Aplicaciones Web con Visual Basic 6

Cómo crear aplicaciones en tres capas con los nuevos módulos web

Sumario

Con los módulos web de Visual Basic 6 crear aplicaciones de servidor es una tarea sencilla. Le contamos cómo utilizarlos para crear contenido dinámico.

De entre las novedades que aportó en su día Visual Basic 6, una de las más interesantes es la posibilidad de desarrollar librerías que se ejecutan como aplicaciones de servidor con *Internet Information Server*. Desarrollar sedes que ofrecen contenido dinámico es, posiblemente, más fácil que nunca para los usuarios de Visual Basic 6. A lo largo de este artículo se expondrán los fundamentos necesarios para que pueda aprovechar este nuevo recurso en sus propios desarrollos.

Para iniciar el desarrollo de una aplicación IIS con Visual Basic 6 es preciso tener instalado *Internet Information Server*, si se trabaja con Windows NT, o *bien Personal Web Server*, en caso de que el sistema de desarrollo sea Windows 95/98. Sin este servicio disponible en el sistema, al seleccionar la opción de nuevo proyecto IIS en Visual Basic se obtendrá un error y el inicio de dicho proyecto no será posible.

Seleccionando la opción **Aplicación IIS** de la ventana **Nuevo proyecto**, tal y como se aprecia en la Figura 1, se creará un nuevo proyecto en el que tan sólo existirá un elemento: un módulo `WebClass`. Observe que el proyecto no es un ejecutable corriente sino una librería de enlace dinámico ActiveX, lo cual permitirá la integración con el modelo de objetos de *Internet Information Server*.

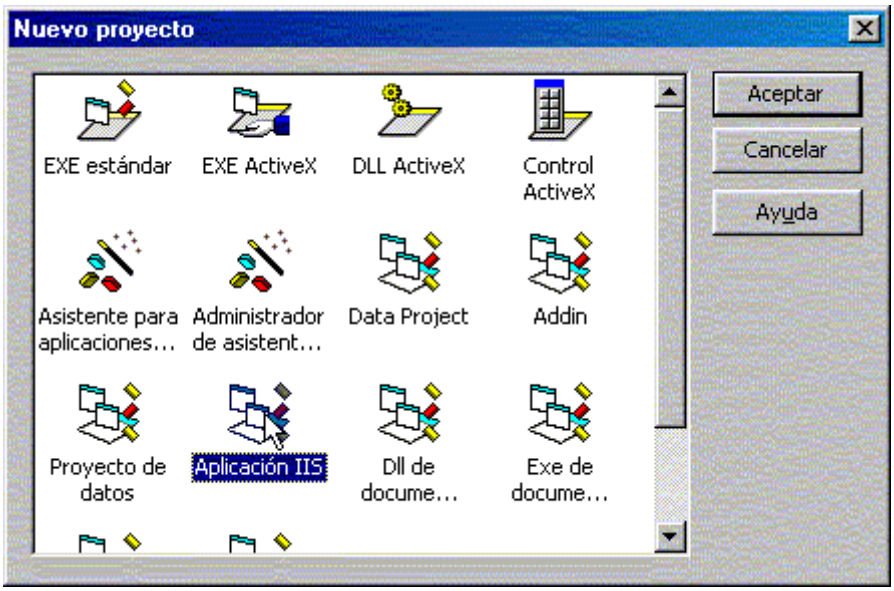

Figura 1. *Para poder iniciar una nueva aplicación IIS en Visual Basic 6 es preciso tener instalado Internet Information Server o Personal Web Server, dependiendo del sistema que esté utilizándose.*

Utilidad de una aplicación de servidor

¿Qué ventajas tiene una aplicación que se ejecuta en un servidor? ¿Cuál es la utilidad que tiene? Seguramente éstas son las preguntas que cualquiera podría plantearse ante una característica relativamente poco habitual hasta ahora. Lógicamente puede utilizarse una aplicación de este tipo para aportar contenido a una sede web cualquiera, pero no es ésta la única aplicación práctica. La mayor utilidad se encontrará, seguramente, en las *intranet* corporativas de las empresas.

Con una configuración de este tipo los módulos web de Visual Basic permiten crear aplicaciones que se ejecutan desde un cliente web cualquiera, puesto que se genera HTML puro, mientras la aplicación, que mantiene las reglas de negocio, se ejecuta en un servidor. Se obtiene, por lo tanto, una distribución/instalación más fácil de la aplicación y un mantenimiento más fácil. La instalación, en realidad, es prácticamente nula, puesto que el cliente no necesita ningún elemento adicional aparte del típico Internet Explorer o Netscape Communicator. El mantenimiento se efectúa sobre un solo equipo: el servidor.

Desde una aplicación web es posible acceder a orígenes de datos que pueden ser tanto locales como servidores de datos remotos. De esta forma es posible crear la típica aplicación en tres capas o niveles que tan de moda está en la actualidad: por una parte tenemos un cliente ligero, por otra una aplicación en un servidor web y por otra un servidor de datos.

En la Figura 2 se muestra un esquema de esta estructura de tres capas. El cliente, mediante el protocolo HTTP, lanza una solicitud al servidor web. Éste ejecuta el código asociado al `WebClass`, que será el que interprete la solicitud y, en caso necesario, acceda al servidor de datos usando ADO (*ActiveX Data Objects*). Obtenidos los datos, la aplicación genera un documento HTML y lo devuelve al cliente, que lo muestra al usuario.

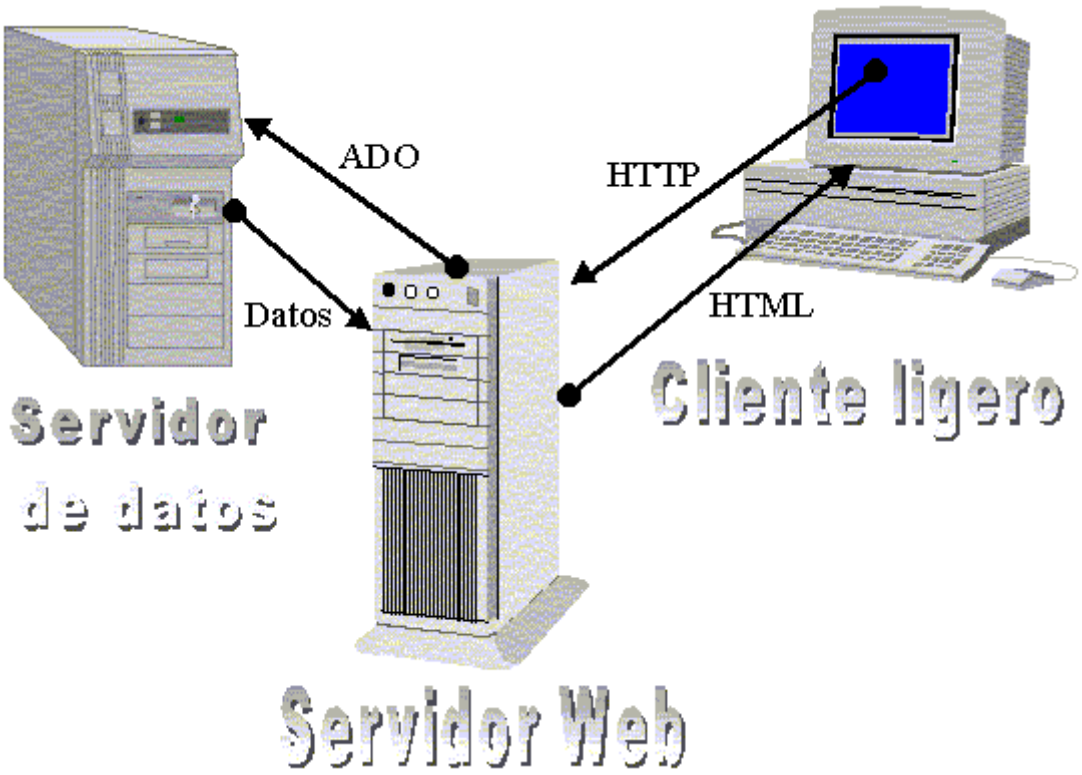

Figura 2. *Estructura típica de tres capas en la que se separa la interfaz de usuario de las reglas de negocio y el tratamiento de datos.*

