ASP.NET. Controles HTML y ASP.NET de Servidor.

Controles HTML de servidor.

Los controles HTML de servidor son los típicos de HTML y por tanto bastante austeros. Permiten diseñar páginas básicas HTML, compatibles con todos (o casi todos) los navegadores.

Son accesibles desde la pestaña HTML de la "Caja de herramientas".

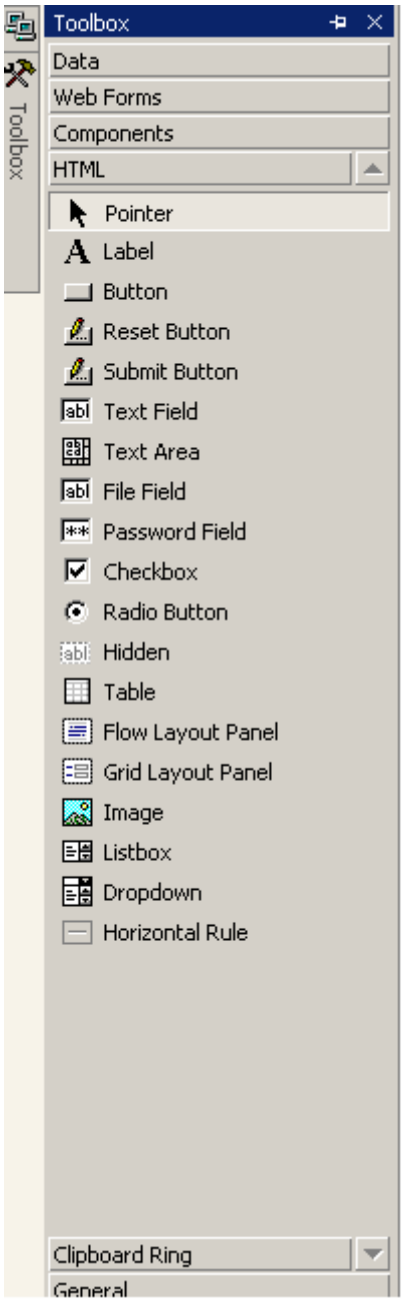

Figura 21.1. Caja de herramientas. Pestaña de controles HTML.

Los controles HTML de servidor pertenecen al `namespace` `System.Web.UI.HTMLControls` y derivan de la clase

`System.Web.UI.HTMLControls.HTMLControl`,

la cual deriva de `System.Web.UI.Control`.

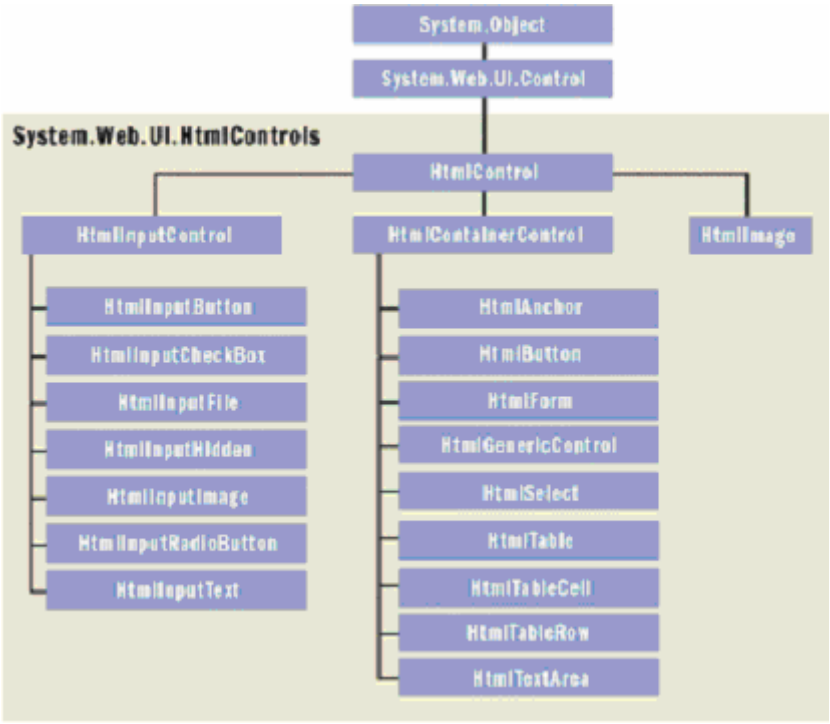

Figura 21.2. `namespace System.Web.UI.HTMLControls` y jerarquía de herencia.

Tienen la particularidad de permitir lanzar eventos en el lado del servidor, al igual que los controles ASP.NET de servidor.
En realidad, cuando se crea un control HTML de servidor lo que se está haciendo es:

- En el fichero con la página Web (`.aspx`): crear una etiqueta HTML correspondiente al control. Por ejemplo, el control `HTMLButton` corresponde a la etiqueta `<button>`.
- En el fichero de código (`.cs`): crear una instancia de la clase a la que pertenece tal control, al igual que con los controles ASP.NET de servidor. Esta clase tiene una serie de métodos y propiedades que permiten acceder a los campos del elemento HTML correspondiente al botón.

Por ejemplo, si se crea un control HTML de servidor de la clase `HTMLButton`, realmente se va a crear un control que permitirá acceder desde el código a un elemento HTML `<button>`. Lo mismo sucede con los controles `HTMLInputButton` (que permite acceder desde el código al elemento `<input type=button>`, `<input type=submit>`, o `<input type=reset>`), `HTMLInputText` (que permite acceder desde el código al elemento `<input type=text>`), etc...

A continuación se va a modificar el ejemplo anterior para que cuando se cambie el valor de la caja de texto desde el servidor cambie también el texto que se muestra en el botón.
El evento que se lanza cuando el texto de la caja de texto -propiedad `Value`- va a cambiar desde el servidor es `ServerChange`.

Si en la ventana de diseño se hace doble click sobre la caja de texto, se crea automáticamente el código correspondiente a tal evento. A partir de aquí sólo resta programar lo que se desee que suceda cuando se dé el evento.

```
private void Text1_ServerChange(object sender, System.EventArgs e)
{
      if (Button1.Text.Equals("Button"))
            Button1.Text = "Hola";
      else
            Button1.Text = "Button";
}
```

Si se ejecuta la aplicación se verá que la primera vez que se pulsa el botón y se cambia en consecuencia la propiedad `Value` de la caja de texto, no cambia el texto del botón. Esto es así porque no se considera cambio el primer paso de la caja de texto vacía a mostrar `Hola Internet`. Para que se lance el evento `ServerChange` ha de existir un texto distinto a `Hola Internet` en la caja de texto antes de pulsar el botón. En tal caso, al pulsar el botón `Button1` se lanzará el evento `Text1_ServerChange` (también se lanzará `Button1_Click()`, por supuesto) y se ejecutará su código, y se3 cambia el texto del botón.

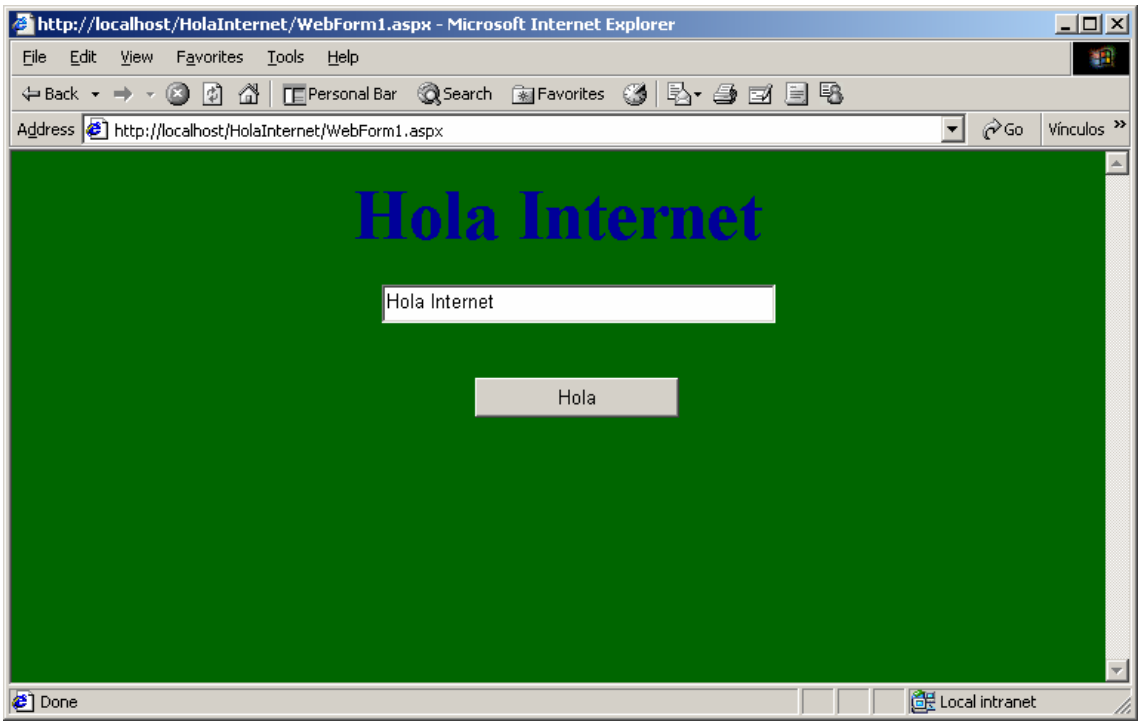

Figura 21.3. Ejecución de la aplicación. Resultado tras lanzar el evento `Text1_Serverchange`.

Controles ASP.NET de servidor.

Los controles ASP.NET de servidor ofrecen una mayor variedad que los controles HTML. Todo control HTML tiene su control ASP.NET equivalente pero ASP.NET ofrece más controles nuevos.

Todos los controles ASP.NET de servidor pertenecen al `namespace` `System.Web.UI.WebControls` y derivan de la clase `System.Web.UI.WebControls.WebControl`,

que a su vez deriva de `System.Web.UI.Control`.

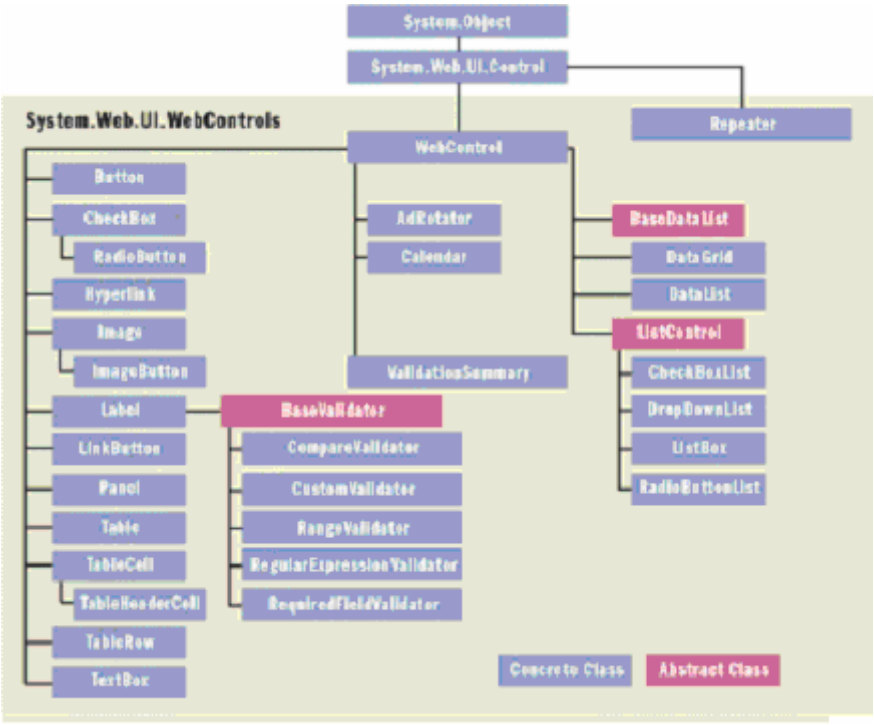

Figura 21.4. `namespace System.Web.UI.WebControls` y jerarquía de herencia.

Cuando se crea un control ASP.NET de servidor lo que se está haciendo es:

- En el fichero con la página Web (`.aspx`): crear una etiqueta HTML especial correspondiente al control. Por ejemplo, el control `Button` corresponde a la etiqueta `<asp:Button>`.
- En el fichero de código (`.cs`): crear una instancia de la clase a la que pertenece tal control. Esta clase tiene una serie de métodos y propiedades que permiten acceder a los campos del elemento HTML especial correspondiente al botón.

Por ejemplo, si se crea un control ASP.NET de servidor de la clase `Button`, realmente se va a crear un control que permitirá acceder desde el código a un elemento HTML especial `<asp:Button>` (como puede verse en la aplicación `HolaInternet`).

Todos estos controles soportan eventos de servidor de un modo más potente que los controles HTML de servidor, ofreciendo un gran abanico de posibilidades a la hora del desarrollo.

En tiempo de diseño, estos controles están disponibles en la pestaña **Web Forms** de la **Caja de herramientas**

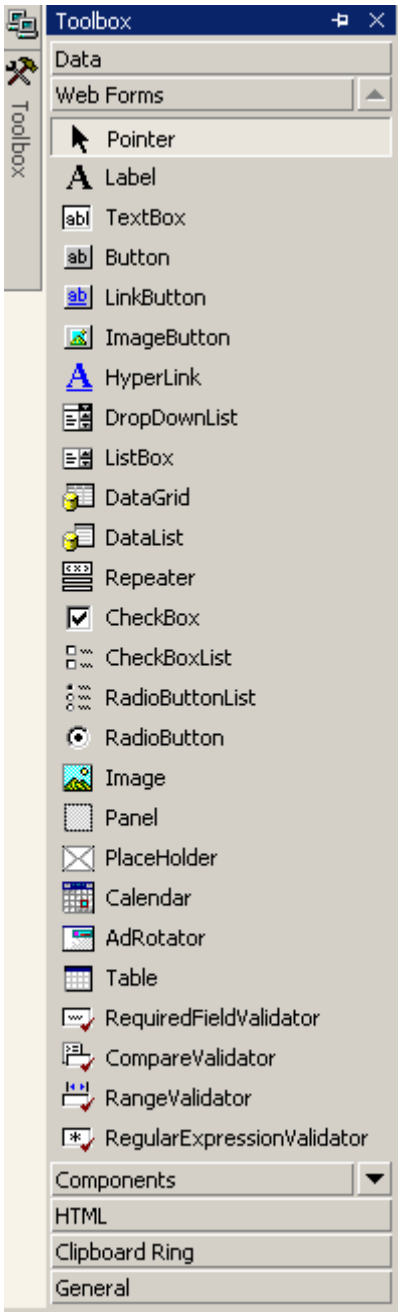

Figura 21.5. Caja de herramientas. Pestaña de controles WebForms.

