Formularios web

En este capítulo, nos centraremos en la construcción de páginas ASP.NET y adquiriremos los conocimientos necesarios para ser capaces de crear nuestras propias páginas web dinámicas con ASP.NET:

- En primer lugar, veremos en qué consisten los formularios web, una parte fundamental de la plataforma .NET.

- A continuación, estudiaremos los controles que podemos incluir dentro de las interfaces web que construiremos con formularios ASP.NET. Primero, analizaremos los tres tipos de componentes estándar incluidos en las bibliotecas de la plataforma .NET correspondientes a ASP.NET. Después, llegaremos a ver cómo podemos crear nuestros propios controles.

- Cuando ya tengamos una buena noción de lo que podemos incluir en un formulario ASP.NET, aprenderemos algo más acerca de su funcionamiento. En concreto, nos interesará saber qué son los *"post backs"* y cómo se mantiene el estado de una página ASP.NET.

Formularios web

Formularios en ASP.NET

En el capítulo anterior, presentamos una panorámica general del desarrollo de aplicaciones web, en la que mostramos las distintas alternativas de las que dispone el programador y situamos en su contexto la tecnología incluidas en la plataforma .NET para la creación de interfaces web: las páginas ASP.NET.

ASP.NET sustituye a las páginas interpretadas utilizadas en ASP por un sistema basado en componentes integrados en la plataforma .NET. De esta forma, podemos crear aplicaciones web utilizando los componentes que vienen incluidos en la biblioteca de clases de la plataforma .NET o, incluso, creando nuestros propios componentes. Lo usual es que estos últimos los implementemos a partir de los componentes existentes por composición; esto es, encapsulando conjuntos de componentes existentes en un componente nuevo. No obstante, también podemos crear nuevos componentes por derivación, creando una nueva clase derivada de la clase del componente cuyo comportamiento deseamos extender (como en cualquier entorno de programación orientado a objetos).

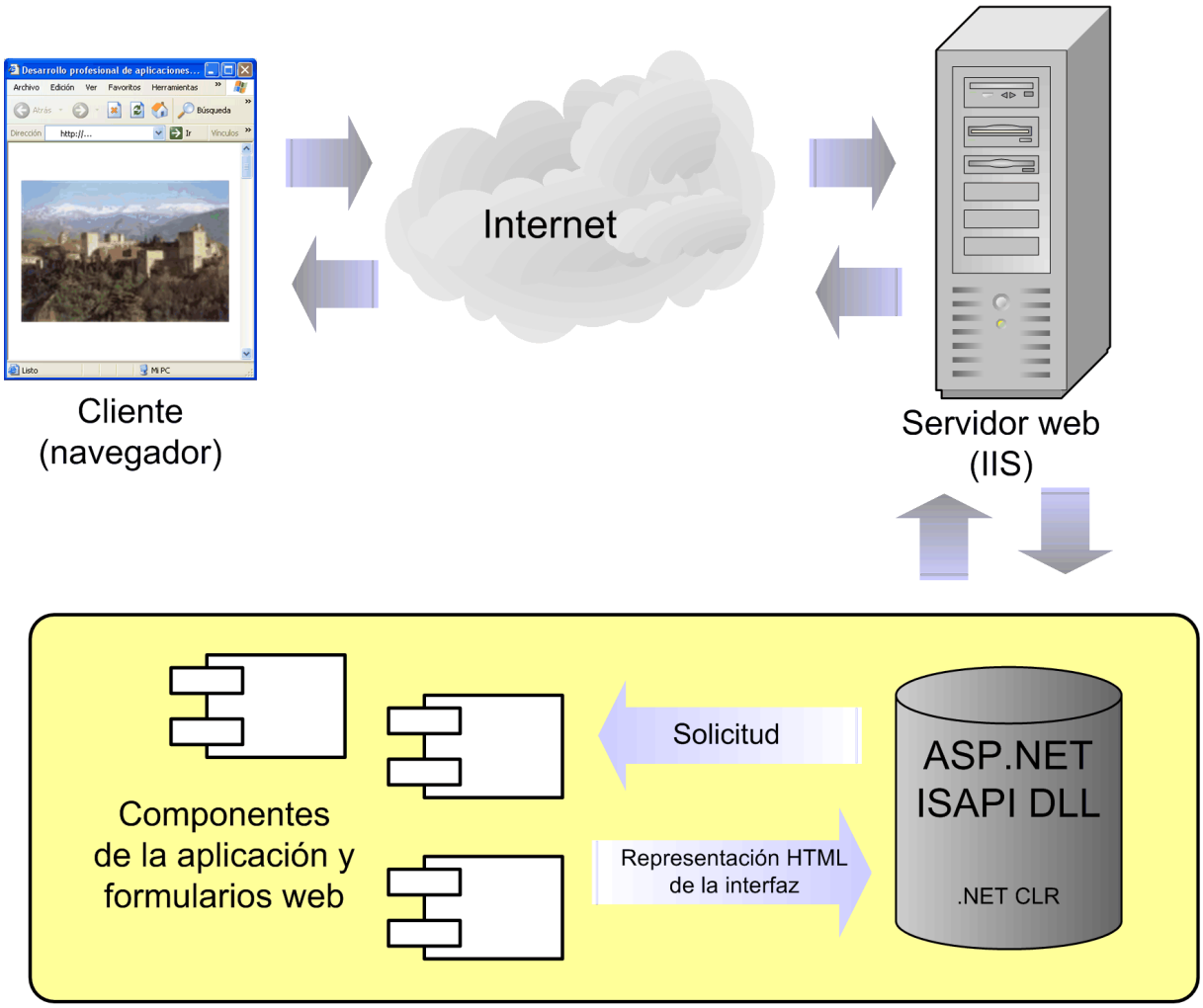
Al construir nuestras aplicaciones utilizando componentes, podemos utilizar un entorno de programación visual (como el Visual Studio .NET). Como consecuencia, al desarrollar aplicaciones web, no hemos de prestar demasiada atención al HTML de nuestras páginas ASP.NET, salvo, claro está, cuando estemos implementando los distintos componentes que utilizaremos para crear los controles de la interfaz de usuario. En otras palabras, los componentes nos permiten crear nuestra aplicación centrándonos en su lógica. Los propios componentes se encargarán de generar los fragmentos de HTML necesarios para construir la interfaz web de la aplicación.

En esta sección veremos en qué consisten y cómo se crean las páginas ASP.NET, aunque antes nos detendremos un poco en analizar cómo se ejecutan las páginas ASP.NET desde el punto de vista físico.

Ejecución de páginas ASP.NET

Los servidores HTTP pueden configurarse de tal forma que las peticiones recibidas se traten de diferentes formas en función del tipo de recurso solicitado. Básicamente, esta decisión la realiza el servidor a partir de la extensión del recurso al que intenta acceder el cliente. En el caso de las páginas ASP convencionales, cuando el usuario intenta acceder a un fichero con extensión `.asp`, el Internet Information Server delega en la biblioteca `asp.dll`, que se encarga de interpretar la página ASP. Cuando se utiliza ASP.NET, el IIS se configura de tal forma que las solicitudes recibidas relativas a ficheros con extensión `.aspx` son enviadas a la biblioteca `aspnet_isapi.dll`.

Como su propio nombre sugiere, la biblioteca `aspnet_isapi.dll` es un módulo ISAPI. Los módulos ISAPI, como vimos en el capítulo anterior, sirven para crear aplicaciones web sin que en el servidor se tengan que crear nuevos procesos cada vez que, como respuesta a una solicitud, se ha de crear dinámicamente una página web. La biblioteca encargada de la ejecución de las páginas ASP.NET (`aspnet_isapi.dll`) encapsula el CLR [*Common Language Runtime*] de la plataforma .NET. De esta forma, podemos utilizar todos los recursos de la plataforma .NET en el desarrollo de aplicaciones web. La DLL mencionada creará las instancias que sean necesarias de las clases .NET para atender las solicitudes recibidas en el servidor web.



Ejecución de páginas ASP.NET: Usando el Internet Information Server como servidor HTTP, una DLL ISAPI se encarga de que podamos aprovechar todos los recursos de la plataforma .NET en el desarrollo de aplicaciones web.

A diferencia de las páginas ASP tradicionales, las páginas ASP.NET se compilan antes de ejecutarse. La primera vez que alguien accede a una página ASP.NET, ésta se compila y se crea un fichero ejecutable que se almacena en una caché del servidor web, un *assembly* si utilizamos la terminología propia de la plataforma .NET. De esta forma, las siguientes ocasiones en las que se solicite la página, se podrá usar directamente el ejecutable. Al no tener que volver a compilar la página ASP.NET, la ejecución de ésta será más eficiente que la de una página ASP convencional.

Configuración del IIS

Para poder ejecutar páginas ASP.NET en el Internet Information Server, primero hemos de indicarle cómo ha de gestionar las peticiones recibidas relativas a ficheros con extensión `.aspx`. Para ello debemos utilizar la herramienta `aspnet_regiis` que se encuentra en el directorio donde se instala la plataforma .NET:

```
<windows>/Microsoft.NET/Framework/v1.X.XXXX
```

donde `<windows>` es el directorio donde esté instalado el sistema operativo Windows (`C:/Windows` o `C:/winNT`, por lo general) y `v1.X.XXXX` corresponde a la versión de la plataforma .NET que tengamos instalada en nuestra máquina. Puede que en nuestro ordenador existan distintas versiones instaladas de la plataforma .NET y, usualmente, escogeremos la más reciente para utilizarla en la creación de nuestras páginas ASP.NET.

Creación de páginas ASP.NET

La biblioteca de clases .NET incluye un conjunto de clases que nos serán de utilidad en la creación de las páginas ASP.NET. Entre dichas clases se encuentra una amplia gama de controles que podremos utilizar en la construcción de interfaces web para nuestras aplicaciones, controles tales como botones, cajas de texto, listas o tablas. Además, la biblioteca de clases estándar también proporciona algunos componentes que nos facilitarán realizar tareas comunes como la gestión del estado de nuestra aplicación web. Conforme vayamos avanzando en la construcción de páginas ASP.NET, iremos viendo cómo funciona cada uno de los componentes suministrados por la plataforma .NET.

Una aplicación web, generalmente, estará formada por varios formularios web. Cada uno de esos formularios lo implementaremos como una páginas ASP.NET. En la plataforma .NET, las páginas ASP.NET se construyen creando clases derivadas de la clase `System.Web.UI.Page`. Dicha clase proporciona la base sobre la que construiremos nuestras páginas ASP.NET, que implementaremos como subclases de `System.Web.UI.Page` en las que incluiremos la funcionalidad requerida por nuestras aplicaciones.

En realidad, para mantener independientes la interfaz de usuario y la lógica asociada a la aplicación, la implementación de las páginas ASP.NET la dividiremos en dos ficheros. En un fichero con extensión `.aspx` especificaremos el aspecto de nuestra interfaz, utilizando tanto etiquetas HTML estándar como etiquetas específicas para hacer referencia a los controles ASP.NET que deseemos incluir en nuestra página. En un segundo fichero, que será un fichero de código con extensión `.cs` si utilizamos en lenguaje C#, implementaremos la lógica de la aplicación.