El diseñador de módulos web

Una vez que sabemos para qué podemos usar una aplicación de servidor, adentrémonos en su desarrollo con los medios que incorpora Visual Basic 6. Iniciada una aplicación de este tipo, según se ha indicado anteriormente, en el proyecto existe un módulo `WebClass`. Al igual que los formularios, los entornos de datos, los informes y los controles, Visual Basic dispone de un diseñador específico para los módulos de clases web. Bastará con hacer doble clic sobre el módulo para abrirlo.

El diseñador de clases web (véase Figura 3) cuenta con una barra de botones y dos paneles. La barra de botones permite ejecutar las acciones más habituales que, en su mayoría, también están accesibles desde el menú emergente. El panel de la izquierda, una lista jerárquica en forma de árbol, muestra todos los elementos que contiene la clase, elementos conocidos como `WebItem`, así como los eventos asociados. El panel derecho, por su parte, es una vista de detalle del elemento seleccionado en el izquierdo. Como tendremos ocasión de ver, desde la vista de detalle es posible ver los atributos de un documento HTML que es posible conectar a eventos propios.

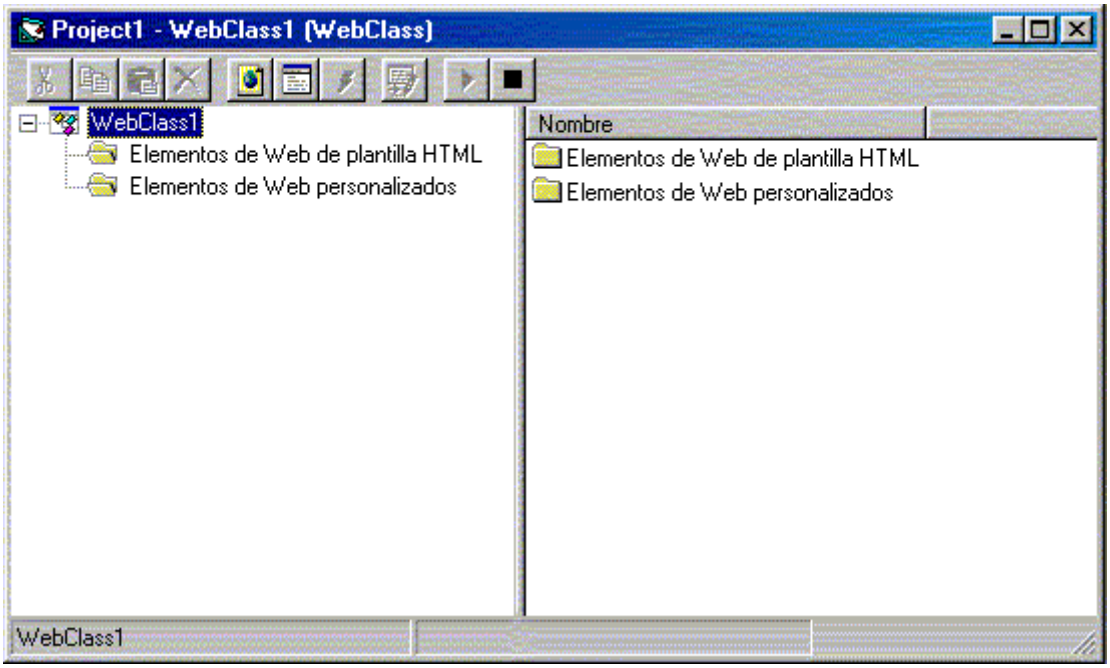

Figura 3. *El diseñador de clases web está dividido en dos paneles que permiten acceder a todos los elementos que forman parte del módulo.*

Como observará, ni el diseñador de clases web ni el propio Visual Basic ofrece ninguna herramienta para escribir código HTML. Será preciso, por lo tanto, crear los documentos desde cualquiera de los editores existentes, importando posteriormente el código que se incorporará al módulo. Antes de poder realizar esta operación, no obstante, es indispensable haber guardado el proyecto.

Un mismo proyecto Visual Basic, consecuentemente una misma librería ActiveX, puede contener múltiples módulos de clases web. Cada uno de ellos aparecerá como un elemento independiente en el gestor de proyectos, de tal forma que para editarlo se abrirá un nuevo diseñador de clases web. Cada módulo, además, cuenta con un conjunto de propiedades específicas e individuales que le diferenciarán de los demás módulos del proyecto.

Objetos en un módulo de clases web

Cada uno de los elementos que se manipula visualmente en el diseñador de clases web es un objeto. Como tales, cuentan con propiedades, métodos y eventos. Existen, además, otra serie de objetos fundamentales para que la aplicación pueda comunicarse con el servidor web que es, a fin de cuentas, quien en definitiva está *hablando* con el usuario.

Al igual que un formulario es la representación de un objeto `Form`, editada visualmente en el diseñador de formularios, una clase web es un objeto `WebClass`. Mediante las propiedades de este objeto podemos acceder a diversos objetos que contienen la solicitud del cliente, la respuesta que se le enviará, información acerca del cliente web, etc. De los tres métodos que implementa el más interesante es `URLFor`, que facilita la resolución de enlaces de forma dinámica. También genera eventos que permiten detectar el inicio y fin de una solicitud o un posible error.

Un objeto `WebClass` es un contenedor, si bien los elementos que puede contener no son componentes habituales como es el caso de un formulario. Los objetos que es posible insertar serán todos de la clase `WebItem`, si bien funcionalmente es posible distinguir dos categorías de estos objetos, como podremos ver después. Un objeto `WebItem` tiene por finalidad enviar código HTML al cliente, bien sea leyéndolo de una plantilla o generándolo dinámicamente en ese momento.

Además de los tres eventos con que cuenta inicialmente un objeto `WebItem`, y mediante los cuales es posible procesar marcas, enviar una respuesta y gestionar eventos de usuario, desde el diseñador de clases web es posible añadir todos los eventos adicionales que se precisen. Los eventos de un `WebItem` sirven, básicamente, para controlar las actuaciones del usuario sobre ciertos elementos del documento HTML como los botones de un formulario o los enlaces.

Compilación y ejecución

Cuando se compila un proyecto de este tipo, o bien cuando es ejecutado desde el entorno de Visual Basic, se genera automáticamente un documento ASP por cada uno de los módulos `WebClass` existentes. El nombre de dicho documento será el que se haya asignado a la propiedad `NameInURL` del correspondiente objeto `WebClass`. Los documentos ASP, como sabrá, son procesados en el servidor cuando un cliente los solicita, ejecutándose el código de *script* que contienen.