Clase Label.

Un control de la clase `Label` muestra un texto en una página Web.
La clase `Label` contiene la propiedad `Text`, a partir de la cual es posible cambiar su texto.
Su uso es muy sencillo y se ha podido comprobar en la aplicación `HolaInternet`.

El asistente de Visual Studio.NET separa el código de la página `.aspx`, generando una página `.aspx.cs`. No obstante, es posible diseñar una página `.aspx` que contenga también el código en forma de scripts de servidor.

Clase TextBox.

Es una caja de texto típica. El control `TextBox` es un control de entrada que permite al usuario introducir texto.
Una propiedad interesante es `TextMode`, cuyos valores pueden ser `SingleLine`, `Multiline` o `Password`.

Clase Button.

Es un botón de comando típico.
Puede ser de dos tipos:

- Control `Submit`: La propiedad `CommandName` en un botón de pulsación de tipo `Submit` está vacía. Un botón `Submit`, al ser pulsado, provoca un envío `Post` de la página al servidor y una validación.
- Control de tipo Comando: Es similar al control `Submit`. La diferencia es que la propiedad `CommandName` tiene un valor, que puede considerarse un modo de nombrar a un control para distinguirlo de otros. Cuando se pulsa un botón de tipo comando, se genera un evento `Click` y también un evento `Command` que pueden ser respondidos o tratados con el método `Button_Click` y el método `Button_Command` respectivamente. El método `Button_Command` recibe un parámetro de la clase `CommandEventArgs`, cuyas dos propiedades más importantes son:
  - `CommandName`: contiene el nombre dado al botón.
  - `CommandArgument`: contiene un argumento pasado al pulsar el control. Permite pasar información al servidor cuando se pulsa un botón de comando.

En el siguiente ejemplo se ilustra como crear directamente una etiqueta, una caja de texto y un botón en una página web (`.aspx`) y cómo utilizarlos mediante un script de servidor.

```
<%@ Page Language="C#" %>
<HTML>

<HEAD>

<script language="C#" runat="server">

void Button_Click(Object Sender, EventArgs e)
{
     Text1.Text = "Hola Internet"
}

</script>

</HEAD>

<body>

<form runat="server" ID="Form1">

<h1 align="center">
<font face="Arial">
```

```
<asp:Label id="Label1" Text="Hola Internet" runat="server" />
</font>
</h1>

<p align="center">
<asp:TextBox id="Text1" Text="" Width="200px" runat="server" />

<p align="center">
<asp:Button     id="Button1"     Text="Hola"     OnClick="Button_Click"
runat="server" />

</form>

</P>

</body>

</HTML>
```

Para probar este ejemplo puede crearse una aplicación Web vacía.

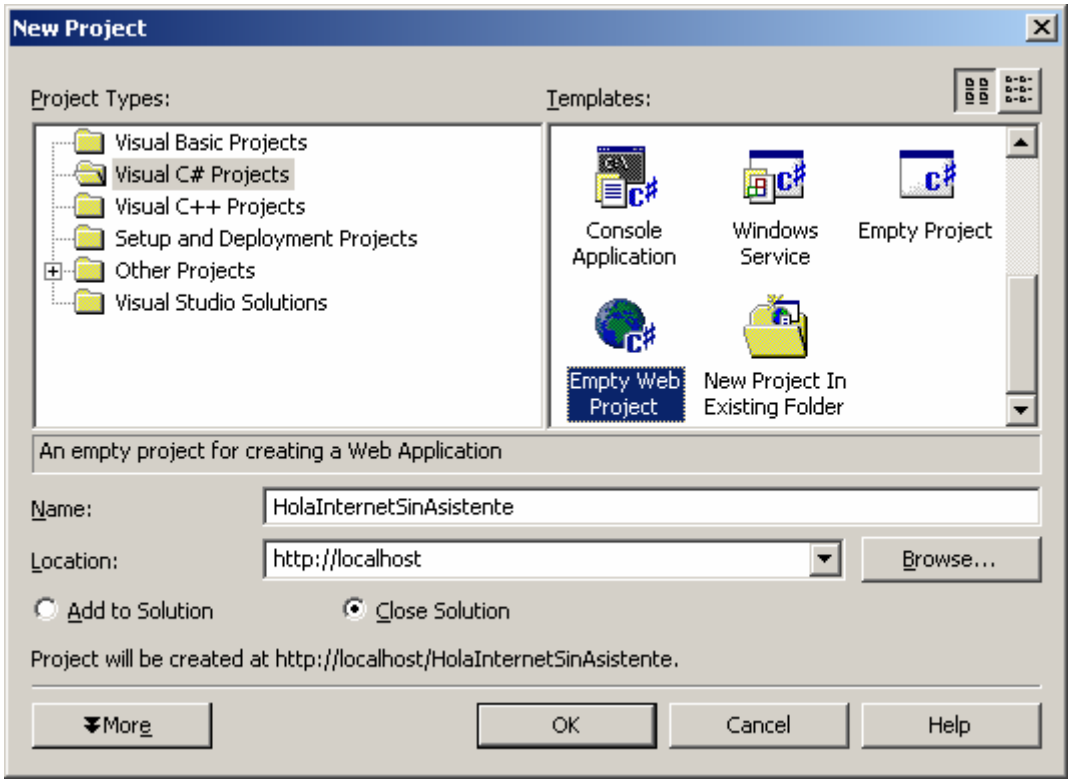

Figura 21.6. Ventana de creación de una nueva aplicación Web ASP.NET vacía, sin asistente.

El siguiente paso es añadir un fichero `.aspx` (menú **Proyecto/Añadir Formulario Web**) vacío, en el que se copiará el código anterior.

Por último habrá que hacer que tal fichero sea la página principal de la aplicación para que se lance al probarla. Conseguir tal cosa implica seleccionar la página en el cuadro **Explorador de soluciones**, pulsar el botón derecho y elegir la opción **Establecer como página de inicio**.

Clase LinkButton.

Esta clase representa un control muy similar, en apariencia, al control de la clase `Hyperlink` pero con la funcionalidad de un botón de comando. Es decir, al pulsarlo no se provoca un salto a un hipervínculo, sino la invocación de un método de servidor de respuesta al evento `Click`.

En el siguiente ejemplo se muestra un uso básico del control `LinkButton`. Al pulsarlo, se ejecutará el evento correspondiente de servidor, que mostrará el texto `Ha sido pulsado el LinkButton1, esto no implica un salto a un hipervínculo` en una caja de texto.

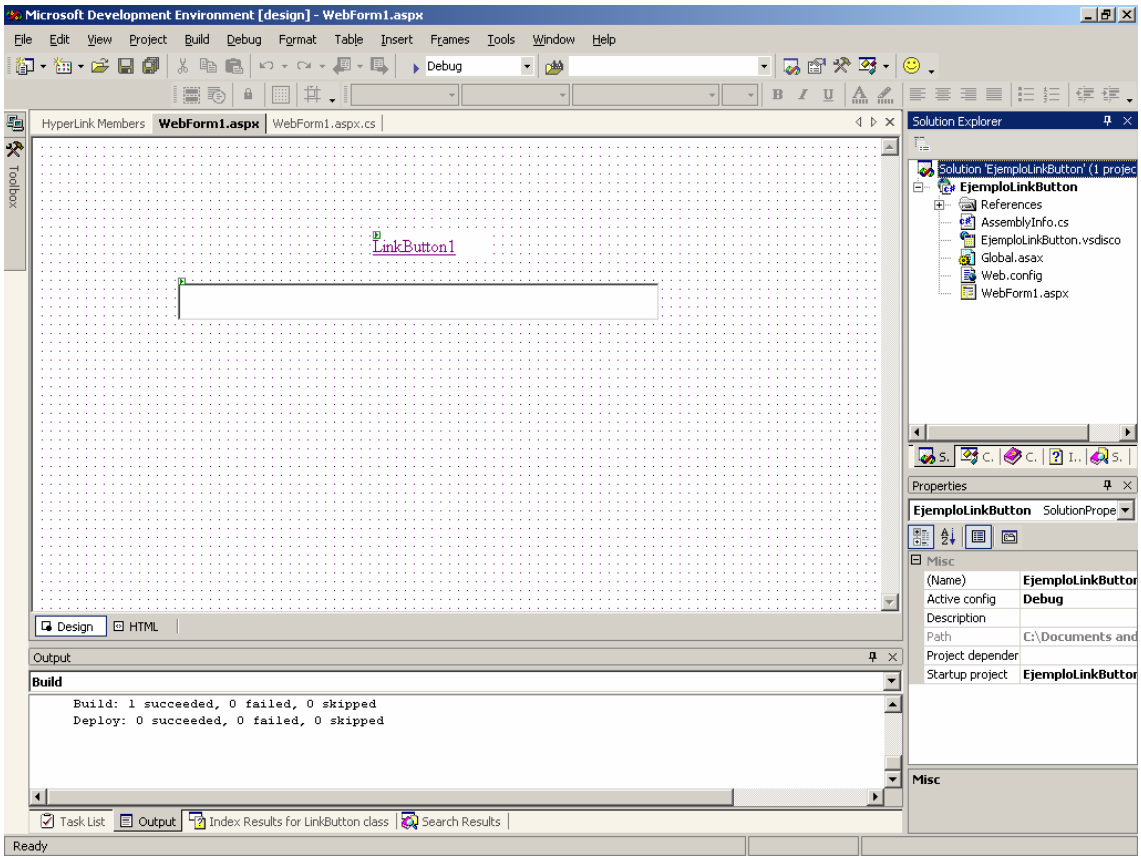

Figura 21.7. Vista diseño de la página WebForm con el control `LinkButton`.

El código HTML es:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="EjemploASPNETControls.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<HTML>

<HEAD>
<meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" Content="C#">
<meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
```

```
<meta                                                name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
</HEAD>

<body MS_POSITIONING="GridLayout" bgColor="#ffffff">

<form id="Form1" method="post" runat="server">

<asp:LinkButton id="LinkButton1" style="Z-INDEX: 101; LEFT: 307px;
POSITION: absolute; TOP: 88px" runat="server" Width="107px"
Height="19px">LinkButton1</asp:LinkButton>

<asp:TextBox id="TextBox1" style="Z-INDEX: 102; LEFT: 131px; POSITION:
absolute;      TOP:      130px"      runat="server"      Width="435px"
Height="34px"></asp:TextBox>

</form>

</body>
</HTML>
```

El código de servidor de respuesta al evento `Click` sobre `LinkButton1` es:

```
private void LinkButton1_Click(object sender, System.EventArgs e)
{
      TextBox1.Text = "Ha sido pulsado el LinkButton1, esto no implica
      un salto a un hipervínculo";
}
```

El resultado de ejecutar la aplicación (antes de pulsar `LinkButton1`) es:

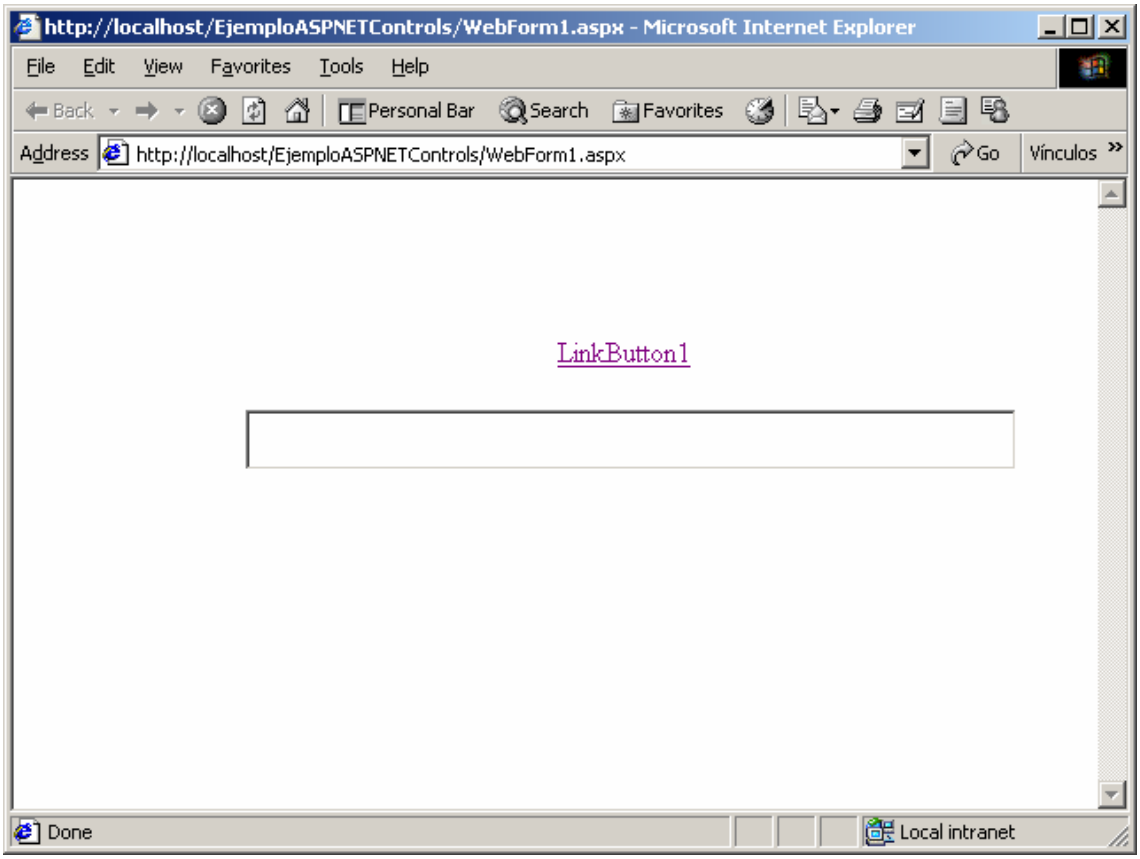

Figura 21.8. Invocación de la página WebForm con el control `LinkButton`.

Es interesante observar el código de cliente que se ha generado, por dos motivos:

- Observar cómo funciona en realidad el control `LinkButton`.
- Por extensión, profundizar en el funcionamiento real de los WebForms.

Para ver el código de cliente se ha de seleccionar la opción **Ver/Fuente** en el navegador (en este caso Internet Explorer):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
<HEAD>
<meta content="Microsoft Visual Studio 7.0" name="GENERATOR">
<meta content="C#" name="CODE_LANGUAGE">
<meta content="JavaScript (ECMAScript)" name="vs_defaultClientScript">
<meta          content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
</HEAD>

<body bgColor="#ffffff" MS_POSITIONING="GridLayout">

<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input  type="hidden"  name="__VIEWSTATE"  value="dDwtMjcyNjU2ODU1Ozs+"
/>

<a  id="LinkButton1"  href="javascript:__doPostBack('LinkButton1','')"
style="height:19px;width:107px;Z-INDEX:  101;  LEFT:  307px;  POSITION:
absolute; TOP: 88px">LinkButton1</a>

<input        name="TextBox1"        type="text"        id="TextBox1"
style="height:34px;width:435px;Z-INDEX:  102;  LEFT:  131px;  POSITION:
absolute; TOP: 130px" />

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />

<script language="javascript">
<!--
      function __doPostBack(eventTarget, eventArgument) {
            var theform = document.Form1;
            theform.__EVENTTARGET.value = eventTarget;
            theform.__EVENTARGUMENT.value = eventArgument;
            theform.submit();
      }
// -->
</script>

</form>
</body>
</HTML>
```

