

EXIT

# ASP.NET 4.5 en C#

## con Visual Studio 2012

Diseño y desarrollo  
de aplicaciones Web

Brice-Arnaud GUÉRIN



INFORMATICA TECNICA



# Índice

- *Información*
  - *Título, autor...*
- *Prólogo*
  - *Prólogo*
- *Visual Studio 2012 y .NET 4.5*
  - *Novedades de Visual Studio 2012*
  - *C#4 de un vistazo*
- *Los sitios web ASP.NET 4.5*
  - *El modelo de compilación*
  - *El rol del servidor web*
  - *El pipeline HTTP de IIS*
- *Los Web Forms*
  - *Los Web Forms*
  - *Organizar la presentación*
  - *Componentes personalizados*
  - *AJAX*
- *Los sitios web MVC*
  - *El enfoque MVC*
  - *Los sitios ASP.NET MVC*
  - *Definición de las rutas*
  - *Ir más allá*
  - *Sitios MVC 4 y Razor para dispositivos móviles*
- *El acceso a datos con ADO.NET 4.5*
  - *Bases de ADO.NET*
  - *Acceso a los datos mediante proveedores*
  - *Componentes gráficos de presentación de datos*
  - *Crear informes con Reporting Services*
- *Gestión del estado*
  - *Los distintos medios para mantener el estado*
  - *Las sesiones*
  - *Los objetos Application y Cache*
- *Personalización y securización*
  - *Securización de los sitios ASP.NET*
  - *Presentación personalizada*
  - *Los Web Parts*
- *Los servicios Web WCF y REST*
  - *Los servicios Web WCF*
  - *Los servicios Web REST*
- *Configuración, despliegue y administración*
  - *Configuración*
  - *Despliegue de aplicaciones ASP.NET*
  - *Supervisión de aplicaciones ASP.NET*

# ASP.NET 4.5 en C# con Visual Studio 2012

## Diseño y desarrollo de aplicaciones Web

Este libro está dirigido a los **desarrolladores, arquitectos y administradores** que deseen adoptar un enfoque profesional en la realización de aplicaciones Web sacando el máximo provecho de ASP.NET. Acompaña al lector en un estudio completo de la tecnología **ASP.NET y de Visual Studio 2012**. Cada tema se aborda con ejemplos prácticos y útiles, que se proveen en **C#**.

El lector comenzará con **Visual Studio y sus herramientas** (depuración, refactoring, pruebas unitarias, pruebas de interfaz gráfica,...) y la evolución del **lenguaje C#**. El segundo capítulo describe el funcionamiento de las **aplicaciones IIS y explica cómo realizar módulos específicos para el servidor Web**. La obra estudia con profundidad los **Web forms, AJAX, JQuery, los sitios MVC** y proporciona componentes personalizados para crear gráficos.

Los siguientes capítulos elaboran soluciones que aportan **rapidez al desarrollo y mejor rendimiento** en el acceso a las bases de datos **ADO.NET**, especialmente con los nuevos componentes basados en **LINQ y entity framework y los estados Reporting Services**. A continuación, se describe la securización de los sitios Web y la personalización de la navegación (**Web Part y servicios Web WCF, REST**). El último capítulo describe la puesta en producción con **ASP.NET** y la infraestructura de supervisión **Health Monitoring**.

---

**Los capítulos del libro:**  
Prólogo – Visual Studio 2012 y .NET 4.5 – Los sitios web ASP.NET 4.5 – Los Web Forms – Los sitios web MVC – El acceso a datos con ADO.NET 4.5 – Gestión del estado – Personalización y securización – Los servicios Web WCF y REST – Configuración, despliegue y administración

### Brice-Arnaud GUÉRIN

Ingeniero por la ESIEA, **Brice-Arnaud GUÉRIN** es responsable de los desarrollos de software en LexisNexis, principal referencia en la información jurídica, económica y financiera, y en soluciones de gestión para profesionales. Sus competencias en desarrollo y el deseo de compartir su conocimiento le llevan, de forma natural, a escribir obras dedicadas a la realización de aplicaciones (.NET, PHP, C++) y la gestión de proyectos.

## Prólogo

ASP.NET y Visual Studio han encontrado un hueco natural en el universo Web, la plataforma se ha enriquecido con nuevos componentes y el entorno de desarrollo integrado proporciona cada vez más herramientas, todas ellas muy útiles. Con el paso de los años, este tandem ha visto crecer su público, yendo más allá de las simples problemáticas de la programación.

Este libro está dirigido a los **desarrolladores, arquitectos y administradores** que deseen adoptar un enfoque profesional en la realización de aplicaciones Web sacando el máximo provecho de ASP.NET 4.5 y de Visual Studio 2012.

El lector comenzará descubriendo **Visual Studio y sus herramientas**. Se presentan, por turno, la interfaz del entorno de desarrollo, el cuadro de herramientas, las ventanas de exploración y el uso del depurador. El lector descubrirá, también, la gestión del código y, en particular, la función de refactorización (refactoring). Este primer capítulo detalla el uso de las pruebas unitarias así como las pruebas de la interfaz gráfica. Gracias al impulso de LINQ, el lenguaje C# ha sufrido modificaciones que se describen con ayuda de ejemplos.

El segundo capítulo describe el funcionamiento de las **aplicaciones ASP.NET funcionando con IIS**. Este servidor Web ha evolucionado considerablemente, puesto que se ha convertido en la columna vertebral del sistema de información de Microsoft. Su conocimiento ayuda, en particular, a los arquitectos a la hora de superar el funcionamiento estándar de un servidor de aplicaciones ASP.NET.

La obra estudia con profundidad los **Web forms**. El lector descubrirá los fundamentos de la programación orientada a eventos para la Web: ciclo de vida de una página ASPX, controles Web, validación de los datos introducidos por los usuarios, componentes de usuario y personalizados. También se describen las buenas prácticas de estructuración de la aplicación, tales como las páginas maestras, los temas y los skins. El framework **AJAX** ha alcanzado cierta madurez en su funcionamiento y su integración; lo estudiaremos desde distintos ángulos, tales como los componentes ScriptManager y UpdatePanel, los callbacks, el código JavaScript orientado a objetos o los componentes personalizados AJAX realizados en combinación con JavaScript y C#. Continuando con la explicación de AJAX, este capítulo estudia **jQuery**, que tiende a imponerse como el framework estándar para crear interfaces ricas en JavaScript.

Microsoft ha completado su servidor con un modelo de aplicaciones ligeras y rápidas, los **sitios MVC**. El capítulo Los sitios web MVC ilustra, a partir de ejemplos concretos, el funcionamiento, la organización y el desarrollo de tales aplicaciones y sirve como toma de contacto con el nuevo motor de renderización **Razor**, en Android.

El siguiente capítulo elabora soluciones que aportan **rapidez al desarrollo** y un **mejor rendimiento** en el acceso a las bases de datos **ADO.NET**. Se presentan con detalle los modos conectado y transaccional y, a continuación, los modos desconectado y proveedor. A estos métodos les siguen los componentes de datos para páginas ASPX, el universo LINQ, su evolución **Entity Framework** y, por último, las herramientas de generación de estado Reporting Services. En el capítulo Gestión del estado se estudian las herramientas que existen a disposición del desarrollador para mantener el estado y optimizar la aplicación con ayuda de cachés de datos.

El servidor ASP.NET dispone de mecanismos de seguridad completos; el capítulo Personalización y securización explora los modos de autenticación y de autorización. A continuación, se estudian los elementos que integran la arquitectura orientada a servicios (SOA); los portales de componentes personalizables (**web.parts**) y los servicios Web se estudian mediante un sitio de ejemplo que proporciona información de cotización bursátil. El conocimiento de las técnicas subyacentes, como las API **WCF** y **REST**, ayudarán a desarrolladores y arquitectos a crear aplicaciones personalizadas ricas y con un buen rendimiento.

El último capítulo describe los elementos indispensables en la puesta en producción de aplicaciones ASP.NET: la estructura de los archivos de configuración, las herramientas de despliegue, la creación de MSI y de carpetas virtuales en IIS. El administrador de aplicaciones se complementa con frecuencia con tareas de supervisión, por este motivo se incluye en este capítulo la infraestructura **Health Monitoring**.

## Novedades de Visual Studio 2012

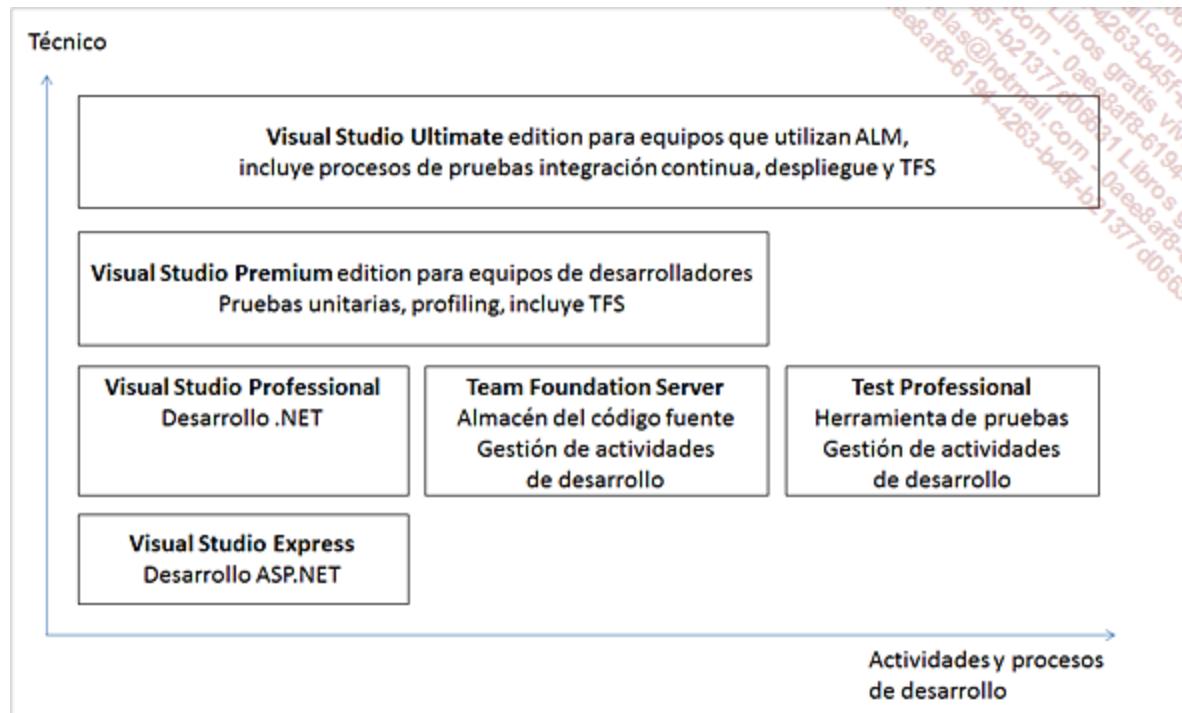
No sin cierta curiosidad muchos desarrolladores han descubierto la nueva versión de Visual Studio. ¿Qué cambios serían necesarios para crear aplicaciones respetando la filosofía de la plataforma ASP.NET? Tras llegar al mercado justo a tiempo, esta versión revela la nueva estrategia de Microsoft en términos de desarrollo de aplicaciones.

De hecho, la aparición de ASP.NET 4.5 coincide con la publicación de una nueva gama de herramientas de desarrollo y de un nuevo framework. Microsoft se dirige a un mercado lo más amplio posible, lo cual reconforta a los programadores en sus elecciones tecnológicas y les asegura al mismo tiempo una paleta de herramientas muy variada.

El framework .NET y la plataforma ASP.NET oficializan los pasos que dieron todos de forma experimental para aprovechar lo mejor de la versión inicial de la tecnología. Esta obra tiene como objetivo proporcionar una síntesis, en forma de escenarios prácticos, a los usuarios de la plataforma, sean nuevos o que ya tengan experiencia con ASP.NET.

La versión 2012 amplía el proceso de desarrollo de una aplicación .NET sobreponiendo el marco de la productividad individual. Si bien muchas aplicaciones ASP.NET se desarrollan según un proceso XP(*eXtreme Programming*), las versiones anteriores de Visual Studio no estaban del todo adaptadas al trabajo en equipo o a la ejecución de un proceso de desarrollo organizado y documentado.

Microsoft ha introducido el modelo ALM (*Application Lifecycle Management*) que rige la creación de aplicaciones, integrando las tecnologías necesarias para realizar el diseño, el desarrollo, las pruebas y el despliegue. La oferta de Visual Studio 2012 es una plataforma que agrupa herramientas destinadas a todos los protagonistas de un proyecto de software, sean o no informáticos.



Para competir con las ofertas existentes con menores costes, Microsoft ha lanzado Visual Studio Express y Visual Studio Web Developer Express. Ambos productos son muy económicos, puesto que soportan el modo wysiwig y aseguran la parte esencial del proceso de desarrollo de aplicaciones .NET y ASP.NET.

La gama Visual Studio 2012 no incluye ningún cambio importante respecto a la versión anterior. Las ediciones estándar (en un idioma) y profesional pueden asociarse a Team Foundation Server 2010 o 2012 para controlar las distintas versiones del código fuente de una aplicación.

La edición Premium de Visual Studio incluye Team Foundation Server y añade la posibilidad de crear planes de pruebas unitarias automáticas. También se permite realizar una revisión de la calidad del código (profiling). Por último, la versión alta de la gama, Ultimate, explota un servidor de integración continua (compilación diaria) y favorece el trabajo en equipo según las reglas de un proceso de desarrollo.

La gama Visual Studio se ve, por tanto, enriquecida considerablemente; esta obra desarrolla los ejemplos creados con Visual Studio edición Profesional, aunque casi todos ellos pueden reproducirse con la edición Visual Web Developer Express.

Desde un punto de vista específico, la tecnología ASP.NET consolida los mecanismos propuestos por la versión anterior, integrando un soporte de nuevas tecnologías esenciales, como AJAX, el desarrollo MVC o los servicios web WCF.

## 1. Instalación

La instalación de Visual Studio 2012 sólo puede realizarse sobre un sistema adaptado. Para los sistemas XP, es preciso instalar previamente el Service Pack 2.

El framework .NET 4.5 también debe instalarse antes de instalar Visual Studio 2012. Si su equipo no tiene el framework instalado, la distribución de Visual Studio incluye el framework 4.5. Para los equipos que ya estén equipados con una versión anterior del framework, la instalación se ejecuta de principio a fin sin problema alguno. Los ensamblados (ejecutables .NET) pueden apuntar a una versión específica del framework y del motor de ejecución CLR.

Para aquél que desarrolle con Visual Web Developer Express, puede resultar útil instalar también la versión Studio Express para desarrollar bibliotecas de componentes DLL o para diseñar aplicaciones no ASP.NET.

El asistente de instalación de Visual Studio proporciona también SQL Server Express, reemplazando a MSDE (*Microsoft Desktop Engine*). Si ya tiene alguna versión anterior de SQL Server (2000 o MSDE), o si piensa migrar a la versión 2008 completa, puede no utilizar este entorno. Por supuesto, ASP.NET sabe cómo explotar otros formatos de base de datos (Oracle, DB/2, MySQL...), aunque es cierto que la nueva versión saca mucho partido de una integración hecha con un formato de base de datos de Microsoft.

La documentación MSDN (*Microsoft Developer Network*) se instala al finalizar la instalación de Visual Studio. Es todavía más voluminosa que lo era antes, y muchos artículos pueden consultarse vía Internet. Para utilizarla sin conexión, es conveniente utilizar el asistente del modo personalizado para instalar el conjunto de documentos sobre el disco duro.

## 2. Interfaz del programa

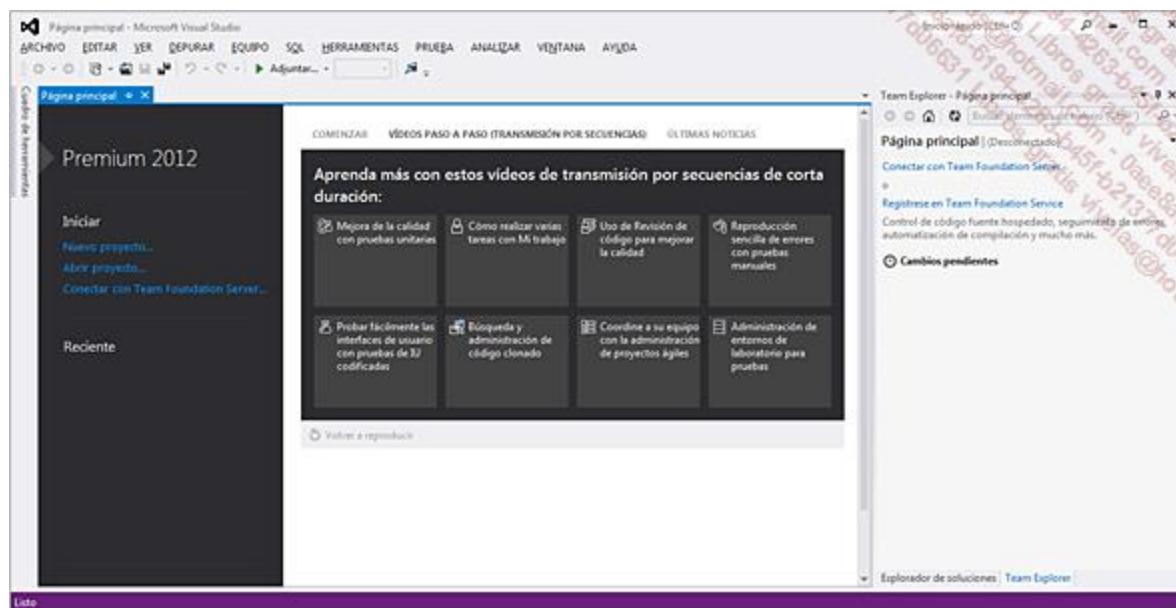
Visual Studio forma parte de la familia de entornos de desarrollo integrados (IDE). Soporta numerosas actividades ligadas al desarrollo de aplicaciones, tales como la creación de interfaces gráficas, la inclusión asistida de código fuente o incluso pruebas unitarias.

Las versiones anteriores de Visual Studio constituían una síntesis de los entornos de desarrollo Visual C++, Visual Basic e Interdev. La versión 2012 persigue cierta lógica de integración agregando la dimensión de la apertura; la creación de extensiones para enriquecer los lenguajes y entornos soportados es, desde ahora, una tarea muy sencilla (snippets, macros, plug-ins). Visual Studio 2012 es una plataforma abierta, de forma similar a Eclipse.

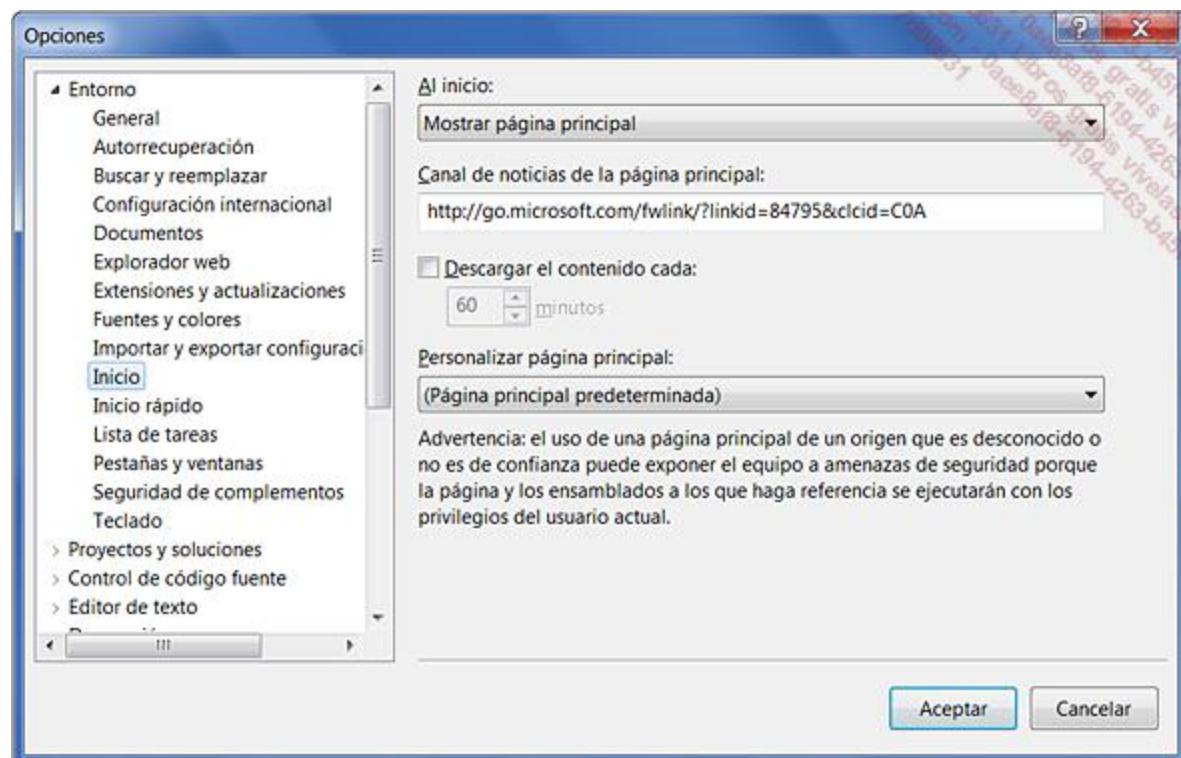
### a. La página de inicio

Tras la ejecución del programa, se muestra la página de inicio. Está compuesta por dos zonas. En la primera, el usuario encuentra la lista de los últimos proyectos abiertos. Crea y abre también nuevos proyectos, como lo haría desde el menú **Archivo** o desde la barra de herramientas.

La segunda zona presenta tres pestañas; la primera facilita la toma de contacto con Visual Studio, la segunda proporciona enlaces a la documentación en línea y recursos. La última pestaña, llamada **Últimas noticias** ofrece información proveniente de una fuente RSS (sindicación de datos vía XML):



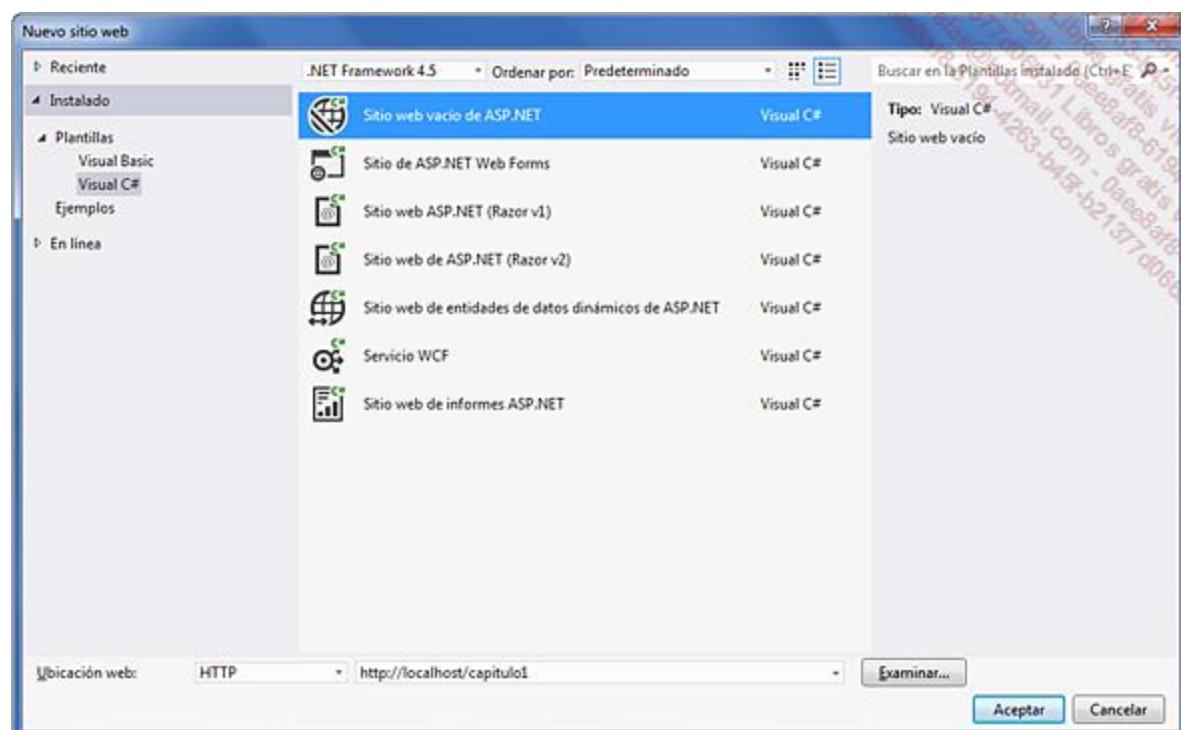
La ventana de diálogo **Opciones** de Visual Studio, que se abre desde el menú **Herramientas - Opciones**, es útil para desactivar que se muestre automáticamente esta página de inicio o para personalizar su presentación; el usuario puede cambiar la dirección de la fuente RSS.



## b. Las ventanas de Visual Studio

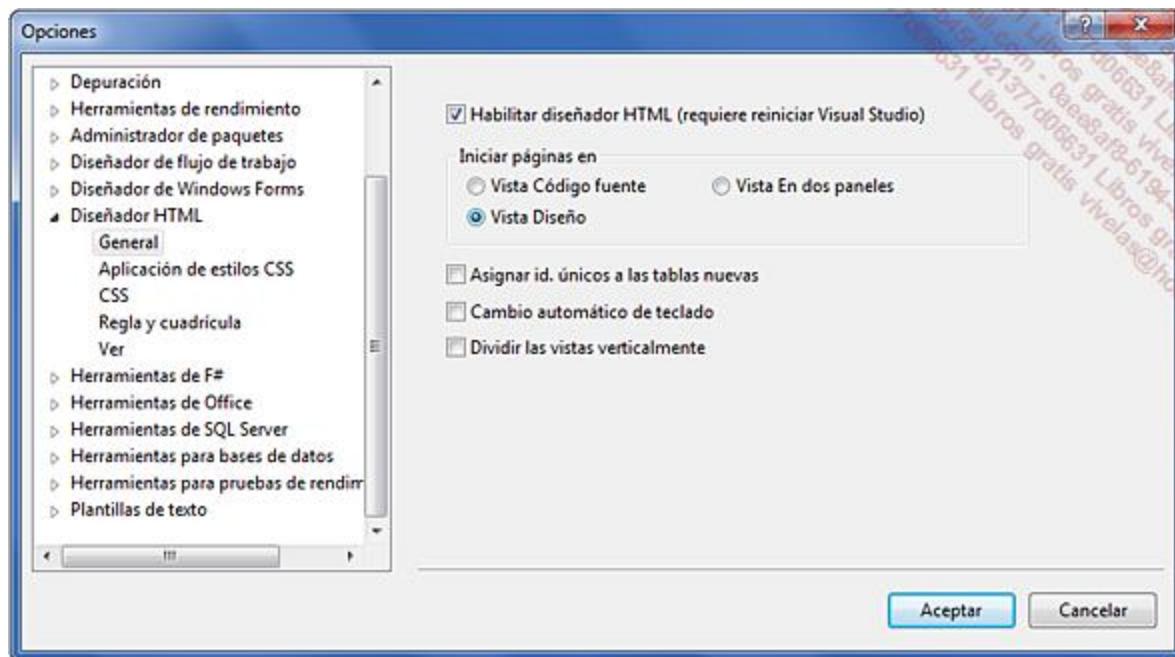
Visual Studio distingue tres tipos de ventana: ventanas de construcción/edición, ventanas de herramientas/navegación y ventanas de estado.

Para conocer con detalle la interfaz de usuario de Visual Studio, cree un primer sitio web llamado **capítulo1**. Utilice para ello el menú **Archivo - Nuevo sitio Web** o bien hágalo desde la página de inicio.

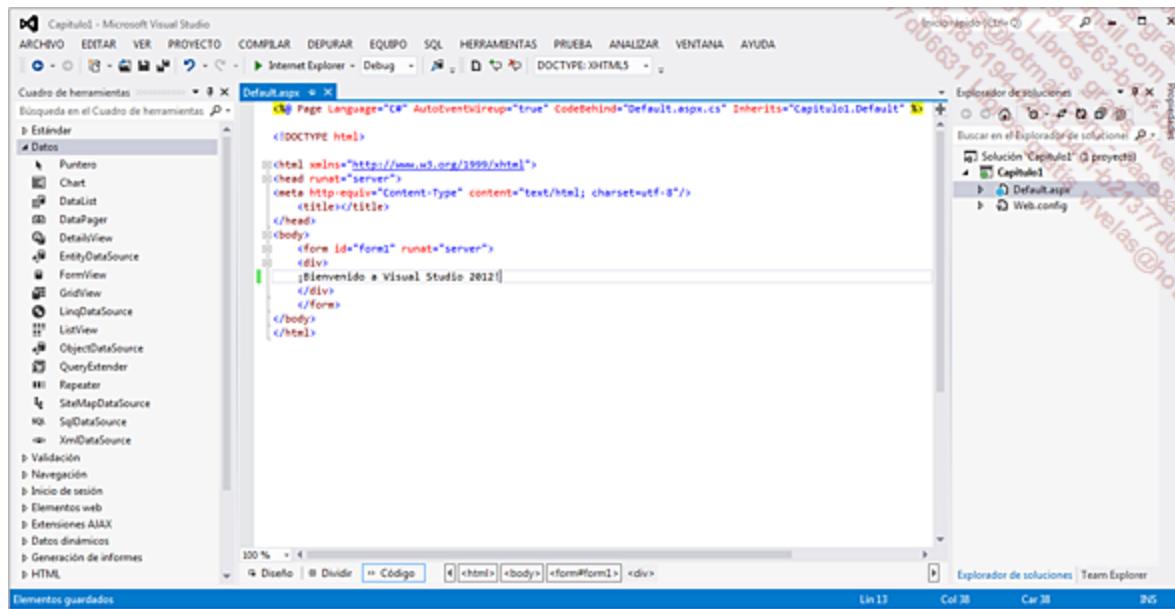


Las ventanas de construcción sirven para crear interfaces de usuario web o Windows. Se trata de archivos con extensión .aspx o .cs que se editan en modo **Diseño** (wysiwyg), en modo **Código**(HTML o C#), o en modo **dividido**. En la versión 2012 de Visual Studio se muestra una página web .aspx en modo Código (HTML).

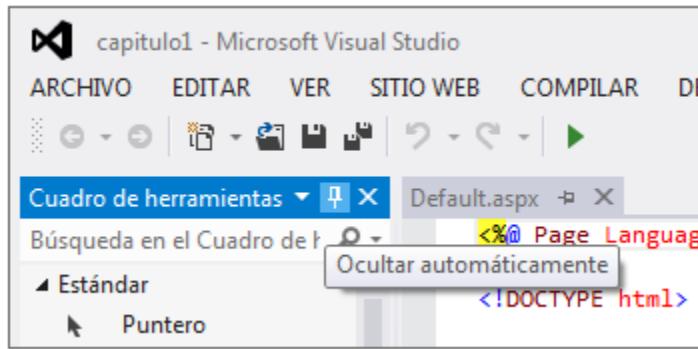
Para que se muestre el modo Diseño al inicio, hay que modificar las opciones del programa Visual Studio de la siguiente manera:



Las ventanas de Visual Studio no se muestran todas a la vez. La primera vez que se ejecuta el programa, algunas están ocultas. Es posible mostrarlas con ayuda del menú **Ver**.



El programador debe organizar de la mejor forma posible su espacio de trabajo, las ventanas se anclan a los distintos bordes de la pantalla. Disponen, también, de un sistema que les permite ocultarse automáticamente, que es posible anular mediante una chincheta.



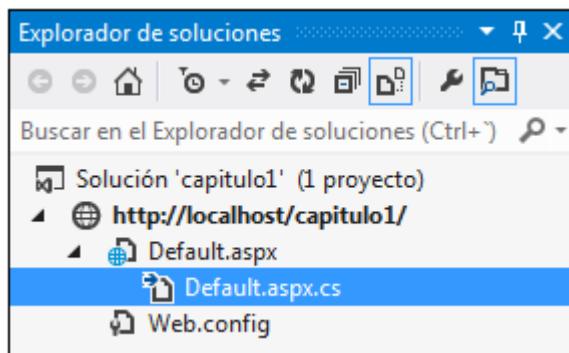
A continuación, presentamos las ventanas más útiles.

### El Explorador de soluciones

Una solución es un grupo de proyectos. Un proyecto es un conjunto de archivos (páginas HTML, código fuente, imágenes...) necesarios para la elaboración y el funcionamiento de una aplicación o de un sitio web.

Como todo proyecto se incluye en una solución, la creación de un proyecto implica, a menudo, la creación de una solución con el mismo nombre. Visual Studio utiliza archivos específicos para gestionar las soluciones (con extensión .sln y .suo para las opciones del usuario) y los proyectos (.csproj).

Los archivos de solución y de proyecto se organizan de forma más sencilla en la carpeta Mis Documentos\Visual Studio 2012\Proyectos. Cada solución crea una carpeta principal y, a continuación, los distintos proyectos asociados a la solución se ubican en subcarpetas. La clasificación parece ser más sistemática que en las versiones anteriores y, sobre todo, no existe diferencia entre VB.NET y C#. Otra característica, que se estudia en el capítulo Los sitios web ASP.NET 4.5, es el modelo de compilación de un sitio web ASP.NET, ya no precisa la creación de un proyecto .csproj o .vbproj. Por último, el usuario **MS Build** que sustituye a **nmake** (make) le otorga al proyecto Visual Studio una dimensión que supera el simple listado de la carpeta.



El explorador de soluciones presenta los archivos relativos a los proyectos asociados a la solución en curso. Lo más habitual es que el usuario abra sus archivos haciendo doble clic sobre ellos.

### La barra de propiedades

Aportada por Microsoft con Visual Basic, la barra de propiedades ha evolucionado muy poco, lo que prueba que es un elemento apreciado por los desarrolladores. La barra muestra y edita las propiedades de un objeto seleccionado, se trate de un control gráfico o de un archivo del explorador de soluciones.

Para el desarrollador ASP.NET 4.5, la barra de propiedades puede, en lo sucesivo, editar las propiedades de un control seleccionado en modo **Código**:



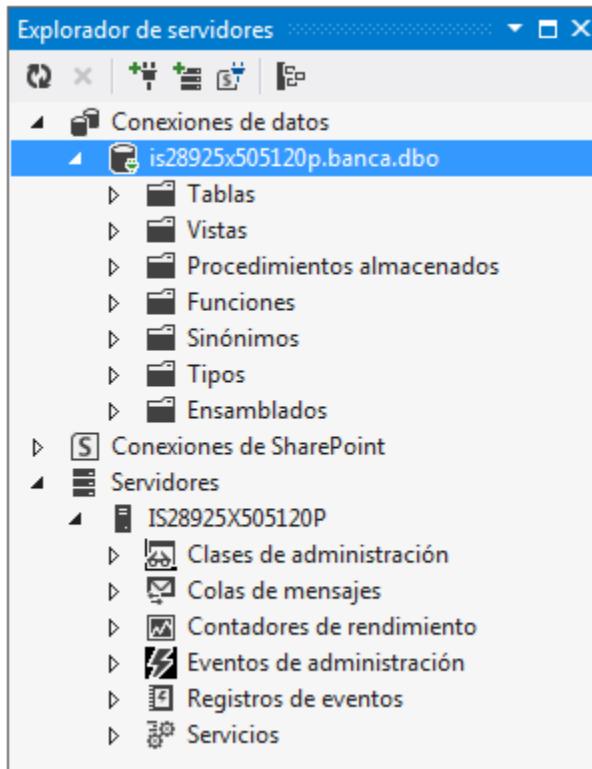
## El cuadro de herramientas

El cuadro de herramientas presenta los distintos controles, gráficos o no, disponibles para el tipo de aplicación que se está construyendo. El desarrollador encuentra aquellos controles que ya estaban disponibles en la versión anterior de Visual Studio: controles básicos web, HTML y componentes de servidor.

Se han agregado algunos controles suplementarios, los cuales se presentan en el capítulo Los sitios web ASP.NET 4.5. Otros controles se han agrupado en una sección separada, tales como controles de validación de datos introducidos por los usuarios.

## El Explorador de servidores

Esta ventana no está disponible con las versiones Express de Visual Studio; se ha reemplazado por una ventana con posibilidades más limitadas, el Explorador de datos.

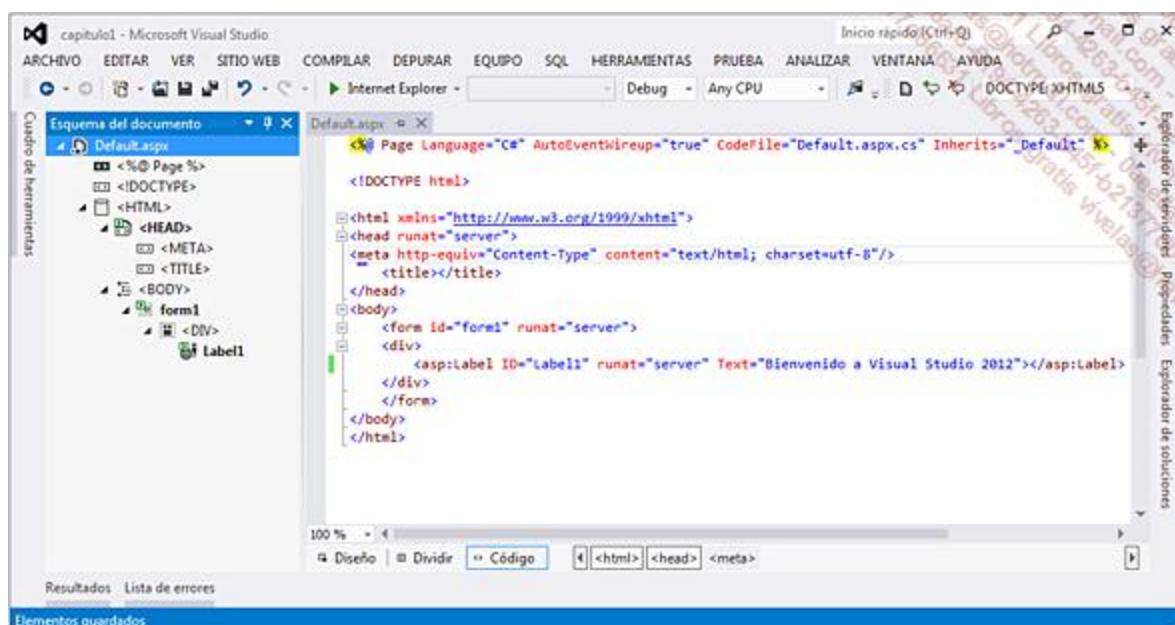


A menudo, el Explorador de servidores se utiliza para acceder a las bases de datos que se requieren para el funcionamiento de una aplicación. Para los usuarios de SQL Server Express, este explorador sirve también para crear bases de datos, su estructura (tablas, vistas y procedimientos almacenados) y para alimentar los datos. De hecho, la versión 2008 de SQL Server tiene una interfaz que recuerda a la de Visual Studio.

El Explorador de servidores también provee información acerca del equipo: estados de Crystal Report, registros (logs), WMI, colas de espera MSMQ, servicios de Windows, indicadores de rendimiento... Toda esta información es accesible a través del panel de configuración.

## El esquema del documento

Para simplificar la edición de documentos basados en HTML, la ventana **Esquema del documento** presenta una vista en árbol del documento que está siendo editado. Esta ventana, ya presente en Visual Studio 2008, se ha completado con una presentación más directa de los tags, recordando así al funcionamiento de Dreamweaver.



### c. Las actividades ligadas al desarrollo

Crear un sitio web o una aplicación consiste en encadenar cierto número de actividades definidas por el proceso de desarrollo. Tratándose de un IDE, Visual Studio 2012 trabaja como una plataforma que incluye las herramientas necesarias para la ejecución de dichas actividades.

#### Codificación

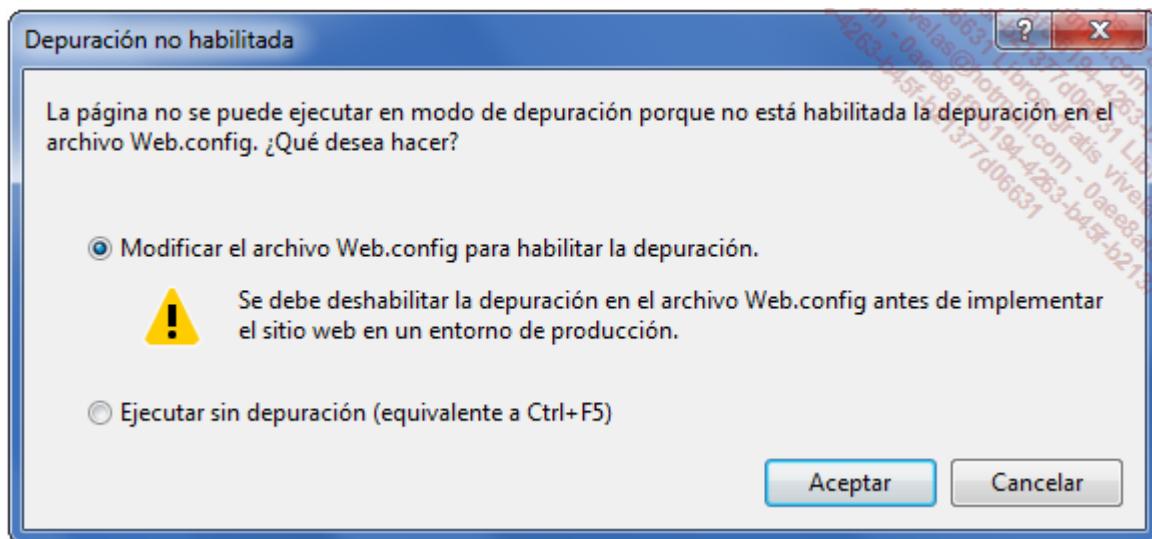
Se trata de la actividad principal del desarrollador en Visual Studio.

#### Depuración

Microsoft ha tenido la fabulosa idea de integrar la depuración en el CLR; todos los ensamblados .NET pueden depurarse. Visual Studio ayuda al desarrollador a situar sus puntos de ruptura, incluso para aplicaciones ASP.NET.

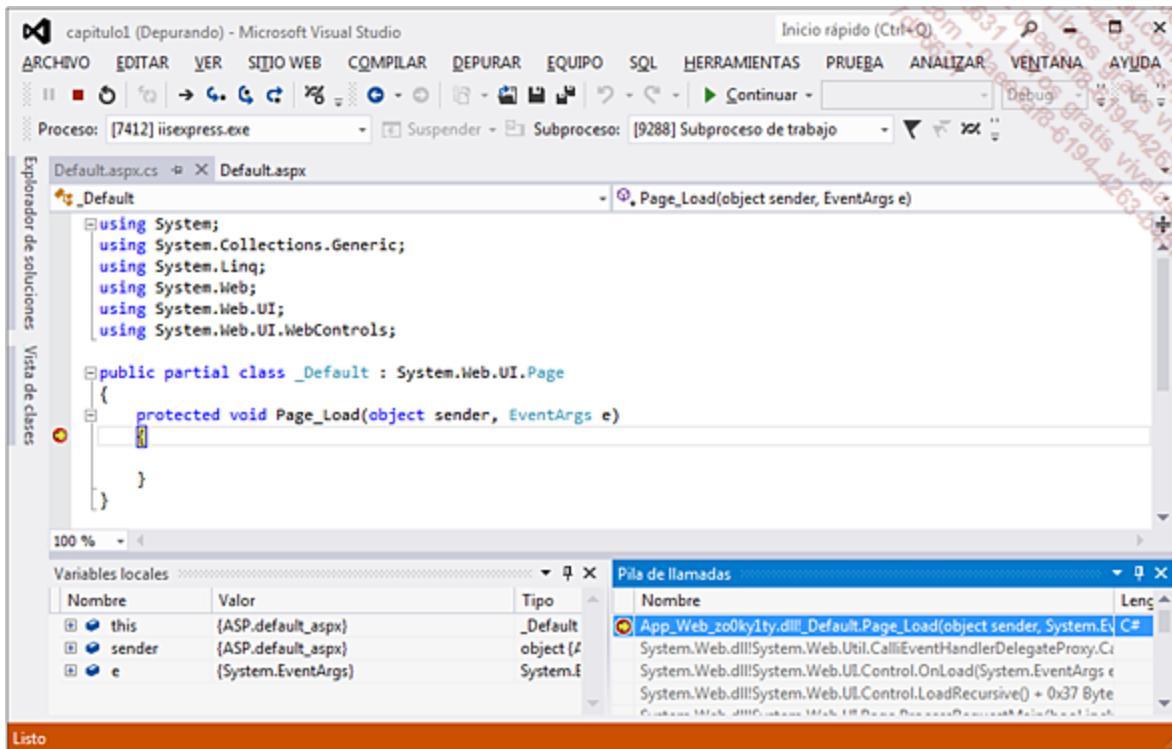
Como el nuevo modelo de compilación no prevé un uso sistemático del archivo Web.config, la primera sesión de depuración provoca que aparezca un mensaje solicitando al desarrollador si

quieres modificar un archivo de configuración Web.config que tenga en cuenta este tipo de explotación:

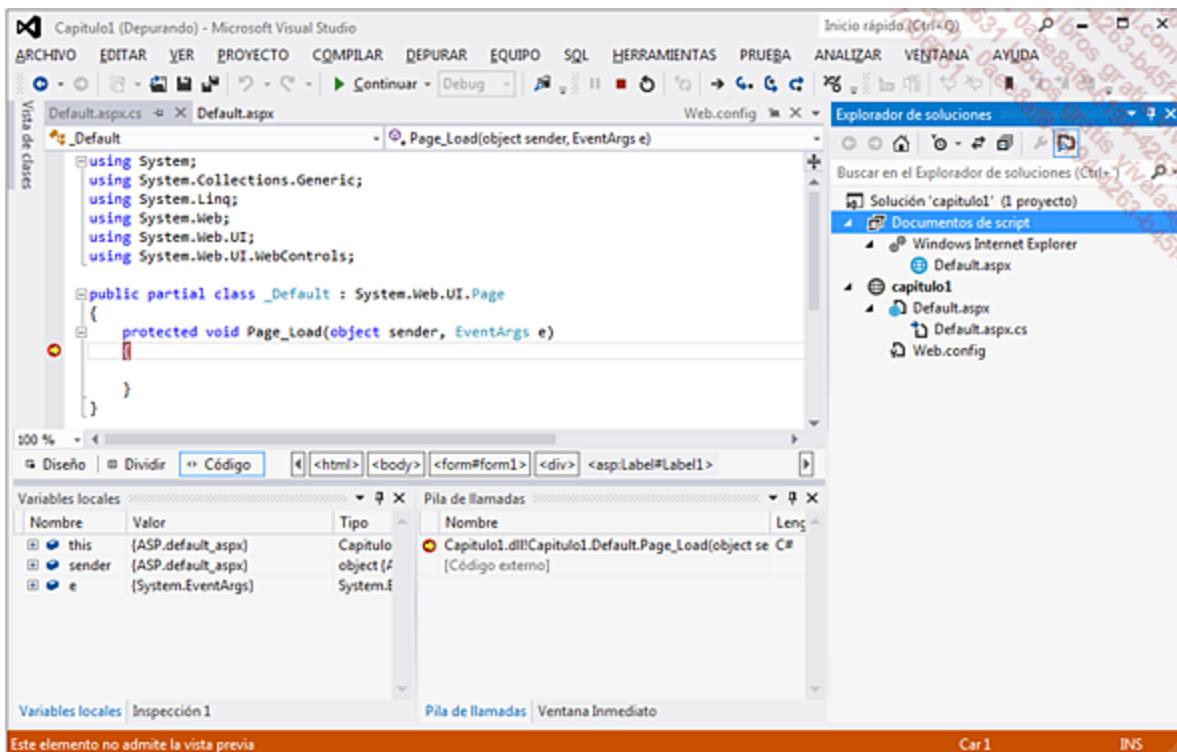


Visual Studio proporciona distintas funciones de ejecución: paso a paso principal, paso a paso detallado, ejecución hasta el final del procedimiento...

Algunas ventanas proporcionan información sobre el contexto de ejecución: variables locales, espías, pila de llamadas, consola de salida. La depuración de Visual Studio es, sin duda, un punto fuerte de la herramienta.



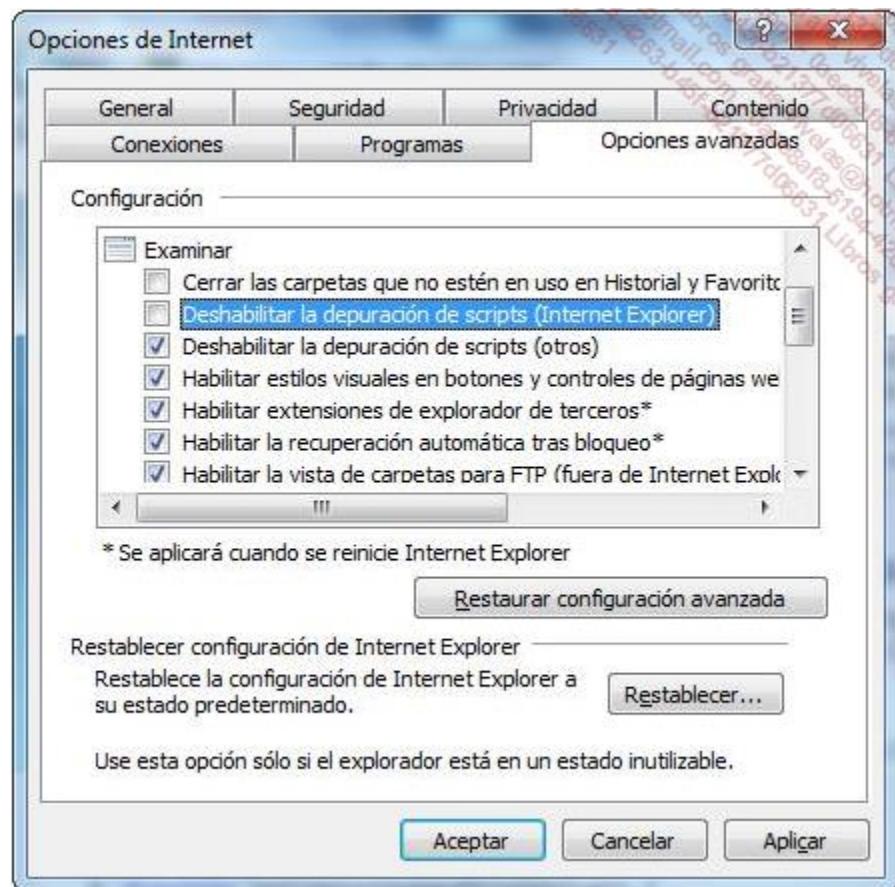
Durante la depuración, el explorador de soluciones se enriquece con un apartado **Documentos de script**. Haciendo doble clic sobre el ítem correspondiente a la página en curso de ejecución se abre una ventana trazando el renderizado HTML de la página:



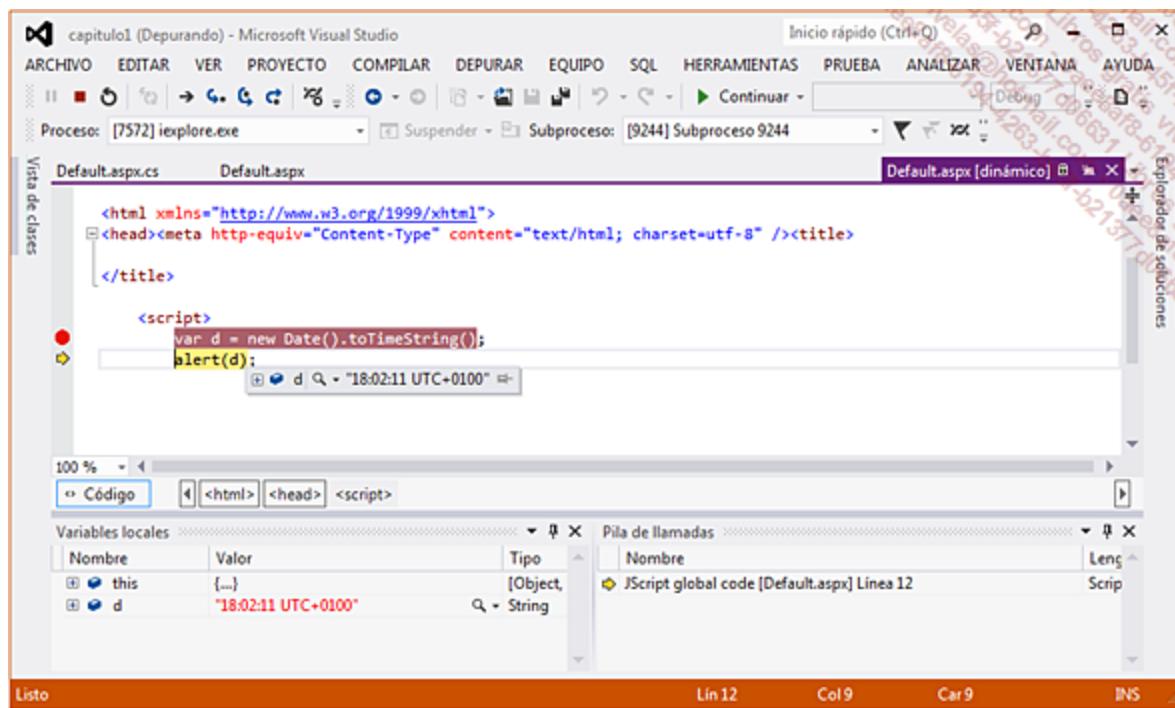
El hecho de incluir la depuración de JavaScript supone una funcionalidad muy valiosa de Visual Studio. Si Internet Explorer no está configurado adecuadamente, Visual Studio genera una alerta cuando se ejecuta la sesión de depuración:



Abra la pestaña **Opciones avanzadas** del menú **Herramientas - Opciones de Internet** del navegador, a continuación detalle la sección **Examinar** y, a continuación, desmarque la opción **Deshabilitar la depuración de scripts (Internet Explorer)**.



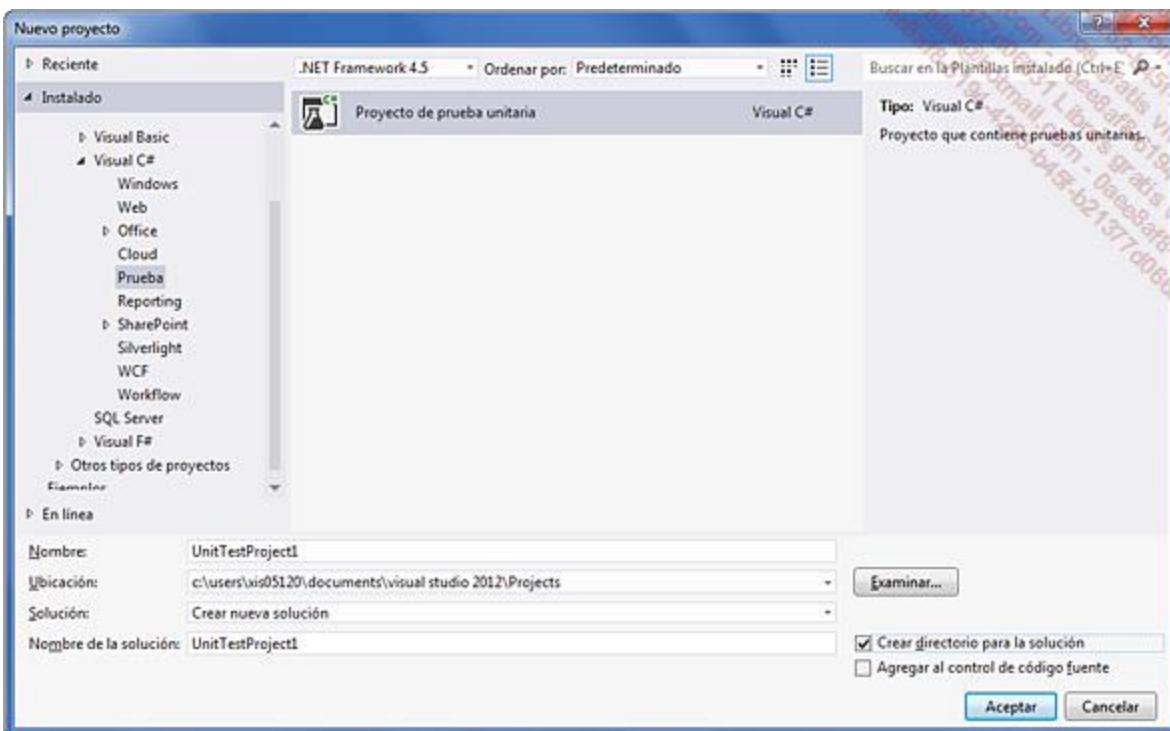
Sólo queda definir puntos de ruptura en su código JavaScript, procediendo de forma similar a como se hizo con el código C#. Los comandos y las ventanas de observación habituales están, ahora, disponibles:



## Pruebas unitarias y profiling

La versión Visual Studio Professional proporciona nuevos servicios de pruebas unitarias y de análisis de calidad de código (profiling). Las pruebas unitarias son clases que despliegan una secuencia de llamadas de métodos para los componentes de una aplicación. Los resultados devueltos por estos métodos se comparan con sus valores teóricos. Se elabora, a continuación, un informe destinado al programador.

Para crear una prueba, es posible partir de un componente que se quiere probar (una clase) y utilizar el menú **Archivo - Nuevo proyecto** y, a continuación, escoger **Prueba** en la lista de tipos de proyecto:



Se creará un proyecto, puesto que las pruebas se ejecutan en un proceso distinto al de la aplicación ASP.NET.

La clase de prueba consiste en una serie de métodos que verifican el funcionamiento de cada función definida en la etapa anterior. El programador debe indicar, para cada una de ellas, los valores de los parámetros, los valores esperados, el tipo de comparación y el mensaje de error.

```
/// <summary>
/// Prueba para cuadrado
/// </summary>
[TestMethod()]
[HostType("ASP.NET")]
[UrlToTest("http://localhost/capitulo01")]
public void cuadradoPrueba()
{
    double param = 2;
    double expected = 4;
    double actual = new Calculadora().cuadrado(param);
```

```

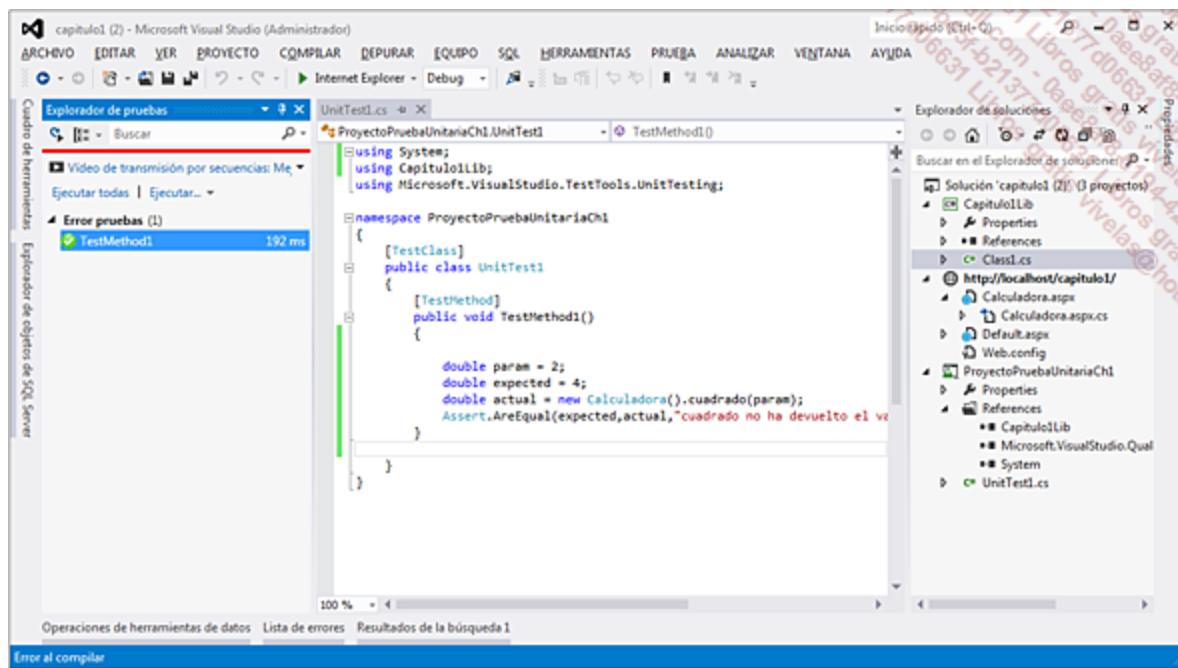
        // La creación del descriptor de acceso privado por
'Microsoft.VisualStudio.TestTools.TypesAndSymbols.Assembly'
ha fracasado

        Assert.AreEqual(expected,actual,"cuadrado no ha devuelto
el valor correcto");
}

```

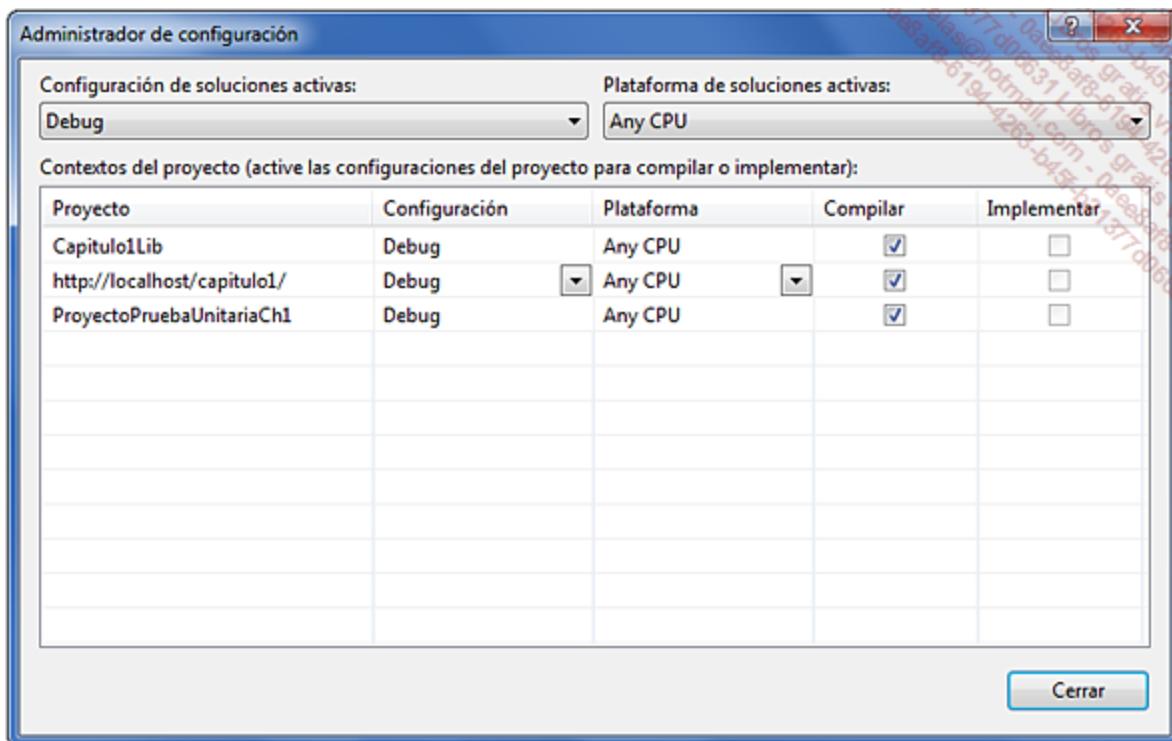
Una vez informados los métodos de prueba, el proyecto de prueba se arranca como los demás. Va a instanciar cada componente y le va a aplicar la secuencia de pruebas.

La ventana **Explorador de pruebas** muestra el conjunto de resultados:



## Configuración

Visual Studio gestiona su proceso de construcción del ensamblado como un **makefile** (véase más adelante MS Build). Para facilitar la industrialización, se crean dos perfiles de compilación para cada proyecto de aplicación: Debug y Release. El programador puede cambiar de perfil y crear otros a partir del gestor de configuración. Éste está accesible desde el menú **Compilar** o desde el menú contextual **Administrador de configuración**.



Una nueva herramienta, accesible desde el menú **Sitio Web, Configuración ASP.NET**, facilita la gestión del archivo Web.config. Se estudiará con detalle en el capítulo Personalización y securización.

**Herramienta Administración de sitios Web**

Aplicación:/capítulo1  
Nombre actual del usuario:SCISB\XIS05120

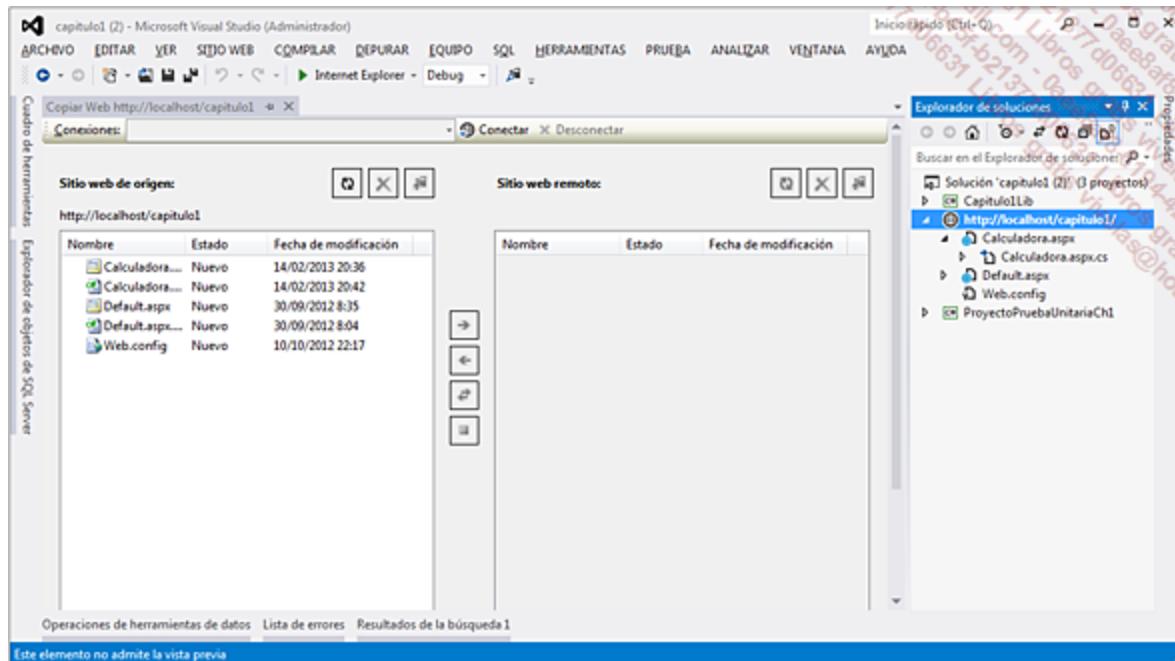
<u>Seguridad</u>	Le permite configurar y editar usuarios, roles y permisos de acceso para el sitio. El sitio está utilizando la autenticación de Windows para la administración de usuarios.
<u>Configuración de la aplicación</u>	Le permite administrar los valores de configuración de la aplicación.
<u>Configuración del proveedor</u>	Le permite especificar dónde y cómo almacenar los datos de administración utilizados por el sitio Web.

## Despliegue

Si bien Microsoft prevé que sus sitios web se desplieguen mediante un asistente MSI (estudiado en el capítulo Configuración, despliegue y administración), muchos sitios se copian simplemente, más o menos de forma manual, desde la carpeta origen hasta la carpeta destino.

Esta función está accesible desde la opción de menú **Sitio Web - Copiar el sitio Web**.

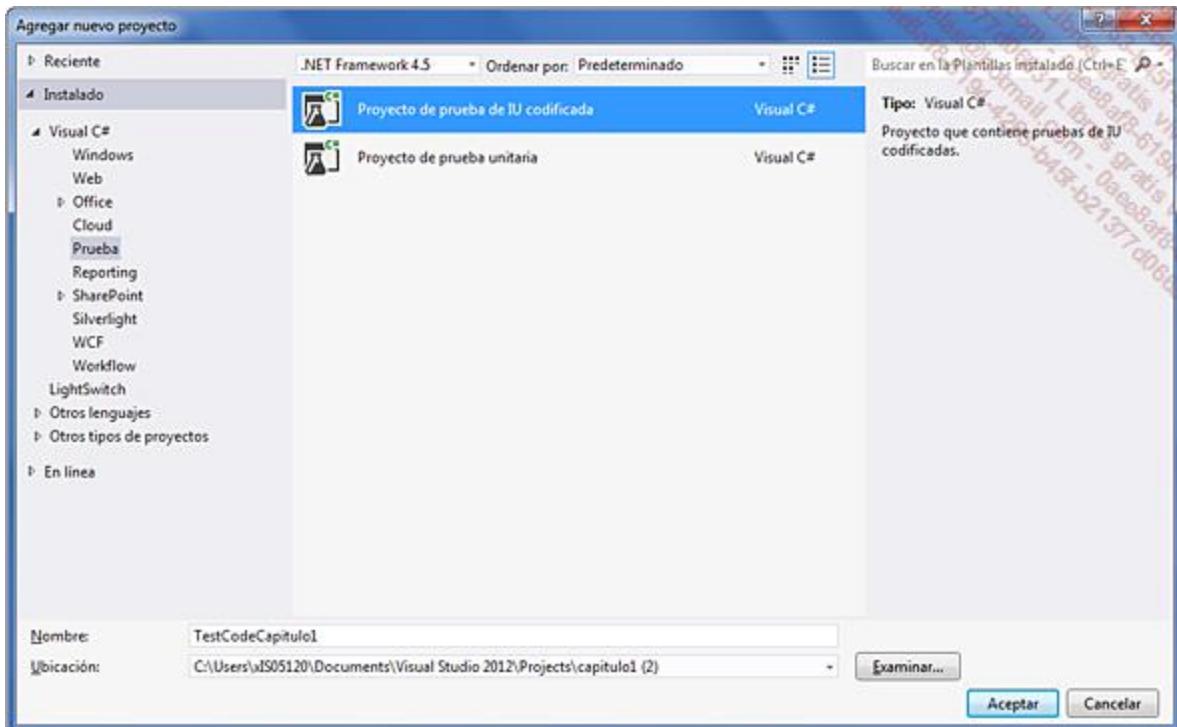
Y, una buena sorpresa, el cliente FTP se ha integrado y mejorado desde Visual Studio 2008. El capítulo Configuración, despliegue y administración ofrece detalles sobre su uso.



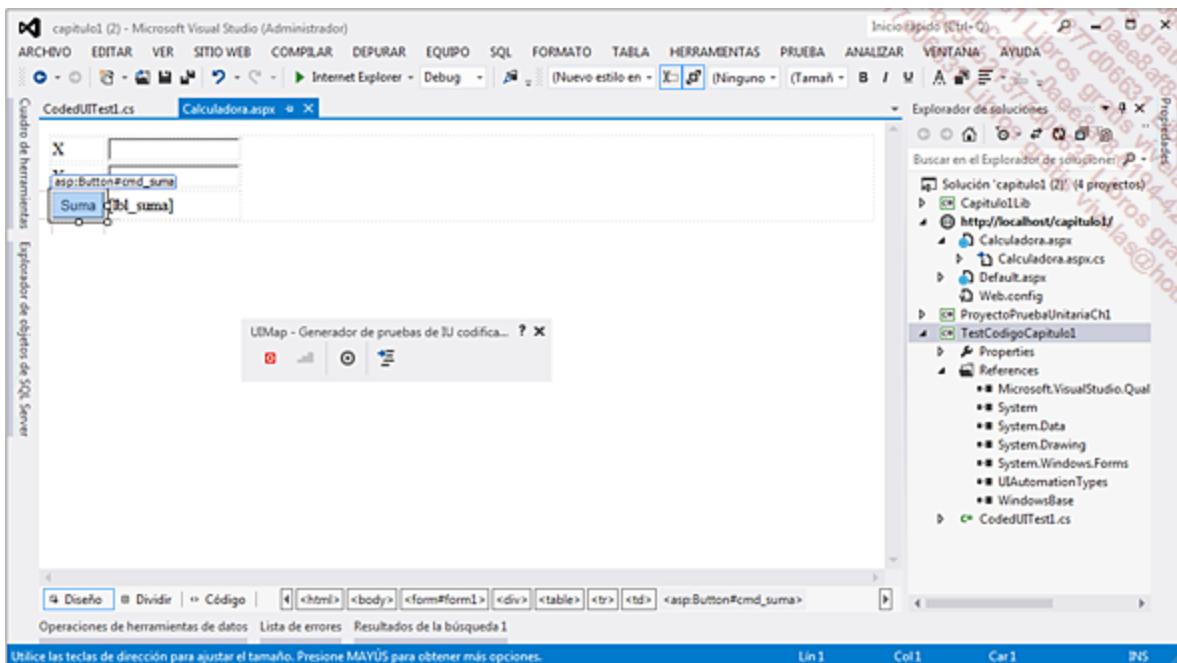
#### d. Las pruebas codificadas de interfaz de usuario

Visual Studio soporta, desde ahora, pruebas automáticas de la interfaz de usuario. Mientras las pruebas unitarias son útiles para probar los módulos de código, su limitación es evidente y conviene, por tanto, simular el encadenamiento de diferentes acciones del usuario.

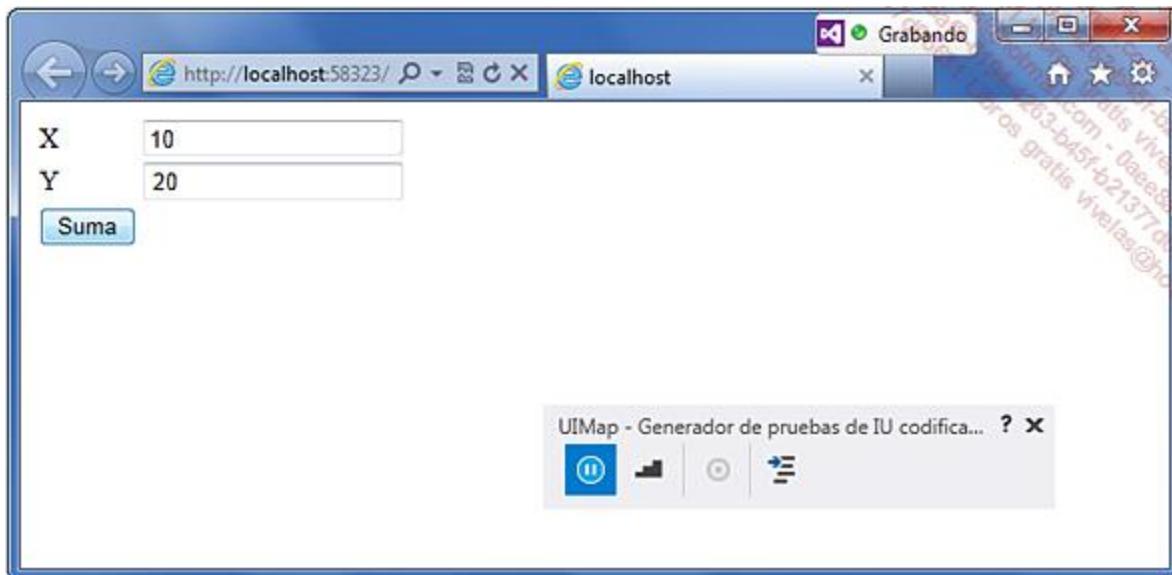
Como con las pruebas unitarias, es preciso utilizar el comando **Nuevo proyecto** para acceder a las pruebas codificadas de interfaz de usuario:



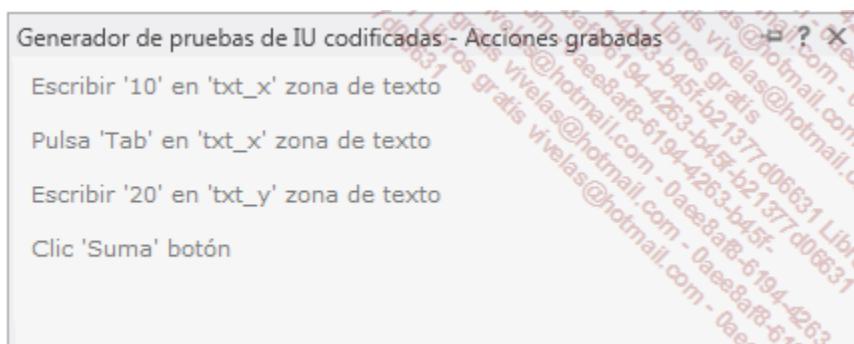
Una prueba codificada de interfaz de usuario se realiza en dos etapas; en primer lugar se registran las acciones del usuario - en nuestro caso sobre una página ASP.NET - y, a continuación, se vuelven a ejecutar de forma automática dichas acciones desde Visual Studio. Existe una barra de herramientas especial que incluye un botón para iniciar la grabación:



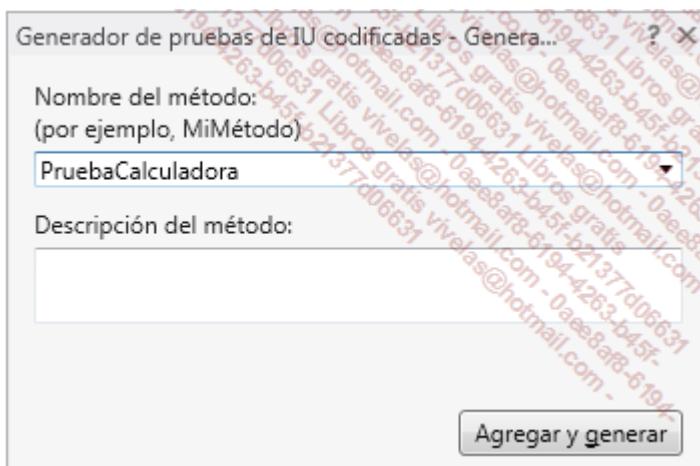
Antes de iniciar la grabación, es preciso "ejecutar" la aplicación que se quiere probar, y, si es posible, en modo de depuración, puesto que podría comprometer la generación de código en función de la secuencia de acciones del usuario. Una vez iniciada la grabación, el usuario sólo tiene que manipular su interfaz gráfica según las acciones previstas:



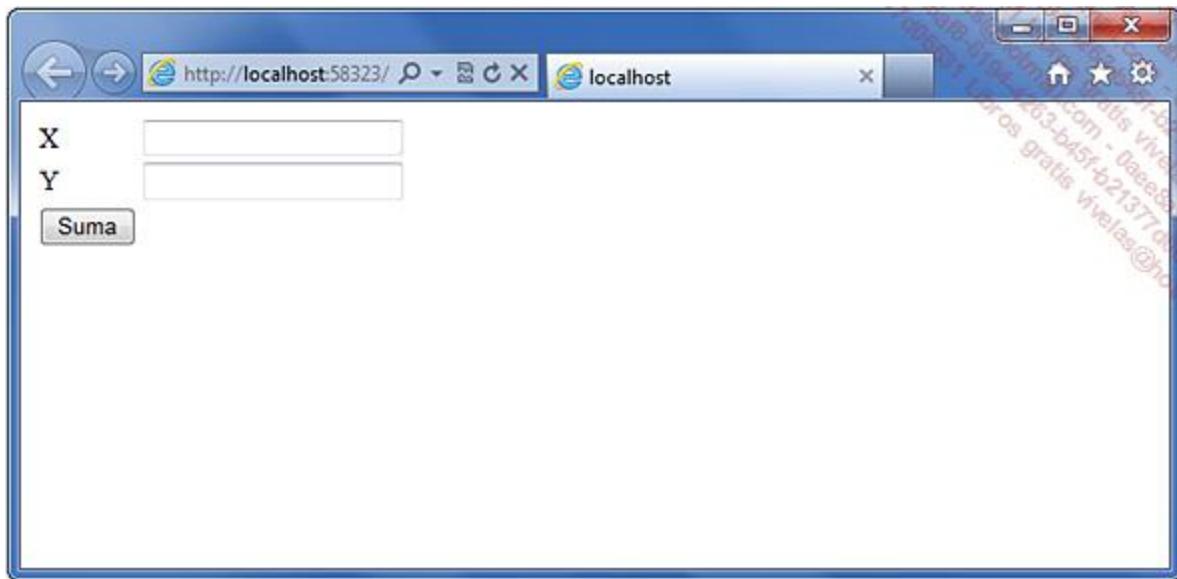
El botón **Pausa** deshabilita la grabación, y otro botón muestra el conjunto de acciones que componen la prueba:



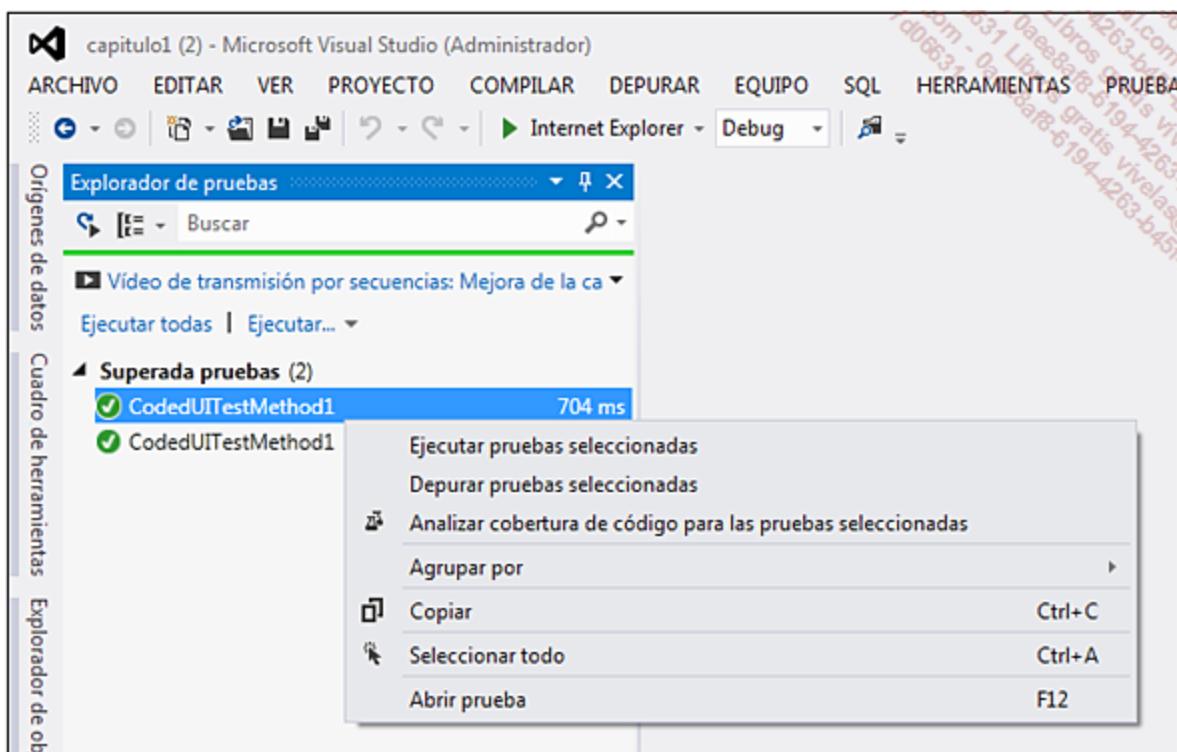
Una vez finalizada la secuencia, conviene materializarla en forma de un archivo de cartografía de prueba (uimap) y de un método - C# o VB.NET - que ejecutará la secuencia bajo demanda.



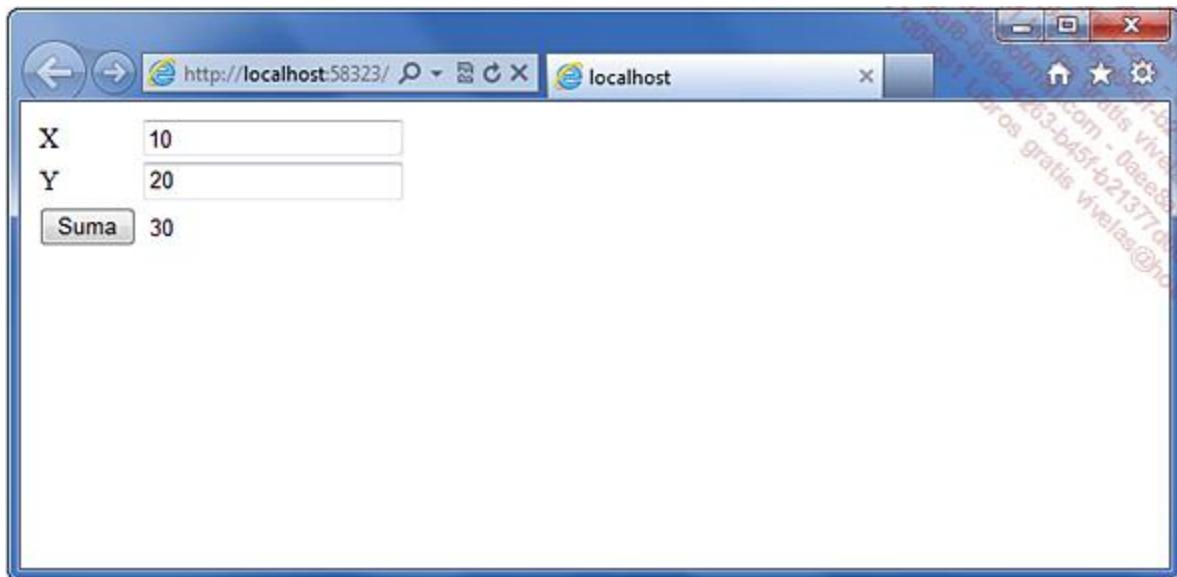
Para repetir la prueba, hay que reiniciar la aplicación:



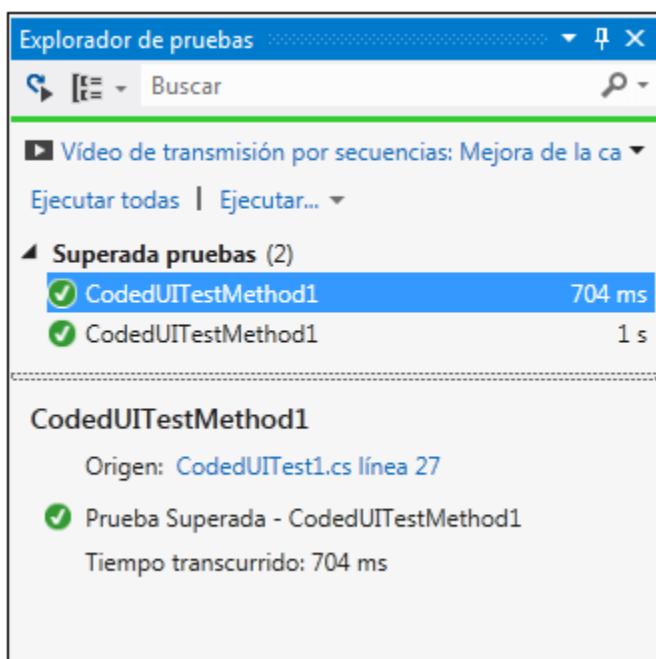
Y, a continuación, ejecutar la prueba desde Visual Studio:



Visual Studio sigue las instrucciones de la prueba y manipula la interfaz de usuario sin acción alguna por su parte, de forma automática:



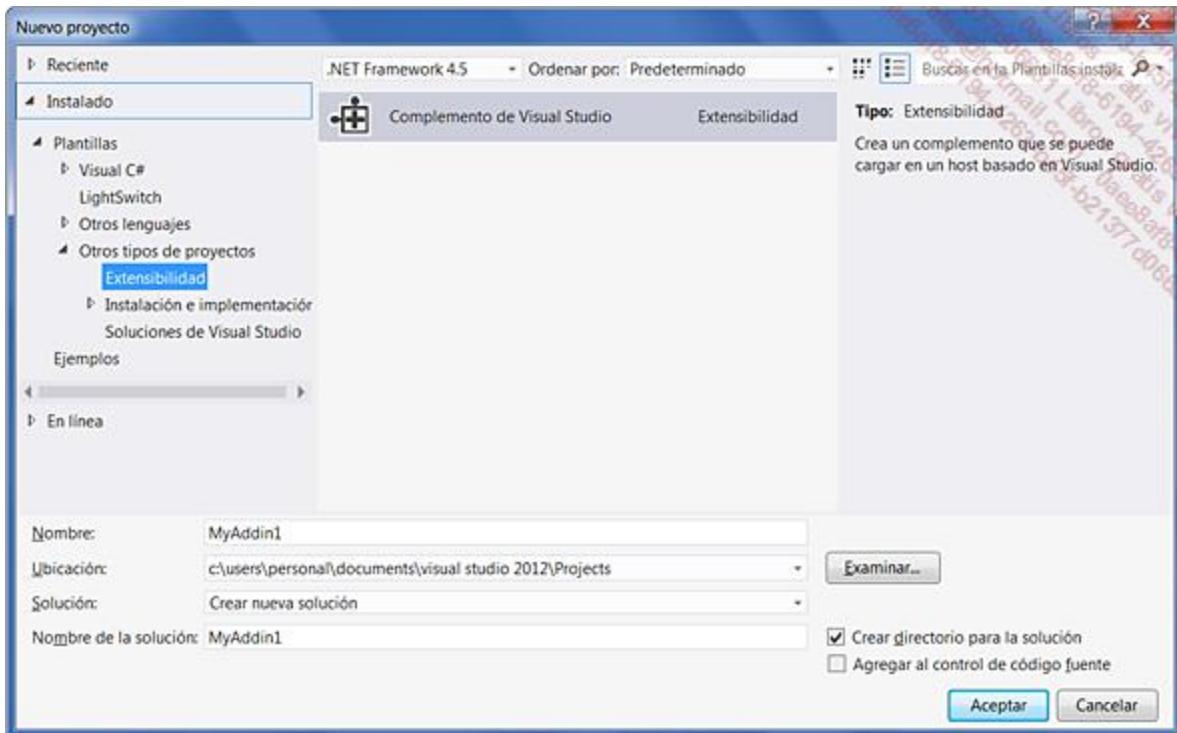
El resultado de la prueba es positivo:



#### e. Las extensiones

Microsoft quiere hacer de Visual Studio una plataforma abierta de desarrollo, de manera similar a Eclipse, ipero sin proveer el código fuente! Esto no es necesario, y el modelo de extensiones puede funcionar perfectamente, pues muchos fabricantes de software proporcionan una integración de su herramienta de desarrollo en Visual Studio. Citemos, por ejemplo, RSM, RAD y DB/2 de IBM, XML Spy de Altova o incluso PHP Phalanger.

A este efecto, Visual Studio provee un modelo de proyecto de extensibilidad.



Sobra decir que la puesta a punto de una extensión requiere mucho trabajo y organización, y se trata de un proyecto independiente. Para la puesta a punto de una aplicación ASP.NET, el valor añadido de tal extensión puede parecer débil. Este tipo de trabajo está, sobre todo, destinado a integrar nuevos entornos en el IDE.

### 3. Gestión del código

Visual Studio 2012 confirma su predilección por la construcción de programas y el código fuente.

#### a. El modo esquema y las regiones

El modo esquema (**Outlining**) es muy útil para navegar en código fuente de gran volumen.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Capitulolib
{
    class Cuenta
    {
        public string Titular { get; set; }
        public int Numero { get; set; }
        public decimal Saldo { get; set; }

        public Cuenta()
        {
            Titular = "";
            Numero = 0;
            Saldo = 0;
        }
    }
}
```

Las regiones son delimitaciones manuales de porciones de código fuente. No afectan a la hora de la compilación, lo que explica que estén precedidas por el símbolo # al comienzo de la línea. A diferencia del modo esquema, las regiones deben respetar ciertas reglas sintácticas; no influyen en la programación, pero el compilador verifica que están ubicadas por parejas.

```
#region Propiedades (estilo C#3)
    public string Titular { get; set; }
    public double Saldo { get; set; }
#endregion
```

Las regiones de código dirigen la presentación del código fuente en el modo esquema: pueden abrirse y cerrarse.

```
public CuentaBancaria()
{
}

#region Propiedades (estilo C#3)

public void Credito(double importe)
{
    saldo += importe;
}
```

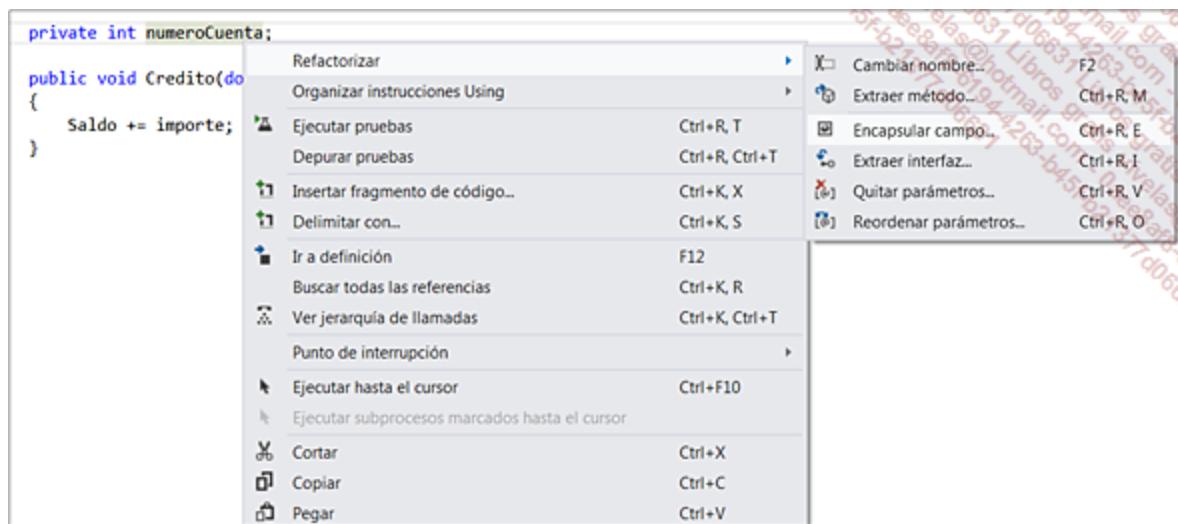
## b. La refactorización (refactoring)

Esta funcionalidad constituye una semi-novedad: los programadores C# o VB.NET ya habían notado que Visual Studio ponía a su disposición servicios de creación o de modificación de código, durante la implementación de una interfaz, por ejemplo, o tras sobrecargar un método.

Estas técnicas se han generalizado bajo la forma de un comportamiento de refactorización (refactoring). Visual Studio provee diversos escenarios:

Renombrar	El nuevo nombre se propaga al conjunto del código fuente. Es un Buscar/Reemplazar inteligente.	Las
Extraer método	Útil para crear una interfaz.	
Encapsular un campo	Crear la lógica de propiedad (get, set) para un campo.	
Extraer interfaz	Útil para componentes de negocio.	
Convertir una variable local en un parámetro de un método	Las llamadas al método se modifican para integrar el nuevo parámetro.	
Suprimir/Reordenar parámetros	Las llamadas al método se modifican para tener en cuenta los cambios.	

herramientas de refactorización están accesibles desde el menú **Refactoring** o desde el menú contextual. Para probar una de ellas, declare un campo privado y utilice la herramienta **Encapsular campo**:



Tras la refactorización, Visual Studio agrega la propiedad NumeroCuenta:

```
private int numeroCuenta;

public int NumeroCuenta
{
    get { return numeroCuenta; }
    set { numeroCuenta = value; }
}
```

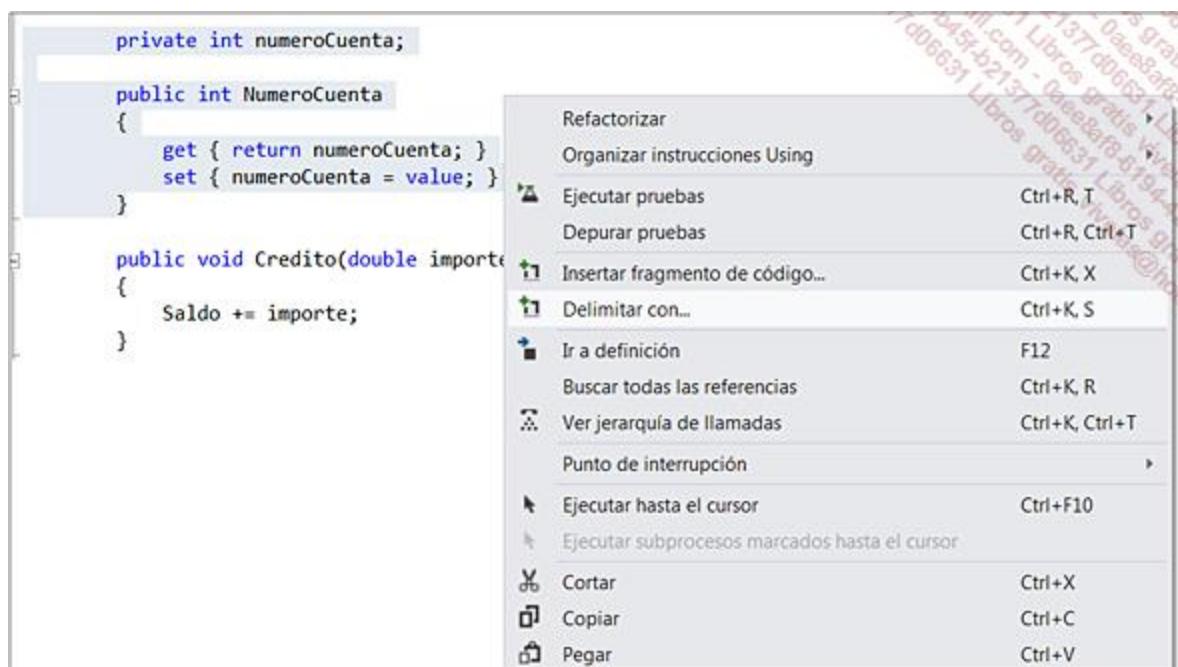
### c. Los fragmentos de código (code snippets)

Los códigos snippets están, principalmente, destinados más bien a la creación contextual de código fuente que a la propagación de modificaciones.

#### Utilizar un snippet existente

Existen tres formas de aplicar un fragmento de código. La primera consiste en introducir el nombre de un fragmento y, a continuación, pulsar dos veces sobre la tecla de tabulación. La segunda utiliza el menú contextual **Insertar fragmento de código**, lo que puede ser muy útil si no tenemos en mente el nombre exacto del fragmento de código. La tercera se basa en el uso del menú contextual **Delimitar con**.

Uno de los fragmentos de código útiles se llama **#region**. Antes de insertarlo, subraye la zona que quiere convertir en región.



Al finalizar la inserción, el código se modifica. Ciertas partes en azul debe completarlas el programador. En nuestro caso, se trata del nombre de la región.

```
#region Propiedad NumeroCuenta
private int numeroCuenta;

public int NumeroCuenta
{
    get { return numeroCuenta; }
    set { numeroCuenta = value; }
}
#endregion
```

#### Descargar snippets

Visual Studio se distribuye con un conjunto de códigos snippets, algunos muy simples pero útiles (escribir un constructor, una secuencia try/catch), otros que se utilizan de forma menos

frecuente. Microsoft difunde en su sitio web bibliotecas de snippets correspondientes a distintos temas: gestión de colecciones, gestión de genéricos, desarrollo web o Windows, construcciones de estados Crystal Report...

Estas bibliotecas pueden descargarse en la siguiente dirección:

<http://msdn.microsoft.com/es-es/library/vstudio/z41h7fat.aspx>

## Crear snippets

Es muy sencillo crear un nuevo fragmento de código. Visual Studio utiliza la carpeta Mis documentos\Visual Studio 2012\Code snippets\C#\My code snippets para integrar los fragmentos de código del usuario a la colección disponible.

He aquí el código de un fragmento **max.snippet** que puede ubicar en la carpeta indicada anteriormente. Este fragmento compara dos variables y devuelve el mayor valor.

```
<?xml version="1.0"?>

<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2012/
CodeSnippet">

    <CodeSnippet Format="1.0.0">

        <Header>

            <Title>Mayor valor de dos variables</Title>
            <Author>Brice-Arnaud Guerin</Author>
            <Description>Compara dos valores y devuelve la de mayor valor
</Description>
            <Shortcut>maxab</Shortcut>
            <SnippetTypes>
                <SnippetType>Expansion</SnippetType>
            </SnippetTypes>
        </Header>
        <Snippet>

            <Declarations>
                <Literal>
                    <ID>A</ID>
                    <ToolTip>Reemplazar por la primera variable</ToolTip>
                    <Default>a</Default>
                </Literal>
                <Literal>
                    <ID>B</ID>
                    <ToolTip>Reemplazar por la segunda variable</ToolTip>
                    <Default>b</Default>
                </Literal>
            </Declarations>
        </Snippet>
    </CodeSnippet>
</CodeSnippets>
```

```

<Code Language="csharp">
    <! [CDATA[ ($A$>$B$?$A$:$B$) $end$ ]]>
</Code>
</Snippet>

```

```

</CodeSnippet>
</CodeSnippets>

```

Se trata de un archivo XML, pero Visual Studio 2012 es muy eficaz para guiar al usuario en su inclusión. Una vez guardado, el fragmento de código ya está activo, no es necesario reiniciar Visual Studio. Su aplicación es inmediata:

Sólo queda reemplazar a y b por los nombres de las variables x e y, Visual Studio ajusta el fragmento de forma adecuada.

## 4. Documentación

El compilador C# permite extraer anotaciones XML situadas en los comentarios que figuran en el código fuente. Dichos comentarios se recopilan en un archivo XML llamado documentación del proyecto.

Esta función ya estaba presente en las versiones anteriores del compilador y de Visual Studio. Tras aplicar el conmutador /doc, el compilador csc genera un archivo XML en el lugar indicado, generalmente cerca del ensamblado resultante.

```

/// <summary>
/// Calcula el nuevo saldo
/// </summary>
/// <param name="importe">importe a aumentar el crédito.</param>
/// <returns>Nuevo saldo</returns>
/// <exception cref="exception.html"></exception>
public double aumentarCredito(double importe)
{
    if (importe < 0)
        throw new System.Exception("Importe negativo");
    saldo += importe;
    return saldo; // no debe ser negativo
}

```

No obstante, esta forma de agregar comentarios al código, mediante etiquetas XML, presenta dos problemas: en la versión 2003, Microsoft no provee una hoja de estilo estándar XSL para traducir la documentación XML en un archivo HTML o en un archivo hlp/chm (archivo de ayuda

de Windows compilado). Gracias a un producto Open Source, NDOC, es posible resolver este inconveniente.

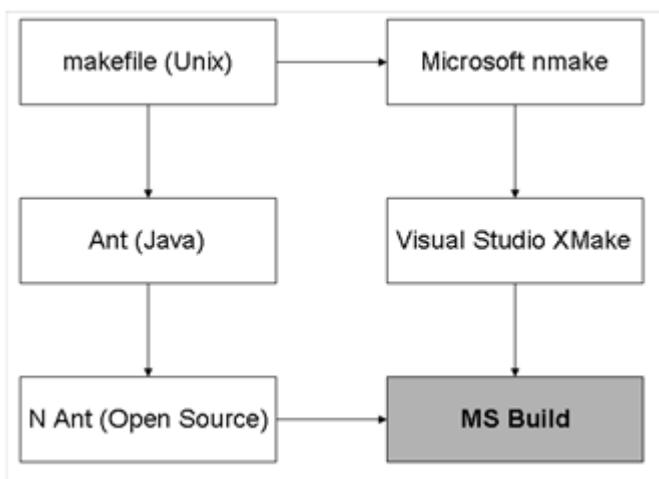
En la versión 2005, el nuevo modelo de compilación de las páginas y los sitios web utiliza el conmutador /doc o la aplicación de una herramienta externa tal como NDOC, delicados. En este caso, es preferible utilizar la reflexión de ensamblados para extraer, al menos, la estructura del programa. Tratándose de ensamblados dinámicos, la operación no es sencilla.

En cualquier caso, estas dificultades invitan al desarrollador a ubicar la mayor parte de su código en DLL referenciadas por el sitio web. El código fuente de estas DLL puede incluirse sin dificultad a la hora de crear una documentación.

## 5. La herramienta MS Build

La compilación de un archivo de código fuente no es sino una etapa más en la formación del ejecutable reconocido por el sistema operativo. Los compiladores explotan, en ocasiones, el flujo de otra herramienta llamada pre-procesador. Como salida, el compilador fabrica un objeto archivo (.netmodule en .NET) que estará enlazado con las bibliotecas necesarias para la ejecución del programa final (ensamblado).

Este proceso se ha automatizado de dos formas: mediante un EDI que gobierna el encadenamiento de las herramientas que intervienen en la elaboración de un programa concreto, o bien a través de un script más o menos general.



La herramienta MS Build generaliza a la vez el enfoque del EDI Visual Studio y del script general N Ant. Tras la versión 2005, MS Build se ha convertido en una herramienta que se incluye en la distribución de la versión 2.0 del framework .NET.

Visual Studio 2012 crea archivos de construcción de proyectos .csproj según las instrucciones MS Build, dichas instrucciones se describen mediante tags XML.

De hecho, un archivo de script MS Build contiene, al menos, la lista de archivos que hay que compilar. Visual Studio completa este script agregando los comandos necesarios para la creación del ensamblado. No obstante, MS Build también puede utilizarse por separado. En este caso, el script contiene no sólo los archivos que se quieren procesar sino también los comandos que hay que aplicar.

### Creación de un script MS Build

MS Build basa su funcionamiento en ANT y makefile. El script define los ensamblados que hay que construir y detalla las operaciones que son necesarias para elaborarlos. Estas operaciones se denominan tareas. Las tareas más útiles consisten en crear carpetas, invocar al compilador o al editor de enlaces.

Para crear un script MS Build, basta con crear un archivo XML que tenga la extensión .proj en un proyecto Visual Studio. Cuando Visual Studio reconoce el espacio de nombres (**xmlns**) asociado a la semántica MS Build, resulta muy sencillo encontrar las distintas construcciones posibles.