El siguiente ejemplo muestra el aspecto que tendrá nuestro fichero `.aspx`:

```
<% @Page Language="C#" Inherits="TodayPage" Src="Today.cs" %>

<html>
<body>
   <h1 align="center">
      Hoy es <% OutputDay(); %>
   </h1>
</body>
</html>
```

El fichero anterior incluye todo lo necesario para generar dinámicamente una página web en la que incluiremos la fecha actual, que aparecerá donde está el fragmento de código delimitado por <% y %>, las mismas etiquetas que se utilizan en ASP tradicional para combinar la parte estática de la página (las etiquetas HTML) con la parte que ha de generarse dinámicamente. En esta ocasión, en vez de introducir el código necesario entre las etiquetas <% y %>, hemos optado por implementar una función auxiliar `OutputDay` que definiremos en un fichero de código aparte. El aspecto de este fichero de código, con extensión `.cs`, será el siguiente:

```
using System;
using System.Web.UI;

public class TodayPage:Page
{
  protected void OutputDay()
  {
    Response.Write(DateTime.Now.ToString("D"));
  }
}
```

Este fichero define una clase (`TodayPage`) que hereda de la clase `System.Web.UI.Page`. En nuestra clase hemos incluido un método que se encargará de generar un mensaje en el que se muestre la fecha actual. Para ello empleamos la clase `Response` que representa la respuesta de nuestra página ASP.NET y la clase `DateTime` que forma parte del vasto conjunto de clases incluidas en la biblioteca de clases de la

plataforma .NET.

Técnicamente, el fichero `.aspx` que creamos representa una clase que hereda de la clase definida en el fichero de código con extensión `.cs`, el cual, a su vez, define una subclase de `System.Web.UI.Page`. Esto explica por qué definimos en método `OutputDay()` como `protected`, para poder acceder a él desde la subclase correspondiente al fichero `.aspx`.

Para poner a disposición de los usuario la no demasiado útil aplicación web que hemos creado con los dos ficheros anteriores, sólo tenemos que copiar ambos ficheros a algún directorio al que se pueda acceder a través del IIS (el directorio raíz `wwwroot`, por ejemplo). Cuando un usuario intente acceder a nuestra aplicación web, éste sólo tendrá que introducir la ruta adecuada en la barra de direcciones de su navegador para acceder a la página `.aspx`. Al acceder por primera vez a ella, el código asociado a la página se compilará automáticamente y se generar un *assembly* en la caché del CLR encapsulado por la biblioteca `aspnet_isapi.dll` en el servidor web IIS.

Exactamente igual que en ASP, si el texto de la página ASP.NET cambia, el código se recompilará automáticamente, por lo que podemos editar libremente nuestra página en el servidor y al acceder a ella ejecutaremos siempre su versión actual (sin tener que desinstalar manualmente la versión antigua de la página e instalar la versión nueva en el servidor web). Si, entre solicitud y solicitud, el texto de la página no cambia, las nuevas solicitudes se atenderán utilizando el código previamente compilado que se halla en la caché del servidor web, con lo cual se mejora notablemente la eficiencia de las aplicaciones web respecto a versiones previas de ASP.

Aunque en el ejemplo anterior hayamos incluido un fragmento de código entre las etiquetas `<%` y `%>` dentro de la página `.aspx`, igual que se hacía con ASP, lo usual es que aprovechemos los recursos que nos ofrece la plataforma .NET para construir aplicaciones web de la misma forma que se construyen las aplicaciones para Windows en los entornos de programación visual. Como sucede en cualquiera de estos entornos, la interfaz web de nuestra aplicación la crearemos utilizando controles predefinidos a los cuales asociaremos un comportamiento específico definiendo su respuesta ante distintos eventos. En la siguiente sección de este capítulo veremos cómo se utilizan controles y eventos en las páginas ASP.NET.

Uso de controles en ASP.NET

Utilizando ASP.NET, las interfaces web se construyen utilizando controles predefinidos. Estos controles proporcionan un modelo orientado a objetos de los formularios web, similar en cierto modo al definido por JavaScript. Sin embargo, a diferencia de JavaScript, en ASP.NET no trabajaremos directamente sobre los objetos que representan las etiquetas HTML del documento que visualiza el usuario en su navegador. Lo que haremos será utilizar controles definidos en la biblioteca de clases .NET.

Al emplear controles predefinidos en nuestra interfaz de usuario en vez de especificar directamente las etiquetas HTML de los formularios web, las páginas ASP.NET se convierten en meras colecciones de controles. Desde el punto de vista del programador, cada control será un objeto miembro de la clase que representa la página (aquélla que hereda de `System.Web.UI.Page`). Cuando deseemos asociar a nuestra página un comportamiento dinámico, lo único que tendremos que hacer es asociar a los distintos controles los manejadores de eventos que se encargarán de implementar la funcionalidad de la página, exactamente igual que cuando se crea una interfaz gráfica para Windows utilizando un entorno de programación visual. Además, en nuestras páginas ASP.NET podremos incluir nuestros propios controles, los cuales crearemos a partir de los controles existentes por derivación (creando clases nuevas que hereden de las clases ya definidas) o por composición (encapsulando una página completa para luego utilizarla como un control más dentro de otras páginas).

En definitiva, en vez de utilizar un intérprete que se encargue de ejecutar los fragmentos de código desperdigados por nuestras páginas web, lo que haremos será utilizar un sistema basado en eventos para generar dinámicamente el contenido de las páginas que se le muestran al usuario. Esto nos permitirá acceder a todas las características de C# y de la plataforma .NET para construir aplicaciones web flexibles, modulares y fáciles de mantener.

Además, aparte de proporcionar un modelo orientado a objetos de la aplicación que evita el código "spaghetti" típico de ASP, los controles web constituyen una capa intermedia entre el código de la aplicación y la interfaz de usuario. Entre las ventajas que proporciona este hecho, destaca la compatibilidad automática de las aplicaciones web con distintos tipos de navegadores. La implementación de los distintos controles se encargará de aprovechar la funcionalidad de los navegadores modernos (como JavaScript o HTML dinámico), sin que esto suponga que nuestra aplicación deje de funcionar en navegadores más antiguos (los que se limitan a soportar HTML 3.2).

Dentro de las páginas ASP.NET, los controles se indican en el fichero `.aspx` utilizando etiquetas de la forma `<asp:... />`. Para implementar el ejemplo de la sección anterior utilizando controles, lo único que tenemos que hacer es crear una página ASP.NET con una etiqueta (componente `asp:Label`). Si estamos utilizando Visual Studio .NET como entorno

de desarrollo, creamos un nuevo proyecto de tipo *Aplicación Web ASP.NET*. Al aparecernos un formulario web vacío, modificamos su propiedad `pageLayout` para utilizar el estilo `FlowLayout` típico de las páginas web, en las que los controles no se colocan en coordenadas fijas (a diferencia de la forma habitual de trabajar con formularios Windows). Para que el contenido de la página aparezca centrado, utilizamos uno de los botones de la barra de herramientas igual que si estuviésemos trabajando con un procesador de textos. Finalmente, añadimos una etiqueta [`Label`] que obtenemos del cajón *Web Forms* del cuadro de herramientas. Una vez que tenemos el control en nuestro formulario web, lo normal es que modifiquemos su identificador (propiedad `(ID)`) y el texto que muestra en el formulario (propiedad `Text`). Como resultado de este proceso obtenemos el siguiente fichero `.aspx`, en el que aparece nuestro control como una etiqueta `asp:Label`:

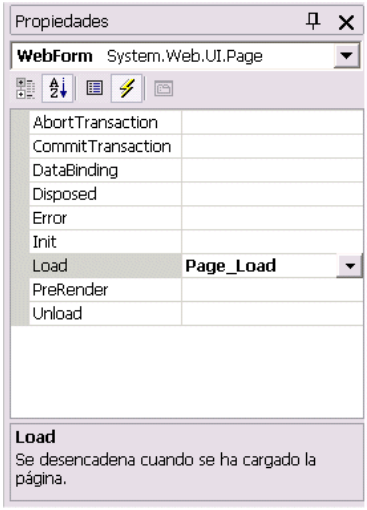
```
<%@ Page language="c#"
         Codebehind="WebControlExample.aspx.cs"
         AutoEventWireup="false"
         Inherits="WebControlExample.WebForm" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>WebForm</title>
    <meta name="GENERATOR" Content="Microsoft Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" Content="C#">
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <P align="center">
        <asp:Label id="LabelFecha" runat="server">Hoy es...</asp:Label>
      </P>
    </form>
  </body>
</HTML>
```

El fichero mostrado define los controles existentes en nuestra interfaz de usuario e incluye información adicional acerca de cómo ha de procesarse la página ASP.NET. De todo lo que ha generado el Visual Studio, el detalle más significativo es la inclusión del atributo `runat="server"` en todos aquellos componentes de nuestra interfaz a los que podemos asociarles manejadores de eventos que implementen la lógica de la aplicación como respuesta a las acciones del usuario. Dichos manejadores de eventos se ejecutarán siempre en el servidor y serán los encargados de que nuestras aplicaciones sean algo más que páginas estáticas en las que se muestra información.

IMPORTANTE

Todos los controles en una página ASP.NET deben estar dentro de una etiqueta `<form>` con el atributo `runat="server"`. Además, ASP.NET requiere que todos los elementos HTML estén correctamente anidados y cerrados (como sucede en XML).

Si deseamos lograr el mismo efecto que conseguíamos en la sección anterior al introducir un fragmento de código dentro de nuestro fichero `.aspx`, podemos especificar la respuesta de nuestro formulario al evento `Page_Load`, que se produce cuando el usuario accede a la página ASP.NET desde su navegador. En un entorno de programación visual como Visual Studio .NET, sólo tenemos que buscar el evento asociado al componente `WebForm` y hacer doble click sobre él.



El evento Page_Load nos permite especificar acciones que deseamos realizar antes de mostrarle la página al usuario.

Acto seguido, implementamos la lógica de la aplicación como respuesta al evento dentro del fichero de código C# asociado a la página ASP.NET. En el ejemplo que nos ocupa sólo tenemos que escribir una línea de código dentro del método creado automáticamente por el Visual Studio .NET para gestionar el evento `Page_Load`:

```
private void Page_Load(object sender, System.EventArgs e)
{
  LabelFecha.Text = "Hoy es " + DateTime.Now.ToString("D");
}
```

En realidad, el código de nuestra aplicación es algo más complejo. Aunque nosotros sólo nos preocupemos de escribir el código que muestra la fecha actual, el entorno de desarrollo se encarga de hacer el resto del trabajo por nosotros. El código completo asociado a nuestro formulario, eliminando comentarios y sentencias `using`, será el mostrado a continuación:

```
public class WebForm : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.Label LabelFecha;

  private void Page_Load(object sender, System.EventArgs e)
  {
    LabelFecha.Text = "Hoy es " + DateTime.Now.ToString("D");
  }

  #region Código generado por el Diseñador de Web Forms

  override protected void OnInit(EventArgs e)
  {
    InitializeComponent();
    base.OnInit(e);
  }

  private void InitializeComponent()
  {
    this.Load += new System.EventHandler(this.Page_Load);
  }

  #endregion
}
```

La región de código que no hemos implementado nosotros se encarga, básicamente, de inicializar los componentes de nuestra página con las propiedades que fijamos en el diseñador de formularios (aquéllas que aparecen en el fichero `.aspx`) y, además, les asigna los manejadores de eventos que hemos implementado a los distintos eventos asociados a los componentes de nuestra interfaz; esto es, enlaza nuestro código con los componentes de la interfaz. En el ejemplo mostrado, el único evento para el cual nuestra aplicación tiene una respuesta específica es el evento `Load` del formulario web.