Como puede verse, el control `LinkButton1` es realmente un hipervínculo (`<a...>LinkButton1</a>`). Lo especial es que es un hipervínculo a una función `javascript` (“`__doPostBack`”). Que lo que hace realmente es ejecutar `submit`, realizando una respuesta POST hacia el servidor. Cuando el servidor reciba esa respuesta la analizará y sabrá que ha de ejecutar el método `LinkButton_Click`. Al

ejecutar tal método se generará una página web de respuesta al cliente en la que el texto `Ha sido pulsado el LinkButton1, esto no implica un salto a un hipervínculo` aparecerá en la caja de texto `TextBox1`.

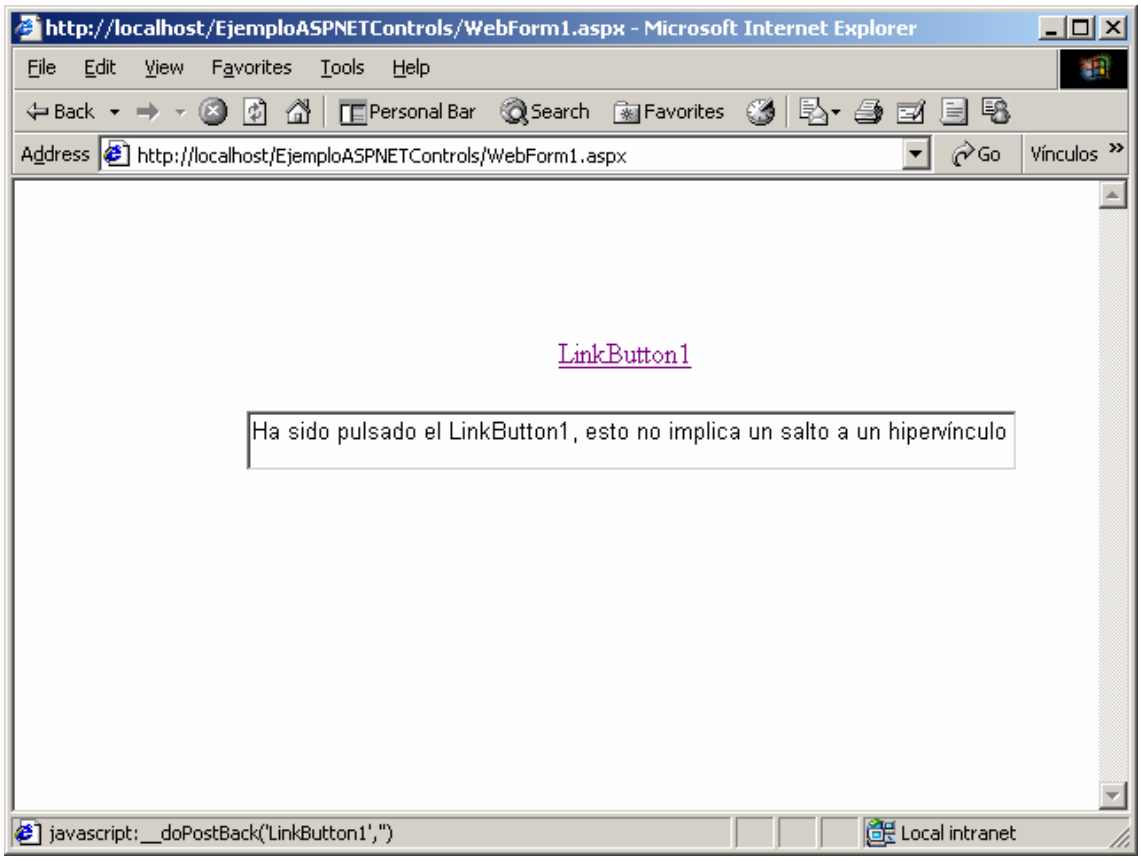

Figura 21.9. Resultado de lanzar el evento `Click` sobre el control `LinkButton` y ejecutar el manejador del evento.

Si se observa el código fuente de esta página se verá que es idéntico a la anterior, con la excepción del texto contenido en la caja de texto `TextBox1`.

De lo visto debe extraerse una idea importante. Una aplicación ASP.NET suele estar formada por una página Web específica (`.aspx`) y un código (`.aspx.cs`). Al invocar la página desde un cliente este recibe no la página pedida, sino una página estándar (HTML + scripts en JavaScript) generada a partir de la pedida y su código asociado.
En este caso, el control `LinkButton1` en la página `.aspx` es:

```
<asp:LinkButton id="LinkButton1" style="Z-INDEX: 101; LEFT: 307px;
POSITION: absolute; TOP: 88px" runat="server" Width="107px"
Height="19px">LinkButton1</asp:LinkButton>
```

y el resultado en la página estándar enviada al cliente es:

```
<a id="LinkButton1" href="javascript:__doPostBack('LinkButton1','')"
style="height:19px;width:107px;Z-INDEX: 101; LEFT: 307px; POSITION:
absolute; TOP: 88px">LinkButton1</a>
```

Clase ImageButton.

Esta clase representa un control que muestra una imagen y responde a la pulsación sobre la imagen del mismo modo que un botón (al pulsar un control `ImageButton` se generan los eventos `Click` y `Command`).

Si se utiliza el método de respuesta al evento `Click`, éste recibe como segundo parámetro `ImageEventArgs`, el cual posee dos propiedades llamadas `X` e `Y` que representan las coordenadas donde ha sido pulsada la imagen.

Para indicar la imagen que se muestra en un control `ImageButton` se utiliza la propiedad `ImageURL`, en la que se indica la URL correspondiente a la imagen.

Supónga que se desea modificar el ejemplo anterior (`LinkButton`) para añadirle un control `ImageButton` tal que al pulsarlo se muestren en la caja de texto las coordenadas donde ha sido pulsada la imagen.

La imagen que se muestra en el control `ImageButton` corresponde a la URL `file:///C:\WINNT\Grano de café.bmp`, aunque evidentemente, puede ser la que se desee.

El método manejador del evento `Click` es:

```
private         void         ImageButton1_Click(object         sender,
                               System.Web.UI.ImageClickEventArgs e)
{
      TextBox1.Text = "Se ha pulsado en el control ImageButton en el
      punto: " + e.X.ToString() + "," + e.Y.ToString();
}
```

El resultado de ejecutar la aplicación será:

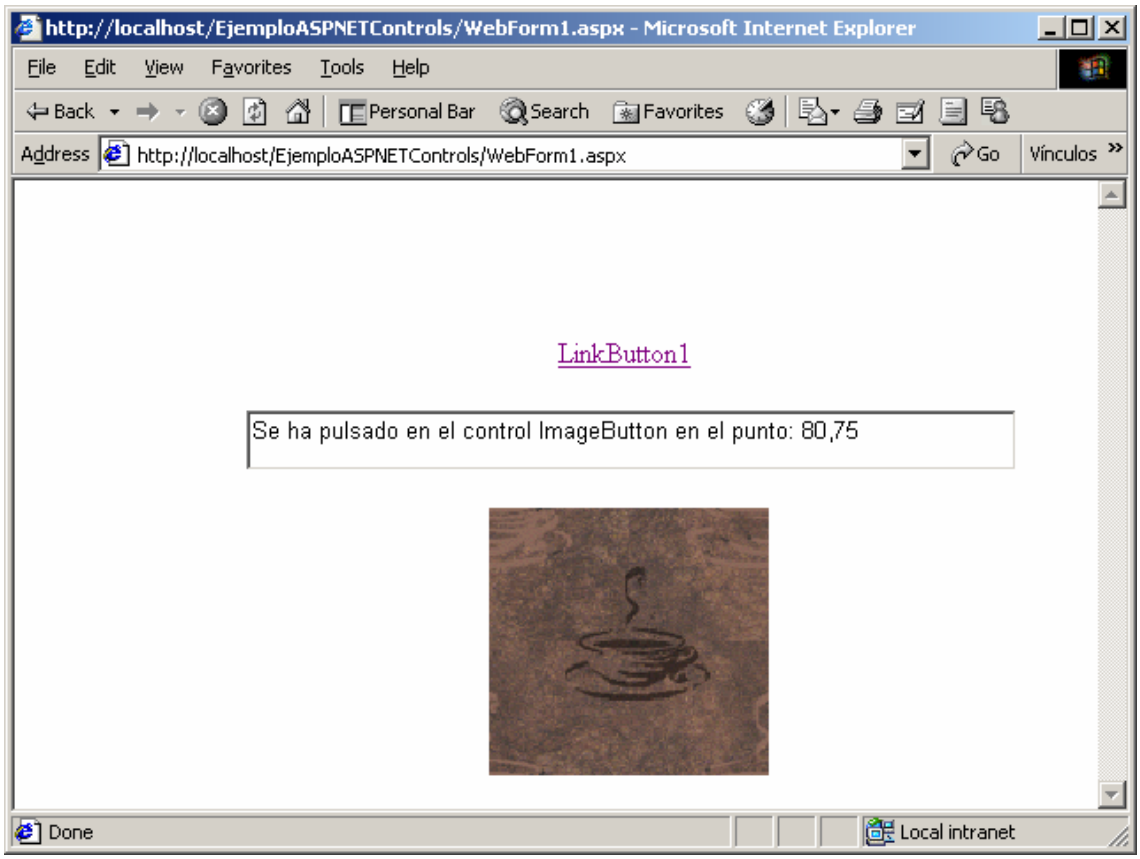

Figura 21.10. Resultado de lanzar el evento `Click` sobre el control `ImageButton` y ejecutar el manejador del evento.

Clase Hyperlink.

La clase `Hyperlink` corresponde a un control que permite realizar un salto a otra página Web. Las propiedades más importantes de este control son:

- `NavigateURL`: indica la dirección a la que salta el control `Hyperlink`.
- `Text`: es el texto que aparece en el control `Hyperlink`. En navegadores que soportan `ToolTips`, la propiedad `Text` corresponde también al texto que se muestra en el `ToolTip`.
- `ImageURL`: es la URL de la imagen que aparece en el control `Hyperlink`. Si `Text` e `ImageUrl` tienen valor a la vez, prevalece `ImageURL`.

A continuación se muestra un ejemplo básico de una página `.aspx` con un control `Hyperlink`.

```
<%@ Page Language="C#" %>
<html>
<head>

</head>
<body>

  <h3><font face="Arial">Ejemplo HyperLink</font></h3>

   Pulsar Hipervínculo:<br>
```

```
   <asp:HyperLink id="hyperlink1"
                  ImageUrl="imagenes/im1.jpg"
                  NavigateUrl="http://www.microsoft.com"
                  Text="Web Site official de Microsoft"
                  Target="_new"
                  runat="server"/>

</body>
</html>
```

Clase DropDownList.