```
<?xml version="1.0" encoding="utf-8" ?>
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="ma_dll_msbuild">
    <MakeDir Directories="salida"/>
    <Csc Sources="Class1.cs" TargetType="library" OutputAssembly=
      "salida/test.dll"/>
  </Target>
</Project>
```

Para ejecutar el script, es preciso utilizar la siguiente línea de comandos:

```
msbuild build1.proj
```

Si el script se desarrolla con normalidad, se construirá una DLL test.dll en la carpeta **salida** a partir del archivo de código fuente Class1.cs.

### **Interés para los sitios web ASP.NET**

Como ocurre con la documentación y, especialmente, con el funcionamiento del nuevo modelo de compilación, el uso de MS Build y de Visual Studio presenta poco interés para un sitio web ASP.NET. No obstante, un sitio web no se limita a un conjunto de páginas y de clases complementarias (Helper). Con frecuencia, los sitios web basan su funcionamiento en objetos soportados por DLL. En tal caso, MS Build puede ser útil para crear procedimientos de compilación diversificados, y ejecutarlos como tareas de fondo. Microsoft explota dicho concepto en su servidor de trabajo colaborativo Team System Server.

## C#4 de un vistazo

Tras haber descrito las características del nuevo entorno de desarrollo y, en particular, de Visual Studio, vamos a continuación a descubrir la evolución del lenguaje C#. En esta presentación figuran aquellos aportes del lenguaje que tienen un impacto directo en el desarrollo ASP.NET 4.5. El lector encontrará un estudio más sistemático del lenguaje C#4 en el libro C# 4.0, de Ediciones ENI.

### 1. Clases parciales

Se trata de un mecanismo que aligeraba el modelo de compilación de los sitios web ASP.NET 1.X. Recordemos que, en este modelo, Visual Studio generaba mucho código y que el resultado se compilaba antes de la ejecución. En el nuevo modelo, el código escrito por el programador se enriquece en el servidor de aplicaciones ASP.NET (bajo la forma de inyección de código intermedio IL o de código C# compilado dinámicamente), y por Visual Studio gracias a las clases definidas en varios archivos de código fuente, las clases parciales.

Desde el punto de vista del CLR las clases están definidas completamente. No obstante, el proceso de compilación admite que la definición de un tipo se reparta en varios archivos de código fuente.

De este modo, podemos tener en un primer archivo Test1.cs la siguiente definición:

```
public partial class Test
{
    private string info;

    public string Info
    {
        get { return Test.info; }
        set { Test.info = value; }
    }
}
```

En un segundo archivo, completamos la definición de la clase con:

```
partial class Test
{
    public override string ToString()
    {
        return "test";
    }
}
```

El programador que utilice una instancia de la clase Test tendrá acceso a ambos miembros Info y ToString. El hecho de que la clase se defina en varios archivos no cambia en nada su uso.

Las clases parciales se introducen para simplificar el desarrollo ASP.NET. A día de hoy, Microsoft no propone ningún otro uso.

## 2. Métodos anónimos

El uso de la programación orientada a objetos se ha democratizado gracias a la llegada de las interfaces gráficas. Para disminuir el número de líneas de código, los desarrolladores se han habituado a crear, para cada elemento gráfico - ventana, botón, área de texto... - una clase asociada. No obstante, los primeros lenguajes orientados a objetos disponibles para implementar estos entornos gráficos, tales como C++ o Java, no conocían la noción de eventos. La programación era particularmente delicada cuando se trataba de responder a una solicitud del usuario. El lenguaje C++ sólo disponía de punteros, y el lenguaje Java 1 de referencias, de modo que los programas gráficos debían seguir las reglas de modelización del programa.

A continuación, Microsoft introduce el concepto de evento en Visual Basic y en sus componentes C++ Active X. Por su lado, Sun propone el concepto de clase anónima anidada. Estos mecanismos tenían como objetivo reducir el fuerte acoplamiento que existía entre un componente que desencadenaba un evento (un botón, por ejemplo) y otro que respondía a dicho evento. Si se atiende al objetivo con el que se han creado, estos constructores generan una sobrecarga de código y un coste de diseño importantes.

Con C#2, Microsoft introduce los métodos anónimos. Estos métodos aligeran el régimen de eventos y de sus delegados, y evitan al programador tener que crear clases o métodos que sólo servirán una única vez; ésta es, en efecto, la primera regla que hay que respetar cuando se diseña un programa. Sólo deben ser reutilizables aquellos fragmentos de código susceptibles de volverse a ejecutar!

### a. Eventos internos

Una clase posee cuatro tipos de miembros: campos, propiedades, métodos y eventos. Los eventos son similares a delegados multipropósito.

Desencadenar un evento supone enviar una señal que provoca una llamada a métodos que lo van a gestionar. Estos métodos reciben parámetros útiles para su procesamiento. Por lo general, los parámetros incluyen las condiciones que envuelven a dicho evento.

Tomemos como ejemplo la clase Termostato. Define un evento OnSobrecalentamiento y su delegado de tipo asociado del\_sobrecalentamiento. El delegado tipo representa la firma del evento.

```
class Termostato
{
    public delegate void del_sobrecalentamiento(int temp);
    public event del_sobrecalentamiento OnSobrecalentamiento;
}
```

Agreguemos a la clase Termosato un método regular() encargado de producir el evento si la temperatura excede el umbral de 100°.

```
public int temperatura;
public void regular(int temp)
```

```

{
    temperatura = temp;
    if (temp > 100)
        if (OnSobrecalentamiento != null)
            OnSobrecalentamiento (temp); // desencadenar evento
}

```

Una vez definido el componente Termostato, creamos en otra clase la lógica indispensable para reaccionar al evento:

```

Termostato tm = new Termostato();
tm.OnSobrecalentamiento += new Termostato.del_sobrecalentamiento
(tm_OnSobrecalentamiento);

```

A continuación, se muestra el método de gestión:

```

void tm_OnSobrecalentamiento(int temp)
{
    Console.WriteLine("¡Temperatura demasiado alta!");
}

```

El método `tm_OnSobrecalentamiento` representa una gestión externa del evento. Se trata de un procedimiento autónomo (`Console.WriteLine`) y no devuelve ningún resultado (`void`) al objeto instancia de `Termostato`.

A primera vista, puede parecer incongruente definir los eventos privados y gestores de eventos que pertenezcan a la misma clase que emite el evento. Si se desea, por tanto, utilizar el patrón de diseño *Evento* en el interior de una clase sin utilizar el miembro evento privado que generaliza inútilmente el concepto señal/procesamiento, los métodos anónimos son una solución elegante.

Para ilustrar esta construcción, acondicionamos la clase `Termostato` con ayuda de un método `init()` que registra un método anónimo con ayuda de un delegado. El método regular() se convierte en `regular_anonimo()`. En el código que aparece a continuación, la parte en negrita se corresponde con el método sin nombre. La palabra reservada `delegate()` soporta únicamente su firma.

```

public delegate void del_sobrecalentamiento_anonimo();
public del_sobrecalentamiento_anonimo regulador;

```

```

public void init()
{
    regulador =
        delegate()
        {
            if (temperatura > 100)
            {

```

```

        Console.WriteLine("¡Temperatura demasiado alta!");
        temperatura = 30;
    }
}

public void regular_anonimo(int temp)
{
    temperatura = temp;
    if (temp > 100)
        regulador(); // invocación de la función anónima
}

```

En esta nueva construcción de la clase Termostato, la señal no se produce al desencadenar un evento sino con la llamada al método anónimo. Esto basta para desacoplar la parte de detección (`regular_anonimo`) y la parte de procesamiento (el método anónimo) sin generalizar una lógica de evento que no se reutilizará.

De paso, destacamos que un método anónimo puede utilizar los parámetros y variables locales del método que lo invoca (`init`, en nuestro ejemplo), pero también acceder a los campos de la clase (`Termostato`). Sin esta posibilidad, su interés sería mucho menor.

Al final, el método anónimo se invoca cuando la temperatura excede los 100 grados. El resultado es idéntico, pero la construcción es distinta a la utilizada en una implementación basada en eventos. Esta diferencia no es significativa en una única clase, pero puede facilitar enormemente la lectura de un programa completo.

## b. Las funciones auxiliares

En una clase, todos los métodos son públicos y todos los campos son privados. ¿Cuántas veces habremos leído esta afirmación? La realidad conlleva ciertos matices. Algunos campos son públicos, mientras que otros se encapsulan en pseudo-métodos `get` y `set`, formando propiedades. Por último, algunos métodos se marcan con el modificador de acceso `private`, puesto que no se quiere que nadie utilice su procesamiento.

Los desarrolladores siguen, también, recomendaciones de modelizadores de objetos (UML) para decidir si tal o cual método deben ser privados o protegidos. De hecho, la programación orientada a objetos antepone el aspecto de interfaz de la programación en detrimento de su aspecto procedural; con el paso del tiempo, las implementaciones se vuelven simples dado que los desarrolladores no tienen la capacidad de análisis suficiente como para elaborar métodos que, aun siendo privados, no aparezcan en los diagramas de clases establecidos por los diseñadores.

Además, los métodos anónimos son muy útiles para crear funciones algorítmicas que no se quieren promover a la categoría de método. Se trata, por tanto, de las famosas funciones auxiliares.

Para ilustrar este aspecto, vamos a estudiar una función que invierte una lista de caracteres (`cadena`). Para evitar definir una lista según las reglas establecidas (encabezado, elemento, constructor...), vamos a utilizar cadenas basadas en la clase **string**. Pero sólo podremos utilizar

tres operaciones: leer el primer carácter de la cadena (encabezado), extraer el carácter siguiente y comprobar si es igual a una cadena vacía.

Una primera versión del programa tendría el siguiente aspecto:

```
public string reverse(string s)
{
    StringBuilder res = new StringBuilder();
    reverse_aux(s, res);
    return res.ToString();
}

private void reverse_aux(string s, StringBuilder res)
{
    if (s == null || s == "")
        return;

    reverse_aux(s.Substring(1), res);
    res.Append(s[0]);
}
```

Esta versión funciona perfectamente, pero un diseñador puntilloso indicará al desarrollador que el método `reverse_aux`, útil para llevar a cabo nuestro objetivo, no forma parte del diagrama de clases. El desarrollador debe, por tanto, modificar su programa para incluir una función anónima:

```
delegate void del_rs(string s, StringBuilder res);
del_rs f;
public string reverse(string ch)
{

    StringBuilder sb = new StringBuilder();
    f= delegate(string s, StringBuilder res)
    {
        if (s == null || s == "")
            return;

        f(s.Substring(1), res);
        res.Append(s[0]);
    };

    f(ch, sb);
    return sb.ToString();
}
```

```
}
```

La nueva versión del método reverse devuelve exactamente el mismo resultado que la anterior, pero sin recurrir a un método privado reverse\_aux.

### c. Simplificar la edición de código

Este tercer ejemplo de método anónimo confortará a aquellos que piensan que el uso de una clase abstracta o de una interfaz pesada complica considerablemente un programa. Muchos algoritmos son genéricos, es decir, se pueden aplicar a distintas situaciones, distintos tipos de datos, distintos contextos funcionales. Para implementar estos algoritmos genéricos, la programación orientada a objetos proporciona, entre otros, los métodos abstractos y los punteros a funciones.

Proponemos ir un poco más lejos pasando como parámetro a un algoritmo no el puntero a una función útil para su ejecución sino la propia función.

Nuestro ejemplo es una clase, DirectoryFilter, que extrae la lista de archivos de una carpeta determinada. El método list() admite como parámetro una función anónima destinada a validar la selección de un archivo en la lista devuelta. La técnica de selección es arbitraria, lo que explica que no se haya sistematizado en la clase DirectoryFilter.

```
class DirectoryFilter
{
    private string path;

    public string Path
    {
        get { return path; }
        set { path = value; }

    }

    public DirectoryFilter(string path)
    {
        this.Path = path;
    }

    public delegate bool del_filtro(string file, string path);
    public string[] list(del_filtro filtro)
    {

        DirectoryInfo dir = new DirectoryInfo(path);
        FileInfo[] files=dir.GetFiles();
        ArrayList ar = new ArrayList();
        for (int i = 0; i < files.Length; i++)
        {
            if (filtro(files[i].Name, files[i].ToString()))
                ar.Add(files[i]);
        }
        return ar.ToArray();
    }
}
```

```

        if (filtro(files[i].FullName, path))
            ar.Add(files[i].FullName);

        return ar.ToArray(typeof(string)) as string[];
    }
}

```

El método `list()` aquí presente invoca, naturalmente, al método del mismo nombre de la clase `java.io.File`. La versión Java admite una interfaz (`java.io.FilenameFilter`) a menudo implementada con ayuda de una clase anónima anidada.

En el caso de C#, nos contentaremos con una función anónima (anidada), pero la construcción es similar:

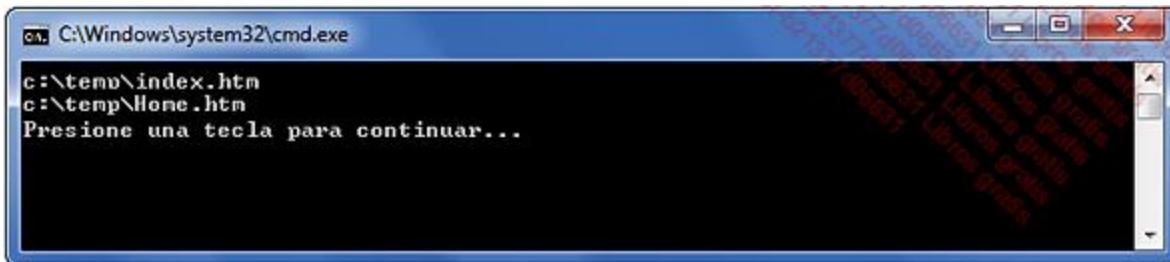
```

DirectoryFilter f = new DirectoryFilter(@"c:\temp");
string[] archivos = f.list(delegate(string file, string path)
{
    return file.EndsWith(".htm");
});

for (int i = 0; i < archivos.Length; i++)
    Console.WriteLine(archivos[i]);

```

¿Qué aporta la función anónima a este ejemplo? El algoritmo que selecciona los archivos que poseen la extensión `.htm` es muy específico. No servirá, desde luego, en otras partes del programa, ni en otras aplicaciones. En este caso no resulta adecuado crear un método, que debe incluirse, obligatoriamente, en una clase. La escritura del código se ve, así, simplificada.



### 3. La inferencia de tipo

Se trata de un mecanismo que requiere que el compilador deduzca, él mismo, el tipo de una expresión y asigne una variable que represente a este tipo. La inferencia de tipo es útil en LINQ, donde la naturaleza de los resultados varía de una consulta a otra.

#### C#

```

int a = 1;

var suma = a + 2; // El compilador deduce que se trata de un entero
Console.WriteLine(suma.GetType().FullName);

```



Para el programador C#, el cambio es más "brutal" que el que supone en VB.NET. En efecto, este último lenguaje es de la familia de lenguajes débilmente tipados, donde el compilador cede, a menudo, al entorno de ejecución la tarea de determinar el tipo y, eventualmente, de realizar la conversión necesaria.

¿Cuál es el interés de la inferencia de tipo? Si nos limitamos a nuestro ejemplo, no supone una gran ventaja. Pero cuando abordemos las consultas LINQ, que encadenan varias series de expresiones, su uso se vuelve crucial para conservar una buena legibilidad del código.

#### a. Las expresiones lambda

Las expresiones lambda generalizan los métodos anónimos, ofreciendo un soporte a VB.NET. Se apoyan, naturalmente, en delegados, que obran por debajo.

Las expresiones lambda se denominan así debido al lenguaje LISP (List Processing), inventado para generalizar el cálculo-λ.

El ejemplo que se muestra a continuación complementa al anterior. Comparte la definición del delegado dsuma. Observe, en esta sintaxis, la desaparición de la palabra clave return, implícita. En C#, el operador => puede leerse "da como resultado".

#### C#

```
// expression lambda
dsuma d3 = (int a, int b) => a + b;

// llamada a la expresión lambda
int s3 = d3(1, 2);
Console.WriteLine("s3=" + s3);
```

### 4. Clases dinámicas y tipos anónimos

La sintaxis de las consultas de selección SQL permite al programador escoger las columnas que quiere incluir en el flujo del resultado. Estas columnas pueden, a continuación, agregarse, filtrarse... Todos los registros de la tabla SQL se componen de las mismas columnas, pero el rendimiento se ve evidentemente afectado por el número que figure en la consulta SELECT.

En programación no SQL, la definición estricta del tipo es la base de un lenguaje fuertemente tipado: todas las instancias de una clase reservan el mismo espacio de memoria para representar el conjunto de campos (atributos), tengan o no valor. Ésta es, por otro lado, una diferencia importante entre SQL y estos lenguajes: estos últimos manipulan datos en memoria, mientras

que SQL aprovecha la durabilidad de los datos almacenando sus valores en archivos indexados. Sólo una parte de ellos se carga en memoria, según las consultas.

Para aligerar la carga del CLR y evitar al programador tener que definir clases con todas las combinaciones de atributos posibles, Microsoft ha incorporado los tipos anónimos en C#3 y VB.NET 9. El siguiente ejemplo indica cómo trabajar con ellos. Las propiedades de los tipos anónimos en las ocurrencias nombre e idp son de sólo lectura.

## C#

```
// construye un tipo anónimo que tiene dos propiedades nombre e idp
var p = new { nombre = "Alberto", idp = 1 };

// muestra el nombre del tipo generado por el compilador
Console.WriteLine(p.GetType().FullName);
```

## 5. Extensión de clases sin herencia

¿Cómo agregar un método a una clase, sin derivarla? Utilizando los últimos aportes de los lenguajes C# y VB.NET. En LINQ, esta sintaxis se utiliza junto a los tipos anónimos para encadenar operaciones tales como SELECT, WHERE, ORDER BY.

Esta disposición recuerda a las funciones afines en C++, que se introdujeron para soportar la sobrecarga del operador de inyección << proporcionado por la STL.

Los procedimientos de definición de una extensión difieren de C# a VB.NET. En C#, para definir una extensión de una clase sin derivarla es preciso crear un método estático en el mismo espacio de nombres que donde se empleará. Este método recibe como primer argumento una instancia de la clase a extender. La palabra reservada this recuerda, precisamente, al compilador que debe tratar este parámetro de forma particular:

## C#

```
static class Usuarios
{
    // observe el uso de this como calificador del parámetro s
    // que indica que la clase string es extendida
    public static bool isDate(this string s)
    {
        try
        {
            DateTime.Parse(s);
            return true;
        }
        catch { }
        return false;
    }
}
```

```

    }

}

class Program
{
    static void Main(string[] args)
    {
        string s = "19/5/2007";

        // el compilador verifica que isDate es conocido
        Console.WriteLine(s.isDate());
    }
}

```

## 6. Tipos nullables

Para resolver ciertas dificultades de los tipos valor (estructura, int, double...), Microsoft ha incluido en C# los tipos nullables.

Los tipos nullables se encapsulan, no mediante boxing (con ayuda de object), sino mediante una estructura genérica **Nullable<T>**. La estructura Nullable<T> contiene dos propiedades HasValue y Value que guían al programador para determinar si una variable contiene valor.

```

Nullable<int> v = null;
Console.WriteLine("¿v tiene valor? " + v.HasValue); // falso
Console.WriteLine("¿v es nulo? " + (v != null)); // falso

v = 30; // provee un valor
Console.WriteLine("¿v tiene valor? " + v.HasValue); // verdadero
Console.WriteLine("¿v es nulo? " + (v != null)); // verdadero

```

Como la escritura Nullable<T> es algo pesada, el lenguaje C# define un alias automático: el tipo T se sigue de un signo ?. Los resultados son idénticos:

```

int? a = null; // idéntico a Nullable<int> a;

Console.WriteLine(a == null); // verdadero
Console.WriteLine(a.HasValue); // falso

a = 2;
Console.WriteLine(a == null); // falso
Console.WriteLine(a.HasValue); // verdadero

```

```
Console.WriteLine(a.Value+" "+a); // 2 2
```

Los tipos nullables se utilizarán para acceder a los datos relacionales. En efecto, los SGBD realizan la distinción entre NULL y una inicialización del valor. Como los tipos SQL están relacionados con los tipos C#, ADO.NET utiliza la constante DBNull, lo que supone encapsular el valor en una estructura. La puesta en marcha se ha visto, por tanto, generalizada.

## 7. Iterador

Los iteradores son construcciones lógicas que sirven para enumerar los elementos de un conjunto de tipo tabla, una colección... Se utilizan con el bucle foreach.

Para estudiar la forma de los iteradores asociados a C#1 y C#3, consideremos la clase Melodia:

```
enum Nota { do,re,mi,mi_bemol,fa,sol,la,si }

class Melodia : ColeccionBase

{
    public Melodia() : base()

    {

    }

    public void add(Nota nota)

    {
        this.List.Add(nota);
    }

    public Nota this[int indice]

    {
        get { return (Nota) List[indice]; }
        set { List[indice] = value; }
    }
}
```

### a. Iterador en C#1

Para ejecutar un bucle foreach sobre una colección con C#1, es preciso implementar la interfaz IEnumerable, que expone el método GetEnumerator(). Este método se invoca desde la instrucción foreach, y devuelve una instancia que implementa IEnumerator.

Para respetar las reglas de visibilidad y de reentrada, la clase que implementa IEnumerator puede ser interna a la clase colección:

```
class Melodia : ColeccionBase, IEnumerable
{
    #region IEnumerable Members
```

```

IEnumerator IEnumerable.GetEnumerator()
{
    return new MelodiaIterator(this);
}
#endregion

// clase encargada de proveer las notas que componen la melodía
private class MelodiaIterator : IEnumerator

```

```

{
    private IList lista;
    private int contador;
    public MelodiaIterator(Melodia melodía)
    {
        lista = melodía.List;
        contador = 0;
    }

    #region IEnumerator Members
    object IEnumerator.Current
    {
        get { return lista[contador]; }
    }

    bool IEnumerator.MoveNext()
    {
        contador++;
        return contador < lista.Count;
    }

    void IEnumerator.Reset()
    {
        contador = 0;
    }
}

#endregion
}
}

```

El programa de prueba puede crear una melodía y reproducirla con ayuda del bucle foreach:

```

Melodia melodía = new Melodia();

```

```

melodia.add(Nota.sol);
melodia.add(Nota.sol);
melodia.add(Nota.sol);
melodia.add(Nota.mi_bemol);
foreach (Nota n in melodía)
    Console.WriteLine(n); // ¡po-po-po-pooo!

```



```

sol
sol
sol
mi_bemol
Presione una tecla para continuar...

```

### b. Iterador a partir de C#3

El lenguaje C#3 proporciona una sintaxis más ligera para construir los iteradores. La palabra reservada `yield return` construye una clase idéntica a la clase `MelodiaIterator`:

```

public IEnumerable Notas
{
    get
    {
        for (int i = 0; i < List.Count; i++)
            yield return List[i];
    }
}

```

Con este ejemplo, la clase `Melodia` no necesita implementar `IEnumerator` - aunque sigue siendo, no obstante, una posibilidad. Cabe destacar que la variable interna `i` se memoriza de una llamada a la siguiente. La palabra reservada `yield return` descompone cada llamada al iterador. Podríamos utilizar también varios `yield return` a continuación para construir un bucle. Observe también la existencia de una instrucción `yield break` para interrumpir la secuencia antes de que termine.

El bucle `foreach` es similar al anterior. Devuelve, evidentemente, el mismo resultado:

```

foreach (Nota n in melodía.Notas)
    Console.WriteLine(n); // ¡Siempre la quinta!

```

## 8. Genericidad

La genericidad es una herramienta de programación que evita al programador tener que practicar secuencias copiar-pegar difíciles de mantener. Hasta el día de hoy, esto era posible utilizando el tipo `object` (alias C# del tipo CTS `System.Object`) que servía de base a todas las clases. Este enfoque débilmente tipado muestra rápidamente sus limitaciones en términos de complejidad o de seguridad en el funcionamiento.

El lenguaje C++ proporciona, desde hace tiempo, un mecanismo de clases plantilla (template). El nombre de la clase introduce uno o varios tipos parámetro, a menudo designados por letras mayúsculas, que retoman los campos, parámetros y métodos de la clase. Para el lector que conozca el lenguaje C, las plantillas presentan un uso mucho más seguro que las macros del pre-procesador; el compilador considera, en efecto, cada instancia del modelo utilizado por el programa con todo el rigor y control necesarios.

El lenguaje C#2 retoma en gran parte el enfoque de las plantillas de C++, afortunadamente simplificando su sintaxis y fijando reglas más estrictas para su aplicación.

### a. Definir un tipo genérico

La sintaxis utilizada para declarar una clase genérica utiliza la notación <Type>. El tipo parámetro (generalmente designado por una letra mayúscula T, U, V) se retoma, a continuación, en el interior de la clase para designar al tipo de ciertos campos, variables y parámetros. El siguiente ejemplo implementa una lista de elementos de tipo "T". Este tipo se precisará en el momento de instanciación de la clase y la lista podrá contener valores de tipo string, int, char...

```
class Lista<T>
{
    #region Propiedad elemento
    private T elemento;
    public T Elemento
    {
        get { return elemento; }
        set { elemento = value; }
    }
    #endregion
    #region Propiedad siguiente
    private Lista<T> siguiente;
    internal Lista<T> Siguiente
    {
        get { return siguiente; }
        set { siguiente = value; }
    }
    #endregion

    // una constante genérica
    public const Lista<T> LISTA_VACIA = null;
    #region Constructores
    public Lista()
    {
        siguiente = LISTA_VACIA;
    }
}
```

```

public Lista(T elemento, Lista<T> siguiente)
{
    this.elemento = elemento;
    this.siguiente = siguiente;
}
#endregion
}

```

La sintaxis del constructor presenta una originalidad: el tipo parámetro `<T>` no aparece en su definición. El nombre que sirve para distinguir el constructor difiere, por tanto, del nombre de la clase.

He aquí, ahora, la definición de otros métodos de la clase `Lista<T>`, estáticos o no.

```

// un método estático con un parámetro genérico
public static bool esta_vacia(Lista<T> l)
{
    return l == LISTA_VACIA;
}

public int longitud()
{
    // llamada al método estático
    return Lista<T>.longitud(this);
}

public static int longitud(Lista<T> l)
{
    if (Lista<T>.esta_vacia(l))
        return 0;
    else
        return 1 + longitud(l.siguiente);
}

public void visualizar()
{
    visualizar(this);
}

```

```

}
```

```

public static void visualizar(Lista<T> l)
{
    if (Lista<T>.esta_vacia(l))
        return;
    Console.WriteLine(l.element.ToString() + ",");
    visualizar(l.siguiente);
}

```

### b. Especialización parcial

La especialización parcial consiste en proveer una implementación específica para ciertos valores del parámetro T:

```

// especialización parcial del método visualizar:
// una versión específica para cadenas
public static void visualizar(Lista<string> l)
{
    if (Lista<string>.esta_vacia (l))
        return;

    Console.WriteLine(l.element + ",");
    visualizar(l.siguiente);
}

```

Este enfoque es útil para aprovechar el conocimiento del tipo <string>. El programador puede, entonces, crear una implementación optimizada para el tipo en cuestión.

### c. Uso de un tipo genérico

La variable L no podría declararse como Lista<T>. Es necesario precisar el parámetro utilizado para instanciarla, en la ocurrencia un string. De este modo, el tipo de la variable L es Lista<string>:

```

Lista<string> l=
    new Lista<string>("hola",
    new Lista<string>("a",
    new Lista<string>("todos",
    Lista<string>.LISTA_VACIA)));

```

```

l.visualizar();
Console.WriteLine("\nLongitud: {0}", l.longitud());

```

A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text:  
holo,a,todos  
Longitud: 3  
Presione una tecla para continuar...  
The background of the window shows a watermark with the text "Libros gratis" repeated diagonally.

#### d. El espacio de nombres System.Collections.Generic

La sintaxis de los tipos genéricos C#2 es muy rica, autorizando a enumerar propiedades de la interfaz gráfica aplicables a parámetros <T>, pero evita también el uso de plantillas del lenguaje C++.

Las clases genéricas se utilizan, a menudo, como estructuras de datos dinámicas: pilas, listas, colas de espera, tablas, tablas hash... Microsoft ha reescrito las clases del espacio de nombres `System.Collections` volviéndolas genéricas. Las nuevas clases se ubican en el espacio de nombres `System.Collections.Generic`.

Comparer<T>	Clase básica para implementar algoritmos de ordenación genéricos.
Dictionary<T>	Tabla de hash genérica.
LinkedList<T>	Lista genérica doblemente encadenada.
List<T>	Lista genérica.
Queue<T>	Cola de espera genérica (también llamada pila FIFO).
SortedList<T>	Lista genérica cuyos elementos pueden ordenarse.
Stack<T>	Pila genérica (también llamada pila LIFO).

La

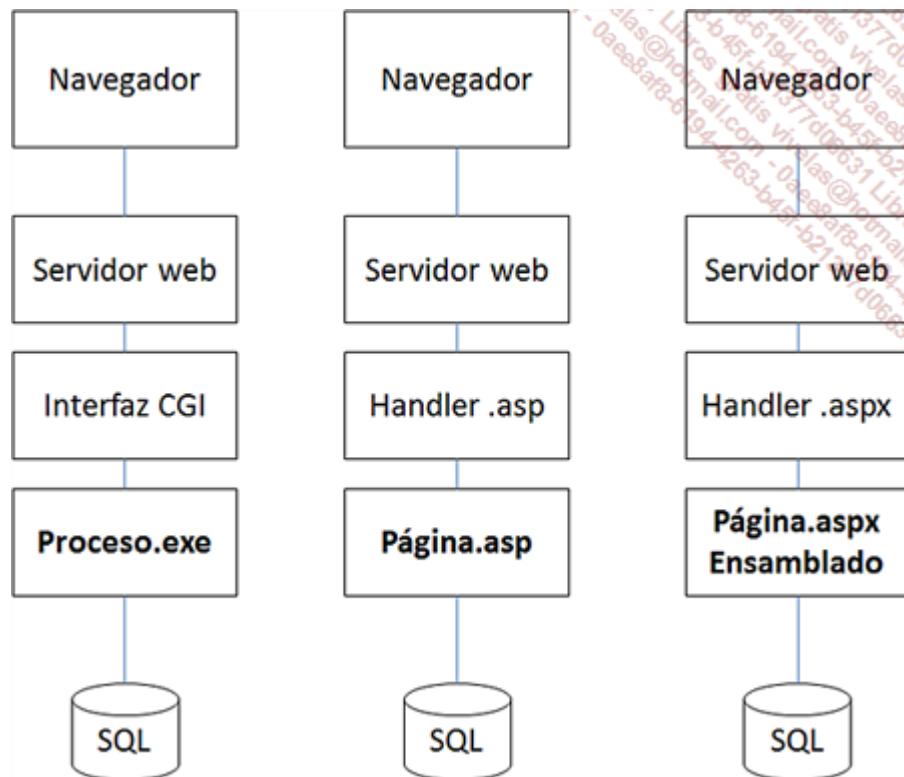
genéricidad no influye en el rendimiento de un programa. Pero la seguridad en el funcionamiento y la robustez se ver mejoradas evitando tipados erróneos desde la clase objeto.

# El modelo de compilación

## 1. Del CGI al modelo ASP.NET 1.X

Para comprender el modelo de compilación de ASP.NET, vamos a trazar la evolución de las aplicaciones web.

El protocolo HTTP ha ido evolucionando de la mano de las páginas HTML. Se ha impuesto con rapidez la idea de que el contenido de las páginas debía generarse bajo demanda, en especial para presentar datos provenientes de una base de datos SQL.



a. La interfaz CGI

La primera técnica disponible fue el CGI (*Common Gateway Interface*). Además de páginas HTML estáticas - archivos cuyo nombre incluye una extensión .html - el servidor web alberga programas ejecutables. Una configuración particular indica al servidor que dichos programas se deben ejecutar cuando se solicitan ciertas URL concretas. El programa ejecutable invocado por el servidor decodifica la petición HTTP realizando una lectura sobre el flujo de entrada estándar (stdin en lenguaje C) y analizando las variables de entorno. La respuesta HTTP se escribe, a continuación, sobre el flujo de salida estándar (stdout); el servidor inserta, en ocasiones, encabezados HTTP y se envía todo al navegador. Si bien fueron indispensables durante la creación de las aplicaciones web, las CGI presentan numerosas limitaciones; el lenguaje de programación empleado, C o, a menudo, PERL, no está realmente adaptado a la situación. Además, la interfaz CGI genera demasiadas implementaciones específicas, que hacen menos robusta la solución. Por último, las CGI no presentan un buen rendimiento.

Con el objetivo de ilustrar la complejidad del desarrollo de una CGI se muestra, a continuación, en lenguaje C, el código de un programa que devuelve la hora al usuario cuyo nombre se pasa en la cadena de petición (query string).

```
#include "stdafx.h"
```

```

#include <time.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    // calcular la hora
    char hora[128];
    _strtime_s( hora, 128 );

    // recuperar el nombre del usuario
    char*q=getenv("QUERY_STRING");
    char*nombre="";

    if(q!=NULL)
    {
        nombre=(char*)malloc(strlen(q));
        char*pn=strstr(q,"nombre=");
        if(pn>0)
            strcpy(nom,pn+strlen("nombre="));
        char*fin;
        if((fin=strstr(nombre,"&"))!=NULL)
            *fin=0;
    }

    // preparar la respuesta HTTP
    printf("Content-Type: text/html\n");
    printf("\n");

```

```

// respuesta HTML
printf("<html>");
printf("<body>");
printf("Hola %s<br>",nombre);
printf("Son las %s",hora);

```

```

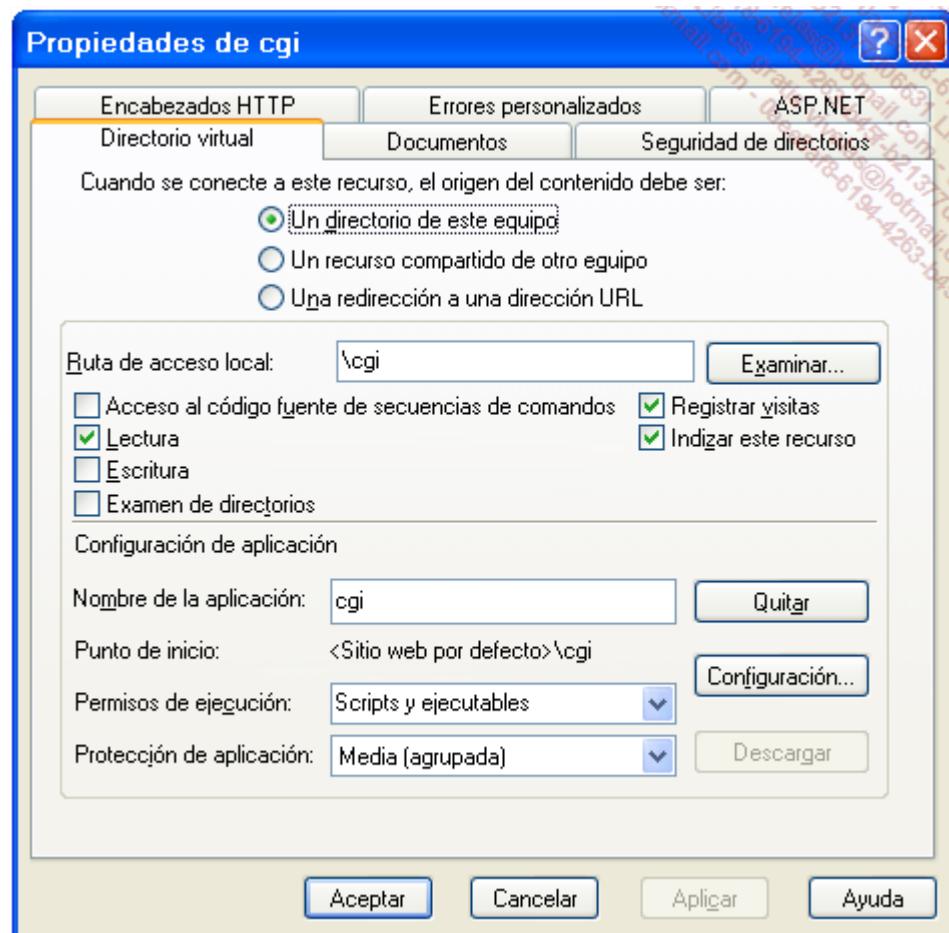
printf("</body>");

printf("</html>");

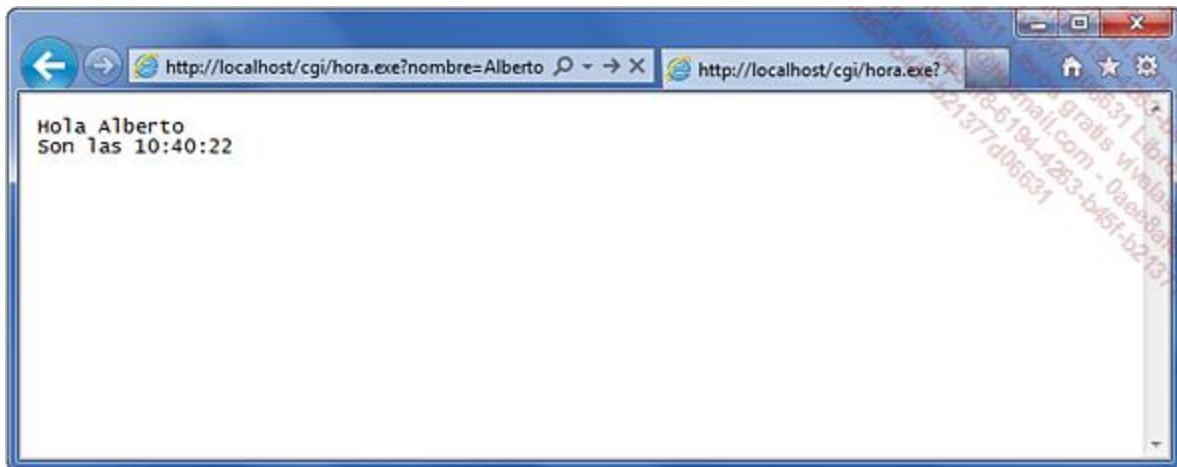
return 0;
}

```

El programa hora.exe debe ejecutarse en una carpeta configurada de manera especial para que el servidor web lo reconozca como módulo CGI. En IIS, cree una carpeta **cgi** en la carpeta **c:\inetpub\wwwroot**. Los modos de ejecución se definen en la ventana de propiedades de la carpeta virtual correspondiente, desde la consola de administración de IIS:



Sólo nos queda probar el programa utilizando la dirección que hace referencia al módulo hora.exe:



### b. Las páginas dinámicas ASP

Los fabricantes de software servidor reaccionaron a las limitaciones de las CGI proponiendo interfaces web (ISAPI de Microsoft, NSAPI de Netscape), e introduciendo páginas dinámicas PHP, JSP y ASP. La implementación de las tres tecnologías es muy similar (a excepción de las JSP, que utilizan un modelo ligeramente diferente). Estas tecnologías se basan en el uso de un lenguaje específico (PHP, Java, VB Script), de una API dedicada a las bases de datos y a la decodificación de consultas HTTP, y de un handler que indica al servidor web que se trata de páginas que deben ejecutarse y no descargarse. Dicho de otro modo, los servidores web reconocen una nueva extensión (asp en el caso de Microsoft) para desencadenar la ejecución de las páginas correspondientes.

A título comparativo, he aquí la versión ASP del código CGI hora.exe. El código es mucho más directo y su aplicación no requiere ninguna configuración, puesto que IIS ha sido concebido para la ejecución de estas páginas dinámicas:

```
<html>
  <body>
    Hola <%= Request("nombre") %><br>
    Son las <%= Now %>
  </body>
</html>
```

Tras la aparición de las páginas dinámicas, los escenarios divergen; Microsoft se mostrará el más innovador con el modelo ASP.NET. Se trata, principalmente, de producir secuencias HTML. Una página .aspx resuelve la dificultad que supone encontrar el compromiso entre HTML y programación, proponiendo un modelo de objetos adaptado. Todas las secuencias HTML están descritas en las páginas .aspx mediante tags específicos (técnica también disponible en JSP y PHP, pero sin un uso generalizado), mientras que el código de los eventos se compila utilizando una librería DLL.NET (ensamblado).

Si bien la base de ASP.NET se ha saneado e industrializado, todavía queda mucho camino por recorrer.

En primer lugar, los sitios web ASP.NET 1.X se basan en un ciclo de creación algo limitado; dado que cada código de página está compilado en una única DLL, la modificación de tan solo una página requiere que se detenga la aplicación completa para actualizar dicha DLL. Además, Visual

Studio 2002 y 2003 generan mucho código C#. La cantidad y la calidad del código generado pueden entorpecer el mantenimiento de las aplicaciones.

A diferencia de Sun, con su modelo Servlet/JSP/Java Bean, Microsoft no propone una arquitectura de tipo aplicación web. Si bien están dotados de interfaces gráficas muy reactivas, los sitios web ASP.NET 1.X carecen de flexibilidad o, dicho de otro modo, es difícil integrar los distintos servicios del sistema de información en una aplicación web ASP.NET 1.X.

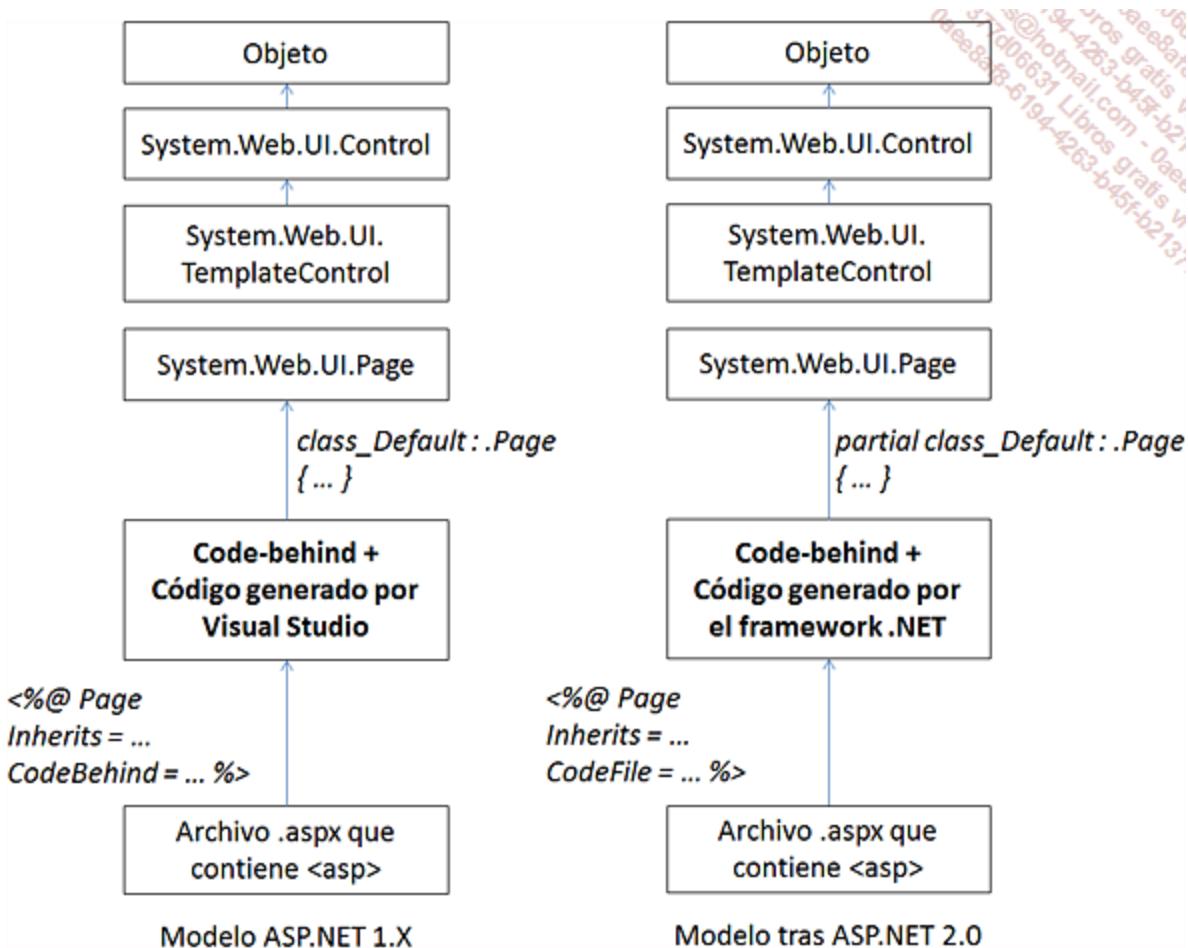
Por último, el objetivo de una página dinámica es producir una secuencia HTML que pueda ser consumida por un navegador. Si bien existía cierta expectación, muchas posibilidades no han podido explotarse con el modelo 1.X: producción dinámica de imágenes, de documentos PDF, servicios web distintos a SOAP... Ni el framework, ni Visual Studio, ni ASP.NET 1.X están realmente adaptados para soportar dichas tareas.

Microsoft no se ha contentado, pues, con adaptar ASP.NET al estado del arte. Ha aportado modificaciones que mejoran los puntos problemáticos. Estas modificaciones se describen en las siguientes secciones, y constituyen un nuevo modelo de compilación.

## 2. Clases parciales para las páginas

### a. Estructura de una página ASPX 4.5

El desarrollador que pase de la versión 1.X a la versión 4.5 constatará que Visual Studio genera, de forma neta, mucho menos código C#. Las páginas ASPX funcionan, todavía, basadas en dos archivos complementarios, pero las reglas de encapsulación han evolucionado.



Una página ASP.NET incluye dos clases: la primera se obtiene compilando el archivo .aspx que contiene segmentos de código C# pero también, sobre todo, etiquetas <asp>. De hecho, todas las etiquetas que poseen un atributo runat="server" constituyen un campo de esta clase. La segunda clase está escrita completamente en C#. En el modelo 1.X, es Visual Studio el encargado de generar el código en esta última clase y, en particular, de declarar campos con los mismos nombres que los controles que tienen un atributo runat="server". Estos campos se declaran con nivel de acceso protected, y se aplica la sobrecarga. El code-behind se basa en campos C# que son, de hecho, controles <asp>.

Tras el modelo 2.0, Visual Studio no genera código; es el framework el que realiza esta tarea. Como dicha generación es dinámica, los ingenieros de Microsoft han tenido que declarar la clase como parcial para soportar una inyección de código.

Consideremos la siguiente página Default.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true"  
CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >  
<body>
```

```
<form id="form1" runat="server">  
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>  
</form>  
</body>  
</html>
```

Esta página constituye una clase que hereda de la clase indicada mediante la directiva <%@ Page %>:

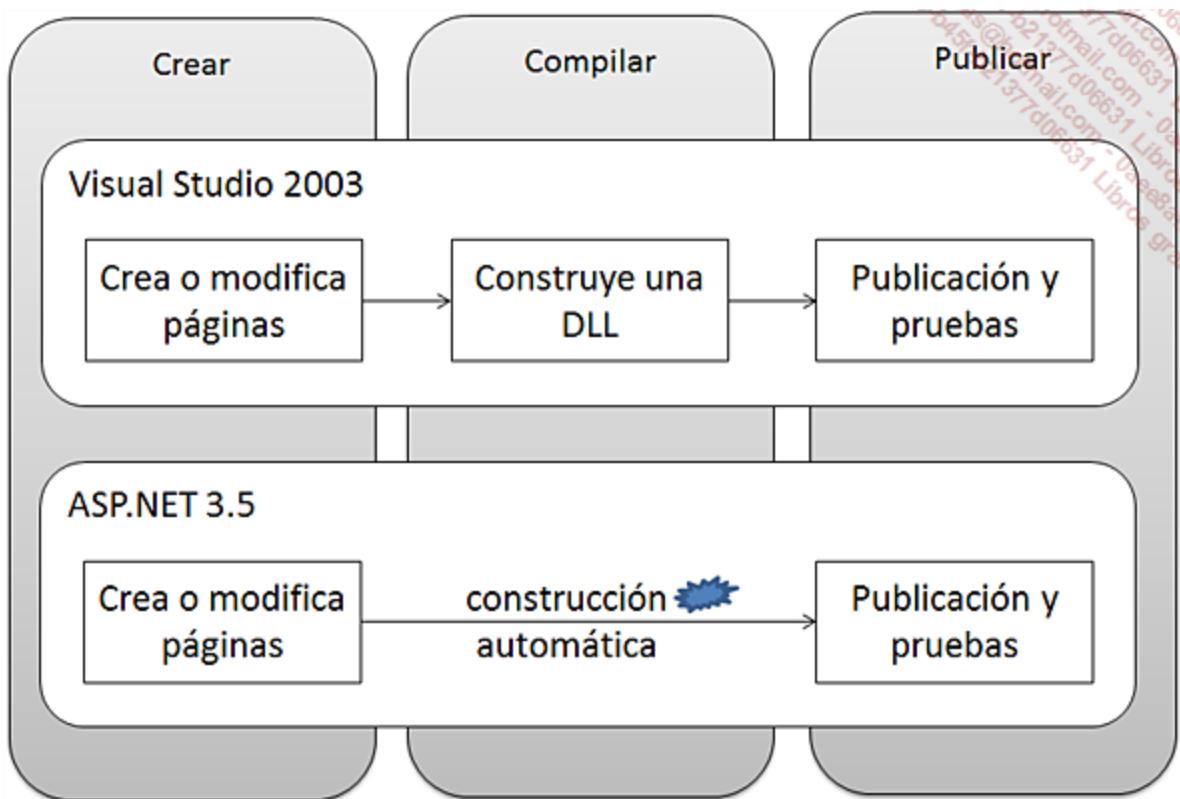
```
public partial class _Default : System.Web.UI.Page  
{  
    /*  
     *   el framework declara aquí los campos protected  
     *   para cada control de la página.  
     *   en este ejemplo,  
     *   protected TextBox TextBox1;  
    */  
    protected void Page_Load(object sender, EventArgs e)  
    {  
        this.TextBox1.Text = "Mensaje";  
    }  
}
```

Para el desarrollador, esta modificación del modelo es transparente, puesto que el modelo de programación no sufre cambios. En efecto, el framework declara los campos de la misma forma que Visual Studio. La evolución del modelo de compilación simplifica, en particular, la implementación de controles de usuario .ascx. Como Visual Studio no los declara, el programador debe realizar esta tarea él mismo. Ahora, es el framework el que utiliza la reflexión, y no olvida ciertos tipos de control.

### b. Modificaciones de una página ASPX 4.5

Podemos, ahora, preguntarnos acerca del sentido de esta evolución del modelo. Visual Studio, sin ser perfecto, lleva a cabo con éxito su tarea de generación de código. Recordemos que en la primera versión de ASP.NET todo el código que figura en un archivo .cs se compilaba bajo la forma de una DLL. La modificación de una sola página .aspx podía suponer cambios en el code-behind; era preciso volver a compilar el conjunto del proyecto y, a continuación, volver a compilar la clase HTML.

Con el nuevo modelo, el servidor de aplicaciones ASP.NET revisa las páginas del sitio. Si se modifica una página, el servidor decide si es necesario volver a compilarla y, en este caso, utiliza técnicas de compilación incremental para no bloquear el ensamblado completo del sitio.



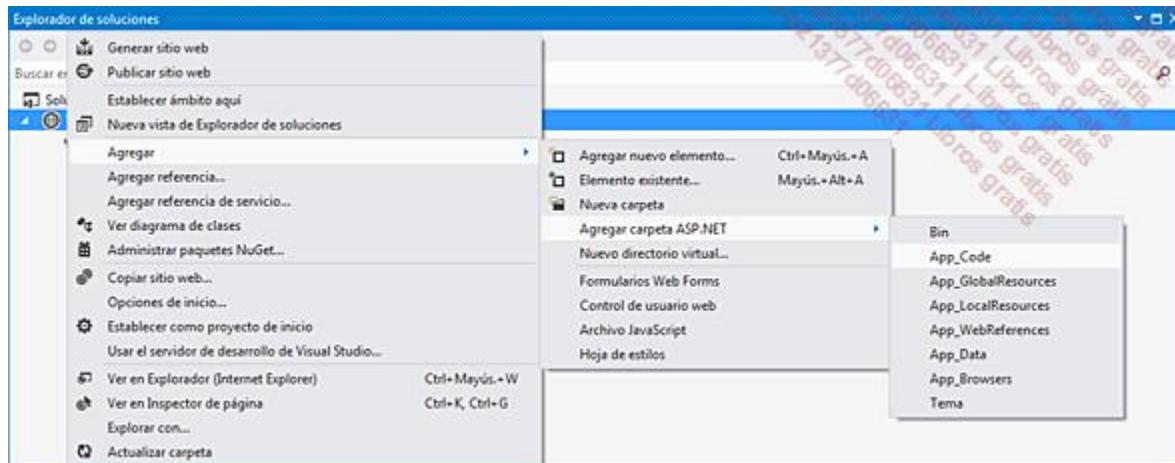
### 3. El código compartido en App\_Code

Una aplicación web no está compuesta solamente por páginas ASPX. Existen clases auxiliares (helper classes) que son necesarias para compartir comportamiento entre distintas clases. Es, así, posible factorizar controles de caché, crear objetos de negocio simples (al estilo de los Java Bean) que no tienen por qué tener la vocación de ser clases en una biblioteca de clases referenciada por el objeto.

Dado el nuevo modelo de compilación, sería una pena, incluso problemático, dejar de lado estas clases del usuario mientras que las páginas se recompilan automáticamente. Ésta es la razón de ser de la carpeta especial App\_Code.

El servidor de aplicaciones ASP.NET escruta así el contenido de dicha carpeta, que contiene el código fuente C#. Cuando se modifica un archivo, se recompila, y se integra en la DLL de salida.

Cuando Visual Studio crea un nuevo proyecto de sitio web, la carpeta especial App\_Code todavía no existe. El programador puede agregarla él mismo, desde el menú contextual que aparece en el explorador de soluciones. En caso contrario, Visual Studio sugerirá ubicar las clases recién creadas en App\_Code.



Cuando se crea o modifica una clase en esta carpeta, se vuelve disponible para utilizarse en otras clases o páginas. Si el programador guarda el archivo de código fuente y éste contiene errores, Visual Studio alimenta la lista de tareas para presentar las observaciones del compilador. Como el servidor de aplicaciones, el compilador y Visual Studio funcionan como tarea de fondo, la aparición de dichos mensajes puede tardar un poco en ocurrir, sobre todo con configuraciones de hardware algo justas. El programador puede, en tal caso, anticiparse y ejecutar la compilación del proyecto mediante el menú habitual **Compilar solución**.

The screenshot shows the Microsoft Visual Studio interface for a web application named "WebSite1(1)". The main window displays a file named "Persona.cs" containing C# code. The code includes standard using statements and defines a class "Persona" with a constructor and an overridden ToString() method. A tooltip "Esto no es código C#" appears over the word "esto". The status bar at the bottom indicates "Lin 18 Col 9 Car 9 INS".

**Lista de errores:**

Descripción	Archivo	Línea
✗ 1 Las directivas de preprocesador deben ser el primer carácter de una línea que no sea un espacio en blanco	Persona.cs	18
✗ 2 No se puede encontrar el tipo o el nombre de espacio de nombres 'esto' (falta una directiva using o una referencia de nombre)	Persona.cs	18

Este servicio de compilación automática implica una contrapartida: el programador debe publicar su código en el servidor de destino.

Preste atención, los entornos web tales como ASP.NET, PHP, J2EE, tienen en cuenta que la compilación de un código fuente es un mecanismo fiable para proteger dicho código fuente. Existen, a día de hoy, muchas herramientas de compilación. Para hacer ilegible un archivo compilado, es necesario utilizar una herramienta llamada ofuscador.

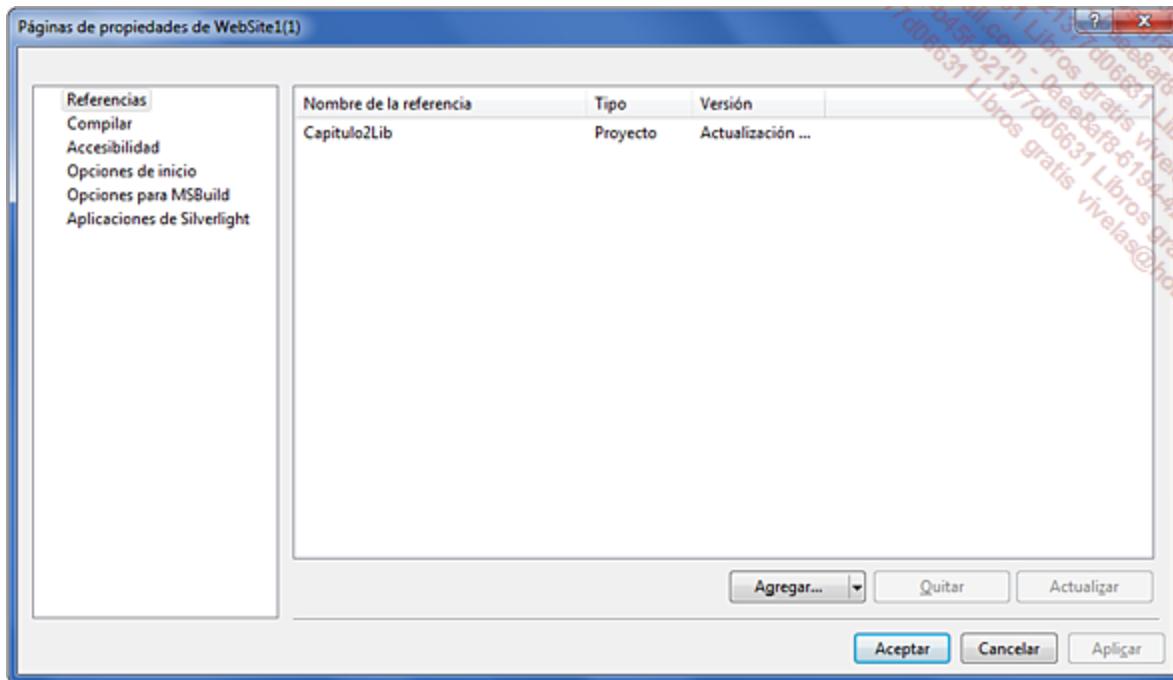
## 4. Los ensamblados referenciados

Hasta la fecha, ningún archivo .csproj se tiene en cuenta para compilar el proyecto. De hecho, el nuevo modelo de compilación de ASP.NET 4.5 ha reducido considerablemente el uso de dicho archivo. El proyecto se describe, ahora, en el archivo de solución. El archivo .csproj permanece tan sólo para la recompilación de la DLL que soporta el conjunto de clases del sitio web, si lo realiza el servidor de aplicaciones, y no Visual Studio, en cuyo caso necesita conocer las referencias a los ensamblados externos.

### a. Referencias dinámicas

El desarrollador puede agregar referencias a DLL siguiendo el procedimiento habitual, es decir, utilizando el menú contextual **Agregar referencia** que aparece en el explorador de soluciones. En el caso de una referencia privada, la DLL se copia en la carpeta bin de la aplicación web. El servidor de aplicaciones reconoce en esta carpeta especial la presencia de una DLL complementaria y la agrega a la lista de referencias que se pasan al compilador C#.

Para poner de manifiesto este mecanismo, es posible crear una DLL y copiarla, a continuación, manualmente en la carpeta bin. El menú contextual **Páginas de propiedades** del explorador de soluciones enumera dichas referencias dinámicas:



La referencia está, de este modo, disponible para el conjunto del código de la aplicación.

### b. Referencias explícitas en el Web.config

A diferencia de las referencias privadas, las referencias compartidas se inscriben en la Global Assembly Cache (GAC) y no tiene interés copiarlas en la carpeta local de la aplicación. Antes de confiar al archivo .csproj la responsabilidad de pasar la lista de referencias compartidas al servidor de aplicaciones, es el archivo de configuración Web.config el que incluye esta lista.

El archivo Web.config ya no lo crea automáticamente Visual Studio, es preciso, por lo general, esperar a la primera depuración para que se cree una configuración particular y ver aparecer dicho archivo en la lista de archivos de la aplicación.

Visual Studio incluye en el archivo de configuración Web.config las referencias compartidas de la aplicación ASP.NET. Esta técnica también puede emplearse para las referencias privadas, pero es inútil dada la supervisión de la carpeta bin que realiza el servidor de aplicaciones.

```
<compilation debug="false">
  <assemblies>
    <!-- aquí es posible declarar una referencia privada
    <add assembly="capítulo_lib"/>          -->

    <!-- referencia compartida (GAC) que debe
        precisar la versión, la cultura y la clave pública -->
    <add assembly="capítulo2_cs_gac_lib, Version=1.0.0.0, Culture=
      neutral, PublicKeyToken=393075C4B6832449"/>
```

```
</assemblies>  
</compilation>
```

## 5. La caché de construcción

El servidor de aplicaciones utiliza una carpeta de trabajo llamada caché de construcción. En esta carpeta se almacenan las distintas versiones de las clases parciales y de las DLL que representan el producto ejecutable de los sitios web ASP.NET 4.5.

La caché se sitúa en la carpeta C:\Windows\Microsoft.Net\Framework\v4.0\Temporary ASP.NET Files.

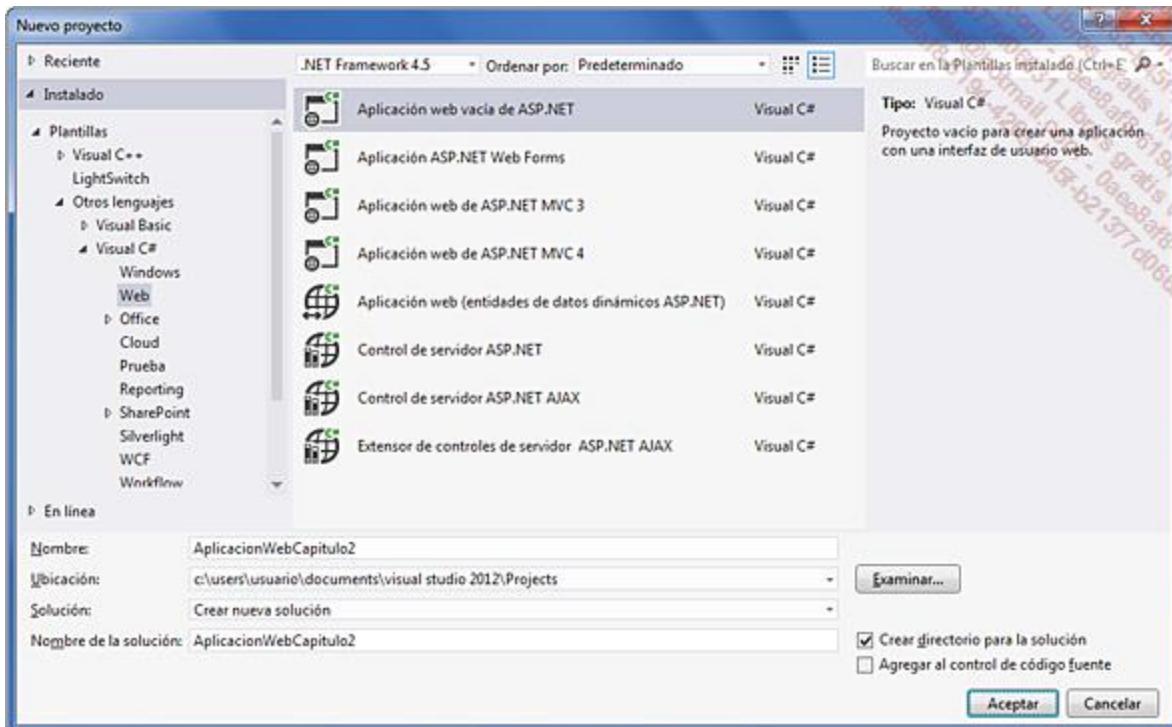
Cada aplicación ASP.NET se ordena en una subcarpeta cuya organización recuerda a la carpeta work del servidor de aplicaciones J2EE Tomcat. De hecho, las páginas ASP.NET y las páginas JSP tienen un funcionamiento muy parecido, y presentan un problema similar: la primera ejecución de una página provoca que se cree una clase, se compile, y se actualice un ensamblado completo. Este tiempo de preparación puede parecer excesivo en tiempo de desarrollo puesto que el proceso se repite cada vez que se guarda el archivo de código fuente.

La herramienta **aspnet\_compiler**, que se estudia en el capítulo que trata el desarrollo de aplicaciones ASP.NET, permite compilar de antemano el conjunto de elementos de un sitio. Esto facilita la instalación de la aplicación, el código fuente ya no tiene por qué distribuirse y se acelera la primera ejecución de un sitio que puede contener muchos elementos de programación.

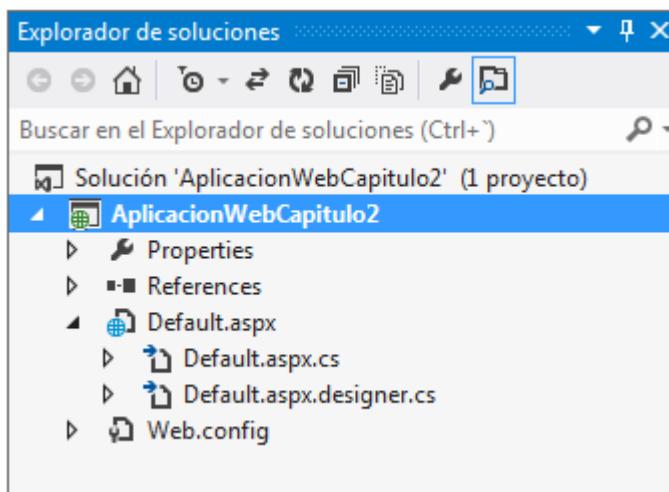
## 6. Las aplicaciones web de Visual Studio

Visual Studio proporciona, del mismo modo, otro modelo de construcción de aplicaciones que recuerda a las primeras versiones de ASP.NET y a las aplicaciones Winforms; se trata de aplicaciones web. Este modelo de aplicación no está enlazado con IIS durante el desarrollo sino con el servidor auxiliar que descubriremos un poco más adelante.

Para acceder a este tipo de aplicaciones, hay que utilizar el comando de Visual Studio **Archivo - Nuevo proyecto** y no **Archivo - Nuevo sitio Web**.



En este modo, Visual Studio juega el rol de servidor de aplicaciones y mantiene en un archivo separado (.designer.cs) el código subyacente.



Este modelo es muy interesante puesto que la compilación, exigida para acceder a la aplicación, incluye numerosas verificaciones que, de otro modo, se harían únicamente en tiempo de ejecución. Además, el acceso inicial a la aplicación en producción es mucho más rápido puesto que ésta ya se encuentra compilada. Otra ventaja es que el código fuente se elimina de la solución final.

Como inconveniente, este tipo de aplicaciones no se enlaza a un IIS en tiempo de desarrollo. Existen, por tanto, dos tiempos, uno sobre el servidor auxiliar y otro tras el despliegue de la versión compilada.

# El rol del servidor web

## 1. El servidor IIS

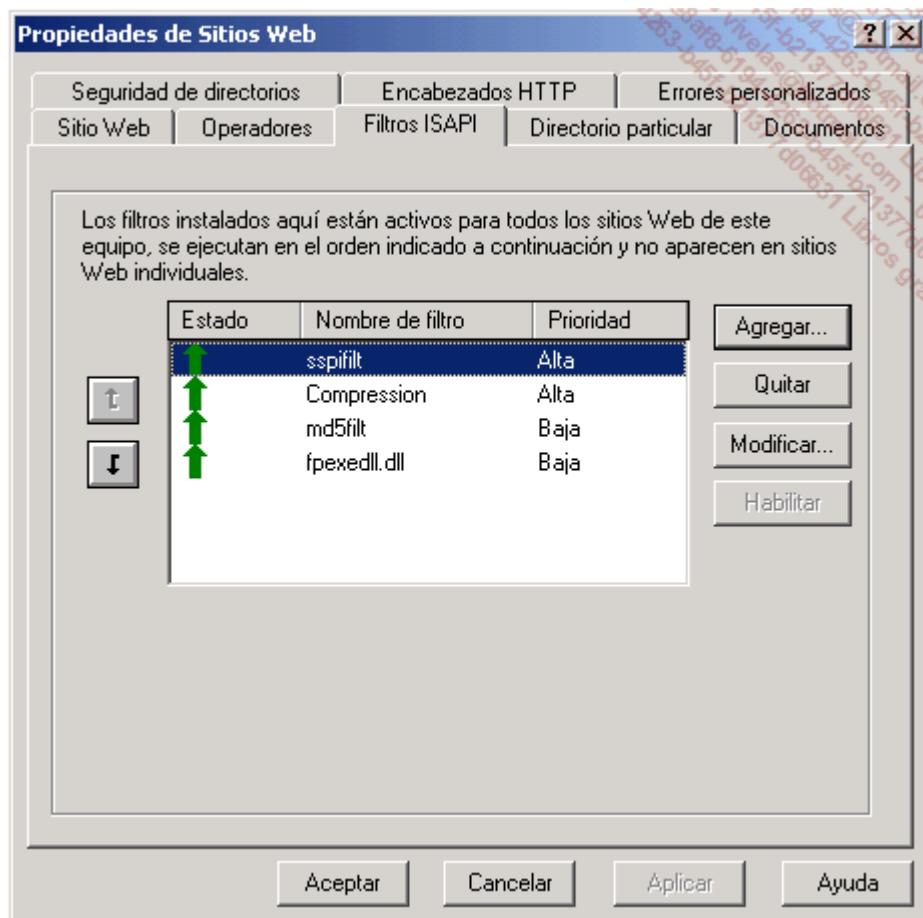
### a. El filtro ISAPI para ASP.NET

El servidor **Internet Information Services** (IIS) se ha integrado a la perfección con el conjunto del sistema operativo Windows. Da soporte a muchas tareas, aunque la implementación del protocolo HTTP sigue siendo su principal actividad.

Es el servidor web el que recibe las consultas http emitidas por el navegador. Estas últimas no distinguen la tecnología del servidor, tan sólo consideran los aspectos del lado cliente HTML, JavaScript, y HTTP.

Cuando se solicita al servidor una página con una extensión particular (.asp o .aspx, por ejemplo), éste delega su ejecución al servidor de aplicaciones correspondiente (ASP o ASP.NET) y, a continuación, envía el flujo HTML al navegador. Desde un punto de vista HTTP, el procedimiento es comparable a la interfaz CGI.

El servidor de aplicaciones ASP.NET se registra en IIS como filtro ISAPI. La lista de filtros ISAPI puede consultarse en la consola MMC de administración de IIS que figura en la carpeta **Herramientas administrativas** del **Panel de control** del equipo.

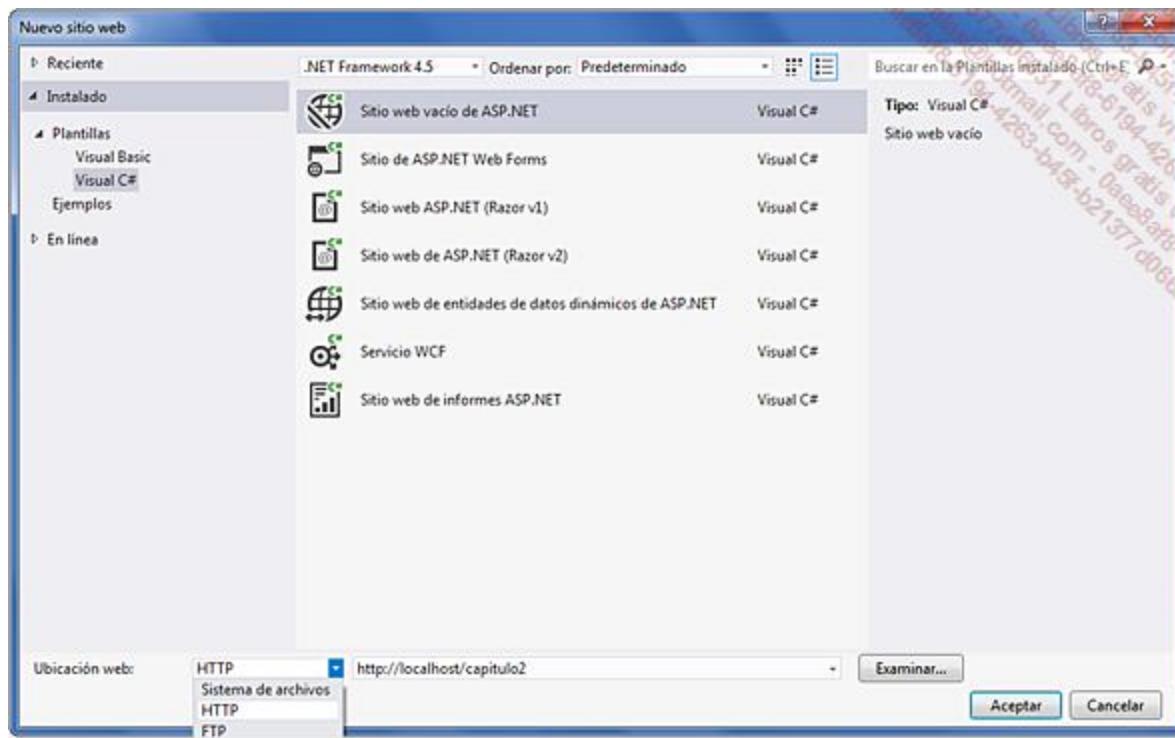


### b. Creación de un sitio web ASP.NET con IIS

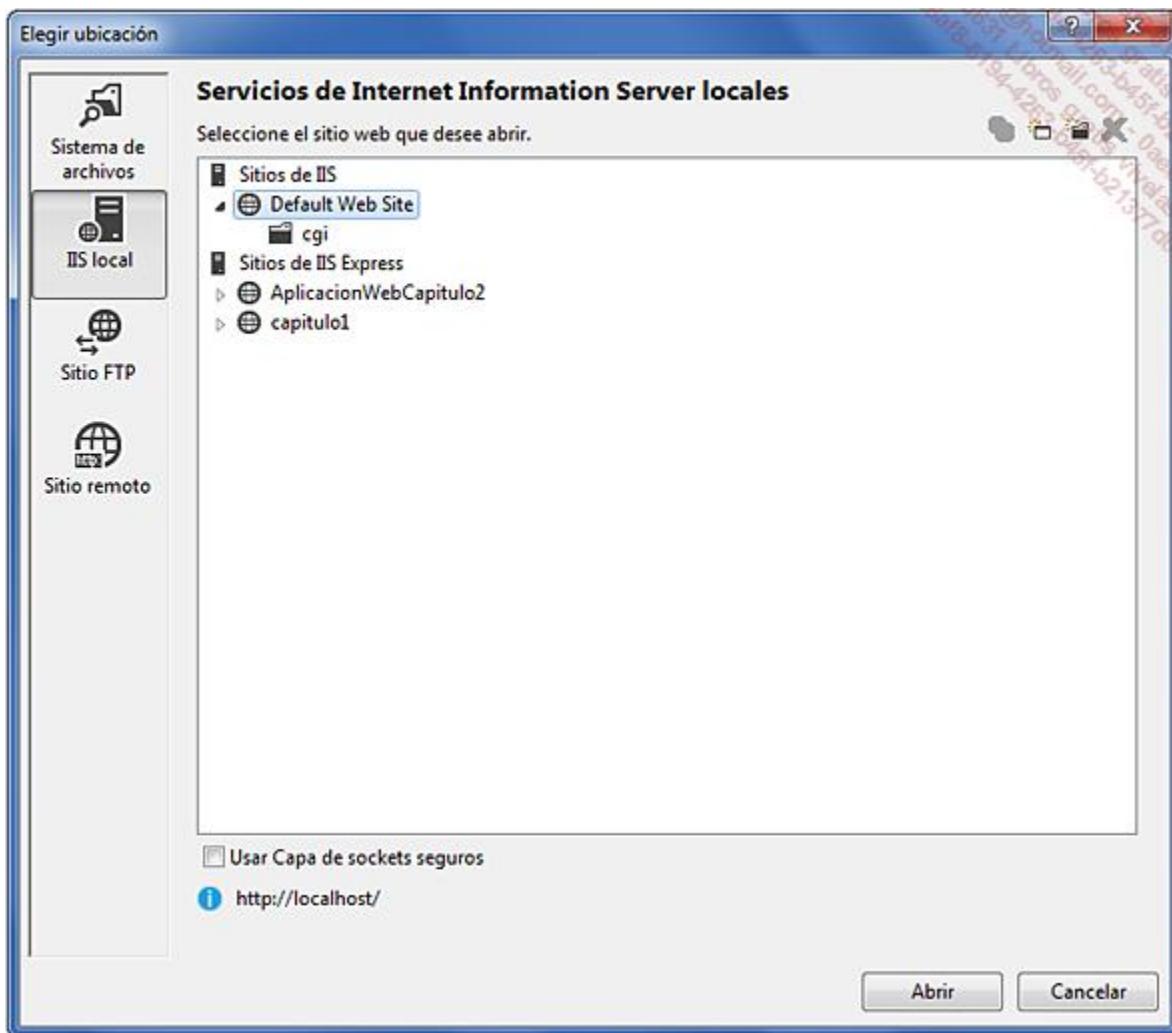
El servidor IIS posee, para cada instancia, una carpeta que se corresponde con la raíz de las URL (el primer símbolo / que figura tras el nombre del servidor en una URL completa). La instalación

por defecto prevé c:\inetpub\wwwroot como raíz, y en esta carpeta instala Visual Studio los sitios web.

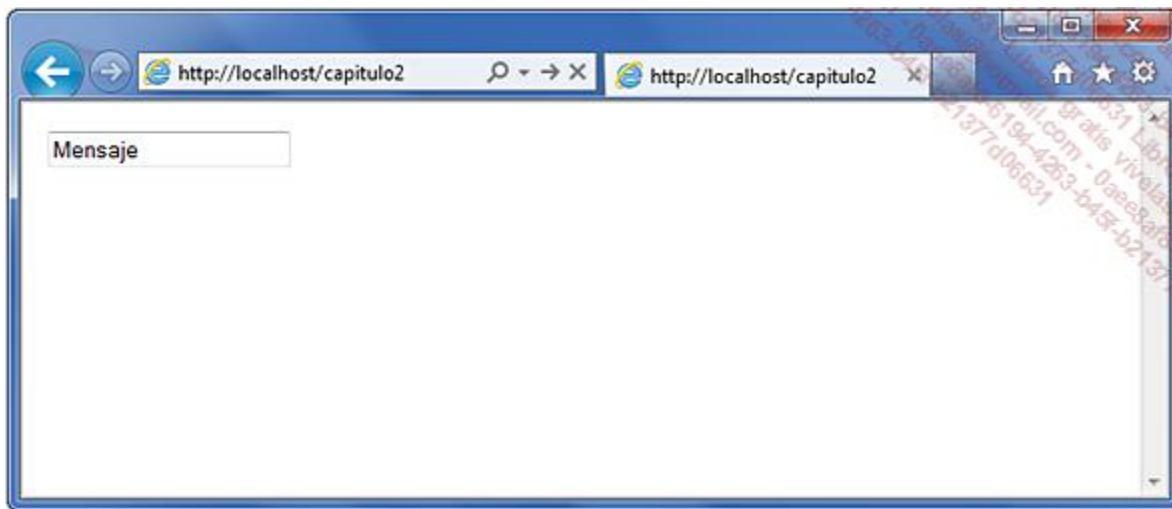
Supone una evolución importante tras Visual Studio 2003. Ahora es posible crear un sitio web ASP.NET eligiendo el modo de acceso: Archivo, HTTP o FTP. El modo HTTP se corresponde con una explotación mediante IIS.



El botón **Examinar** es útil para precisar la ubicación de la futura carpeta virtual. Permite seleccionar un servidor IIS local o remoto.



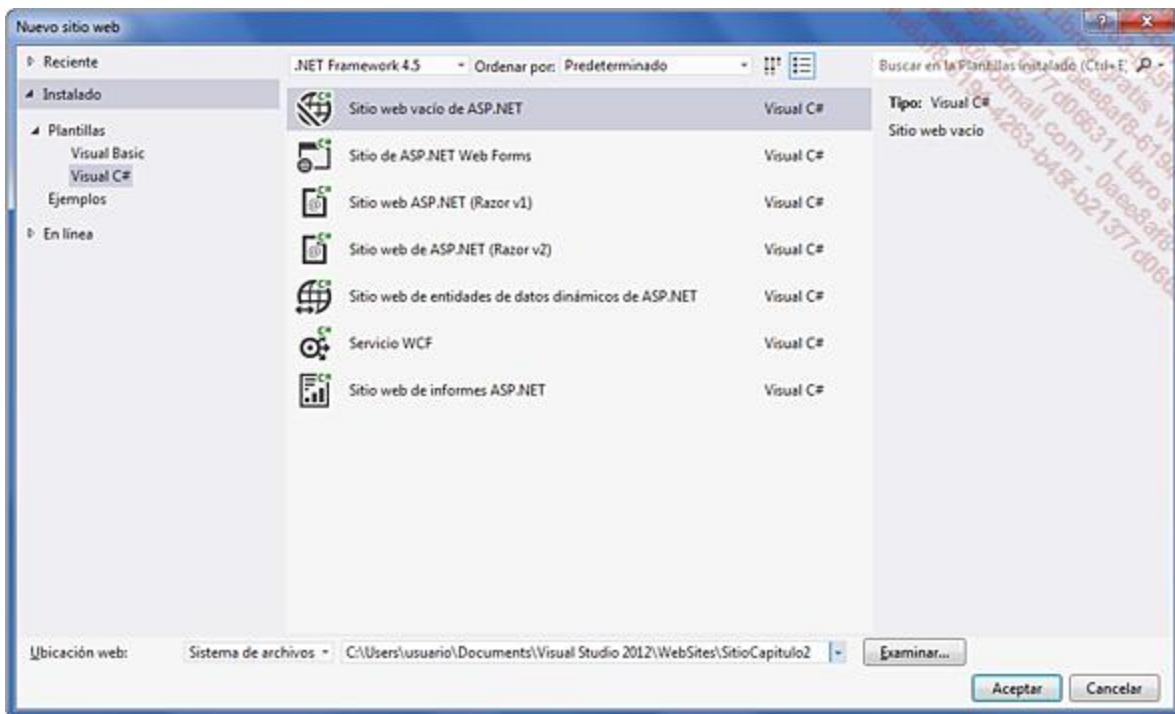
El uso del atajo de ejecución [F5] (depurar) o [Ctrl][F5] (sin depurar) provoca la apertura del navegador por defecto con una URL que se refiere a IIS, en principio **localhost** sobre el puerto web por defecto (**80**). El nombre del servidor y el puerto se corresponden, evidentemente, con información que figura en la URL de creación del sitio web.



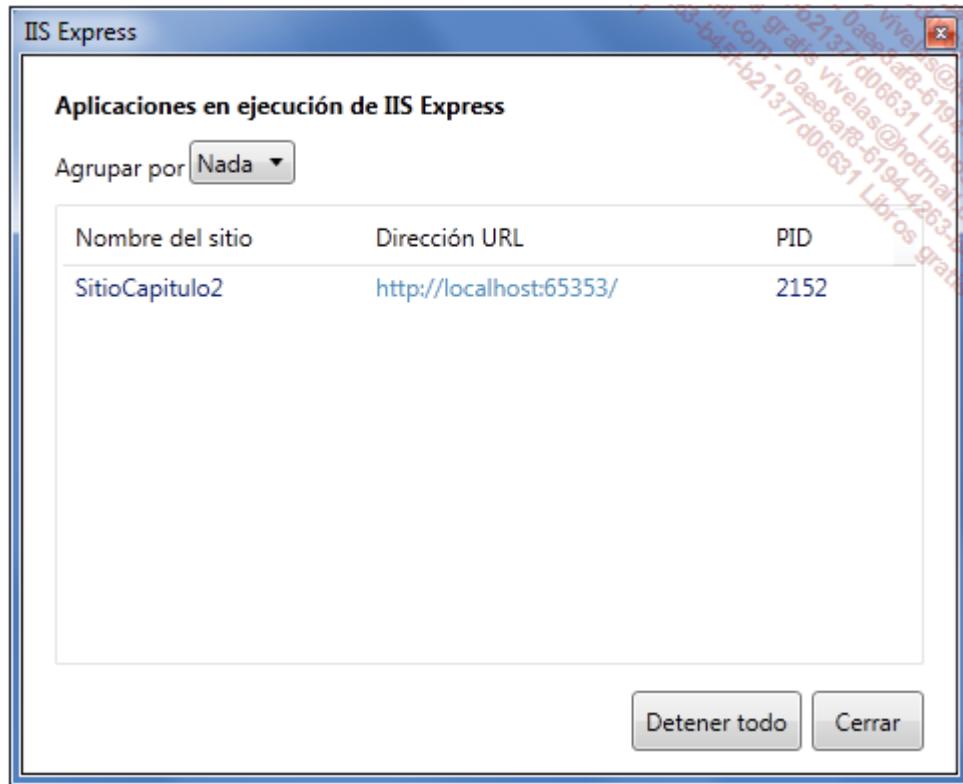
## 2. El servidor de desarrollo ASP.NET

Microsoft provee una alternativa a IIS que no está disponible con todas las distribuciones de Windows, en particular las ediciones Puesto de trabajo o Familiar. El servidor web personal (IIS Express) se basa en este servicio. Se trata de un servidor de desarrollo y de pruebas. No está, por tanto, previsto para realizar la explotación, a diferencia de IIS Standard, disponible con las ediciones servidor de Windows.

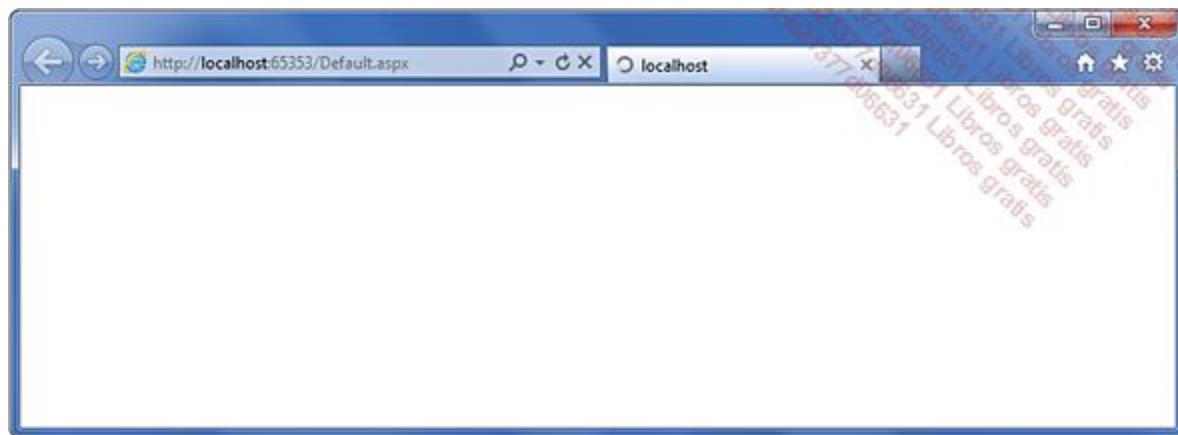
El servidor de desarrollo ASP.NET permite crear un sitio web ASP.NET en cualquier carpeta. Para crear un sitio explotado por dicho servidor hay que escoger la opción **Sistema de archivos** y, a continuación, indicar la carpeta de destino.



Cuando se ejecuta una página del sitio, mediante la tecla [F5], el servidor de desarrollo se enlaza a un puerto que cambia con cada sesión. El motivo de este funcionamiento es, sin duda, disuadir al usuario de utilizarlo como servidor de producción. Aparece un ícono en la barra de tareas para indicar el puerto y permitir detener el servidor.



El puerto lo envía Visual Studio al navegador por defecto, que puede pasar de una página a otra con la misma dirección de base (protocolo, servidor, puerto, carpeta web):



### 3. Otros servidores

La tecnología ASP.NET funciona también con otros servidores web. Gracias a la puesta a punto de un módulo, el servidor Apache (<http://www.apache.org>) es compatible.

La configuración Apache/ASP.NET puede probarse con éxito en el marco del proyecto de portabilidad de .NET en Linux (Mono). Existen guías de instalación disponibles en la dirección <http://www.mono-project.com>.

# El pipeline HTTP de IIS

## 1. Funcionamiento de IIS

Nos interesaremos, ahora, por el funcionamiento del servidor web IIS. El protocolo HTTP se basa en un mecanismo de peticiones y respuestas entre el cliente (el navegador) y el servidor. Este mecanismo no varía de una tecnología de páginas dinámicas a otra.

### a. Primeros pasos en HTTP con Telnet

Para estudiar los intercambios entre cliente y servidor es posible utilizar Telnet. Telnet es comparable a Minitel, pero se basa en el protocolo TCP/IP. Se trata, por tanto, de un terminal pasivo (teclado y pantalla) que es posible utilizar, afortunadamente, sobre cualquier puerto. Abriendo una sesión sobre el puerto 80 es posible hacerse pasar por un agente HTTP.

Telnet se ejecuta desde la línea de comandos y es aconsejable activar el echo local antes de realizar consultas HTTP. En efecto, el protocolo Telnet prevé que el servidor solicite o no al cliente mostrar por pantalla lo que se introduce por teclado. El protocolo Telnet también sabe interpretar la tecla de borrado de carácter. Aunque este no es el caso del protocolo HTTP que, por lo general, lo emplea un agente automático. En las manipulaciones siguientes se introduce un error en la consulta y habrá que retomarlo desde el principio. El echo local obliga al usuario a ver los caracteres conforme los introduce por el teclado, incluso si el servidor no se lo pide expresamente.

He aquí la sintaxis Telnet de activación del echo local en Windows XP y 7:

```
set localecho
```

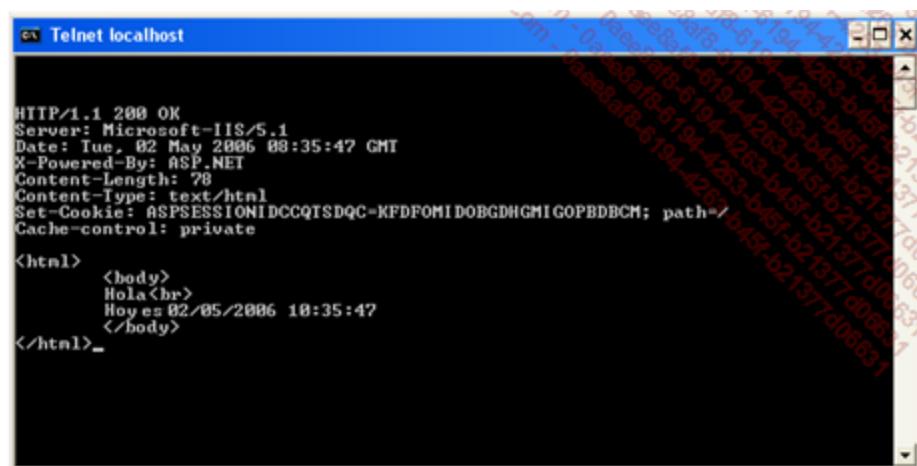
El comando que establece la conexión con el servidor en el puerto 80 es idéntico en ambas versiones.

```
open localhost 80
```

A continuación, hay que ejecutar una consulta HTTP sobre una página web existente. Para realizar una primera prueba, vamos a escoger una página cuyo contenido sea corto, con el objetivo de facilitar el análisis, hora.asp, por ejemplo.

```
GET /hora.asp HTTP/1.1
```

```
Host: yo
```

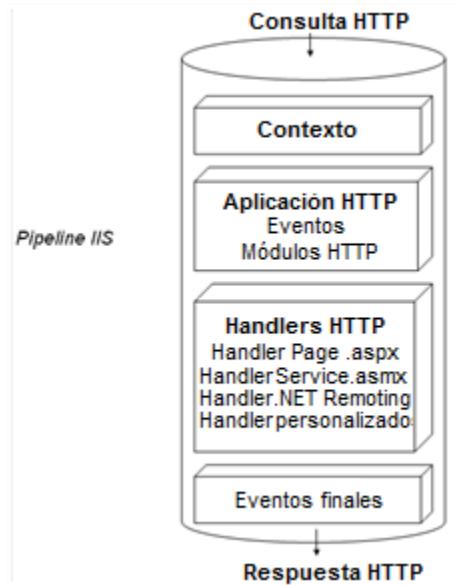


La respuesta del servidor está compuesta por un encabezado y un cuerpo separados por una línea vacía. El encabezado define, en particular, el código de error (200=OK), el formato de salida (text/html), el tamaño en bytes del cuerpo (70 bytes). El cuerpo es, de hecho, la secuencia HTML interpretada por el navegador según el formato especificado (text/html). De este modo el navegador ignora el formato de dicha secuencia hasta la recepción del encabezado Content-Type. La extensión indicada en la URL (.asp) no es una regla fiable, puesto que una página .asp también puede emitir un flujo PDF.

Tras el primer intercambio, las versiones del protocolo HTTP difieren: la versión 1.0 prevé que la conexión se interrumpa entre el navegador y el cliente, mientras que la versión actual, la 1.1, conserva la conexión abierta para poder realizar un nuevo intercambio. A este respecto, la versión 1.1 es mucho más eficaz para servir páginas HTML que referencian imágenes, pues el principal consumo de tiempo se da a la hora de establecer las distintas conexiones. Una página HTML que refiere a 3 imágenes requiere 4 conexiones con HTTP 1.0 y una única con HTTP 1.1.

## b. Detalle del procesamiento IIS

Tras estudiar el servidor IIS desde el punto de vista del plan de ejecución entrada (consulta) - salida (respuesta), vamos a describir el proceso de procesamiento de la consulta. El servidor web IIS pasa la consulta a una capa llamada pipeline. La consulta se deriva a las distintas unidades hasta formar la respuesta que se envía al navegador.



### El contexto HTTP

En primer lugar, la consulta HTTP, que puede estar formada por muchas líneas, en particular en el caso de POST HTTP, se deserializa. Esta operación se asemeja a la decodificación CGI de los distintos elementos que la constituyen: información acerca del cliente, operación solicitada, parámetros de la solicitud, formato de la solicitud... Esta información se clasifica, codifica y forma el contexto de la petición.

### La aplicación HTTP

Una aplicación HTTP coincide, más o menos, con una carpeta virtual. A cada aplicación le corresponde una instancia de la clase **HttpApplication** o una forma derivada. Esta clase crea una instancia de la clase **HttpContext** que expone la información de la petición y soporta los elementos de la respuesta en curso de ser formada.

La clase `HttpApplication` orquesta la progresión del procesamiento de una petición mediante eventos. Los programadores ASP y ASP.NET 1.X conocen algunos de estos eventos por haberlos visto en los archivos `Global.asa` y `Global.asax.cs`.

La petición la procesan, de este modo, en el marco de una aplicación, distintos módulos HTTP (autenticación, autorización...). Para intervenir sobre el procesamiento, el programador puede tomar control sobre los eventos (`Session_Start`, por ejemplo) o registrar nuevos módulos.

### Los gestores (handlers)

A cada extensión de recurso (URI) le corresponde un gestor HTTP. Las páginas `.aspx` las procesa un gestor específico, los servicios web `.asmx` los procesa otro, y así con los demás elementos.

El rol de un gestor consiste en procesar la consulta de manera efectiva, desde un punto de vista de aplicación. Son, por tanto, los gestores los encargados de responder a las consultas.

El servidor IIS prevé varias alternativas para extender dichos gestores. Se detallan a continuación.

## 2. La clase `HttpContext`

La clase `HttpContext` es accesible por todos los elementos de una aplicación ASP.NET. Expone una propiedad estática `Current` que devuelve una gran cantidad de información relativa a la petición en curso. Los objetos que componen `HttpContext` se utilizan habitualmente en páginas ASP.NET.

De este modo, el objeto `Application` es un alias de `HttpContext.Current.Application`.

He aquí los principales objetos que forman la clase `HttpContext`:

Application	HttpApplicationState	Objeto que conserva los datos compartidos por todos los usuarios.	Gracias a la
Application Instance	HttpApplication	Aplicación HTTP en curso.	
Cache	System.Web.Caching.Cache	Objeto que comparte los datos entre todos los usuarios con reglas de conservación.	
PreviousHandler	HttpHandler	Apunta al anterior gestor. Es útil para gestionar los post back cruzados (cross post back).	
Profile	HttpProfileBase	Perfil del usuario.	
Request	HttpRequest	Petición HTTP.	
Response	HttpResponse	Respuesta HTTP.	
Server	HttpServerUtility	El servidor Soporta los métodos Execute, Transfer, MapPath...	
Session	HttpSessionState	Objeto que conserva los datos propios de cada usuario.	

propiedad estática `Current` es, por tanto, posible explotar esta información desde una librería dinámica referenciada por el sitio web. A continuación, es necesario definir una referencia al ensamblado `System.Web`.

### 3. La clase **HttpApplication**

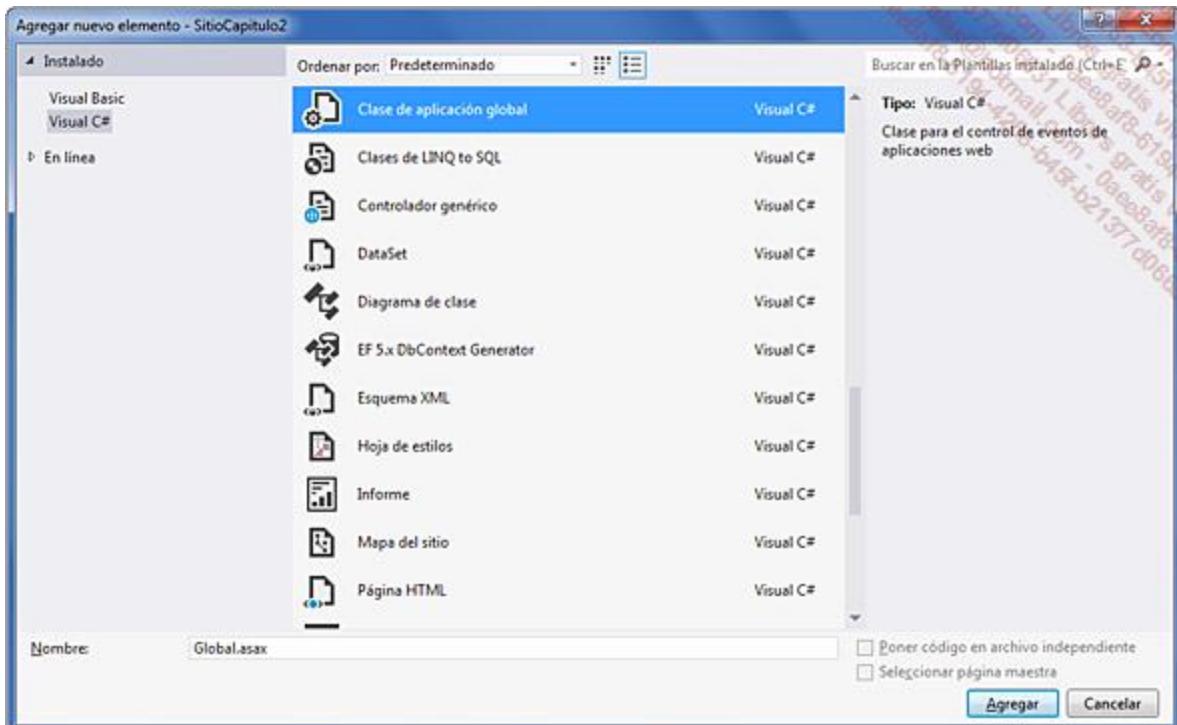
#### a. Ciclo de vida de la aplicación

La clase **HttpApplication** desencadena toda una serie de eventos en el transcurso del procesamiento de una consulta. Estos eventos se gestionan, por lo general, en el archivo **Global.asax**, donde se describe una versión derivada de **HttpApplication**.

BeginRequest	Primer evento del pipeline. Inicio del procesamiento.	b.
PreAuthenticateRequest AuthenticateRequest PostAuthenticateRequest	Muy útil para personalizar el enfoque de autenticación con los modos de Windows, Forms y Passport.	
PreAuthorizationRequest AuthorizationRequest PostAuthorizationRequest	Utilizado por el Administrador de roles de ASP.NET y por el módulo de autorización de acceso a los archivos y a las URL.	
PreResolveRequestCache ResolveRequestCache PostResolveRequestCache	Utilizado por el módulo de actualización de caché de HTML.	
PreMapRequestHandler PostMapRequestHandler	Nuevo evento destinado a ejercer una influencia sobre los gestores empleados para procesar la solicitud.	
PreAcquireRequestState AcquireRequestState PostAcquireRequestState	El estado (session) obtenido a partir de distintos canales (memoria, servidor, base de datos SQL...).	
PreRequestHandlerExecute Pagehandler PostRequestHandlerExecute	Tras el evento PreRequestHandlerExecute, se invoca el método ProcessRequest del gestor para procesar la consulta.	
PreReleaseRequestState ReleaseRequestState PostReleaseRequestState	El estado (session) se salvaguarda.	
PreUpdateRequestCache UpdateRequestCache PostUpdateRequestCache	Utilizado por el módulo de caché de salida para transmitir la secuencia de retorno del gestor hacia la caché del servidor web y la caché del navegador.	
EndRequest	Última posibilidad de intervenir sobre el flujo de salida antes de que deje de pertenecer a IIS.	
PreSendRequestHandlers PreSendRequestContent	Renderizado de los encabezados HTTP y, a continuación, del cuerpo HTTP.	

#### Agregar un Global.asax

El archivo **Global.asax** implementa una versión derivada de la clase **HttpApplication**. Recordemos que esta clase desencadena eventos que pueden gestionarse para intervenir en el procesamiento de la petición. Tras la versión 2005 de Visual Studio, este archivo ya no se crea de manera sistemática. Sólo aquellas aplicaciones que lo necesiten, podrán agregarlo desde el explorador de soluciones o desde el menú **Sitio Web**:



## La versión simplificada

La versión 2008 de este archivo es, claramente, menos rica que la anterior. Para empezar, el código figura en el Global.asax y no en un archivo code-behind Global.asax.cs. Además, sólo los eventos más comunes poseen gestores: Application\_Start, Application\_End, Application\_Error, Session\_Start, y Session\_End. Nos recuerda, un poco, al archivo de configuración de ASP Global.asa. El programador puede, a continuación, iniciar sin dificultad su aplicación o crear variables de sesión para cada usuario.

```
<%@ Application Language="C#" %>

<script runat="server">
void Application_Start(object sender, EventArgs e)
{
    // Código ejecutado tras el inicio de la aplicación:
    // primera consulta del primer usuario
}

void Application_End(object sender, EventArgs e)
{
    // Código ejecutado antes de que la aplicación termine
    // coincide, a menudo, con la muerte del proceso
}
```

```
// aspnet_wp.exe
}
```

```

void Application_Error(object sender, EventArgs e)
{
    // Código ejecutado cuando se produce un error no manejado
}

void Session_Start(object sender, EventArgs e)
{
    // Código ejecutado cuando inicia una sesión de usuario
}

void Session_End(object sender, EventArgs e)
{
    // Código ejecutado cuando finaliza una sesión de usuario
    // El evento se produce únicamente con la persistencia
    // InProc (véase Web.config)
}

</script>

```

### **La versión completa**

El código del archivo Global.asax puede enriquecerlo el servidor de aplicaciones para formar la clase derivada de `HttpApplication`. Este cambio de estrategia responde a la mayoría de desarrolladores de ASP.NET, que no necesitan el conjunto de eventos producidos por la clase `HttpApplication`.

Para poder procesar todos estos eventos, es necesario transformar el archivo Global.asax haciendo referencia a una clase creada en la carpeta App\_Code:

```
<%@ Application Language="C#"
inherits="AplicacionCapitulo2" %>
```

El extracto de código C# debe suprimirse o comentarse. La clase AplicacionCapitulo2 describirá los métodos agregando ciertos gestores de eventos:

```

public class AplicacionCapitulo2 : HttpApplication
{
    public AplicacionCapitulo2() : base()

    {
        this.BeginRequest += new EventHandler(AplicacionCapitulo2_
BeginRequest);
    }

```

```

void AplicacionCapitulo2_BeginRequest(object sender, EventArgs e)
{
    Response.Write("Comienzo del procesamiento de la petición <br>");
}

void Application_Start(object sender, EventArgs e)
{
    Application["nb_user"] = 0;
}

void Session_Start(object sender, EventArgs e)
{
    int nb_user = (int)Application["nb_user"];
    nb_user++;
    Application["nb_user"] = nb_user;
}

void Session_End(object sender, EventArgs e)
{
    int nb_user = (int)Application["nb_user"];
    nb_user--;
    Application["nb_user"] = nb_user;
}

```

A continuación, una página ASPX puede trabajar de distintas maneras con la clase `HttpApplication`:

```

public partial class visitas : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // utilizar el objeto Application para contar
        // el número de visitantes
        int nb_user = (int)Application["nb_user"];
        Label1.Text = string.Format("Número de visitas: {0}", nb_user);

        // trabajar con la clase derivada de HttpApplication
    }
}

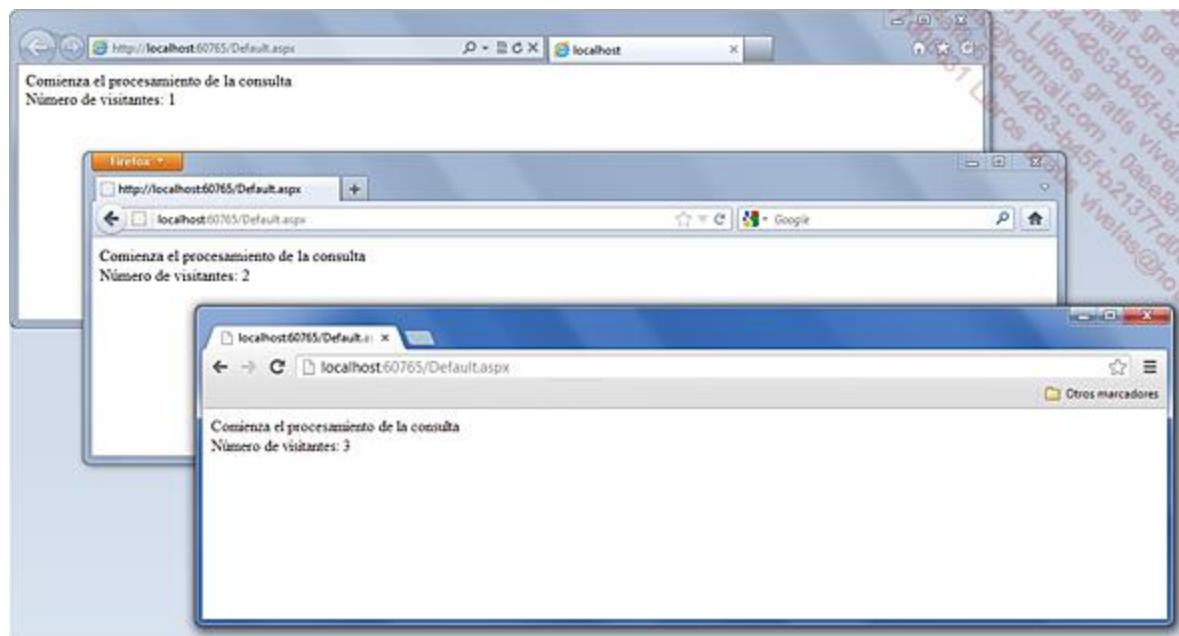
```

```

AplicacionCapitulo2 app =
    HttpContext.Current.ApplicationInstance as AplicacionCapitulo2;
}

}

```



### c. Crear un módulo HTTP

Un módulo HTTP es comparable a una clase derivada de `HttpApplication` compartida entre varias aplicaciones. Si bien es posible ubicar el código del `Global.asax` en una DLL compartida, esta opción presenta varios problemas que hacen del uso de un módulo una opción mucho más recomendable.