Si un cliente cualquiera quiere acceder a nuestra aplicación tendrá, por lo tanto, que introducir la URL correcta para acceder al documento ASP generado por Visual Basic. En ese momento IIS recuperará dicho documento, lo ejecutará y enviará al cliente la salida que proceda. El código a ejecutar, en el caso que nos ocupa, será similar al mostrado en el Listado 1. Utilizando el gestor de aplicaciones web, que es creado si procede, se pone en marcha un componente llamado `Principal` alojado en el servidor `WebPCWorld`. Éste es el nombre del proyecto, mientras que `Principal` es el nombre de la `WebClass`.

```
<%
Response.Buffer=True
Response.Expires=0

If (VarType(Application("~WC~WebClassManager")) = 0) Then
        Application.Lock

        If (VarType(Application("~WC~WebClassManager")) = 0) Then
                Set Application("~WC~WebClassManager") = _
            Server.CreateObject("WebClassRuntime.WebClassManager")
        End If

        Application.UnLock
End If

Application("~WC~WebClassManager").ProcessNoStateWebClass
```

```
                "WebPCWorld.Principal", _
                Server, _
                Application, _
                Session, _
                Request, _
                Response
%>
```

Listado 1. *Código ASP generado por Visual Basic para un módulo de clase web*

Al compilar el proyecto se añade al registro de configuraciones de Windows, concretamente a la rama `HKEY_CLASSES_ROOT\CLSID`, una entrada con el `CLSID` o identificador global del componente que se ha creado. En la Figura 4 puede ver la entrada correspondiente al componente `WebPCWorld.Principal`. IIS obtiene el nombre del componente a partir del código ASP, a continuación busca en el registro para encontrar la librería de enlace dinámico ActiveX, crea el componente y lo pone en marcha.

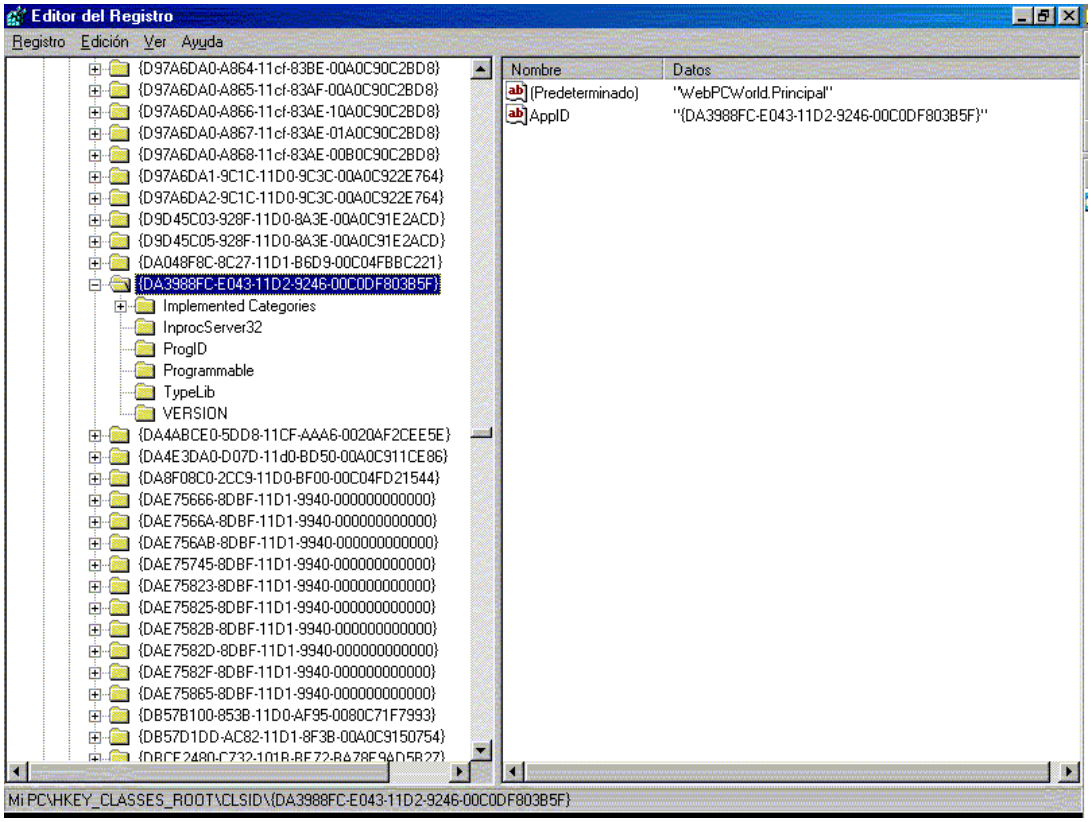

Figura 4. *Cada módulo* *WebClass* *tiene asociada una entrada en el registro de Windows, de tal forma que IIS puede encontrar la librería de enlace dinámico y ejecutar la aplicación.*

Observe al final del código ASP, en el Listado 1, que al ejecutar el componente se le facilitan una serie de objetos de *Internet Information Server* como `Application`, `Request` o `Response`. Estos objetos estarán disponibles para nuestro proyecto a través de las propiedades del mismo nombre del objeto

`WebClass`, permitiéndonos obtener la petición de cliente, recuperar información diversa, trabajar con *cookies* y, lógicamente, enviar la respuesta.

Envío de información al cliente

Existen dos métodos para enviar información a nuestro cliente: usando el método `Write` del objeto `Response` o bien el método `WriteTemplate` de un objeto `WebItem`. Para acceder al objeto `Response` hay que usar la propiedad del mismo nombre del objeto `WebClass`, mientras que para utilizar un `WebItem` se introducirá, como es lógico, el nombre que se le haya asignado en el editor de propiedades.

Si inicia un nuevo proyecto IIS, abre el diseñador de clases web y hace doble clic sobre el primer elemento del árbol, que es el objeto `WebClass`, verá abrirse el editor de código con un contenido similar al mostrado en la Figura 5. Se utiliza la propiedad `Response` para, mediante el método `Write`, enviar código HTML al cliente. Esto permite probar el funcionamiento del proyecto sin necesidad de dar ningún paso adicional, basta con añadir el módulo web y pulsar F5 para comprobar cómo se abre el cliente web por defecto y muestra el documento.

