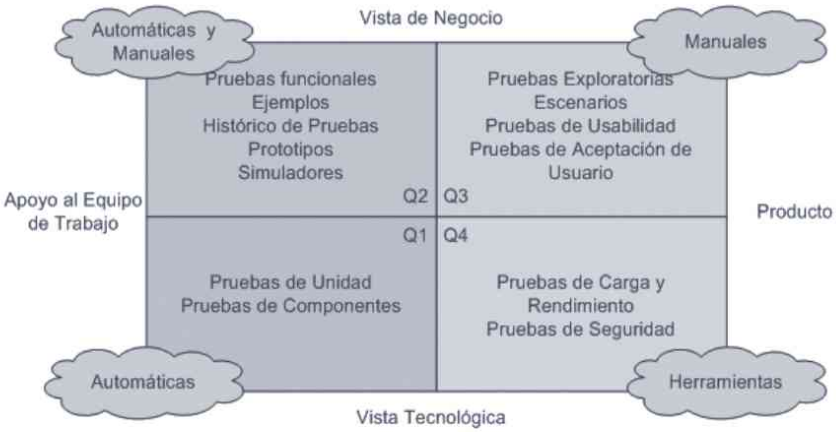


Figura 5.5. *Tipos de pruebas y como realizarlas, de acuerdo a (Crispin & Gregory, 2009)*

Cada cuadrante representa un conjunto de pruebas mientras que las nubes representan el mecanismo que se recomienda para facilitar tanto su ejecución. Veamos cada cuadrante con un poco más de detenimiento:

- **Cuadrante 1**. Pruebas unitarias y de componentes, que normalmente se recomienda automatizar.
- **Cuadrante 2**. Pruebas que pueden realizarse de manera automática o manual, y que suelen ser las pruebas funcionales, simulaciones, prototipos, etc.
- **Cuadrante 3**. Pruebas manuales, que suelen ser las de usabilidad, aceptación, de exploración y alpha/beta testing.
- **Cuadrante 4**. Pruebas que se hacen con el apoyo de herramientas, como son las de rendimiento y carga.

Existen varios tipos de herramientas para apoyar el desarrollo de pruebas. Algunas de las más usadas son:

- **Depurador de código**. Herramienta para probar y encontrar errores en la ejecución de software. Rastrea cambios en variables puede detener la ejecución, etc. Casi todos los entornos de desarrollo poseen un depurador y es aquel que nos permite ejecutar la aplicación paso a paso o configurar ventanas de inspección para controlar variables en el sistema.
- **Analizador de rendimiento**. Estas herramientas se encargan de capturar el perfil de comportamiento de una pieza de software. Permite conocer exactamente que partes son los cuellos de botella. Muestra el comportamiento real de las diversas partes, lo que a veces no corresponde con lo diseñado o modelado.

5.5.3 EJEMPLO DE HERRAMIENTA PARA PRUEBAS UNITARIAS: PHPUNIT

PHPUnit es una aplicación que nos proporciona un excelente conjunto de características para ayudarnos a probar nuestro código. Para instalar PHPUnit, tenemos que hacerlo a través de su canal PEAR (*PHP Extensión and Application Repository*). PEAR (*http://pear.php.net/*) es un marco de trabajo y distribución de componentes PHP reutilizables. El instalador de PEAR instalará todas las dependencias que podamos necesitar. El siguiente código muestra las instrucciones para la correcta instalación de PHPUnit:

```
pear channel-discover pear.phpunit.de
pear channel-discover components.ez.no
pear channel-discover pear.symfony-project.com
pear install phpunit/PHPUnit
```

También es posible instalar PHPUnit de forma manual, las instrucciones detalladas de instalación pueden encontrarse en el manual *on line* de la aplicación (*http://www.phpunit.de/manual/current/en/*). PHPUnit se basa en varias librerías de PEAR, por tanto, es importante tener esta aplicación para que funcione correctamente.

Una vez instalado PHPUnit, vamos a escribir los casos de prueba respectivos. Los casos de prueba tienen que ser programados, para que los mismos puedan ser guardados y puedan volverse a ejecutar fomentando así la reutilización de la prueba. De manera general, la clase de prueba extenderá de la clase `PHPUnit_Framework_TestCase`. Esto nos dará acceso a ciertas funcionalidades como los métodos para las pruebas `setUp()` y `tearDown()`. Una buena práctica es que el nombre de la clase de prueba imite al nombre de la clase que estemos probando. Por ejemplo, si queremos probar la clase `conexiónRemota`, utilizaremos como nombre de la clase de prueba `TestConexionRemota`.

Al crear métodos de prueba, es necesario que empiecen con la palabra "test", por ejemplo `testConexionBD()`. Los métodos de la clase de prueba deben ser públicos ya que si tenemos métodos privados en las pruebas, estas no podrán ser ejecutadas como pruebas por PHPUnit. Es deseable que cuando se escriba una prueba, éstas sean lo más autónomas posibles.

Empezaremos con algunas funciones de prueba, y aquí presentamos la clase con la que vamos a trabajar en los siguientes ejemplos y que llamaremos `RemoteConnect.php`:

```
<?php
  class RemoteConnect{
    public function connectToServer($serverName=null){
      if($serverName==null){
        throw new Exception("¡Este no es un nombre de
          servidor!");
      }
      $fp = fsockopen($serverName,80);
      return ($fp) ? true : false;
    }
    public function returnSampleObject(){
      return $this;
    }
  }
?>
```

Esta funcionalidad la probaremos haciendo una petición a un servidor remoto, la clase test para dicha prueba puede tener el siguiente aspecto:

```
<?php
  require_once('RemoteConnect.php');
  class RemoteConnectTest extends PHPUnit_Framework_TestCase{
    public function setUp(){ }
    public function tearDown(){ }
    public function testConnectionIsValid(){
      //prueba para asegurarse de que el objeto de un
        fsockopen es válido
      $connObj = new RemoteConnect();
      $serverName = 'www.google.com';
      $this->assertTrue($connObj->
        connectToServer($serverName) !==false);
    }
  }
?>
```

Podemos ver que la clase `RemoteConnectTest` extiende la clase `TestCase` de PHPUnit, por lo que una gran cantidad de funcionalidades provienen con ella. Los dos primeros métodos `setUp` y `tearDown` son ejemplos de funcionalidades integradas. Estas son funciones de ayuda que se ejecutan como parte de la prueba normal de funcionamiento y se ejecutan antes de las pruebas y después de que todo haya sido ejecutado, respectivamente. A pesar de su utilidad, no vamos a centrarnos aún en ellos. En este caso, el método interesante es el método `testConnectionIsValid()`. Este método establece el entorno necesario con la creación de una nueva instancia de nuestra clase `RemoteConnect` y llama al método `connectToServer()`. Dentro de este método encontramos la función `assertTrue`. Esa es una de las funciones de ayuda de PHPUnit, y comprueba que una expresión booleana tenga el valor `true`. Otras funciones de ayuda pueden poner a prueba las propiedades del objeto, la existencia de archivos, la presencia de una clave dada en una matriz o una coincidencia en una expresión regular, solo por mencionar algunas. En este caso, queremos asegurar que el resultado de `connectToServer` no es `false`, esto querría decir que la conexión ha fallado por alguna razón.

Para ejecutar las pruebas hay que llamar al ejecutable PHPUnit y señalar el archivo que contiene el código de las mismas. Para nuestro ejemplo, la forma de llamar a la prueba establecida en la clase `RemoteConnectTest` es:

```
phpunit /path/to/tests/RemoteConnectTest.php
```

La salida es fácil de interpretar. Por cada una de las pruebas del caso de prueba, PHPUnit se ejecuta a través de ellos y recoge algunas estadísticas como el número de pruebas y aserciones. He aquí una vista de la salida de nuestro ejemplo:

```
PHPUnit 3.4 by Sebastian Bergmann
.
Time: 1 second
Tests: 1, Assertions: 1, Failures 0
```

Para cada prueba que ejecutamos, veremos un punto (.) si se tiene éxito (como en el ejemplo), una "F" si ocurrió un error, y una "I" si la prueba está marcada como incompleta o una "S" si se ha marcado como Omitida (*Skipped*). De forma predeterminada, PHPUnit está configurado para ejecutarse a través de un conjunto de pruebas a la vez e informar las estadísticas totales en un informe sencillo.

La prueba de ejemplo muestra una prueba de paso ya que siempre que proporcionemos los parámetros correctos, nuestro método funcionará como se espera. Pero, también es necesario configurar una prueba para ver el comportamiento si los parámetros son incorrectos. Por ejemplo, ¿qué sucede si el nombre del host que se proporciona al método no existe? ¿El método produce una excepción? Hay que asegurarnos de que a la hora de escribir las pruebas, controlaremos tanto los parámetros validos como los parámetros inválidos. En el caso del método `connectToServer()` de nuestra clase de ejemplo, el proporcionar un nombre de host no válido para la conexión lanzará una excepción. Para conocer más acerca del manejo de excepciones con PHPUnit recomendamos consultar la documentación *on line* (*http://www.phpunit.de/manual/current/en/writing-tests-for-phpunit.html*).

PHPUnit se vale de aserciones para ayudar al probador a validar el software. Una aserción es una afirmación que se considera cierta. Existen muchas y diferentes aserciones y cada una de ellas nos ayuda a probar los resultados de todos los tipos de llamadas en nuestras aplicaciones. De hecho, las aserciones provistas por PHPUnit cubren la mayoría de los casos que queramos probar. La siguiente lista muestra algunas de las aserciones más comunes:

- **AssertTrue/AssertFalse**. Comprueba la entrada para verificar si es igual a `true`/`false`.
- **AssertEquals**. Comprueba el resultado frente a otra entrada en busca de coincidencias.
- **AssertGreaterThan**. Comprueba el resultado para ver si es mayor que un valor (también hay `LessThan`, `GreaterThanOrEqual`, y `LessThanOrEqual`).
- **AssertContains**. Comprueba que la entrada contiene un valor específico.
- **AssertType**. Comprueba que una variable es de un cierto tipo.
- **AssertNull**. Comprueba que una variable sea nula.
- **AssertFileExists**. Comprueba que un archivo exista.
- **AssertRegExp**. Comprueba la entrada con una expresión regular.

Por ejemplo, si deseamos obtener un objeto proveniente de un método llamado `returnSampleObject` y queremos comprobar si el objeto devuelto es una instancia de una clase en particular. En ese caso el código sería el siguiente:

```
<?php
  function testIsRightObject() {
    $connObj = new RemoteConnect();
    $returnedObject = $connObj->returnSampleObject();
    $this->assertType('remoteConnect', $returnedObject);
 }
?>
```

Al igual que en cualquier área de desarrollo de software, la especificación de pruebas también cuenta con un conjunto de mejores prácticas (*best practices*). La primera es "una prueba, una aserción". Esta estrategia indica que para cada una de las pruebas, solo puede haber una comprobación o aserción, así que, cada prueba solo llama a un método de aserción. Algunos desarrolladores, pueden pensar que esto es una gran pérdida de espacio, sin embargo, seguir este principio puede además garantizar la reutilización de pruebas. Un ejemplo de un caso de prueba que evalúe más de una aserción es la siguiente:

```
<?php

  public function testIsMyString(){
    $string = "Muy inofensivo";
    $this->assertGreaterThan(0,strlen($string));
    $this->assertContains("42",$string);
 }
?>
```

En este ejemplo, la prueba muestra comprueba dos cosas diferentes. En primer lugar, comprueba si la cadena está vacía (longitud mayor a 0) y luego comprueba si la cadena contiene el número "42". De inmediato, podemos ver cómo esto se torna difícil: La prueba podría fallar si la cadena fuera, por ejemplo, "cuarenta y dos". Pero, veríamos exactamente el mismo error si la cadena estuviera vacía, lo cual podría ser causado por un error totalmente diferente. En este caso, el "error" resultante podría ser engañoso y podría causar cierta confusión en cuanto a lo que realmente está reportando.

Algunos de los marcos de trabajo para construir aplicaciones PHP, tales como Zend Framework ó Symphony incluyen la posibilidad de escribir pruebas para testar la funcionalidad de los módulos programados.

5.5.4 TENDENCIAS EN EL DESARROLLO DE PRUEBAS

Cuando se habla de pruebas, muchos desarrolladores las asocian con algunas de las últimas tendencias dentro de la informática, donde las pruebas están involucradas de manera directa. Aquí particularmente mencionaremos tres de estas tendencias: el desarrollo dirigido por pruebas, la integración continua y modelos de madurez orientados a modelos de pruebas.

El desarrollo basado en pruebas (*Test Driven Development* o TDD). TDD es un tipo de proceso de desarrollo de software basado en la premisa de que debemos escribir las pruebas, antes de escribir una sola línea de código de la aplicación. Aquí podemos preguntarnos, ¿cómo saber que escribir en la prueba si no se tiene el código de la aplicación a la vista? Ese es el punto. En TDD se escribe la prueba para comprobar la funcionalidad de lo previsto y luego se escribe el código que coincide exactamente con la prueba. TDD se basa en la realización de pruebas unitarias por lo que puede ser un método difícil de practicar para un novato en pruebas unitarias. Cuando se empieza en el TDD seguramente

el primer conjunto de pruebas fallarán. Sin embargo, al escribir el código de la aplicación, nuestro trabajo estará centrado en la prueba con lo que al final todos los casos de prueba serán exitosos. Este método permite centrarse más en los requerimientos teniéndolos siempre presentes, en vez de perderse en los detalles del código.

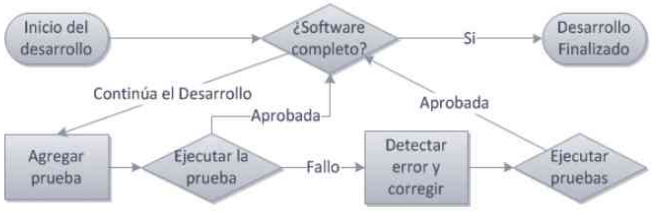


Figura 5.6. *Proceso de Desarrollo Dirigido por Pruebas*

Otra de las técnicas nuevas para facilitar la realización de pruebas es la integración continua. La integración continua (del inglés, *continuous integration*) es una metodología informática propuesta inicialmente por Martin Fowler (Fowler & Foemmel, 2005) que promueve la realización de integraciones automáticas de un proyecto de manera frecuente y lo más pronto posible para así poder detectar fallos cuanto antes. Entendemos por integración a la compilación y ejecución de test de todo un proyecto. Esta propuesta suele asociarse a las metodologías de programación extrema y desarrollo ágil. Un sistema de integración continua está compuesto por una serie de piezas dispuestas de forma que permitan realizar integraciones automáticas frecuentes del proyecto, con todas las ventajas que ello supone. Estas piezas que necesitamos son, como mínimo:

- Sistema de control de versiones (por ejemplo, Subversión SVN).
- Sistema de construcción de proyectos (por ejemplo, Phing).
- Servidor de integración continua (por ejemplo, Xinc para PHP).

Figura 5.7. *Modelo de integración continua*

Finalmente y coincidiendo con el auge de los modelos de calidad en ingeniería del software como CMMI (Modelo de Madurez de Capacidad Integrado) o ISO/IEC 15504 SPICE, se empiezan a escuchar otros modelos más especializados y enfocados a cierto tipo de organizaciones, como son, por ejemplo, los modelos de calidad para pruebas. Citando los más populares, podemos encontrar TMM (Modelo de Madurez de Pruebas - *Test Maturity Model*), TMMi (Modelo de Madurez de Pruebas Integrado - *Test Maturity Model Integration*) o TPI (Proceso de Mejoramiento de Pruebas - *Test Process Improvement*). TMMi es la evolución de TMM, creado por el Instituto de Tecnología de Illinois en 1996. Siguiendo la línea de CMMI, tanto TMM como TMMi definen cinco niveles de madurez específicos para pruebas. Actualmente, este modelo es desarrollado por la Fundación TMMi (*http://www.tmmifoundation.org*). Por su parte, TPI, cuya primera referencia es de 1999 (Koomen & Pol, 1999), ofrece 20 áreas de proceso, y su modelo de evaluación es más cercano a niveles de capacidad que a niveles de madurez. TMM nació en EEUU y TPI en Europa, como sucedió con CMMi e ISO 15504.

ACTIVIDADES 5.6

- Uno de los mejores *blogs* en español sobre nuevas tendencias, tanto en calidad de software como en el desarrollo de pruebas, es el dirigido por el Dr. Javier Garzás Parra (*http://www.javiergarzas.com/*). Consúltenlo.

RESUMEN DEL CAPÍTULO

En este capítulo hemos estudiado cómo establecer algunos de los requerimientos de seguridad y pruebas necesarios a la hora de implementar una aplicación web.

El mantenimiento del estado en una aplicación web permite almacenar información dada por el usuario a la aplicación. Existen dos formas de manejo del estado: por medio de *cookies* y por medio de sesiones. Las *cookies* son pequeños archivos que almacenan información en el cliente. Por otro lado, las sesiones son mecanismos manejados por el servidor que pueden tomar los datos enviados por el cliente a través de los métodos *POST* y *GET*.

Un aspecto complementario del manejo del estado es el control de usuarios. Una de las estrategias más difundidas para este control es la división entre usuarios, perfiles y roles. A nivel informático, esa estrategia puede ser implementada a través de un ACL o Lista de Control de Acceso. LDAP (Protocolo Ligero de Acceso al Servicio de Directorios) es una de las implementaciones que pueden apoyar a un ACL. LDAP es un directorio, similar a una lista de teléfonos, que organiza los objetos (usuarios, hardware) de forma jerárquica. Cada tipo de objeto se concretará a través de varias entradas que describen a través de un atributo a un usuario o hardware.

Otro aspecto importante a la hora de crear una aplicación web es la realización de pruebas que garanticen que el software desarrollado funciona correctamente. Existen diversos tipos de pruebas y, en muchos casos, existen entornos de pruebas que nos facilitan la realización de las mismas, como PHPUnit. Además, existen varias tendencias actuales que involucran al campo de las pruebas, como son el Desarrollo Dirigido por Pruebas (TDD) o la Integración Continua o los Modelos de Madurez de Pruebas.

TEST DE CONOCIMIENTOS

1 Seleccione cuál de las siguientes opciones contiene mecanismos de control de estados en la Web:
- a) Sesiones y *cookies*.
- b) Pruebas y LDAP.
- c) ACL y LDAP.
- d) Sesiones, *cookies* y ACL.

2 Si deseo crear una sesión en PHP, deberé escribir el siguiente código:
- a) `$HTTP_SESSION('Nombre_Sesión').`
- b) `$_SESSION('Nombre_Sesión').`
- c) `session().`
- d) `session_start().`

3 Acerca de las *cookies* podemos afirmar:
- a) Son soportadas por todos los navegadores.
- b) Son pequeños archivos que se guardan en el cliente.
- c) Algunas veces son bloqueadas por los navegadores.
- d) Todas las anteriores.

4 Las siglas ACL significan:
- a) Acceso Controlado a Listas.
- b) Lista de Control de Acceso.
- c) Lista de Acceso a Componentes.
- d) Acceso a Componentes de Listas.

5 Si hablamos de la autenticación universal, podemos afirmar que:
- a) Es una tendencia poco aceptada por los problemas de seguridad que una autenticación única puede generar.
- b) Las iniciativas que hay en este campo son de origen privado, por lo que hay que pagar para su utilización.
- c) Permite al usuario registrarse en un servicio provisto en Internet a través de su cuenta en otro servicio.
- d) Requiere una gran formación por parte de los desarrolladores.

6 ¿Cuál de las siguientes afirmaciones es falsa hablando de un sistema que contenga un directorio manejado por LDAP?
- a) El directorio posee un modelo de nombrado.
- b) La organización del directorio es jerárquica.
- c) Podemos afirmar que la información del directorio es exclusivamente de usuarios.
- d) Las entradas están constituidas por conjuntos de atributos.

7 Refiriéndonos a la ejecución de pruebas, señale cuál de las siguientes afirmaciones es correcta:
- a) Los casos de prueba tienen por objetivo probar el software.
- b) Es posible probar un software de manera exhaustiva.
- c) La realización de pruebas manuales es recomendable no importando el tamaño del proyecto.
- d) La documentación es parte fundamental del desarrollo de pruebas.

8 Acerca de los tipos de pruebas podemos afirmar que:
- a) Las pruebas de caja blanca permiten examinar la estructura del programa.
- b) Las pruebas funcionales se realizan con técnicas de caja negra.
- c) Las pruebas de unidad son para comprobar que el software funcione como un todo.
- d) El objetivo de las pruebas de volumen es calcular el número de líneas de código de un programa.

6 Utilización de técnicas de acceso a datos

OBJETIVOS DEL CAPÍTULO

- ✓ Aprender a establecer una conexión con la base de datos en los diferentes lenguajes.
- ✓ Estudiar la sintaxis necesaria para ejecutar las distintas sentencias SQL: inserción, borrado, actualización, selección, etc.
- ✓ Aprender a recuperar y utilizar la información de la base de datos.
- ✓ Estudiar las posibles transacciones que se pueden realizar en una base de datos utilizando los distintos lenguajes.
- ✓ Aprender los distintos niveles de aislamiento de una transacción entre las que destaca el modo serializable.
- ✓ Aprender a obtener la información de otros orígenes distintos a una base de datos como ficheros de texto, CSV y XML.

En el capítulo anterior hemos visto cosas referentes a las pruebas a las que hay que someter a las aplicaciones web durante su desarrollo, la autenticación de los usuarios y el control de sesiones entre otras muchas cosas. Sin embargo, en este capítulo vamos a ver cómo gestionar la comunicación de estas aplicaciones web con una base de datos, la cual la vamos a utilizar para almacenar toda la información. Esto conlleva aprender la sintaxis necesaria para conectarnos con la base de datos y la de las sentencias SQL de inserción, actualización, borrado y selección. También veremos cómo utilizar la información recuperada de la base de datos, cómo realizar transacciones y cómo acceder a la base de datos de manera serializable para que la base de datos no se quede en ningún momento inconsistente. Esto puede ocurrir debido a que la base de datos con frecuencia deberá soportar el acceso de varias personas a la vez. Por último explicaremos cómo obtener la información de otro tipo de fuentes que no sean la propia base de datos.

6.1 ESTABLECIMIENTO DE CONEXIONES

A la hora de interactuar con una base de datos lo primero que tenemos que hacer es establecer la conexión entre nuestra aplicación web y ésta para que posteriormente podamos ejecutar las sentencias SQL (*Structured Query Language*) que necesitemos de entre todo el abanico de sentencias que nos ofrece este lenguaje.

A continuación vamos a mostrar la sintaxis necesaria en los distintos lenguajes para conectarnos a una base de datos utilizando distintos gestores de base de datos. En el caso de PHP solo explicaremos las funciones para MySQL, pues para cada gestor de base de datos cuenta con unas funciones distintas.

Con respecto a PHP, la última versión cuenta con dos conjuntos de funciones: una de ellas son las funciones `mysqli`, que se usan con MySQL 4.1 y posteriores y las otras son las funciones `mysql`, que se usan con la versión de MySQL 4.0 y anteriores.

```
$conector=mysqli_connect($host, $user, $password [,
dbname]);

$conector=mysql_connect($host, $user, $password);
mysql_select_db($dbname);
```

Como podemos ver, las funciones `mysqli` y `mysql` son prácticamente iguales, ya que las funciones `mysqli` son una extensión o mejora de las otras y lo que se pretendió es que fueran lo más parecidas posibles para que la migración de la versión antigua a la nueva fuera casi inmediata. Más concretamente, para conectarnos con la base de datos tenemos que facilitar el servidor donde se encuentra la base de datos (`$host`), el usuario (`$user`) y la contraseña (`$password`) de la base de datos. El servidor se puede especificar facilitando el nombre del servidor, seguido del puerto (`nombre_host:puerto`) o mediante una ruta para un servidor local (`:ruta`). Con respecto a la función `mysqli_connect()` podemos definir opcionalmente que base de datos del servidor queremos utilizar o no especificarlo y hacerlo más tarde (`mysqli_select_db($dbname)`) como hacemos con las funciones `mysql`. Con la funciones de `mysql` primero hay que conectarse al servidor (`mysql_connect()`) y después seleccionar la base de datos (`mysql_select_db($dbname)`) pasándole como parámetro el nombre de la base de datos.

¿SABÍAS QUE...?

Para utilizar en PHP base de datos Oracle deberemos utilizar las funciones del paquete OCI8, para SQL Server las del paquete MySQL y, por último, para Access las del paquete ODBC.

A continuación vamos a mostrar como conectar una aplicación web escrita en JSP a una base de datos MySQL:

```
<%@page import="java.sql.*" %>
<%
  Connection conexion=null;
  Statement st=null;
  Class.forName("org.gjt.mm.mysql.Driver");
  conexion=DriverManager.getConnection(
    "jdbc:mysql://localhost/directorios...",
    "nombre_usuario","contraseña");
%>
```

Lo que tenemos que hacer antes que nada en JSP para conectarlo con una base de datos es importar el paquete `java.sql.*`, que contiene todos los métodos y clases relacionados con el manejo de las base de datos. Posteriormente se crea una instancia de los drivers específicos de la base de datos con la que vamos a trabajar. En el ejemplo que presentamos arriba se realiza una conexión con una base de datos MySQL. Una vez creada la nueva instancia de los drivers que necesitamos, los utilizamos para conectarnos con la base de datos haciendo uso del método `DriverManager.getConnection()`. Este método recibe como parámetros una URL y opcionalmente podemos especificar el nombre de usuario y la contraseña de la base de datos a la que nos queremos conectar. La URL representa la información de la base de datos concreta a la que nos vamos a conectar que presenta la siguiente estructura: `jdbc:protocolo://localhost/ruta_BD`.

Si queremos en JSP conectarnos a otras base de datos tendríamos que cambiar la siguiente sentencia:

- **SQL Server**:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver
  ");
Connection conexion=DriverManager.getConnection(
  "dbc:sqlserver://localhost;databaseName=ruta_BD",
  "nombre_usuario","contraseña");
```

- **MS Access**:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conexion=DriverManager.getConnection
  ("jdbc:odbc:Driver={Microsoft Access Driver
  (*.mdb)};DBQ=ruta_BD");
```

- **Oracle**:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conexion =
 DriverManager.getConnection("jdbc:oracle:thin:@miservidor:
 puerto:esquema","usuario","contraseña");
```

Por otro lado, para conectarnos a una base de datos MySQL en ASP la sintaxis es la siguiente:

```
<%
  Dim conn
  Set conexion=Server.CreateObject("ADODB.connection")
  conexion.open "Driver={MySQL}; Server=nombre_servidor;
  Port=puerto; Database=Nombre_db; Uid=Nombre_usuario;
  Pwd=contraseña"
%>
```

Como vemos en el ejemplo para conectarnos en ASP a una base de datos MySQL, creamos una variable sobre la que llamaremos al método `server.CreateObject()`. Este método crea una instancia de un objeto del servidor. Después debemos crear la conexión especificando los *drivers* que se van a utilizar para ello, esto es lo que deberemos cambiar si en vez de conectarnos a una base de datos MySQL queremos conectarnos a otro tipo de base de datos. También podemos especificar el puerto, el nombre de la base de datos, el usuario y contraseña. Por último, también podremos especificar la ruta del servidor que contiene la base de datos (Si el acceso es a una base de datos local lo indicaremos poniendo `Server=localhost`).

Si queremos conectarnos a otra base de datos lo único que cambiaría sería la siguiente sentencia:

- **SQL Server**:

```
conexion.open "Driver={SQL Server}; Server=nombre_servidor;
  Database=Nombre_db; Uid=Nombre_usuario; Pwd=contraseña"
```

- **Oracle**:

```
conexion.Open "Provider=MSDAORA.1;Password=contaseña;User
  ID=nombre_usuario;Data Source=data.world"
```

- **MS Access**:

```
conexion.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data
  Source= nombre_BD
```

6.2 EJECUCIÓN DE SENTENCIAS SQL (STRUCTURED QUERY LANGUAGE)

Una vez que hemos logrado conectar la base de datos a nuestra aplicación, ya estamos preparados para ejecutar cualquier tipo de sentencias SQL. Más tarde aprenderemos a utilizar los resultados obtenidos tras la ejecución de estas sentencias.

Dentro de las sentencias SQL que podemos ejecutar, existen tres tipos de sentencias: las sentencias de definición de datos (*Data Definition Language*), las sentencias de manipulación de datos (*Data Manipulation Language*) y las sentencias de control (*Data Control Language*).

Antes de pasar a especificar la sintaxis de las distintas sentencias SQL vamos a explicar lo que tenemos que hacer para preparar la ejecución de una sentencia SQL en los distintos lenguajes.

En primer lugar en PHP existe el método `mysql_query()` que ejecuta las consultas en la base de datos y devuelve un valor booleano o un objeto. Este método devolverá un valor booleano en el caso de ejecutar sentencias SQL que no devuelvan información al usuario, como por ejemplo: la inserción, el borrado de datos, etc. En estos casos el método devuelve únicamente un valor booleano que informa de si la ejecución de la consulta se realizado de forma satisfactoria (`true`) o por el contrario se ha producido algún error (`false`). Por otro lado, este método a veces devuelve un objeto resultado (`ResultSet`) que contiene el resultado de ejecutar la sentencia. Este es el caso de ejecutar sentencias como las de selección, etc.

```
$result=mysql_query($sentencia[,$conector]);
$result=mysqli_query($conector,$sentencia[,$modo_resultado]
  );
```

Para utilizar el método `mysql_query()` necesitamos pasarle como parámetro un *string* que contenga la sentencia SQL a ejecutar en la base de datos y de forma opcional el conector de la base de datos. Cuando hablamos del conector de la base de datos nos referimos al valor resultante que obtenemos de llamar a la función `mysql_connect()` o `mysqli_connect()`, cuyo valor tenemos que almacenarlo en una variable para posteriormente poder usarlo. Si por el contrario necesitamos usar la función `mysqli_query()` aparte de pasarle como parámetro la sentencia SQL y el conector (en este caso de forma obligatoria), también acepta como parámetro una constante que puede tomar los siguientes valores: `MYSQLI_STORE_RESULT` y `MYSQLI_USE_RESULT` (si no se especifica toma por defecto `MYSQLI_USE_RESULT`).

En JSP para ejecutar una sentencia SQL en la base de datos es necesario crear un objeto `Statement` que se usa para enviar la sentencia SQL a la base de datos. Una vez creado el objeto `Statement` con el método `createStatement()` a través del conector de la base de datos pasamos a ejecutar la sentencia SQL. Para ello hay que llamar al método `executeQuery()` para ejecutar consultas de selección (`SELECT`) o `executeUpdate()` para ejecutar actualizaciones, inserciones o borrados (`DELETE`, `UPDATE` o `INSERT`). Ambos métodos deben llamarse a partir del objeto `Statement` y pasarle como parámetro un *string* que contenga la sentencia SQL. El resultado de ejecutar la sentencia se guardará en un objeto `ResultSet`.

```
st=conexion.createStatement();
ResultSet rs= st.executeQuery(sentencia);

int n=st.executeUpdate(sentencia);
```

Por último en ASP, lo único que necesitamos es llamar al método `execute()` a través del conector de la base de datos y pasarle al método como parámetro la sentencia que queremos ejecutar. El resultado lo almacenaremos en una variable para posteriormente poder utilizarlo (`RecordSet`).

```
Set rs = conexion.Execute(sentencia)
```

En los subapartados siguientes mostraremos la sintaxis de las distintas sentencias SQL que podemos ejecutar en una base de datos. Vamos a dividir este apartado en tres subapartados: uno para las sentencias de definición de datos, otra para las sentencias relacionadas con la manipulación de datos y otra para las sentencias de control.

6.2.1 SENTENCIAS DE DEFINICIÓN DE DATOS (DDL, DATA DEFINITION LANGUAGE)

En este subapartado presentamos la sintaxis de las distintas sentencias de definición de datos para crear, modificar y borrar una base de datos o una tabla.

SENTENCIA PARA CREAR UNA BD

La sintaxis para crear una base de datos es la siguiente:

```
CREATE DATABASE nombre_db
```

Para crear una base de datos hay que poner las palabras reservadas `CREATE DATABASE` seguido del nombre que queramos darle a la base de datos.

SENTENCIA PARA BORRAR UNA BD

La sintaxis para borrar una base de datos es la siguiente:

```
DROP DATABASE nombre_db
```

Para borrar una base de datos hay que poner `DROP DATABASE` seguido del nombre de la base de datos que deseamos borrar.

SENTENCIA PARA CREAR UNA TABLA

La sintaxis para crear una tabla en la base de datos es la siguiente:

```
CREATE TABLE nombre_tabla(
nombre_columna1 tipo1,
nombre_columna2 tipo2,
...
nombre_columnaN tipoN)
```

Como podemos, ver para crear una nueva tabla en la base de datos solo necesitamos poner `CREATE TABLE` seguido del nombre que queramos que tenga. Entre paréntesis hay que especificar los campos que deseemos que tenga nuestra tabla. Para ello, tendremos que poner el nombre de cada columna seguido de su tipo de datos. La definición de cada columna se separará de la siguiente utilizando comas (,).

SENTENCIA PARA BORRAR UNA TABLA

Para borrar una tabla de la base de datos la sintaxis es la siguiente:

```
DROP TABLE nombre_tabla
```

Si lo que queremos es borrar una tabla, lo único que tenemos que hacer es crear un *string* en el que pongamos `DROP TABLE` seguido del nombre de la tabla que queremos borrar.

SENTENCIA PARA BORRAR EL CONTENIDO DE UNA TABLA

A diferencia del subapartado anterior si lo que deseamos no es borrar la tabla en sí misma si no los datos que esta contiene deberemos utilizar la sentencia `TRUNCATE` como podemos ver a continuación:

```
TRUNCATE TABLE nombre_tabla
```

Como podemos ver es tan sencillo como poner `TRUNCATE TABLE` seguido del nombre de la tabla.

SENTENCIA PARA MODIFICAR UNA TABLA

Esta sentencia se utiliza para modificar la estructura de una tabla, ya sea para añadir, modificar o eliminar columnas.

La sintaxis es la siguiente:

```
ALTER TABLE nombre_tabla opciones
```

Para modificar la estructura de una tabla solo necesitamos escribir `ALTER TABLE` seguido del nombre de la tabla sobre la que queremos realizar los cambios y las opciones. Las opciones representan todos los posibles cambios que podemos realizar sobre una tabla. Las opciones que podemos utilizar son las siguientes:

- **Añadir una columna a una tabla**. Podemos añadir una columna nueva en una tabla con `ADD COLUMN` seguido del nombre de la nueva columna y el tipo de datos.

  ```
  ADD COLUMN nombre_columna tipo
  ```

- **Eliminar una columna de una tabla**. Elimina la columna cuyo nombre hemos especificado. Para ello la sintaxis es la siguiente:

  ```
  DROP COLUMN nombre_columna
  ```

- **Modificar una columna**. Cuando modificamos una columna de una tabla nos referimos a modificar su tipo de datos. Para modificarlo solo hay que escribir `ALTER COLUMN` seguido del nombre de la columna y su nuevo tipo de datos.

  ```
  ALTER COLUMN nombre_column tipo_nuevo
  ```

AÑADIR RESTRICCIONES A UNA TABLA

Al definir una tabla podemos además agregarle una serie de restricciones para limitar los tipos de datos que pueden tener las columnas. Las restricciones las podemos definir al crear una tabla o una vez creada con la sentencia `ALTER TABLE`. A continuación mostraremos los tipos de restricciones que existen en SQL:

- **NOT NULL**. Esta restricción índica que la columna es obligatoria y no puede contener un valor nulo. Esto quiere decir que cuando queramos insertar o actualizar un registro en una tabla es necesario asignarle un valor a este campo. Como podemos ver, para definir esta restricción tenemos que poner `NOT NULL` a continuación del tipo de datos de la columna que no queramos que acepte valores nulos justo en el momento en el que estamos definiendo la estructura de la tabla.

  ```
  CREATE TABLE nombre_tabla(
    nombre_columna1 tipo1 NOT NULL,
    nombre_columna2 tipo2,
    ...
    nombre_columnaN tipoN
  )
  ```

- **UNIQUE**. Esta restricción garantiza la unicidad de una columna o conjunto de columnas e identifica de forma única cada registro dentro de la base de datos. Para que una columna de una tabla sea `UNIQUE` debe ser también `NOT NULL`. Un ejemplo de campo único en la vida cotidiana sería el DNI que nos identifica de forma unívoca a cada uno de nosotros o la matrícula de un vehículo.

Podemos ver la sintaxis para añadir esta restricción para las distintas base de datos:

✓ **MySQL**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  UNIQUE(nombre_columna1)
)
```

✓ **SQL Server/Oracle/MS Access**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL UNIQUE,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN
)
```

✓ **MySQL/SQL Server/Oracle/MS Access**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  CONSTRAINT nombre_restricción UNIQUE(nombre_columna1,...)
)
```

En MySQL una de las formas posibles formas de definir esta restricción cuando solo se le añade a una columna es agregándola en el momento en el que definimos la tabla detrás de la última columna que esta contenga. Para ello hay que poner `UNIQUE` y entre paréntesis el nombre de las columna que queramos que posean esta restricción.

El resto de las base de datos definen esta restricción cuando solo se refiere a una columna al definir la columna. La columna que queramos que sea `UNIQUE` se definirá poniendo el nombre de la columna, seguido del tipo de dato de la columna, `NOT NULL` (ya que para ser `UNIQUE` no debe aceptar valores nulos) y por último `UNIQUE`.

Existe otra forma de definir esta restricción común a todas las bases de datos que se utiliza cuando queremos añadirla a múltiples columnas. Esta forma consiste en añadir detrás de la definición de la última columna de la tabla `CONSTRAINT` seguido del nombre que le queramos dar a la restricción, `UNIQUE` y entre paréntesis el nombre de todas las columnas que queramos que tenga esta restricción.

Una vez que la tabla ya ha sido creada la sintaxis para añadir esta restricción varía:

– **Forma 1**:

```
ALTER TABLE nombre_tabla
ADD UNIQUE(nombre_columna1)
```

– **Forma 2**:

```
ALTER TABLE nombre_tabla ADD CONSTRAINT nombre_restricción
UNIQUE(nombre_columna1,...)
```

Las dos formas son válidas para cualquier base de datos para definir esta restricción una vez que ya ha sido creada la tabla.

- **PRIMARY KEY**. Cada tabla de una base de datos debe contener una y solo una clave primaria. Esta clave primaria no puede contener valores nulos. Los valores de esta clave son únicos e identifican a los registro de una tabla de la base de datos de forma unívoca. A diferencia de la restricción `UNIQUE`, puede haber varias columnas `UNIQUE` pero solo una que sea `PRIMARY KEY`. Habitualmente cuando no hay ninguna columna que pueda identificar una tabla, se usa como `PRIMARY KEY` un índice que se incrementa de forma automática y que actúa como identificador de la tabla.