Técnicamente, un módulo es una clase que implementa la interfaz `IHttpModule` y declarada en el `Web.config`. El módulo se creará en una DLL que referencia al ensamblado `System.Web`.

La interfaz `IHttpModule` define dos métodos: `Init` y `Dispose`. El método `Init` sirve para registrar los gestores para los eventos de la aplicación HTTP. El método `Dispose` se utiliza cuando se detiene la aplicación.

```

public class MarcadorModulo : IHttpModule
{

```

```

    public void Init(HttpContext context)
    {
    }

    public void Dispose()
    {
    }
}

```

He aquí el código de un módulo que agrega una entrada en el Query String (la parte que sigue al ? en una petición HTTP).

```
public class MarcadorModulo: IHttpModule
{
    private HttpApplication app;

    public void Dispose()
    {
    }

    public void Init(HttpContext context)
    {
        app = context; // memoriza la referencia del contexto
        context.BeginRequest += new EventHandler(context_BeginRequest);
    }

    void context_BeginRequest(object sender, EventArgs e)
    {
        string marcador = "marca=un+mensaje";

        // modifica el Query String si existe
        string qs = app.Context.Request.QueryString.ToString();
        if (qs != null && qs != "")
            qs += "&" + marcador;
        else
            qs = marcador;
    }
}
```

```
// redirección
app.Context.RewritePath( app.Context.Request.FilePath,
    app.Context.Request.PathInfo, qs);
}
```

El módulo debe registrarse en el archivo Web.config mediante el elemento <HttpModule> ubicado en la sección <system.web>:

```
<httpModules>
    <add name="MarcadorModulo"
```

```
    type="capitulo2_cs_module. MarcadorModulo, capitulo2_cs_module"/>
</httpModules>
```

La sintaxis del atributo type es un clásico del género: el tipo cualificado completo (espacio de nombres de la clase) seguido del ensamblado referenciado por el proyecto de la aplicación web.

Para probar los efectos del módulo, una página ASPX sencilla muestra la petición y su cadena Query String:

```
public partial class test_module : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = "Petición: "+Request.Path+
                      "<br>QueryString : "+Request.QueryString;
    }
}
```

Tan sólo queda probar:



## 4. Los controladores (handlers) HTTP

Los controladores HTTP tienen la responsabilidad de responder, en términos de aplicación, a las peticiones enviadas por los navegadores.

En lo que respecta a las páginas ASPX, se trata de una instancia de la clase `System.Web.Page` (o de una clase derivada) que realiza el trabajo de renderizado HTML. Esta clase la solicita el controlador `System.Web.UI.PageHandlerFactory` que responde a la URL que incluye la extensión `.aspx`.

Una URI (*Uniform Resource Identifier*) es un fragmento de URL (*Uniform Resource Locator*).

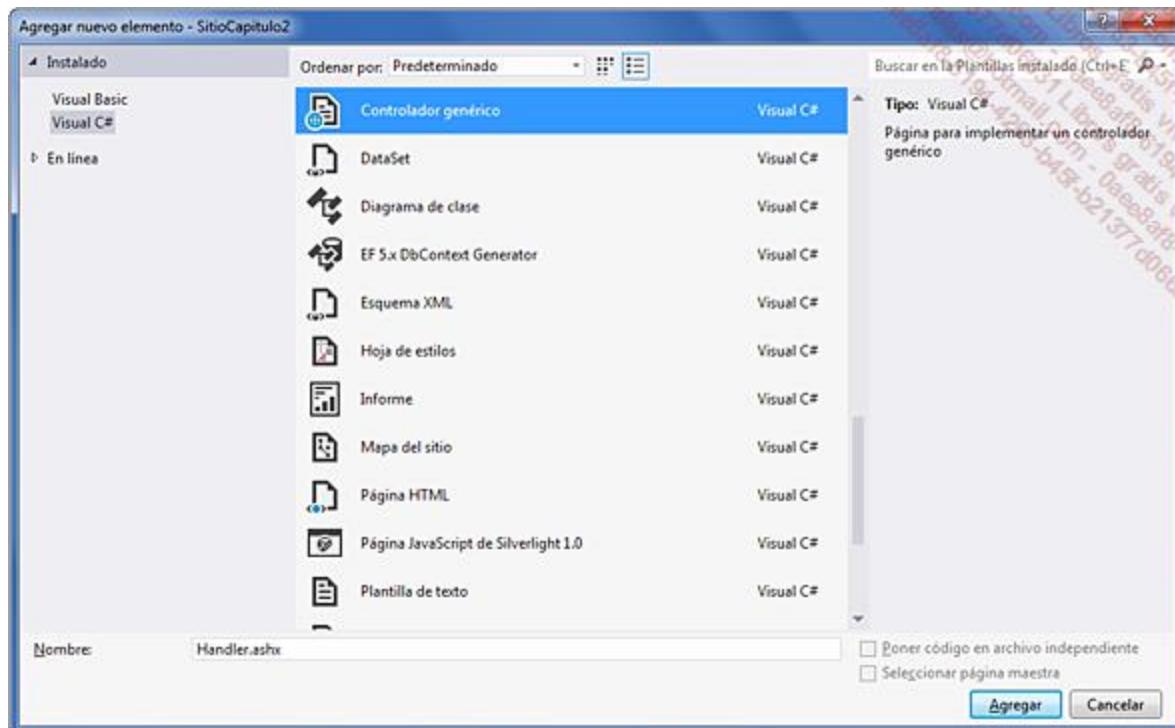
El servidor de aplicaciones .NET ya proporciona varios controladores adaptados a distintas situaciones y responden a varios tipos de URL:

System.Web.Handlers.TraceHandler	trace.axd	Visor de trazas.
System.Web.Handlers.WebAdminHandler	WebAdmin.axd	Interfaz de administración web para el sitio en curso.
System.Web.Handlers.AssemblyResourceLoader	WebResource.axd	Extrae los recursos de un ensamblado.
System.Web.Handlers.PrecompHandler	precompile.axd	Compila todas las páginas y el código de una aplicación web.
System.Web.Handlers.WebPartExportHandler	WebPartExport.axd	Utilizado por los Web Parts.
System.Web.UI.PageHandlerFactory	*.aspx	Redirige todas las peticiones destinadas a páginas ASPX hacia el código adecuado.
System.Web.UI.SimpleHandlerFactory	*.ashx	Proporciona una infraestructura simplificada para crear controladores personalizados.
System.Web.StaticFileHandler	variable	Útil para securizar el acceso a los formatos de archivo no dinámicos (.html, .jpeg...).
System.Web.Services.Protocols.WebServiceHandlerFactory	*.asmx	Procesa las peticiones de prueba o explotación de servicios web SOAP.
System.runtime.Remoting.Channels.Http.HttpRemotingHandlerFactory	*.rem *.soap	Controlador anidado en la jerarquía de componentes .NET Remoting por IIS.
System.Web.HttpForbiddenHandler	*.cs *.vbWeb.Config...	Bloquea el acceso (código HTTP 403) a algunos tipos de archivo.

## a. Crear un handler ASHX

Tal y como hemos indicado, la tecnología ASP.NET no está limitada únicamente a páginas dinámicas ASPX. A menudo es útil para responder a consultas HTTP con otros formatos distintos a HTML. Los desarrolladores J2EE utilizan, con este fin, los servlets, especie de CGI escritos en Java. En el universo Microsoft, el servlet se implementa mediante un controlador ASHX.

Los controladores .ashx existen antes que .NET; se trataban, en su origen, de una simple extensión de IIS. Desde la versión 2005 de Visual Studio es posible crear un controlador desde la ventana de diálogo **Agregar nuevo elemento**:



Técnicamente, el controlador es una clase que implementa la interfaz `IHttpHandler`. Esta interfaz impone dos métodos, `IsReusable` y `ProcessRequest`.

Nuestro controlador de ejemplo va a crear una imagen JPEG que representa un diagrama sectorial. Las características del diagrama se especifican mediante Query String.

```
<%@ WebHandler Language="C#" Class="Diagrama" %>

using System;
using System.Web;
using System.Drawing;
using System.Drawing.Imaging;

public class Diagrama : IHttpHandler
{
    public void ProcessRequest (HttpContext context) {
```

```

// preparar una imagen

int longitud = 400, altura = 400;
Bitmap bmp = new Bitmap(longitud, altura);
Graphics g = Graphics.FromImage(bmp);

// diseñar un diagrama sectorial

g.FillRectangle(Brushes.White, new Rectangle(0,0,longitud,
altura));
Pen p=new Pen(Color.Red,5);
float angulo=200;
try
{
    angulo=float.Parse(context.Request["angulo"]);
}
catch{};

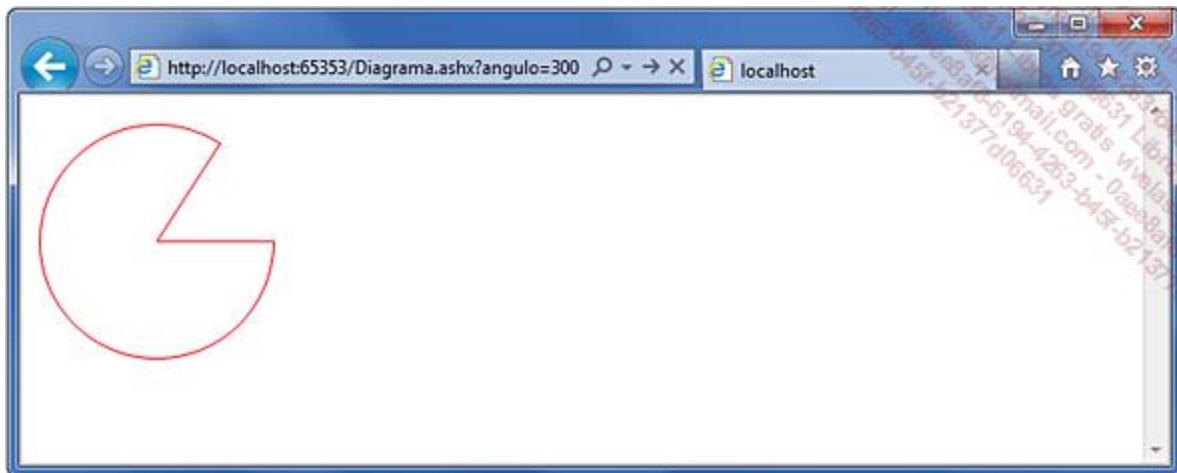
g.DrawPie(p,(float) 50,
           (float) 50,
           (float) (longitud-100),
           (float) (altura-100),
           (float) 0,angulo);

// devuelve la imagen por el flujo de salida
context.Response.ContentType = "image/jpeg";
context.Response.ClearContent();
bmp.Save(context.Response.OutputStream, ImageFormat.Jpeg);
context.Response.Flush();
}

public bool IsReusable {
    get {
        return true;
    }
}
}

```

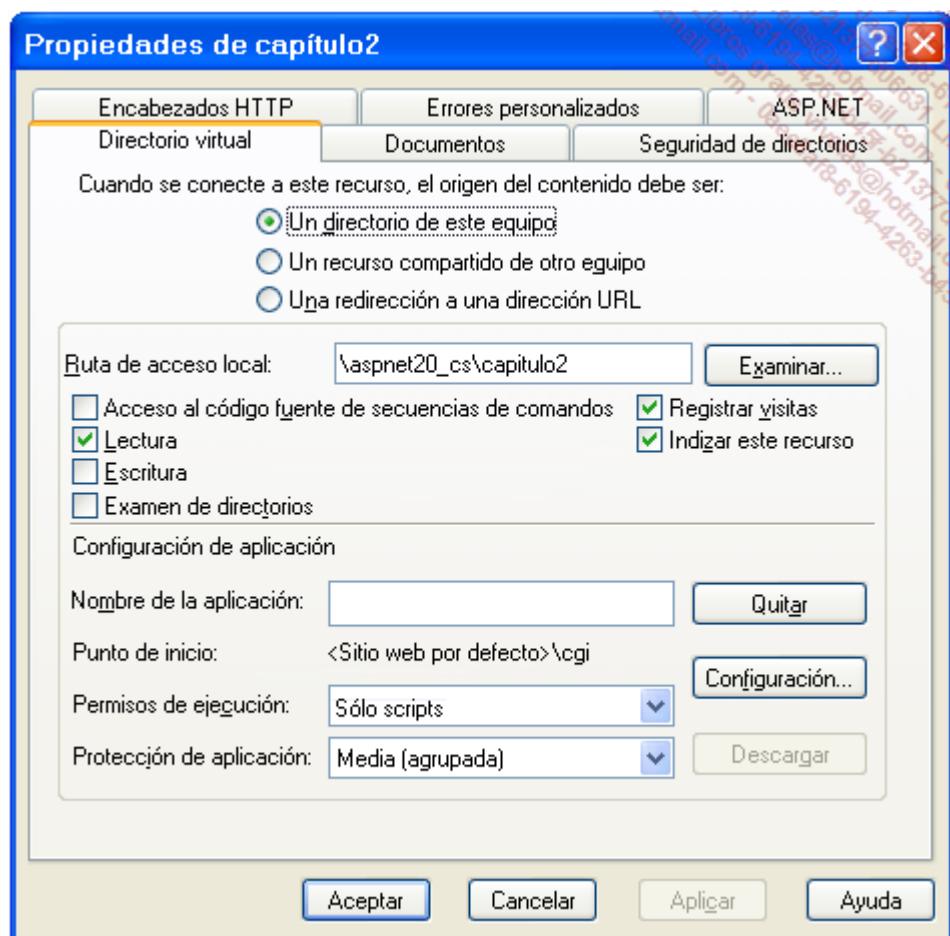
Es posible probar el controlador mediante una URL que haga intervenir a la extensión .ashx:



### b. Crear una extensión personalizada

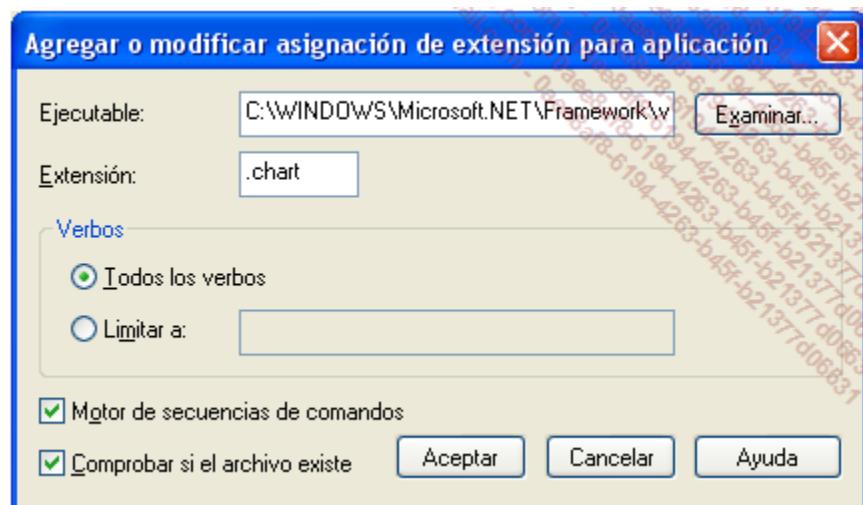
Es posible remplazar la extensión genérica .ashx por una extensión personalizada que se declara en la consola IIS. En este caso, el código del controlador no se ubica en un archivo .ashx sino en una clase que implementa IHttpHandler.

La asociación se realiza editando las propiedades de la carpeta virtual desde la consola de administración de IIS. Haga clic en el botón **Configuración** y, a continuación, especifique la DLL que contiene el controlador.



Para ilustrar este procedimiento, vamos a crear un nuevo tipo de archivo con la extensión .chart.

La ventana de diálogo **Agregar** asocia esta extensión con el filtro ISAPI ASP.NET. Éste se encuentra en la carpeta de instalación del framework: C:\WINDOWS\Microsoft.NET\Framework\v4\aspnet\_isapi.dll. Esta asociación controlará las peticiones de archivos .chart hacia la aplicación ASP.NET. De este modo, todas las peticiones que se refieran a la extensión .chart se enviarán a ASP.NET, haciendo posible su procesamiento mediante un controlador personalizado.



A continuación, quedaría el procesamiento de la extensión a través de la aplicación C#. La clase DiagramaHandler, que recupera la lógica del controlador ASHX, se crea dentro de App\_Code. En lugar de decodificar el ángulo mediante Query String, el controlador lee la información en el archivo .chart. El resto de la clase no cambia.

```
try
{
    string PhysicalPath = context.Server.MapPath(context.Request.Path);
    using (FileStream fs = new FileStream(PhysicalPath, FileMode.Open))
    {
        StreamReader reader = new StreamReader(fs);
        string contenido = reader.ReadToEnd();
        angle = float.Parse(contenido.Substring("angulo=".Length));
    }
}
catch { }
```

El archivo Web.config registra la siguiente regla: las consultas que incluyan .chart en su extensión se procesarán a través de la clase DiagramaHandler. El atributo verb se corresponde con un método HTTP, GET, POST, HEAD, TRACE, DEBUG o \* para todos los métodos:

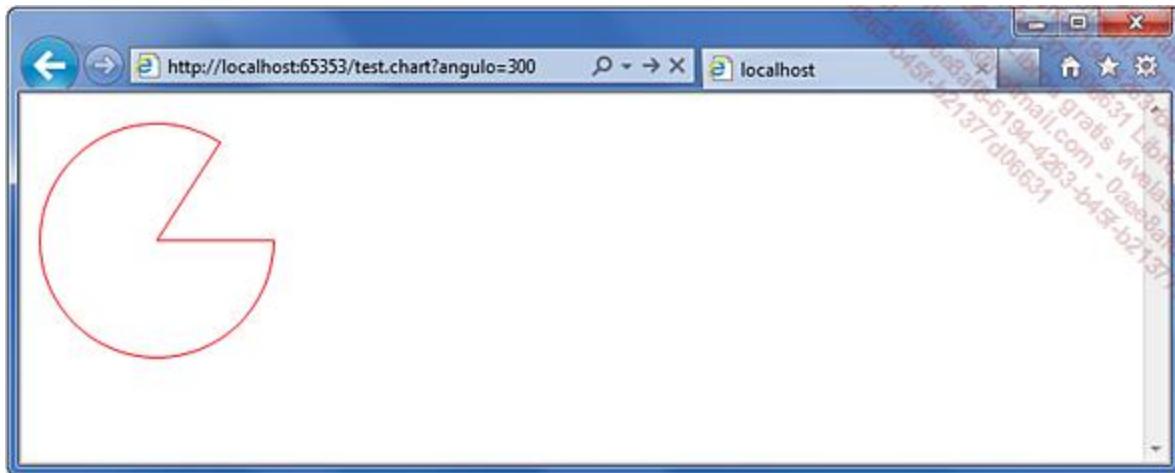
```
<httpHandlers>
    <add path="*.chart"
        type="Manejadores.DiagramaHandler"
        verb="*"/>
```

```
</httpHandlers>
```

Tan sólo quedaría crear un archivo dibujo1.chart:

```
angulo=300
```

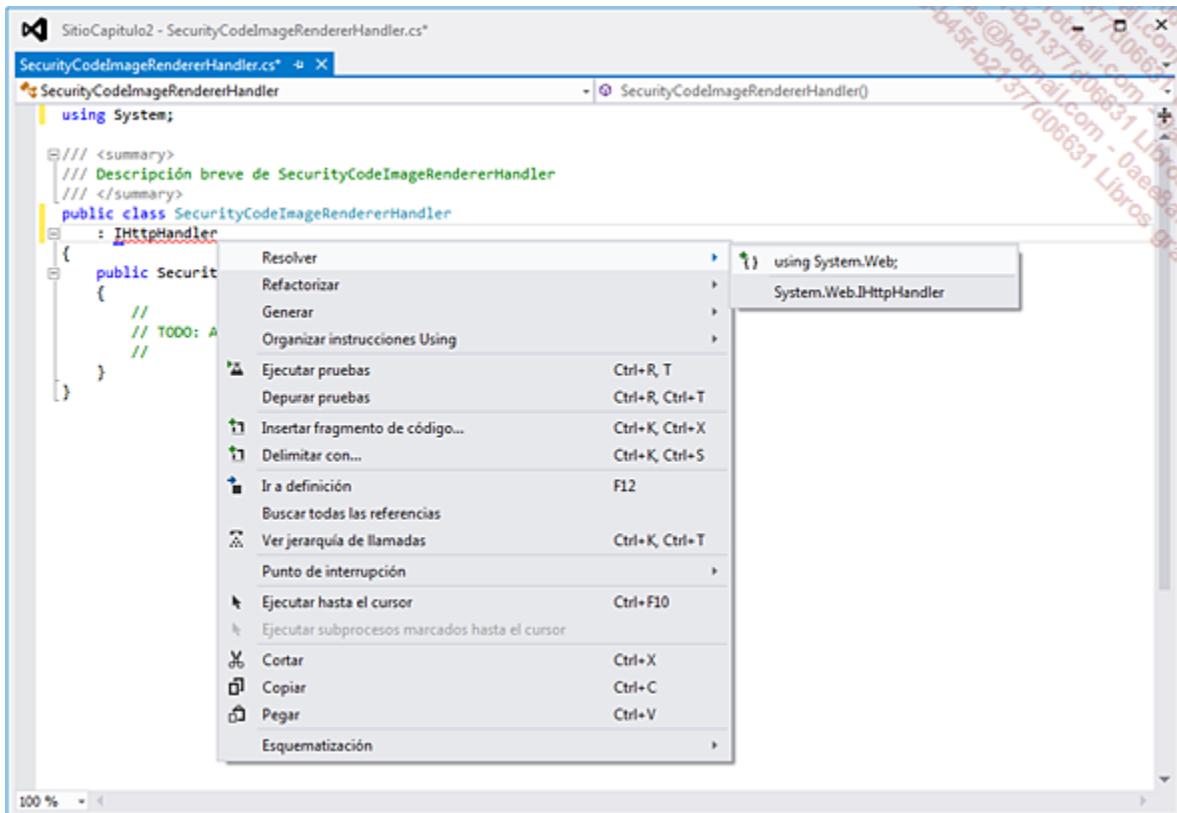
El dibujo se genera de la siguiente forma:



### c. Crear un handler en una DLL

En ocasiones puede resultar útil compartir la definición de un controlador personalizado entre varios proyectos, especialmente cuando éste reposa en lógica de aplicación que también está compartida. Es, por tanto, posible crear una implementación del controlador en una DLL y declararlo en el archivo Web.config del sitio web.

Tomemos, como ejemplo, un controlador que calcula una imagen que incluye un código de tipo captcha que el usuario tiene que reproducir. Comenzamos creando una clase `SecurityCodeImageRendererHandler` dentro de un proyecto de componente servidor web (antes llamado proyecto de librería de clases web). Como controlador HTTP, esta clase debe implementar `IHttpHandler`. Con ayuda del menú contextual **Resolver**, pedimos a Visual Studio que agregue la directiva `using` necesaria para la resolución de nombres de la interfaz.



He aquí el código completo del controlador:

```
class SecurityCodeImageRenderer : IHttpHandler
{
    #region IHttpHandler Miembros

    public bool IsReusable
    {
        get { return false; }
    }

    public Bitmap GetImage()
    {
        int longitud = 220, altura = 80;
        System.Drawing.Bitmap bmp = new System.Drawing.Bitmap
        (longitud, altura);
        System.Drawing.Graphics g = System.Drawing.Graphics.
        FromImage(bmp);
    }
}
```

```
Brush b = new HatchBrush(HatchStyle.DarkDownwardDiagonal,
ColorTranslator.FromHtml("#F0C0C0"), Color.White);
```

```

        Brush bt = new HatchBrush(HatchStyle.DiagonalBrick,
Color.Black);

        g.FillRectangle(b, new Rectangle(0, 0, longitud, altura));

        string text="     A2S0P1.2N ET";
        var font = new Font("Arial", 14, FontStyle.Strikeout);

        int centerX = longitud / 2;
        int centerY = altura / 2;

        System.Drawing.Drawing2D.GraphicsPath path =
            new System.Drawing.Drawing2D.GraphicsPath();

        path.AddString(text, font.FontFamily, 1, 15,
            new PointF(0.0F, 0.0F),
            new StringFormat(StringFormatFlags.DirectionVertical));

        System.Drawing.Drawing2D.Matrix rotateMatrix =
            new System.Drawing.Drawing2D.Matrix();

        rotateMatrix.RotateAt(-65.0F, new PointF(15, altura/4));
        path.Transform(rotateMatrix);

        g.SmoothingMode =
            System.Drawing.Drawing2D.SmoothingMode.HighQuality;

        g.FillPath(bt, path);

        path.Dispose();

        return bmp;
    }
}

```

```

public void ProcessRequest(HttpContext context)
{
    Bitmap bmp = GetImage();
}

```

```

        context.Response.ContentType = "image/png";
        context.Response.ClearContent();
        bmp.Save(context.Response.OutputStream,
System.Drawing.Imaging.ImageFormat.Png);
        context.Response.Flush();
    }

    #endregion
}

```

A continuación, es necesario registrar el controlador en el archivo Web.config, dentro de la sección <system.web><httpHandlers>. La URI es arbitraria:

```

<httpHandlers>
    <add verb="GET" path="SecurityCodeImageRenderer.axd"
        type="capitulo2lib.SecurityCodeImageRendererHandler,
capitulo2lib"/>
</httpHandlers>

```

Para probar nuestro controlador, hemos preparado una página que referencia a esta URI en un tag de servidor image:

```

<table border="0">
    <tr>
        <td>Código de seguridad</td>
        <td><asp:Image ID="is" runat="server" ImageUrl=
"SecurityCodeImageRenderer.axd" /></td>
    </tr>
    <tr>
        <td>Introduzca el código de seguridad que aparece en la imagen:</td>
        <td><asp:TextBox ID="ts" runat="server" /></td>
    </tr>
    <tr>
        <td colspan="2"><asp:Button ID="bs" runat="server"
Text="Aceptar" /></td>
    </tr>
</table>

```



# Los Web Forms

Los formularios web (Web Forms) representan la parte más visible de los sitios web ASP.NET y, en consecuencia, la más popular. Se basan en un reparto de responsabilidades de tipo **MVC**: modelo, vista, controlador. Cuando se escribe un formulario utilizando el estilo **código independiente**, la página HTML .aspx se encarga de la representación (vista), la clase C# gestiona los datos y los cálculos realizados con ellos (modelo), mientras que el servidor de aplicaciones ASP.NET coordina el conjunto (controlador). Este análisis resultará familiar, sin duda, a los desarrolladores Java en lo relativo a la organización de sitios web ASP.NET.

Por otro lado, los formularios web son el resultado de la transposición que realiza Microsoft del modelo Visual Basic 6, y una forma original y productiva de desarrollar interfaces gráficas para Internet. El éxito de este modelo ha sido tal, que Sun lo ha replicado por su cuenta en la tecnología de desarrollo web JSF (*Java Server Faces*).

## 1. Estructura de una página ASPX

En el capítulo Los sitios web ASP.NET 4.5 nos hemos puesto al día con la estructura de una página ASPX desde el punto de vista de la compilación. Ahora se trata de comprender su estructura lógica.

Estudiemos el código que aparece en una página Default.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

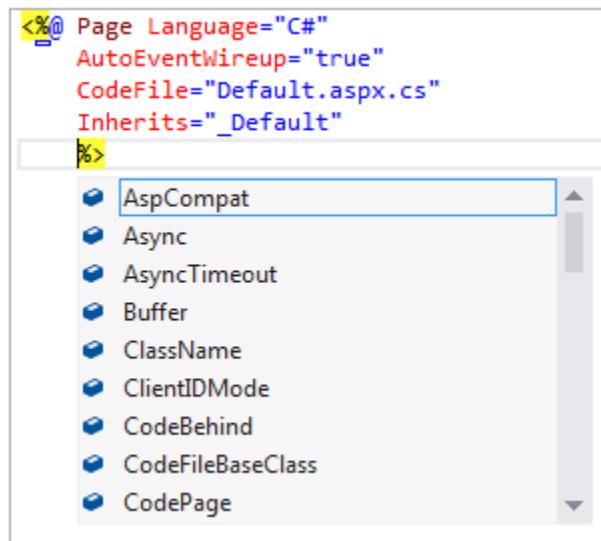
        </div>
    </form>
</body>
</html>
```

Este código está formado por tres partes: una directiva Page, una declaración de DTD y código XHTML.

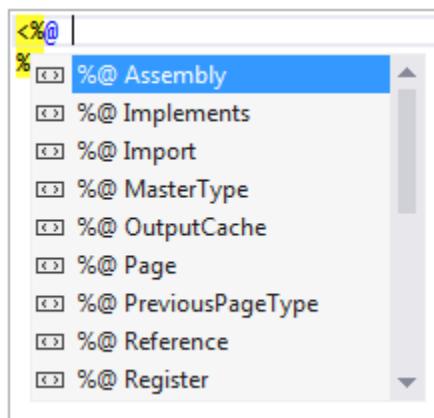
### La directiva Page

Las directivas organizan la lectura de una página ASPX en el servidor de aplicaciones. En la página Default.aspx, el atributo **Language** define el lenguaje - C#, VB.NET, C++ - utilizado para escribir los scriptlets. Hay otros atributos presentes, que sirven para la comunicación con la página de codebehind (**AutoEventWireup**, **CodeFile**, **Inherits**), para aplicar temas, para la gestión de trazas... Descubriremos el uso de estos atributos conforme avance nuestro estudio.

Por suerte, Visual Studio proporciona distintos atributos aplicables utilizando la combinación de teclas [Ctrl][Espacio].



Existen otras directivas disponibles para incluir recursos en el entorno de la página: estrategia de caché, componentes, ensamblados, tipos de página maestra...



## Las DTD

Las definiciones de tipo de documento (*Document Type Definition*) las establece el consorcio W3C. Se trata de una norma aplicable a los documentos SGML, XML y HTML que fija las reglas sintácticas y semánticas de la construcción de un documento basado en tags (marcadores).

Los navegadores son bastante tolerantes en lo que respecta a las DTD. Con la versión ASP.NET 1.X, el flujo HTML de salida es compatible con la DTD **HTML transicional de nivel 4**. Salvo el atributo MS\_POSITIONNING que no estaba filtrado, el código HTML era completamente estándar. Es cierto que una página ASPX contiene etiquetas especiales (<asp:label>, por ejemplo) que se traducen por una secuencia HTML accesible desde el navegador.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

La versión 2.0 aporta una conformidad con **XHTML**, una declaración mucho más estricta del lenguaje HTML. Los puristas pueden dirigirse al sitio web de W3C e introducir una página ASP.NET en el motor de verificación ubicado en la dirección <http://validator.w3.org>. Las páginas deben estar en conformidad con la DTD correspondiente.

Preste atención, para realizar esta prueba, es preciso guardar el flujo HTML en un bloc de notas abierto mediante el comando ver código fuente. La función Guardar como - Página HTML del navegador Internet Explorer modifica el archivo y desvirtualiza la prueba.

Para el desarrollador de páginas web, la conformidad con una versión específica del lenguaje HTML no es suficiente para garantizar que una página tenga la misma presentación sea cual sea el navegador. De entrada, los navegadores tienen la responsabilidad de interpretar las reglas de representación tal y como ellos las entiendan. El lenguaje HTML describe el contenido, pero no la representación. Además, las páginas incluyen código JavaScript y estilos CSS, que difieren en su interpretación en función del navegador.

El servidor ASP.NET 2.0 introduce otro cambio: desaparece la noción de esquema de navegador de destino (Netscape 3, 4, 5, IE 3, 4, 5+). Es cierto que esta directiva no ha podido estar a la par con la evolución de los navegadores, sin contar con la aparición de otros dispositivos de navegación. En su lugar, los sitios web ASP.NET poseen una carpeta **App\_Browsers** que considera las características de cada navegador. Este aspecto se estudiará cuando aparezcan los componentes personalizados.

Para ciertos navegadores y programas JavaScript que intervienen en el DOM y que no sean compatibles con la norma XHTML, el servidor de aplicaciones puede configurarse para utilizar el modo HTML transicional. La directiva se ubica en el archivo Web.config:

```
<xhtmlConformance mode="Legacy"/>
```

El atributo mode acepta tres valores:

Legacy	Antiguo formato HTML transicional
Strict	XHTML strict
Transitional	XHTML transicional

EI

### código XHTML

Si bien es cierto que el servidor de aplicaciones ASP.NET 1.X emitía un flujo conforme a la DTD HTML 4 transicional, la propia sintaxis de las páginas ASPX mezclaba secuencias HTML con secuencias XML. Visual Studio 2003 se encargaba de controlar la coherencia del conjunto y generar advertencias cuando era necesario, y el servidor de aplicaciones debía realizar una lectura más atenta (y costosa) para separar las secuencias HTML de las secuencias XML.

Ahora, el elemento <html> contiene una referencia al espacio de nombres XHTML:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

Dicho de otro modo, las etiquetas de una página ASPX deben respetar la sintaxis XHTML. De este modo, las etiquetas que comienzan por asp (controles web), uc (controles de usuario) o cc (controles personalizados) no forman parte del vocabulario XHTML. Pero, al menos, la sintaxis es mucho más próxima y más precisa. Y el flujo de salida permanece, en cualquier caso, conforme a la DTD declarada.

Por último, Visual Studio hace todo lo posible para validar de antemano las secuencias HTML que figuran en una página ASPX. Se generan mensajes de advertencia para llamar la atención del desarrollador acerca de las no conformidades.

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** Capítulo - Microsoft Visual Studio
- Menu Bar:** ARCHIVO, EDITAR, VER, SITIO WEB, COMPILAR, DEPURAR, EQUIPO, SQL, HERRAMIENTAS, PRUEBA, ANALIZAR, VENTANA, AYUDA
- Toolbar:** Includes icons for file operations like Open, Save, Print, and a dropdown for browser selection set to Internet Explorer.
- Code Editor:** Displays the Default.aspx file with C# code embedded in the HTML. The code uses a foreach loop to generate a list of days of the week.
- Syntax Highlighting:** Colors code elements (HTML tags, C# keywords, comments) in red, blue, and black.
- Toolbars:** Cuadro de herramientas (Toolbox) on the left, and Explorador de servidores, Propiedades, and Explorador de soluciones on the right.
- Status Bar:** Shows the status "Listo".
- Bottom Status Bar:** Shows "Lin 19 Col 12 Car 12 INS".
- Toolbox:** Shows the "Equipo" tab selected.
- Properties Window:** Shows the "Propiedades" tab selected.
- Solutions Explorer:** Shows the "Explorador de soluciones" tab selected.
- Task List:** Shows "Lista de errores" with one warning: "Validación (XHTML5): el elemento 'li' no se puede anidar dentro del elemento 'div'." at line 17.

### a. Estilo anidado, en línea y separado

La organización de una página dinámica es una simple cuestión de estilo. Según la naturaleza de la secuencia HTML que se quiera describir, es preferible optar por la versión anidada o por la versión en línea (inline). Sólo el estilo separado (code behind) supone un cambio radical y aporta una distinción neta entre la presentación y el cálculo. Éste es el motivo por el que se le da privilegio en Visual Studio.

## El estilo anidado

Son las primeras generaciones de las páginas dinámicas (ASP, PHP) las que imponen el estilo anidado. Con los modelos de componentes web ASP.NET, deja de tener tanto sentido, aunque sigue siendo aplicable. También puede servir en el caso de controles a base de modelos tales como los Repeater o los Data List.

He aquí un ejemplo de código basado en este estilo:

```

<body>

    <form id="form1" runat="server">

        <ul>
            <%
                int i;

                string[] dias = { "lunes", "martes", "miércoles", "jueves",
"viernes", "sábado", "domingo" };

                for(i=0; i< dias.Length; i++)
                {
                    %>
                    <li><%= dias[i] %></li>
                }
            <% } %>
        </ul>
    </form>
</body>

```

El desarrollador debe trabajar de la mejor forma para alinear su código como si se tratase de un programa escrito completamente en C#.



### El estilo en línea (inline)

El estilo anidado se utiliza, principalmente, en la presentación. No conviene utilizarlo cuando se planifica el procesamiento. La versión en línea separa el código C# y el código HTML en dos partes del mismo archivo .aspx. Las etiquetas `<script runat="server">` indican al compilador que se trata de código C#, aunque puedan reemplazarse por scriptlets `<% %>`.

```

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

<script runat="server">
    // contiene el código de procesamiento de eventos
    void procesar_click(object sender, EventArgs e)

```

```

    {
        mensaje.Text = "¡Ha hecho clic!";
    }


```

</script>

<!-- límite entre el código C# y el código HTML -->

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Estilo en línea</title>
</head>

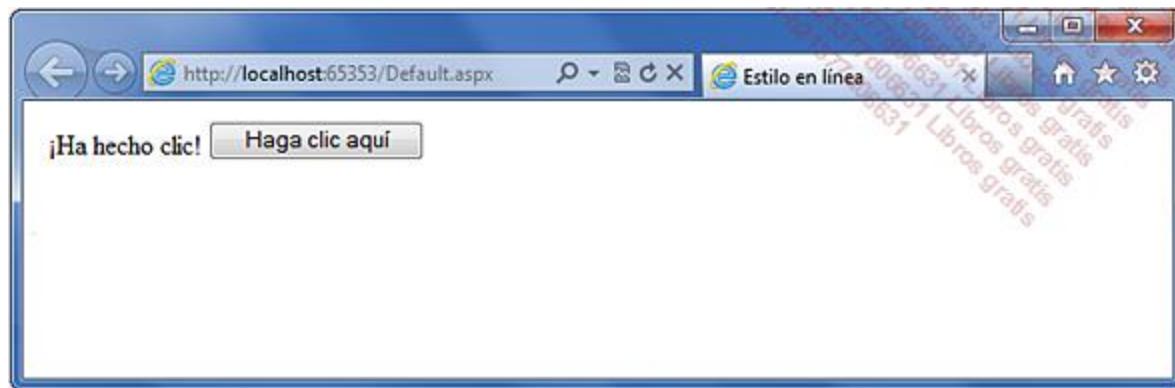
```

```

<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="mensaje" runat="server"></asp:Label>
            <asp:Button ID="cmd" runat="server" Text="Haga clic aquí"
                OnClick="procesar_click" />
        </div>
    </form>
</body>
</html>

```

El límite entre ambas secciones de código es responsabilidad del desarrollador, quien tiene la libertad para respetar o no esta división.



### El estilo separado

Cuando el procesamiento es complejo, no es recomendable ubicarlo en una página HTML. La calidad del desarrollo se verá afectada. El estilo separado (también llamado code-behind) funciona tal y como hemos descrito en el capítulo Los sitios web ASP.NET 4.5: la vista HTML forma una clase que hereda de la clase C# que figura en el archivo de código subyacente. Esto explica por qué los controladores de eventos están cualificados mediante la palabra reservada **protected** (miembro accesible por las subclases).

Más allá de una organización más clara, la clase de código subyacente impone también una cronología de eventos. Para implementar un procesamiento, el desarrollador debe determinar dónde debe ubicarse su código. Para responder a esta cuestión, hay que preguntarse cuándo debe ejecutarse el código. A cada momento le corresponde un evento (`init`, `load`, `click`...).

## b. Los scriptlets

Los scriptlets son fragmentos de código que figuran en una página ASPX. Están delimitados por marcadores, que los distinguen de las secuencias HTML.

ASP.NET cuenta con cuatro tipos de scriptlets:

Los	
<% instructions %>	Instrucciones ejecutadas de arriba a abajo, anidadas en el código HTML.
<%= expression %>	Expresión evaluada durante el renderizado de la página.
<%# expression %>	Expresión evaluada cuando se invoca el método de la página o del control DataBind() (véase el capítulo El acceso a datos con ADO.NET 4.5).
<%\$ expression %>	Expresión analizada en tiempo de compilación de la página y evaluada tras cada petición.

### **bloques de instrucciones <% %>**

Estos bloques de instrucciones se ejecutan durante la visualización de la página. Pueden influir en la fabricación de secuencias HTML, tal y como muestra el siguiente ejemplo de código anidado:

```
<%  
    int i;  
    string[] dias = { "lunes", "martes", "miércoles", "jueves",  
    "viernes", "sábado", "domingo" };  
    for(i=0; i<dias.Length; i++)  
    {  
        %>  
        <li><%= dias[i] %></li>  
    } %>
```

Se generarán exactamente 7 etiquetas `<li>...</li>`, tantas como iteraciones del bucle `for`.

### **Las expresiones <%= %>**

Las expresiones que figuran entre <%= %> se evalúan sistemáticamente en el contexto de ejecución. Puede tratarse de valores literales, de variables o de llamadas a métodos.

```
<li><%= dias[i] %></li>  
<%= DateTime.Now.ToString() %>
```

### Las expresiones anidadas <%# %>

Desde un punto de vista sintáctico, podemos considerar las expresiones anidadas <%# %> como una variación de las expresiones sistemáticas <%= %>. Ciertos controles, tales como las listas o las tablas de datos, iteran sobre registros con datos. El origen de los datos se enlaza a estos componentes mediante su propiedad DataSource y, a continuación, se invoca el método DataBind(). Esto establece el orden de resolución de las expresiones <%# %> que hacen referencia a las columnas del origen de datos:

```
<asp:Repeater ID="rep1" runat="server">  
  
    <ItemTemplate>  
        <%# DataBinder.Eval(Container.DataItem, "precio") %>  
    </ItemTemplate>  
</asp:Repeater>
```

El estudio de los controles de datos (véase el capítulo El acceso a datos con ADO.NET 4.5) y de los controles basados en un modelo (véase la sección Componentes personalizados, en este capítulo) detalla esta sintaxis, algo compleja.

### Las \$-expressions <%\$ %>

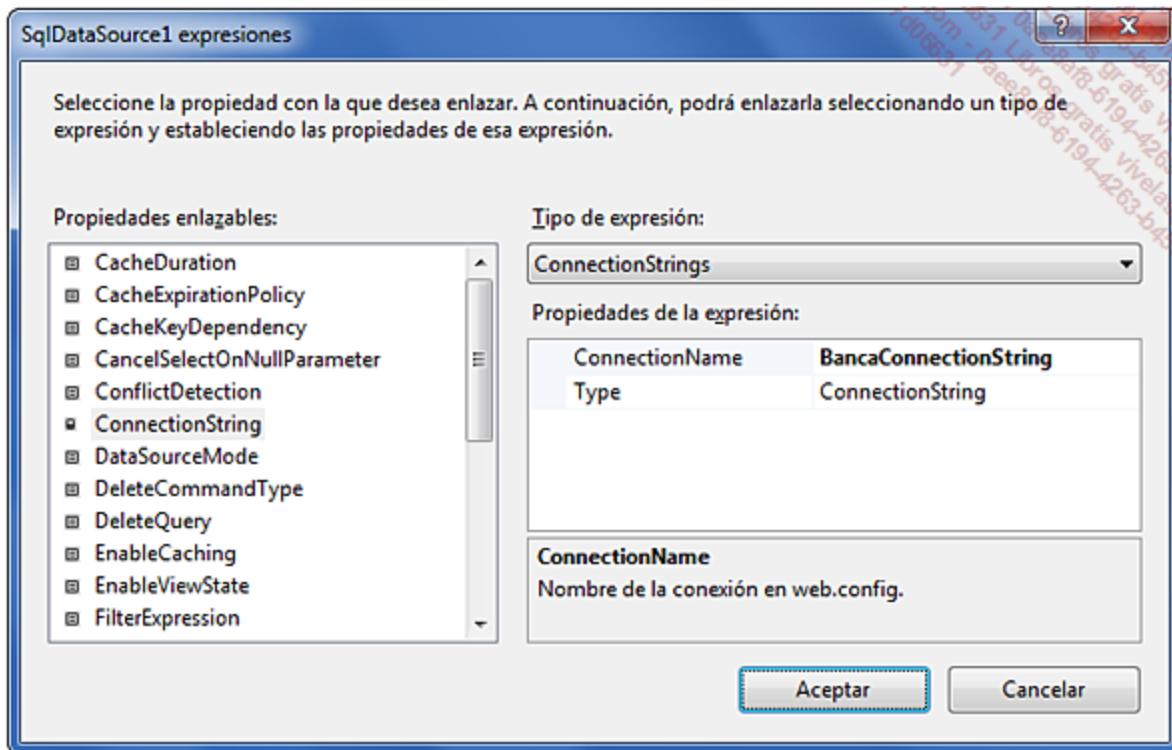
Las expresiones ligadas son útiles a la hora de acceder a las bases de datos. Aunque estas expresiones no se evalúan hasta el momento en que se produce la llamada al método DataBind(). Pueden aparecer errores de contexto que se producen demasiado tarde como para ser corregidos.

Además, las expresiones <%= %> no pueden figurar como valor de atributo, de modo que la siguiente línea sería incorrecta:

```
<asp:Label ID="lbl" runat="server" Text='<%= 10 %>' />
```

Para satisfacer ambos requisitos, Microsoft ha dotado a ASP.NET de las \$-expressions. Se trata de expresiones de análisis en tiempo de compilación, que limitan el riesgo de errores contextuales y que pueden figurar como valor de un atributo.

El servidor de aplicaciones ASP.NET y Visual Studio explotan, ambos, las \$-expressions. Ciertas expresiones estándar se reconocen directamente en Visual Studio y el desarrollador las aprovecha sin tener que introducir código; la propiedad (Expressions) es accesible mediante ciertos controles web y reemplaza al anterior sistema de propiedades dinámicas.



El código generado por Visual Studio es una \$-expression:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:BancaConnectionString %>" 
SelectCommand="select * from dbo.cuenta"></asp:SqlDataSource>
```

ASP.NET proporciona, de manera estándar, tres tipos de expresiones:

ConnectionString	Lee directamente la sección connectionStrings del archivo Web.config.
AppSettings	Lee la sección appSettings del archivo Web.config.
Resources	Lee una entrada de un archivo de recursos.

### Creación de \$-expressions personalizadas

Es posible crear \$-expressions personalizadas. Esta operación se realiza mediante la API Code DOM, que gestiona la inyección de código en una página web.

He aquí el código de una expresión que es capaz de decodificar cadenas de conexiones inscritas en el archivo Web.config. En efecto, este archivo, si bien no es accesible a través del canal HTTP, contiene, a menudo, información sensible (logins y contraseñas de acceso a la base de datos). IIS es capaz de trabajar con este archivo Web.config codificado, aunque debe encriptarse el archivo completo. Es, por tanto, oportuno mejorar la \$-expression ConnectionString agregando un método de desencriptación.

```
[ExpressionPrefijo("CSCodee")]
```

```
public class CStringCodeeExpressionBuilder : ExpressionBuilder
{
```

```

public CStringCodeeeExpressionBuilder() : base()
{
}

#region Desencriptación
public static string descripta(string cs)
{
    ...
}
#endregion

public static string GetString(string name)
{
    return descripta(ConfigurationManager.ConnectionStrings[name].
ConnectionString);
}

public override CodeExpression GetCodeExpression(BoundProperty
Entry entry, object parsedData, ExpressionBuilderContext context)
{
    string name = entry.Expression; // nombre de la cadena de conexión

    Type type = entry.DeclaringType;
    PropertyDescriptor descriptor;
    descriptor = TypeDescriptor.GetProperties(type)[entry.
 PropertyInfo.Name];

    CodeExpression[] extab = new CodeExpression[1];
    extab[0] = new CodePrimitiveExpression(name);

    // será preciso invocar a la evaluación de la expresión
    // el método estático GetString pasándole como parámetro
    // el nombre de la cadena de conexión
    return new CodeCastExpression(

```

```

        descriptor.PropertyType,
        new CodeMethodInvokeExpression(new CodeTypeReference
Expression(base.GetType()),
        "GetString", extab));

```

```
}
```

```
}
```

El algoritmo de desencriptación no se ha reproducido aquí para no sobrecargar la presentación. La versión disponible para su descarga contiene un programa completo.

Antes de emplearse en una página ASPX, la expresión debe registrarse en el archivo Web.config. El elemento <compilation> recibe la declaración correspondiente:

```
<compilation debug="true">
```

```
  <expressionBuilders>
```

```
    <add expressionPrefix="CSCode"
```

```
      type="CStringCodeeeExpressionBuilder"/>
```

```
  </expressionBuilders>
```

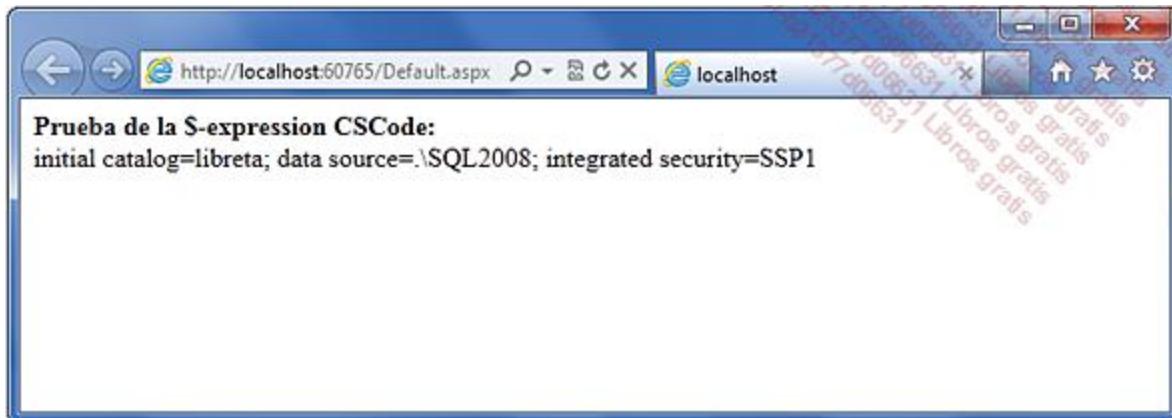
```
</compilation>
```

El comando <add> admite dos atributos, uno que indica el prefijo de llamada de la expresión y otro que especifica la clase que implementa. Sobra decir que esta clase puede y debe declararse en una librería de clases separada. El atributo type respetará, por tanto, el formato "namespace.clase, ensamblado".

Sólo queda probar el funcionamiento de dicha expresión personalizada. Aplicada, aquí, a un literal, la expresión puede emplearse en un componente de acceso a datos SqlDataSource.

```
<asp:Literal ID="Literal2" runat="server"
```

```
  Text="<%$ CSCode:cx_libreta %>"></asp:Literal>
```



### c. Jerarquía de controles

Un formulario web está compuesto de controles web - zonas de texto, listados, opciones a marcar... Para Microsoft, los términos controles y componentes son, prácticamente, idénticos. Un componente es una clase compilada, un control es un componente dotado de responsabilidades de aplicación o gráficas.

En realidad, la clase Page hereda, ella misma, de System.Web.UI.TemplateControl quedaría de System.Web.UI.Control. Es,

por tanto, esta última clase por la que nos debemos interesar si queremos comprender la jerarquía de controles de una página.

He aquí algunas propiedades de la clase `System.Web.UI.Control`:

Controls	Lista (o, mejor dicho, colección) de controles anidados.	Una página es, por tanto, un control un tanto
ControlState	Se trata de una novedad en ASP.NET 2.0. Complemento de ViewState.	
EnableViewState	Activa o desactiva la inscripción del control en el ViewState de la página.	
ID	Nombre del control.	
Page	Página a la que está enlazado el control.	
Parent	Control que contiene el control en curso, si existe (la Page no tiene control Parent).	
Visible	Si esta propiedad vale falso, la renderización HTML del control no se realiza, y tampoco para los controles siguientes (anidados).	
HasControls	Indica si la colección Controls contiene, al menos, un control.	
Init, Load, PreRender, Render, Unload	Eventos que gobiernan la vida de un control.	

particular. Es preferible considerar que su colección `Controls` es el punto de partida de la jerarquía. Ésta puede, por otro lado, mostrarse activando la traza de un formulario web:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="jerarquia.aspx.cs" Inherits="jerarquia" Trace="true"
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Jerarquia</title>
</head>
<body>
    <form id="form1" runat="server">
```

```
<div>
    <asp:TextBox ID="TextBox1" runat="server" />
    <asp:Button ID="Button1" runat="server" Text="Button" />
</div>
</form>
</body>
</html>
```

En tiempo de ejecución de la página, se inserta la traza en el flujo HTML, permitiendo seguir el desarrollo de la petición, y visualizar la jerarquía de controles:



Es, por otro lado, interesante subrayar que las etiquetas HTML forman parte del árbol como controles literales. El servidor de aplicaciones les asigna, a cada una, un nombre diferente, pero difícil de prever. A este propósito, la clase **Control** posee un método **FindControl** que permite buscar un control a partir de su nombre. Esta operación se produce cuando no es posible determinar el nombre de un control en tiempo de compilación.

```
// buscar un control en la página
Control c = Page.Controls[0];

// buscar un control a partir del nombre
Control f = Page.FindControl("form1");
TextBox t = f.FindControl("TextBox1") as TextBox;

// el método FindControl puede buscar de forma recursiva
TextBox t1=Page.FindControl("TextBox1") as TextBox;
```

#### d. Agregar controles dinámicamente

El modelo ASP.NET permite agregar controles dinámicamente. Esta forma de trabajar se utiliza, principalmente, en los controles personalizados compuestos, aunque funciona también a nivel de página. En este último caso, el programador debe prestar atención y crear controles web o HTML en la jerarquía del formulario y no a nivel de la página. Esto equivaldría a situar una etiqueta `<asp:...runat="server">` en lugar de la etiqueta `<form runat="server">` produciendo un error en el servidor de aplicaciones en tiempo de análisis de la página.

El siguiente programa ilustra cómo crear dinámicamente un botón capaz de responder a un clic.

```
public partial class ct_dinamico : System.Web.UI.Page
{
    private Button b;
```

```

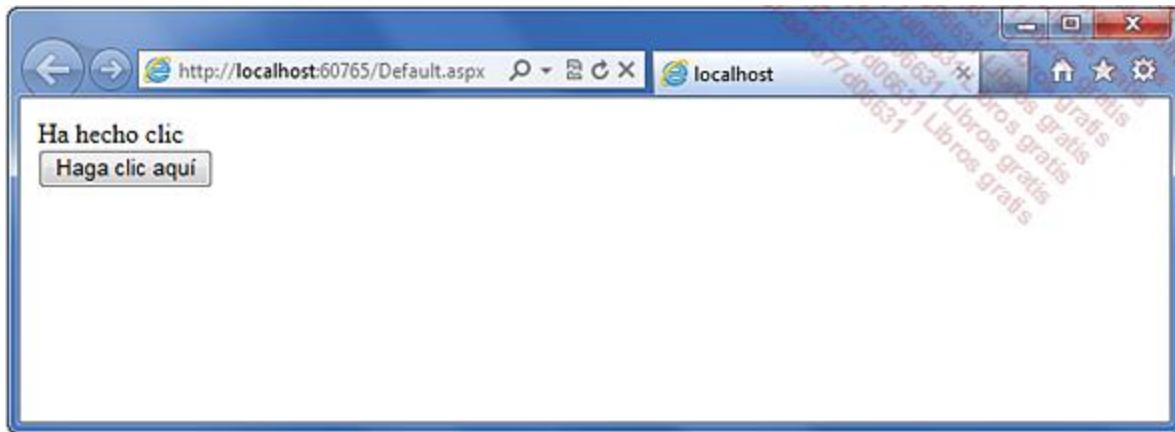
protected void Page_Load(object sender, EventArgs e)
{
    // creación dinámica del botón
    b = new Button();
    b.Text = "Haga clic aquí";
    b.ID = "boton";

    // el botón debe ser un descendente del formulario
    form1.Controls.Add(b);

    // inscribimos el controlador del evento
    b.Click += new EventHandler(b_Click);
}

void b_Click(object sender, EventArgs e)
{
    Label1.Text = "Ha hecho clic";
}

```



#### e. Objetos intrínsecos

La clase `Page` expone varias propiedades públicas que llamaremos objetos intrínsecos. Estos objetos se corresponden, de hecho, con los miembros del contexto `http` del capítulo Los sitios web MVC. De este modo, `Page.Request` es idéntico a `HttpContext.Current.Request`.

Tres de estos objetos ya estaban disponibles antes de ASP.NET: **Request**, **Form** y **Response**. En el marco del estudio de los formularios web, vamos a presentar estos tres objetos en detalle.

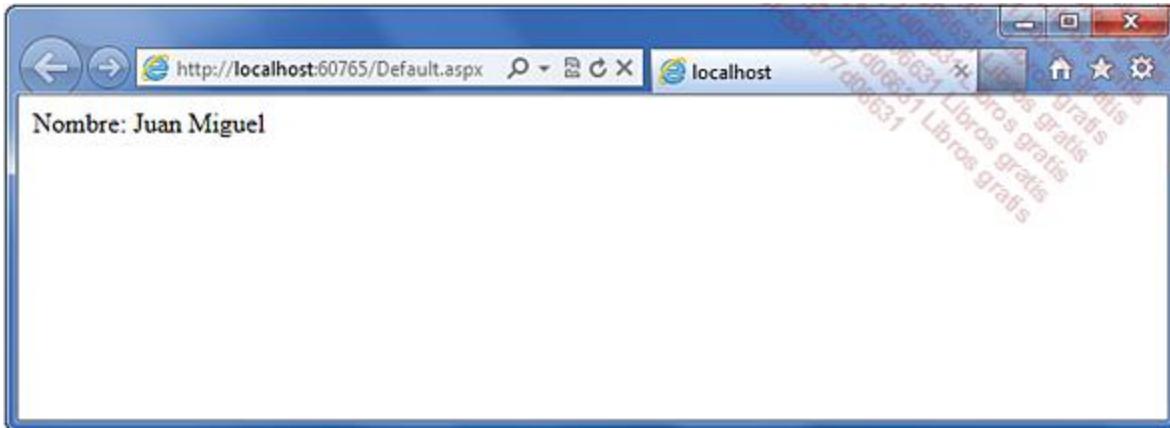
#### **Request**

El objeto `Request` representa a los parámetros enviados desde el navegador al servidor cuando se produce una petición de tipo **GET**. En los formularios web ASP.NET, las peticiones GET se corresponden, a menudo, con enlaces simples de hipertexto. En el caso de una petición de tipo POST (es decir, un **postback** en ASP.NET), el objeto `Request` puede, también, incluir información de tipo parámetro: se trata de la cadena de interrogación **Query String**.

Esta cadena figura en la URL tras el nombre de la página .aspx. Comienza con un signo ? y está formada por una lista de pares **clave=valor** separados por el símbolo &.

El siguiente extracto de código muestra cómo decodificar esta cadena:

```
Label1.Text = "Nombre: "+Request.QueryString["nombre"];
```



A pesar de la riqueza de los controles web de servidor, la cadena de interrogación sigue siendo útil para configurar una página que se invoca tras la selección de un registro. Consideremos, por ejemplo, la siguiente dirección:

```
http://localhost/ventas/articulo.aspx?id_articulo=15
```

Para determinar qué ficha de producto debemos presentar, la página `articulo.aspx` decodifica la cadena de interrogación mediante la expresión:

```
Request.QueryString["id_articulo"]
```

El objeto `Request` incluye, a su vez, el conjunto de parámetros de la petición HTTP:

- Dirección (URL).
- Agente (Browser).
- Variables de servidor (ServerVariables).
- Tipo de contenido, en el caso del POST.

Incluso si la clase `Page` y los controles web tienen la potencia suficiente como para presentarnos la petición desde un enfoque de alto nivel, el desarrollador puede contar con el objeto `Request` para controlar con detalle las condiciones de ejecución de una página ASPX.

## Form

La clase Page posee una propiedad Form de tipo **HtmlForm** que representa a la etiqueta <form>; existe un segundo objeto **Form**, propiedad esta vez del objeto Request.

Este último objeto, de tipo **NameValueCollection**, representa los datos enviados. Con la decodificación de los controles web de servidor existentes, es raro acceder a dicho objeto en ASP.NET, aunque siempre es una posibilidad a la hora de construir extensiones o para mantener un formato compatible con ASP.

## Response

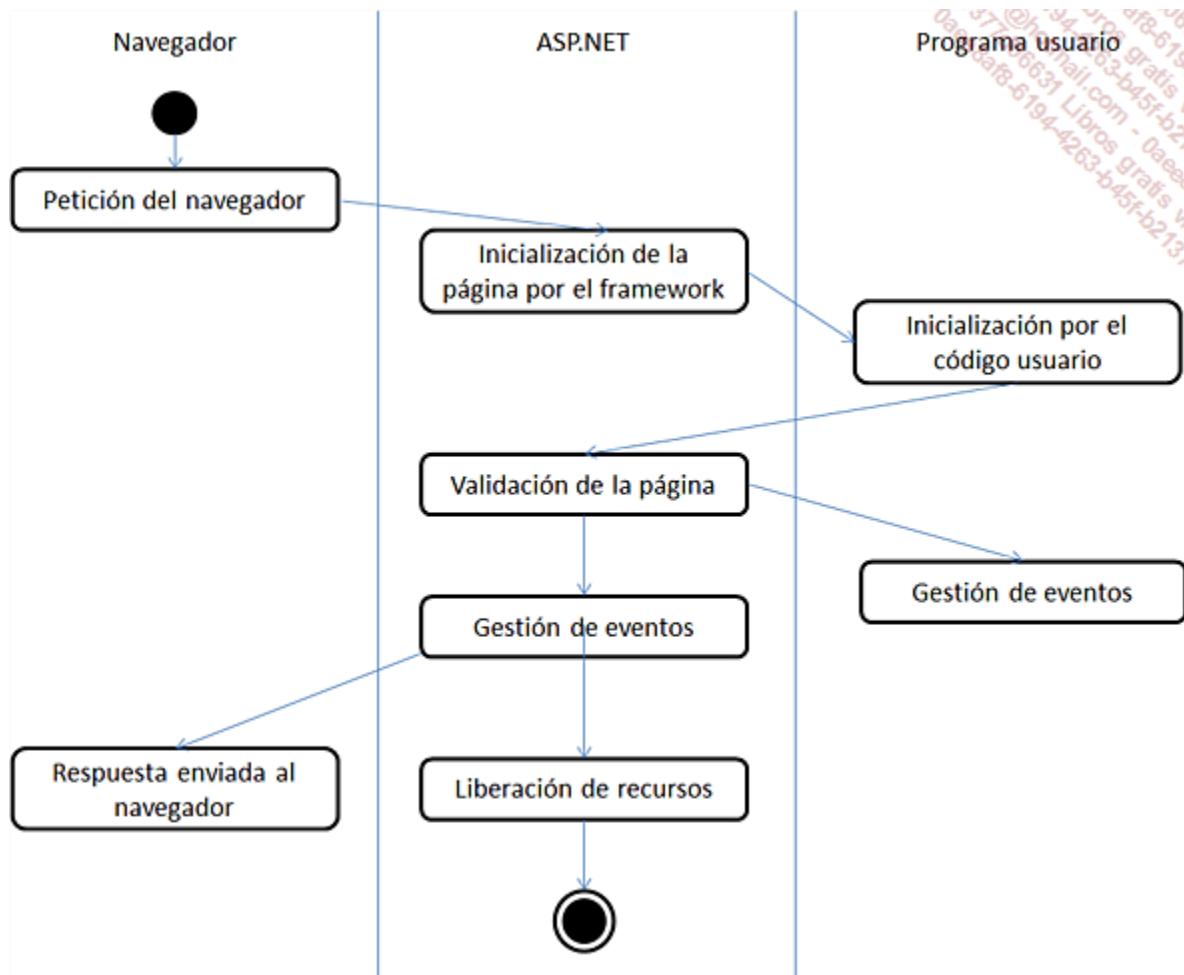
En ASP, el objeto **Response** se empleaba para escribir en el flujo de salida HTML sin tener que salir de la programación. Este procedimiento no ha variado con ASP.NET, puesto que los controles web son capaces de asegurar su propia visualización.

No obstante, el objeto Response sigue estando disponible para escribir otro tipo de flujo - PDF, Excel a partir de una página ASPX. Además, el objeto **Response** se utiliza en la gestión de la caché HTML.

## 2. Ciclo de vida de una página

### a. El ciclo nominal

Una página dinámica ASPX es un programa que se ejecuta según un proceso basado en eventos. Antes de detallar las diferentes señales que puede recibir un programador para diseñar su programa, veremos este proceso a grandes rasgos.



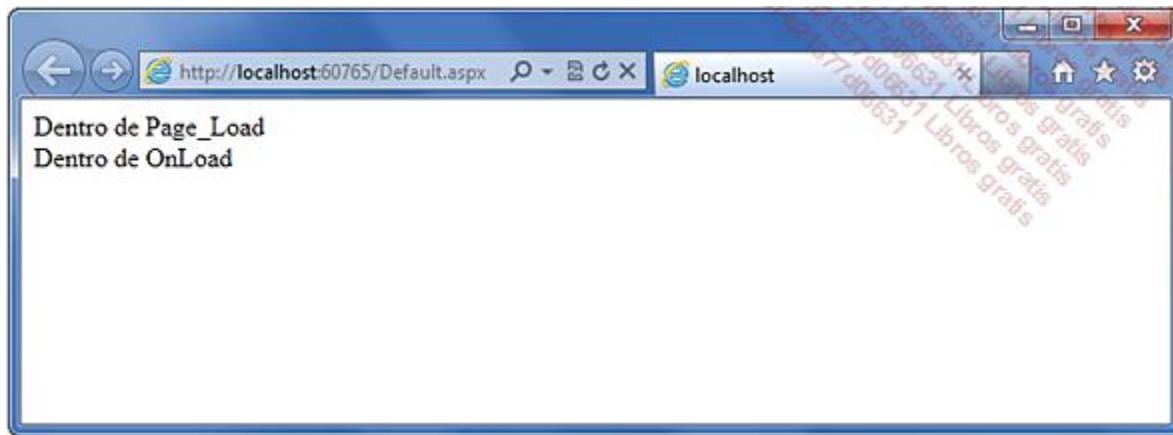
## Inicialización de la página por el framework

Se trata de la primera fase del proceso de ejecución de una página. El framework instancia la página e inicializa sus controles. El ViewState se carga y, a continuación, se decodifica. Esta fase se corresponde con los eventos **FrameworkInitialize**, **PreInit** e **Init**. El desarrollador rara vez ubica su código de aplicación a este nivel, pues los controles todavía no existen.

## Inicialización mediante el código de usuario

Esta fase se corresponde con el evento **OnLoad** y con el método **Page\_Load**. A diferencia de los formularios Windows Winform, el evento **OnLoad** se produce tras cada petición. Por lo general, el método **Page\_Load** también se invoca, a menos que el valor del atributo **AutoEventWireUp** de la directiva <%@ Page %> tenga un valor igual a false.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="autoevent.aspx.cs" Inherits="autoevent" %>
```



```
public partial class autoevent : System.Web.UI.Page
{
    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);
        Label1.Text += "Dentro de OnLoad<br>";
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        // se invoca solamente si AutoEventWireUp=true
        Label1.Text += "Dentro de Page_Load<br>";
    }
}
```

Como Visual Studio crea formularios con el atributo **AutoEventWireUp** con el valor true, y genera el método **Page\_Load()**, el código de usuario prácticamente siempre se ubica en este método y no en **OnLoad**.

El programador debe prestar atención a no ubicar todo su código en Page\_Load o en OnLoad. Por un lado, la petición no ha terminado de ser procesada, los eventos de carga y de acción llegarán más tarde, tras la validación de los datos introducidos por el usuario. Por otro lado, el programa se volverá ineficaz o demasiado complejo de mantener.

## Validación de datos introducidos por el usuario

ASP.NET dispone de controles integrados para controlar los datos introducidos por los usuarios, según los formatos estándar (campos obligatorios, fechas bien formadas, números, e-mails...). Estos controles se validan, en primer lugar, mediante JavaScript. Además, el servidor de aplicaciones les aplica una validación adicional del lado servidor. Por último, la página dispone de una propiedad **IsValid** para determinar si se han realizado con éxito todas las verificaciones.

## Gestión de eventos

Los eventos son clases ubicadas en cuatro categorías que se producen en el orden siguiente:

- Eventos que se producen al inicio de un retorno de una llamada a una función JavaScript `_doPostBack()`.
- Eventos de cambio de ubicación en caché, a menos que la propiedad **AutoPostBack** no valga true.
- Eventos de cambio, tales como **TextChanged** o **SelectedIndexChanged**.
- Eventos de acción tales como **Click**.

El desarrollador incluye, habitualmente, controladores para cada uno de estos eventos en el archivo de código subyacente. Al finalizar las distintas operaciones, se invocan los métodos de enlace, se fija el estado de la vista ViewState, y se produce la visualización de la página.

Justo antes de la transformación del modelo C# en una secuencia HTML, se produce el evento **PreRender**: se trata de una señal que indica que las nuevas modificaciones en el modelo C# ya no se tendrán en cuenta. A menos que el evento **Render** no se controle, para ciertos tipos de controles (por ejemplo un calendario en el que se quiere modificar la apariencia de los días festivos), se produce la visualización en base a los valores de los atributos y propiedades de los distintos componentes de la página.

## Liberación de recursos

Mientras el servidor web IIS dirige la página HTML al navegador, el servidor de aplicaciones libera los recursos. La instancia de Page se destruye, todos sus campos pierden su valor y el recolector de basura (garbage collector) recupera la memoria de forma periódica.

### b. Identificar las peticiones de tipo postback

El postback se corresponde con un retorno de la página después de que el usuario haya realizado alguna acción sobre un control web de tipo botón, calendario... Algunos eventos sólo se producen en este preciso instante, pero el método de Page\_Load() se invoca sistemáticamente. Para determinar si la página se ejecuta mediante un GET HTTP - un enlace desde otra página, una dirección introducida manualmente en la barra de direcciones del navegador - o si la petición se genera a partir de un retorno de formulario (postback), la página expone la propiedad estática **IsPostBack**.

A menudo se comprueba la negación de esta propiedad para saber cuándo hay que inicializar los controles de la página.

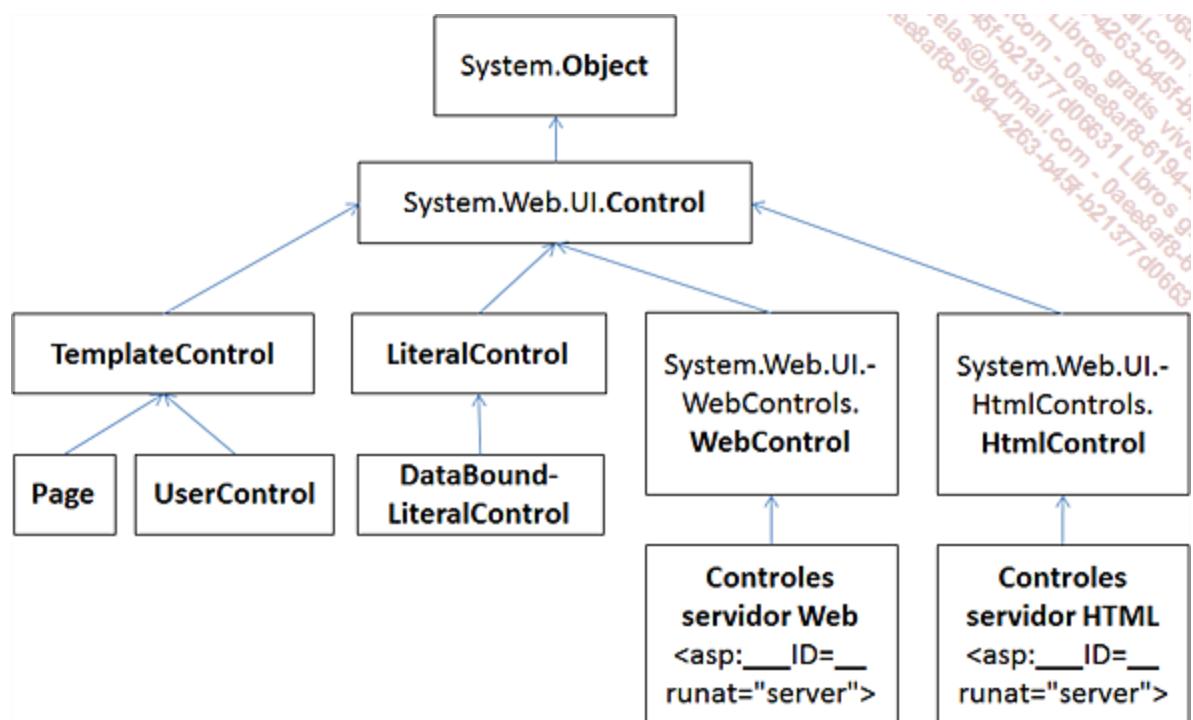
```
protected void Page_Load(object sender, EventArgs e)
{
    if( ! IsPostBack)
    {
        ListBox1.Items.Add("Las Palmas");
        ListBox1.Items.Add("Lanzarote");

        ListBox1.Items.Add("Fuerteventura");
    }
}
```

Si no se realizara esta comprobación, la lista `ListBox1` se incrementaría de tres en tres ítems tras cada ejecución de la página.

### 3. Los controles web

Las páginas ASP.NET tienen como principal objetivo exponer controles web. Éstos interactúan con el usuario aceptando datos introducidos manualmente y mostrando sus valores. Microsoft ha definido un verdadero DOM (*Document Object Model*) del lado servidor, lo que permite explotar estos controles de una forma mucho más productiva que los objetos `Request` y `Response`. De este modo, el objetivo del programador de páginas ASP.NET no es crear código HTML "a mano" sino crear un servicio funcional de cara al usuario. Los controles web, más o menos integrados, tienen la responsabilidad de decodificar aquellos elementos de las peticiones que les afectan y fabricar, bajo demanda, parte de la respuesta que se devuelve al navegador.



## a. Las etiquetas HTML

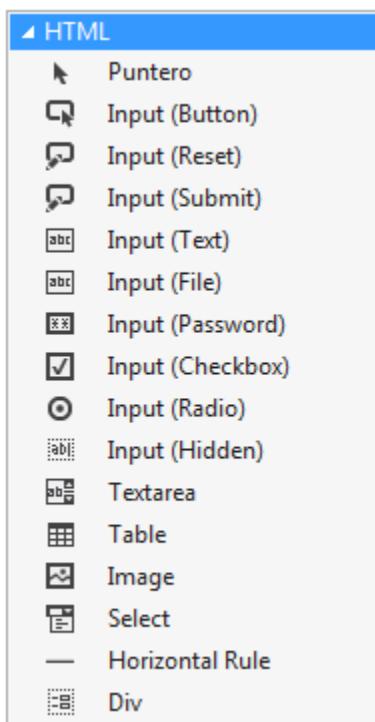
Tras las primeras versiones del framework, el código 100% HTML ha encontrado su lugar. Microsoft parece haber abandonado la representación absoluta (top, left) mediante CSS, sugerente dado que es similar a los métodos de diseño utilizados en Windows, pero del todo desalineada con las restricciones en la creación de sitios web. Este modo de crear pantallas sigue siendo aplicable, pero Visual Studio 2005 invita al programador a diseñar sus páginas creando un reparto del espacio con la ayuda de tablas (`<table>`) y de divisiones (`<div>`).

Las etiquetas HTML se integran en el DOM bajo la forma de controles LiteralControl. Es, de este modo, posible agregar nuevas secuencias HTML instanciando esta clase, pero esta operación tiene, a menudo, lugar en el método de visualización del control personalizado.

Cuando se edita una página ASPX en modo Diseño, Visual Studio trabaja como un procesador de texto web: los retornos de carro se convierten en etiquetas `<br>`, la negrita ([Ctrl] B) delimita el texto mediante los tags `<b>...`. Sin esperar grandes maravillas en lo relativo a la ergonomía, los menús y los cuadros de herramientas resultan bastante prácticos para diseñar una página HTML. Los más valientes pasarán a la vista Código para editar directamente el código HTML.

En la práctica, se combinan, habitualmente, ambos enfoques. A menudo, el diseño general se realiza con ayuda del ratón en modo Diseño, mientras que las definiciones se realizan introduciéndolas directamente en las etiquetas. También es habitual utilizar otras herramientas de diseño gráfico de páginas web, tales como Dreamweaver, y programar el funcionamiento de las páginas con Visual Studio.

En cuanto al cuadro de herramientas de Visual Studio, está más orientado a los controles que a las etiquetas. Éste es el motivo por el que la sección HTML no contiene prácticamente campos de formulario HTML:



## b. El atributo runat="server"

Consideremos la siguiente página HTML:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="etiquetas_html.
aspx.cs" Inherits="etiquetas_html" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <input type="text" name="nombre" />
            <input type="submit" name="boton1" value="Enviar" />
        </div>
    </form>
</body>
</html>
```

Se trata de una página ASPX que incluye un formulario, el cual contiene dos campos HTML **nombre** y **botón**

# Organizar la presentación

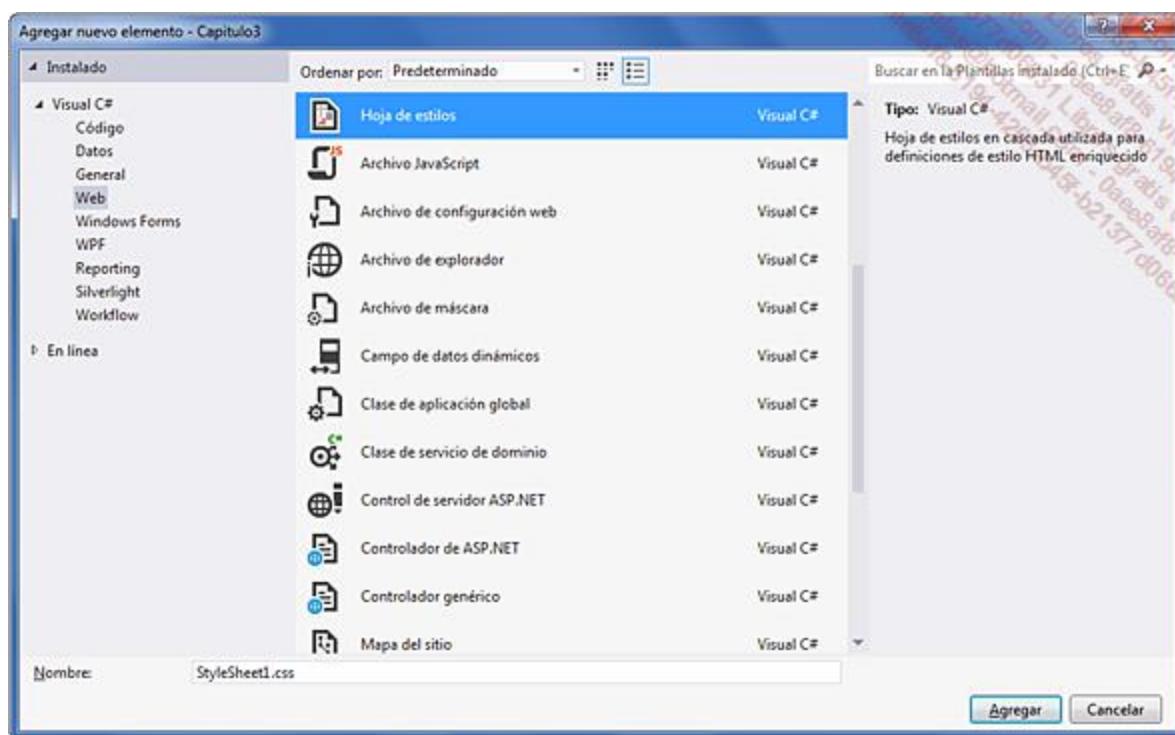
Los sitios web ASP.NET disponen de nuevas herramientas para organizar su presentación.

## 1. Temas y máscaras

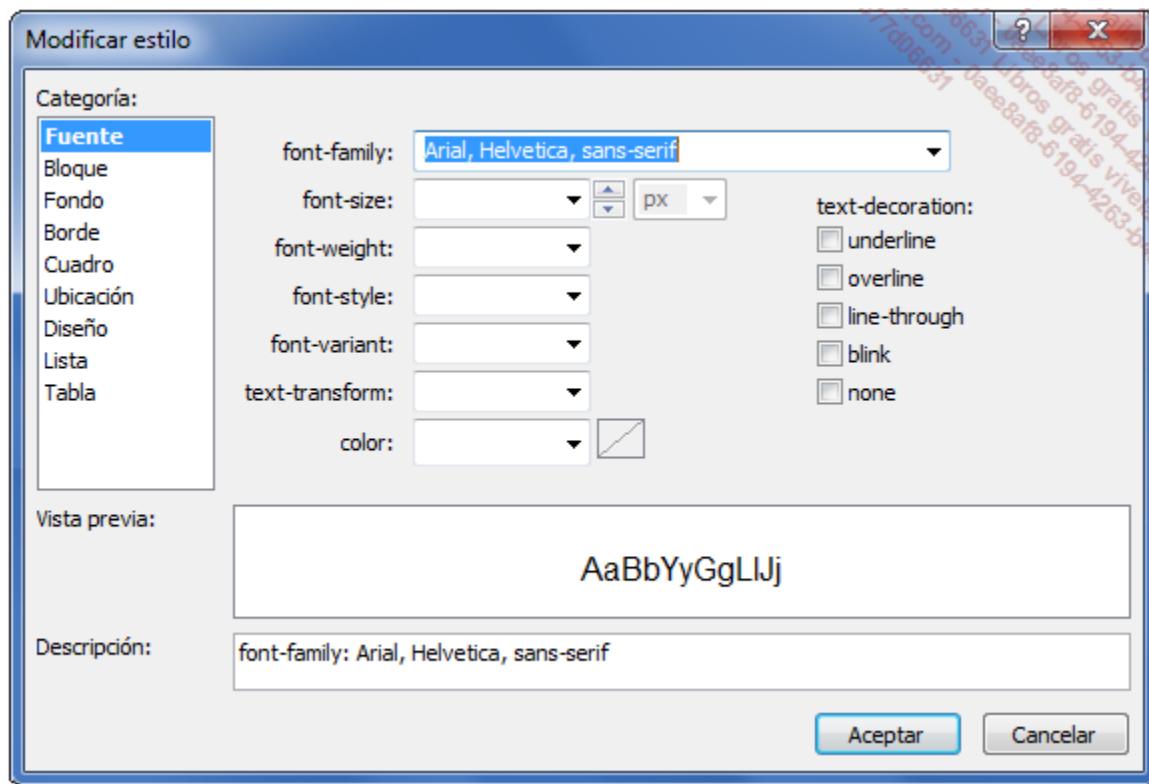
Los temas y las máscaras (skins) son herramientas destinadas a controlar la apariencia de los elementos de una página ASPX. Para ello, se apoyan en los estilos CSS (*Cascaded Style Sheet*).

### a. Hojas de estilo CSS

Los sitios ASP.NET soportan, de forma natural, hojas de estilo CSS. Estas hojas de estilo homogeneizan la apariencia de las páginas web definiendo clases de estilo personalizadas o redefiniendo las etiquetas HTML. A este respecto, Visual Studio conoce el formato de archivo css y proporciona un editor de estilo CSS completo.



Si bien la función Intellisense, que nos ayuda en la escritura de código CSS, es muy útil, los menús contextuales **Agregar una regla de estilo** y **Compilar estilo** facilitan la creación de reglas de estilo mediante una ventana emergente temática:



Para asociar una hoja de estilos CSS a una página ASPX, basta con deslizarla desde el explorador de soluciones hasta el formulario. Esto tiene como efecto la inserción de la etiqueta HTML <link>:

```
<head runat="server">
    <title>Inicio</title>
    <link href="verano.css" rel="stylesheet" type="text/css" />
</head>
```

## b. Temas

Los temas definen juegos de hojas de estilo, imágenes y skins que pueden agregarse fácilmente a las páginas web ASPX. Los temas se almacenan en una carpeta especial **ASP.NET, App\_Themes**:

### Creación de un tema

Un tema tiene el nombre de la carpeta en la que está contenido. En nuestro caso, se trata de tema **verano**. Es mejor no utilizar acentos para escribir el nombre de un tema, pues se trata de una carpeta, y las URL no permiten acentos en su definición.

### Aplicación de un tema

Nuestro tema contiene la hoja de estilo verano.css definida en el párrafo anterior. Para aplicar el tema - y, por tanto, la hoja de estilo - a la página Default.aspx, no es necesario utilizar ninguna etiqueta <link>. Por el contrario, el atributo **Theme** de la directiva @Page debe tener el valor "verano":

```

<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default"
Theme="verano"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Inicio</title>
</head>
<body>
    <form id="form1" runat="server">
    </form>
</body>
</html>

```

En tiempo de ejecución, la etiqueta del encabezado `<head>` contendrá una referencia a la hoja de estilos verano.css. Esto explica por qué el elemento `<head>` está marcado como `runat="server"`; su representación se personaliza en función del tema aplicado.

### **Construcción de un tema**

Un tema contiene hojas de estilo CSS, archivos .skin e imágenes. El nombre de los archivos .css no tiene ninguna importancia, el framework los asociará todos a las páginas a las que se aplique el tema.

Como el orden de aplicación de las hojas de estilo no puede decidirlo el programador, hay que prestar atención a que la ejecución de las reglas "en cascada" no esté repartida en varios archivos .css.

### **Aplicación a todas las páginas**

Es posible aplicar un tema al conjunto de páginas de un sitio definiendo el elemento `<pages>` del archivo Web.config:

```

<system.web>
    <pages theme="invierno"/>
</system.web>

```

Esto evita tener que repetir el tema="invierno" en la directiva `@Page` de cada Web Form. Está, también, previsto que una página pueda derogar la aplicación de un tema definido por el Web.config. Para ello debe marcar el atributo `EnableTheming` a `false`:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default"
EnableTheming="false"
%>
```

## Aplicación dinámica de un tema a una página

El modelo de programación de una página ASPX prevé cambiar dinámicamente el tema. Para ello, el atributo Theme de la directiva @Page no debe estar informado. A continuación, hay que definir el tema mediante el evento PreInit (Init interviene demasiado tarde). Es bastante delicado controlar la aplicación de un tema en función de los controles web presentes en la página. En efecto, estos controles todavía no estás listos en el momento de la ejecución del método OnPreInit.

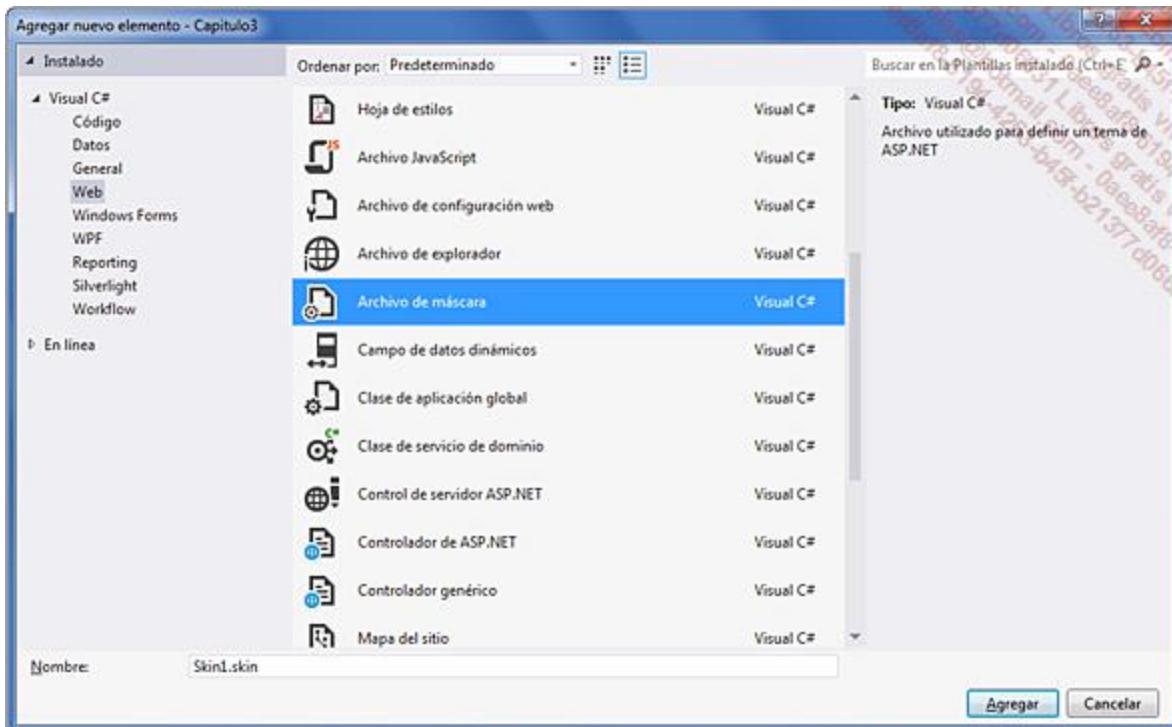
```
protected override void OnPreInit(EventArgs e)
{
    base.OnInit(e);
    Theme = "verano"; // nombre del tema, escrito literalmente
}
```

### c. Máscaras

Las máscaras definen juegos de propiedades textuales (colores, imágenes, estilos) que se aplican a los controles web que utiliza un sitio ASP.NET.

Estos juegos de propiedades se definen en uno o varios archivos con la extensión .skin y se ubican en una carpeta-tema. Visual Studio no proporciona ayuda en la inserción de archivos .skin. Para evitar los errores, es preferible, hasta que se tenga soltura, utilizar el copiar-pegar.

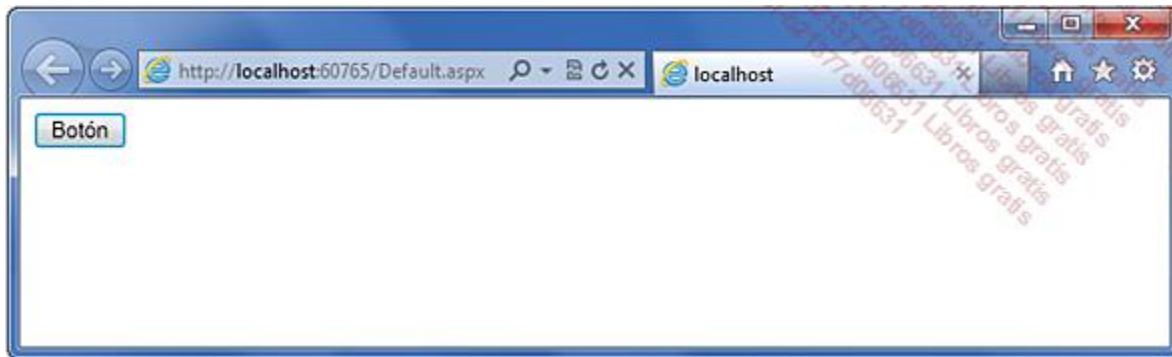
Como con la hoja de estilos CSS, el nombre del archivo .skin no importa. El framework aplicará todos los archivos .skin de un tema.



Consideremos el siguiente botón, que figura en la página Default.aspx:

```
<form id="form1" runat="server">
    <asp:Button ID="boton1" runat="server" Text="Botón" />
</form>
```

En la publicación, la representación de dicho botón consiste en una etiqueta `<input type="submit">` de apariencia clásica y color gris.

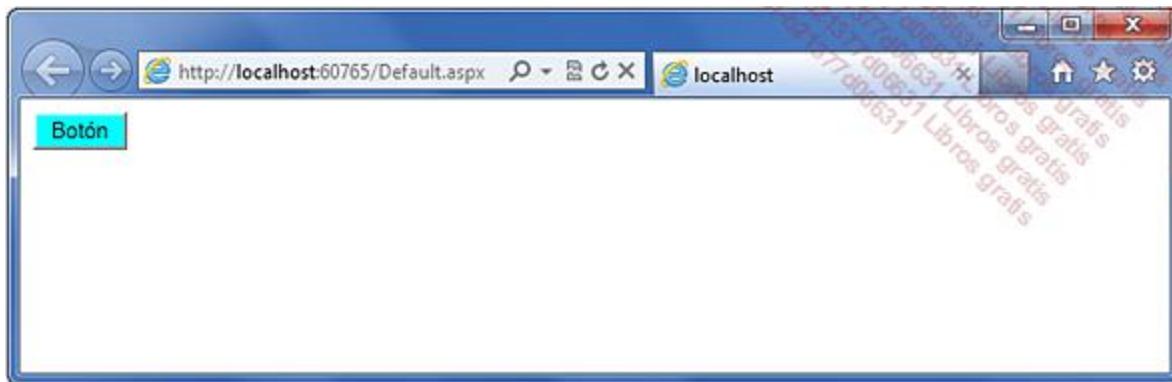


### Aplicación de un skin

Redefiniendo la etiqueta `<asp:Button>` en el archivo `.skin`, es posible afectar al conjunto de botones del sitio web:

```
<asp:Button runat="server" BackColor="Aqua" />
```

En su publicación, un estilo CSS modifica la representación de la etiqueta HTML `<input type="submit">`:



```
<input type="submit" name="boton1" value="Botón"  
id=" boton1" style="background-color:Aqua;" />
```

Las máscaras constituyen una forma sencilla y cómoda de aplicar estilos CSS a los controles web.

### Máscaras alternativas

Un tema puede declarar varias versiones de una máscara de control. Cada versión se caracteriza por su SkinID, referenciado por el control web:

```
<%-- Versión por defecto para todos los botones (guardar) --%>  
<asp:Button runat="server" />  
  
<%-- Versión específica --%>  
  
<asp:Button runat="server" SkinID="agosto" BackColor="Aqua" />
```

Si el control botón no referencia el SkinID, se aplica la versión por defecto. En caso contrario, se aplica el color de fondo azul.

```
<asp:Button ID="boton1" runat="server" SkinID="agosto" Text="Botón" />
```

### Derogación de la aplicación de una máscara

Un control web puede derogar la aplicación de una máscara declarando su propiedad EnableTheming a false:

```
<asp:Button ID="boton1" runat="server" SkinID="agosto" Text="Botón"  
EnableTheming="false" />
```

Las modificaciones definidas por las reglas del archivo .skin no se tendrán en cuenta, incluso si se define la propiedad SkinID.

## 2. Componentes de usuario .ascx

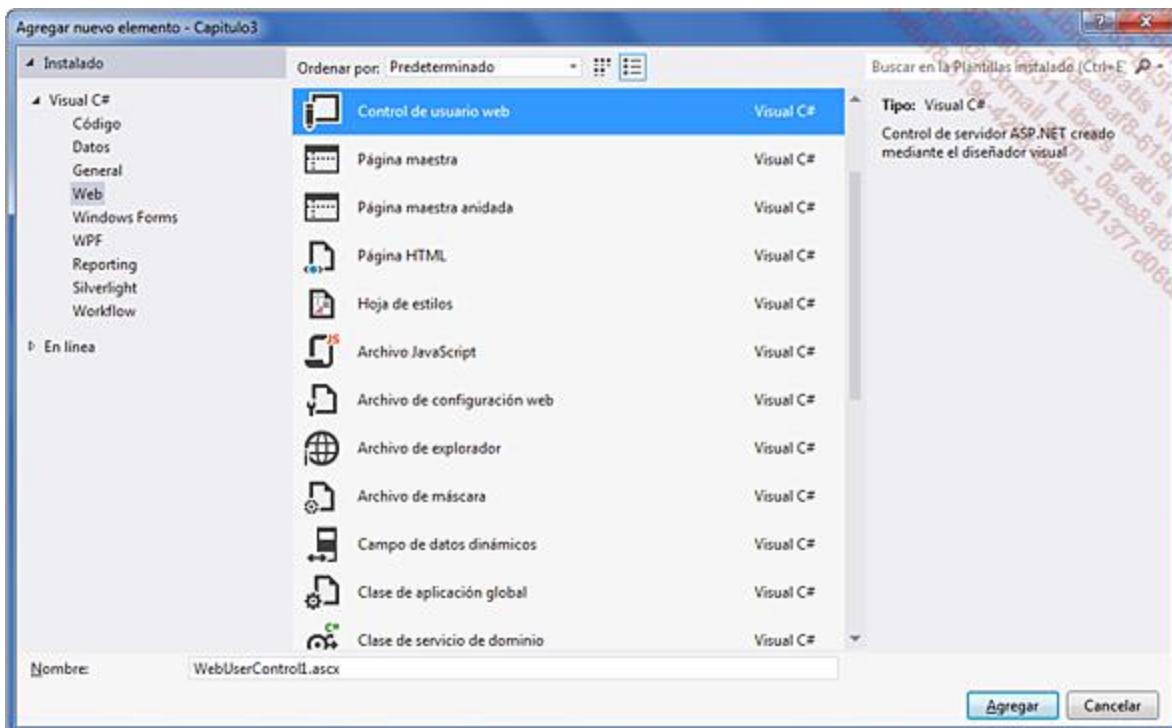
Los componentes de usuario representan sub-formularios, fragmentos de páginas. En relación a las páginas maestras, introducidas en la versión de ASP.NET 2.0, los componentes .ascx tienen características que los hacen, todavía, indispensables en el desarrollo de sitios web:

- Se integran en las páginas como cualquier componente, y se comunican con las páginas según los procedimientos vistos anteriormente.
- Los componentes .ascx autorizan la caché parcial de la salida HTML.

Desde un punto de vista de representación de la página, no existe gran diferencia entre un componente .ascx y una página maestra. Pero, desde un punto de vista de integración y de programación, las diferencias son importantes.

### a. Crear un componente de usuario

Visual Studio provee un asistente para crear componentes de usuario. Técnicamente, es posible partir de una página .aspx para crear un control .ascx, aunque esta operación puede resultar delicada.



El código HTML de un componente de usuario contiene una directiva @Control y un conjunto de etiquetas. Pero, a diferencia de una página ASPX, no puede contener ni <html> ni <body> ni <form>.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="menu.ascx.cs" Inherits="menu" %>

```

```

        NavigateUrl="default.aspx" runat="server">></asp:HyperLink></td>
        <td><asp:HyperLink ID="12" Text="Productos"
        NavigateUrl="productos.aspx" runat="server"></asp:HyperLink></td>
        <td><asp:HyperLink ID="13" Text="Servicios"
        NavigateUrl="servicios.aspx" runat="server"></asp:HyperLink></td>
    </tr>
</table>

```

En efecto, este componente se inserta en páginas ASPX que ya poseen etiquetas que forman la estructura de la página - <head>, <body> y <form> -, y HTML no prevé que aparezcan más de una vez.

### b. Utilizar un componente de usuario

Antes de instanciarlo y ubicarlo en una página ASPX, el componente de usuario debe registrarse mediante la directiva @Register. Esta directiva designa la ruta del archivo .ascx que debe formar parte del mismo proyecto web que la página, y especifica un par (espacio de nombres, etiqueta) que se utiliza para instanciar el componente en la página:

```
<%@ Register Src="menu.ascx" TagName="menu" TagPrefix="uc1" %>
```

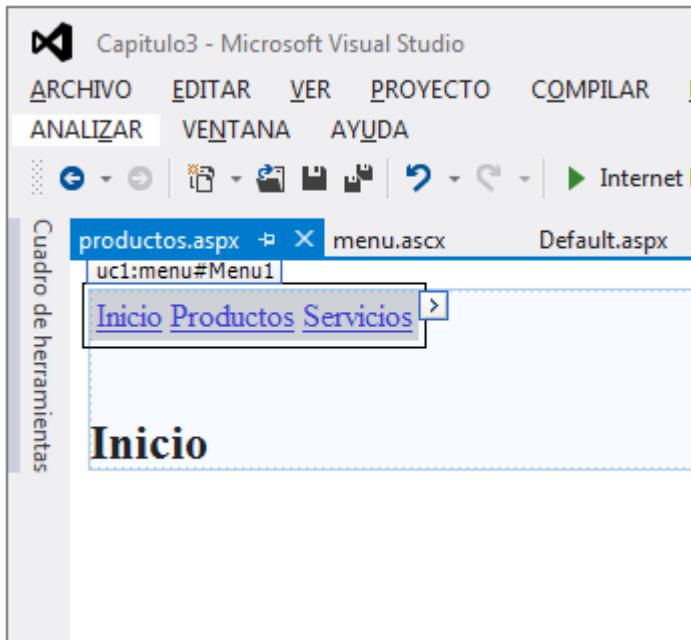
La página ASPX empleará tantas instancias del componente como desee, cada una de ellas con un identificador único. Esta característica distingue a los componentes de las páginas maestras.

```

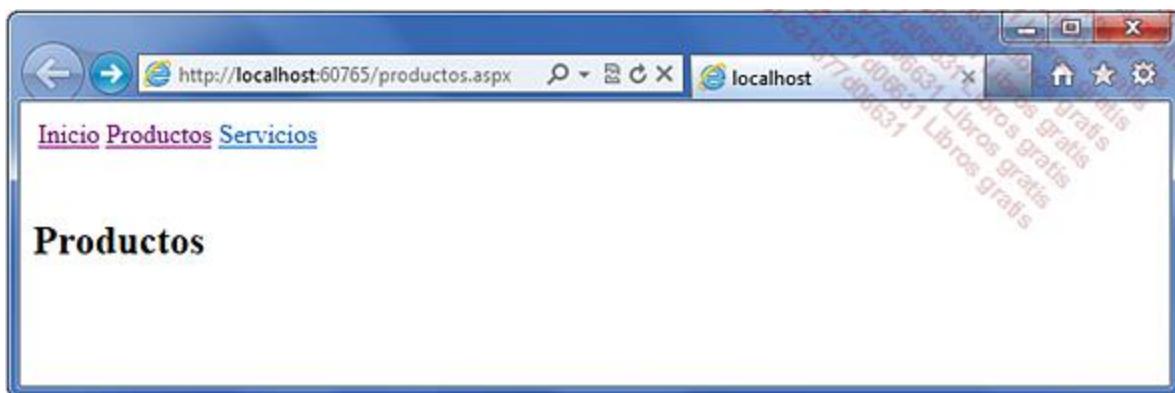
<form id="form1" runat="server">
    <div>
        <uc1:menu ID="Menu1" runat="server" />
    <br />
    <h2>Inicio</h2>
    </div>
</form>

```

A partir de la versión 2005, Visual Studio es capaz de representar el componente de usuario en modo Diseño:



Sólo queda probar la página que contiene el componente de usuario:



### c. Agregar propiedades y eventos

Los componentes de usuario .ascx soportan un modelo de programación completo. Disponen, en particular, de un modelo separado con una parte HTML y otra parte escrita en C#.

```
public partial class menu : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

La clase menu hereda de **UserControl** (en vez de Page) pero posee también un método **Page\_Load** que se invocará justo tras el método **Page\_Load** de la página que contenga la instancia del componente.

## Gestionar los eventos de los controles

Es posible crear controladores de eventos para los controles que componen el fragmento .ascx. Para ilustrar este mecanismo, agregaremos al control menú.ascx un Panel y un LinkButton:

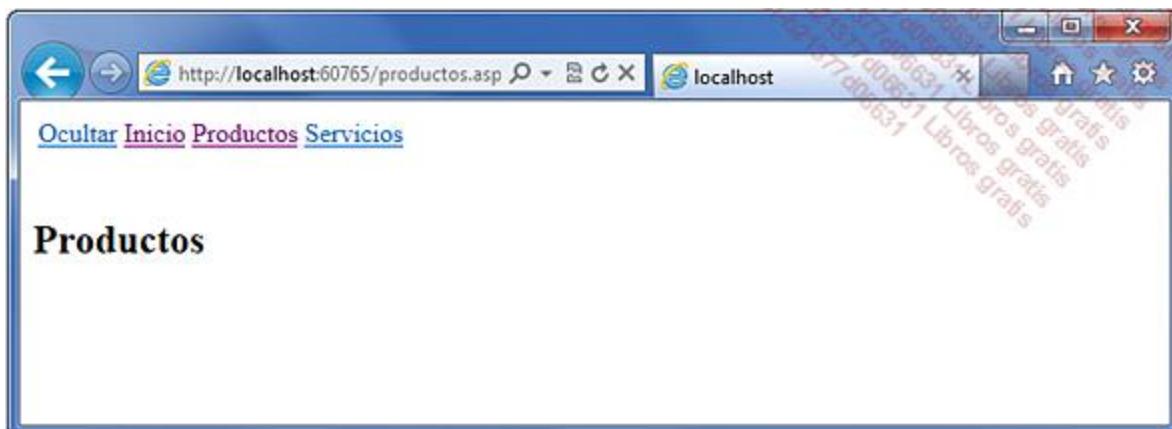
```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile=
"menu.ascx.cs" Inherits="menu" %>


```

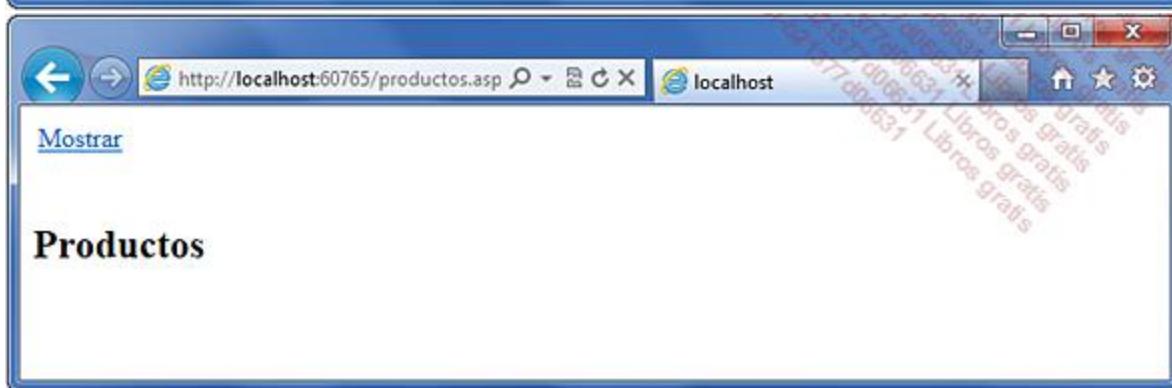
Al hacer doble clic en LinkButton se declara el siguiente método:

```
protected void cmd_ocultar_Click(object sender, EventArgs e)
{
    mp.Visible = !mp.Visible;
    cmd_aff.Text = (mp.Visible ? "Ocultar" : "Mostrar");
}
```

Podemos verificar que el menú se oculta y, a continuación, se muestra:



## Productos



## Productos

### Agregar propiedades

De cara a la página que alberga una instancia del componente ASCX, estos controles no están accesibles, pues se definen como elementos privados (private). Lo adecuado es, en este caso, definir propiedades públicas para no exponer la estructura interna del componente. De este modo, el desarrollador podrá hacer evolucionar esta estructura y modificar la ergonomía, sin comprometer la integración de las páginas que explotan el componente.