El ejemplo anterior puede conducir a conclusiones erróneas si no se analiza con detalle. Puede parecer que el formulario web construido utilizando controles resulta excesivamente complejo en comparación con la implementación equivalente que vimos en la sección anterior de este capítulo. No obstante, la organización modular de la página ASP.NET simplifica el desarrollo de interfaces complejos y su posterior mantenimiento. Resulta mucho más sencillo modificar el código incluido en un método de una clase que el código disperso entre etiquetas propias de la interfaz de usuario. Además, la comprobación del correcto funcionamiento del código implementado también será más sencilla cuando sólo incluimos en el fichero `.aspx` los controles de la interfaz de usuario e implementamos su funcionalidad aparte. Por otra parte, la complejidad extra que supone utilizar los controles no repercute en el trabajo del programador, ya que el entorno de desarrollo se encarga de generar automáticamente el código necesario para enlazar los controles con el código implementado por el programador.

Como hemos visto, al usar controles en nuestra interfaz web, la lógica de la aplicación se implementa como respuesta de la interfaz de usuario a los distintos eventos que puedan producirse, exactamente igual que en cualquier entorno de programación visual para

Windows. El servidor web será el encargado de interpretar las etiquetas correspondientes a los controles ASP.NET y visualizarlos correctamente en el navegador del usuario, para lo cual tendrá que generar el código HTML que resulte más apropiado.

Una vez que hemos visto el funcionamiento de los controles en las páginas ASP.NET, podemos pasar a analizar los distintos tipos de controles que podemos incluir en nuestras páginas web. En la plataforma .NET, existen tres tipos de controles predefinidos que se pueden utilizar en las páginas ASP.NET:

- Los controles HTML representan etiquetas HTML tradicionales y funcionan de forma similar a los objetos utilizados en JavaScript para manipular dinámicamente el contenido de una página web.

- Los controles web son los controles asociados a las etiquetas específicas de ASP.NET y facilitan el desarrollo de interfaces web utilizando un entorno de programación visual como Visual Studio .NET.

- Por último, existe una serie de controles de validación que se emplean para validar las entradas introducidas por el usuario de una forma relativamente cómoda (aunque no siempre resulta la más adecuada desde el punto de vista del diseño de la interfaz de usuario).

Aparte de estos controles, que ya vienen preparados para su utilización, el programador puede crear sus propios controles para simplificar la creación de interfaces consistentes y evitar la duplicación innecesaria de código en distintas partes de una aplicación web.

En los siguientes apartados de esta sección analizaremos con algo más de detalle los distintos tipos de controles que se pueden utilizar en las páginas ASP.NET y veremos cómo podemos crear nuestros propios controles fácilmente.

Controles HTML

Por defecto, las etiquetas HTML incluidas en una página ASP.NET se tratan como texto en el servidor web y se envían tal cual al cliente. No obstante, en determinadas ocasiones puede que nos interese manipular su contenido desde nuestra aplicación, que se ejecuta en el servidor. Para que las etiquetas HTML sean programables en el servidor, sólo tenemos que añadirles el atributo `runat="server"`.

En el siguiente ejemplo veremos cómo podemos hacer que un enlace HTML apunte dinámicamente a la URL que nos convenga en cada momento. En HTML, los enlaces se marcan con la etiqueta `<A>`. Para poder modificar las propiedades del enlace en HTML, sólo tenemos que añadir el atributo `runat="server"` a la etiqueta estándar `<A>` y asociarle al enlace un identificador adecuado. El fichero `.aspx` de nuestra página ASP.NET quedaría como se muestra a continuación:

```
<html>
  ...
  <body>
    <form id="HTMLControl" method="post" runat="server">
      <a id="enlace" runat="server">¡Visite nuestra página!</a>
    </form>
  </body>
<html>
```

Una vez que hemos marcado el enlace con el atributo `runat="server"`, podemos modificar sus propiedades accediendo a él a través del identificador que le hayamos asociado. En ASP.NET, los enlaces HTML de una página se representan mediante el control `HtmlAnchor`. Una de las propiedades de este control (`HRef`) indica la URL a la que apunta el enlace, por lo que sólo tenemos que establecer un valor adecuado para esta propiedad en el código asociado a alguno de los eventos de la página ASP.NET. El fichero de código resultante tendría el siguiente aspecto:

```
public class HTMLControl : System.Web.UI.Page
{
  protected System.Web.UI.HtmlControls.HtmlAnchor enlace;

  private void Page_Load(object sender, System.EventArgs e)
  {
    enlace.HRef = "http://csharp.ikor.org/";
  }

  override protected void OnInit(EventArgs e)
  {
    this.Load += new System.EventHandler(this.Page_Load);
    base.OnInit(e);
  }
}
```

Si utilizamos Visual Studio .NET, para poder manipular un control HTML en el servidor sólo tenemos que seleccionar la opción "Ejecutar como control del servidor" del menú contextual asociado a la etiqueta HTML en el diseñador de formularios web. Esto hace que se añada al atributo `runat="server"` a la etiqueta en el fichero `.aspx` y que se incluya la declaración correspondiente en la clase que define nuestro formulario, con lo cual ya podemos programar el comportamiento del control HTML en función de nuestras necesidades accediendo a él como un miembro más de la clase que representa la página ASP.NET.

La biblioteca de clases de la plataforma .NET incluye una gama bastante completa de componentes que encapsulan las distintas etiquetas que pueden aparecer en un documento HTML. Dichos componentes se encuentra en el espacio de nombres `System.Web.UI.HtmlControls`.

Las etiquetas más comunes en HTML tienen su control HTML equivalente en ASP.NET. Este es el caso de los enlaces o las imágenes en HTML, los cuales se representan como objetos de tipo `HtmlAnchor` y `HtmlImage` en las páginas ASP.NET, respectivamente. Si, por cualquier motivo, nos encontramos con que no existe un control HTML específico para representar una etiqueta HTML determinada, esto no impide que podamos manipularla desde nuestra aplicación web. Existe un control genérico, denominado `HtmlGenericControl`.

La siguiente tabla resume los controles HTML disponibles en ASP.NET, las etiquetas a las que corresponden en HTML estándar y una breve descripción de su función en la creación de páginas web:

Control HTML	Etiqueta HTML	Descripción
`HtmlAnchor`	`<a>`	Enlace
`HtmlButton`	`<button>`	Botón
`HtmlForm`	`<form>`	Formulario
`HtmlGenericControl`		Cualquier elemento HTML para el cual no existe un control HTML específico
`HtmlImage`	`<image>`	Imagen
`HtmlInput...`	`<input type="...">`	Distintos tipos de entradas en un formulario HTML: botones (`button`, `submit` y `reset`), texto (`text` y `password`), opciones (`checkbox` y `radio`), imágenes (`image`), ficheros (`file`) y entradas ocultas (`hidden`).
`HtmlSelect`	`<select>`	Lista de opciones en un formulario
`HtmlTable...`	`<table>` `<tr>` `<td>`	Tablas, filas y celdas
`HtmlTextArea`	`<textarea>`	Texto en un formulario

Controles web

La principal aportación de ASP.NET a la creación de interfaces web es la inclusión de controles específicos que aíslan al programador del HTML generado para presentar el formulario al usuario de la aplicación. Como ya mencionamos anteriormente, estos controles

permiten desarrollar aplicaciones web compatibles con distintos navegadores y facilitan la tarea del programador al ofrecerle un modelo de programación basado en eventos.

En el fichero `.aspx` asociado a una página ASP.NET, los controles web se incluyen utilizando etiquetas específicas de la forma `<asp:... />`. La sintaxis general de una etiqueta ASP.NET es de la siguiente forma:

```
<asp:control id="identificador" runat="server" />
```

Dada una etiqueta como la anterior:

- `control` variará en función del tipo de control que el programador decida incluir en su página ASP.NET. La etiqueta concreta de la forma `asp:... />` será diferente para mostrar un texto (`asp:Label`), un botón (`asp:Button`), una lista convencional (`asp:ListBox`) o una lista desplegable (`asp:DropDownList`), por mencionar algunos ejemplos.

- `identificador` especifica el identificador que le asociamos a la variable mediante la cual accederemos al control desde el código de nuestra aplicación.

- Finalmente, la inclusión del atributo `runat="server"` es necesaria para indicar que el programador puede manipular la etiqueta ASP.NET desde el servidor implementando manejadores de eventos para los distintos eventos a los que pueda responder el control representado por la etiqueta ASP.NET.

Igual que sucedía con las etiquetas HTML, en la biblioteca de clases de la plataforma .NET existen componentes que nos permiten manipular dichas las etiquetas ASP.NET desde el código de la aplicación. Dichos componentes encapsulan a las distintas etiquetas ASP.NET y se encuentran en el espacio de nombres `System.Web.UI.WebControls`. De hecho, todos ellos derivan de la clase `WebControl`, que se encuentra en el mismo espacio de nombres.

Uso de controles ASP.NET

Para mostrar el uso de los controles ASP.NET, podemos crear una aplicación web ASP.NET desde el Visual Studio .NET. Dicha aplicación contiene, por defecto, un formulario web vacío al que denominamos `WebControl.aspx`. A continuación, le añadimos un botón al formulario utilizando el control `Button` que aparece en la sección *Web Forms* del *Cuadro de herramientas* de Visual Studio .NET.

Al añadir el botón, en el fichero `.aspx` de la página ASP.NET aparece algo similar a lo siguiente:

Uso de controles ASP.NET

```
<form id="WebControl" method="post" runat="server">
  <asp:Button id="Button" runat="server" Text="Pulse el botón">
  </asp:Button>
</form>
```

Así mismo, en el fichero de código asociado a la página aparece una declaración de la forma:

```
protected System.Web.UI.WebControls.Button Button;
```

Ahora sólo nos falta añadirle algo de funcionalidad a nuestra aplicación. Para lograrlo, buscamos los eventos a los cuales ha de reaccionar nuestro formulario web e implementamos los manejadores de eventos correspondientes. Por ejemplo, desde el mismo diseñador de formularios web del Visual Studio .NET, podemos hacer doble click sobre el botón para especificar la respuesta de este control al evento que se produce cuando el usuario pulsa el botón desde su navegador web:

```
private void Button_Click(object sender, System.EventArgs e)
{
  Button.Text = "Ha pulsado el botón";
}
```

Obviamente, la lógica asociada a los eventos de una aplicación real será bastante más compleja (y útil), si bien la forma de trabajar del programador será siempre la misma: implementar la respuesta de la aplicación frente a aquellos eventos que sean relevantes para lograr la funcionalidad deseada.