  ✓ **MySQL**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  PRIMARY KEY(nombre_columna1)
)
```

  ✓ **SQL Server/Oracle/MS Access**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL PRIMARY KEY,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN
)
```

  ✓ **MySQL/SQL Server/Oracle/MS Access**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  CONSTRAINT nombre_PK PRIMARY KEY (nombre_columna1,...)
)
```

La forma de definir esta restricción para indicar que una columna va a ser una `PRIMARY KEY` es exactamente igual que la restricción `UNIQUE`. Existe dos formas para definirla cuando se refiere únicamente a una columna. Una de ellas es solo válida para MySQL que es añadiendo la cláusula `PRIMARY KEY` y entre paréntesis la columna que queramos que sea la clave primaria de la tabla al final de todas las columnas. La otra posibilidad es añadirla en la definición de la columna, después de indicar el nombre, el tipo, `NOT NULL` (para que la columna sea `PRIMARY KEY` es necesario que no acepte valores nulos) y en último lugar `PRIMARY KEY`. Esta última opción es posible en todas las bases de datos excepto en MySQL.

Por otro lado, si lo que deseamos es crear una sola clave primaria formada por la unión de los valores que toman varias columnas necesitamos escribir `CONSTRAINT` seguido del nombre que queramos poner a la clave primaria y entre paréntesis el nombre de las columnas implicadas separadas por comas.

Si necesitamos añadir esta restricción una vez que la tabla ya ha sido creada, lo haremos de la siguiente manera:

- **Forma 1**:

```
ALTER TABLE nombre_tabla
ADD PRIMARY KEY(nombre_columna1)
```

- **Forma 2**:

```
ALTER TABLE nombre_tabla ADD CONSTRAINT nombre_PK
PRIMARY KEY(nombre_columna1,...)
```

La primera de ellas es para crear una clave primaria en una tabla con el valor de una columna y la segunda crea una clave primaria con el valor conjunto de varias columnas. Ambas formas son válidas para todas las base de datos.

- **FOREIGN KEY**. Esta restricción se usa para relacionar la información de una tabla con la de otra distinta. A continuación mostramos la sintaxis para implementar esta restricción en las distintas base de datos posibles:

✓ **MySQL**:

```
CREATE TABLE nombre_tabla1(
  nombre_columna1 tipo1,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  FOREIGN KEY(nombre_columna) REFERENCES
  nombre_tabla2(PK_tabla2)
)
```

✓ **SQL Server/Oracle/MS Access**:

```
CREATE TABLE nombre_tabla1(
  nombre_columna1 tipo1,
  nombre_columna2 tipo2 FOREIGN KEY REFERENCES
  nombre_tabla2(PK_tabla2),
  ...
  nombre_columnaN tipoN
)
```

✓ **MySQL/SQL Server/Oracle/MS Access**:

```
CREATE TABLE nombre_tabla1(
  nombre_columna1 tipo1 NOT NULL,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  CONSTRAINT nombre_FK FOREIGN KEY
  (nombre_columna) REFERENCES nombre_tabla2(PK_tabla2,...)
)
```

Para definir una columna como clave ajena de otra tabla existen distintas formas. MySQL sigue la misma dinámica para definir el resto de restricciones que anteriormente hemos visto. Solo necesitamos poner después de la última columna `FOREIGN KEY` seguido del nombre de la columna que queremos que se comporte como

clave ajena. Después pondremos `REFERENCES` seguido del nombre de la tabla a la que hace referencia y entre paréntesis la clave primaria (PK) de la misma (la tabla referenciada). En el resto de base de datos definen esta restricción en el momento justo en el que se está definiendo la columna que va a realizar la función de clave ajena. En este caso no hace falta especificar el nombre de la columna antes de la palabra reservada `REFERENCES`, puesto que como está en la misma línea en la que se define la columna se entiende a cuál nos referimos.

Por último, existe una sintaxis común a todas para definir una `FOREIGN KEY` compuesta por varias columnas de la tabla ajena. Para ello, utilizamos `CONSTRAINT`, el nombre de la clave ajena, `FOREIGN KEY`, el nombre de la columna de la tabla actual que hace de clave ajena, `REFERENCES`, el nombre de la tabla a la que hace referencia y entre paréntesis los campos de esta a los que hace referencia la clave ajena.

Si necesitamos definir claves ajenas una vez ya creadas las tablas lo haremos de la siguiente forma:

- **Forma 1**:

```
ALTER TABLE nombre_tabla1
ADD FOREIGN KEY(nombre_columna)REFERENCES
nombre_tabla2(PK_tabla2)
```

- **Forma 2**:

```
ALTER TABLE nombre_tabla ADD CONSTRAINT nombre_FK
FOREIGN KEY(nombre_columna)REFERENCES
nombre_tabla2(PK_tabla2,...)
```

La primera forma es para crear una clave ajena que referencia a una columna de otra tabla y la segunda la utilizaremos en el caso de que tengamos que crear una clave ajena formada por varias columnas. La sintaxis de las dos formas que facilitamos es válida para todas las base de datos.

- **DEFAULT**. Con esta restricción podemos asignarle un valor por defecto a una columna de una tabla. Este será el valor que se le añada a los nuevos campos si no especificamos su valor.

La sintaxis para especificar el valor que tomará por defecto una columna es la siguiente:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1,
  nombre_columna2 tipo2 DEFAULT valor,
  ...
  nombre_columnaN tipoN
)
```

Como podemos ver, para definir el valor por defecto de una columna en cualquier base de datos solo hace falta añadir al final de la definición de la misma `DEFAULT` y el valor que está toma. En el caso de tratarse de una columna de tipo `VARCHAR` es necesario expresar el valor entre comillas simples (').

En el caso de tener que definir el valor por defecto de una columna una vez creada la tabla utilizaremos esta sintaxis:

- ✓ **MySQL**:

```
ALTER TABLE nombre_tabla
ALTER nombre_columna SET DEFAULT valor
```

- ✓ **SQL Server/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla
ALTER COLUMN nombre_columna SET DEFAULT valor
```

- **CHECK**. Por último, esta restricción limita el rango de valores que puede tomar una determinada columna de una tabla.

  A continuación presentamos la sintaxis para definir restricciones que limitan los valores que puede tomar una determinada columna:

  ✓ **MySQL**:

  ```
  CREATE TABLE nombre_tabla(
    nombre_columna1 tipo1,
    nombre_columna2 tipo2,
    ...
    nombre_columnaN tipoN,
    CHECK(condición)
  )
  ```

  ✓ **SQL Server/Oracle/MS Access**:

  ```
  CREATE TABLE nombre_tabla(
    nombre_columna1 tipo1 CHECK(condición),
    nombre_columna2 tipo2,
    ...
    nombre_columnaN tipoN
  )
  ```

  ✓ **MySQL/SQL Server/Oracle/MS Access**:

  ```
  CREATE TABLE nombre_tabla(
    nombre_columna1 tipo1,
    nombre_columna2 tipo2,
    ...
    nombre_columnaN tipoN,
    CONSTRAINT nombre_restricción CHECK(condición1 AND ...)
  )
  ```

  La primera de ellas define esta restricción en MySQL añadiendo `CHECK` seguido de la condición. La segunda de ellas es igual que en MySQL solo que en vez de definirlo después de todas las columnas, tenemos que definirla sobre la columna que debe cumplir la condición. Esta es válida para todas las bases de datos excepto para MySQL. Por último, existe una forma válida para todas las base de datos que se utiliza cuando se establecen restricciones en varias columnas.

  Para definir esta restricción una vez que la tabla ya ha sido creada, la sintaxis es la siguiente:

  – **Forma 1**:

  ```
  ALTER TABLE nombre_tabla
  ADD CHECK(condición)
  ```

  – **Forma 2**:

  ```
  ALTER TABLE nombre_tabla
  ADD CONSTRAINT nombre_restricción CHECK(condición1 AND …)
  ```

  La primera forma es para definirlo sobre una columna y la segunda sobre varias. Ambas válidas para todas las base de datos.

- **AUTO_INCREMENT**. Permite enumerar de manera automática cada registro que se inserta en una tabla. Suele utilizarse junto con `PRIMARY KEY` para crear una clave unívoca.

  ✓ **MySQL**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 NOT NULL AUTO_INCREMENT,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
  PRIMARY KEY(nombre_columna1)
)
```

  ✓ **SQL Server**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 PRIMARY KEY IDENTITY,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
)
```

  ✓ **MS Access**:

```
CREATE TABLE nombre_tabla(
  nombre_columna1 tipo1 PRIMARY KEY AUTOINCREMENT,
  nombre_columna2 tipo2,
  ...
  nombre_columnaN tipoN,
)
```

  ✓ **Oracle**:

```
CREATE SEQUENCE nombre_seq MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10
```

  En Oracle vemos que es algo más complejo, hay que crear una secuencia en el que se especifica el valor de inicio, el valor de fin y de cuánto va a ser el incremento. Para usar esta secuencia hay que hacer lo siguiente:

```
INSERT INTO nombre_tabla (nombre_columna1,...)VALUES
  (nombre_secuencia.nextval,valor2,...)
```

ELIMINAR RESTRICCIONES DE UNA TABLA

Una vez creada la base de datos puede darse el caso de que con el tiempo tengamos la necesidad de modificar la base de datos o incluso que necesitemos eliminar algunas de las restricciones que hubiéramos definido en un principio. Vamos a explicar cómo eliminar cada una de las restricciones que hemos explicado con anterioridad.

- **UNIQUE**. Para eliminar la restricción `UNIQUE` tenemos que utilizar la sintaxis siguiente:

  ✓ **MySQL**:

```
ALTER TABLE nombre_tabla DROP INDEX nombre_columna
```

  ✓ **SQL Server/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla DROP CONSTRAINT nombre_columna
```

- **PRIMARY KEY**. La sintaxis para eliminar esta restricción es la siguiente:
  - ✓ **MySQL**:

```
ALTER TABLE nombre_tabla DROP PRIMARY KEY
```

  - ✓ **SQL Server/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla DROP CONSTRAINT nombre_columna
```

- **FOREIGN KEY**. Para eliminar una clave ajena hay que hacer lo siguiente:
  - ✓ **MySQL**:

```
ALTER TABLE nombre_tabla DROP FOREIGN KEY nombre_columna
```

  - ✓ **SQL Server/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla DROP CONSTRAINT nombre_columna
```

- **DEFAULT**. La sintaxis para eliminar el valor por defecto que pueda tener cualquier columna de una tabla es la siguiente:
  - ✓ **MySQL/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla
ALTER nombre_columna DROP DEFAULT
```

  - ✓ **SQL Server/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla
ALTER COLUMN nombre_columna DROP DEFAULT
```

- **CHECK**. Para eliminar cualquier limitación con respecto al rango de valores que puede tomar una columna dentro de una tabla la sintaxis es la siguiente para las distintas base de datos:
  - ✓ **SQL Server/Oracle/MS Access**:

```
ALTER TABLE nombre_tabla
DROP CONSTRAINT nombre_restricción
```

CREAR VISTAS EN UNA BD

Una vista en el contexto de las bases de datos, no es más que una tabla virtual que contiene el resultado de la ejecución de una consulta de selección. Los campos que componen una vista son campos que pertenecen a uno o más tablas reales de la base de datos.

A continuación presentamos la sintaxis para definir una vista:

```
CREATE VIEW nombre_vista AS SELECT nombre_columna1, ...
FROM nombre_tabla WHERE condición
```

Como podemos ver, solo hace falta poner `CREATE VIEW` seguido del nombre que queramos darle a la vista, `AS SELECT` el nombre de las distintas columnas que queremos que formen parte de la vista, `FROM` el nombre de la tabla a la que pertenecen dichas columnas y por último `WHERE` y la condición que las columnas deben cumplir para incluirse en la vista.

ELIMINAR UNA VISTA EN UNA BD

Para eliminar una vista creada por nosotros la sintaxis es mucho más sencilla todavía. Para ello solo debemos escribir `DROP VIEW` seguido del nombre de la vista que queramos borrar:

```
DROP VIEW nombre_vista
```

CREAR UN ÍNDICE EN UNA TABLA

Los índices de las tablas de una base de datos permiten realizar búsquedas o consultas dentro de estas de manera más rápida y eficaz. Tenemos que tener en cuenta que la actualización de una tabla con índices tarda más tiempo que una tabla que no los tenga, ya que no solo tenemos que actualizar esta, sino que también deben actualizarse los índices. Por este motivo, es necesario utilizar índices solo en las tablas que sepamos que los vamos a utilizar.

La sintaxis para definir estos índices son las siguientes:

- **Forma 1**:

```
CREATE INDEX nombre_índice ON TABLE
nombre_tabla(nombre_columna)
```

- **Forma 2**:

```
CREATE UNIQUE INDEX nombre_índice ON TABLE
nombre_tabla(nombre_columna)
```

Existen dos formas para definir estos índices. La primera de ellas permite crear índices que pueden tener valores duplicados, mientras que la segunda no los acepta.

ELIMINAR UN ÍNDICE DE UNA TABLA

Al igual que el resto de propiedades o elementos que tiene una base de datos podemos eliminar los índices creados para una tabla. Para ello presentamos la sintaxis que tenemos que utilizar según la base de datos con la que estemos trabajando para poder eliminarlos:

- **MS Access**:

```
DROP INDEX nombre_índice ON nombre_tabla
```

- **SQL Server**:

```
DROP INDEX nombre_tabla.nombre_índice
```

- **DB2/Oracle**:

```
DROP INDEX nombre_índice
```

- **MySQL**:

```
ALTER TABLE nombre_tabla DROP INDEX nombre_índice
```

SENTENCIAS DE CONTROL DE DATOS (DCL, DATA CONTROL LANGUAGE)

Dentro de esta subsección explicaremos las sentencias de control de acceso, que sirven para otorgar o eliminar permisos.

AÑADIR PERMISOS

A veces es necesario diferenciar a los usuarios en distintos grupos y asignarles ciertos permisos a estos grupos dependiendo de la actividad que estos realicen. De manera que solo el personal autorizado para realizar ese tipo de operaciones pueda hacerlo permitiendo tener un control mayor sobre quien accede y manipula la base de datos, la cual puede contener información muy valiosa y delicada perteneciente a una empresa u otra organización.

La sintaxis básica para agregar permisos es la siguiente:

```
GRANT privilegio1,... [(nombre_columna,...)] ON
  nombre_objeto TO {nombre_usuario| PUBLIC | nombre_rol}
```

Como podemos ver, para definir permisos necesitamos la sentencia `GRANT` seguido del nombre de los privilegios que queramos. Aunque existen más privilegios, los más utilizados suelen ser: `all`, `select` (consulta la BD), `insert` (añadir registros), `delete` (borrado de registros) y `update` (actualización de registros). También es posible que en un momento determinado nos surja la necesidad de declarar ciertos privilegios sobre algunas columnas de una tabla y no sobre la tabla entera. Para ello tendríamos que poner entre paréntesis el nombre de las columnas de la tabla separadas por comas, situadas entre el último privilegio que queramos otorgarle y la palabra `ON`. Después de la palabra `ON` tendremos que escribir el nombre del objeto sobre el que se dan los permisos. Generalmente suelen ser tablas, pero pueden ser vistas, etc. Por último, se escribiría `TO` y a continuación a quien se le da los permisos, que puede ser a un usuario/s, a un role/s o a todos (`PUBLIC`).

ELIMINAR PERMISOS

Al igual que es posible agregar permisos a los elementos de una base de datos también es posible eliminar estos. Ya que puede que cierto usuario que tiene una serie de privilegios sobre la base de datos con el tiempo puede que sea conveniente que no los tenga.

Para eliminar los permisos la sintaxis básica es la siguiente:

```
REVOKE privilegio1,... [(nombre_columna,...)] ON
  nombre_objeto FROM {nombre_usuario/s| PUBLIC |
  nombre_rol/es}
```

La sintaxis es prácticamente igual a la que se utiliza para agregar privilegios con la salvedad de que hay que escribir `REVOKE` en vez de `GRANT`, y `FROM` y no `TO` antes de escribir las personas o grupos a los que se les quita los permisos.

6.2.3 SENTENCIAS DE MANIPULACIÓN DE DATOS (DML, DATA MANIPULATION LANGUAGE)

Una vez que ya sabemos cómo crear la base de datos y cómo administrar los permisos de esta, debemos aprender a utilizar la base de datos, almacenar información nueva, actualizar el contenido de los campos pertenecientes a las tablas, borrar el contenido de los registros y por último a consultar la base de datos para obtener la información que necesitemos en un momento dado.

INSERCIÓN DE DATOS EN LAS TABLAS DE LA BD

Una vez creada la base de datos con sus respectivas tablas se debe realizar la carga de datos, para inicializar las tablas con los valores que queremos almacenar en ellas. Para ello, adjuntamos la sintaxis necesaria para realizar la inserción de datos en las tablas de una base de datos:

- **Forma 1**:

```
INSERT INTO nombre_tabla VALUES(valor1, valor2,...)
```

- **Forma 2**:

```
INSERT INTO nombre_tabla
  (nombre_columna1,nombre_columna2,...) VALUES(valor1,
valor2,...)
```

Existen dos formas de insertar los valores en la tabla: la primera solo especificando el valor que tomarán las columnas y la segunda en el que además se especifica el nombre de las columnas.

REALIZACIÓN DE CONSULTAS EN LA BD

Para obtener información de la base de datos es necesario realizar consultas de selección sobre ella. Existen distintos operadores que nos permiten realizar consultas de selección muy variadas y que nos facilitan la tarea a la hora de trabajar con las bases de datos. A continuación se presenta la sintaxis de varios de ellos:

- **SELECT...FROM**. Consulta el valor de determinadas columnas de una tabla:

```
SELECT nombre_columna1,... FROM nombre_tabla
```

También es posible consultar una tabla en su totalidad con la ayuda del símbolo "*":

```
SELECT * FROM nombre_tabla
```

- **DISTINCT**. Consulta solo los valores de las columnas especificadas de una tabla, cuyo valor no se repite:

```
SELECT DISTINCT nombre_columna1,... FROM nombre_tabla
```

Tenemos que recordar que si quisiéramos consultar el valor de los campos no repetidos de todas las columnas de una tabla utilizaríamos el símbolo "*".

- **WHERE**. Consulta los campos de una columna que cumplan cierta condición:

```
SELECT nombre_columna1,... FROM nombre_tabla WHERE
  nombre_columna operador valor
```

Para definir la condición podemos utilizar los siguientes operadores: = (igual), <> (distinto), > (mayor que), < (menor que), >= (mayor o igual), <= (menor o igual), `BETWEEN`, `IN` y `LIKE`. Estas tres últimas la explicaremos más adelante. El valor que pongamos en la condición tiene que ser acorde con el tipo de dato con el que está definida la columna que vamos a utilizar en la condición.

La condición no tiene porque ser simple sino que podemos definir condiciones compuestas utilizando los operadores lógicos `AND` y `OR`. Para negar la condición utilizaremos el operador `NOT`.

- **ORDER BY**. Consulta los campos especificados de una tabla devolviendo el resultado ordenado:

```
SELECT nombre_columna1,... FROM nombre_tabla ORDER BY
  nombre_columna1,... [ASC | DESC]
```

Esta consulta, como hemos dicho antes, devuelve el resultado de una consulta de selección ordenado de forma ascendente (`ASC`) o descendente (`DESC`). Por defecto la cláusula `ORDER BY` si no se le especifica si tiene que ordenarlo de manera ascendente o descendente, lo hace de manera ascendente.

- **UNION**. Une el resultado de realizar varias consultas de selección:

```
SELECT nombre_columna1,... FROM nombre_tabla1
  UNION SELECT nombre_columna1,... FROM nombre_tabla2
```

- **INNER JOIN**. Devuelve las filas cuando al menos existe una coincidencia en ambas tablas.

```
SELECT nombre_columna1,... FROM nombre_tabla1
  INNER JOIN nombre_tabla2 ON nombre_tabla1.nombre_columna=
  nombre_tabla2.nombre_columna
```

- **LEFT JOIN**. Devuelve todas las filas de la tabla de la izquierda (`nombre_tabla1`), e incluso cuando no coincide con la tabla de la derecha (`nombre_tabla2`). No se mostrarán las columnas de la derecha que no coincidan con las de la izquierda.

```
SELECT nombre_columna1,... FROM nombre_tabla1
  LEFT JOIN nombre_tabla2 ON nombre_tabla1.nombre_columna=
  nombre_tabla2.nombre_columna
```

- **RIGHT JOIN**. Devuelve todas las filas de la tabla de la derecha (`nombre_tabla2`), e incluso cuando no coincide con la tabla de la izquierda (`nombre_tabla1`). No se mostrarán las columnas de la izquierda que no coincidan con las de la derecha.

```
SELECT nombre_columna1,... FROM nombre_tabla1
  RIGHT JOIN nombre_tabla2 ON nombre_tabla1.nombre_columna=
  nombre_tabla2.nombre_columna
```

- **FULL JOIN**. Devuelve todas las filas de las dos tablas.

```
SELECT nombre_columna1,... FROM nombre_tabla1
  FULL JOIN nombre_tabla2 ON nombre_tabla1.nombre_columna=
  nombre_tabla2.nombre_columna
```

- **SELECT...INTO**. Se utiliza para crear copias de seguridad (*backup*) de las tablas.

```
SELECT nombre_columna1,... INTO nombre_tabla_copia [IN
  nombre_BD_externa] FROM nombre_tabla_original
```

  Como hemos dicho anteriormente, esta sentencia se utiliza para realizar copias de seguridad de toda una tabla (utilizando "*") o de solo de un determinado número de columnas de una tabla. También es posible crear una copia de una tabla perteneciente a otra base de datos distinta, especificándolo con la cláusula `IN` seguido del nombre de la base de datos entre comillas simples.

- **BETWEEN**. Se utilizan en la condición de las consultas de selección con la cláusula `WHERE` para seleccionar solo aquellos campos que estén dentro de un rango de valores.

```
SELECT nombre_columna1,... FROM nombre_tabla WHERE
  nombre_columna BETWEEN valor1 AND valor2
```

- **IN**. El operador `IN` se utiliza en la condición de las consultas de selección con la cláusula `WHERE`. Este operador hace que la consulta devuelva el resultado, solo si la columna especificada en la condición toma un valor de los múltiples valores especificados con el operador `IN`.

```
SELECT nombre_columna1,... FROM nombre_tabla WHERE
  nombre_columna IN (valor1, valor2,...)
```

- **TOP**. Especifica el número de filas que debe devolver una consulta:
  - ✓ **SQL Server/MS Access**:

```
SELECT TOP número [PERCENT] nombre_columna1,... FROM
  nombre_tabla
```

  - ✓ **MySQL**:

```
SELECT nombre_columna1,... FROM nombre_tabla  LIMIT número
```

  - ✓ **Oracle**:

```
SELECT nombre_columna1,... FROM nombre_tabla  WHERE ROWNUM
  <= número
```

  En SQL Server y MS Access para especificar el número de filas que muestra una consulta podemos utilizar un número o por el contrario el porcentaje de filas que queremos que se muestren con respecto a las que tiene en total añadiendo `PERCENT`.

- **ALIAS**. Los alias son nombres abreviados que se le pueden asignar a columnas o tablas para utilizarlos dentro de la misma consulta.

```
SELECT nombre_columna...FROM nombre_tabla AS alias_tabla

SELECT nombre_columna AS alias_columna FROM nombre_tabla
```

- **PATRONES**. Estos patrones se utilizan para sustituir uno o más caracteres cuando realizamos búsquedas dentro de la base de datos. Es decir, que se utilizan cuando queremos que una consulta nos devuelva solo ciertas filas en las que el valor que toma un campo de una columna cumpla una cierta estructura. Como por ejemplo en una base de datos de de un hospital cuando solo queremos saber la información de los pacientes cuyos apellidos empiecen por "B".

  A continuación mostramos una tabla con los patrones que se pueden utilizar:

Tabla 6.1 Patrones

Patrón	Descripción	Ejemplo
`%`	Sustituye a cero o más caracteres.	Campos que empiecen por es: 'es%'.
`_`	Sustituye solo a un carácter.	Campos que empiecen por "ho" tengan un carácter y después "a". Lo cumpliría la cadena 'hola' 'ho_a'.
`[lista_caracteres]`	Solo un carácter de los que estén en la lista.	'%[nl]%' lo cumplirían cadenas como 'mano' o 'palo'.
`[^lista_caracteres]` o `[¡lista_caracteres]`	Solo un carácter de los que no estén en la lista.	'%[^nl]%' o '%[!nl]%' lo cumplen cadenas como 'paso'.

- **LIKE**. Este operador se utiliza para buscar determinados patrones en las columnas especificadas en una tabla.

```
SELECT nombre_columna1,... FROM nombre_tabla WHERE
    nombre_columna LIKE patrón
```

ACTUALIZAR CAMPOS DE UNA TABLA

Otra de las operaciones básicas en una base de datos es la actualización de los campos de una tabla, ya que con el tiempo puede ser necesario actualizar la información contenida en ella.

La sintaxis para ello es la siguiente:

```
UPDATE nombre_tabla SET nombre_columna1=valor1,
  nombre_columna2=valor2,... WHERE nombre_columna operador
  valor
```

Como podemos ver se indica `UPDATE` seguido del nombre de la tabla a la que pertenecen las columnas que queremos actualizar. Después añadimos `SET` seguido de pares con el nombre y el nuevo valor de las columnas a actualizar. Por último, utilizando la cláusula `WHERE` se especifica la condición que deben cumplir, para realmente saber que son esas columnas y no otras las que debe actualizar.

BORRAR CAMPOS DE UNA TABLA

Para borrar los campos de una tabla que cumplan cierta condición la sintaxis es la siguiente:

```
DELETE FROM nombre_tabla WHERE nombre_columna operador
  valor
```

Solo se borrarán los campos de la tabla especificada que cumplan la condición expresada en la cláusula `WHERE`.

También podemos borrar todos los campos de una tabla. La sintaxis para hacerlo es:

- **Forma 1**:

```
DELETE FROM nombre_tabla
```

- **Forma 2**:

```
DELETE * FROM nombre_tabla
```

Tenemos que tener mucho cuidado al borrar los campos o registros de una tabla, ya que una vez ejecutada la sentencia no se puede deshacer. Por este motivo es necesario que estemos seguros de los campos o registros que se van a borrar al ejecutar la sentencia y de que precisamente sean los que deseamos borrar. Al principio, cuando no se tiene práctica con las bases de datos es conveniente mostrar con una consulta de selección qué campos se van a borrar y después borrarlos.

6.3 UTILIZACIÓN DEL CONJUNTO DE RESULTADOS

Una vez ejecutadas las sentencias en la base de datos para obtener la información deseada, necesitamos procesar y utilizar dicha información. Por este motivo a continuación vamos a explicar qué conjunto de resultados utiliza cada lenguaje y cómo obtener la información a través de sus métodos.

En JSP la información se almacena en una estructura llamada `ResultSet`. Esta estructura almacena todos los registros o filas resultantes de la ejecución de una sentencia SQL y proporciona acceso a los datos de dichos registros a partir de una serie de métodos `get` que nos permiten acceder a los distintos campos o columnas del registro actual. La estructura de datos `ResultSet` contiene un puntero que señala al registro actual y que lo utiliza para acceder al resto de registros que almacena.

Los métodos y propiedades para recuperar la información de la estructura `ResultSet` los explicamos en la siguiente tabla:

Tabla 6.2 Métodos y propiedades del conjunto de resultados JSP

Método	Descripción
`getLong()`	Se utiliza para recuperar datos de tipo `BIGINT` y `BIT` de la base de datos.
`getBytes()`	Se utiliza para recuperar datos de tipo `BINARY` y `VARBINARY` de la base de datos.
`getString()`	Se utiliza para recuperar datos de tipo `CHAR, VARCHAR` y `VARCHAR2` de la base de datos.
`getDate()`	Se utiliza para recuperar datos de tipo `DATE` de la base de datos.
`getBigDecimal()`	Se utiliza para recuperar datos de tipo `DECIMAL` y `NUMERIC` de la base de datos.
`getDouble()`	Se utiliza para recuperar datos de tipo `DOUBLE` y `FLOAT` de la base de datos.
`getInt()`	Se utiliza para recuperar datos de tipo `INTEGER` de la base de datos.
`getBinaryStream()`	Se utiliza para recuperar datos de tipo `LONGVARCHAR` de la base de datos.
`getAsciiStream()`	Se utiliza para recuperar datos de tipo `LONGVARBINARY` de la base de datos.
`getUnicodeStream`	Se utiliza para recuperar datos de tipo `LONGVARCHAR` de la base de datos.

`getFloat()`	Se utiliza para recuperar datos de tipo `REAL` de la base de datos.
`getShort()`	Se utiliza para recuperar datos de tipo `SMALLINT` de la base de datos.
`getTime()`	Se utiliza para recuperar datos de tipo `TIME` de la base de datos.
`getTimestamp()`	Se utiliza para recuperar datos de tipo `TIMESTAMP` de la base de datos.
`getByte()`	Se utiliza para recuperar datos de tipo `TINYINT` de la base de datos.
`first()`	Apunta el cursor la primera fila almacenada en ResultSet.
`last()`	Apunta el cursor a la última fila almacenada en ResultSet.
`beforeFirst()`	Apunta el cursor antes de la primera fila almacenada en ResultSet.
`afterLast()`	Apunta el cursor después de la última fila almacenada en ResultSet.
`absolute(int num)`	Apunta el cursor a la fila que se encuentra en la posición que se le pasa como argumento.
`next()`	Mueve el cursor a la fila siguiente.
`previous()`	Mueve el cursor a la fila anterior.
`getRow()`	Devuelve la fila en la que se encuentra el cursor.
`isFirst()`	Devuelve un valor booleano que indica si el cursor está en la primera posición.
`isLast()`	Devuelve un valor booleano que indica si el cursor está en la última posición.
`isBeforeFirst()`	Devuelve un valor booleano que indica si el cursor está antes de la primera posición.
`isAfterLast()`	Devuelve un valor booleano que indica si el cursor está después de la última posición.

En PHP el resultado de ejecutar una sentencia SQL se almacena en una variable cualquiera. Dado que en PHP no se definen las variable previamente a su uso. Existen los siguientes métodos en PHP para recorrer la estructura que devuelve la ejecución de una sentencia. A continuación mostramos alguna de las funciones y propiedades que resultan más útiles en el procesamiento de los datos obtenidos de una base de datos.

Tabla 6.3 Métodos y propiedades del conjunto de resultados PHP

Método	Descripción
`mysql_num_rows($result)/ mysqli_num_rows($result)`	Devuelve el número de filas que tiene un resultado.
`mysql_num_fields($result)/ mysqli_num_fields($result)`	Devuelve el número de campos que tiene un resultado.
`mysql_affected_rows([$link]) /mysqli_affected_rows($link)`	Devuelve el número de filas afectadas por una operación SQL.
`mysql_free_result($result)/ mysqli_free_result()`	Libera la memoria reservada para almacenar un resultado.
`mysql_fetch_array($result)/ mysqli_fetch_array($result)`	Devuelve un *array* con las filas obtenidas o un valor booleano que índica que no tiene más filas.
`mysql_fetch_row($result)/ mysqli_fetch_row($result)`	Devuelve una fila de resultados como un *array* enumerado.
`mysql_fetch_field($result, [$pos_campo])/ mysqli_fetch_field($result)`	Devuelve el siguiente campo del conjunto de resultados.
`mysql_fetch_assoc($result)/ mysqli_fetch_assoc($result)`	Devuelve una fila de resultados como un *array* asociativo.
`mysql_fetch_object($result)/ mysqli_fetch_object($result)`	Devuelve la fila actual del resultado en forma de un objeto.
`mysql_result($result)`	Devuelve el resultado.
`mysql_field_len($result, $pos_campo)`	Devuelve el tamaño de un campo específico.
`mysql_fetch_lengths($result, $num)/ mysqli_fetch_lengths($result)`	Devuelve la longitud de las columnas de la fila actual en el conjunto de resultados.

Por último ASP utiliza una estructura de datos llamada `RecordSet` que almacena el resultado de las sentencias ejecutadas en SQL y contiene los siguientes métodos y propiedades para poder acceder a la información que almacena.

Tabla 6.4 Propiedades y métodos del conjunto de resultados ASP

Método	Descripción
`EOF`	Devuelve un booleano que indica si se ha acabado el `RecordSet`.
`RecordCount`	Devuelve el número de registros que tiene un RecordSet.
`AbsolutePosition`	Devuelve el valor que contiene el registro actual del `RecordSet` en la posición especificada.
`MoveFirst`	Mueve el cursor al primer registro del `RecordSet`.
`MoveLast`	Mueve el cursor al último registro del `RecordSet`.
`MoveNext`	Mueve el cursor al registro siguiente del `RecordSet`.
`MovePrevious`	Mueve el cursor al registro anterior del `RecordSet`.
`GetRow`	Devuelve los registros del `RecordSet` en un *array* bidimensional.
`GetString`	Devuelve la estructura `RecordSet` en forma de *string*.

6.4 CIERRE DE CONEXIONES

Una vez que ya hemos terminado de trabajar con la base de datos es necesario cerrar la conexión. A continuación explicaremos el código necesario para llevarlo a cabo:

- **ASP**:

```
conexion.close()
Set rs=nothing
Set conexion=nothing
```

- **JSP**:

```
rs.close();
st.close();
conexion.close();
```

- **PHP**:

```
mysql_free_result($result);
mysql_close($conector);
```

En JSP los objetos `Statement`, `Connection` y `ResultSet` deben llamar al método `close()`. En PHP se liberará la memoria reservada para el objeto `ResultSet` devuelto como resultado de ejecutar una consulta (únicamente con la sentencia `SELECT`) y después se cerrará la base de datos pasándole como argumento el conector obtenido al abrir la conexión. Por último en ASP se cerrará la conexión y se inicializará con el valor `nothing` tanto el objeto de la conexión como el que almacenaba el resultado de la ejecución de una consulta.

ACTIVIDADES 6.1

Pruebe el siguiente código en ASP que realiza una consulta a la base de datos Access "prueba.mdb" que adjuntamos.

```
<html>
<head>
  <title>Ejemplo de Consulta a una base de datos en
      ASP
  </title>
</head>
<body>
  <h1>Ejemplo de Consulta a una base de datos en
      ASP</h1>
  <table border="1">
    <tr>
      <td>IdAlumno</td>

<td>Nombre</td>

<td>Apellidos</td>

<td>Direccion</td>

<td>Telefono</td>
    </tr>
  <%
    Dim conexion, registros, basedatos
      strcon = Server.MapPath("prueba.mdb")


Set conexion=
        Server.CreateObject("ADODB.Connection")

    conexion.Open
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
          & strcon & ";"


Set rs = conexion.Execute("SELECT * FROM Alumnos")


Do While Not rs.EOF
        response.write "<tr>"
```

```
  response.write "<td>" & rs("IdAlumno") & "</td>"
        response.write "<td>" & rs("Nombre") & "</td>"

  response.write "<td>" & rs("Apellidos") &
            "</td>"
        response.write "<td>" & rs("Dirección") &
            "</td>"
        response.write "<td>" & rs("Telefono") & "</td>"

  response.write "</tr>"

  rs.MoveNext

Loop


response.write "</table>"

conexion.close

Set rs = Nothing
      Set conexion = Nothing

    %>
</body>
 </html>
```

6.5 TRANSACCIONES

Actualmente utilizamos bases de datos que albergan a un gran número de usuarios. Estos interactúan con la base de datos de manera simultánea lo que puede provocar inconsistencias dentro de la misma, ya que si varios usuarios modifican de manera independiente la base de datos la probabilidad de colisiones entre los usuarios al actualizar los datos es bastante grande. Las transacciones surgieron como medida para controlar el acceso a los datos ante la necesidad de acceder de manera simultánea a la base de datos y el riesgo que esto acarreaba.

Cuando hablamos de transacciones nos referimos a un conjunto de instrucciones que se ejecutan de manera atómica, es decir como si fuera una sola instrucción o sentencia SQL. Gracias a la atomicidad que nos otorgan las transacciones podemos realizar operaciones complejas sobre la base de datos sin que por ello pongamos en peligro la integridad de la misma.

Como hemos dicho anteriormente, una transacción está compuesta por varias instrucciones que se comportan como una sola. Por ello, solo si todas y cada una de las instrucciones que la componen se ejecutan con éxito esta lo hará también. De lo contrario, si alguna de las instrucciones falla se anulan todas las instrucciones que se hayan ejecutado hasta el momento para dejar la base de datos igual que estaba antes de que se empezara a ejecutar la transacción.

Para que un grupo de sentencias sean consideradas una transacción debe cumplir las siguientes propiedades del test ACID:

- **Atomicidad**. El grupo de sentencias debe comportarse como un "todo o ninguno". Si alguna de ellas no se pudiera ejecutar se debería poder deshacer o revertir.
- **Consistencia**. La base de datos debe quedarse en un estado consistente y coherente al finalizar la transacción.
- **Aislamiento**. Los datos modificados durante una transacción no deben ser visibles para otros usuarios hasta que esta no finalice y se hayan realizado todos los cambios de manera definitiva.
- **Permanencia**. Los efectos de una transacción se almacenarán de forma permanente.

Por defecto las bases de datos se comportan como si cada sentencia fuera una transacción (transacciones implícitas). Esto quiere decir, que cuando ejecutamos una sentencia que modifica los datos de una tabla (sentencias DML) se almacena el resultado en el disco de forma automática. Para definir transacciones de manera explícita necesitamos escribir la sentencia que da comienzo a la transacción, que varía ligeramente dependiendo de la base de datos. Después del inicio de la transacción, incluiremos todas las sentencia SQL que queremos que formen parte de esta. Por último, finalizaremos la transacción.

Existen dos formas de finalizar una transacción, una de ellas es `commit` y otra `rollback`. La primera de ellas se ejecuta para finalizar la transacción de forma exitosa y hacer permanentes los cambios realizados durante la misma. Por otro lado, la instrucción `rollback` se ejecutará para abortar la transacción porque algunas de las instrucciones que la componen se ha ejecutado y ha fallado. Esta instrucción deshace todos los cambios temporales que se pudieran haber hecho en la base de datos de manera que la base de datos quede igual que estaba antes de ejecutar la transacción.

A continuación vamos a mostrar la sintaxis para definir una transacción en las distintas bases de datos:

✓ **MS Access**:

```
BEGIN TRANSACTION
Sentencia1_SQL
Sentencia2_SQL
...
ROLLBACK [WORK|TRANSACTION]
COMMIT [WORK|TRANSACTION]
```

✓ **SQL Server**:

```
BEGIN TRAN|TRANSACTION [nombre_transacción]
Sentencia1_SQL
Sentencia2_SQL
...
ROLLBACK TRAN|TRANSACTION [nombre_transacción|
  punto_recuperación]|[WORK]
COMMIT TRAN|TRANSACTION [nombre_transacción] | [WORK]
```

✓ **Oracle**:

```
Sentencia1_SQL
Sentencia2_SQL
...
ROLLBACK [WORK] [TO SAVEPOINT nombre|TO punto_recuperación]
COMMIT [WORK]
```

✓ **MySQL**:

```
START {TRANSACTION|BEGIN [WORK]}
Sentencia1_SQL
Sentencia2_SQL
...
ROLLBACK [WORK][TO SAVEPOINT nombre |TO punto_recuperación]
COMMIT [WORK]
```

Dentro de una transacción podemos definir distintos puntos a través de los cuales volver atrás con un `ROLLBACK TO` sin deshacer toda la transacción. La sintaxis para definir estos puntos de retorno es la siguiente:

✓ **SQL Server**:

```
SAVE TRAN|TRANSACTION punto_recuperación
```

✓ **Oracle/MySQL**:

```
SAVEPOINT nombre
```

Es más, en MySQL existe una sentencia para borrar puntos de recuperación creados en la transacción actual:

```
RELEASE SAVEPOINT punto_recuperación
```

ACTIVIDADES 6.2

Partiendo de este ejemplo teórico de uso de transacciones en SQL Server:

```
begin transaction prueba
update estudiante …
save transaction prueba2
delete asignatura…
rollback transaction prueba2
commit transaction prueba
```

¿Qué sentencias SQL se harán efectivas en la base de datos?

6.5.1 SERIALIZACIÓN O NIVELES DE AISLAMIENTO

Existen distintos tipos de transacciones dependiendo del tipo de acceso que permitamos durante la ejecución de una transacción. A esto se le llama niveles de aislamiento.