```
public bool visible
{
    get { return mp.Visible; }
    set { mp.Visible = value; }
}

private string url;
```

```
protected void Page_Load(object sender, EventArgs e)
{
    url = Request.Url.Segments[ Request.Url.Segments.Length-1 ];
    switch (url.ToLower())
    {
        default:
            ll.Text = "Inicio";
    }
}
```

```

        12.Text = "Productos";
        13.Text = "Servicios";
        break;

    case "default.aspx":
        11.Text = "[ Inicio ]";
        12.Text = "Productos";
        13.Text = "Servicios";
        break;

    case "productos.aspx":
        11.Text = "Inicio";
        12.Text = "[ Productos ]";
        13.Text = "Servicios";
        break;

    case "servicios.aspx":
        11.Text = "Inicio";
        12.Text = "Productos";
        13.Text = "[ Servicios ]";
        break;
    }
}

public string Url
{
    get { return url; }
}

```

Una página que albergue el componente accederá, ahora, a sus propiedades públicas:

```

public partial class productos : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        lbl_url.Text = Menu1.Url;
    }
}

```

### **Desencadenar eventos para las páginas**

Como el componente ASCX sigue las reglas de funcionamiento de los demás controles de página, no siempre es evidente estar trabajando en el momento temporal que nos conviene. De este modo, el código no ofrece ningún resultado: el método Page\_Load de las páginas se ejecuta antes que el del control Menu, y la propiedad URL no se asignará correctamente.

Es preferible, en este caso, instalar un evento que indique la presencia de datos válidos en las propiedades:

```
public delegate void del_url_valida(string url);
public event del_url_valida OnUrlValida;

protected void Page_Load(object sender, EventArgs e)
{
    url = Request.Url.Segments[ Request.Url.Segments.Length-1 ];
    if (OnUrlValida != null)
        OnUrlValida(url); // indica la presencia de una URL válida

    #region "Switch(url)"
    #endregion
}
```

Esto le da una nueva vuelta de tuerca a la página productos.aspx:

```
public partial class productos : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Menu1.OnUrlValida += new menu.del_url_valida( Menu1_OnUrlValida );
    }
}
```

```
void Menu1_OnUrlValida(string url)
{
    lbl_url.Text = url;

    // la siguiente línea produce el mismo resultado
    lbl_url.Text = Menu1.Url;
}
```

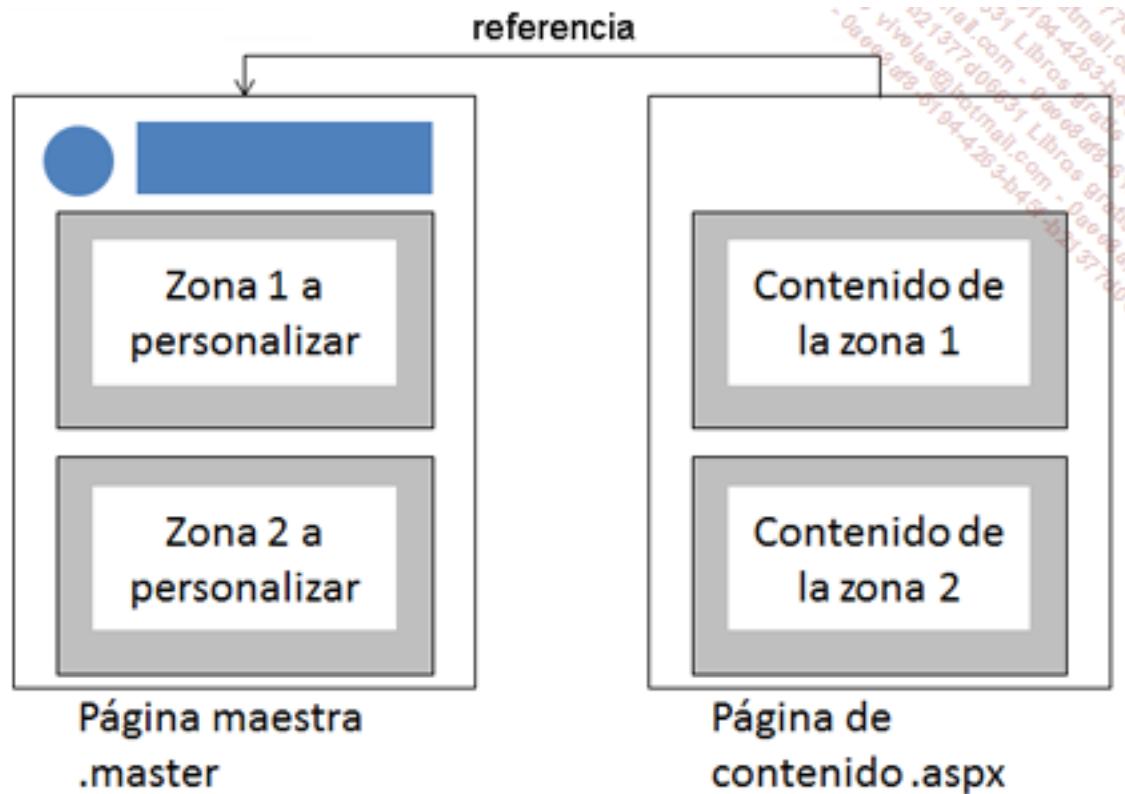
### 3. Las páginas maestras

En el enfoque anterior, la página integraba componentes de usuario .ascx para complementar su representación. Este enfoque era relativamente coherente en 2002, tras la aparición de

ASP.NET 1.0. Después, la concepción y el diseño de un sitio web han cambiado bastante. Los sitios los explotan equipos muy variados, y los ciclos de desarrollo se han visto reducidos. En estas condiciones, las páginas maestras (master page) retoman, con éxito, el relevo de los controles de usuario. Estos últimos guardan algunas funcionalidades, como acabamos de ver.

#### a. Crear una página maestra

La técnica de las master pages utiliza páginas maestras (.master) y páginas de contenido (.aspx).

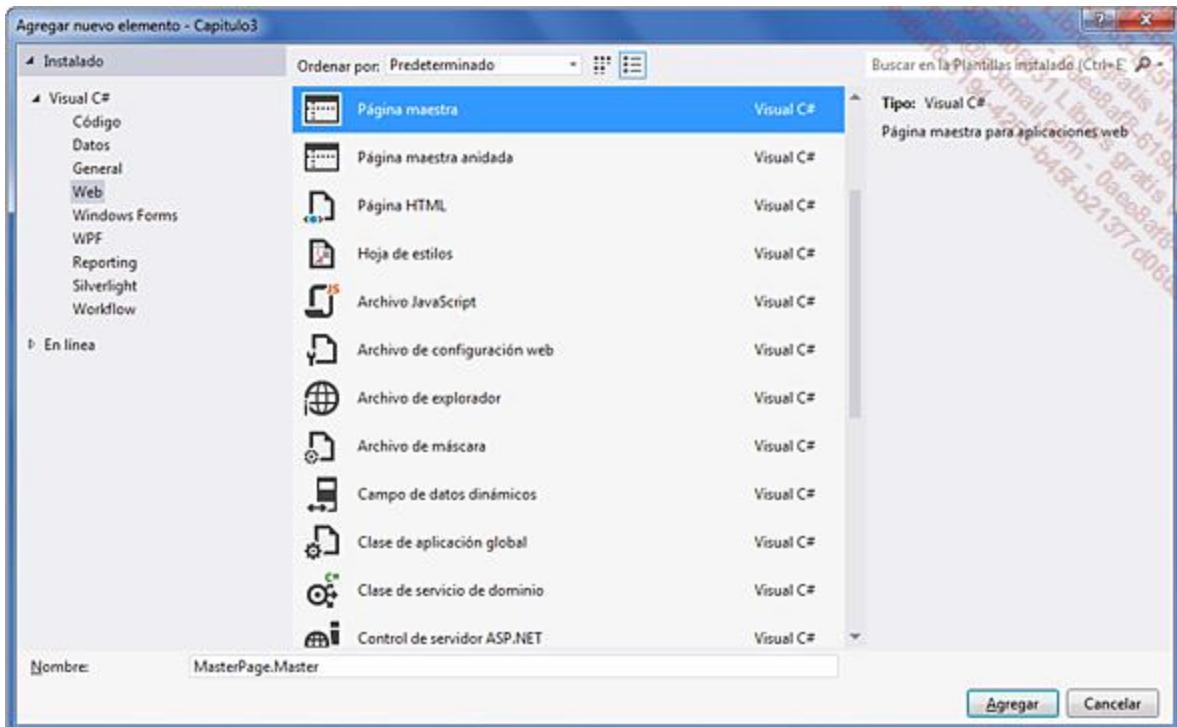


La página maestra se concibe como una página .aspx más; contiene etiquetas HTML, controles web y código de control de eventos. Define, además, zonas a personalizar mediante páginas de contenido.

Las páginas de contenido referencian a una página maestra. En su zona HTML, no contienen tags <html>, <body> ni <form>, sino únicamente etiquetas que referencian y delimitan las zonas que se desea personalizar.

Las páginas .master no pueden invocarse desde una URL. Sólo las páginas .aspx pueden responder a solicitudes del navegador.

Visual Studio provee un asistente que permite crear una página maestra:



Tras validar el nombre de la página, genera una matriz que contiene una primera zona a personalizar:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
```

```
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
            </asp:contentplaceholder>
        </div>
    </form>
</body>
</html>
```

La primera zona destinada a páginas de contenido se define mediante la etiqueta `<asp:contentplaceholder>`. Una página maestra puede contener varias etiquetas de este tipo, situadas en distintos lugares (en una celda de una tabla, en una etiqueta `<div>...`), e identificadas mediante su atributo **id**.

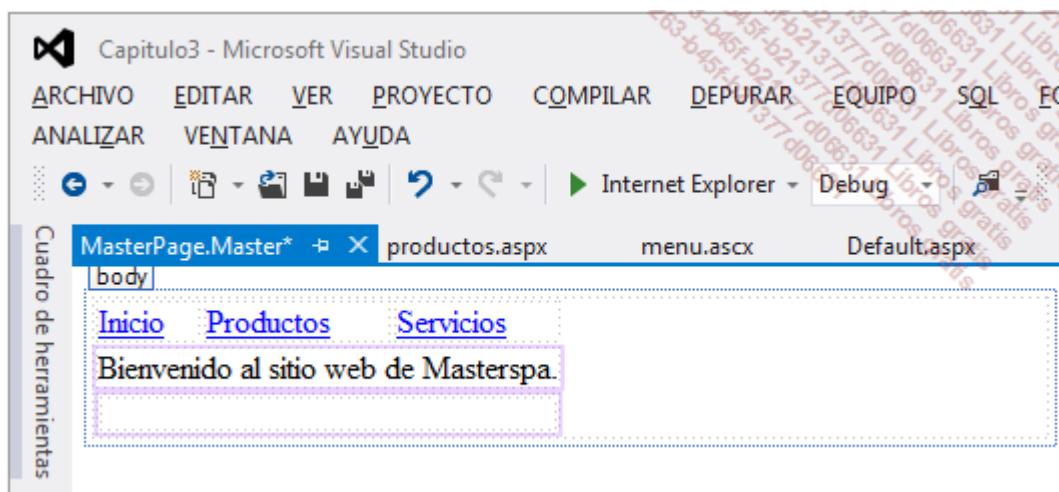
```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Masterspa, el lugar del bienestar</title>
</head>
<body>
    <form id="form1" runat="server">
        <table>
            <tr>
                <td><a href="inicio.aspx">Inicio</a></td>
                <td><a href="productos.aspx">Productos</a></td>
                <td><a href="servicios.aspx">Servicios</a></td>
            </tr>
            <tr>
                <td colspan="3">
                    <asp:contentplaceholder id="encabezado" runat="server">
                        Bienvenido al sitio web de Masterspa.
                    </asp:contentplaceholder>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

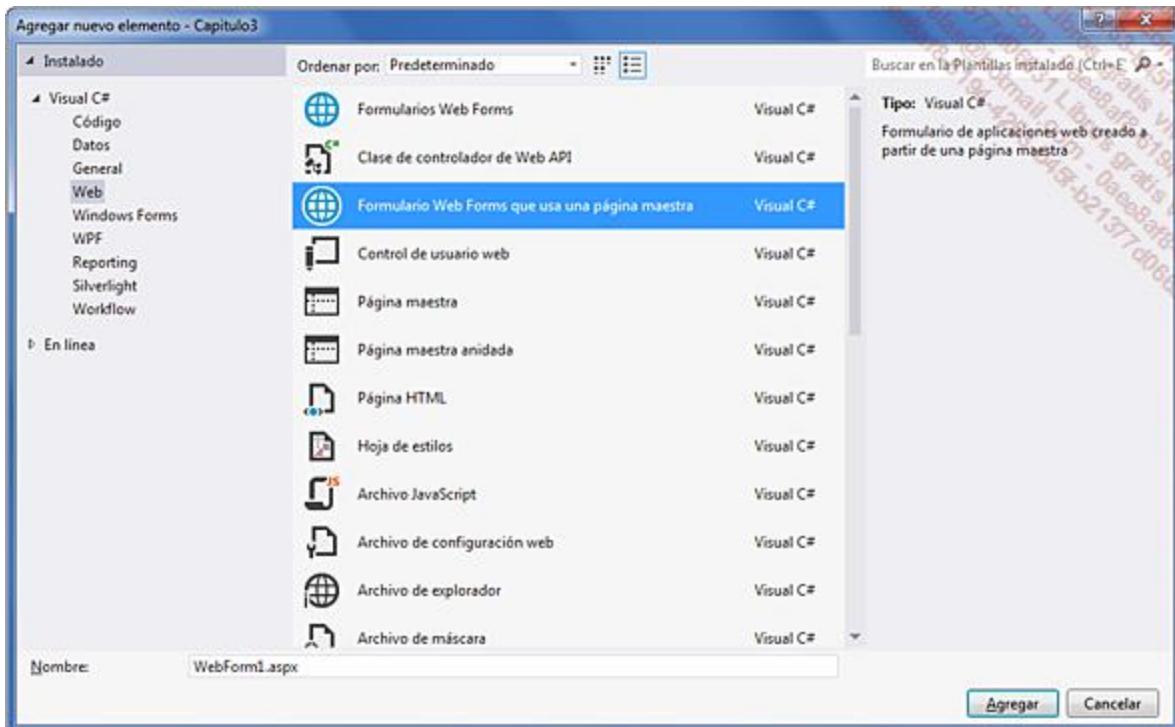
```
</asp:contentplaceholder>
</td>
</tr>
<tr>
    <td colspan="3">
        <asp:contentplaceholder id="principal" runat="server">
            </asp:contentplaceholder>
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

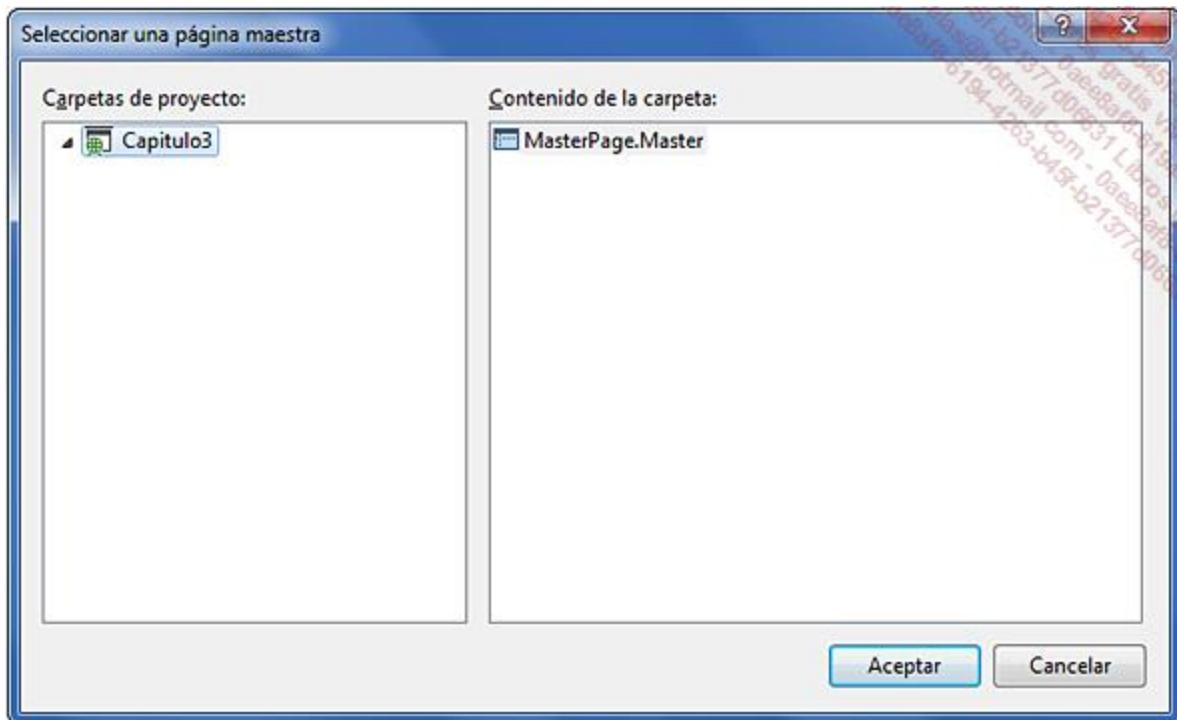
Como con los componentes de usuario, el programador tiene acceso al modo Diseño para editar las páginas maestras.



### b. Crear una página de contenido

Desde el explorador de soluciones, el menú contextual **Agregar una página de contenido** es el primer medio para crear una página de contenido a partir de una página maestra. Como Visual Studio llama a las páginas, por defecto, Default1.aspx, es preferible partir con un nombre definido de la página y utilizar el asistente **Agregar nuevo elemento** activando la opción **Seleccionar la página maestra** para el ítem Web Form.





Si bien la zona HTML está mucho más restringida que en una página ASPX normal, el modelo de programación es idéntico.

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
AutoEventWireup="true" CodeFile="inicio.aspx.cs" Inherits="inicio"
```

```
Title="Untitled Page" %>

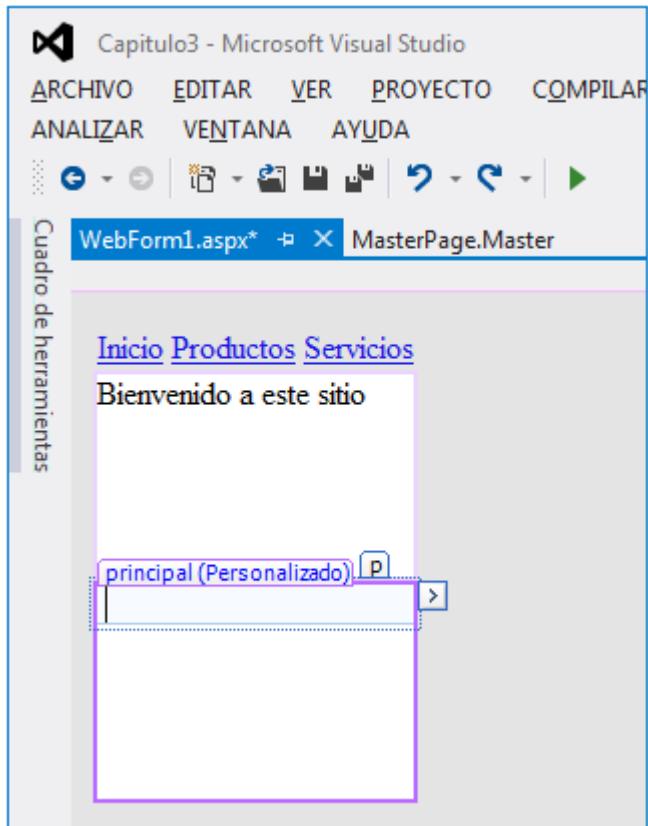
<asp:Content ID="Content1" ContentPlaceHolderID="encabezado" Runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="principal" Runat=
"Server">

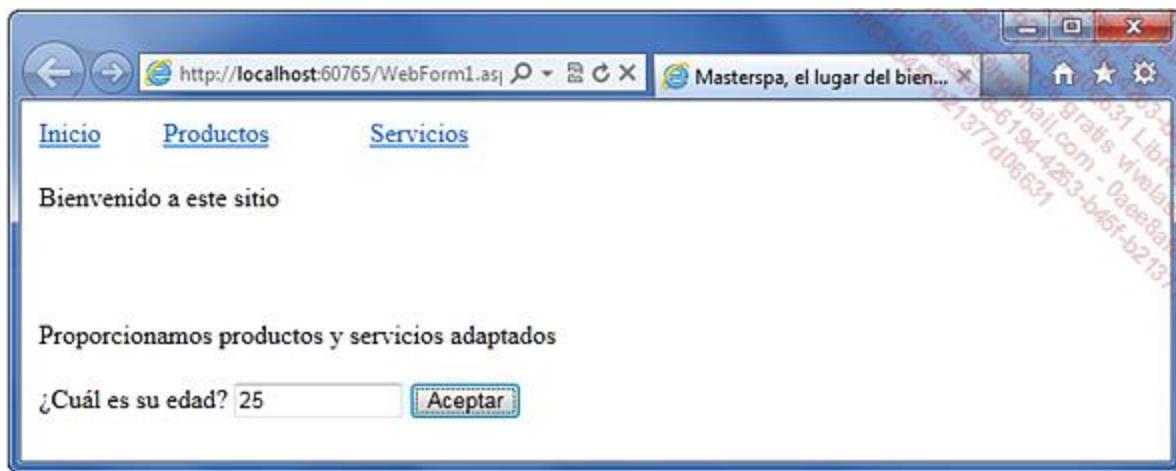
</asp:Content>
```

La página de contenido puede contener tantos paneles `<asp:content>` como referencia su página maestra. Cuando falta un panel, se muestra el contenido HTML de la página maestra. Cuando se define el panel en la página de contenido, incluso vacío, sustituye al código HTML de la página maestra.

Si pasamos al modo Diseño, el programador sólo puede modificar aquellas zonas correspondientes a los paneles de contenido. El resto se queda en gris, pues se trata de código HTML de la página maestra.



Tras la publicación de la página, no existe ninguna señal que indique que la página es resultado de la fusión de una página maestra y otra de contenido:



Destaquemos, también, que una página de contenido puede reemplazar el título definido por la etiqueta <title> en la página maestra mediante el atributo Title de la directiva @Page:

```
<%@ Page Language="C#"
    MasterPageFile("~/MasterPage.master"
    AutoEventWireup="true" CodeFile="inicio.aspx.cs"
    Inherits="inicio"
    Title="Bienvenida Masterspa" %>
```

### c. Programar páginas maestras y páginas de contenido

Desde un punto de vista del diseño, la asociación entre una página .master y una página de contenido es idéntica a la que existe entre una página .aspx y un control .ascx; los controles web presentes de una parte y de otra los gestiona cada una de su lado.

Existe, no obstante, el caso en el que una página de contenido .aspx debe acceder a las propiedades de una página .master.

```
public partial class ModeloEncabezado : System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    public string encabezado
    {
        get { return lbl_encabezado.Text; }
        set { lbl_encabezado.Text = value; }
    }
}
```

Si quisieramos acceder a la propiedad encabezado desde la página de contenido comunicado.aspx, la propiedad Master del objeto Page debería modificar su tipo por ModeloEncabezado:

```
public partial class comunicado : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // se requiere modificación de tipo
        ModeloEncabezado maestra = Page.Master as ModeloEncabezado;
        maestra.encabezado = "Para comenzar...";
    }
}
```

Es posible definir indirectamente el tipo de la página maestra mediante la directiva @MasterType y así evitar esta modificación de tipo:

```
<%@ Page Language="C#" MasterPageFile="~/ModeloEncabezado.master"
AutoEventWireup="true" CodeFile="comunicado.aspx.cs"
Inherits="comunicado" Title="Untitled Page" %>
```

```
<%@ MasterType VirtualPath="~/ModeloEncabezado.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
</asp:Content>
```

El código resulta, así, mucho más directo:

```
protected void Page_Load(object sender, EventArgs e)
{
    // sin modificación de tipo
    Master.encabezado = "Para comenzar...";
}
```

#### d. Aplicar dinámicamente una página maestra

El programador puede aplicar dinámicamente una página maestra a una página de contenido definiendo la propiedad **MasterPageFile**.

```
public partial class Default2 : System.Web.UI.Page
{
    protected override void OnPreInit(EventArgs e)
    {
        base.OnPreInit(e);
        Page.MasterPageFile = "Master1024.master";
    }
}
```

De esta forma, la representación de la página permite respetar las capacidades del navegador. El usuario del sitio web realiza una elección al inicio de su navegación (o se recupera su configuración mediante su perfil), y todas las páginas del sitio se muestran sin la barra de navegación ni espacio vacío alrededor del contenido.

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="Master1024.master.cs" Inherits="Master1024" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Untitled Page</title>
```

```
</head>

<body>

    <form id="form1" runat="server">

        <h2>Resolución 1024x768</h2>

        <table border="1" width="1024">

            <tr>

                <td>

                    <asp:contentplaceholder id="ContentPlaceHolder1"
runat="server">

                    </asp:contentplaceholder>

                </td>

            </tr>

        </table>

    </form>

</body>

</html>
```

# Componentes personalizados

Los componentes personalizados aseguran a ASP.NET flexibilidad y versatilidad. El propósito de Microsoft no es producir soluciones específicas, por ello estos componentes abren la vía a un mercado más dinámico.

## 1. Funcionamiento de los componentes personalizados

Muy diferentes a los componentes de usuario .ascx por su propósito, los componentes personalizados se desarrollan también de forma particular. Se diseñan, exclusivamente, mediante instrucciones C# o VB.NET. En contrapartida a este esfuerzo de diseño, el programador controla todo su funcionamiento.

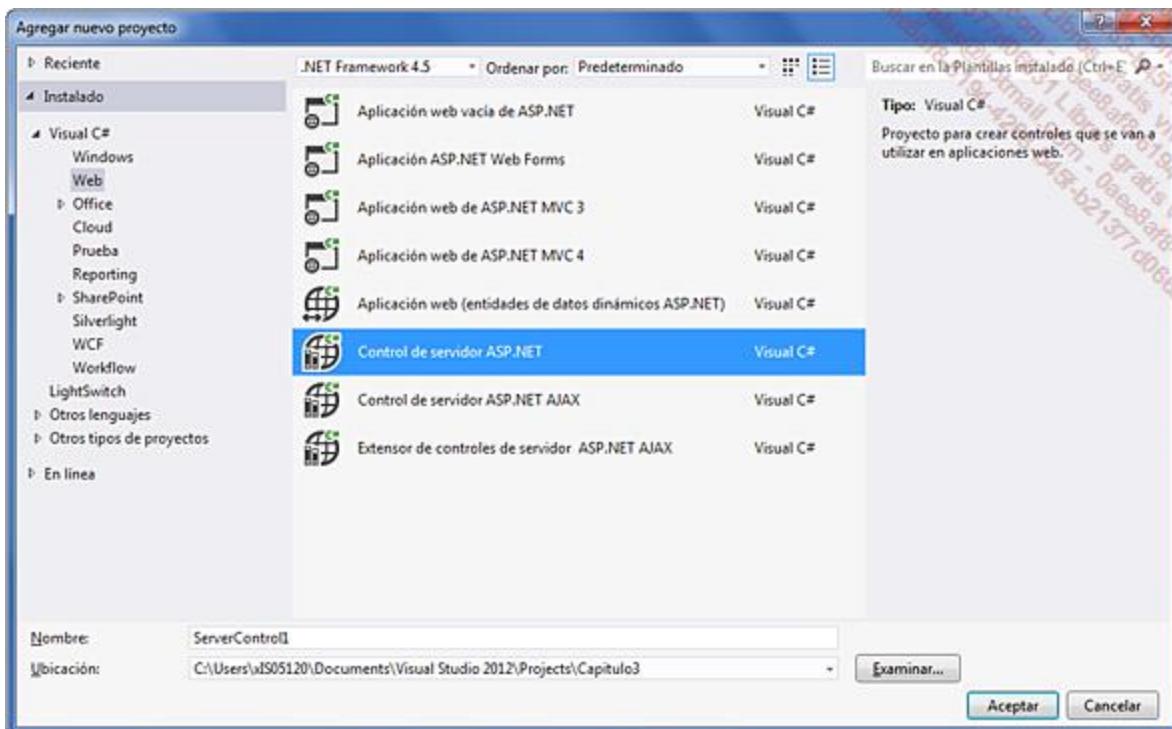
### a. Tipos de componentes personalizados (custom controls)

Con ASP.NET, es posible distinguir entre varios tipos de componentes personalizados. A este respecto, Microsoft se ha esforzado por documentar esta parte, importante, del framework ASP.NET.

Tipo	Hereda de	Aplicación	b.
Control web	WebControl	Define un nuevo tipo de control.	
Control derivado	TextBox...	Especializa el funcionamiento de un control web.	
Control compuesto	CompositeControl	Control similar a los componentes de usuario .ascx que integra varios controles web.	
Control basado en plantilla	TemplateControl	Control que puede contener secuencias de controles web, característica muy práctica para operaciones de Databinding.	

### Creación de una librería de componentes

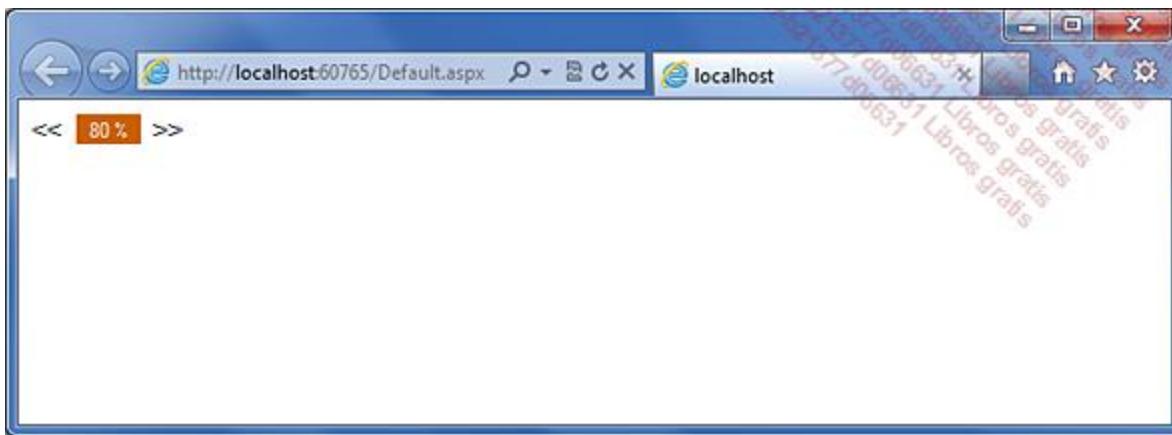
Un componente personalizado puede invocarse desde varios proyectos web. Éste es el motivo por el que se ubica en una librería de componentes. Se trata de un ensamblado .NET que trabaja, en particular, con la referencia System.Web.dll y en el seno de la cual Visual Studio genera el esqueleto de un primer control web personalizado.



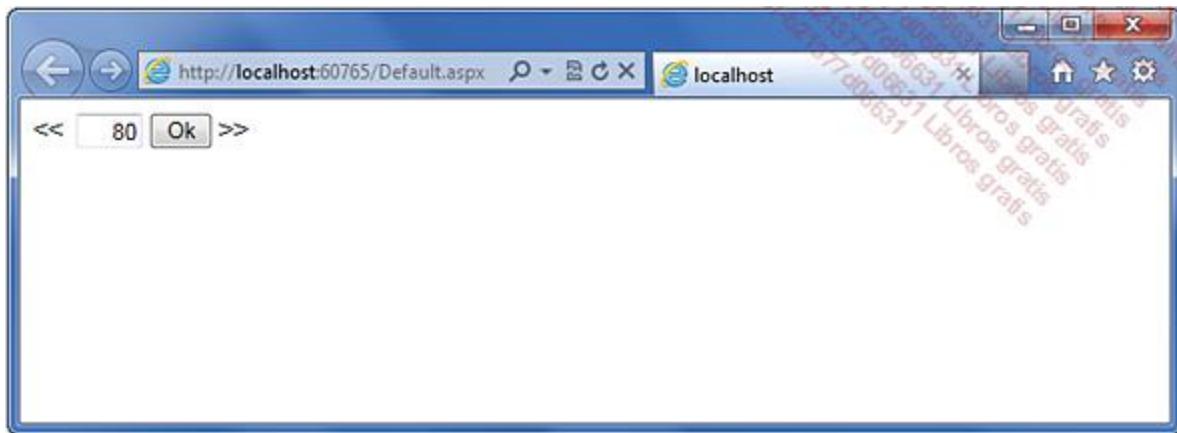
### c. Creación del componente ColoredPad

#### Estructura y funcionamiento

El componente ColoredPad representa un cuadrado cuyo color varía en función de su propiedad Value. Cuando Value = 0, el cuadrado es negro, y para otros valores, el color es un matiz de la propiedad BaseColor.



También se ha previsto un modelo de eventos; el control dispone de botones de desplazamiento << y >> que sirven para aumentar (o, respectivamente, disminuir) la propiedad Value según un valor especificado en la propiedad Step. Además, haciendo clic en el cuadrado de color, el control cambia de aspecto y pasa al modo edición. El usuario define, así, directamente el valor de la propiedad Value. A continuación, se desencadena la acción OnValue Changed se desencadena.



La implementación del componente ColoredPad se basa en una clase. Los atributos C# que se informan entre [ ] y ] se utilizan en Visual Studio para instanciar el control y gestionar sus eventos.

```
[DefaultProperty("Value")]
[DefaultEvent("OnValueChanged")]
[ToolboxData("<{0}:ColoredPad runat=server>
</{0}:ColoredPad>")]
public class ColoredPad :
WebControl, IPostBackDataHandler, IPostBackEventHandler
{
    #region OnInit
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);

        // indica que el componente tiene que decodificar
        // los datos del postback
        Page.RegisterRequiresPostBack(this);
    }
    #endregion
}
```

El atributo **ToolboxData** indica, en particular, qué etiqueta está asociada con el componente de una página. El formateador **{0}** se corresponde con el espacio de nombres **cc1** como veremos más adelante.

La clase ColoredPad hereda de **WebControl**, pues el componente no se corresponde con ninguna implementación existente. Se trata, por tanto, de un control nuevo. Las interfaces **IPostBackDataHandler** e **IPostBackEventHandler** definen los métodos que manejan el postback y los eventos. En el mismo sentido, el método sobrecargado **OnInit()** invoca **Page.RegisterRequiresPostBack()**.

## Propiedades y persistencia

Las propiedades de un control web se corresponden con los atributos de la etiqueta XHTML. Recordemos que un componente es un campo de una página y que las instancias se destruyen tras la ejecución de la página. Por ello, los campos de la página, y también sus componentes, desaparecen.

Para superar esta dificultad, la página pone a nuestra disposición su colección de persistencia ViewState. Las propiedades de los controles la utilizan para recordar sus valores entre un postback y el siguiente.

```
#region Propiedad Valor
[Category("Appearance")]
[DefaultValue("0")]
[Description("Valor entre 0 y 100")]
public double Value
{
    get
    {
        if (ViewState["Value"] != null)
            return (double)ViewState["Value"];
        else
            return 0;
    }
    set
    {
        ViewState["Value"] = value;
    }
}
#endregion
```

Las propiedades están marcadas mediante atributos que son útiles en Visual Studio para exponerlos mediante el control PropertyGrid presente en la barra de propiedades.

El control PropertyGrid es un control estándar .NET que utilizan los desarrolladores en sus aplicaciones Winform.

## Eventos

La derivación de WebControl implica la herencia de sus eventos: Init, Load, PreRender, Render, Unload y Dispose. Es habitual completar esta lista mediante otros eventos de interés en la aplicación. Estos eventos no respetan, obligatoriamente, el formato recomendado por el delegado System.EventHandler, también es preciso definir los suyos específicos:

```
public delegate void del_value_changed(double new_value);
public event del_value_changed OnValueChanged;
```

```

public void Up()
{
    Value += Step;
    if (Value > 100)
        Value = 100;

    if (OnValueChanged != null)
        OnValueChanged(Value);
}

```

## Representación

La representación gráfica consiste en crear una secuencia de HTML, JavaScript y estilos CSS. Este código reemplazará a la etiqueta XHMTL del flujo que se devuelve al navegador.

Existen varios métodos para organizar esta representación, pasando el constructor, y mediante métodos auxiliares. En nuestro caso vamos a dar preferencia a la sobrecarga del método `Render()`, puesto que resulta más explícito.

```

#region Representación HTML

protected override void Render(HtmlTextWriter output)
{
    // calcula el color
    Color c;

    double r = BaseColor.R;
    double v = BaseColor.G;
    double b = BaseColor.B;
    r = r * Value / 100;
    v = v * Value / 100;
    b = b * Value / 100;

    int wc = ((int)r) << 16;

    wc |= ((int)v) << 8;
    wc |= (int)b;
    c = Color.FromArgb(wc);

    // forma una etiqueta <span>
    output.WriteBeginTag("span");
    output.WriteAttribute("name", this.ID);
    output.Write(HtmlTextWriter.TagRightChar);
}

```

```

// disminuir

if (DisplayUpDown)
{
    string l1 = "javascript:"+Page.GetPostBackEventReference(this,
"down");

    output.WriteBeginTag("a");
    output.WriteAttribute("href", l1);
    output.WriteAttribute("style", "color: black;
background-color:white; text-decoration: none; font-weight: bold;");
    output.Write( HtmlTextWriter.TagRightChar);
    output.Write( "&nbsp;&lt;&lt;&nbsp;");
    output.WriteEndTag("a");
}

// escribe el valor formateado

if (Edit)
{
    // campo de texto que permite modificar el valor
    output.WriteBeginTag("input");
    output.WriteAttribute("type", "text");
    output.WriteAttribute("name", this.ID + "_value");
    output.WriteAttribute("size", "3");
    output.WriteAttribute("value", string.Format("{0}", Value));
    output.Write( HtmlTextWriter.TagRightChar);

    //botón de validación
    output.WriteBeginTag("input");
}

```

```

        output.WriteAttribute("type", "button");
        output.WriteAttribute("name", this.ID+_valid");
        output.WriteAttribute("value", "ok");
        output.WriteAttribute("onclick",
Page.GetPostBackEventReference(this,"valid"));

        output.Write( HtmlTextWriter.TagRightChar);
    }

    else
    {
        // haciendo clic pasamos al modo de edición
    }
}

```

```

        string l1 = "javascript:";

+ Page.GetPostBackEventReference(this, "edit");

        output.WriteBeginTag("a");

        output.WriteAttribute("href", l1);

        output.WriteAttribute("style", "text-decoration: none; color: " +
+ ColorTranslator.ToHtml(TextColor));

        output.Write( HtmlTextWriter.TagRightChar);

        output.WriteBeginTag("span");

        output.WriteAttribute("style",
            string.Format( "background-color: {0}; color: {1}",
ColorTranslator.ToHtml(c), ColorTranslator.ToHtml(TextColor)));

        output.WriteAttribute("name", this.ID);

        output.Write( HtmlTextWriter.TagRightChar);

        if (DisplayValue)
        {
            if (Format != string.Empty)
                output.Write( string.Format(Format, Value));
            else
                output.Write(Value);
        }
        else
            output.Write(" &nbsp;&nbsp;");
    }
}

```

```

        output.WriteEndTag("span");

        output.WriteEndTag("a");
    }

    // aumentar

    if (DisplayUpDown)
    {
        string l1 = "javascript:"+Page.GetPostBackEventReference(this,
"up");

        output.WriteBeginTag("a");

        output.WriteAttribute("href", l1);

        output.WriteAttribute("style", "color: black;

```

```

background-color: white; text-decoration: none; font-weight: bold;");

        output.Write( HtmlTextWriter.TagRightChar);

        output.Write( "&nbsp;&gt;&gt;&nbsp;");
        output.WriteEndTag("a");

    }

    // cierra la etiqueta </span>
    output.WriteEndTag("span");
}

#endregion

```

El método anterior es un poco extenso pero no presenta ninguna dificultad particular en la programación. Para su diseño, el desarrollador debe crear una maqueta 100% HTML y, a continuación, pensar el algoritmo que permite obtenerlo. Tras la publicación de la página, es posible comparar el código HTML de la página con la maqueta y reajustar nuestro programa si fuera necesario. He aquí el fragmento de código generado para nuestro componente:

```

<span name="ColoredPad1">

<!-- disminuir -->
<a href="javascript: doPostBack('ColoredPad1','down')"
style="color: black; background-color: white;
text-decoration: none; font-weight: bold;">
&nbsp;&lt;&lt;&nbsp;</a>

<!-- pasar al modo de edición -->
<a href="javascript:__doPostBack('ColoredPad1','edit')" style=
```

```

"text-decoration: none; color: White">
<!-- valor -->
<span style="background-color: #CC6600; color: White" name=
"ColoredPad1"> 80 % </span>
</a>
<!-- aumentar -->
<a href="javascript:__doPostBack('ColoredPad1','up')"
style="color: black; background-color: white;
text-decoration: none; font-weight: bold;">
&nbsp;&gt;&gt;&nbsp;</a>
</span>
```

## Activar el postback

El lector se habrá dado cuenta de que en el método `Render()` existen llamadas al método `GetPostBackEventReference()`. Esto crea la secuencia JavaScript de invocación de la función `_doPostBack` con el objetivo de producir un postback y, eventualmente, generar eventos.

Como respuesta a un postback, la clase implementa la interfaz `IPostBackEventHandler`. El siguiente método recibe como parámetro la cadena que identifica al evento que ha provocado el postback:

```
#region IPostBackEventHandler: RaisePostBackEvent

public void RaisePostBackEvent(string eventArgument)
{
    switch (eventArgument)
    {
        case "up":
            Up();
            break;

        case "down":
            Down();
            break;

        case "edit":
            Edit = true;
            break;
    }
}
```

```
case "valid":
    Edit = false;
    break;
}

#endregion
```

## Gestión de los datos del postback

Un postback está asociado a una acción interna del control, como hemos visto en la sección anterior, aunque también realiza la decodificación de los datos enviados mediante post. En efecto, un control puede crear campos HTML destinados a interactuar con el usuario.

El método `OnInit()` realiza, precisamente, una llamada a `Page.RegisterRequiresPostBack`. Esto quiere decir que la clase implementa la interfaz `IPostBackDataHandler` y que sus métodos deben invocarse para recoger los datos enviados.

```
#region IPostBackDataHandler LoadpostData y
```

```
RaisePostDataChangedEvent

public bool LoadpostData(string postDataKey, System.Collections.
Specialized.NameValueCollection postDataCollection)
{
    // este método se invoca si los datos del postback
    // le interesan al control

    // la llamada a Page.RegisterRequiresPostBack(this)
    // indica que el control necesita obtener los datos
    // del postback
    if (postDataCollection[this.ID + "_value"] == null)
        return false;

    string nuevo_valor = postDataCollection[ this.ID+"_value"];
    double nv = 0;
    try
    {
        nv = double.Parse(nuevo_valor);
    }
```

```
catch { }

if (nv != Value)
{
    Value = nv;
    return true; // el valor ha cambiado
}

// indica si es necesario invocar a RaisePostDataChangeEvent
return false;
}

public void RaisePostDataChangedEvent()
{
    // este método se invoca para indicar que ha habido un cambio en
    // el estado del control tras un post de los datos
    if(OnValueChanged!=null)
        OnValueChanged(Value);
}
```

```
#endregion
```

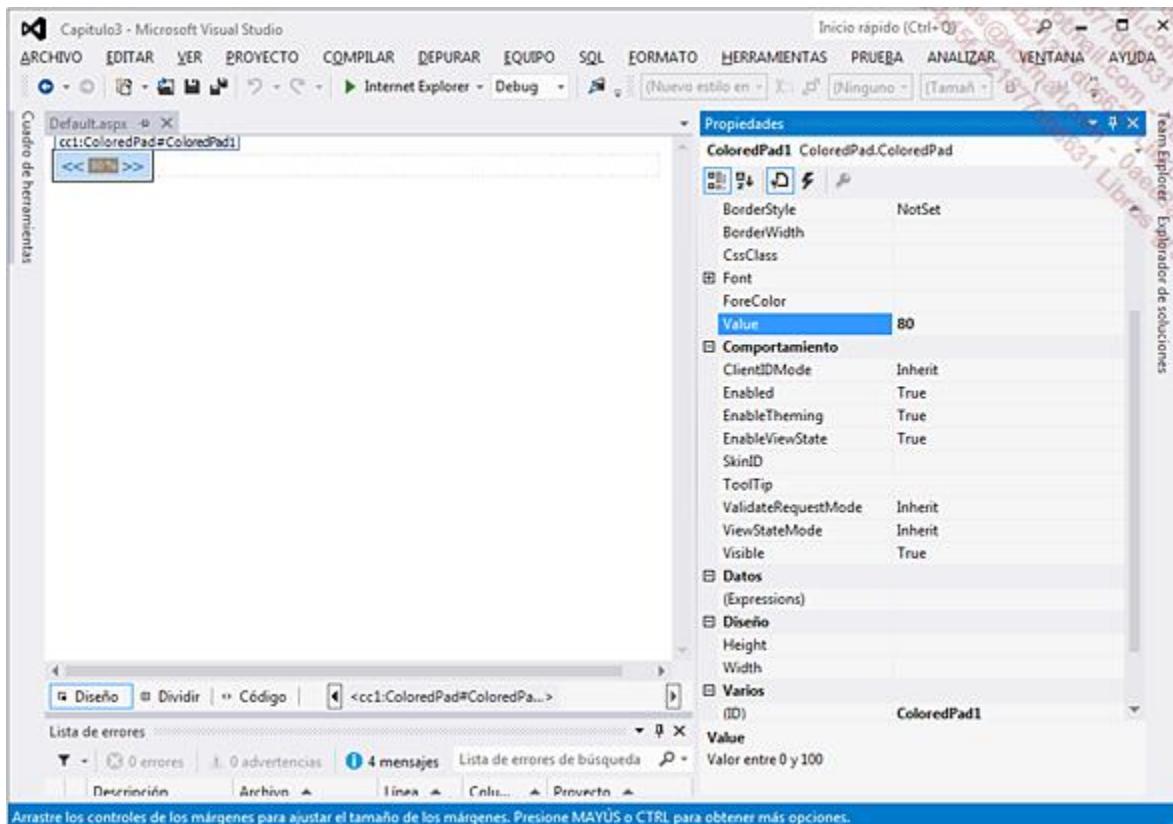
Estos dos métodos no se invocan consecutivamente. El primero recoge los datos y los analiza. Decide, a continuación, si los datos enviados justifican la generación de un evento de tipo cambio. Para ello, el método **LoadpostData** reenvía un valor booleano. Al finalizar la ejecución, el framework invoca a los demás métodos de la página y del control. Cuando llega el momento de generar un evento de tipo cambio (véase la sección Ciclo de vida de una página en este capítulo), se invoca el método **RaisePostDataChangedEvent** a su retorno, a condición de que **LoadpostData** haya devuelto true al framework.

#### d. Empaquetado y pruebas

Como cualquier componente de usuario .ascx, los componentes personalizados deben empaquetarse y registrarse en las páginas que los van a instanciar. Preferiblemente, el programador habrá definido una referencia a la DLL que contiene los componentes.

En la siguiente captura de pantalla, el lector podrá constatar el esfuerzo de Visual Studio para hacer que los componentes personalizados estén accesibles del mismo modo que los componentes estándar definidos por Microsoft. Por un lado, el cuadro de herramientas contiene una sección propia de la DLL referenciada. Además, el componente personalizado dispone de una representación gráfica en el seno de Visual Studio.

Por último, es posible editar las características directamente desde la barra de propiedades:



En cuanto a la integración en la página, responde a una sintaxis idéntica a los controles de usuario:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile=
```

```

"Default.aspx.cs" Inherits="_Default" %>

<%@ Register Assembly="aspnet20_cs_herramientas" Namespace=
"aspnet20_cs_herramientas" TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <form id="form1" runat="server">
        <cc1:ColoredPad ID="ColoredPad1" runat="server"
DisplayValue="True" Format="&nbsp;{0} &nbsp;" TextColor="White"
Value="80" BaseColor="255, 128, 0" DisplayUpDown="True" />
    </form>
</body>
</html>

```

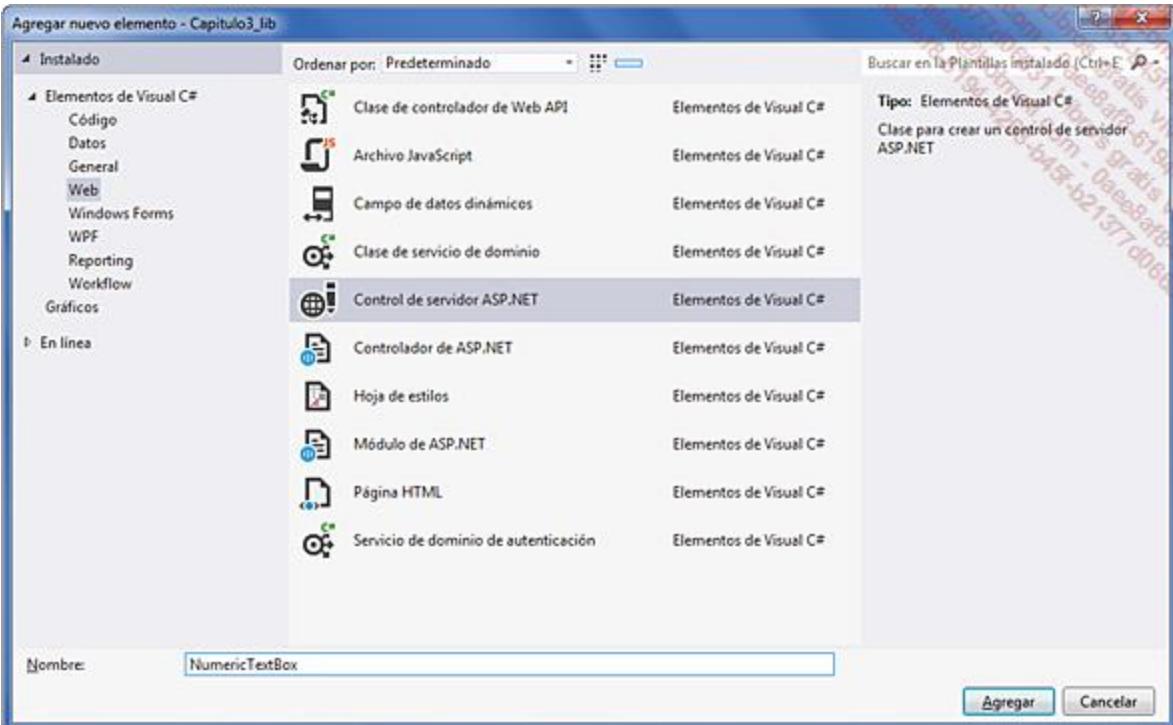
## 2. NumericTextBox, componente derivado de TextBox

Los componentes estándar del framework ASP.NET forman una paleta bastante rica de posibilidades, mientras que los tipos de controles de formularios HTML son limitados. El programador tiene la posibilidad de enriquecer esta paleta especializando el comportamiento de los controles web existentes.

### a. Creación del control

Nos interesaremos, a continuación, por NumericTextBox, un control web que deriva de TextBox y sólo acepta valores numéricos.

Si bien el control está compuesto por una simple clase, partir de un asistente nos facilita la operación declarando los espacios de nombres necesarios para su funcionamiento:



### b. Propiedades y eventos

Además de las propiedades del control TextBox, el componente NumericTextBox expone Value y DisplayNegativeInRed. La primera propiedad establece una contraparte numérica de la propiedad Text. La segunda aplicará una representación en color rojo si se trata de un valor negativo.

```
#region Propiedad Valor
[Description("Valor del control")]
public double Value
{
    set
    {
        Text = value.ToString();
    }
    get
    {
        try
        {
            return double.Parse(Text);
        }
        catch { }
        return 0;
    }
}
```

```
#endregion
```

Para conservar el valor de la propiedad `DisplayNegativeInRed`, nos apoyaremos en una construcción diferente a la de `ColoredPad`: el valor se hace persistente gracias al `ControlState` y no al `ViewState`.

El `ControlState` tiene la misma vocación que el `ViewState`, pero no se puede desactivar - no tiene `EnableControlState`. Microsoft ha hecho (deliberadamente) su programación mucho más restrictiva que la del `ViewState` con el objetivo de que no se pueda sobrecargar el campo oculto y compartido `__VIEWSTATE`.

Los métodos **LoadControlState** y **SaveControlState** aseguran la carga y la salvaguarda del conjunto de valores registrados en el `ControlState` del componente. El acceso utiliza una lista (`Pair`), que resulta mucho menos práctica que el diccionario `ViewState`.

```
#region Control State
protected override void LoadControlState(object savedState)
{
    Pair p = savedState as Pair;
    if (p != null)
    {
        base.LoadControlState(p.First);
        displayNegativeInRed = (bool)p.Second;
    }
}
```

```
protected override object SaveControlState()
{
    object estado_base = base.SaveControlState();
    return new Pair(estado_base, displayNegativeInRed);
}

#endregion

#region Propiedad DisplayNegativeInRed
private bool displayNegativeInRed;
public bool DisplayNegativeInRed
{
    get { return displayNegativeInRed; }
    set { displayNegativeInRed = value; }
}
#endregion
```

El control del formato numérico del texto se realiza con ayuda de un procedimiento interno; el componente declara, él mismo, un controlador de eventos para TextChanged:

```
#region Test texto numérico

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

    this.TextChanged += new EventHandler( NumericTextBox_TextChanged);
}

void NumericTextBox_TextChanged(object sender, EventArgs e)
{
    // verifica que el texto es un número
    try
    {
        double.Parse(Text);
    }
    catch
    {
        Value = 0;
    }
}

#endregion
```

### c. Representación

La ventaja de derivar un control web existente es que podemos utilizar su representación gráfica y, de este modo, limitar la cantidad de código que es necesario escribir.

```
#region Representación gráfica

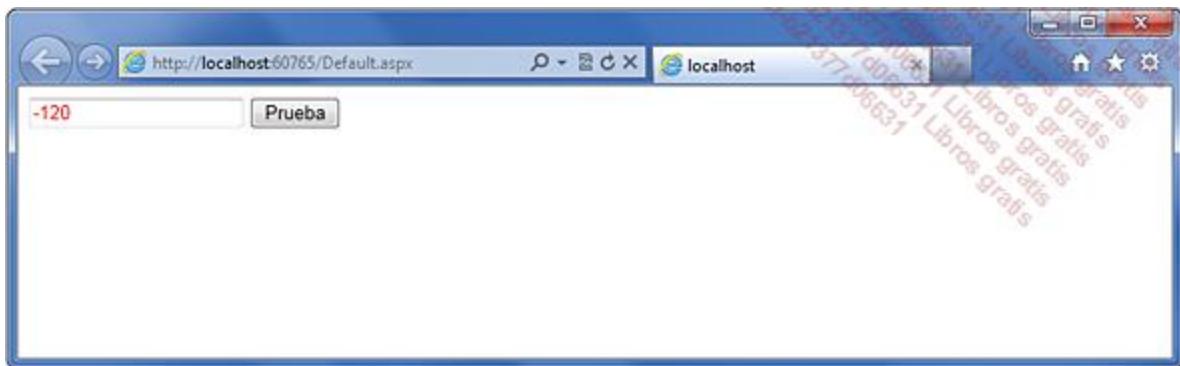
protected override void Render(HtmlTextWriter writer)
{
    Color c = Color.Black;

    if (Value < 0 && DisplayNegativeInRed)
    {
        c = this.ForeColor;
        ForeColor = Color.Red;
    }

    base.Render(writer);

    if (Value < 0 && DisplayNegativeInRed)
    {
```

```
    ForeColor = c;  
}  
}  
#endregion
```



### 3. ChartControl, componente gráfico que utiliza GDI+

Presentamos, a continuación, un componente que crea gráficos "ofimáticos". Si bien estos servicios se soportan, generalmente, mediante controles Active X de la suite MS Office, el framework .NET no provee un servicio estándar de creación de gráficos profesionales.

Técnicamente, el programador utiliza la biblioteca GDI+, versión específica de .NET del sistema gráfico GDI de Windows.

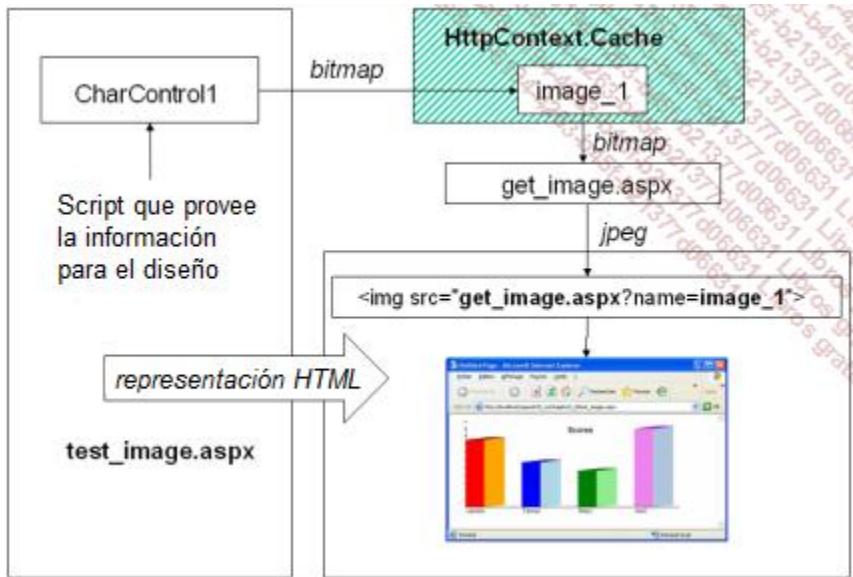
#### a. Funcionamiento

El funcionamiento del componente ChartControl es bastante original; los componentes web de ASP.NET no están previstos para fabricar secuencias binarias, sino únicamente secuencias HTML. Precisamente por ello, la etiqueta que define una imagen en una página web no hace sino referenciar a la URL donde se ubica su archivo binario. Esta URL no puede ser la de la página que describe la etiqueta ``.

Por este motivo, utilizaremos una página auxiliar, `get_image.aspx`. Tendremos, también, que utilizar un Web Handler. Lo que distingue al ChartControl de los enfoques habituales es el hecho de que la página `get_image.aspx` no fabrica el diseño de la imagen, sino que se contenta con ubicarla en el flujo de salida con formato JPEG.

Los enfoques clásicos dejan el control al componente "gráfico" para generar una etiqueta `<img>` que hace referencia a la página auxiliar pasándole toda la información necesaria para su diseño (tamaño, valores, texto, reglas de representación...). Esto es mucho más restrictivo, pues la única forma de comunicación con la página auxiliar es la `QueryString`, canal expuesto y con posibilidades muy limitadas. Por otro lado, es la página la que instancia el componente gráfico que tiene toda la información. Comunicarlo a otra página no es muy eficaz.

En el caso de ChartControl, el objeto Cache sirve como memoria buffer entre la representación del componente y la página auxiliar. Todo el diseño lo realiza directamente el componente, que memoriza el mapa de bits (el soporte), en el objeto Cache utilizando un nombre concreto. A continuación, genera una etiqueta `<img>` que apunta a la página auxiliar `get_image.aspx` y le indica el nombre de la imagen en caché.



## b. Representación

He aquí la parte del método `Render()` que controla la ubicación en caché:

```

protected override void RenderContents(HtmlTextWriter output)
{
    // generación del bitmap
    // ...

    // ubicación en caché
    long t=DateTime.Now.Ticks;
    string nom = string.Format("image_{0}", t);
    HttpContext.Current.Cache.Add(
        nombre, bmp,
        null, DateTime.Now.AddSeconds(10), TimeSpan.Zero,
        CacheItemPriority.Normal, null);

    // representación gráfica
    output.WriteBeginTag("img");
    output.WriteAttribute("width", PixelWidth.ToString());
    output.WriteAttribute("height", PixelHeight.ToString());
    output.WriteAttribute("src", string.Format("get_image.aspx?name="
    {0}", nombre));
}

```

El nombre de la imagen en caché debe ser único. Hemos utilizado, aquí, la unidad de tiempo más pequeña, los Ticks (1/18,2 segundos). Sucesivas consultas muy próximas entre sí podrían tener el mismo nombre, de modo que se incluye un código hash MD5 que depende, a la vez, del tiempo y de la consulta, limitando así los riesgos de corrupción de memoria.

La imagen se conserva en caché durante 10 segundos, lo que da tiempo suficiente a un navegador para recibir la página e ir a buscar la imagen al servidor. Un programador avisado podría crear, fácilmente, estrategias de caché más adecuadas basadas en una combinación de valores de diseño.

### c. Integración y pruebas

La página get\_image.aspx funciona como nuestro generador de imágenes del capítulo Los sitios web ASP.NET 4.5. Empieza buscando el mapa de bits en la caché antes de escribir el flujo de salida:

```
public partial class get_image : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // recupera el mapa de bits de la caché
    }
}
```

```
Bitmap bmp = (Bitmap)Cache[Request.QueryString["name"]];
if(bmp==null)
    return;

Response.ClearContent();
Response.AddHeader("Content-Type","image/jpeg");
bmp.Save(Response.OutputStream,ImageFormat.Jpeg);
Response.Flush();
}
}
```

Sólo nos quedaría probar:



## 4. PictureBrowser, componente basado en una plantilla

Como novedad del framework ASP.NET, los componentes basados en plantillas ofrecen a los programadores la posibilidad de crear nuevos Repeater. Un componente basado en una plantilla se caracteriza por su funcionamiento de aplicación fija mientras que la representación final en la página es específica de cada instancia, en la página en que se integra.

### a. Funcionamiento

El componente PictureBrowser se define para la representación de imágenes según dos modos, miniatura o imagen por imagen. Un DataTable contiene la lista de imágenes que se quiere representar. Este objeto es un fragmento de DataSet y puede inicializarse mediante un archivo XML.

Para funcionar, el componente PictureBrowser utiliza un control compuesto Picture. Los controles compuestos derivan de CompositeControl, o bien heredan de WebControl e implementan INamingContainer:

```
#region Picture
public class Picture : WebControl,INamingContainer
{
    public Picture(string name, string src,int index)
    {
        Name = name;
        Src = src;
        pictureIndex = index;
    }
    private string name, src;
    private int pictureIndex;

    public int PictureIndex
    {
        get { return pictureIndex; }
        set { pictureIndex = value; }
    }

    public string Src
    {
        get { return src; }
        set { src = value; }
    }

    public string Name
    {
```

```

        get { return name; }

        set { name = value; }

    }

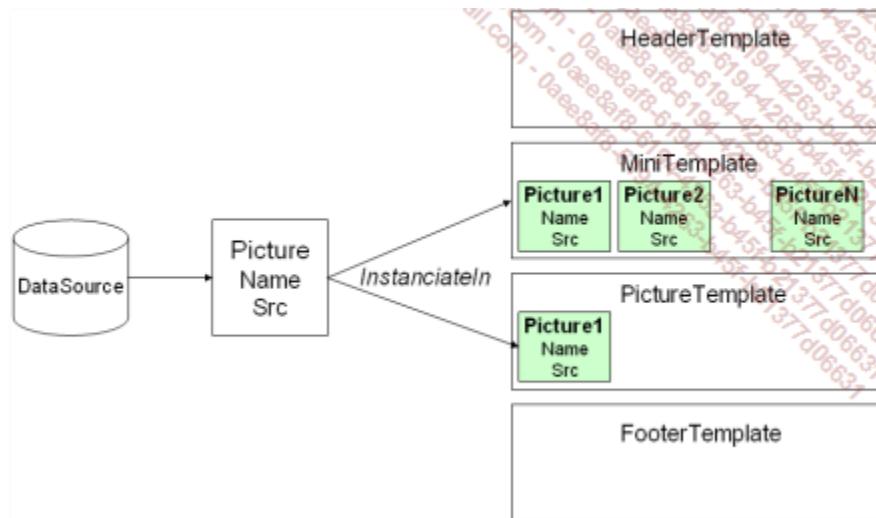
}

#endregion

```

Esta clase no sobrecarga el método Render; es, de hecho, su método **CreateChildControls()** el que construye de forma automática la secuencia HTML.

En su instanciación, el control Picture se configura con los datos de una imagen (propiedades Name y Src) e integra una plantilla del control PictureBrowser:

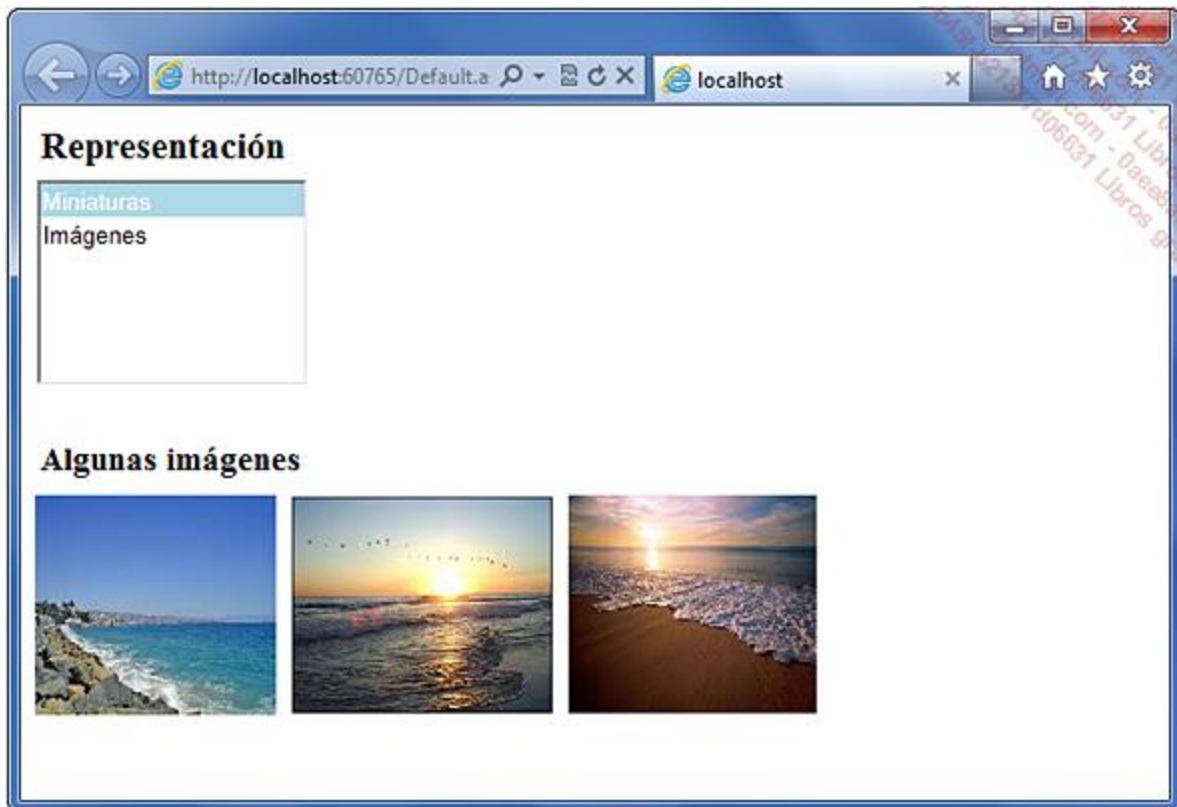


Además de los modelos MiniTemplate (representación de todas las imágenes en miniatura) y PictureTemplate (representación en tamaño real, imagen por imagen), el control PictureBrowser dispone de los modelos HeaderTemplate y FooterTemplate para mejorar la representación.

He aquí las principales propiedades del control PictureBrowser:

PictureCount	Número de imágenes presentes en el DataSource.
CurrentPictureIndex	Índice de la imagen en curso en el modo Picture.
DisplayNavigator	Indica si los botones de navegación (imagen anterior/siguiente) deben generarse en modo Picture.
Mode	Pasa de la representación en miniaturas a la representación imagen por imagen.

b.



### Implementación del componente

El componente PictureBrowser es, él mismo, un componente composite. Como procesa eventos de tipo postback, implementa la interfaz IPostBackEventHandler. La interfaz INamingContainer hace referencia a propiedades de tipo ITemplate e indica al framework la naturaleza composite del control. No existe ningún método ligado a esta interfaz.

```
[PersistChildren(true)]  
[Designer(typeof(PictureBrowserDesigner))]  
[ToolboxData("<{0}:PictureBrowser runat=server></{0}:PictureBrowser>")]  
public class PictureBrowser : WebControl, INamingContainer,  
IPostBackEventHandler  
{  
}
```

Los atributos que cualifican a la clase tienen, cada uno, un rol determinado:

PersistChildren	Indica a Visual Studio que los sub-controles son propiedades o forman parte de la colección de controles.
Designer	Designa una clase usuaria para facilitar la integración del control en Visual Studio.
ToolboxData	Forma la sintaxis XHTML para instanciar el control en la página.

plantillas

c. Las

Las plantillas son elementos XHTML que reciben secuencias HTML y controles web.

```

<ccl:PictureBrowser ID="PictureBrowser1" runat="server">

    <HeaderTemplate>
        <h2>Algunas imágenes</h2>
    </HeaderTemplate>

    <MiniTemplate>
        <asp:Image ImageUrl='<%# Container.Src %>' Width="80"
Height="80" runat="server" />
    </MiniTemplate>

    <PictureTemplate>
        <asp:Image ID="Image1" ImageUrl='<%# Container.Src %>'
runat="server" />
    </PictureTemplate>
</ccl:PictureBrowser>

```

Para autorizar esta construcción, el control define propiedades del tipo ITemplate. Un atributo específico indica que la propiedad de la clase componente se corresponde con un subelemento XHTML y no con un atributo, como es generalmente el caso:

```

#region Propiedad FooterTemplate
private ITemplate footerTemplate;

[PersistenceMode(PersistenceMode.InnerProperty)]
public ITemplate FooterTemplate
{
    get { return footerTemplate; }
    set { footerTemplate = value; }
}
#endregion

#region Propiedad MiniTemplate
private ITemplate miniTemplate;
[TemplateContainer(typeof(Picture))]
[PersistenceMode(PersistenceMode.InnerProperty)]

```

```

public ITemplate MiniTemplate
{
    get { return miniTemplate; }
    set { miniTemplate = value; }
}

```

```
#endregion
```

La propiedad MiniTemplate está, del mismo modo, cualificada por el atributo TemplateContainer. En efecto, está ligada a una instancia del control Picture que define las propiedades Name y Src, que pueden estar ligadas mediante Data binding al origen de datos DataSource.

#### d. Representación

En un control composite (y, por tanto, un control basado en una plantilla), es el método `CreateChildControls()` el que asegura la representación. El método `Render`, en su versión básica, invoca a `CreateChildControls` en el momento oportuno para construir la secuencia HTML definitiva.

#### DataBind

Cuando el programador aplica el método `.DataBind()` al componente `PictureBrowser`, pide la resolución de las secuencias de enlace `<%# %>`. Es, entonces, momento de invocar a `CreateChildControls`. Esta invocación se realiza de forma indirecta a través del método `EnsureChildControls`. Esta organización concreta evita al framework tener que llamar varias veces a `CreateChildControls`.

```
public override void DataBind()
{
    if (DataSource == null)
        return;

    PictureCount = DataSource.Rows.Count;

    EnsureChildControls(); // invoca a CreateChildControls
    base.DataBind();
}
```

#### CreateChildControls

El método `CreateChildControls` va a instanciar a cierto número de controles web y los va a agregar a la colección `Controls`. Escoge, en particular, los modelos apropiados según el modo de representación (Miniatura, Imagen).

```
protected override void CreateChildControls()
```

```
{
    Controls.Clear();

    #region HeaderTemplate
    if (HeaderTemplate != null)
    {
        Panel head = new Panel();
```

```

        HeaderTemplate.InstantiateIn(head);
        Controls.Add(head);
    }

#endregion

#region Mode==PictureMode.Picture
if (Mode == PictureMode.Picture)
{
    if (PictureTemplate != null && DataSource != null)
        try
    {
        DataRow dr = DataSource.Rows[ CurrentPictureIndex];
        Picture pic =
            new Picture(
                dr["name"] as string,
                dr["src"] as string,
                CurrentPictureIndex);

        PictureTemplate.InstantiateIn(pic);
        Controls.Add(pic);
    }
    catch { }
}

#endregion

```

```

#region Mode==PictureMode.Miniature
if (Mode == PictureMode.Miniature)

```

```

{
    if (DataSource != null && MiniTemplate == null)
    {
        int index = 0;
        foreach (DataRow dr in DataSource.Rows)
        {
            Picture pic =
                new Picture(dr["name"] as string,
                dr["src"] as string, index);

```

```

        MiniTemplate.InstantiateIn(pic);

        Controls.Add(pic);
        index++;
    }

}

}

#endregion

#region FooterTemplate
if (FooterTemplate != null)
{
    Panel foot = new Panel();
    FooterTemplate.InstantiateIn(foot);
    Controls.Add(foot);
}
#endregion
}

```

Dado que el modelo no itera sobre los datos, se instancia un Panel que contiene la secuencia de controles web, como es el caso del HeaderTemplate y FooterTemplate. El método **InstantiateIn** vuelve a copiar los controles web definidos en XHTML en el contenedor correspondiente.

En el caso de los modelos MiniTemplate y PictureTemplate, es la clase composite Picture la que sirve como contenedor. De esta forma, la secuencia XHTML puede utilizar el enlace de datos y referenciar a las propiedades Name, Src y PictureIndex.

#### e. Eventos

Para generar eventos en un control composite, podemos hacerlo de forma análoga al código del método Render de los componentes web que derivan de WebControl, es decir, utilizando GetPostBackEventReference e implementando la interfaz IPostBackEventHandler.

```

#region DisplayNavigator
if (DisplayNavigator)
{
    string tag =
        string.Format(
            "<input type=\"button\" name=\"{1}_next\" value=\"Imagen
Siguiente\" onclick=\"{0}\">",
            Page.GetPostBackEventReference(this, "next"), this.ID);
}

```

```

Literal bn = new Literal();
bn.Text = tag;

tag =
    string.Format(
        "<input type=\"button\" name=\"{1}_prev\" value=\"Imagen
Anterior\" onclick=\"{0}\">>",
        Page.GetPostBackEventReference(this, "previous"), this.ID);

Literal bp = new Literal();
bp.Text = tag;

Literal salto = new Literal();
salto.Text = "r>r>";

Controls.Add(bp);
Controls.Add(bn);
Controls.Add(salto);
}

#endregion

```

En el caso de un control composite, el programador debe prestar atención a la hora de agregar (Add) los controles que incluyan eventos una vez que todos los controles hayan sido instanciados. De otro modo, el acceso a la colección Controls parece tener repercusiones sobre el buen funcionamiento de la información que se envía con el postback.

Por último, sólo queda implementar la interfaz IPostBackEventHandler para producir los eventos OnPrevious y OnNext:

```

#region IPostBackEventHandler Members

public event EventHandler OnNext, OnPrevious;
public void RaisePostBackEvent(string eventArgument)
{
    switch (eventArgument)
    {
        case "previous":
            if (OnPrevious != null)
                OnPrevious(this, EventArgs.Empty);
            break;

        case "next":
    }
}

```

```
        if (OnNext != null)
            OnNext(this, EventArgs.Empty);
        break;
    }
}

#endregion
```

#### f. Información relativa al diseño en Visual Studio

Como la salida HTML de nuestro control depende de la invocación a Create- ChildControls y el método EnsureChildControls no se invoca en otras ocasiones más allá de DataBin

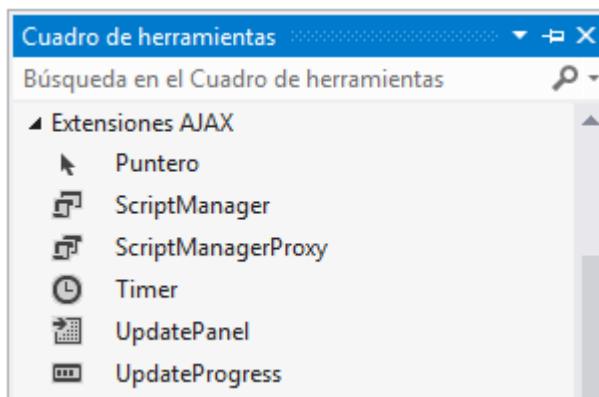
# AJAX

## 1. Del callback a AJAX

El mecanismo de callbacks, incorporado con la versión 2.0 de ASP.NET, suponía una primera etapa antes de disponer de un framework completo de tipo AJAX. De hecho, no existe ninguna diferencia técnica entre los callbacks y AJAX (*Asynchronous JavaScript And XML*). No obstante, la implementación de los callbacks se vuelve, en ocasiones, delicada, pues se trata de una simple solución técnica.

Microsoft ha publicado, a finales de 2006, una extensión de ASP.NET bajo la forma de una DLL que contiene componentes esenciales para dar soporte a un framework completo. Estos componentes son el controlador del script y el panel de actualización. Paralelamente, el fabricante de software ha promovido (financiado) la puesta a punto de un kit de componentes AJAX publicado en Codeplex (el sitio equivalente a Source Forge para Microsoft).

La versión 3.5 de ASP.NET oficializa e integra ambas iniciativas, volviendo su uso mucho más sencillo y, sobre todo, más duradero.



## 2. El script manager

El controlador de script es el componente esencial del framework AJAX de Microsoft. Sirve como base para cargar las librerías de código JavaScript dependientes del navegador e indispensables para el funcionamiento de los componentes AJAX.

Debe, en cualquier caso, ser el primer componente de una página web, y puede, también, instanciarse desde una página maestra.

```
<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div>
    </div>
</form>
```

Entre sus propiedades, se encuentran Scripts y Services para precargar las librerías de código JavaScript y alcanzar servicios web proveedores de datos.

Esto es suficiente para incluir un componente UpdatePanel (véase a continuación). Aun así, el componente ScriptManager sirve, también, para manejar servicios web desde JavaScript.

### a. Usar ScriptManager con servicios web

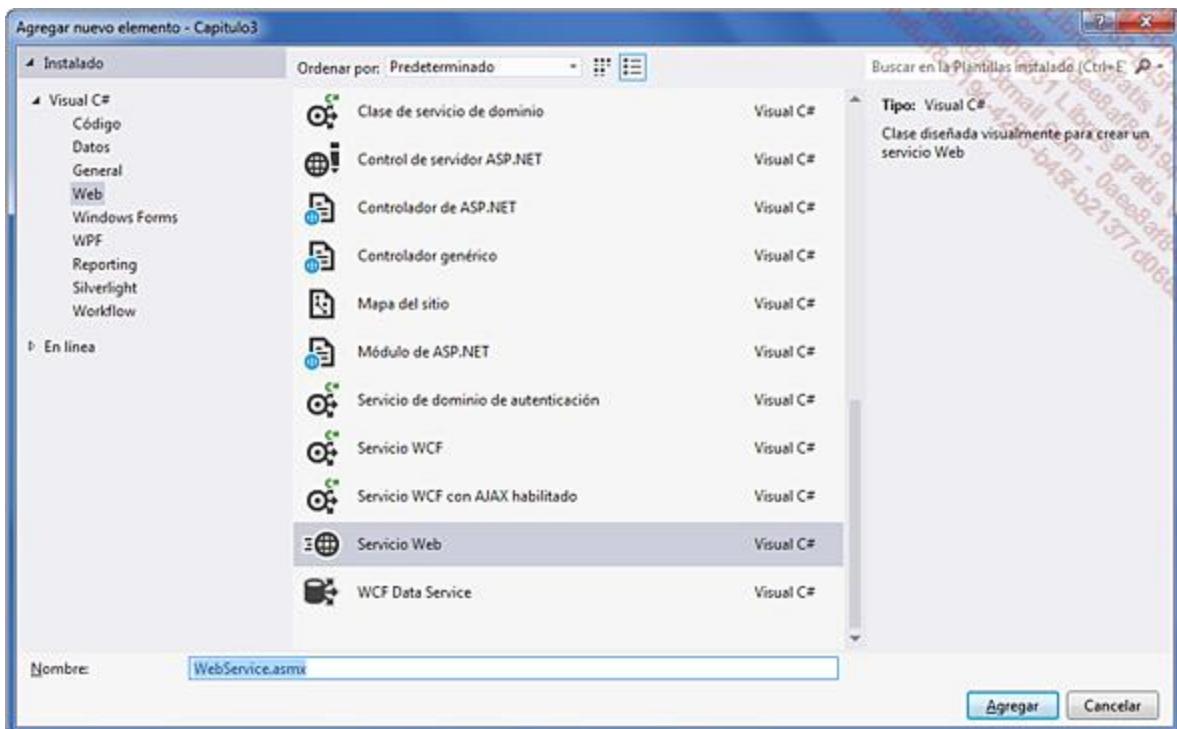
El ScriptManager instancia un repetidor (proxy) JavaScript hacia los servicios web. Veamos el ejemplo de la clase Libro, que se muestra a continuación:

```
public class Libro
{
    public Libro()
    {
    }

    public Libro(string titulo, string autor)
    {
        Autor = autor;
        Titulo = titulo;
    }

    public string Autor { get; set; }
    public string Titulo { get; set; }
}
```

Vamos a construir una página capaz de consultar un servicio web que nos devuelva información acerca de las obras. La primera etapa consiste en agregar un servicio web SOAP de tipo ASMX(véase el capítulo Los servicios Web WCF y REST si desea más detalles):



Visual Studio crea un punto de acceso `LibreriaWebService.asmx` y una clase de código subyacente. Conviene activar el atributo **ScriptService** que habilita el acceso al servicio desde JavaScript mediante un controlador de script. El método `GetObra()` no presenta ninguna particularidad salvo que está marcado por el atributo **WebMethod**, como todos los métodos públicos de servicio web SOAP.

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class LibreriaWebService : System.Web.Services.WebService {

    public LibreriaWebService () {
    }

    [WebMethod]
    public Libro GetObra(int index) {
        List<Libro> l = new List<Libro>();

        l.Add(new Libro("ASP.NET 4.5 en C#", "Brice-Arnaud"));
        l.Add(new Libro("ASP.NET 4.5 en VB.NET", "Brice-Arnaud"));

        return l[index];
    }
}
```

```
}
```

La página ASPX instancia, si nos fijamos, un controlador de script y declara una referencia al servicio web:

```
<asp:scriptmanager ID="Scriptmanager1" runat="server">
    <Services>
        <asp:ServiceReference Path("~/LibreriaWebService.asmx" />
    </Services>
</asp:scriptmanager>
```

Esta declaración tiene como resultado la instantiación dinámica de un repetidor (proxy) hacia el servicio, que puede utilizarse desde una función de llamada:

```
<script language="javascript">
    // función de llamada
    function muestraObra(index) {
        LibreriaWebService.GetObra(index,GetObraCompleta);
    }

    // función de recepción
    function GetObraCompleta(result) {
        var o = document.getElementById("obra");
        o = $get("obra"); // sintaxis equivalente

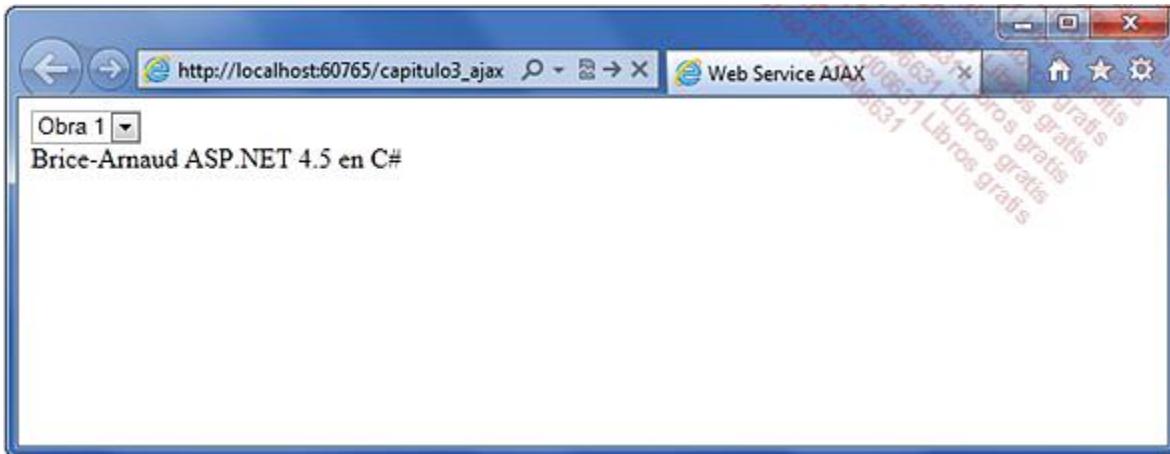
        // muestra el detalle de la obra
        o.innerHTML = result.Autor + " " + result.Titulo;
    }
</script>
```

Tan sólo queda invocar a la función de llamada desde un evento concreto para hacer funcionar nuestro servicio web:

```
<div>
    <asp:DropDownList ID="ddl_obra" runat="server"
onchange="muestraObra(this.value);">

        <asp:ListItem Text="Obra 1" Value="0"></asp:ListItem>
        <asp:ListItem Text="Obra 2" Value="1"></asp:ListItem>
    </asp:DropDownList>
</div>
```

```
<div id="obra"></div>
```



### 3. El componente UpdatePanel

#### a. Funcionamiento

Este componente tiene la capacidad de realizar un postback parcial si el control que inicia la llamada forma parte de su modelo ContentTemplate. Todos los controles del panel se refrescarán, sin que el programador tenga que escribir una sola línea de código.

#### b. Implementación

En el siguiente ejemplo, se utilizan dos etiquetas (label) para mostrar la hora en curso. El primer label está integrado dentro de un control UpdatePanel, de modo que se actualiza con cada clic en el botón Button1. El segundo label se inicializa tras el primer acceso a la página.

```
<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">

    </asp:ScriptManager>
    <div>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Button ID="cmd_act_hora" runat="server"
                    Text="Actualizar" OnClick="cmd_act_hora_Click" />
                <br />
                Hora Ajax:
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
</form>
```

```
<asp:Label ID="lbl_hora_panel"
runat="server"></asp:Label>
<br />
</ContentTemplate>
</asp:UpdatePanel>
</div>
```

```

<br />
Hora Page:
<asp:Label ID="lbl_hora_page" runat="server"></asp:Label>
</form>

```

El código no necesita implementar ninguna interfaz: basta con incluir un simple controlador de código C# en el botón:

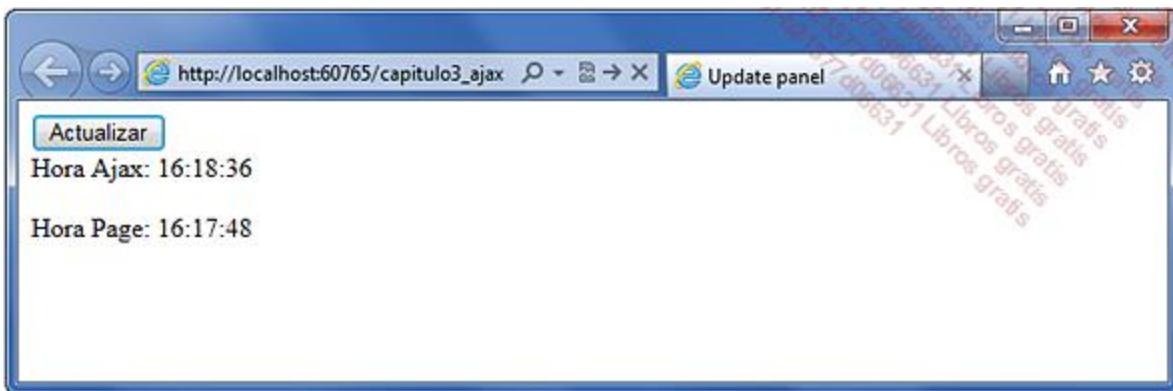
```

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack && !IsCallback)
        {
            lbl_hora_page.Text = DateTime.Now.ToString();
        }
    }

    protected void cmd_act_hora_Click(object sender, EventArgs e)
    {
        lbl_hora_panel.Text = DateTime.Now.ToString();
    }
}

```

Tras la ejecución, comprobamos que el primer label, que figura en el panel UpdatePanel, muestra una hora posterior al segundo label, que permanece estática (y no se refresca más).



### c. Gestión de errores

El mecanismo AJAX permite recoger, en el navegador, excepciones de aplicación producidas en el servidor. Veamos la siguiente página:

```

<asp:ScriptManager ID="scriptManager1" runat="server">
</asp:ScriptManager>

```

```

<div>

<asp:UpdatePanel ID="panel1" runat="server">
<ContentTemplate>
    <asp:Button ID="cmd_error" runat="server" Text="Error"
    OnClick="cmd_error_Click"/>
</ContentTemplate>
</asp:UpdatePanel></div>

```

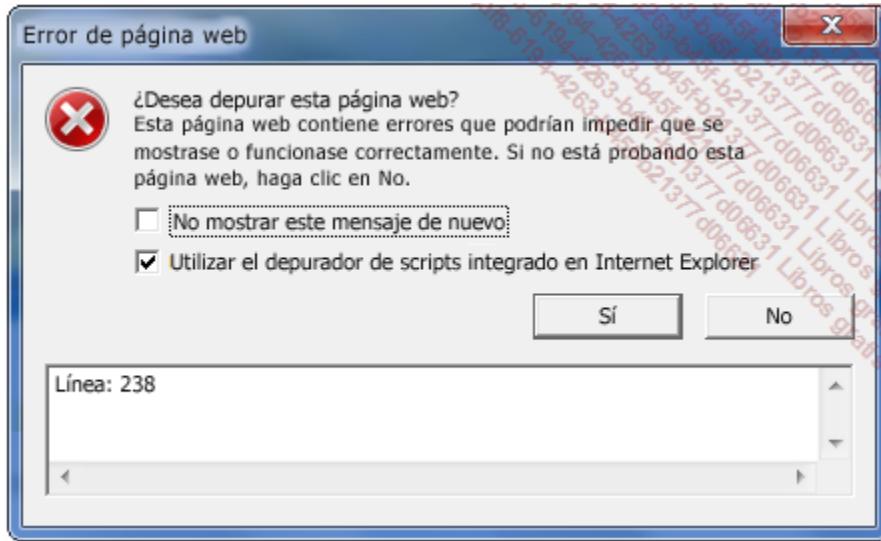
El código referente a los eventos del botón no hace sino producir una excepción:

```

protected void cmd_error_Click(object sender, EventArgs e)
{
    throw new ApplicationException("Error en el postback");
}

```

Sin existir un UpdatePanel, este error produce la detención de la ejecución de la página y elWeb.config indica cómo debe reaccionar ASP.NET: mostrar un error, redirigir al usuario a una página de error... En el caso de un componente UpdatePanel, se captura el error y se devuelve al navegador:



La segunda etapa consiste en capturar el error mediante código JavaScript personalizado. La función JavaScript pageLoad() siguiente declara un controlador de eventos que se invoca tras cada petición:

```

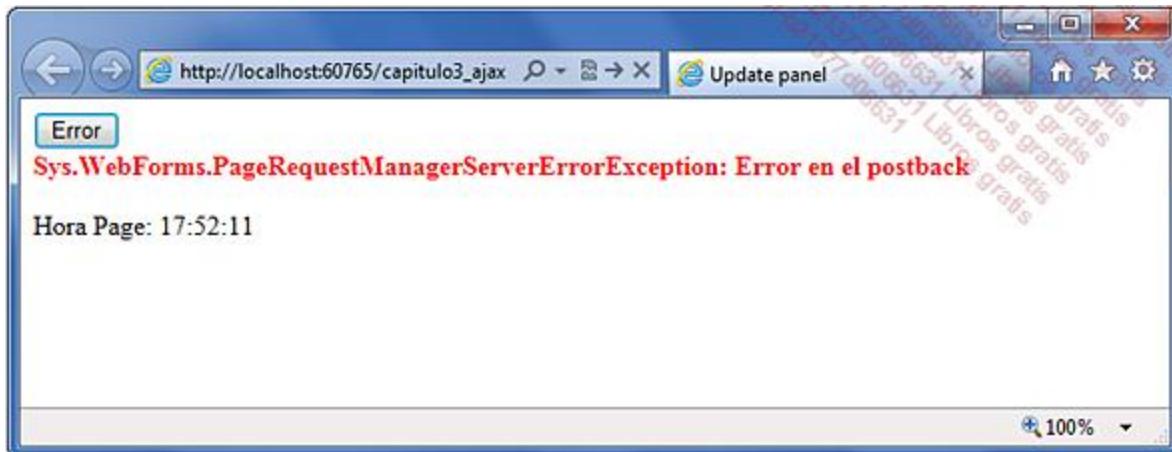
function pageLoad(sender, args) {
    var pageManager = Sys.WebForms.PageRequestManager.getInstance();
    pageManager.add_endRequest(finPeticion);
}

```

La definición del gestor de fin de petición aparece a continuación. Detecta la presencia de cualquier error durante el procesamiento de la petición y extrae el mensaje para mostrarlo en un lugar específico:

```
function finPeticion(sender,args){  
    if(args.get_error() != null){  
        var mensaje=args.get_error().mensaje;  
        $get("lbl_error").innerHTML=mensaje;  
  
        // indica que se ha controlado el error  
        args.set_errorHandled(true);  
    }  
}
```

En esta ocasión, el error no lo captura el navegador sino el código personalizado, lo cual resulta indispensable en el marco de una aplicación:



#### d. Los triggers

Los triggers permiten actualizar el contenido de un UpdatePanel capturando los eventos de los controles situados fuera de su extensión. Hemos visto, anteriormente, cómo un botón situado en un UpdatePanel provocaba implícitamente un postback parcial al realizar un clic. Gracias a los triggers, es posible realizar este postback parcial desde un botón situado en el exterior:

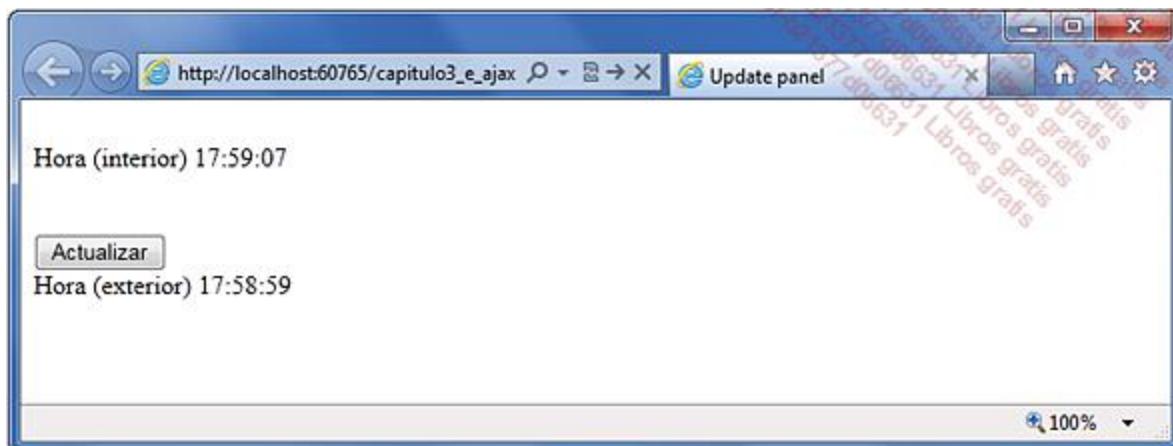
```
<asp:UpdatePanel ID="panel1" runat="server">  
    <ContentTemplate>  
        Hora (interior) <%= DateTime.Now.ToString() %>  
    </ContentTemplate>  
    <Triggers>  
        <asp:AsyncPostBackTrigger ControlID="cmd_fuera_panel"  
            EventName="Click" />  
    </Triggers>  
</asp:UpdatePanel>
```

```

<%--botón situado en el exterior--%>
<asp:Button ID="cmd_fuera_panel" runat="server" Text="Actualizar" />
<br />
Hora (exterior) <%= DateTime.Now.ToString() %>

```

El evento Click del control cmd\_fuera\_panel se captura gracias al trigger. Un clic producirá, así, un postback parcial y la actualización del control UpdatePanel:



#### 4. El componente UpdateProgress

Este componente está activo (se muestra) únicamente durante las fases de postback parcial, es decir, cuando un control ha iniciado el refresco de un UpdatePanel. La propiedad **AssociatedUpdatePanelID** indica qué panel debe supervisarse.

```

<asp:UpdateProgress ID="UpdateProgress1" runat="server"
AssociatedUpdatePanelID="UpdatePanel1">
    <ProgressTemplate>
        Refresco en curso
    </ProgressTemplate>
</asp:UpdateProgress>

```

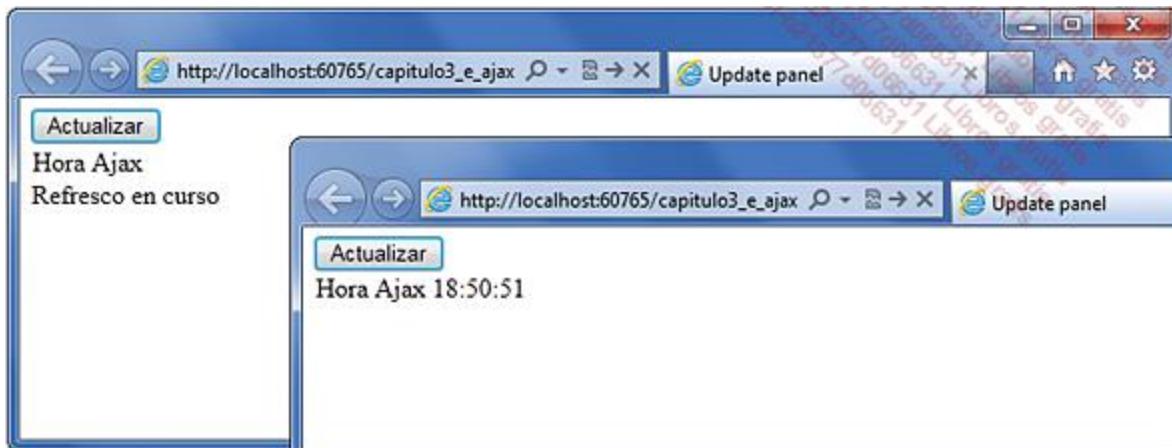
Con el objetivo de mostrarlo aquí, hemos agregado un temporizador en el controlador del clic del botón:

```

protected void cmd_act_hora_Click(object sender, EventArgs e)
{
    lbl_hora_panel.Text = DateTime.Now.ToString();
    System.Threading.Thread.Sleep(4000);
}

```

Tenemos, así, tiempo para apreciar el mensaje de espera cuando se produce una actualización de la hora:



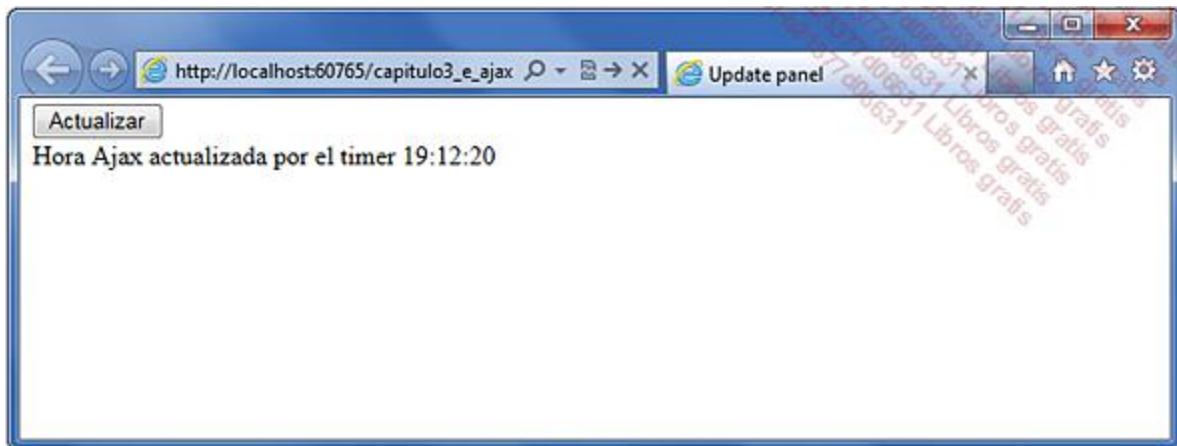
En la práctica, solemos encontrar una imagen de tipo GIF animado que simula una barra de progreso. AJAX nos da, en ese caso, la impresión de tener una interfaz próxima a la de un cliente Windows.