Como se puede comprobar, el desarrollo de aplicaciones web con controles ASP.NET es completamente análogo al desarrollo de aplicaciones para Windows. Sólo tenemos que seleccionar los controles adecuados para nuestra interfaz e implementar la respuesta de nuestra aplicación a los eventos que deseemos controlar.

Aunque la forma de desarrollar la interfaz de usuario sea la misma cuando se utilizan formularios web ASP.NET y cuando se emplean formularios para Windows, los controles que se pueden incluir cambian en función del tipo de interfaz que deseemos crear. En el caso de los formularios web, podemos utilizar cualquiera de los controles definidos en el espacio de nombres `System.Web.UI.WebControls` de la biblioteca de clases de la plataforma .NET. La mayoría de ellos corresponden a los controles típicos que uno podría esperar encontrarse en cualquier interfaz gráfica actual, si bien también existen otros muy útiles para el programador en la creación de interfaces más sofisticadas (como es el caso de los componentes `asp:Repeater` y `asp:DataGrid`, que mencionaremos más adelante en este mismo capítulo).

La tabla que ofrecemos a continuación resume, a modo de guía, cuáles son los controles ASP.NET y menciona su función en la creación de interfaces web con ASP.NET:

Control	Descripción
`AdRotator`	Muestra una secuencia de imágenes (a modo de banner)
`Button`	Botón estándar
`Calendar`	Calendario mensual
`CheckBox`	Caja de comprobación (como en los formularios Windows)
`CheckBoxList`	Grupo de cajas de comprobación
`DataGrid`	Rejilla de datos
`DataList`	Muestra una lista utilizando plantillas (*templates*)
`DropDownList`	Lista desplegable
`HyperLink`	Enlace
`Image`	Imagen
`ImageButton`	Botón dibujado con una imagen
`Label`	Etiqueta de texto estático
`LinkButton`	Botón con forma de enlace
`ListBox`	Lista (como en los formularios Windows)
`Literal`	Texto estático (similar a `Label`)
`Panel`	Contenedor en el que se pueden colocar otros controles
`PlaceHolder`	Reserva espacio para controles añadidos dinámicamente
`RadioButton`	Botón de radio (como en los formularios Windows)
`RadioButtonList`	Grupo de botones de radio
`Repeater`	Permite mostrar listas de controles
`Table`	Tabla
`TextBox`	Caja de edición
`Xml`	Muestra un fichero XML o el resultado de una transformación XSL

Como se puede apreciar a partir de la lista de controles de la tabla, la gama de controles estándar ya disponibles es bastante amplia, lo suficiente como para poder construir interfaces gráficos estándar sin tener que preocuparnos demasiado de los detalles de implementación de los controles de la interfaz de usuario.

Controles de validación

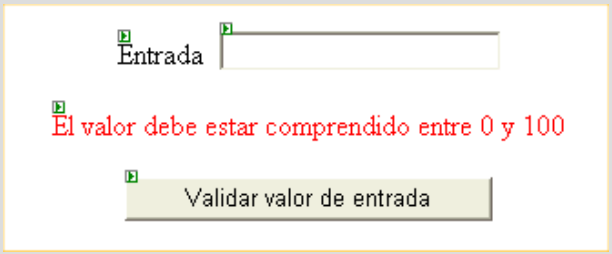
Para finalizar nuestro recorrido por los componentes predefinidos en la biblioteca de clases de la plataforma .NET para la creación de interfaces web, mostraremos un tercer grupo de componentes que puede utilizarse para imponer ciertas restricciones a los datos introducidos por el usuario: los controles de validación.

Los controles de validación son un tipo especial de controles ASP.NET, por lo que también están incluidos en el espacio de nombres `System.Web.UI.WebControls`. Estos controles se enlazan a controles ASP.NET de los descritos en el apartado anterior para validar las entradas de un formulario web. Cuando el usuario rellena los datos de un formulario y alguno de los datos introducidos no verifica la restricción impuesta por el control de validación, este control se encarga de mostrarle un mensaje de error al usuario.

La validación de las entradas de un formulario web se realiza automáticamente cuando se pulsa un botón, ya tenga éste la forma de un botón estándar (`Button`), de una imagen (`ImageButton`) o de un enlace (`LinkButton`). No obstante, se puede desactivar esta validación si establecemos a `false` la propiedad `CausesValidation` del botón correspondiente. Igualmente, se puede forzar la validación manualmente en el momento que nos interese mediante el método `Validate` asociado a la página ASP.NET.

Ejemplo de uso de controles de validación

Podemos forzar a que una entrada determinada esté dentro de un rango válido de valores con un control de validación del tipo `RangeValidator`:



Para ver cómo funcionan los controles de validación, podemos crear una aplicación web ASP.NET con un formulario como el mostrado en la imagen de arriba, en el cual incluimos una etiqueta (`Label`), una caja de texto (`TextBox`), un botón (`Button`) y un control de validación (`RangeValidator`), todos ellos controles incluidos en la pestaña *Web Forms* del cuadro de herramientas de Visual Studio .NET.

Para que nuestro formulario web utilice el control de validación, sólo tenemos que establecer

Ejemplo de uso de controles de validación

la propiedad `ControlToValidate` de este control para que haga referencia al `TextBox` que usamos como entrada. Mediante la propiedad `ErrorMessage` especificamos el mensaje de error que se mostrará (en rojo) cuando el valor introducido por el usuario no cumpla la condición impuesta por el control de validación. Finalmente, esta condición la indicamos utilizando las propiedades específicas del control de validación que hemos escogido: el tipo de dato (`Type=Integer`) y el rango de valores permitidos (entre `MinimumValue` y `MaximumValue`).

Cuando los datos introducidos por el usuario son válidos, la aplicación prosigue su ejecución de la forma habitual. Cuando no se verifica alguna condición de validación, se muestra el mensaje de error asociado al control de validación y se le vuelve a pedir al usuario que introduzca correctamente los datos de entrada. Obviamente, la validación la podríamos haber realizado implementando la respuesta de nuestra aplicación a los eventos asociados utilizando controles web estándar, aunque de esta forma, ASP.NET se encargará de generar automáticamente las rutinas de validación y automatizar, al menos en parte, algunas de las comprobaciones más comunes que se realizan al leer datos de entrada en cualquier aplicación.

Por cuestiones de seguridad, los controles de validación siempre validan los datos de entrada en el servidor, con el objetivo de que un cliente malintencionado no pueda saltarse algunas comprobaciones acerca de la validez de los datos. Aun cuando la comprobación en el servidor siempre se realiza, si el navegador del usuario acepta HTML dinámico, la validación también se efectúa en el cliente. Al realizar la comprobación antes de enviar los datos al servidor, se consigue disminuir la carga del servidoral ahorrarnos un viaje de ida y vuelta cuando los datos de entrada no verifican las restricciones impuestas por los controles de validación.

A pesar de su evidente utilidad, no conviene abusar demasiado del uso de controles de validación porque, desde el punto de vista del diseño de interfaces de usuario, su implementación dista de ser adecuada. Cuando el usuario introduce algún dato erróneo, el error que se ha producido no siempre llama su atención. De hecho, lo único que sucede en la interfaz de usuario es la aparición de una etiqueta, usualmente en rojo, que indica en qué consiste el error. Desafortunadamente, esta etiqueta puede que se encuentre bastante lejos del punto en el cual el usuario tiene fija su atención. Por tanto, para el usuario el error pasa desapercibido y no siempre le resulta obvio darse cuenta de por qué se le vuelve a presentar el mismo formulario que acaba de rellenar. Para evitar situaciones como la descrita resulta aconsejable utilizar el control `asp:ValidationSummary`, que resume en una lista los errores de validación que se hayan podido producir al rellenar un formulario web.

Los controles de validación predefinidos en la biblioteca de clases de la plataforma .NET se encuentran, como mencionamos al comienzo de esta sección, en el espacio de nombres `System.Web.UI.WebControls`. Todos los controles de validación derivan de la clase `System.Web.UI.WebControls.BaseValidator`. En la siguiente tabla se recoge cuáles son y qué utilidad tienen:

Control de validación	Descripción
`CompareValidator`	Compara el valor de una entrada con el de otra o un valor fijo.
`CustomValidator`	Permite implementar un método cualquiera que maneje la validación del valor introducido.
`RangeValidator`	Comprueba que la entrada esté entre dos valores dados.
`RegularExpressionValidator`	Valida el valor de acuerdo a un patrón establecido como una expresión regular.
`RequiredFieldValidator`	Hace que un valor de entrada sea obligatorio.

Controles creados por el usuario

En las secciones anteriores de este capítulo hemos visto cuáles son los controles predefinidos que podemos utilizar en la creación de interfaces web con ASP.NET. En esta, veremos cómo podemos crear nuestros propios controles.

En ASP.NET, el programador puede crear sus propios controles de dos formas diferentes:

- **Por composición:** A partir de una colección de controles ya existentes, el programador decide cómo han de visualizarse conjuntamente. En vez de tener que repetir la disposición del conjunto de controles cada vez que hayan de utilizarse en la aplicación, la colección de controles se encapsula en un único control. Cada vez que se desee, se puede añadir el control a un formulario web para mostrar el conjunto completo de controles a los que encapsula.

- **Por derivación:** Igual que en cualquier otro entorno de programación orientado a objetos, los controles se implementan como clases. Estas clases heredarán, directa o indirectamente, de la clase base de todos los controles web: la clase `System.Web.UI.WebControl` (la cual está definida en la biblioteca `System.Web.dll`).

En esta sección nos centraremos en la primera de las alternativas, la creación de controles de usuario por composición, porque su uso es muy habitual en la creación de aplicaciones web en ASP.NET. Como veremos en los próximos capítulos del libro, los controles ASP.NET creados de esta manera son los que nos permiten construir aplicaciones modulares que sean fácilmente mantenibles y extensibles.

Dado que los controles definidos por composición se utilizan como bloque básico en la construcción de aplicaciones web con ASP.NET, lo usual es que dichos controles se adapten a las necesidades de cada aplicación particular. Esto, obviamente, no impide que podamos crear controles reutilizables en distintas aplicaciones si hemos de realizar el mismo tipo de tareas en diferentes proyectos de desarrollo.

Supongamos ahora que deseamos crear una aplicación web en la cual se almacene información de contacto relativa a diferentes personas, como podría ser el caso de las entradas de una sencilla agenda, un cliente de correo electrónico o una completa aplicación de gestión en la que tuviésemos que mantener información acerca de clientes o proveedores. En cualquiera de las situaciones mencionadas, en nuestra aplicación existirá una clase que encapsule los datos relativos a distintas personas. La implementación de dicha clase podría tener un aspecto similar, aunque nunca igual, al que se muestra a continuación:

```
public class Contact
{
  public string Name;        // Nombre
  public string EMail;       // Correo electrónico
  public string Telephone;   // Teléfono
  public string Mobile;      // Teléfono móvil
  public string Fax;         // Fax
  public string Address;     // Dirección
  public string Comments;    // Anotaciones
  public string ImageURL;    // URL de su fotografía
}
```

Encapsulación

Cualquier purista de la programación orientada a objetos se estremecería al ver un fragmento de código como el anterior, y tendría razones para ello. La idea básica de la utilización de objetos es encapsular la implementación de su interfaz, algo que no se consigue si todas las variables de instancia son públicas. Siempre resulta aconsejable mantener los datos en variables privadas las de una clase y acceder a ellos mediante métodos.