Antes de ver los distintos tipos de aislamiento que existen y como configurarlo en nuestra base de datos es necesario que entendamos una serie de conceptos previos:

- **Lectura sucia**. Esta situación se da cuando una transacción T1 modifica los datos que lee T2, y luego se hace un `rollback` de T1.Por lo tanto T2 ha leído unos datos que no existen.

- **Lectura no repetible**. En una transacción T1 queremos leer dos veces los mismos datos, pero entre ambas lecturas T2 elimina algunos datos. Esto provoca que tengamos resultados distintos en cada una de las lecturas, ya que en la segunda lectura nos faltarán datos con respecto a la lectura anterior.
- **Lectura fantasma**. Es parecida a la anterior, ya que en una transacción T1 queremos leer dos veces los mismos datos, pero entre ambas lecturas T2 en vez de eliminar añade algunos datos. Esto provoca que al leer por segunda vez nos encontremos tuplas que anteriormente no existían.

Una vez que conocemos estos conceptos vamos a ver los distintos tipos de aislamiento y en cuáles de ellos se producen los fenómenos anteriormente explicados:

- **Serializable**. Si aplicamos el nivel serializable podremos ver la base de datos antes o después de la ejecución de una transacción, pero nunca mientras se está realizando. Por lo tanto da la sensación de que las instrucciones se ejecutan una detrás de otra. Esto solo es aplicable al usuario que lo elige, es decir que otro usuario puede tener definido otro nivel de aislamiento. Este es el nivel que viene por defecto en las base de datos.
- **Read commited**. Este nivel evita la lectura sucia permitiendo solo la lectura de los datos que han sido confirmados mediante la realización de un `commit` de otra transacción.
- **Repeatable read**. Este nivel en cambio evita la lectura no repetible. Es igual que "*Read commited*" añadiéndole la restricción de que todo lo que veamos en una lectura inicial de una transacción debemos poder verlo en lecturas posteriores.
- **Read uncommited**. Si una transacción se ejecuta con este nivel de aislamiento puede ver las modificaciones realizadas por otra transacción, sean actualizaciones, borrados o inserciones a pesar de no haber hecho un `commit`.

Por último, vamos a ver cómo configurar los niveles de aislamiento en la base de datos:

✓ **MySQL**:

```
SET [SESSION|GLOBAL] TRANSACTION ISOLATION LEVEL READ
  UNCOMMITED| READ COMMITED|SERIALIZABLE| REPETEABLE READ
```

✓ **SQL Server**:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED| READ
  COMMITED|SERIALIZABLE| REPETEABLE READ | SNAPSHOT
```

✓ **Oracle**:

```
SET TRANSACTION ISOLATION LEVEL {READ COMMITED |
  SERIALIZABLE}
```

Como podemos observar SQL Server a parte de los niveles explicados anteriormente añade uno más `SNAPSHOT`. Este nivel de aislamiento hace que las instrucciones que se ejecuten en una transacción no vean las modificaciones de datos efectuadas por otras transacciones, si no que en su lugar reciban una "copia coherente" de cómo estaban los datos al comienzo de la transacción.

Por otro lado Oracle solo define dos de los niveles explicados anteriormente (*read commited* y *serializable*).

En cuanto a MySQL podemos especificar si queremos que el nivel de aislamiento se aplique para la sesión actual (`SESSION`) o sea el que se use siempre desde ese momento (`GLOBAL`).

Tabla 6.5 Niveles de aislamiento

Nivel	Lectura Sucia	Lectura No Repetible	Lectura Fantasma
`Serializable`	NO	NO	NO
`Repeteable Read`	NO	NO	SÍ
`Read Commited`	NO	SÍ	SÍ
`Read Uncommited`	SÍ	SÍ	SÍ
`Snapshot`	NO	NO	NO

En la Tabla 6.5 podemos ver cuáles de las situaciones explicadas anteriormente se evitan con cada nivel de aislamiento. En conclusión la más estricta es el nivel *Serializable* mientras que *Read Uncommited* es la más permisiva.

6.6 UTILIZACIÓN DE OTROS ÓRIGENES DE DATOS

En apartados anteriores hemos visto cómo obtener información de una base de datos, pero este no es el único método para almacenar información. También podemos almacenar información en ficheros por lo que resulta necesario ser capaces de trabajar con ellos. Por este motivo en este apartado explicaremos como trabajar con distintos tipos de ficheros: CSV, XML y fichero de texto. Los ficheros CSV son documentos de formato abierto para representar datos en forma de tabla y que pueden ser procesados en Excel.

En estos ficheros separan las filas por saltos de línea y las columnas por un separador (en general suele utilizarse ","). Por otra parte, los ficheros XML se utilizan hoy en día para el intercambio de información estructurada y está basado en un lenguaje de etiquetas.

Para procesar un fichero lo primero que tenemos que hacer es abrirlo (si este existe) o crearlo mediante una función. Una vez abierto podremos tanto leer como escribir en él y una vez que hemos terminado de utilizarlo se cierra.

Aquí podemos ver las funciones que disponemos en PHP para interactuar con ficheros:

Tabla 6.6 Funciones para procesar ficheros en PHP

Funciones PHP	Descripción
`$descriptor=fopen($nombre, $tipo_acceso)`	Crea y abre un fichero (válido para todo tipo de ficheros). Los tipos de acceso más utilizado son: r (lectura), w (escritura), a (escritura añadiendo al final del fichero).
`fclose($descriptor)`	Cierra un fichero (válido para todo tipo de ficheros).
`fread($descriptor, $tamaño)`	Lee de un fichero de texto un tamaño determinado.
`fwrite($descriptor, $texto, [$tamaño])`	Escribe en un fichero de texto la cadena que le pasamos como parámetro.
`fgetcsv($descriptor,[$tamaño], [$delimitador],[$cierre_campo], [$carácter_escape])`	Lee un fichero CSV. Devuelve en una matriz los valores de un campo.
`fputcsv($descriptor, $matriz_valores, [$delimitador],[$cierre_campo])`	Escribe en un fichero CSV. Devuelve la longitud de la cadena escrita.
`feof($descriptor)`	Devuelve un booleano indicando si se ha llegado al final del fichero (válido para todo tipo de ficheros).

Para ficheros XML podemos usar la función `simplexml_load_file(ruta)` que interpreta un fichero XML como un objeto y se puede acceder a sus elementos de forma más sencilla.

Para utilizar ficheros en ASP primero tenemos que crear el objeto `File System`:

```
Set fs=Server.CreateObject("Scripting.FileSystemObject")
```

En la Tabla 6.7 están disponibles las funciones relacionadas con los ficheros en el lenguaje ASP.

Tabla 6.7 Funciones para procesar ficheros en ASP

Funciones ASP	Descripción
`file=FileSystem.OpenTextFile(ruta,[tipo_acceso],[crear],formato)`	Abre el fichero especificado en la ruta. El parámetro `tipo_acceso` indica para que se va a usar (`ForReading, ForWriting, ForAppending`), el parámetro crear es un booleano que dice si crear un fichero en caso de que no exista y el formato del fichero que por defecto es ASCII (válido para todo tipo de ficheros).
`file=FileSystem.CreateTextFile(ruta,[sobreescribe],[unicode])`	Crea un fichero en la ruta especificada. Los parámetros `sobrescribe` y `Unicode` son booleanos y opcionales (válido para todo tipo de ficheros).
`file.Close()`	Cierra un fichero (válido para todo tipo de ficheros).
`file.Read(tamaño)`	Leer de un fichero un tamaño determinado (válido para todo tipo de ficheros).
`file.ReadAll()`	Lee el fichero entero (válido para todo tipo de ficheros).
`file.ReadLine()`	Lee una línea del fichero (válido para todo tipo de ficheros).
`file.Write(texto)`	Escribe un texto en el fichero (válido para todo tipo de ficheros).
`file.WriteLine(texto)`	Escribe un texto en el fichero y al final le añade un salto de línea (válido para todo tipo de ficheros).

Por otra parte, en JSP lo primero que tenemos que hacer para utilizar ficheros de texto es crear un fichero si este no existe (`File`), un objeto `PrintWriter` para escribir en él y/o un objeto `BufferedReader` si lo que queremos es leer la información que este contiene.

```
File fichero= new File(ruta_fichero);
  if(!fichero.exists()){
    fichero.createNewFile();
  }

BufferedReader br=BufferedReader(new FileReader(fichero));
PrintWriter pw = new FileWriter(ruta, [flag_concatenar]);
```

Al objeto `FileWriter` se le puede pasar como parámetro un valor booleano (`flag_concatenar`) que si toma como valor `true` concatena lo que vayamos a escribir en el fichero con el contenido que tuviera anteriormente. Si toma como valor `false` entonces el fichero se sobrescribe.

Si en lugar de ficheros de texto trabajamos con ficheros CSV tendremos que crear un objeto `CsvWriter` para escribir en ellos y `CsvReader` para leerlos:

```
File f= new File(ruta_fichero);
  if(!f.exists()){
    f.createNewFile();
  }

FileReader fr=new FileReader(f);
CsvReader cvsr= new CvsReader(fr,delimitador))

FileWriter fw=new FileWriter(f);
CsvWriter cvsw= new CvsWriter(fr,delimitador)
```

Por último, en JSP se utilizan las siguientes funciones para interactuar con ficheros:

Tabla 6.8 Funciones para procesar ficheros en JSP

Funciones JSP	Descripción
`BufferedReader.read(buffer,offset, tamaño)`	Leer el tamaño indicado de un fichero de texto.
`BufferedReader.readLine()`	Leer una línea de un fichero de texto.
`PrintWriter.println(texto)`	Escribir en un fichero de texto.
`BufferedReader.close()` `PrintWriter.close()`	Cierra un fichero de texto.
`CvsReader.getHeaders()`	Devuelve el valor de las cabeceras de las columnas de un fichero CSV en forma de *array*.
`CvsReader.readHeaders()`	Devuelve un booleano que indica si tiene cabeceras un fichero CSV.
`CvsReader.readRecord()`	Devuelve un booleano que indica si tiene más registros un fichero CSV.
`CvsReader.get(cabecera)` `CvsReader.get(posición)`	Devuelve el valor de una columna dada la posición de una columna o el nombre de su cabecera.
`CsvWriter.write(texto)`	Escribe otra columna de datos en este registro.
`CsvWriter.endRecord()`	Termina el registro actual mediante el delimitador del registro.
`CsvWriter.close()` `CsvReader.close()`	Cierra un fichero CSV.
`File.exists()`	Devuelve un booleano que indica si un fichero existe.
`File.createNewFile()`	Crea un fichero.

En PHP y JSP una de las alternativas para leer ficheros XML es convertir este en un objeto DOM que permita recorrer la estructura en forma de árbol. Existen métodos para crear documentos XML y para acceder a las distintas partes que este posee:

Tabla 6.9 Funciones para documentos XML en PHP

Funciones PHP	Descripción
`DomDocument=domxml_open_file($fichero,[$modo],[$error])`	Crea un objeto `Document` a partir de un fichero XML.
`DomDocument-> create_element($nombre)`	Crea un elemento.
`DomDocument-> create_attribute($nombre, $valor)`	Crea un atributo.
`DomDocument->add_root($nombre)`	Crea un nodo raíz.
`DomDocument-> get_elements_by_tagname($nombre)`	Devuelve los elementos con dicho nombre.
`DomDocument->document_element()`	Devuelve el nodo raíz del objeto `Document`.
`DomElement->get_attribute($nombre)`	Devuelve el valor de un atributo dado.
`DomNode->next_sibiling()`	Devuelve el siguiente nodo hermano.
`DomDocumet-> dump_file($fichero, $modo_compacto],[$formato])`	Vuelca el objeto `Document` en un fichero XML. Los dos últimos parámetros son booleanos y se refieren al formato.

Si vamos a utilizar PHP 5.0 o superiores recomendamos ver las funciones `SimpleXMLElement`, que simplifica mucho el trabajo con XML.

En JSP también podemos leer un fichero XML pero antes debemos crear un documento DOM a partir del XML, para lo que hay que hacer lo siguiente:

```
DocumentBuilderFactory factory =
  DocumentBuilderFactory.newInstance();

try{
  DocumentBuilder builder = factory.newDocumentBuilder();
  Document document= builder.parse(new File(ruta_fichero));
}catch(Excetion e){
    out.println(e);
}
```

Una vez tenemos el objeto `Document` podemos utilizar las siguientes funciones para leer las distintas partes del fichero XML:

Tabla 6.10 Funciones para documentos XML en JSP

Funciones PHP	Descripción
`Document.getFirstChild()`	Devuelve el primer hijo del nodo.
`Document.getLastChild()`	Devuelve el último hijo del nodo.
`Document.getChildNodes()`	Devuelve una lista con los hijos que contiene el nodo.
`Document.getNextSibiling()`	Devuelve el nodo siguiente.
`Document.getNodeName()`	Devuelve el nombre de un nodo.
`Document.getNodeValue()`	Devuelve el valor de un nodo.
`Document.getAttributes()`	Devuelve los atributos de un nodo.

Para escribir un documento XML a partir de un objeto `Document` tendríamos que hacer lo siguiente:

```
Document documento=...;
TransformerFactory transformerFactory=
  TransformerFactory.newInstance();
Transformer trans= transformerFactory.newTransformer();
DOMSource source= new DOMSource(documento);
StreamResult result= new StreamResult(new
  File(fichero_xml));
trans.transform(source,result);
```

ACTIVIDADES 6.3

- Copie y pruebe este código en PHP que escribe una cadena en un fichero y lo lee posteriormente para mostrarlo por pantalla:

```
<html>
  <head>
    <title>Ejemplo de utilización de ficheros de texto
      en PHP</title>
  </head>
  <body>
    <h1>Ejemplo de utilización de ficheros de texto en
      PHP</h1>
    <?php
      $DescriptorFichero = fopen("prueba.txt","w");
      fwrite($DescriptorFichero,"Este mensaje es una
        prueba");

      $DescriptorFichero = fopen("prueba.txt","r");
      $texto=fread($DescriptorFichero,1024);

      echo $texto;
      fclose($DescriptorFichero);
    ?>
  </body>
</html>
```

RESUMEN DEL CAPÍTULO

En este capítulo hemos aprendido a acceder a bases de datos utilizando distintos tipos de gestores de base de datos. Además, hemos aprendido cómo ejecutar distintas sentencias SQL para interactuar, obtener y almacenar en una base de datos la información que necesitemos.

Otra de las cosas que hemos podido ver es cómo mostrar o manipular los datos obtenidos de la base de datos y el uso de transacciones.

Por último, hemos explicado métodos para acceder a otro tipo de orígenes de datos de los que obtener información, como son los ficheros de texto, ficheros XML y ficheros CSV.

EJERCICIOS PROPUESTOS

- **1.** Basándose en la actividad 6.1, escriba el mismo código para realizar una consulta a la base de datos de Access que facilitamos con el nombre de "`prueba.mdb`" para JSP y que muestre el resultado con una tabla HTML.
- **2.** Basándose en la actividad 6.1, escriba el mismo código para realizar una consulta a la base de datos de MySQL que facilitamos con el nombre de "`prueba.sql`" para PHP y que muestre el resultado con una tabla HTML.
- **3.** Escriba el código en JSP para realizar una inserción de datos a la tabla "Alumnos" de la base de datos Access que adjuntamos ("`prueba.mdb`").
- **4.** Basándose en la actividad 6.3, programe el mismo código en ASP que escriba un texto en un fichero y lo lea para mostrarlo por pantalla.
- **5.** Basándose en la actividad 6.3, programe el mismo código en JSP que escriba un texto en un fichero y lo lea para mostrarlo por pantalla.

TEST DE CONOCIMIENTOS

1 ¿Qué significan las siglas SQL?
- a) *Structured Query Language*.
- b) *Structured Question Language*.
- c) *Standard Query Language*.
- d) Ninguna de las anteriores.

2 ¿Qué es una transacción?
- a) Cada una de las sentencias SQL que ejecutamos.
- b) Aquellas sentencias que deben ejecutarse de manera repetitiva.
- c) Un conjunto de sentencias que deben ejecutarse de forma atómica.
- d) Ninguna de las anteriores.

3 ¿Cuál de estos fenómenos se evitan si utilizamos como nivel de aislamiento "`serializable`"?
- a) Lectura sucia.
- b) Lectura no repetible.
- c) Lectura fantasma.
- d) Todas las anteriores.

4 ¿Cuál de estas propiedades debe cumplir una transacción SQL?
- a) Atomicidad y permanencia.
- b) Consistencia.
- c) Aislamiento.
- d) Todas las anteriores.

5 En PHP, ¿cuál es la diferencia entre el método `mysql_connect()` y `mysqli_connect()`?
- a) `5mysqli_connect()` se utiliza para conectarse a bases de datos SQLite mientras que `mysql_connect()` se utiliza para conectarse a bases de datos MySQL.
- b) `mysql_connect()` se emplea con MySQL 4.1 y `mysqli_connect()` con MySQL 4.0 y anteriores.
- c) Con `mysqli_connect()` crea un fichero *log* con los errores producidos al intentar conectar con la base de datos mientras que el método `mysql_connect()` informa al usuario del error si falla pero no lo almacena en un fichero *log*.
- d) Ninguna de los anteriores.

6 ¿Cuál es el objetivo de una `PRIMARY KEY`?
- a) Asignar permisos de acceso especiales a los registros de una base de datos.
- b) Permitir asignar prioridades en la lectura y escritura de los registros de una base de datos.
- c) Identificar de manera unívoca y exclusiva un registro dentro de una tabla de la base de datos.
- d) Sirve como llave de acceso a contenidos ocultos de la base de datos solo disponibles para el administrador.

7 DML engloba las instrucciones SQL que permiten:
- a) Ejercer control sobre los procesos que ejecutan transacciones y los permisos de la base de datos.
- b) Definir la estructura de una base de datos como, por ejemplo, crear una base de datos o crear y eliminar tablas.
- c) Realizar cambios en los datos contenidos en la base de datos: incluir registros, borrarlos, editarlos, etc.
- d) Todas las anteriores son ciertas.

8 ¿Qué sentencia utilizaremos si queremos consultar una base de datos?
- a) `SELECT…INTO`.
- b) `SELECT`.
- c) Se pueden utilizar ambas.
- d) Ninguna de las anteriores.

7 Programación de servicios web

OBJETIVOS DEL CAPÍTULO

- ✓ Reconocer las características y ámbito de aplicación de los servicios web.
- ✓ Reconocer las ventajas de utilizar servicios web para dar funcionalidad a la lógica de la capa de negocio de una aplicación.
- ✓ Identificar las tecnologías y protocolos para publicar y utilizar servicios web.
- ✓ Programar un servicio web y crear el documento de descripción del mismo.
- ✓ Verificar y consumir el funcionamiento de un servicio web.

Con la aparición de Internet, a mediados de la década de los 90, comenzaron a extenderse una gran variedad de desarrollos software y hardware. Estos sistemas, tenían unas peculiaridades muy diferentes. Es cierto que las diferentes ramificaciones surgidas enriquecían el panorama tecnológico, no solo a nivel software si no a nivel hardware también. Al existir tal diversidad, las empresas vieron la necesidad de integrar estos sistemas, de tal forma que pudieran comunicarse, para poder colaborar en sus objetivos unos sistemas con otros. El afán de las empresas se encauzo en desarrollar la mejor tecnología que integrara los sistemas. Pero según avanzaba la competencia, resultaba más complicado desarrollar este producto. Debido a la rápida aceptación del fenómeno de Internet y el fuerte impacto que causaron las tecnologías de la información, las relaciones de negocio y comunicación cambiaron radicalmente. El uso de las tecnologías de la información en la vida cotidiana, las exigencias de integrar diferentes sistemas (tanto hardware como software), se volvió algo prioritario. Las empresas veían inviable crear una plataforma integradora de forma individual. Por esta razón algunas empresas emprendedoras, buscaron un lenguaje común que intercambiara información. Para ello aprovecharon estándares existentes en el mercado, que además iban a facilitar que se extendiera más esta tecnología. A partir de estas necesidades surgidas, se crean los servicios web.

Un *servicio web* es un conjunto de protocolos y estándares que permiten comunicar dos sistemas a través de una red. Habitualmente los servicios web actúan para intercambiar datos (comunicarse), entre dos aplicaciones. Estas aplicaciones suelen estar desarrolladas en lenguajes de programación distintos. Además pueden estar en plataformas (sistemas operativos o arquitecturas) diferentes. Es habitual, y el uso más generalizado que los servicios web, intercambien los datos a través de Internet. Aunque esta es la generalidad podrían implementarse servicios web en otro tipo de red.

La operabilidad de los servicios web se consigue a través de estándares abiertos que gestionan organizaciones como OASIS (*Organization for the Advancement of Structured Information Standards*), consorcio internacional, sin ánimo de lucro, que orienta su desarrollo a la adopción de estándares para el comercio electrónico y los servicios web. Otro de los consorcios internacionales que produce recomendaciones para los servicios web, es W3C. Estos son los comités responsables de la reglamentación de los servicios web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios web, se ha creado el organismo WS-I. Más adelante veremos la importancia que tiene el permitir la ejecución de servicios web en múltiples plataformas tanto software como hardware.

A continuación vamos a ver el esquema conceptual de cómo se comunican varios servicios web. Una tienda *on line* ofrece servicio a través de Internet a sus clientes. Para completar el servicio, necesita comunicarse, con la aplicación del banco y con la aplicación de pago por Paypal.

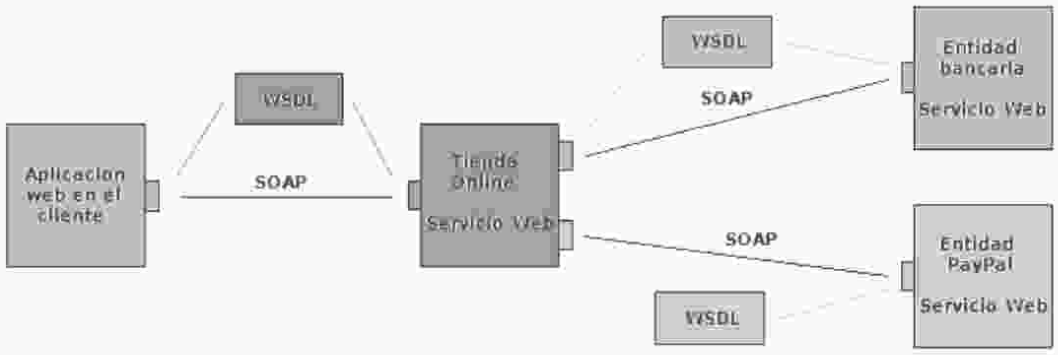


Figura 7.1. *Vista general de varios servicios web*

En la Figura 7.1, observamos como el perfil de usuario (aplicación web en el cliente), que tiene el rol de cliente del servicio web. Solicita información sobre los productos de una tienda *on line*. Este usuario realiza una petición a la tienda *on line* a través de un servicio web. En última instancia, el usuario (aplicación web en el cliente), compra un producto y realiza el pago al servicio web de la entidad bancaria o al de Paypal dependiendo el modo de pago que haya elegido, a través de la tienda *on line*. En todo este proceso intervienen una serie de tecnologías que hacen posible la circulación de la información. En el punto siguiente vamos a describir cuales son y cómo se comunican.

7.1 MECANISMOS Y PROTOCOLOS IMPLICADOS

Como hemos indicado anteriormente, un servicio web es un conjunto de protocolos que sirven para intercambiar información entre aplicaciones. Estos protocolos se consideran independientes y los autores, acostumbran a no encuadrarlos en una documentación común. El objetivo, es dinamizar las posibilidades de comunicación, seleccionando el protocolo adecuado a casa caso particular. Aun así, al conjunto de servicios y protocolos de un servicio web, se le llama *Web Services Protocol Stack* (pila de protocolos de servicios web). Esta colección de protocolos y estándares, sirve para definir, implementar, localizar y hacer que un servicio web interactúe con un cliente. La pila de protocolos se divide en cuatro partes que podemos denominar, servicios, a continuación vamos a describir cada uno de ellos. En la siguiente imagen mostramos la pila de protocolos.

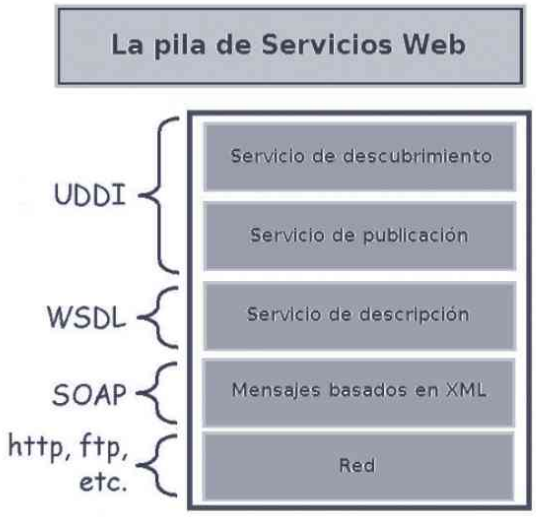


Figura 7.2. *Pila de protocolos de un servicio web*

En la imagen anterior, visualizamos los cuatro servicios de la pila de protocolos de un servicio web. En la capa inferior, se encuentra el servicio de transporte. En la siguiente capa (*XML Based Menssaging*), está el servicio de mensajería. En la tercera capa (*Service Description*), se describirá el servicio web. La capa superior es la encargada de facilitar el descubrimiento de los servicios web. A continuación vamos a ver detenidamente cada uno de los servicios que integran la pila de protocolos.

7.1.1 SERVICIO DE TRANSPORTE

El servicio de transporte, es el responsable del envío de mensajes entre las aplicaciones a través de la Red. Este servicio, trabaja al nivel más bajo, encargándose de cómo se codifica la información, sin preocuparse de su formato. Es el encargado de establecer la conexión y el puerto que se va a usar. Generalmente se utiliza el protocolo HTTP, el mismo protocolo que utiliza la *World Wide Web*. La razón de que la mayoría de los servicios web hagan uso de este protocolo, es el uso tan extendido de los navegadores web. A través de Internet, los clientes hacen uso de los servicios web. Como los servicios web originalmente no fueron pensados expresamente para utilizar este protocolo, se pueden utilizar otros, como, SMTP (protocolo de correo electrónico), FTP (protocolo de trasferencia de ficheros) o BEEP (*Blocks Extensible Exchange Protocol*). Este último, es un protocolo específico para servicios web. A diferencia de los protocolos anteriores, BEEP, no es un protocolo cliente-servidor. Los dos ordenadores entre los que establece comunicación actúan a la vez como cliente y como servidor. Además es extensible y su especificación está hecha con XML. Este protocolo se está haciendo muy popular entre las aplicaciones web.

7.1.2 SERVICIO DE MENSAJERÍA

Este es el servicio responsable de la codificación de los mensajes. En él se especifica que contienen los datos que se intercambian entre las maquinas. El lenguaje utilizado para los mensajes es XML. *Extensible Markup Languaje* (XML), traducido, lenguaje de marcas extensible. XML es un metalenguaje extensible de etiquetas. Este lenguaje es una adaptación y simplificación del lenguaje SGML. XML nos permite definir la gramática del propio lenguaje. Por ejemplo una etiqueta que se llame casa: `<casa>...</casa>`. XML se propone como un lenguaje estándar, para el intercambio de información entre diferentes plataformas. Por esta razón XML es ideal para encargarse de la mensajería de los servicios web. Además este lenguaje es independiente del protocolo de transporte. De esta forma tenemos un bajo acoplamiento entre los servicios.

Existen varios protocolos para este servicio, que interactúan con el lenguaje XML para ofrecer el servicio de mensajería. Uno de los más usados es SOAP, pero también hay otros, como, XML-RPC (*remote procedure call mediante XML*) que utilizan el servicio de transporte HTTP exclusivamente. Otro de los protocolos en este servicio es REST (*Representational State Transfer*), una técnica de arquitectura software para sistemas, que maneja tipos de datos distribuidos en la Web. La forma más simple de implementar el protocolo, es enviar páginas XML directamente sobre el protocolo HTTP.

- **XML-RPC** (*Remote Procedure Call* mediante XML). El nombre del protocolo, está formado por dos palabras, separadas por un guion. La primera palabra, hace mención al lenguaje de marcas extensible que hemos visto en el punto anterior. La segunda palabra, RPC (*remote procedure call*), llamada a procedimiento remoto, es un protocolo que permite a una aplicación, ejecutar código en otra máquina sin preocuparse de las comunicaciones de ambos. Por lo tanto, XML-RPC es un protocolo que hace una llamada a un procedimiento remoto (RPC), que codifica las llamadas utilizando el lenguaje XML. Este protocolo fue creado en el año 1998 por el Neoyorkino Dave Winer. Para la comunicación de los mensajes en la Red utiliza HTTP.

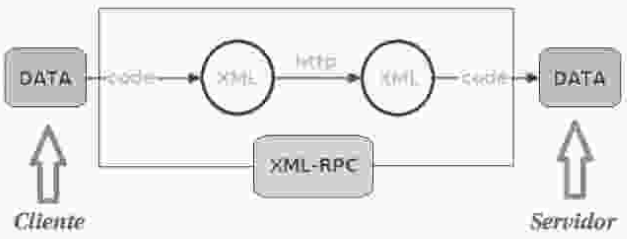


Figura 7.3. *Comunicación del protocolo XML-RPC*

En la figura anterior vemos la comunicación de datos a través del protocolo XML-RPC. En un lado se sitúa el cliente y en el otro la máquina del servidor. El intercambio de datos se realiza mediante XML.

- **SOAP** (*Simple Object Access Protocol*). El protocolo simple de acceso a objetos, es la evolución del protocolo XML-RPC. El XML es el lenguaje de comunicación que utiliza, al igual que su antecesor. Una de las diferencias que aporta este protocolo, es la independencia con el protocolo de trasporte. Como indicamos al comienzo del punto, este protocolo, permite utilizar diferentes protocolos de comunicación, como FTP (*File Transfer Protocol*) o SMTP (*Simple Mail Transfer Protocol*). SOAP permite la ejecución de métodos, que actúan sobre diferentes objetos, en diferentes plataformas, para que estos puedan comunicarse entre sí. Parece complicado conseguir que dos métodos escritos en lenguajes diferentes y que a su vez están en maquinas que tienen sistemas operativos distintos, puedan comunicarse. El abanico de posibilidades es inmenso. Podemos tener clientes programados en cualquier lenguaje y bajo cualquier plataforma. Estos clientes podrán acceder a múltiples servidores, programados en diferentes lenguajes y en múltiples plataformas.

  Para ello SOAP utiliza un lenguaje neutro (descendiente de XML), en el cual define los métodos a invocar y los parámetros que hay que pasar a los métodos. El XML generado por el cliente, se envía a través de la Red. El servidor al recibir los datos (XML generado), los convierte a su lenguaje de programación. Una vez que el servidor ha realizado las operaciones solicitadas en el método, convierte nuevamente el código propio del servidor, en el código neutro de programación, convirtiéndolo así en un documento (XML). Una vez se ha generado, se lo vuelve a enviar al cliente a través de la Red.

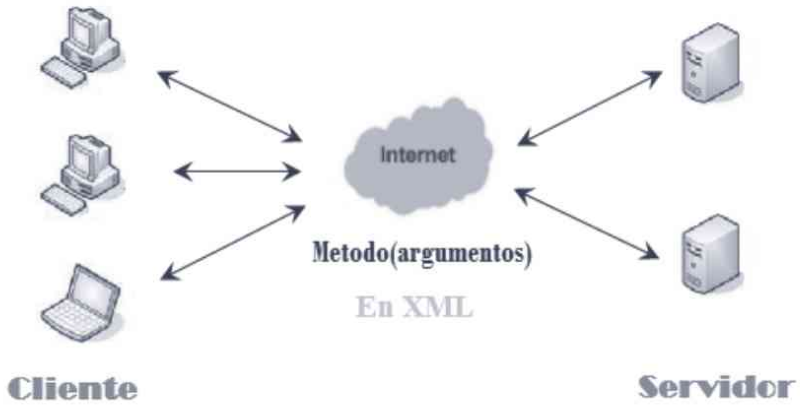


Figura 7.4. *Comunicación del protocolo SOAP*

En la Figura 7.4 vemos como los clientes transforman al lenguaje neutro (XML), los métodos para ser enviados a través de Internet. El servidor convierte los mensajes a su lenguaje, los interpreta y los vuelve a convertir al lenguaje neutro para enviárselo al cliente a través de Internet.

Para conseguir que los mensajes entre el servidor y el cliente se entiendan entre sí, se utiliza un lenguaje neutro descendiente de XML. Este lenguaje genera una descripción de los métodos y funciones permitidas por el servidor. Los clientes deberán adherirse a esta descripción, como si fuera un contrato. La descripción permite que se puedan adherir clientes que estén implementados en diferentes lenguajes, siempre que sigan las normas de la descripción. A la capa correspondiente a esta descripción se la llama servicio de descripción y vamos a estudiarla en el punto siguiente.

7.1.3 SERVICIO DE DESCRIPCIÓN

A la hora de comunicarse, el cliente de un servicio web y el propio servicio web, tienen que llegar a un acuerdo. Deben decidir los detalles del transporte de los mensajes y el contenido de los mismos a través de un documento. Resulta necesario que se especifique la sintaxis y los mecanismos de intercambio de mensajes. De esa forma las aplicaciones involucradas conocerán de forma automática el formato que tienen que utilizar para comunicarse. A esta especificación se le llama descripción del servicio y se usa para describir la interfaz pública de un servicio web. Habitualmente se utiliza WSDL (*Web Services Definition Service*). Este protocolo se encarga de describir la funcionalidad del servicio web, cuando es publicado. El lenguaje utilizado para describir los servicios web es XML.

- **Descripción del servicio en XML-RPC**. Cuando utilizamos este protocolo, necesitamos conocer de antemano las funciones y los métodos que proporciona el servidor. Además es necesario conocer el lenguaje en el que están escritos en el servidor. Este protocolo es más limitado en comparación con WSDL y no existe una metodología para descubrir servicios en una red, como ocurre con su predecesor. En cambio la implementación del mismo es mucho más sencilla, aunque menos escalable.
- **WSDL** (*Web Services Definition Service*). WSDL se encarga de publicar la interface pública de un servicio web. Dicho de otra manera, específica la sintaxis y los mecanismos para el intercambio de mensajes. Existe una primera versión, la 1.0. Esta fue la primera recomendación de W3C. Y de esta se ha pasado a la versión 2.0, que es la recomendación actual de la entidad. WSDL está basado en XML y define los requisitos del protocolo y el formato de los mensajes, para interactuar con el listado de operaciones que proporciona el servidor. Estas operaciones que soporta, se describen de una forma abstracta. Una vez descritas se casan con el protocolo de red elegido (servicio de transporte) y el formato del mensaje (servicio de mensajería). Un cliente que se va a conectar, puede leer el WSDL y visualizar que funciones están disponibles para el servidor y como puede utilizarlas. WSDL tiene una estructura que integra una serie de elementos, que son; los tipos de datos, los mensajes, los tipos de puerto, los *bindigs* (protocolo de comunicación usado), los servicios (conjunto de puertos y direcciones). A continuación mostramos una imagen conceptual de como se comunican el cliente, el WDSL y el servidor.

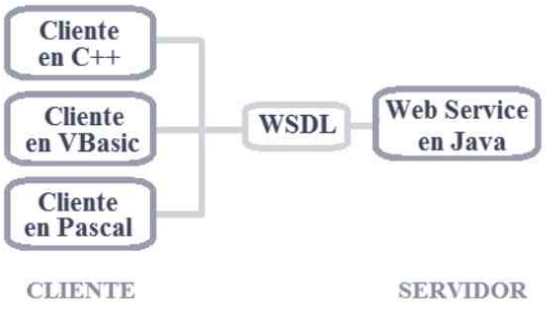


Figura 7.5. *Esquema de comunicación del WSDL*

En la figura anterior vemos como en el WSDL, se describen las funcionalidades que puede ofrecer el servicio web. En el protocolo XML-RPC era necesario conocer las funciones y métodos del servidor. Además era necesario conocer el lenguaje de programación utilizado por el servidor. A través del WSDL (lenguaje neutro), se consigue aislar el lenguaje específico del servicio web. Esto permite el acceso a clientes con diferentes plataformas y lenguajes, a través del lenguaje neutro, sin que estos se preocupen del lenguaje del servicio web.

7.1.4 SERVICIO DE DESCUBRIMIENTO

Hasta ahora hemos visto como funciona y se comunica un servicio web. El punto anterior, nos indica que WSDL utiliza un lenguaje neutro. Se define un lenguaje común, a partir del cual se va generando el código correspondiente adaptado a cada cliente. Por lo tanto, el servicio de definición (WSDL) va a ser reutilizable para todos los clientes que se quieran conectar a él. Una vez que se ha construido una descripción WSDL para un servicio web, este va a poder ser consultado por los posibles clientes del servicio web, para implementar los métodos que permite el servidor. Esta reutilización de código hace necesario un sistema de publicación de los WSDL que estén construidos. El servicio de descubrimiento, es el encargado de esta misión.

Un servicio de descubrimiento **es** el que centraliza un registro común de servicios web, de manera que las empresas que generan servicios web, puedan publicar su localización y descripción. De esta forma podemos descubrir los servicios web que hay disponibles en la Red. El consorcio internacional sin ánimo de lucro OASIS sufraga el principal catalogo de negocios de Internet, al cual se le denomina UDDI.

UDDI (*Universal Description, Discovery and Integration*) es un servicio de directorio donde las empresas pueden registrar y buscar servicios web. Este registro en el catalogo se realiza en XML. UDDI es un marco independiente a la capa del servicio de descripción visto anteriormente. Recordamos que uno de los principales objetivos de las capas de los servicios web, es el bajo acoplamiento entre ellas.

Las características de UDDI son las siguientes:

- Es sinónimo de *Universal Description Discovery and Integration*.
- Es un directorio para almacenar información de servicios web.
- Es un directorio de interfaces de servicio que se describe a través del protocolo WSDL.
- UDDI se comunica a través de SOAP.
- Utiliza los estándares de la W3C y de OASIS.
- Sigue las recomendaciones de la IETF (*Internet Engineering Task Force*), grupo especial sobre ingeniería de Internet.

En un mundo tan globalizado, y en un área tan expandida como es Internet, se hace patente que las empresas compartan sus descripciones de servicios web a través de la Red. Por lo tanto, existen múltiples beneficios a la hora de publicar la descripción de un servicio web.

Las ventajas de utilizar UDDI son las siguientes:

- Descubrir servicios web en línea de su área de negocio.
- Conocer la definición de como habilitar el comercio una vez descubierto el área de negocio.
- Localizar nuevos clientes y dar acceso a los actuales.
- Ampliar la oferta y extender el alcance del mercado.
- Eliminación de barreras, permitiendo la participación de clientes a través de Internet.
- La descripción de los servicios y procesos de negocio, en entorno de programación único, estándar, escalable y seguro.

Los servicios UDDI, se diseñan como una caja negra, de esta forma, se oculta la complejidad del sistema. Permitiendo así una fácil comunicación con los métodos y funcionalidades del servidor. Para que las empresas puedan organizar, publicar la información de los servicios y localizar los ya publicados, UDDI provee de un repositorio en el que se ubican los documentos XML y un esquema. Esto viene definido por un mensaje SOAP para el registro y petición de información.

Los documentos XML se guardan en sistemas UDDI de compañías que aceptan mantener los documentos. A estos documentos se los llama, nodos. Además estas empresas respetan el consorcio UDDI. A los documentos almacenados se los denomina ficheros de registro. Los ficheros de registro constan de las siguientes partes:

- **Paginas blancas**. Esta parte, asemejándose a la funcionalidad de la guía de teléfonos habitual, especifica los datos personales de la empresa propietaria del servicio, como, dirección, contactos, identificación de la empresa y otros identificadores conocidos.
- **Páginas amarillas**. En este bloque, se especifica la categoría industrial. Esta es descrita en base a la taxonomía propuesta por UDDI.
- **Páginas verdes**. Este último bloque, describe la información técnica del servicio web. Esta información la aporta quien publica el servicio web en el UDDI.

Las diferentes capas de la pila de protocolos, son independientes, pero mantienen una relación, para que la conexión entre ellas sea fuerte. Al comienzo de la sección 7.2, veíamos en la Figura 7.2 como la pila se proyectaba como una sucesión de capas que se ordenaban desde el nivel más bajo, el transporte hasta la publicación de los servicios. Una vez desarrollado cada protocolo, tenemos capacidad para visualizar la relación existente en un servicio web. A continuación mostramos un triangulo que especifica esas relaciones.

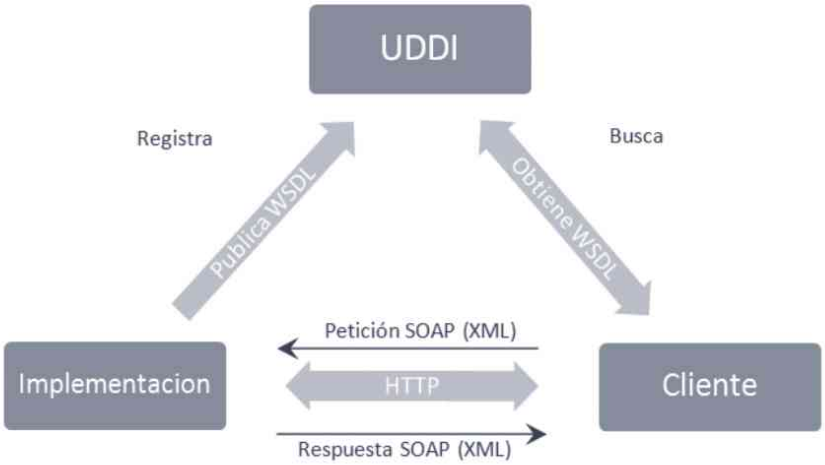


Figura 7.6. *Esquema de la relación de un servicio web*

En la imagen, vemos como en el lado izquierdo se implementa la descripción del WSDL. Una vez realizada la implementación, se publica en el UDDI. El cliente que quiere utilizar un servicio web, busca la especificación del WSDL adaptado a su negocio. Cuando lo encuentra lo obtiene y genera peticiones SOAP, siguiendo la descripción del WSDL. En la zona izquierda de implementación, se han generado previamente las respuestas SOAP, que envían en el lenguaje neutro (XML), los datos al cliente. El protocolo utilizado en este caso es HTTP, aunque podría implementarse para otro protocolo.

7.2 GENERACIÓN DE UN SERVICIO WEB

A la hora de generar servicios web, es habitual el uso de entornos de desarrollo integrados, IDE (en inglés *Integrated Development Environment*) para facilitar la tarea. Estos entornos aglutinan una serie de herramientas que ayudan a en el desarrollo del software. En la creación de servicios web, es especialmente importante disponer de una ayuda a la hora de generar los diferentes tipos de ficheros. Algunas de las razones, son que existen muchas estructuras de código que se generan de forma automática al crear un nuevo fichero. En el siguiente punto vamos a ver cómo se facilitan muchas tareas con estas herramientas. Existen multitud de entornos de desarrollo para la generación de servicios web. Normalmente estos entornos facilitan el uso de uno o varios lenguajes y además incorporan otros módulos como, servidores web, creación de proyectos web, herramientas para la creación de servicios web, etc. Algunos de estos entornos de desarrollo integrado son:

- **Eclipse**. Código abierto, multiplataforma. Para generar servicios web con este entorno Java necesitamos; Tener instalada una versión de JDK y un servidor de aplicaciones (ejemplo, Tomcat). Así como el uso de librerías para la implementación de servicios web (Por ejemplo: axis.jar, commons-discovery.jar, jaxrpc.jar, assj.jar, wsdl4j.jar). Observamos como existen librerías para realizar todas las tareas asociadas con los servicios web. Con Eclipse también se pueden desarrollar servicios web con el lenguaje PHP.
- **NetBeans**. Entorno de desarrollo libre, existen muchos módulos para extenderlo. Uno de los módulos relacionados con los servicios web, *NetBeans Enterprise Pack*, provee de soporte para la creación de aplicaciones orientadas a servicios (SOA), incluyendo herramientas de esquemas XML, un editor WSDL y un editor BPEL para servicios web. Permite utilizar más de un servidor de aplicaciones como GlassFish o Apache Tomcat.
- **Microsoft Visual Studio**. Es un entorno de desarrollo para sistemas operativos Windows. Soporta varios lenguajes como Visual C++, Visual C#, Visual J#, ASP.Net y Visual Basic .Net. Permite desarrollar servicios web en cualquiera de los entornos que soporta la plataforma .Net.

Para el desarrollo del ejemplo que vamos a mostrar vamos a utilizar el entorno de desarrollo NetBeans en su versión 7.0.1 como plataforma de desarrollo. A continuación visualizamos una imagen del aspecto que tiene el entorno de trabajo de NetBeans.

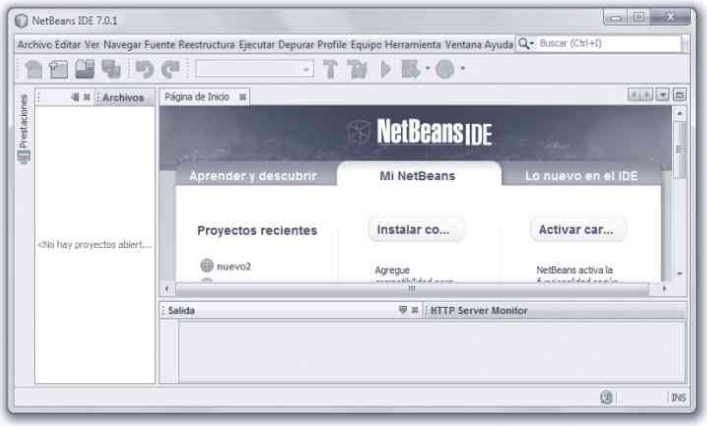


Figura 7.7. *Imagen del NetBeans*

El lenguaje que vamos a utilizar para genera el servicio web, es Java. Lo primero que tenemos que hacer para generar un servicio web, es crear un proyecto o aplicación web. Este será el encargado de llamar a los procesos que consuman los recursos del servicio web o generar nuevos procesos.

7.2.1 CREACIÓN DE UN PROYECTO WEB

Para crear un proyecto web a través del IDE de NetBeans pulsamos en el programa NetBeans IDE 7.0.1, **Archivo → proyecto nuevo**. Seleccionamos **Java Web → Web Application**.

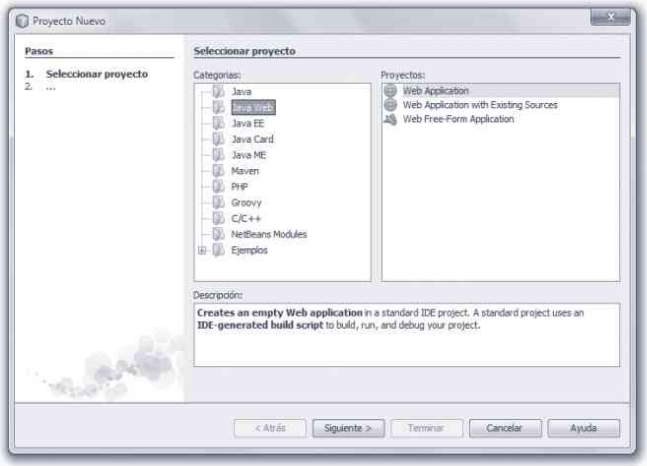


Figura 7.8. *Creación de un proyecto web nuevo*

En la ventana siguiente indicamos en *Project Name*, WebService como nombre del proyecto. En el caso de querer cambiar la ruta de localización del proyecto, pulsaremos sobre **Browse...** para seleccionar otra ruta.

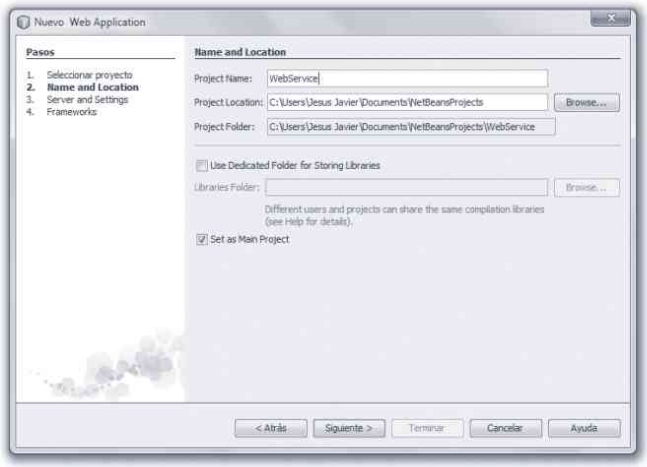


Figura 7.9. *Inserción del nombre y ruta en la creación de aplicación web*

Pulsamos **Siguiente** y seleccionamos primero el *Server*: En nuestro caso vamos a elegir **GlashFish server 3.1**. Luego elegimos en *Java EE Versión*: El Java EE 6 Web. Es importante elegir la versión 6 para poder implementar los servicios web.

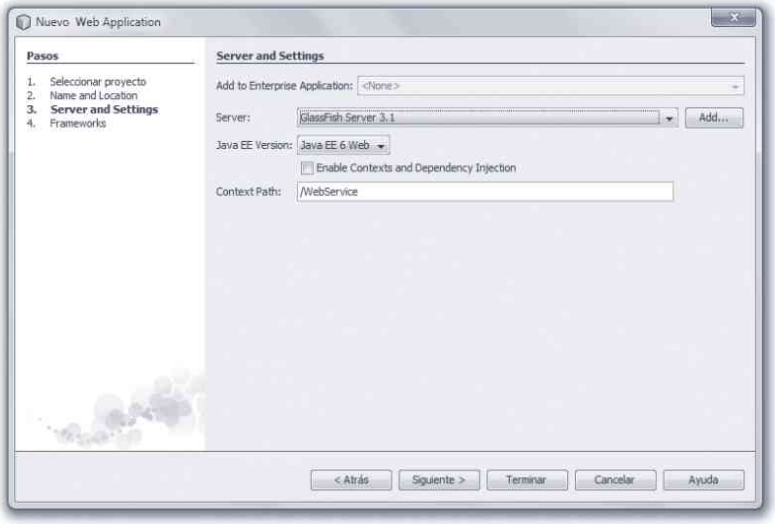


Figura 7.10. *Inserción del servidor y opciones*

Volvemos a pulsar **Siguiente** y no seleccionamos ninguna casilla de las que aparecen. No queremos incluir ningún *framework* para no confundir los conceptos propios de los servicios web con arquitecturas más específicas.

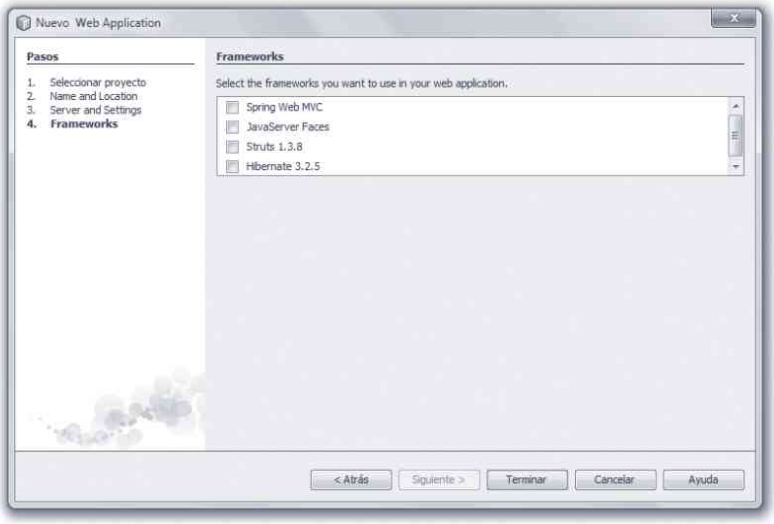


Figura 7.11. *Selección de frameworks*

Una vez que visualizamos la ventana pulsamos en terminar, para crear la aplicación web.

Al pulsar terminar se crea un proyecto web, dentro de él se genera un JSP que se llamara `index.jsp`.

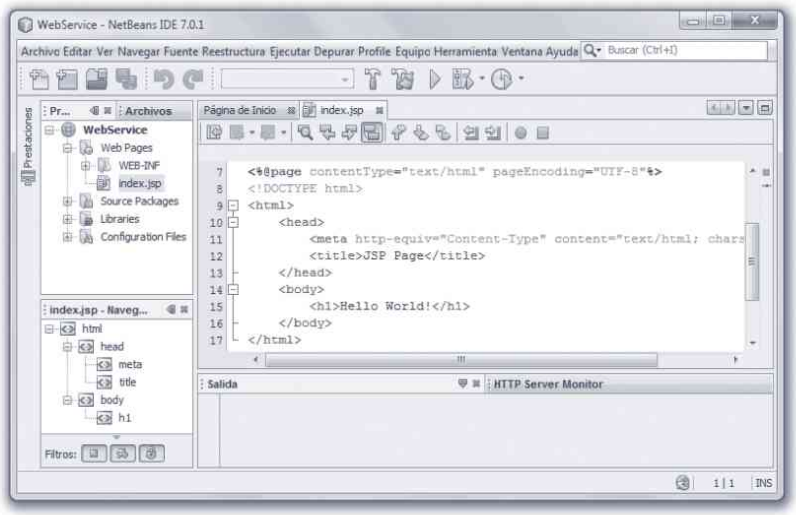


Figura 7.12. *Mostrar* `index.jsp`

Una vez que hemos creado el proyecto, vamos a probar que funciona correctamente. Para realizar la prueba de funcionamiento, seleccionamos con el botón derecho del ratón el directorio raíz del proyecto, en nuestro caso `WebService`. Una vez se despliega el menú pulsaremos **Ejecutar**.

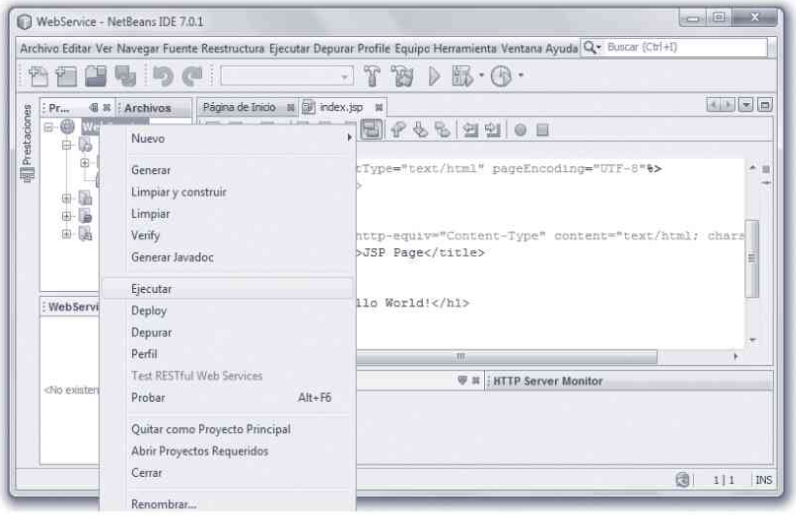


Figura 7.13. *Ejecutar proyecto web*

En este paso se empaqueta el proyecto, se realizan las compilaciones necesarias y se inicia el servidor web, en nuestro ejemplo el servidor es GlashFish. Si todo ha ido correctamente, se deberá abrir una ventana del navegador que tiene por defecto el sistema operativo mostrando la página `index.jsp`. Recordemos que la página `index.jsp` se había creado como base en la creación del proyecto web. La página tendrá el siguiente aspecto:

Figura 7.14. *Página* `index.jsp`

Una vez comprobamos que se visualiza la página `index.jsp` aseguramos que el proyecto se ha creado correctamente y funciona. Es ahora cuando estamos preparados para generar un servicio web.

7.2.2 CREACIÓN DE UN SERVICIO WEB

Para crear un servicio web en NetBeans, tenemos que seguir una serie de pasos. Lo primero será situarnos con el ratón sobre el directorio principal del árbol del proyecto, en nuestro caso se llama `WebService`. Pulsamos el botón derecho, en **Nuevo → Web Service**.

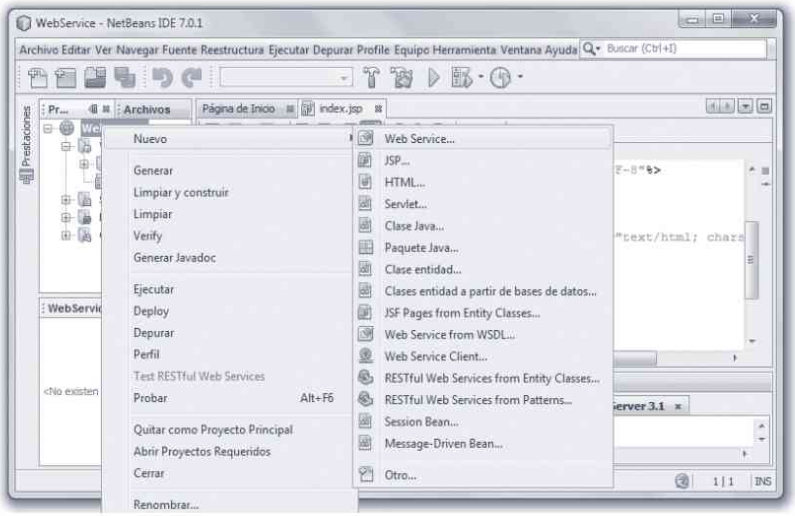


Figura 7.15. *Creación del servicio web*

Cuando se abra la ventana, introducimos en nombre de clase, vamos a llamar a la clase y al servicio web `WebServ1`. En el apartado paquete, crearemos un paquete para que los servicios web se almacenen en él. El paquete se llamará `ServiciosWeb`.

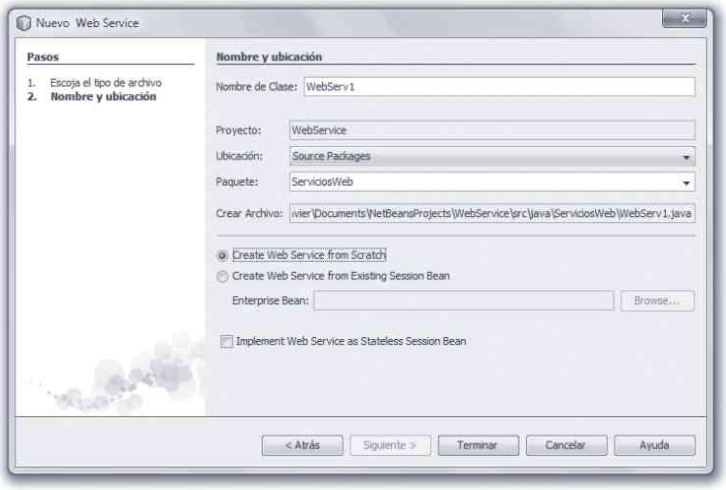


Figura 7.16. *Inserción de datos del servicio web*

Una vez que insertamos los datos pulsamos en terminar. NetBeans crea una clase con el nombre del *Web Services*. La clase incorpora los paquetes necesarios para poder implementar el servicio web. En el siguiente punto vamos a ver cuáles son las diferentes partes que aparecen en el código. El proyecto en NetBeans, quedaría de la siguiente manera.

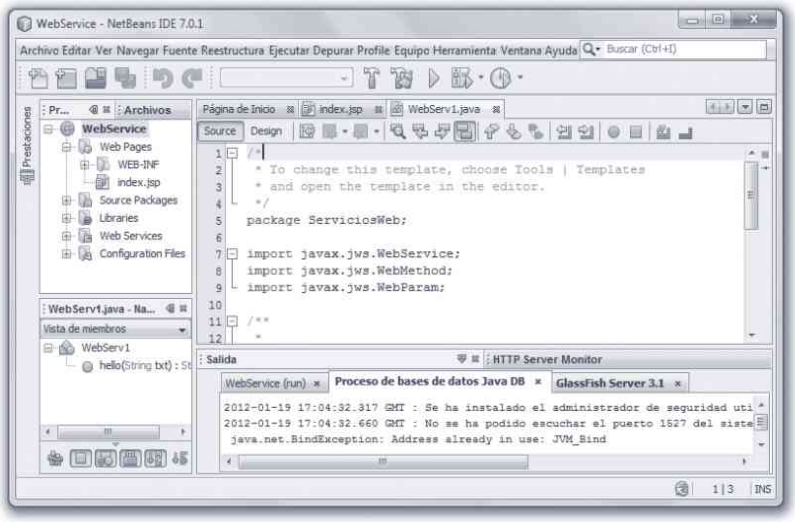


Figura 7.17. *NetBeans con el servicio web creado*

El código correspondiente al servicio web que hemos creado tiene una estructura tipo. En el siguiente párrafo vemos la estructura que NetBeans genera por defecto cuando creamos un servicio web.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ServiciosWeb;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author Jesus Javier
 */
@WebService(serviceName = "WebServ1")
public class WebServ1 {

    /** This is a sample web service operation */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt)
    {
      return "Hello " + txt + " !";
    }
}
```

