Interfaces web

En este capítulo aprenderemos en qué consisten las aplicaciones web y mencionaremos algunas de las herramientas que los programadores tenemos a nuestra disposición para construir este tipo de aplicaciones. Para ser más específicos, en las siguientes secciones trataremos los temas que aparecen a continuación:

- En primer lugar, comenzaremos presentando la evolución histórica de las aplicaciones web para comprender cómo se ha llegado a su arquitectura actual desde los modestos comienzos de las páginas web.

- A continuación, pasaremos a describir las principales alternativas de las que dispone el programador para construir sus aplicaciones web.

- Finalmente, comentaremos las soluciones que oferta Microsoft para el desarrollo de interfaces web y empezaremos a ver cómo se construyen aplicaciones web en la plataforma .NET utilizando páginas ASP.NET.

Interfaces web

Evolución de las aplicaciones web 7
HTML estático 7
Aplicaciones web 9
Servicios web 11

Desarrollo de aplicaciones para Internet 13
En el cliente 13
HTML dinámico y JavaScript 14
Controles ActiveX 16
Applets 17
Plug-ins específicos 17
En el servidor 18
Aplicaciones web compiladas: CGI 19
Servlets 20
Aplicaciones web interpretadas: CGI scripts & Scripting languages 21
Páginas de servidor: ASP y JSP 22

ASP: Active Server Pages 24

ASP.NET: Aplicaciones web en la plataforma .NET 29
Un ejemplo 29
Dos estilos 32

Apéndice: Aprenda HTML en unos minutos 35

Evolución de las aplicaciones web

Como dijimos en la introducción a esta parte del libro, las aplicaciones web son aquellas cuya interfaz se construye utilizando páginas web. Dichas páginas son documentos de texto a los que se les añaden etiquetas que nos permiten visualizar el texto de distintas formas y establecer enlaces entre una página y otra.

La capacidad de enlazar un texto con otro para crear un hipertexto es la característica más destacable de las páginas web. Aunque su éxito es relativamente reciente, sus orígenes se remontan al sistema Memex ideado por Vannevar Bush (*"As we may think"*, Atlantic Monthly, julio de 1945). El término hipertexto lo acuñó Ted Nelson en 1965 para hacer referencia a una colección de documentos (nodos) con referencias cruzadas (enlaces), la cual podría explorarse con la ayuda de un programa interactivo (navegador) que nos permitiese movernos fácilmente de un documento a otro.

De hecho, la versión que conocemos actualmente del hipertexto proviene del interés de los científicos en compartir sus documentos y hacer referencias a otros documentos. Este interés propició la creación de la "tela de araña mundial" (*World-Wide Web*, WWW) en el Centro Europeo para la Investigación Nuclear (CERN). Tim Berners-Lee, uno de los científicos que trabajaba allí, ideó el formato HTML para representar documentos con enlaces a otros documentos. Dicho formato fue posteriormente establecido como estándar por el W3C (*World-Wide Web Consortium*, http://www.w3c.org/), el organismo creado por el MIT que fija los estándares utilizados en la web desde 1994.

HTML estático

Inicialmente, las páginas web se limitaban a contener documentos almacenados en formato HTML [*HyperText Markup Language*]. Dichos documentos no son más que ficheros de texto a los que se le añaden una serie de etiquetas. Dichas etiquetas delimitan fragmentos del texto que han de aparecer en un formato determinado y también sirven para crear enlaces de un documento a otro (o, incluso, de una parte de un documento a otra parte del mismo documento). Al final de este capítulo puede encontrar una pequeña introducción al formato HTML para refrescar sus conocimientos o aprender a escribir sus propias páginas web.

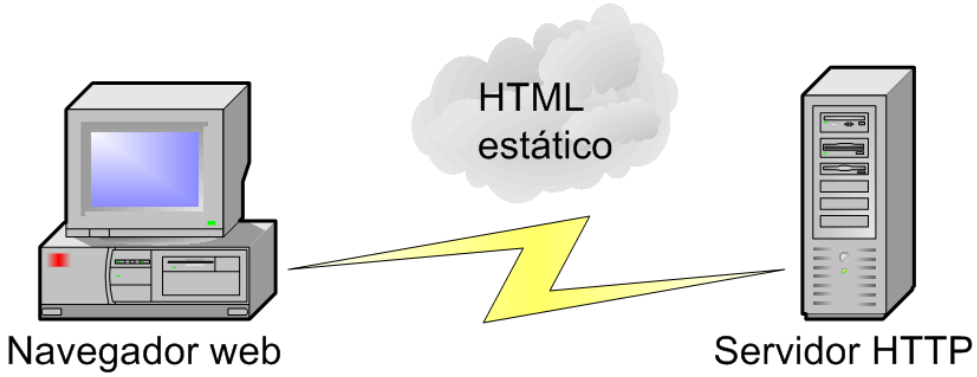
Con unos conocimientos mínimos de HTML, crear un sitio web resulta relativamente sencillo. Sólo hay que preparar los documentos HTML tal y como queramos que los visualicen los visitantes de nuestra página. Cuando podemos predecir con antelación cuál es la información que tenemos que mostrarle al usuario, crear una página web estática resulta la opción más sencilla. Incluso cuando el contenido de nuestra página web ha de cambiar periódicamente, en ocasiones es suficiente con escribir pequeños programas que generen los documentos HTML

a los que accederá el visitante de nuestra página. Este es el caso, por ejemplo, de las páginas web utilizadas por medios de comunicación como periódicos. Cada día, o incluso más a menudo, una aplicación se encarga de generar los documentos HTML con el formato visual más adecuado para nuestro sitio web. Dichos documentos HTML quedan almacenados de forma permanente en ficheros y el usuario accede a ellos directamente.

En realidad, el usuario no accede directamente a los ficheros que contienen los documentos HTML, sino que utiliza un navegador para visualizarlos cómodamente. Dicho navegador es, en realidad, una aplicación cliente que utiliza el protocolo HTTP [*HyperText Transfer Protocol*] para acceder a la máquina en la que hayamos alojado nuestros ficheros en formato HTML. Por tanto, para que los usuarios puedan acceder a nuestra página web, sólo necesitaremos un servidor web que atienda las peticiones HTTP generadas por el navegador web del usuario. En respuesta a esas peticiones, el servidor HTTP que utilicemos le enviará al navegador los documentos que haya solicitado. Cuando nuestro servidor se limita a servir documentos HTML previamente preparados, podemos utilizar cualquier servidor HTTP de los existentes, como el Internet Information Server de Microsoft o el Apache (el cual se puede descargar gratuitamente de http://www.apache.org/).

Aparte del servidor HTTP que hemos de tener funcionando en la máquina que sirva los documentos HTML, nos hará falta disponer de una dirección IP fija para la máquina donde alojemos el servidor HTTP. Dicha dirección resulta imprescindible para que el usuario pueda saber dónde ha de conectarse para obtener los documentos que le interesan. De hecho, la mayoría de los usuario ni siquiera son conscientes de que cada vez que acceden a un sitio web están utilizando una dirección IP a modo de "número de teléfono", sino que escriben en la barra de direcciones de su navegador una URL [*Uniform Resource Locator*] que identifica unívocamente el recurso en Internet al que desea acceder y suele incluir un nombre de dominio. Dicho nombre de dominio es una cadena de texto fácil de recordar que el servicio de nombres DNS [*Domain Name Service*] traduce a la dirección IP necesaria para establecer con éxito una conexión con el servidor. En consecuencia, resulta aconsejable (y no demasiado caro) reservar un nombre de dominio que asociaremos a la dirección IP de la máquina donde instalemos nuestro servidor HTTP. Sólo así podrá el usuario escribir una URL de la forma http://csharp.ikor.org/ para acceder cómodamente a los datos que nosotros previamente habremos dejado a su disposición en nuestro servidor web.

Si bien esta forma de construir un sitio web puede ser suficiente para muchas aplicaciones, en ocasiones necesitaremos que el contenido de nuestra página web se genere dinámicamente en función de las necesidades y deseos de nuestros usuarios. Cuando nos interesa algo más que mostrar siempre los mismos datos de la misma forma, tener que actualizar periódicamente los ficheros HTML no siempre es una buena idea. El inconveniente de utilizar simples ficheros HTML es que estos ficheros son estáticos y, mientras no los actualicemos de forma manual o automática, mostrarán siempre la misma información independientemente de quién acceda a nuestra página. Por ejemplo, si lo que queremos es construir una página web cuyo contenido cambie en función del usuario la visite y de la tarea que desee realizar, la solución pasa ineludiblemente por generar dinámicamente los documentos HTML cada vez que le llega una solicitud a nuestro servidor HTTP. Éste es el principio de funcionamiento de las aplicaciones web que describiremos en el siguiente apartado.



HTML estático: Configuración típica de una "aplicación web" que se limita a ofrecer la información almacenada en páginas HTML a las que el usuario final accede desde su navegador web utilizando el protocolo HTTP.

Aplicaciones web

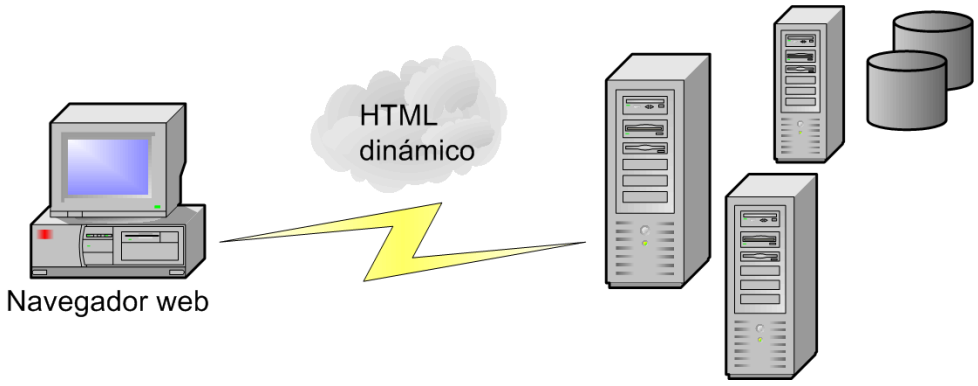
Aunque la utilización de documentos HTML estáticos puede ser la solución más adecuada cuando nuestra página web se limite a ofrecer siempre la misma información o podamos automatizar la realización de actualizaciones de los documentos HTML que la constituyen, la naturaleza dinámica de la web y las expectativas que ha creado en la actualidad hacen necesaria la implementación de aplicaciones web que generen dinámicamente el contenido que finalmente se les ofrece a los usuarios. De esta forma podemos seleccionar, filtrar, ordenar y presentar la información de la forma más adecuada en función de las necesidades de cada momento. Si bien esto se podría conseguir con páginas HTML estáticas si dispusiésemos de espacio suficiente en disco (y, de hecho, esta es una estrategia que se utiliza para disminuir la carga de la CPU de los servidores), las aplicaciones web nos permiten ofrecer la información más actual de la que disponemos al poder acceder directamente a las bases de datos que contienen los datos operativos de una empresa.