El lenguaje de programación C# nos ofrece un mecanismo muy cómodo para conseguir el mismo resultado: el uso de propiedades. Las propiedades nos permiten seguir utilizando la clase de la misma forma y, al mismo tiempo, mantener la encapsulación de los datos. Si decidimos emplear propiedades, la implementación de la clase anterior resulta algo más extensa. Este es el único motivo por el que, inicialmente, decidimos utilizar variables públicas en el fragmento de código anterior.

Por tanto, cada variable de instancia de las que antes habíamos declarado públicas debería, al menos, convertirse en un fragmento de código como el siguiente:

```
...
private string _variable;

public string Variable {
  get { return _variable; }
  set { _variable = value; }
}
...
```

Una vez que tenemos clases que encapsulan los datos con los que nuestra aplicación ha de trabajar, hemos de crear una interfaz de usuario adecuada para que el usuario de nuestra aplicación pueda trabajar con los datos.

Clases como la anterior se denominan habitualmente clases modelo. Este apelativo proviene del hecho de que estas clases son las que modelan el dominio del problema que nuestra aplicación pretende resolver (la representación de datos de contacto en nuestro ejemplo). Éstas son las clases que suelen aparecer en un diagrama de clases UML o en el modelo de datos de una base de datos.

Para construir la interfaz de usuario asociada a nuestra clase modelo, lo que haremos será crear vistas que nos permitan mostrar de distintas formas los datos que encapsula la clase modelo. Por ejemplo, puede que nos interese mostrar los datos de contacto de una persona en una tabla como la siguiente:

Nombre	Nombre del contacto	
E-mail	mailbox@domain	
Teléfono	999 999 999	
Móvil	999 999 999	
Fax	999 999 999	
Dirección	Calle Localidad CP Provincia	
Comentarios		

Presentación visual de los datos de contacto de una persona

La presentación visual de los datos de contacto requiere la utilización de distintos controles, como las etiquetas que nos permitirán mostrar los datos concretos del contacto o la imagen que utilizaremos para visualizar la fotografía de nuestro contacto. Si en nuestra aplicación tuviésemos que mostrar la información de contacto en diferentes situaciones, resultaría bastante tedioso tener que copiar la tabla anterior y repetir todo el código necesario para rellenarla con los datos particulares de una persona. Aun pudiendo usar copiar y pegar, esta estrategia no resulta demasiado razonable. ¿Qué sucedería entonces si hemos de cambiar algún detalle de la presentación visual de los datos de contacto? Tendríamos que buscar todos los sitios de nuestra aplicación en los que aparezca una tabla como la anterior y realizar la misma modificación en diferentes lugares. Esto no solamente resulta incómodo, sino una situación así es bastante proclive a que, al final, cada vez que mostremos los datos de una persona lo hagamos de una forma ligeramente diferente y el comportamiento de nuestra aplicación no sea todo lo homogéneo que debería ser. En otras palabras, olvidar la realización de las modificaciones en alguna de las apariciones del fragmento duplicado provoca errores. Además, todos sabemos que la existencia de código duplicado influye negativamente en la mantenibilidad de nuestra aplicación y, a la larga, en su calidad.

Por suerte, ASP.NET incluye el mecanismo adecuado para evitar situaciones como las descritas en el párrafo anterior: los controles de usuario creados por composición. Un conjunto de controles como el que utilizamos para mostrar los datos de contacto de una persona se puede encapsular en un único control que después emplearemos en la creación de nuestros formularios web. El mecanismo utilizado es análogo a los `frames` de Delphi o C++Builder.

Para encapsular un conjunto de controles en un único control, lo único que tenemos que hacer es crear un "control de usuario Web", tal como aparece traducido en Visual Studio .NET el término inglés *web user control*, aunque quizá sería más correcto decir "control web de usuario". En realidad, un control de este tipo se parece más a un formulario web que a un control de los que existen en el "cuadro de herramientas" (la discutible traducción de *toolbox* en Visual Studio).

Un control web de usuario nos permite definir el aspecto visual conjunto de una colección de controles a los que encapsula. El control es similar a una página ASP.NET salvo que hereda de la clase `System.Web.UI.UserControl` en vez de hacerlo de `System.Web.UI.Page`.

Por convención, los controles web de usuario se almacenan en ficheros con extensión `.ascx` (en lugar de la extensión `.aspx` de los formularios ASP.NET). Dicho fichero será el que se incluya luego dentro de una página ASP.NET, por lo que en él no pueden figurar las etiquetas `<html>`, `<head>`, `<body>`, `<form>` y `<!DOCTYPE>`, las cuales sí aparecen en un formulario ASP.NET.

Teniendo en cuenta lo anterior, podemos crear un control web de usuario para mostrar los datos de contacto de una persona. Dicho control se denominará `ContactViewer` porque nos permite visualizar los datos de contacto pero no modificarlos y se creará de forma análoga a como se crean los formularios ASP.NET.

En primer lugar, creamos un fichero con extensión `.ascx` que contenga los controles de la interfaz de usuario que nuestro control ha de encapsular. Dicho fichero se denominará `ContactViewer.ascx` y tendrá el siguiente aspecto:

```
<%@ Control Language="c#"
            AutoEventWireup="false"
            Codebehind="ContactViewer.ascx.cs"
            Inherits="WebMail.ContactViewer"%>
<DIV align="center">
  <TABLE id="TableContact" width="90%" border="0">
    <TR>
      <TD width="100" bgColor="#cccccc">
          Nombre
      </TD>
      <TD bgColor="#e0e0e0">
        <asp:Label id="LabelName" runat="server">
          Nombre del contacto
        </asp:Label>
      </TD>
      <TD vAlign="middle" align="center" width="20%" rowSpan="6">
        <asp:Image id="ImagePhoto" runat="server" Height="200px">
        </asp:Image>
      </TD>
    </TR>
    <TR>
      <TD bgColor="#cccccc">
          E-mail
      </TD>
      <TD bgColor="#e0e0e0">
        <asp:Label id="LabelEMail" runat="server">
           mailbox@domain
        </asp:Label>
      </TD>
    </TR>
    <!-- Lo mismo para los demás datos del contacto -->
    ...
    <TR>
      <TD colSpan="3">
        <P>
          <asp:Label id="LabelComments" runat="server">
            Comentarios
          </asp:Label>
        </P>
      </TD>
    </TR>
  </TABLE>
</DIV>
```

Como se puede apreciar, su construcción es similar al de un formulario ASP.NET. Sólo cambia la directiva que aparece al comienzo del fichero. En vez de utilizar la directiva `<%@ Page...` propia de los formularios ASP.NET, se emplea la directiva `<%@ Control..` específica para la creación de controles por composición. Aparte de eso y de la ausencia de algunas etiquetas (`<html>`, `<head>`, `<body>`, `<form>` y `<!DOCTYPE>`), no existe ninguna otra diferencia entre un fichero `.ascx` y un fichero `.aspx`.

Igual que sucedía en las páginas ASP.NET, al crear un control separaremos la lógica de la interfaz implementando el código asociado al control en un fichero de código aparte. A este fichero hace referencia el atributo `Codebehind` de la directiva `<%@ Control..` que aparece al comienzo del fichero `.ascx`. En el ejemplo que estamos utilizando, el contenido del fichero de código `ContactViewer.ascx.cs` será el que se muestra a continuación:

```
public class ContactViewer : System.Web.UI.UserControl
{
  // Controles de la interfaz de usuario

  protected...

  // Modelo asociado a la vista

  public Contact DisplayedContact {
      set { UpdateUI(value); }
  }

  private void UpdateUI (Contact contact) {

    if (contact!=null) {
       LabelName.Text = contact.Name;
       LabelEMail.Text = contact.EMail;
       LabelTelephone.Text = contact.Telephone;
       LabelMobile.Text = contact.Mobile;
       LabelFax.Text = contact.Fax;
       LabelAddress.Text = contact.Address;
       LabelComments.Text = contact.Comments;

       ImagePhoto.ImageUrl = "contacts/"+contact.ImageURL;
    }
  }

  // Código generado por el diseñador de formularios
  ...
}
```

Como se puede apreciar, el código que hemos de implementar se limita a mostrar en las distintas etiquetas de nuestra interfaz los datos correspondientes al contacto que queremos visualizar. Para mostrar la fotografía correspondiente, lo único que hacemos es establecer la URL de la imagen mostrada en la parte derecha de la tabla. Cuando queramos utilizar este control para mostrar los datos de contacto de alguien, lo único que tendremos que hacer es establecer un valor adecuado para su propiedad `DisplayedContact`.

Ya hemos visto en qué consiste un control ASP.NET definido por el usuario, aunque aún no hemos dicho nada acerca de cómo se crean y cómo se utilizan los controles web en la práctica.

Para crear un control web de usuario como el que hemos utilizado de ejemplo en este apartado, podemos utilizar tres estrategias diferentes:

- **Creación manual:** Podemos seguir el mismo proceso que hemos visto en el ejemplo para construir un control ASP.NET escribiendo el contenido del fichero `.ascx` e implementando el código asociado al control en un fichero aparte con extensión `.ascx.cs`. Sólo hemos de tener en cuenta que, al comienzo del fichero `.ascx`, debe aparecer la directiva `<%@ Control...` y que el control debe heredar de la clase `System.Web.UI.UserControl`.

- **Conversión de una página ASP.NET en un control:** Para que resulte más cómoda la creación del control, podemos comenzar creando un formulario ASP.NET convencional. Una vez que el formulario haga lo que nosotros queramos, podemos convertir la página ASP.NET en un control modificando las extensiones de los ficheros. Esta estrategia también es válida cuando ya tenemos la página hecha y queremos reutilizarla. La conversión de una página en un control requiere cambiar la extensión de los ficheros (de `.aspx` y `.aspx.cs` a `.ascx` y `.ascx.cs`), eliminar las etiquetas que no pueden aparecer en el control (`<html>`, `<head>`, `<body>` y `<form>`, así como la directiva `<!DOCTYPE>`), cambiar la directiva que aparece al comienzo de la página (de `<%@ Page...` a `<%@ Control...`) y modificar la clase que implementa la funcionalidad del control para que herede de `System.Web.UI.UserControl` en lugar de hacerlo de `System.Web.UI.Page`.

- **Con Visual Studio .NET**: Cuando estamos trabajando con una aplicación web, podemos añadir un "control de usuario web" desde el menú contextual asociado a nuestro proyecto en el *Explorador de Soluciones* de Visual Studio .NET. Esto nos permite utilizar el diseñador de formularios para implementar nuestro control igual que se implementa un formulario web ASP.NET.