En el código anterior, la primera línea indica al paquete al que pertenece la clase del servicio web que hemos creado. En nuestro caso pertenece al paquete `ServiciosWeb`. A continuación vemos cómo se importan algunas clases que necesitamos para poder implementar los servicios web.

La primera clase, `javax.jws.WebService` se encarga de las tareas de los servicios web, por ejemplo del nombre que se le asigna al servicio web.

La clase `javax.jws.WebMethod`, posibilita el uso de los métodos que se crean en la clase.

La clase `javax.jws.WebParam` se encarga de los parámetros del servicio web.

La primera línea que comienza por `@WebService()` indica cuál es el nombre del servicio web con la estructura mostrada en el código situado a continuación.

```
@WebService(serviceName = "WebServ1")
```

La estructura que sigue al nombrado del servicio web es siempre la misma. Mostramos en negrita en el código anterior el valor que indicará cómo se llamará cada servicio web.

A continuación declaramos la clase, en nuestro caso de tipo público, que llamaremos (`WebServ1`) como ocurre en Java con todas las clases, como el nombre del fichero.

La siguiente línea muestra el método que contiene el servicio web, en el caso de nuestro ejemplo NetBeans ha creado un método llamado `hello`. Antes de implementar el método según la estructura Java es necesario indicar el nombre del método con la estructura mostrada en el código.

```
@WebMethod(operationName = "hello")
```

Observamos como para los métodos el nombre que se indica después de la arroba es `WebMethod`. En el interior se le asigna `operationName` al valor del método `hello`.

La primera parte del código correspondiente a la implementación del método, es similar a la generación de las clases en Java, la nota asonante viene en la anteposición de la estructura mostrada en negrita. Al igual que hicimos con la declaración de la clase del servicio web y el método asociado a esta debemos seguir una estructura previa. Mostramos en negrita la estructura `WebParam(name = "name")` que hay que anteponer.

```
public String hello(@WebParam(name = "name") String txt) {
```

El parámetro que recibe el método `hello` es `txt`. En el caso de nuestro ejemplo, el método devuelve una cadena de texto más el valor de la variable `txt` y un signo de admiración cerrado al final. La forma que tiene el método del servicio web de devolver el valor es utilizando `return`, igual que se hace habitualmente en Java.

Una vez que hemos estudiado las diferentes partes de las que consta un servicio web en Java, vamos a realizar un test del servicio web para comprobar que este funciona correctamente. NetBeans provee de la utilidad para realizar esta tarea. Desplegamos el proyecto, en el directorio `Web Services`, pulsamos con el botón derecho del ratón en el menú **Test Web Services**.

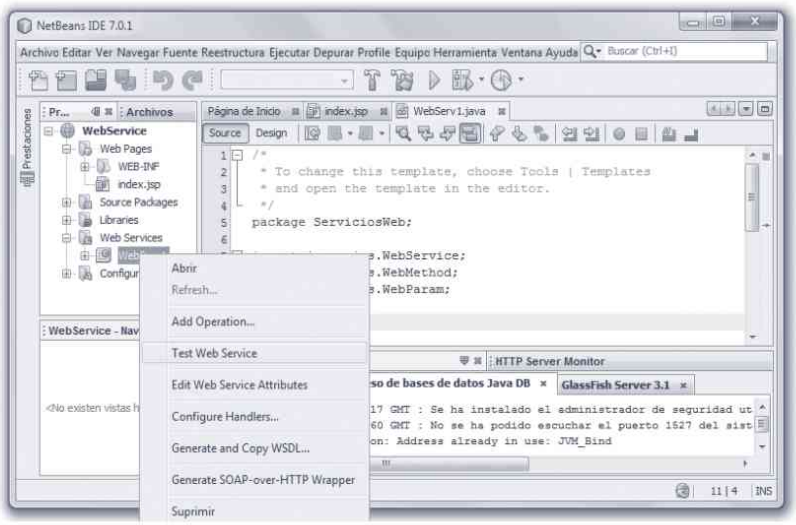


Figura 7.18. *Menú del test del servicio web*

A continuación se deberá abrir una ventana del navegador mostrando una interfaz para probar el servicio web. La ruta está compuesta por la dirección IP (en nuestro caso *localhost*), el puerto del servidor GlassFish (el 8084), el nombre del proyecto (*WebService*) y el nombre del servicio web seguido de una interrogación y la palabra "Tester" (`WebServ1?Tester`).

Figura 7.19. *Menú del test del servicio web*

En la imagen anterior podemos ver una página donde podremos probar que el parámetro que recibe el método se recibe y devuelve correctamente junto con el resto de la frase que indicábamos en el código del servicio web.

En primer lugar nos encontramos el nombre del servicio web con un texto que nos indica *tester*.

A continuación se muestra un enlace al fichero de descripción WSDL. Este fichero lo veremos en el siguiente punto del tema.

En el siguiente bloque aparecen los métodos implementados en la clase. En nuestro caso solo tenemos uno. Este método solo recibe un parámetro de tipo `String` como se indica en el test. Para probar que el servicio funciona, deberemos introducir un valor de tipo `String` en el campo de entrada. Se muestra entre paréntesis para que la sensación de estar pasando un valor al método sea más real. Nosotros vamos a introducir la palabra "mundo" para realizar la comprobación. Una vez introducida pulsamos en el botón gris **hello**.

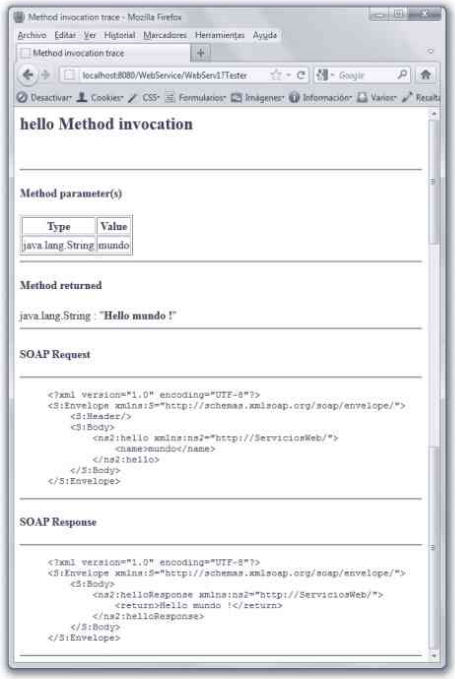


Figura 7.20. *Resultado del test del servicio web*

En la imagen anterior podemos visualizar el resultado del la prueba del servicio web. En primer lugar, visualizamos el texto en negrita "hello Method invocation". Este nos indica que se ha invocado al método `hello`.

En el siguiente bloque aparece una tabla con el tipo de parámetro enviado al método y el valor asociado. En nuestro caso el tipo es `String` y el valor `mundo`.

El tercer bloque muestra el valor devuelto por el método. Primero indica el tipo de dato devuelto, *string* y después de los dos puntos en negrita el texto devuelto, "Hello mundo !".

Los dos últimos bloques corresponden a la petición y la respuesta de los mensajes SOAP, puesto que es este el servicio de mensajería que estamos utilizando. A continuación explicamos detalladamente el proceso de petición y respuesta de mensajes:

- **Petición del mensaje**. El primero de ellos *SOAP Request*, es la petición del mensaje. En la primera línea indica la versión y codificación del XML. El siguiente elemento (`Envelope`), es el elemento raíz del mensaje SOAP. Este elemento define al documento XML como un mensaje SOAP. El `xmlns` define el espacio de nombres de codificación SOAP que debe respetar el XML. El `Header` corresponde a la cabecera del mensaje, en este caso como no contiene nada se cierra así misma. El `body` contiene el método, en nuestro caso `hello`. Entre las etiquetas correspondientes al método viaja el parámetro enviado en la petición. En nuestro caso el valor del parámetro es `mundo`. En el código siguiente podemos verlo:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:hello xmlns:ns2="http://ServiciosWeb/">
      <name>mundo</name>
    </ns2:hello>
  </S:Body>
</S:Envelope>
```

- **Respuesta del mensaje**. En la respuesta del mensaje, las dos primeras etiquetas del XML realizan la misma función que en la petición. La siguiente etiqueta muestra el cuerpo del mensaje a través de la etiqueta `Body`. `helloResponse`, es en este caso la respuesta al método invocado anteriormente. Para ofrecer la respuesta se utiliza la etiqueta `<return></return>` con el mensaje "Hello mundo !".

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:helloResponse xmlns:ns2="http://ServiciosWeb/">
      <return>Hello mundo !</return>
    </ns2
:helloResponse>
  </S:Body>
</S:Envelope>
```

Estos son los mensajes utilizados para comunicar dos sistemas con plataformas diferentes, estén o no en lenguajes de programación diferentes. Por un lado está la petición, como hemos visto, que se envía vía HTTP y por otro lado está la respuesta que también es enviada por este método.

7.2.3 AÑADIR MÉTODOS A UN SERVICIO WEB

En el punto anterior hemos visto como generar un servicio web por defecto. El servicio explicado es el que crea NetBeans por defecto. Los objetivos de los servicios web son muy variados y en cada caso vamos a tener que implementar una casuística determinada. Por esta razón es necesario comprender como podemos ampliar y modificar la funcionalidad de un servicio web. Si trabajamos con NetBeans, podemos hacer uso de la utilidad que permite la inserción de nuevos métodos en un servicio web. Esta utilidad facilita enormemente la implementación puesto que la base de la estructura se genera de una forma automática. A continuación, vamos a ver cómo añadir un método en Netbeans utilizando las herramientas de las que nos provee el asistente. Nos situamos en el programa NetBeans, en el proyecto que hemos creado anteriormente. A continuación vamos a añadir un método nuevo que lea la hora del sistema y la devuelva por pantalla. Para ello tenemos que abrir el código correspondiente al servicio web que creamos anteriormente. El servicio se llama `WebService1.java`. Hacemos un doble clic y cuando visualicemos el código, pulsaremos en el botón **Design**.

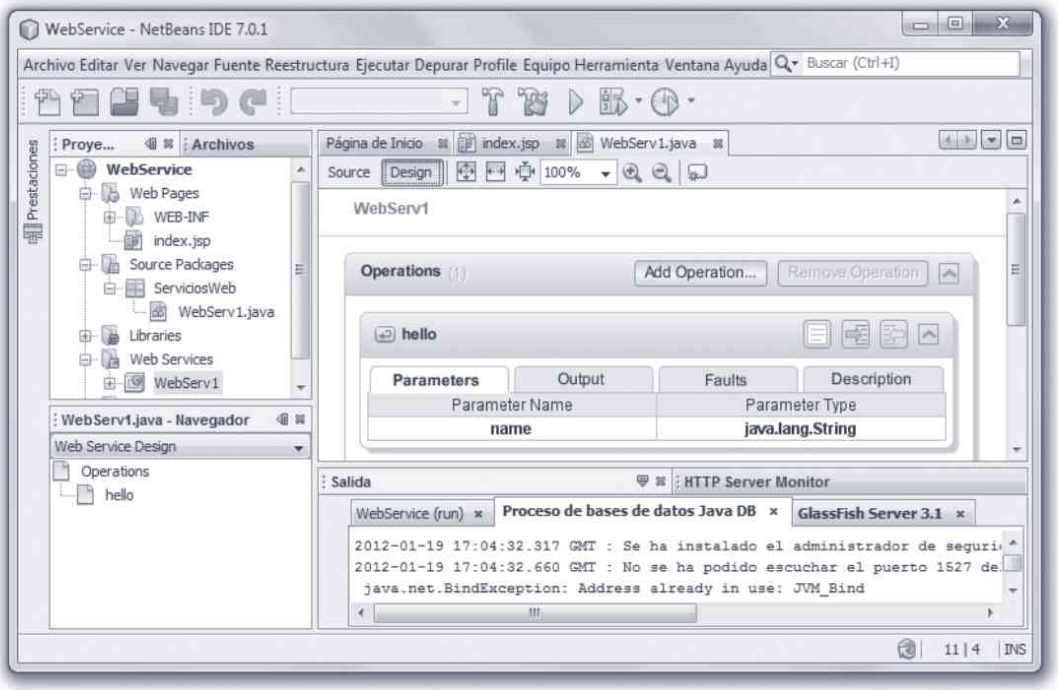


Figura 7.21. *Añadir método en un servicio web*

Una vez pulsado el botón **Design** para cambiar el tipo de vista, pulsamos en el botón **Add Operation...**. La anterior operación nos mostrará la siguiente pantalla:

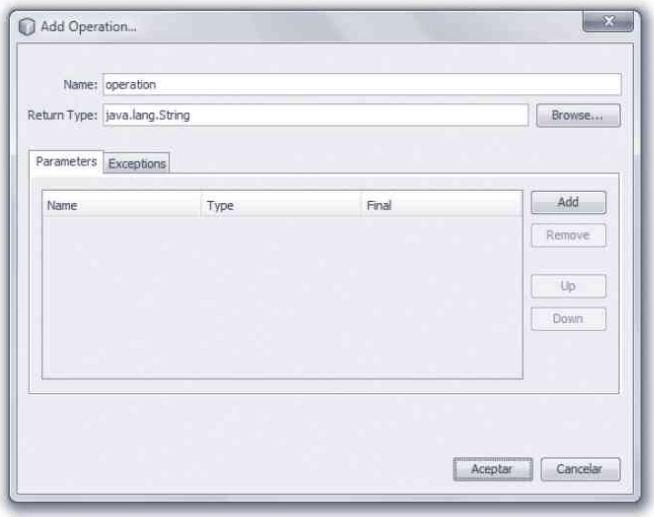


Figura 7.22. *Introducir nombre y parámetros en el método*

En esta pantalla vamos a indicar el nombre del método. Por defecto nos indica "operation", pero nosotros vamos a cambiar el nombre y vamos a poner `MuestraHora`. En los *Parameters* añadiremos un parámetro pulsando en el botón **Add**. Este parámetro se llamará `hora`. Observamos que existe una pestaña al lado de *Parameters* en la que indica *Exceptions*. En esta pestaña podemos añadir excepciones al código. El aspecto de la ventana después de añadir estos valores quedará como en la imagen siguiente:

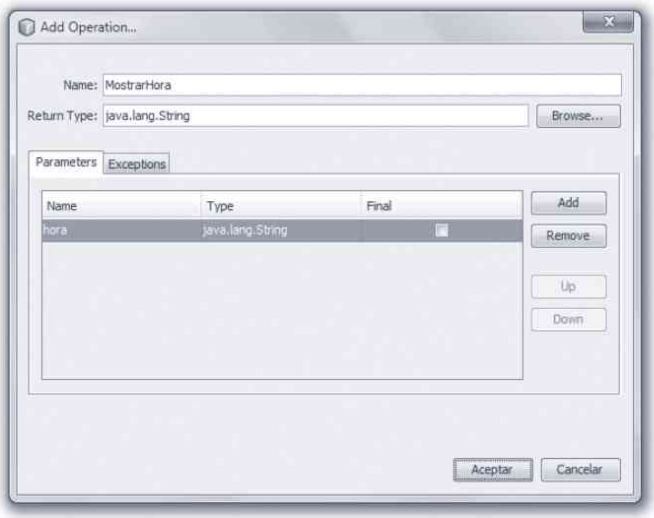


Figura 7.23. *Añadir nombre del método y parámetros*

Una vez hemos introducido el nombre del método y los parámetros, pulsamos **Aceptar**. Automáticamente, Java crea un método llamado `MostrarHora`. Vamos a retocar el código generado automáticamente para que en el caso de que se envíe a través del método el parámetro `Madrid`, se muestre la fecha del sistema. En el caso de que se envíe a través del método otro valor indicará que no puede mostrarse la hora.

```
package ServiciosWeb;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import java.util.Date;

@WebService(serviceName = "WebServ1")
public class WebServ1 {

  /** This is a sample web service operation */
  @WebMethod(operationName = "hello")
  public String hello(@WebParam(name = "name") String txt)
  {
    return "Hello " + txt + " !";
  }

  /**
   * Web service operation
  */
  @WebMethod(operationName = "MostrarHora")
  public String MostrarHora(@WebParam(name = "hora") String
    hora) {
    //TODO write your implementation code here:

    if(hora.equals("Madrid")){
      Date fecha = new Date();
      return "FECHA--> "+fecha;
    }else{
      return "...No puede mostrarse la hora...";
    }
  }
}
```

El código que hemos modificado se muestra en negrita. No olvidemos incluir la clase `java.util.Date`, necesaria para el manejo de la fecha del sistema. Una vez hemos realizado las modificaciones, volveremos a pulsar con el botón derecho del ratón en el nombre del servicio web, en nuestro caso `WebServ1`. Cuando salga el menú pulsaremos **Test Web Service**. Observamos como ahora aparece en la interfaz de prueba los dos métodos creados para el servicio web. En primer lugar aparecerá el que se genero de forma automática y debajo el que hemos creado ahora. Procederemos a introducir `Madrid` y visualizaremos la fecha del sistema. En el caso de incluir otro valor, visualizaremos el mensaje "…No puede mostrarse la hora…".

A partir de este asistente podemos crear tantos métodos como queramos y modificar su funcionalidad o bien podemos generar a mano los métodos siguiendo la estructura que hemos visto en los casos anteriores.

7.3 DESCRIPCIÓN DEL SERVICIO

Como indicábamos en los primeros puntos del tema, este servicio es el encargado de describir los servicios web y de cómo acceder a ellos. Este servicio está basado en XML, de esta forma permite un entendimiento entre distintas tecnologías y arquitecturas, puesto que es un lenguaje común.

WSDL es el lenguaje que la organización W3C recomienda para la describir un servicio web. Este lenguaje no solo se utiliza para describir servicios web, también para localizarlos.

¿SABÍAS QUE...?

WSDL es un documento escrito en XML. El documento describe el servicio web. En él se especifican tanto la ubicación del servicio como las operaciones (métodos) que el servicio permite realizar.

Para que los diferentes intermediarios en el servicio web, puedan comunicarse a través del WSDL, es necesario tener unas normas comunes. Existen unos elementos que representan una definición común. A continuación vamos a visualizar una tabla con los elementos principales de un WSDL y su definición.