La creación de aplicaciones web, en consecuencia, requiere la existencia de software ejecutándose en el servidor que genere automáticamente los ficheros HTML que se visualizan en el navegador del usuario. Exactamente igual que cuando utilizábamos páginas estáticas en formato HTML, la comunicación entre el cliente y el servidor se sigue realizando a través del protocolo HTTP. La única diferencia consiste en que, ahora, el servidor HTTP delega en otros módulos la generación dinámica de las páginas HTML que se envían al cliente. Ya que, desde el punto de vista del cliente, la conexión se realiza de la misma forma y él sigue recibiendo páginas HTML estándar (aunque éstas hayan sido generadas dinámicamente en el servidor), el navegador del cliente es independiente de la tecnología que se utilice en el servidor para generar dichas páginas de forma dinámica.

Desde el punto de vista del programador, existe una amplia gama de herramientas a su disposición. Para generar dinámicamente el contenido que se le ofrece al usuario, puede optar por desarrollar software que se ejecute en el servidor o, incluso, en la propia máquina del usuario. Algunas de las opciones entre las que puede elegir el programador serán comentadas en las siguientes secciones y una de ellas será estudiada con mayor detalle en éste y los siguientes capítulos: las páginas ASP.NET incluidas en la plataforma .NET.

Básicamente, las distintas alternativas disponibles para el desarrollo de aplicaciones web ofrecen la misma funcionalidad. No obstante, en función de las necesidades de cada proyecto y de su envergadura algunas resultarán más adecuadas que otras. Igual que en cualquier otro aspecto relacionado con el desarrollo de software, no existen "balas de plata" y cada tecnología ofrece una serie de facilidades que habremos de estudiar en función de lo que tengamos que hacer. Por ejemplo, el protocolo HTTP es un protocolo simple en el que se establece una conexión TCP independiente para cada solicitud del cliente. Esto es, cada vez que el usuario accede a un fichero de nuestro servidor (o, lo que es lo mismo, a una página generada dinámicamente), lo hace de forma independiente. Por tanto, la herramienta que utilicemos para crear nuestra aplicación web debería facilitarnos el mantenimiento de sesiones de usuario (conjuntos de conexiones independientes relacionadas desde el punto de vista lógico).

En resumen, independientemente de la forma en que implementemos nuestra aplicación web, el navegador del cliente es independiente de la tecnología que se utilice en el servidor, ya que a él sólo le llegará una página HTML estándar que mostrará tal cual. En la siguiente sección repasaremos las formas más comunes de desarrollar aplicaciones web. Usualmente, las páginas web que se le muestran al usuario se generan dinámicamente en el servidor, si bien también se puede introducir cierto comportamiento dinámico en el navegador del cliente a costa de perder parte de la independencia entre el navegador y nuestra aplicación web.



Aplicaciones web: El contenido que se le muestra al usuario se genera dinámicamente para cada solicitud proveniente del navegador web instalado en la máquina cliente.

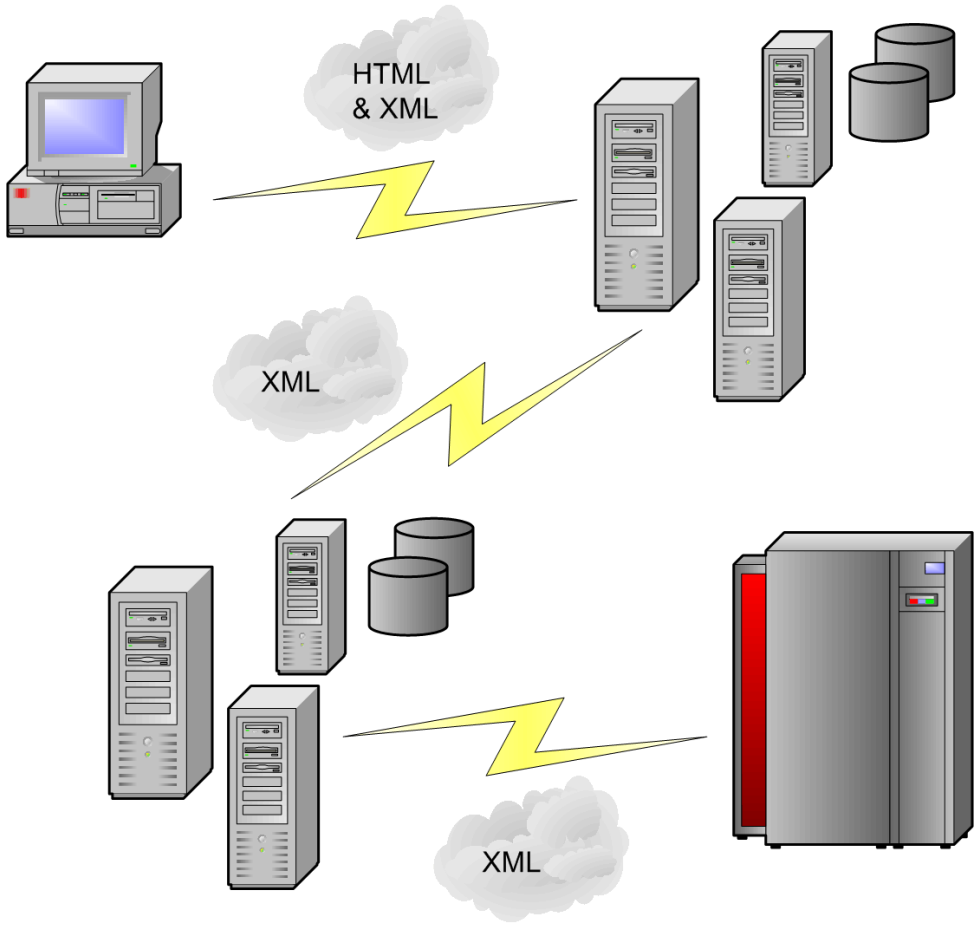
Antes de analizar las distintas herramientas y tecnologías que nos facilitan el desarrollo de aplicaciones web, para terminar de repasar la evolución de las páginas web desde sus comienzos estáticos hasta la actualidad, en el siguiente apartado comentaremos brevemente cuál es la configuración que suelen tener las aplicaciones web más recientes.

Servicios web

Las aplicaciones web han sufrido una evolución análoga a la que ya padecieron las aplicaciones de escritorio que utilizan los recursos propios de cada sistema operativo para construir su interfaz de usuario. Inicialmente, estas aplicaciones se ejecutaban en una única máquina, que era además la máquina donde se almacenaban los datos que manipulaban. Posteriormente, se hicieron populares las arquitecturas cliente/servidor, en las que la interfaz de usuario de las aplicaciones de gestión se ejecuta en la máquina del cliente pero los datos se suelen almacenar en un sistema gestor de bases de datos. La aplicación cliente se conecta al sistema gestor de bases de datos de forma similar a como el navegador web accede al servidor HTTP en una aplicación web como las descritas en el apartado anterior. Finalmente, para determinadas aplicaciones de gestión se han impuesto las arquitecturas multicapa y el uso de *middleware* (por ejemplo, CORBA). En estas aplicaciones, la máquina cliente sólo ejecuta la interfaz de usuario y la lógica de la aplicación se ejecuta en un servidor de aplicaciones independiente tanto de la interfaz de usuario como de la base de datos donde se almacenan los datos.

Las aplicaciones web sólo se distinguen de las aplicaciones de escritorio tradicionales en que, en vez de implementar la interfaz de usuario utilizando un lenguaje particular como C/C++ o Java, se utilizan páginas web como punto de acceso a las aplicaciones. Por consiguiente, no es de extrañar que también se construyan aplicaciones web multicapa. Dichas aplicaciones construyen su interfaz utilizando formularios HTML, implementan su lógica en sistemas distribuidos y suelen almacenar sus datos en sistemas gestores de bases de datos relacionales.

De hecho, en el caso de las aplicaciones web incluso se han propuesto estándares que utilizan los mismos protocolos que las aplicaciones web cliente/servidor como canal de comunicación entre las distintas partes de una aplicación distribuida. Este es el caso de los servicios web, que intercambian mensajes en formato XML utilizando protocolos de transporte como HTTP. Los servicios web, básicamente, establecen un lenguaje común mediante el cual distintos sistemas puedan comunicarse entre sí y, de esta forma, facilitan la construcción de sistemas distribuidos heterogéneos. Dada su importancia actual, les dedicaremos a ellos un capítulo completo en la siguiente parte de este libro. Por ahora, nos centraremos en la construcción de interfaces web. Más adelante ya veremos cómo encajan todas las piezas en la construcción de arquitecturas software.



Servicios web: La lógica de la aplicación se distribuye. El intercambio de mensajes en formato XML y el uso de protocolos estándares de Internet nos permiten mantener conectadas las distintas partes de una aplicación, aunque ésta haya de funcionar en un sistema distribuido heterogéneo.

Desarrollo de aplicaciones para Internet

Como hemos ido viendo en las páginas anteriores, actualmente se observa una tendencia muy definida que fomenta la utilización los estándares de Internet para desarrollar aplicaciones de gestión en medianas y grandes empresas. Si centramos nuestra atención en el desarrollo del interfaz de usuario de estas aplicaciones, lo que encontramos es un uso extensivo de los estándares abiertos utilizados en la web, aquéllos promovidos por el W3C, si bien es cierto que también se utilizan algunas tecnologías propietarias.

La característica común que comparten todas las aplicaciones web es el hecho de centralizar el software para facilitar las tareas de mantenimiento y actualización de grandes sistemas. Es decir, se evita tener copias de nuestras aplicaciones en todos los puestos de trabajo, lo que puede llegar a convertir en una pesadilla a la hora de distribuir actualizaciones y garantizar que todos los puestos de trabajo funcionen correctamente. Cada vez que un usuario desea acceder a la aplicación web, éste se conecta a un servidor donde se aloja la aplicación. De esta forma, la actualización de una aplicación es prácticamente trivial. Simplemente se reemplaza la versión antigua por la versión nueva en el servidor. A partir de ese momento, todo el mundo utiliza la versión más reciente de la aplicación sin tener que realizar más esfuerzo que el de adaptarse a los cambios que se hayan podido producir en su interfaz.

Aunque todas las aplicaciones web se diseñen con la misma filosofía, existen numerosas alternativas a la hora de implementarlas en la práctica. A grandes rasgos, podemos diferenciar dos grandes grupos de aplicaciones web en función de si la lógica de la aplicación se ejecuta en el cliente o en el servidor. Esta distinción nos permitirá, en los dos próximos apartados, organizar un recorrido sobre las tecnologías existentes para el desarrollo de interfaces web.