Utilizando cualquiera de las tres estrategias mencionadas, al final obtendremos un control implementado en dos ficheros, uno para definir el aspecto visual de la interfaz de usuario (el fichero con extensión `.ascx`) y otro para el código que implementa la funcionalidad del control (el fichero que tiene extensión `.ascx.cs`).

Una vez que hemos creado el control, sólo nos falta ver cómo utilizarlo. Básicamente, el control que acabamos de mostrar lo emplearemos en la creación de formularios web ASP.NET como cualquier otro de los controles que ya vimos en los apartados anteriores de este capítulo. La única diferencia es que nuestro control no podemos ponerlo en el "cuadro de herramientas" del Visual Studio .NET.

La forma más sencilla de utilizar un control web creado por el usuario es, en el mismo entorno de desarrollo, arrastrar el control desde el *Explorador de Soluciones* hasta el lugar del formulario web en el que deseamos que aparezca nuestro control. Obviamente, primero hemos de incluir el control en nuestro proyecto actual.

Tras arrastrar el control a una página, en el diseñador de formularios veremos algo similar a lo mostrado en la siguiente imagen:

El control que hemos creado aparece en nuestra página ASP.NET como un simple botón. Será cuando ejecutemos la página cuando aparezca el control tal como lo hemos diseñado.

La sencilla operación de arrastrar y soltar que realizamos en el diseñador de formularios web se traduce en una serie de cambios en el fichero de nuestra página ASP.NET. En el ejemplo mostrado, la inclusión del control web en la página ASP.NET provoca que nuestro fichero `.aspx` tenga el siguiente aspecto:

```
<%@ Page language="c#"
         Codebehind="Contacts.aspx.cs"
         AutoEventWireup="false"
         Inherits="WebMail.Contacts" %>
<%@ Register TagPrefix="user"
             TagName="ContactViewer"
             Src="ContactViewer.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>Información de contacto</title>
  </HEAD>
  <body>
    <form id="FormContacts" method="post" runat="server">
      <P align="center">
        <asp:DropDownList id="ContactList"
                          runat="server" Width="80%"
                          AutoPostBack="True">
        </asp:DropDownList>
        <asp:Button id="ButtonOK" runat="server" Text="Mostrar">
        </asp:Button></P>
      <DIV align="center">
        <P align="center" id="Header" runat="server">
          Información de contacto de
          <asp:Label id="LabelContact" runat="server">
            ...
          </asp:Label></P>
      </DIV>
      <P align="center">
        <user:ContactViewer id="ContactView" runat="server">
        </user:ContactViewer>
      </P>
    </form>
  </body>
</HTML>
```

Si observamos detenidamente la página ASP.NET anterior, podremos apreciar los cambios introducidos por la inclusión de un control web de usuario. Conociendo en qué consisten estos cambios podríamos utilizar un control definido por el usuario sin necesidad de la ayuda del entorno de desarrollo.

En realidad, la utilización de un control web de usuario ocasiona que en nuestro fichero `.aspx` aparecen dos novedades: una directiva al comienzo de la página (la directiva `<%@ Register...`) y una etiqueta ASP.NET que hace referencia a nuestro control (`<user:ContactViewer...`).

La directiva `<%@ Register...` registra el uso del control en nuestra página ASP.NET, especificando el fichero donde se encuentra el control (`Src="ContactViewer.ascx"`) y la etiqueta que utilizaremos para incluir el control en nuestro formulario web (mediante los atributos `TagPrefix` y `TagName`). En cierto modo, esta directiva viene a ser como un `#define` de C.

```
<%@ Register TagPrefix="user"
             TagName="ContactViewer"
             Src="ContactViewer.ascx" %>
```

Una vez que hemos creado la forma de hacer referencia al control con la directiva `<%@ Register...`, para utilizar el control en la página lo incluiremos como incluiríamos cualquier otro control ASP.NET, utilizando la etiqueta que hemos definido (`user:ContactViewer` en el ejemplo). Cuando lo hacemos, hemos de especificar que se trata de un control de servidor; esto es, que el control tiene asociada funcionalidad que ha de ejecutarse al procesar la página ASP.NET en el servidor. Esto lo conseguimos con el atributo `runat="server"`. Finalmente, lo usual es que le asociemos un identificador al control que acabamos de incluir en la página ASP.NET. Esto se logra con el atributo `id="ContactView"`:

```
<user:ContactViewer id="ContactView" runat="server">
</user:ContactViewer>
```

Con el control incluido correctamente en la página ASP.NET, podemos pasar a ver cómo se pueden manipular sus propiedades en tiempo de ejecución. Si estamos trabajando con el Visual Studio .NET, pulsar la tecla `F7` nos llevará del diseñador de formularios web al código asociado a la página.

Si queremos utilizar el control desde el código asociado a la página, hemos de asegurarnos de que el control aparece como una variable de instancia de la clase que implementa la funcionalidad de la página ASP.NET. En otras palabras, en el fichero de código con extensión `.aspx.cs` debe existir una declaración de una variable protegida que haga referencia al

control de usuario incluido en la página. El tipo de esta variable coincidirá con el tipo de nuestro control y su identificador ha de coincidir con el identificador que hayamos indicado en el atributo `id` de la etiqueta `user:ContactViewer`:

```
  protected ContactViewer ContactView;
```

Volviendo al ejemplo anterior, el fichero de código completo asociado a nuestra página ASP.NET será el siguiente:

```
public class Contacts : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.DropDownList ContactList;
  protected System.Web.UI.WebControls.Label LabelContact;
  protected System.Web.UI.WebControls.Button ButtonOK;
  protected System.Web.UI.HtmlControls.HtmlGenericControl Header;

  protected ContactViewer ContactView;

  private void Page_Load(object sender, System.EventArgs e)
  {
    // Inicialización de la lista desplegable
    ...
    UpdateUI(contact);
  }

  // Código generado por el diseñador (OnInit & InitializeComponents)
  ...

  // Manejadores de eventos
  ...

  // Actualización de la interfaz

  private void UpdateUI (Contact contact)
  {
    ContactView.DisplayedContact = contact;
    Header.Visible = true;
    ContactView.Visible = true;
  }
}
```

El código anterior se limita a establecer la propiedad `DisplayedContact` de nuestro control. El valor adecuado para esta propiedad lo obtendremos previamente de algún parámetro que le llegue a nuestra página ASP.NET, de un fichero o de una base de datos. Nuestro control se encargará internamente de visualizar los datos del contacto de la forma adecuada. Al ejecutar nuestra página ASP.NET, el resultado final que se le mostrará al usuario será similar al recogido en la siguiente figura:

Página ASP.NET con un control web definido por el usuario.

Creación de controles a medida

En este apartado hemos visto cómo se pueden crear controles por composición, aunque ya mencionamos al principio que ésta no es la única opción disponible. También se pueden crear controles por derivación a partir de clases que implementen controles ya existentes o directamente a partir de la clase base de todos los controles web: `System.Web.UI.WebControl`.

En la plataforma .NET, la creación de controles web de usuario se puede realizar de tres formas diferentes cuando optamos por crear el control por derivación:

- Podemos crear una clase que herede directamente de alguno de los controles ASP.NET existentes y le añada algo de funcionalidad, sin tener que preocuparnos de la presentación visual del control. Por ejemplo, podemos crear una clase que herede de `TextBox` y sólo permita la introducción de datos en un determinado formato interceptando algunos eventos de un `TextBox` convencional.

Creación de controles a medida

- Podemos crear un control completamente nuevo, para lo cual implementaremos una clase que herede directamente de `System.Web.UI.WebControl`. Además, el control puede que tenga que implementar interfaces como `IPostBackDataHandler` o `IPostBackEventHandler`, los cuales son interfaces relacionados con el funcionamiento de las páginas ASP.NET (que veremos en la siguiente sección de este mismo capítulo). La clase creada de esta forma ha de implementar todo el código que sea necesario para mostrar el control en la interfaz de usuario.

- Como última alternativa, podemos crear un "control compuesto" implementando una clase que simule el funcionamiento de un control creado por composición. En este caso, es necesario que el control incluya el interfaz `INamingContainer` como marcador y redefina el método `CreateChildComponents`, en el cual se crearán los controles que conforman el control compuesto.

Sea cual sea la estrategia utilizada para implementar el control, en la creación de controles por derivación deberemos emplear las técnicas habituales de programación orientada a objetos (encapsulación, herencia y polimorfismo). En otras palabras, hemos de implementar nuestras propias clases "a mano", sin ayuda de un diseñador visual como el diseñador de formularios web de Visual Studio .NET que se puede emplear en la creación de un control por composición.

Los controles creados por composición resultan adecuados para mostrar datos de una forma más o menos estática, mientras que los controles creados por derivación nos permiten ser más flexibles a la hora de generar el aspecto visual del control (que hemos de crear dinámicamente desde el código del control).

Al emplear composición, la reutilización del control se realiza a nivel del código fuente (el fichero `.ascx` y su fichero de código asociado). Por otro lado, si creamos un control por derivación, podemos compilarlo e incluirlo en un *assembly*. Esto nos permite añadir nuestro control al "cuadro de herramientas" de Visual Studio .NET. Además, al tener compilado el control, en tiempo de diseño podremos ver el aspecto visual del control en el diseñador de formularios, así como acceder a sus propiedades y eventos desde la "ventana de propiedades" del Visual Studio .NET.

En definitiva, la elección de una u otra alternativa dependerá, en gran medida, de lo que queramos conseguir y del tiempo del que dispongamos. A la hora de elegir cómo crear nuestros propios controles hemos de optar por la simplicidad de la creación por composición o la flexibilidad de uso que nos ofrecen los controles creados por derivación.

Funcionamiento de las páginas ASP.NET

En lo que llevamos de este capítulo, hemos visto cómo se pueden construir aplicaciones web utilizando formularios con controles de forma completamente análoga al desarrollo de aplicaciones para Windows en un entorno de programación visual. Aunque disponer de un diseñador de formularios para construir aplicaciones web resulta excepcionalmente cómodo para el programador, la semejanza con los entornos de desarrollo para Windows puede conducir a algunos equívocos. El objetivo de esta sección es aclarar aquellos aspectos que diferencian la creación de interfaces web de la creación de interfaces para Windows utilizando entornos de programación visual. La correcta comprensión de estas diferencias resulta esencial para la correcta implementación de aplicaciones web.

En una página ASP.NET, todos los controles cuyo funcionamiento haya de controlarse en el servidor de algún modo deben estar incluidos dentro de un formulario HTML. Los formularios HTML han de ir delimitados por la etiqueta estándar `<form>`. En el caso de ASP.NET, dicha etiqueta ha de incluir el atributo `runat="server"`. Este atributo le indica al servidor web (Internet Information Server) que la página ASP.NET ha de procesarse en el servidor antes de enviársela al cliente. Como consecuencia, el esqueleto de una página ASP.NET será siempre de la forma:

```
<form runat="server">
  ...
  <!-- HTML y controles ASP.NET -->
  ...
</form>
```

El formulario HTML incluido en la página ASP.NET, que ha de ser necesariamente único, es el encargado de facilitar la interacción del servidor con el navegador web que el usuario final de la aplicación utiliza en su máquina. Aunque la forma de programar los formularios web ASP.NET sea prácticamente idéntica a la creación de formularios para Windows, el modo de interacción característico de las interfaces web introduce algunas limitaciones. Estas limitaciones se deben a que cada acción que el usuario realiza en su navegador se traduce en una solicitud independiente al servidor web.