Corresponde a un control de tipo "lista desplegable". Muestra una serie de elementos y permite seleccionar uno. Las propiedades más interesantes son:

- `Items`: es una colección a la que se pueden ir añadiendo los elementos de la lista. Cada elemento es un objeto de la clase `ListItem` (básicamente es un par texto-valor). Si se maneja esta propiedad, en diseño, desde el entorno de Visual Studio, éste muestra un cuadro de diálogo para añadir elementos a la lista.
-

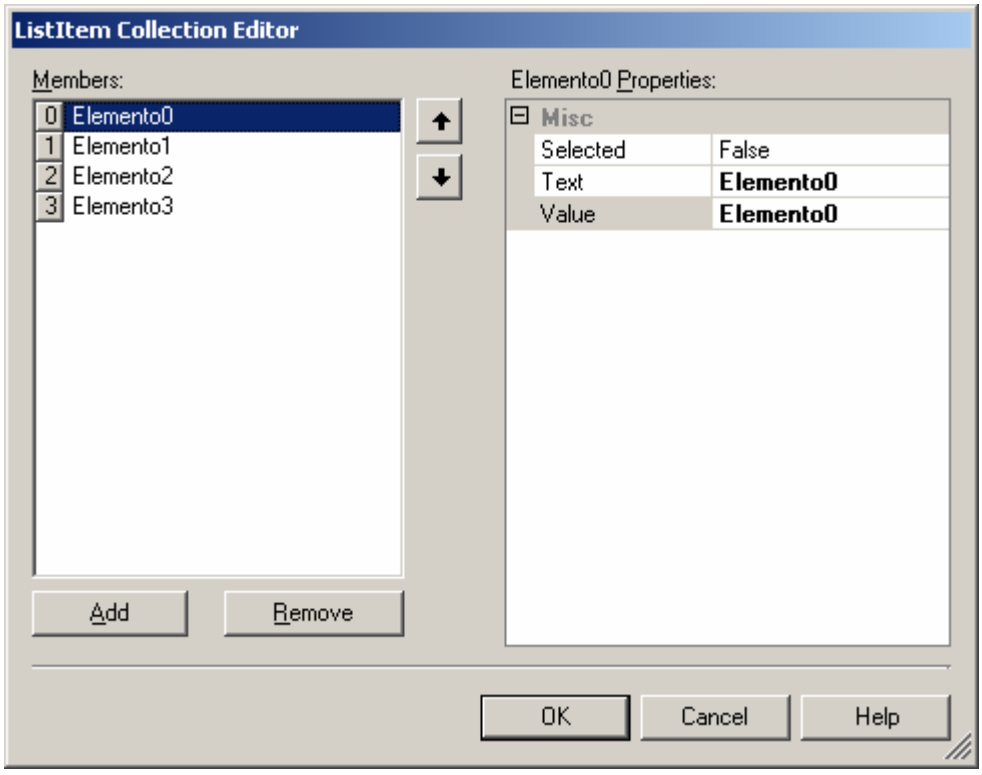

Figura 21.11. Cuadro de diálogo de edición de los elementos de un control `DropDownList`.

Si se observa el código de la página `.aspx` que contiene el `DropDownList`, se verá que cada elemento se ha insertado en la lista con la etiqueta `ListItem`.

```
<asp:DropDownList
id="DropDownList1" style="Z-INDEX: 101; LEFT: 51px; POSITION:
absolute; TOP: 38px" runat="server" Width="171px" Height="165px"
AutoPostBack="true">

<asp:ListItem Value="Elemento0">Elemento0</asp:ListItem>
<asp:ListItem Value="Elemento1">Elemento1</asp:ListItem>
<asp:ListItem Value="Elemento2">Elemento2</asp:ListItem>
<asp:ListItem Value="Elemento3">Elemento3</asp:ListItem>

</asp:DropDownList>
```

- `AutoPostBack`: si esta propiedad está a `true`, indica que se provocará un `submit`, es decir una petición `Post` hacia el servido cada vez que se seleccione un elemento del control `DropDownList`.
- `SelectedIndex`: es el índice (empezando desde 0) del elemento actualmente seleccionado en el control `DropDownList`.
- `SelectedItem`: es el elemento actualmente seleccionado.
- `DataSource`: representa el origen de datos del que se obtienen los datos del `DropDownList`.
- `DataMember`: se utiliza cuando el `DataSource` es un `DataSet`, es decir, conecta a una base de datos. Representa una tabla de datos.
- `DataValueField`: representa el campo del origen de datos del que se obtiene el valor de los elementos del `DropDownList`.

Cuando se selecciona un elemento de un control de tipo `DropDownList` se lanza un evento `SelectedIndexChanged` (manejado por el método `DropDownList_SelectedIndexChanged`).

A continuación se muestra un ejemplo básico en el que cada elemento seleccionado en un control del tipo `DropDownList` se muestra en una caja de texto. Para realizar este ejemplo se debe:

- añadir un control de la clase `DropDownList` a la página Web de la aplicación ASP.NET.
- utilizar su propiedad `Items` (o `DataSource`) para añadir los elementos a la lista.
- cambiar la propiedad `AutoPostBack` a `true` para que se el evento `SelectedIndexChanged` se genere en el servidor cada vez que se seleccione un elemento.
- codificar el método de respuesta al evento `SelectedIndexChanged`.

```
private    void    DropDownList1_SelectedIndexChanged(object    sender,
                                                System.EventArgs e)
{
      TextBox1.Text = DropDownList1.SelectedItem.ToString();
}
```

El resultado de ejecutar la aplicación es:

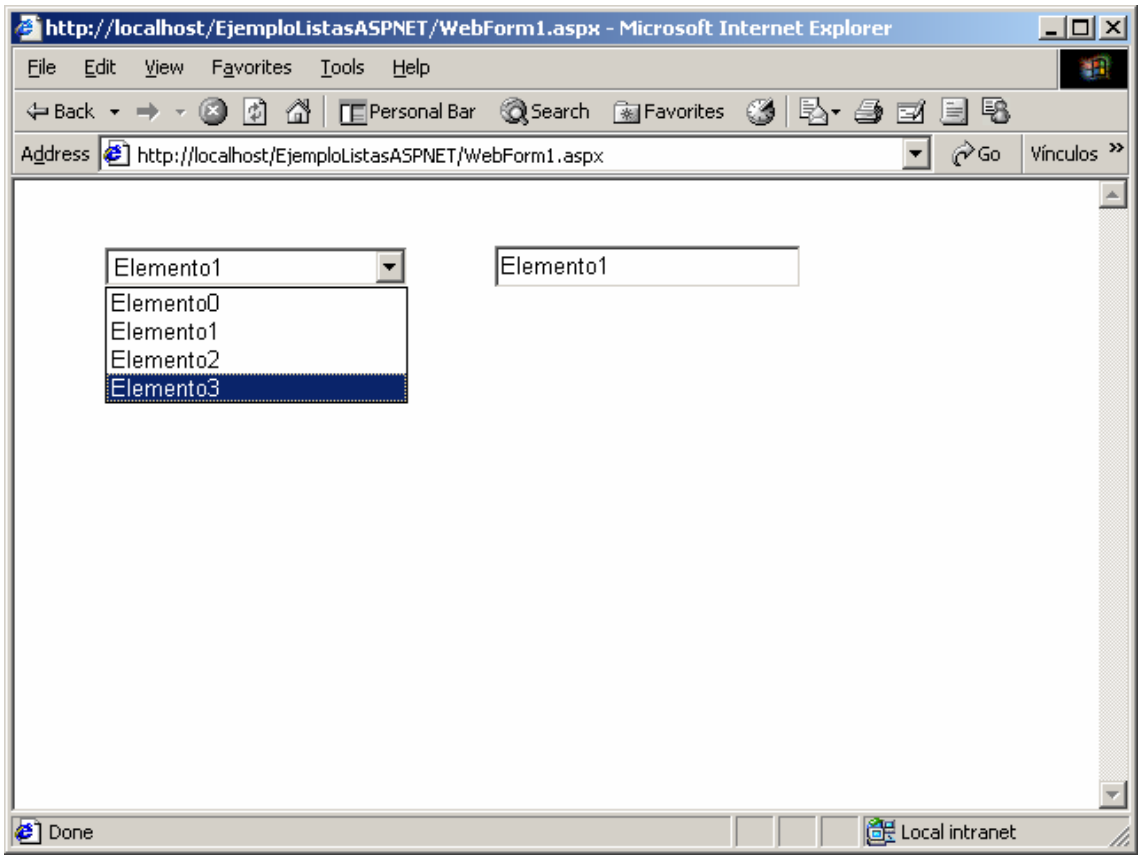

Figura 21.12. Resultado de lanzar el evento `SelectedIndexChanged` sobre el control `DropDownList` y ejecutar el manejador del evento.

Clase ListBox.

Representa un control típico de tipo lista . Es similar a `DropDownList`, con la salvedad de que permite selección múltiple y su altura no tiene porque ser de una fila. Aparte de las propiedades en común con `DropDownList`, un control de la clase `ListBox` tiene otras propiedades, como:

- `SelectionMode`: su valor puede ser `Single` o `Multiple`. Indica si se permite o no seleccionar más de un elemento a la vez.
- `Rows`: esta propiedad indica la altura en filas del control `ListBox`, es decir, que tamaño tiene (independientemente del número de elementos que realmente contenga).

Como ejemplo se añadirá un control de tipo `ListBox` y una caja de texto al ejemplo anterior. Su comportamiento será similar al del control `DropDownList` y la caja de texto ya existentes.

Clase DataGrid.

Representa un control que permite mostrar los campos de un origen de datos (`DataSource`) como columnas de una tabla, donde cada fila de la tabla (del `DataGrid`)

corresponde a un registro. El control `DataGrid` soporta selección, edición, borrado, paginación y ordenación.

Tanto esta clase, como la clase `DataList` y la clase `Repeater` se explicarán en detalle en la parte dedicada a ADO.NET.

Clase DataList.

Es similar al `DataGrid` pero menos potente. Muestra los datos en forma de lista, en lugar de en forma de rejilla o `grid`.

Clase Repeater.

Representa un control similar a `DataGrid` y `DataList`. Una diferencia importante es que permite aplicar una plantilla personalizada a los datos a mostrar. Por lo demás es más limitado que `DataGrid` y `DataList`.

Clase CheckBox.

Representa un control casilla de verificación. Permite al usuario seleccionarlo o no dejándolo en estado `true` o `false` respectivamente.
Si existen varios controles `CheckBox` en una página ASP.NET éstos son independientes entre sí, es decir, el estado de uno no determina el del resto.
En caso de tener varios `CheckBox` en una página ASP.NET puede ser interesante agruparlos en una `CheckBoxList`.
La clase `CheckBox` permite mayores posibilidades en cuanto al formato individual de un `CheckBox` concreto que lo que permite la clase `CheckBoxList`.
La propiedad más interesante de la clase `CheckBox` es `Selected`, que valdrá `true` si el `CheckBox` está seleccionado y `false` en caso contrario.
Otra propiedad interesante es `Appearance`, que permite indicar cual es la apariencia del `CheckBox` (botón, casilla...).
El evento generado al seleccionar o deseleccionar un `CheckBox` es `CheckedChanged`.

Clase CheckBoxList.

Representa una lista de casillas de verificación o `CheckBox`. Su propiedad más interesante es `Items` que representa una colección de `CheckBox`. Otra propiedad interesante es `RepeatDirection`, que indica si los controles `CheckBox` se alinearán vertical u horizontalmente. Cuando se selecciona un control `CheckBox` en una `CheckBoxList`, se produce un evento `SelectedIndexChanged`.

Clase RadioButtonList.

Es similar a `CheckBoxList`. Representa una lista de botones de opción. La particularidad de las listas de botones de opción es que sólo uno puede estar pulsado.

Clase RadioButton.

Representa un botón de opción. Si existen varios controles `RadioButton` en una página ASP.NET éstos son dependientes entre sí, es decir, sólo puede haber uno seleccionado.

El evento generado al seleccionar o deseleccionar un `RadioButton` es `CheckedChanged`.

En el siguiente ejemplo se completa el diseñado anteriormente para los controles `DropDownList` y `ListBox`. Se le añaden:

- Una `CheckBoxList` con dos `CheckBox`, el valor del primero determina si el control `DropDownList1` estará activo (`Enabled`) o no y el del segundo determina si el control `ListBox1` estará activo o no.
- Una `RadioButtonList` con dos `RadioButton`, el valor del primero determina si el control `TextBox1` estará activo (`Enabled`) o no y el del segundo determina si el control `TextBox2` estará activo o no.

En ambas listas se ha puesto la propiedad `AutoPostBack` a `true`, de modo que cualquier cambio en la selección provoque un evento `SelectedIndexChanged`.
El control de cuándo poner o no `Enabled` los controles `DropDownList1`, `ListBox1`, `TextBox1` y `TextBox2` se hace en los métodos de respuesta al evento `SelectedIndexChanged` de cada una de las listas.

```
private void CheckBoxList1_SelectedIndexChanged(object sender,
                                                System.EventArgs e)
{
   DropDownList1.Enabled = CheckBoxList1.Items[0].Selected;
   ListBox1.Enabled = CheckBoxList1.Items[1].Selected;
}

private void RadioButtonList1_SelectedIndexChanged(object sender,
                                                System.EventArgs e)
{
   TextBox1.Enabled = RadioButtonList1.Items[0].Selected;
   TextBox2.Enabled = RadioButtonList1.Items[1].Selected;
}
```

El resultado de ejecutar la aplicación es:

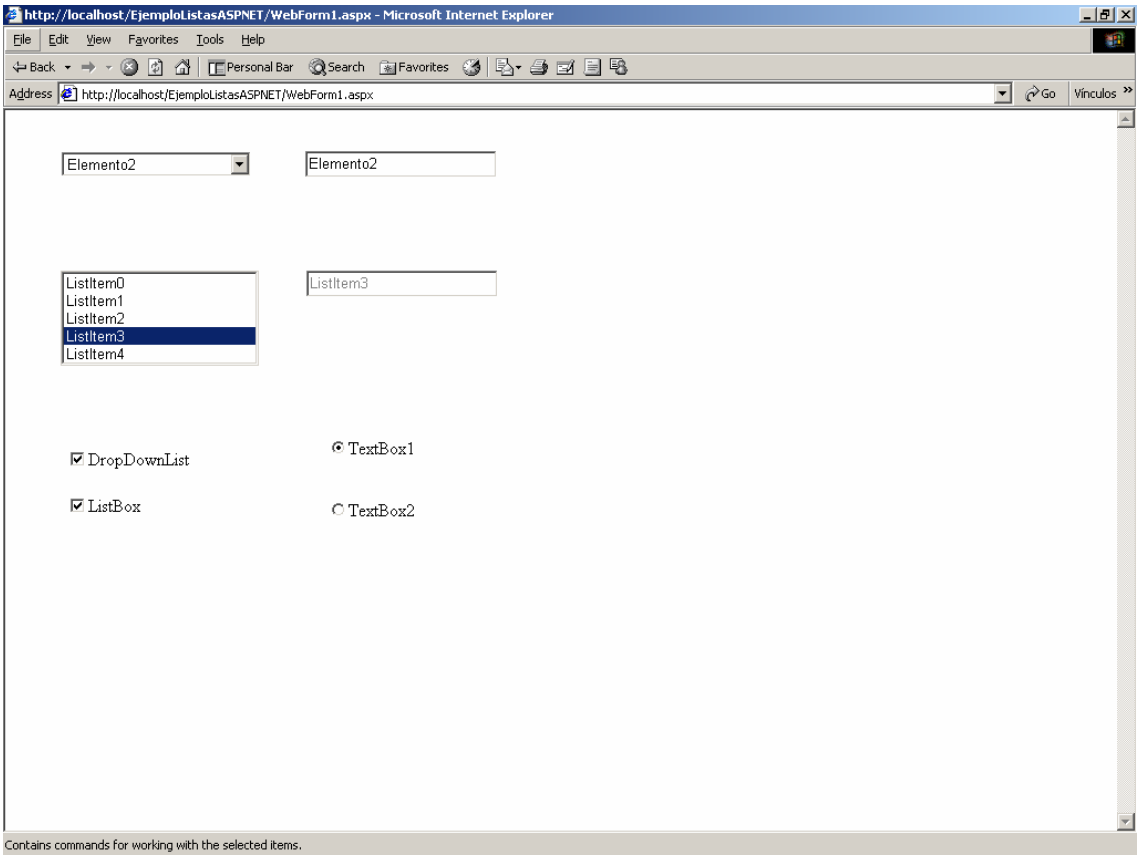

Figura 21.13. Ejecución de la aplicación. Desactivación de la caja de texto `TextBox2`.