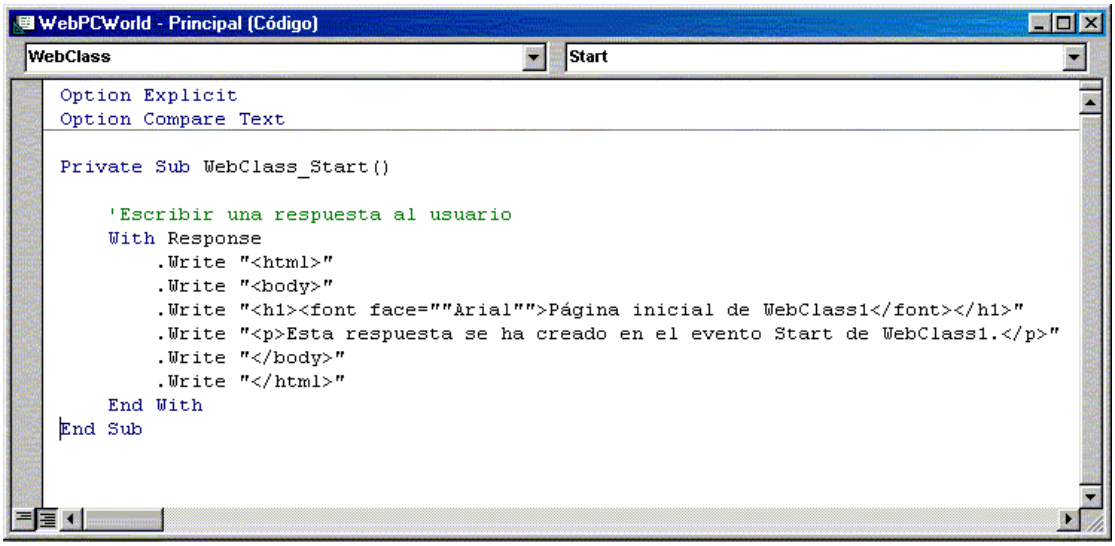

Figura 5. *Por defecto el evento `Start` de cada objeto `WebClass` cuenta con el código necesario para comprobar el funcionamiento del proyecto.*

Obviamente podemos modificar o eliminar el código de ejemplo que contiene el módulo, insertando el apropiado para conseguir nuestro objetivo. Supongamos que estamos creando una aplicación que al ejecutarse debe mostrar una página inicial de presentación, página que hemos diseñado en una herramienta externa y almacenado en un archivo HTML, como se aprecia en la Figura 6. El siguiente paso consistirá en añadir ese documento a nuestro proyecto, para lo cual habrá que seleccionar la opción **Agregar plantilla HTML** del menú emergente o bien pulsar el botón equivalente. Se añadirá un nuevo `WebItem`, al que vamos a llamar `Portada`, quedando el diseñador de clases web como puede verse en la Figura 7.

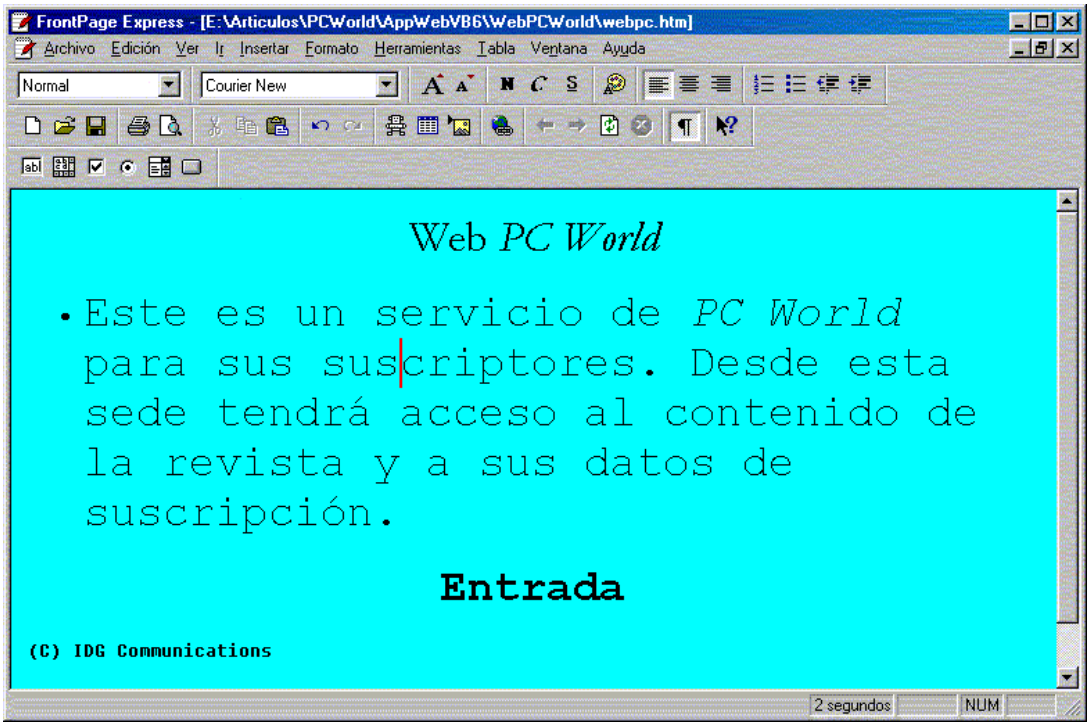

Figura 6. *Preparamos nuestro documento inicial usando cualquiera editor HTML para, posteriormente, importarlo como plantilla en nuestro proyecto.*

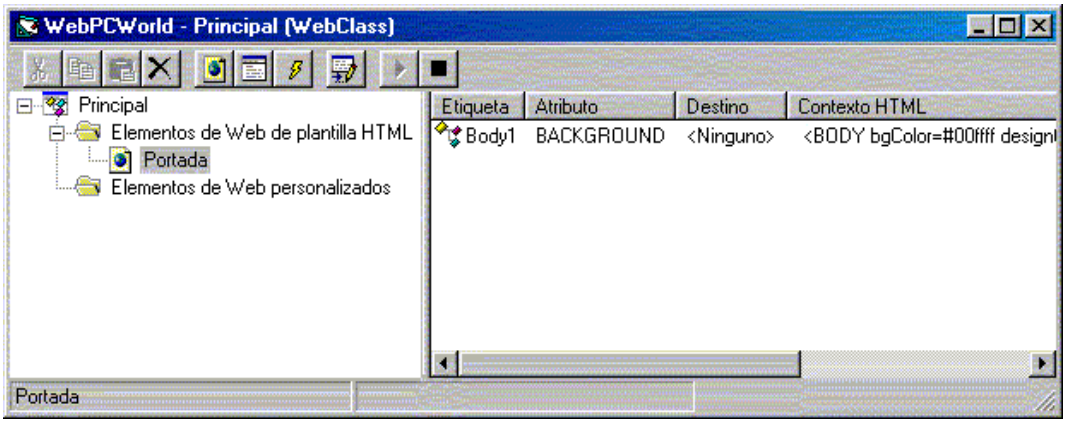

Figura 7. *Una vez añadida la plantilla al proyecto se crea un nuevo objeto* `WebItem`*. Seleccionándolo es posible ver los elementos del documento que es posible conectar a eventos.*

Lo único que resta es modificar el código asociado al evento `Start` del objeto `WebClass`. En lugar de usar el objeto `Response` para enviar el código HTML, se usará el método `WriteTemplate` de la plantilla `Portada`. Por lo tanto, tras eliminar el contenido por defecto del método `WebClass_Start` habrá que añadir la sentencia `Portada.WriteTemplate`, quedando el método tal y como se muestra en el siguiente fragmento.

```
Private Sub WebClass_Start()
    Portada.WriteTemplate
End Sub
```

Sustitución de etiquetas

Está claro que para enviar al cliente un documento estático, como el que hemos diseñado, no es preciso crear una aplicación de servidor web, bastaría con hacer accesible directamente el documento HTML, sin más. Que el documento sea procesado por nuestra aplicación, sin embargo, tiene una serie de ventajas, si bien hasta ahora no estamos aprovechando ninguna de ellas.