## 5. El Timer

El componente AJAX **Timer** es capaz de iniciar ciclos de actualización de un panel UpdatePanel:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Button ID="cmd_act_hora" runat="server"
            Text="Actualizar" OnClick="cmd_act_hora_Click" />
        <asp:Timer ID="Timer1" runat="server"
            Interval="10000" ontick="Timer1_Tick">
        </asp:Timer>
        <br />
        Hora AJAX:
        <asp:Label ID="lbl_hora_panel" runat="server"></asp:Label>
        <br />
    </ContentTemplate>
</asp:UpdatePanel>
```

Para que funcione es preciso tener en cuenta el evento **Tick** de servidor:



## 6. Programación orientada a objetos con JavaScript

El modelo de extensión AJAX de Microsoft facilita la programación orientada a objetos en JavaScript aportándole estructura y legibilidad.

### a. Inserción de código JavaScript en una página

Si bien los DTD HTML prevén la inserción de etiquetas `<script>` en la sección `<head>` del documento, no es raro insertar fragmentos de código casi en cualquier sitio. Existen varios modos de inserción posibles:

<pre>&lt;script&gt; Código &lt;/script&gt;</pre>	Modo en línea	Práctico durante las fases de puesta a punto pero no muy modular; el código no está mutualizado entre las distintas páginas.
<pre>&lt;script src="modulo.js" /&gt;</pre>	Módulo referenciado	El módulo está referenciado entre varias páginas del mismo sitio.
<pre>string url = Page.ClientScript.GetWeb ResourceUrl(     this.GetType(),     "ComponentesJavascript.     ModuloIncorporado.js" );  Page.ClientScript. RegisterClientScript Include(     "ModuloIncorporado.js", url);</pre>	Referencia dinámica	El módulo JavaScript puede encapsularse en una DLL.

La

referencia dinámica tiene el mismo efecto que la inserción mediante una etiqueta `<script src=/>`. Cuando el módulo se incorpora en una DLL, el controlador de

<pre> &lt;asp:ScriptManager     ID="scriptManager1"     runat="server"&gt;      &lt;Scripts&gt;         &lt;asp:ScriptReference             Name =             "ComponentesJavaScript.             ModuloIncorporado.js"             Assembly =             "Componentes-JavaScript"         /&gt;      &lt;/Scripts&gt; &lt;/asp:ScriptManager&gt; </pre>	Controlador de script	Como el anterior, pero utiliza plenamente las posibilidades del script manager.
--	-----------------------	---

scriptClientScriptManager o ScriptManager utiliza una URL especial, basada en un controlador HTTP WebResource.axd (respectivamente ScriptManager.axd):

```

<script
src="/capitulo3_e_ajax/WebResource.axd?d=a8y_A0VzeDJbEflZkv9LVA2&am
p;t=634183748232812500" type="text/javascript"></script>

<script src="/capitulo3_e_ajax/ScriptResource.axd?d=Xhmrt2e-
9Pa7XqB0amjfzS7VVMSRQsfIfMl-DF89x7nEg4WdrvBli-WDpK6eL2Xuz0bWPjCM-
ad_UAwz_Wnppj2i8SpkrRfBgq-OxsFG8A1& t=16ab2387"
type="text/javascript"></script>

```

Como se ha descrito anteriormente, el archivo .js se integra en una DLL, se marca como recurso embebido (*Embedded resource*) y se declara mediante el atributo [assembly: WebResource] para que el navegador pueda acceder a él.

### b. Crear objetos y clases JavaScript

Los siguientes ejemplos están realizados mediante código JavaScript; el código fuente correspondiente figura, de este modo, en el interior de una sección `<script>...</script>` o bien los descarga el navegador mediante archivos .js, incorporados o no.

JavaScript es un lenguaje débilmente tipado: es interpretado y la constitución de clases definida por el usuario no forma parte de su definición original. Esto no es, por tanto, contradictorio con una orientación a objetos, como muestra este primer ejemplo:

```

// declaración de un objeto y de dos atributos
var personal = new Object();

```

```

personal.nombre = "Diego";
personal.apellido = "Velázquez";

alert(personal.nombre + " " + personal.apellido);

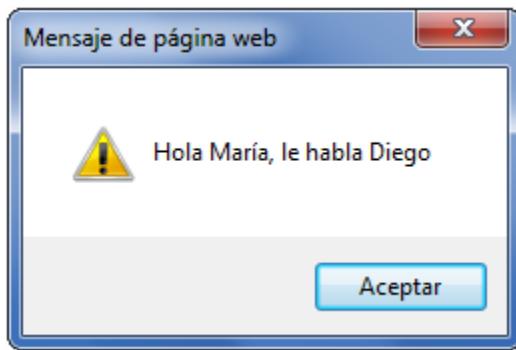
```

Esta escritura indica cómo se puede, de manera dinámica, agregar miembros a los objetos, aunque la noción de método (función) también es indispensable:

```

// agregar métodos a un objeto
personal.presentacion = function (otroNombre) {
    alert("Hola " + otroNombre + ", le habla " + this.nombre);
}
personal.presentacion("María");

```



Ahora que hemos comprobado la capacidad de JavaScript en términos de programación orientada a objetos, vamos a abordar el concepto de clases y espacios de nombres. Una clase es un modelo (una matriz) de objetos. A partir de una clase es posible fabricar objetos en función de las necesidades. Igual que un framework, ASP.NET AJAX está compuesto por miles de clases, organizadas en grupos llamados espacios de nombres (o packages, en Java). Se trata de nociones idénticas a las clases y a los espacios de nombres .NET de C#.

El uso de un **ScriptManager** en una página provee el soporte del framework AJAX. La clase **Type** sirve para declarar espacios de nombres:

```

<asp:ScriptManager ID="scriptManager1" runat="server">
</asp:ScriptManager>

```

```

<script language="javascript">
    // declaración de un namespace
    Type.registerNamespace("agenda");

```

El espacio de nombres **agenda** es una declaración a la que pueden asociarse las clases. Es, precisamente, el caso de la clase **Persona** cuya definición se muestra a continuación:

```

// declaración de una clase
agenda.Persona = function(nombre, apellido) {

```

```

// atributos (privados)

var _nombre = nombre, _apellido = apellido, _edad;

// accesos (públicos)

this.set_nombre = function (value) {
    _nombre = value;
}

this.get_nombre = function () {
    return _nombre;
}

this.set_apellido = function (value) {
    _apellido = value;
}

this.get_apellido = function () {
    return _apellido;
}

```

El lector observará el uso de accesos para imitar la noción de propiedad C#. Los atributos, privados, se declaran mediante la instrucción **var**. Los accesos **get\_** y **set\_** se definen en el cuerpo de la clase (se habla de **closure** en lengua inglesa), que en realidad es un constructor:

```
agenda.Persona = function(nombre, apellido) { ... }
```

Expicaremos esta sintaxis. La función anónima recibe dos argumentos - **nombre**, **apellido** - y su definición se describe entre llaves. No es, realmente, anónima, puesto que su referencia se asocia a la clase **agenda.Persona**. Se podría escribir en C#:

```

namespace agenda
{
    class Persona
    {

        Public Persona(object nombre, object apellido)
        {
        }
    }
}
```

El código JavaScript entre llaves representa instrucciones o declaraciones. Las declaraciones son variables (llamadas atributos o campos), o funciones (llamadas métodos). Las instrucciones se ejecutan cuando se instancia la clase, como ocurre con cualquier constructor. En general, se limita a la inicialización de campos del nuevo objeto, ejerciendo el rol clásico de un constructor.

Se muestra, a continuación, una sintaxis que permite definir métodos y exponerlos al exterior de la clase:

```
// métodos públicos  
agenda.Persona.prototype.presentarse = function () {  
    alert(this.get_nombre() + " " + this.get_apellido());  
}
```

La palabra reservada **prototype** agrupa el conjunto de métodos de la clase. Siguiendo la lógica anterior, es posible asociar una función (anónima) con el nombre **presentarse** al prototipo de la clase **agenda.Persona**.

A continuación, es posible instanciar dicha clase y utilizar el objeto resultante:

```
// instanciación  
var persona2 = new agenda.Persona("Simón", "López");  
alert(persona2.get_nombre() + " " + persona2.get_apellido());  
persona2.presentarse();
```

Generalmente, registraremos la clase desde el framework AJAX:

```
// registro de la clase  
agenda.Persona.registerClass("agenda.Persona");  
alert(Object.getTypeName(persona2)); // requiere que la clase esté  
registrada
```

Esto tendrá como efecto inscribir la clase en una jerarquía y aprovechará el soporte del framework, tal y como descubriremos a continuación.

### c. El estilo AJAX

El framework proporciona, también, una sintaxis optimizada. La clase está siempre compuesta por un constructor, campos y métodos, pero el estilo evoluciona:

```
Type.registerNamespace("pantalla");  
  
// constructor  
pantalla.Punto = function (x, y) {  
    // datos privados  
    this._x = x;  
    this._y = y;  
}  
  
// definición desviada de los métodos públicos
```

```

function pantalla$Point$set_x(value) {
    this._x = value;
}

function pantalla$Point$get_x() {
    return this._x;
}

function pantalla$Point$set_y(value) {
    this._y = value;
}

function pantalla$Point$get_y() {
    return this._y;
}

// composición de métodos públicos
pantalla.Punto.prototype = {
    set_x: pantalla$Point$set_x,
    get_x: pantalla$Point$get_x,
    set_y: pantalla$Point$set_y,
    get_y: pantalla$Point$get_y
};

// registro de la clase

```

```
pantalla.Punto.registerClass("pantalla.Punto");
```

Esta sintaxis resulta más legible y más rápida de ejecutar para el framework. No obstante, la instanciaión y el uso de objetos no cambian:

```

// instanciaión y prueba
var p = new pantalla.Punto(5, 10);
alert(p.get_x());

```

#### d. Clases derivadas

El framework AJAX soporta la derivación (herencia) de clases. La función **registerClass** permite asociar la clase base (Punto) con la clase derivada (PuntoColor):

```

// clase derivada
pantalla.PuntoColor = function (x, y, color) {
    pantalla.PuntoColor.initializeBase(this, [x, y]);
    this._color = color;

```

```

}

pantalla.PuntoColor.prototype.set_color = function(value) {
    this._color = value;
}

pantalla.PuntoColor.prototype.get_color = function() {
    return this._color;
}

// asociación de la clase PuntoColor como derivada de Punto
pantalla.PuntoColor.registerClass("pantalla.PuntoColor", pantalla.Punto);

```

El lector habrá observado la llamada a **initializeBase** como primera instrucción del constructor, que recuerda, naturalmente, a la palabra reservada C# `base()`. La referencia `this` transmite para que el constructor de la clase madre pueda atender los atributos que debe inicializar. Los valores de los parámetros requeridos por el constructor se agrupan en una tabla, de ahí la sintaxis con corchetes [ ].

#### e. Implementar interfaces

JavaScript debe hacer frente a las mismas restricciones que otros lenguajes orientados a objetos y, por tanto, la implementación de interfaces resulta algo necesario. Recordemos que una interfaz es una clase cuyos métodos no contienen un cuerpo; es la clase la que implementa las instrucciones que componen dichos métodos. Tomemos como ejemplo la interfaz **IDisposable** que exige la implementación de un método `Dispose`. El mecanismo se realiza en dos etapas: en primer lugar la definición del método, respetando la firma impuesta por la interfaz y, a continuación, la declaración de la implementación de la interfaz:

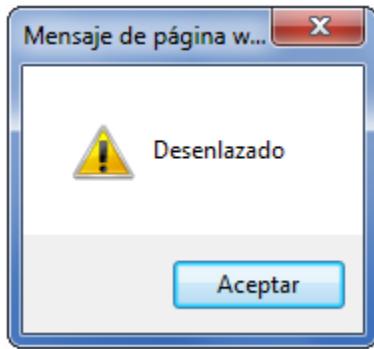
```

// versión con la implementación de una interfaz
pantalla.PuntoColor.prototype.dispose = function () {
    alert("Desenlazado");
}

// declaración de la implementación, derivando de Punto
pantalla.PuntoColor.registerClass("pantalla.PuntoColor",
pantalla.Punto, Sys.IDisposable);

// instanciación y prueba
var pc = new pantalla.PuntoColor(1, 2, "#00FF00");
alert(pc.get_x() + " " + pc.get_y() + " " + pc.get_color());
pc.dispose();

```



## 7. El modelo de extensión AJAX

El framework AJAX se ha propuesto, inicialmente, como una extensión al framework ASP.NET 2.0. Desde la versión 3.5, Microsoft ha tratado de hacer converger ambos modelos y la versión ASP.NET 4 realiza una integración completa.

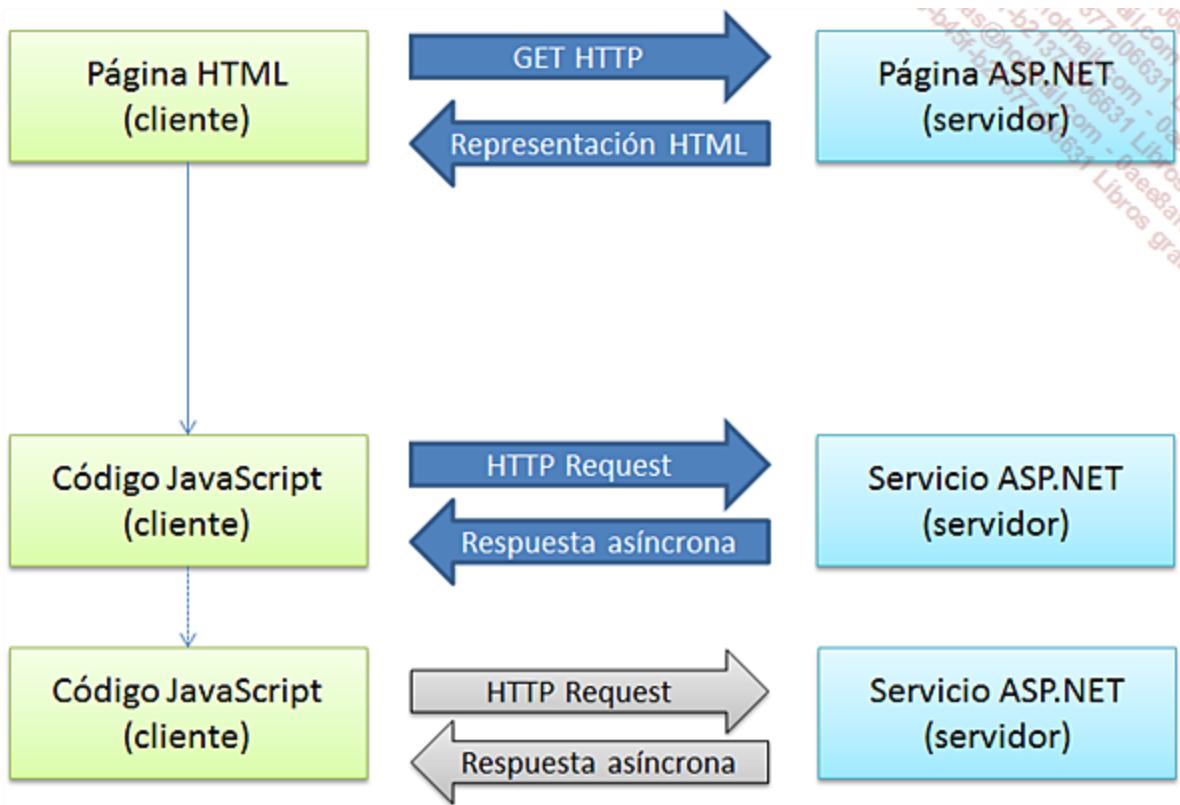
### a. Estructura del framework

Con el objetivo de simplificar el descubrimiento del framework AJAX, recordamos que el navegador (cliente) no conoce más que algunas pocas tecnologías muy simples y limitadas: HTML-CSS, JavaScript, DOM (*Document Object Model*). JavaScript permite, recientemente, el acceso a intercambios mediante una tarea de fondo (llamada asíncrona) por medio del objeto **XMLHttpRequest**.

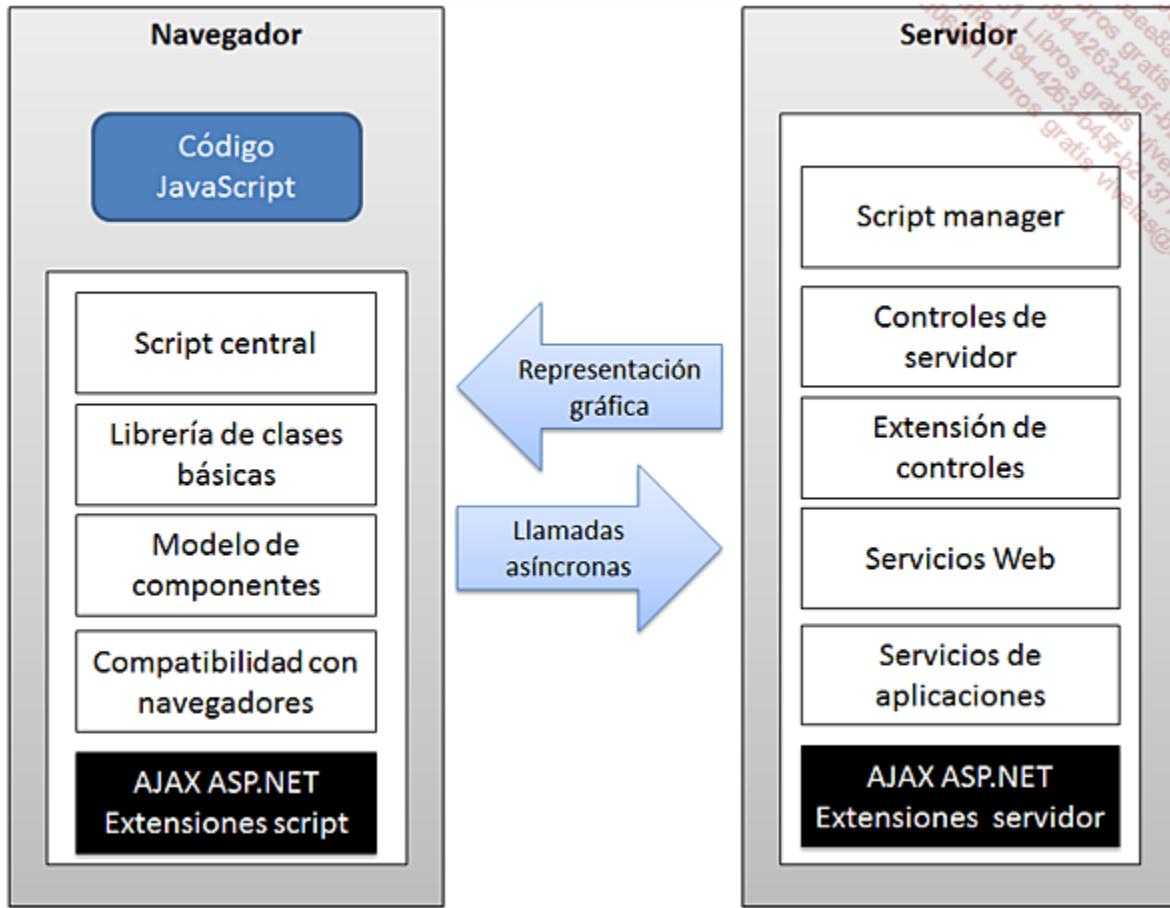
El funcionamiento habitual es el siguiente: el navegador solicita al servidor una URL (GET), que le responde como de costumbre. El framework ASP.NET prevé que la respuesta contenga un formulario HTML <form> que se volverá a enviar (postback) indefinidamente sobre la misma URL, para avanzar en el contenido de la página. El proceso se detiene cuando el usuario sigue algún enlace de hipertexto que le lleva a una página diferente o cuando el servidor realiza, él mismo, una redirección (Response.Redirect). Hemos visto anteriormente que este modelo, muy potente, resultaba también algo pesado de implementar y la experiencia de usuario exige el uso de modelos más ligeros, más reactivos: los callbacks y los postback parciales.

El enfoque del framework AJAX parte del mismo principio, el refresco completo de una página supone un ciclo costoso que conviene limitar al máximo. Gracias al apoyo de componentes de cliente más autónomos, a servicios Web y a peticiones como tarea de fondo (asíncronas), es posible trabajar con componentes de usuario mucho menos costosos.

La lógica AJAX completa, de manera natural, a la introducida por el framework ASP.NET. Tras la carga completa de una página (representación), el framework AJAX toma el relevo y desencadena una serie de operaciones asíncronas hacia el servidor. Es decir, ya no resulta interesante incluir lógica funcional en las páginas; ésta se maneja, a menudo, desde servicios web ASMX o REST.



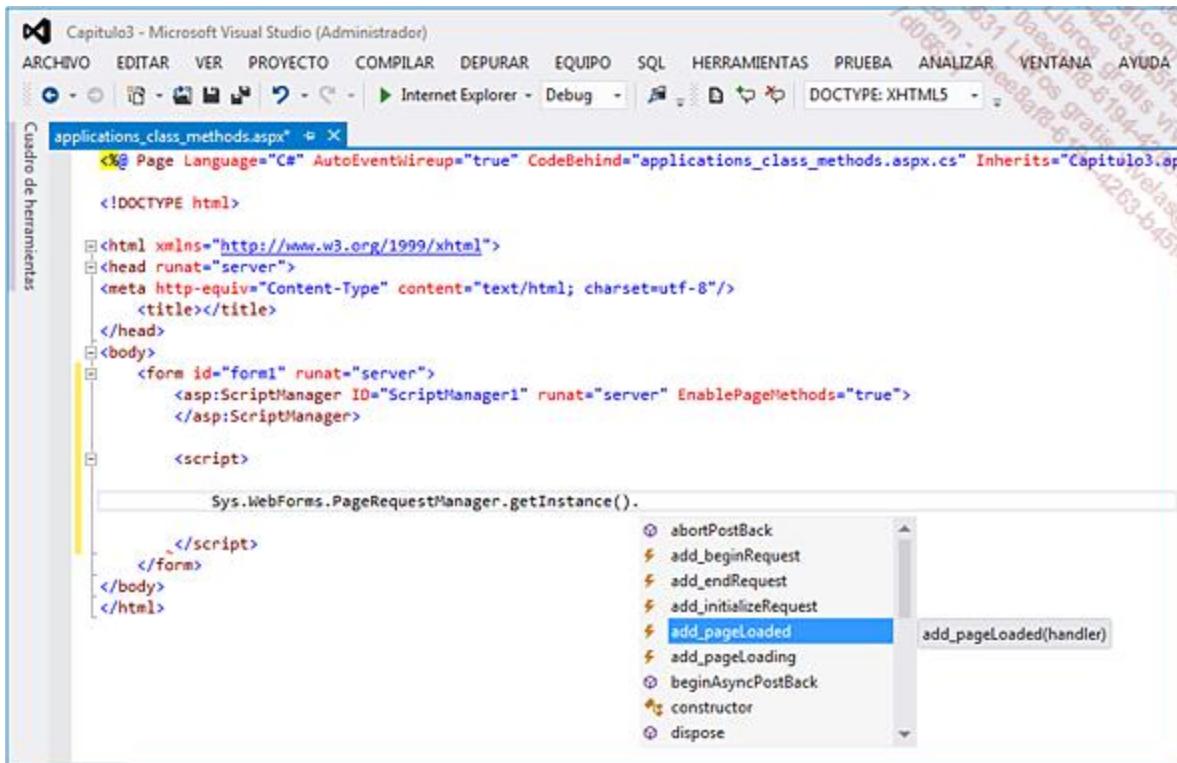
Los diseñadores de Microsoft han decidido ahorrar a los desarrolladores AJAX las trabas encontradas en los frameworks existentes; sin un enfoque estructurado de los componentes, de los comportamientos, representaciones HTML eficaces... lo esencial del código JavaScript se centra en problemáticas de serialización y de transmisión, y el código resultante es muy difícil de capitalizar y de mantener. El framework AJAX ASP.NET es, por tanto, un modelo completo:



### b. La clase aplicación

Ya hemos tenido la ocasión de familiarizarnos con ciertos elementos de este framework: el controlador de script, los servicios web, el modelo de componentes... Veamos, a continuación, qué aporta la clase aplicación. Se trata de un conjunto de métodos asociados a la clase **PageRequestManager** y que controlan el ciclo de vida AJAX.

Esta clase incluye eventos accesibles a través del nombre `Sys.WebForms.PageRequestManager.getInstance()`:



Los eventos siguientes resultan muy útiles para controlar este ciclo de vida:

beginRequest	Se produce tras initializeRequest pero antes de que se ejecute la petición HTTP.
endRequest	Se produce al finalizar la petición.
initializeRequest	Se produce antes de que comience una llamada asíncrona. La llamada puede bloquearse gracias a la propiedad Cancel del objeto <code>Sys.WebForms.InitializeRequestEventArgs</code> que se pasa como argumento al controlador de eventos.
pageLoaded	Interviene cuando la página ha terminado su carga.
pageLoading	Interviene cuando se carga la página.

El siguiente ejemplo nos muestra cómo interceptar estos eventos. Se trata de anotar en un registro la supervisión de cada uno de ellos:

```

<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"
        EnablePageMethods="True">
    </asp:ScriptManager>

    <script type="text/javascript">
        // método que permite anotar un mensaje en un registro
        function log(msg) {

```

```

        $get("registro").innerHTML += msg + "<br>";

    }

    // registro de los controladores de eventos
    // preste atención, si existe una función pageLoad() pues será
    // invocada automáticamente por el framework
    // con el evento pageLoaded.
    // No es preciso registrarla, si no, se invocará
    // dos veces.
    // Sys.WebForms.PageRequestManager.getInstance()
    .add_pageLoaded(pageLoad);

Sys.WebForms.PageRequestManager.getInstance().add_initializeRequest
(initializeRequest);
Sys.WebForms.PageRequestManager.getInstance().add_beginRequest
(beginRequest);
    Sys.WebForms.PageRequestManager.getInstance().add_endRequest
(endRequest);

    // implementación de los controladores
    function pageLoad() {
        log("pageLoad");
    }

    function initializeRequest(sender,e) {
        log("initializeRequest");
    }

    function beginRequest() {
        log("beginRequest");
    }

    function endRequest() {
        log("endRequest");
    }

}

</script>

```

```

<asp:UpdatePanel ID="panel_hora" runat="server">
    <ContentTemplate>
        <asp:Label ID="lbl_hora" runat="server" Font-
Bold="True"></asp:Label>
        <br />
        <asp:Button ID="cmd_actualizar" runat="server" Text="Actualizar"
            onclick="cmd_actualizar_Click" />
    </ContentTemplate>
</asp:UpdatePanel>

<hr />
<div id="registro"></div>
</form>

```

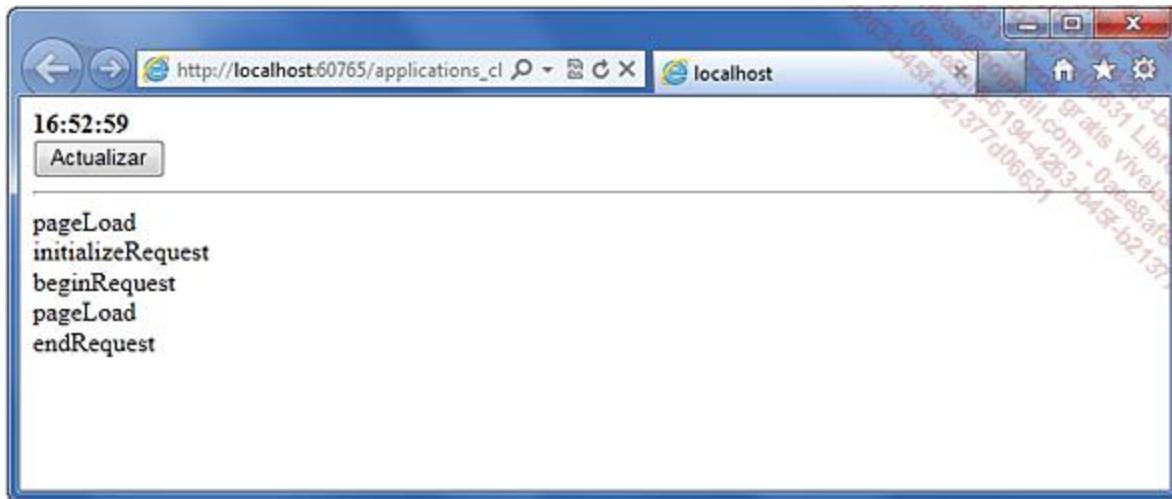
El código de evento cmd\_actualizar\_Click se resume en una simple actualización de la hora:

```

protected void cmd_actualizar_Click(object sender, EventArgs e)
{
    lbl_hora.Text = DateTime.Now.ToString("HH:mm:ss");
}

```

Tras la ejecución del programa, se constata que los eventos se producen en el orden previsto:

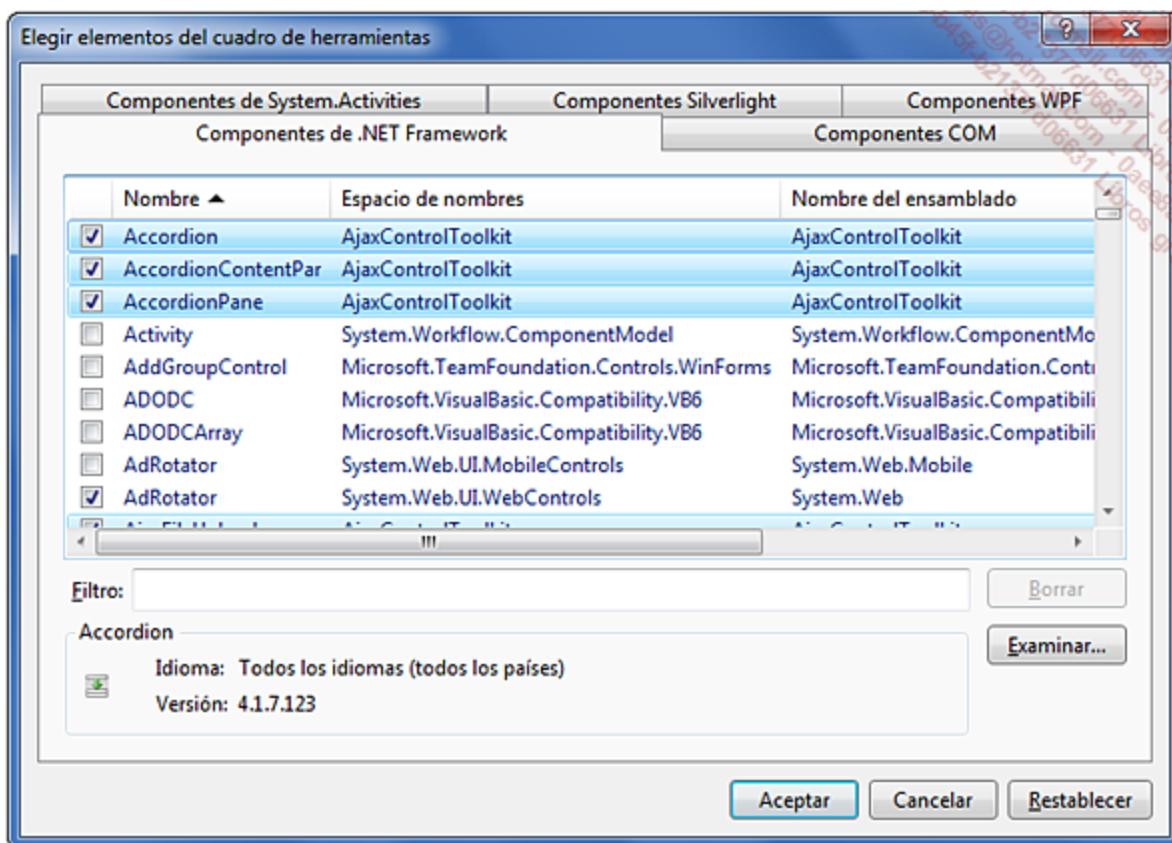


### c. Los controles AJAX del toolkit

El sitio web [www.codeplex.com](http://www.codeplex.com) publica un kit de componentes adicionales para el framework AJAX ASP.NET. Es posible encontrarlo, también, en el sitio <http://ajax.asp.net>. Llamado inicialmente "atlas", este kit se llama, en lo sucesivo, "ajax toolkit".

El uso del kit es muy sencillo; basta con agregar una referencia a la DLL ajax-Toolkit.dll, bien como referencia privada (en la carpeta bin), o bien como referencia compartida (en el GAC).

Por comodidad, los controles de la DLL se agregan al cuadro de herramientas en una pestaña particular.

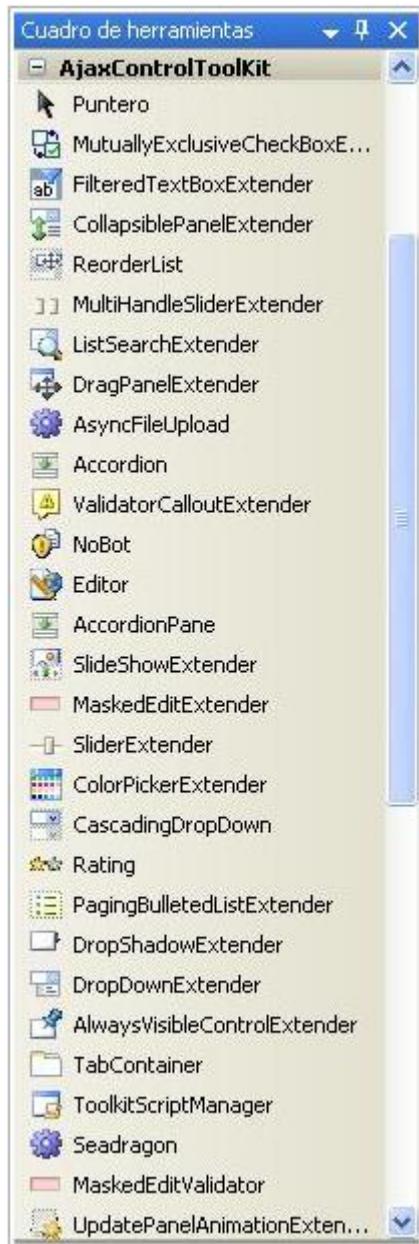


El juego de componentes provee distintos tipos de control:

- Controles adicionales.
- Dos extensores de controles.

Los extensores de controles permiten incluir nuevos comportamientos a los controles existentes, o nuevas propiedades.

He aquí una muestra de los componentes disponibles:



Los controles cuyo nombre termina por Extender son extensores.

A modo de ejemplo, veamos cómo guiar al usuario en una lista deslizante. El comportamiento estándar HTML sólo prevé una selección basada en la primera letra de las distintas opciones posibles. El control ListExtender va a permitir a un componente <asp:DropDownList/> tener un comportamiento algo más refinado:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="lista_completa.aspx.cs" Inherits="lista_completa" %>
<%@ Register Assembly="AJAXControlToolkit"
Namespace="AJAXControlToolkit"

```

```
TagPrefix="ajaxToolkit" %>
```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <div>
            <asp:DropDownList ID="lst_ciudades" runat="server" Width="200px">
                <asp:ListItem>Madrid</asp:ListItem>
                <asp:ListItem>Barcelona</asp:ListItem>
                <asp:ListItem>Valencia</asp:ListItem>
                <asp:ListItem>Bilbao</asp:ListItem>
                <asp:ListItem>Sevilla</asp:ListItem>
                <asp:ListItem>Vigo</asp:ListItem>
                <asp:ListItem>Santander</asp:ListItem>
                <asp:ListItem>León</asp:ListItem>
                <asp:ListItem>Salamanca</asp:ListItem>
            </asp:DropDownList>
            <br />
            <br />
            <ajaxToolkit:ListSearchExtender ID="ListSearchExtender1"
runat="server" TargetControlID="lst_ciudades">
            </ajaxToolkit:ListSearchExtender>
        </div>
    </form>
</body>
</html>

```

El atributo `TargetControlID` determina a qué control se aplicará el comportamiento adicional introducido por el extensor de la lista; no es necesario realizar ninguna programación adicional para que funcione este ejemplo.

#### d. Definir controles personalizados en JavaScript

El framework AJAX es, por naturaleza, un framework extensible; los desarrolladores del kit de componentes han aprovechado esta posibilidad para construir su juego de controles.

Un componente es una clase JavaScript que deriva de `Sys.UI.Control`.

```
Type.registerNamespace("ComponentesPersonalizados");
ComponentesPersonalizados.BotonHover.registerClass
('ComponentesPersonalizados.BotonHover',
 Sys.UI.Control);
```

Esto significa que hereda todos los métodos, propiedades y todos los eventos de dicha clase. No obstante, el desarrollador tiene la libertad de agregar más. Veamos el caso de un botón que reaccione cuando se pasa el ratón por encima (`handle over` y `unhandle over`). Para ser más claros, responderá a tres tipos de evento: `click`, `mouseOver`, `mouseOut`. Como ocurre en programación .NET, el framework AJAX utiliza la noción de delegado para gestionar los controladores de eventos. Esto nos ofrece el siguiente código:

```
// Constructor
ComponentesPersonalizados.BotonHover= function (element) {

    ComponentesPersonalizados.BotonHover.initializeBase
(this, [element]);

    // miembros privados: controladores de eventos
    this._clickDelegate = null;
    this._mouseOverDelegate = null;
    this._mouseOutDelegate = null;
}
```

El prototipo contiene, de por sí, accesores para sus propiedades (eventos), a las que se agrega una propiedad `text` que no requiere campo privado en la medida en que el elemento HTML asociado al botón ya expone una propiedad `innerHTML`:

```
// Texto del botón
get_text: function () {
    return this.get_element().innerHTML;
},
set_text: function (value) {
    this.get_element().innerHTML = value;
},
// Agregar y suprimir el controlador de evento click
add_click: function (handler) {
```

```

        this.get_events().addHandler('click', handler);
    },
    remove_click: function (handler) {
        this.get_events().removeHandler('click', handler);
    },
    // Agregar y suprimir el controlador de evento mouseOver
    add_mouseOver: function (handler) {
        this.get_events().addHandler('mouseOver', handler);
    },
    remove_mouseOver: function (handler) {
        this.get_events().removeHandler('mouseOver', handler);
    },
    // Agregar y suprimir el controlador de evento mouseOut
    add_mouseOut: function (handler) {
        this.get_events().addHandler('mouseOut', handler);
    },
    remove_mouseOut: function (handler) {
        this.get_events().removeHandler('mouseOut', handler);
    },

```

El constructor de aplicación - el método `initialize` - precisa que el componente reaccione frente a ciertos eventos HTML. Siendo rigurosos, deberíamos invocar a los elementos del botón `hover`, `unhover` y `click`, pero no existe ninguna limitación técnica a este respecto. Escogemos, por tanto, nombres distintos para que el lector pueda distinguir ambos niveles:

```
// constructor de aplicaci&oa
```

# El enfoque MVC

Una vez pasada la época en la que se discutía la estructura de una aplicación web, el universo Java ha popularizado el uso de frameworks tales como Struts o Spring. Éste, y Struts en primer lugar, han sentado las bases de una separación de responsabilidades entre los distintos niveles de una aplicación web. Es cierto que las primeras tecnologías web no invitaban a los programadores a organizar sus aplicaciones; el mantenimiento se vuelve muy delicado, al tiempo que el rendimiento es ridículo.

## 1. El patrón de diseño MVC

La expresión MVC se refiere a un enfoque de diseño generalizado, o patrón de diseño. El objetivo consiste en no reinventar la rueda con cada aplicación. Como veremos, el MVC es un patrón bastante simple. No utilizarlo supone, realmente, dirigirse hacia una aplicación complicada y, por tanto, mal hecha, lo que nos recuerda al pasado tal y como veíamos antes.

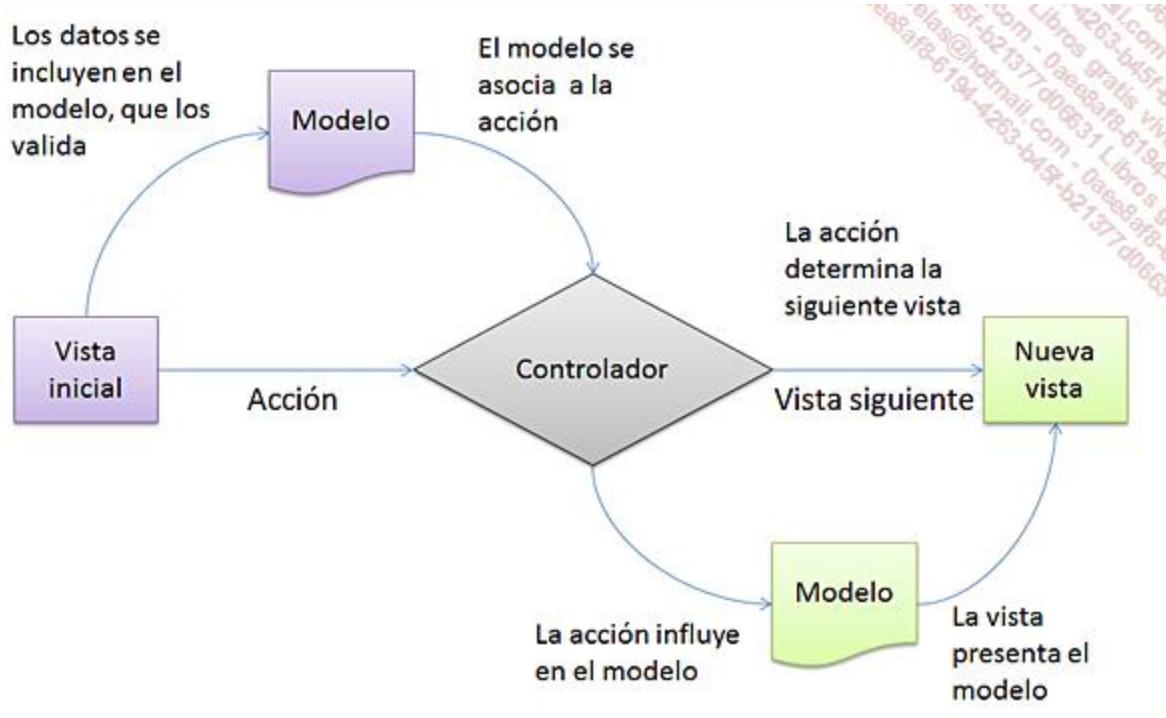
Cada letra del acrónimo MVC se corresponde con un rol bien definido; el modelo, la vista y el controlador.

El modelo es un objeto "de negocio" que agrupa sus datos, su comportamiento (métodos) y sus reglas de validación. No contiene, por lo general, ninguna lógica técnica (presentación, navegación). Es posible atribuirle aspectos (inyección de servicios tales como persistencia de archivos o SQL, transacciones, seguridad...). En los enfoques menos completos, el objeto de negocio se asocia con una clase de servicios que sirve de interfaz (API).

La vista se encarga de restituir el modelo en el seno de una interfaz gráfica (web, en nuestro caso), y permite al usuario interactuar con el modelo.

El controlador define las reglas de navegación (también llamada la cinemática). El paso de una vista a otra se realiza mediante acciones dirigidas por un controlador. El modelo se interroga, o enriquece, para condicionar el desarrollo de acciones.

La siguiente ilustración describe la secuencia de interacciones entre estos objetos:



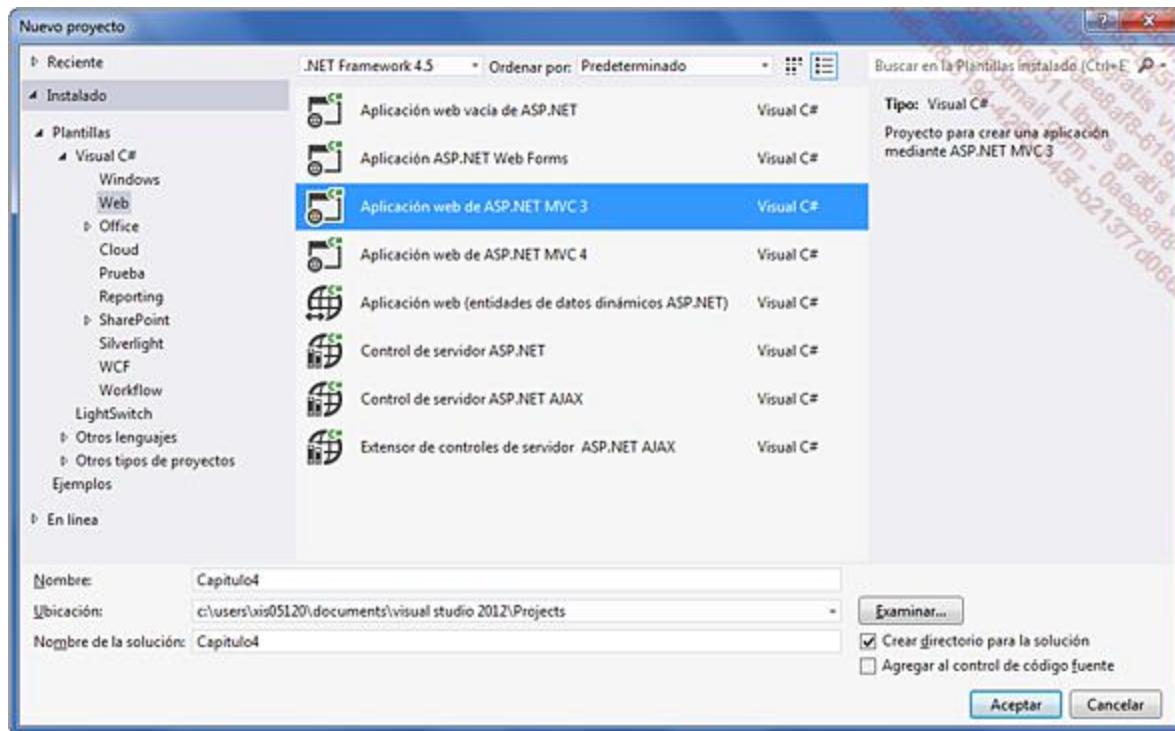
## 2. Evolución de MVC

El enfoque MVC 2 es, principalmente, una evolución del framework; consiste en utilizar únicamente un controlador para varias acciones. Esta evolución reduce considerablemente el esfuerzo en cuanto a programación y a configuración. Por suerte, el framework ASP.NET soporta, en lo sucesivo, el nivel MVC 2 mediante MVC 3 y MVC 4.

# Los sitios ASP.NET MVC

## 1. Creación de un sitio

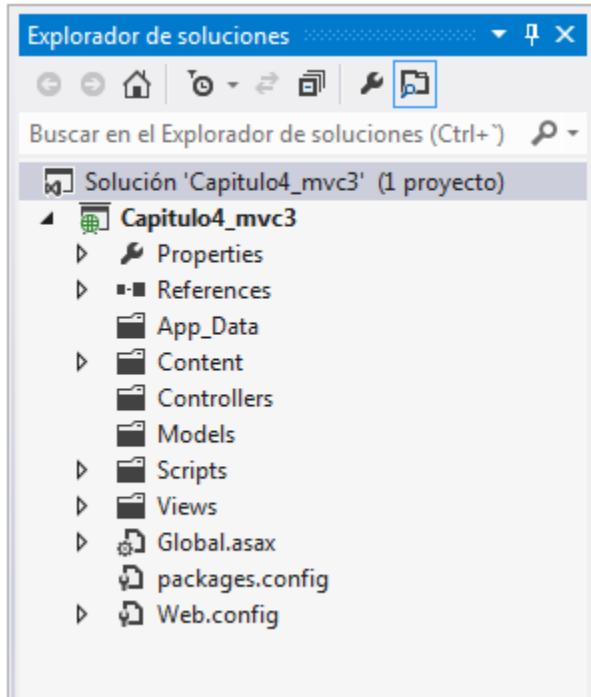
Visual Studio 2012 provee dos modelos de proyecto para MVC según el nivel de soporte MVC 3 o MVC 4. Escogeremos el nivel MVC 3 para nuestro ejemplo:



Como la aplicación MVC requiere el uso de clases que no están en el código subyacente (como con los Web Forms), el modelo Visual Studio está disponible únicamente para un proyecto web, y no para un sitio web.

## 2. Organización de carpetas

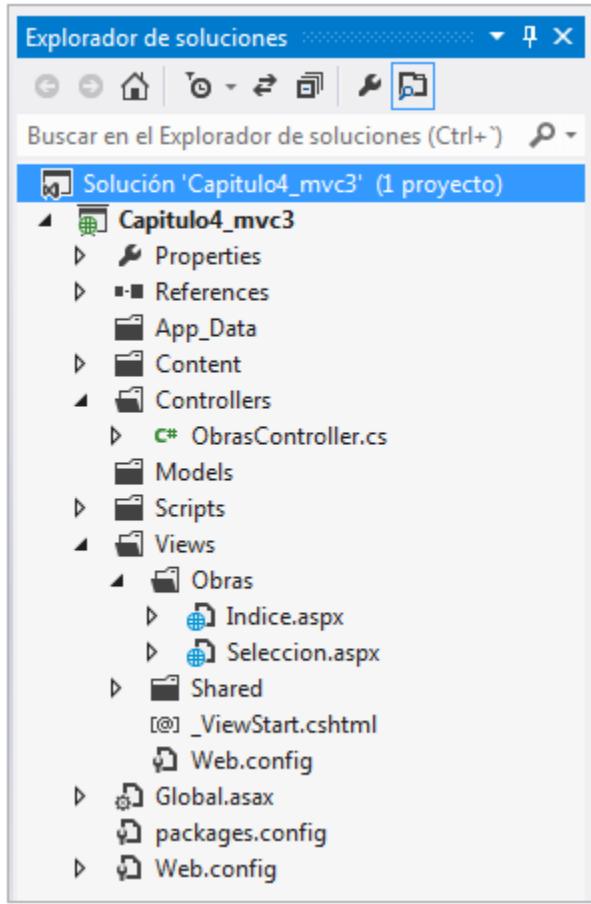
La solución del proyecto web contiene muchas más carpetas que un proyecto Web Forms.



Estas carpetas tienen, como objetivo, guiar al programador:

Content	Contiene las hojas de estilo CSS y demás recursos compartidos.
Controllers	Agrupa los controladores destinados a procesar las acciones.
Models	Agrupa los modelos que son componentes funcionales.
Scripts	Conjunto de módulos JavaScript, jQuery y AJAX.
Views	Páginas .aspx.

Las vistas son páginas ASPX, aunque no tienen código subyacente (como veremos a continuación). Se agrupan, en principio, en carpetas llamadas zonas, las cuales se corresponden con controladores. Esta regla no tiene ningún carácter obligatorio desde un punto de vista técnico, aunque es más sencillo utilizar el framework si nos ceñimos a ella.



### 3. Creación del modelo

Un modelo es una clase cuyas instancias se denominan "objetos de negocio". Esto significa que no contiene ninguna lógica técnica, y que el framework se encarga de gestionar el ciclo de vida del componente de negocio, aportándole servicios técnicos de alto nivel tales como la seguridad, las transacciones, la validación, la persistencia...

```
#region Modelo Obra
    /// <summary>
    /// Objeto de negocio Obra
    /// </summary>

    public class Obra
    {
        /// <summary>
        /// Por convención, ID es una clave primaria
        /// </summary>
        public int ID { get; set; }
```

```
    /// <summary>
    /// Campo Autor
```

```

/// </summary>
public string Autor { get; set; }

/// <summary>
/// Campo Título
/// </summary>
public string Titulo { get; set; }

public Obra()
{
    ID = 0;
    Autor= "";
    Titulo = "";
}

public Obra(int id, string autor, string titulo)
{
    ID = id;
    Autor = autor;
    Titulo = titulo;
}

}

#endregion

```

Hemos asociado una clase de servicio que contiene algunos métodos ineludibles. Las clases de servicio implementan bastante a menudo el patrón CRUD (*Create Read Update Delete*). Nosotros obviaremos esta práctica, sin implementar la persistencia en base de datos SQL, para centrarnos en la arquitectura MVC.

```

#region ObraServicio
/// <summary>
/// Clase de servicio para Obra
/// </summary>
public class ObraServicio
{
    /// <summary>

```

```

    /// Devuelve una lista de obras
    /// </summary>
    /// <returns></returns>
    public List<Obra> Select()

```

```

{
    List<Obra> l = new List<Obra>();
    l.Add(new Obra(1, "Brice-Arnaud", "ASP.NET 4.5 en
C#"));
    l.Add(new Obra(2, "Brice-Arnaud", "ASP.NET 4.5 en
VB.NET"));

    return l;
}

/// <summary>
/// Busca una obra a partir de su clave primaria
/// </summary>
/// <param name="id">Clave primaria</param>
/// <returns></returns>
public Obra Select(int id)
{
    // utiliza LINQ para realizar la búsqueda
    // también podríamos usar un bucle FOR
    var q = from o in Select() where o.ID == id select o;
    return q.ToArray()[0];
}

public void Insert(Obra obra)
{
    // ...
}

public void Update(Obra obra)
{
    // ...
}

```

```

public void Delete(int id)
{
    // ...
}

```

```
#endregion
```

## 4. Definición del controlador

Un controlador es una clase que deriva de **Controller**. Esta clase se activa mediante una URL que se configura a través de rutas. Estas URL se muestran mediante comentarios en el siguiente código:

```
public class ObrasController : Controller
{
    // URL de acceso:
    // GET: /Obras/
    /// <summary>
    /// Acción Índice.
    /// Lleva a la vista Índice encargada de mostrar el listado
    /// de las obras
    /// </summary>
    /// <returns>Vista en curso</returns>
    public ActionResult Index()
    {
        var p = new ObraServicio();

        // modelo
        var obras = p.Select();

        // la vista se encarga de representar el modelo de obras
        return View(obras);
    }

    // URL de acceso:
    // GET: /Obras/Selección/3
    /// <summary>
```

```
    /// Acción Selección.
    /// Lleva a la vista Selección encargada de mostrar el detalle
    /// de una obra
    /// </summary>
    /// <param name="index">clave primaria</param>
    /// <returns></returns>
    public ActionResult Seleccion(int? id)
    {
```

```

var p = new ObraServicio();

// modelo correspondiente a la búsqueda
var obra = p.Select(id.HasValue ? id.Value : 1);

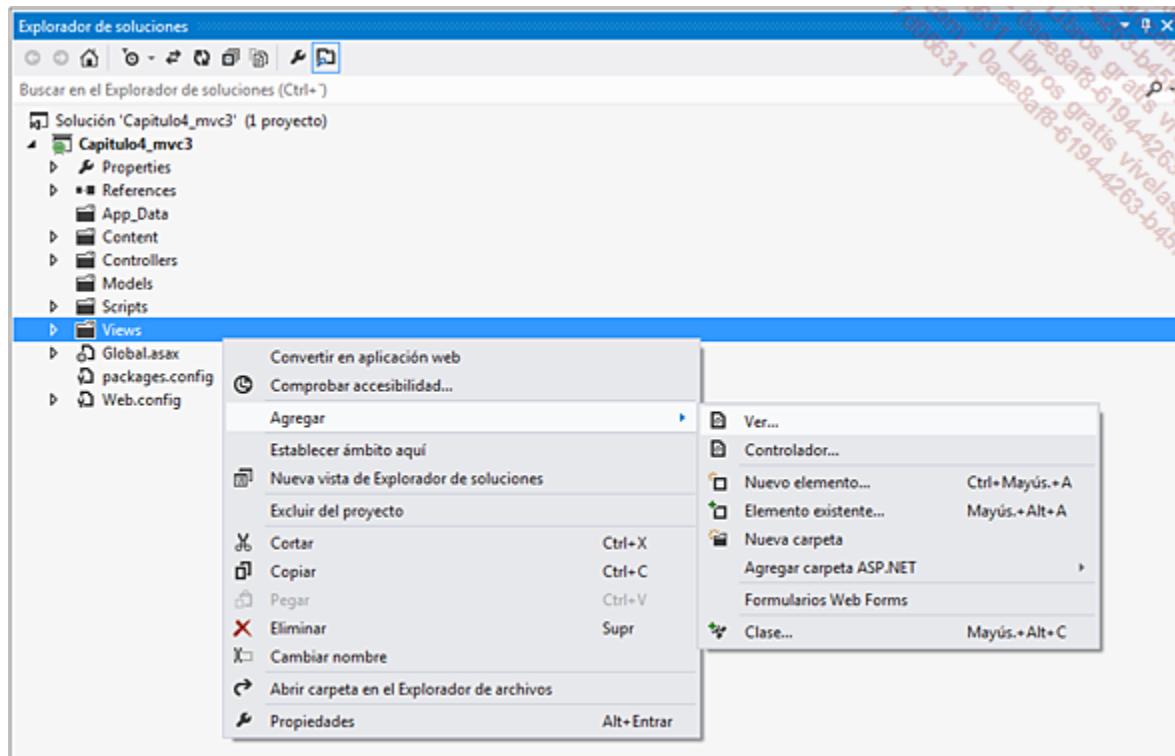
// la vista se encarga de representar el modelo obra
return View(obra);
}

}

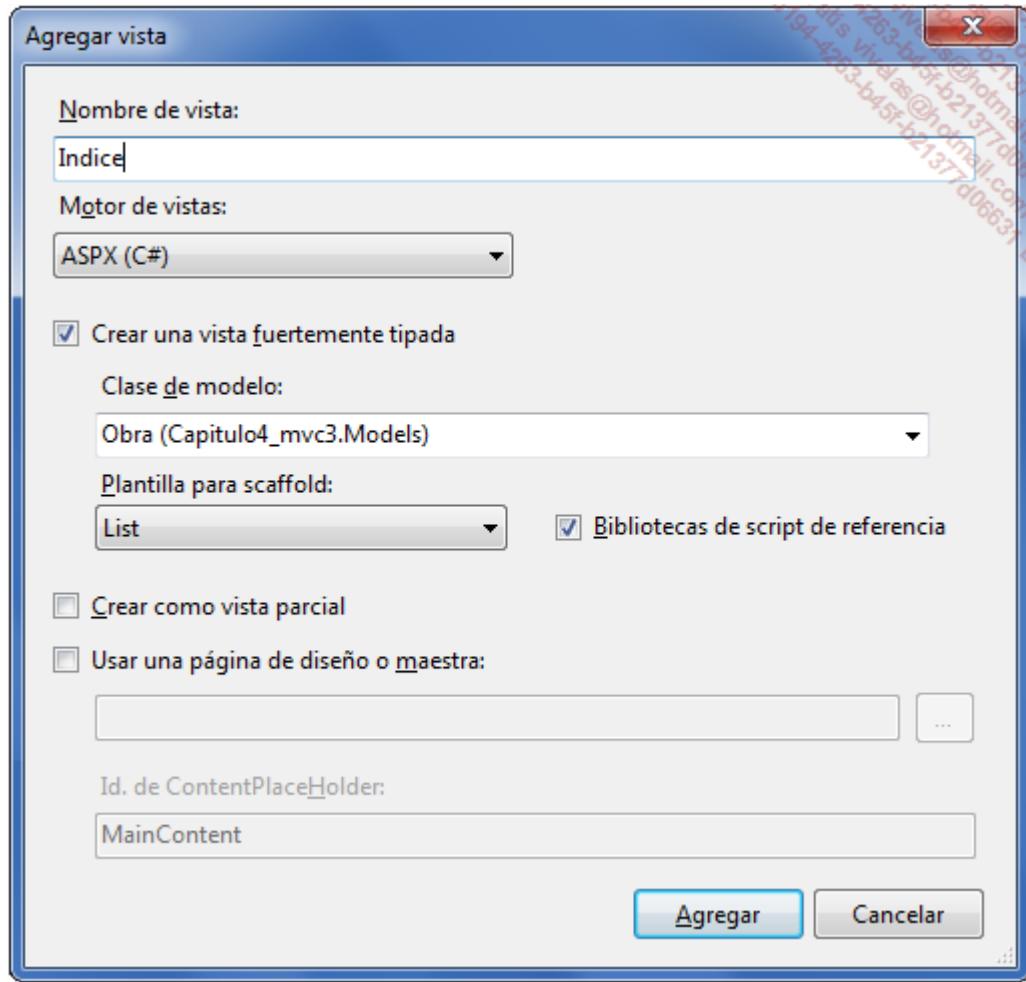
```

## 5. Agregar vistas

Visual Studio provee un asistente que permite crear nuevas vistas a partir de la información de un modelo y según las indicaciones del programador.



Sólo es obligatorio informar el nombre de la vista.



La siguiente información es opcional:

Crear una vista parcial	Genera un archivo .ascx en lugar de una página .aspx.
Crear una vista fuertemente tipada	Tras seleccionar la clase modelo, Visual Studio utiliza la reflexión para preparar un formulario basado en el nombre de las propiedades de la clase modelo.
Contenido de la vista	Visual Studio utiliza el valor de esta lista para preparar un diseño HTML destinado a iterar o a detallar los valores del modelo.
Selección de una página maestra	Asocia la página .aspx a una página .master y logra representar el contenedor indicado.
Motor de vista	La vista puede implementarse mediante etiquetas webform ASPX y código C# o bien a partir del nuevo motor Razor.

La

implementación de la vista Indice resulta bastante sencilla:

```
<%@ Page Language="C#" 
Inherits="System.Web.Mvc.ViewPage<IEnumarable<capitulo4_mvc3.Model
s.Obra>>" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
```

```

<title>Indice</title>
</head>
<body>
    <table>
        <tr>
            <th></th>
            <th>
                Autor
            </th>
            <th>
                Titulo
            </th>
        </tr>

        <% foreach (var item in Model) { %>

            <tr>
                <td>
                    <%: Html.ActionLink("Detalle", "Seleccion", new {
id=item.ID })%> |
                </td>
                <td>
                    <%: item.Autor %>
                </td>
                <td>
                    <%: item.Titulo %>
                </td>
            </tr>

        <% } %>

        </table>
    </body>
</html>

```

En nuestro caso, la vista Vista hereda de Modelo. Este enfoque difiere de las prácticas Java, aunque es idéntica a la arquitectura Web Form. Destaquemos, también, que la clase de base es un tipo genérico configurado mediante un modelo particular:

```
System.Web.Mvc.ViewPage<IEnumerable<capitulo4_mvc3.Models.Obra>>
```

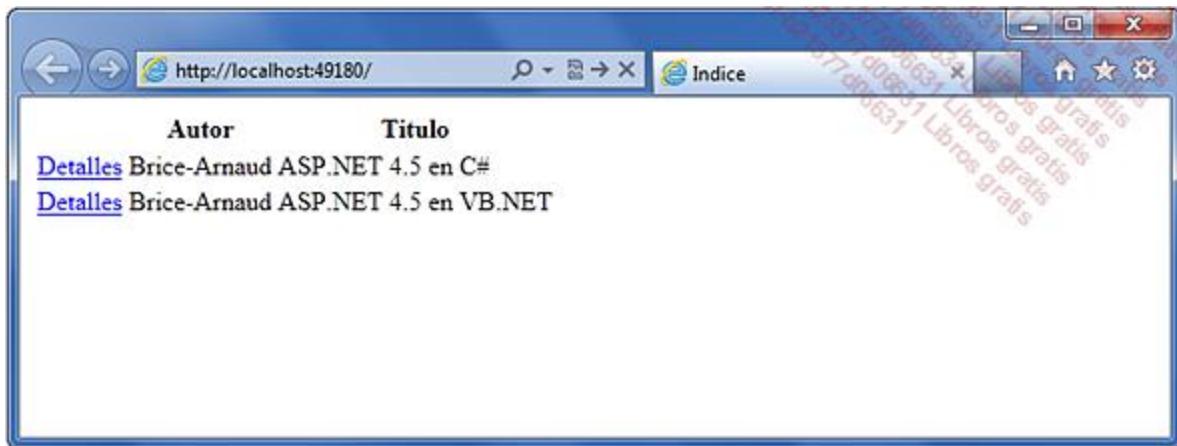
Tratándose de una lista, la vista realiza una iteración mediante un scriptlet `for`. Deducimos, de aquí, la existencia de una variable `Model` de tipo `IEnumerable<Obra>`:

```
<% foreach (var item in Model) { %>
```

El objeto HTML expone el método `ActionLink` que genera un enlace correspondiente a la acción `Selección`. El tercer parámetro es el valor del atributo `ID`, que sirve como clave primaria. El primer parámetro es, simplemente, el texto del enlace:

```
<%: Html.ActionLink("Detalle", "Seleccion", new { id=item.ID })%> |
```

En tiempo de ejecución, comprobamos que el enlace generado se corresponde con el formato de la URL descrito en la clase controlador.



## Definición de las rutas

El archivo Global.asax contiene la definición de las rutas; una ruta es una URL que asocia un controlador, una acción y parámetros opcionales:

```
public class MvcApplication : System.Web.HttpApplication
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

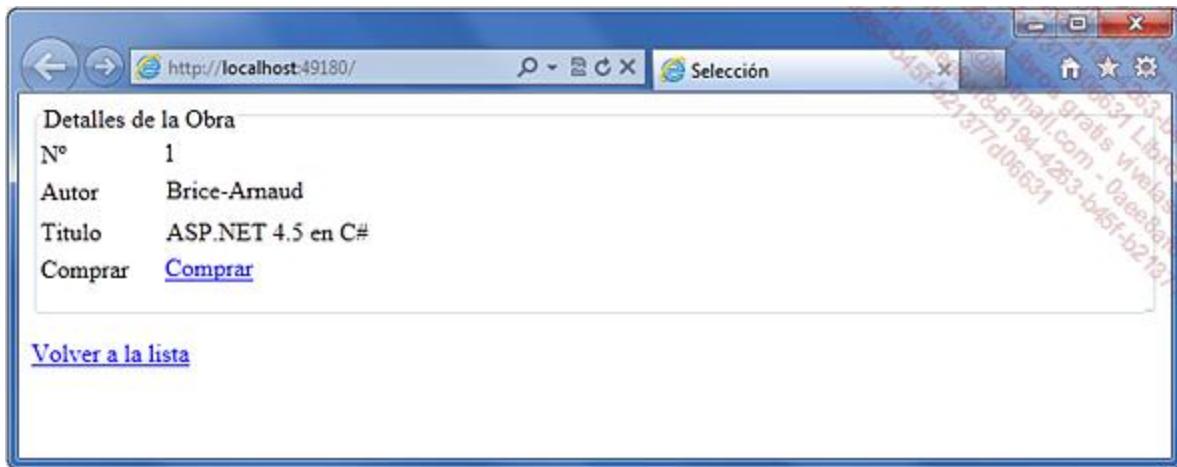
        routes.MapRoute(
            "Default", // Ruta
            "{controller}/{action}/{id}", // URL con parámetros
            new { controller = "Obras", action = "Indice", id =
UrlParameter.Optional } // Parámetros por defecto
        );
    }

    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();

        RegisterRoutes(RouteTable.Routes);
    }
}
```

Cuando el framework busca una ruta, se recorre la lista de arriba a abajo, en orden, y la localización se detiene cuando se encuentra una correspondencia. De este modo, la ruta por defecto debe figurar, siempre, en último lugar.

En nuestro caso, como el parámetro ID es opcional, la acción Seleccion debe recibir directamente un tipo nullable int?. Así, en el siguiente ejemplo, este parámetro se omite y la implementación de la acción asume que ID tiene el valor implícito 1.



## Ir más allá

### 1. De una acción a otra

Vamos a hacer evolucionar nuestro ejemplo de librería agregando un modelo de solicitud:

```
public class Solicitud
{
    public int IDSolicitud { get; set; }
    public Obra obra { get; set; }
    public int Cantidad { get; set; }
    public string Email { get; set; }

    public Solicitud()
    {
        Cantidad = 1;
        IDSolicitud = 0;
        obra = new Obra();
        Email = "";
    }

    public Solicitud(int idSolicitud, Obra obra, int
cantidad, string email)
    {
        this.IDSolicitud = IDSolicitud;
        this.obra = obra;
        this.Cantidad = cantidad;
        this.Email = email;
    }
}
```

```
}
```

Existe una clase de servicio que da soporte a la operación básica y utiliza la sesión HTTP, en lugar de SQL, como medio de persistencia.

```
public class SolicitudServicio
{
    #region Solicitudes
    public List<Solicitud> Solicitudes
    {
        get
        {
            List<Solicitud> l = null;
            l =
                (List<Solicitud>) HttpContext.Current.Session["Solicitudes"];
            if (l == null)
            {
                l = new List<Solicitud>();
                HttpContext.Current.Session["Solicitudes"] = l;
            }

            return l;
        }
    }
    #endregion

    #region IDSolicitud
    public int IDSolicitud
    {
        get
        {
            int? id = null;
            if (HttpContext.Current.Session["IDS"] == null)
            {
                id = 1;
                HttpContext.Current.Session["IDS"] = id;
            }

            return id.Value;
        }
    }
}
```

```

    }

    set
    {
        HttpContext.Current.Session["IDS"] = value;
    }
}

#endregion

#region CrearSolicitud
public void CrearSolicitud(Solicitud solicitud)
{
    solicitud.IDSolicitud = IDSolicitud++;
    Solicitudes.Add(solicitud);
}
#endregion

#region BuscarSolicitud
public Solicitud BuscarSolicitud(int idSolicitud)
{
    var q = from s in Solicitudes where s.IDSolicitud ==
idSolicitud select s;
    return q.ToArray()[0];
}
#endregion

#region ModificarSolicitud
public void ModificarSolicitud(Solicitud solicitud)
{
    var q = from s in Solicitudes where s.IDSolicitud ==
solicitud.IDSolicitud select s;
    var sl = q.ToArray()[0];

```

```

        sl.obra.Autor = solicitud.obra.Autor;
        sl.obra.Titulo = solicitud.obra.Titulo;
        sl.obra.ID = solicitud.obra.ID;

        sl.Email = solicitud.Email;
    
```

```

    sl.Cantidad = solicitud.Cantidad;

}

#endregion
}

```

Se agrega, también una vista Solicitud/Create.aspx que permite al usuario realizar la solicitud de una obra:

```

<%@ Page Language="C#"
Inherits="System.Web.Mvc.ViewPage<capitulo4_mvc.Models.Solicitud>" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Crear</title>
</head>
<body>
    <% using (Html.BeginForm()) {%
        <%: Html.ValidationSummary(true, "Errores") %>

        <fieldset>
            <legend>Detalles de la solicitud</legend>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.IDSolicitud) %>
            </div>
            <div class="editor-field">
                <%: Html.TextBoxFor(model => model.IDSolicitud) %>

```

```

                <%: Html.ValidationMessageFor(model =>
model.IDSolicitud) %>
            </div>

            <div class="editor-label">
                <%: Html.LabelFor(model => model.Cantidad) %>
            </div>
            <div class="editor-field">

```

```

        <%: Html.TextBoxFor(model => model.Cantidad) %>
        <%: Html.ValidationMessageFor(model =>
model.Cantidad) %>
    </div>

    <div class="editor-label">
        <%: Html.LabelFor(model => model.Email) %>
    </div>
    <div class="editor-field">
        <%: Html.TextBoxFor(model => model.Email) %>
        <%: Html.ValidationMessageFor(model =>
model.Email) %>
    </div>

    <div class="editor-label">
        Autor
    </div>
    <div class="editor-field">
        <%: Html.TextBoxFor(model =>
model.obra.Autor)%>
    </div>

    <div class="editor-label">
        Título
    </div>
    <div class="editor-field">
        <%: Html.TextBoxFor(model =>
model.obra.Titulo)%>

```

```

    </div>

    <p>
        <input type="submit" value="Crear" />
    </p>
</fieldset>

<% } %>
```

```

<div>
    <%: ViewData["message"] %>
</div>
<div>
    <%: Html.ActionLink("Lista de solicitudes ", "Indice") %>
</div>

</body>
</html>

```

Para que esta vista sea operacional, hay que invocarla desde la pantalla de detalle de una obra, y pasar a la acción un parámetro identificativo de la obra.

Para ello, necesitamos un controlador:

```

public class SolicitudesController : Controller
{
    //
    // GET: /Solicitudes/Create/5

    public ActionResult Create(int? IdObra)
    {
        var s = new Solicitud();
        var obra = new
        ObraServicio().Select(idObra.HasValue?idObra.Value:1);
        s.obra = obra;

        return View(s);
    }
}

```

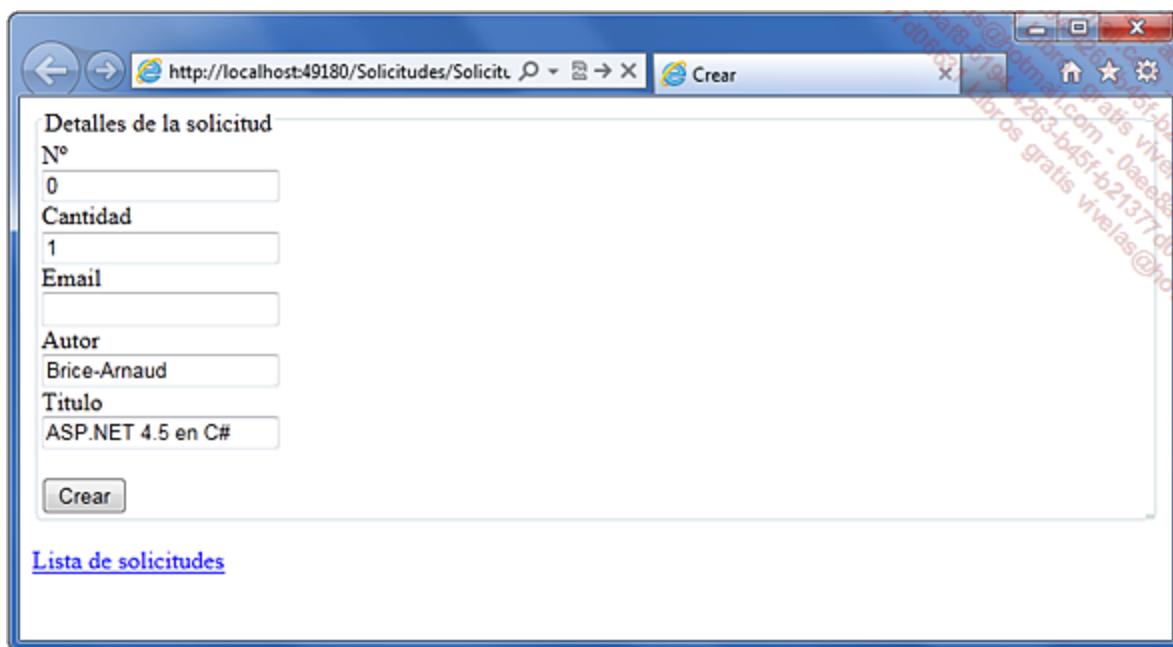
En la vista `Seleccion.aspx`, agregamos un enlace destinado a la acción `Create` del controlador `Solicitudes`. Destacamos el parámetro `idObra`, que viene identificado y con valor:

```

<tr>
    <td>Solicitar</td>
    <td><%: Html.ActionLink("Solicitar", "Create",
    "Solicitudes", new { idObra = Model.ID } , null) %></td>
</tr>

```

En tiempo de ejecución, verificamos que el parámetro se incluye en la URL y que la obra seleccionada está referenciada correctamente en la solicitud:



## 2. Actualización del modelo y redirección

El controlador `SolicitudesController` recibe una segunda definición de la acción `Create`. Ésta se basa en un POST HTTP y tiene como objetivo guardar una nueva solicitud y, a continuación, devolver al navegador la vista `Indice`:

```
//  
// POST: /Solicitudes/Create  
[HttpPost]  
public ActionResult Create(int? idObra, Solicitud post)
```

```
{  
    var p = new SolicitudesServices();  
    var s = new Solicitud();  
  
    // actualiza la solicitud a partir del formulario  
    UpdateModel(c);  
    var obra = new ObraService().Select(idObra.HasValue  
        ? idObra.Value : 1);  
    s.obra = obra;  
  
    p.CrearSolicitud(s);  
    return RedirectToAction("Indice");  
}
```

La acción y la vista Indice se encargan de enumerar las solicitudes y su diseño es, por tanto, idéntico al que hemos mostrado anteriormente para las obras.

### 3. Validación

Como la validación de los datos de negocio no tiene nada de técnico, puede realizarse mediante anotaciones, es decir, atributos que se especifican en las propiedades del modelo.

```
public class Solicitud  
{  
    [DisplayName("Nº")]  
    public int IDSolicitud { get; set; }  
  
    public Obra obra { get; set; }  
  
    [Range(1, 100, ErrorMessage = "La cantidad debe estar comprendida  
entre 1 y 10")]  
    public int Cantidad { get; set; }  
  
    [Required(ErrorMessage = "Introduzca un Email")]  
    [DataType(DataType.EmailAddress, ErrorMessage = "Informe  
una dirección de E-mail")]  
    public string Email { get; set; }  
}
```

La acción Create del controlador ya no debe verificar si el modelo es válido; en caso afirmativo, puede proceder a almacenar la solicitud, en caso contrario, reenviará al usuario la pantalla de crear una solicitud informando los datos que no están conformes a lo esperado:

```
[HttpPost]  
public ActionResult Create(int? idObra, Solicitud post)  
{  
    if (!ModelState.IsValid)  
    {  
        ViewData["message"] = "La solicitud no es válida";  
        return View();  
    }  
  
    var p = new SolicitudServices();  
    var s = new Solicitud();  
  
    // actualiza la solicitud a partir del formulario
```

```
UpdateModel(c);

var obra = newObraService().Select(idObra.HasValue
? idObra.Value : 1);

s.obra = obra;

p.CrearSolicitud(s);

return RedirectToAction("Index");

}
```

El objeto `Html` dispone, por sí mismo, de un método `ValidationSummary` encargado de mostrar el conjunto de errores devueltos por el sistema de validación integrado en el framework:

```
<body>

<% using (Html.BeginForm()) { %>

<%: Html.ValidationSummary(true, "Errores") %>
```

The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost:49180/Solicitudes/Solicitar>. The page title is "Crear". The form has the following fields:

- Nº**: A text input containing "0".
- Cantidad**: A text input containing "0". A validation message next to it says "La cantidad debe estar comprendida entre 1 y 10".
- Email**: A text input with placeholder text "Informe una dirección de Email".
- Autor**: A text input containing "Brice-Arnaud".
- Título**: A text input containing "ASP.NET 4.5 en C#".

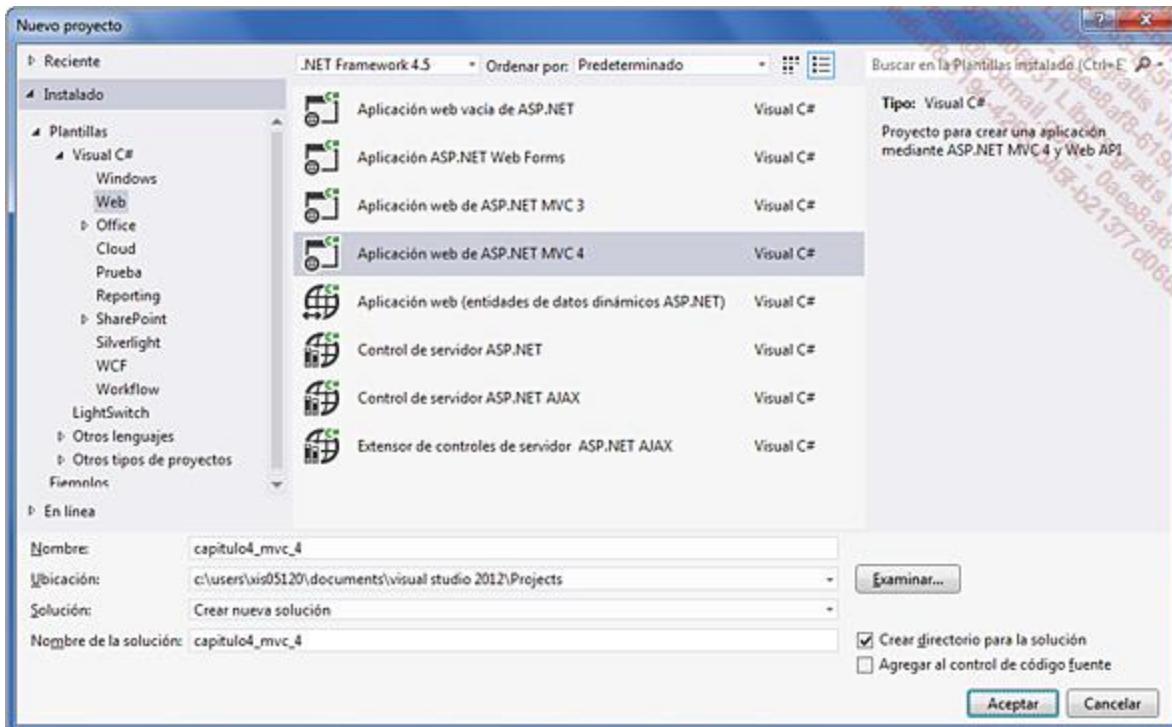
Below the form is a button labeled "Crear". At the bottom of the page, there is a message "La solicitud no es válida" and a link "[Lista de solicitudes](#)".

## Sitios MVC 4 y Razor para dispositivos móviles

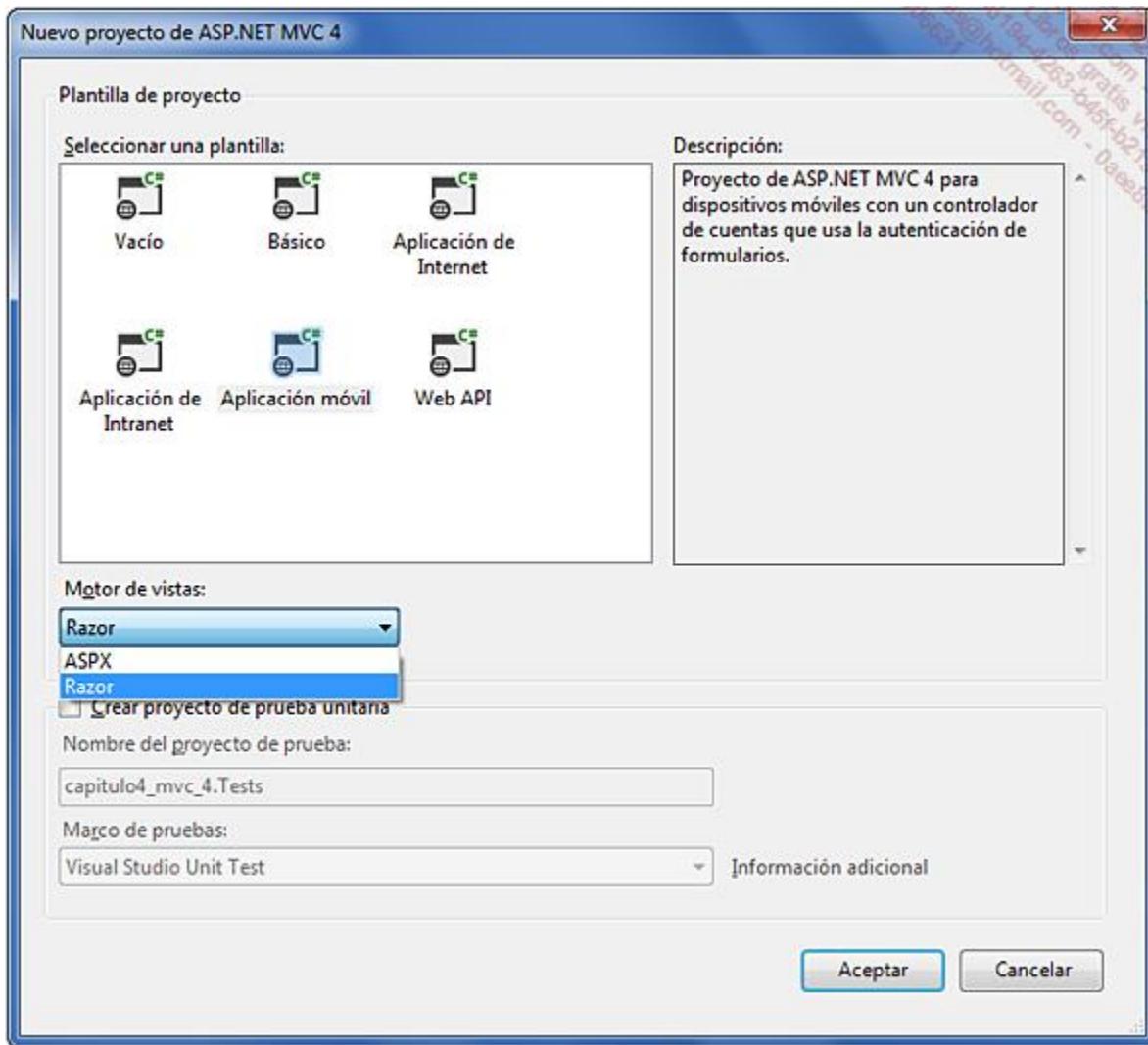
Visual Studio 2012 provee, a su vez, la cuarta generación del framework MVC que viene acompañado de un motor de contenidos Razor. Este último dispositivo tiene como misión simplificar la escritura de vistas, y también la representación de un flujo HTML más ligero y más rápido, mejor adaptado a dispositivos móviles en los que el ancho de banda sigue siendo reducido.

### 1. Crear un proyecto MVC 4

Como con el conjunto de modelos MVC, la creación de un nuevo proyecto MVC 4 se realiza desde el comando **Archivo - Nuevo proyecto**:



En la segunda etapa, debe seleccionar la apariencia de las páginas con ayuda de la sección **Seleccionar una plantilla**. En nuestro ejemplo, escogeremos **Aplicación móvil**, aunque invitamos al lector a que investigue él mismo las demás opciones. En esta misma pantalla es posible escoger el motor de vistas, ASPX o Razor, siendo este último con el que continuaremos.

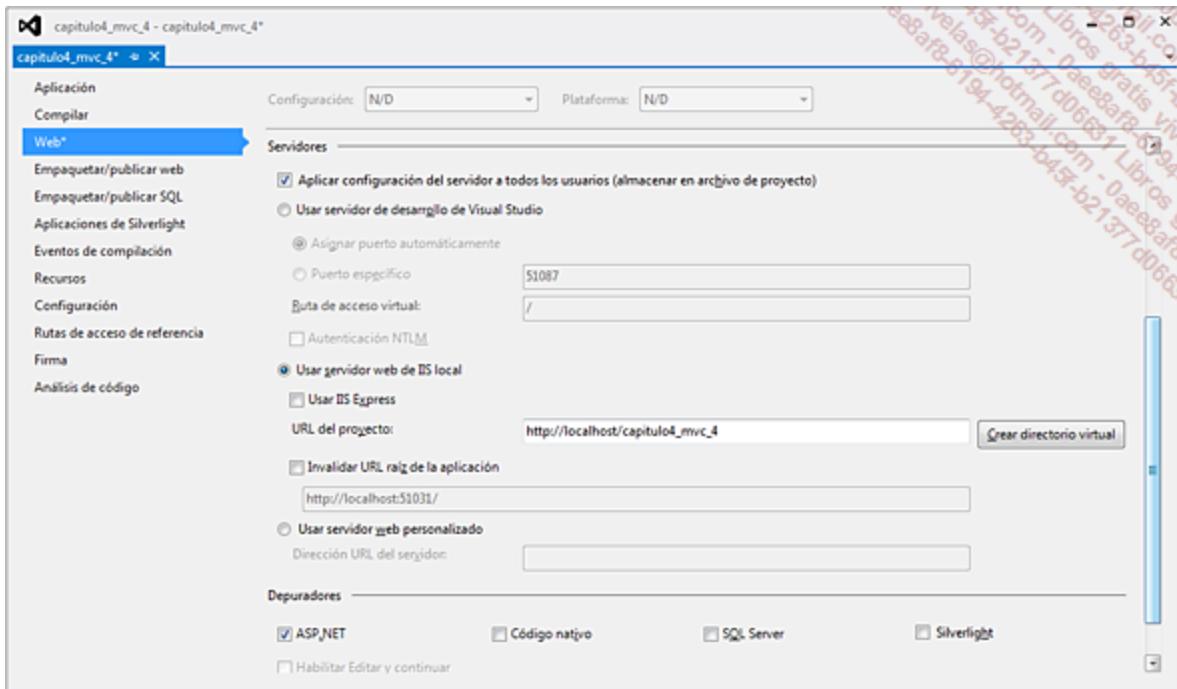


Visual Studio inicializa la solución con carpetas que tienen nombres y roles evidentemente muy similares a MVC 3. La arquitectura se basa, también, en la noción de modelo, vistas y controladores.

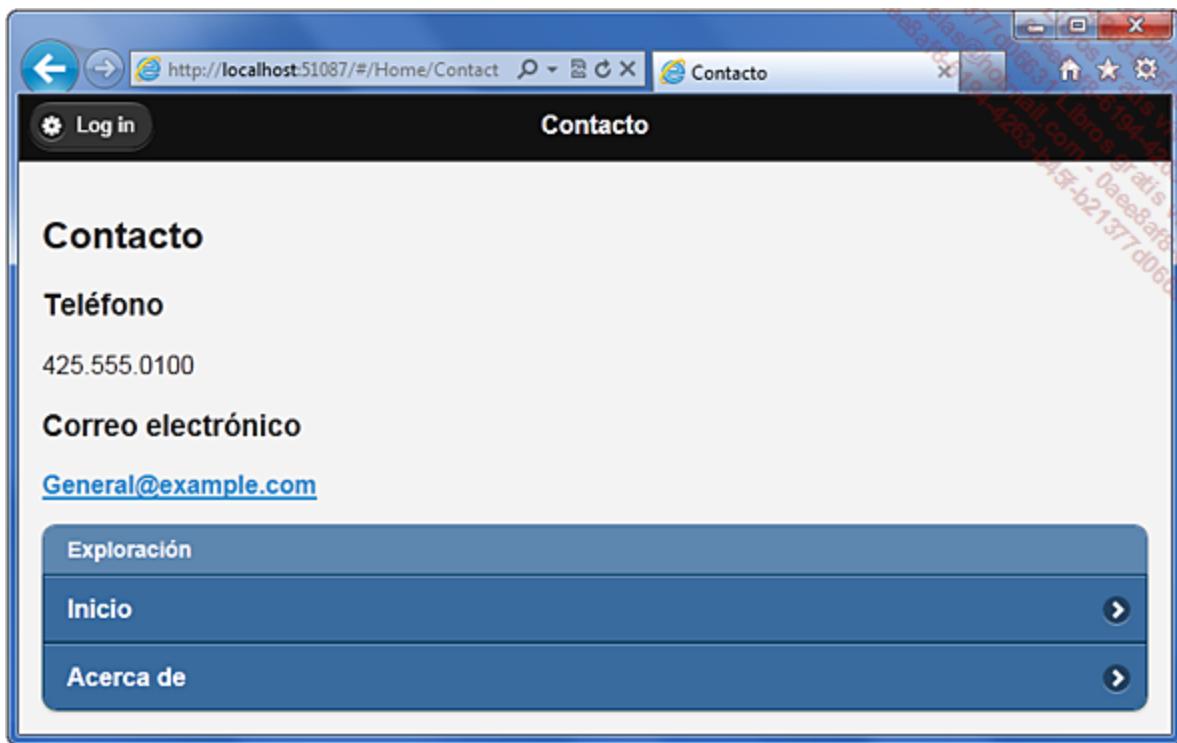
## 2. Pasar a IIS estándar

Como nuestro objetivo es acceder al sitio web desde un emulador, es más sencillo pasar a IIS (generalmente utilizando el puerto 80). Esta operación se realiza desde el menú **Sitio Web - Propiedades del proyecto**.

Conviene, a continuación, activar la opción **Usar servidor web de IIS local** desde la pestaña **Web**, y permitir a Visual Studio crear una carpeta virtual para el proyecto:



La navegación del sitio en un navegador utilizará, en lo sucesivo, el puerto por defecto del servidor IIS (en principio el puerto 80).



### 3. Páginas provistas por Razor

El motor Razor integra los lenguajes C# y VB.NET en sus vistas, teniendo como base de su sintaxis una simplificación de etiquetas scriptlets.

Estudiando el ejemplo de esquema `_Layout.cshtml` ubicado en la carpeta Shared, observaremos que el signo @ sirve de marcador al motor para distinguir las etiquetas HTML de las secuencias C#:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <meta charset="utf-8" />
        <title>@ViewBag.Title</title>
        <meta name="viewport" content="width=device-width" />
        <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        @Styles.Render("~/Content/mobileCss", "~/Content/css")
        @Scripts.Render("~/bundles/modernizr")
    </head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<body>
    <div data-role="page" data-theme="b">
        <div data-role="header">
            @if (IsSectionDefined("Header")) {
                @RenderSection("Header")
            } else {
                <h1>@ViewBag.Title</h1>
                @Html.Partial("_LoginPartial")
            }
        </div>
        <div data-role="content">
            @RenderBody()
        </div>
    </div>

    @Scripts.Render("~/bundles/jquery",
        "~/bundles/jquerymobile")
        @RenderSection("scripts", required: false)
    </body>
</html>
```

Esta novedad vuelve inservibles los marcadores `<% %>` y `<%= %>`, pues el motor reconoce directamente en la página instrucciones C# - como `RenderSection("Header")` - y expresiones como `@ViewBag.Title`.

Existe una forma de escapar el signo @, doblándolo, lo cual resulta útil para direcciones de correo electrónico. El siguiente ejemplo es interesante, puesto que el signo @ no se dobla en el atributo href, sino en el texto de la etiqueta:

```
@{  
    ViewBag.Title = "Contacto";  
}  
  
<h2>Contacto</h2>  
  
<h3>Por email</h3>  
<span><a href="aspnet4.5@ediciones-eni.es">  
aspnet4.5@ediciones-eni.es</a></span>  
  
<ul data-role="listview" data-inset="true">  
    <li data-role="list-divider">Navegación</li>  
    <li>@Html.ActionLink("Inicio", "Index", "Home")</li>  
    <li>@Html.ActionLink("Acerca de", "About", "Home")</li>  
</ul>
```

Si nos damos cuenta, Razor se encarga de representar secuencias HTML en lugar de los scriptlets. Cabe destacar la brevedad del código generado respecto a una página ASPX:

```
<!DOCTYPE html>  
<html lang="es">  
    <head>  
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>  
        <meta charset="utf-8" />  
        <title>Inicio</title>  
        <meta name="viewport" content="width=device-width" />  
        <link href="/capitulo4_mvc_4/favicon.ico" rel="shortcut icon"
```

```
        type="image/x-icon" />  
        <link href="/capitulo4_mvc_4/Content/jquery.mobile-1.1.0.css" rel="stylesheet"/>  
        <link href="/capitulo4_mvc_4/Content/jquery.mobile.structure-1.1.0.css" rel="stylesheet"/>  
        <link href="/capitulo4_mvc_4/Content/jquery.mobile.theme-1.1.0.css" rel="stylesheet"/>
```

```

<link href="/capitulo4_mvc_4/Content/site.css" rel="stylesheet"/>

    <script src="/capitulo4_mvc_4/Scripts/modernizr-2.5.3.js">
</script>

</head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<body>
    <div data-role="page" data-theme="b">
        <div data-role="header">
            <h1>Inicio</h1>
<a data-icon="gear" href="/capitulo4_mvc_4/Account/Login">Conexión</a>
</div>
        <div data-role="content">

            <h2>Razor, ¡muy eficaz para dispositivos Android!</h2>
<p>
    Desarrollar con ASP.NET
</p>

<ul data-role="listview" data-inset="true">
    <li data-role="list-divider">Navegación</li>
    <li><a href="/capitulo4_mvc_4/Home/About">Acerca de</a></li>
    <li><a href="/capitulo4_mvc_4/Home/Contact">Contacto</a></li>
</ul>
        </div>
    </div>

```

```

<script src="/capitulo4_mvc_4/Scripts/jquery-1.7.1.js"></script>
<script src="/capitulo4_mvc_4/Scripts/jquery.mobile-1.1.0.js"></script>

</body>
</html>

```

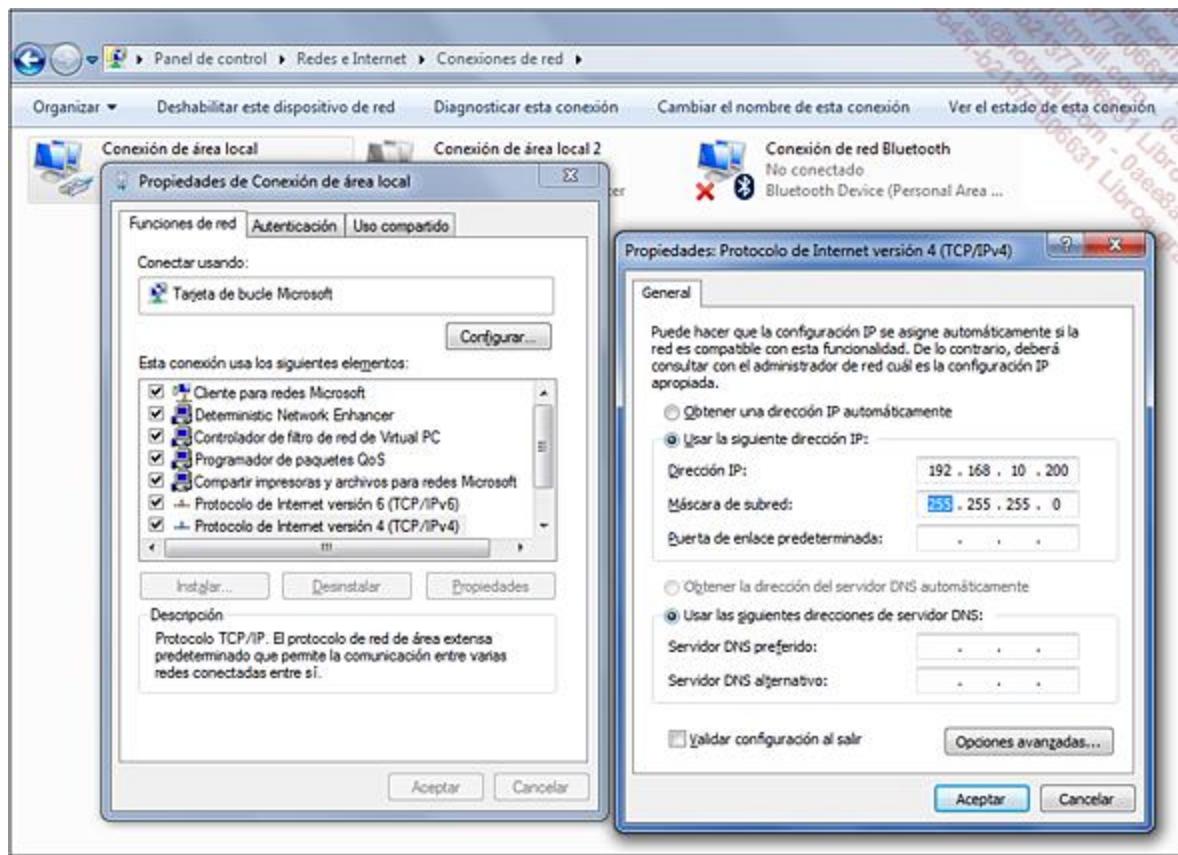
## 4. Probar con un emulador Android

Con el objetivo de probar nuestro sitio "móvil", vamos a utilizar un emulador Android. Lo más sencillo para conseguir uno consiste en instalar el kit de desarrollo de Android para Eclipse

(entorno Java). No importa qué versión de Android escojamos para nuestra prueba, pues las vistas no contienen funciones JavaScript demasiado elaboradas. Por nuestra parte, usaremos una versión 2.2:



Para comunicar el emulador con IIS, puede ser necesario instalar en los puestos equipados con un sistema operativo cliente como Windows 7 una tarjeta de bucle a la que podamos dar una dirección IP estática.



De esta forma podremos informar una dirección IP en la URL de nuestro navegador Android para realizar la solicitud a IIS.



## Bases de ADO.NET

En .NET, el acceso a los datos se realiza mediante un bloque de servicios ADO.NET. Si bien el framework ASP.NET se ha enriquecido con nuevos controles que facilitan la lectura y la presentación de datos SQL, el desarrollo debería considerar el uso del modo conectado para elaborar una aplicación ASP.NET. En efecto, las restricciones de carga, de integración y de ejecución de la web influyen de manera importante en la eficiencia final de una aplicación ASP.NET.

### 1. El modo conectado

En el modo conectado, todos los formatos básicos de datos adoptan el mismo funcionamiento. Lo ilustraremos con SQL Server, y para pasar a otros formatos bastará con cambiar el espacio de nombres y modificar el prefijo de cada clase.

La siguiente tabla muestra los cambios necesarios:

	Espacio de nombres	Conexión	Comando	Las demás clases se
SQL Server	System.Data.SqlClient	SqlConnection	SqlCommand	
Access	System.Data.OleDb	OleDbConnection	OleDbCommand	
Oracle	System.Data.Oracleclient	OracleConnection	OracleCommand	

nombran siguiendo el mismo principio.

#### a. La conexión

La conexión SqlConnection designa el canal por el que se intercambian las órdenes y los registros SQL. Este canal lo programan C# y la base de datos.

El objeto SqlConnection posee varios estados, cabe destacar dos de ellos: abierto y cerrado. Los demás estados están activos de forma transitoria o cuando ocurre algún error.

El programa interactúa con una conexión mediante la propiedad **ConnectionString** y los métodos **Open()**, **Close()**. Lo esencial de las operaciones ligadas a una base de datos sólo puede realizarse a través de una conexión abierta.

```
// inicialización de la conexión  
  
string c_string=@"data source=.\SQL2005; initial catalog=  
agenda; integrated security=true";  
  
SqlConnection cx_agenda;  
  
cx_agenda=new SqlConnection();  
cx_agenda.ConnectionString=c_string;  
  
  
// apertura  
cx_agenda.Open();  
  
  
// operaciones SQL  
  
  
  
// Cierre  
cx_agenda.Close();
```

La cadena de conexión está formada por distintos segmentos que indican el nombre de la máquina que alberga la base de datos, el nombre de la base de datos, las credenciales del usuario... La sintaxis depende de cada formato de base de datos. En SQL Server, la cadena de conexión comprende la siguiente información:

data source	nombre del servidor y de la instancia que ejecutan SQL Server.
initial catalog	nombre de la base de datos.
integrated security	true o sspi: la seguridad integrada está activa. false: la seguridad integrada no está activa.
user id	nombre del usuario que accede a la base de datos.
password	contraseña del usuario que accede a la base de datos.

La

documentación MSDN provee los detalles de los fragmentos que constituyen la cadena de conexión.

El programador debe estar especialmente atento a la manipulación de la conexión. Una vez abierta, consume recursos del sistema y no puede reabrirse antes de cerrarse; una conexión mal cerrada representa, por tanto, un peligro para la integridad y el rendimiento del sistema.

La sintaxis try... catch... finally es, por tanto, la única construcción posible para estar seguros de que se cierra una conexión y se liberan los recursos reservados tras su apertura:

```
try
```

```

{
    // apertura
    cx_agenda.Open();

    // operaciones SQL
    // ...
}

catch (Exception err)
{
    Trace.Write(err.Message);
}

finally
{
    try
    {
        // cierre
        cx_agenda.Close();
    }

    catch (Exception err2)
    {
        Trace.Write(err2.Message);
    }
}
}

```

## Autentificación y cadena de conexión

SQL Server dispone de dos modos de autentificación. El primero consiste en proveer, con cada conexión, un par (usuario, contraseña) que se verifica en una tabla de usuarios. Este enfoque, clásico, puede resultar delicado cuando la información de conexión se almacena en un archivo de configuración de tipo texto o se transmiten con demasiada frecuencia por la red.

El segundo modo de autentificación, llamado seguridad integrada, no necesita que se transmitan un nombre de usuario y una contraseña por la red. El sistema Windows autentifica el programa cliente (ASP.NET, en nuestro caso) y transmite el testigo de autentificación a SQL Server. Este testigo está codificado y tiene un tiempo de vida limitado, lo que aumenta la seguridad del conjunto.

Cuando la cadena de conexión comprende el segmento `integrated security= true` (o `=sspi`), se activa la seguridad integrada. El usuario de ASP.NET debe estar, previamente, autorizado por SQL Server para acceder a la base de datos deseada. En el caso en el que la seguridad integrada no esté activa, debe figurar en la cadena de conexión el segmento `user id=xxx` y `password=xxx`.

```
string c_string = @"data source=.\SQL2012; initial catalog=agenda;
```

```
integrated security=false; user id=sa; password=";
```

## Declarar en SQL Server 2012 el usuario ASP.NET

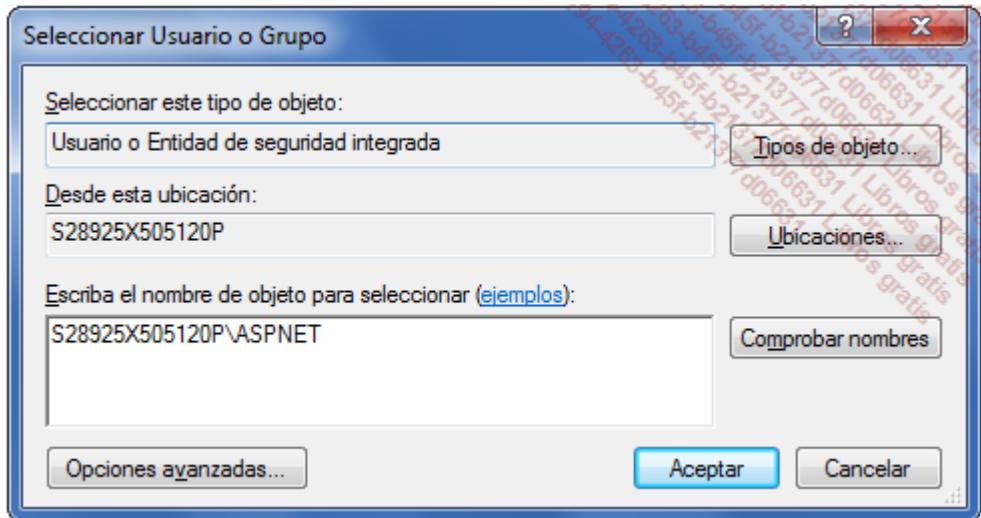
Se muestra, a continuación, la manipulación necesaria para declarar el usuario ASP.NET en SQL Server 2012.

Desde SQL Server Management Studio, conéctese a la instancia en la que se ejecuta la base de datos que quiere consultar desde ASP.NET. El nombre de la instancia depende de la versión del programa que haya instalado (SQL Server 2000, MSDE, 2005 Express...) y de las opciones con que se haya instalado.

El tipo de autenticación no tiene impacto para la aplicación ASP.NET. Se trata, solamente, de la identidad del administrador que exista en el servidor SQL.

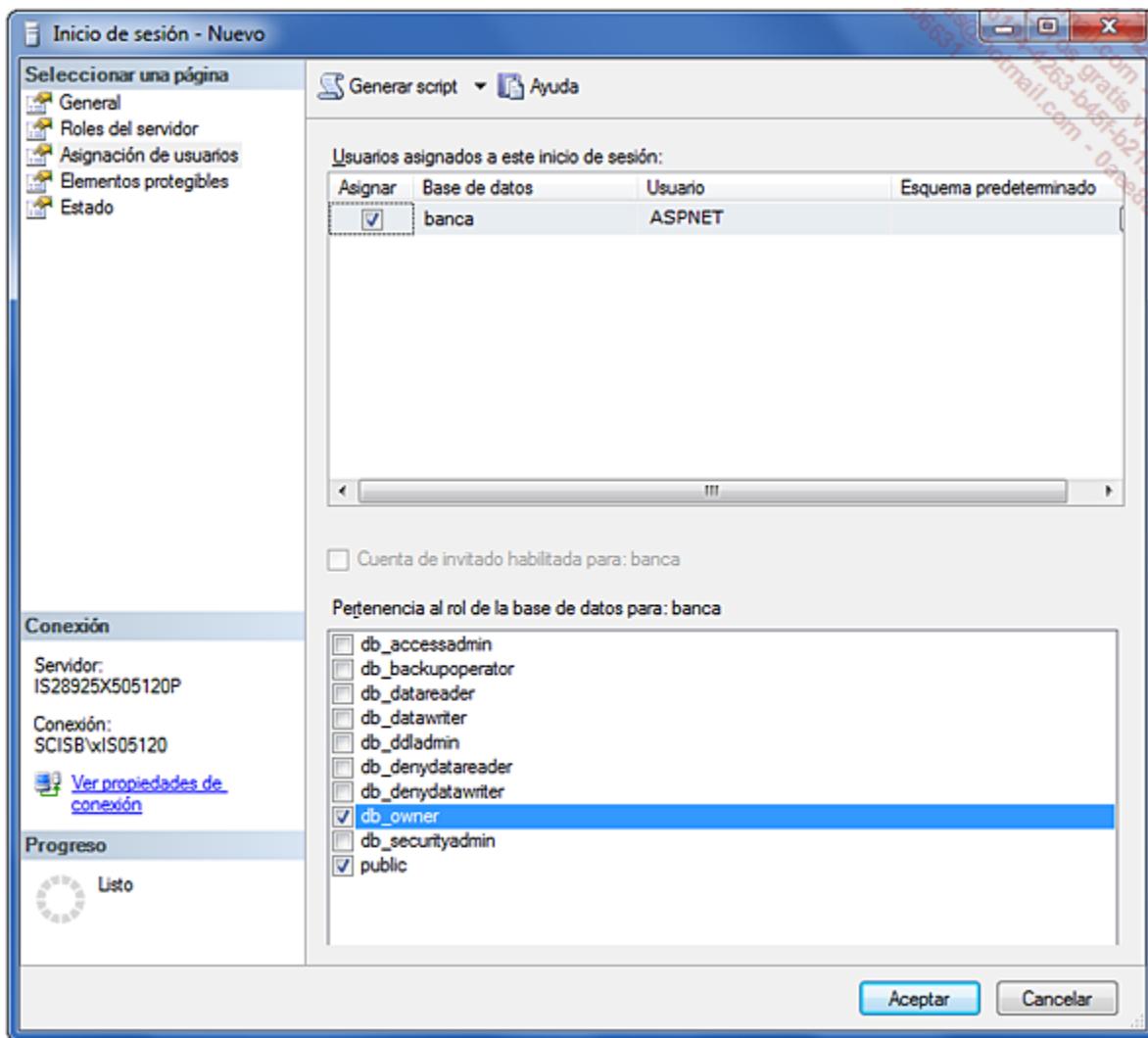


En la sección **Seguridad - Inicio de sesión** del explorador de objetos, utilice el menú contextual **Nuevo inicio de sesión**. Utilice la función de búsqueda para seleccionar el usuario ASP.NET que figura en la máquina que ejecuta el servidor web IIS.



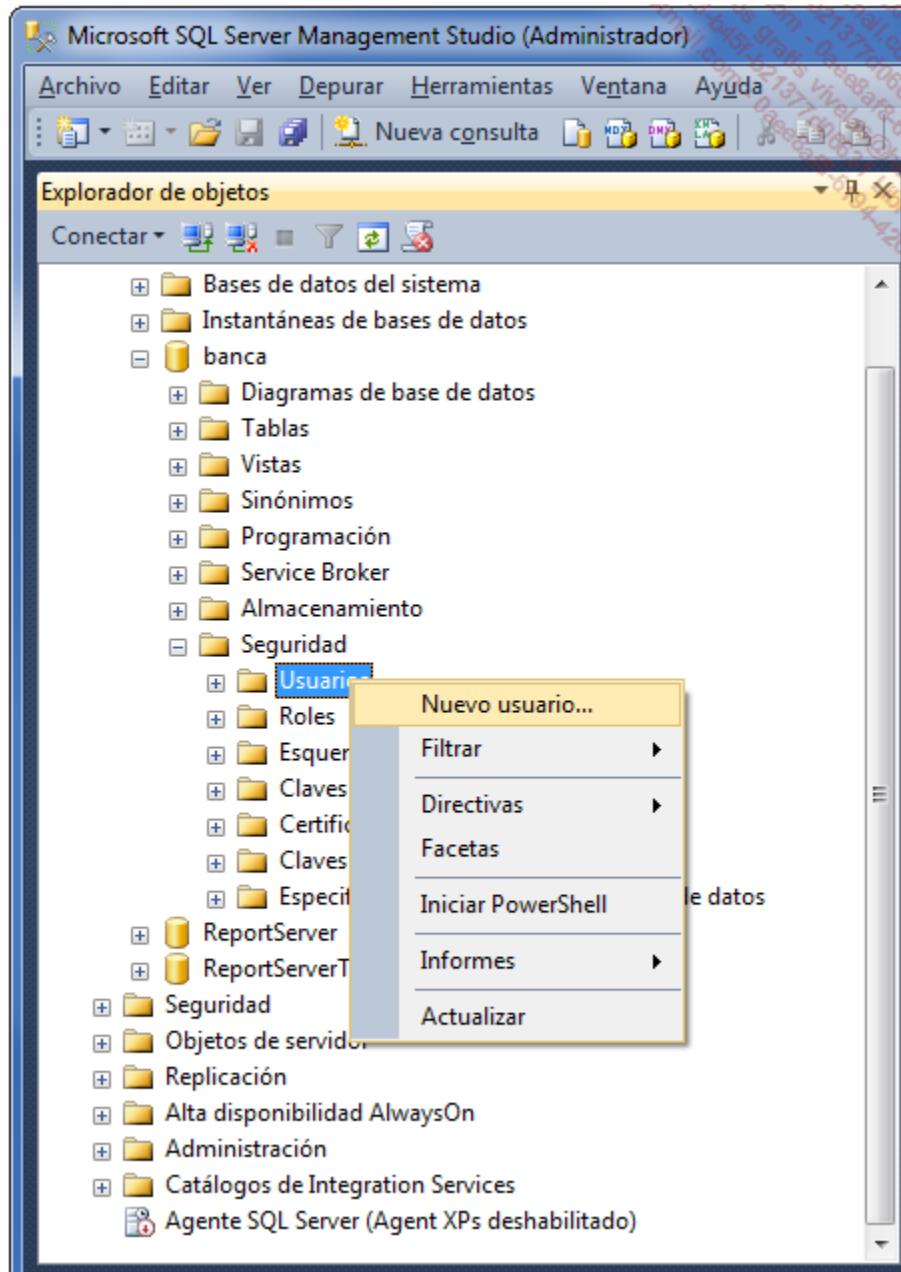
En la pestaña **Asignación de usuario** presente en la ventana de diálogo **Nuevo inicio de sesión**, seleccione la base de datos de trabajo y aplique los roles necesarios para la aplicación ASP.NET. Se activan todos salvo los rechazos (deny), puesto que un rechazo es prioritario a un permiso.