El hecho de que cada solicitud recibida por el servidor sea independiente de las anteriores ocasiona que, al desarrollar aplicaciones web, tengamos que ser conscientes de cuándo se produce cada solicitud y de cómo podemos enlazar solicitudes diferentes realizadas por un mismo usuario. En los dos siguientes apartados analizaremos cómo se resuelven estas dos cuestiones en las páginas ASP.NET. Dejaremos para el capítulo siguiente la forma de identificar las solicitudes provenientes de un único usuario cuando nuestra aplicación consta de varias páginas.

Solicitudes y "postbacks"

Al solicitar una página ASP.NET desde un cliente, en el servidor se dispara el evento `Page_Load` asociado a la página antes de generar ninguna salida. Es en el manejador asociado a este evento donde debemos realizar las tareas de inicialización de la página. Dichas tareas suelen incluir el establecimiento de valores por defecto o el rellenado de las listas de valores que han de mostrarse al usuario.

El evento `Page_Load` se dispara **cada vez** que el usuario accede a la página. Si lo que deseamos es realizar alguna tarea **sólo la primera vez** que un usuario concreto accede a la página, hemos de emplear la propiedad `Page.IsPostBack`. Esta propiedad posee el valor `false` cuando el cliente visualiza por primera vez la página ASP.NET, mientras que toma el valor `true` cuando no es la primera vez que la página ha de ejecutarse para ser mostrada. Esto sucede cuando el usuario realiza alguna acción, como pulsar un botón del formulario web, que tiene como consecuencia volver a generar la página para presentar datos nuevos o actualizados en la interfaz de usuario.

En ASP.NET y otras muchas tecnologías de desarrollo de interfaces web, cuando el usuario realiza una acción que requiere actualizar el contenido de la página que está visualizando, la página "se devuelve al servidor". De ahí proviene el término inglés *post back*. El uso de *postbacks* es una técnica común para manejar los datos de un formulario que consiste en enviar los datos a la misma página que generó el formulario HTML.

Page.IsPostBack

Una vez visto en qué consiste la ejecución repetida de una página ASP.NET como respuesta a las distintas acciones del usuario, estamos en disposición de completar el código correspondiente al ejemplo con el que terminamos la sección anterior de este capítulo (el del uso de controles definidos por el usuario en la creación de formularios web). Para que nuestro formulario muestre en cada momento los datos del contacto que el usuario selecciona de la lista desplegable, hemos de implementar el evento `Page_Load`. Dicho evento se ejecuta cada vez que se accede a la página y ha de seleccionar el contacto adecuado que se mostrará en el navegador web del usuario:

```
private void Page_Load(object sender, System.EventArgs e)
{
  if (!Page.IsPostBack) {

     // Inicialización de la interfaz
     // - Rellenado de la lista desplegable de contactos
     ...
     Header.Visible = false;
     ContactList.SelectedIndex = 0;
```

```
  } else {

     // Visualización del contacto actual
     contact = ...
     UpdateUI(contact);
  }
}

private void InitializeComponent()
{
  this.Load += new System.EventHandler(this.Page_Load);
}
```

Como se puede apreciar, hemos de diferenciar la primera vez en que el usuario accede a la página de las ocasiones en las que el acceso se debe a que el usuario haya pulsado el botón de su interfaz para mostrar los datos de un contacto diferente. La primera vez que el usuario accede a la página, cuando se cumple la condición `!Page.IsPostBack`, se tiene que inicializar el estado de los distintos controles del formulario web. En el ejemplo que nos ocupa, se ha de rellenar la lista desplegable de valores que le permitirá al usuario seleccionar el contacto cuyos datos desea ver.

AutoPostBack

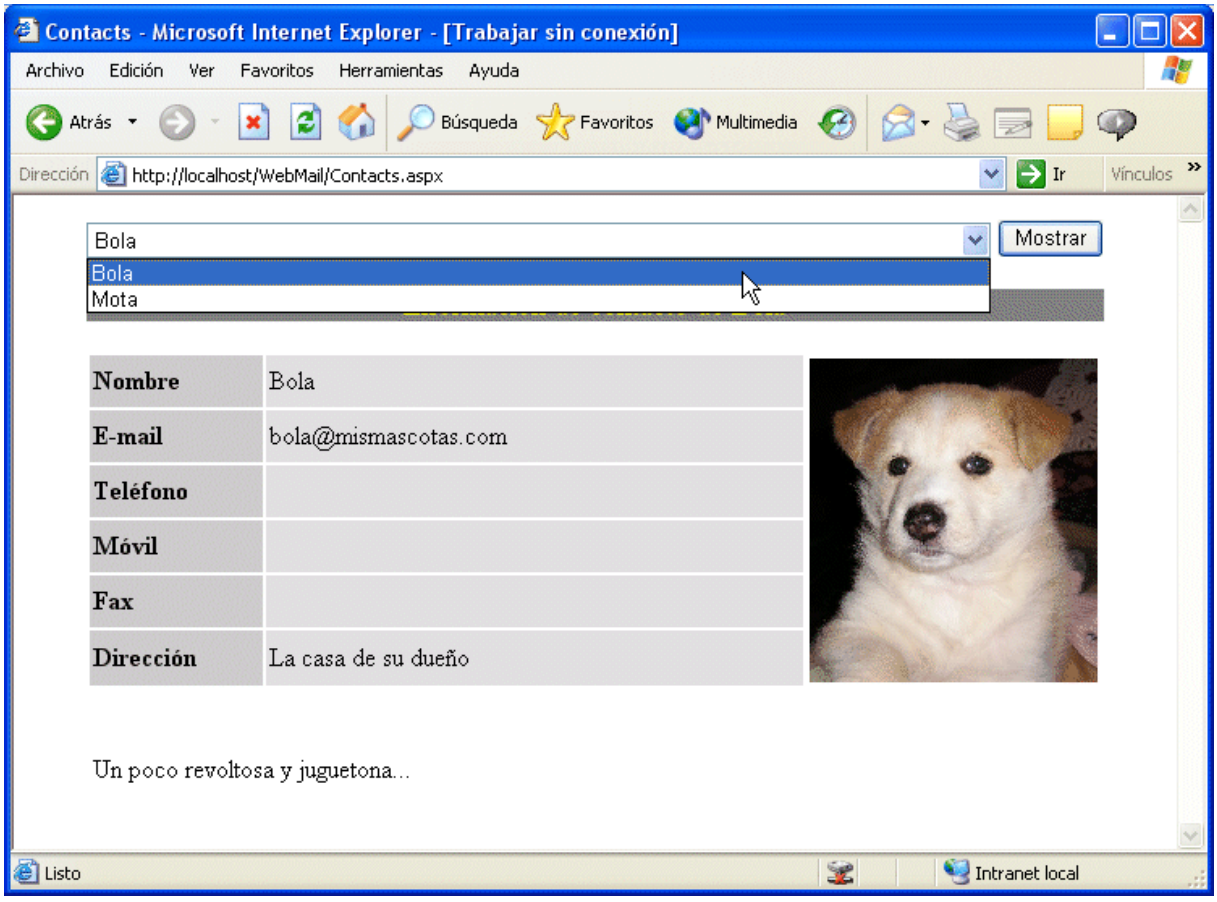
Utilizando únicamente el manejador correspondiente al evento `Page_Load` podemos conseguir una página dinámica cuya actualización se realiza cada vez que el usuario pulsa un botón, pulsación que se traduce en una nueva solicitud al servidor web (*post back* si empleamos la terminología habitual). No obstante, en determinadas ocasiones nos puede interesar que la interfaz de nuestra aplicación web responda a otras acciones del usuario, no sólo a la pulsación final de un botón del formulario.

En el ejemplo que venimos desarrollando en este capítulo, puede que nos interese mostrar los datos de contacto de alguien en el mismo momento en que el usuario selecciona un nombre de la lista desplegable. De esta forma, la respuesta inmediata de la aplicación facilita que el usuario perciba el efecto de las acciones que realiza. Al ver reflejadas sus acciones de forma inmediata en la ventana de su navegador web, el usuario ve reducida la distancia existente entre sus acciones y la respuesta del sistema con el que interactúa, uno de los principios fundamentales del diseño de interfaces de usuario. En el caso de la lista desplegable, debemos implementar la respuesta del control al evento `SelectedIndexChanged`:

```
private void ContactList_SelectedIndexChanged
                 (object sender, System.EventArgs e)
{
    // Contacto seleccionado de la lista
    contact = ...
    UpdateUI(contact);
}
```

```
private void InitializeComponent()
{
  ...
  this.ContactList.SelectedIndexChanged += new
      System.EventHandler(this.ContactList_SelectedIndexChanged);
  ...
}
```

Implementar la respuesta del control a este evento no es suficiente para que la aplicación responda de forma inmediata a un cambio en el elemento seleccionado de la lista desplegable. Si queremos que la página se actualice en el mismo momento en que el usuario selecciona un elemento de la lista, hemos de establecer a `true` la propiedad `AutoPostBack` del control de tipo `DropDownList` que corresponde a la lista desplegable.



Uso de la propiedad AutoPostBack para mejorar la usabilidad de una página ASP.NET: En cuanto el usuario selecciona un contacto de la lista desplegable, en el control de usuario de debajo aparecen todos sus datos.

La propiedad `AutoPostBack` existente en algunos de los controles ASP.NET sirve para que, ante determinados eventos relacionados con acciones del usuario, el estado de los controles de la página se envíe automáticamente al servidor. Esto permite actualizar el contenido de la página conforme el usuario interactúa con la aplicación. Si el retardo existente entre la solicitud del usuario y la respuesta del servidor web no es demasiado elevado, la usabilidad de la aplicación mejora. Sin embargo, el uso indiscriminado de la propiedad `AutoPostBack` genera una mayor carga sobre el servidor web al realizar cada cliente más solicitudes relativas a la misma página. Aparte de consumirse un mayor ancho de banda en la red, conforme aumenta la carga del servidor web, su tiempo de respuesta aumenta y se anulan las ventajas que supone utilizar `AutoPostBack` en primera instancia.