Comencemos viendo cómo un `WebItem` puede procesar ciertas etiquetas del documento, identificadas como marcas, para aportarle algunos elementos dinámicos. Todo objeto `WebItem` cuenta con una propiedad, llamada `TagPrefix`, cuyo valor podemos editar en la ventana de propiedades de Visual Basic. Dicha propiedad indica el prefijo que habrán de tener las marcas de la plantilla HTML que quieren sustituirse. Por defecto el valor de esa propiedad es `WC@`. Tras seleccionar el elemento `Portada` en el diseñador de clases web, nos vamos a la ventana de propiedades y la modificamos para que sea simplemente `wc`.

Acto seguido tendremos que modificar la plantilla HTML, el documento creado inicialmente, añadiendo las marcas necesarias para alojar un contador y la fecha y hora actuales. En la Figura 8 puede ver el documento original tras insertar dos nuevas líneas, que aparecen resaltadas del resto. La primera es un párrafo en el que existe una etiqueta llamada `WC:CONTADOR`, mientras que en la segunda existe otra con el nombre `WC:FECHAHORA`. El contenido de estas etiquetas no es de utilidad alguna, ya que será sustituido por nuestro código al procesar la página. Tras guardar las modificaciones será preciso actualizar la plantilla en el proyecto, para lo cual basta con abrir el menú contextual del elemento `Portada` y elegir la opción correspondiente.

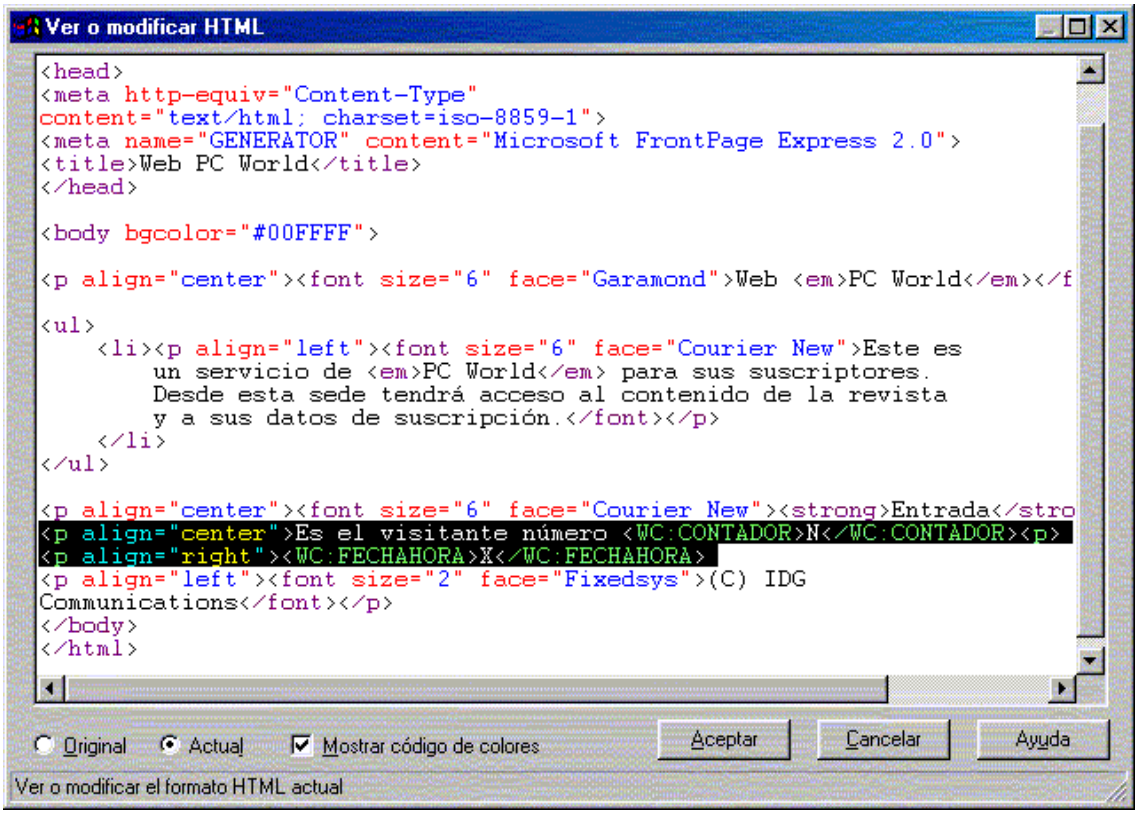

Figura 8. *El contenido de las etiquetas delimitadas por marcas que comienzan con* `WC` *será sustituidas por los valores apropiados.*

Por último, tenemos que escribir el código necesario para sustituir las dos etiquetas citadas. Cada vez que se utiliza un `WebItem` para enviar una plantilla HTML al cliente, como en este caso, se hace una exploración y se genera un evento `ProcessTag` cada vez que se encuentra una etiqueta que comienza con las iniciales asignadas a la propiedad `TagPrefix`. El método correspondiente recibe como primer parámetro una cadena con el nombre de la etiqueta encontrada, en este caso `WC:CONTADOR` o `WC:FECHAHORA`, mientras que el segundo es otra cadena con el contenido de la etiqueta. Bastará con cambiar el valor de ese segundo parámetro para sustituir el contenido de la etiqueta.

```
    ' Marcas sustituibles en la plantilla Portada
    Private Sub Portada_ProcessTag(ByVal TagName As String, _
                  TagContents As String, SendTags As Boolean)
            ' Si es la marca de fecha y hora
        If TagName = "WC:FECHAHORA" Then
            TagContents = Now ' devolvemos la fecha y hora
            Exit Sub ' salimos
        End If
      ' en otro caso es la marca CONTADOR
  Dim Contador As Integer
  On Error Resume Next ' prevenimos posible error
   ' abrimos el archivo donde está el contador
  Open "Contador.dat" For Random As 1 Len = 2
  Get #1, , Contador ' lo leemos
  Contador = Contador + 1 ' incrementamos
  Put #1, 1, Contador ' y reescribimos
  Close #1 ' cerrando el archivo
```

```
    On Error GoTo 0
    TagContents = Contador ' devolvemos el contador
End Sub
```

Listado 2. *Código para sustituir el contenido de las etiquetas existentes en la plantilla HTML*

En el Listado 2 se muestra el código que sería preciso para insertar en nuestra plantilla el contador de peticiones, la fecha y la hora. En caso de que la etiqueta encontrada sea `WC:FECHAHORA` se asigna a `TagContents` el valor devuelto por la función `Now` y se abandona el procedimiento. Si la etiqueta no es la anterior tan sólo puede ser el contador, cuyo valor está almacenado en un archivo de sólo dos bytes. Abrimos dicho archivo, recuperamos el valor, lo incrementamos y rescribimos, terminando por devolverlo como contenido de la etiqueta. En la Figura 9 puede ver el nuevo aspecto del documento tras incluir los cambios.