Tabla 7.1 Elementos y definición de un WSDL

Elemento	Definición
`<definitions>`	Agrupa el WSDL de principio a fin.
`<types>`	Especifica los tipos de datos que utiliza el servicio web.
`<message>`	Especifica los mensajes que utiliza el servicio web.
`<portType>`	Especifica las operaciones (métodos) que puede realizar y los mensajes que están involucrados.
`<binding>`	Especifica los protocolos de comunicación utilizados por el servicio web.

A continuación vamos a generar el WSDL del ejemplo anterior a través de las herramientas de las que nos provee el entorno de trabajo.

En el punto anterior, hemos visto como generar un servicio web. NetBeans cuando genera un servicio web, también crea la descripción del servicio (WSDL). Una de las formas acceder a la descripción del servicio WSDL es a través del enlace que mostrábamos en el punto anterior (en la página de Test Web Service). Cuando pulsamos con el botón derecho del ratón sobre el servicio web, en nuestro caso `WebServ1`, nos muestra el menú. Pulsamos sobre *Test Web Service*. En la parte superior aparecerá un enlace en el que indica "WSDL File". Si pulsamos en él se abrirá el fichero de descripción. La composición de la URL que abre la descripción del servicio web es la siguiente: *http://localhost:8080/WebService/WebServ1?WSDL*

Vamos a describir los campos de la URL que abre en el navegador el fichero de descripción WSDL.

- **http**. Protocolo de transferencia de hipertexto (del servicio web).
- **localhost**. Dirección de *loopback* (dirección IP pública fuera de intranet).
- **8084**. Puerto que utiliza GlassFish Server.
- **WebServ1**. Nombre del proyecto.
- **WebServ1?WSDL**. Nombre del servicio web. La convención es que la descripción del servicio web termine con una interrogación y con la cadena de texto "WSDL". Al poner la URL en el navegador visualizaremos la descripción del servicio web.

A partir de esta descripción que aparece en el navegador dispondremos de la información necesaria para conocer el servicio web. Así este servicio puede ser consumido desde cualquier otra plataforma o desde cualquier lenguaje de programación. El WSDL define el acuerdo que se establece entre un cliente y el servicio web. En este acuerdo se establecen modelos de mensajería que el servicio web define y admite. Un modelo de mensajería es el envío desde el cliente de una ciudad, en nuestro caso "Madrid", y el servicio web envía un mensaje que muestra la fecha del sistema. El modelo WSDL indica de una forma genérica y entendible lo que el consumidor del servicio web puede esperar que suceda cuando se envía un mensaje con formato correcto al servicio web. En la Figura 7.24, vemos como el WSDL está compuesto por los elementos descritos en la Tabla 7.1. Recordemos la funcionalidad que habíamos creado en el ejemplo anterior de servicio web:

- **Servicio web "WebServ1"**. Clase pública. Nombre de la clase `WebServ1`.
- **Método hello**. Recibe un parámetro de entrada llamado `txt`. Devuelve la cadena de texto "Hello" más la variable recibida en el método `txt` y al final un signo de admiración.
- **Método MostrarHora**. Recibe el parámetro "hora". Si hora es igual a "Madrid", devuelve la fecha del sistema (fecha del servidor). Si la "hora" no es igual a "Madrid", devuelve la cadena de texto "...No puede mostrarse la hora...".

Figura 7.24. *WSDL mostrado en el navegador*

En la imagen del código anterior visualizamos la descripción del servicio. A continuación vamos a analizar cada uno de los bloques que aparecen con respecto a la Tabla 7.1 que veíamos en páginas anteriores.

El elemento `<definitions>` agrupa el WSDL. Será la primera y última etiqueta a interpretar. En el atributo `targetNameSpace` se incluirá la URL del proyecto. En el atributo `name` aparece el nombre del servicio web.

```
<definitions targetNamespace="http://ServiciosWeb/"
  name="WebServ1">

  ...Todas las etiquetas...

</definitions>
```

El elemento `<types>` define los tipos de datos que son utilizados por el servicio web. Para ello hace uso de un esquema XSD.

```
<types>
  <xsd:schema>
   <xsd:import namespace="http://ServiciosWeb/"
     schemaLocation=
     "http://localhost:8080/WebService/WebServ1?xsd=1"/>
  </xsd:schema>
</types>
```

Si introducimos la URL situada en el `schemaLocation` en una ventana del navegador, se mostraran los tipos de datos asociados. Observamos en la siguiente imagen (Figura 7.25), que se define en el XSD un nombre para el envío del método y otro para la respuesta. En el caso de la hora, el método de envío `MostrarHora` lleva asociado su parámetro de entrada de tipo `String` llamado `hora`. La respuesta del método `MostrarHora` se define como `MostrarHoraResponse`. Este método lleva asociado un valor de devolución (`return`), de tipo `string`.

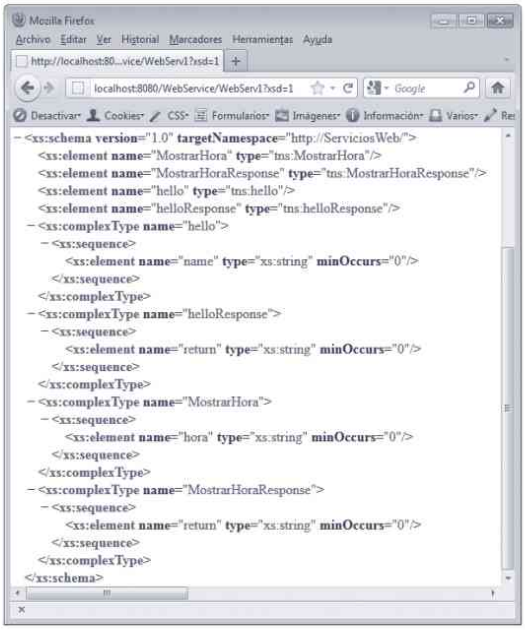


Figura 7.25. *WSDL mostrado en el navegador*

En el XSD mostrado en la imagen se muestran los tipos de datos de los dos métodos generados en el ejemplo con el que estamos trabajando. Es importante tener en cuenta que existen un tipo de datos asociado para el envío del método y otro para la respuesta.

El elemento `<mensajes>` define los elementos de datos de una operación. Cada mensaje puede consistir en una o más partes. Cada una de las partes de los mensajes, son el equivalente a los parámetros de la llamada a una función, en los lenguajes de programación tradicionales.

```
<message name="hello">
  <part name="parameters" element="tns:hello"/>
</message>

<message name="helloResponse">
  <part name="parameters" element="tns:helloResponse"/>
</message>

<message name="MostrarHora">
  <part name="parameters" element="tns:MostrarHora"/>
</message>

<message name="MostrarHoraResponse">
  <part    name="parameters"
    element="tns:MostrarHoraResponse"/>
</message>
```

En nuestro caso tenemos 4 mensajes. Dos para cada uno de los métodos implementados en el servicio web.

El elemento `<portType>` es el más importante del WSDL. En él se describen los servicios web. Están definidas las operaciones que se pueden realizar. También están descritos los mensajes involucrados.

```
<portType name="WebServ1">
  <operation name="hello">
    <input  wsam:Action=
      "http://ServiciosWeb/WebServ1/helloRequest"
      message="tns:hello"/>
    <output wsam:Action=
      "http://ServiciosWeb/WebServ1/helloResponse"
      message="tns:helloResponse"/>
  </operation>

  <operation name="MostrarHora">
    <input wsam:Action=
      "http://ServiciosWeb/WebServ1/MostrarHoraRequest"
      message="tns:MostrarHora"/>
    <output wsam:Action=
      "http://ServiciosWeb/WebServ1/MostrarHoraResponse"
      message="tns:MostrarHoraResponse"/>
  </operation>
</portType>
```

Podemos observar en el código anterior como existen dos operaciones correspondientes a las acciones del servicio web `hello` y `MostrarHora`.

El elemento `<binding>` define el formato del mensaje. También se encarga de detalles del protocolo para cada puerto.

```
<binding name="WebServ1PortBinding" type="tns:WebServ1">
  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="document"/>

    <operation name="hello">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>

    <operation name="MostrarHora">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
</binding>
```

Observamos en el código anterior como se define el `style` como "document" y en las operaciones los campos se definen como literales para las entradas (`input`) y para las salidas (`output`).

Este documento tiene esa especie de contrato que tienen que firmar el servidor que ofrece el servicio web y el cliente que los consume. Por lo tanto, para poder utilizar recursos de un servicio web, necesitamos conocer la descripción del servicio web y acogernos a las condiciones que nos exige este, sabiendo interpretar cada uno de los elementos que hemos visto en este punto.

7.4 INTERFAZ DE UN SERVICIO WEB

En los puntos anteriores hemos visto cómo generar un servicio web y cómo crear la descripción del servicio a través del documento WSDL. El último paso para poder terminar el ciclo de vida de un servicio web, es que un cliente interprete el documento de descripción (WSDL) e implemente el código necesario para invocar a los métodos permitidos en este documento. A esta tarea, se la llama consumir el servicio. Para consumir un servicio web, el cliente debe tener implementado el código necesario para realizar las llamadas a los métodos y hacer alguna llamada a estos con éxito.

Como hemos podido comprobar a la hora de generar clases y documentos XML para servicios web, la complejidad del desarrollo aumenta considerablemente, comparado con las habituales clases de java, JSP y demás tipos de ficheros de otras tecnologías de servidor. Habitualmente las relaciones entre ficheros son muy claras y estos gozan de cierta independencia. El acoplamiento entre módulos es fácilmente comprensible. En cambio, en los servicios web, existen muchos ficheros que tienen una relación fuerte entre sí. Comenzar a realizar de cero una interface de un servicio web puede resultar una tarea muy tediosa y no demasiado productiva. Netbeans al igual que otros entornos de trabajo que permiten desarrollar servicios web, tiene un asistente para crear la interfaz de un servicio web a partir de su WSDL.

Antes de comenzar a desarrollar la interfaz del cliente, conviene saber que algunos autores consideran que los servicios web deberían comenzarse a construir en el orden inverso al que lo hemos realizado nosotros en este capítulo. En realidad tiene lógica pensar que es más adecuado crear la interfaz para después implementarla. Este es el primer paso a la hora de analizar un desarrollo orientado al cliente. Primero se mantienen conversaciones y reuniones con el cliente para definir qué es lo que quiere desde un punto de vista funcional y visual. En el caso de comenzar a desarrollar un servicio web comenzando por el servidor y terminando por la interfaz del cliente, puede ocurrir que el cliente tenga alguna incompatibilidad a la hora de enviar o recuperar variables. Por ejemplo si los tipos de datos no coinciden en el lenguaje de origen y de destino. Y nos podemos encontrar con alguna inconsistencia que perjudique o impida al cliente desarrollar la interfaz del servicio web. La forma de solucionar problemas de incompatibilidades, es ceñirse lo más posibles a los estándares que apoya la organización W3C, con el fin de que todas las herramientas y tecnologías trabajen con el fin de estandarizar al menos la forma de comunicarse entre los diferentes lenguajes y plataformas.

Nosotros pensamos que un servicio web debe dar servicio a la totalidad o mayoría de los potenciales clientes que pueda haber en el mercado. Por esta razón justificamos la creación de un servicio web en el orden en el que lo hemos expuesto. Comenzando en la implementación en el servidor, posteriormente con la creación del WSDL e implementación del cliente a partir del WSDL. Lógicamente es necesario crear unos requisitos que debe cumplir el servicio web a nivel de cliente, pero desde un punto de vista funcional, nunca asociado a ninguna tecnología. Esto ya ocurre en los modelos de negocio de los proyectos web, en la toma de requisitos e incluso en la fase de análisis de los proyectos. Recordemos que hasta que no llega la fase de diseño, no se debe implicar a ninguna tecnología concreta en el proyecto.

7.4.1 CREACIÓN DE LA INTERFAZ DE USUARIO

Para crear la interfaz del servicio web, vamos a utilizar el entorno de trabajo NetBeans. Lógicamente podemos realizar esta implementación con cualquier otro lenguaje de programación, plataforma y entorno de trabajo. La única condición es que el entorno de trabajo permita la implementación de la interfaz de usuario a través de WSDL.

Para poder implementar la interfaz de usuario, primero necesitamos crear un proyecto web. Como hemos visto en los puntos anteriores la forma de crear un proyecto web en NetBeans, no indicaremos como se hace paso a paso. Podemos consultarlo en la sección anterior como crear un proyecto web en NeBeans. Creamos un proyecto web, independiente del que ya teníamos creado. En este caso lo vamos a llamar `WebCliente`. Dentro del proyecto, pulsamos botón derecho, "Nuevo"→ "Web Service Client...".

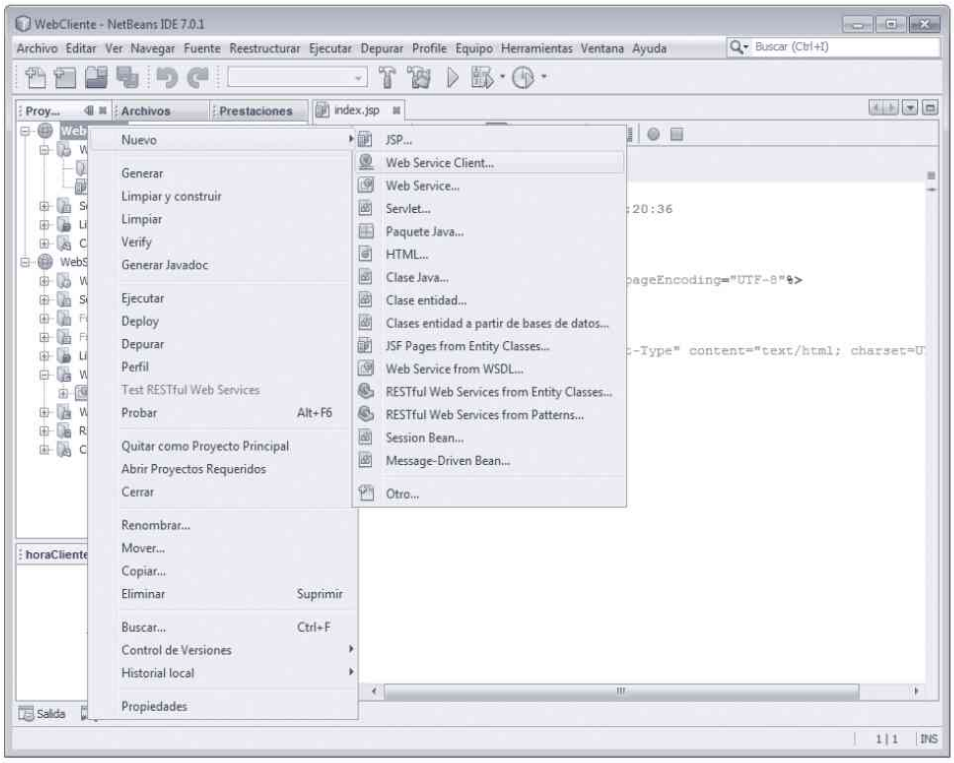


Figura 7.26. *Creación de un Web Service Client*

Una vez visualizamos el menú, hacemos clic con el botón izquierdo del ratón. Nos mostrará una nueva ventana donde podremos especificar la fuente del WSDL, elegir un proyecto donde se encuentre un fichero local o una URL.

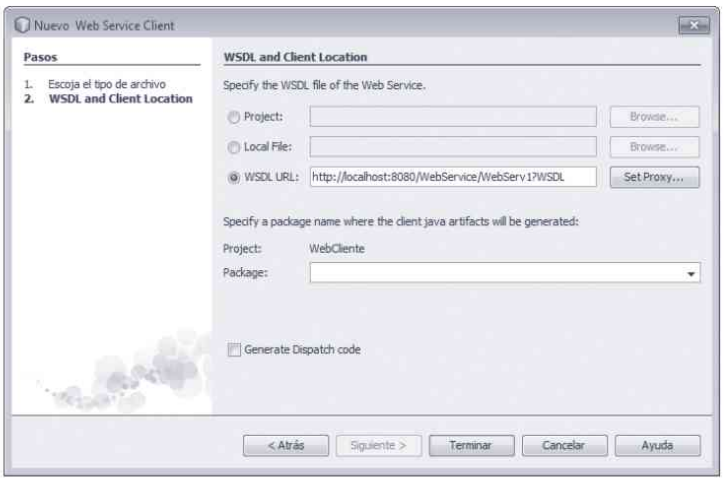


Figura 7.27. *Inserción del WSDL*

En la imagen anterior debemos pulsar en el último botón de radio, que indica "*WSDL URL*". En el campo de entrada, incluiremos la URL del WSDL, que vimos en el punto anterior de la descripción del servicio web. Una vez introducida, nuestro NetBeans, crea unas fuentes generadas. Podemos indicar un paquete para incluir dentro de él las clases de las fuentes generadas a partir del WSDL. En estas fuentes, se encuentran los métodos que contiene la descripción del servicio (WSDL).

Ahora vamos a crear un nuevo JSP en el que vamos a incluir las llamadas a los métodos del servicio web. Para ello creamos un JSP normal. Este se llamará `clienteWeb`. Una vez que hemos creado el JSP `clienteWeb` nos situamos con el botón izquierdo del ratón en el JSP y pulsamos la última opción del menú que indica **Web Service Client Resources**.

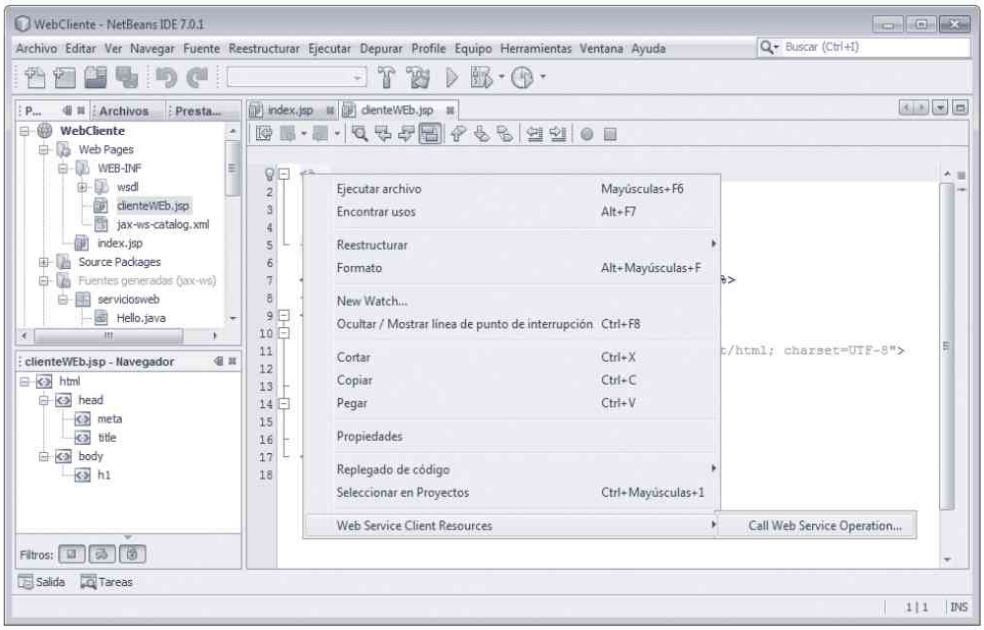


Figura 7.28. *Llamada a operaciones del servicio web*

Una vez que pulsamos en la llamada a la operación del servicio web, se nos muestra una pantalla para seleccionar el método que queremos elegir. Elegimos uno de ellos, en nuestro caso vamos a elegir `MostrarHora`.

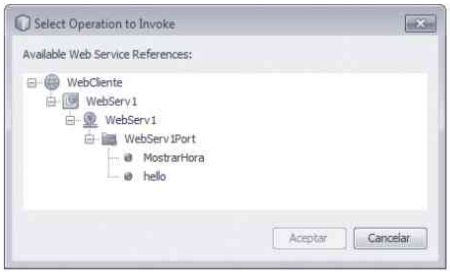


Figura 7.29. *Selección de operación del servicio web*

Una vez pulsemos el botón **Aceptar**, se genera un código en el JSP. El código que se genera de una forma genérica. Esto quiere decir que normalmente vamos a tener que retocar el código para conseguir los objetivos. El código generado en nuestro ejemplo es el siguiente. Vamos a inicializar la variable `hora` a "Madrid" para que entre en la condición y ver cómo el servicio web muestra la hora.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <%-- start web service invocation --%><hr/>
    <%
      try {
      serviciosCliente.WebServ1_Service service = new
          serviciosCliente.WebServ1_Service();

        serviciosCliente.WebServ1 port =
          service.getWebServ1Port();

        // TODO initialize WS operation arguments here
      java.lang.String hora = "Madrid";
      // TODO process result here
      java.lang.String result = port.mostrarHora(hora);
      out.println("Result = "+result);
      } catch (Exception ex) {
        // TODO handle custom exceptions here
      }
    %>
    <%-- end web service invocation --%><hr/>
  </body>
</html>
```

Para comprobar que nuestro servicio web funciona vamos a seleccionar con el ratón el fichero que hemos creado, `clienteWeb.jsp`. Una vez que hemos seleccionado con el ratón pulsamos **Ejecutar archivo** y se nos abrirá una ventana del navegador con la petición HTTP. Si todo ha ido correctamente, nos deberá salir una página HTML como la siguiente:

Figura 7.30. *Servicio web consumido*

Una vez hemos ejecutado la página JSP del cliente, hemos consumido el servicio web. El cliente a través de las clases generadas a partir del WSDL envía los XML generales para solicitar y recibir las peticiones del servicio web. Aunque en el ejemplo hemos realizado tanto la parte del servidor como del cliente con el mismo entorno de trabajo, los servicios se pueden implementar en diferentes entornos, plataformas y tecnologías. Si los WSDL siguen los estándares establecidos por la organización W3C se debería poder implementar la comunicación a través del servicio web.

7.5 SERVICIOS

Los servicios son los encargados de realizar cada una de las tareas en un servicio web. El objetivo de dividir en servicios cada una de las tareas de un servicio web es que la escalabilidad sea muy grande. Además necesitamos un bajo acoplamiento para poder multiplicar el número de combinaciones entre los servicios que ofrecen funcionalidad, los que intermedian y los que reciben el servicio. Al separar los bloques conseguimos además simplificar la lógica y, por tanto, entender su funcionamiento es mucho más intuitivo. Algunos de los servicios son lo que vamos a ver en los puntos siguientes del tema.

7.5.1 WSDL (WEB SERVICES DESCRIPTION LANGUAJE)

El WSDL es un lenguaje basado en XML para describir servicios web. También se encarga de como acceder a ellos. WSDL es un documento. Este documento puede ser de varios tipos según las operaciones que se pueden realizar y los mensajes que están involucrados. El puerto define el punto de conexión a un servicio web. Según los tipos de operación la de petición-respuesta es la más común pero existen 4 tipos definidos para WSDL:

Tabla 7.2 Tipos y definición de los WSDL según su operación

Tipo	Definición
De un solo sentido (*One-Way*)	La operación puede recibir un mensaje, pero no devuelve una respuesta.
De petición-respuesta (*Request-Response*)	La operación puede recibir una solicitud y devuelve una respuesta.
De peticion-respuesta (*Solicit-Response*)	La operación puede enviar una solicitud y puede esperar una respuesta.
Notificación (*Notification*)	La operación puede enviar un mensaje, pero no se espera una respuesta.

7.5.2 SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

Dentro del servicio de mensajería, está el protocolo simple de acceso a objetos. Un protocolo basado en XML para permitir el intercambio de información a través de un protocolo de comunicaciones como HTTP. Algunas aplicaciones, hoy en día, se comunican mediante llamadas a procedimientos remotos RPC, como DCOM, CORBA. Lamentablemente el protocolo HTTP no fue diseñado para esto y ofrece algunos problemas a la hora de implementar estas tecnologías. Existen problemas de compatibilidad y los elementos de seguridad como los *firewall* y servidores *proxy* normalmente bloquean este tráfico. Por esta razón SOAP es una de las mejores maneras de comunicarse en los servicios web. Es compatible con todos los navegadores de Internet y servidores. Y por ello SOAP es uno de los estándares que apoya la organización W3C en la implementación de servicios web.

7.5.3 XML-RPC (XML REMOTE PROCEDURE CALLING)

XML-RPC, es un protocolo de llamada a un procedimiento remoto que utiliza XML para la codificación de los datos y HTTP como protocolo de transmisión de mensajes. Este protocolo define algunos tipos de datos y comandos. Además define una descripción completa. Este protocolo requiere bastante soporte de software para su uso.

Este protocolo fue creado por Microsoft en 1998. Al considerar que era muy simple Microsoft decidió añadirle funcionalidad. Al introducir ciertas características en las diferentes etapas de desarrollo, el estándar dejo de ser sencillo y se convirtió en lo que actualmente conocemos como SOAP, el servicio que hemos visto brevemente en el punto anterior y con el que hemos trabajo durante el capítulo. Una diferencia entre SOAP y XML-RPC es que en el primero los parámetros tienen nombre y no importa su orden mientras que en el caso de XML-RPC sí.

ACTIVIDADES 7.1

- Instale Netbeans e implemente los ejemplos que hemos desarrollado en el libro.
- Cree un servicio web que recoja los gastos de la compra en el supermercado en el cliente y que los introduzca el servicio web en un Excel y lo guarde en la carpeta "C://" del sistema operativo.
- Realice un esquema de cómo se comunica todo el conglomerado del los servicios web. Realice otro paralelo comparando en los casos que se puedan las funcionalidades de un servicio web con las de un programa habitual.
- Genere un documento WSDL a mano que permita introducir dos números en el cliente y reciba en el servidor. Compruebe si le permite generar el servicio web en el lado del cliente, el realizado en el punto 7.4.1.
- Genere a mano el fichero XSD asociado al WSDL que se encuentra en el elemento `types`.

RESUMEN DEL CAPÍTULO

Un servicio web es un conjunto de protocolos y estándares que permiten comunicar dos sistemas a través de una red.

En los servicios web más habituales aparecen mensajes SOAP, ficheros WSDL, el propio servicio web y el cliente que consume el servicio.

Dentro de los mecanismos y protocolos implicados tenemos, el servicio de transporte, el de mensajería, el de descripción del servicio y el de descubrimiento.

El servicio de transporte es el encargado del envió de mensajes a través de la Red.

El servicio de mensajería se encarga de la codificación de los mensajes.

El servicio de descripción se encarga de detallar los acuerdos a los que llegan el cliente y el propio servicio web.

El servicio de descubrimiento se encarga de centralizar en un registro común los servicios web.

Para generar servicios web es recomendable utilizar entornos de trabajo que faciliten la implementación de los mismos.

A la hora de crear un servicio web, primero se crea un proyecto web, luego un servicio web, más tarde se genera a través del servicio el WSDL. Con el WSDL se generan las clases para incluir en las páginas del cliente los métodos que se pueden invocar.

Cuando un servicio web se implementa, se genera el WSDL y las clases y métodos necesarios para realizar las llamadas desde el cliente y tener un resultado satisfactorio. Se dice que el servicio web ha sido consumido.

La interfaz de un servicio web es lo que ve el cliente para realizar las actividades de las que le provee el servicio web.

EJERCICIOS PROPUESTOS

1. Un jubilado cuando echa la lotería guarda su boleto en algún bolsillo. Casi todas las semanas no sabe dónde ha guardado el boleto. El jubilado sabe que el boleto no está perdido, pero a la hora de comprobar la lotería no tiene los números para saber si le ha tocado. En caso de que le toque buscará el boleto, pero si no le ha tocado no importa demasiado dónde lo haya dejado. Hablando con sus nietos le han dicho que puede guardar los números en Internet. Uno de sus nietos le ha dicho que le va ayudar a que los guarde en la nube. El nieto ya sabe lo que quiere, pero no sabe implementarlo. Necesita que le implemente un servicio web para poder dar servicio a su abuelo con la lotería. El objetivo es crear un servicio web que tenga un método que permita escribir los números de la lotería y guardarlos en una fila del fichero. Cada vez que se introduzca una fila de números de lotería en el fichero *txt* se insertará también la fecha actual del sistema. Existirán otros dos métodos que permitirán recuperar los números de la lotería. Uno de ellos mostrará todos los números. El otro solo mostrara el número que corresponda a una fecha igual o inferior a 6 días anteriores.

Implemente el servicio web, genere el WSDL e implemente también los JPS en cliente para que el abuelo y su nieto puedan introducir y visualizar los números de lotería.

TEST DE CONOCIMIENTOS

1 El WSDL es un:
- a) *Service Discovery*.
- b) *Service Publication*.
- c) *Service Description*.
- d) *Service Pack*.

2 El servicio de transporte y de mensajería de un servicio web:
- a) Son lo mismo.
- b) Se encargan del envío y la codificación de mensajes.
- c) Utilizan XML para todo.
- d) Las 3 respuestas anteriores son correctas.

3 SOAP es:
- a) *Sample Object Access Protocol*.
- b) *Simple Object Access Protocol*.
- c) *Simple Object Asics Protocol*.
- d) *Simple Object Access Transfer*.

4 El servicio XML-RPC:
- a) Es mucho más escalable que WSDL y necesitamos conocer de antemano las funciones y métodos que proporciona el servidor.
- b) Es mucho menos escalable que WSDL y necesitamos conocer de antemano las funciones y métodos que proporciona el servidor.
- c) El protocolo está menos limitado en comparación con WSDL.
- d) No es un protocolo.

5 WSDL es un documento que:
- a) Publica la interfaz pública de un servicio web.
- b) Está escrito en XML.
- c) Las operaciones que realiza se describen de una forma abstracta.
- d) Todas las anteriores son ciertas.

6 Un servicio de descubrimiento:
- a) Centraliza un registro común de servicios web.
- b) Comparte en la Red las cosas que se descubren sobre servicios web.
- c) UDDI es un servicio de directorios.
- d) Las respuestas anteriores son ciertas.

7 Para generar un servicio web:
- a) Se necesita un entorno de desarrollo.
- b) Es recomendable utilizar un entorno de desarrollo.
- c) Hacen falta clases específicas que nos permiten generarlo.
- d) Las respuestas *b* y *c* son correctas.

8 Para crear un servicio web:
- a) Es necesario crear un proyecto web.
- b) Es recomendable crear un proyecto web.
- c) No es recomendable crear un proyecto web.
- d) Las respuestas *a* y *b* son correctas.

9 La librería WebService sirve:
- a) Para hacer un servicio web solo hace falta esa librería.
- b) No sirve para gestionar servicios web.
- c) Es una librería que además de otras es necesaria para generar servicios web.
- d) Ninguna de las anteriores es cierta.

10 Los mensajes SOAP:
- a) Tienen una respuesta.
- b) Tienen una petición.
- c) Tienen una petición y una respuesta.
- d) Todas las anteriores son ciertas.

8 Generación dinámica de páginas web interactivas

OBJETIVOS DEL CAPÍTULO

- ✓ Identificar las diferencias de la ejecución de código en el servidor y en el cliente.
- ✓ Reconocer las ventajas de unir tecnologías que se ejecutan en el cliente y en el servidor.
- ✓ Identificar librerías y tecnologías relacionadas con la generación por parte del servidor de páginas web con guiones embebidos.
- ✓ Utilizar estas tecnologías para generar páginas web que incluyan interacción con el usuario en forma de advertencias y peticiones de confirmación.
- ✓ Utilizar estas tecnologías para generar páginas web que verifiquen formularios.
- ✓ Utilizar estas tecnologías para generar páginas web que incluyan modificación dinámica de su contenido y estructura.

Una página web, tienen como principal objetivo, transmitir información a través de un documento electrónico. El primer paso para que los datos sean ofrecidos al usuario, es que este previamente haya realizado una solicitud. En el momento en el que el usuario realiza la petición y el servidor ofrece la respuesta, la comunicación vía HTTP finaliza. La navegación entre páginas a través de hiperenlaces, establece comunicaciones que comienzan y finalizan con la petición y respuesta del servidor. Esta es la interacción que el cliente tiene con una página web estática. Como hemos ido viendo en los capítulos anteriores, este tipo de páginas, que son habitualmente utilizadas para mostrar información, han quedado relegadas a un segundo plano. Las necesidades en la Red han impulsado tecnologías que ayudan complementar las carencias del protocolo HTTP, para dar dinamismo a la navegación. Estas tecnologías se ejecutan a nivel de navegador y a nivel de servidor, y aunque ambas pueden realizar la misma funcionalidad, combinarlas es lo que convierte una página web en potente, rápida, segura, dinámica e interactiva.

Una página web dinámica interactiva es la que permite variar el aspecto y comportamiento en función de decisiones que toma el usuario a través de la interfaz gráfica de la página. Para ello se varía la estructura de la página.

Para conseguir interactividad en una página web podemos utilizar código de *script*, que se ejecuta en el navegador. Otra opción es enviar una petición al servidor, de esta forma el código se ejecuta en el servidor y una vez generada la nueva página web vuelve a ser enviada al navegador. Al tratarse de una página web dinámica, en función de los datos introducidos por el usuario, la página se verá modificada mostrando información definida en la programación. Vamos a ver más detenidamente cómo funciona la ejecución de código en el cliente y el servidor, así como cuáles son los ciclos y el orden de prioridad de cada uno. Un navegador web y un servidor se comunican según el esquema mostrado en la imagen que mostramos a continuación:

Figura 8.1. *Elementos donde se ejectua código*

Según el esquema de la figura anterior, existen dos partes diferenciadas donde se puede interpretar o ejecutar código. En el navegador se ejecuta una parte del código. Otra parte se ejecuta en el servidor. Para que en el ordenador cliente, se ejecute el código, es necesario tener una aplicación, habitualmente se utiliza un navegador web. En el servidor, necesitamos un motor que interprete el código. A continuación vamos a ver cuáles son los tipos de código que se ejecutan en el navegador y en el servidor.

- **Código que se ejecuta en el navegador**. Cuando el navegador realiza una petición al servidor, este responde. En esa respuesta el servidor envía una página con una extensión. En el caso de no utilizar un lenguaje de servidor, las páginas pueden tener la extensión `.html`. Si existe un lenguaje de servidor, las páginas tienen otras extensiones, como, `.asp`, `.jsp`, `.php`, etc., dependiendo del lenguaje de servidor que estemos utilizando. Aunque parezca extraño, el que estos lenguajes de programación del servidor cambien, no afecta al resultado del tipo de código que le llega al navegador.

  El navegador recibe principalmente 3 tipos de código. Como hemos dicho anteriormente independientes del código que se ejecuta en el servidor. Estos códigos se relacionan entre sí, y se corresponden con el patrón de arquitectura, modelo-vista-controlador (MVC).

- **El código HTML**. Se corresponde con el modelo de la arquitectura MVC. HTML es la base de las páginas web, este lenguaje describe la estructura y el contenido. La forma en que se interpreta el código por el navegador, es de principio a fin. Primero se interpreta la primera línea y así sucesivamente hasta la última. El navegador interpreta todo el código de una página, cuando este ha terminado de interpretarla hasta el final, se dice que se ha cargado la página completamente.

- **El lenguaje JavaScript.** Se corresponde con el controlador de la arquitectura MVC. Este lenguaje permite configurar la interactividad y el dinamismo de la página web. A través de los eventos de la página, las acciones programadas en este lenguaje, modificaran el aspecto, estructura o visualización de la página. La ventaja de este lenguaje es que es interpretado por el navegador. El código JavaScript viaja a través de la Red desde el servidor tal cual se ha programado. Este código habitualmente se encuentra situado en la cabecera (`<head>...JavaScript...</head>`) de la página web. Lo normal es disponer de un enlace a otro documento con extensión `.js`, de forma que es en el `js` donde se aloja el código JavaScript. El comportamiento del navegador en este caso es similar, realiza las llamadas a las funciones JavaScript programadas, como si este estuviera embebido en la página web. El orden en el que el navegador interpreta este código, viene fijado por las llamadas a cada función del código. Supongamos que en la cabecera se han establecido 3 funciones, `fun1`, `fun2` y `fun3`. Si dentro de la página web se llama a la función `fun1` a través de un evento, esta función se ejecutará. En el caso de que las otras funciones implementadas `fun2` y `fun3` no sean llamadas en algún evento de la página web, estas no serán ejecutadas por el navegador. Como los eventos son acciones que pueden suceder una vez se ha cargado la página completamente, el código JavaScript, se puede ejecutar cada vez que ocurra un evento que tenga asociada una función JavaScript.

- **El lenguaje CSS**. Se corresponde con la vista de la arquitectura MVC. Este lenguaje, por tanto, es el encargado de definir la presentación de los datos que se indican en el código HTML. Para incluir las hojas de estilos CSS en la página web, es recomendable incluir un enlace a la dirección donde se ubica el fichero CSS. El navegador interpreta las características CSS. Este lenguaje es enviado directamente desde el servidor. El navegador interpreta el código CSS y lo aplica a los elementos HTML. En el caso de que exista código CSS que no esté relacionado con ningún elemento HTML, este no afectará a la página web.

- **El lenguaje XML**. Es otro lenguaje muy común en el navegador que también se encapsula dentro de los que se ejecutan en el navegador. Este es un lenguaje de marcas extensible que además permite definir la gramática. El navegador es el encargado de interpretar el código y la forma que tiene de hacerlo es similar a HTML. Este lenguaje necesita de CSS para mostrar los datos. En el caso de no utilizar estilos CSS, el navegador muestra la estructura igual que el código original.

Existen otros lenguajes que se ejecutan en el navegador como son los applets de Java. Estos son pequeños programas realizados en Java, que se envían desde el servidor al navegador y es en este donde son ejecutados. Un lenguaje similar a JavaScript es, **VBScript**. Este lenguaje solo es compatible con Internet Explorer, por lo que se desaconseja el uso de este lenguaje, en beneficio de la estandarización. Una tecnología que se utiliza para poder implementar la visualización de vídeos u otras imágenes en movimiento es Flash. Esta tecnología aunque no puede denominarse un lenguaje de programación, se ejecuta en el lado del cliente y es necesario instalar complementos en el navegador para poder soportarlo. En HTML versión 5 pretende integrar en los estándares elementos de tipo vídeo que sean interpretados directamente por el navegador.

- **Código que se ejecuta en el servidor**. Según el servidor existen múltiples lenguajes de programación que se pueden ejecutar. Estos lenguajes permiten definir parámetros en función de perfiles de usuarios, datos a mostrar, consultas a bases de datos, tipo de dispositivo que se usa, etc. Además con estos lenguajes, podemos realizar peticiones al servidor y recibir respuestas, actualizando la página web en función de parámetros enviados. Al contrario que ocurría con los lenguajes del navegador, en el servidor existen infinidad de lenguajes de programación. Estos lenguajes a su vez interactúan con diferentes bases de datos y otras tecnologías relacionadas con el servidor. Habitualmente estos lenguajes son los que condicionan el nombre de la extensión de la página web, algunos de ellos son, CGI, Perl, ASP, PHP, JSP. A continuación vamos a ver las formas en las que se ejecutan algunos de los lenguajes de programación en el servidor:
  - **CGI**. Es uno de los sistemas más antiguos, en la programación de páginas dinámicas. Un CGI es un programa que se ejecuta en el servidor para enviar datos al navegador.
  - **Perl**. Es un lenguaje de programación interpretado. Por lo tanto, no existe una compilación del código en el servidor. El código se ejecuta e interpreta directamente en el servidor. Este lenguaje permite realizar llamadas a otros subprogramas. Perl puede ser ejecutado desde otros códigos Perl.
  - **ASP** (*Active Server Pages*). Es la tecnología que Microsoft ha creado para generar páginas web dinámicas en el servidor. Este lenguaje se ha comercializado como un anexo al *Internet Information Services* (IIS). Para poder ejecutar el código ASP en el servidor, es necesario que este sea compatible con el lenguaje. El IIS soporta código ASP. El código se mezcla entre las sentencias HTML, y para que el interprete de ASP reconozca el código, se embebe entre los símbolos `<%...código ASP...%>`. Un buen truco para recordar cómo debe quedar el código ASP mezclado con HTML, es fijarse en que los picos del comienzo y finalización de ASP coinciden con los picos de las etiquetas HTML. El programador construye la página, siguiendo estos criterios. La página antes de ser interpretada en el servidor tiene este aspecto aproximadamente:

```
<html>
  <head></head>
  <body>
    <p> <% Response.Write "hola mundo" %> </p>
  </body>
</html>
```

En el código anterior, se visualiza una estructura que devuelve la cadena de texto entre comillas. Este código solo es visualizado por el programador, el servidor antes de enviar el código al navegador, interpreta la estructura ASP y genera un documento similar al siguiente.