Basta con asignar el rol db\_owner:



Es posible validar la creación del usuario mediante el botón **Aceptar**.

Si el usuario ASP.NET ya existe, debe recibir la autorización necesaria para acceder a la base de datos. Esto se realiza desde el menú contextual **Nuevo usuario** del ítem **Seguridad - Usuarios** de la base de datos de trabajo:



### b. Los comandos

Un comando se corresponde con una instrucción SQL ejecutada desde el programa que se aplica sobre la base de datos designada por la conexión asociada.

```
// apertura
cx_agenda.Open();

// operaciones SQL
string rq = "select count(*) from usuario";
```

```
SqlCommand sql;
sql = new SqlCommand();
```

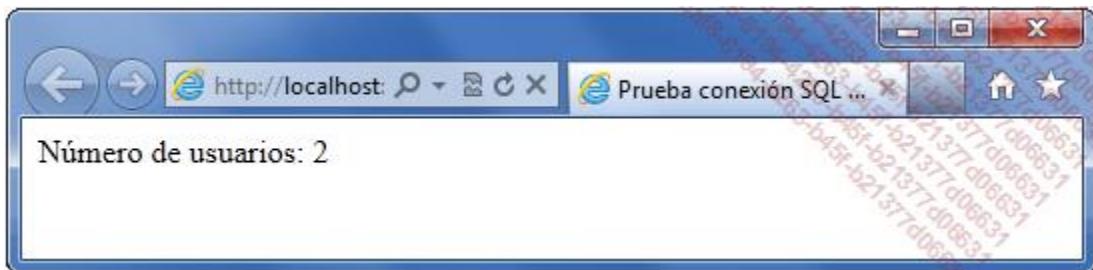
```

sql.CommandText = rq;
sql.CommandType = CommandType.Text; // valor por defecto
sql.Connection = cx_agenda; // asociación de conexión

int cu = (int)sql.ExecuteScalar();
Label1.Text = string.Format("Número de usuarios: {0}", cu);

// cierre
cx_agenda.Close();

```



La propiedad **CommandType** pregunta si **CommandText** contiene una instrucción SQL - como en este ejemplo - o bien se refiere a un procedimiento almacenado. **CommandType.Text**, valor por defecto, hace que no sea preciso informar la línea **CommandType**.

El objeto comando expone cuatro métodos para ejecutar consultas SQL. Cada una de ellas está indicada en una situación concreta:

<b>ExecuteScalar ()</b>	Ejecuta una instrucción SQL que devuelve un valor único (agregado).	c. El
<b>ExecuteReader ()</b>	Ejecuta una instrucción SQL que devuelve, al menos, un valor o un registro.	
<b>ExecuteNonQuery ()</b>	Ejecuta una instrucción SQL que no devuelve ningún valor. Es útil para consultas de creación, actualización, y para invocar a ciertos procedimientos almacenados.	
<b>ExecuteXmlReader ()</b>	Método específico de SQL Server. Ejecuta un instrucción SELECT y devuelve un flujo XML.	

### DataReader

El DataReader - o, mejor dicho, el **SqlDataReader** - es un cursor que se posiciona de registro en registro. Lo instancia el método **ExecuteReader**, y avanza fila a fila gracias al método **Read**.

El programa sólo puede avanzar el cursor, hasta que alcanza la última fila de la selección o hasta que se cierra. Los datos indexados por el cursor no pueden modificarse mediante su intermediario. En contrapartida a estas limitaciones, el DataReader resulta el método más eficaz para leer registros.

### Ejecutar una consulta SELECT que devuelve un DataReader

Las consultas de tipo SELECT afectan a varios registros o varias columnas y devuelven un DataReader. Su aplicación sigue, siempre, la misma lógica:

```
// preparar la conexión
string c_string=@"data source=.\SQL2012; database=agenda;
integrated security=true";
SqlConnection cx_agenda=new SqlConnection(c_string);

// una consulta select devuelve un SqlDataReader
string rq="select idu,nombre,telefono,ids from usuario";
SqlCommand sql=new SqlCommand(rq,cx_agenda);

// abrir la conexión
cx_agenda.Open();

// ejecutar la consulta y recuperar el cursor
SqlDataReader reader=sql.ExecuteReader();

// avanzar fila a fila
while(reader.Read())
{
}

// cerrar, siempre, el reader tras su uso
reader.Close();

// cerrar la conexión
cx_agenda.Close();
```

Como las cláusulas where o having que aparecen en la consulta SELECT pueden omitir todos los registros, el DataReader devuelto se sitúa siempre antes del primer registro, si existe. El bucle que permite recorrerlo debe ser un while (y no un do).

Es obligatorio cerrar el DataReader antes de ejecutar otra consulta sobre la misma conexión. La estructura de control try... catch... finally resulta, por tanto, muy conveniente.

## Leer los valores de las columnas

Una vez posicionado en un registro, el DataReader se comporta como un diccionario cuyas entradas están indexadas mediante números de orden en la consulta (primera columna, segunda columna...) y mediante los nombres de las columnas. La sintaxis correspondiente no es fácilmente intercambiable y el programador debe ser muy riguroso en cuanto a las transformaciones de tipo necesarias.

```

while(reader.Read())
{
    int idu;
    idu = (int)reader[0];           // sintaxis 1
    idu = (int)reader["idu"];       // sintaxis 2
    idu = reader.GetInt32(0);       // sintaxis 3, sin conversión de tipo
    string nombre;
    nombre = (string)reader[1];     // sintaxis 1
    nombre = (string)reader["nombre"]; // sintaxis 2
    nombre = reader.GetString(1);   // sintaxis 3
}

```

Las tres sintaxis presentadas devuelven el mismo dato y, de forma global, con el mismo tiempo de respuesta. La segunda sintaxis, muy directa, requiere en sí misma un **cast** (conversión de tipo) puesto que el indexador de SqlDataReader es de tipo `object`. Es muy importante distinguir la conversión de tipo de una conversión implícita. Una conversión de tipo es un mecanismo que acuerda los tipos de una parte y otra del operador de afectación `=`. El compilador verifica que el tipo del valor a afectar no se promueve sin que el programador tome la responsabilidad. En tiempo de ejecución, el CLR aplica una nueva verificación y produce una excepción si el tipo de la columna consultada no se corresponde con el tipo indicado en el programa.

Por el contrario, la conversión implícita constituye un cambio de tipo. Para los tipos numéricos, es posible aplicar el operador `( )` dando lugar a una doble conversión de tipo. Para los tipos cadena de caracteres que se quiere convertir en números o fechas, los métodos de análisis textual (parsing) permiten realizar la conversión a los tipos correspondientes (`int.Parse` o `DateTime.Parse`).

La tercera sintaxis no da lugar a una conversión de tipo, puesto que el DataReader expone métodos específicos para cada tipo. No obstante, el framework verifica, en tiempo de ejecución, que los métodos se aplican correctamente. NO es posible leer un número doble utilizando una columna de tipo **varchar** sin utilizar un operador de conversión.

Por último, el programador aplica la sintaxis que le parezca más adecuada, sin tener impacto en el resto de código.

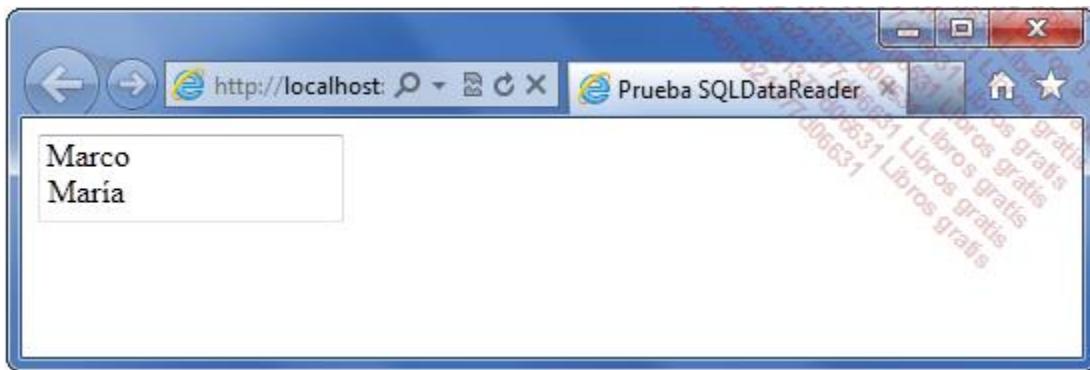
```

while(reader.Read())
{
    int idu;
    idu = (int)reader["idu"];

    string nombre;
    nombre = (string)reader["nombre"];

    ListBox1.Items.Add(new ListItem( nombre, idu.ToString()));
}

```



#### d. Los parámetros

Algunas consultas SQL pueden resultar difíciles de codificar en C# o en VB.NET. Aunque el método `string.Format()` es útil para sustituir variables (llamado interpolar, en PHP), los valores SQL no numéricos requieren procesamientos específicos; los textos están delimitados por apóstrofes, las fechas mediante #...

Es preferible, en este caso, parametrizar estas consultas. El funcionamiento es muy parecido al del método `string.Format`, pero ADO.NET aplicará automáticamente a los valores el formato que mejor le convenga según el tipo del parámetro.

```
// preparar la conexión
string c_string = @"data source=.\SQL2012; database=agenda;
integrated security=true";
SqlConnection cx_agenda = new SqlConnection(c_string);

// una consulta select devuelve un SqlDataReader
string rq = "select count(idu) from usuario where nombre like @nombre+'%'";
SqlCommand sql = new SqlCommand(rq, cx_agenda);

// crear el parámetro @nombre de tipo varchar(100)

SqlParameter nombre = new SqlParameter("@nombre", SqlDbType.VarChar, 100);
sql.Parameters.Add(nombre);

// abrir la conexión
cx_agenda.Open();

// dar un valor al parámetro
sql.Parameters["@nombre"].Value = "a";

// ejecutar la consulta
int c = (int)sql.ExecuteScalar();

// cerrar la conexión
```

```

cx_agenda.Close();

// mostrar el resultado
Label1.Text = string.Format("Cantidad de nombres que comienzan por {0}:
{1}", nombre.Value, c);

```

Es, así, posible asociar varios parámetros a una consulta SQL. Los procedimientos almacenados utilizan parámetros auxiliares para controlar su funcionamiento. Pueden, a su vez, devolver valores mediante parámetros de salida.

### e. Las transacciones

Los sistemas de gestión de bases de datos ofrecen, a menudo, la posibilidad de anular las operaciones de modificación de datos (actualización, inserción, borrado). Las operaciones tienen lugar en un marco temporal llamado transacción. Las transacciones, en primer lugar, se inician (**BEGIN TRANS**), a continuación se resuelven mediante una validación (**COMMIT**) o se anulan (**ROLLBACK**).

Por defecto, SQL Server es auto-validante; el sistema garantiza la integridad de los datos, pero valida lo más rápido posible las operaciones de modificación. Utilizando la instrucción Transact **SQLBEGIN TRANS**, SQL Server pasa al modo transaccional. Todas las operaciones de modificación deberán validarse o anularse.

## Las transacciones ADO.NET 1

El objeto ADO.NET, instancia de **SqlTransaction**, inicia y resuelve una transacción SQL. Está asociado a la vez a una conexión y a los comandos que se ejecutan en dicha conexión.

El siguiente programa utiliza **SqlTransaction** para controlar una inserción en la tabla de usuarios. En caso de error SQL o de duplicidad en la columna nombre, la transacción se anula.

```

// preparar la conexión
string c_string = @"data source=.\SQL2012;database=agenda;
integrated security=true";
SqlConnection cx_agenda = new SqlConnection(c_string);
SqlTransaction trans = null;

try
{
    // abrir la conexión e iniciar la transacción
    cx_agenda.Open();
    trans = cx_agenda.BeginTransaction();

    // prepara la consulta de inserción
    string rq = "insert into usuario(nombre,telefono) values(@nombre,
@telefono)";

```

```

SqlCommand sql = new SqlCommand(rq, cx_agenda);

sql.Parameters.Add("@nombre", SqlDbType.VarChar, 100);
sql.Parameters.Add("@telefono", SqlDbType.VarChar, 10);

// dar valor a los parámetros
sql.Parameters["@nombre"].Value = txt_usuario.Text;
sql.Parameters["@telefono"].Value = txt_telefono.Text;

// ejecutarla en el contexto de la transacción
sql.Transaction = trans; // importante: asociar la
transacción con el comando
sql.ExecuteNonQuery();

// verificar que no exista el usuario
SqlCommand verif = new SqlCommand("select count(idu) from usuario
where nombre=@nombre", cx_agenda);
verif.Parameters.Add("@nombre", SqlDbType.VarChar, 100);
verif.Parameters["@nombre"].Value = txt_usuario.Text;
verif.Transaction = trans;
int n = (int)verif.ExecuteScalar();
if (n == 1)

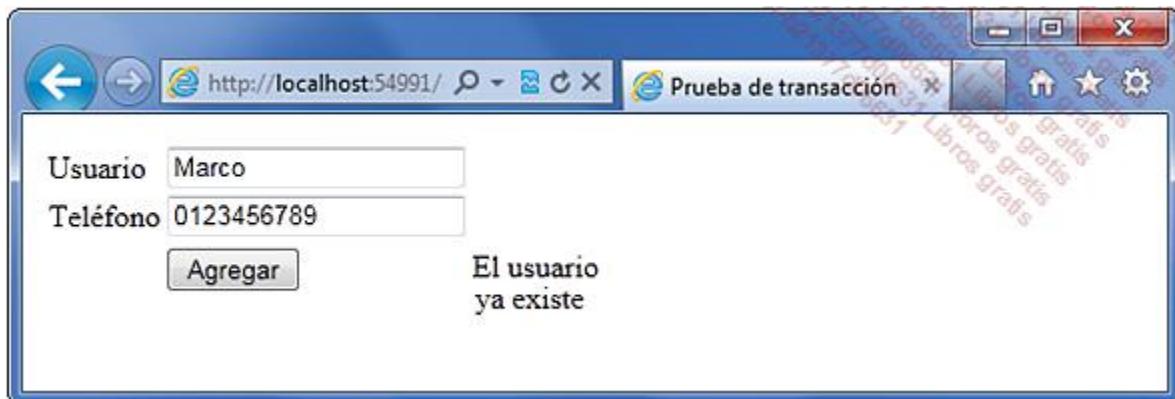
```

```

{
    trans.Commit(); // validar la transacción
    lbl_mensaje.Text = "Usuario agregado";
}
else
    throw new Exception("El usuario ya existe ");
}
catch (Exception err)
{
    if (trans != null)
        trans.Rollback(); // anular la transacción
    lbl_mensaje.Text = err.Message;
}
finally
{
    // cerrar la conexión
    cx_agenda.Close();
}

```

```
}
```



## Las transacciones distribuidas

También llamadas transacciones COM+, tienen en cuenta componentes que pertenecen a dominios de aplicación distintos. El programador define los puntos de la transacción mediante atributos `TransactionOption`, `AutoComplete`, y la clase `ContextUtil`.

El uso de transacciones distribuidas COM+ se describe en Aplicaciones de servidor en .NET.

## Las transacciones ADO.NET 2

Desde la versión 2 de ADO.NET, el bloque de servicios proporciona una sintaxis simplificada para gestionar las transacciones locales con un estilo declarativo que recuerda a las transacciones COM+. Para ilustrar su funcionamiento, vamos a adaptar el ejemplo desarrollado para las transacciones ADO.NET 1.

Para explotar la API de transacciones simplificadas, el proyecto debe hacer referencia al ensamblado `System.Transactions.dll`.

```
using (TransactionScope trans = new TransactionScope  
(TransactionScopeOption.RequiresNew))  
{  
    // preparar la conexión  
    string c_string = @"data source=.\SQL2012; database=agenda;  
    integrated security=true";  
    SqlConnection cx_agenda = new SqlConnection(c_string);  
  
    try  
    {  
        // abrir la conexión en el marco de la transacción en curso  
        cx_agenda.Open();  
    }
```

```

// prepara la consulta de inserción
string rq = "insert into usuario(nombre,telefono) values(@nombre,
@telefono)";

SqlCommand sql = new SqlCommand(rq, cx_agenda);
sql.Parameters.Add("@nombre", SqlDbType.VarChar, 100);
sql.Parameters.Add("@telefono", SqlDbType.VarChar, 10);

// dar valor a los parámetros
sql.Parameters["@nombre"].Value = txt_usuario.Text;
sql.Parameters["@telefono"].Value = txt_telefono.Text;

// ejecutar en el contexto de la transacción
sql.ExecuteNonQuery();

// verificar que no exista el usuario
SqlCommand verif = new SqlCommand("select count(idu) from
usuario where nombre=@nombre", cx_agenda);

```

```

verif.Parameters.Add("@nombre", SqlDbType.VarChar, 100);
verif.Parameters["@nombre"].Value = txt_usuario.Text;
int n = (int)verif.ExecuteScalar();
if (n == 1)
{
    trans.Complete(); // validar la transacción
    lbl_mensaje.Text = "Usuario agregado";
}
else
{
    throw new Exception("El usuario ya existe");
}
catch (Exception err)
{
    // rollback implícito
    lbl_mensaje.Text = err.Message;
}
finally
{
    // cerrar la conexión
    cx_agenda.Close();
}

```

}

La palabra reservada C# using delimita una sección particular del código que se ejecuta en el contexto del objeto transacción trans, el cual inicia una nueva transacción (RequiresNew). El framework ADO.NET supervisa el acceso a los datos que figuran en la sección. Todas las conexiones y comandos SQL se asocian, automáticamente, a la transacción en curso. Ya no es necesario realizar manualmente esta asociación.

Por otro lado, el framework espera la llamada del método Complete() antes de terminar la sección. Si este método, efectivamente, se invoca, la transacción se valida (lo que corresponde a un Commit() en la versión anterior). Si el método Complete() no se invoca, el rollback está implícito.

## 2. Las bases de datos SQL Server

### a. Las versiones de SQL Server

SQL Server es un servidor de bases de datos adaptado por Microsoft a partir de un programa diseñado por Sybase hace ya varios años. Posteriormente, Microsoft ha enriquecido, de manera constante, su capacidad de procesamiento así como las funciones ofrecidas.

La siguiente tabla describe qué versiones del servidor son accesibles al desarrollador:

SQL Server 2000	Versión mayor. Utilizada en la producción y el desarrollo.
MSDE	Versión limitada de SQL Server 2000. Es gratuita, y se utiliza para el desarrollo y para la pequeña producción (contenedores ASP.NET).
SQL Server 2005	Nueva versión mayor. SQL Server Management Studio reemplaza a Enterprise Manager.
SQL Server Express 2005	Como MSDE, adaptación gratuita y limitada de la versión completa de SQL Server 2005.
SQL Server 2008	Nueva versión mayor que soporta, en particular, tipos de datos espaciales destinados a los sistemas de información geográfica.
SQL Server 2008 Express	Versión gratuita.
SQL Server 2008 Express Advanced Series	Versión gratuita que incluye la herramienta de gestión Management Studio.
SQL Server 2012	Nueva versión que mejora el rendimiento y el almacenamiento.

La

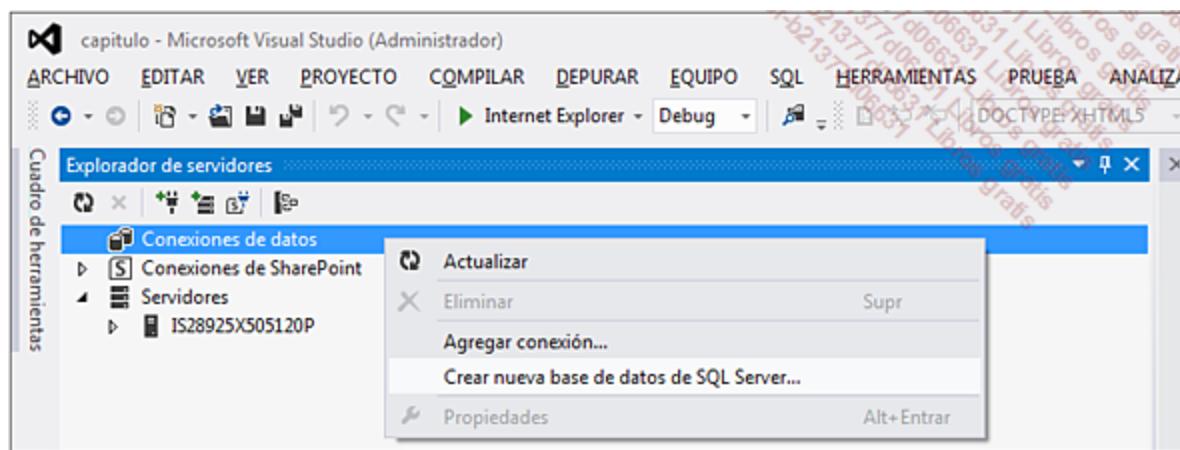
instalación estándar de Visual Studio prevé la instalación de SQL Server Express. Algunos archivos de configuración (machine\_config) se utilizan como valores por defecto. Si la base de datos es un SQL Server (edición clásica), es preferible actualizar el archivo machine\_config. Esta etapa se detalla un poco más adelante.

### b. Creación de bases de datos

La creación de bases de datos SQL Server se realiza con ayuda de un programa específico (Enterprise Manager, SQL Server Management Studio), o bien desde Visual Studio, método que mostramos a continuación.

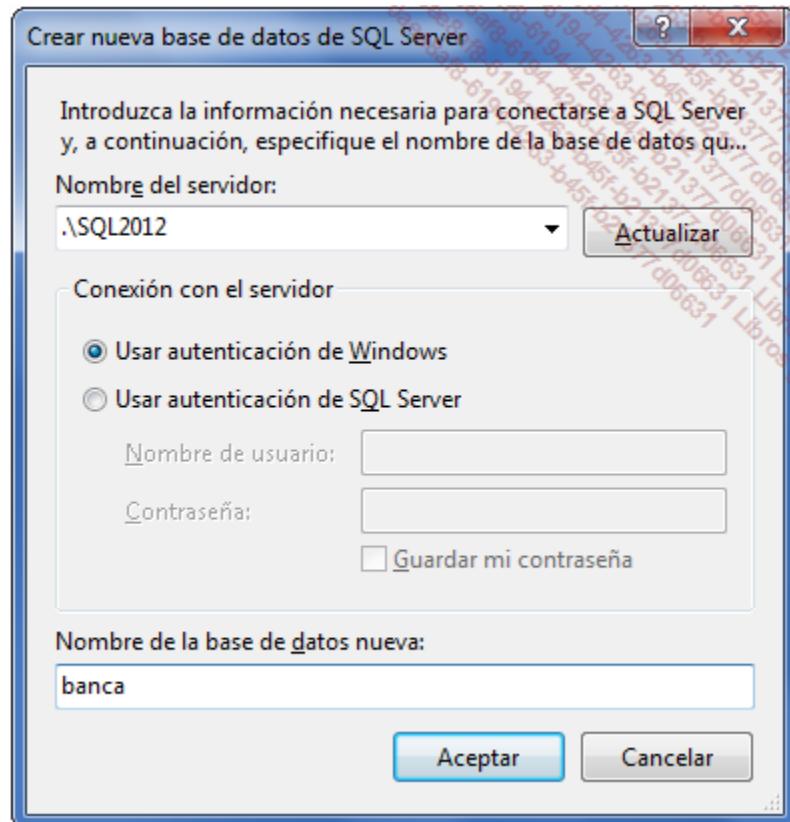
#### Crear una nueva base de datos

Las bases de datos se crean y administran desde el Explorador de servidores (Explorador de datos en Visual Web Development).



Es necesario indicar el nombre del servidor y el nombre de la instancia sobre la que queremos crear la base de datos. Cuando se instalan distintas versiones en el mismo servidor, se distinguen mediante el nombre de la instancia.

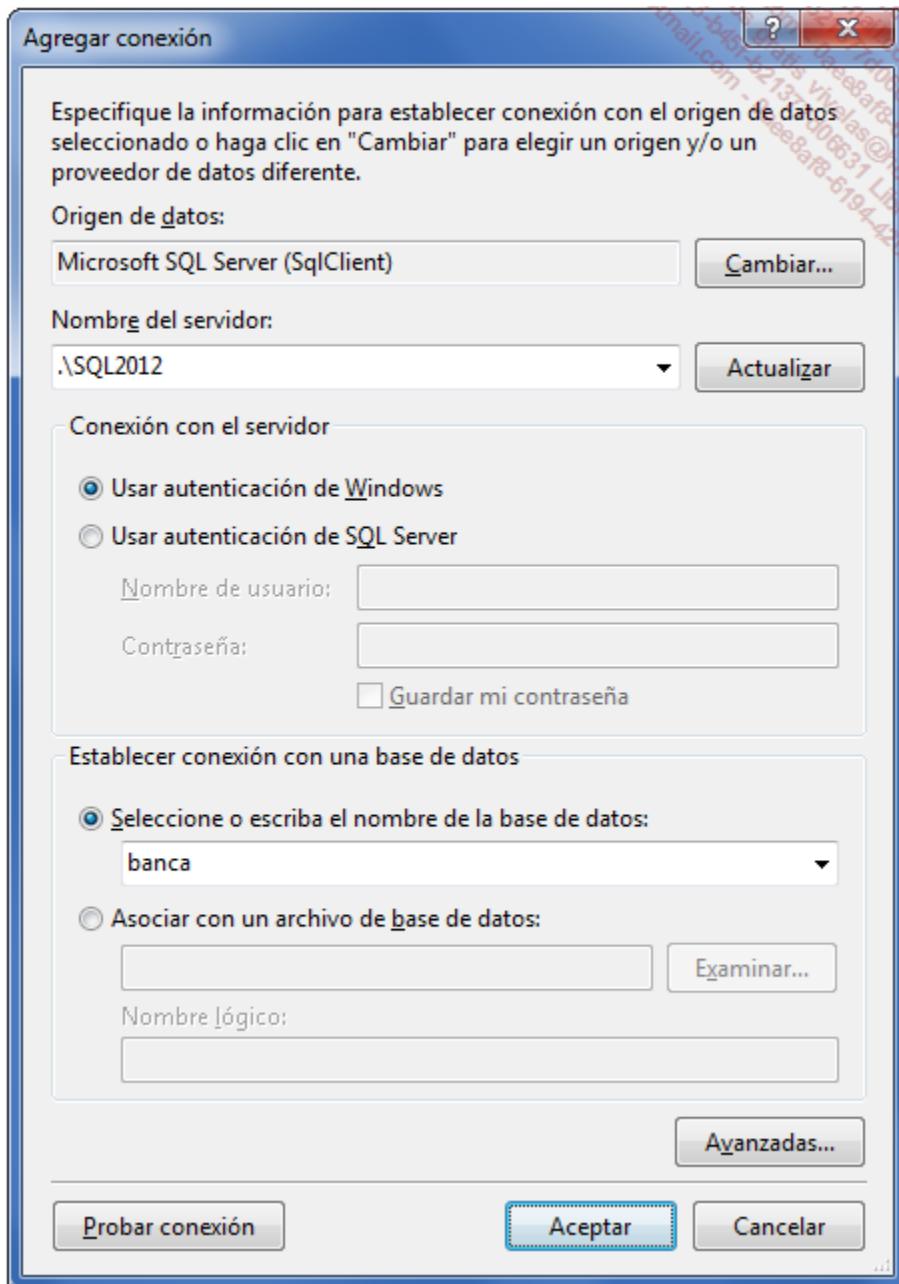
Para el nombre del servidor, podemos utilizar indiferentemente la dirección IP (localhost, por ejemplo) o el nombre Windows. Si nos referimos al equipo de trabajo, las sintaxis **localhost, (local)** o **.** son equivalentes.



## Conectarse a una base de datos existente desde Visual Studio

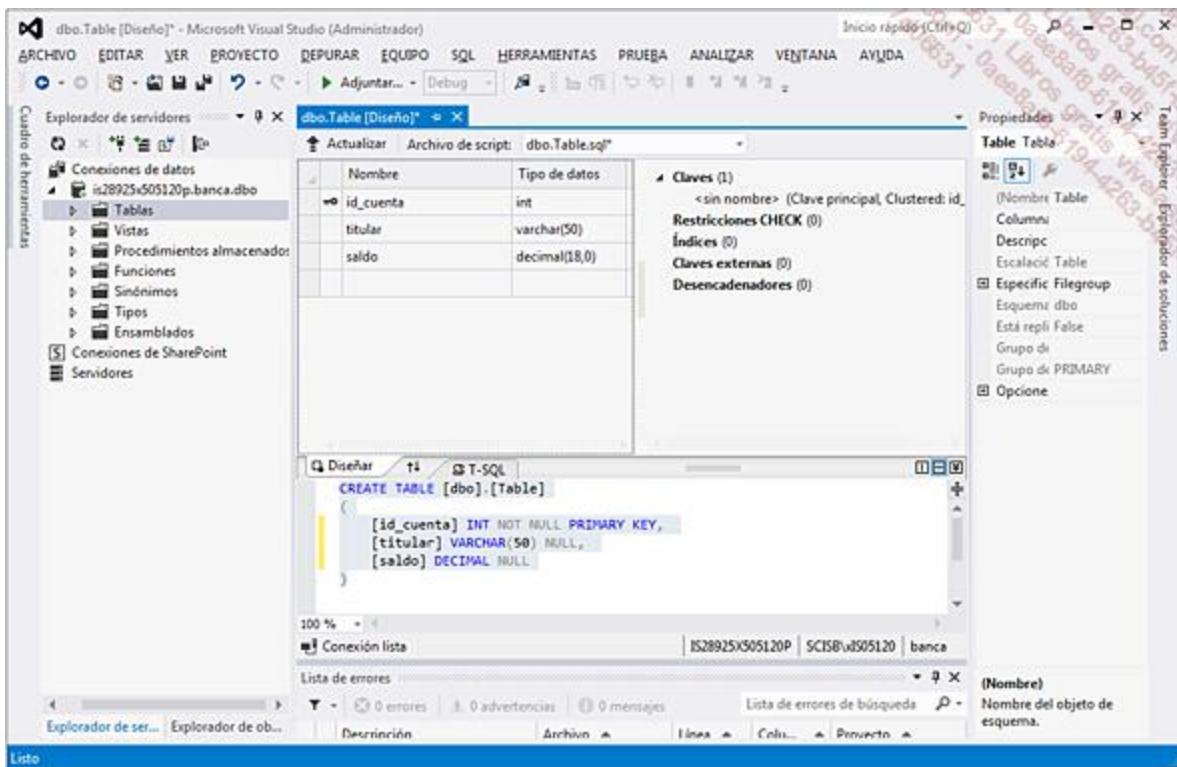
Es útil crear un acceso directo desde Visual Studio hacia la base de datos utilizada en un proyecto ASP.NET. El desarrollador tiene, de este modo, acceso a su estructura (tablas, vistas, procedimientos almacenados) y puede aprovechar esta información para escribir un programa.

La operación de creación de un acceso directo se realiza desde el menú contextual **Agregar conexión** desplegado desde el Explorador de servidores. Respecto a la versión anterior de Visual Studio, la selección de un driver (provider o data source) se realiza mediante el botón **Modificar** y no a través de una pestaña específica.



### c. Creación de tablas

Una vez definida la conexión en Visual Studio, la creación o la modificación de tablas se realiza mediante el menú contextual. El programador dispone, ahora, de todas las facilidades para editar sus tablas y definir sus columnas, claves primarias, relaciones...



Gracias al menú contextual **Mostrar datos de tabla**, es posible insertar y modificar registros en una tabla sin salir de Visual Studio.

dbo.cuenta [Datos]			
	id_cuenta	titular	saldo
	1	Juan Miguel	1000
	2	Fernando	250
►*	NULL	NULL	NULL

#### d. Las vistas

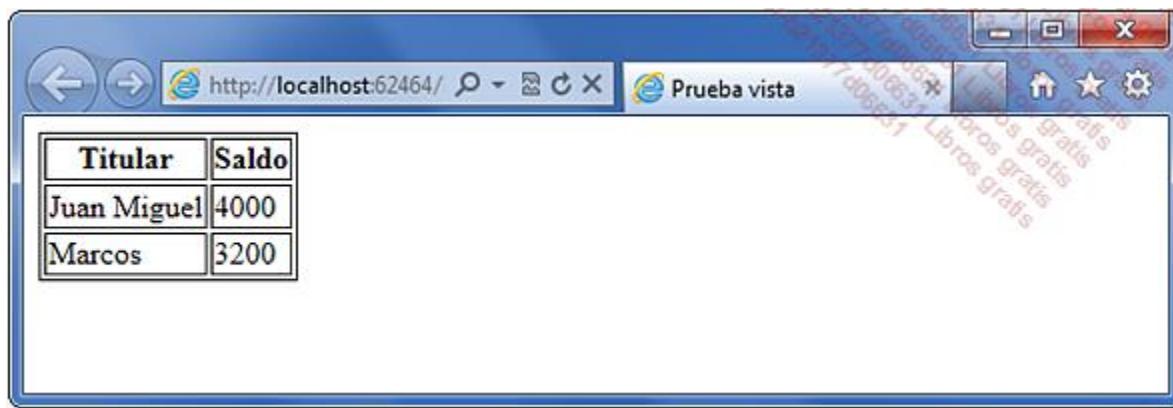
Las vistas son consultas realizadas sobre una o varias tablas y a las que se atribuye un nombre. A continuación, pueden manipularse mediante ADO.NET como si fueran tablas.

La creación de una vista se realiza desde SQL Server Management Studio o desde Visual Studio. En este caso, se realiza desde el menú contextual **Agregar nueva vista** desde el Explorador de servidores.

```
dbo.View.sql * ➔ X
↑ Actualizar
CREATE VIEW [dbo].[VistaGrandesSaldos]
AS SELECT cuenta.titular, cuenta.saldo from cuenta where saldo > 500
```

El siguiente fragmento de código muestra cómo utilizar una vista desde ADO.NET. La técnica es idéntica a la lectura de registros de una tabla. Presenta, no obstante, la particularidad de que se utiliza el data binding para mostrar los resultados dentro de un GridView:

```
string c_string = @"data source=.\SQL2012; database=banca;integrated security=true";  
SqlConnection cx_agenda= new SqlConnection(c_string);  
  
// La vista se considera como una tabla  
SqlCommand sql = new SqlCommand(  
"select * from VistaGrandesCuentas", cx_agenda);  
  
// abrir  
cx_agenda.Open();  
  
// ejecutar la consulta  
SqlDataReader reader = sql.ExecuteReader();  
  
// utilizar el databinding para mostrarla  
GridView1.DataSource = reader;  
GridView1.DataBind();  
  
// cerrar  
reader.Close();  
cx_agenda.Close();
```



#### e. Los procedimientos almacenados

Los procedimientos almacenados son programas escritos en SQL que se ejecutan directamente en el servidor SQL. Un procedimiento tiene un nombre, admite parámetros y posee un conjunto de instrucciones.

Como con las vistas, los procedimientos almacenados pueden crearse desde Visual Studio seleccionando la opción **Agregar nuevo procedimiento almacenado** del menú contextual del Explorador de servidores.

```
ALTER PROCEDURE crear_cuenta
(
    @titular varchar(100),
    @numero int output
)
as
/*
    crear un registro según el valor
    del parámetro @titular
*/
insert into cuenta(titular,saldo) values(@titular,0)

/*
    devuelve el identificador del último registro creado
*/
select @numero=scope_identity()
```

El siguiente programa utiliza un comando SQL parametrizado para invocar al procedimiento almacenado `crear_cuenta`. Cabe destacar la sintaxis, particular, utilizada para declarar el parámetro de salida:

```
// declarar el comando
SqlCommand sql = new SqlCommand("crear_cuenta", cx_agenda);
sql.CommandType = CommandType.StoredProcedure;
sql.Parameters.Add("@titular", SqlDbType.VarChar, 100);

sql.Parameters.Add(new SqlParameter("@numero", SqlDbType.Int, 4,
ParameterDirection.Output , false, 0, 0, "id_cuenta", DataRowVersion.
Default, 0));

// abrir
cx_agenda.Open();

// ejecutar y leer el parámetro de salida
sql.Parameters["@titular"].Value = txt_titular.Text;
sql.ExecuteNonQuery();
int numero = (int)sql.Parameters["@numero"].Value;

// cerrar
cx_agenda.Close();

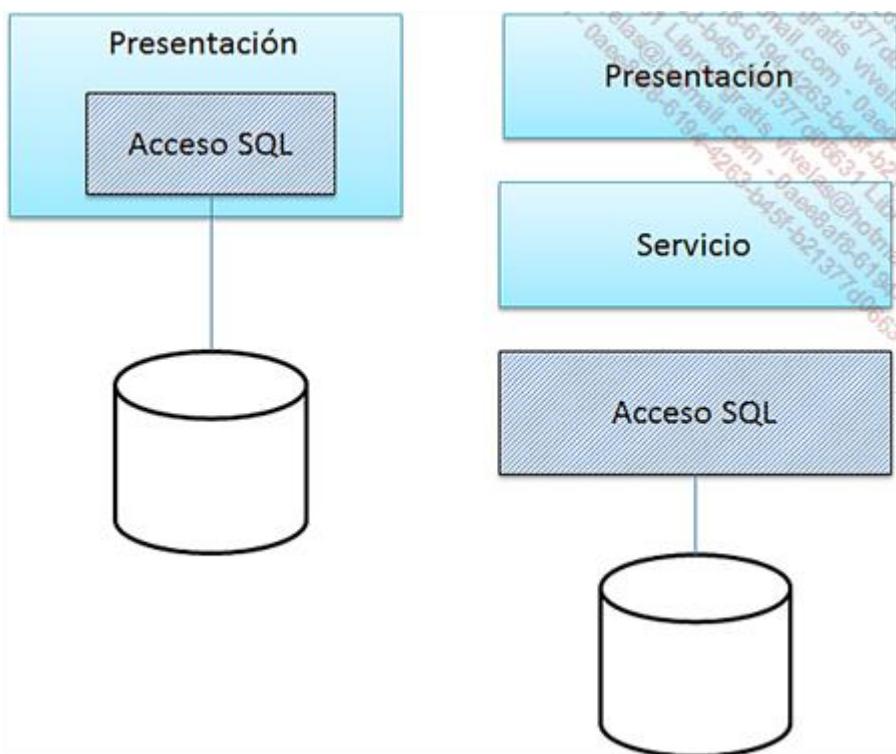
// mostrar
```

```
lbl_info.Text = string.Format("Su número de cuenta es: {0}", numero);
```



### 3. Hacer transparente el acceso a las bases de datos

Las aplicaciones ASP.NET requieren una arquitectura adaptada al acceso a los datos. Los ejemplos de código relativos al modo desconectado no siempre son aplicables tal cual. En este sentido, estudiaremos el acceso transparente a las bases de datos. De esta transparencia se desprende una separación de la lógica de presentación y la lógica de almacenamiento, donde la capa de servicio se intercala entre ambas.



a. El modo desconectado

Bien conocido por los desarrolladores de ASP.NET 1.X, el modo desconectado y su DataSet han cambiado radicalmente la construcción de una aplicación web. El DataSet es una caché de datos (Microsoft lo llama, también, grupo) estructurados bajo la forma de DataTable y de DataRelation. Una tabla DataTable está, en sí misma, organizada en filas (DataRow) y en columnas ( DataColumn).

Cada DataTable está asociado con un flujo XML (ReadXml, WriteXml) o bien con un dispositivo de transmisión SQL, el DataAdapter. El adaptador de datos DataAdapter soporta cuatro comandos - SelectCommand, InsertCommand, UpdateCommand, DeleteCommand - y expone, en particular, los métodos Fill y Update. El método Fill alimenta el DataSet a partir de datos SQL, mientras que el método Update sincroniza la base de datos a partir de intercambios de información mantenidos en cada línea del DataSet.

Dado que los DataAdapter trabajan con distintos formatos de bases de datos (SQL Server, Oracle...) y pueden alimentar las tablas de un mismo DataSet, la arquitectura autoriza el cambio de formato de base de datos a lo largo del desarrollo.

Además, los DataSet han sido diseñados para facilitar las operaciones que enlazan datos (Data Binding) con controles web ASP.NET. Esto explica el número de ejemplos que los hacen intervenir. Instancias de una forma variante, los DataSet tipados son clases que incluyen un esquema XSD y derivan de DataSet. Permiten realizar maquetas rápidamente y crear estados de Crystal Report.

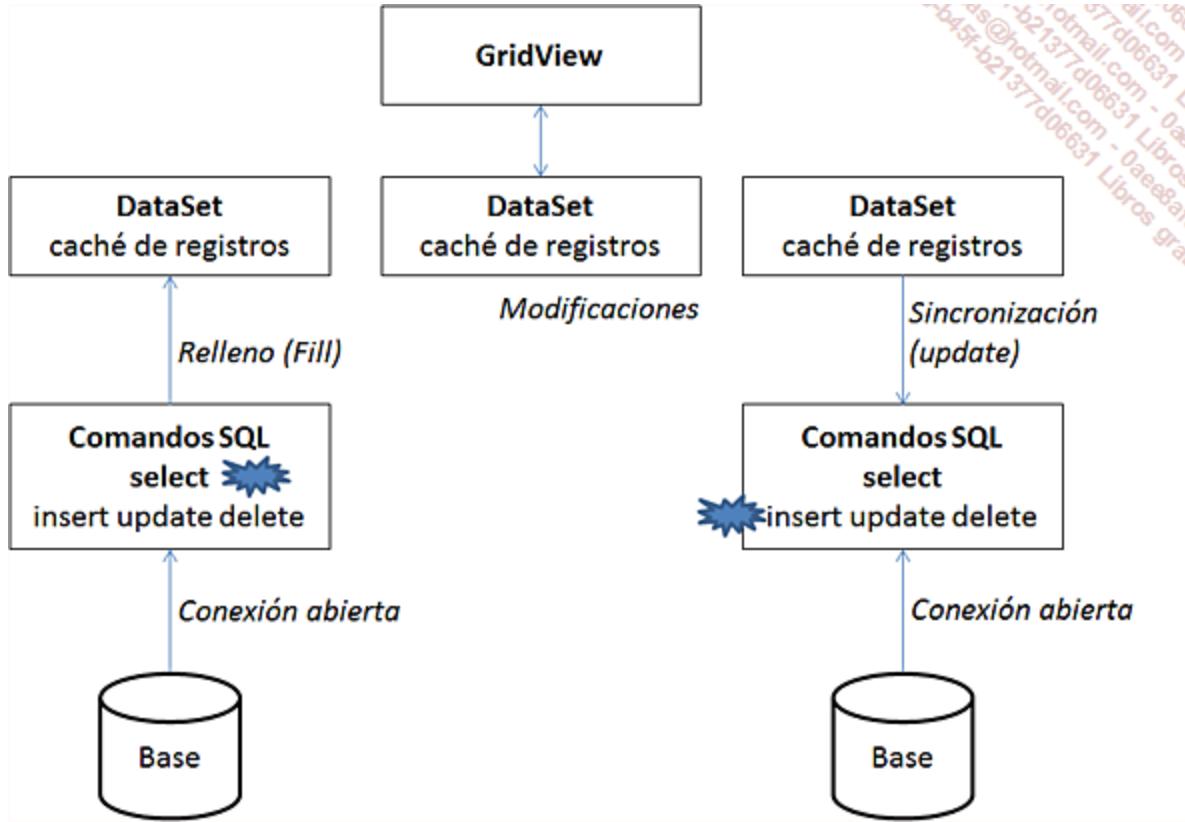
Entonces, ¿todo son ventajas con los DataSet? La realidad es mucho más compleja. Si bien ofrecen métodos de consulta relacionales muy interesantes, el DataSet 1.X está limitado en el número de registros que puede manipular en condiciones razonables de rendimiento (en torno a 400 registros). Si bien la versión 2.0 de este componente se ha reescrito por completo de cara a manejar una volumetría mayor, las aplicaciones ASP.NET encuentran siempre las mismas dificultades.

Por un lado, la volumetría debe contabilizar todas las tablas, incluidas las tablas necesarias para las relaciones. Ninguna base de datos es capaz de paginar los resultados de una consulta para sus tablas satélite. Por otro lado, el servidor de aplicaciones ASP.NET debe evitar duplicar la base de datos en el objeto Application. El rendimiento no se verá mejorado (puesto que el objeto Application puede ser persistente de cara a una base de datos SQL), y este modo de trabajo supone un error en la arquitectura. Sólo un servidor SQL garantiza la integridad de los datos, concurrencia en su acceso y transaccionalidad.

Como consecuencia, podemos considerar que el DataSet es un resultado necesario pero no suficiente. Sería preciso agregar dispositivos de persistencia en caché, mejorando el rendimiento y asegurando el funcionamiento. Para obtener este resultado, el DataAdapter, disponible desde la primera versión del framework ADO.NET, se ha mejorado con otros componentes.

## b. DataAdapter y TableAdapter

Los DataAdapter son capaces de alimentar y de sincronizar un DataTable a partir de una tabla SQL. La conexión entre el DataSet y la base de datos SQL no se abre más que en dos ocasiones, el resto del tiempo los datos se manipulan desde el programa y no necesitan ningún acceso al servidor de base de datos. Este funcionamiento explica el sentido del término **modo desconectado**.



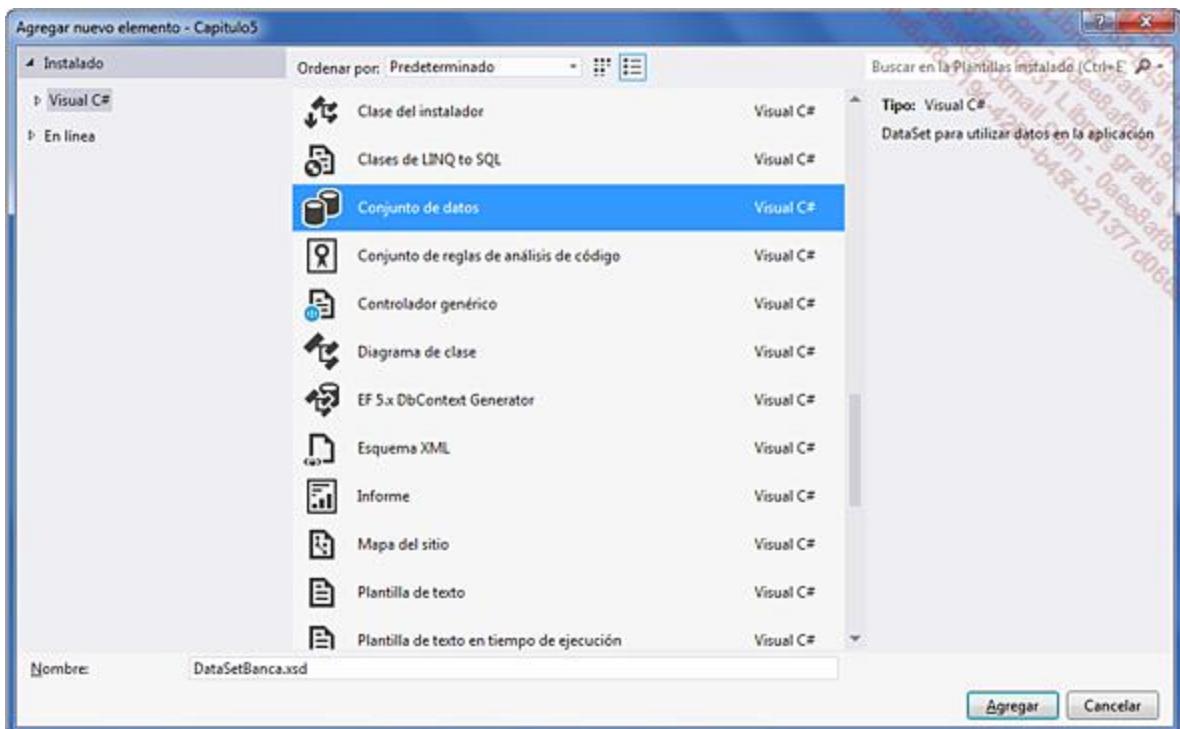
## El TableAdapter

El TableAdapter transforma el DataAdapter facilitando su creación en el seno de un DataSet tipado. La apariencia Page/Table, inconcebible, obligaba a externalizar la definición de las tablas en forma de esquemas XSD.

El enfoque TableAdapter supone una novedad. Consiste en ubicar archivos XSD en la carpetaApp\_Code. El lector adivinará que el servidor de aplicaciones ASP.NET 4.5 genera, dinámicamente, el código de un DataSet tipado en lugar de Visual Studio.

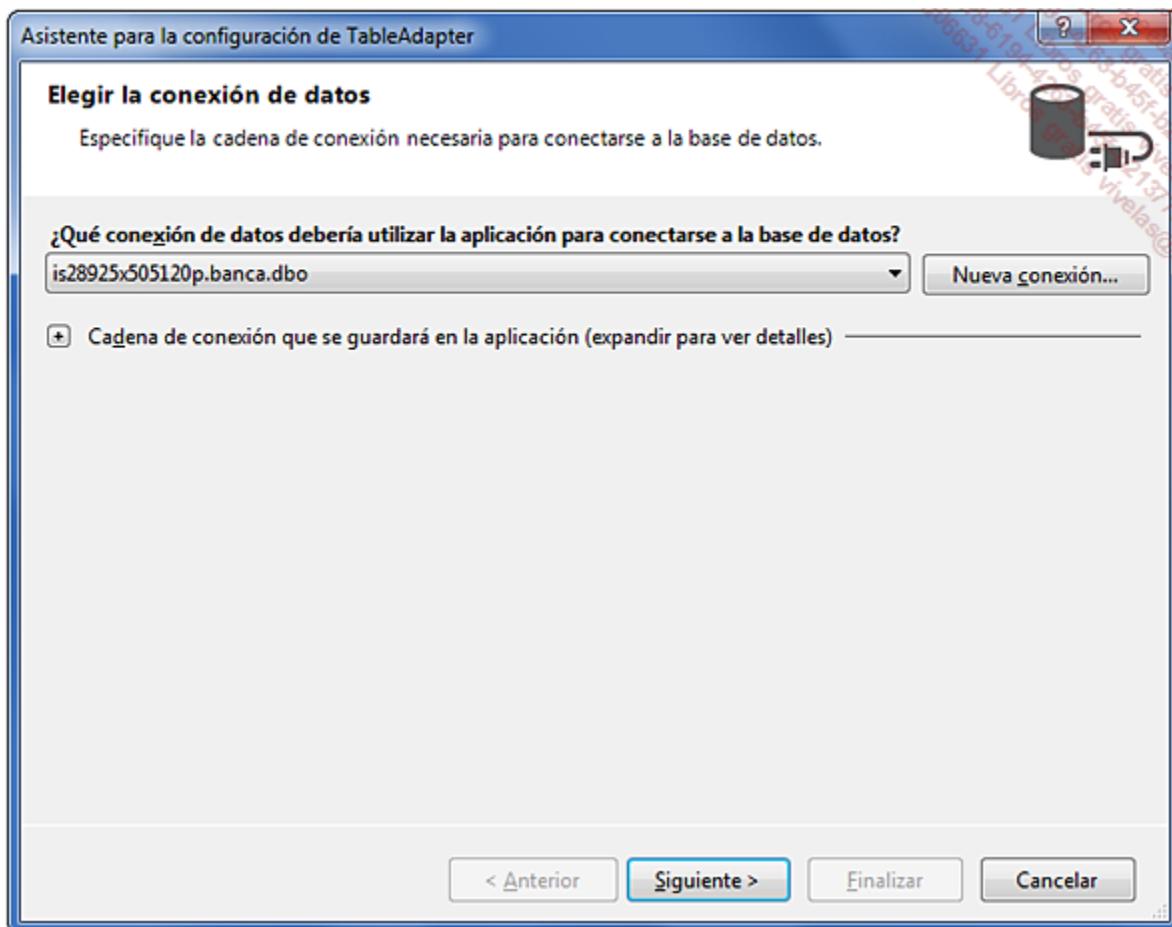
El TableAdapter supone, también, una novedad para el modelo de programación, puesto que el adaptador soporta, además de los cuatro comandos de un DataAdapter, tantos como consultas requiera.

Es posible incluir un TableAdapter al mismo tiempo que la definición de un DataSet tipado.

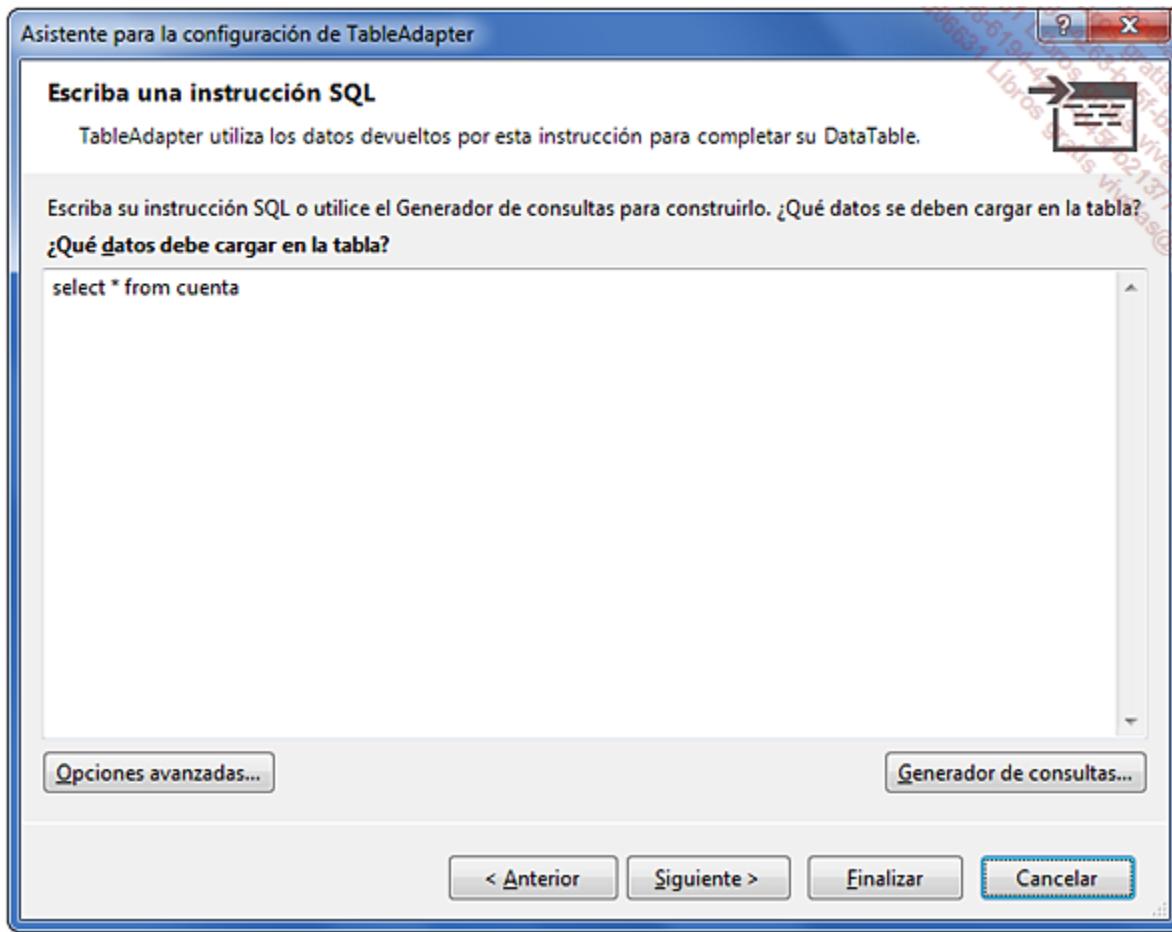


Si la carpeta App\_Code todavía no existe, Visual Studio le propone crearla.

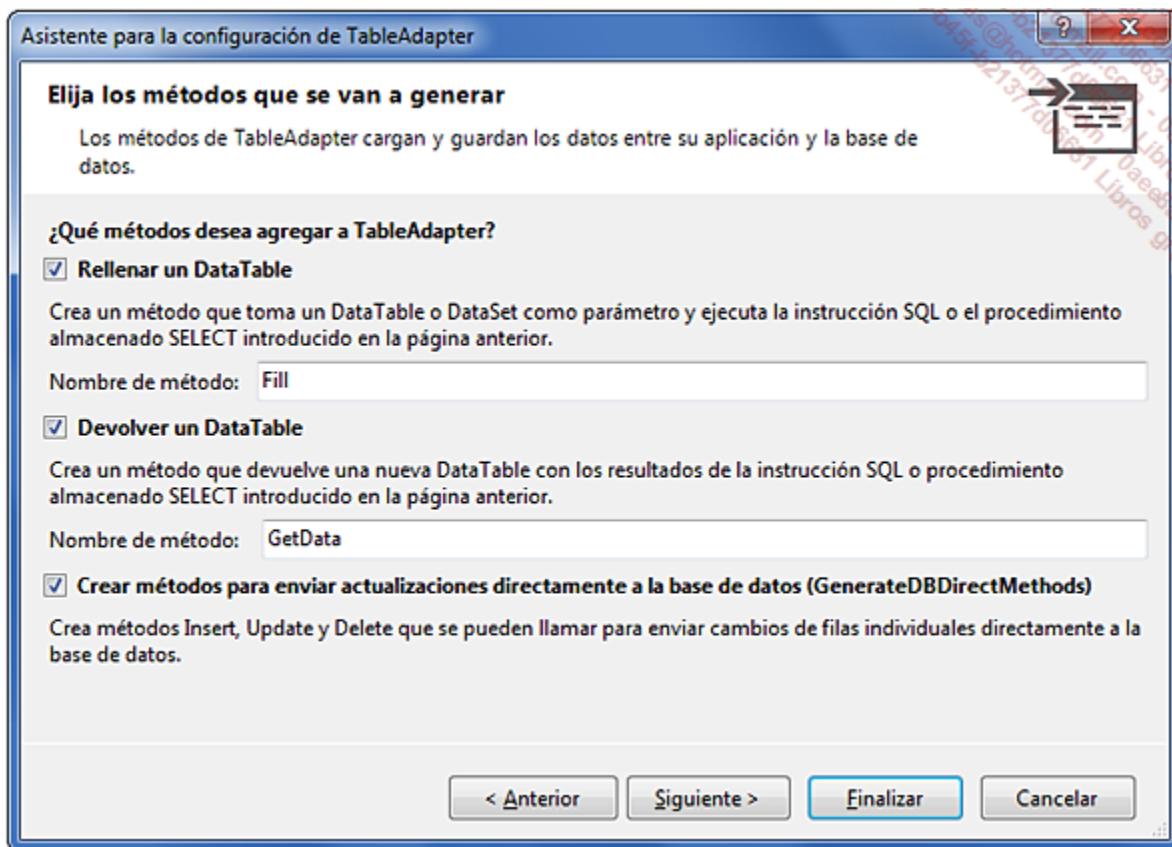
La segunda etapa de la secuencia consiste en la definición de una conexión utilizada por los comandos del TableAdapter. La cadena de conexión correspondiente se declara en el archivo Web.config.



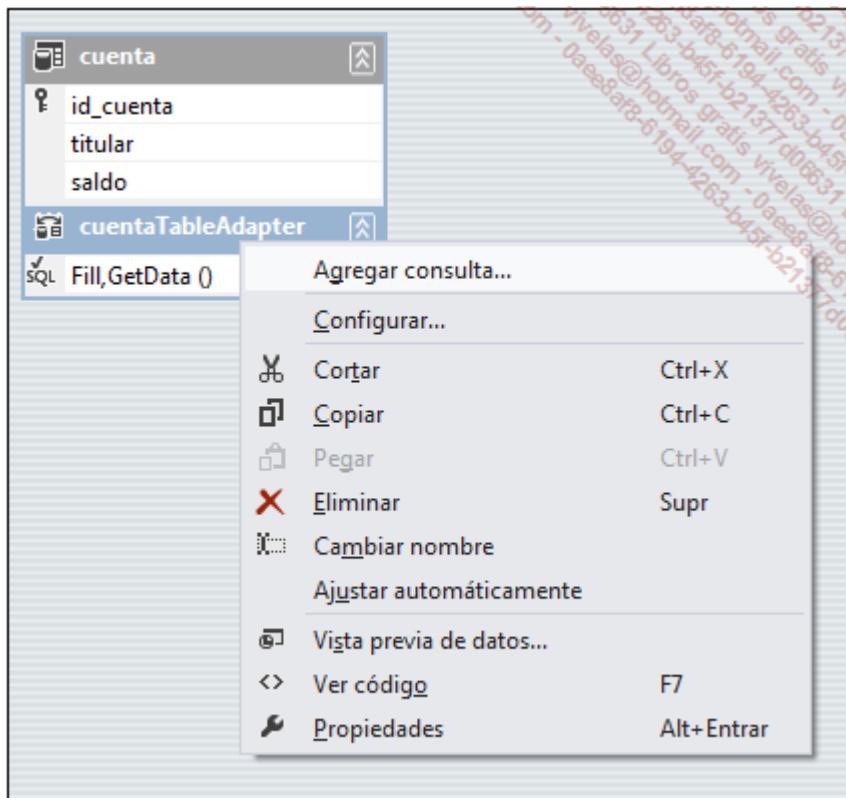
El asistente le pregunta si los comandos están soportados por instrucciones SQL o por procedimientos almacenados. Del mismo modo, la operación de formación de la consulta principal Select debería repetirse para cada tabla del DataSet tipado. Es, por tanto, posible hacer uniones de tipo join, pero las operaciones de actualización pueden verse comprometidas.



Respecto al DataAdapter, el asistente de TableAdapter le proporciona una etapa adicional para definir el nombre del método Fill (para llenar el DataSet), el nombre de un método que devuelve el DataTable correspondiente al comando Select, e indicar si el programador desea métodos de acceso directo a la base de datos.



Una vez finalizado el asistente, el programador tiene la posibilidad de crear nuevos métodos-consulta para cada TableAdapter:



## Acceso a los datos mediante proveedores

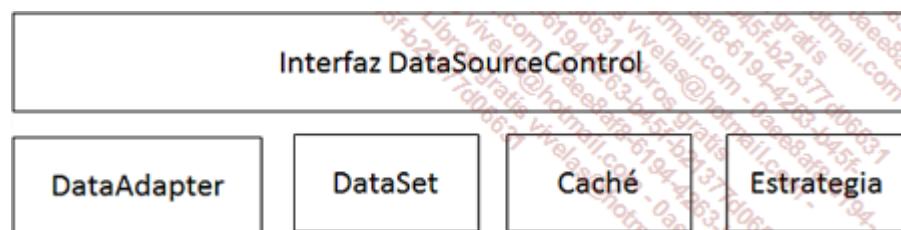
### 1. Introducción al desarrollo por proveedores

El acceso a los datos desde un sitio ASP.NET es, sin duda, uno de los aspectos más delicados de afrontar; para ayudar en la arquitectura de esta tarea, el framework .NET proporciona varios niveles de desarrollo:

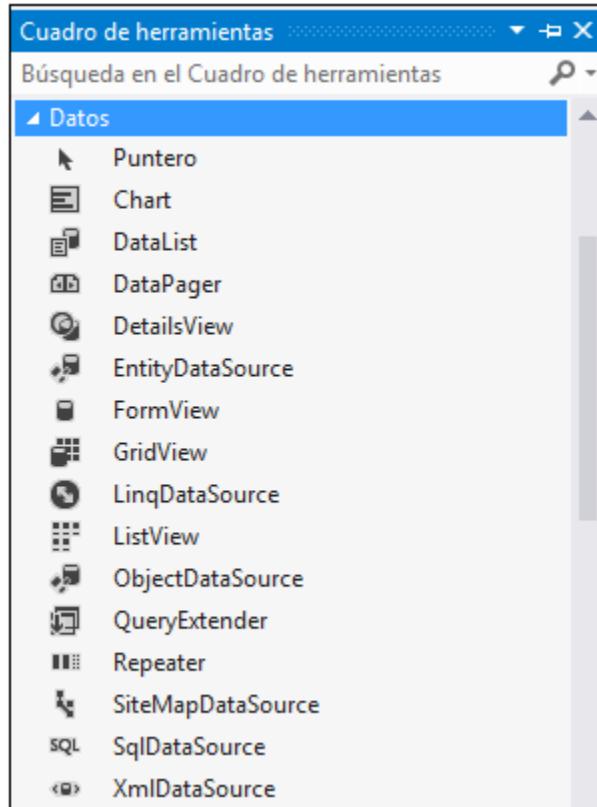
Modo conectado	Clases y componentes en contacto directo con la base de datos.
Modo desconectado	Componentes y herramientas que desacoplan la interfaz gráfica de la base de datos.
Modo proveedor de datos	Sistematización de una organización de acceso a datos.

El

desarrollo basado en un proveedor no consiste en enriquecer la API con nuevas clases. Se trata, más bien, de una estrategia, de una organización para obtener un buen resultado. En lo relativo al acceso a los datos, Microsoft define los proveedores como controles que agrupan una interfaz para el data binding, un almacén de datos (digamos un DataSet), un mecanismo de adaptación, una caché de datos así como una estrategia.



El framework ASP.NET define controles que implementan el acceso a datos según el **modo proveedor**. La sección Data (datos) del cuadro de herramientas de Visual Studio proporciona, precisamente, este tipo de control. De forma complementaria, existen otros controles útiles para la presentación y la edición de datos.



#### a. Controles origen de datos en modo proveedor

Estos controles tienen en común el hecho de adoptar un estilo declarativo: se trata de controles Web forms, sus propiedades vienen determinadas por el código XHTML. En la mayoría de ocasiones con muy poco código de usuario, a excepción de ObjectDataSource que es, precisamente, un control previsto para tener en cuenta las especificidades del programador.

SqlDataSource	Control origen de datos especializado en el acceso a SQL Server.
AccessDataSource	Control origen de datos especializado en el acceso a MS-Access.
ObjectDataSource	El programador define su propia fábrica de objetos que se asocia, a continuación, a una instancia de Object-DataSource para las operaciones de data binding.
XmlDataSource	Control origen de datos especializado en la lectura filtrada de archivos XML.
SiteMapDataSource	Control origen de datos asociado al archivo de definición del mapa del sitio.
LinqDataSource	Control origen de datos asociado a un contexto de objetos LINQ.
EntityDataSource	Control origen de datos que se conecta a un contexto de objetos LINQ to Entity (para SQL Server).

b.

#### Controles de presentación de datos

Estos controles se distinguen por su organización gráfica en línea, en columna y por la naturaleza de las operaciones que soportan.

GridView	Sucesor de DataGrid, presenta los datos en forma de tabla: un registro en cada fila.
DataList	Menos integrado que el GridView, presenta cada registro en una celda de una tabla.
DetailsView	Presenta los datos registro a registro. Los campos están alineados en una columna.
FormView	Presenta los datos registro a registro. El modo (Template) es libre.
ListView	Funciona de forma similar a ListView Windows proporciona varios modos de visualización, bajo la forma de modelos (templates).
Repeater	Componente de bajo nivel destinado a crear presentaciones completamente personalizadas.

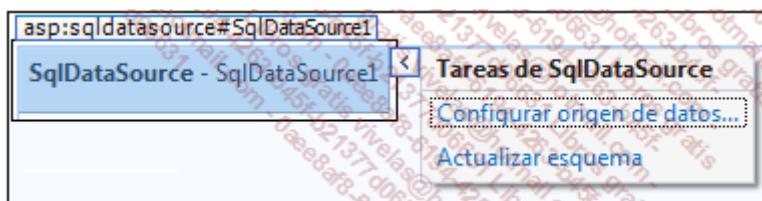
programador decide la disposición general de los controles.

## 2. Los orígenes de datos SqlDataSource y AccessDataSource

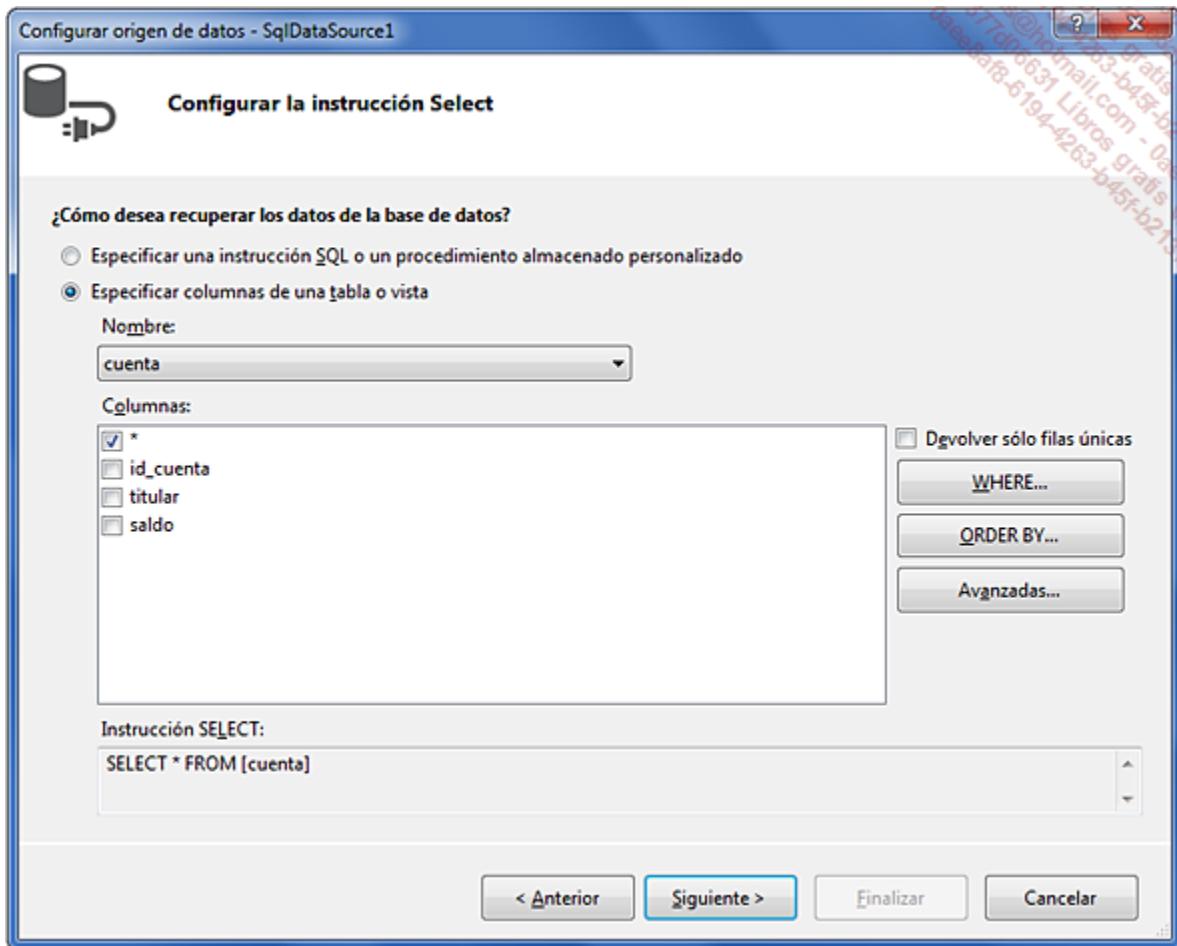
Muchas páginas web ASP.NET son formularios que presentan datos SQL. Los controles de origen de datos SqlDataSource y AccessDataSource están especializados en el acceso bidireccional a datos SQL.

### a. Consultas de selección

Para configurar un control origen de datos, se utiliza, a menudo, la **smart tag** que se sitúa en la parte superior derecha de su representación gráfica:



En el caso de los controles SqlDataSource y AccessDataSource, la opción **Configurar origen de datos** inicia el asistente. El programador puede, de este modo, escoger la conexión SQL y construir su consulta de selección.



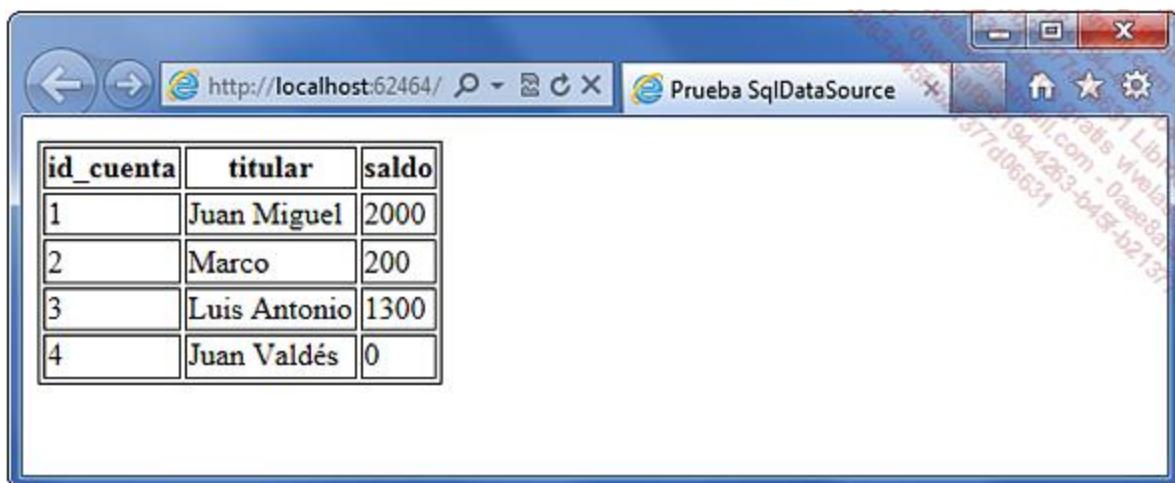
El programador habría podido, también, editar el código correspondiente en la sección <form> de su página web ASPX:

```
<form id="form1" runat="server">  
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"  
        ConnectionString="<%$ ConnectionStrings:bancaConnectionString %>"  
        SelectCommand="SELECT * FROM [cuenta]">  
    </asp:SqlDataSource>  
</form>
```

El control implementado es capaz de cambiar los datos correspondientes a la consulta en el momento adecuado y efectuar un data binding sin intervención por parte del programador. Para probar esta característica, basta con asociar el control a un componente de tipo GridView:

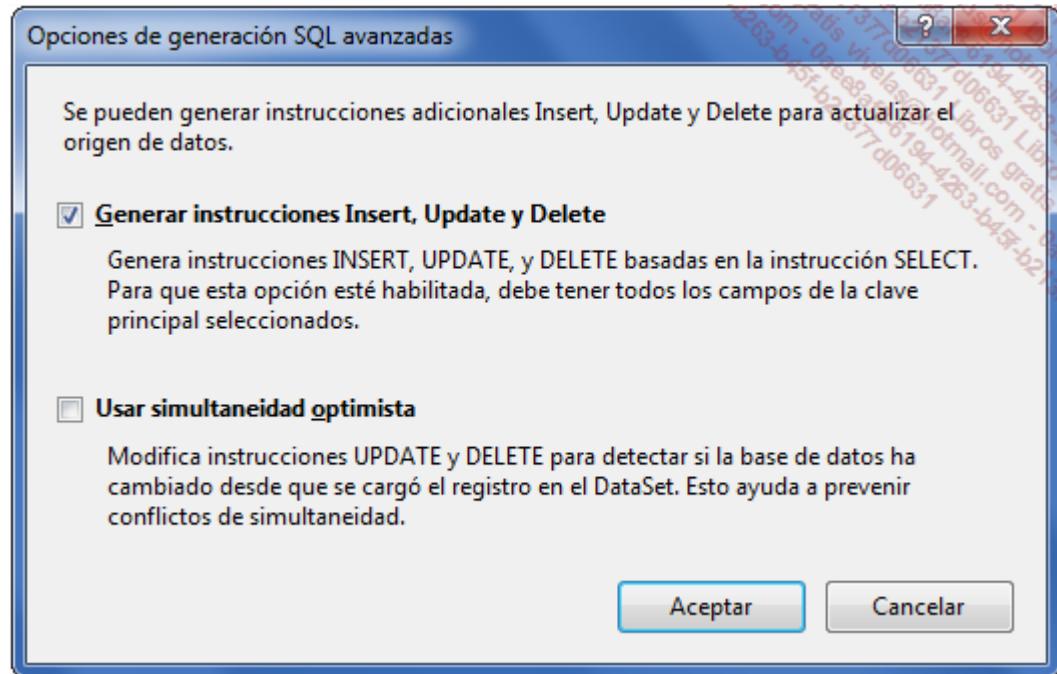


Cuando se escoge el origen de datos, Visual Studio y el servidor de aplicaciones ASP.NET trabajan de manera conjunta para representar las columnas presentes en la consulta Select. Sólo queda verificar el resultado en el navegador.



#### b. Consultas de actualización

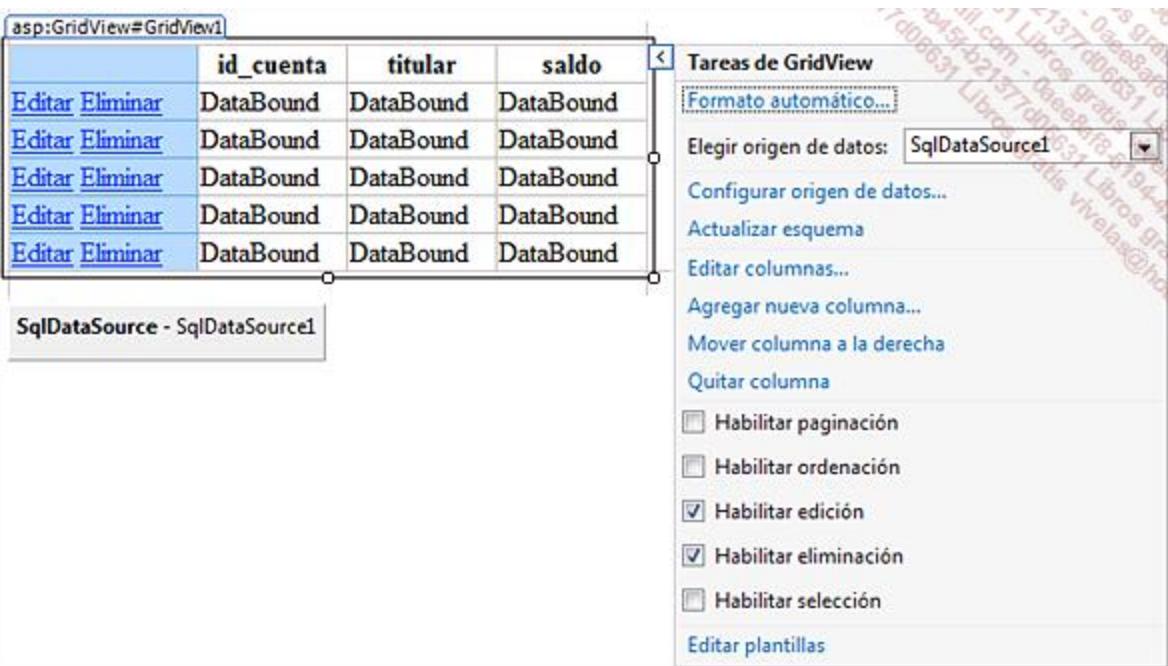
El DataAdapter, integrado dentro del control de origen de datos SQL, soporta también consultas de actualización Update, Insert y Delete. La activación de estos comandos se realiza haciendo clic en el botón **Opciones avanzadas** del asistente de configuración del control:



Tras completar todas las etapas del asistente, éste completa la definición XML del control:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:bancaConnectionString %>" 
SelectCommand="SELECT * FROM [cuenta]" DeleteCommand="DELETE FROM [cuenta]
WHERE
[id_cuenta] = @id_cuenta"
InsertCommand="INSERT INTO [cuenta] ([titular], [saldo]) VALUES
(@titular, @saldo)" UpdateCommand="UPDATE [cuenta] SET [titular] =
@titular, [saldo] = @saldo WHERE [id_cuenta] = @id_cuenta">
<DeleteParameters>
<asp:Parameter Name="id_cuenta" Type="Int32" />
</DeleteParameters>
<UpdateParameters>
<asp:Parameter Name="titular" Type="String" />
<asp:Parameter Name="saldo" Type="Decimal" />
<asp:Parameter Name="id_cuenta" Type="Int32" />
</UpdateParameters>
<InsertParameters>
<asp:Parameter Name="titular" Type="String" />
<asp:Parameter Name="saldo" Type="Decimal" />
</InsertParameters>
</asp:SqlDataSource>
```

Los componentes de presentación de datos asociados al origen de datos tendrían acceso al modo de edición:



### c. Parámetros

Las consultas de un control de origen de datos admiten **parámetros** declarados en XML en lugar de en C#. El programador los utiliza libremente, en particular a nivel de cláusulas **Where**.

Los valores de los parámetros dependen del entorno de la página ASPX:

CookieParameter	Una cookie le asigna valor al parámetro.
ControlParameter	Un control Webform provee el valor (lista, texto...).
FormParameter	El valor se basa en los datos enviados por HTTP.
ProfileParameter	El valor forma parte del perfil del usuario.
QueryStringParameter	Como con FormParameter, pero basado en la cadena de consulta.
SessionParameter	Un dato en la Session del usuario fija el valor.

Para ilustrar el uso de

parámetros, crearemos una página que contiene dos orígenes SQL. El primero alimenta un ComboBox. El segundo se asocia a un control DetailsView. Este último se configura a partir del valor seleccionado en la lista desplegable:

```
<!-- lista desplegable -->
<asp:DropDownList ID="DropDownList1" runat="server" DataSourceID=
"ds_lista">
```

```
    DataTextField="titular" DataValueField="id_cuenta"
    AutoPostBack="True">
</asp:DropDownList><br />
```

```

<!-- origen que alimenta el desplegable -->

<asp:SqlDataSource ID="ds_lista" runat="server"
ConnectionString=
"<%$ ConnectionStrings:bancaConnectionString %>" 
SelectCommand="SELECT * FROM [cuenta]">
</asp:SqlDataSource>

<br />
<!-- representación registro por registro -->

<asp:DetailsView ID="dv" runat="server" AutoGenerateRows="False"
DataKeyNames="id_cuenta"
DataSourceID="ds_detalle" Height="50px" Width="125px">
<Fields>
    <asp:BoundField DataField="id_cuenta" HeaderText="id_cuenta"
InsertVisible="False"
        ReadOnly="True" SortExpression="id_cuenta" />
    <asp:BoundField DataField="titular" HeaderText="titular"
SortExpression="titular" />
    <asp:BoundField DataField="saldo" HeaderText="saldo"
SortExpression="saldo" />
</Fields>
</asp:DetailsView>

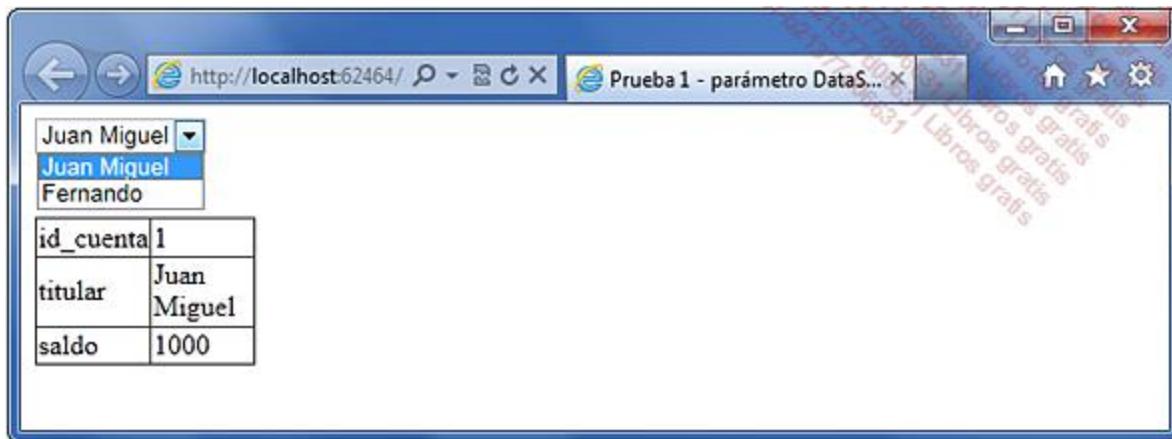
<!-- origen parametrizado -->

<asp:SqlDataSource ID="ds_detalle" runat="server"
ConnectionString=
"<%$ ConnectionStrings:bancaConnectionString %>" 
SelectCommand=
"SELECT * FROM cuenta WHERE ([id_cuenta] = @id_cuenta)">
<SelectParameters>
    <asp:ControlParameter ControlID="DropDownList1"
Name="id_cuenta" PropertyName="SelectedValue"

```

```

Type="Int32" />
</SelectParameters>
</asp:SqlDataSource>
```



## Parámetros con valores concretos por aplicación

El control SqlDataSource admite, a su vez, valores configurados por programación. En principio, los parámetros se declaran en SQL Server mediante el prefijo @ que los distingue de las columnas SQL. Hemos visto que estos parámetros sirven para las cláusulas Where y para los procedimientos almacenados que soportan los comandos.

Para crear un parámetro cuyo valor esté asignado por el programa y no por el entorno, no debe declararse con el prefijo @.

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1">
</asp:GridView>
<br />
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString=
"<%$ ConnectionStrings:bancaConnectionString %>">
  SelectCommand=
    "SELECT * FROM [cuenta] where [saldo]>@saldo">
  <SelectParameters>
    <asp:Parameter Name="saldo" Type="int32" DefaultValue="30" />
  </SelectParameters>
</asp:SqlDataSource>
```

El programa le asigna, a continuación, el valor al parámetro utilizando la propiedad DefaultValue, de tipo string:

```
protected void Page_Load(object sender, EventArgs e)
{
  SqlDataSource1.SelectParameters["saldo"].DefaultValue = "100";
}
```

<b>id_cuenta</b>	<b>titular</b>	<b>saldo</b>
1	Juan Miguel	1000
2	Fernando	250

#### d. Caché

Los controles de origen de datos SqlDataSource y AccessDataSource poseen propiedades que permiten determinar la estrategia de caché de los datos. Por defecto, la propiedad **EnableCaching** vale False y, por tanto, los datos se recargan prácticamente tras cada postback. Activando la caché, el programador puede fijar la estrategia mediante las propiedades **CacheDuration**, **CacheKeyDependency** y **CacheExpirationPolicy**.

Para alojar los datos en caché durante 30 segundos, configure las propiedades de la siguiente forma:

CacheDuration	30
CacheExpirationPolicy	Absolute

Para indicar que los datos deben invalidarse pasados 30 segundos de inactividad (si el programa no accede durante este tiempo), ajuste el valor de **CacheExpirationPolicy** a Sliding. Observe que los valores de estas propiedades pueden fijarse mediante código C#.

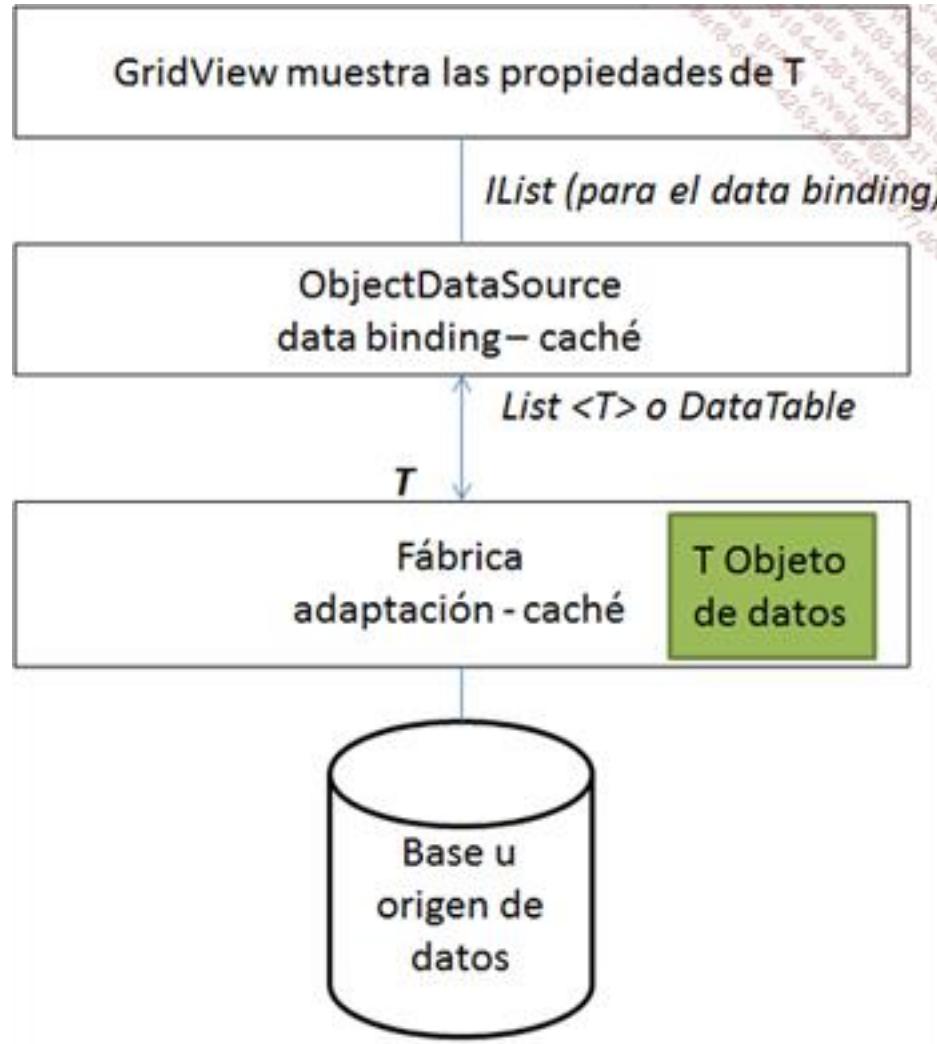
### 3. El proveedor ObjectDataSource

#### a. Principio

El control de origen de datos ObjectDataSource presenta características similares a los controles SqlDataSource y AccessDataSource, aunque el programador debe implementar toda la lógica de la persistencia.

Utilizar este control presenta un interés doble. Por un lado, el control SqlDataSource puede resultar exigente, aunque está bien integrado, y sus consultas deben copiarse de una página ASPX a otra, lo que genera mucho copiar-pegar escrito a fuego en el código. Por otro lado, los orígenes de datos no siempre son bases de datos SQL. Encontramos, con frecuencia, objetos de negocio más o menos dependientes de un servidor de aplicaciones (COM+, por ejemplo).

La implantación de un ObjectDataSource asegura al programador una continuidad en su desarrollo de interfaces gráficas, pues el control presenta la misma morfología que los demás controles de origen de datos, dejando una implementación totalmente libre. La arquitectura general se apoya en el concepto (patrón de diseño) de la fábrica de objetos:



Es importante para Visual Studio y ASP.NET que el tipo de objeto de datos se resuelva de forma clara, sin ello los componentes de presentación tendrían dificultades para identificar las "columnas" a representar. Como con los sistemas de mapping de objeto relacional, la hipótesis consiste en hacer que una instancia de objeto de datos equivalga a un registro SQL.

Si el objeto de datos es de tipo T, la fábrica utiliza los formatos List<T> y DataTable para alimentar el ObjectDataSouce. Cuando el control solicita la actualización del origen de datos (Update, Insert, Delete), utilizará directamente el tipo T como parámetro de los métodos que implementan el adaptador.

El componente gráfico encargado de representar los valores de cada objeto utiliza propiedades públicas de la clase T.

## b. Implementación

La implementación del control ObjectDataSource requiere el desarrollo de dos clases, la fábrica y el componente de datos.

### **El componente de datos CompteDataObject**

Un componente de datos posee, al menos, un constructor por defecto, accesible al control ObjectDataSource que debe poder instanciarlo. El programador agrega, a continuación, otros constructores en función de las necesidades de la aplicación. El componente expone, también,

propiedades públicas, tantas como "columnas" existan en la tabla de datos correspondiente a la clase.

```
public class CuentaDataObject
{
    #region Propiedad Id_cuenta
    private int id_cuenta;

    public int Id_cuenta
    {
        get { return id_cuenta; }
        set { id_cuenta = value; }
    }
    #endregion

    #region Propiedad Titular
    private string titular;

    public string Titular
    {
        get { return titular; }
        set { titular = value; }
    }
    #endregion

    #region Propiedad Saldo
    private decimal saldo;

    public decimal Saldo
    {
        get { return saldo; }
        set { saldo = value; }
    }
    #endregion
```

```
// prevemos un constructor público
public CuentaDataObject()
{
}
```

```

    // otro constructor

    public CuentaDataObject(int id_cuenta, string titular, decimal
saldo)
    {
        this.id_cuenta = id_cuenta;
        this.titular = titular;
        this.saldo = saldo;
    }
}

```

## La fábrica de objetos

La fábrica necesita la exposición de un constructor por defecto. Veremos más adelante cómo transmitir estos parámetros para facilitar su inicialización.

Es posible definir hasta cinco método públicos. Su nombre realmente no importa, pues se precisará en ObjectDataSource en tiempo de ejecución.

Método	Firma	Rol	He aquí el código del método
GetAll	List<T> GetAll() o DataTable Getall()	Implementa la instrucción SQL Select.	
GetCount	int GetCount()	Opcional. Cuenta el número de registros (útil para la paginación).	
Insert	void Insert(T row)	Implementa la instrucción SQL Insert. El control ObjectDataSource comunica los objetos por defecto mediante el objeto row.	
Update	void Update(T row)	Implementa la instrucción SQL Update. El control ObjectDataSource comunica los nuevos valores y la clave primaria mediante el objeto row.	
Delete	void Delete(T row)	Implementa la instrucción SQL Delete. La clave primaria se comunica mediante el objeto row.	

GetAll(). No presenta ninguna dificultad particular, se basa en una lista genérica List<T>:

```

public List<CuentaDataObject> GetAll()
{
    SqlConnection cx=null;
    List<CuentaDataObject> lista = new List<CuentaDataObject>();
}

```

```

try
{
    string cs;
    cs = ConfigurationManager.ConnectionStrings["bancaConnection
String"].ConnectionString;
}

```

```

        cx = new SqlConnection(cs);

        // todos los registros de la tabla cuenta
        string c = "select * from cuenta";
        SqlCommand sql = new SqlCommand(c, cx);
        cx.Open();
        SqlDataReader reader = sql.ExecuteReader();
        while (reader.Read())
        {
            // convierte cada linea en un objeto
            CuentaDataObject row = new CuentaDataObject();
            row.Id_cuenta = (int) reader["id_cuenta"];
            row.Titular = (string) reader["titular"];
            row.Saldo = (decimal) reader["saldo"];

            // agregar a la lista
            liste.Add(row);
        }
        reader.Close();
    }

    catch (Exception err)
    {
        throw err; // prever una acción específica
    }

    finally
    {
        if (cx != null)
            cx.Close();
    }
    return lista;
}

}

```

Los demás métodos se basan en el mismo principio.

### **Asociación de la fábrica al ObjectDataSource**

Como con un SqlDataSource, el asistente de configuración de un ObjectDataSource se ejecuta desde el Web Form o mediante la smart tag **Configure Data Source**.

El programador puede escoger el objeto que implementa la fábrica y los métodos asociados a cada operación.

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
DataObjectType="CuentaDataObject"
DeleteMethod="Delete"
InsertMethod="Insert"
SelectMethod="GetAll"
TypeName="FabricaCuenta"
UpdateMethod="Update"></asp:ObjectDataSource>
```

El atributo **TypeName** se corresponde con el tipo que implementa la fábrica. El tipo del objeto de datos se obtendrá por reflexión.

## Pruebas

El control ObjectDataSource se asocia, a continuación, a un control de presentación, un GridView en nuestro caso:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns=
"False" DataSourceID="ObjectDataSource1">
    <Columns>
        <asp:BoundField DataField="Id_cuenta" HeaderText=
"Id_cuenta" SortExpression="Id_cuenta" />
        <asp:BoundField DataField="Titular" HeaderText="Titular"
SortExpression="Titular" />
        <asp:BoundField DataField="Saldo" HeaderText="Saldo"
SortExpression="Saldo" />
    </Columns>
</asp:GridView>
```

Visual Studio ha sabido determinar los nombres de las propiedades públicas que servirán para comunicar los datos. Sólo queda realizar la prueba.

id_cuenta	titular	saldo
1	Juan Miguel	1000
2	Fernando	250
3	Luis Antonio	100
4	Juan Valdés	0

### c. Parámetros de creación

El programador dispone de tres eventos para controlar la instancia de la clase Fabrica.

ObjectCreating	Se produce antes de la instancia
ObjectCreated	Se produce después de la instancia
ObjectDisposing	Se produce tras su destrucción

Gestionando el evento **ObjectCreating**, es posible invocar un constructor específico y pasarle los parámetros:

```
protected void
ObjectDataSource1_ObjectCreating(object sender,
ObjectDataSourceEventArgs e)
{
    string cs = ConfigurationManager.ConnectionStrings
    ["bancaConnectionString"].ConnectionString;
    e.ObjectInstance = new FabricaCuenta(cs);
}
```

### d. Gestión de la caché

Como con todos los controles de origen de datos, ObjectDataSource es capaz de gestionar una caché. No obstante, esta operación sólo está accesible si el método GetAll devuelve un DataSet o un DataTable. En caso contrario, la propiedad EnableCaching debe desactivarse y el programador deberá gestionar, él mismo, su caché mediante HttpContext.Current.Cache.

### e. Una versión avanzada

Nuestra primera implementación no es perfecta, se contenta con ejecutar el proceso de construcción de un origen de datos de objetos, pero el valor agregado se basa débilmente en un control SqlDataSource.

Conviene modificar la construcción para promover el objeto de datos CuentaDataObject, simple reflejo de la tabla SQL Cuenta, a un objeto de negocio CuentaBusinessObject, que soporte los métodos de ejecución que SQL no puede gestionar por sí solo.

Po otro lado, la escritura de los procedimientos GetAll()... Delete() en el estilo anterior es contraproducente: es mucho más adecuado utilizar un TableAdapter.

## Modificación del objeto

La primera mejora supone que las propiedades del objeto Cuenta se asocien a las columnas de un DataRow fuertemente tipado. Esto evita tener que copiar los valores mediante un método GetAll.

Estas propiedades deben, también, funcionar sin un DataRow asociado: el control ObjectDataSource instancia, él mismo, el objeto de datos en el marco de las operaciones Insert, Update y Delete.

Por último, el objeto incluye nuevos métodos de negocio, basados en un componente Java EJB Entidad.

Todas las propiedades de tipo "columna" se basan en el mismo principio, por lo que sólo mostraremos el código correspondiente a la primera de ellas, id\_cuenta.

```
public class CuentaBusinessObject
{
    #region Propiedad row
    private BancaDataSet.cuentaRow row;
    /// <summary>
    /// Asocia las propiedades del objeto a una fila DataRow
    /// </summary>
    public BancaDataSet.cuentaRow Row
    {
        get { return row; }
        set { row = value; }
    }
    #endregion

    #region Propiedad Id_cuenta
    private int id_cuenta;
    public int Id_cuenta
    {
        get
        {
            if (row != null)
                return row.id_cuenta;
            else
                return id_cuenta;
        }
        set
    }
}
```

```

{
    if (row != null)
        row.id_cuenta = value;
    else
        id_cuenta = value;
}
}

#endifregion

#region "Propiedad Titular"
#endifregion

#region Propiedad Saldo
#endifregion

#region Constructor
public CuentaBusinessObject()
{
    row = null;
    id_cuenta = 0;
    titular = null;
    saldo = 0;
}
#endifregion

#region "Métodos de negocio"
#endifregion
}

```

## Instalar la caché en la fábrica

La implementación de la caché agrega una nueva propiedad a la fábrica, CacheDuration. Existen otras estrategias de gestión caché, también aplicables. El constructor fija una duración de caché por defecto; la propiedad, accesible desde el evento **ObjectCreating**, puede configurar otro valor.

```

public FabricaBusinessCuenta()
{
    cacheDuration = 45;
}

#region Propiedad CacheDuration

```

```

private int cacheDuration;
/// <summary>
/// Fija la duración en segundos para almacenar los datos en Caché
/// </summary>
public int CacheDuration
{
    get { return cacheDuration; }
    set { cacheDuration = value; }
}
#endregion

```

De hecho, todos los valores leídos se alojan en caché. El procedimiento interno GetTable tiene la responsabilidad de actualizarlos cuando expiran. Se utiliza un TableAdapter para facilitar esta extracción:

```

#region GetTable()
/// <summary>
/// Busca los datos alojados en caché y los renueva si es precis
/// </summary>
/// <returns>cuentaDataTable que contiene los registros </returns>
protected BancaDataSet.cuentaDataTable GetTable()
{
    BancaDataSet.cuentaDataTable tabla;
    tabla = HttpContext.Current.Cache["tabla_cuenta"] as BancaDataSet.
cuentaDataTable;
    if (tabla == null)
    {
        // los datos ya no están en la caché
        // es preciso renovarlos
        BancaDataSetTableAdapters.cuentaTableAdapter da = new
BancaDataSetTableAdapters.cuentaTableAdapter();
        tabla = da.GetData();
    }
}

```

```

// duración en caché 45 segundos
HttpContext.Current.Cache.Add( "tabla_cuenta",
    tabla,
    null,
    DateTime.Now.AddSeconds(cacheDuration),
    TimeSpan.Zero,
    CacheItemPriority.Normal,

```

```

        null);
    }

    return tabla;
}

#endregion
```

## Evolución de los métodos de la fábrica

Los cinco métodos públicos de la fábrica utilizan, en adelante, el procedimiento interno GetTable. Su implementación es mucho más directa:

```

#region GetAll()
/// <summary>
/// Devuelve los datos utilizando una lista de objetos
/// </summary>
/// <returns>Lista de objetos de negocio</returns>
public List<CuentaBusinessObject> GetAll()
{
    BancaDataSet.cuentaDataTable tabla = GetTable();

    // convertir la lista de registros
    // en una lista de objetos de negocio
    List<CuentaBusinessObject> lista = new List<CuentaBusinessObject>();
    for (int i = 0; i < table.Rows.Count; i++)
    {
        CuentaBusinessObject objeto = new CuentaBusinessObject();

        objeto.Row = tabla[i];
        lista.Add(objeto);
    }
}

return lista;
}

#endregion

#region GetCount()
/// <summary>
/// Método que cuenta el número de objetos
/// </summary>
/// <returns>Número de objetos</returns>
public int GetCount()
```

```

{
    return GetTable().Rows.Count;
}

#endregion

#region Update()
/// <summary>
/// Actualiza la base de datos a partir de un objeto
/// </summary>
/// <param name="objeto">Valores a actualizar</param>
public void Update(CuentaBusinessObject objeto)
{
    BancaDataSetTableAdapters.cuentaTableAdapter da = new
BancaDataSetTableAdapters.cuentaTableAdapter();
    BancaDataSet.cuentaDataTable tabla = GetTable();

    // encuentra la fila y la actualiza con los datos recibidos
    BancaDataSet.cuentaRow fila = tabla.FindByid_cuenta
(objeto.Id_cuenta);
    fila.titular = objeto.Titular;
    fila.saldo = objeto.Saldo;
    // actualiza la base de datos
    da.Update(tabla);
}
#endregion

```

```

#region Insert()
/// <summary>
/// Agrega un registro.
/// </summary>
/// <param name="objeto">Valores iniciales a insertar en la base
de datos</param>
public void Insert(CuentaBusinessObject objeto)
{
    BancaDataSetTableAdapters.cuentaTableAdapter da = new
BancaDataSetTableAdapters.cuentaTableAdapter();
    BancaDataSet.cuentaDataTable tabla = GetTable();

```

```

// crea una fila nueva y le asigna los valores recibidos
BancaDataSet.cuentaRow fila = tabla.NewcuentaRow();

fila.id_cuenta = objeto.Id_cuenta;
fila.titular = objeto.Titular;
fila.saldo = objeto.Saldo;

tabla.Rows.Add(fila);

// actualiza la base de datos
da.Update(tabla);

// sincroniza el objeto fila
objeto.Row = fila;
}

#endregion

#region Delete()
/// <summary>
/// Borra un registro / objeto
/// </summary>
/// <param name="objeto">Registro que se suprime</param>
public void Delete(CuentaBusinessObject objeto)
{
    BancaDataSetTableAdapters.cuentaTableAdapter da = new
    BancaDataSetTableAdapters.cuentaTableAdapter();
}

```

```

BancaDataSet.cuentaDataTable tabla = GetTable();

// crea una fila y le asigna los valores recibidos
BancaDataSet.cuentaRow fila = tabla.FindByid_cuenta
(objeto.Id_cuenta);

fila.Delete();

// actualiza la base de datos
da.Update(tabla);
}

#endregion

```

## Pruebas

Para probar cada operación, el componente GridView no basta. Utilizaremos también un DetailsView, encargado de seguir el registro seleccionado por el GridView. La asociación se establece mediante un evento:

```
protected void GridView1_SelectedIndexChanged(object sender,
EventArgs e)
{
    DetailsView1PageIndex = GridView1.SelectedIndex;
}
```

The screenshot shows a web browser window with the URL <http://localhost:62464/>. The title bar says "Prueba ObjectDataSource". The main content area displays a GridView with four rows of data. The first row is selected. Below the GridView, a DetailsView is displayed for the selected row (Id\_cuenta 2). The DetailsView shows the Titular (Fernando) and Saldo (250) for the selected row. At the bottom of the DetailsView, there are links for Editar, Eliminar, and Nuevo.