> En realidad, casi todas las aplicaciones web reales utilizan tecnologías tanto del lado del cliente como del lado del servidor. Utilizar unas u otras en una cuestión de diseño que habrá de resolverse en función de lo que resulte más adecuado para satisfacer las necesidades particulares de cada aplicación.

En el cliente

En principio, todo el software asociado a una aplicación web se puede desarrollar de forma que el trabajo lo realice el servidor y el navegador instalado en la máquina cliente se limite a mostrar páginas HTML generadas en el servidor. De esta forma, al usuario de la aplicación web le basta con tener instalado cualquier navegador web. Esta estrategia es la que resulta

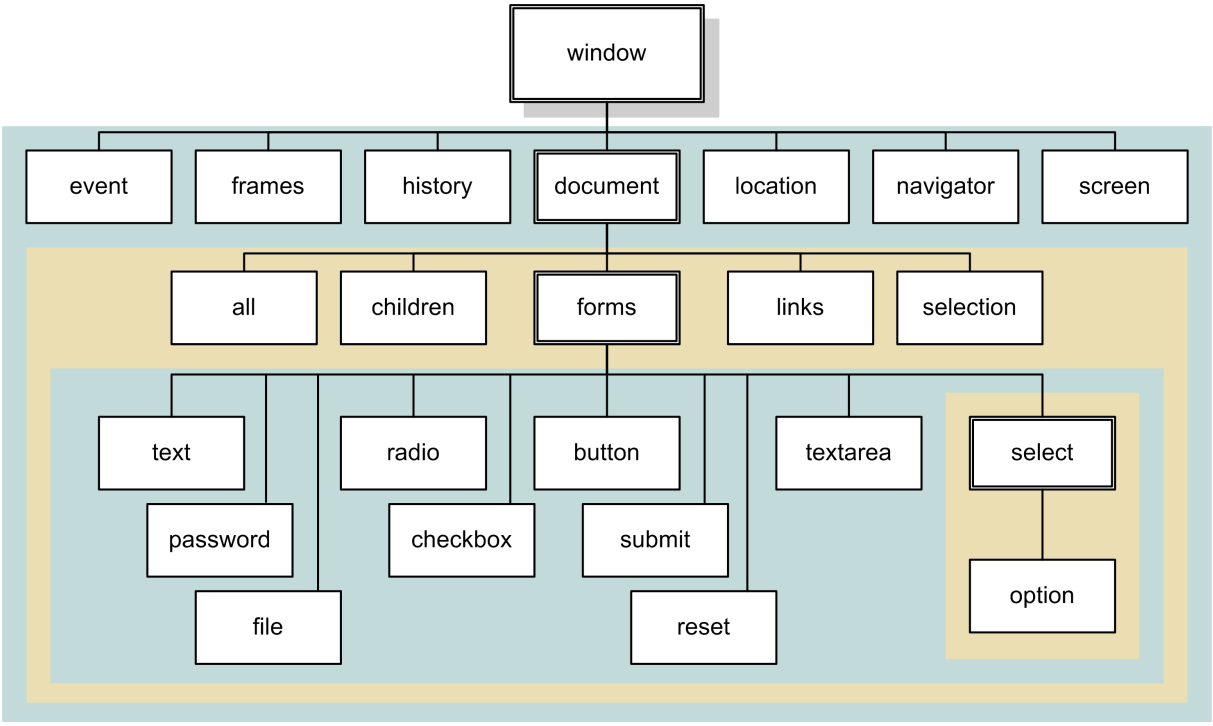
más cómoda para el programador. Sin embargo, desde el punto de vista del usuario final de la aplicación, esta opción no resulta demasiado atractiva. Las limitaciones de los formularios estándar de HTML hacen difícil, si no imposible, construir interfaces de usuario amigables restringiéndonos al estándar.

Las limitaciones del formato HTML para construir interfaces de usuario (algo para lo que nunca fue diseñado) ha propiciado la aparición de numerosas tecnologías que permiten ejecutar código en la máquina del cliente, generalmente dentro del propio navegador web. Con estas tecnologías se consigue mejorar la escalabilidad de las aplicaciones, ya que se realiza menos trabajo en el servidor y éste puede atender a más clientes. Por otro lado, se mejora tanto la productividad como la satisfacción del usuario final, al emplear interfaces de usuario más depuradas y fáciles de manejar. Finalmente, estas tecnologías permiten conseguir aplicaciones muy atractivas desde el punto de vista estético, algo a lo que los programadores no suelen prestar demasiada atención pero que resulta de vital importancia a la hora de vender el producto, ya sea una aplicación a medida para un cliente particular o un sistema que se pone en el mercado a disposición de todo aquél que quiera usarlo desde la comodidad de su casa. Al fin y al cabo, ¿hace cuánto tiempo que no ve una aplicación web nueva en la que no se haya incluido una animación o presentación espectacular en Flash para agradar al cliente? Y eso que la dichosa presentación suele tardar un buen rato en descargarse y sólo supone una pérdida de tiempo para los usuarios habituales de la aplicación web (la cual, en demasiadas ocasiones, ni siquiera funciona correctamente).

A continuación mencionaremos algunas de las herramientas y tecnologías que se suelen utilizar para ejecutar parte de la aplicación web en la máquina del propio cliente:

HTML dinámico y JavaScript

Sin lugar a dudas, la herramienta más utilizada a la hora de dotar a nuestras páginas HTML de cierto comportamiento dinámico. El HTML dinámico (DHTML) se basa en construir un modelo basado en objetos del documento HTML, de forma que podamos acceder fácilmente a los distintos elementos que lo componen (véase la figura). La modificación dinámica de la página HTML se realiza a través de pequeñas macros o *scripts* que suelen incluirse en el mismo fichero que la página, si bien siempre es conveniente intentar separarlas del HTML para no mezclar los detalles del HTML de la interfaz con la lógica que implementan dichas macros.



DHTML DOM [Document Object Model]: Facilita la creación de páginas web dinámicas al ofrecer una forma cómoda de acceder a los distintos elementos que componen una página web.

En HTML dinámico, cada etiqueta HTML se convierte en un objeto con sus propiedades y eventos asociados. Los scripts han de proporcionarle al navegador el código correspondiente a la respuesta prevista por el programador para los distintos eventos que se pueden producir. Esto es, las macros se ejecutan cuando se produce algún evento asociado a alguno de los elementos de la página web de modo análogo a como se programa en cualquier entorno de programación visual para construir interfaces de usuario.

Usualmente, las macros se escriben utilizando JavaScript por cuestiones de portabilidad, si bien navegadores web como el Internet Explorer de Microsoft también permiten otros lenguajes como VBScript [Visual BASIC Script]. En realidad, aunque existe un estándar oficial de JavaScript ratificado por ECMA (por lo que se le suele llamar ECMAScript), cada navegador implementa versiones sutilmente diferentes de JavaScript, con los consiguientes dolores de cabeza que esto conlleva para el programador. Pese a ello, JavaScript resulta una opción atractiva ya que no resulta difícil encontrar en Internet bibliotecas gratuitas de ejemplos que funcionan en los navegadores web más comunes (desde los típicos menús desplegables, banners, relojes y calendarios hasta juegos de ajedrez).

JavaScript es un lenguaje interpretado originalmente llamado LiveScript que Netscape desarrolló para sus productos relacionados con la web. De hecho, JavaScript funciona tanto en navegadores web como en el servidor HTTP de Netscape, al más puro estilo de las páginas ASP de Microsoft.

La sintaxis de JavaScript muy similar a la de Java y resulta fácil de aprender para cualquiera que tenga unos conocimientos básicos de C. Por ejemplo, para declarar una variable no hay que especificar su tipo; basta con asignarle un valor que podemos obtener de alguno de los elementos de nuestra página web (el primer elemento del primer formulario, por ejemplo):

```
var dato = document.forms[0].elements[0].value;
```

Inicialmente, sólo el navegador de Netscape soportaba JavaScript, si bien Microsoft no tardó en incorporar una versión ligeramente modificada de JavaScript denominada JScript (cuando Netscape acaparaba el mercado de los navegadores web y Microsoft era un aspirante). Las resultantes inconsistencias hacen difícil escribir código que funcione correctamente en ambos navegadores, si bien Microsoft ha ganado la batalla y ahora son los demás los que tienen que intentar que sus navegadores interpreten el HTML dinámico de la forma que lo hace el de Microsoft.

Si bien Netscape y Sun Microsystems se han beneficiado mutuamente de su cooperación para facilitar el intercambio de mensajes y datos entre Java y JavaScript, JavaScript es independiente de Java. JavaScript es hoy un estándar abierto, ratificado por ECMA igual que el lenguaje C#, mientras que Java es propiedad de Sun Microsystems.

Controles ActiveX

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 como los que se utilizan en el desarrollo de aplicaciones para Windows. Los controles ActiveX están construidos sobre COM [*Component Object Model*], el modelo de Microsoft para desarrollo de componentes anterior a la plataforma .NET. A diferencia de JavaScript, que es un lenguaje totalmente interpretado, los controles ActiveX se compilan previamente, lo que permite su ejecución más eficiente.

Al ser una tecnología específica de Microsoft, la inclusión de controles ActiveX en páginas web sólo funciona correctamente en el navegador web de Microsoft, el Internet Explorer. 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 uno puede permitirse el

lujo de que su aplicación web no sea realmente portable.

En cierta medida, se puede decir que los controles ActiveX fueron la primera respuesta de Microsoft a los applets de Java promovidos por Sun Microsystems. La segunda, mucho más ambiciosa, fue la creación de la plataforma .NET.

Applets

Los applets son aplicaciones escritas en Java que se ejecutan en el navegador web. A diferencia de las macros interpretadas de JavaScript, un applet de Java es una aplicación completa compilada para la máquina virtual de Java (similar a la máquina virtual de la plataforma .NET). Los applets se adjuntan a las páginas web y pueden ejecutarse en cualquier navegador que tenga instalada una máquina virtual Java (básicamente, un intérprete del código intermedio que genera el compilador de Java).

Si bien su utilización se puede haber visto limitada por algunos problemas de rendimiento (la ejecución de un applet es, en principio, más lenta que la de un control ActiveX equivalente), los litigios existentes entre Microsoft y Sun Microsystems han ocasionado que su uso se haya realizado con cautela en la construcción de aplicaciones web. A pesar del eslogan de Java, "escribe una vez, ejecuta en cualquier sitio", la existencia de distintos navegadores y de sus sutiles diferencias restan atractivo a la construcción de interfaces web a base de applets. De hecho, el uso de Java está más extendido en el servidor, donde es muy común implementar las aplicaciones con servlets y páginas JSP.