```
<html>
  <head></head>
  <body>
    <p> hola mundo </p>
  </body>
</html>
```

  - **PHP** (*Hipertext Preprocesor*). Es un lenguaje de programación interpretado, que se ejecuta en el lado del servidor. Este lenguaje tiene una licencia de software libre y de código abierto. Además al contrario que el anterior, este lenguaje es independiente de la plataforma. Es un lenguaje rápido, con una gran liberaría de funciones y abundante documentación. El tipo de implementación con respecto a cómo se combina con el código, es similar a ASP. El código PHP se interpreta en el servidor y se envía al navegador en forma de HTML.

- **JSP** (*Java Server Pages*). Traducido, páginas de servidor Java, es otra tecnología que se ejecuta en el servidor. Este lenguaje permite crear páginas con programación Java. Java es un lenguaje multiplataforma, por tanto, estas aplicaciones se pueden ejecutar en todos los servidores que tengan la maquina virtual instalada. El coste económico de mantenimiento de esta tecnología es sensiblemente más alto que PHP por el incremento de tecnología (maquina virtual), que necesita. A continuación mostramos un ejemplo de código JSP:

```
<%
  String titulo = "Página de prueba";
  if (request.getAttribute("titulo") != null) {
    titulo = (String) request.getAttribute ("titulo");
  }
%>
...
<title><%=titulo%></title>
....
```

El primer bloque de código define una variable. Después compara si el atributo título del objeto `request` es nulo. En el caso de que no sea nulo, asigna la variable definida anteriormente al mismo.

En el bloque inferior, el código HTML generado inicializa el contenido de la etiqueta titulo a la variable titulo, antes asignada.

El primer bloque se interpreta en el servidor, por tanto, el navegador solo recibiría, del código anterior el siguiente código:

```
...
 <title>...valor del titulo...</title>
....
```

Una vez que hemos visto las diferencias que existen entre ejecutar el código en el navegador y en el servidor, podemos estudiar cuales son las ventajas y desventajas de utilizar estas tecnologías. Una de las mayores ventajas que aporta utilizar el lenguaje de *script* (JavaScript), es la rapidez, no es necesario enviar peticiones al servidor para que se ejecute allí una validación, por ejemplo. Tenemos que tener en cuenta que la inclusión de código de *script* dentro de la página puede complicar la estructura de la página web. Por esta razón siempre es recomendable hacer referencia a uno o varios ficheros donde se almacene el código de *script*. De esta forma los buscadores por ejemplo, al leer la cabecera de las páginas no tendrán que recorrer el código de *script*. Una vez que la validación en el navegador se ha realizado, la petición en el servidor será correcta casi con toda seguridad. De esta forma evitamos sobrecargar el servidor y ralentizar la iteración del usuario con su aplicación web, puesto que la respuesta JavaScript dentro del navegador es mucho más rápida. A esta mejora de iteración se le llama, mejora de experiencia del usuario. Esto no quiere decir que las validaciones en el cliente por si solas sean suficientes. Es imprescindible que las validaciones se realicen en el servidor, para dar seguridad y fiabilidad a la aplicación. Es más, una vez superada la validación en el servidor, es conveniente realizar otra validación en la base de datos, en caso de insertar datos en ella. La organización estandarizadora W3C, apuesta por las validaciones y comprobaciones en el servidor, como la base para evitar fallos, inconsistencias y vulnerabilidades en las aplicaciones web. Por estas razones en los puntos siguientes, vamos a estudiar cuales son las tecnologías que permiten realizar validaciones y comprobaciones cuando se realizan peticiones al servidor. En el servidor, el lenguaje se ejecuta y se genera la página. Esto aporta mucha seguridad puesto que solo los resultados son enviados al navegador. La respuesta adoptada por ejemplo, por una condición. En el ejemplo anterior si el objeto `request` es igual a `null`, la sentencia interna en la condición `if` no será verdadera. Esto resulta transparente para el navegador que solo obtendrá el resultado de la variable titulo.

8.1 LIBRERÍAS Y TECNOLOGÍAS RELACIONADAS

Como hemos visto en los puntos anteriores existen multitud de lenguajes de programación en el servidor. Cada lenguaje ha ido adoptando unas tecnologías particulares. De esta forma unos lenguajes permiten con más facilidad realizar determinadas operaciones. Algunas de las comprobaciones que se realizan en el servidor son muy generales, y se repiten en las aplicaciones de cada construcción web. Por esta razón casi todos los lenguajes han desarrollados librerías para realizar las tareas más generales. Por ejemplo en el caso que queramos comprobar si un DNI es correcto, necesitaríamos una función que recoja el DNI y nos devuelve una respuesta. Esta función sería, por tanto, una de las muchas que podríamos encontrarnos en una librería. Además, algunos lenguajes de programación en el servidor han desarrollado su propio *framework* con el fin de facilitar la operativa en la creación de páginas web. Como la lista de tecnologías asociadas a cada lenguaje es bastante extensa, en cada lenguaje intentaremos sintetizar los conceptos fundamentales de cada una. Para ello es importante conocer el significado de algunos términos, como librería, interfaz de programación de aplicaciones (API) ó *framework*.

El termino librería proviene de la traducción del inglés (*library*). Si fuéramos puristas, la traducción correcta es biblioteca. Se aceptan los dos términos por la similitud en el significado según la RAE. Por lo tanto, una biblioteca o librería es un conjunto de subprogramas que sirven para programar una aplicación. Las librerías contienen código y datos que pueden ser llamados desde otro programa principal. De esta forma se evita que cada programa principal desarrolle el código de la librería. En la inclusión del código de librería, éste pasa a formar parte del programa principal.

La interfaz de programación de aplicaciones (API), es un conjunto de funciones y procedimientos que proveen de bibliotecas (librerías) para ser utilizado por otro programa (generan una capa de abstracción). La diferencia con la librería es que las funciones de una API acceden habitualmente a programas diferentes, permitiendo así generar transparencia al programador.

Un *framework* es un conjunto estándar de conceptos y tecnologías con un soporte definido. Habitualmente se emplea para ello una serie de módulos concretos. Estos módulos facilitan la organización y desarrollo del nuevo software. La mayoría de los *framework* incluyen soporte para programas, bibliotecas y uno o más lenguajes que son interpretados por los programas base del *framework* para facilitar el desarrollo, agrupar y organizar los componentes del proyecto.

8.1.1 TECNOLOGÍAS Y LIBRERÍAS RELACIONADAS CON ASP

Cada lenguaje de programación en el servidor ha desarrollado sus propias tecnologías, para maximizar las opciones y adaptarse a las necesidades del mercado web. Según el consorcio W3C, el lenguaje ASP es capaz de:

- ✓ Dinamizar, editar, cambiar, añadir, cualquier contenido de una página web.
- ✓ Responder a las consultas de los usuarios y a los datos de los formularios HTML que se han enviado.
- ✓ Acceder a datos y a bases de datos, devolviendo resultados al navegador.
- ✓ Personalizar las aplicaciones web para que sean más útiles para usuarios individuales.
- ✓ Es una tecnología veloz y sencilla. Además garantiza la seguridad, ya que el código no se ve en el navegador. Si se programa de una forma correcta minimiza el tráfico de la Red.

Uno de los métodos asociados a la dinamización que nos vamos a encontrar en casi todos los lenguajes de servidor, es introducir código embebido con las etiquetas HTML. De esta forma si se genera una condición de verdadero o falso, solo se incluirá la sentencia que haya tenido un resultado positivo. Por ejemplo si tenemos dos números e indicamos la siguiente condición (suponemos que el código va integrado en una página HTML):

```
<%
  Dim numUno
  Dim numDos
  If numUno == numDos Then
    <%<p> NumUno es igual a numDos </p>%>
  Else
    <%<p> Los números no son iguales </p>%>
  End If
%>
```

En el caso de que los números uno y dos sean iguales, el código solo devolverá la sentencia "NumUno es igual a numDos". El resto del código al generarse la página HTML desaparecerá.

```
<p> NumUno es igual a numDos </p>
```

Esta es la forma más habitual de generar páginas dinámicas con los lenguajes de *scripts*. En una página real, el cogido va integrado entre condiciones de código ASP. Así se consigue generar una página en base a unas condiciones que son analizadas en el momento de interpretar el código. En contraposición con esta forma, ASP dispone de una tecnología que implementa alguno de los elementos HTML en el servidor. Con esta tecnología los elementos HTML de servidor, en vez de ser sujetos pasivos esperando a que una condición ASP los deje aparecer, tienen unas propiedades intrínsecas que los otorga autonomía. La razón de que estos elementos puedan disponer de unas propiedades, es porque estas solo están disponibles en el lado del servidor. A continuación vamos a ver que son y cómo funcionan. A estos elementos se los llama controles.

Los controles de servidor en ASP son objetos de las páginas web que se ejecutan cuando el usuario solicita una página. Estos representan el formato en el navegador. Muchos de estos controles son similares a los elementos habituales de HTML, como botones, cajas de texto, etc. Además de encontrarnos los elementos HTML, los controles ASP nos proporcionan otros más complejos, como calendarios, o conexiones a bases de datos. El *framework* que Microsoft ha facilitado para hacer uso de estos controles es ASP.Net. Por lo tanto, estos controles permiten manejar eventos en el servidor dependiendo de la iteración del usuario. Debido a que estos eventos son manejados desde el servidor, es necesario acceder a las propiedades, métodos o eventos de los elementos en el código de servidor.

Existen varios controles de servidor en ASP.Net:

- **HTML Server Controls**. Estos controles son los mismos que los de HTML. La diferencia es que estos son a nivel de servidor.
- **ASP.Net Web Forms Controls**. También estos controles son como los de HTML en el lado del servidor. por ejemplo, *TextBox*, *Buttons*, etc.
- **ASP.Net List Controls**. Este control permite mostrar el contenido de las bases de datos.
- **ASP.Net Mobile Controls**. Estos controles tienen la misma funcionalidad que los *Web Forms* y *List Controls* de los dos puntos anteriores. La diferencia reside en que se aplican a los dispositivos cliente, como por ejemplo un dispositivo móvil.

- **ASP.Net Validation Controls**. Estos son los controles de validación. Permiten en el servidor, validar entradas a través de la iteración que el usuario tiene con la aplicación. La ventaja que presentan, es la variedad de casos para validar. También es posible crear un criterio propio de validación.

A continuación vemos un ejemplo del código correspondiente a un *Control Web Form* de tipo *TextBox* en un formulario:

```
<html>
  <body>
    <form runat="server">
    <asp:TextBox id="tb1" Text="Name" runat="server" />
    </form>
  </body>
</html>
```

Una de las formas de reducir la redundancia de código, es crear librerías para que las páginas llamen a los bloques de código. Para ello se genera un fichero de código con la extensión `.asp`. Por ejemplo, `libreria.asp` dónde va la implementación general. A este fichero se le llama librería.

Las librerías en ASP nos permiten reutilizar código. De esta forma el código común reside en un único fichero. En las páginas que deseemos utilizar el código común, solo tendremos que llamar a la función correspondiente de la librería. Una utilidad de las librerías puede ser; a la hora de generar páginas dinámicas. En una página podemos encontrar que ciertas partes se repiten. Imaginemos que tenemos una página con una cabecera, un menú, una parte central y un pie de página. Supongamos que la cabecera, el menú y el pie se repiten para todas las páginas. Deberíamos repetir el código correspondiente a estas partes en cada página. Una forma de evitar la repetición del código, es utilizar una librería que contenga cada parte, cabecera, menú y pie.

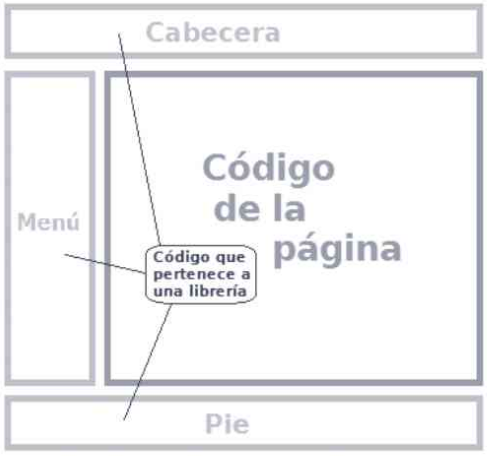


Figura 8.2. *Secciones de una página*

La representación de la imagen anterior muestra como dentro de una página se llama al contenido de la cabecera, menú y pie que residen en una librería. El código que representa esta estructura es el siguiente:

```
<html>
  <head><title>Título de ASP</title></head>
  <body>
    <!-- #include file="libreria.asp" -->
    <% call CabeceraPagina %>
    <% call MenuPagina %>

    Contenido de la Página

    <% call PiePagina %>
  </body>
</html>
```

En el código anterior, vemos como se incluye la librería con `<!-- #include file="libreria.asp" -->`. Debemos inicializar la variable `file` al valor donde se encuentra la librería. Es habitual encontraros direcciones relativas cuando las páginas están implantadas en servidores. Para ello se utiliza el punto o el doble punto. En el caso de no poner nada, se entiende que la ruta donde se encuentra el fichero es la actual. Las llamadas al código se realizan mediante la función `call`. Esta llamada devuelve el código correspondiente. Según la ruta y la jerarquía en la que se encuentran, podemos llamar a una librería de 3 formas. Aunque existen más, estas son las más habituales:

```
<!-- #include file="libreria.asp"    // Directorio actual -->
<!-- #include file="./libreria.asp"  // Directorio padre  -->
<!-- #include file="../libreria.asp" // Un nivel mas atrás-->
```

Por último, faltaría implementar el código correspondiente a la librería. En el siguiente código mostramos las funciones que contienen el código de cada una de las partes de la página, la cabecera, el menú y el pie de página.

```
<%
  sub CabeceraPagina
%>
  <p>Cabecera de la páginas</p><hr>
<%
  end sub
  sub MenuPagina
%>
  <p>Menu de la página</p>
<%
  end sub
  sub PiePagina
%>
  <p>Pie de la página</p>
<%
  end sub
%>
```

Otra de las funcionalidades más comunes en las librerías, es utilizar una función para realizar operaciones que se repiten. De esta forma introduciendo unos parámetros de entrada recuperaríamos un resultado. Cualquier tarea que se repita y que cumpla una serie de patrones en una página web, se puede incluir en una librería. En estos casos es necesario conocer el patrón de conducta para que la función de la librería pueda predecir el resultado.

8.1.2 TECNOLOGÍAS Y LIBRERÍAS RELACIONADAS CON PHP

La tecnología PHP es uno de los lenguajes de generación de páginas web dinámicas e interactivas más popular. Las razones son la independencia de la plataforma, la libertad en el tipo de licencia y la rapidez de aprendizaje junto con la multitud de documentación y tecnologías asociadas. La entidad estandarizadora W3C describe en su página oficial que PHP:

- Funciona en diferentes plataformas (Windows, Linux, Unix, etc.).
- Es compatible con casi todos los servidores que se utilizan en la actualidad (Apache, IIS, etc.).
- Se puede descargar gratis desde los recursos oficiales de PHP.
- Es fácil de aprender y se ejecuta de manera eficiente en el lado del servidor.

En PHP también es posible tener código embebido para generar de forma dinámica las páginas de una aplicación web. De esta forma, en función de unos parámetros y las decisiones en el código, se va conformando la página web. En la siguiente estructura, vemos una condición que compara dos cadenas de texto:

```
<p><?php
  $nombre = "Javi";
  if($nombre != "Javi"){
    echo "Esta página no es de Javi";
  }else{
    echo "Javi, esta es tu página";
  }
?></p>
```

En el código anterior, vemos como, si el valor de la variable `$nombre` es igual a "Javi", el código devolverá: "Javi, esta es tu página". El código de respuesta del motor PHP dejaría la siguiente sentencia:

```
<p>Javi, esta es tu página</p>
```

Otra de las posibilidades para generar código de forma dinámica, es el uso de librerías. En el punto anterior referente al código ASP, hemos visto como una librería incluía un bloque de código que se repetía en todas las páginas de la aplicación web. En PHP también podemos realizar esta funcionalidad. No obstante vamos a contextualizar el ejemplo con una función que nos devuelve un valor a partir de dos parámetros que recibe. En el siguiente código se pasan dos valores a la función `calculaHipotenusa`. El valor se guarda en la variable `$hipotenusa`. La función `calculaHipotenusa` se encuentra en otro fichero llamado `funciones.php`. Para que en la llamada a la función, el código sea reconocido, es necesario incluir la ruta del fichero que contiene las funciones mediante la sentencia `include`.

```
<?php
  include "funciones.php";

  $cateto1 = 10;
  $cateto2 = 5;
  $hipotenusa = calculaHipotenusa($cateto1, $cateto2);

  print"<p>La hipotenusa del triángulo es $hipotenusa </p>";
?>
```

La librería en este caso, alberga la implementación de la función `calculaHipotenusa`. Dentro de la implementación, se calcula el otro lado del triángulo. Estas operaciones solo se programan una vez. De esta forma se reutiliza código, aumentando la claridad del código y minimizando la probabilidad de errores de cálculo.

```
<?php
  function calculaHipotenusa($arg1, $arg2){
    $hipotenusa = sqrt($arg1*$arg1+$arg2*$arg2);
    return $hipotenusa;
  }
?>
```

El código anterior, está incluido en `funciones.php`. La forma jerárquica de llamar a los directorios a través del punto, es igual que en la tecnología ASP. En el ejemplo anterior, hemos supuesto que el fichero "`funciones.php`", está en el mismo directorio que la página que llama a la función de la librería.

8.1.3 TECNOLOGÍAS Y LIBRERÍAS RELACIONADAS CON JSP

Cuando hablamos de JSP, nos estamos refiriendo a páginas Java que se ejecutan en el servidor. Esta tecnología nos permite ejecutar código Java en el servidor a través de *scripts*. De esta forma generamos páginas web dinámicas. La respuesta de la ejecución del código en el servidor, da como resultado documentos HTML o XML. No obstante, en Java existen dos tipos de fichero principales que se ejecutan en el servidor. Ambos tipos de fichero pueden realizar las mismas operaciones. Uno de ellos es JSP, el cual facilita la presentación de los documentos. El otro tipo de fichero es el *servlet*, este provee de una mayor flexibilidad en la programación de la lógica de negocio. La ventaja de las tecnologías Java relacionadas con el servidor, es que estas se pueden ejecutar en múltiples plataformas. Habitualmente para servir documentos web, es necesario disponer de un servidor de aplicaciones. El nombre genérico se refiere a un servidor en una red que ejecuta diferentes aplicaciones. El servidor de aplicaciones gestiona las funciones de la lógica de negocio y el acceso a datos. Aunque Microsoft llama al IIS (*Internet Information Server*), servidor de aplicaciones, el nombre es popularmente conocido para referirse al servidor de aplicaciones de Java EE. A continuación vamos a ver cuáles son las diferencias entre JSP y *servlets*:

- JSP es un acrónimo de *Java Server Pages*. El objetivo de los JSP es estructurar la interfaz de usuario, de una forma clara y ordenada.
- Los *Servlet* son programas que se ejecutan en el lado del servidor para generar páginas web. Estos actúan normalmente a partir de parámetros enviados en una petición desde el navegador. Por su forma facilitan en su implementación la programación del modelo de negocio.

A continuación mostramos un diagrama de la comunicación que existe si utilizamos los ficheros tipo JSP como la vista de una aplicación web. De esta forma el servidor devuelve al navegador las respuestas. Estas son páginas `.jsp`. En el lado del cliente, el navegador envía las peticiones al servidor. Estas peticiones son dirigidas a los *servlets*, que en este caso hacen de controlador. El acceso a datos se realiza a través de la interfaz de usuario (JSP).

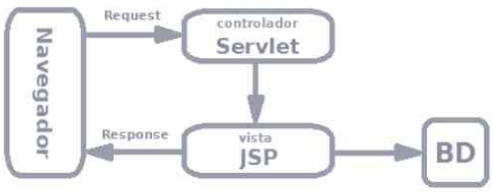


Figura 8.3. *Modelo vista controlador JSP y Servlet*

A continuación, vamos a ver un ejemplo de una JSP dinámica que suma uno cada vez que se carga la página en el navegador. La estructura de las JSP al igual que en ASP y en PHP, llevan el código embebido con el HTML.

```
<%@ page language='java' contentType="text/html" %>
<%! int count=0; %>
<html>
  <head><title>Ejemplo JSP</title></head>
  <body>
    Ha entrado
    <%= count++ %>
    <% if (count == 1) { %>
      vez
    <% } else { %>
      veces
    <% } %>
  </body>
</html>
```

El *servlet* en cambio tiene una estructura de programación similar a la de una clase, en programación orientada a objetos. Esto no quiere decir que un *servlet* no pueda generar un documento HTML, pero si es necesario utilizar mecanismo para devolver la estructura. A continuación mostramos que un *servlet* devuelve una página HTML que muestra "hola mundo".

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaMundo extends HttpServlet {

  public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException,
    ServletException{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Título Hola Mundo!</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Hola Mundo!</h1>");
      out.println("</body>");
      out.println("</html>");
  }
}
```

En el código anterior mostramos como un *servlet* devuelve un documento HTML al navegador. Como vemos por la forma de un *servlet* es más fácil implementar la lógica que mostrar un documento HTML. La seguridad que ofrece un *servlet* a la hora de abstraer y evitar que los datos de la lógica puedan llegar al navegador es mayor. En el caso de que se produzca un error o una petición al *servlet* no deseada, este solo enviará al navegador las trazas correspondientes a las salidas por el `response` (código incluido en `out.println("...")`). En el caso de los JSP, puede ocurrir que un error devuelva código al navegador, enviando información cuyo ámbito es solo del servidor.

8.2 GENERACIÓN DINÁMICA DE PÁGINAS INTERACTIVAS

En el punto anterior hemos visto como las diferentes tecnologías existentes, nos permiten generar páginas web dinámicas. Cuando una página web se convierte en interactiva, no solo tiene que dinamizar el resultado, tiene que responder a las peticiones del usuario. Por lo tanto, debe existir una forma en la que el usuario pueda comunicar al servidor su respuesta. El servidor, debe ser capaz de ofrecer las páginas adecuadas. La manera de conseguir interactividad es comunicar al servidor con las acciones tomadas en el navegador. Las tecnologías de servidor, adecuan sus herramientas para mejorar el dinamismo y la facilidad de la programación. Cada lenguaje tiene sus peculiaridades, aunque el objetivo es el mismo, dar dinamismo e interacción a las páginas web.

8.2.1 PÁGINAS INTERACTIVAS EN ASP

Al estar embebido el código ASP junto al HTML, cuando se programa una interacción con el usuario, la página puede tener varios comportamientos. En las interacciones debemos tener en cuenta a que página se dirige en el servidor. Vamos a ver algunos de los casos más comunes de direccionamiento.

Ejecutar código de *script* de navegador para solicitar otra página al servidor con una advertencia es una de las tareas más comunes para establecer una comunicación con el usuario. En este caso el código a ejecutar son *scripts* de navegador (JavaScript). La respuesta al evento capturado es la redirección a otra página. En la petición de la nueva página, podemos enviar variables a través de la URL. De esta forma podemos controlar desde que evento o página viene el usuario.

```
<html>
  <head><title>Título origen ASP</title>
    <script languaje="JavaScript">
      function envio(){
        if(confirm('¿Pulsa una opcion?')){
          location.href='pagina.asp?id=1';
        }else{
          location.href='pagina.asp?id=2';
        }
      }
    </script>
  </head>
  <body>
    <p onclick="envio()">Pagina nueva</p>
  </body>
</html>
```

En el código anterior podemos ver cómo existe un párrafo que lleva asociado el evento `onclick` (dentro del `body`). En la parte superior, una función controla la redirección. La función `confirm`, despliega un cuadro de diálogo con las opciones por defecto: Aceptar y Cancelar.

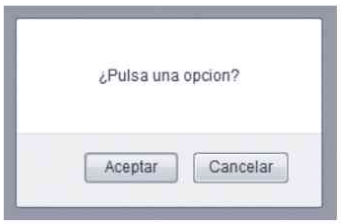


Figura 8.4. *Cuadro de diálogo que se muestra al hacer clic*

En el caso de hacer clic con el ratón sobre el botón **Aceptar** del cuadro de diálogo, la aplicación solicita `pagina.asp`. El valor de la variable `id` será 1. En el caso de que hagamos clic sobre el botón **Cancelar** del cuadro de diálogo, el valor enviado a través de la URL es "02". A continuación vamos a ver cómo podemos controlar en la página de destino qué botón pulsó el usuario.

```
<html>
  <head><title>Título destino ASP</title></head>
  <body>
    <%
      valor = Request.QueryString("id")

      if valor=1 then
      response.write("Has pulsado si, valor: " & valor)
      else
      response.write("Has pulsado no, valor: " & valor)
      end if
    %>
  </body>
</html>
```

El primer lugar recogemos la variable que hemos pasado en la URL. Esto se realiza con la función `QueryString` del objeto `Request`. En el siguiente paso comparamos el valor recogido con las opciones "1" para el clic *Aceptar* y "2" para el clic *Cancelar* de la página anterior. Si el valor es 1 se muestra en la página el primer diálogo ("Has pulsado si...") en el caso de que sea 2 se mostrará el diálogo segundo. Es importante tener en cuenta que cuando no se pasen las variables en la URL con valores 1 y 2, el diálogo que se muestra es el perteneciente al `else` de la estructura. Si queremos controlar más opciones necesitamos incluir condiciones en la estructura del `if`.

8.2.2 PÁGINAS INTERACTIVAS EN PHP

En el lenguaje PHP existe una estructura de código embebido similar a la de ASP. Por lo tanto, vamos a implementar una redirección a una nueva página. En este ejemplo solicitaremos páginas diferentes según las diferentes tomas de decisiones del usuario. Como en el caso anterior, mostraremos un cuadro de diálogo en el que aparecerán dos botones para pulsar, *Aceptar* y *Cancelar*. Dependiendo de la elección en este caso vamos a solicitar páginas diferentes. A continuación mostramos el código necesario para que un evento recoja el clic del usuario y tome la decisión oportuna, redireccionando a dos páginas PHP.

```
<html>
  <head><title>Título origen PHP</title>
    <script languaje="JavaScript">
      function envio(){
        if(confirm('¿Pulsa una opcion?')){
          location.href='destinoUNO.php?id=1';
        }else{
          location.href='destinoDOS.php?id=2';
        }
      }
    </script>
  </head>
  <body>

    <p onclick="envio()">Destino Pagina nueva</p>
  </body>
</html>
```

En el código anterior, cuando el usuario hace clic en el párrafo que se encuentra entre las etiquetas del `body` se activa el evento. En la parte superior, es llamada la función `envío`. En el primer caso solicita la página `destinoUno.php`, si el usuario pulsa aceptar en el cuadro de diálogo. En el segundo caso `destinoDos.php`, si el usuario pulsa cancelar. A continuación, vamos a ver cómo se evalúa en la página de destino que variable ha sido pasada a través de la URL. De esta forma, podemos controlar, al igual que ocurría en ASP, que página origen solicito la página de destino.

```
<html>
  <head><title>Título destino uno PHP</title></head>
  <body>
    <%
      $valor = $_GET["id"];
      if ($valor == 1){
      echo "Estas en el DESTINO 1, valor: ".$valor);
      }else{
      echo "El parámetro id no es 1, es: ".$valor);
      }
    %>
  </body>
</html>
```

El código anterior se corresponde con la página, `destinoUno.php`. Según la estructura del lenguaje PHP, en primer lugar se recoge la variable `id` (que previamente se ha pasado a través de la URL). Para ello accedemos al *array* `$_GET`. Este vector nominal contiene las variables que se han pasado a través de la URL de la página actual. En el caso de que éste sea 1, la página escribe, "Estas en el DESTINO...". En el ejemplo visto no cabe la posibilidad de que la variable que recibe (`id`) no sea uno. Si existiera otra página diferente que llamara a `destinoUno.php`, y pasara otro valor de `id` a través de la URL, el resultado de escritura en la página sería el indicado en el `else` de la estructura ("El parámetro id no es 1, es:...").

El ejemplo de `destinoDos.php` puede ser igual, la única diferencia es que el valor del `if` es 2 en vez de 1 y el nombre del fichero, `destinoDos.php`.

8.2.3 PÁGINAS INTERACTIVAS EN JSP

Los JSP llevan un tipo de código Java embebido con el código HTML, como ocurre con las tecnologías PHP y ASP. Con un JSP podemos reproducir todos los comportamientos que hemos visto en los ejemplos anteriores. *Java Enterprise Edition* o Java EE de forma reducida son los nombres genéricos que se utilizan para referirse a las tecnologías web de Java. En el apartado 8.1.3 veíamos como existían dos tipos de ficheros que podían emular el mismo comportamiento, los JSP y los *servlets*. Los JSP eran utilizados para mostrar las interfaces graficas al usuario. Los *servlets* en cambio facilitaban la implementación de la lógica y la toma de decisiones en la programación. Además aumentan notablemente la seguridad y aseguran que el fichero solo se vaya a ejecutar en el servidor. La razón es que la tipología de de un *servlet* no lleva implícito el envió de datos al navegador. Los *servlets* son clases que atienden peticiones HTTP. Esto se hace a través del método `getWriter()`. Así se crea una abstracción entre el servidor y el navegador. Debido a esta utilización, vamos a ver cómo se implementa una página interactiva a través de estos dos tipos de ficheros. En primer lugar representamos la página origen que incluye la solicitud del usuario. Al igual que en los ejemplos anteriores, utilizamos el evento `onclick` y una función para redireccionar a otra página cuando se capture el evento.

```
<html>
  <head><title>JSP Page</title>
    <script languaje="JavaScript">
      function envio(){
        if(confirm('¿Pulsa una opcion?')){
          location.href='/capitulo/intermediario?id=1';
        }else{
          location.href='/capitulo/intermediario?id=2';
        }
      }
    </script>
  </head>
  <body>
    <p onclick="envio()">Destino pagina nueva</p>
  </body>
</html>
```

En el código anterior, al hacer clic en el texto **Destino página nueva**, se lanza un evento que solicita el *servlet* `intermediario`. Dependiendo del parámetro que le pasemos a través de la URL se envía el `id` con valor 1 ó 2. La decisión de enviar un valor u otro es del usuario. Según pulse *Aceptar* o *Cancelar* en el cuadro de diálogo (`función confirm('...')`) permite la entrada por una u otra opción en la estructura condicional.

A continuación vamos a ver cómo se comporta el *servlet* `intermediario`.

```
package es.serv.paquete;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet(name = "intermediario", urlPatterns =
  {"/intermediario"})
public class intermediario extends HttpServlet {

  protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)throws ServletException,
    IOException {

    String idServlet = request.getParameter("id");
    response.setContentType("text/html;charset=UTF-8");

    if("1".equals(idServlet)){
      response.sendRedirect("destinoUNO.jsp");
    }else{
      response.sendRedirect("destinoDOS.jsp");
    }
  }

  @Override
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response)throws ServletException,
    IOException {
    processRequest(request, response);
  }
}
```

La primera línea del *servlet* indica en que paquete se encuentra este. Las líneas posteriores importan las clases Java necesarias para poder implementarlo. La clase `HttpServlet` se encarga de gestionar las peticiones HTTP. Las clases `HttpServletRequest` y `HttpServletResponse` se encargan de la petición realizada por el navegador (cliente) y la contestación de una petición del cliente respectivamente.

Los *servlets* deben ser mapeados a una URL, en las versiones anteriores, era necesario crear un `web.xml`. A partir de la versión 3.0, según se define en el estándar JSR 135, el mapeo se puede hacer mediante anotaciones. La más importante es `@webServlet`. En nuestro caso hacemos uso de esta anotación y la mapeamos en la ruta "/intermediario" con el nombre `intermediario`.

A continuación se definen las clases necesarias. Para recuperar el valor enviado de la página anterior, hacemos uso del objeto `request` llamando al método `getParameter()`. Una vez hemos recuperado el valor lo comparamos en la estructura condicional (`if`). En el caso de que el valor recuperado sea 1, redireccionamos a través del método `sendRedirect()` al `destinoUno.jsp`. En caso de no ser 1 redireccionamos al `destinoDos.jsp`.

```
<html>
  <head><title>JSP Page</title></head>
  <body>
    <h1>Pulso ACEPTAR. Llega al destino UNO</h1>
  </body>
</html>
```

En el código anterior presentamos la implementación de `destinoUno.jsp`. La petición de esta página se realiza cuando la condición del *servlet* es igual a 1. El `destinoDos.jsp` es igual que el uno pero con otro mensaje que indique que se pulsó el botón *Cancelar*.

A continuación vamos a ver un esquema de cómo se comportan las interacciones entre el usuario y el servidor a través de las interfaces gráficas y la lógica que se encuentra incluida en el *servlet*.

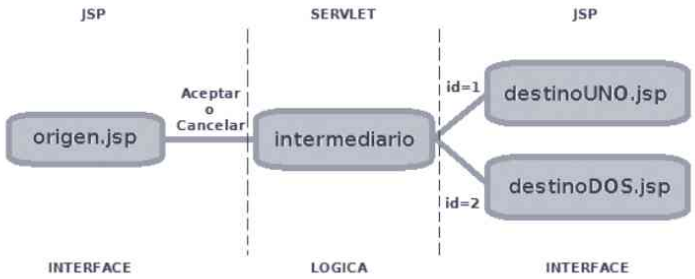


Figura 8.5. *Modelo Vista controlador de la implementación del ejemplo*

Las combinaciones que podemos encontrar en las interacciones del usuario con las páginas web, son muy extensas dependiendo del caso. No obstante con el ejemplo que hemos visto en cada unas de las tecnologías, tenemos herramientas suficientes para implementar todos los casos. Es importante controlar el envío y recepción de variables entre el código que se ejecuta en el navegador y el servidor. Otro punto para conseguir la interacción es conocer la forma de redirección automática entre páginas, incluyendo en ella, el envío de variables y su posterior análisis. En algunas implementaciones en las que se separa la lógica del resto de capas, se dedica un fichero común al encaminamiento de páginas. De esta forma todas las paginas pasan por el fichero común, este es el encargado de redireccionar a cada destino, en base a los parámetros recibidos.

8.3 OBTENCIÓN REMOTA DE INFORMACIÓN

Una de las necesidades más usuales a la hora de implementar una página web interactiva, es la recolección de cierta información que proporciona el usuario. Hasta ahora en el capítulo hemos visto como los datos que se enviaban al servidor, habitualmente de una forma trasparente para el usuario y nos servían para decidir el camino que iba a seguir el flujo de páginas web. Las variables que recuperábamos en las peticiones a las nuevas páginas se evaluaban con el fin de conocer la respuesta desde el servidor. Otro de los objetivos principales de la mayoría de las aplicaciones web, es el almacén de variables, enviadas por el usuario, con el fin de poder ofrecerle un servicio. Un ejemplo puede ser el registro en un foro, así en el futuro la aplicación conocerá los datos de un usuario. En base a un usuario y una contraseña (almacenados), es posible autenticar a un usuario. Además existen casos en los que se necesita conocer un valor que introdujo previamente el usuario, para evaluarlo y ofrecer una respuesta. En este caso, un ejemplo puede ser, una aplicación que calcule la hipoteca mensual en función del coste total de la vivienda, los años, etc.

Para recuperar campos en el servidor, se utilizan los formularios de HTML. Al enviar el formulario desde el cliente al servidor, se indica cuál es la página web que va a recuperar estos datos. Esta página recupera los valores apoyándose en las diferentes tecnologías que hemos visto en los puntos anteriores. ASP, PHP, JSP. No obstante cuando los valores

de las variables llegan al servidor y son recuperados, el servidor, no puede garantizar que el tipo de dato coincida con el que se esperaba obtener. Supongamos que en un formulario se espera un número de 9 cifras que corresponde con un teléfono. El usuario ha introducido un número de 10 cifras. Podemos predecir de antemano que un número de 10 cifras no corresponde con ningún teléfono valido en el territorio nacional. Además en el caso de que la base de datos para ese campo solo permita la inclusión de un número de 9 cifras, se producirá un error.

Por esta razón es necesario que los valores que llegan desde el navegador, sean previamente comprobados. En otros capítulos veíamos como existía una validación en el lado del cliente. Esta validación se realizaba a través de *scripts* de navegador (JavaScript o VBScript). A nivel de integridad de datos, no es posible incluir los *script*, como un método fiable de validación. En el lado del cliente perdemos la visibilidad de ejecución del código. Es en el cliente donde se ejecuta el código enviado, concretamente en la aplicación de su navegador. Por lo tanto, este tipo de comprobaciones en el navegador están más orientadas a evitar excesivas peticiones al servidor. También para que el cliente reciba unas orientaciones en el navegador, de esta forma, cuando los datos llegan al servidor tienen menos posibilidades de contener datos inconsistentes. Con esto se consigue disminuir el número de veces que el servidor atiende peticiones para cada cliente.

Figura 8.6. *Esquema de validación en el servidor*

A continuación, vamos a ver cómo comprueban las diferentes tecnologías de servidor los datos recibidos por el usuario. Para implementar las comprobaciones, haremos uso de mensajes, que se mostraran en el navegador cuando un dato no sea correcto. Es importante tener en cuenta que una excesiva severidad a la hora de validar los datos, pueden provocar un rechazo por parte del usuario en el uso de la aplicación, aumentando la curva de aprendizaje y empeorando la experiencia del usuario.

8.3.1 VALIDAR DATOS CON ASP

En ASP podemos validar los campos a través del código embebido con el HTML. En este caso, deberemos incluir una estructura condicional. Si la estructura supera la condición establecida para el campo, se aceptará el parámetro. En caso de no superar la condición, se devolverá la página de formulario al usuario con un mensaje, indicando que campo contiene el error. En las tecnologías relacionadas con ASP, existen herramientas específicas para desarrollar la validación. Como veíamos en los puntos anteriores, Microsoft incorpora ASP.Net como una tecnología más avanzada que ASP para mejorar la combinación de código HTML con código de servidor. En el punto 8.1.1 veíamos como ASP.Net implementa una serie de herramientas, como son los controles, para mejorar el código en el lado del servidor. Un grupo de controles, según veíamos en dicho apartado, son los de validación. Los controles de validación, se utilizan para validar los datos de entrada del usuario. En el caso de que el control de entrada no pase la validación, se muestra un mensaje de error al usuario. La diferencia con el caso que explicábamos al principio del párrafo es que ahora gestionamos la validación a través de una estructura que tiene un nombre y una serie de parámetros.

La sintaxis básica para crear un control de validación de servidor es la siguiente:

```
<asp:control_name id="some_id" runat="server" />
```

En el código anterior el tipo de control lo especificamos a través del campo `control_name`. Además dispondrá de un identificador `id` y en el atributo `runat` que indica que este control se ejecuta en el servidor. Hay más campos para cada control de validación, aunque los anteriores son los que forman la estructura principal. Existen los siguientes tipos de controles de validación. Las funcionalidades de cada uno la especificamos en la segunda columna de la tabla.