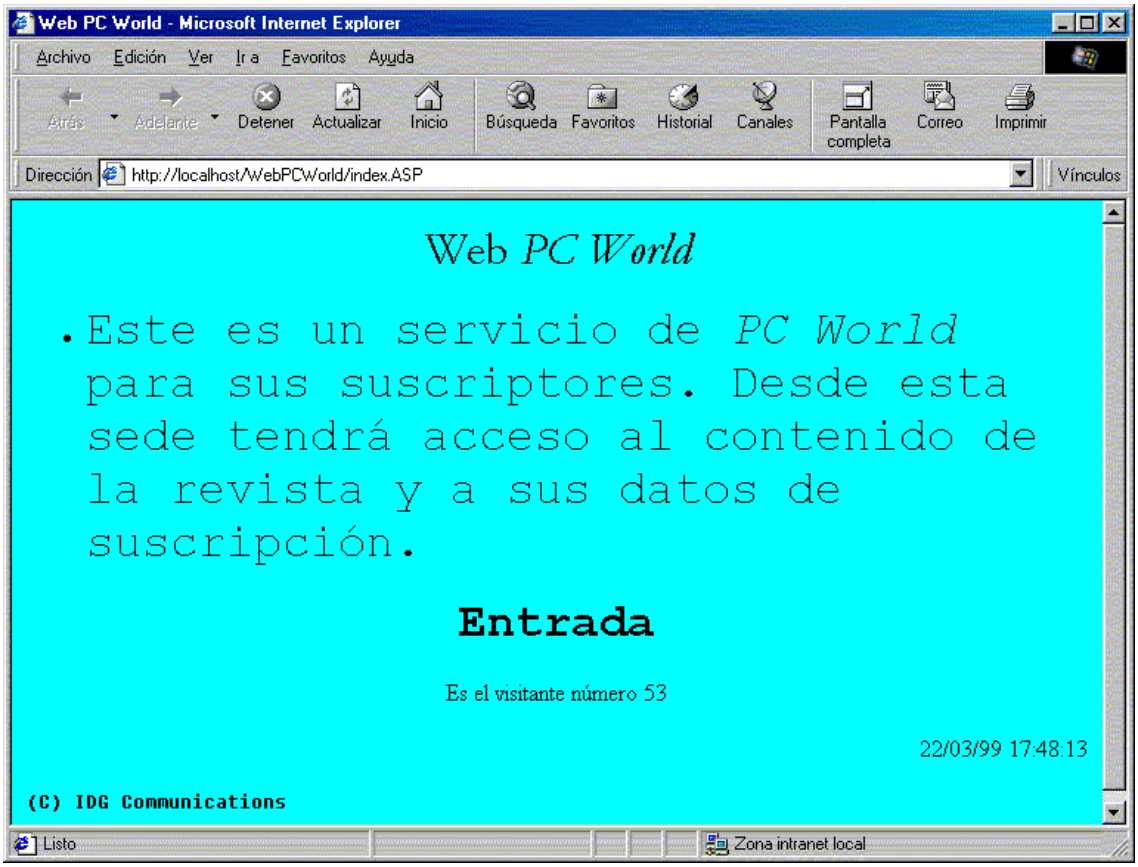

Figura 9. *Tras insertar el código en el evento `ProcessTag`, al solicitar el documento éste muestra elementos no estáticos como un contador, la fecha y hora.*

Objetos `WebItem` personalizados

Un objeto `WebItem` que tiene asociada una plantilla se procesa según acaba de describirse, no pudiendo tener mucho más control sobre él que el explicado trabajo de sustitución de etiquetas. Es posible, sin embargo, crear objetos `WebItem` personalizados, que se caracterizan por no tener asociada una plantilla HTML. Cuando el cliente envía una petición de ese elemento el objeto genera un evento `Respond`. Será en ese evento donde, usando el objeto `Response`, se genere dinámicamente el código HTML que se enviará como contestación.

Continuando con el proyecto de ejemplo anterior, suponga que dispone de una base de datos con los artículos publicados en una revista y quiere ofrecer a los clientes un documento que sea una lista del contenido de esa tabla. Lógicamente no puede preparar una página HTML estática ya que ésta se encontraría desactualizada la mayor parte del tiempo, a menos que cualquier modificación sobre la base de datos se efectuase asimismo sobre el mencionado documento, duplicando todo el trabajo. La mejor opción consiste en generar dinámicamente el código HTML.

Asumiendo que tenemos creada ya la base de datos, con una tabla que contiene datos de los artículos, añadiríamos al proyecto un entorno de datos con una conexión a dicha base y definiríamos un solo comando con el que se podría acceder a la tabla. En la Figura 10 se muestra el entorno de datos con la conexión, el comando y las columnas asociadas.

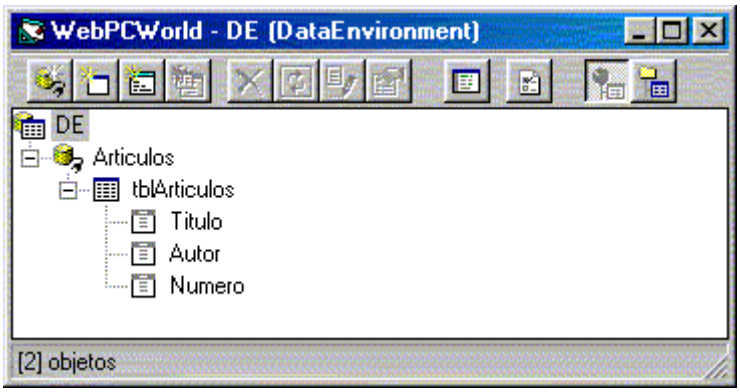

Figura 10. *Añadimos al proyecto un entorno de datos con una conexión a la base de datos en la que están almacenados los artículos.*

A continuación abrimos el diseñador de módulos web y añadimos un `WebItem` personalizado llamándolo `Articulos`. Hacemos doble clic sobre ese nuevo elemento, para abrir el correspondiente método `Respond`, y añadimos el código del Listado 3. Básicamente abrimos el comando, en este caso la tabla de revistas, y recorremos todas sus filas añadiendo una entrada por cada artículo. Observe que se incluyen, al principio y al final, todas las etiquetas necesarias para que el cliente interprete la información como un documento HTML.

```
    ' Cuando se solicite la lista de artículos
    Private Sub Articulos_Respond()
      With Response ' preparamos un documento
```

```
          .Write "<HTML><body><h1 align=center>Relación de
artículos</h1><ol>"
          DE.rstblArticulos.Open ' abriendo la tabla Articulos
      Do While Not DE.rstblArticulos.EOF
                ' y facilitando una lista de su contenido
              .Write "<li>" & DE.rstblArticulos("Titulo") & "</li>"
             DE.rstblArticulos.MoveNext
          Loop
    DE.rstblArticulos.Close ' cerramos la tabla
    .Write "</ol></body></HTML>" ' y el documento
  End With
End Sub
```

Listado 3. *Al producirse el evento* *Respond* *del elemento* *Articulos* *se genera la lista de artículos*

Ya tenemos codificado nuestro `WebItem` personalizado. Si tras realizar los cambios se ejecuta el proyecto, sin embargo, sigue apareciendo la portada y, desde ella, no hay ningún enlace que nos permita acceder a la lista de artículos. Podríamos modificar el evento `Start` del `WebClass` para que al ejecutar se facilitase la lista de artículos en lugar de la portada, pero lo lógico es que ésta aparezca antes y cuente con un enlace al segundo documento.