Cuando se utiliza un applet, se descarga del servidor web el código intermedio del applet correspondiente a la máquina virtual Java; esto es, sus *bytecodes*. Al no tener que difundir el código fuente de la aplicación y disponer de una plataforma completa para el desarrollo de aplicaciones, se suelen 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.

Igual que JavaScript, los applets tienen la ventaja de funcionar sobre cualquier navegador que se precie (Netscape Navigator, Internet Explorer de Microsoft o HotJava de Sun). Además de por su portabilidad, garantizada con que exista un intérprete de bytecodes para la máquina virtual Java, los applets destacan por su seguridad: cada aplicación se ejecuta en un espacio independiente [sandbox] que, en principio, no puede acceder al hardware de la máquina del cliente (salvo que éste, explícitamente, lo autorice).

Plug-ins específicos

Los navegadores web pueden extenderse con *plug-ins*. Los *plug-ins* son componentes que permiten alterar, mejorar o modificar la ejecución de una aplicación en la que se instalan. Por ejemplo, los navegadores web suelen incluir *plug-ins* para visualizar documentos en formato PDF (de Adobe), ejecutar presentaciones Flash (de Macromedia), escuchar sonidos

RealAudio, mostrar imágenes vectoriales en formato SVG [Scalable Vector Graphics], ejecutar applets escritos para la máquina virtual Java o recorrer mundos virtuales VRML [Virtual Reality Markup Language].

Para que nuestra página web incluya un fichero para el cual necesitemos un plug-in específico, basta con incluir una etiqueta `EMBED` en el HTML de la página web. Los plug-ins, usualmente, son gratuitos. Basta con descargar de Internet la versión adecuada para nuestro sistema operativo e instalarla una única vez, tras lo cual queda almacenada localmente y podemos utilizarla cuantas veces necesitemos.

Existen distintos productos que nos permiten construir aplicaciones web utilizando plug-ins, como es el caso de **Curl**, un curioso lenguaje comercializado por un spin-off del MIT. Curl mejora la capacidad de HTML a la hora de construir interfaces de usuario sin salirse del navegador web. Más que por sus características (Curl viene a ser una especie de LaTeX para el desarrollo de interfaces), este producto resulta curioso por el modelo de negocio sobre el que espera prosperar. En vez de pagar los gastos fijos de una licencia para instalar Curl en el servidor web, el que lo utiliza ha de pagar en función de la cantidad de datos que se transmiten comprimidos desde el servidor. Igual que en el caso de los controles ActiveX, esta tecnología puede que tenga éxito en el desarrollo de aplicaciones web para intranets, en situaciones en las que el ancho de banda utilizado por las aplicaciones web tradicionales pueda llegar a ser un obstáculo para su implantación.

En el servidor

Las tecnologías específicas descritas en la sección anterior permiten, fundamentalmente, mejorar la interfaz de usuario de nuestras aplicaciones web en el navegador del cliente, al menos en cierta medida. Independientemente de que utilicemos o no dichas tecnologías, la funcionalidad de las aplicaciones web usualmente la tendremos que implementar en el lado del servidor.

Para construir aplicaciones web que se ejecuten en el servidor disponemos, si cabe, de más alternativas que en el cliente. Además, ya que la aplicación se ejecutará en el servidor, no necesitamos tener ningún plug-in instalado en la máquina del cliente y, en muchas ocasiones, nos bastará con utilizar el HTML dinámico disponible en los navegadores web actuales.

Las aplicaciones que se ejecutan en el servidor pueden recibir información del cliente de distintas formas. Por ejemplo, los datos recogidos por un formulario HTML pueden enviarse al servidor codificados en la propia URL (el identificador unívoco que aparece en la barra de direcciones del navegador) o enviarse en la cabecera del mensaje HTTP que se envía automáticamente cuando el usuario pulsa un botón del formulario. Además, las aplicaciones web pueden almacenar pequeñas cadenas de texto en el navegador web del cliente para realizar tareas como el mantenimiento de sesiones del usuario (las famosas *cookies*).

Generalmente, las herramientas que utilicemos nos simplificar el acceso a los datos facilitados desde el cliente. Una vez que se adquieren estos datos, la aplicación web ha de procesarlos de

acuerdo a sus requisitos funcionales, los cuales pueden involucrar el acceso a bases de datos, el uso de ficheros, el envío de mensajes a otras máquinas utilizando algún tipo de middleware o, incluso, el acceso a otros servidores web, posiblemente utilizando los protocolos asociados a los servicios web.

Finalmente, 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 su accesibilidad (desde cualquier punto de Internet), su fácil mantenimiento (no hay que distribuir el código de las aplicaciones ni sus actualizaciones), su relativa seguridad (el código no puede manipularlo el usuario, al que sólo le llega una representación de los datos que le incumban) y su escalabilidad (utilizando arquitecturas multicapa y clusters de PCs resulta relativamente sencillo ampliar en número de clientes a los que puede dar servicio la aplicación).

Dada la gran variedad de herramientas y tecnologías que se pueden utilizar para construir aplicaciones web en el servidor, intentaremos agruparlas en unas pocas categorías mencionando algunas de las más representativas, ya que continuamente aparecen nuevas formas de implementar aplicaciones web e intentar enumerarlas todas sólo serviría para que este libro quedase obsoleto antes de su publicación:

Aplicaciones web compiladas: CGI

CGI es el nombre que se le da a una aplicación web que recibe sus parámetros utilizando el estándar *Common Gateway Interface*, de ahí su nombre. El estándar establece cómo han de comunicarse la aplicaciones con el servidor web. Por extensión, se denomina CGI a un módulo de una aplicación web que se implementa utilizando el estándar CGI en un lenguaje de programación tradicional como C. En realidad, un CGI se encarga únicamente de implementar la respuesta de la aplicación web a un tipo concreto de solicitud proveniente del cliente. Por tanto, una aplicación web estará formada, en general, por muchos CGIs diferentes. Cada uno de ellos será responsable de un contexto de interacción de la aplicación con el usuario (la interacción del usuario con la aplicación que se realiza como una única acción del usuario).

El estándar CGI permite ejecutar programas externos a un servidor HTTP. El estándar especifica cómo se pasan los parámetros al programa como parte de la cabecera de la solicitud HTTP y también define algunas variables de entorno. En realidad un CGI puede ser cualquier programa que pueda aceptar parámetros en su línea de comandos. En ocasiones se utilizan lenguajes tradicionales como C y, otras veces, se emplean lenguajes ideados expresamente para construir aplicaciones web (como es el caso de Perl).

Si escribimos un CGI en C, lo único que tenemos que hacer es escribir un programa estándar

que acepte como parámetros los datos recibidos desde el cliente. Dicho programa ha de generar la respuesta HTTP correspondiente a través de su salida estándar, `stdout` en el caso de C. Usualmente, el programa CGI generará un documento en HTML, si bien también puede redirigir al usuario a otra URL, algo permitido por el protocolo HTTP y que se utiliza mucho para registrar el uso de los enlaces en buscadores y banners publicitarios. Al tener que realizar todo esto a bajo nivel, la tarea se vuelve harto complicada en aplicaciones web de cierta envergadura, lo que nos hará buscar soluciones alternativas como las que se describen en los apartados siguientes.

Habitualmente, un CGI es una aplicación independiente que compilamos para el sistema operativo de nuestro servidor web (esto es, un fichero EXE en Windows). Dicha aplicación se suele instalar en un subdirectorio específico del servidor web, el directorio `/cgi-bin`, si bien esto no es estrictamente necesario. Cada vez que el servidor web recibe una solicitud para nuestra aplicación, el servidor web crea un nuevo proceso para atenderla. Si la ejecución del proceso falla por algún motivo o se reciben más solicitudes que procesos pueden crearse, nuestra aplicación dejará de responder a las solicitudes del usuario.

Dado que lanzar un proceso nuevo cada vez que recibimos una solicitud del usuario puede resultar bastante costoso, los servidores web suelen incluir la posibilidad de implementar nuestros CGIs como bibliotecas que se enlazan dinámicamente con el servidor web y se ejecutan en su espacio de direcciones (DLLs en el caso de Windows). De esta forma se elimina la necesidad de crear un proceso independiente y realizar cambios de contexto cada vez que nuestra aplicación web deba atender una petición HTTP. Éste el mecanismo que utilizan los módulos ISAPI [Internet Server API] del Internet Information Server de Microsoft o los módulos NSAPI [Netscape Server API] del servidor HTTP de Netscape. Facilidades como ISAPI o NSAPI sirven para mejorar el rendimiento de los CGIs y poco más. Un fallo en la implementación de un módulo de nuestra aplicación web puede ocasionar que "se caiga" en servidor web por completo, algo que normalmente no sucederá con los CGIs que se ejecutan de forma independiente al servidor web.

Servlets

El término servlet, por analogía con el término applet, hace referencia a un programa escrito en Java que se ejecuta en el servidor en vez de ejecutarse en el navegador del cliente como los applets.

En realidad, un servlet extiende el comportamiento del servidor web de la misma forma que un CGI tradicional. La principal diferencia es que Java nos ofrecen una completa biblioteca de clases para implementar cómodamente la funcionalidad de nuestra aplicación web.

El uso de servlets, que encapsulan la comunicación con el servidor web y el usuario final de la aplicación, permite que el programador se centre en la lógica de su aplicación sin tener que preocuparse en exceso de los detalles del protocolo HTTP. Además, al ser Java un lenguaje orientado a objetos, las aplicaciones web resultantes son más elegantes, modulares y flexibles que las que se construyen con CGIs.

Aplicaciones web interpretadas: CGI scripts & Scripting languages

Un script CGI es igual que una aplicación CGI, salvo que su implementación se realiza utilizando lenguajes interpretados de propósito específico, de ahí el término con el que a menudo se hace referencia a ellos: *scripting languages*. **Perl**, **PHP** [*Personal Home Page*] y **ColdFusion** (extensión *.cfm*) o **Groovy** (basado en Java) son algunos de los lenguajes más populares pertenecientes a esta categoría, si bien también se pueden utilizar directamente macros del shell de UNIX o incluso lenguajes antiguos como **REXX** [*Restructured EXtended eXecutor*, 1979].