Tabla 8.1 Controles de servidor en el lenguaje ASP

Nombre del Control	Descripción del Control
`CompareValidator`	Compara el valor de un control de entrada al valor de otro control de entrada o de un valor fijo.
`CustomValidator`	Permite escribir un método para controlar la validación de un valor introducido.
`RangeValidator`	Comprueba que el usuario introduce un valor que esté entre dos valores.
`RegularExpressionValidator`	Asegura que el valor de un control de entrada coincide con un patrón determinado.
`RequiredFieldValidator`	Hace que el control de un campo de entrada sea obligatorio.
`ValidationSummary`	Muestra un informe de todos los errores de validación que se produjeron en una página web.

A continuación, vamos a ver un ejemplo de cómo implementar un control de validación en el servidor de tipo `RangeValidator`. En este ejemplo vemos los diferentes campos que tiene este tipo de validación. Estos son: `ControlToValidate`; indica el campo del formulario al que se refiere, `MinimumValue`; valor mínimo del rango, `MaximumValue`; valor máximo del rango, `Type`; tipo de dato, `EnableClientScript`; indica si se comprueba también en los *scripts* de cliente (JavaScript), `Text`; mensaje de error que aparecerá en el navegador, `runat;` donde se ejecuta.

```
<script  runat="server">
  Sub submit(sender As Object, e As EventArgs)
    If Page.IsValid Then
      lbl1.Text="La página es válida."
    Else
      lbl1.Text="La página no es válida!!"
    End If
  End Sub
</script>

<html>
  <body>
    <form runat="server">
```

```
        Introduzca un teléfono móvil valido:
        <asp:TextBox id="tbox1" runat="server" />
        <br /><br />
        <asp:Button Text="Submit" OnClick="submit"
          runat="server" />
        <br /><br />
        <asp:Label id="lbl1" runat="server" />
        <br />
        <asp:RangeValidator ControlToValidate="tbox1"
          MinimumValue="600000000" MaximumValue="699999999"
          Type="Integer" EnableClientScript="false"
          Text="El teléfono introducido no es válido!"
          runat="server" />
      </form>
    </body>
</html>
```

En el ejemplo, observamos lo sencillo que es comprobar a partir de dos parámetros, que solo se considere valido un rango de números. Aprovechamos esta característica para restringir la entrada de datos del usuario a los teléfonos móviles validos en el territorio nacional. En el caso de que el usuario introduzca un teléfono móvil que no sea válido, se mostrara el mensaje indicado en el campo `Text`.

8.3.2 VALIDAR DATOS CON PHP

El lenguaje PHP se sirve de filtros para validar y filtrar datos provenientes de fuentes inseguras, como es el envió de datos desde el navegador del usuario. Los tipos de datos que validan los filtros son, campos de entrada de formularios, *cookies*, datos de servicios web, variables y resultados de consultas (*querys*). A continuación visualizamos una tabla con los principales tipos de filtro y su descripción.

Tabla 8.2 Filtros en PHP

Nombre del Filtro	Descripción del Filtro
`filter_has_var()`	Comprueba si existe una variable de un tipo concreto.
`filter_var()`	Filtra una variable con un filtro específico.
`filter_var_array()`	Filtra varias variables con el mismo o diferente filtro.
`filter_input()`	Obtiene un campo de entrada y un filtro para este.
`filter_input_array()`	Obtiene varios campos de entrada y filtros con el mismo o diferente filtro.

En este punto nos vamos a centrar en la validación de los campos de entrada de los formularios. Las funciones de la tabla anterior, reciben diferentes parámetros, vamos a ver cómo validar un campo de entrada con la función, `filter_input()`.

```
<html>
  <head><Title>Validar e-mail</Title><head>
  <body>
    <form method="GET" action="validaEmail.php">
      <p>Introduce un e-mail a validar <input type="text"
        name="email" size="30"></p>
      <?php
        if(!filter_has_var(INPUT_GET, "email")){
          echo("Aun no has introduce un e-mail");
        }else{
          if (!filter_input(INPUT_GET, "email",
            FILTER_VALIDATE_EMAIL)){
            echo "El e-mail no es válido";
          }else{
            echo "El e-mail es válido";
          }
        }
      ?>
      <p><input type="submit" value="Validar"
        name="enviar"></p>
    </form>
  </body>
<html>
```

En el código anterior vemos como la función `filter_input(tipo_envio, nom_var, filtro)`, recibe 3 parámetros. Estos parámetros son: el tipo de envío a través del formulario (*GET* o *POST*), el nombre del campo de entrada del formulario y el filtro que se aplica al campo de entrada. Según sea la respuesta de la función, devolverá, "El e-mail es válido", en caso de que sea correcto y "El e-mail no es válido", si este no se ha introducido respetando la estructura de un correo electrónico. Observamos que primero se comprueba si existe el campo de entrada del formulario. Para ello se hace uso de la función `filter_has_var(tipo_envio, nom_var)`. Es conveniente hacer esta comprobación puesto que la primera vez que accedemos a la página no se ha enviado a través del método *GET* o *POST* la variable (campo de entrada del formulario) que se desea analizar.

8.3.3 VALIDAR DATOS CON JSP

Las tecnología relacionadas con Java EE (en versiones anteriores, llamado J2EE), tienen numerosas formas de validar. Existen clases que nos permiten comprobar si una variable cumple con una serie de condiciones, el uso de *beans*, que son componentes de software reutilizables. Además existen otras tecnologías basadas en Java EE, como son JSF o Struts, que implementan el modelo vista controlador y facilitan las labores de validación. Debido al aumento de complejidad de los modelos basados en Java EE, en este punto vamos a utilizar las funcionalidades de los JSP para implementar la capa de interfaz de usuario y los *servlets* para implementar la lógica. Según la imagen siguiente el usuario enviara y recibirá los datos a través de un formulario generado en un JSP. Las peticiones en cambio se enviaran a un *servlet* que será el encargado de validarlos.

Figura 8.7. *Esquema de validación en un servlet*

Siguiendo estos criterios la interfaz de usuario de una página JSP queda de la siguiente manera:

```
<html>
  <head><title>JSP Page</title></head>
  <body>
    <%
      String nombre = "";

      if(request.getParameter("nombre")!=null){
        nombre = request.getParameter("nombre");
      }
      String error1 = request.getParameter("error1");
    %>
    <form method="GET" action="/capitulo/validar">
      <p>Introduce nombre<input type="text" name="nombre"
        value="<%=nombre%>" size="30"></p>
      <%
        if("1".equals(error1)){
          out.println("Es necesario rellenar el nombre");
        }else{
          if("2".equals(error1)){
            out.println("Nombre validado");
          }
        }
      %>
      <p><input type="submit" value="Validar"
        name="enviar"></p>
    </form>
  </body>
</html>
```

En el código anterior vemos como la variable `nombre`, es susceptible de ser o no válida dependiendo de si el usuario ha enviado algún valor en el formulario. La página hace a la vez de origen de los datos para su envío como de presentación y respuesta en los casos de que se haya o no introducido un nombre en el formulario. El campo de entrada del formulario `nombre`, se inicializa con la variable enviada a través de la URL. A través de la variable `error1` (es enviada a través de la URL por el *servlet* a la siguiente página) se generan dos posibles respuestas. Si el usuario no ha introducido un nombre en el campo de entrada y pulsa **Enviar** visualizará el siguiente mensaje: "Es necesario rellenar el nombre". Si el usuario introdujo un nombre en el campo de entrada se mostrara: "Nombre validado".

A continuación vamos a ver la implementación del *servlet* de validación. En él recogemos el campo de entrada `nombre` y según este vacío o tenga valor envía a través de la URL del JSP unos valores u otros.

```
package es.serv.capitulo2;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "validar", urlPatterns = {"/validar"})
public class validar extends HttpServlet {

  protected void processRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    String nombre = request.getParameter("nombre");

    if("".equals(nombre)){
      response.sendRedirect("
      formulario.jsp?nombre=&error1=1");
    }else{
      response.sendRedirect("
      formulario.jsp?nombre="+nombre+"&error1=2");
    }
  }

  @Override
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    processRequest(request, response);
  }
}
```

En el ejemplo hemos enviado los valores a través de la URL con el fin de facilitar la implementación y corrección de errores. Algunos desarrolladores comienzan la implementación del código utilizando este método para darle mayor visibilidad a las variables enviadas a través de las páginas. No obstante es más correcto enviar a través del método *POST* los datos.

Para darle una mayor fiabilidad al transporte de los datos y mejorar la comodidad también es habitual el uso de sesiones. En las tecnologías vistas, se puede crear una sesión y transportar entre páginas solo los datos que sean necesarios. Es importante tener en cuenta que cuando trabajamos con variables de sesión, es necesario comprobar con una estructura condicional si la página recibe ese dato dependiendo de la forma de envío *GET* o *POST*.

8.4 MODIFICACIÓN DE LA ESTRUCTURA DE LA PÁGINA WEB

Existen varias formas para modificar una página web. Una de las más revolucionarias en la actualidad, es la que se efectúa en el lado del cliente. A través de la estructura de árbol DOM que propone la organización estandarizadora W3C, es posible modificar la estructura de una página web sin que el código tenga que salir del propio navegador. Una de las ventajas de la utilización de esta tecnología, es la flexibilidad que supone a la hora de generar páginas interactivas tanto a nivel visual como funcional. Sin embargo, no podemos olvidar que si el código se ejecuta en el navegador del cliente, la fiabilidad de que ciertos procesos se realizan de una forma correcta es nula. Por poner un ejemplo, podemos fiarnos al enviar código que muestre una u otras capas en función de un menú. Pero no podríamos confiar en que el usuario guarde valores que previamente necesitemos tener almacenados para poder trasmitirle otra información. Por esta razón, es necesario combinar ambas tecnologías para dinamizar y modificar la estructura de una página web. En los casos que hemos visto a lo largo del capítulo, ya hemos visto como las páginas se generaban de una forma diferente en el cliente en función de las opciones del cliente o de las decisiones tomadas en el servidor.

En este punto, vamos a implementar un ejemplo basado en la validación del punto 8.3.3 referente al `nombre`. En el ejemplo vamos a generar una página que nos permita incluir una dirección de correo electrónico solo en el caso de que el usuario haya introducido previamente un nombre en el formulario. Para ello vamos a reutilizar el *servlet* de forma idéntica y solamente vamos a modificar el JSP. Como la variación en el JSP es mínima solo incluiremos el bloque donde se modifica parte de la estructura. El cambio se indica en negrita. La implementación del JSP queda de la siguiente manera:

```
....
<%
  if("1".equals(error1)){
    out.println("Es necesario rellenar el nombre");
  }else{
    if("2".equals(error1)){
      out.println("Nombre validado");

%>
<p>Introduce e-mail<input type="text" name="email"
  size="30"></p>
<%
    }
  }
%>
... .
```

La línea marcada en negrita, es incluida en el JSP del punto anterior. De esta forma el campo de entrada `email` solamente aparece si el campo de entrada nombre ha sido rellenado previamente. Es importante tener en cuenta que al generarse desde el servidor, este puede guardar previamente el nombre del usuario aunque este no vuelve a enviar más peticiones al servidor. Es una petición común en algunas páginas web, cuando para ofrecerte una información previamente te solicitan, un número de teléfono, un correo electrónico, etc.

ACTIVIDADES 8.1

- Configure un servidor para ofrecer páginas en cada una de las tecnologías vistas en el capítulo. Para ASP puede utilizar el que ofrece Windows incluido en el sistema operativo. La ruta por defecto donde se alojan las páginas es `inetpub`. Para la implementación de los JSP, puede utilizar un servidor de aplicaciones, como JDK, que incluye el servidor GlassFish o un servidor como Apache Tomcat. Existen *frameworks* como Eclipse o Netbeans que facilitan enormemente la generación de JSP y *servlets*. En el caso de PHP, puede utilizar un paquete tipo XAMP. Implemente el "hola mundo" en cada una de las tecnologías, visualice y compare las diferencias del resultado mostrado en el navegador.
- Cree una librería en ASP que tenga varias funciones que generen el área de distintas figuras geométricas planas. Implemente una página que permita introducir los datos para llamar a cada una de las funciones.
- Genere una librería de funciones en PHP que contenga la cabecera y el menú de una secuencia de 6 páginas. Añada en cada página un texto que la identifique con un número. Cree una serie de enlaces que le permitan redirigirse a las otras 5 páginas de forma directa.
- Implemente los casos de los puntos 8.2.1, 8.2.2. y 8.2.3.
- Implemente un formulario con ASP que contenga un campo a validar para cada uno de los controles vistos en el apartado 8.3.1.
- Implemente un formulario en PHP en el que utilice cada uno de los filtros del apartado 8.3.2.
- Realice la implementación de una validación con JSP y *servlets.* Deberá existir una página tipo JSP origen que contendrá un formulario. El formulario será enviado a un *servlet*. El *servlet* evaluará los datos. En el caso de que los datos no sean correctos, el *servlet* redireccionará a la página con cada mensaje de error. Si los datos son correctos el *servlet* enviará los datos a una página de destino nueva.

RESUMEN DEL CAPÍTULO

Las necesidades en la Red han propulsado que nazcan tecnologías que complementen las carencias del protocolo HTTP. Las páginas web dinámicas interactivas permiten que los objetivos actuales de las páginas web se cumplan.

Existe código se ejecuta en el navegador como HTML, XML, JavaScript, VBScript CSS, *applets* de Java. Por otro lado, están los lenguajes de servidor que dinamizan las páginas como son, ASP, JSP, *servlet,* PHP, CGI, PERL. Ambos se combinan para dinamizar los resultados en las aplicaciones web.

Los lenguajes de servidor implementan tecnologías y librerías para facilitar las necesidades web como reutilizar código, validar datos o combinar código. Unos lenguajes permiten con mayor facilidad que otros realizar según qué tareas.

Las tecnologías relacionadas con Java EE tienen dos tipos de ficheros básicos, uno es el JSP, orientado a la implementación de la interfaz y otro es el *servlet*, orientado al control.

ASP implementa el uso de controles para ejecutar código en el servidor. A través de ellos se pueden validar los datos introducidos por el usuario en el formulario.

PHP implementa el uso de filtros para la validación de código. Esto facilita la tarea, pudiendo incluir filtros referentes a un tipo de dato.

JSP dispone de la implementación de los *servlet* para dar el control de la validación a estos ficheros y aislar así la interfaz gráfica del usuario.

Al igual que ocurría a nivel de navegador, en el servidor se modifica la estructura de la página en función de las condiciones que marca el código.

La generación de las páginas interactivas permite que las nuevas necesidades en la Web se hagan realidad. La emulación a través de Internet de aplicaciones que habitualmente residen en local, exige que los protocolos y lenguajes en la Red completen sus carencias para permitir emular todos los comportamientos de una aplicación web en local.

EJERCICIOS PROPUESTOS

1. Realice una aplicación en la que existan tres formularios con los distintos tipos de validación vistos en el tema. Por ejemplo, el primer formulario puede tener nombre y correo electrónico, el segundo, los datos personales y, el tercero, los datos bancarios. Valide cada campo para que el usuario no pueda introducir un tipo de datos diferente al requerido. Valide también para que no pueda dejar campos en blanco. Se mostrará un mensaje por cada dato mal introducido en el formulario. Cuando el usuario introduzca los datos correctamente en el formulario introduzca estos datos en una sesión y muestre el segundo formulario. El formulario de datos deberá estar distribuido en 3 ventanas (3 formularios), de forma que no se pueda pasar a la segunda ventana del formulario hasta que no se introduzcan correctamente los campos del primer formulario. En una cuarta ventana se mostraran los datos introducidos en los formularios.

TEST DE CONOCIMIENTOS

1 Una página web interactiva:
- **a)** Permite variar el aspecto y comportamiento en función de las decisiones que toma el usuario.
- **b)** Permite variar el aspecto y comportamiento en función de las decisiones que toma el servidor.
- **c)** Permite variar el aspecto, pero no el comportamiento en función de las decisiones que toma el usuario.
- **d)** No permite variar el aspecto, pero sí el comportamiento en función de las decisiones que toma el usuario.

2 El código que se ejecuta en el navegador:
- **a)** No puede ser visualizado por el usuario.
- **b)** Es interpretado por el navegador y los subprogramas asociados al navegador.
- **c)** Puede ser visualizado por el usuario una vez se ha cargado la página.
- **d)** Las respuestas *b* y *c* son correctas.

3 El código que se ejecuta en el servidor:
- **a)** Resulta transparente para el navegador.
- **b)** Resulta transparente para el usuario.
- **c)** No puede ser visto fuera del ámbito del servidor.
- **d)** Las respuestas anteriores son correctas.

4 El termino *librería* visto en el tema proviene de:
- **a)** Una mala traducción de *library*.
- **b)** Que en inglés *librería* y *biblioteca* no son la misma palabra.
- **c)** Que las librerías son más pequeñas que las bibliotecas.
- **d)** La traducción correcta de *library*.

5 Los controles de ASP sirven para:
- **a)** Validar datos en el servidor.
- **b)** Emular el comportamiento de elementos de HTML en el servidor.
- **c)** Proporcionar elementos complejos como calendarios, en el servidor.
- **d)** Las respuestas anteriores son correctas.

6 Las librerías en los lenguajes de servidor permiten:
- **a)** Reutilizar código.
- **b)** Dificultar el entendimiento del código.
- **c)** Solo la respuesta *a* es correcta.
- **d)** Las respuestas *a* y *b* son correctas.

7 En las tecnologías relacionadas con Java EE:
- **a)** Se separan fácilmente la vista y el controlador.
- **b)** Se puede hacer lo mismo con los *servlets* que con los JSP.
- **c)** Se puede mostrar una página web solo con un *servlet*.
- **d)** Las respuestas anteriores son correctas.

8 En las tecnologías relacionadas con Java EE:
- **a)** Un JSP genera un fichero `.class` para poder ejecutarse.
- **b)** Un *servlet* genera un fichero `.class` para poder ejecutarse.
- **c)** Los *servlet* y los JSP genera un fichero `.class` para poder ejecutarse.
- **d)** Las respuestas *b* y *c* son incorrectas.

9 En ASP `CustonValidator` es:
- **a)** Un control de servidor que permite escribir un método para controlar un valor introducido por el usuario.
- **b)** Un control de cliente que permite escribir un método para controlar un valor introducido por el usuario.
- **c)** Un control que valida que los datos introducidos por el usuario que estén entre dos rangos.
- **d)** Un control que compara el valor de un control de entrada a un valor fijo.

10 En la validación con PHP:
- **a)** *filter_input()* obtiene un campo de entrada.
- **b)** *filter_input()* obtiene un filtro de entrada.
- **c)** *filter_input()* obtiene un campo de entrada y un filtro para éste.
- **d)** *filter_input()* obtiene datos de formularios.

9 Desarrollo de aplicaciones web híbridas

OBJETIVOS DEL CAPÍTULO

- ✓ Reconocer las ventajas de la reutilización de código y el aprovechamiento de la información existente.
- ✓ Identificar las librerías de código y las tecnologías para la creación de aplicaciones web híbridas.
- ✓ Crear aplicaciones web que recuperen y procesen repositorios de información ya existente.
- ✓ Crear repositorios específicos a partir de información existente en Internet y almacenes de información.
- ✓ Utilizar librerías de código para incorporar funcionalidades a una aplicación web.
- ✓ Programar servicios y aplicaciones web utilizando información y código generado por terceros.
- ✓ Probar, depurar y documentar las aplicaciones generadas.

En los capítulos anteriores hemos visto como desarrollar aplicaciones web para cualquier tipo de negocio. A menudo hemos podido comprobar cómo ciertos servicios que ofrecían otras páginas web nos podían resultar interesantes para el objetivo de nuestra aplicación web. Con el paso de los años los desarrollos web se están extendiendo cada vez a más ámbitos. Además las aplicaciones web que se desarrollan cada vez son más complejas y requieren de más utilidades para completar su funcionalidad. Pongamos un ejemplo, queremos construir una aplicación web que ofrezca un servicio de alquiler de apartamentos en la playa. Es sabido que uno de los condicionantes del precio a la hora de alquilar un apartamento en la playa es la cercanía a la orilla del mar. Por lo tanto, uno de los ejes centrales de nuestra aplicación web va a ser la posibilidad de poder orientar cada apartamento en un mapa. El verdadero objetivo de nuestro negocio es gestionar los apartamentos a alquilar y ofrecer el servicio a los clientes potenciales. No obstante no podemos prescindir del mapa para situar cada uno de estos apartamentos. Desarrollar un sistema de mapas para una aplicación de este calado, convierte el proyecto en inviable. El desarrollo de un sistema de mapas supera con creces el presupuesto con respecto al beneficio que podemos obtener ofreciendo alquileres de pisos en la playa. Afortunadamente existen desarrollos que implementan mapas, como por ejemplo Google Maps. Si analizamos un momento la manera de proceder con nuestra aplicación, sería la siguiente: buscamos un apartamento a alquilar, una vez encontramos uno que nos gusta, abrimos la página de Google Maps e introducimos la calle y el número correspondiente. Una vez realicemos la búsqueda Google Maps nos mostrara la ubicación del apartamento. En caso de que el apartamento sea adecuado para el cliente, volverá a la aplicación web y alquilará el apartamento.

Esta funcionalidad parece solucionar nuestro problema. Aun así, nos quedan dudas sobre si la trazabilidad de funciones se puede mejorar. Salir de una página web para entrar en otra y luego volver no es lo más aconsejable para ofrecer un servicio. Quizás perdamos clientes que no sepan continuar este flujo, que se aburran de cambiar de navegador o que simplemente no lleguen a cumplir los pasos que hemos fijado para poder visualizar la orientación del apartamento. Puesto que la plataforma del mapa ya tiene todo preparado para introducir la calle y el número, ¿no podríamos desde la aplicación de alquiler solicitar el mapa de Google Maps pasando como parámetro la calle y el número? En el caso de conseguir que Google nos facilitara mostrar sus mapas en nuestra aplicación, nuestra página ya no estaría formada solo por código nuestro, tendría también código desarrollado por Google. A esto se le llama, aplicación web híbrida. En el siguiente tema vamos a ver qué es una aplicación web híbrida, que tipos existen, cuales son las posibilidades y como se implementan.

Una aplicación web híbrida (en inglés *mashup*) es, es una página o aplicación web que utiliza contenidos o servicios de terceros y los combina para crear una aplicación web nueva.

9.1 REUTILIZACIÓN DE CÓDIGO E INFORMACIÓN

Las aplicaciones web híbridas o *mashup*, son accedidas a través de una interfaz pública o usando un API. El concepto de *mashup* implica una integración fácil y rápida. Este tipo de servicios son ofrecidos y consumidos a través del protocolo de comunicaciones HTTP.

Existen compañías como YouTube, Yahoo, Microsoft, Google, Flickr o eBay que ofrecen *mashups* para integrar sus productos o servicios con otras aplicaciones web.

Debemos tener en cuenta que incluir código embebido en una aplicación web, no puede considerarse como una aplicación web híbrida. Por ejemplo, incluir un vídeo embebido de YouTube en una aplicación web, no convierte a esta en híbrida. Es necesario que la aplicación web origen acceda a la información externa o procese los datos del sitio que nos ofrece el servicio, para que este servicio ofrezca un valor añadido al usuario.

9.1.1 ARQUITECTURA DE UNA APLICACIÓN WEB HÍBRIDA

La arquitectura de una aplicación web híbrida está compuesta por 3 partes principales. Por un lado el proveedor de contenidos, el servidor del sitio web *mashup* y el cliente web. A continuación visualizamos las tres partes de la arquitectura.

Figura 9.1. *Elementos de la arquitectura mashup*

En la imagen vemos las tres partes de un *mashup*. En el lado derecho visualizamos el proovedor de contenidos, en esta caso es Google Maps, que ofrece su servicio de mapas. En el centro esta situado el servidor de la nueva página híbrida generada. Por último, en la parte izquierda se encuentra el cliente, que es quien accede a la página híbrida.

- **Proveedor de contenidos**. Es la fuente de datos. Los datos suelen estar disponibles a través de una interfaz pública o utilizando un API y con diferentes protocolos como RSS, Atom, Screen scraping o servicios web.
  - **RSS**. Son las siglas de *Really Simple Syndication*. Traducido significa, sindicación realmente simple. Teniendo en cuenta que sindicar en el caso que nos atañe, significa tomar nota, el resultado de la traducción de RSS sería algo como, tomar notas de forma simple. Esta tecnología está basada en una estructura XML que permite compartir contenido de una web determinada. RSS permite distribuir contenidos de una web sin necesidad de un navegador. De esta forma estos contenidos pueden ser mostrados en una aplicación o en otra página web. Los formatos de extensión para RSS son `.rss` o `.xml` y representan objetos de tipo MIME, application/rss+xml. Conviene saber que la primera versión de RSS, también es conocida como RDF, por tanto, algunos canales RSS se etiquetan como RSS 1.0 ó RDF. La extensión en este caso es `.rdf`.

Figura 9.2. *Icono representativo de RSS*

En la imagen anterior visualizamos una de las imágenes habituales que encontramos en las páginas web que ofrecen canales de redifusión de contenido RSS. Es habitual que esta imagen proporcione un enlace al documento XML correspondiente al contenido a distribuir.

  - **Atom**. Esta tecnología está basada en el lenguaje XML permite estructurar y distribuir contenidos web, como ocurría en RSS. No obstante Atom no se basa ni se corresponde con ninguna versión de RSS. El formato del documento XML, es similar y tiene un mismo objetivo. Se trata de poder distribuir información de una web a través del documento XML. Las mejoras que incorpora con respecto a la tecnología RSS es que un documento Atom tiene capacidad para contener más información, siendo esta más compleja y consistente. Debido a esta mejora en la capacidad aumenta el grado de dificultad de esta tecnología. El

tipo de extensión que maneja es `.atom` o `.xml` y construye un objeto tipo MIME, application/atom+xml. Existen dos estándares que son RFC 4287 y RFC 5023. Los símbolos que utilizan las páginas para indicar que disponen de canales Atom para permitir la distribución de contenidos a un lector de Atom son similares a los de RSS, aunque indican que se trata de Atom y no de RSS.

Figura 9.3. *Icono representativo de ATOM*

En la figura anterior vemos dos tipos de representaciones que utilizan las páginas para indicar que su web dispone de canales ATOM. En el de la izquierda no se indica la versión, en el de la derecha indica que la tecnología que ofrece en canal es ATOM.

- **Screen scraping**. Traducido al castellano significa *raspado de pantalla*, es una técnica de programación que toma la presentación de una información que se visualiza por pantalla. Normalmente se extrae texto aunque también se puede seleccionar información gráfica. El programa *screen scraping* accede a través de ingeniería inversa a la página del proveedor y procesa el contenido HTML recopilando la información que se desea extraer. Para utilizar esta tecnología no es necesario que el proveedor del contenido realice ninguna tarea adicional como ocurría en RSS o Atom. Esta técnica esta en desuso debido a los inconvenientes que presenta como, las webs que no la autorizan, la complejidad, sobrecarga de los servidores y la propensión a fallos inesperados. En algunos casos puede ser la única opción de recuperar contenidos aunque la tendencia a la hora de crear nuevos *mashup* es utilizar API.
- **API**. Una interfaz de programación de aplicaciones es un conjunto de funciones y procedimientos que ofrecen una biblioteca para utilizar otro software, creando de esta forma una capa de abstracción. De esta manera, a través de una API, el cliente accede un servicio web del proveedor de contenidos. Con la respuesta obtenida y el código generado en el servidor, se ofrece una respuesta para conformar la aplicación web híbrida en el cliente. Los beneficios de esta tecnología con respecto a *screen scraping* son notables; La API es del proveedor, por tanto, existe un acuerdo entre proveedor y cliente, es más sencillo y está documentado. El proveedor puede disponer de servidores específicos para este servicio, dando además robustez y una gestión de errores adecuada.

Figura 9.4. *Elementos de una arquitectura mashup con API*

En la figura anterior vemos como el proveedor de contenidos facilita una API al servidor de páginas híbridas. El servidor une los contenidos propios de la página con los datos recibidos del proveedor y genera una página híbrida que pone a disposición del cliente.

- **El sitio mashup**. Es la aplicación web nueva que provee de un nuevo servicio utilizando diferentes fuentes de información de las que no es dueño. El sitio *mashup*, hará uso de las tecnologías de las que disponga el proveedor de contenidos para conseguir generar la aplicación híbrida.
- **El web browser cliente**. Es la interfaz del usuario del *mashup*. Es una aplicación web. En este bloque debemos tener en cuenta que el contenido puede ser mezclado también en el navegador web del cliente. Para ello se puede hacer uso de lenguajes como JavaScript o AJAX.

9.1.2 COMUNICACIÓN EN LA ARQUITECTURA MASHUP

A continuación vamos a ver un esquema de las posibles vías de comunicación desde el navegador web del cliente hasta el proveedor de contenidos, pasando o no por el servidor de aplicaciones.

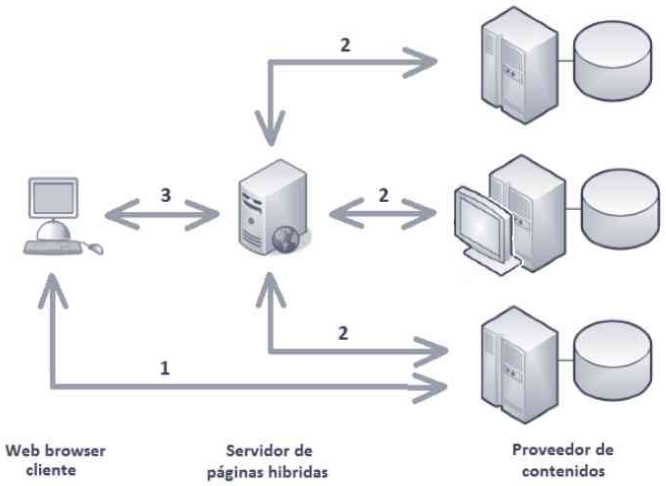


Figura 9.5. *Comunicación de elementos en una arquitectura mashup*

En la imagen anterior observamos las formas en que podemos acceder a una página web híbrida o a un proveedor de contenidos. Hemos indicado los casos con un número desde el 1 al 3, siendo estos casos los siguientes.

- **Caso 1**. El usuario accede directamente al contenido que ofrece el proveedor. En este caso el contenido se mezcla en el cliente a través de JavaScript.
- **Caso 2**. En este caso el acceso al proveedor de los contenidos se realiza desde el sitio web del *mashup*. Los contenidos en este caso se mezclan en el servidor de aplicaciones. Una vez mezclados son enviados al navegador web cliente.
- **Caso 3**. En el tercer caso el usuario accede desde el navegador al servidor de páginas híbridas. En los casos 2 y 3 es posible acceder a través de las API y con las técnicas no demasiado recomendables de *Screen Scraping*.

9.1.3 DIVISIÓN POR CATEGORÍAS DE LOS MASHUP

Las aplicaciones web híbridas suelen estar divididas en cuatro grandes categorías. Esta división enmarca unos objetivos diferentes y muy extendidos en la Red. Por un lado están las aplicaciones híbridas compuestas a través de mapas, otra categoría son las fotos y los vídeos, en un tercer marco se encuentran las búsquedas y compras y por último las noticias.

- **Mashups de Mapas**. Los *mashups* de mapas muestran información sobre un mapa o imagen satélite. Existen unas normativas con respecto a los mapas, para controlar ciertos parámetros incluso algunos países no permiten que ciertas zonas de los mapas se visualicen en estos. Estos *mashups* están muy extendidos en infinidad de aplicaciones web en las que es necesario precisar la situación de algo.

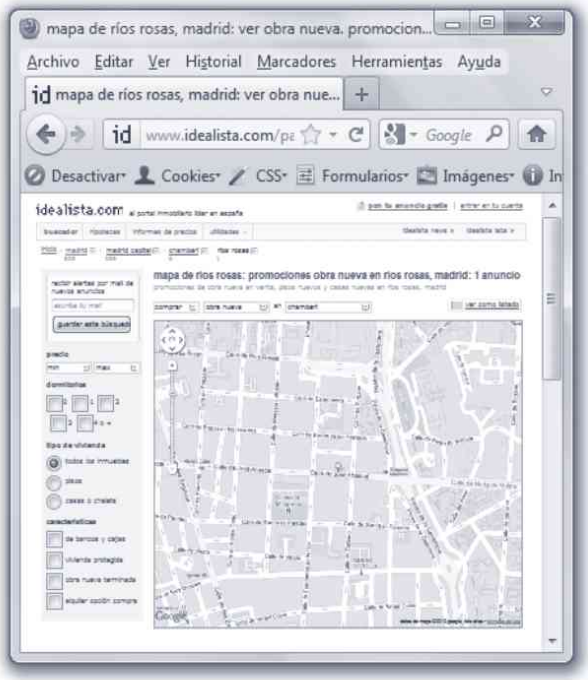


Figura 9.6. *Portal de idealista con un mashup de mapa*

- **Mashups de Fotos y Vídeo**. Este tipo de *mashups* generan el contenido a partir de servidores web que almacenan grandes cantidades de fotografías y vídeos. Normalmente el proveedor de contenidos permite realizar ciertas tareas sobre estos objetos, como, editar, ordenar, coleccionar, compartir, etc.

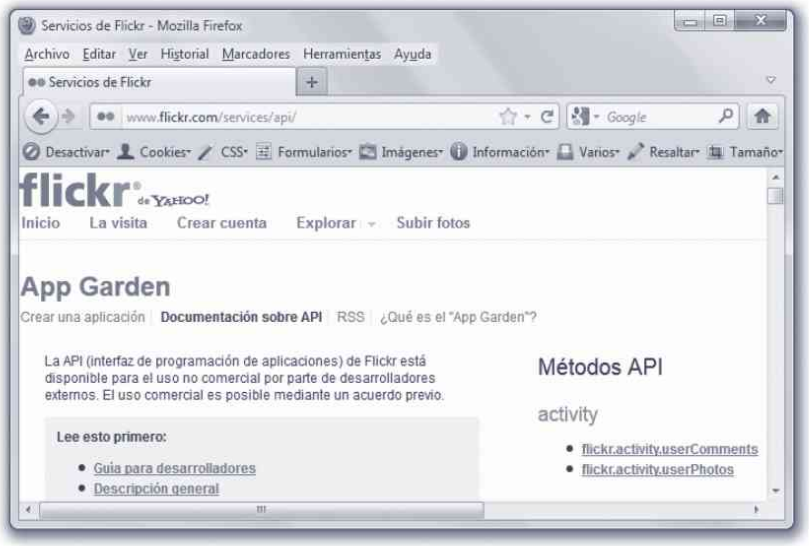


Figura 9.7. *Portal de Flickr que provee imágenes y vídeos*

En la imagen anterior podemos observar como el portal Flickr proporciona en la propia página web los métodos de la API para crear una aplicación web nueva híbrida que acceda a sus contenidos.

- **Mashups de Compras**. Estas aplicaciones híbridas permite realizar búsquedas de las mejores ofertas comerciales. Hace años eran conocidas como *web aggregators*, aunque ese término se refiere a los agregadores en general que nos permiten incluir fuentes de noticias en formatos RSS y Atom entre otros. Un *mashup* de datos facilita la comunicación *Bussiness to Bussiness* (B2B). Estos *mashups* conducen a servicios comerciales en la Red a través de API.

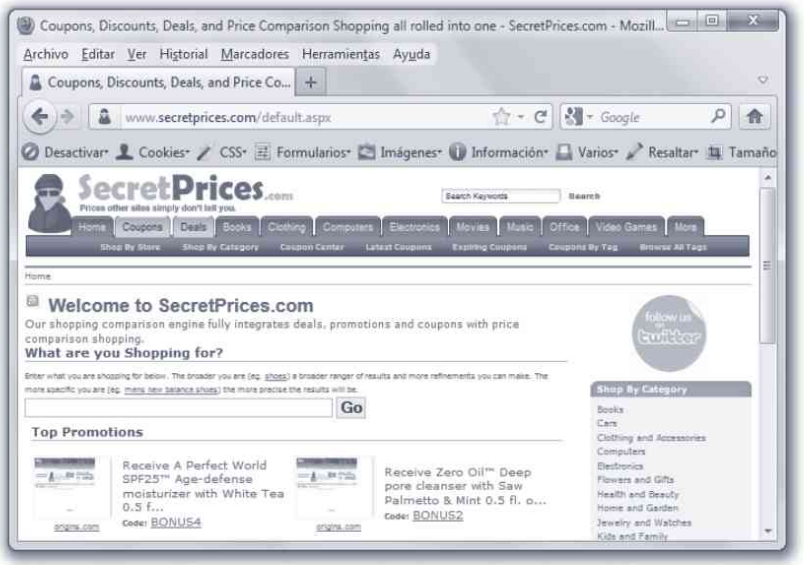


Figura 9.8. *Portal de SecretPrices, mashup de compras*

En la imagen anterior vemos el portal de SecretPrite. En este portal podemos buscar productos a los mejores precios, cuyos productos pertenecen a otros portales como Amazon, eBay, fnac, etc.

- **Mashups de Noticias**. Los *mashups* de noticias seleccionan fuentes de noticias a través de los protocolos RSS y Atom. Existe la posibilidad de agrupar noticias según ciertos parámetros de preferencias del usuario. Es importante que sepamos que las visitas de los portales que proveen las noticias disminuyen, debido a que las noticias son leídas desde el navegador web cliente.

Figura 9.9. *Portal digg.com, mashup de noticias*

En la imagen anterior vemos el portal digg.com que reune noticias de diferentes proveedores. Este portal está controlado casi enteramente por los usuarios del sitio web.

9.2 UTILIZACIÓN DE INFORMACIÓN PROVENIENTE DE RESPOSITORIOS. UDDI (UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION)

UDDI, como vimos en el capítulo de los servicios web, proporciona un método estandarizado para la publicación y descubrimiento de información sobre servicios web. Este proyecto que esta sufragado por la organización sin ánimo de lucro OASIS, es una iniciativa de la industria con el fin de crear una plataforma independiente para la publicación de servicios web. UDDI se centra en el proceso de descubrimiento de la arquitectura orientada a servicios.

El proyecto UDDI es una iniciativa que se comunica con los clientes a través de un directorio común que cumple con los estándares de UDDI. Este es un registro único conceptual que pretende distribuir en muchos nodos replicas de los datos de los servicios web publicados por las empresas.

En todo este proceso existen una serie de pasos que debemos seguir para descubrir un servicio. A continuación vamos a describir cuáles son cada uno de estos pasos a seguir.

9.2.1 EL SERVICIO DE DIRECTORIO

A la hora de publicar un servicio web XML, debemos buscar un servicio de directorio. Para ello debemos ubicar el servicio web XML en el UDDI o en otro servicio de directorio.

Como ocurre con otros recursos en Internet, para buscar un servicio web XML, necesitamos un medio. Los directorios de servicios web XML, proporcionan una ubicación centralizada en la que los proveedores pueden publicar información acerca de sus servicios web. La especificación UDDI (*Universal Descripcion, Discovery and Integration*) define un medio estándar para publicar y descubrir información acerca de los servicios web XML. Los esquemas de los que dispone UDDI en XML definen cuatro tipos de informaciones que permitirán a un programador utilizar servicios web publicados. Estos tipos son: información comercial, información de servicios, información de enlace e información de la especificación del servicio.

En la imagen siguiente mostramos como se publica un servicio web XML en el servicio de directorio UDDI.

Figura 9.10. *Publicación de un servicio web XML*

En la figura anterior vemos como el cliente ubica un servicio web XML en un servicio de directorio UDDI. Cuando este está ubicado se asocia una dirección URL al documento de descubrimiento.