Clase Image.

Representa un control que permite mostrar una imagen en una página Web. Las propiedades más interesantes son:

- `ImageURL`: contiene la URL de la imagen.
- `ImageAlign`: permite alinear la imagen respecto al resto de elementos de la página Web en la que está.
- `AlternateText`: permite indicar un texto a mostrar cuando la imagen no esté disponible (por el motivo que sea).

Clase Panel.

Esta clase representa un control que sirve de contenedor para otros controles. Toda página ASP.NET tiene un Panel principal; no obstante, se pueden crear más dentro de éste para por ejemplo, agrupar los controles.

Para añadir un `Panel` a una página Web simplemente hay que arrastrarlo desde la Caja de herramientas y dimensionarlo. A partir de ahí se le pueden ir añadiendo controles.

En el siguiente ejemplo se muestra cómo decidir si un panel es o no visible en base al estado de un `CheckBox`. El nombre del `Panel` añadido es `Panel1`, de modo que el código de respuesta al evento `CheckedChanged` es:

```
private void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
      Panel1.Visible = CheckBox1.Checked;
}
```

Y el resultado de ejecutar la aplicación debe ser (si el `CheckBox` está a `true`):

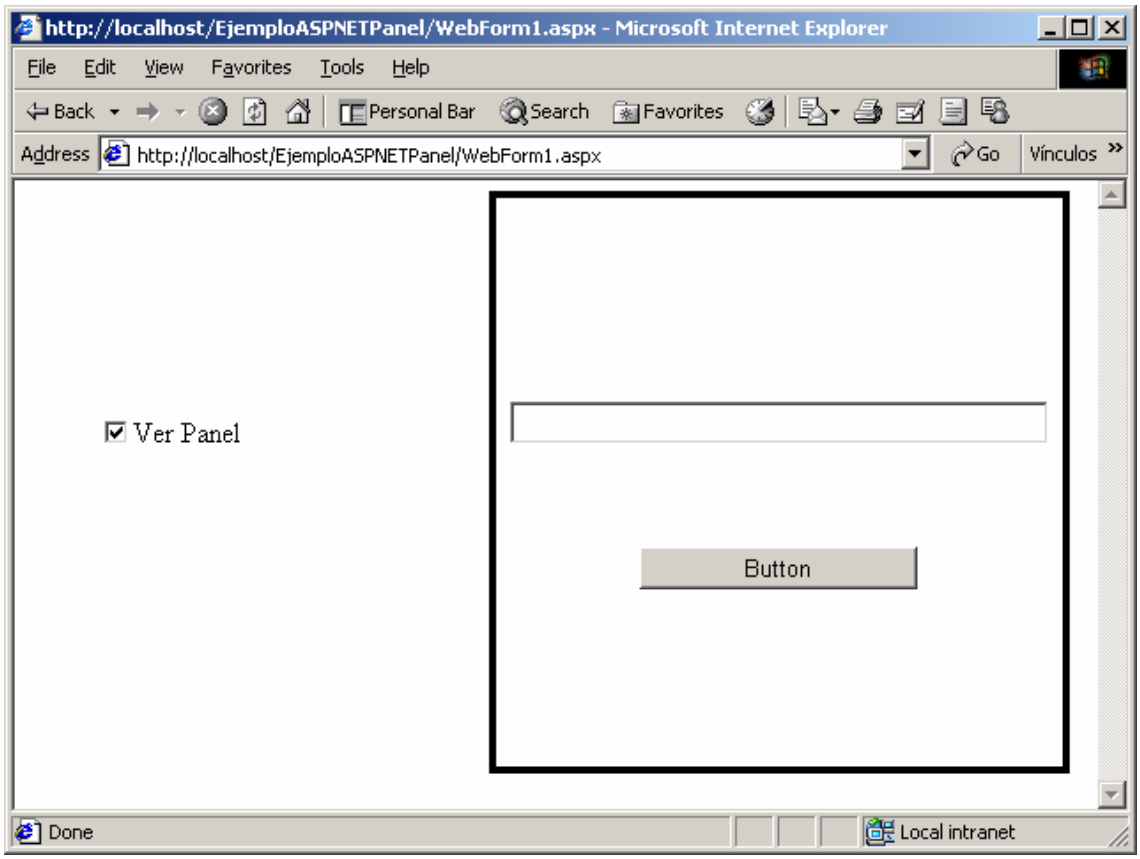

Figura 21.14. Ejecución de la aplicación. Propiedad `Visible` del `Panel` a `true`.

Si se pone el `CheckBox` a `false`:

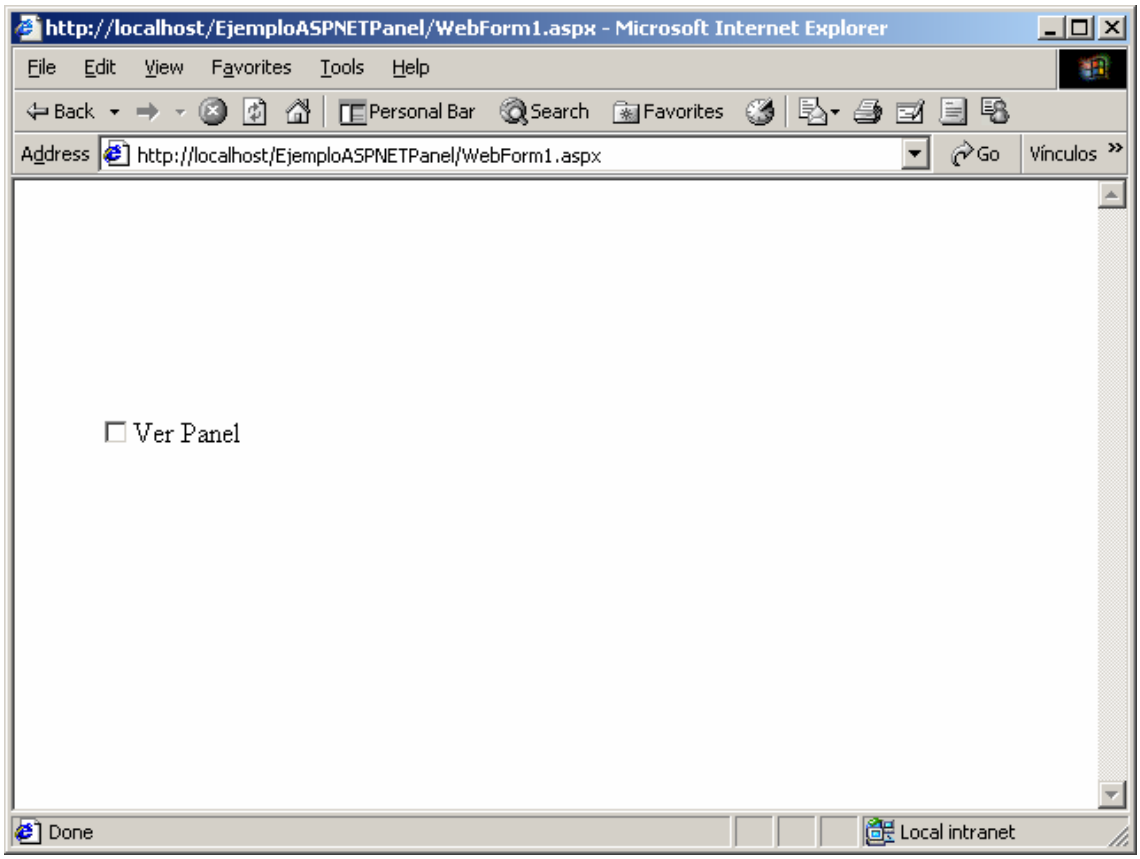

Figura 21.15. Ejecución de la aplicación. Propiedad `Visible` del `Panel` a `false`.

Clase PlaceHolder.

Esta clase representa un control similar a `Panel`, con la diferencia de que un `PlaceHolder` puede contener controles (generados dinámicamente) pero no es visible. Es sólo un contenedor.

Clase Calendar.

Representa un control que muestra un calendario de mes, que permite al usuario seleccionar fechas y moverse al anterior y/o posterior mes.

Por defecto, un control de tipo `Calendar` muestra los días del mes, cabeceras para los días de la semana, un título con el nombre del mes e hipervínculos para moverse al mes anterior y posterior.

Las propiedades más interesantes de un control `Calendar` son:

- `SelectedDate`: esta propiedad indica la fecha seleccionada. Un objeto de esta propiedad es de la clase `DateTime` y tiene por tanto todas sus propiedades (`Day`, `Month`, `Year`, `Date`, etc...).
- `SelectedDates`: esta propiedad contiene una colección con todos los días (o semanas..., depende de `SelectedMode`) seleccionados.
- `SelectedMode`: permite indicar si la selección será de un día, una semana o un mes.

- `FirstDayofWeek`: esta propiedad permite indicar qué día se pondrá en la primera columna de la semana (lunes, domingo...).

Al seleccionar un día (o una semana...) en el control Calendar se provoca un evento `SelectionChanged`, en el método asociado es donde se ha de meter el código de respuesta al evento.

En el siguiente ejemplo se muestra cómo hacer que se muestre la información sobre un día seleccionado en cuatro cajas de texto: la fecha, el año, el mes y el día.

El código de respuesta al evento generado al seleccionar un día en el calendario es:

```
private void Calendar1_SelectionChanged(object sender, EventArgs e)
{
      TextBox1.Text = Calendar1..SelectedDate.ToString();
      TextBox2.Text = Calendar1.SelectedDate.Year.ToString();
      TextBox3.Text = Calendar1.SelectedDate.Month.ToString();
      TextBox4.Text = Calendar1.SelectedDate.Day.ToString();
}
```

El resultado de ejecutar la aplicación es:

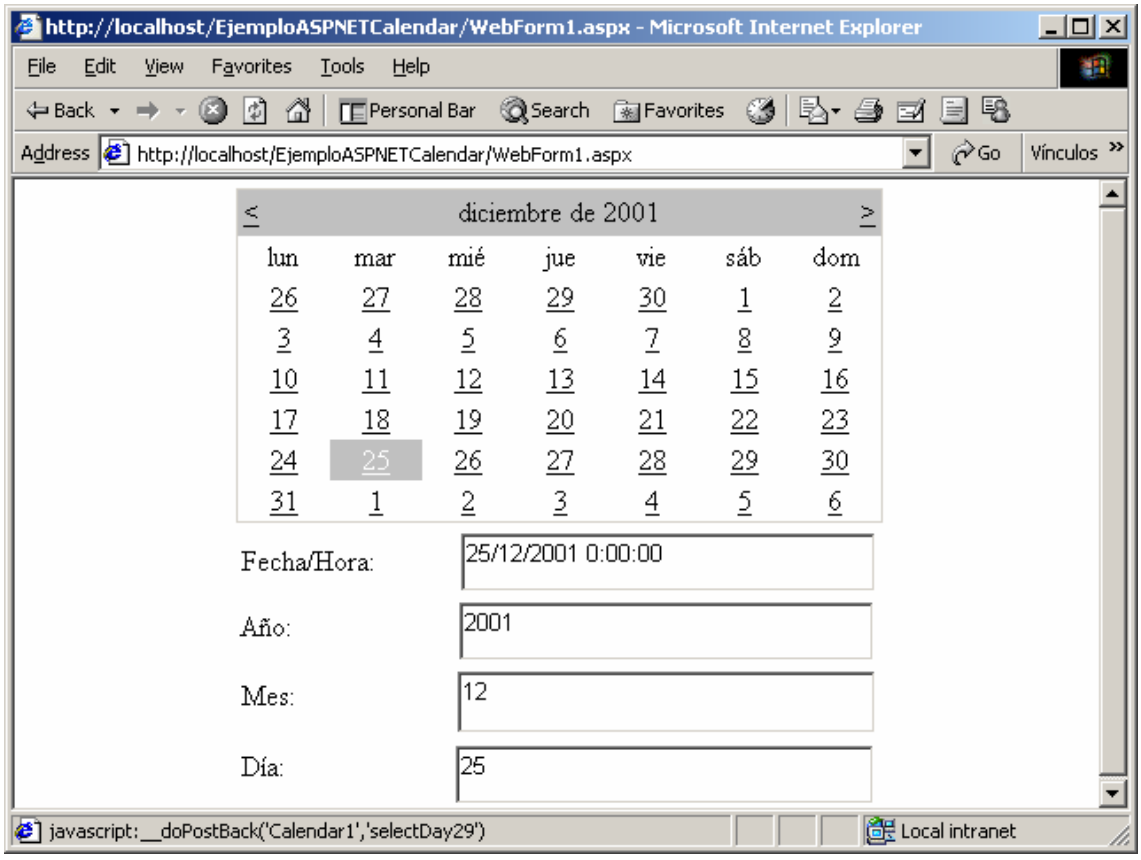

Figura 21.16. Lanzamiento del evento `Selection_Changed` sobre el control `Calendar` y resultado de ejecutar el método manejador.