	<b>Id_cuenta</b>	<b>Titular</b>	<b>Saldo</b>
<a href="#">Editar</a>	1	Juan Miguel	1000
<a href="#">Eliminar</a>	2	Fernando	250
<a href="#">Seleccionar</a>	3	Luis Antonio	100
<a href="#">Editar</a>	4	Juan Valdés	0

**Id\_cuenta** 2  
**Titular** Fernando  
**Saldo** 250  
[Editar](#) [Eliminar](#) [Nuevo](#)  
1234

#### 4. El proveedor XmlDataSource

El proveedor XmlDataSource tiene como objetivo adaptar los archivos XML a los controles de presentación. Un archivo XML puede considerarse como un conjunto de tablas SQL cuya estructura se determina de antemano mediante un esquema XSD o incluso dinámicamente en tiempo de análisis.

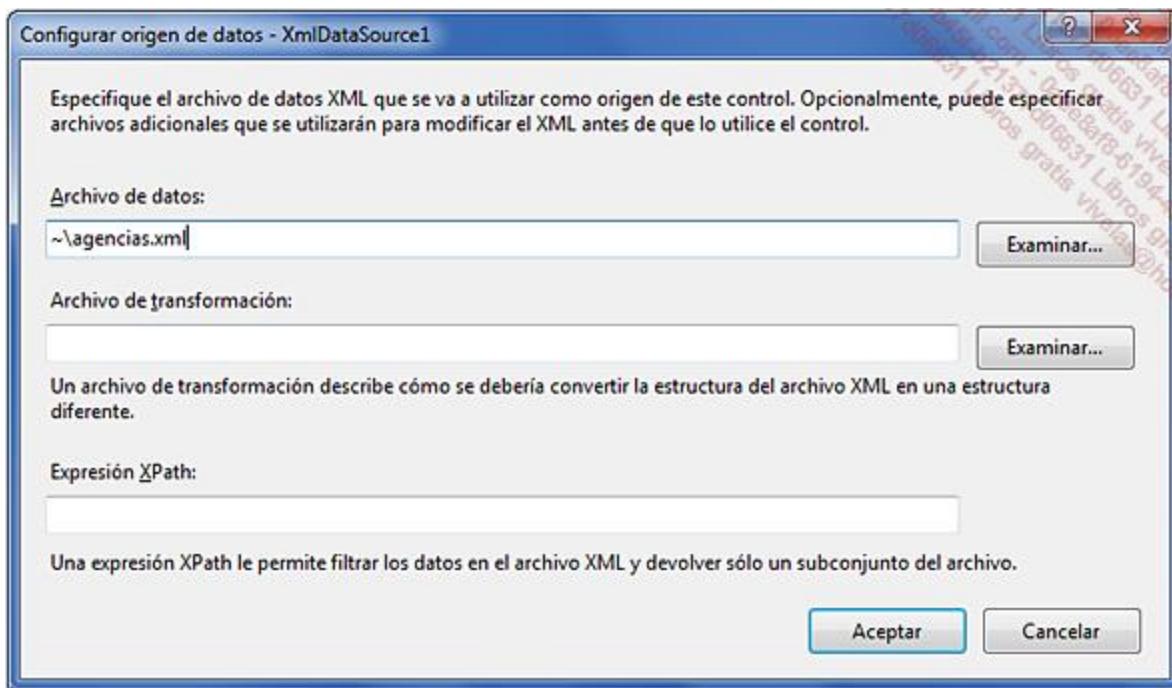
Para ilustrar su funcionamiento, partimos de un documento XML llamado agencias.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<agencias>
    <agencia nombre="Castaños" direccion="Avenida de los castaños 23"
ciudad="Madrid" codigopostal="28000" abierto_sabado="true" />
    <agencia nombre="Plaza de armas" direccion="Plaza de armas 2"
ciudad="Barcelona" codigopostal="08000" abierto_sabado="false">
    </agencia>
    <agencia nombre="Cipreses" direccion="Paseo de los cipreses 43"
```

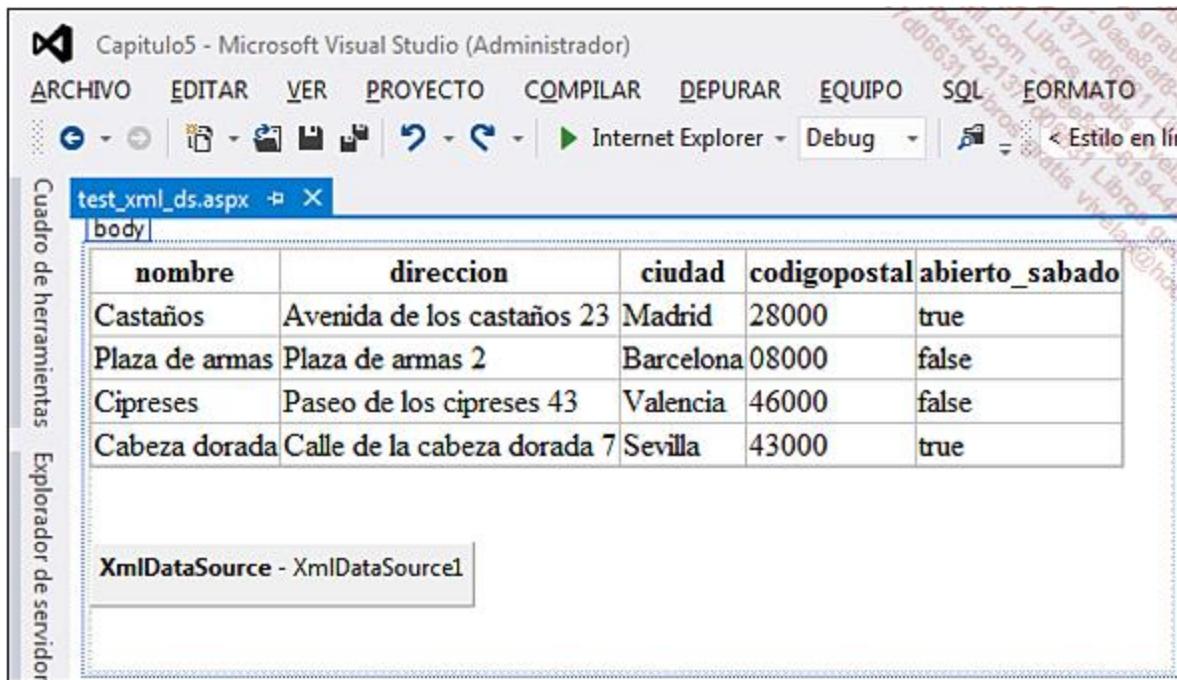
```
ciudad="Valencia" codigopostal="46000" abierto_sabado="false">
</agencia>
<agencia nombre="Cabeza dorada" direccion="Calle de la cabeza
dorada 7"
ciudad="Sevilla" codigopostal="43000" abierto_sabado="true">
</agencia>
</agencias>
```

Analizando su estructura, deducimos la existencia de una tabla llamada agencia que está formada por las columnas nombre, direccion, ciudad, codigopostal y abierto\_sabado.

La configuración del control XmlDataSource es muy sencilla. La única propiedad obligatoria es **esDataFile**. El carácter ~ (tilde) indica que la ruta es relativa a la carpeta del proyecto.



Una vez configurado, el origen de datos se asocia mediante el procedimiento habitual a un control de presentación. Es posible aplicar numerosas operaciones, a excepción de las que generan modificaciones de datos. Un archivo XML es, en efecto, similar a un origen de datos relacionales de sólo lectura, las solicitudes concurrentes de escritura no están soportadas.



## Agregar un procesado XSL

Los archivos XML pueden preprocesarse mediante programas XSL (hojas de estilo XML). En nuestro caso, el procesamiento consistirá en agregar una columna id\_agencia autonómica. El programa XSL correspondiente se almacenará en el archivo agencias.xsl:

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="agencias">
        <agencias>
            <xsl:apply-templates select="agencia"/>
        </agencias>
    </xsl:template>

    <xsl:template match="agencia">
```

# Componentes gráficos de presentación de datos

## 1. El componente GridView

El componente GridView es el sucesor de DataGrid, disponible todavía por motivos de mantenimiento pero que se ha vuelto obsoleto. El GridView retoma las directrices de funcionamiento y la implantación de su predecesor. Es, también, mucho más práctico para las operaciones de actualización.

### a. Presentación tabular de datos

Los orígenes de datos SQL están organizados en tablas y en registros. El GridView presenta cada registro en una fila, las columnas de la tabla serán las columnas de la tabla en pantalla.

#### Origen de datos

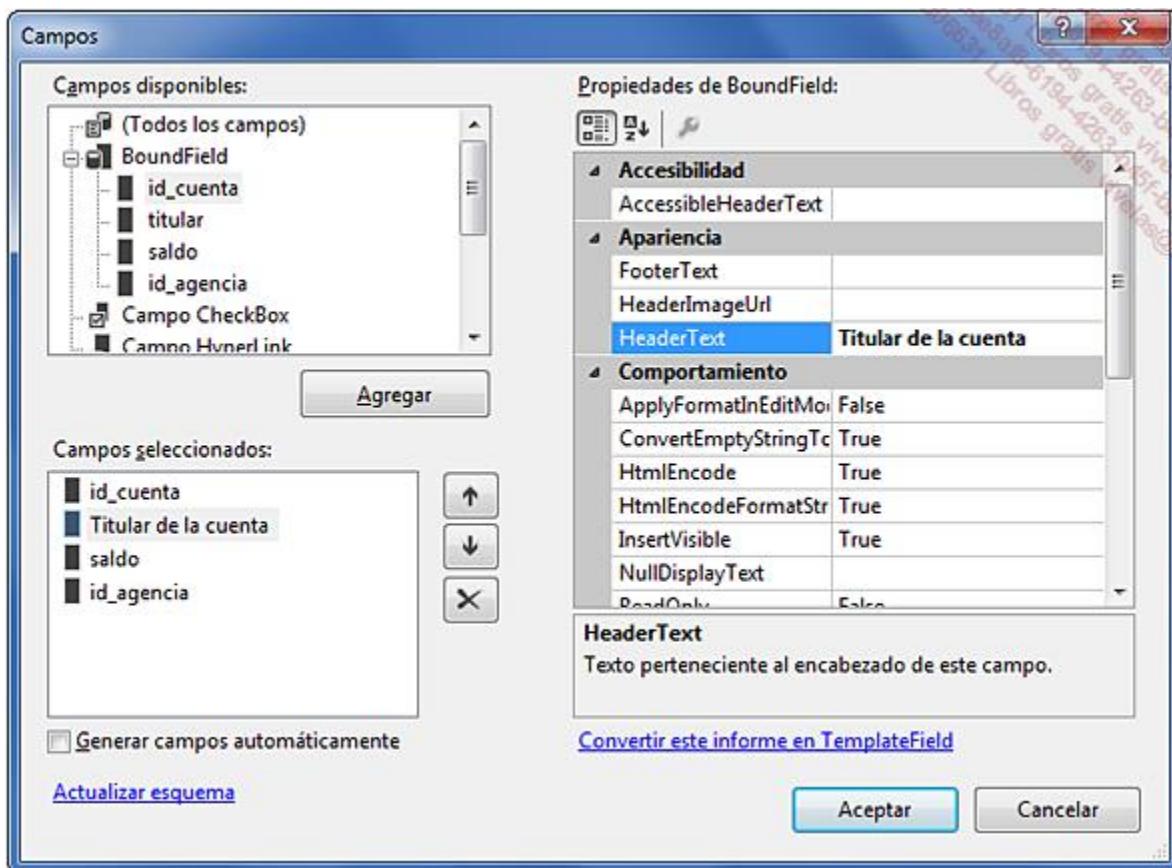
Las smart tags (accesos directos gráficos) resultan muy útiles para configurar un componente GridView y, en particular, su origen de datos (**Configurar origen de datos**). Si se trata de un control de origen (SqlDataSource, por ejemplo), no es necesario invocar al método DataBind(). Si se trata de un DataSet, el programador deberá realizar esta llamada como se describe en el caso de la versión 1.X.

#### Formato general

El formato general (colores, tamaño, apariencia) se define mediante la barra de propiedades. Como con el DataGrid que reemplaza, es posible aplicar formatos estándar mediante la smart tag**Formato automático**. No es posible personalizar la lista de formatos estándar que tiene por defecto Visual Studio.

#### Columnas

Por defecto, el GridView detecta los campos presentes a nivel de origen de datos y genera, para cada uno de ellos, una columna "HTML". El programador decide el orden de las columnas y aplica un formato a los valores correspondientes. Utiliza, para ello, la smart tag **Editar columnas**.



Como el DataGrid, GridView proporciona distintos tipos de columnas:

BoundField	En el modelo por defecto, utiliza un Label y un TextBox para presentar y editar el valor.
CheckboxField	Utiliza una casilla de selección para editar el valor. Funciona bien con el tipo SQL bit (equivalente a bool).
HyperLinkField	El valor de la columna se utiliza para formatear un enlace hipertexto.
ImageField	El valor de la columna configura la visualización de una imagen.
ButtonField	El valor de la columna configura la representación de un botón.
CommandField	Acciones sobre el GridView: selección, edición, validación, anulación y borrado.

A

menudo, las columnas se asocian a BoundField, que se corresponde con BoundColumn del DataGrid. El uso del tipo de columna HyperLinkField está indicado para crear URL parametrizadas con la clave primaria de un registro.

Consideremos la página operaciones.aspx, encargada de mostrar las operaciones relacionadas con una cuenta. Esta página recibe en su cadena de consulta (Query String) el identificador de la cuenta, id\_cuenta. Esto configura las siguientes URLs:

```
http://localhost/capitulo5_c/operacion.aspx?id_cuenta=1
http://localhost/capitulo5_c/operacion.aspx?id_cuenta=2
```

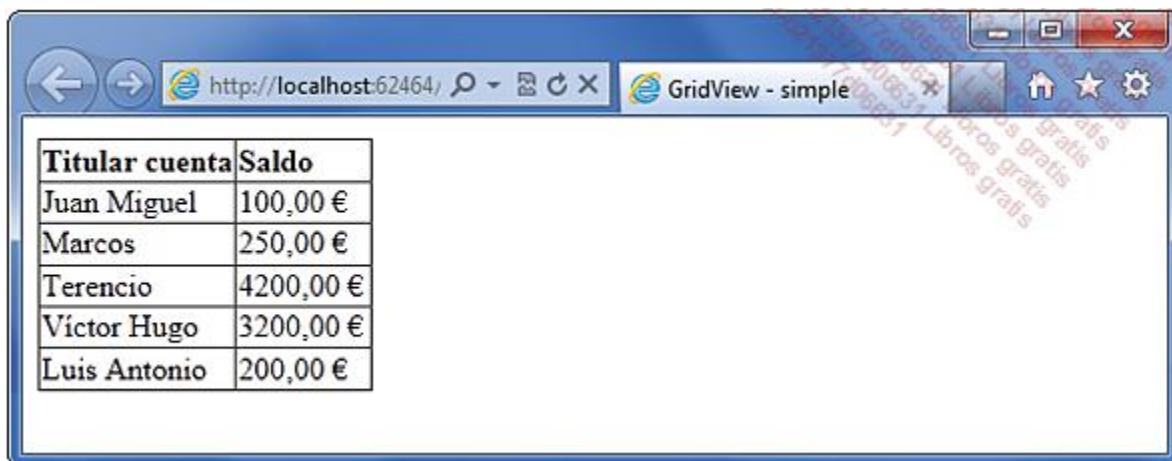
```
...
```

```
http://localhost/capitulo5_c/operacion.aspx?id_cuenta=15
```

Para crear un enlace parametrizado, agregamos una columna de tipo HyperLinkField. Sus propiedades se definen de la siguiente manera:

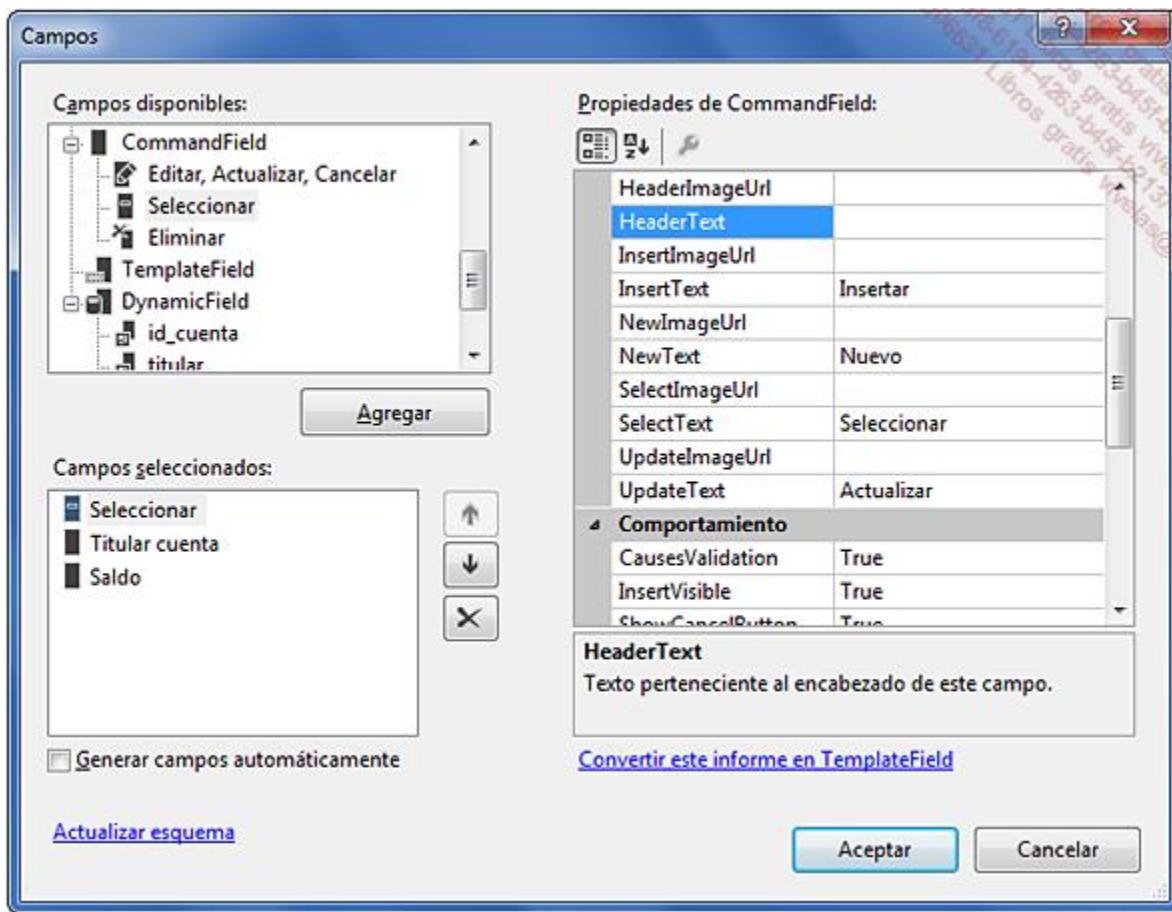
DataNavigatesUrlFields	id_cuenta	Los
DataNavigateUrlFormatString	operacion.aspx?id_cuenta={0}	
TextField	titular	
TextFormatString	Operaciones de {0}	

formateadores {0} se reemplazan en tiempo de ejecución por los valores de los campos indicados, id\_cuenta y titular respectivamente.



## b. Operaciones de selección y de navegación

El GridView tiene una propiedad, **SelectedIndex**, que indica qué fila está seleccionada. El valor de esta propiedad es o bien negativo (cuando no hay ninguna fila seleccionada), o bien positivo o nulo. El programador tiene la posibilidad de escribir en esta propiedad para seleccionar una fila, y el usuario puede hacer lo mismo gracias a la columna botón de comando Select. Una smart tag, **Habilitar selección**, evita tener que editar manualmente las columnas, operación que puede resultar, a pesar de todo, necesaria para personalizar la apariencia del comando.



La selección de una fila puede detectarse gracias a los eventos **SelectedIndexChanged** y **SelectedIndexChanging**. El programa sincroniza de este modo el GridView con otros controles que trabajan con un único registro, como por ejemplo un DetailsView o un FormView.

### Paginación de los registros

Como las tablas SQL son susceptibles de almacenar varios miles de registros, el GridView tiene la posibilidad de paginar su representación, es decir, segmentar su presentación en páginas con un número de filas fijado por el programador.

La activación de la paginación se realiza mediante la propiedad **AllowPaging** o la smart tag **Habilitar paginación**. La propiedad **PageSize** indica cuántos registros pueden figurar en la misma página. Cuando se activa la paginación, el GridView muestra un selector de página de tipo numérico donde figura cada índice de página. El otro tipo de selector está equipado con dos comandos < página anterior y página siguiente >.

Existe, también, la propiedad **PageCount** que indica cuántas páginas se definen a partir del origen de datos. De hecho, PageCount es igual al número total de registros dividido entre el valor de la propiedad **PageSize**, redondeado al entero superior.

Por último, la propiedad **PageIndex** determina qué página está activa.

El programador debe ser muy riguroso en sus cálculos de índice. La propiedad **SelectedIndex** varía entre 0 y **PageSize**. Cuando está activa la paginación, sin

ordenar los registros, el verdadero índice en el origen de datos viene determinado por la siguiente fórmula: índice = SelectedIndex + PageIndex\*PageSize

El siguiente programa nos servirá para validar nuestro cálculo:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
    lbl_info.Text = string.Format("PageIndex =
{0}<br>PageCount =
{1}<br>PageSize = {2}<br>SelectedIndex = {3}",
        GridView1.PageIndex,
        GridView1.PageCount,
        GridView1.PageSize,
        GridView1.SelectedIndex);
}
```

### c. Claves y operaciones de actualización

El cálculo del índice de la fila seleccionada no siempre es suficiente, ni práctico. Resulta, también, útil determinar el valor de la clave de la fila en curso. El GridView permite, precisamente, la posibilidad de definir una o varias columnas clave mediante la propiedad **DataKeyNames**. En nuestro ejemplo, esta propiedad toma el valor id\_cuenta, clave primaria de la tabla cuenta.

Tras cambiar o refrescar el esquema del origen de datos, es obligatorio informar de nuevo el valor de esta propiedad, sin ella las operaciones de actualización no funcionarán correctamente y, lo que es peor, sin un mensaje de error explícito.

#### Valor de la clave seleccionada

Cuando la clave está formada por una única columna, el programador determina su valor a partir de la fila seleccionada mediante la propiedad **SelectedDataKey.Value**:

```
int id_cuenta_seleccionada = (int)
GridView1.SelectedDataKey.Value;
```

Existe una colección **Value** para las claves formadas por varias columnas.

#### Operaciones de actualización

Cuando el origen de datos dispone de comandos de actualización, y a condición de que la propiedad DataKeyNames esté correctamente informada, el GridView se encarga de la edición y el borrado de los registros. Las smart tags respectivamente **Habilitar edición** y **Habilitar eliminación** corresponden con las columnas con los comandos Edit y Delete.

Respecto al DataGrid, el GridView no es, realmente, mucho más eficaz realizando estas operaciones. Cuando no se enlaza, no es necesaria ninguna programación.

Si no se realiza una personalización explícita, los campos SQL se muestran mediante controles Label y se editan mediante TextBox. Para modificar esta ergonomía es preciso convertir las columnas enlazadas en columnas plantilla (template). Esta manipulación se describe un poco más adelante.

	Titular cuenta	Saldo
<a href="#">Editar</a> <a href="#">Seleccionar</a>	Juan Miguel	100,00€
<a href="#">Editar</a> <a href="#">Seleccionar</a>	Marcos	250,00€
<a href="#">Editar</a> <a href="#">Seleccionar</a>	Terencio	4200,00€
<a href="#">Editar</a> <a href="#">Seleccionar</a>	Víctor Hugo	3200,00€
<a href="#">Actualizar</a> <a href="#">Cancelar</a>	Luis Antonio	200

#### d. Formateo y ordenación

##### Formateo de los valores

Es posible aplicar formatos a los valores mostrados en las distintas columnas de un GridView. Como en el ejemplo del enlace hipertexto parametrizado, la sintaxis se basa en el formateador general {0} o {0:format}.

Los valores más habituales son los siguientes:

c	Currency. Aplica el formato moneda.
D	Fecha.
d MMMM yyyy	Día, nombre del mes, año con cuatro cifras.
N	Número.

El

programador encontrará la lista completa de formatos aplicables en la documentación del framework .NET buscando, por ejemplo, la entrada correspondiente al método Format() de la clase string.

El formato de una columna está definido por su propiedad **DataFormatString**. El formateador{0} puede estar precedido y seguido de texto, como, por ejemplo, en string.Format() o en Console.WriteLine.

Existen otras tres propiedades muy importantes para controlar la apariencia de una columna:

HorizontalAlign	Izquierda, derecha, centrado.
ApplyFormatInEditMode	Puede ser necesario para eliminar el símbolo de moneda cuando la columna está en modo edición.
HtmlEncode	Debe valer False para que funcione el formateo.

##### Ordenación por columna

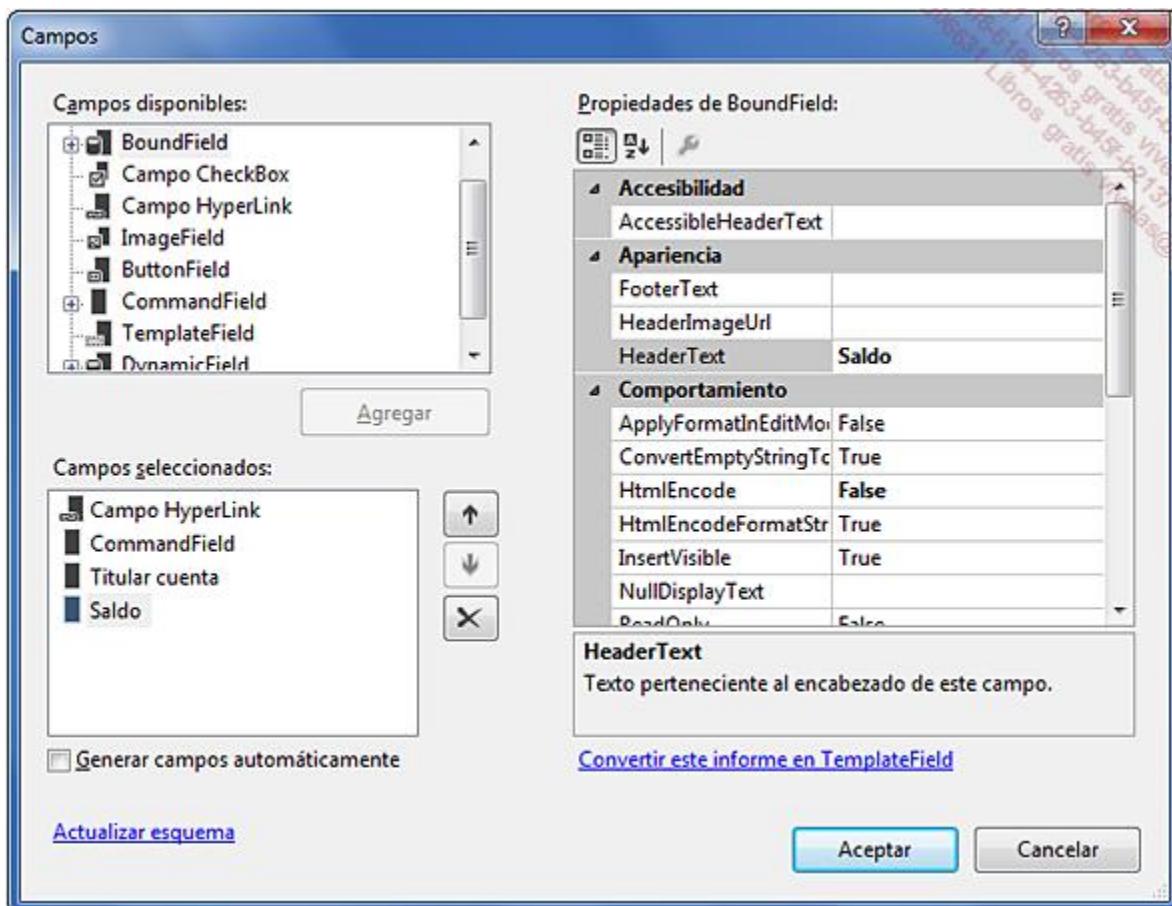
Activando la propiedad **AllowSorting**, el GridView utiliza un DataView para ordenar los datos según el valor de la propiedad **SortExpression** asociada a cada columna.

Cuando la propiedad **SortExpression** está definida por una columna, aparece un enlace en la representación gráfica del encabezado que permite realizar la ordenación de los datos. Cuando la propiedad está vacía, la columna no puede utilizarse para ordenar.

	<b>Titular cuenta</b>	<b>Saldo</b>
<a href="#">Editar Seleccionar</a>	Juan Miguel	100,00€
<a href="#">Editar Seleccionar</a>	Luis Antonio	200,00€
<a href="#">Editar Seleccionar</a>	Marcos	250,00€
<a href="#">Editar Seleccionar</a>	Víctor Hugo	3200,00€
<a href="#">Editar Seleccionar</a>	Terencio	4200,00€

#### e. Columnas plantilla

Es frecuente personalizar la ergonomía de una columna. La pareja Label - TextBox no siempre se adapta bien a la situación. Para poder personalizar una columna de datos, ésta debe convertirse en una columna plantilla (**Convertir este campo en TemplateField**).



El GridView, que deriva de TemplateControl, integra dos modelos HTML explícitos para la columna convertida, uno para la representación, ItemTemplate, y otro para la edición, EditItemTemplate:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="id_cuenta" DataSourceID="SqlDataSource1"
    OnSelectedIndexChanged="GridView1_SelectedIndexChanged" PageSize="3">
    <Columns>
        <asp:BoundField DataField="titular" />
        <asp:BoundField DataField="saldo" />
        <asp:TemplateField HeaderText="id_agencia">
            <EditItemTemplate>
                <asp:TextBox ID="TextBox1" runat="server" Text=
                    '<%# Bind("id_agencia") %>'></asp:TextBox>
            </EditItemTemplate>
            <ItemTemplate>
                <asp:Label ID="Label1" runat="server" Text=
                    '<%# Bind("id_agencia") %>'></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

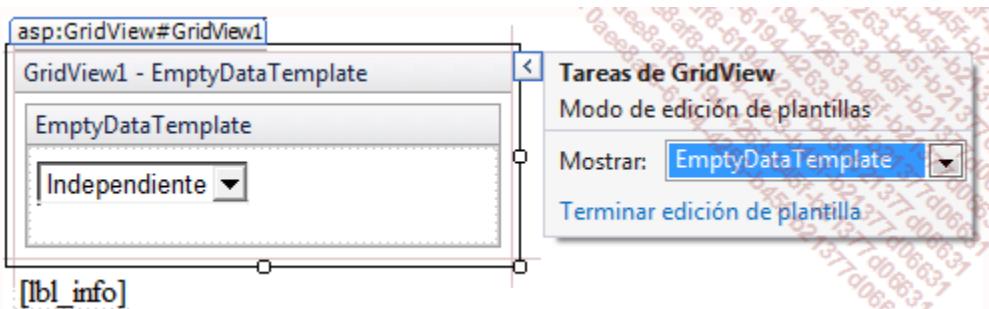
```

</asp:TemplateField>
</Columns>
</asp:GridView>

```

El programador tiene, ahora, la libertad para reemplazar el Label y el TextBox por controles escogidos entre los controles estándar de ASP.NET, controles de usuario o controles personalizados.

La definición de plantillas se realiza directamente desde el código HTML, o en modo diseño gracias a la smart tag **Editar plantillas**:



#### f. Enlace bidireccional

El GridView incluye el enlace de datos (data binding), sin apenas escribir código! Hemos visto cómo el framework ASP.NET aporta su ayuda cuando es necesario realizar programaciones repetitivas y, por tanto, fáciles de sistema-tizar.

Para ello, la sintaxis habitual de data binding, compleja y débilmente tipada, se reemplaza por una sintaxis más corta y más directa:

ASP.NET 1.X	A partir de ASP.NET 2.0.
DataBinder.Eval(Container.DataItem, "columna")	Eval("columna")

La  
sintaxis  
1.X  
sigue  
siendo,

no obstante, aplicable en ciertos componentes, como ocurría con nuestro control personalizado PictureBrowser.

A continuación, el enlace de datos se vuelve bidireccional. El valor de la columna de un registro sirve, en primer lugar, para su representación y, a continuación, un control capaz de realizar la edición actualiza dicho valor. En este caso, la palabra reservada **Eval** se convierte en **Bind**.

Por ello, podríamos modificar nuestro modelo de columna "agencia" para utilizar Eval en la representación y Bind en la edición. El siguiente modelo daría, exactamente, el mismo resultado:

```

<asp:TemplateField HeaderText="id_agencia">

    <EditItemTemplate>
        <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind
("id_agencia") %>'></asp:TextBox>
    </EditItemTemplate>

```

```

<ItemTemplate>
    <asp:Label ID="Label1" runat="server" Text='<%# Eval
("id_agencia") %>'></asp:Label>
</ItemTemplate>
</asp:TemplateField>

```

### g. Gestionar los enlaces

El caso de la columna Agencia ilustra una de las restricciones del uso de GridView: el enlace. La columna cuenta.id\_agencia es, en efecto, una clave externa hacia la tabla Agencia. Conviene representar el nombre de la agencia en lugar de su código, y proporcionar una lista desplegable en tiempo de edición en lugar de un área de texto.

#### Resolver el enlace en SQL

El método más sencillo para resolver el problema del enlace consiste en escribir una pequeña consulta SQL que "recupera" el nombre de la agencia.

El generador de consultas del control SqlDataSource facilita enormemente la elaboración de la sentencia SQL deseada:

Columna	Alias	Tabla	Resul...	Tipo de orden	Criterio de or...	Filtro	O...	O...
<input checked="" type="checkbox"/> id_cuenta		cuenta	<input checked="" type="checkbox"/>					
<input checked="" type="checkbox"/> titular		cuenta	<input checked="" type="checkbox"/>					
<input checked="" type="checkbox"/> saldo		cuenta	<input checked="" type="checkbox"/>					
<input checked="" type="checkbox"/> nombre_agen...		agencia	<input checked="" type="checkbox"/>					

```

SELECT cuenta.id_cuenta, cuenta.titular, cuenta.saldo, agencia.nombre_agencia
FROM cuenta INNER JOIN
     agencia ON cuenta.id_agencia = agencia.id_agencia

```

Sólo queda asociar el origen SQL al control GridView para verificar la resolución del enlace:

<b>id_cuenta</b>	<b>titular</b>	<b>saldo</b>	<b>nombre_agencia</b>
1	Juan Miguel	100	Castaños
2	Marcos	250	Plaza de armas
3	Terencio	4200	Cipreses
4	Víctor Hugo	3200	Castaños
5	Luis Antonio	200	Plaza de armas

Este método presenta, no obstante, un inconveniente: es difícil (casi imposible) elaborar consultas para la actualización de la tabla completa. Esta solución no es, por tanto, aplicable en todos los casos.

### **Utilizar un control doblemente enlazado**

Otra forma de abordar el problema consiste en usar un control doblemente enlazado, como por ejemplo una lista (desplegable o no). Por un lado, la lista se alimenta con un primer SqlDataSource correspondiente a la tabla agencia, mientras que la propiedad SelectedValue está ligada a la columna cuenta.id agencia, objeto de un segundo SqlDataSource:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
DataKeyNames="id_cuenta"
DataSourceID="ds_cuenta">

    <Columns>
        <asp:BoundField DataField="titular" />
        <asp:TemplateField>
            <EditItemTemplate>

<asp:DropDownList
    ID="lst_edititem_agencia" runat="server"
    DataSourceID="ds_agencia"
    DataTextField="nombre_agencia"
    DataValueField="id_agencia"
    SelectedValue='<%# Bind("id_agencia") %>'>
</asp:DropDownList>
            </EditItemTemplate>

            <ItemTemplate>

<asp:DropDownList ID="lst_item_agencia" runat="server"
    DataSourceID="ds_agencia"
    DataTextField="nombre_agencia"
    DataValueField="id_agencia"
```

```

    SelectedValue='<%# Eval("id_agencia") %>'>
</asp:DropDownList>
</ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>

```

	<b>Titular</b>	<b>Agencia</b>
<a href="#">Editar</a>	Juan Miguel	Castaños <input type="button" value="▼"/>
<a href="#">Actualizar Cancelar</a>	Marcos	Plaza de armas <input type="button" value="▼"/>
<a href="#">Editar</a>	Terencio	Cipreses <input type="button" value="▼"/>
<a href="#">Editar</a>	Víctor Hugo	Castaños <input type="button" value="▼"/>
<a href="#">Editar</a>	Luis Antonio	Plaza de armas <input type="button" value="▼"/>

Respecto al enlace SQL, esta solución tiene la ventaja de que funciona cuando se editan los registros. No obstante, el número de controles que podemos enlazar doblemente está limitado. Además, es casi imposible impedir al usuario seleccionar en la representación un valor que no se corresponde con el que hay registrado en la base de datos.

### Utilizar una función auxiliar

Las listas no están, por tanto, adaptadas a la representación; utilizar su propiedad `Enable=false` perjudicaría, a la vez, la estética y la ergonomía de la página. En este caso, es preferible utilizar un label alimentado con un método de tipo code-behind.

La construcción HTML es una mezcla de los dos primeros enfoques:

```

<asp:TemplateField HeaderText="Agencia">
    <EditItemTemplate>
        <asp:DropDownList ID="lst_edit_agencia" runat="server"
            DataSourceID="ds_agencia"
            DataTextField="nombre_agencia"
            DataValueField="id_agencia"
            SelectedValue='<%# Bind("id_agencia") %>'>
    </asp:DropDownList>
</EditItemTemplate>

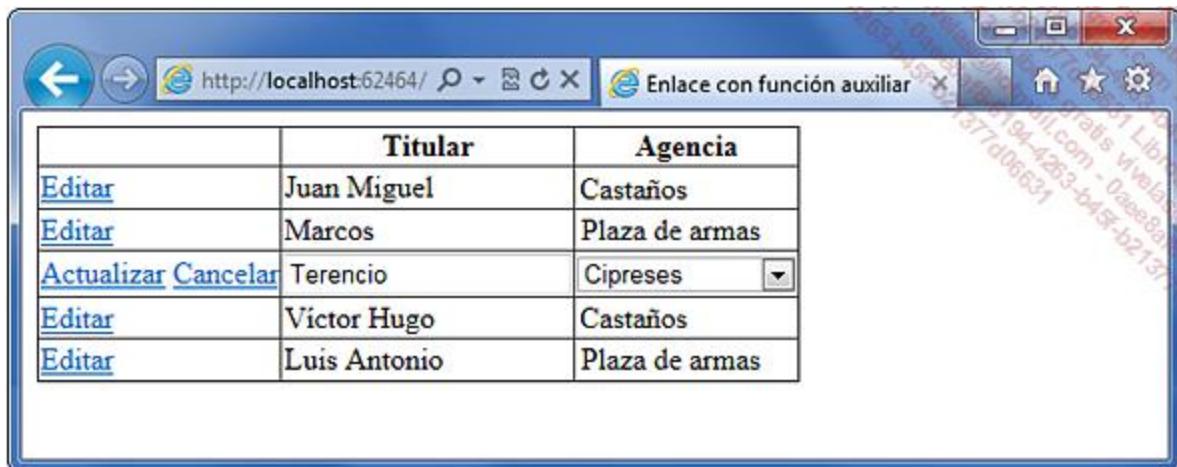
    <ItemTemplate>
        <asp:Label ID="Label1" runat="server"
            Text='<%# getNombreAgencia((int) Eval("id_agencia")) %>'>
    </ItemTemplate>

```

```
</asp:Label>  
</ItemTemplate>  
</asp:TemplateField>
```

La función getNombreAgencia está implementada en el código subyacente. Utiliza un TableAdapter para encontrar rápidamente la información. La tabla Agencia se carga una sola vez durante la carga de la página, incluso podría memorizarse en caché.

```
public partial class enlace_aux : System.Web.UI.Page  
{  
    private DataSetAgencia dsa;  
    protected void Page_Load(object sender, EventArgs e)  
    {  
        dsa = new DataSetAgencia();  
        DataSetAgenciaTableAdapters.agenciaTableAdapter da =  
new DataSetAgenciaTableAdapters.agenciaTableAdapter();  
        da.Fill(dsa.agencia);  
    }  
  
    protected string getNombreAgencia(object oid)  
    {  
        if (oid == null || !(oid is int))  
            return "";  
  
        int id_agencia=(int) oid;  
        try  
        {  
            return dsa.agencia.FindByid_agencia(id_agencia).nombre_agencia;  
        }  
        catch  
        {}  
        return "";  
    }  
}
```



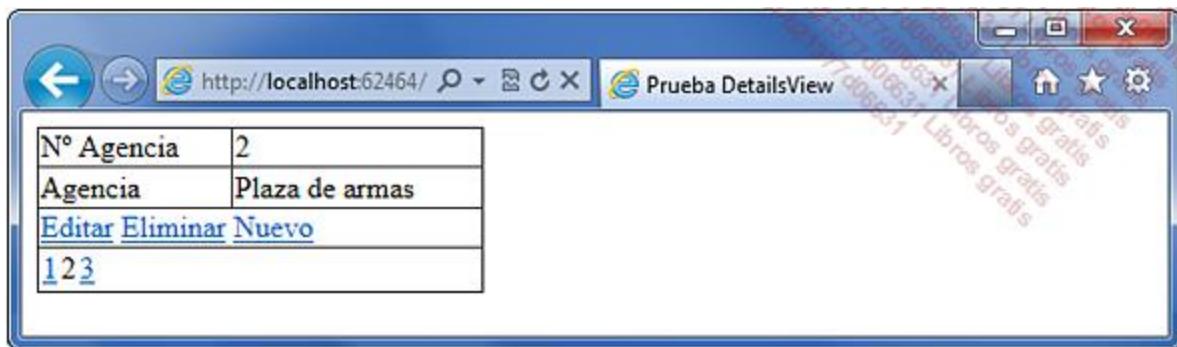
## 2. El componente DetailsView

### a. Presentación de DetailsView

A diferencia del GridView, el componente DetailsView trabaja sobre un registro a la vez, presentando verticalmente. Para el DetailsView, cada registro se llama Page, y la implantación de la navegación de un registro a otro se realiza mediante la smart tag **Habilitar paginación**.

Posee, prácticamente, el mismo juego de propiedades y funciona de manera análoga al GridView en lo relativo al formateo y a las plantillas de las columnas.

El DetailsView es, a su vez, capaz de insertar registros. La función se activa mediante la smart tag **Habilitar inserción**.



### b. Los eventos

Como los demás controles de presentación de datos (GridView, FormView), DetailsView expone eventos ligados a la actualización de datos. Los eventos funcionan en pareja, uno se produce antes de realizar la operación, y el otro después. El primer tipo de evento puede inhibir la operación o modificar los valores introducidos por el usuario. El segundo sirve, por lo general, para controlar el buen funcionamiento de la operación, por ejemplo el registro efectivo de los datos en la base de datos SQL.

ItemCreated	Se produce cuando se crea un registro.	De este modo,
ItemDeleted	Se produce cuando se elimina un registro.	
ItemInserting	Se produce antes de realizar una operación INSERT.	
ItemInserted	Se produce tras realizar una operación INSERT.	

ItemUpdating	Se produce antes de realizar una operación UPDATE.
ItemUpdated	Se produce tras realizar una operación UPDATE.

programa puede controlar los datos antes de que se envíen a la base de datos:

```
protected void DetailsView1_ItemUpdating(object sender,
DetailsViewUpdateEventArgs e)
{
    string agencia = (string) e.OldValues["nombre_agencia"];
    e.NewValues["nombre_agencia"] = agencia.ToLower();

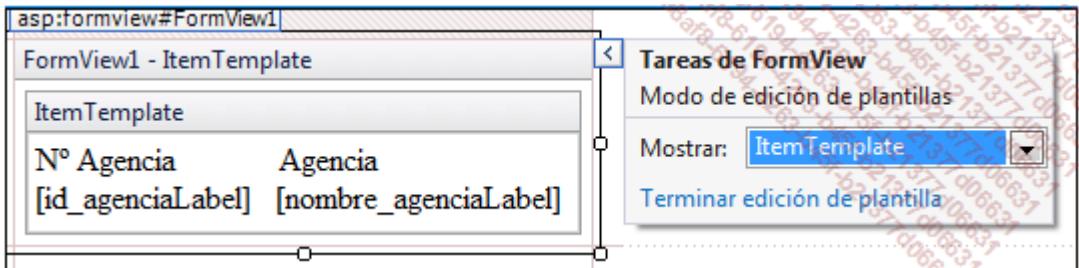
    e.Cancel = false; // ;Valor por defecto!
}
```

La propiedad **e.Cancel** vale, por defecto, false. La última línea del método no tiene ningún efecto, aunque muestra cómo podría inhibirse el proceso de actualización.

### c. El componente FormView

Muy similar a DetailsView, FormView difiere en dos puntos:

- Su presentación de un registro es totalmente libre.
- No existe el concepto de columna; todos los valores se exponen mediante modelos particulares - ItemTemplate, InsertItemTemplate, EditTemplate...

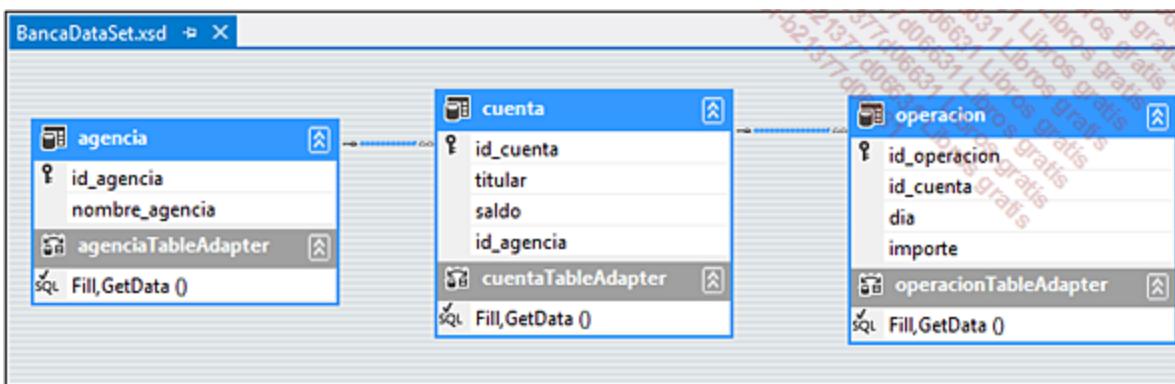


## Crear informes con Reporting Services

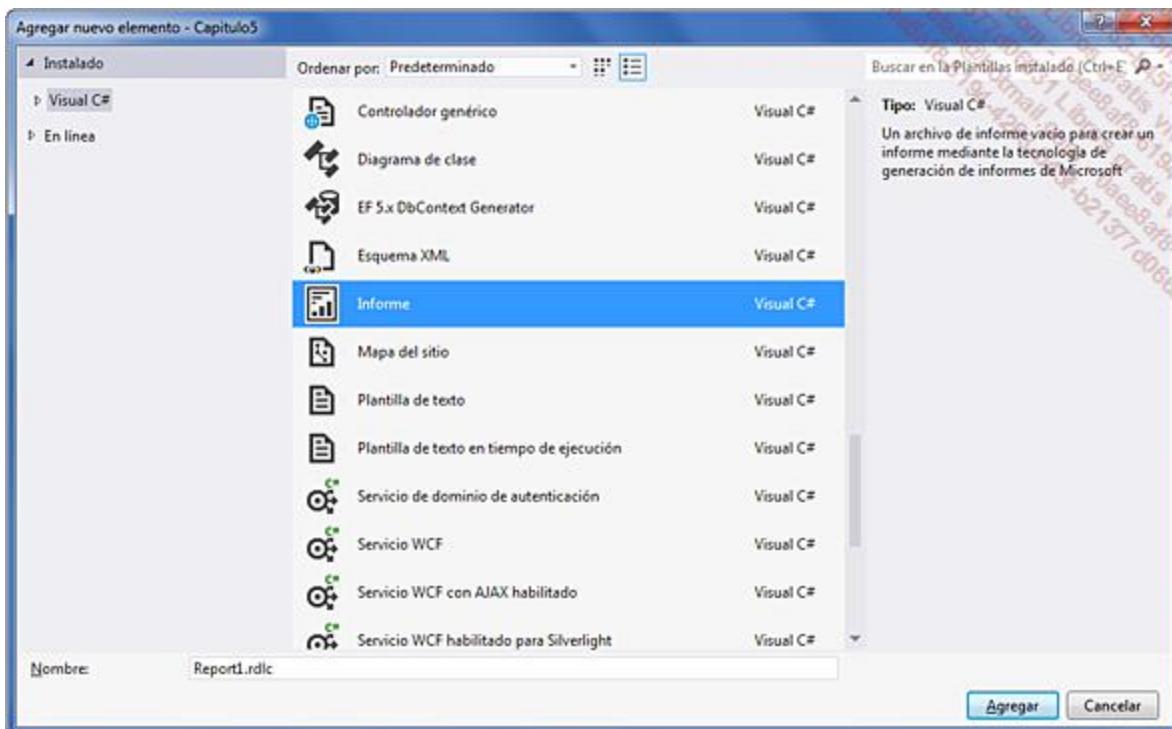
Cuando se quiere imprimir datos desde aplicaciones web, consideramos los informes (report en inglés). Es la función del componente Reporting Services disponible en ASP.NET.

### 1. Crear un informe

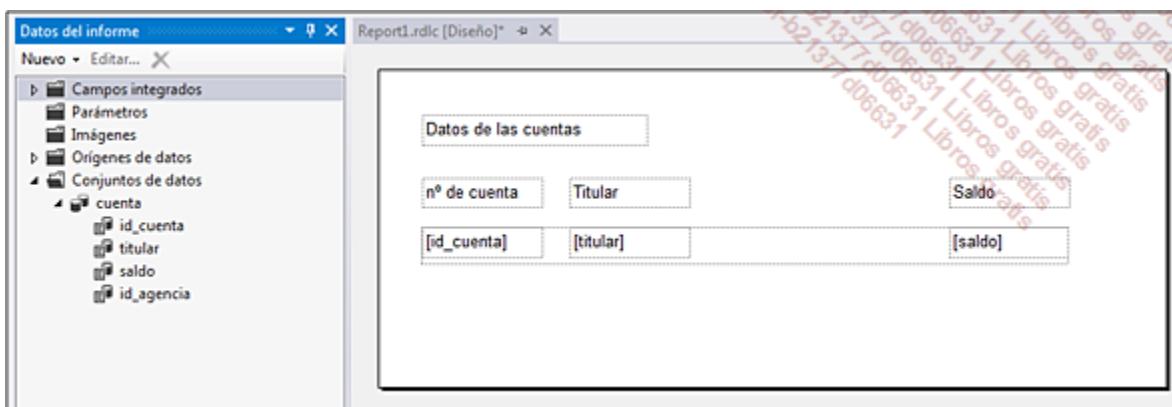
Resulta muy práctico definir un informe a partir de un DataSet tipado:



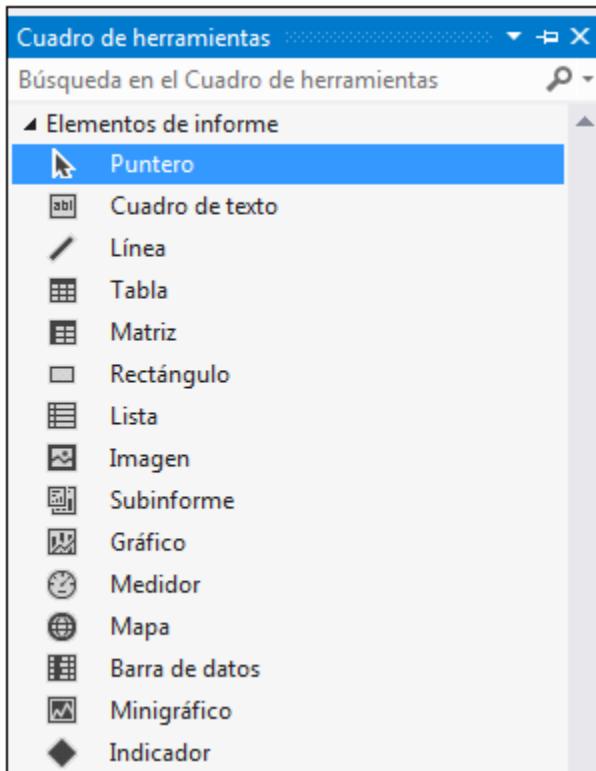
Agregamos un informe Reporting Services mediante Visual Studio, que sirve de herramienta de diseño:



En la sección DataSets, declaramos grupos de datos (de hecho, DataTable) que alimentarán las listas de campos. Estas listas contienen tantas filas como registros haya en DataSets.

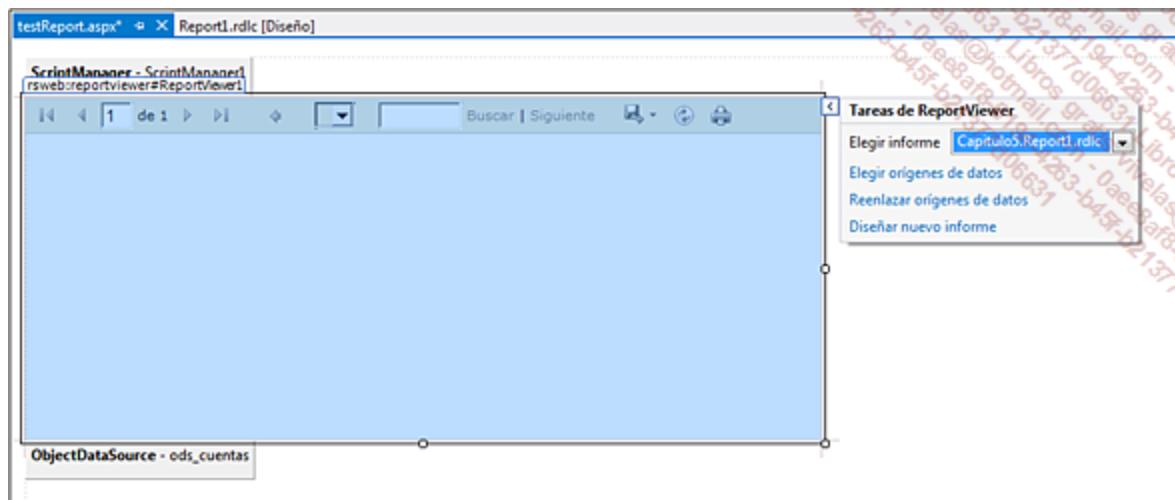


La caja de herramientas específica de los informes.rdlc proporciona elementos que pueden situarse sobre el informe para definir su lógica y su implementación. Entre los más interesantes se encuentran el texto - que sirve tanto de label como de campo - y la lista que itera sobre las filas de un DataSet.



## 2. El componente de visualización

Para visualizar el informe, se utiliza un componente **ReportViewer**. Es preciso agregar en la página un componente ScriptManager, así como un origen de datos de tipo ObjectDataSource encargado de recuperar los datos de los TableAdapters del DataSet.



He aquí el aspecto de la página completa, observe que no es necesario ningún código C# para que funcione:

```
<form id="form1" runat="server">  
  
<asp:ScriptManager ID="ScriptManager1" runat="server">  
  </asp:ScriptManager>
```

```

<rsweb:ReportViewer ID="report_cuentas" runat="server">
    <LocalReport ReportPath="App_Code\Report.rdlc">
        <DataSources>
            <rsweb:ReportDataSource DataSourceId="ods_cuentas" Name="cuentas" />
        </DataSources>
    </LocalReport>
</rsweb:ReportViewer>

<asp:ObjectDataSource ID="ods_cuentas" runat="server" DeleteMethod="Delete" SelectMethod="GetData" TypeName="BancaDataSetTableAdapters.CUENTATableAdapter" UpdateMethod="Update">
</asp:ObjectDataSource>
</form>

```

nº de cuenta	Titular	Saldo
1	Juan Miguel	100
2	Marcos	250
3	Terencio	4200
4	Víctor Hugo	3200
5	Luis Antonio	200

### 3. Parámetros

Nuestro segundo ejemplo es más ambicioso: los datos que se muestran en el informe están filtrados por el usuario. Por un lado, el usuario indica la cuenta de la que quiere ver las operaciones, por otro lado, transmite al informe .rdlc los límites del periodo para el que quiere extraer las operaciones.

Se muestra, a continuación, el código de la página; existen dos componentes ObjectDataSource que recuperan respectivamente la cuenta y sus operaciones. Están configurados mediante la lista desplegable cb\_cuenta, alimentada a su vez mediante un SqlDataSource:

```

<asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <table width="80%">
        <tr>
            <td>Mostrar operaciones para</td>
            <td><asp:DropDownList ID="cb_cuenta" runat="server"
AutoPostBack="True"
DataSourceID="ds_cuentas" DataTextField="TITULAR"
DataValueField="ID CUENTA"

OnSelectedIndexChanged="cb_cuenta_SelectedIndexChanged">
</asp:DropDownList></td>
            <td>&nbsp;</td>
            <td>Desde</td>
            <td>
                <asp:TextBox ID="txt_inicio" runat="server"
Width="70px">1/9/12</asp:TextBox>
            </td>
            <td>Hasta</td>
            <td>
                <asp:TextBox ID="txt_fin" runat="server"
Width="70px">30/9/12</asp:TextBox>
            </td>
        </tr>
    </table>
    <div style="height: 541px">
        <br />
        <rsweb:ReportViewer ID="ReportViewer1" runat="server"
Font-Names="Verdana"
Font-Size="8pt" WaitMessageFont-Names="Verdana"
WaitMessageFont-Size="14pt"

```

```

        Width="739px">
        <LocalReport ReportPath="App_Code\ReportOperaciones.rdlc">
            <DataSources>
                <rsweb:ReportDataSource DataSourceId="ods_cuenta"
Name="cuenta" />
                <rsweb:ReportDataSource DataSourceId="ods_operaciones"
Name="operacion" />

```

```

        </DataSources>
    </LocalReport>
</rsweb:ReportViewer>
<br />
<br />
<br />

<asp:SqlDataSource ID="ds_cuentas" runat="server"
    ConnectionString="<%$ ConnectionStrings:bancaConnectionString %>"
    SelectCommand="SELECT [titular], [id_cuenta] FROM [cuenta]">
</asp:SqlDataSource>
<asp:ObjectDataSource ID="ods_cuenta" runat="server"
    OldValuesParameterFormatString="original_{0}">
SelectMethod="GetDataByID_CUENTA"
    TypeName="BancaDataSetTableAdapters.CUENTATableAdapter">

    <SelectParameters>
        <asp:ControlParameter ControlID="cb_cuenta" Name="id_cuenta"
            PropertyName="SelectedValue" Type="Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
<asp:ObjectDataSource ID="ods_operaciones" runat="server"
    OldValuesParameterFormatString="original_{0}">
SelectMethod="GetDataByID_CUENTA"
    TypeName="BancaDataSetTableAdapters.OPERACIONTableAdapter">
    <SelectParameters>
        <asp:ControlParameter ControlID="cb_cuenta" Name="id_cuenta"
            PropertyName="SelectedValue" Type="Int32" />
    </SelectParameters>
</asp:ObjectDataSource>

```

```
</div>
```

En esta ocasión, hay algo de código C# para proveer los parámetros esperados por el informe:

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack && !IsCallback)
    {
        ReportParameter p1 = new ReportParameter("Inicio", txt_inicio.Text);
    }
}

```

```

ReportParameter p2 = new ReportParameter("Fin", txt_fin.Text);
ReportViewer1.LocalReport.SetParameters(new
ReportParameter[] { p1, p2 });
ReportViewer1.Visible = true;

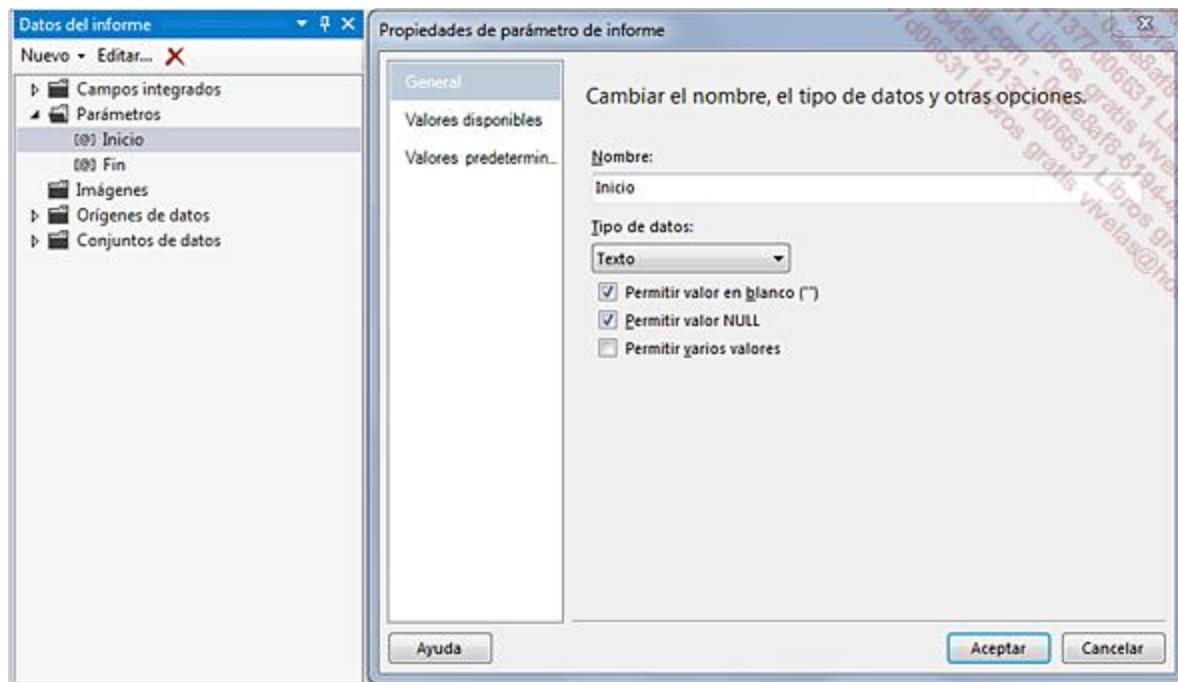
ReportViewer1.LocalReport.Refresh();
}

}

protected void cb_cuenta_SelectedIndexChanged(object sender, EventArgs e)
{
    ReportViewer1.LocalReport.Refresh();
}

```

En cuanto al informe, se definen dos parámetros mediante la sección **Parámetros**:



Podemos probar, a continuación, el informe completo:

Mostrar el reporte para Juan Miguel del 1/3/13 al 31/3/13

1 de 1 100% Buscar | Siguiente

Extracto de la cuenta Periodo del 1/3/13 al 31/3/13

nº de cuenta	Titular	Saldo
1	Juan Miguel	100,00 €

Operaciones

Fecha	Importe
01/03/2013	400,00 €
10/03/2013	200,00 €

# Los distintos medios para mantener el estado

El protocolo HTTP, al ser un protocolo desconectado, hace que el servidor Web y la aplicación no tengan ningún medio para "recordar" una consulta formulada por un agente. Los distintos procedimientos que sirven para superar esta dificultad se denominan técnicas de gestión del estado.

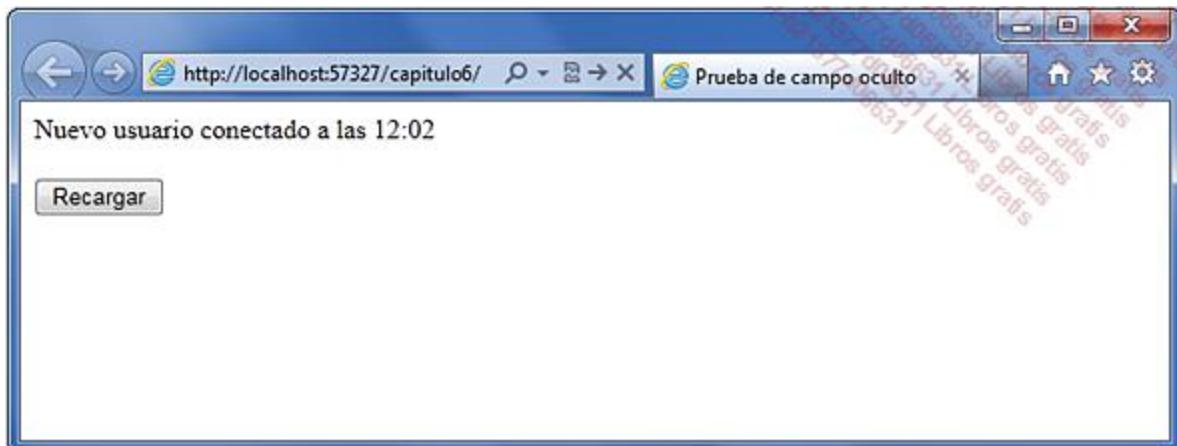
## 1. Campos ocultos

El lenguaje HTML dispone de campos ocultos. Estos campos se caracterizan por su nombre y su valor, de tipo texto. Esto se genera en tiempo de publicación de la página y se devuelve en el post del formulario.

```
<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">
    <title>Prueba de campo oculto</title>
</head>
<body>
<script runat="server">
    string valor;
</script>
<%
    if (Request["campo_oculto"] != null && Request["campo_oculto"] != "")
        valor = Request["campo_oculto"];
    else
    {
        valor = DateTime.Now.ToShortTimeString();
        Response.Write("Nuevo usuario conectado a las " + valor);
    }
%>
<form id="form1" runat="server">
    <input type="hidden" name="campo_oculto" value="<%= valor %>" />
    <input type="submit" name="b" value="Recargar" />
</form>
</body>
</html>
```

Tras la primera ejecución de la página, el objeto Request no se alimenta con una entrada llamada `campo_oculto`. El valor se inicializa con la hora en curso, antes de publicarse mediante el atributo `value` del campo oculto del formulario. Cuando el usuario hace clic en el botón **Recargar**, se asigna valor al campo y desaparece el mensaje para el usuario.



## 2. El ViewState

El diseño anterior no es útil para una página ASPX puesto que ya existe un mecanismo de persistencia llamado **ViewState**. El ViewState utiliza, a su vez, un campo oculto para recordar el valor de una colección entre dos postback. Hemos visto en el capítulo Los Web Forms que el ViewState lo emplean, normalmente, los controles de un formulario Web para conservar los valores de las propiedades usadas en la programación.

### a. Usar el ViewState en un Web Form

El ViewState sirve, también, para organizar datos de múltiples maneras, según un criterio de ordenación, por ejemplo. Esta información es de utilidad durante el tiempo de vida de la página, y el hecho de enlazar con otra página hace que se reinicialice el ViewState (salvo en el caso del postback cruzado).

El siguiente ejemplo ordena una lista de cuentas según un criterio definido en el ViewState:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        ViewState["ordenacion"] = "titular";
        mostrar();
    }
}

private void mostrar()
{
    string ordenacion;
    ordenacion = ViewState["ordenacion"] as string;
    string c = "SELECT titular, nombre_agencia FROM agencia INNER JOIN
cuenta ON agencia.id_agencia = cuenta.id_agencia GROUP BY titular,
nombre_agencia ORDER BY "
```

```

+ ordenacion;

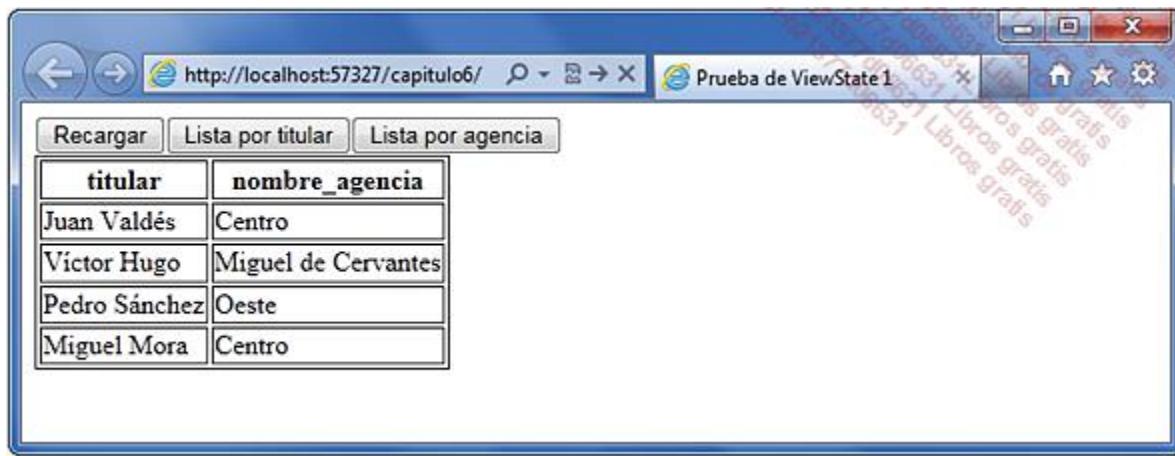
    SqlConnection cx = new SqlConnection(@"initial catalog=banca;
data source=.\SQL2005; integrated security=true");
    SqlCommand sql = new SqlCommand(c, cx);

    cx.Open();
    GridView1.DataSource = sql.ExecuteReader(CommandBehavior.
CloseConnection);
    GridView1.DataBind();
}

protected void cmd_ordenacion_titular_Click(object sender, EventArgs e)
{
    ViewState["ordenacion"] = "titular";
    mostrar();
}

protected void cmd_agencia_Click(object sender, EventArgs e)
{
    ViewState["ordenacion"] = "nombre_agencia";
    mostrar();
}

```



En este ejemplo, el ViewState sirve para almacenar el criterio de ordenación definido por el usuario. La ergonomía, basada en botones, habría sido delicada si el criterio de ordenación se hubiera manejado con una lista. El uso de la variable Session no es aconsejable, pues el criterio de ordenación de los registros no tiene sentido más que para esta página Web.

## b. Controlar la aplicación del ViewState

El diccionario ViewState se renderiza tras cada publicación de la página. El tiempo necesario para su codificación y su decodificación es despreciable, pero el volumen de datos que se inserta en el flujo HTML no lo es. Hemos visto en el capítulo Los Web Forms cómo desactivar el ViewState para ciertos controles o en la página entera.

De este modo, el programador no debería utilizar el ViewState para almacenar información que pudiera impactar en el rendimiento de las páginas HTML.

Tras la serialización de la colección, el flujo ViewState sufre tres transformaciones: se encripta (seguridad de los datos), una parte se convierte en código hash (suma de control), y se codifica en formato base 64 de modo que pueda insertarse en un campo de tipo texto como, por ejemplo, el campo oculto \_\_VIEWSTATE.

La operación de encriptación puede generar problemas cuando el sitio Web se explota en una granja de servidores. Las claves de codificación son específicas de cada máquina. En estas condiciones es preferible desactivar la encriptación, pues una representación por un servidor puede que se analice, posteriormente, en otro servidor distinto. Es por este motivo por el que existe la propiedad de página EnableViewStateMac, que, afortunadamente, tiene el valor por defecto False.

Si bien es posible inicializar esta propiedad mediante programación, conviene hacerlo mediante la directiva <%@ Page %>:

```
<%@ Page Language="C#" AutoEventWireup="true"  
EnableViewStateMac="true" %>
```

Algunos proxy HTTP limitan el tamaño del valor de los campos que se intercambian entre el agente y el servidor. Como el valor del campo ViewState alcanza rápidamente varios kilobytes, ASP.NET permite, para ello, fragmentarlo. Esta operación, llamada chunking en inglés, se realiza mediante el archivo machine.config:

```
<system.web>  
    <pages maxPageStateFieldLength="1024"/>  
</system.web>
```

## 3. Cadena de consulta (Query String) y URI

El protocolo HTTP se basa en el mecanismo de URI (*Uniform Resource Identifier*), fragmentos de URL. Podemos considerar que la URL `http://localhost/siteweb/page.aspx` da pie a la URI /siteweb/page.aspx.

Para conservar valores entre dos consultas HTTP, es posible utilizar la URI. Precisamente, HTTP define la cadena de consulta, situada tras el punto de interrogación que sigue al nombre del recurso:

```
http://localhost/siteweb/page.aspx?cadena_de_consulta
```

Si bien el programador tiene completa libertad en cuanto a la formación de la cadena de consulta, las herramientas implicadas en su decodificación consideran que está formada por segmentos clave=valor separados mediante el carácter &.

Se utiliza, generalmente, la cadena de consulta para:

- Parametrizar una página Web con ayuda de un identificador de registro (véase el capítulo El acceso a datos con ADO.NET 4.5, sección Presentación tabular de datos).
- Comunicar los valores de los campos del formulario cuando el atributo Method del formulario HTML es GET.
- Conservar un testigo de sesión o de autentificación.

El modelo de programación ASP.NET está lo suficientemente elaborado para no tener que recorrer la cadena de consulta más que en el primer caso: los datos se envían, por lo general, utilizando el método POST mientras que los testigos de sesión y de autentificación, si no se intercambian mediante cookies, están integrados en la URI (véase más adelante).

## 4. Las cookies

Las **cookies** son segmentos de datos (limitados a 4 KB) que emite el servidor Web. Cuando un agente HTTP recibe una cookie, la asocia a la cadena de consulta que se envía al servidor que la ha emitido.

Si la cookie se emite sin fecha límite, el navegador la conserva hasta el cierre de la sesión del usuario; la cookie se borra una vez se cierra la ventana del navegador.

Las cookies persistentes (dotadas de una fecha límite) se conservan en una carpeta del sistema operativo donde está instalado el navegador.

```
protected void Page_Load(object sender, EventArgs e)
{
    HttpCookie c = Request.Cookies["contador"];
    if (c == null)
    {
        c = new HttpCookie("contador");

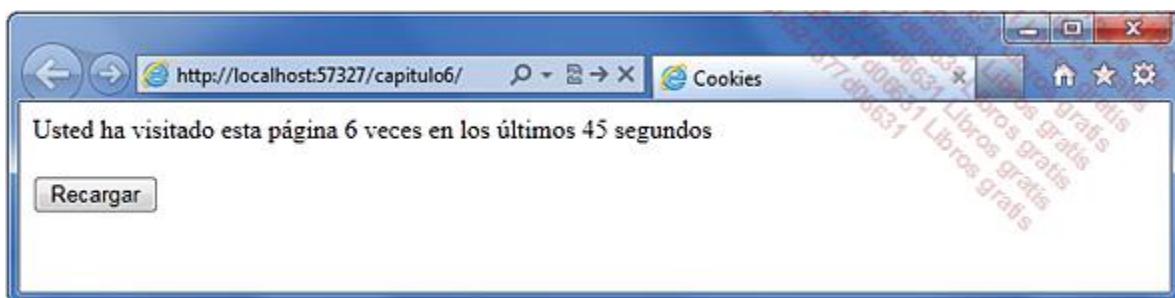
        // el valor de la cookie está limitado a 4 KB por
        // el protocolo HTTP
        c.Value = 1.ToString();
    }
    else
    {
        c.Value = (int.Parse(c.Value) + 1).ToString();
        c.Expires = DateTime.Now.AddSeconds(45);
    }
    Response.Cookies.Add(c);
    // fecha de expiración, no se trata, por tanto,
    // de una cookie de sesión
}
```

```

c.Expires = DateTime.Now.AddSeconds(45);

// uso de la cookie
Label1.Text = "Usted ha visitado esta página " + c.Value + " veces
en los últimos 45 segundos";
}

```



## Las sesiones

Es muy frecuente conservar para cada usuario datos de todo tipo: preferencias de presentación, criterios de ordenación, selección de artículos... Los períodos de tiempo en los que se mantienen estos datos en la aplicación se denominan sesiones.

### 1. Uso del objeto Session

Respecto a las técnicas anteriores de mantenimiento de estado, las sesiones apenas consumen ancho de banda; sólo se intercambia un testigo (formado por una cookie o una URI) que permite identificar al usuario. Los datos se conservan, de hecho, en el servidor de aplicaciones.

#### a. Memorización y búsqueda de un objeto

El objeto **Session** (propiedad de `HttpContext.Current`) se utiliza como una tabla hash: los objetos (o valores) memorizados están identificados mediante una clave que sirve de índice:

```

protected void cmd_redir_Click(object sender, EventArgs e)
{
    Session["nombre"] = txt_nombre.Text;
    Response.Redirect("session_page2.aspx");
}

```

Para encontrar un valor almacenado en la sesión desde otra página, conviene utilizar la misma clave y proceder a un tipado explícito:

```

protected void Page_Load(object sender, EventArgs e)
{
    string nombre = Session["nombre"] as string;
    Label1.Text = nombre;
}

```

```
}
```

Es, a su vez, necesario verificar que la entrada correspondiente a la clave existe, en caso contrario el indexador del objeto Session devuelve null y falla la conversión de tipo:

```
// la conversión de tipo peligra si no existe el dato idu de usuario  
int id_user = (int)Session["idu"];
```

El siguiente código es, de este modo, más seguro:

```
// verificación  
int id_user;  
if(Session["idu"] != null)  
    id_user = (int)Session["idu"];
```

### b. Inicialización del objeto Session

Como hemos mencionado en el capítulo Los sitios web ASP.NET 4.5, el servidor de aplicaciones ASP.NET produce los eventos globales, tales como Session\_Start() y Session\_End(). Estos eventos puede interceptarlos el programador definiendo el archivo Global.asax.

```
void Session_Start(object sender, EventArgs e)  
{  
    // este código se ejecuta cuando un usuario  
    // inicia una sesión  
    System.Data.DataSet ds = new System.Data.DataSet();  
    Session["seleccion"] = ds;  
}
```

```
void Session_End(object sender, EventArgs e)  
{  
    // este código se ejecuta cuando la sesión  
    // de un usuario falla (tras 20' de inactividad)  
    System.Data.DataSet ds = Session["seleccion"]  
    as System.Data.DataSet;  
}
```

### c. Securización del testigo de sesión

Si bien los datos que se mantienen en sesión no se comunican al navegador, los piratas pueden, en ocasiones, robar el testigo de sesión, especialmente si se transmite mediante una cookie.

Si el sitio se explota mediante el protocolo HTTPS (Secure HTTP, utilizando el subprotocolo SSL), el programador puede restringir la cookie de sesión a las consultas que se realizan desde el puerto seguro 443.

El siguiente código nos indica cómo proceder:

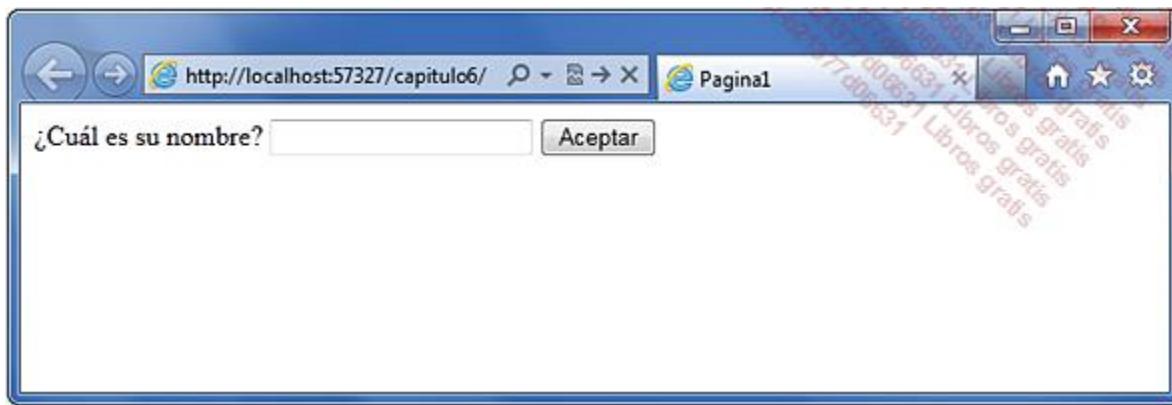
```
// indica que no se transmite el testigo de sesión  
// salvo con HTTPS  
  
Request.Cookies["ASP.NET_SessionId"].Secure = true;
```

## 2. Sesiones sin cookie y tiempo de abandono de sesión

### a. Sesiones sin cookie

Gracias al elemento <sessionState> del archivo de configuración Web.config, es posible configurar el módulo de sesión de forma que utilice la URL en lugar de las cookies para comunicar el testigo de sesión.

```
<sessionState cookieless="true" />
```



Esta regla es interesante cuando la política de seguridad prohíbe a los puestos cliente intercambiar cookies. Algunos proxy Web filtran, también, las cookies.

Como inconveniente, el programador no puede crear enlaces absolutos a los elementos de su sitio, puesto que el testigo de sesión se incluye en la URL en lugar del nombre de una subcarpeta ficticia.

### b. Timeout

En principio, las sesiones se abandonan tras 20 minutos de inactividad; si el testigo no se presenta tras este tiempo, el servidor de aplicaciones considera que hay que reiniciar la sesión, aunque se presente un "antiguo" testigo.

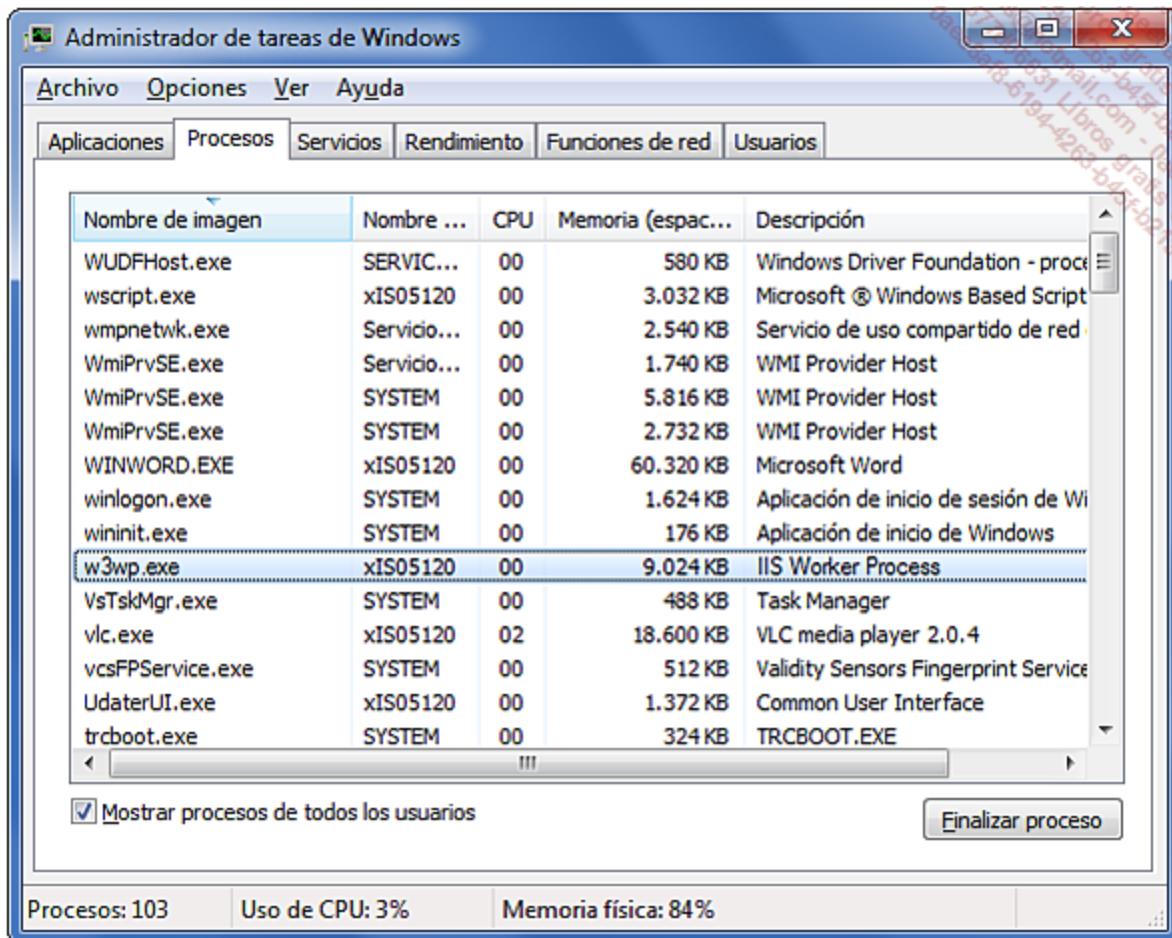
```
<sessionState timeout="40"/>
```

## 3. Servicios de conservación de datos en sesión

El servidor de aplicaciones ASP.NET utiliza distintos mecanismos para conservar los datos ubicados por el programador en sesión.

### a. El proceso en memoria InProc

El proceso ASP.NET w3wp.exe (Worker Process) se utiliza para almacenar datos de sesión. Tiene varias responsabilidades dentro del servidor de aplicaciones, y su ciclo de vida es bastante independiente del servicio IIS.



La activación de este modo de conservación se realiza mediante el archivo Web.config:

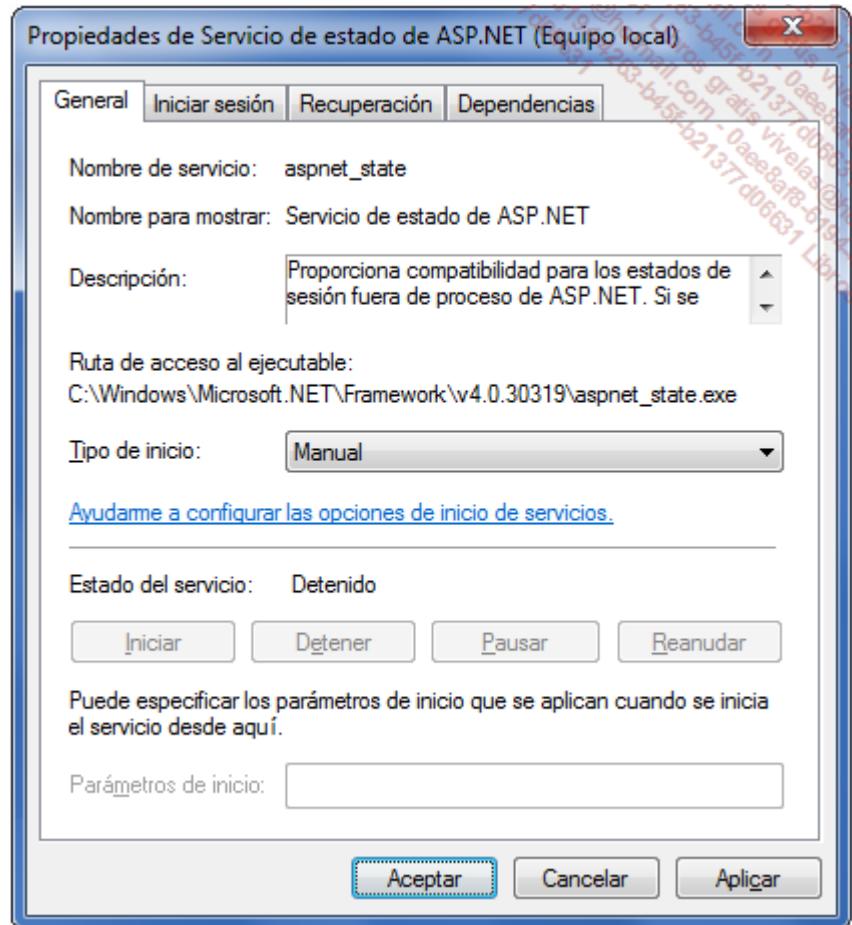
```
<sessionState mode="InProc"/>
```

Como no hay ninguna configuración específica para su uso, se trata del valor por defecto. El programador puede, no obstante, escoger otros modos cuando el reparto de carga entre varios servidores impide la compartición de sesión por el proceso de ejecución propio de cada máquina.

También, el proceso se ejecuta a menudo con un nivel de privilegios muy bajo, siendo muy vulnerable de cara a otros procesos. Dicho de otro modo, no es imposible que los datos de sesión se pierdan si el proceso se mata y se reinicia a continuación.

### b. El servicio Windows ASP.NET State Service

Tras la versión 2.0, Microsoft provee, a su vez, un servidor de conservación de estado basado en un servicio de Windows que debe iniciarse antes de su uso:



```
<sessionState mode="StateServer"/>
```

Más allá de la separación de responsabilidades de ejecución y de conservación de los datos en dos procesos diferentes, el servicio Windows es capaz de mutualizar los datos entre varios servidores. En este caso, el atributo `StateConnectionString` indica la dirección IP de la máquina donde se encuentran accesibles los datos para los distintos servidores:

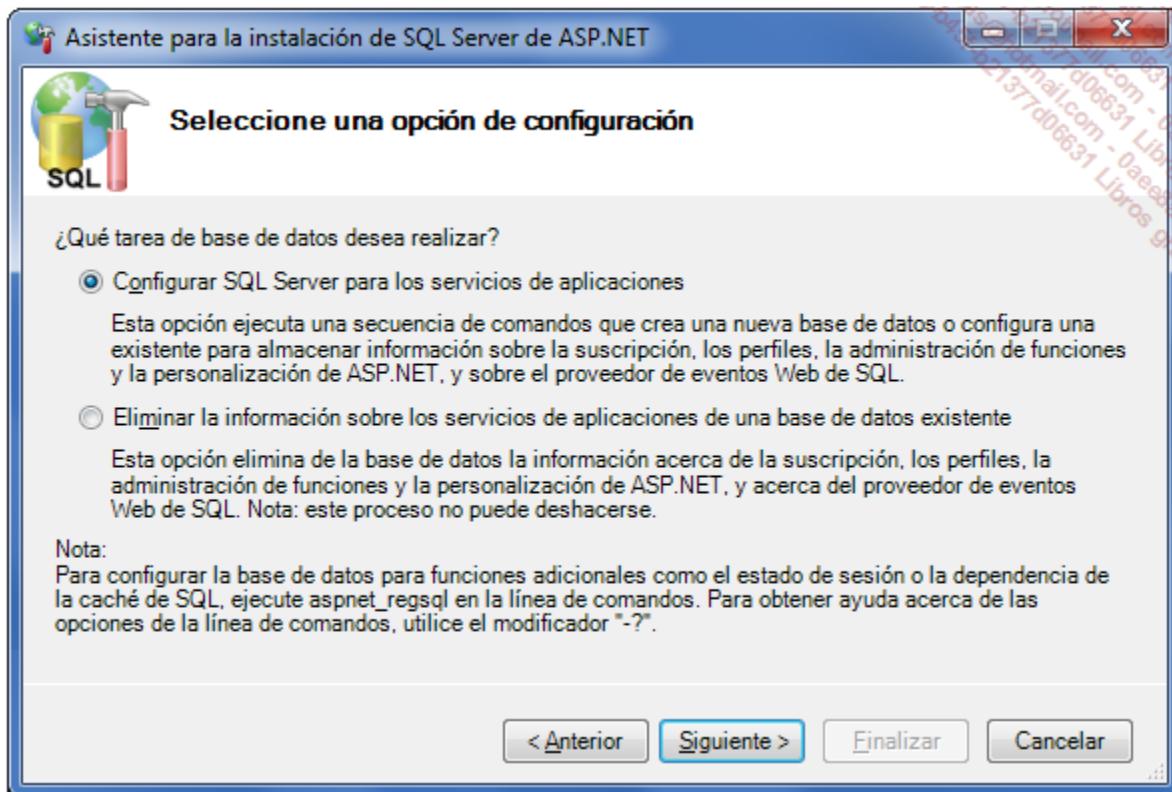
```
<sessionState mode="StateServer"  
stateConnectionString="tcpip=127.0.0.1:42424"/>
```

El valor 42424 se corresponde con el puerto por defecto que se utiliza para la comunicación entre el servidor de aplicaciones ASP.NET y el servicio Windows.

### c. El servicio SQL Server

El tercer modo de conservación es la base de datos SQL Server **ASPState** o **ASPNetdb**.

La base de datos ASPNetDB debe crearse mediante la herramienta `aspnet_regsql`. Ejecutada sin parámetros, abre un asistente que guía al programador en la creación de la base.



También es posible utilizar la herramienta `aspnet_Regsql` con las siguientes opciones:

```
aspnet_Regsql ssadd -sstype p
```

Para utilizar la base de datos ASPState, hay que ejecutar (con `sqlcmd` o `SQL Server Management Studio`) el script SQL `InstallSQLState.sql` ubicado en la carpeta `\windows\Microsoft.NET\Framework\v2.0`. Existe, también, un script de instalación `UninstallSQLState.sql`.

Antes de proceder a realizar pruebas, el programador debe pensar en:

- Adaptar los permisos del usuario ASP.NET para aprovechar la seguridad integrada.
- Ajustar, si es necesario, SQL Server para que acepte conexiones remotas (TCP/IP o túneles).

Una vez instalada la base de datos, la cadena de conexión del archivo `Web.config` indica cómo acceder a ella:

```
<sessionState mode="SQLServer">
    stateConnectionString="data source=.\SQL2005;initial
    catalog=aspnetstate;
    integrated security=true"/>
```

#### d. Servicios personalizados

Señalemos también que, con ASP.NET, el programador puede crear su propio servicio de persistencia de datos de sesión. Se trata, no obstante, de una tarea delicada, pero que podría

resolver problemas de compatibilidad con otros formatos de base de datos como Oracle o DB/2 para conservar los datos de sesión.

# Los objetos Application y Cache

## 1. El objeto Application

A diferencia del objeto Session, el objeto **Application** conserva los datos accesibles para todos los usuarios de un sitio Web.

### a. Uso

Como el objeto Session, Application se utiliza como un diccionario de términos identificados mediante claves.

```
protected void Page_Load(object sender, EventArgs e)
{
    int c=0;
    if (Application["contador"] != null)
        c = (int)Application["contador"];
    c++;
    Application["contador"] = c;
    Label1.Text = "Página visitada " + c + " veces";
}
```

Los datos se conservan tanto tiempo como permita el proceso de gestión del estado (aspnet\_wp.exe, el servicio Windows de gestión del estado o SQL Server). No es necesaria ninguna cookie.

El objeto Application memoriza cualquier objeto que se le pase como referencia. El programador debe, no obstante, prestar atención para mantener los campos estáticos que, generalmente, no están serializados.

### b. Bloqueo

Como varias páginas, solicitadas por distintos usuarios, pueden acceder simultáneamente al objeto Application, Microsoft proporciona un mecanismo de bloqueo.

```
Application.Lock();
Application["contador"] = c;

Application.UnLock();
```

Este mecanismo debe, por tanto, utilizarse con prudencia; el bloqueo afecta al conjunto de datos almacenados en el objeto Application. Como se trata de una sección crítica, el bloqueo creado bloquea los demás threads hasta que se libera. Esto entraña, a menudo, una degradación en el rendimiento. Es preciso, por tanto, asegurar el funcionamiento respecto al rendimiento general de la aplicación.

## 2. La caché de datos de aplicación Cache

El objeto **Cache** hace su aparición con ASP.NET 1.0. Igual que hace el objeto Application, conserva datos accesibles por todos los usuarios de un sitio Web. No obstante, estos datos se pueden borrar cuando existen dependencias asociativas.

La sintaxis de inserción es, por tanto, un poco más compleja, puesto que hace falta precisar las dependencias.

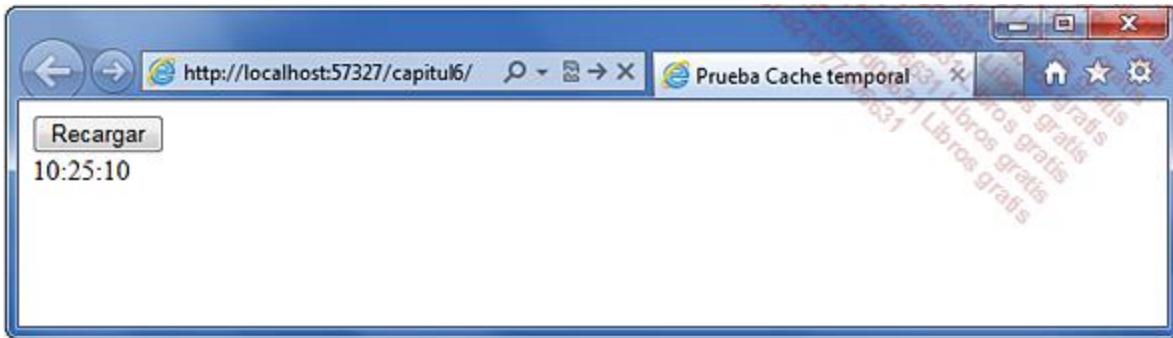
### a. Las dependencias temporales

El primer tipo de dependencia es el tiempo; el dato almacenado se conserva hasta una fecha (hora) absoluta.

```
protected void Page_Load(object sender, EventArgs e)
{
    string hora;
    if (Cache["hora"] == null)
    {
        hora = DateTime.Now.ToString();
        Cache.Add("hora", // clave
                  hora, // valor
                  null, // depende de archivo o de SQL
                  DateTime.Now.AddSeconds(30), // tiempo
                  TimeSpan.Zero, // tiempo relativo
                  CacheItemPriority.Default, // prioridad
                  null); // callback
    }
    else
        hora = (string)Cache["hora"];
    Label1.Text = hora;
}
```

El parámetro `TimeSpan.Zero` indica que la expiración es absoluta y no relativa. Como `TimeSpan` es una estructura, no es posible precisar `null` como valor del quinto parámetro. Es, a su vez, posible utilizar la constante `Cache.NoSlidingExpiration`.

Este ejemplo conserva una cadena, `hora`, que se renueva cada treinta segundos.



Otra estrategia de caché basada en el tiempo consiste en invalidar los datos tras un periodo de inactividad. En el siguiente ejemplo, cuando la página se recarga una vez, al menos, cada quince segundos, el dato con la hora se conserva.

```
protected void Page_Load(object sender, EventArgs e)
{
    string hora;
    if (Cache["hora"] == null)
    {
        hora = DateTime.Now.ToString();
        Cache.Add("hora", // clave
                  hora, // valor
                  null, // depende de archivo o de SQL
                  Cache.NoAbsoluteExpiration,
                  new TimeSpan(0,0,15),
                  CacheItemPriority.Default, // prioridad
                  null); // callback
    }
    else
        hora = (string)Cache["hora"];
    Label1.Text = hora;
}
```

## b. El callback

Almacenar un dato en caché permite asociar un procedimiento de recuperación (callback) cuando se borra el dato. El delegado **CacheItemRemoveCallback** admite como parámetro la clave y el valor asociado, así como una instancia de un tipo enumerado que indica el motivo de su eliminación. El tipo enumerado **CacheItemRemovedReason** presenta los valores DependencyChanged, Expired, Removed y Underused.

```
protected void Page_Load(object sender, EventArgs e)
{
    string hora;
```

```

if (Cache["hora"] == null)
{
    hora = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
    Cache.Add("hora", // clave
              hora, // valor
              null, // depende de archivo o de SQL
              Cache.NoAbsoluteExpiration,
              new TimeSpan(0, 0, 15),
              CacheItemPriority.Default, // prioridad
              new CacheItemRemovedCallback(cb_cache));
}
else
    hora = (string)Cache["hora"];

Label1.Text = hora;
}

private void cb_cache(string clave, object valor,
                      CacheItemRemovedReason motivo)
{
    // borrado de datos dependientes
}

```

El callback se invoca únicamente cuando se borra el dato, aunque con la próxima ejecución de la página, que es la que incluye el procedimiento. Podemos concluir que no se trata de un procedimiento de renovación puesto que la llamada puede haber tenido lugar, efectivamente, tras el borrado del dato. Por el contrario, es posible aprovechar la situación para borrar otros datos que se conservan y que eran dependientes.

### c. Dependencias de archivos

Los datos almacenados en la caché pueden depender de archivos cuyos cambios escruta el servidor de aplicaciones ASP.NET. Estos cambios invalidan los datos, que pasan automáticamente a renovarse.

El siguiente ejemplo utiliza este mecanismo para supervisar el archivo c:\temp\novedades.txt. Cuando este archivo se modifica, el dato se invalida y el texto que aparece en la caché se recarga desde el archivo:

```

protected void Page_Load(object sender, EventArgs e)
{
    string contenido;
    if (Cache["novedades"] == null)
    {

```

```

// lectura del archivo

string ruta = @"c:\temp\novedades.txt";

FileStream f;

f = new FileStream(ruta, FileMode.Open, FileAccess.Read);

TextReader reader = new StreamReader(f);

contenido = reader.ReadToEnd();

reader.Close();

f.Close();


// almacenamiento en caché

CacheDependency dep = new CacheDependency( ruta );

Cache.Add("novedades", contenido, dep, Cache.NoAbsoluteExpiration,
Cache.NoSlidingExpiration, CacheItemPriority.Default, null);

}

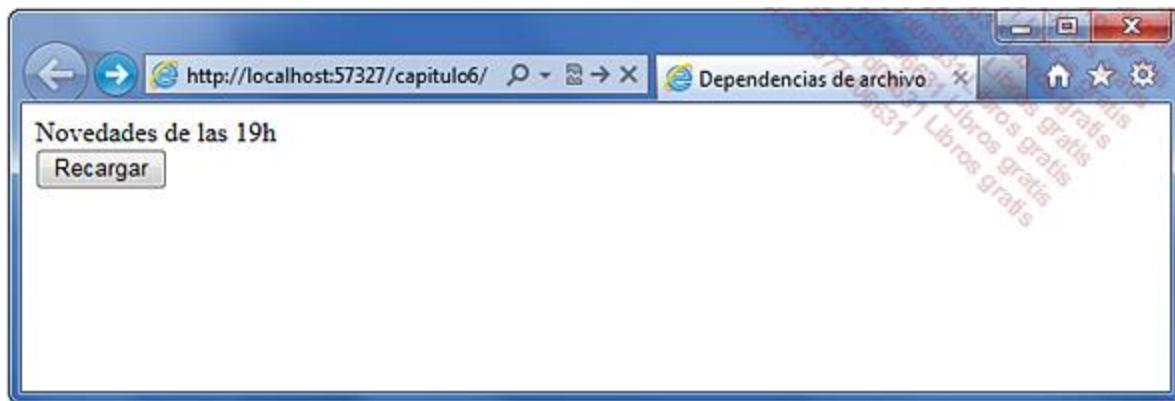
else

contenido = Cache["novedades"] as string;

Label1.Text = contenido;

}

```



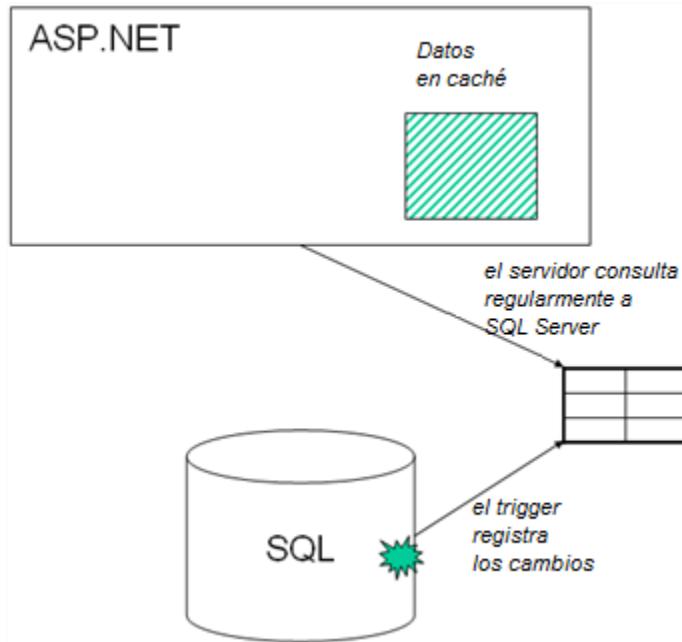
#### d. Dependencias SQL con SQL Server

ASP.NET provee un mecanismo de dependencia basado en los cambios aportados por datos SQL. Se trata de dependencias SQL Server. La implantación difiere ligeramente entre SQL Server 2000 y 2005-2012 puesto que esta última versión aprovecha un nuevo sistema de alertas llamado servicio de notificación.

### Preparar la base de datos

En la medida en que las dependencias SQL 2000 funcionan también con SQL Server 2005, presentamos aquí la puesta en marcha con los resultados de ambos entornos.

Con esta técnica, cada parte realiza la mitad del trabajo: la base de datos detecta los cambios con ayuda de un trigger, y el servidor de aplicaciones ASP.NET la interroga regularmente (polling) para determinar si es preciso invalidar los datos.



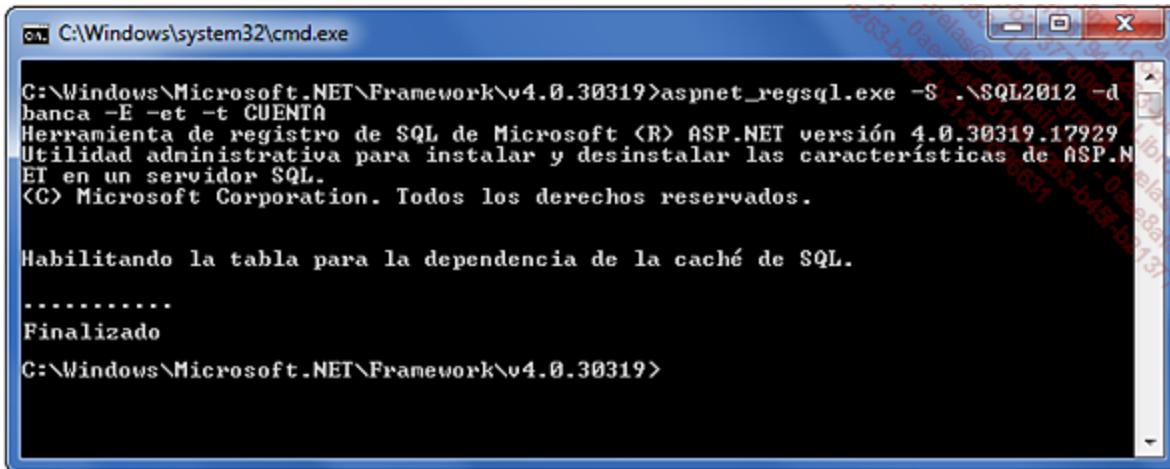
La base de datos que se quiere supervisar debe haberse preparado con la herramienta `aspnet_Regsql`:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319>aspnet_Regsql.exe -S .\SQL2012 -d banca -ed -E
Herramienta de registro de SQL de Microsoft (R) ASP.NET versión 4.0.30319.17929
Utilidad administrativa para instalar y desinstalar las características de ASP.N
ET en un servidor SQL.
(C) Microsoft Corporation. Todos los derechos reservados.

Habilitando la base de datos para la dependencia de la caché de SQL.
-----
Finalizado
C:\Windows\Microsoft.NET\Framework\v4.0.30319>
```

La opción `-ed` (enable dependency) permite tener en cuenta las dependencias SQL para la base de datos indicada, `banca`, en nuestro ejemplo. Esta operación crea una tabla `AspNet_SqlCacheTableForChangeNotification` que registrará cada evento de modificación aportado a una tabla que esté bajo supervisión.

La opción `-et` crea un trigger en la tabla de destino cuenta sobre la que se basa la consulta de cambio. Cualquier operación de tipo agregar, actualizar o suprimir se registrará en la tabla anteriormente creada.