9.2.2 EL DESCUBRIMIENTO DE UN SERVICIO

El descubrimiento de servicios web XML es un proceso que consiste en localizar o descubrir uno o varios documentos relacionados que describen un servicio web XML. Para descubrir estos servicios se utiliza el documento de descubrimiento de servicios web (archivo `.disco`).

Un documento de descubrimiento (archivo `.disco`), es un documento XML que contiene vínculos a otros recursos que describen al servicio web XML. De esta forma se permite el descubrimiento mediante programación de un servicio web XML. En el siguiente código mostramos un ejemplo de documento de descubrimiento XML que incluye referencias (URL) a otros documentos.

```
<?xml version="1.0" encoding="utf-8" ?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://schemas.xmlsoap.org/disco/">

   <contractRef ref="http://www.ejemplo.com/pagina.asmx?wsdl"
     docRef="http://www.ejemplo.com/pagina.asmx"
     xmlns="http://schemas.xmlsoap.org/disco/scl/" />

   <soap address="http://www.ejemplo.com/pagina.asmx"
     xmlns:q1="http://tempDIR.org/" binding="q1:paginaSoap"
     xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

A través del proceso de descubrimiento, los clientes del servicio web XML, obtienen la información de que un servicio web XML existe y el sitio web donde se puede buscar el documento de descripción del servicio web XML (WSDL).

El documento de descubrimiento es un contenedor que suele incluir vínculos (URL) a otros recursos que proporcionan información del servicio web XML.

En la siguiente figura podemos ver cómo se establece la comunicación para descubrir un servicio web XML.

Figura 9.11. *Descubrimiento de un servicio web*

En la figura anterior, observamos como el cliente solicita el documento de descubrimiento de un servicio web. Una vez se ha solicitado el servidor devuelve al cliente este documento. En este caso el documento se llama `desc.disco`.

Aunque aparentemente parece que un sitio web que implementa un servicio web XML permite el descubrimiento, no tiene porque ser así. Puede existir un sitio para la descripción del servicio y otro para el descubrimiento. También puede haber casos en los que el uso del servicio web sea privado por lo que no existirá un medio público para buscarlo.

9.2.3 DESCRIPCIÓN DE UN SERVICIO

Los servicios web se comunican mediante mensajes basados en XML, cuando estos cumplen una descripción de servicio publicada. Como hemos visto en capítulos anteriores la descripción de un servicio, está escrito con una gramática XML a la cual se la denomina WSDL (Lenguaje de descripción de servicios web). Este documento define el formato de los mensajes comprensible para el servicio web XML. Este documento sirve como acuerdo entre el proveedor del servicio y el cliente que consume el servicio. Así el cliente sabe como interactuar con el proveedor del servicio. Este comportamiento se basa en determinar los modelos de mensajes que se definen y admiten en el documento. Por lo tanto, estos modelos dictan a nivel de concepto al consumidor del servicio lo que puede ocurrir cuando se envía un mensaje con el formato correcto al servicio web XML.

En un modelo de solicitud respuesta como los que vimos en capítulos anteriores, el WSDL define por un lado el esquema de mensaje que SOAP utilizará para invocar a un método concreto. El modelo por otro lado determina el mensaje SOAP que deberá recibirse de respuesta. Estos esquemas que definen los formatos de mensajes SOAP se pueden definir dentro de la descripción del servicio o externamente y ser importados en la descripción.

En la imagen siguiente visualizamos como el cliente solicita la descripción de un servicio. El servidor devuelve un fichero WSDL al cliente del servicio web XML.

Figura 9.12. *Descripción de un servicio web*

En la figura anterior observamos como el cliente solicita la descripción del servicio `WebService.WSDL`. El servidor devuelve el documento del servicio web XML asociado.

9.2.4 FORMATO DE CONEXIÓN DE UN SERVICIO

Los servicios web no necesariamente tienen que comunicarse a través del protocolo HTTP. Pueden utilizar otro tipo de comunicación puesto que esta capa es independiente de las demás. Aun así, es cierto que este protocolo es uno de los más convenientes para utilizar. Las razones son la expansión de Internet y el uso generalizado del protocolo HTTP en prácticamente todos los medios que se conectan a Internet.

HTTP

Dentro del protocolo HTTP, existe el protocolo que envía las variables a través de la URL, *GET* y el que las enviar en oculto *POST*. Estos protocolos envían pares de parámetros, nombre y valor, junto con la solicitud asociada. Cada uno se compone de una serie de encabezados en la solicitud HTTP, que entre otras define lo que el cliente le solicita al servidor. El servidor a su vez responde al cliente con una respuesta HTTP similar, si procede.

SOAP

SOAP es un protocolo simple y ligero basado en XML. Sirve para intercambiar información estructurada y de tipos. El objetivo del diseño SOAP consiste en conseguir la mayor simplificación y además proporcionar un mínimo de funcionalidad. Este protocolo define un marco para la mensajería que excluye del mensaje la semántica de la aplicación y el transporte. Por esta razón se puede implementar este protocolo con diferentes métodos de transporte. Este protocolo es muy modular y extensible.

En SOAP se puede utilizar la arquitectura existente de Internet. Además es fácil obtener la aceptación por parte de cualquier sistema compatible con los estándares de Internet.

SOAP se compone de 4 partes principales:

- La primera parte define un sobre extensible para encapsular los datos. Este sobre es obligatorio. Este sobre define un mensaje SOAP y es la unidad más básica de intercambio entre mensajes SOAP. Este punto es el único obligatorio de la especificación.
- La segunda parte define las reglas de la codificación de los datos opcionales para representar tipos de datos definidos por cada aplicación y un modelo uniforme para datos no sintácticos.
- La tercera parte define el intercambio de mensajes, solicitud y respuesta. Cada mensaje SOAP, es una transmisión en un sentido. Esta parte es opcional.
- La cuarta parte de la especificación, define un enlace entre SOAP y HTTP. Esta especificación también es opcional, de esta forma se puede combinar SOAP con cualquier otro protocolo de transporte.

Figura 9.13. *Formato de conexion de un servicio web*

En la imagen anterior vemos como el cliente del servicio web XML, solicita el servicio, en el otro lado el servicio devuelve la respuesta.

9.2.5 IMPLANTACIÓN DE UDDI EN LA NUBE

Los servicios web han sido bastante bien acogidos para comunicar diferentes plataformas, como hemos ido viendo en los capítulos, estos están basados en XML, que es un lenguaje universalmente aceptado. Los diferentes elementos que se utilizan para crear y consumir un servicio web, son usados para comunicar plataformas con diferentes arquitecturas o sistemas, las empresas ven necesidades de comunicar sus tecnologías, e implantan servicios web para comunicarlas. Por el contrario, la publicación de servicios web a través de UDDI, no es una realidad. Los esfuerzos por crear un dominio único, replicando en cada uno de los documento de descubrimiento de cada servicio web, no han dado los frutos que se esperaba. Quizás las pretensiones de implementar todas las etapas de los servicios web a la vez, sea demasiada carga para que los estándares maduren de una forma adecuada. A la hora de valorar los resultados de la creación de servicios web, se han ido viendo algunos cambios en la arquitectura que mejoran notablemente el funcionamiento. Tampoco queda demasiado claro cuáles son los métodos más apropiados para publicar servicios web. Quizás la rápida extensión de las API vaya en detrimento de la publicación de estos servicios, puesto que estas cumplen con algunos objetivos de comunicación entre diferentes plataformas.

Las empresas implementan sus servicios web a nivel privado para comunicar sus plataformas, pero no ven la necesidad de publicar estos servicios en la Red para que otros usuarios puedan utilizarlos.

Es posible que cuando los servicios web maduren, se encuentren necesidades suficientes para justificar la publicación de servicios web. Al surgir estas necesidades los servicios serían publicados en dominios de acceso público como prometía el UDDI.

No debemos pensar que los servicios web están incompletos o que carecen de la funcionalidad para la que fueron pensados. La potencia de los servicios web queda intacta aun con la falta de expansión de UDDI. El poder comunicar dos o más maquinas que tienen plataformas diferentes, era la necesidad prioritaria cuando se pensó en esta arquitectura. Por lo tanto, este objetivo esta conseguido, quizás no haya suficientes motivos para crear un dominio común de servicios web ó quizás en el futuro dispondremos de un directorio común en el que probablemente podríamos consultar a nivel mundial todos los servicios web existentes. Puede que todas las empresas incluyan sus servicios web publicados o quizás ninguna quiera tenerlos en la Red. Al no publicar un servicio web, aumenta la transparencia de la comunicación entre las diferentes plataformas.

9.3 INCORPORACIÓN DE FUNCIONALIDADES ESPECÍFICAS

Como veíamos al principio del capítulo, una de las formas de generar una página web híbrida era hacer uso de una API. Una API es una interfaz de programación de aplicaciones. A través de los métodos de la API, podemos comunicarnos con una aplicación externa. La interfaz de programación de aplicaciones (API), proveerá de los métodos para poder realizar las tareas para las que está programada. En la API se describen los métodos que realizan las tareas y provee de acceso a ellos desde una aplicación cliente.

Una API puede incluir especificaciones para manejar objetos, estructuras de datos, variables, rutinas, etc. Una API puede variar su forma adaptándose al entorno de la arquitectura, puede seguir normas de estándares, como POSIX, realizar funcionalidades sobre sistemas operativos, como la API de Windows, o servir como biblioteca de un lenguaje de programación, como la API de Java.

Las API dependen del lenguaje, responden a una sintaxis y unos elementos del lenguaje en particular. De esta forma se facilita el uso de la misma. Aunque una API estará escrita en un lenguaje de programación, o puede que la combinación de varios, es deseable que los métodos de la API puedan ser llamados por el mayor número de lenguajes posibles, sobre todo si la API no está vinculada a un sistema especifico.

Para poder conocer cuáles son las tareas de una API, normalmente suele tener una descripción de cuáles son las acciones que se pueden llevar a cabo. Por ejemplo si queremos incluir la posibilidad de manejar fechas en Java deberemos incluir la el fichero `import java.util.Date`; De esta forma importaremos el fichero que contiene los métodos que podemos usar para el manejo de fechas en Java. La tecnología Java, alberga un fichero en la Red que detalla la descripción de todas sus funciones. Al ser un lenguaje orientado a objetos, la API provee de un conjunto de clases para efectuar las tareas que se necesitan dentro de un programa. Si ponemos en google API de Java 7, probablemente nos salga el primero el enlace al API, que actualmente es el siguiente: http://docs.oracle.com/javase/7/docs/api/.

En el enlace se incluyen todas las características, funcionalidades y capacidades de la API de la versión 7 de Java.

Otra forma de llamar a una API, es realizar una inclusión a través de una URL desde una página web, al incluir una URL donde venga la implementación de la API, podremos acceder a sus métodos y ejecutar toda la funcionalidad, mostrando en nuestro navegador los resultados. Este es el caso por ejemplo de Google Maps, que ofrece varias API para gestionar sus mapas desde un *mashup*. La inclusión de la URL donde se encuentra la API se ejecutan en JavaScript, y podemos realizarla de la siguiente manera:

```
<script src="http://maps.google.com/maps?
  file=api&v=2&key=abcdefg&sensor=true_or_false"
  type="text/javascript">
</script>
```

Esta es la estructura que presenta la llamada a la carga de la API desde el lenguaje JavaScript, lenguaje de navegador. Al realizarse las llamadas desde un lenguaje que es compatible con el resto de tecnologías web, todos los usuarios pueden ejecutar código JavaScript para implementar la funcionalidad de los mapas. El lenguaje de servidor utilizado resulta transparente con respecto al uso de esta API que ofrece GoogleMaps.

Cuando hablamos de una API en la Web, como la que ofrece GoogleMaps, solemos referirnos a las llamadas o acciones que devuelven hipertexto, dicho de otra manera código HTML. Se envía una solicitud a través de los métodos implementados en la API y se recibe una respuesta, que en algunos casos es código HTML o indicaciones para modificar elementos de este código enviado. En algunos casos también se devuelve código XML. La Web 2.0 se convierte en realidad gracias en parte a las API y tecnologías que nos permite implementar los *mashups*.

9.3.1 FUNCIONALIDADES PARA COMPARTIR CONTENIDO

En la Red se ha extendido mucho el uso de API, la facilidad de uso y la baja curva de aprendizaje, convierten a las API en un buen aliado para complementar una Web con información proveniente de terceros. Esto unido al gran auge que han tenido los portales sociales, tanto los relacionados con las redes sociales, como Tuenti, Facebook, o los de compartición de fotos como Flickr o Photobucket, hace que estas compañías faciliten API para que los usuarios incluyan funcionalidades en sus propias páginas. Una interfaz muy popular es la que ofrece YouTube para incorporar vídeos y funcionalidades relacionadas con estos en una página *mashup*. Estudiar cada una de las API requiere de una extensión amplia de la que no disponemos en esta sección del capítulo, no obstante vamos a indicar un fragmento de texto en el que recuperamos la descripción de un vídeo a través del API de YouTube:

```
<feed>
  <entry>
    ...
    <media:group>
      ...
    </media:group>
    <gd:comments>
      <gd:feedLink href='http://gdata.youtube.com
        /feeds/api/videos/VIDEO_ID/comments'/>
    </gd:comments>
  </entry>
</feed>
```

En el código anterior recuperamos los comentarios de un *feed* de tipo vídeo, la URL incluida en el feedLink, nos sirve para comunicar la petición al proveedor de los contenidos.

9.3.2 FUNCIONALIDADES PARA MOSTRAR MAPAS

En infinidad de webs nos encontramos con la necesidad de utilizar mapas para situar zonas de venta de inmuebles, alquileres de coches, servicio de hoteles, hostales, localización de establecimientos, etc. Casi todos los servicios de la Web, están relacionados de alguna manera con una localización geográfica. Como ya hemos indicado en los apartados anteriores, implementar un mapa para ofrecer un servicio de este tipo, es inviable. Además el uso de los servidores de mapas, es algo que está sometido a regulaciones, cuyo cumplimiento es obligatorio. Uno de los mapas más extendidos es el que ofrece el portal de búsquedas Google en su subsección de mapas. Google, implementa varias API para poder ejecutar sus objetos. A continuación vamos a ver cuál es la implementación del manejo de un mapa a través de una de las API que nos presenta.

A continuación vamos a ver el código que presenta Google en su ejemplo "Hello World" de GoogleMaps, es decir, que quiere decir el "hola mundo" de Google Maps.

```
<!DOCTYPE html "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html;
      charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src= "http://maps.google.com/maps?file=api&amp;
      v=2&amp; key=abcdefg&sensor=true_or_false"
      type="text/javascript">
    </script>
    <script type="text/javascript">
      function initialize() {
        if (GBrowserIsCompatible()) {
          var map = new
            GMap2(document.getElementById("map_canvas"));

        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
        }
```

```
        }
      </script>
    </head>
    <body onload="initialize()" onunload="GUnload()">
      <div id="map_canvas" style="width: 500px; height:
        300px"></div>
    </body>
  </html>
```

En el código anterior, presentamos la visualización de un mapa de 500 por 300 que centra la zona en el Palo Alto de California. Para modificar el ejemplo que se muestra a continuación en nuestro servidor de aplicaciones, es necesario solicitar una clave para el API de Google Maps. Esto se puede realizar en una página del propio dominio de Google. Sustituyendo esta clave podremos visualizar y modificar el código para ver las diferentes funcionalidades. La clave está identificada con el nombre de variable `key`.

En la primera etiqueta del *script*, se incluye la ubicación de las funcionalidades de la API.

A través de la clase `GMap2`, se genera un único mapa en una página. Se pueden crear más instancias de mapas en la página instanciando la clase.

Inicializamos los parámetros del mapa a través del método `setCenter`.

Una vez creado el mapa e inicializado los valores, visualizamos el mapa. En este caso hacemos uso de una estructura `div`. Como el objeto creado con el mapa se llama `map_canvas`, incluiremos el objeto con `id=map_canvas` dentro de la estructura `div` para visualizar aquí el mapa. Con el fin de evitar posibles errores, comprobamos a través los eventos del `body` que la página se ha cargado completamente antes de inicializarla.

Aunque la lista de opciones de las que dispone el API de GoogleMaps es muy extensa, solo incluimos un ejemplo simple, puesto que no podemos abarcar todos los puntos con la profundidad que se merecen.

9.4 SINDICACIÓN Y FORMATOS DE REDIFUSIÓN. RSS (RICH SITE SUMMARY), ATOM

La sindicación o redifusión consiste en el reenvío de contenidos desde el origen de los datos hasta un destinatario, que a su vez puede convertirse en nuevo emisor de esos contenidos. En Internet se publican contenidos en las páginas a través de los editores de los contenidos o bien incluyendo el propio contenido en el código que se ejecuta en el servidor. Esta es la forma habitual de mostrar contenidos en una página web. La gran expansión de algunas aplicaciones web hace casi momentánea la actualización de estos contenidos. Esto ha propiciado que algunos portales no puedan competir en rapidez a la hora de actualizar los contenidos de sus páginas web. Además la fiabilidad que dan ciertos contenidos provenientes de portales ampliamente conocidos es bastante mayor.

Pongamos un ejemplo, si existen varios portales que publican noticias diariamente, y solo uno de ellos es un portal perteneciente a un conocido periódico de tirada mundial, ¿cuál es la noticia que tiene más credibilidad?

Los portales web de contenidos de gran magnitud y con amplio público, por otro lado pueden centrase en emitir noticias a nivel global y convertir la plataforma web, en excesivamente amplia. Imaginemos que queremos ver las noticias relacionadas con un equipo de futbol en un portal de noticias deportivas. Si entramos en la página, nos vemos obligados a recorrer el menú para llegar a esa sección. Podríamos incluir en favoritos la ruta y pulsar en el enlace cuando queramos ver las noticias del equipo de futbol. A veces, consultar un solo diario deportivo, nos va a

proporcionar una visión un tanto subjetiva. Por lo tanto, lo ideal sería poder leer noticias de varios diarios deportivos. En este caso deberíamos recorrer las rutas de todos los diarios hasta llegar a nuestro equipo o incluir en favoritos cada una de las URL de la sección del equipo. Esto se puede convertir en una tarea algo pesada, ocasionándonos una pérdida de tiempo. Es muy probable que muchos días no visitemos las páginas. En el caso de que surja una noticia importante sobre nuestro equipo y se publique en alguno de los portales que visitamos, nos quedaremos sin conocerla.

A partir de estas suposiciones, surge cierta necesidad de agrupar los contenidos de las diferentes páginas. De esta forma el lector podrá acceder de una forma mucho más sencilla a los contenidos específicos de una página web. Sería ideal, por tanto, poder recoger solo ciertos contenidos asociados a una categoría de noticias y poder visualizarlos, en una nueva página web. Podríamos realizar esta operación extrayendo los contenidos de la categoría elegida de todas las páginas web que queramos visitar frecuentemente. Estos contenidos los agruparemos en una la nueva página. Esta página estará formada por contenidos de los diferentes portales y además tendrá el código de la propia página que mostrara y agrupara los contenidos recibidos. Por lo tanto, la nueva página web será un *mashup* o página web híbrida.

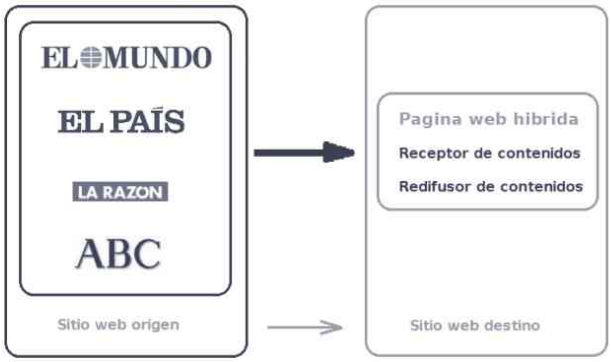


Figura 9.14. *Diagrama de redifusión de contenidos*

Vamos a seguir analizando el planteamiento del comienzo del párrafo, en el decíamos que las grandes estructuras web, actualizan sus contenidos muy rápidamente y además son fuentes muy creíbles. Por otro lado existen portales que ofrecen servicios web en Internet, pero que no tienen capacidad para actualizar con rapidez sus contenidos. Estos portales podrían incluir contenidos de las secciones de otras páginas web, con previo consentimiento, sin tener que preocuparse de actualizar los datos.

Conviene saber que cuando hablamos de redifusión, existen diferentes tipos que no están relacionados con la Web. Un tipo de redifusión puede ser la televisiva, la de prensa o de radio. No vamos a profundizar en estos tipos de redifusión, pero si es aconsejable enmarcar la redifusión en el ámbito web para no generar confusión. Por esta razón existe el término ampliamente utilizado para referirse a la redifusión web, que precisamente esta en el nombre del título de este punto. El término, que ya hemos mencionado es, sindicación web.

9.4.1 FUENTE WEB O CANAL WEB

La redifusión o sindicación, como hemos indicado antes, trata la distribución de contenidos de un portal de Internet. De esta forma es posible seleccionar parte de información de una página web y ponerla a disposición de otros portales o clientes finales. Para ofrecer el contenido, debemos hacer uso de una fuente web, que provee de estos datos a la página de destino.

Según la RAE una fuente es el principio, fundamento u origen de algo, en su acepción 8. Teniendo en cuenta este significado, podemos hacernos una idea de que es una fuente web.

Una fuente web o canal web, en inglés, *web feed* ó *news feed*, es el medio utilizado para la redifusión de contenidos web. Para visualizar una fuente web podemos suscribirnos al servicio de la página web que ofrece los contenidos. Otra opción es utilizar un programa llamado agregador que permite visualizar las fuentes web.

Los dos tipos más extendidos de fuentes web en la actualidad son RSS y Atom. Estos formatos como vimos en el primer punto del capítulo, utilizan el lenguaje de marcado XML para su construcción.

Cuando los portales web, comenzaron a facilitar fuentes web a sus usuarios, las llamaron con el término anglosajón, *web feed*. En la actualidad, son ya muchos los portales que utilizan el término de fuente web, fuente RRS, o fuente Atom.

Conviene saber que no es lo mismo una fuente web que RSS. En la actualidad se utiliza RSS para referirse a una fuente web, esto no es correcto. Una fuente web hace mención al medio de redifusión web. RSS en cambio se refiere al formato de la fuente web. Esta confusión de significados, origina en que el único formato de fuente web era RSS, por tanto, se usaba indistintamente. En cambio actualmente existe otro formato de fuente web muy popular, que es Atom.

Resumiendo, una fuente web es el medio de redifusión de contenido web. RSS y Atom son el formato que adopta dicha fuente web.

Las fuentes web tienen una serie de campos que contienen la información necesaria para poder visualizar el contenido. La separación en campos permite que esta información se visualice con cierta división y orden. Los items que suelen tener las fuentes web son: el título, un resumen del contenido, un enlace para ampliar la información y una fecha de publicación. Como vimos en la primera parte del capítulo, RSS tiene una estructura de fuente más simple que Atom. En los siguientes apartados profundizaremos en la estructura de cada uno de estos formatos.

9.4.2 BENEFICIOS DE LA SINDICACIÓN O REDIFUSIÓN WEB

Al comienzo de este apartado, hemos visto algunas de las ventajas que nos proporciona la sindicación o redifusión web. A continuación vamos a detallar y repasar estos beneficios, para comprender y evaluar la utilidad que puede presentar el uso de las siguientes tecnologías que vamos a mencionar.

Los portales de las grandes corporaciones actualizan sus contenidos frecuentemente. Como los usuarios no podemos estar visualizando y evaluando constantemente estos cambios. A través de las fuentes web, podemos visualizar los cambios de los canales de interés. Además al poder unificar la información no tenemos que buscar la información por diferentes páginas web.

El ahorro de tiempo es considerable, al disponer de los contenidos de una forma fácil y rápida. Por ejemplo podemos buscar los titulares de varios portales de noticias unificándolo todo en un solo sitio.

Se evita el uso de direcciones de correo electrónico, puesto que aquí los datos viajan a través del protocolo HTTP. De esta forma se evita el envío de publicidad, *spam*, etc.

El control de cancelación o inclusión de fuentes, va a ser ejecutado por el usuario. De esta forma un usuario puede modificar las fuentes que desea visualizar sin necesidad de realizar ninguna petición.

9.4.3 UTILIZACIÓN DE UNA FUENTE WEB O CANAL WEB

Un fichero de una fuente web, es un documento XML que puede ser abierto directamente por un navegador. El formato de este fichero está basado en etiquetas. Algunas versiones actuales, reconocen el formato de las fuentes y las visualizan directamente, supliendo las etiquetas que lo agrupan por un orden y presentación establecidos.

No obstante lo más recomendable es utilizar un programa que lea estos documentos XML (fuente web), y los analice, mostrando el resultado en el propio programa. A estos programas se los llama agregadores. En la actualidad existen dos tipos de agregadores principalmente. Los que disponen de un acceso en línea, basta logarse para acceder a ellos. Los que requiere de la instalación de un software en local.

Un agregador es una aplicación web o local que permite la suscripción a diferentes fuentes web, principalmente de los dos principales formatos RSS y Atom, aunque existen algunos derivados de estos. Por lo tanto, un agregador permite reunir datos que se publican en portales que permiten la redifusión web. La aplicación se encarga de actualizar los vínculos e indicar cuándo se ha añadido nueva información en el programa. Algunos de estos agregadores, permiten filtrar u ordenar los contenidos por palabras clave.

Algunos agregadores en la Red son, IGoogle, Netvibes, Blogvibes, My Yahoo o Google Reader.

Entre los agregadores locales más extendidos están Liferea o Naufrago (para Linux), NetNewsWire o FeedReader.

Figura 9.15. *Agregador NetNewsWire para Mac*

En la imagen anterior, vemos el aspecto del agregador NetNewsWire. Esta aplicación es un agregador de noticias compatible con los sistemas operativos Mac OS X y iOS de Apple.

9.4.4 EL FORMATO RSS

Como dijimos al principio RSS son las siglas de *Really Simple Syndication*. El lenguaje que utiliza RSS para formar sus documentos es XML. Como hemos visto a lo largo de todo el capítulo, XML es el lenguaje por antonomasia que permite intermediar en la comunicación de casi todas las plataformas y sistemas existentes en el mercado. Actualmente la última versión del formato RSS es la 2.0. En el siguiente punto vamos a pormenorizar cuales son los componentes de esta versión del formato.

Las primeras etiquetas que conforman un documento con formato RSS son la cabecera del documento XML con su `versión` y el `encoding`. Posteriormente indicaremos la etiqueta que agrupa a todos los elementos contenidos en un documentos RSS, esta etiqueta es `<rss>...</rss>`. Esta etiqueta recibe el atributo de la `versión` (en nuestro caso será 2.0) y el `xmlns`, indicando el espacio de nombres. Otra etiqueta que forma parte del bloque principal es `<chanel>...</chanel>`. Dentro de esta etiqueta se incluyen todos los items. Según esta descripción la estructura principal de un documento RSS queda de la siguiente manera:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0"
  xmlns:media="http://search.yahoo.com/mrss/">

  <channel>
    ...items...
  </channel>
</rss>
```

En el código anterior vemos como la etiqueta `rss`, agrupa todo el contenido del documento. Dentro de esta etiqueta se encuentra `chanel` que también va a recoger al resto de etiquetas que vamos a describir a continuación.

Una vez que hemos definido el cuerpo principal del documento RSS, debemos crear noticias asociadas a este canal. Cada noticia se deberá agrupar con las etiquetas `<item></item>`. Dentro de esta etiqueta tendremos un subgrupo de etiquetas que identificaran y estructurarán la noticia. Estas etiquetas son: `title`, indica el título de la noticia, **`link`**, enlaza con el detalle de la noticia, `description`, que describe una pequeña reseña de la noticia, `author`, define al autor de la noticia y `pubDate`, que define la fecha de publicación de la noticia. Aunque existen más etiquetas, estas son las principales que conforman un documento RSS.

```
<item>
  <title><![CDATA[Nieve para el fin de semana]]></title>
  <link><![CDATA[http://tiempo.com/0001.html]]></link>
  <description><![CDATA[Ya no nieva tanto]]></description>
  <author><![CDATA[Javier]]></author>
  <pubDate><![CDATA[30/01/2012]]></pubDate>
</item>
```

Observamos como los campos de las etiquetas se incluyen en `CDATA`, puesto que estos son datos, según la normativa de XML de tipificación de tipos de datos.

Por lo tanto, un documento RSS completo de ejemplo quedaría de la siguiente manera. La estructura `<item>...</item>` se repetirá tantas veces como sea necesario a lo largo del documento de forma que si se repite 5 veces, existirán 5 noticias para visualizar en el canal del documento.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0" xmlns:media="http://dom.pagina.com/rss/">
  <channel>
    <title><![CDATA[TITULO - Portada]]></title>
    <link><![CDATA[http://www.pagina.com]]></link>
    <description>
      <![CDATA[Descripción título - Portada]]>
    </description>
    <lastBuildDate>
      Mon, 30 Jan 2012 16:14:07 +0100
    </lastBuildDate>
    <language>es-es</language>
    <copyright><![CDATA[Copyright - Derechos]]></copyright>
    <ttl>0</ttl>

    <item>
      <title><![CDATA[Titulo de la noticia]]></title>
      <link><![CDATA[http://domin.com/not/0001.html]]></link>
      <description>
        <![CDATA[Texto descripción]]>
      </description>
      <author><![CDATA[Nombre del autor]]></author>
      <pubDate><![CDATA[Fecha de la publicación]]></pubDate>
    </item>
  </channel>
</rss>
```

Este documento se debe generar en el servidor. Tendrá una extensión `.rss` y tendrá asociada una ruta con dirección pública para que los clientes que pretendan tener acceso al documento lo hagan a través de ella. Una forma de facilitar a través de otra página la URL de la fuente web sería la siguiente:

```
<a type="application/rss" href="nombrefichero.rss">
  Noticias del canal anterior
</a>
```

Si colocamos este enlace en la interfaz que provee de los enlaces al cliente de la fuente web, podremos copiar y pegar el enlace o pinchar en él para asociar la suscripción con el software que tengamos instalado en el ordenador.

9.4.5 EL FORMATO ATOM

Atom al igual que RSS, es una forma simple de leer y escribir información de una web con una forma estructurada y organizada, en otro lugar elegido. Permite mantener el seguimiento de más de un sitio web, simplificando la pérdida de tiempo y facilitando el acceso a las zonas que se desean agrupar.

El tipo de estructura Atom tiene una serie de etiquetas para formar los documentos que servirán de alimento a las aplicaciones o webs que los reciban. El elemento principal es la etiqueta `<feed>` que contiene un atributo `xmlns` como ocurría en el caso de RSS.

Dentro del la etiqueta principal contiene, etiquetas de tipo `id`, que identifica la información, `título`, que da título a la fuente, `actualizado`, indica la última vez que fue modificada la fuente. Estos elementos son necesarios para la generación de una estructura Atom.

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<feed xmlns="http://www.w3.org/201/Atom">
  <id> http://ejemplo .com/ </id>
  <title> Titulo </title>
  <updated> 20 de enero de 2011 </updated>
 </Feed>
```

Además de las etiquetas obligatorias, existen una serie de etiquetas que se pueden añadir al final. Estas etiquetas deberán ir dentro de otra etiqueta llamada `<entry>`. A continuación vemos cuales son; `autor`, indicaremos el nombre del autor, `enlace`, para mostrar una página web relacionada, `categoría`, especifica una categoría de la información, `contribuyente`, indica algún colaborador de la información, `generador`, software utilizado, `icono`, una pequeña imagen, debe ser cuadrada, `logo`, una imagen más grande que el icono, `derechos`, indica los derechos de la información, `subtitulo`, una descripción. El código correspondiente a estos parámetros quedaría de la siguiente manera:

```
<entry>
  <author>Javier</author>
  <link type="text/html" href="http://ejemplo.org/"/>
  <category term="sports"/>
  <contributor>
    <name>Madona</name>
  </contributor>
  <generator uri="/myPagina.asp" version="3.0">
    Generado por ASP
  </Generator>
  <icon> /icono.jpg </icon>
  <logo> /logo.jpg </logo>
  <rights> © 2012 MAdrid </rights>
  <subtitle> Un comentario </ subtitle>
</entry>
```

La estructura de Atom, es más compleja que la de RSS. Quizás, por esa razón, se ha extendido más RSS. No obstante la potencia de Atom es mayor y permite configurar y detallar más las noticias.

Las características de los tipos de documento Atom son más amplias. Algunas de las etiqueta mostradas admiten atributos, al igual que ocurre en las etiquetas HTML.

Conviene saber que si se abusa de subscribirse a excesivas noticias y contenidos por el hecho de que estos sean de fácil y automático acceso, puede que sea tan densa la información que sea difícil digerirla diariamente. Por esta razón es recomendable aconsejar a los usuarios de fuentes web, que moderen la inclusión de subscripciones en sus agregadores con el fin recoger en ellos la información que de verdad queremos visualizar en ellos.

ACTIVIDADES 9.1

- Realice un resumen de los puntos del capítulo haciendo una pequeña descripción de cada uno de los puntos anteriores.
- Busque agregadores en la Red e instale algunos de ellos. Instale también algún agregador para el sistema operativo Linux.
- Realice un listado de páginas web que utilicen *mashups* y asócielas a cada una de las categorías.
- Desarrolle un documento de descubrimiento UDDI e inclúyalo en un fichero para que éste pueda ser accedido desde un servidor de aplicaciones.
- Cree dos fuentes web equivalentes en RSS y Atom. Compare en qué se diferencian cada una de ellas.

RESUMEN DEL CAPÍTULO

Una aplicación web híbrida es una página web que utiliza contenidos o servicios de terceros y los combina para crear una aplicación nueva.

La arquitectura de una aplicación web híbrida está compuesta por tres partes principales, el *proveedor de contenidos*, *el servidor del sitio web* y *el cliente web*.

Existe proveedores de contenidos para los mashups, entre ellos están RSS, Atom, *Screen scraping* (aunque no es recomendable su uso) y el uso de API.

La división en categorías está compuesta por los mashups de mapas, de foto y vídeos, de compras y de noticias.

La arquitectura de un repositorio UDDI está formada por el servicio de directorio, el descubrimiento de un servicio, la descripción de un servicio y el formato de conexión.

Existen muchas API, entre ellas para manejar e incluir fotos, vídeos, mapas, etc. Todas estas son muy utilizadas debido a la proliferación de portales que ofrecen servicios con este tipo de contenidos.

Una fuente web, o canal web, es el medio utilizado para la redifusión de contenidos web. Los agregadores son programas que agrupan estas fuentes web en un solo sitio. RSS y Atom son dos formatos de fuente web existentes.

La inclusión de excesivas suscripciones en un agregador dificulta la facilidad de lectura de los contenidos, minimizando los efectos beneficiosos de unificar los contenidos en un solo lugar.

EJERCICIOS PROPUESTOS

1. Cree una página web en la que se acceda a un mapa de GooleMaps. Sitúe el mapa en un punto e incluya algunos de los elementos del mapa, como los enlaces para acercar el mapa, mover el mapa con las flechas de desplazamiento, etc. Utilizando la documentación que ofrece Google, estudie cuál sería la forma de incluir un icono dentro del mapa. Pruebe a implementarlo.

TEST DE CONOCIMIENTOS

1 La arquitectura de una aplicación híbrida está formada por:
- a) Proveedor de contenido, el servidor del sitio web *mashup* y el cliente web.
- b) El proveedor de contenidos y el *mashup*.
- c) El *mashup* y el cliente.
- d) El servidor, el *mashup* y el cliente.

2 RSS y Atom son:
- a) Dos fuentes web.
- b) Dos proveedores de contenidos.
- c) Un tipo de estructura de fuente web.
- d) Todas las respuestas anteriores son ciertas.

3 *Screen scraping* es:
- a) Un método que no es aconsejable utilizar.
- b) Un método con el que no hace falta que el proveedor haga nada.
- c) En algunos casos puede ser la única opción para recuperar contenidos.
- d) Todas las respuestas anteriores son ciertas.

4 El navegador web cliente es:
- a) Es la interfaz del usuario del *mashup*.
- b) Es una aplicación web.
- c) Las respuestas *a* y *b* son ciertas.
- d) Solo la pregunta *a* es cierta.

5 Las divisiones por categorías de los *mashups* son:
- a) Mapas, fotos, compras y noticias.
- b) Mapas, fotos y vídeos, compras y noticias.
- c) Mapas, fotos y vídeos, ventas y noticias.
- d) Mapas, fotos, ventas y noticias.

6 El descubrimiento de un servicio se encapsula en:
- a) El fichero WSDL.
- b) En un XML que contiene vínculos a otros recursos.
- c) En un archivo con extensión `.ico`.
- d) En un `xmlns`.

7 Una API:
- **a)** Crea una capa de abstracción entre el cliente y el proveedor.
- **b)** Permite enviar una solicitud y recibir una respuesta.
- **c)** Exige que esté instalada en un equipo local.
- **d)** Las respuesta *a* y *b* son ciertas.

8 La función `GMap2` de GoolgeMaps permite:
- **a)** Localizar un mapa en la Web.
- **a)** Generar un único mapa en una página.
- **b)** Generar varios mapas a la vez.
- **c)** Llamar a un mapa de tipo 2.

9 Una fuente web es:
- **a)** El medio utilizado por RSS o Atom.
- **a)** El medio utilizado por RSS.
- **b)** El medio utilizado por Atom.
- **c)** Ninguna de las anteriores.

10 La forma de leer fuentes web es:
- **a)** Utilizar un agregador.
- **b)** Utilizar un agregador o un navegador.
- **c)** Utilizar un agregador o una versión actual de navegador.
- **d)** Todas las respuestas anteriores son ciertas.

Índice Alfabético

La presente obra está dirigida a los estudiantes del Ciclo Formativo **Desarrollo de Aplicaciones Web** de Grado Superior, en concreto para el módulo profesional **Desarrollo Web en Entorno Servidor**.

Los contenidos incluidos en este libro abarcan los conceptos básicos y las técnicas habituales para el desarrollo de aplicaciones web que serán ejecutadas en un servidor web. Además, se presentan acompañados de ejemplos intuitivos que sirven para ilustrar dichos conceptos y técnicas. Como punto de partida se introducen brevemente los tipos y arquitecturas de servidores web, además de presentar las diferentes alternativas tecnológicas que un desarrollador web tiene a su disposición a la hora de crear soluciones informáticas en el entorno del servidor. Se abordan los puntos principales relacionados con el uso de lenguajes que intercalan su código con el de las páginas web (PHP, ASP, JSP, etc.), ofreciendo una descripción detallada de su sintaxis y de las estructuras y funciones primordiales. Se estudia el desarrollo de aplicaciones web dinámicas. Se hace un recorrido por los diferentes mecanismos de separación de la lógica de negocio y de generación dinámica de las interfaces web que se envían al cliente, detallando el soporte de aspectos tales como la seguridad, la gestión y mantenimiento del estado como parte de la interacción con el cliente o la implementación de técnicas avanzadas de control de usuarios. También se presta una atención especial a los aspectos de conexión y acceso a fuentes de datos desde las aplicaciones web del servidor, entre otros temas.

Todos los capítulos incluyen actividades y ejemplos con el propósito de facilitar la asimilación de los conocimientos tratados. Así mismo, se incorporan test de conocimientos y ejercicios propuestos con la finalidad de comprobar que los objetivos de cada capítulo se han asimilado correctamente.

Además, reúne los recursos necesarios para incrementar la didáctica del libro, tales como un glosario con los términos informáticos necesarios, bibliografía y documentos para ampliación de los conocimientos.

En la página web de **Ra-Ma** (**www.ra-ma.es**) se encuentra disponible el material de apoyo y complementario.

ra-ma.es

Ra-Ma®