Clase AdRotator.

Esta clase representa un control que permite mostrar un `banner` con imágenes e hipervínculos, cambiando de modo aleatorio su contenido, cada vez que la página es refrescada.
Las distintas imágenes con sus hipervínculos y otros datos se guardan en un fichero XML, indicado mediante la propiedad `AdvertisementFile` (al fichero se le llama fichero de `Advertisement`).

A continuación se muestra un ejemplo de un `AdRotator` que utiliza una lista de dos imágenes, basado en un documento XML.

El fichero `.aspx` (`WebForm1.aspx`) contiene:

```
<%@ Page Language="CS" %>
<html>

 <head>
 </head>

 <body>
    <form runat="server">

       <h3><font face="Arial">Ejemplo AdRotator</font></h3>

       <asp:AdRotator id="AdRotator1" runat="server"
            Target="_self"
            AdvertisementFile="XMLFich.xml"/>

    </form>
 </body>

 </html>
```

El fichero `XMLFich.xml` contiene:

```
<Advertisements   xmlns="http://schemas.microsoft.com/AspNet/AdRotator-
Schedule-File">

     <Ad>

         <ImageUrl>elipse.jpg</ImageUrl>
         <NavigateUrl>http://www.microsoft.com</NavigateUrl>
         <AlternateText>Web Site Microsoft </AlternateText>
         <Impressions>80</Impressions>
         <Keyword>Topic1</Keyword>

     </Ad>

     <Ad>

         <ImageUrl>circulos.jpg</ImageUrl>
         <NavigateUrl>http://www.pandasoftware.es</NavigateUrl>
         <AlternateText>Panda Software</AlternateText>
         <Impressions>80</Impressions>
         <Keyword>Topic2</Keyword>

     </Ad>

 </Advertisements>
```

Si se copia la página `.aspx`, el fichero `XMLFich.xml` y los dos ficheros `.gif` a un directorio del servidor web (o se crea uno) puede probarse la página.
Cada vez que se refresque la página se cambiará la imagen del `AdRotator`.

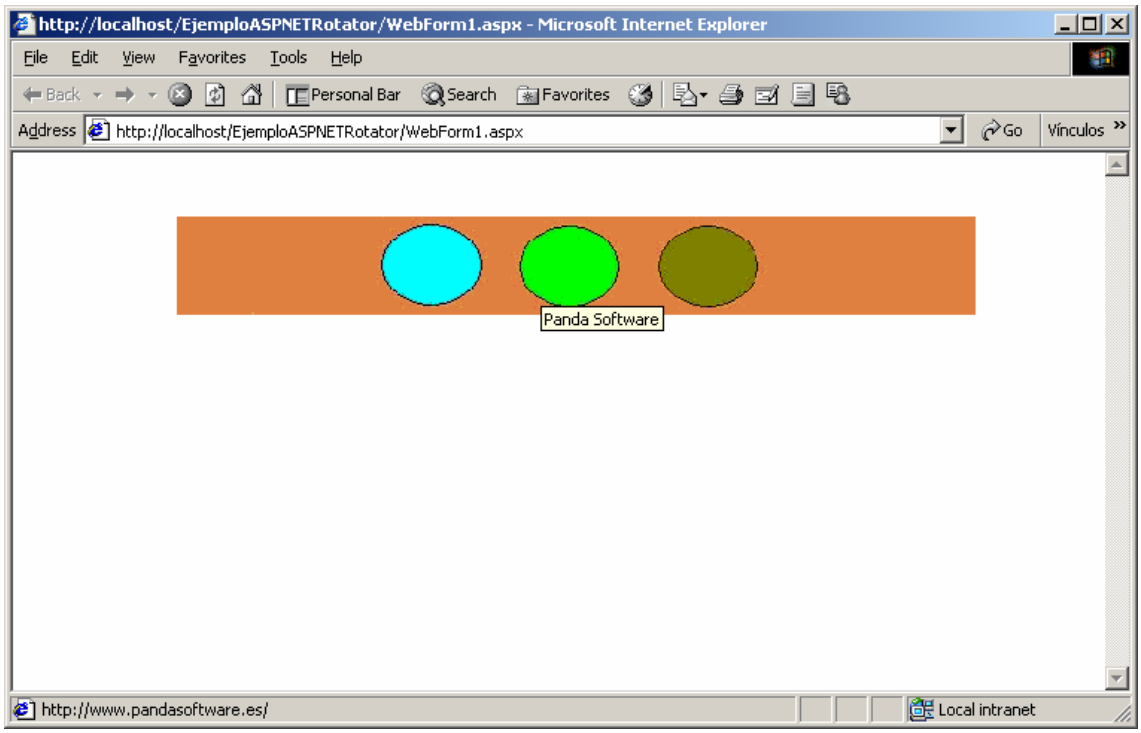

Figura 21.17. Ejecución de la aplicación. Tras actualizar la página el control `AdRotator` cambia de imagen. Al detener el puntero del ratón sobre la imagen se muestra un Texto obtenido de la elemento `AlternateText`.

Visual Studio tiene un asistente para la creación de un fichero XML asociado a un `AdRotator`. Para utilizarlo se pueden seguir los pasos que se indican:

- En el menú **Proyecto** elegir **Agregar nuevo elemento** y seleccionar **Archivo XML**:

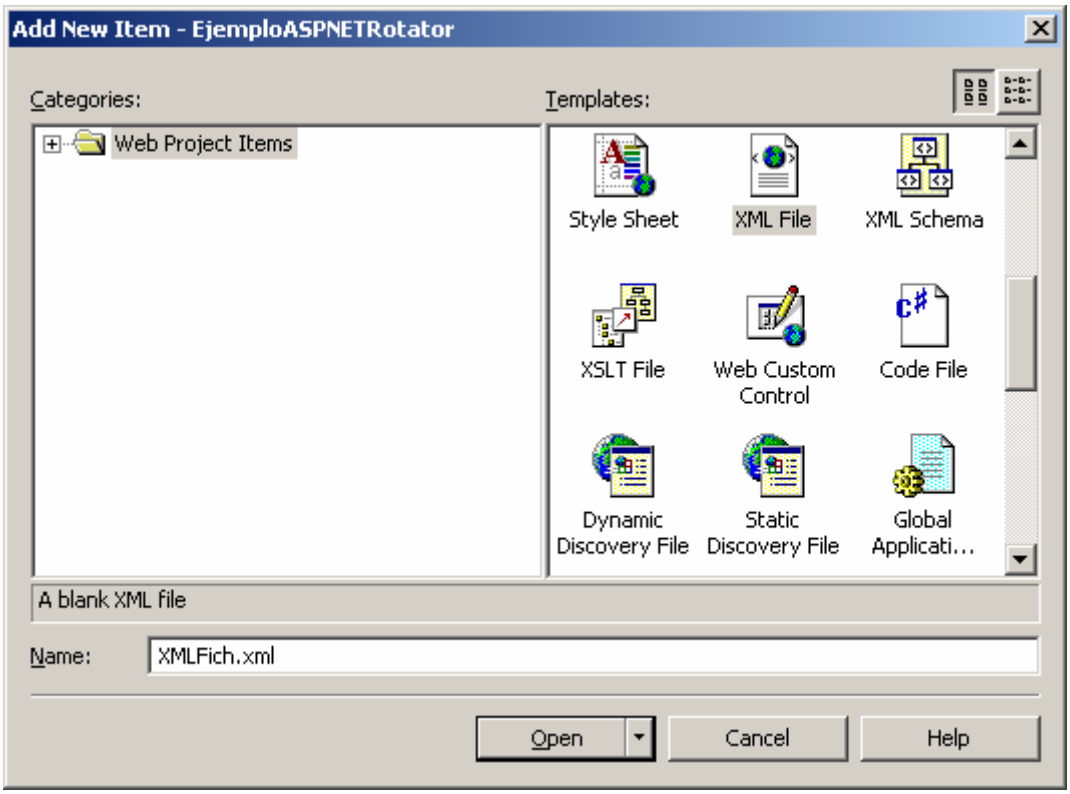

Figura 21.18. Adición de un fichero XML a la aplicación.

- Una vez se está en el editor del fichero XML, en la **ventana de propiedades** elegir, para la propiedad `TargetSchema`, el valor `ADRotator Schedule File`, con lo cual se le añadirá el elemento `<Advertisement>` al fichero XML.
- Añadir los elementos `<Ad>` que se desee al fichero. El único elemento obligatorio es `<ImageURL>`.
- Guardar el fichero XML (y las imágenes) en el directorio de la aplicación (realmente puede ser cualquier directorio válido del WebSite).

Una vez creado el fichero XML y el `AdRotator` se asocia el fichero XML al `AdRotator` mediante la propiedad `AdvertisementFile` del `AdRotator`. Visual Studio .NET permite hacerlo mediante un cuadro de diálogo.

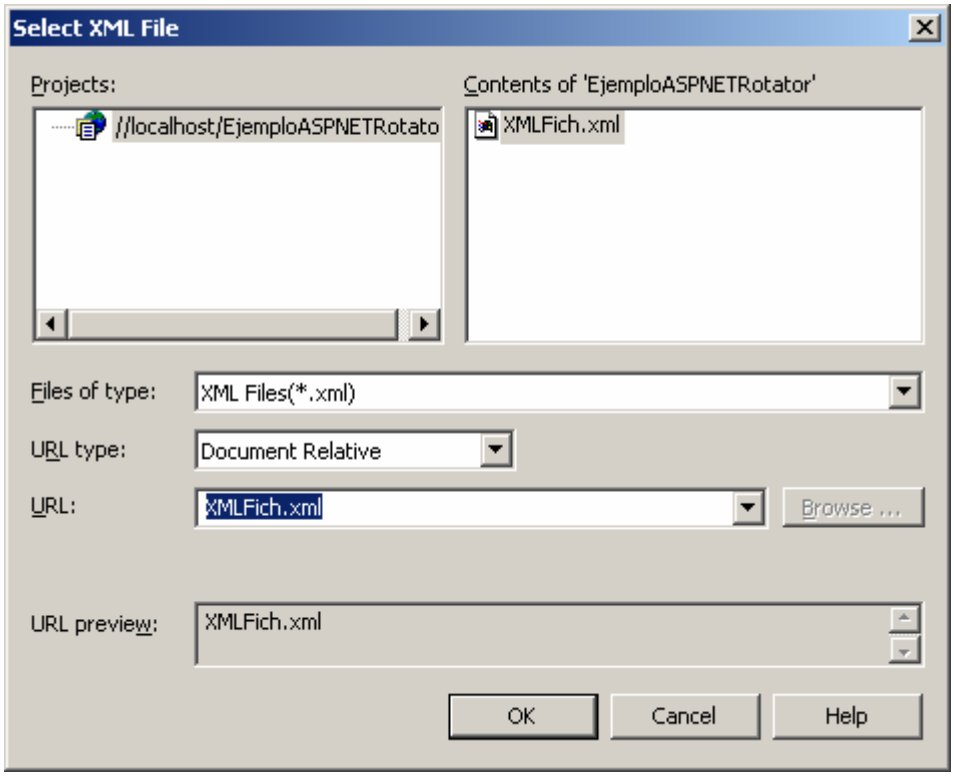

Figura 21.19. Selección de un fichero XML para la propiedad `AdvertisementFile`. El fichero debe haber sido añadido a la aplicación previamente.

A partir de aquí ya se puede probar (siempre que las imágenes indicadas en el fichero XML existan, por supuesto).

En caso de que haya problemas para ver las imágenes o el asistente anterior no muestre el fichero XML se puede intentar solucionarlo añadiendo las imágenes al proyecto (y el fichero XML si es que no se ha creado añadiéndolo al proyecto).

Clase Table.

Representa un control que permite construir una tabla HTML. La construcción puede ser mediante código (más potente) o en modo diseño.
Las propiedades más importantes de la clase `Table` son:

- `Rows`: es una colección de controles de la clase `TableRow`. Cada control de la clase `TableRow`, a su vez, contiene una colección de controles de la clase `TableCell` (cada control representa una celda).
- `BackImageURL`: permite especificar una imagen que se utilizará de fondo en la tabla
- `BorderStyle`, `BorderWidth`, `BorderColor`, `CellPadding`: estas propiedades permiten especificar el formato de los bordes de la tabla y la separación entre celdas.

Cuando se crea una tabla con el asistente de Visual Studio .NET, éste ofrece cuadros de diálogo para ir añadiendo y definiendo controles de la clase `TableRow` y controles de la clase `TableCell`.

Para añadir un control de tipo `TableRow` se ha de elegir la propiedad `Row` y seguir las indicaciones.