Internamente, el uso de la propiedad `AutoPostBack` para forzar la actualización de la página cuando el usuario realiza alguna acción se traduce en la inclusión de un fragmento de JavaScript en la página que se le envía al cliente. Este fragmento de código se le asocia, en el navegador web del cliente, al evento correspondiente del control cuya respuesta implementamos en la página ASP.NET (que se ejecuta en el servidor). Si volvemos al ejemplo anterior, el control `DropDownList` da lugar al siguiente fragmento de HTML dinámico:

```
<select name="ContactList"  id="ContactList"
        onchange="__doPostBack('ContactList','')"
        language="javascript" style="width:80%;">
  <option value="bola.xml" selected="selected">Bola</option>
  <option value="mota.xml">Mota</option>
</select>
```

Ya que el uso de `AutoPostBack` se traduce en la utilización de HTML dinámico en el cliente, el navegador del usuario ha de ser compatible con JavaScript (ECMAScript para ser precisos). Además, el usuario ha de tener habilitado el uso de secuencias de comandos en su navegador web.

Como consecuencia del uso interno de la versión de JavaScript estandarizada por ECMA, el estándar determina en qué controles y sobre qué eventos se puede emplear la propiedad `AutoPostBack`. Afortunadamente, dicha propiedad está disponible para la mayoría de las situaciones en las que nos pueda interesar refrescar el contenido de la página que ve el usuario. Como se mostró en el ejemplo anterior, podemos hacer que nuestra aplicación responda a un cambio en el elemento seleccionado de una lista, ya sea desplegable (de tipo `DropDownList`) o no (`ListBox`). También podemos conseguir que la aplicación web responda inmediatamente a acciones como el marcado de una caja de comprobación (`CheckBox` y `CheckBoxList`) o la selección de un botón de radio (`RadioButton` y `RadioButtonList`). Incluso podemos lograr que se genere una solicitud al servidor en cuanto el usuario modifique el texto contenido en un control de tipo `TextBox`.

En cualquier caso, cuando decidamos utilizar la propiedad `AutoPostBack` debemos tener en cuenta la carga adicional que esto supone sobre el servidor web y el consumo de ancho de

banda que las solicitudes adicionales suponen. Al sopesar los pros y los contras de la utilización de `AutoPostBack`, nunca debemos olvidar que, aunque mejoremos la realimentación que recibe el usuario como respuesta a sus acciones individuales, esta mejora puede llegar a ser contraproducente si ralentiza la realización de tareas completas.

Estado de una página ASP.NET

A diferencia de las aplicaciones para Windows, en las cuales el usuario interactúa con una instancia concreta de un formulario, en la aplicaciones web cada acción del usuario se trata de forma independiente. En otras palabras, cada vez que se le muestra una página al usuario, la página se construye de nuevo. Esto implica que el objeto concreto que recibe una solicitud del usuario no es el mismo aunque al usuario le dé la sensación de estar trabajando con "su" página. Desde el punto de vista práctico, este hecho tiene ciertas implicaciones que comentamos a continuación retomando el ejemplo de la lista de contactos.

Si recordamos, cuando creamos un control web de usuario para mostrar los datos de contacto de alguien, creamos una clase `ContactViewer` con una propiedad `DisplayedContact` que utilizábamos para mostrar los datos de un contacto concreto. Ingenuamente, puede que nos olvidamos del contexto en el que funcionan las aplicaciones web y escribamos algo como lo siguiente:

```
public class ContactViewer : System.Web.UI.UserControl
{
  ...
  private Contact contact;

  public Contact DisplayedContact {
    get { return contact; }
    set { contact = value; }
  }
  ...
}
```

Aparentemente, podríamos utilizar la variable de instancia `contact` para acceder a los datos que hemos de mostrar en pantalla. Sin embargo, cada vez que el usuario realiza alguna acción que se traduce en una nueva solicitud ál servidor (lo que denominábamos *post back* en la sección anterior), en el servidor web se crea un objeto nuevo encargado de atender la solicitud. Por consiguiente, el valor que habíamos guardado en la variable de instancia del objeto con el que se trabajaba en la solicitud anterior se habrá perdido.

Como consecuencia, la construcción de controles y de páginas en ASP.NET no puede realizarse de la misma forma que se implementaría una clase convencional. En una aplicación Windows, cuando se tiene una referencia a una instancia de una clase, siempre se trabaja con la misma referencia. En ASP.NET, cada acción se realiza sobre un objeto diferente, por lo que

debemos utilizar algún mecanismo que nos permita mantener el estado de una página o de un control entre distintas solicitudes procedentes de un mismo usuario.

Por suerte, los diseñadores de ASP.NET decidieron incluir un sencillo mecanismo por el cual se puede almacenar el estado de una página. Este mecanismo se basa en la utilización de un array asociativo en el cual podemos almacenar cualquier objeto, siempre y cuando éste sea serializable. La forma correcta de implementar una propiedad en el control web de usuario que diseñamos antes sería la siguiente:

```
public class ContactViewer : System.Web.UI.UserControl
{
  ...
  public Contact DisplayedContact {
    get { return (Contact) ViewState["contact"]; }
    set { ViewState["contact"] = value; }
  }
  ...
}
```

En lugar de utilizar una variable de instancia, como haríamos habitualmente al diseñar una clase que ha de encapsular ciertos datos, se emplea el array asociativo `ViewState` para garantizar que el estado de la página se mantiene entre solicitudes diferentes.

En el caso de los controles ASP.NET predefinidos, su implementación se encarga de mantener el estado de los controles cuando se recibe una solicitud correspondientes a un *post back*. Cuando se recibe una nueva solicitud, aunque se trabaje con objetos diferentes, lo primero que se hace es reconstruir el estado de los controles de la interfaz a partir de los datos recibidos del usuario.

Internamente, el estado de una página ASP.NET se define mediante un campo oculto incluido en el formulario HTML correspondiente a la página. Este campo oculto, denominado `__VIEWSTATE`, se le añade en el servidor a cada página que tenga un formulario con el atributo `runat="server"` (de ahí la necesidad de que el formulario HTML de la página ASP.NET incluya este atributo). De esta forma, es la página web que visualiza el usuario la que se encarga, sin que el usuario sea consciente de ello, de mantener el estado de la página ASP.NET en el servidor. Esto lo podemos comprobar fácilmente si visualizamos el código fuente de la página HTML que se muestra en el navegador web de la máquina cliente. En él podemos ver algo así:

```
<input type="hidden" name="__VIEWSTATE"
       value="dDwtMjEwNjQ1OTkwMDs7PiTPnxCh1VBUIX3K2htmyD8Dq6oq" />
```

Este mecanismo, que supone una novedad en ASP.NET con respecto a versiones anteriores de ASP, nos ahorra tener que escribir bastantes líneas de código. Al encargarse `ViewState` de

mantener automáticamente el estado de los controles de los formularios web, el programador puede despreocuparse del hecho de que cada solicitud del usuario se procese de forma independiente.

En ASP clásico, al enviar un formulario, el contenido de la página HTML que ve el usuario se pierde, por lo que el programador debe reconstruir el estado de la página manualmente. Por tanto, si se produce un error, por pequeño que sea, al rellenar alguno de los valores de un formulario, el programador debe implementar el código que se encargue de rellenar los valores que sí son correctos. Esto resulta aconsejable siempre y cuando queramos evitarle al usuario tener que introducir de nuevo todos los datos de un formulario cuando, a lo mejor, se le ha olvidado introducir un dato que era obligatorio. Tener que hacer algo así es demasiado común en muchas aplicaciones web, además de resultar bastante irritante.

Por desgracia, la labor del programador para evitarle inconvenientes al usuario final de la aplicación resulta bastante tediosa y es, en consecuencia, muy propensa a errores. Afortunadamente, en ASP.NET, el formulario web reaparece automáticamente en el navegador del cliente con el estado que sus controles ASP.NET tuviesen anteriormente.

Si seguimos utilizando el estilo tradicional de ASP clásico, una sencilla página podría tener el aspecto siguiente:

```
<%@ Page language="c#" %>
<html>
 <body>

  <form runat="server" method="post">
    Tu nombre:
    <input type="text" name="nombre" size="20">
    <input type="submit" value="Enviar">
  </form>

  <%
    string name = Request.Form["nombre"];

    if (name!=null && name!="") {
       Response.Write("Hola, " + name + "!");
    }
  %>

 </body>
</html>
```

Esta página incluye un formulario en el que el usuario puede introducir su nombre. Cuando el usuario pulsa el botón "Enviar", en su navegador web aparecerá de nuevo el formulario seguido de un mensaje de bienvenida. Sin embargo, el campo del formulario cuyo valor había rellenado aparecerá vacío. Si utilizamos los controles ASP.NET, no obstante, podemos evitar que el valor introducido desaparezca tecleando lo siguiente en nuestro fichero `.aspx`:

```
<%@ Page language="c#" %>
<html>
 <script runat="server">
  void enviar (object sender, EventArgs e)
  {
    label.Text = "Hola, " + textbox.Text + "!";
  }
 </script>
 <body>
  <form runat="server" method="post">
   Tu nombre: <asp:TextBox id="textbox" runat="server" />
   <asp:Button OnClick="enviar" Text="Enviar" runat="server" />
   <p><asp:Label id="label" runat="server" /></p>
  </form>
 </body>
</html>
```

El mantenimiento automático del estado de la página que ofrecen los formularios web cuando utilizamos controles ASP.NET nos sirve para evitar que el usuario "pierda" los datos que acaba de introducir. Como es lógico, aunque el mantenimiento del estado de la página es automático en ASP.NET, puede que nos interese que la página no mantenga su estado. Esto puede suceder cuando el usuario teclea una clave privada o cualquier otro tipo de dato cuya privacidad se haya de preservar. También puede ser recomendable que una página no mantenga su estado cuando se están introduciendo series de datos, ya que mantener el estado de la página podría ocasionar la recepción de datos duplicados cuya existencia habría que detectar en el código de la aplicación.

ASP.NET nos ofrece dos alternativas para indicar explícitamente que el estado de un formulario web no debe mantenerse entre solicitudes independientes aunque éstas provengan de un único usuario:

- A nivel de la página, podemos emplear la directiva `<%@ Page EnableViewState="false" %>` en la cabecera del fichero `.aspx`. Esto deshabilita el mantenimiento del estado para todos controles de la página ASP.NET.

- A nivel de un control particular, podemos establecer su propiedad `EnableViewState` a `false`. De esta forma, podemos controlar individualmente el comportamiento de las distintas partes de una página en lo que se refiere al mantenimiento de su estado.

Resumen del capítulo

En este capítulo hemos aprendido todo lo que necesitamos para ser capaces de construir interfaces web utilizando la tecnología ASP.NET incluida en la plataforma .NET de Microsoft. Para ser más precisos, nos centramos en la construcción de páginas ASP.NET individuales:

- En primer lugar, descubrimos en qué consiste el modelo de programación de los formularios web, que está basado en el uso de controles y eventos.

- A continuación, vimos cómo podemos crear nuestros propios controles para construir interfaces web más modulares y flexibles.

- Finalmente, estudiamos el funcionamiento de las páginas ASP.NET, haciendo especial hincapié en los detalles que diferencian la creación de interfaces web de la construcción de aplicaciones para Windows. En este apartado llegamos a aprender cómo se puede controlar el mantenimiento del estado de una página ASP.NET.

Sin embargo, una aplicación web rara vez consta de una única página, por lo que aún nos quedan por ver algunos aspectos relacionados con la construcción de aplicaciones web. Este será el hilo conductor del siguiente capítulo.