Por ejemplo, a continuación se muestra un pequeño programa escrito en Perl que almacenaríamos en un fichero de nombre `prueba.cgi`:

```
#!/usr/bin/perl
use CGI qw(:standard);

print header,
      start_html,
      h1("CGI de ejemplo"),
          "Su dirección IP es: ", remote_host(),
          end_html;
```

Cuando se ejecuta el programa anterior, se genera automáticamente la cabecera HTTP asociada a la respuesta y un documento HTML que muestra la dirección IP de la máquina que realizó la solicitud HTTP en primera instancia. Si lo ejecutamos localmente 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>
```

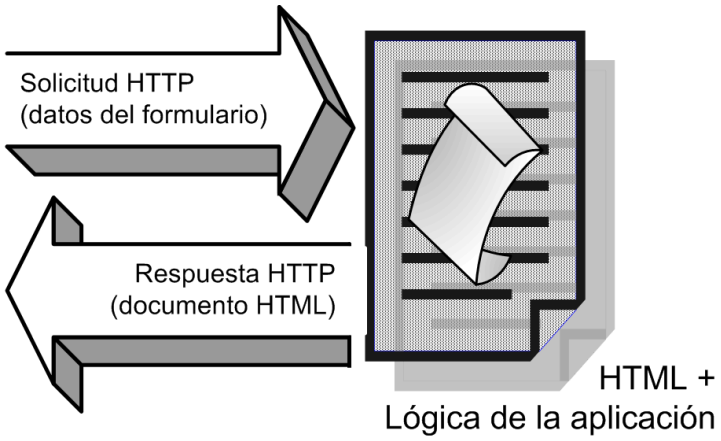
Para ejecutar la aplicación web, el usuario no tendría más que escribir la URL correspondiente en la barra de direcciones de su navegador:

```
http://csharp.ikor.org/cgi-bin/prueba.cgi
```

suponiendo, claro está, que nuestro servidor web fuese `csharp.ikor.org`.

Páginas de servidor: ASP y JSP

Existen otras tecnologías, similares a los lenguajes interpretados como Perl, que nos permiten preparar documentos HTML dentro de los cuales podemos introducir fragmentos de código que será interpretado por el servidor web cuando atienda las solicitudes HTTP que reciba. 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. Las páginas **JSP** [*Java Server Pages*] de Java y las páginas **ASP** [*Active Server Pages*] de Microsoft son los ejemplos más representativos de este tipo de sistemas.



Funcionamiento de las páginas de servidor: Una página ASP/JSP contiene HTML estático intercalado con scripts que se encargan de generar HTML de forma dinámica.

Igual que los lenguajes de propósito específico o los servlets, las páginas de servidor (ASP o JSP) resultan más cómodas para el programador que los CGIs, ya que no tiene que tratar directamente con los mensajes HTTP que se transmiten desde y hasta el navegador web del cliente. Sin embargo, el diseño de las aplicaciones resultantes no suele ser demasiado elegante, pues mezcla la interfaz de usuario con la lógica de la aplicación (y siempre deberíamos aspirar a construir aplicaciones modulares con módulos débilmente acoplados).

Las páginas ASP permiten crear aplicaciones web fácilmente incluyendo en los documentos HTML fragmentos de código escrito en un lenguaje como VBScript, lo más usual, o JScript, la versión de JavaScript de Microsoft. En la plataforma .NET, las páginas ASP.NET reemplazan a las páginas ASP y el resto de este capítulo y los siguientes los dedicaremos a

estudiar la construcción de aplicaciones web utilizando esta tecnología en el servidor HTTP de Microsoft (el Internet Information Server).

JSP (http://java.sun.com/products/jsp/) funciona de forma análoga a las páginas ASP, si bien utiliza el lenguaje de programación Java. En realidad, JSP no es más que una extensión de los servlets Java que permiten programar las aplicaciones web tal como estaban acostumbrados los programadores que utilizaban páginas ASP antes de que JSP existiese. Por este motivo se suele considerar que las páginas JSP son más sencillas que los servlets o los CGIs.

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. JSP permite separar mejor la interfaz de usuario de la generación de contenido que las páginas ASP tradicionales, de forma que los diseñadores web pueden modificar el aspecto de una página web sin que eso interfiera en la programación de la aplicación web. Esto se debe a que JSP utiliza etiquetas al estilo de XML para generar el contenido de forma dinámica, generación de la que se harán cargo componentes instalados en el servidor (*JavaBeans*), el cual puede ser un servidor HTTP como Apache, un servidor escrito en Java como Tomcat (conocido familiarmente como "el gato Tom") o cualquiera de los muchos servidores de aplicaciones existentes que se ajustan al estándar J2EE [*Java 2 Enterprise Edition*].

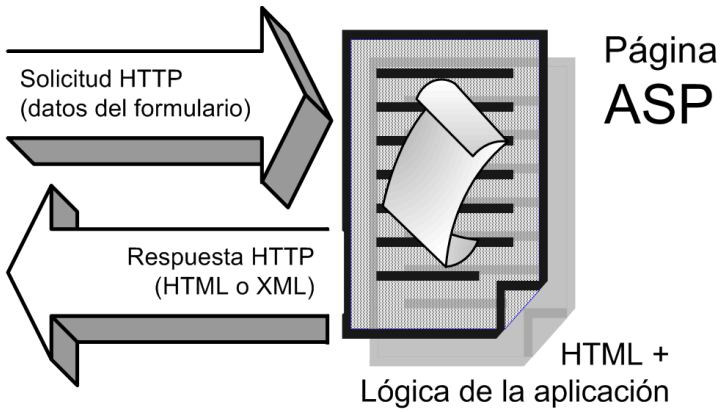
Recordatorio

Algo que nunca debemos olvidar es que tanto ASP como JSP gozan de gran aceptación a pesar de que no fuerzan uno de los principios básicos de diseño de software: la separación entre la interfaz de una aplicación y su lógica interna.

ASP: Active Server Pages

Tal como mencionamos en la sección anterior, ASP es la tecnología de Microsoft que permite desarrollar aplicaciones web que ejecuten en el servidor HTTP de Microsoft, el Internet Information Server (IIS). El desarrollo de aplicaciones utilizando páginas ASP consiste, básicamente, en intercalar macros o 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 generar la parte dinámica. Esto suele conducir a mezclar los detalles de la interfaz con la lógica de la aplicación, algo que, repetimos, no suele ser demasiado recomendable.

Una página ASP no es más que un fichero HTML con extensión .asp (.aspx en el caso de ASP.NET) al que le añadimos algo de código. Este código se pueden implementar utilizando distintos lenguajes interpretados. Por lo general, se emplea una variante de Visual Basic conocida como VBScript [*Visual Basic Script*]. Cuando alguien accede a la página, el Internet Information Server interpreta el código que incluye la página y combina el resultado de su ejecución con la parte estática de la página ASP (la parte escrita en HTML convencional). Una vez interpretada la página ASP, el resultado final es lo que se envía al navegador web instalado en la máquina del usuario que accede a la aplicación.



Funcionamiento de las páginas ASP: La parte estática de la página se envía junto con el resultado de ejecutar el código en el servidor, de ahí el "AS" de ASP.

Para desarrollar páginas ASP con comodidad, el programador dispone de una serie de objetos predefinidos que simplifican su trabajo ocultando los detalles de la comunicación del

navegador web del cliente con el servidor HTTP. Igual que en el caso de las páginas JSP en Java y de otras muchas alternativas para desarrollar aplicaciones web en el servidor, las mencionadas en el apartado anterior de este capítulo, los objetos definidos en ASP proporcionan distintos servicios útiles en el desarrollo de aplicaciones web (véase la tabla adjunta), además de facilidades para acceder a componentes COM (por ejemplo, se puede utilizar ADO [ActiveX Data Objects] para acceder a bases de datos).

Objeto	Encapsula
`Request`	La solicitud HTTP recibida
`Response`	Las respuesta HTTP generada
`Server`	El estado del servidor
`Application`	El estado de la aplicación web
`Session`	La sesión de usuario

Igual que sucede con las demás tecnologías basadas en lenguajes interpretados, para dejar nuestra aplicación web a disposición del usuario basta con escribir las páginas ASP y guardarlas en algún directorio al que se pueda acceder a través del Internet Information Server, sin tener que compilarlas previamente.

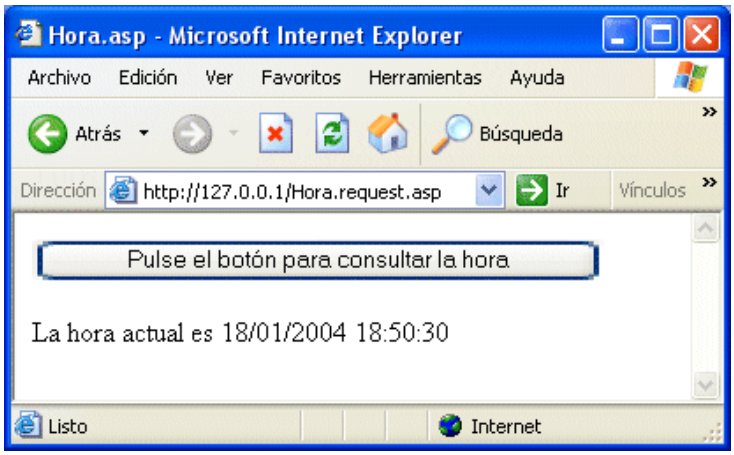
El siguiente ejemplo muestra cómo se pueden crear fácilmente páginas ASP que generen su contenido dinámicamente. Para crear nuestra primera página web, sólo tenemos que crear un fichero de texto llamado `Hora.asp` y colocarlo en algún directorio al que se pueda acceder desde un navegador web utilizando el IIS. Dentro del fichero podemos escribir lo siguiente:

```
<html>
 <head>
  <title>Hora.asp</title>
 </head>
 <body>
  <h2> Hora actual </h2>
  <% Response.Write(Now()) %>
 </body>
</html>
```

El código que aparece entre las etiquetas `<%` y `%>` contiene la parte de la página ASP se interpreta en el servidor antes de enviarle nada al navegador del cliente. El método `Response.Write` sirve para escribir algo en la página HTML que se genera como resultado de la ejecución de la página ASP mientras que la función `Now()` es una función predefinida que nos devuelve la fecha y la hora que marque el reloj del sistema donde esté instalado el servidor web. Siempre que queramos generar dinámicamente parte de la página utilizaremos el objeto `Response` para hacerle llegar al usuario aquello que nos interese.

En el ejemplo anterior, el fragmento de código incluido entre las etiquetas <% y %> se ejecuta cuando el usuario accede por primera vez a la página `Hora.asp`. Una vez que se carga la página, el resultado de la ejecución de la página ASP queda almacenado en la caché del navegador del cliente y la hora no se actualiza hasta que el usuario solicite explícitamente que la página se vuelva a descargar del servidor (pinchando en el botón actualizar de la barra de herramientas del navegador web que esté utilizando).

Podríamos modificar la página ASP que utilizamos en el ejemplo de arriba para mostrar cómo se puede utilizar el objeto `Request`. En esta ocasión, nuestra página ASP incluirá un botón gracias al cual el usuario podrá visualizar la hora actual del servidor en el momento que le interese sin necesidad de conocer el comportamiento interno del navegador web que utilice:



Aspecto visual de nuestra sencilla página ASP.

Para incluir controles en nuestra página ASP a través de los cuales el usuario pueda comunicarse con el servidor web, lo único que tenemos que hacer es incluir un formulario HTML en nuestra página ASP:

```
<html>
 <head>
  <title>Hora.Request.asp</title>
 </head>
 <body>
  <form method="post">
     <input type="submit" id=button name=button
            value="Pulse el botón para consultar la hora" />
     <%
       if (Request.Form("button") <> "") then
          Response.Write "<p>La hora actual es " & Now()
       end if
     %>
```

```
    </form>
  </body>
</html>
```

En esta ocasión, hemos utilizado el objeto `Request` para recoger los datos de entrada que llegan a nuestra página ASP desde el formulario incluido en la página HTML que se visualiza en el navegador web del cliente. `Request` encapsula los datos enviados desde el cliente y nos permite programar nuestra página sin tener que conocer los detalles del protocolo HTTP relativos al envío de datos asociados a un formulario HTML.

Para comprobar los ejemplos anteriores funcionan, sólo hay que guardar los ficheros ASP correspondientes en algún sitio que cuelgue del directorio `wwwroot` del IIS y acceder a él desde el navegador utilizando una URL de la forma `http://localhost/...`.

Como no podía ser de otra forma, en la plataforma .NET se ha incluido una versión mejorada de ASP denominada ASP.NET. Entre los principales inconvenientes asociados a las versiones de ASP anteriores a ASP.NET, a las que se suele hacer referencia por "ASP Clásico", se encuentra el hecho de ASP suele requerir escribir bastante código. Por ejemplo, es necesario escribir código para mantener el estado de la página ASP cuando, por cualquier motivo, el usuario ha de volver a ella. Omitir dicho código provoca la frustrante sensación que todos hemos experimentado alguna vez cuando, tras rellenar un extenso formulario HTML, se nos informa de un error en uno de los campos y nos vemos forzados a volver a rellenar el formulario completo.

Por otro lado, el código incluido en las páginas ASP resulta, además de excesivamente extenso, poco legible y difícil de mantener al estar mezclado con el HTML de la interfaz gráfica. En ASP, el fragmento de código ha de colocarse justo en el sitio donde queremos que su salida aparezca, lo que hace prácticamente imposible separar los detalles de la interfaz de usuario de la lógica de la aplicación. Este hecho, no sólo dificulta la reutilización del código y su mantenimiento, sino que también complica el soporte de nuestra aplicación para múltiples navegadores (recordemos, por ejemplo, que JavaScript y JScript no son exactamente iguales).

Finalmente, el "ASP Clásico" presenta otros inconvenientes que hacen su aparición a la hora de implantar sistemas reales de cierta envergadura. Como muestra, valga mencionar la dificultad que conlleva la correcta configuración de una aplicación ASP de cierta complejidad, los problemas de eficiencia que se nos pueden presentar cuando hemos de atender más peticiones que las que nuestro servidor puede atender, o los innumerables problemas que puede suponer la depuración de una aplicación construida con páginas ASP. En realidad, muchos de estos inconvenientes provienen de las limitaciones de los lenguajes interpretados que se utilizan para escribir los fragmentos de código incluidos en las páginas ASP.

Las limitaciones e inconvenientes mencionados en los párrafos anteriores propiciaron la

realización de algunos cambios notables en ASP.NET. ASP.NET, por tanto, no es completamente compatible con ASP, si bien la mayor parte de las páginas ASP sólo requieren pequeños ajustes para pasarlas a ASP.NET. De hecho, el primero de los ejemplos que hemos mostrado funciona perfectamente como página ASP.NET (sin más que cambiarle la extensión `.asp` por la extensión utilizada por las páginas ASP.NET: `.aspx`). El segundo de los ejemplos funcionará correctamente si tenemos en cuenta que, en Visual Basic .NET ,los paréntesis son obligatorios en las llamadas a los métodos (algo que era opcional en versiones anteriores de Visual Basic).

La siguiente sección de este capítulo y los capítulos siguientes los dedicaremos a tratar los detalles de funcionamiento de ASP.NET, la versión de ASP incluida en la plataforma .NET

ASP.NET: Aplicaciones web en la plataforma .NET

ASP.NET es el nombre con el que se conoce la parte de la plataforma .NET que permite el desarrollo y ejecución tanto de aplicaciones web como de servicios web. Igual que sucedía en ASP, ASP.NET se ejecuta en el servidor. En ASP.NET, no obstante, las aplicaciones web se suelen desarrollar utilizando formularios web, que están diseñados para hacer la creación de aplicaciones web tan sencilla como la programación en Visual Basic (.NET, claro está).

Un ejemplo

Para hacernos una idea de cómo es ASP.NET, retomemos el ejemplo de la sección anterior, que en ASP.NET queda como sigue si empleamos el lenguaje de programación C#:

Ejemplo de página ASP.NET

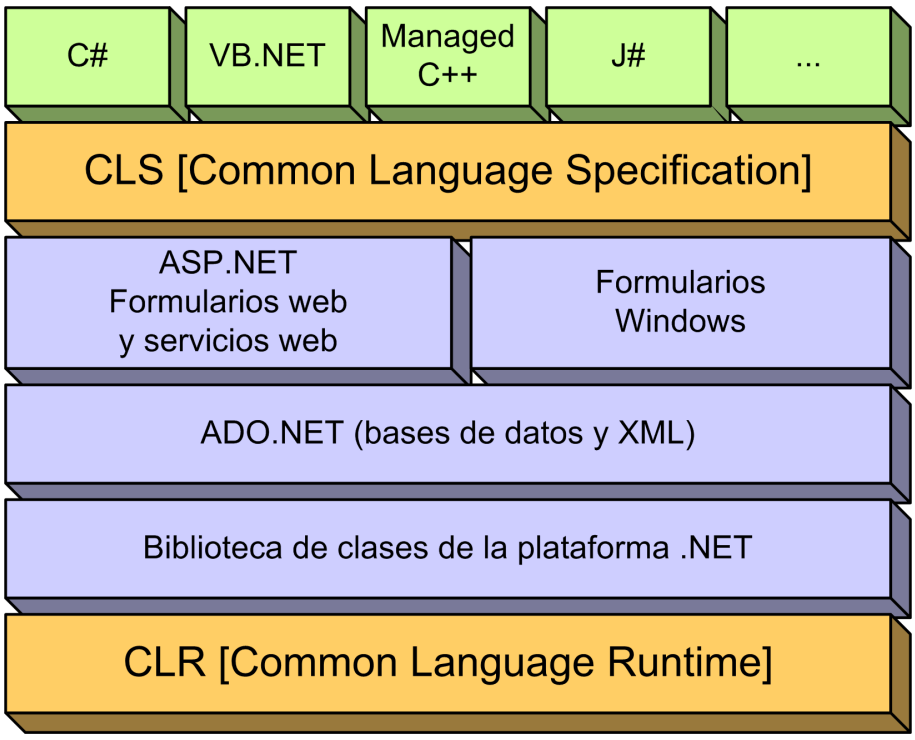
```
<%@ Page language="c#" %>
<html>
 <head>
   <title>Hora.aspx</title>
 </head>
 <script runat="server">
  public void Button_Click (object sender, System.EventArgs e)
  {
    LabelHora.Text = "La hora actual es " + DateTime.Now;
  }
 </script>
 <body>
  <form method="post" runat="server">
    <asp:Button onclick="Button_Click" runat="server"
                Text="Pulse el botón para consultar la hora"/>
    <p>
    <asp:Label id=LabelHora runat="server" />
  </form>
 </body>
</html>
```

Este sencillo ejemplo ilustra algunas de las características más relevantes de ASP.NET. Por ejemplo, podemos apreciar cómo el código de nuestra aplicación ya no está mezclado con las etiquetas HTML utilizadas para crear el aspecto visual de nuestra aplicación en el navegador del usuario. En vez de incluir código dentro de la parte correspondiente al HTML estático, algo que todavía podemos hacer al estilo de las páginas ASP tradicionales, hemos preferido utilizar un par de controles ASP.NET que nos permiten manipular en el servidor los elementos de nuestra página web con más comodidad (de forma similar a como JavaScript

nos da la posibilidad de modificar dinámicamente, en el cliente, el aspecto de los distintos elementos de la página).

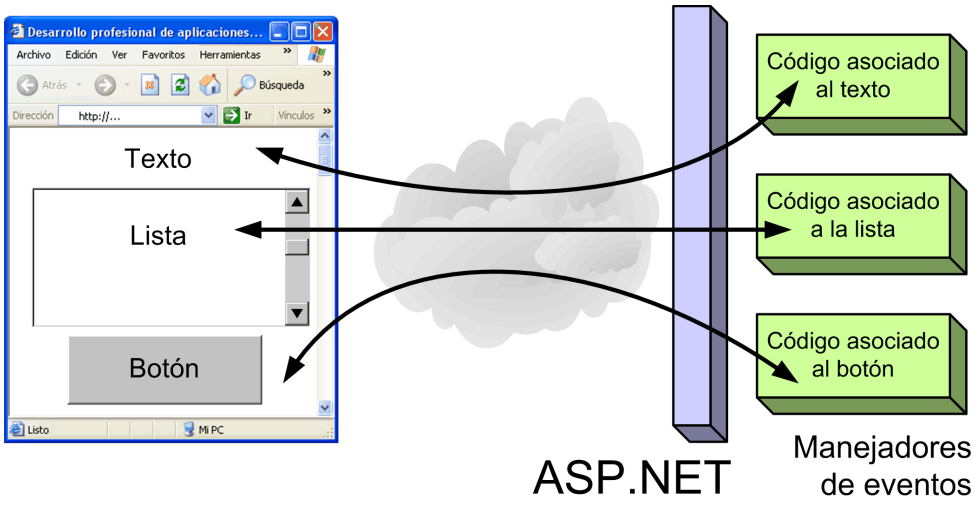
Para probar el funcionamiento del ejemplo anterior, sólo tenemos que guardar la página ASP.NET con la extensión `.aspx` en uno de los directorios de nuestra máquina a los que da acceso el Internet Information Server (por ejemplo, el directorio `/wwwroot`). Una vez hecho esto, podremos acceder a la página desde nuestro navegador web utilizando la URL adecuada. El Internet Information Server se encargará de interpretar la página ASP.NET de la forma adecuada.

ASP.NET forma parte de la plataforma .NET. De hecho, los formularios Windows y los formularios ASP.NET son las dos herramientas principales con las que se pueden construir interfaces de usuario en .NET. Aunque no son intercambiables, ya que aún no existe una forma estándar de crear una interfaz de usuario que funcione tanto para aplicaciones Windows como para aplicaciones web, tanto unos formularios como los otros comparten su posición relativa dentro de la familia de tecnologías que dan forma a la plataforma .NET.



La plataforma .NET: Los formularios ASP.NET y los formularios Windows son las dos alternativas principales de las que dispone el programador para crear las interfaces de usuario de sus aplicaciones.

Igual que sucede en el caso de los formularios Windows (y, de hecho, en cualquier entorno de programación visual para un entorno de ventanas como Windows, desde Visual Basic y Delphi hasta Oracle Developer), la programación en ASP.NET está basada en el uso de controles y eventos. Las páginas ASP.NET, en vez de aceptar datos de entrada y generar su salida en HTML como sucede en ASP, implementan su funcionalidad en fragmentos de código que se ejecutan como respuesta a eventos asociados a los controles de la interfaz con los que puede interactuar el usuario. Esta forma de funcionar le proporciona a ASP.NET un mayor nivel de abstracción, requiere menos código y permite crear aplicaciones más modulares, legibles y mantenibles.

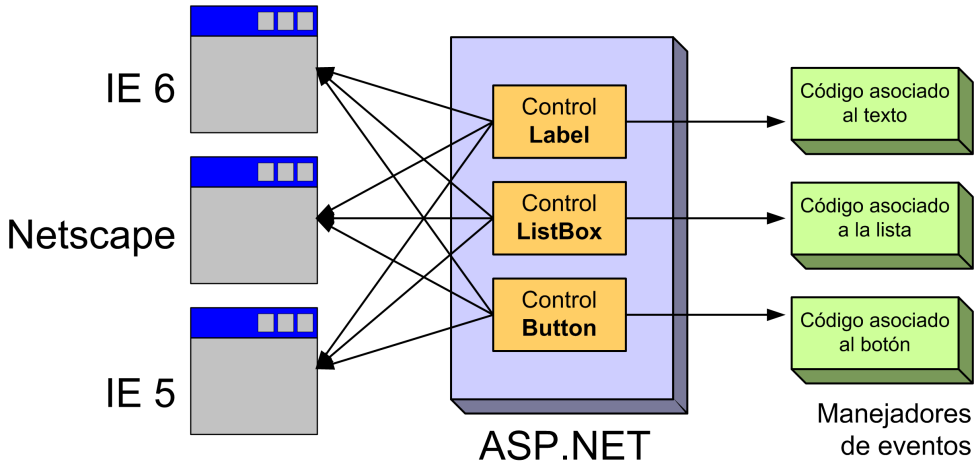


Eventos en ASP.NET: La respuesta de la aplicación web se obtiene como resultado de ejecutar los manejadores de eventos asociados a los controles incluidos en la interfaz de usuario.

Como muestra la figura, el código se ejecuta en el servidor web en función de los manejadores de eventos que definamos para los controles y páginas que conforman la interfaz de nuestra aplicación web. En ASP.NET, todos los controles que aparecen en la interfaz derivan de la clase `System.Web.UI.Control`. La página ASP.NET, en sí misma, también es un objeto. En este caso, la página hereda de `System.Web.UI.Page`, que a su vez deriva de `System.Web.UI.Control`.

Otra de las características destacables de ASP.NET es que las etiquetas que introducimos en la página HTML para incluir controles en la interfaz de usuario son independientes del HTML que después se genera para construir la interfaz de usuario que le llega al navegador del cliente. Es el caso, por ejemplo, de los controles `<asp:Label...>` y `<asp:Button...>` que aparecen en el ejemplo que abre este apartado. ASP.NET se encarga de convertir estas etiquetas en el fragmento de HTML que resulte más adecuado para

mostrar los controles en función del navegador web que utilice el usuario de nuestra aplicación. De esta forma, ASP.NET garantiza la compatibilidad de los controles de nuestra aplicación web con distintos navegadores, sin que el programador tenga que preocuparse demasiado de las diferencias existentes entre los diferentes navegadores web que puede emplear el usuario final para acceder a nuestra aplicación.



ASP.NET se encarga de mostrar los formularios web de la forma que resulte más adecuada para el navegador que utilice el usuario final de la aplicación.

Dos estilos

Si bien el ejemplo incluido en el apartado anterior puede que no nos permita apreciar la importancia de las mejoras de ASP.NET en cuanto a la separación de la interfaz de la lógica de la aplicación, lo cierto es que ASP.NET nos permite aprovechar todas las características de la plataforma .NET para el diseño de aplicaciones modulares utilizando técnicas de orientación a objetos. De hecho, ASP.NET nos permite utilizar dos estilos bien diferenciados para la confección de páginas ASP.NET:

- El primero de ellos consiste en incluir tanto los controles como el código en un único fichero `.aspx`, tal y como hicimos en el ejemplo anterior. A pesar de que los cambios introducidos por ASP.NET suponen una notable mejora respecto al ASP Clásico, esta forma de crear páginas ASP.NET nos impide aprovechar al máximo las ventajas de ASP.NET. Por ejemplo, puede que el responsable de implementar el código asociado a la funcionalidad de la página sea una persona diferente al diseñador gráfico que se encarga del aspecto visual de la aplicación. Tener todo en un único fichero puede provocar más que un simple conflicto de

intereses entre programadores y diseñadores gráficos. Baste con pensar qué podría suceder si el diseñador gráfico crea sus diseños utilizando algún editor visual de HTML, del tipo de los que no se suelen comportar correctamente con los fragmentos del documento HTML que no comprenden.

- Afortunadamente, ASP.NET también nos permite mantener los controles de nuestra interfaz en un fichero `.aspx` y dejar todo el código en un lugar aparte [*code-behind page*]. De esta forma, se separa físicamente, en ficheros diferentes, la interfaz de usuario del código de la aplicación. Esta alternativa es, sin duda, más adecuada que la anterior. En la situación antes descrita, tanto el programador como el diseñador gráfico pueden centrarse en su trabajo y el riesgo de que uno de ellos "estropee" el trabajo del otro es bastante reducido. Incluso aunque no exista la división de trabajo entre programador y diseñador gráfico, mantener el código separado facilita la construcción de aplicaciones web con diferentes aspectos visuales. Esto puede llegar a resultar necesario cuando un mismo producto se personaliza para distintos clientes o cuando nuestra empresa pretende crear una línea de productos sin hipotecar su futuro, ya que tener que mantener código duplicado puede llegar a hundir el mejor de los planes de negocio.

Veamos, a continuación, cómo quedaría nuestro ejemplo de antes si separamos físicamente el diseño de nuestra interfaz de la implementación de los manejadores de eventos que implementan la respuesta de nuestra aplicación cuando el usuario pide la hora.

En la página ASP.NET propiamente dicha, sólo incluiremos el HTML estático que se le envía al cliente junto con las etiquetas correspondientes a los controles ASP.NET que se mostrarán en el navegador del cliente como elementos de un formulario web convencional. Aparte de esto, también hemos de introducir una directiva en la cabecera de nuestra página que le permita saber al IIS dónde está el código que se ejecuta para conseguir el comportamiento dinámico de nuestra página. Todo esto lo pondremos en un fichero llamada `HoraWebForm.aspx`:

```
<%@ Page language="c#" Codebehind="HoraWebForm.aspx.cs"
Inherits="HoraWeb.WebForm" %>
<html>
 <head>
   <title>Hora.aspx</title>
 </head>
 </script>
 <body>
  <form method="post" runat="server">
    <asp:Button id="ButtonHora" runat="server"
                Text="Pulse el botón para consultar la hora" />
    <p>
    <asp:Label id="LabelHora" runat="server" />
  </form>
 </body>
</html>
```

Una vez que tenemos el diseño de nuestro formulario, sólo nos falta implementar su comportamiento dinámico en un fichero aparte. Este fichero de código lo escribiremos como cualquier otro fichero de código en C#, creando para nuestra página una clase que herede de `System.Web.UI.Page`. Tal como podemos deducir de lo que aparece en la directiva que abre nuestro fichero `.aspx`, el código lo guardaremos en el fichero `HoraWebForm.aspx.cs`:

```
namespace HoraWeb
{
  public class WebForm : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.Button ButtonHora;
    protected System.Web.UI.WebControls.Label LabelHora;

    override protected void OnInit(EventArgs e)
    {
      this.ButtonHora.Click += new
System.EventHandler(this.ButtonHora_Click);
      base.OnInit(e);
    }

    private void ButtonHora_Click(object sender, System.EventArgs e)
    {
      LabelHora.Text = "La hora actual es "+DateTime.Now;
    }
  }
}
```

Como se puede ver, el código asociado a nuestra aplicación web se escribe de forma completamente independiente a su presentación en HTML. En realidad, cuando creamos la página `.aspx`, lo que estamos haciendo es crear una subclase de la clase implementada en el fichero de código. Esta subclase se limita únicamente a organizar los elementos de nuestra interfaz de usuario de la forma que resulte más adecuada.

Como habrá podido observar, la creación de una página ASP.NET con C# resulta relativamente sencilla si se sabe programar en C# y se tienen unos conocimientos básicos de HTML. Sólo tenemos que aprender a manejar algunos de los controles utilizados en la construcción de formularios ASP.NET. Hasta ahora, todo lo hemos implementado a mano con el objetivo de mostrar el funcionamiento de ASP.NET y de desmitificar, en parte, la creación de aplicaciones web. No obstante, como es lógico, lo normal es que creemos las páginas ASP.NET con la ayuda del diseñador de formularios que nos ofrece el entorno de desarrollo Visual Studio .NET. Éste se encargará de rellenar muchos huecos de forma automática para que nos podamos centrar en la parte realmente interesante de nuestras aplicaciones. Pero antes de seguir profundizando en los detalles de ASP.NET tal vez convendría refrescar nuestros conocimientos de HTML.

Apéndice: Aprenda HTML en unos minutos

Sin lugar a dudas, el servicio de Internet más utilizada hoy en día es la World Wide Web (WWW), una aplicación que nos permite enlazar unos documentos con otros fácilmente. Para escribir documentos web, más conocidos como páginas web, se utiliza el formato HTML [*HyperText Markup Language*].

Un documento HTML es un fichero de texto normal y corriente (un fichero ASCII, por lo general). Este fichero incluye ciertas marcas o etiquetas que le indican al navegador, entre otras cosas, cómo debe visualizarse el documento. Las etiquetas, igual que en XML, se escriben encerradas entre ángulos: `<` y `>`. Además, dichas etiquetas usualmente van por parejas: `<etiqueta>` y `</tag>`.

Cualquier documento en formato HTML, delimitado por la pareja de etiquetas `<HTML>` y `</HTML>`, tiene dos partes principales:

- La cabecera (entre `<HEAD>` y `</HEAD>`) contiene información general sobre el documento que no se muestra en pantalla: título, autor, descripción...

- Las etiquetas `<BODY>` y `</BODY>` definen la parte principal o cuerpo del documento.

Documento HTML de ejemplo