En la portada existe un párrafo con el texto "`Entrada`" que en principio no tiene asociado enlace alguno. Vamos a incluir ese texto como contenido de una etiqueta a la que llamaremos `WC:ENLACE`, con la finalidad de que sea sustituida en el evento `ProcessTag` por el enlace correspondiente. Será necesario, por lo tanto, modificar el código del método asociado a ese evento, que ahora quedará como puede verse en el Listado 4. Al encontrarse la mencionada marca se sustituye su contenido por una marca de enlace (`<a href>`) cuya dirección apunta al `WebItem` llamado `Articulos`. Para obtener el URL de ese `WebItem` se usa el método `URLFor`, evitando así codificar directamente el enlace.

```
    ' Marcas sustituibles en la plantilla Portada
    Private Sub Portada_ProcessTag(ByVal TagName As String, _
                  TagContents As String, SendTags As Boolean)

      ' Según la marca encontrada
  Select Case TagName
      Case "WC:FECHAHORA":
          TagContents = Now ' devolvemos la fecha y hora
      Case "WC:CONTADOR":
              ' procesamos el contador
          Dim Contador As Integer
          On Error Resume Next ' prevenimos posible error
           ' abrimos el archivo donde está el contador
          Open "Contador.dat" For Random As 1 Len = 2
          Get #1, , Contador ' lo leemos
          Contador = Contador + 1 ' incrementamos
          Put #1, 1, Contador ' y reescribimos
          Close #1 ' cerrando el archivo
          On Error GoTo 0
```

```
            TagContents = Contador ' devolvemos el contador
        Case "WC:ENLACE": ' o establecemos el enlace
                ' a la lista de artículos
            TagContents = "<a href=" & URLFor(Articulos) & _
                ">Entrada</a>"
    End Select
End Sub
```

Listado 4. *Evento* `ProcessTag` *después de incluir el proceso de la etiqueta* `WC:ENLACE`

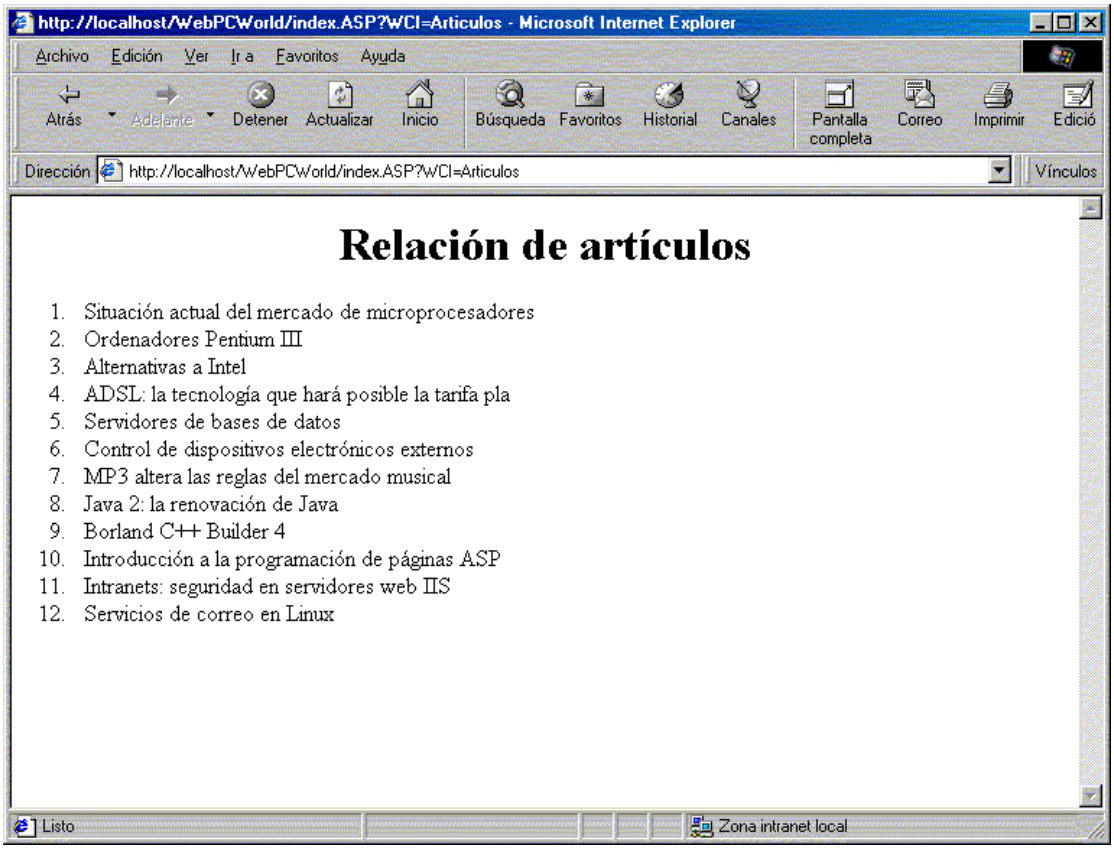

Figura 11. *Al solicitar la lista de artículos nuestra aplicación accede a la base de datos y genera dinámicamente el documento HTML.*

Usar el método `URLFor` del objeto `WebClass` es siempre mejor que insertar enlaces estáticos en los documentos. Si en cualquier momento se realiza un cambio en la estructura de la aplicación, cambiando nombres de elementos, añadiendo otros nuevos, etc., el método `URLFor` siempre facilitará el URL correcto, mientras que los introducidos manualmente pueden quedar perdidos. Ahora al ejecutar el proyecto en la portada hay un enlace que, al ser pulsado, abre la página que contiene la lista de artículos generada dinámicamente.

Formularios y *cookies*

En teoría la aplicación que estamos construyendo, accesible a través de la Web, está pensada para que la utilicen sólo los suscriptores de la revista. Hasta ahora, no obstante, no se ha realizado control alguno sobre el acceso, lo que significa que cualquiera que conociese el URL podría abrirla y leer los artículos. Lo más lógico es solicitar la introducción de una clave, en un campo de edición, y comprobarla en el servidor. Para evitar pedir la clave varias veces, cada vez que el usuario cambie de una página a otra, podemos almacenarla en una *cookie* y recuperarla cuando se precise.

Hasta ahora tan sólo hemos enviado información al cliente, mediante el objeto `Response`, porque los documentos que estamos generando no permiten al usuario introducir información alguna. En caso de que así fuese, para acceder a los distintos campos habría que usar el objeto `Request`, concretamente la propiedad `Form`, facilitando como índice el nombre del elemento a recuperar. Esta operación de lectura sólo se efectúa cuando el cliente envía la información al servidor, habitualmente usando el método POST.

Desde un módulo web de Visual Basic los datos de un formulario se recuperan conectando la acción, por lo general el mencionado POST por parte del cliente, con un `WebItem` personalizado. Esta conexión se efectúa desde el diseñador de clases web. Cuando se produce el evento `Respond` significa que los datos del formulario acaban de recibirse y se encuentran en el objeto `Request`, no tenemos mas que recuperarlos.

Utilizar las conocidas *cookies* desde una aplicación web en Visual Basic es igualmente sencillo. Los objetos `Response` y `Request` cuentan con una propiedad, llamada `Cookies`, que es una colección de las *cookies* existentes. La colección del objeto `Response` se usa para crear una nueva *cookie*, mientras que la del objeto `Request` sirve para recuperar *cookies* previamente almacenadas.