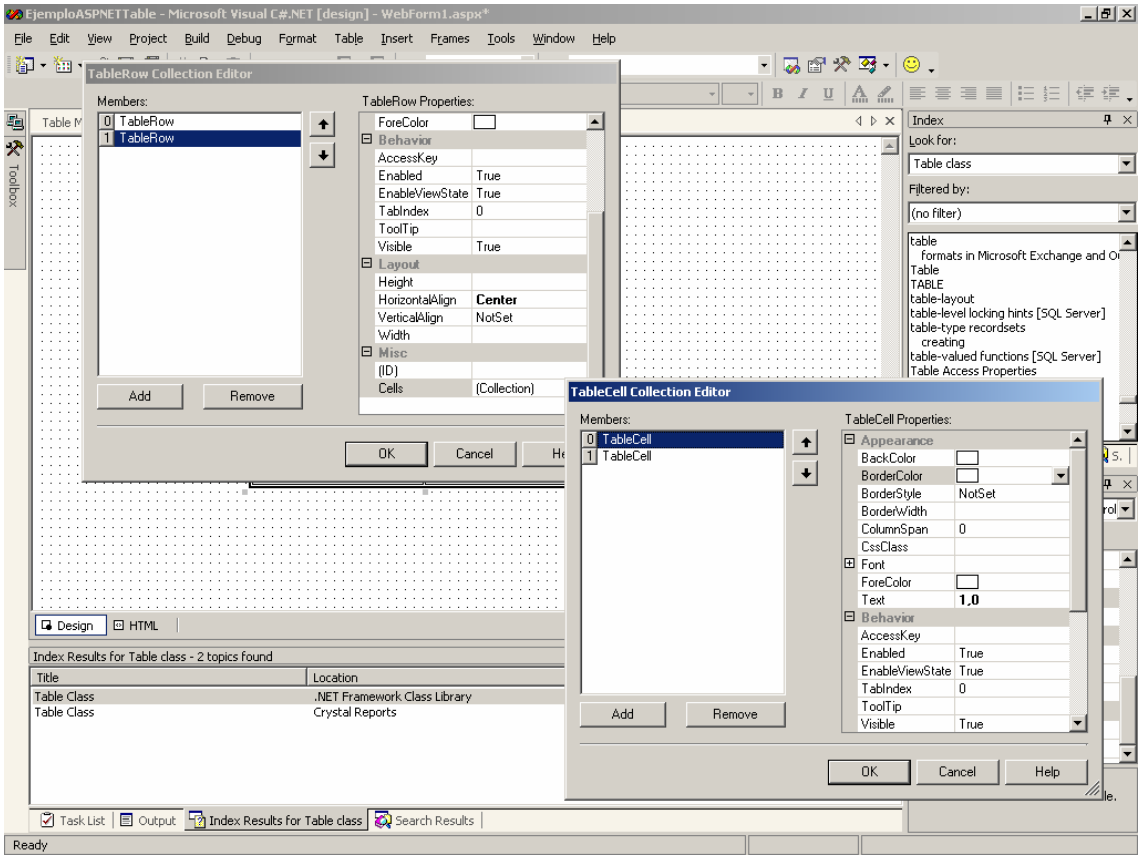

Figura 21.20. Cuadros de diálogo de edición de filas y columnas.

El resultado de ir añadiendo filas y celdas a una tabla en diseño queda como sigue.

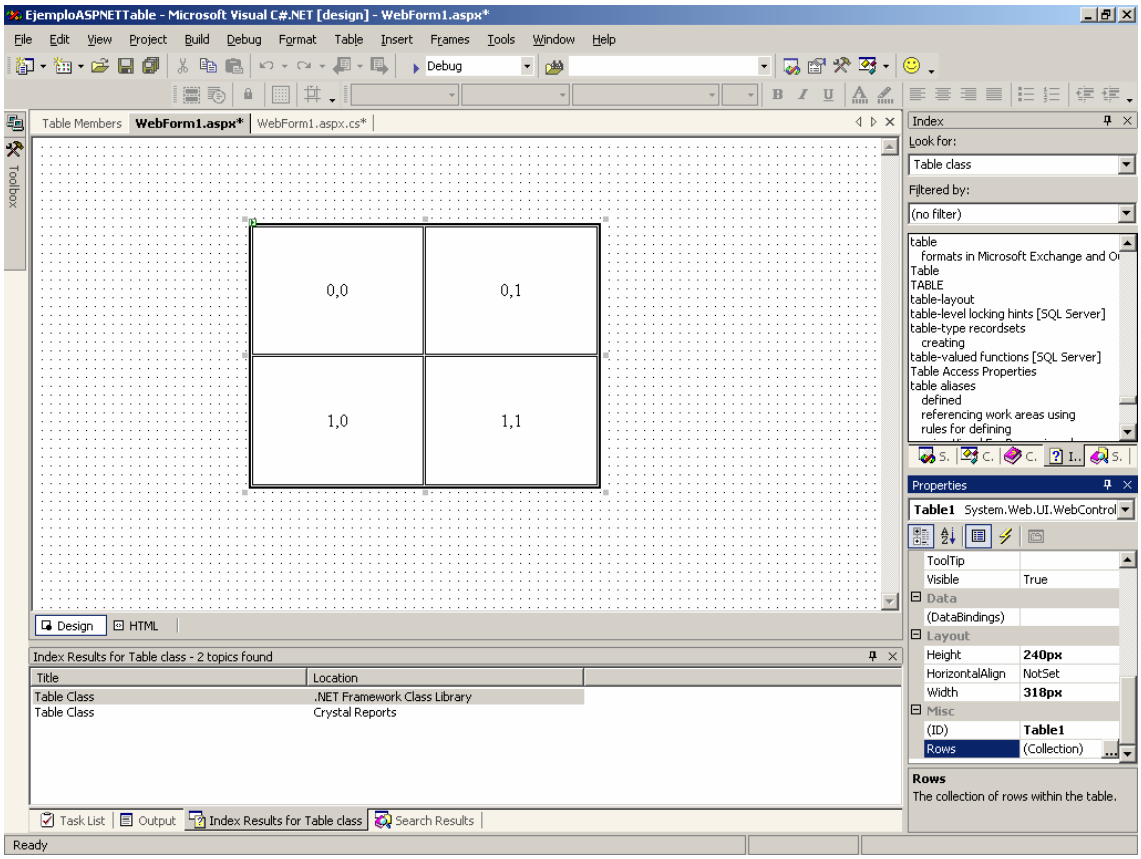

Figura 21.21. Vista diseño de la tabla.

Si se observa el código de la página `.aspx`, se verá la estructura ASP.NET real de la tabla:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false" Inherits="EjemploASPNETTable.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<HTML>

<HEAD>
<meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" Content="C#">
<meta name="vs_defaultClientScript" content="JavaScript (ECMAScript)">
<meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
</HEAD>

<body MS_POSITIONING="GridLayout">
<form id="Form1" method="post" runat="server">

<asp:Table id="Table1" style="Z-INDEX: 101; LEFT: 195px; POSITION:
absolute; TOP: 77px" runat="server" Width="318px" Height="240px"
BorderColor="Black" BorderStyle="Solid" CellPadding="1"
CellSpacing="1" BorderWidth="2px" ForeColor="Transparent"
GridLines="Both">

    <asp:TableRow BorderWidth="7px" BorderColor="White"
    BorderStyle="Dashed" HorizontalAlign="Center"
    ForeColor="Transparent">
```

```
            <asp:TableCell   BorderStyle="Solid"   BorderColor="Black"
            Text="0,0"></asp:TableCell>
            <asp:TableCell Text="0,1"></asp:TableCell>
      </asp:TableRow>
      <asp:TableRow HorizontalAlign="Center">
            <asp:TableCell Text="1,0"></asp:TableCell>
            <asp:TableCell Text="1,1"></asp:TableCell>
      </asp:TableRow>
      </asp:Table>

</form>

</body>

</HTML>
```

Es posible construir una tabla dinámicamente, mediante código. Haciéndolo de este modo las posibilidades son más amplias que en modo diseño. Por ejemplo:

```
<html>

<head>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
      // Generate rows and cells
      int numrows = 4;
      int numcells = 2;
      for (int j=0; j<numrows; j++)
      {
                TableRow r = new TableRow();
                for (int i=0; i<numcells; i++)
          {
                    TableCell c = new TableCell();
            c.Controls.Add(new LiteralControl("row " + j.ToString() +
            ", cell " + i.ToString()));
                        r.Cells.Add(c);
                }
                Table1.Rows.Add(r);
             }
         }
    </script>

</head>

<body>
  <h3><font face="Arial">Ejemplo de tabla construída desde el
  código</font></h3>
    <form runat=server>
        <asp:Table id="Table1"
         GridLines="Both"
         HorizontalAlign="Center"
         Font-Name="Arial"
         Font-Size="8pt"
         CellPadding=15
         CellSpacing=0
         Runat="server"/>
    </form>

</body>
```

```
</html>
```

Es importante saber que cuando una tabla o un control en general es creado en modo diseño forma parte de la página en la que está y seguirá existiendo tras cualquier `PostBack` que se haga hacia el servidor desde la página.
En cambio, si la tabla (o el control), ha sido generado dinámicamente, es decir, ejecutando un código y se ha añadido a la página que se ha enviado, en cuanto se haga un `Postback` se perderá a menos que sea regenerado en el mensaje de repuesta.

Clase XML

Representa al control XML. Este control no deriva de WebForms sino directamente de la clase `Control` (que a su vez deriva de `Object`). No obstante su `namespace` es `System.Web.UI.WebControls.Xml`.

El control XML permite escribir en una página Web un documento XML o el resultado de aplicar una transformación XSL a un documento XML.

Las propiedades más interesantes son:

- `Document` (no disponible en tiempo de diseño): permite indicar un objeto de la clase `System.XML.XMLDocument`, cuyo contenido se mostrará.
- `DocumentContent` (no disponible en tiempo de diseño): esta propiedad contiene y permite mostrar un `string` que ha de estar en formato XML.
- `DocumentSource`: esta propiedad contiene una referencia al documento XML que se desea mostrar.
- `Transform` (no disponible en tiempo de diseño): permite indicar un objeto de la clase `System.XML.XMLTransform`, que se aplicará al documento XML a mostrar, dándole un formato determinado por la página XSL que utilice el objeto `XMLTransform`.
- `TransformSource`: permite indicar directamente el fichero XSL que se utilizará para transformar el fichero XML a mostrar.

Es obligatorio que una y sólo una de las tres primeras propiedades tenga valor (`Document`, `DocumentContent`, `DocumentSource`), ya que de otro modo no habría documento XML que mostrar.

En cuanto a la transformación del documento XML, es opcional. No obstante, si se decide hacer, sólo una de las propiedades correspondientes (`Transform`, `TransformSource`) debe tener valor.

A continuación se va a realizar una aplicación `EjemploASPNETXML` que va a utilizar el control XML para mostrar la página `libros.xml`.

La página `libros.xml` contiene información sobre libros obtenida de una base de datos en formato XML.

```
<?xml version='1.0' ?>
<biblioteca>
```

```
      <libro  genero="novela"  fechapublicacion="1981"  ISBN="1-861003-
      11-0">
            <titulo>El Quijote</titulo>
            <autor>
                  <nombre>Miguel</nombre>
                  <apellido>de Cervantes</apellido>
            </autor>
            <precio>12.25</precio>
      </libro>
      <libro   genero="novela"   fechapublicacion="1967"   ISBN="0-201-
      63361-2">
            <titulo>Cien años de soledad</titulo>
            <autor>
                  <nombre>Gabriel</nombre>
                  <apellido>Garcia Marquez</apellido>
            </autor>
            <precio>10.5</precio>
      </libro>
      <libro  genero="poesía"  fechapublicacion="1991"  ISBN="1-861001-
      57-6">
            <titulo>Rimas</titulo>
            <autor>
                  <nombre>Gustavo Adolfo</nombre>
                  <apellido>Becquer</apellido>
            </autor>
            <precio>9.75</precio>
      </libro>
</biblioteca>
```

El primer paso es crear la aplicación y añadir el control XML a la página `WebForm1.aspx`. El siguiente paso es indicar el fichero XML que va a mostrar el control XML. Para ello ha de utilizar la propiedad `DocumentSource`.

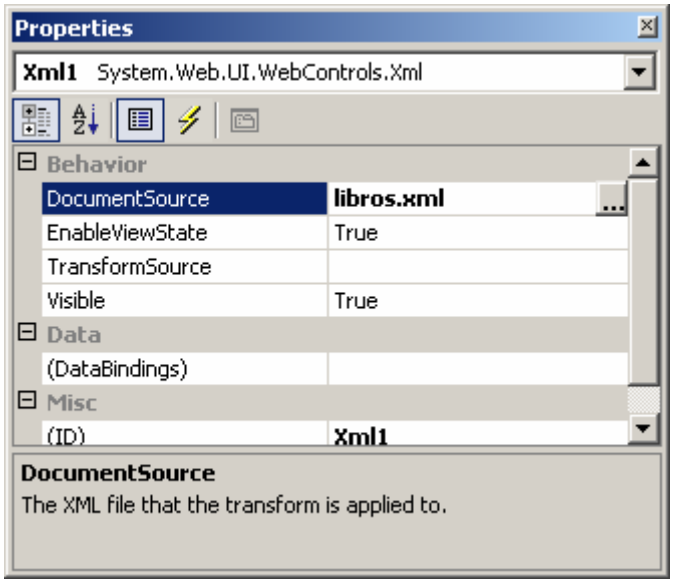

Figura 21.22. Propiedades del control XML.

Al pulsar el botón "...", muestra un cuadro de diálogo que permite elegir el documento XML a mostrar por el control XML (el documento XML tiene que haber sido añadido al proyecto).

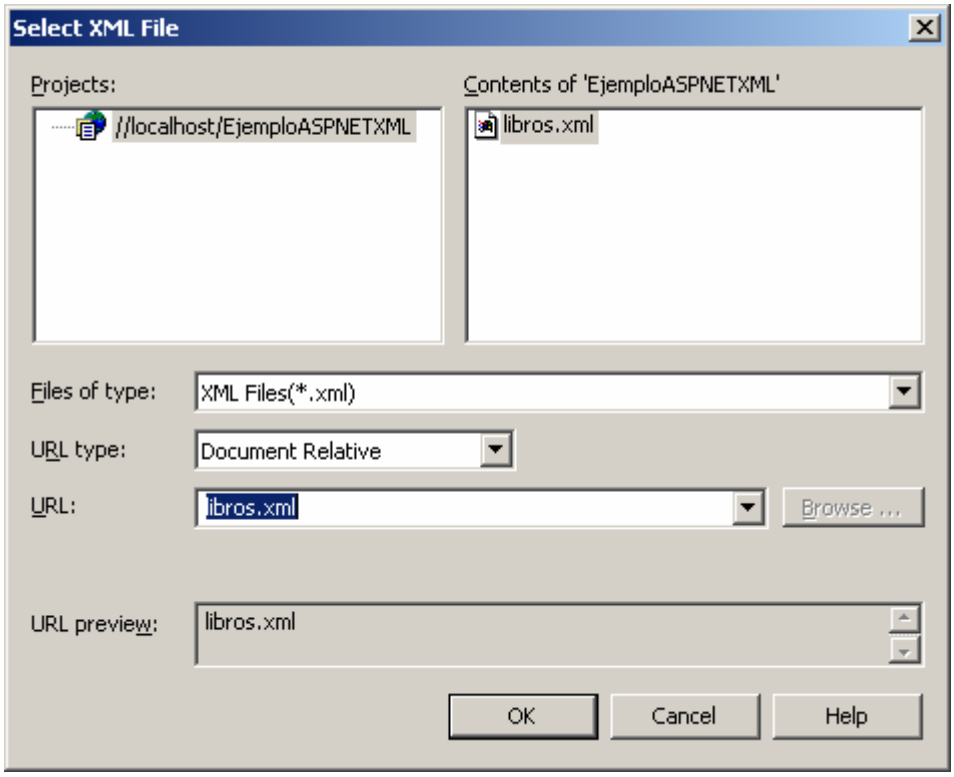

Figura 21.23. Cuadro de diálogo de selección del documento XML.

Una vez realizado este paso, el contenido de `WebForm1.aspx` tiene ya actualizado el elemento correspondiente al control XML:

```
<asp:xml id="Xml1" DocumentSource=”libros.xml”
runat="server"></asp:xml>
```

Ahora ya puede ser probada la aplicación. El resultado será:

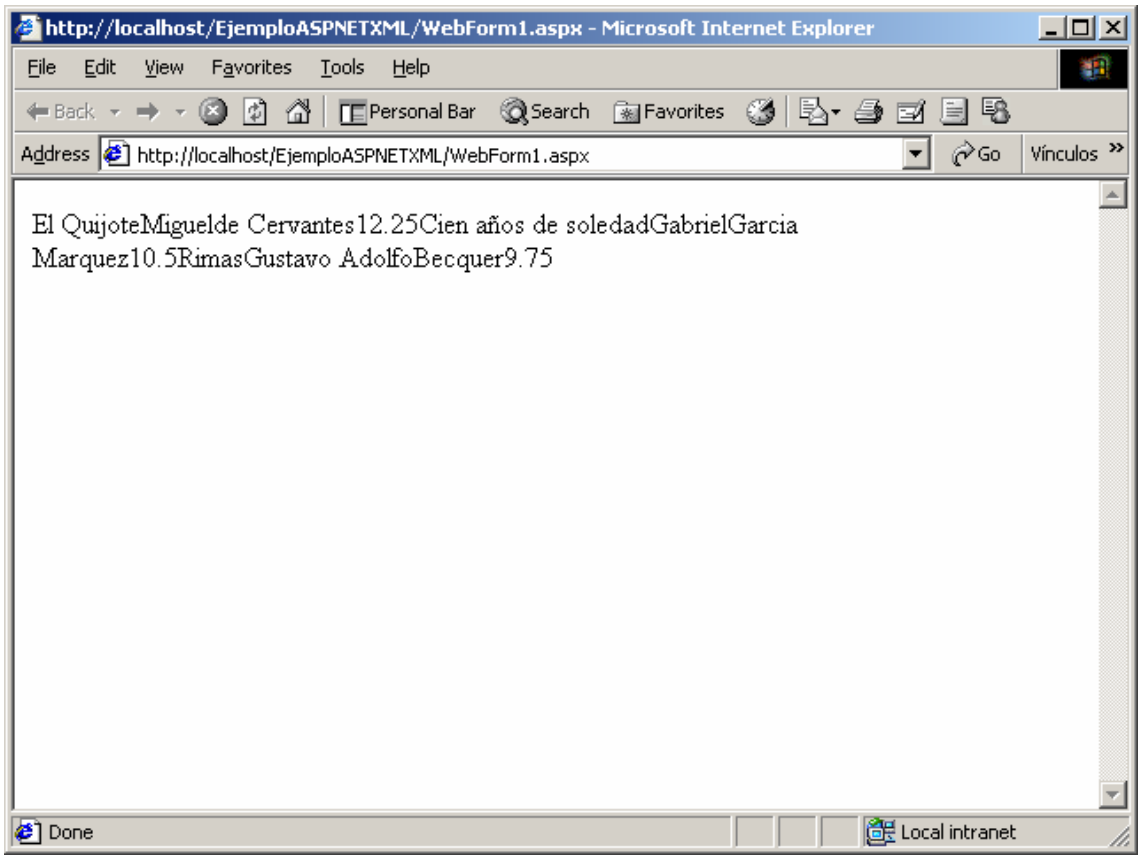

Figura 21.24. Invocación de la página desde el Internet Explorer. El control XML devuelve el fichero XML sin tratar y el Internet Explorer lo "parsea" (analiza) por defecto.

Como puede verse, el formato deja mucho que desear. El navegador muestra todos los elementos del documento XML seguidos, ya que no sabe cómo interpretar las etiquetas que tiene. No obstante, los datos son mostrados.

El control XML permite mejorar esta situación utilizando páginas XSL. Una página XSL tiene instrucciones sobre como hay que leer una página XML y de qué modo hay que transformar su contenido. Una utilidad enorme de las transformaciones XSL es poder convertir una página XML (de un lenguaje tipo XML, para ser más exacto) en otra HTML o WML, o cHTML, etc...

En este caso se va a utilizar una página (`libros.xsl`) que va a permitir obtener, a partir de la página `libros.xml`, una lista de libros en la que sólo se indicará el ISBN, título y precio. La página XSL `libros.xsl` contiene comentarios explicativos.

Se recomienda no utilizar tildes directamente en las páginas XSL y XML porque se puede solucionar con caracteres especiales.

```
<xsl:stylesheet                                          version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <!-- Esta primera parte define la estructura del documento
      transformado  -->
      <!-- La línea siguiente busca el nodo raíz del documento -->
      <xsl:template match="/">
```

```
            <html>
                  <head>
                        <title>Prueba xsl</title>
                  </head>
                  <body>
                        <!--La línea siguiente pide que se apliquen las
                        plantillas-->
                        <!-- Las plantillas que se indican más abajo
                        son las que se aplican -->
                        <!-- Se comienza por el elemento más cercano a
                        la raíz, que es biblioteca-->
                        <xsl:apply-templates />
                  </body>
            </html>
      </xsl:template>

      <xsl:template match="biblioteca">
            <!-- La plantilla biblioteca sólo pide aplicar la plantilla
            libro -->
            <xsl:apply-templates select="libro" />
      </xsl:template>

      <!-- La plantilla libro muestra el valor de ciertos elementos
      (ISBN, titulo, precio) de cada libro-->
      <<!-- Además inserta texto y algún otro elemento válido en HTML-
      ->
      <xsl:template match="libro">
            ISBN:
            <xsl:value-of select="@ISBN" />
            <br></br>
            Titulo:
            <xsl:value-of select="titulo" />
            <br></br>
            Precio:
            <xsl:value-of select="precio" />
            <br></br>
            <br></br>
            <text></text>
      </xsl:template>
</xsl:stylesheet>
```

Si se desea utilizar este documento XSL hay que indicarlo a través de la propiedad `TransformSource` del objeto XML. En la página `WebForm1.aspx`, la entrada correspondiente al control quedará:

```
<asp:xml             id="Xml1"             DocumentSource=”libros.xml”
TransformSource=”libros.xsl” runat="server"></asp:xml>
```

En este caso, el resultado de ejecutar la aplicación será.

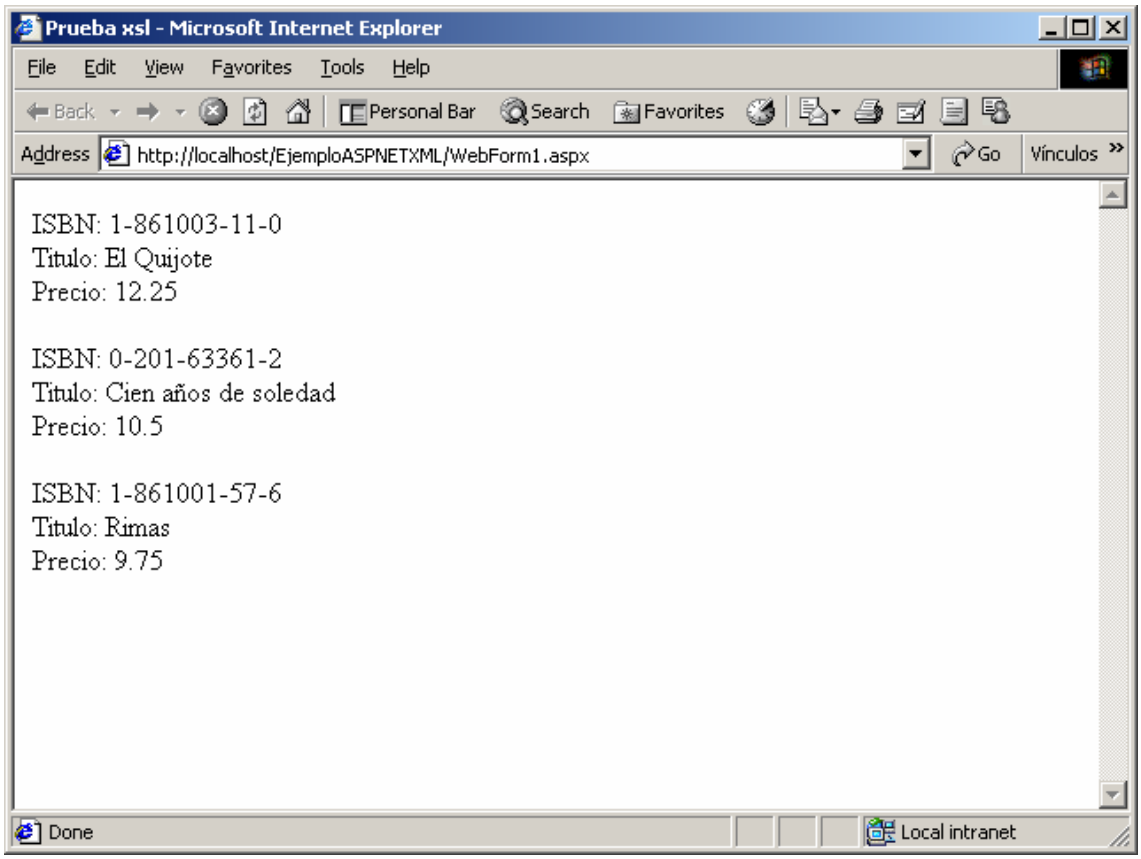

Figura 21.25. Invocación de la página desde el Internet Explorer. El control XML tiene una página XSL asociada y devuelve el fichero XML convertido a HTML.

Como puede verse, el formato está bastante mejor.

Hasta aquí, todo el ejemplo se ha hecho en modo diseño. No obstante, es posible programar la creación y manejo de un objeto XML.

Por ejemplo: supónga que se desea inicializar un objeto XML `xml1` al cargar la página a la que pertenece (`Page_Load`) , indicando los documentos XML y XSL que utilizará. Hacerlo es tan sencillo como lo que sigue:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Xml;
using System.Xml.Xsl;

namespace EjemploASPNETXML
{
      public class WebForm1 : System.Web.UI.Page
      {
```

```
            protected System.Web.UI.WebControls.Xml Xml1;

            public WebForm1()
            {
                  Page.Init += new System.EventHandler(Page_Init);
            }

            private void Page_Load(object sender, System.EventArgs e)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(Server.MapPath("libros.xml"));
                  XslTransform trans = new XslTransform();
                  trans.Load(Server.MapPath("libros.xsl"));
                  Xml1.Document = doc;
                  Xml1.Transform = trans;

            }
...
...
...
}
```

Como puede verse es muy sencillo. Se crean los objetos `XmlDocument` y `XslTransform`, se inicializan y se asignan a las propiedades `Document` y `Transform` de `XML1` (el control) respectivamente.

Existe un modo sencillo de asociar un documento XML a un control XML y es indicando el contenido del documento en la propia página `.aspx` (donde está la entrada correspondiente al control). Este modo es menos utilizado.

```
<asp:xml             id="Xml1"            TransformSource=”libros.xsl”
runat="server"></asp:xml>
<biblioteca>
      <libro  genero="novela"  fechapublicacion="1981"  ISBN="1-861003-
      11-0">
            <titulo>El Quijote</titulo>
            <autor>...
      ...
      ...
</biblioteca>
</asp:xml>
```

Clase Literal

El control `Literal` muestra un texto estático en una página Web, de modo similar al control `Label`. La diferencia entre ambos es que el control `Literal` no permite aplicar estilos al texto que muestra.

Ejemplo:

```
<html>
<head>
</head>
<body>
   <form runat="server">
      <h3><font face="Arial">Literal Example</font></h3>
```

```
      <asp:Literal id=”Literal1”
           Text=”Hola Internet”
           runat="server"/>
   </form>
</body>
</html>
```

Clase CrystalReportViewer

Esta clase representa un control que proporciona una serie de métodos y propiedades que permiten manejar el visor de `CrystalReports` (`CrystalReportViewer`).
El `namespace` al que pertenece esta clase es `CrystalReports.Web`. Este `namespace` provee soporte para el visor Web de `CrystalReports` y sus clases asociadas.
`CrystalReports` es una herramienta de creación de formularios que se empezó a utilizar para diseñar formularios en Visual Basic. `CrystalReports` es ahora la herramienta estándar de creación informes en Visual Studio .NET. Permite crear presentaciones interactivas, con contenido de calidad. Con `CrystalReports` para Visual Studio .NET es posible hospedar `Reports` en plataformas Windows y Web y publicar `CrystalReports` como un servicio web de Reportes (informes).

Como puede verse, en la pestaña **Web Forms** existen una serie de controles que no se han comentado como `RequiredFieldValidator`, `CompareValidator`, `RangeValidator`, `RegularExpresionValidator`, `CustomValidator`, `ValidationSummary`. Estos son los llamados *controles de validación* y se explican en el siguiente capítulo.

Por supuesto, existen y pueden crearse otros controles (no sólo ASP.NET, sino WinForms Controls e incluso binarios COM, etc...). Esos controles pueden añadirse también a la Caja de herramientas (opciones **añadir pestaña** y **personalizar Caja de herramientas** a las que se llega pulsando el botón derecho del ratón sobre la **Caja de herramientas**).