```
<HTML>
  <HEAD>
    <TITLE> Título del documento </TITLE>
    <META NAME="Author" CONTENT="Fernando Berzal et al.">
    <META NAME="Keywords" CONTENT="HTML">
    <META NAME="Description" CONTENT="Conceptos básicos de HTML">
  </HEAD>
  <BODY>
    Cuerpo del documento...
  </BODY>
</HTML>
```

El cuerpo del documento HTML puede incluir, entre otros elementos:

- **Párrafos** (delimitados por la etiqueta `<P>`).

- **Encabezados**, empleados para definir títulos y subtítulos (de mayor a menor nivel, con las etiquetas `<H1>` a `<H6>`).

- **Enlaces** para enganchar unas páginas con otras: `<A HREF="url"> texto`

</A>, donde `url` indica la URL del documento al que apunta el enlace y `texto` es el fragmento del documento de texto sobre el cuál ha de pinchar el usuario para acceder al documento enlazado.

- **Imágenes** (en formato GIF, PNG o JPG): `<IMG SRC="fichero" BORDER=0 ALT="texto descriptivo">`, donde `url` indica la URL mediante la cual se puede acceder al fichero que contiene la imagen y `texto` es un texto alternativo que se le presenta al usuario cuando éste no ve la imagen.

- **Listas** numeradas (con `<OL>` y `</OL>`) o no numeradas (con `<UL>` y `</UL>`) cuyos elementos se indican en HTML utilizando la etiqueta `<LI>`.

Con el fin de personalizar la presentación del texto del documento, HTML incluye la posibilidad de poner el texto en negrita (`<B> ... </B>`), en cursiva (`<I> ... </I>`), subrayarlo (`<U> ... </U>`) o centrarlo (con `<CENTER> ... </CENTER>`). Además, se puede modificar el tamaño y color del tipo de letra con `<FONT SIZE="+1" COLOR="#rrggb"> ... </FONT>`, donde `SIZE` indica el tamaño (usualmente, un tamaño relativo al del texto actual, p.ej. +2 +1 -1 -2) y el color se suele representar en hexadecimal como una combinación de rojo, verde y azul. Por ejemplo, el negro es `#000000`, el blanco `#ffffff`, el rojo `#ff0000`, el verde `#00ff00` y el azul `#0000ff`.

Otras etiquetas habituales en los documentos HTML son `<BR>`, que introduce saltos de línea, y `<HR>`, que muestra en el navegador una línea horizontal a modo de separador. En los siguientes apartados presentaremos otros aspectos de HTML que conviene conocer:

Caracteres especiales

En lenguajes derivados de SGML, como es el caso de HTML o XML, las vocales acentuadas, las eñes y otros caracteres "no estándar" en inglés, incluyendo los ángulos que se utilizan para las etiquetas HTML, requieren secuencias especiales de caracteres para representarlos. La siguiente tabla recoge algunas de ellas:

Carácter	Secuencia HTML	Carácter	Secuencia HTML
ñ	ñ	Ñ	Ñ
&	&	€	€
<	<	á	á
>	>	è	è
"	"	ê	ê
©	©	ü	ü
®	®	™	™

Tablas

Las tablas se delimitan con las etiquetas <TABLE> y </TABLE>. Entre estas dos etiquetas se han de incluir una serie de filas delimitadas por <TR> y </TR>. Cada fila, a su vez, incluye una serie de celdas <TD> y </TD>. Por ejemplo:

```
<TABLE border=2>
  <TR bgcolor="#ccccee">
    <TH COLSPAN=2> <IMG SRC="cogs.gif"> Tabla en HTML </TH>
  </TR>
  <TR bgcolor="#e0e0e0">
    <TH> Datos</TH>
    <TH> Valores</TH>
  </TR>
  <TR>
    <TD> Dato 1</TD>
    <TD> Valor 1</TD>
  </TR>
  <TR>
    <TD> Dato 2</TD>
    <TD> Valor 2</TD>
  </TR>
  <TR>
    <TD> Dato 3</TD>
    <TD> Valor 3</TD>
  </TR>
</TABLE>
```

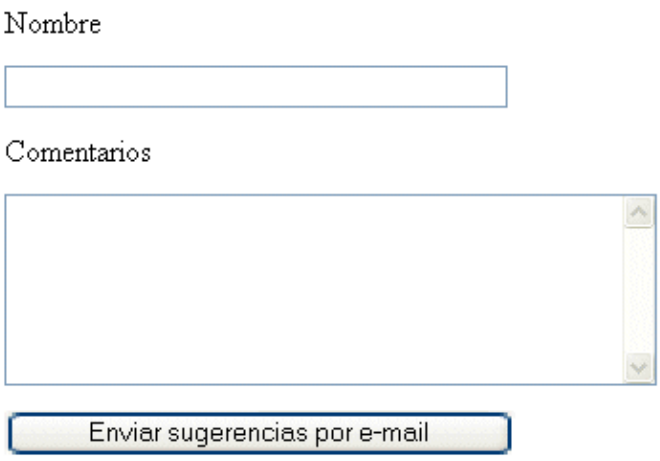
El fragmento de HTML anterior aparecerá en el navegador del usuario como:

Formularios

HTML también permite que el usuario no se limite a leer el contenido de la página, sino que también pueda introducir datos mediante formularios (la base de cualquier aplicación web. Por ejemplo, el siguiente fragmento de HTML contiene un formulario:

```
<FORM METHOD="POST" ACTION="mailto:berzal@acm.org">
  <INPUT TYPE="text" NAME="NOMBRE" SIZE=30 MAXLENGTH=40>
  <TEXTAREA NAME="COMENTARIOS" ROWS=6 COLS=40> </TEXTAREA>
  <INPUT TYPE="submit" VALUE="Enviar sugerencias por e-mail">
</FORM>
```

Este formulario, dentro de una tabla, se mostraría de la siguiente forma en el navegador del usuario:



En los formularios HTML estándar se pueden incluir:

- Cuadros de texto para que el usuario pueda escribir algo (`<INPUT TYPE="TEXT"...>`).
- Cuadros de texto que no muestran lo que el usuario escribe (`<INPUT TYPE="PASSWORD"...>`), indispensables cuando queremos que el usuario introduzca información privada.
- Textos de varias líneas (`<TEXTAREA ...> ... </TEXTAREA>`).

- Opciones que se pueden seleccionar o no de forma independiente (<INPUT TYPE="CHECKBOX"...>).
- Opciones mutuamente excluyentes (<INPUT TYPE="RADIO"...>).
- Listas para seleccionar valores (<SELECT> <OPTION> <OPTGROUP>).
- Ficheros adjuntos (<INPUT TYPE="FILE"...>).
- E, incluso, datos ocultos para el usuario (<INPUT TYPE="HIDDEN"...>).

Para enviar los datos del formulario a la URL especificada en FORM ACTION, que sería la dirección adecuada para nuestra aplicación web, se puede utilizar un botón (<INPUT TYPE="SUBMIT"...>) o una imagen (<INPUT TYPE="IMAGE"...>). En este último caso, la acción que se realice puede depender de la zona de la imagen que seleccione el usuario, pues, además de los datos rellenados en el formulario, recibiremos las coordenadas de la imagen correspondientes al punto donde el usuario pulsó el botón del ratón.

Hojas de estilo

El principal inconveniente que tiene el formato HTML a la hora de crear páginas web es que debemos indicar la forma junto con el contenido del documento. Cuando hay que mantener un gran número de páginas, sería conveniente disponer de un mecanismo que nos facilitase darles a todas las páginas un formato coherente. Las hojas de estilo en cascada, CSS [*Cascading Style Sheets*] son el estándar del W3C que nos permite facilitar el mantenimiento de un conjunto grande de páginas HTML y asegurarnos de que se mantiene cierta coherencia en su presentación de cara al usuario.

Para emplear una hoja de estilo en la presentación de nuestra página web, sólo tenemos que incluir la siguiente etiqueta antes del cuerpo de nuestro documento HTML:

```
<link REL=STYLESHEET TYPE="text/css" HREF="style.css">
```

donde style.css es el fichero que contiene la hoja de estilo que se empleará para visualizar el documento HTML.

El texto de la hoja de estilo ha de escribirse de acuerdo a la siguiente sintaxis:

```
ETIQUETA {
  propiedad1: valor1;
  propiedad2: valor2;
}
```

```
ETIQUETA1, ETIQUETA2 {
  propiedad: valor;
}

.CLASE {
  propiedad: valor;
}
```

donde las etiquetas y las clases son las que se utilizan en los documentos HTML, mientras que las propiedades aplicables a cada elemento y los valores que pueden tomar dichas propiedades están definidas en un estándar emitido por el W3C.

Por ejemplo, podemos hacer que el cuerpo de esta página se visualice con una imagen de fondo y se dejen márgenes alrededor del texto si escribimos la siguiente hoja de estilo:

```
BODY
{
 background-image: url(http://csharp.ikor.org/image/csharp.jpg);
 color: #000000;
 margin-left: 10%;
 margin-right: 10%;
 margin-top: 5%;
 margin-bottom: 5%;
}
```

También podemos definir estilos que nos permitan ver fragmentos de nuestros documentos con el fondo en gris de la misma forma que aparecen los ejemplos de esta sección. Sólo tenemos que definir una clase `ejemplo`:

En el documento HTML:

```
...
<table class="example">
...
```

En la hoja de estilo CSS:

```
.example
{
  background-color: #e0e0e0;
}
```

Incluso podemos definir características comunes para distintas etiquetas de las que aparecen en un documento HTML. La siguiente hoja de estilo hace que el texto incluido en párrafos,

citas, listas y tablas aparezca justificado a ambos márgenes:

```
P, BLOCKQUOTE, LI, TD
{
 text-align: justify;
}
```

Finalmente, algunos navegadores nos permiten modificar la forma en la que se visualizan los enlaces de una página web, para que estos cambien al pasar el cursor del ratón sobre ellos:

```
A
{
  text-decoration: none;
}

A:hover
{
  color: #009999;
}
```

Jugando un poco con las posibilidades que nos ofrecen las hojas de estilo CSS se puede conseguir que nuestras páginas HTML estándar tengan buen aspecto sin tener que plagarlas de etiquetas auxiliares como `FONT`, las cuales lo único que consiguen es que el texto de nuestro documento HTML sea menos legible y más difícil de mantener.

Información adicional

Toda la información relativa a los estándares relacionados con la web se puede encontrar en la página del organismo que los establece, el *World Wide Web Consortium* (W3C). En dicha página encontrará el estándar oficial, si bien, en la práctica, puede que le interese más visitar sitios como *Index DOT*. El ellos encontrará toda la información que necesite acerca de las páginas HTML y las hojas de estilo CSS, incluyendo comentarios acerca de cómo funciona cada etiqueta con las distintas versiones de los navegadores web más populares.

- W3C: http://www.w3c.org

- Index DOT: http://www.blooberry.com/indexdot/index.html