Por defecto las *cookies* sólo tienen validez durante una sesión. Es decir, podremos recuperar el valor de la *cookie* mientras el cliente se mueva de un punto a otro de nuestra sede sin llegar a cerrarse la conexión. En el momento en que se cierre y vuelva a abrir, la *cookie* habrá desaparecido. Es posible especificar una fecha de caducidad usando la propiedad `Expires` de la *cookie* que nos interese. De esta forma sería posible, por ejemplo, conservar información como la última fecha en que nos visitó el cliente, dato que usaríamos para mostrar una lista personalizada de los artículos nuevos. Esta lista sería diferente para cada cliente que conectase, porque la fecha de última visita se almacenaría en el ordenador del propio cliente.

Modificaciones al proyecto

Tomemos una vez más la plantilla HTML que actúa como portada e incluyamos una nueva modificación. En este caso se añadirá un campo de texto con capacidad para tres caracteres que mostrará en forma de clave. Este campo se

llamará `Clave` y será el único elemento del formulario, por lo que el envío de la información al servidor se producirá en el momento en que se pulse Intro.

Tras actualizar la plantilla, seleccionándola en el diseñador de clases web y ejecutando la opción correspondiente, veremos aparecer en el panel derecho un nuevo elemento con el nombre `Form1`. Para recoger los datos de ese formulario es preciso que creemos otro `WebItem` personalizado, le llamaremos `RegistroUsuario`, y lo conectemos. Esta última operación, representada en la Figura 12, se consigue seleccionando el formulario en el diseñador, abriendo el menú emergente y eligiendo la opción de conexión a `WebItem`. Aparece una ventana en la que podemos elegir el elemento a conectar.

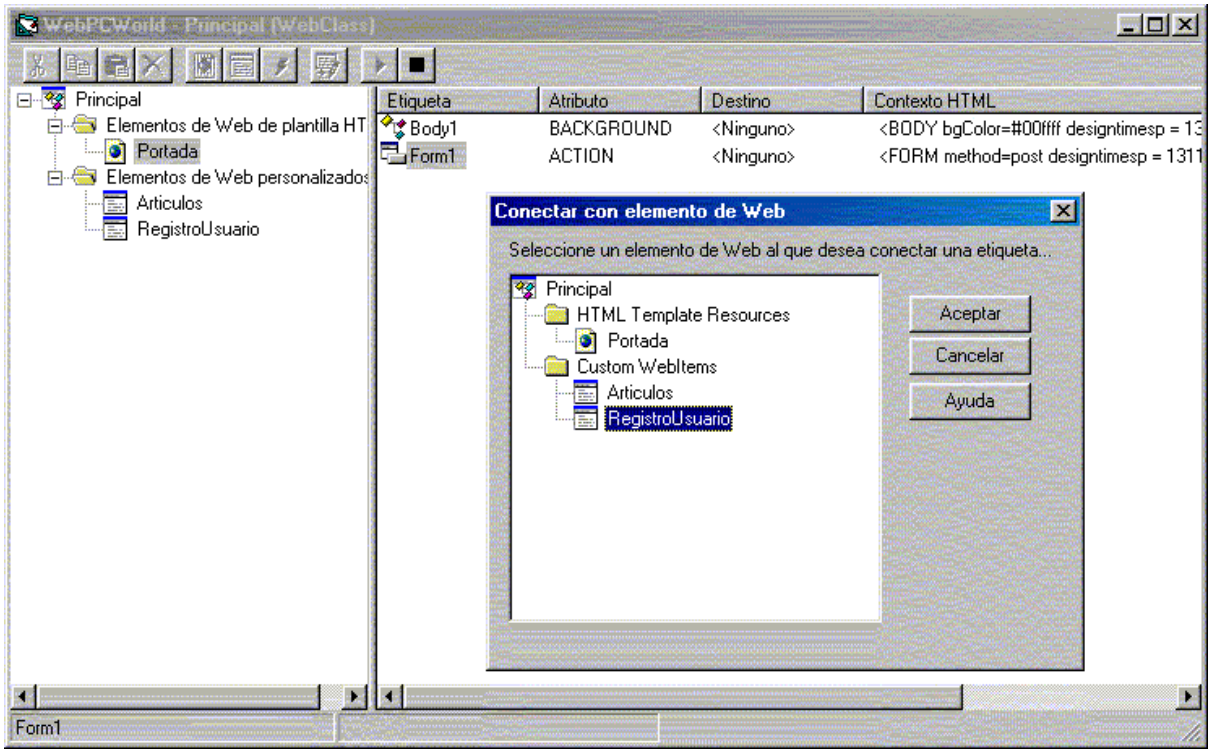

Figura 12. *Conectamos el envío del formulario al servidor con un* `WebItem` *personalizado que se encargará de recoger y procesar los datos.*

Establecida la conexión habrá que escribir el código encargado de procesar el formulario. En este caso dicho código es breve y simple, como puede ver en el siguiente fragmento, limitándose a tomar el valor del campo `Clave` del formulario almacenándolo como una *cookie* a la que también llamamos `Clave`.

```
Private Sub RegistroUsuario_Respond()
    Response.Cookies("Clave") = _
        Request.Form("Clave")

    Set NextItem = Articulos
End Sub
```

Observe que tras guardar la clave en la *cookie* se asigna a la propiedad `NextItem` una referencia al `WebItem Articulos`, provocando su ejecución.

Dicho elemento no es una plantilla, por lo que para enviarlo al cliente no es posible usar el método `WriteTemplate` explicado anteriormente.

Lógicamente solicitar la clave y almacenarla no servirá de nada si posteriormente no se utiliza para controlar el acceso a los documentos, en este caso a la lista de artículos. Al inicio de ese `WebItem`, o de cualquier otro que tenga que protegerse, se realizará una llamada a la función `VerificaUsuario` mostrada en el Listado 5. El documento se enviará al cliente sólo si esta función devuelve el valor `True`. En este caso la clave se ha incluido directamente en el código del programa pero, como es obvio, en una aplicación real las claves de los suscriptores serían distintas y se recuperarían de una base de datos.

```
    ' Esta función verifica la clave del usuario
    Private Function VerificaUsuario() As Boolean
            ' si la clave no es correcta
        If Request.Cookies("Clave") <> "pcw" Then
              ' indicamos que no puede acceder
            Response.Write "No es suscriptor"
            VerificaUsuario = False ' y devolvemos false
        Else ' en caso contrario devolvemos true
            VerificaUsuario = True
        End If
    End Function
```

Listado 5. *Esta función se encarga de verificar la validez de la clave permitiendo o denegando el acceso*

Resumiendo

Como ha podido ver en este artículo, el desarrollo de aplicaciones de servidor con los nuevos módulos web de Visual Basic 6 resulta una tarea muy sencilla. A pesar de la funcionalidad demostrada en los distintos ejemplos, lo cierto es que tan sólo se han visto algunas de las posibilidades de los módulos web, quizá las más interesantes.

La construcción de aplicaciones web con Visual Basic 6 es una seria alternativa al uso de otros métodos, como Win-CGI o la construcción de páginas ASP. No hay que olvidar que todo el código de la aplicación está compilado en una librería ActiveX y, por lo tanto, se ejecutará más rápidamente que una página ASP en la que el código tiene que ser interpretado por un motor de *script*. No obstante, conocer el modelo de objetos usado en ASP, que es el de *Internet Information Server*, es útil también para utilizar los módulos web de Visual Basic.