```
C:\Windows\Microsoft.NET\Framework\v4.0.30319>aspnet_regsql.exe -S .\SQL2012 -d banca -E -et -t CUENTA
Herramienta de registro de SQL de Microsoft (R) ASP.NET versión 4.0.30319.17929
Utilidad administrativa para instalar y desinstalar las características de ASP.N
ET en un servidor SQL.
(C) Microsoft Corporation. Todos los derechos reservados.

Habilitando la tabla para la dependencia de la caché de SQL.
-----
Finalizado
C:\Windows\Microsoft.NET\Framework\v4.0.30319>
```

## Crear la dependencia en ASP.NET

El archivo Web.config debe completarse para que el servidor de aplicaciones ASP.NET pueda buscar las señales de modificación en la base de datos asociada:

```
<connectionStrings>
  <add name="banca"
    providerName="System.Data.SqlClient"
    connectionString="data source=.\SQL2005; initial catalog=banca;
    integrated security=true"/>

</connectionStrings>
<system.web>
  <caching>
    <sqlCacheDependency enabled="true">
      <databases>
        <add name="banca"
          connectionStringName="banca" pollTime="500"/>
      </databases>
    </sqlCacheDependency>
  </caching>
</system.web>
```

El retardo indicado (pollTime) significa que ASP.NET verificará cada 500 milisegundos si hay disponible alguna modificación.

La página Web que utilice los registros no tendrá más que crear una dependencia SQL. En nuestro ejemplo, es la que inicia el proceso de escucha. Una vez realizadas las comprobaciones, la instrucción `Start()` debería desplazarse al método `Application_Start()` del archivo `Global.asax`.

```

void Page_Load(object sender, EventArgs e)
{
    SqlConnection cx = new SqlConnection();
    cx.ConnectionString = ConfigurationManager.ConnectionStrings["banca"].
ConnectionString;

    if (!IsPostBack)
    {
        // El siguiente código puede figurar también en Application_Start
        // aunque para realizar pruebas es más práctico
        // integrarlo en la página

        // arranca el listener de notificaciones SQL
        SqlDependency.Start(cx.ConnectionString);

        // para las pruebas, asegura que el dato
        // no está presente en la cache
        if(Cache["ds"]!=null)
            Cache.Remove("ds");
    }

    Label1.Text = "";
    DataSet ds;
    ds = (DataSet)Cache["ds"];
    if (ds == null)
    {
        SqlCommand cmd = new SqlCommand("select * from cuenta", cx);
        SqlDataAdapter da = new SqlDataAdapter(cmd);

```

```

        // preste atención al orden de ejecución de las instrucciones
        ds = new DataSet();
        da.Fill(ds, "cuenta");

        // base de datos y tabla bajo supervisión
        SqlCacheDependency dep = new SqlCacheDependency("banca","cuenta");

        Label1.Text = "Relectura de la tabla: " + dep.HasChanged + " " +
dep.UtcLastModified;

```

```

        Cache.Add("ds", ds,
        dep,
        Cache.NoAbsoluteExpiration,
        Cache.NoSlidingExpiration,
        CacheItemPriority.Normal, null);
    }

    GridView1.DataSource = ds;
    GridView1.DataBind();

}

```

## **Mejoras aportadas por la versión SQL 2005 (también funcionan en 2008 y en 2012)**

El nuevo sistema de notificación SQL 2005 simplifica la implementación de dependencias SQL. Ya no es necesario preparar la base de datos con `aspnet_Regsql` ni activar el polling.

Además, las notificaciones son más precisas y más eficaces, puesto que pueden apuntar a una fila en lugar de a una tabla, y no generar señales sucesivas para modificaciones consecutivas sobre el mismo dato.

El servicio de notificación debe instalarlo el administrador de SQL 2005. No forma parte de las opciones por defecto.

Para ilustrar el funcionamiento de este servicio vamos a aplicarlo a un control `SQLDataSource`:

```

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:bancaConnectionString %>" 
EnableCaching="True"
OnSelecting="SqlDataSource1_Selecting"
SelectCommand="SELECT id_agencia,nombre_agencia FROM dbo.agencia"
SqlCacheDependency="CommandNotification"> </asp:SqlDataSource>

```

El atributo `SqlCacheDependency`, asignado por `CommandNotification` en lugar de la sintaxis `base:table`, indica al servidor de aplicaciones que debe crear una dependencia automática ligada a una notificación SQL Server 2005 en lugar de una dependencia de tipo SQL Server 2000.

El siguiente código arranca el listener de las notificaciones y muestra un mensaje cada vez que se modifican los datos de la tabla agencia:

```

protected void Page_Load(object sender, EventArgs e)

```

```

{
    if (!IsPostBack)
    {
        // arranca el listener de las notificaciones
        string cs = ConfigurationManager.ConnectionStrings
        ["bancaConnectionString"].ConnectionString;
        System.Data.SqlClient.SqlDependency.Start(cs);
    }
}

protected void SqlDataSource1_Selecting(object sender,
SqlDataSourceSelectingEventArgs e)
{
    Label1.Text = "Recarga de datos a las "
+ DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss");
}

```

Es, a su vez, posible obtener el mismo resultado sin utilizar un control integrado `SqlDataSource`. El siguiente código permite implementar un `ObjectDataSource`:

```

protected void Page_Load(object sender, EventArgs e)
{
    DataSet ds;
    if (Cache["ds2"] == null)
    {
        string cs = ConfigurationManager.ConnectionStrings
        ["bancaConnectionString"].ConnectionString;
        SqlConnection cx = new SqlConnection(cs);
        SqlDataAdapter da = new SqlDataAdapter("select id_agencia,
nombre_agencia from dbo.agencia", cx);

        // activa la recepción de notificaciones
        da.SelectCommand.NotificationAutoEnlist = true;
        System.Web.Caching.SqlCacheDependency dep=dep =
new SqlCacheDependency
(da.SelectCommand);

        ds = new DataSet();
        da.Fill(ds);
    }
}

```

```

        Cache.Insert("ds2", ds, dep);

        Label1.Text = "Recarga de datos a las "
+ DateTime.Now.ToString();
    }

    else

        ds = (DataSet)Cache["ds2"];

        GridView1.DataSource = ds;
        GridView1.DataBind();
    }
}

```

Para funcionar, el código debe, obligatoriamente, activar el listener de notificaciones emitidas por el servidor SQL. Es el objetivo de la siguiente instrucción:

```

// activa la recepción de notificaciones
da.SelectCommand.NotificationAutoEnlist = true;

```

Como con los demás sistemas de dependencia, la supervisión (escucha) se inicia en el archivoGlobal.asax.

```

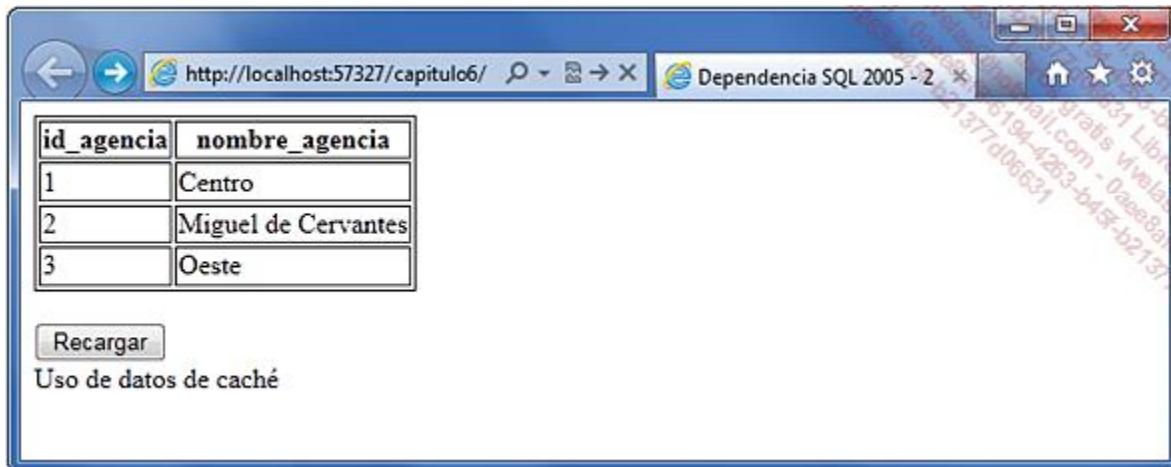
void Application_Start(object sender, EventArgs e)
{
}

```

```

// activa la recepción de notificaciones
string cs = ConfigurationManager.ConnectionStrings
["bancaConnectionString"].ConnectionString;
System.Data.SqlClient.SqlDependency.Start(cs);
}

```

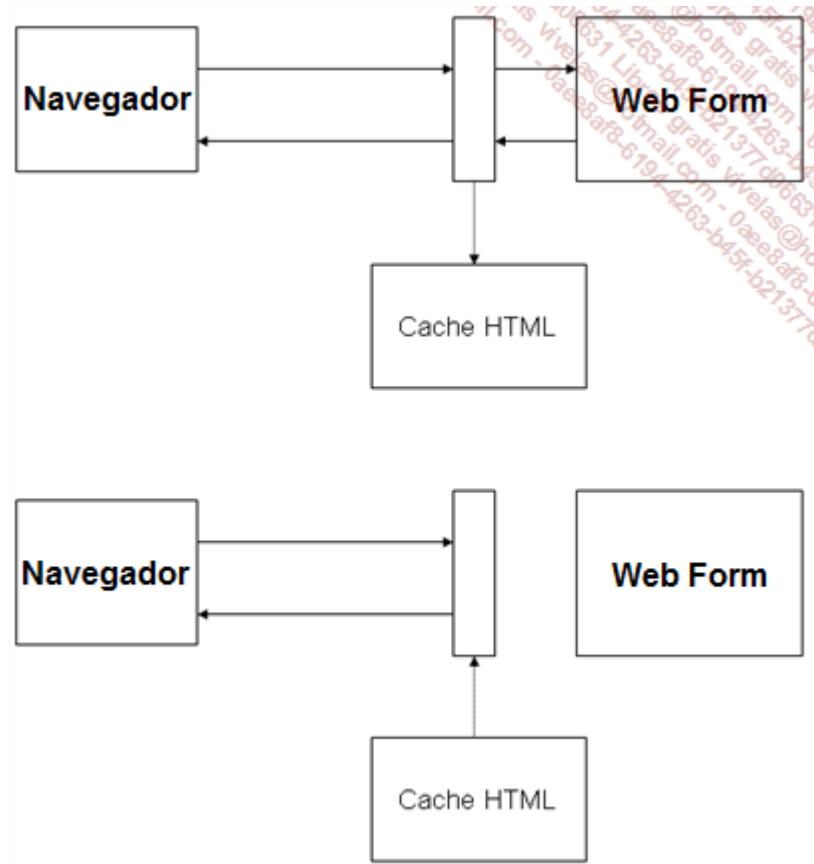


### 3. La caché HTML

Siempre tratando de mejorar el rendimiento, la tecnología ASP.NET proporciona una caché de salida HTML. Se trata, normalmente, de una prerrogativa del servidor Web IIS, aunque su consideración por el servidor de aplicaciones permite una gestión muy fina.

#### a. Caché de salida

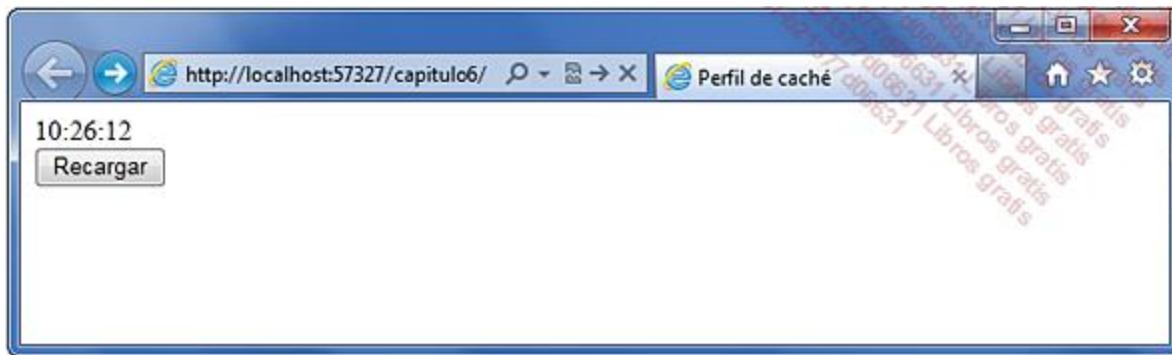
La caché de salida es un dispositivo que trabaja en dos tiempos. Por un lado, toma una fotografía instantánea del flujo de salida HTML. A continuación, cortocircuita el proceso de procesamiento de una consulta y sustituye su flujo por la copia que se conserva en caché.



Para ilustrar el funcionamiento de caché de salida HTML, creamos un Web Form que contiene una tabla y un botón. En el método Page\_Load(), mostramos la hora en curso mediante un label. En principio, cada clic en el botón provoca un postback y, en consecuencia, el refresco del label. Pero la directiva <%@ OutputCache %> puede modificar este comportamiento:

```
<%@ OutputCache Duration="15" VaryByParam="none" %>
```

Tras el primer acceso, la salida HTML se almacena en caché durante 15 segundos. El atributo `VaryByParam`, que debe estar presente, sirve para controlar la caché según algún parámetro que figura en la cadena de consulta Query String.



Es, también, posible controlar la caché mediante un encabezado HTTP, un control, o una dependencia SQL:



### b. Fragmentos de páginas en caché

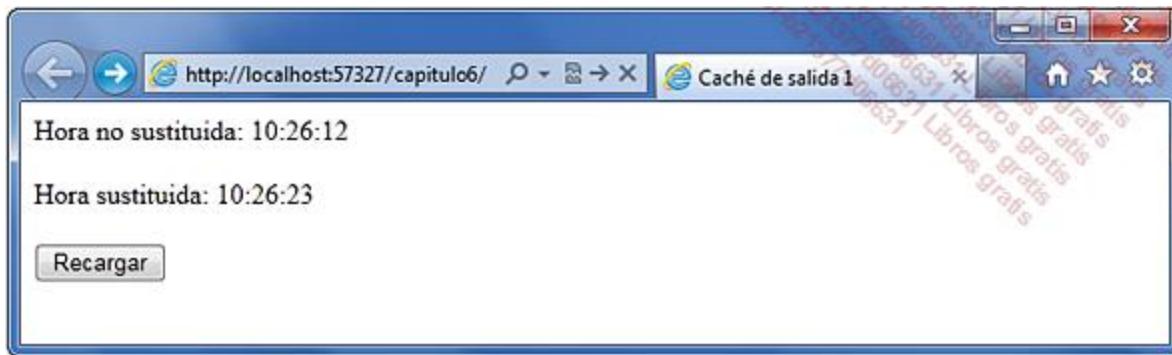
El programador tiene la posibilidad de alojar en caché solamente una parte de la página HTML; conviene aplicar la directiva `<%@ OutputCache %>` a un control .ascx en lugar de a la página que lo contiene. Las reglas de funcionamiento son idénticas.

### c. Sustituciones

El control **Substitution** resulta útil cuando una parte de la página debe escapar a la caché. Posee una propiedad **MethodName** que designa al método estático que se invoca aunque esté activa la caché:

```
protected static string get_hora(HttpContext ctx)
{
    return DateTime.Now.ToString("yyyy-MM-dd");
}
```

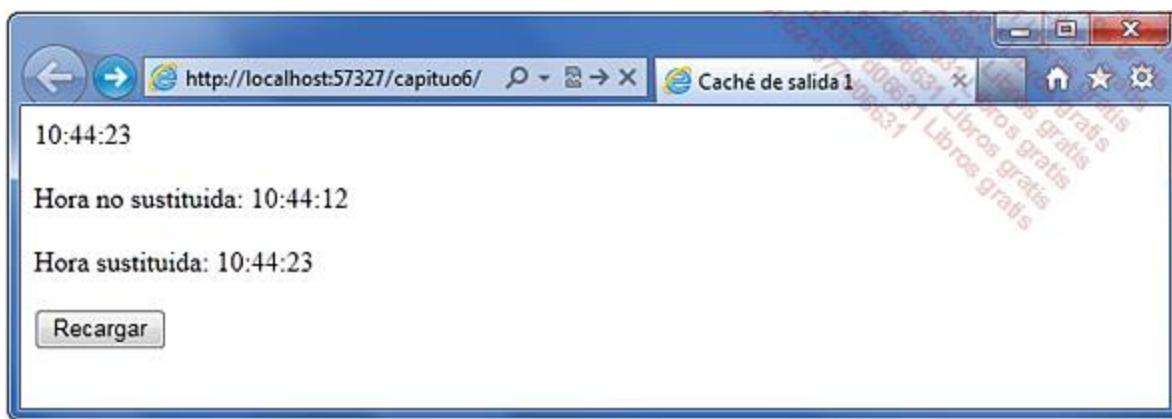
La siguiente captura de pantalla muestra que el control sustituido representa una hora posterior al label cuyo contenido HTML está alojado en caché:



Para anular los efectos de la caché es, también, posible utilizar el método **WriteSubstitution**:

```
// otra forma de anular la caché  
  
Response.WriteSubstitution(new HttpResponseSubstitutionCallback(get_hora));
```

Esta última forma presenta, no obstante, un inconveniente: es difícil controlar la posición del texto de reemplazo en el flujo de salida.



#### d. Perfiles de caché

Cuando una estrategia de caché afecta a varias páginas, el programador puede crear perfiles en el archivo Web.config:

```
<system.web>  
  <caching>  
    <outputCacheSettings>  
      <outputCacheProfiles>  
        <add name="perfil1" duration="15" varyByParam="none"/>  
      </outputCacheProfiles>  
    </outputCacheSettings>  
  </caching>  
</system.web>
```

Los perfiles se referencian en las páginas Web que los requieren:

```
<%@ OutputCache CacheProfile="perfill" %>
```

# Securización de los sitios ASP.NET

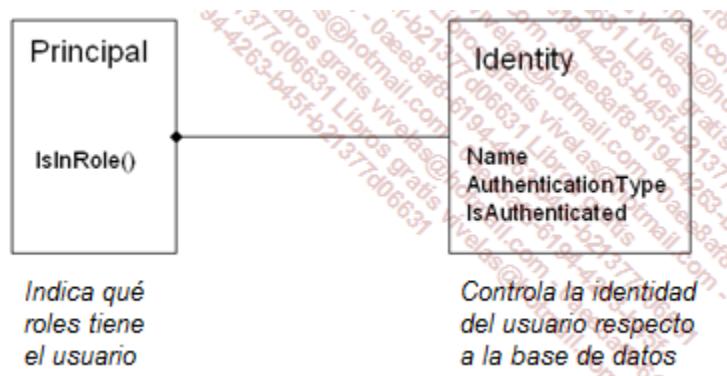
Los sitios Web exponen sus páginas a una gran cantidad de usuarios. Estén preparados para Internet o Intranet, es necesario definir reglas de accesibilidad a los distintos elementos del sitio.

## 1. Modelo de securización del sitio

La securización de un sitio se basa en los mismos principios que cualquier aplicación .NET. Comparte, además, la tipología de objetos asociada a la securización, basada en los roles (**RBS**, *Role Based Security*) de las aplicaciones Windows.

### a. Objetos de seguridad

Todo usuario de una aplicación .NET representa una entidad (**Principal**, en inglés). Su identidad(**Identity**) se basa en su reconocimiento dentro de una base de datos estándar de usuarios (Windows, Active Directory, AspNetDb) o personalizada.



El siguiente extracto de código muestra información conocida acerca del usuario:

```
TextBox1.Text =  
string.Format("Nombre del usuario: {0}\nBase de datos:  
{1}\nAutentificado:{2}\nTipo{3}",  
User.Identity.Name,  
User.Identity.AuthenticationType,  
User.Identity.IsAuthenticated,  
User.GetType().FullName);
```

Aunque, en ausencia de una identificación precisa, estos campos no se informan correctamente. Será, entonces, necesario modificar el archivo Web.config para exigir una identificación:

```
<authentication mode="Windows"/>  
<authorization>  
    <!-- prohibido desconocidos (anónimos) -->
```

```
    <deny users="?" />  
</authorization>
```

## b. Autentificación

La autentificación (**authentication**, en inglés) consiste en verificar las credenciales (**credentials**) de un usuario en una lista. ASP.NET tiene acceso a varias bases de datos, tales como Windows, Active Directory (o ADAM para las aplicaciones), AspNetDb. También puede definir sus propias bases de datos.

A cada repositorio de datos puede corresponderle una técnica de verificación. A menudo, se trata de un par (usuario, contraseña), aunque es posible aplicar otras técnicas. Desde el momento en que se identifica un usuario, las credenciales asociadas a su autentificación serán consistentes con independencia del repositorio de datos; el sistema no confiará más en un usuario identificado mediante NTLM, seguridad integrada, que en uno identificado mediante el esquema básico.

Un sitio Web ASP.NET escoge un modo de autentificación Windows, Forms, o Passport. Según este modelo, hay uno o varios repositorios de datos disponibles.

En el caso del **modo Windows**, es IIS el encargado de identificar al usuario. Las bases de datos disponibles incluyen el repositorio de cuentas de usuario de Windows y Active Directory. Desde un punto de vista técnico, utilizaremos en este caso un **WindowsPrincipal** (User) y un **WindowsIdentity** (User.Identity).

Active Directory es el servicio de directorio de Windows. Está integrado con Windows Server 2003. En XP, utilizamos ADAM, una adaptación de la aplicación (Active Directory in Application Mode).

En el caso del **modo Forms**, una página de conexión, llamada a menudo login.aspx en los ejemplos, reconoce al usuario. Se basa en una base de datos propietaria o sobre la nueva **APIMembership** (que puede enlazarse con la base de datos AspNetDb). En este modo, el objeto User se convierte en un **GenericPrincipal** y el tipo del objeto User.Identity se convierte en un **FormsIdentity**.

Microsoft proporciona, a su vez, el **modo Passport** que funciona con ayuda de un servicio Web público de pago.

El archivo de configuración Web.config precisa qué modo se aplica a un sitio Web. La sintaxis para cada modo se detalla en los párrafos a continuación.

## c. Autorización

La autorización permite, o deniega, explícitamente el acceso a ciertos recursos (URL) a los usuarios del sitio. La autorización se define en el archivo Web.config que se lee de arriba a abajo. Cuando se encuentra una correspondencia para el usuario en curso, el sistema detiene la lectura del archivo y declara al usuario apto (**allow**) o no (**deny**) para acceder al recurso.

La autorización puede definirse para el conjunto de páginas de un sitio gracias al elemento **<authorization>**:

```
<authorization>
    <!-- prohíbe el acceso a desconocidos (anónimos) -->
    <deny users="?" />
</authorization>
```

Para asignar autorizaciones específicas a una URL, es preciso crear un nuevo elemento directamente bajo el súperelemento **<configuration>**:

```
<location path="compras.aspx">
  <system.web>
    <authorization>
      <allow roles="compradores"/>
      <deny users="*"/>
    </authorization>
  </system.web>
</location>
```

El lector verá que se comienza abriendo la página `compras.aspx` para los miembros del grupo compradores y, a continuación, se deniega el acceso a los demás. Si procediéramos de forma opuesta, el sistema reconocería a todos los usuarios de la primera regla y no tendría en cuenta la segunda, de modo que ningún usuario tendría acceso a la página `compras.aspx`.

## 2. Securización en modo Windows

Como con cualquier sistema operativo, Windows tiene la responsabilidad de identificar a los usuarios y permitirles el acceso a las aplicaciones. Sabiendo que el servidor Web IIS funciona bajo este sistema operativo, parece oportuno aprovechar su conocimiento en este dominio para apoyarse en el repositorio de cuentas de usuario integrado en el OS.

### a. Activación del modo de autentificación

El elemento **<authentication>** del archivo Web.config aplica este modo de autentificación. No es necesario ningún otro parámetro:

```
<authentication mode="Windows"/>
```

### b. Configuración de IIS

Desde la consola de administración de IIS, se puede escoger el esquema de autentificación. El término esquema se traduce por protocolo. Se trata de una convención que se intercambia entre el servidor Web y el navegador para determinar las reglas de identificación del usuario.

La consola de administración IIS se encuentra en el panel de control. Para acceder al sitio Web en el que se desea incluir las reglas, hay que desplegar el árbol **Sitio Web por defecto**, seleccionar el sitio Web afectado y, a continuación, la opción **Authentication**.

Para la autentificación efectiva de los usuarios, los esquemas que están soportados por IIS son los siguientes:

Basic	El nombre de usuario y la contraseña se comunican sin cifrar (base64) entre el navegador y el servidor. Es el esquema estándar HTTP disponible en todos los navegadores.	Seguridad débil
Basic con SSL	Similar al Basic, pero el protocolo SSL encripta la información.	Seguridad intermedia

Digest	Se requiere un plug-in para Internet Explorer. Está asociado a Active Directory, sólo se intercambia un testigo de autenticación.	Seguridad intermedia	Por defecto
Integrada o NTLM	Se requiere un plug-in estándar para IE y Firefox. Sólo se comunica el testigo de autenticación Windows. Es incompatible con el uso de un cortafuegos.	Seguridad elevada	

defecto, la seguridad integrada es la única activa. En este esquema, el sistema cliente Windows ya ha autenticado al usuario. El navegador obtiene, a continuación, la identidad del usuario desde IIS.

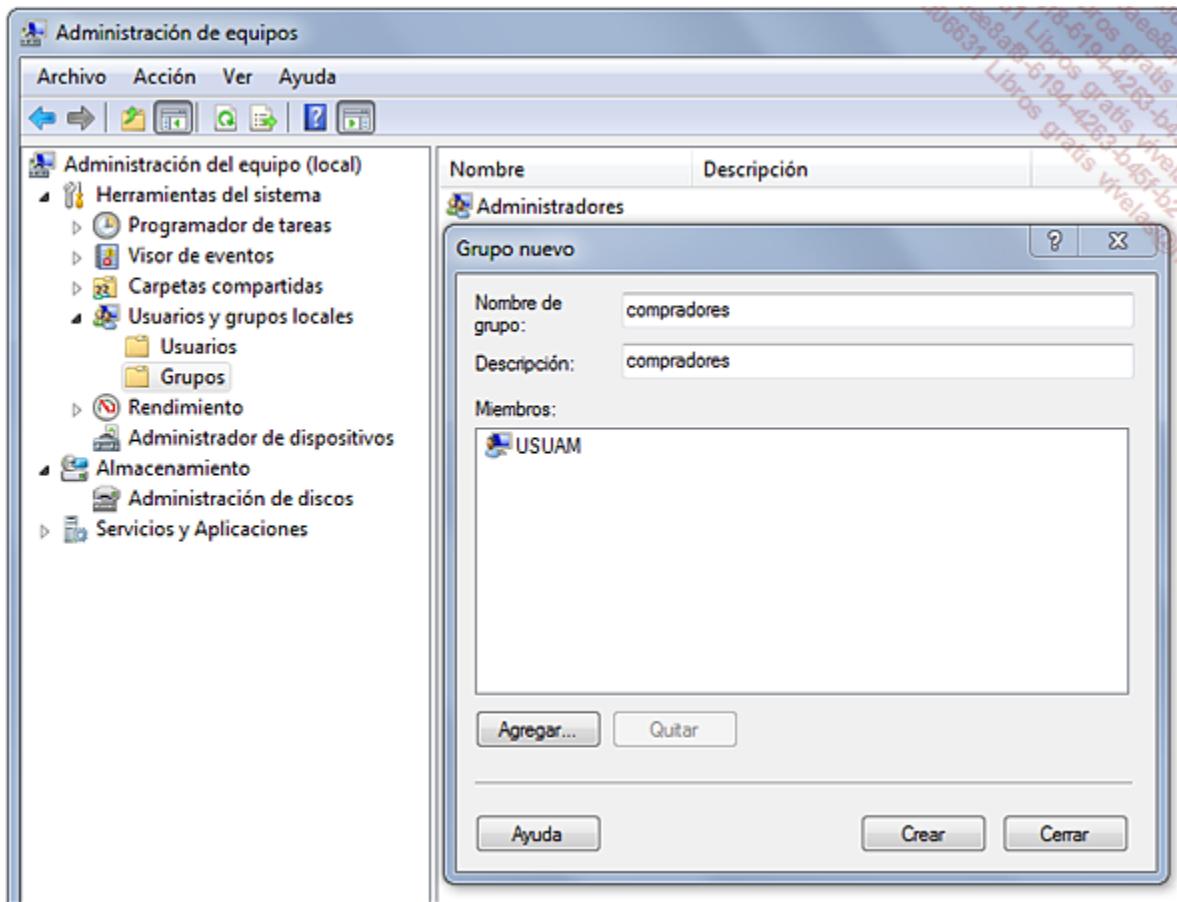
### c. Autorización

Es aconsejable configurar usuarios y grupos de Windows para realizar pruebas. Con Windows 7, estas operaciones se realizan desde la consola **Administración de equipos**, accesible desde la carpeta **Herramientas administrativas** del panel de control o del menú contextual **Administrar el equipo** aplicada al puesto de trabajo.

Tras la creación del grupo **Compradores** y una vez designados sus miembros, es necesario reiniciar el equipo para poder realizar las pruebas.

La primera prueba que vamos a realizar consiste en verificar que el usuario se ha creado correctamente en el grupo **Compradores**. Esto puede hacerse mediante programación:

```
TextBox1.Text =
string.Format("Nombre de usuario: {0}\nBaseDeDatos: {1}\nAutentificado?
{2}\nTipo: {3}\nMiembro de compradores? {4}",
User.Identity.Name,
User.Identity.AuthenticationType,
User.Identity.IsAuthenticated,
User.GetType().FullName,
User.IsInRole(@"Compradores")); // o bien BRICE-ARNAUD\Comprador
```



Podemos, a continuación, probar la eficacia de la protección en la página `compras.aspx`. En la captura de pantalla siguiente, el usuario `db2admin` no es miembro del grupo **Compradores**. IIS (de hecho HTTP) le ofrece la posibilidad de identificarse de nuevo. En caso de persistencia, el navegador muestra la página de error correspondiente al estado HTTP Not Authorized (códigos 401 o 403).

En modo Windows, los nombres de los roles se corresponden con grupos de Windows. Éstos pueden prefijarse por el dominio al que pertenecen o por `BUILTIN` en el caso de roles integrados.

### 3. Securización en modo Forms

Desde su introducción con ASP.NET 1.0, la securización en modo Forms ha sufrido cambios y mejoras sustanciales. Es, por fin, compatible con los navegadores que prohíben el uso de cookies y puede asociarse con la base de datos de usuarios de Windows mediante Active Directory.

Respecto al modo Windows, el programador construye una página de conexión que se encarga de la representación gráfica del sitio.

#### a. Activación del modo Forms y creación de una página de conexión

El modo Forms se activa en el archivo `Web.config` mediante el atributo `mode`, que pertenece al elemento `<authentication>`:

```
<authentication mode="Forms">
    <forms loginUrl="conexion.aspx"/>
```

```
</authentication>
```

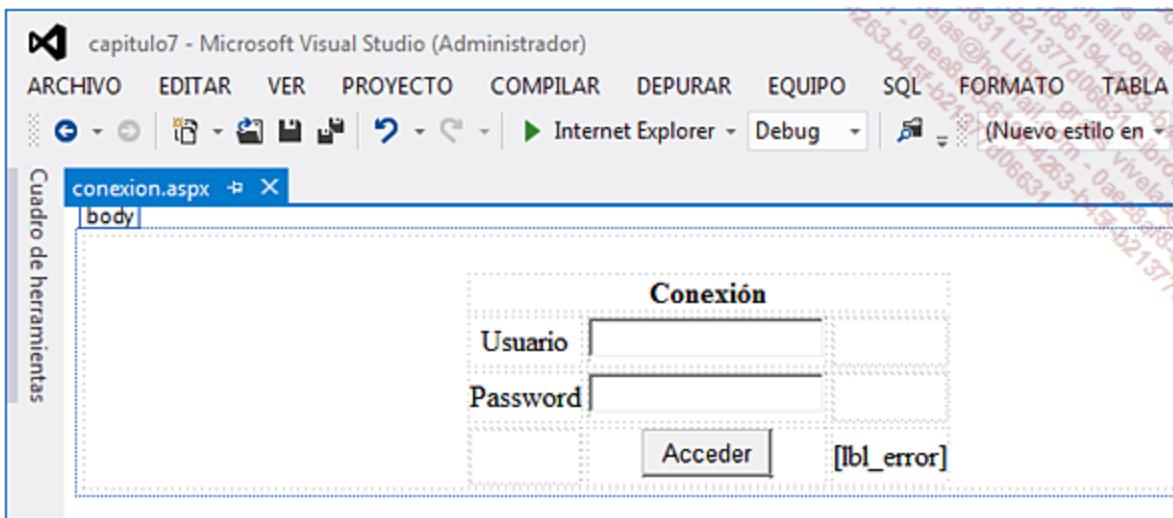
El subelemento **<form>** posee, al menos, el atributo **loginUrl** que designa una página responsable de la identificación de usuarios. Hay otros atributos disponibles según se quiera crear un testigo de autenticación persistente (lo cual es poco habitual) o compartir el testigo entre varias aplicaciones.

Como con el modo Windows, bloqueamos el acceso a los usuarios anónimos con el objetivo de probar la autenticación Forms.

```
<authorization>
  <deny users="?" />
</authorization>
```

Así configurado, el servidor de aplicaciones desviará a todo usuario que no esté autenticado hacia la página de conexión o le bloqueará el acceso hasta que provea credenciales válidas. A continuación, se le redirigirá a la página solicitada inicialmente.

La página de conexión es muy sencilla de implementar. Basta con crear una pareja de campos TextBox, un botón y una etiqueta label que permita mostrar un mensaje de error:



Su programación no presenta ninguna dificultad. Es preciso importar el espacio de nombres **System.Web.Security** y, a continuación, validar a los usuarios según la base de datos apropiada. En esta primera prueba, nos conectaremos con una tabla de hash. Esta técnica no está, evidentemente, recomendada en producción.

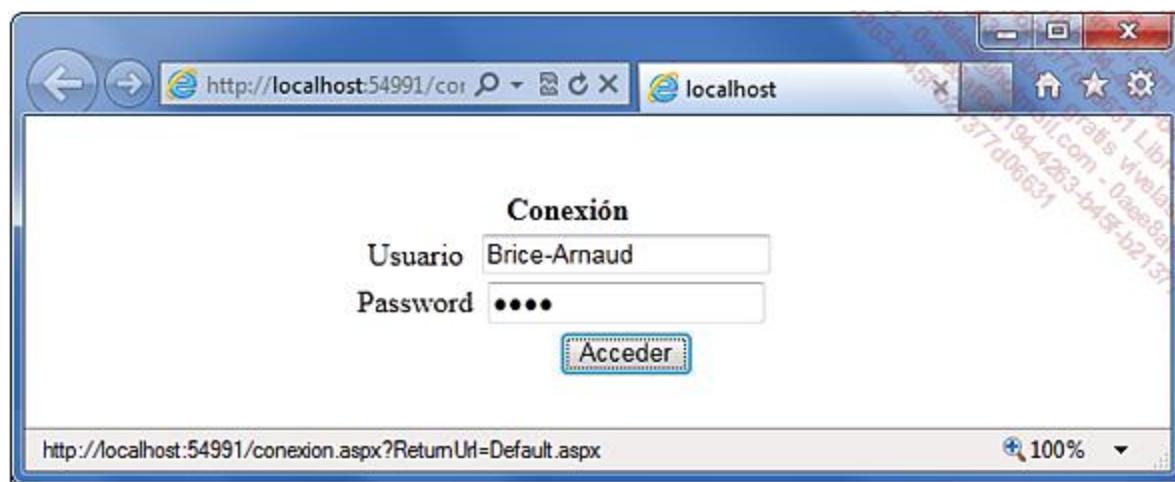
```
protected void cmd_conexion_Click(object sender, EventArgs e)
{
    Hashtable t = new Hashtable();
    // usuario = password
    t["bill gates"] = "bill";
    t["brice-arnaud"] = "brice";
```

```

if (t[txt_usuario.Text] != null &&
    (t[txt_usuario.Text] as string) == txt_password.Text)
    FormsAuthentication.RedirectFromLoginPage( txt_usuario.Text, false);
else
    lbl_error.Text = "Error de conexión. Inténtelo de nuevo.";
}

```

Sólo queda probar el sitio desde la página Default.aspx. En primer lugar, se redirige al usuario a la página de conexión. Destacamos que la página solicitada en primer lugar aparece en la cadena de consulta Query string:



Tras introducir y verificar las credenciales, el usuario llega a la página Default.aspx. Nuestra prueba del objeto User aporta información importante: por un lado, la base de datos utilizada es Forms (y User.Identity es de tipo FormsIdentity); por otro lado, el Principal empleado es un GenericPrincipal.



## b. Asignación de roles

En el marco del modo de autenticación Forms, es la aplicación la que asocia los roles a los usuarios. La operación se realiza mediante el procedimiento **Application\_AuthenticateRequest** del archivo Global.asax.

La técnica depende de si el administrador de roles (Role Manager) está activo o no mediante el atributo <roleManager enabled="true"> en el archivo Web.config.

El siguiente procedimiento indica cómo proceder en ambas situaciones. En la práctica, el código puede hacerse más sencillo si se confirma una u otra hipótesis.

Preste atención, el método Role.AddUserInRole define la pertenencia del usuario a un grupo de forma permanente (véase el párrafo acerca del proveedor de roles). El código que se muestra a continuación es útil únicamente en el marco de las pruebas, salvo para aplicaciones ASP.NET 1.X.

```
void Application_AuthenticateRequest(Object sender, EventArgs e)
{
    if (User != null && User.Identity.IsAuthenticated)
    {
        if (Roles.Enabled)
        {
            // Conveniente en aplicaciones ASP.NET 2.0
            // cuando está activo el administrador de roles
            // Código para pruebas
        }
    }
}
```

```
string role = "Comprador";

// es inútil asociar varias veces
// el mismo rol
if (User.IsInRole(role))
    return;

// verifica que existe el rol
if (!Roles.IsUserInRole(role) &&
    Roles.RoleExists(role) &&
    User.Identity.Name=="bill gates")
    Roles.AddUserToRole(User.Identity.Name, role);

}
```

```

// Conveniente en aplicaciones ASP.NET 1.X
// y 2.0 cuando el administrador de roles
// no está activo

if(User.Identity.AuthenticationType == "Forms")
{
    // crea una identidad genérica
    FormsIdentity nueva_identidad = (FormsIdentity) User.Identity;

    string[] roles = null;
    if (nueva_identidad.Name == "bill gates")
    {
        // crea una entidad principal (con roles)
        roles = new string[1];
        roles[0] = "Comprador";
    }

    System.Security.Principal.GenericPrincipal myPrincipal =
    new System.Security.Principal.GenericPrincipal( nueva_identidad, roles);
    // asociar la entidad principal al thread
    HttpContext.Current.User=myPrincipal;
}

```

```

} // Forms
} // !Roles.Enabled

} // User!=null
}

```

Ya no es preciso probar la pertenencia del usuario en la página Default.aspx, por ejemplo:

```

TextBox1.Text =
    string.Format("Nombre de usuario: {0}\nBase de datos: {1}\nAutentificado?
{2}\nTipo: {3}\nComprador? {4}",
    User.Identity.Name,
    User.Identity.AuthenticationType,
    User.Identity.IsAuthenticated,
    User.GetType().FullName,

    User.IsInRole("Comprador"));

```

Se reserva, a continuación, el acceso a la página compras.aspx a los miembros del grupo Compradores:

```
<location path="compras.aspx">  
    <system.web>  
        <authorization>  
            <allow roles="comprador"/>  
            <deny users="*"/>  
        </authorization>  
    </system.web>  
  
</location>
```



#### c. El modo Forms sin cookie

Con ASP.NET 4.5, es muy sencillo pasar de las cookies al modo Forms:

```
<authentication mode="Forms">  
    <forms loginUrl="conexion.aspx"  
    cookieless="UseUri"/>  
</authentication>
```

De este modo, el testigo de autentificación está integrado con la URL.

#### d. Autorización

Cuando se activa el modo Forms, el nombre de los roles los define el programador (o el administrador). No se prevé ningún prefijo.

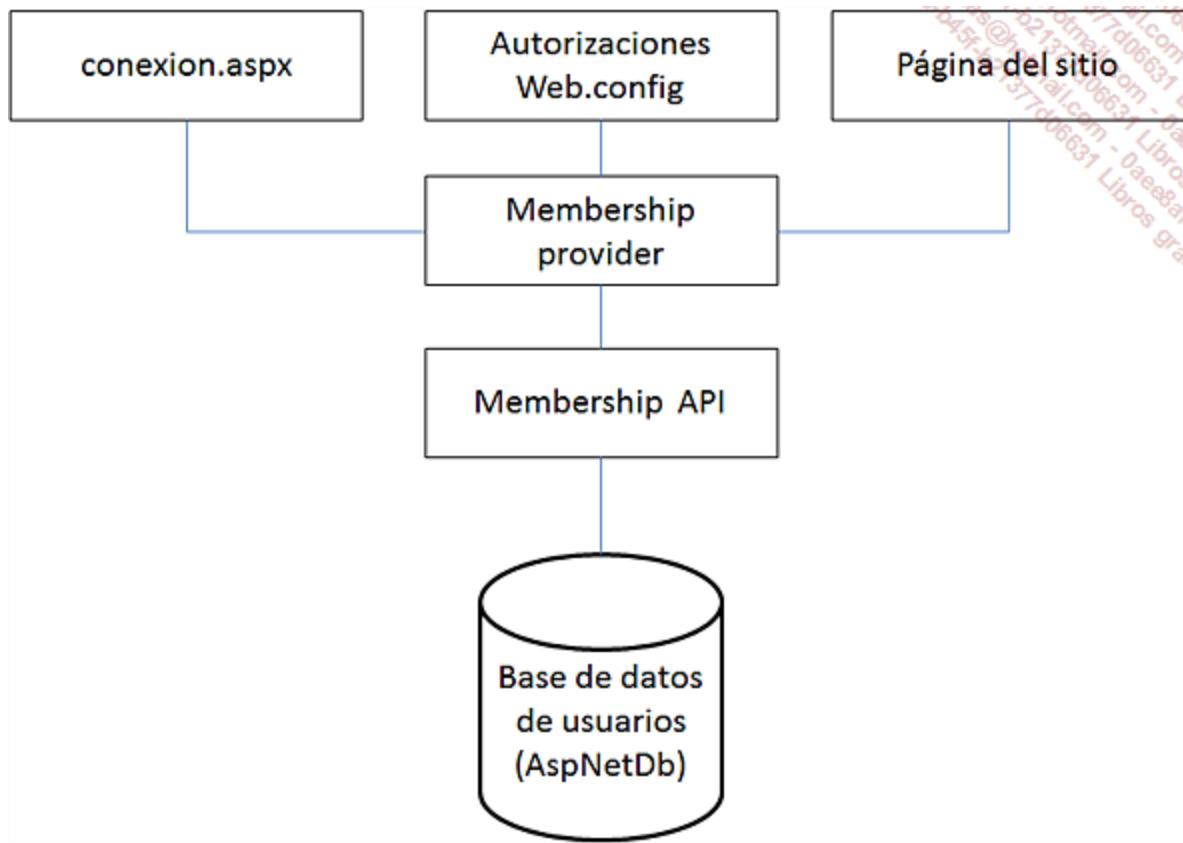
### 4. El proveedor MemberShip

Con el modo Forms de ASP.NET 1.X, lo que ganaba el programador en cuanto a flexibilidad lo perdía en aportaciones del framework. Dicho de otro modo, no había ninguna herramienta disponible para construir su propio repositorio de usuarios.

La versión 2.0 incluye el proveedor de pertenencia (Membership provider) que establece un marco de trabajo para la autentificación y la gestión de los usuarios.

## a. Funcionamiento del proveedor

El proveedor coordina la API de pertenencia (Membership API), la página de conexión, las autorizaciones declaradas en el archivo Web.config y las páginas del sitio sobre las que los usuarios deben realizar consultas.



La API Membership es un conjunto de clases destinadas a autenticar, crear, suprimir y mantener el repositorio de cuentas de usuarios. La lista de usuarios está constituida por una base de datos SQL Server (AspNetDb, véase a continuación) o por un servicio de directorio como, por ejemplo, Active Directory. Dicho de otro modo, el sistema es extensible a cualquier construcción definida por el programador.

### Conexión de los usuarios

La API Membership simplifica el proceso de autenticación de un usuario:

```
if (Membership.ValidateUser(txt_usuario.Text, txt_password.Text))  
    FormsAuthentication.RedirectFromLoginPage(txt_usuario.Text,  
    false);  
else  
    lbl_error.Text = "Error de conexión. Intente de nuevo.";
```

El método **ValidateUser()** verifica, en la base de datos seleccionada en la aplicación, la validez de las credenciales comunicadas. No es necesario abrir una conexión con la base de datos ni ejecutar una consulta para comprobar la información transmitida por el usuario.

## Enumerar los usuarios

La clase **Membership** sirve, también, para obtener la lista de todos los usuarios. La fábrica de objetos siguiente, asociada a un ObjectDataSource, la utiliza para presentar toda la información "pública" de los usuarios de la base de datos:

```
public class FabricaUsuario
{
    public FabricaUsuario ()
    {
    }

    public MembershipUserCollection GetAll()
    {
        return Membership.GetAllUsers();
    }
}
```

## Administrar la lista de usuarios

La API Membership proporciona también métodos para administrar la lista de usuarios. Esta lista se almacena en una base de datos llamada almacén (Store).

CreateUser	Agrega un usuario nuevo en el almacén asociado a la aplicación.	Se trata, a
DeleteUser	Borra un usuario del almacén.	
FindUserByEmail FindUserByName	Busca usuarios en la lista a partir de distintos criterios. Estas funciones devuelven una colección MembershipUserCollection.	
GetUser	Busca un usuario en la lista.	
UpdateUser	Actualiza algunas propiedades asociadas a un usuario.	

menudo, de métodos estáticos aplicados en la programación o actualizaciones a partir de controles de seguridad integrados.

### b. Utilizar AspNetSqlMembershipProvider

Microsoft integra en su framework ASP.NET 2.0 una herramienta destinada a crear una base de datos que almacena los usuarios de distintas aplicaciones.

#### Crear la base

El programa **aspnet\_reqsql** se utiliza con o sin parámetros. Si no se provee ningún parámetro, arranca un asistente gráfico que guía al usuario en la creación de la base de datos.

El desarrollador puede, también, inicializar la base **AspNetDb** (su nombre por defecto) pasándole parámetros por línea de comandos:

-A all m r p c w	Agrega a la base de datos servicios de aplicaciones: Todos, Membership, Roles, Perfiles, Personalización (Web parts) y Evento Web SQL.	De este modo, para crear e
-d	Nombre de la base de datos.	
-S	Nombre del servidor.	
-E	Seguridad integrada para la conexión con SQL Server.	

inicializar la base de datos sobre el servidor local, el comando es el siguiente:

```
aspnet_reqsql -d aspnetdb -S .\SQL2012 -E -A all
```

La cadena de conexión utilizada por ASP.NET para acceder a esta base de datos figura en el archivo **Machine.Config** (ubicado en la carpeta `\Windows\Microsoft.NET\Framework\v4.0\Config`). Como este archivo es voluminoso, conviene abrirlo con Visual Studio (tras realizar una copia de seguridad del mismo) y buscar la cadena de conexión LocalSqlServer :

```
<connectionStrings>
    <add name="LocalSqlServer" connectionString="data source=.
    \SQL2012;Integrated Security=SSPI;initial catalog=aspnetdb" providerName=
    "System.Data.SqlClient" />
</connectionStrings>
```

Esta cadena debe configurarse para alcanzar al servidor y la base de datos AspNetDb. Para aquellos que utilicen una versión beta de Visual Studio o que empleen SQL Server Express, es probable que la cadena definida en la instalación del producto utilice una versión de archivo de AspNetDb. Recordamos que este modo no está soportado por la versión completa de SQL Server.

En el ejemplo de configuración anterior, el servidor de aplicaciones ASP.NET se conecta con la base de datos utilizando la seguridad integrada. Para poder trabajar, el programador debe asignar, a mano, los permisos a la cuenta de ejecución ASP.NET como hemos visto en el capítulo El acceso a datos con ADO.NET 4.5. Si, por el contrario, la seguridad no está integrada, será preciso indicar las credenciales de conexión en este archivo (lo cual no resulta seguro) o en el archivo de configuración Web.config de cada aplicación.

### Configurar el proveedor y el usuario de la base de datos común

Las aplicaciones ASP.NET que funcionan con **AspNetSqlMembershipProvider** deben escoger este proveedor en su archivo Web.config:

```
<system.web>
    <membership defaultProvider="AspNetSqlMembershipProvider">
    </membership>
</system.web>
```

### Crear un (sub-)proveedor y utilizar una base de datos personalizada

Algunas aplicaciones requieren su propia lista de usuarios. Basta con crear un proveedor personalizado a partir del proveedor **SqlMembershipProvider**; no es necesario crear una base de datos separada que contenga esta lista.

```
<connectionStrings>
    <add name="miBaseAspNetDb"
        connectionString="data source=.\SQL2005;initial catalog=
aspnetdb;integrated security=true"/>
</connectionStrings>

<system.web>
    <membership defaultProvider="miProvider">
        <providers>
            <add name="miProvider"
                type="System.Web.Security.SqlMembershipProvider"
                connectionStringName="miBaseAspNetDb"
                />
        </providers>
    </membership>
</system.web>
```

El proveedor **SqlMembershipProvider** admite numerosos parámetros que se definen como atributos. Dado que se acceden como atributos, los nombres comienzan por minúscula. Cuando se trata de propiedades accesibles en programación, los nombres comienzan por una mayúscula.

name

Nombre del proveedor, devuelto por el atributo **defaultProvider** del elemento **<membership>**.

applicationName

Nombre de la aplicación asociada a los usuarios.

description

Opcional.

passwordFormat

Define el formato de almacenamiento de las contraseñas: Clear, Encrypted o Hashed.

minRequiredNon-AlphanumericCharacter

Número de caracteres no alfanuméricos en una contraseña.

minRequiredPasswordLength

Longitud mínima de una contraseña.

passwordStrengthRegularExpression

Permite imponer una máscara de expresión regular para formatear la contraseña.

enablePasswordReset

Indica si la API Membership puede utilizarse para reinicializar la contraseña mediante correo electrónico.

enablePasswordRetrieval

Indica si la API Membership puede utilizarse para encontrar una contraseña mediante correo electrónico.

maxInvalidPasswordAttempts

Indica el número de intentos antes de bloquear la cuenta de un usuario.

passwordAttemptWindow

Retardo, en minutos, antes de invalidar una conexión inactiva.

requiresQuestionAndAnswer

Indica si el dispositivo de pregunta-respuesta para restablecer una contraseña está activo en la aplicación.

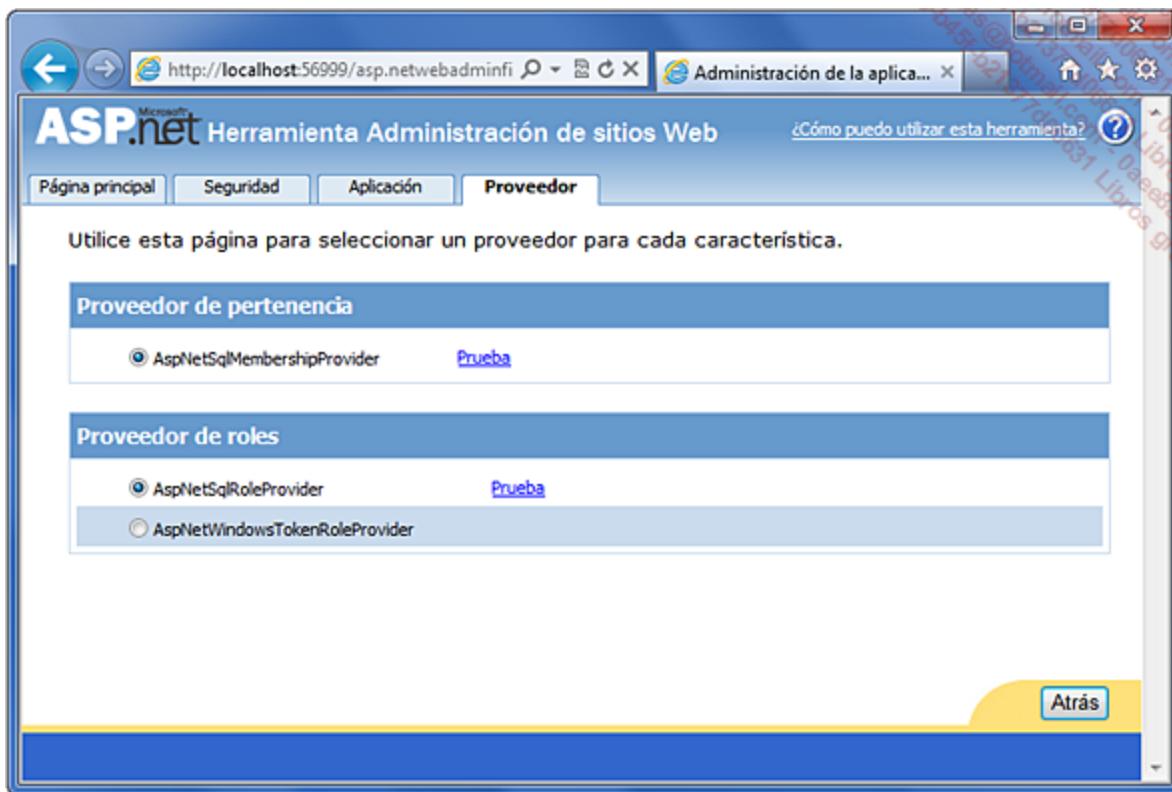
requiresUniqueEmail

Indica si se aplica una restricción de unicidad a la columna e-mail.

### **Administrar los usuarios mediante WAT**

La herramienta **Web Administration Tools (WAT)** se ejecuta desde el explorador de soluciones a partir del menú **Proyecto - Configuración ASP.NET**.

Una vez iniciado, el programador verifica el buen funcionamiento del proveedor registrado mediante la pantalla **Proveedor**:



Si la prueba no funciona, el fallo se debe, probablemente, a un error de configuración. En este caso es necesario corregirlo, detener el servidor Web y relanzar la consola WAT.

Una vez establecido el enlace, la pestaña **Seguridad** administra la lista de usuarios. Existen asistentes para registrar nuevos usuarios y asignarles roles. Recordemos que esta operación puede realizarse también mediante la API Membership que utiliza, de todos modos, la herramienta WAT.

Una vez hayamos usado la consola WAT, el archivo Web.config se modifica para tener en cuenta estos cambios de configuración solicitados por el administrador.

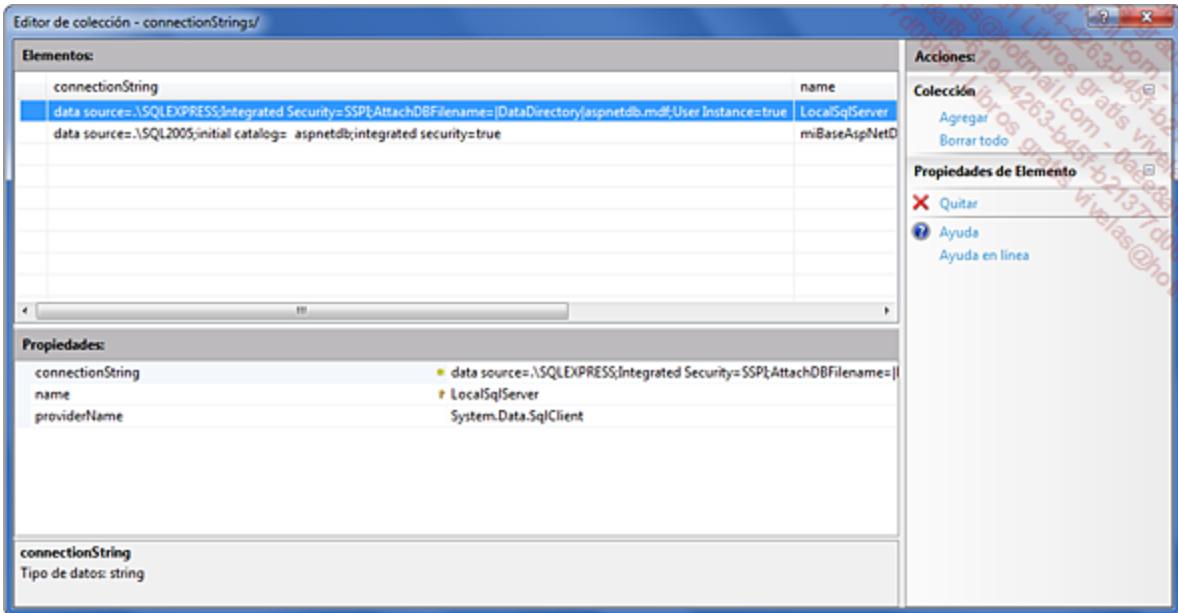
Una modificación algo incómoda que se produce consiste en la inserción del atributo xmlns en el elemento principal <configuration>:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
```

Este atributo ayuda a la consola WAT cuando se modifica el archivo Web.config. Una consecuencia de su aplicación es que Visual Studio deja de proporcionar la funcionalidad de asistente en la inserción de código intellisense. Para volver a activar esta ayuda, basta con suprimir el atributo xmlns y guardar el archivo.

### Utilizar la consola IIS

La consola IIS sirve, a su vez, para configurar la seguridad de un sitio Web ASP.NET. Es preciso editar las propiedades del sitio mediante el botón **Editor de configuración** en la pestaña **ASP.NET**:



Los parámetros son los mismos que con WAT, salvo la creación de usuarios, que no es posible.

## 5. El proveedor de roles

Tras detallar los mecanismos del proveedor de pertenencia, he aquí el estudio del proveedor de roles.

### a. AspNetSqlRoleProvider

Como con el proveedor de pertenencia, un elemento del archivo Web.config permite utilizar el proveedor de roles y precisa con qué tipo de proveedor trabajará.

```
<roleManager enabled="true">
    defaultProvider="AspNetSqlRoleProvider">
</roleManager>
```

Hay otros atributos disponibles para alojar en caché los roles del usuario en curso mediante cookies. Como el testigo de autenticación se aloja en caché, esta solución evita una conexión a la base de datos con cada petición HTTP.

cacheRolesInCookie	Indica si los roles se alojan en caché.	Una vez
maxCachedResults	Número de roles que se alojan en caché.	
cookieName	Nombre de la cookie.	
cookiePath	Ruta de la cookie.	
cookieProtection	All, Encryption, Validation, None.	
cookieRequiresSSL	True o False.	
cookieSlidingExpiration	True o False.	

declarado el proveedor, es posible crear roles mediante la interfaz WAT:

The screenshot shows a user management interface. At the top, there's a search bar with fields for 'Buscar por' (Nombre de usuario) and 'buscar'. Below it is a list of letters from A to Z for filtering. A modal window is open for the user 'admin1', showing its status as active and listing its functions: 'Comprador'. There's also a link to 'Añadir función'.

Como con el proveedor de pertenencia, existe una API para generar los roles y probarlos con el usuario en curso:

```
Label1.Text = (Roles.IsUserInRole("Comprador") ? "Usted es miembro del grupo Comprador" : "");
```

La clase rol dispone de los siguientes miembros:

Provider	Proveedor usado actualmente por la aplicación.	b.
Providers	Lista de proveedores registrados.	
AddUser(s) ToRole(s)	Asocia roles a los usuarios.	
CreateRole	Crea un rol.	
DeleteRole	Borra un rol.	
FindUsersInRole	Encuentra los usuarios afectados por un rol.	
GetAllRoles	Enumera todos los roles.	
GetRolesForUser	Enumera los roles de un usuario.	
IsUserInRole	Verifica si el usuario posee un rol. Es equivalente a User.IsInRole.	
RemoveUser(s) FromRole(s)	Desenlaza los roles a los usuarios.	
RoleExists	Verifica que existe un rol.	

### WindowsRoleTokenProvider

Se trata de un tipo de proveedor que permite utilizar la API Rol con el modo de autenticación Windows. Como el nivel de seguridad de los usuarios y de los grupos del sistema operativo es algo crucial, sólo es posible utilizar este proveedor para probar la pertenencia de un usuario, pero no para registrar nuevos roles. Se trata, por tanto, de un proveedor de "sólo lectura".

Conviene aplicar la sintaxis siguiente para declarar el uso de este proveedor:

```
<roleManager enable="true" defaultProvider=
```

```
"AspNetWindowsTokenRoleProvider" />
```

### c. Crear un proveedor personalizado para Active Directory

No existe, de manera estándar, un proveedor de roles adaptado a Active Directory (o ADAM). Proponemos, a continuación, crear un proveedor personalizado. De cara a simplificar, nuestro proveedor es de sólo lectura, aunque el programador lo completará fácilmente.

#### **Registro en el archivo Web.config**

El proveedor se implementa a partir de la clase MyAdamRoleProvider ubicado en la DLL myadamroleprov. Tras agregar una referencia a esta DLL, la sintaxis siguiente permite declarar el uso del proveedor personalizado:

```
<roleManager enabled="true" defaultProvider="miProviderAdam">
    <providers>
        <add name="miProviderAdam"
            type="myadamroleprov.MyAdamRoleProvider, myadamroleprov"
            connectionStringName="ADService"
        &n
```

# Presentación personalizada

El interés de los proveedores de pertenencia y de roles no se limita únicamente a la securización de una aplicación. El usuario espera que un sitio web tenga una presentación personalizada, una navegación adaptada y que los mensajes se muestren en su idioma.

## 1. Perfiles de usuario

Los perfiles de usuario aprovechan el conocimiento del servidor de aplicaciones ASP.NET en materia de generación dinámica de código.

Se trata de generar una clase en el Web.config de la que existe una instancia por cada usuario. Esta instancia se memoriza en la base de datos AspNetDb (o en la base de datos asociada al proveedor) y se asocia al usuario con cada nueva conexión.

El perfil expone propiedades que siempre tienen valor. Habitualmente no es necesario que se guarde el perfil, salvo que contenga elementos de tipo colección a los que el servidor de aplicaciones le costará seguir su evolución.

### a. Formación del perfil

El perfil se forma en el archivo Web.config. Es preciso indicar qué proveedor se desea utilizar y si es necesario describir las características de este proveedor. En nuestro caso, se trata del proveedor estándar ligado a la base de datos AspNetDb.

Las propiedades del perfil definen propiedades accesibles por programación. Tienen un nombre, un tipo y un valor por defecto:

```
<profile enabled="true" defaultProvider="AspNetSqlProfileProvider">
    <properties>
        <add name="Theme" type="System.String" defaultValue="blanco"/>
        <add name="MensajeDeBienvenida" type="System.String" defaultValue=
        "Bienvenido"/>
    </properties>
</profile>
```

Cuando se registra el archivo Web.config, el servidor de aplicaciones genera una clase que tiene estas propiedades.

### b. Uso del perfil

La clase generada se instancia bajo el nombre Profile. El siguiente programa utiliza el perfil del usuario para escoger su mensaje de bienvenida así como su tema de presentación.

```
public partial class _Default : System.Web.UI.Page
{
    protected override void OnPreInit(EventArgs e)
    {
```

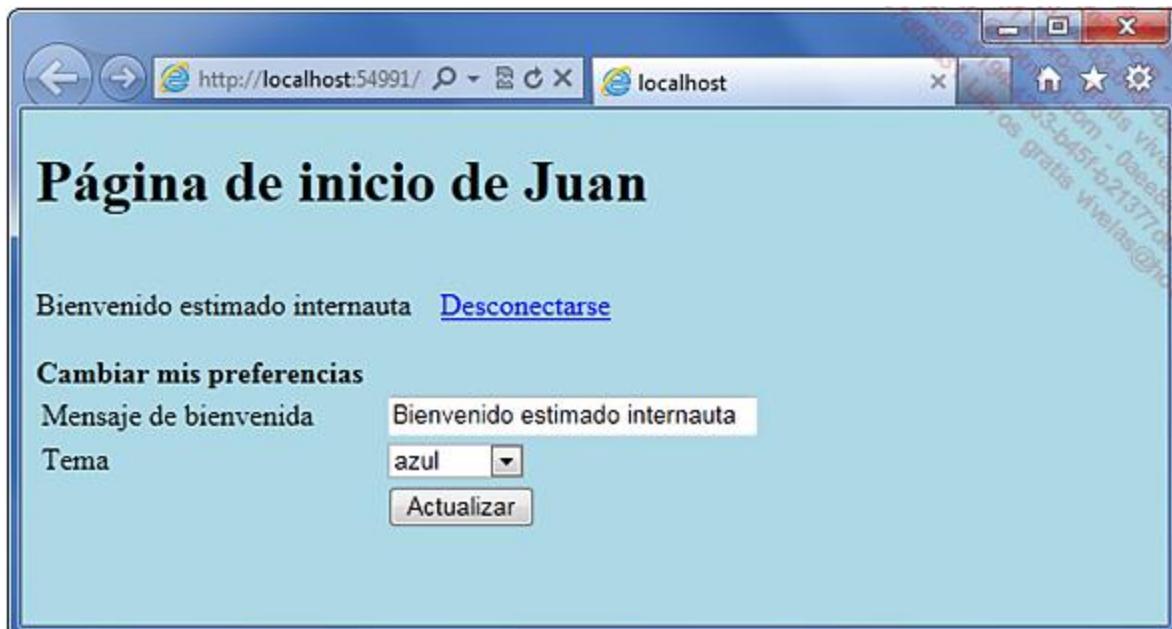
```
        this.Theme = Profile.Theme;
        base.OnPreInit(e);
    }

protected void Page_Load(object sender, EventArgs e)
{
    lbl_bienvenida.Text = Profile.MensajeDeBienvenida;

    if (!IsPostBack)
    {
        txt_bienvenida.Text = Profile.MensajeDeBienvenida;
        switch (Profile.Theme)
        {
            default:
            case "blanco":
                lst_theme.SelectedIndex = 0;
                break;

            case "azul":
                lst_theme.SelectedIndex = 1;
                break;
        }
    }
}

protected void cmd_cambio_perfil_Click(object sender, EventArgs e)
{
    Profile.MensajeDeBienvenida = txt_bienvenida.Text;
    Profile.Theme = lst_theme.SelectedValue;
}
```



### c. Agrupación y tipos complejos

Cuando el perfil expone muchas propiedades, el programador puede estructurarlos recurriendo a la agrupación:

```
<properties>
    <add name="Theme" type="System.String" defaultValue="blanco"/>
    <add name="MensajeDeBienvenida" type="System.String" defaultValue=
        "Bienvenido"/>
    <group name="Seguridad">
        <add name="RetardoDeConexion" defaultValue="10" type="System.Int32"/>
        <add name="ConexionesConcurrentes" defaultValue="true" type=
            "System.Boolean"/>
    </group>
</properties>
```

El grupo crea un nivel de agrupación en el objeto perfil:

```
int retardo = Profile.Seguridad.RetardoDeConexion;
```

Generalmente, es posible utilizar tipos complejos - como colecciones - o tipos personalizados. El siguiente ejemplo utiliza una colección de cadenas (string):

```
<add name="FraseDelDia" type="ListString"
    serializeAs="Binary"/>
```

La definición de la clase `ListString` aparece a continuación. Es indispensable aplicar el atributo `[Serializable]` para indicar al framework que la clase es compatible con una serialización binaria:

```

[Serializable]

public class ListString : CollectionBase
{
    public ListString() : base()
    {
    }

    public void Add(string s)
    {
        this.List.Add(s);
    }

    public string this[int index]
    {
        set { List[index] = value; }
        get { return List[index] as string; }
    }
}

```

Sólo queda probar esta propiedad del perfil:

```

if (Profile.FraseDelDia != null)
{
    int max = Profile.FraseDelDia.Count - 1;
    if (max >= 0)
    {
        Random r = new Random(DateTime.Now.Second);
        int pi = r.Next(max);
        lbl_frase.Text = Profile.FraseDelDia [pi];
    }
    else
        lbl_frase.Text = "";
}

```

Y:

```

if (Profile.FraseDelDia == null)
    Profile.FraseDelDia = new ListString();
if (txt_frase.Text != "")
{

```

```
        Profile.FraseDelDia.Add(txt_frase.Text);
        Profile.Save(); // necesario
    }
```



## 2. Navegación dentro del sitio

Hemos visto en el capítulo Los Web Forms cómo las páginas maestras y los controles .ascx facilitaban la reutilización de partes de interfaces gráficas. Nos interesamos, a continuación, en la navegación entre las distintas páginas del sitio.

### a. El archivo de definición del sitio

El servidor de aplicaciones ASP.NET define un nuevo formato de archivo, el Web.sitemap. Este archivo constituye el mapa del sitio, es decir, detalla la jerarquía de las páginas. Se trata, naturalmente, de un archivo XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="Default.aspx" title="Inicio" description=
    "Inicio">
```

```
        <siteMapNode url="mis cuentas.aspx" title="Mis cuentas" description=
        "Listado de mis cuentas">
            <siteMapNode url="operaciones.aspx" title="Operaciones" description=
            "Detalle de las operaciones de una cuenta"/>
        </siteMapNode>

        <siteMapNode url="Movimientos.aspx" title="Movimientos" description=
```

```

"Movimientos">

    <siteMapNode url="nuevomov.aspx" title="Nuevo movimiento"
description="Realizar un movimiento"/>

    <siteMapNode url="seguirmov.aspx" title="Seguir un movimiento"
description="Seguir un movimiento"/>

</siteMapNode>
</siteMapNode>
</siteMap>

```

Este archivo debe respetar la construcción en árbol. Por un lado, la sintaxis XML impone que un elemento de origen `<siteMap>` agrupe elementos descendientes `<siteMapNode>`. Esta es la estructura de un árbol. Aun así, el servidor de aplicaciones va más allá, puesto que la URL de una página no puede figurar varias veces en la jerarquía creada.

No se trata de una limitación tecnológica, sino de una restricción aplicada a la programación con el objetivo de que no se defina un grafo (un árbol cuyos nodos puedan tener varios padres). En efecto, este archivo Web.sitemap puede explotarse mediante distintos controles gráficos tales como el menú. Admitir que una página aparezca más de una vez en el archivo sería molesto para el usuario.

Esta limitación debe, no obstante, superarse cuando el programador prepara distintas navegaciones según el rol del usuario. Veremos, un poco más adelante, cómo proceder en este caso.

### b. El proveedor SitemapProvider, la API Sitemap y el SitemapDataSource

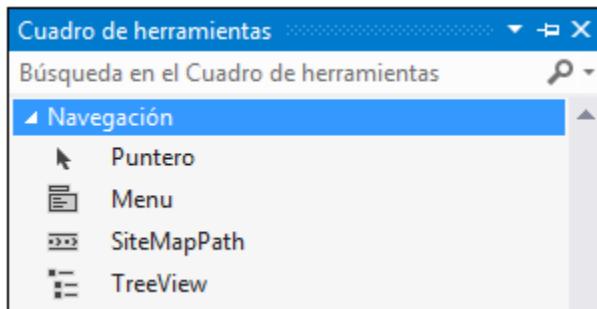
En sí mismo, el archivo Web.sitemap no se explota mediante controles gráficos. Existe una doble capa de abstracción formada por la API y un proveedor que se comporta como un aislante.

Para no especializar los controles, ASP.NET dispone también del SitemapDataSource, enlazado al proveedor. A menudo, basta con instanciar este control para aprovechar la información contenida en el archivo Web.sitemap:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

### c. Controles asociados a la navegación

El cuadro de herramientas proporciona varios controles asociados a la navegación:



SiteMapPath

Muestra la posición del usuario dentro del mapa del sitio.

Para  
unificar

Menu	Proporciona un sistema de navegación para pasar directamente de una página a otra.	la
Treeview	Proporciona un sistema de navegación que se presenta como un explorador de datos jerárquicos (parecido al panel izquierdo del explorador de archivos de Windows).	

presentación, el programador puede querer ubicar estos controles en una página maestra (master-page).

#### d. Filtrar la representación en función del usuario

##### **Filtrado mediante un proveedor personalizado**

Es posible restringir el acceso a una entrada del archivo Web.sitemap utilizando el atributo roles:

```
<siteMapNode url="movespecial.aspx" title="Movimiento especial"
roles="super_usuario"/>
```

Sólo los titulares con el rol super\_usuario podrán realizar la función "movimiento especial". De hecho, esta sintaxis está destinada principalmente a la escritura de un proveedor de mapa personalizado.

La implementación correspondiente es muy simple:

```
public class MySiteMapProvider : XmlSiteMapProvider
{
    public MySiteMapProvider()
    {
    }

    // indica, para cada nodo, si el usuario tiene acceso
    public override bool IsAccessibleToUser(HttpContext context,
SiteMapNode node)
    {
        if (!base.SecurityTrimmingEnabled)
            return true;
    }
}
```

```
    return Roles.IsUserInRole(node.Roles[0].ToString());
}

}
```

##### **Filtrado mediante servidor de aplicaciones**

Ocultar una URL en el menú no es suficiente. La sección <authorizations> del sitio debe bloquear el acceso a ciertas categorías de usuario. Resulta, por tanto, implícito ocultar las entradas

correspondientes en el menú. Reconfigurando el proveedor por defecto es posible alcanzar este resultado sin escribir un proveedor personalizado:

```
<siteMap defaultProvider="XmlSiteMapProvider">
    enabled="true">
    <providers>
        <add name="XmlSiteMapProvider"
            description="Proveedor por defecto con seguridad activa"
            type="System.Web.XmlSiteMapProvider, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
            securityTrimmingEnabled="true"
            siteMapFile="Web.sitemap" />
    </providers>
</siteMap>
<location path="movespecial.aspx">
    <system.web>
        <authorization>
            <allow roles="super_usuario"/>
            <deny users="*"/>
        </authorization>
    </system.web>
</location>
```

### 3. Internacionalización

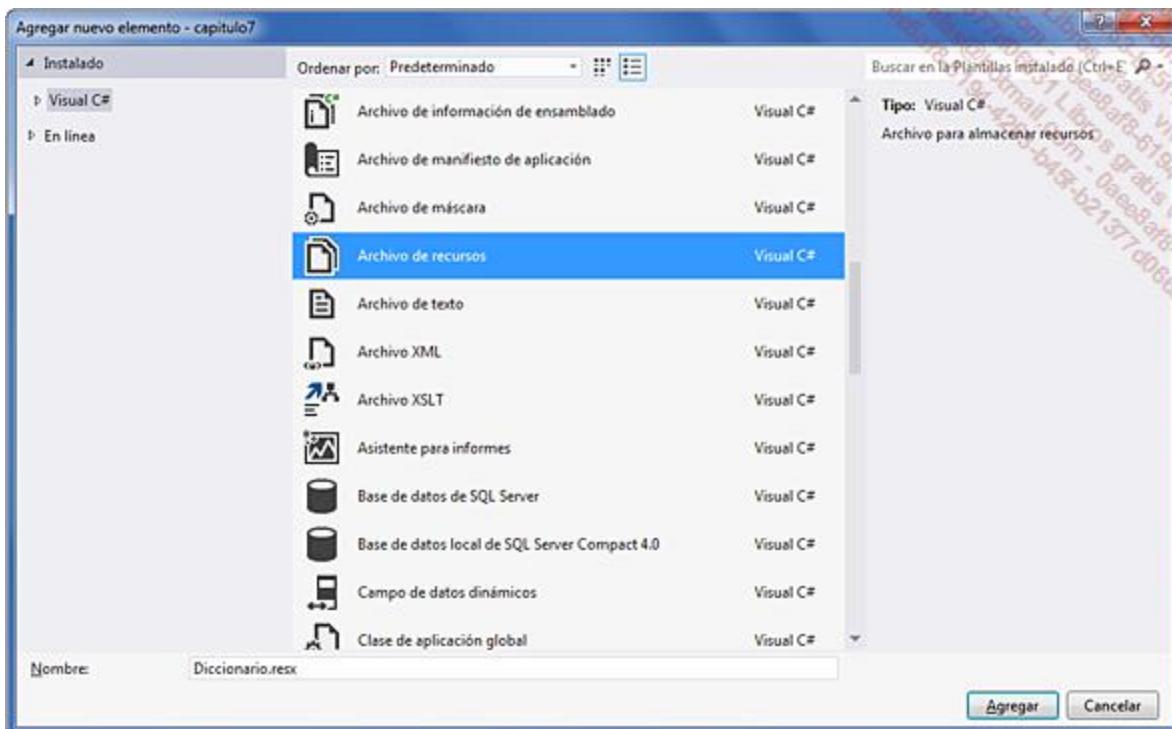
El framework .NET proporciona varios mecanismos para construir una aplicación accesible a un público internacional. Es posible, de entrada, formatear divisas, fechas y números según una culturaespecífica. Recordemos que el framework conoce tres tipos de cultura: invariante, neutra y específica. La cultura invariante sirve para comparar cadenas de caracteres independientemente de su página de código. La cultura neutra, español es, inglés en, etc., resulta útil para crear diccionarios de datos, crear pantallas personalizadas o conocer el nombre de los días en un idioma particular. La cultura específica, español en España es-es, español en Argentina es-ar, etc., es capaz de formatear números y divisas.

#### a. Recursos globales

Los recursos globales son archivos XML compilados que definen términos de acceso mediante un sistema de claves. La técnica, específica desde la versión 2.0 de ASP.NET, se asemeja a los ensamblados satélite.

Los archivos .resx se crean en la carpeta especial App\_GlobalResources. Como vemos, el servidor de aplicaciones va a generar una clase fuertemente tipada para cada mensaje, lo que disminuye el riesgo ligado al enfoque débilmente tipado de los ensamblados satélite.

Partimos de un archivo llamado Diccionario.resx que incluye los mensajes en el idioma por defecto:



Se define una cadena llamada bienvenida:

Nombre	Valor
bienvenida	Con Banca, gestione sus cuentas online
*	

El archivo se copia, a continuación, con el nombre Diccionario.en.resx y se traduce el mensaje al inglés.

A continuación, se extrae de la clase Diccionario asociada al espacio de nombres **Resources**:

```
protected void Page_Load(object sender, EventArgs e)
{
    lbl_titulo.Text = Resources.Diccionario.bienvenida;
}
```

Es posible escoger una cultura mediante el atributo **UICulture**:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
```

```
AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
Title="Untitled Page" Theme="general"
UICulture="en" %>
```

Aunque el programador puede modificarla dinámicamente:

```
protected override void FrameworkInitialize()
{
    switch (Profile.idioma)
    {
        case 0:
        default:
            this.UICulture = "es";
            break;

        case 1:
            this.UICulture = "en";
            break;
    }
    base.FrameworkInitialize();
}
```

## b. Recursos locales

Mientras que algunos elementos se comparten entre varias páginas, otros no tienen más que un sentido local. Los recursos locales adaptan la localización de los Winforms a las aplicaciones ASP.NET.

La creación de un archivo de recursos locales se realiza mediante el comando **Herramientas - Generar recurso local**, desde una página en modo Diseño.

Visual Studio genera, entonces, un archivo con el nombre de la página en la carpetaApp\_LocalResources.

Como en el caso de los recursos globales, el archivo se copia con el nombreDefault.aspx.en.resx para crear un juego de mensajes en inglés.

Los controles Web que tengan que extraer mensajes de los recursos locales utilizan un mecanismo de atributo **meta:resourceKey**:

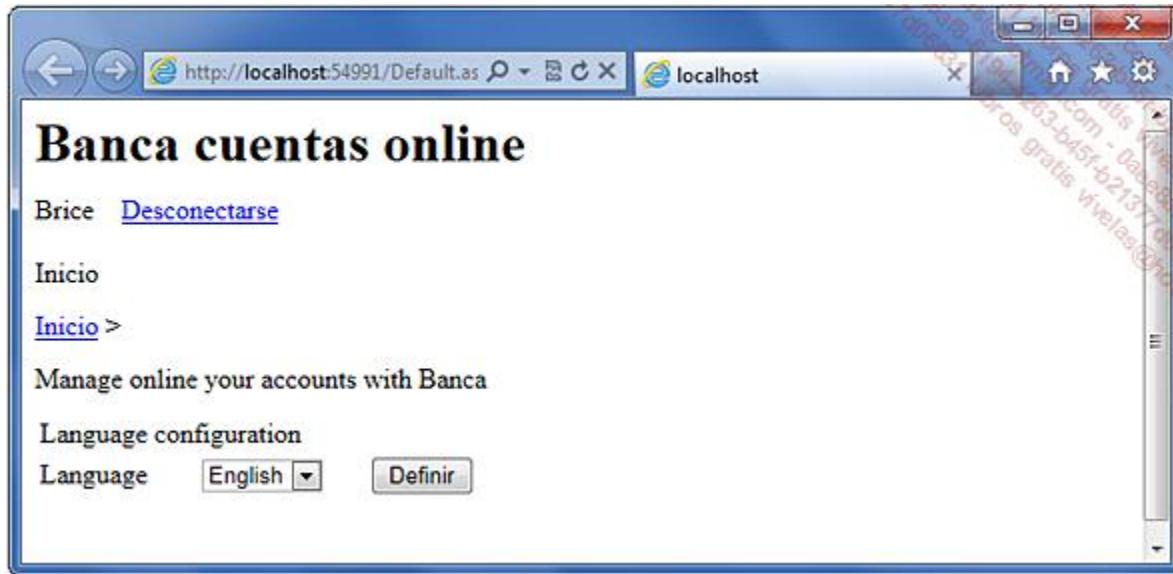
```
<asp:Label ID="lbl_idioma" runat="server"
    Text="Idioma de visualización" meta:resourcekey="lbl_idiomaRecurso1">
</asp:Label>&nbsp;
<asp:DropDownList ID="lst_idioma" runat="server"
```

```

meta:resourcekey="lst_idiomaRecurso1">
    <asp:ListItem Value="0" meta:resourcekey="ListItemRecurso1">
Español</asp:ListItem>
    <asp:ListItem Value="1" meta:resourcekey="ListItemRecurso2">
Inglés</asp:ListItem>
</asp:DropDownList>&nbsp;
<asp:Button ID="cmd_definir_perfil" runat="server" Text="Definir"
meta:resourcekey="cmd_definir_perfilRecurso1" OnClick="cmd_definir_perfil_Click" />

```

Sólo queda probar la aplicación en distintos idiomas. En la captura de pantalla que se muestra a continuación, los controles específicos de la página Default.aspx se han traducido al inglés. Por el contrario, los mensajes dispuestos en la página maestra están escritos a fuego, aunque es posible generar recursos locales para este tipo de página.



### c. El componente Localize

El componente Localize funciona como un recurso local. La diferencia está en el hecho de que el mensaje extraído no es un valor de atributo.

```

<asp:Localize ID="Localize1" runat="server" meta:resourcekey="parametro">
    Parámetros lingüísticos
</asp:Localize>

```

### d. Localización de las validaciones

Los controles integrados en ASP.NET encargados de validar los datos introducidos por los usuarios funcionan, básicamente, con JavaScript. Este lenguaje dispone de algunas funciones básicas tales como IsNaN (Not a number) útiles para verificar si el contenido textual de un campo puede aceptarse como número decimal.

La ausencia de un soporte internacional para JavaScript hace que este dispositivo sea inútil en muchos países. Es preciso, por tanto, utilizar controles que verifican los datos introducidos por los usuarios mediante una máscara de tipo expresión regular, como por ejemplo RegularExpressionValidator.

El caso de una aplicación multiidioma complica un tanto las cosas, pues cada cultura define sus propias máscaras de validación. Lo mejor es, por tanto, codificar las expresiones regulares en un archivo de recursos globales, puesto que se comparten entre las distintas páginas Web del sitio. Utilizando una \$-expression, es, por tanto, posible aplicar a cada control de validación la expresión conveniente, sin escribir una sola línea de código relativo a la cultura en curso.

Tomemos el ejemplo de un campo destinado a recibir un número decimal. Proponemos, para las culturas **es** (cultura por defecto) y **en** las siguientes expresiones:

es	[\+\-]?\d+\.\d{2}
en	[\+\-]?\d+\.\d{2}

Entre una cultura y otra sólo varía el separador decimal. La máscara obliga a introducir un signo (opcional), una parte entera, un separador y exactamente dos cifras decimales para la parte fraccionaria.

Una expresión compilada \$-expression sirve de enlace entre el archivo y el componente RegularExpressionValidator:

```
<asp:RegularExpressionValidator ID="reg_validate" runat="server"
    ControlToValidate="txt_number"
    ErrorMessage="Se esperaba un número decimal"
    meta:resourcekey="reg_validarRecurso1"
    ValidationExpression="<%$ Resources:ResourceFormat,
floatexpression %>" ></asp:RegularExpressionValidator>
```

Según la cultura definida en tiempo de ejecución, una de las expresiones de validación se transmite al componente antes de realizar su control mediante JavaScript.

# Los Web Parts

## 1. Del sitio Web al portal

Los perfiles son un medio práctico para conservar las preferencias de un usuario. Los elementos Web (Web Parts) van más allá permitiendo a los usuarios personalizar la presentación, tanto en su disposición como en su contenido.

Se trata, a menudo, de páginas de inicio que contienen cierta personalización. Se construyen a base de zonas enlazadas con distintos orígenes de información: flujos RSS, servicios Web, datos SQL... El usuario selecciona los canales de información que le convienen, y los organiza para acceder rápidamente a los temas que le interesan.

Un sitio Web capaz de adaptarse a este funcionamiento se convierte en un portal de información. Microsoft proporciona, por sí mismo, el programa Sharepoint Portal Server, ligado a distintos flujos de un sistema de información. La tecnología de Web Parts es la ocasión para el desarrollador del sitio Web ASP.NET de obtener el mismo tipo de resultados.

## 2. Crear un portal

Para construir un portal, partimos de una página de inicio. Ésta se desarrolla como cualquier otra página Web ASPX. Su implementación general se organiza mediante tablas, controles Web... Agregamos, a continuación, los elementos necesarios para la tecnología de Web Parts.

Los Web Parts requieren una base de datos persistente que puede ser AspNetDb. El archivo Web.config indica al servidor de aplicaciones el proveedor y la base de datos que se utilizará:

```
<webParts>
    <personalization defaultProvider="MyProvider">
        <providers>
            <add name="MyProvider"
                type="System.Web.UI.WebControls.WebParts.
                SqlPersonalizationProvider"
                connectionStringName="localSql" />
        </providers>
    </personalization>
</webParts>
```

### a. El gestor WebPartManager

El primer componente que hay que ubicar en una página portal es el control **WebPartManager**. Tiene como objetivo coordinar la comunicación entre los distintos elementos de la página.

```
<asp:WebPartManager ID="MyPartManager" runat="server"
    CloseProviderWarning="Desea cerrar este elemento? Si"
```

```
trata de pasarle los datos a otro elemento, la conexión
y el elemento se cerrarán. Para cerrar este elemento, haga clic
```

```

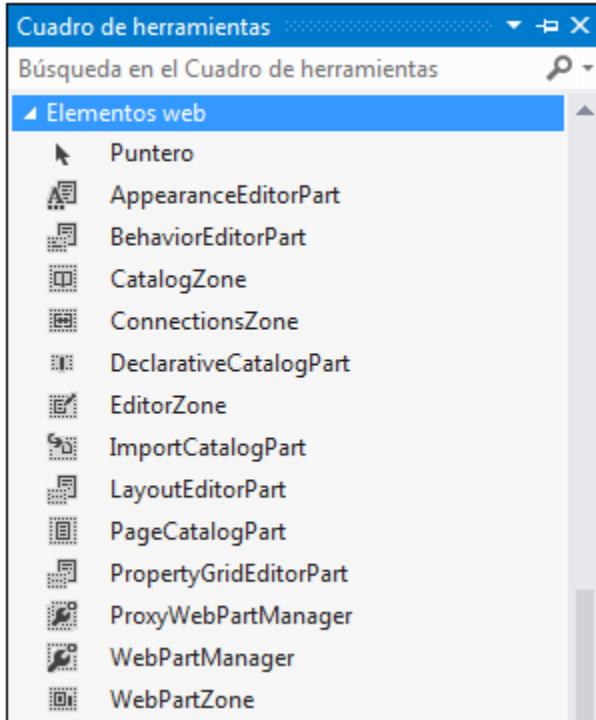
en Aceptar. Para mantenerlo, haga clic en Cancelar."
DeleteWarning="Va a borrar un elemento.
Confirme con Aceptar, o bien haga clic en Cancelar"
ExportSensitiveDataWarning="Esta página contiene elementos
personalizados. Verifique que las contraseñas no son accesibles."
>
</asp:WebPartManager>
```

Si los elementos no están conectados, las propiedades del gestor no tienen mucha importancia.

### b. Las zonas WebPartZone

Una zona es un contenedor de elementos. El programador del portal posiciona inicialmente elementos (Web Parts) en el interior de estas zonas, pero el usuario podrá desplazarlas a continuación.

Las zonas son, pues, superficies activas. Algunos navegadores aprovechan mejor el HTML dinámico, mientras que otros obligan al usuario a organizar su página mediante un catálogo de elementos.



La sección **WebParts** del cuadro de herramientas proporciona todos los controles necesarios para la realización de un portal. El control **WebPartZone** puede, así, instanciarse tantas veces como sea necesario en el formulario ASP:

```

<asp:WebPartZone ID="zonaPrincipal" runat="server" EmptyZoneText=
"Agreece elementos desplazándolos sobre esta linea.">
<ZoneTemplate>
```

```

</ZoneTemplate>
</asp:WebPartZone>

```

<b>id_empresa</b>	<b>empresa</b>	<b>compra</b>	<b>evolucion</b>
0	abc	0	0
1	abc	0,1	0,1
2	abc	0,2	0,2
3	abc	0,3	0,3
4	abc	0,4	0,4

El valor del atributo **EmptyZoneText** da a entender que varios elementos pueden coexistir en el interior de una zona, lo cual es cierto.

Una zona dispone de un cuadro, llamado cromo. Sobre este cromo (que puede tener varios aspectos) se anclan los comandos, llamados verbos. Estos comandos sirven para controlar la representación gráfica de los elementos contenidos en la zona: cierre, borrado, ocultación...

### c. Los elementos WebPart

Cuando controles de distintos tipos, Web, de usuario o personalizados se instancian en el interior de una zona, se vuelven automáticamente elementos **WebPart**.

```

<asp:WebPartZone ID="zonaPrincipal" runat="server" EmptyZoneText="Agregue
elementos deslizándolos sobre esta línea.">

    <ZoneTemplate>

        <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns=
"True" DataKeyNames="id_soc" DataSourceID="SqlDataSource1" OnLoad=
"GridView1_Load">

            </asp:GridView>
    </ZoneTemplate>

</asp:WebPartZone>

```

Desde un punto de vista HTML, su sintaxis no cambia. Pero en modo Diseño o en funcionamiento se convierten en elementos de contenido que el usuario consulta, oculta, desplaza o borra.

<b>id_empresa</b>	<b>empresa</b>	<b>compra</b>	<b>evolucion</b>
1	Mar y Montaña	123,12	2,12
2	Especias de oriente	89,98	1,92
3	Cloud-W	72,69	3,01
4	BricoSalud	87,25	1,09

<b>id_empresa</b>	<b>empresa</b>	<b>compra</b>	<b>evolucion</b>
1	Mar y Montaña	123,12	2,12
2	Especias de oriente	89,98	1,92
3	Cloud-W	72,69	3,01
4	BricoSalud	87,25	1,09

### 3. Los controles de catálogo CatalogZone y PageCatalogPart

#### a. El catálogo de zonas

Parte de la página portal centraliza, a menudo, comandos que ayudan al usuario en su organización: se trata de la zona catálogo.

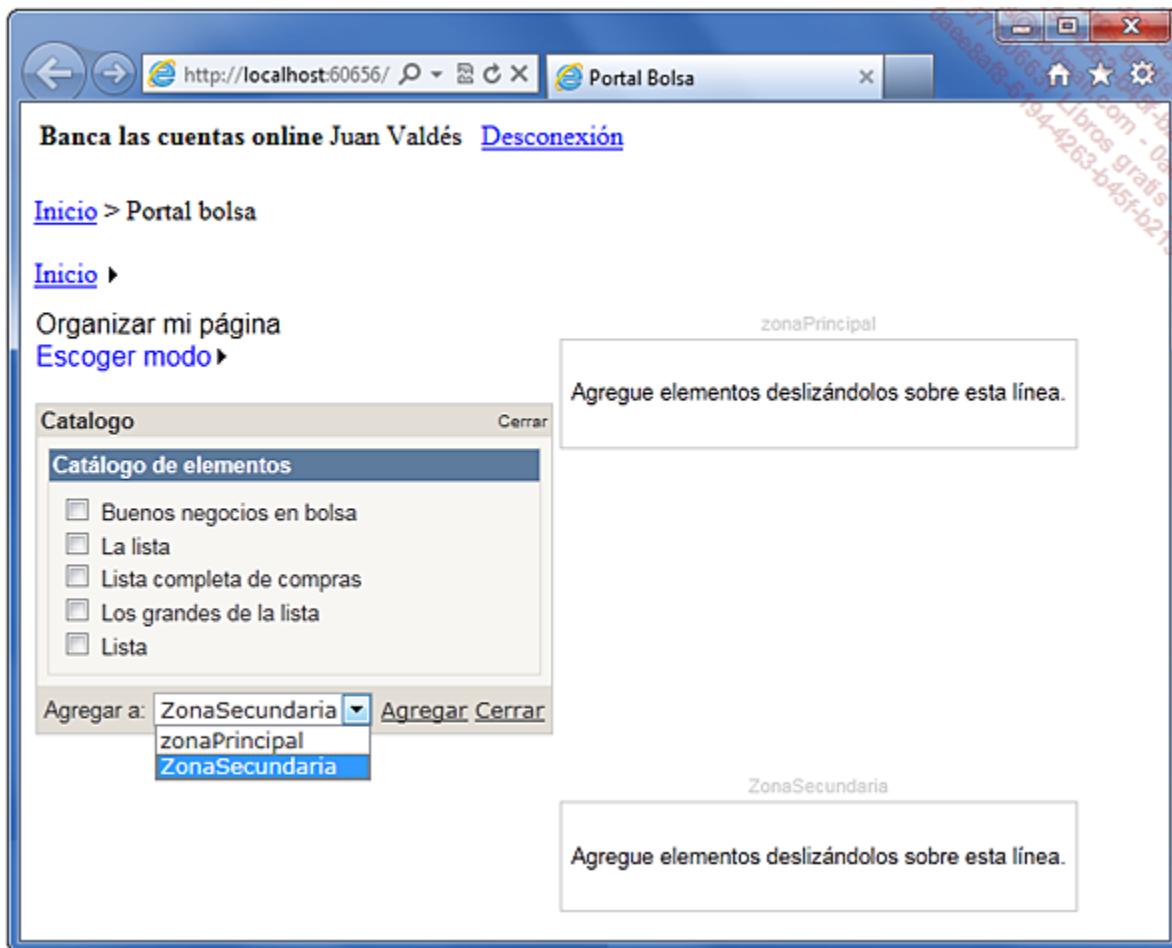
```
<asp:CatalogZone ID="CatalogZone1" runat="server" EmptyZoneText=
"El catálogo no contiene ningún elemento"
HeaderText="Catalogo" InstructionText="Seleccione en el catálogo
el elemento que quiere mostrar" SelectTargetZoneText="Agregar a:
" VerbButtonType="Link">
<ZoneTemplate>
```

```
<asp:PageCatalogPart ID="PageCatalogPart1" runat="server"
Title="Catálogo de elementos" />
</ZoneTemplate>
<AddVerb Description="Agregar un elemento a una zona"
Text="Agregar" />
<CloseVerb Text="Cerrar" />
```

```
<HeaderCloseVerb Description="Cerrar la zona" />  
</asp:CatalogZone>
```

Como vemos en el extracto de código, proporciona también verbos para agregar, desplazar o borrar elementos en el interior de las zonas de la página.

La lista de elementos disponibles para la página viene dada por el elemento **PageCatalogPart**. En tiempo de ejecución, el catálogo muestra todos los elementos que no están activos en alguna zona:



### b. Menú para cambiar de modo

El usuario no siempre quiere organizar su página. A menudo, encuentra una disposición que le conviene, y realiza cambios sólo de vez en cuando. Podemos agregar a nuestra página portal un menú alimentado dinámicamente mediante acciones "portal" soportadas por la página:

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if (!this.IsPostBack)  
    {  
        MenuItem Root = new MenuItem("Escoger un modo");
```

```

foreach (WebPartDisplayMode mode in MyPartManager.DisplayModes)

    if (mode.IsEnabled(MyPartManager) )

        Root.ChildItems.Add(new MenuItem(mode.Name) );

    PartsMenu.Items.Add(Root);

}

}

```

Cuando se elige una opción, se notifica al administrador de elementos del cambio de comportamiento:

```

protected void PartsMenu_MenuItemClick(object sender, MenuEventArgs e)
{
    foreach (WebPartDisplayMode modo in MyPartManager.DisplayModes)
        if (modo.Name.Equals(e.Item.Text))
            MyPartManager.DisplayMode = modo;
}

```

Los distintos modos disponibles son:

Browse	La interfaz funciona con normalidad, todos los elementos están en su sitio.	Cada modo
Catalog	Está activa la zona de catálogo, el usuario realiza cambios en los elementos.	
Design	El usuario desplaza los elementos de una zona a otra desplazando y soltando.	
Edit	El usuario personaliza los elementos que componen su interfaz: nombre, apariencia, comportamiento.	
Connect	Se inician las comunicaciones entre los elementos.	

necesita una o varias zonas y elementos específicos para funcionar. Así, el modoCatalog requiere el uso de una zona catálogo. Si se activa un modo sin que esté soportado por alguna zona, se obtiene un error y se interrumpe la ejecución de la página.

### c. Dar nombre a los elementos

El programador habrá observado que los elementos tienen el nombre "Untitled" (sin título) en Visual Studio. Para evitar que este nombre aparezca en tiempo de ejecución, los elementos deben tener un nombre. No obstante, la instanciación de un elemento (como objeto) no tiene lugar sino después de la ejecución de la página. Lo mejor es, entonces, proceder dinámicamente:

```

protected void GridView1_Load(object sender, EventArgs e)
{
    GenericWebPart part = (GenericWebPart) GridView1.Parent;
    part.Title = "Lista de compra completa";
}

protected void GridView2_Load(object sender, EventArgs e)

```

```

{
    GenericWebPart part = (GenericWebPart) GridView2.Parent;
    part.Title = "Los grandes de la lista";
}

```

#### d. Los editores

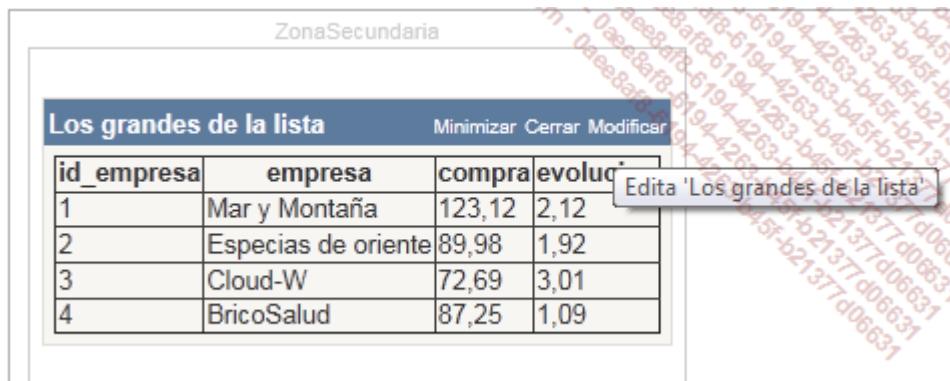
El usuario aprovecha los editores para personalizar elementos de su portal. Un editor está formado por una zona editor (**EditorZone**) que contiene uno o varios elementos de edición:

```

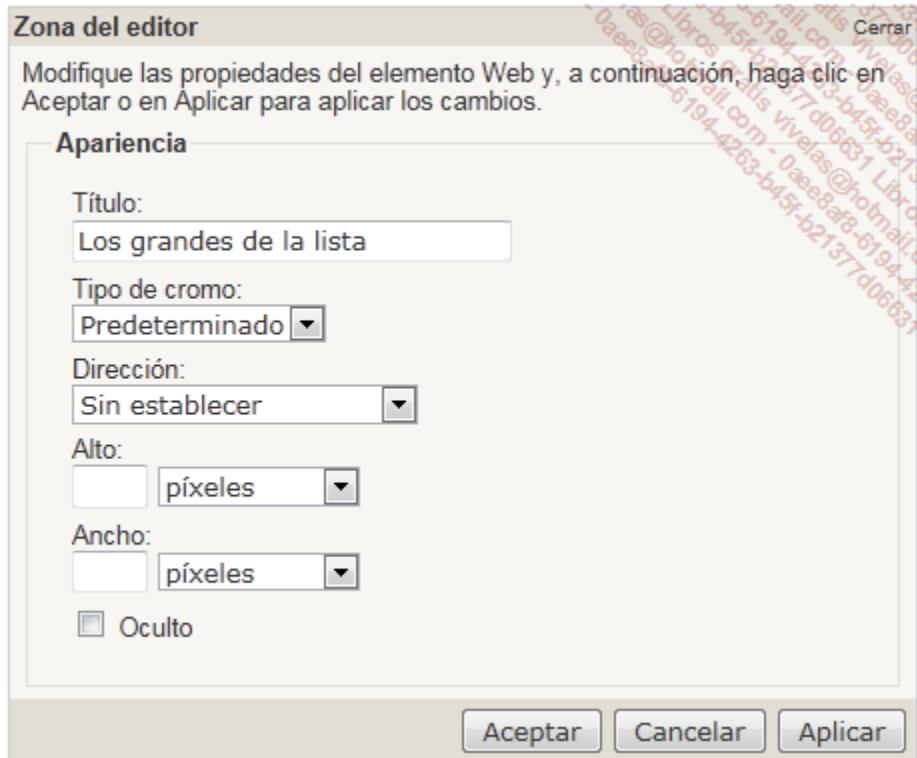
<asp:EditorZone ID="EditorZone1" runat="server">
    <ZoneTemplate>
        <asp:PropertyGridEditorPart ID="PropertyGridEditorPart1" runat="server" />
        <asp:AppearanceEditorPart ID="AppearanceEditorPart1" runat="server" />
    </ZoneTemplate>
</asp:EditorZone>

```

En tiempo de ejecución, será necesario, en primer lugar, escoger el modo **Edit** y, a continuación, utilizar el verbo **Modificar** a nivel del elemento a personalizar:



Después, el usuario modifica las características del elemento que se almacenarán en la base de datos AspNetDb.



## 4. Crear elementos personalizados

Si todo control Web es susceptible de servir de WebPart, es más fácil partir de un componente de usuario o personalizado para controlar su funcionamiento.

### a. Crear un WebPart a partir de un componente de usuario

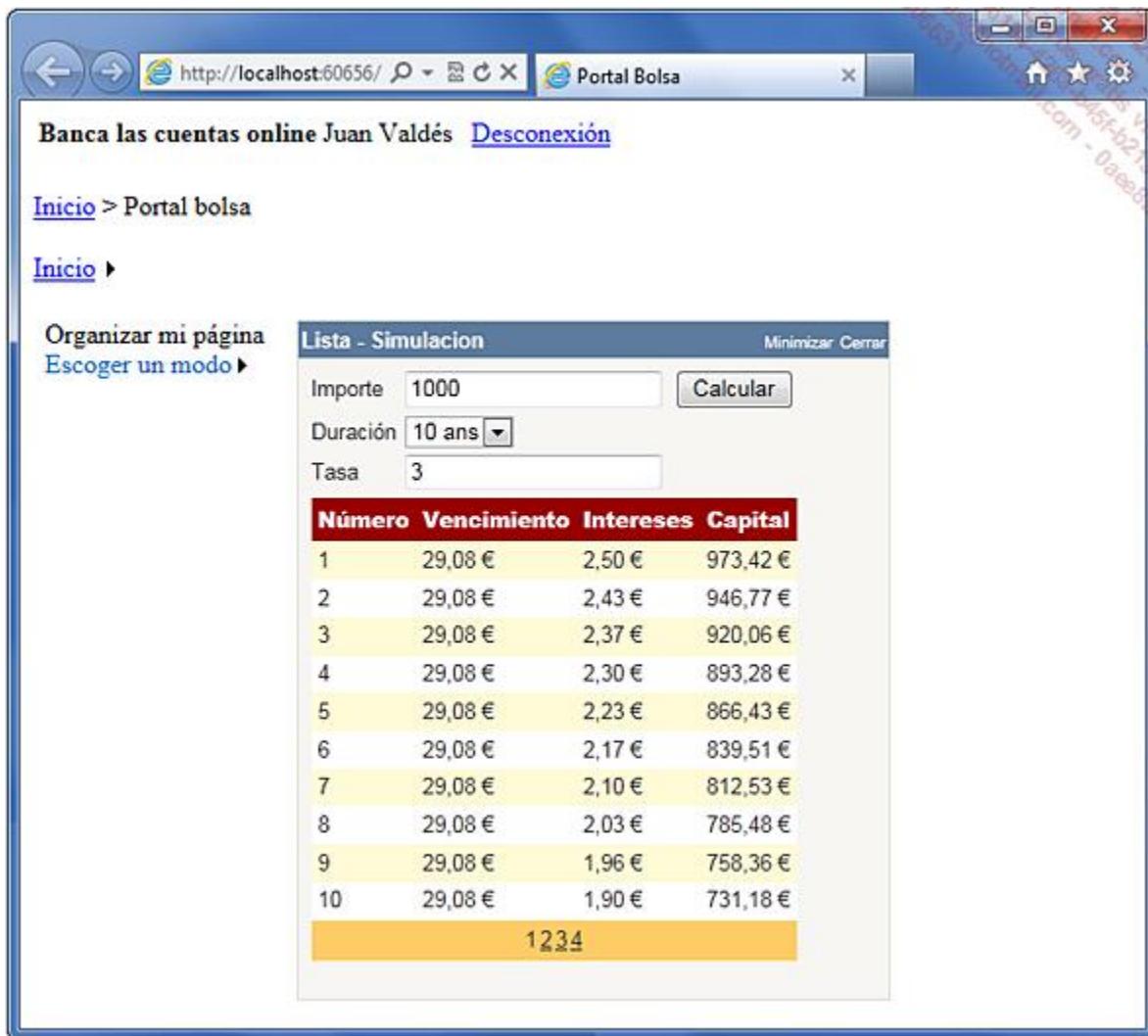
La interfaz `IWebPart` puede implementarse mediante un componente de usuario ASCX. Define las características del WebPart personalizado:

```
public partial class simulacion : System.Web.UI.UserControl, IWebPart
```

Entre las propiedades expuestas por la interfaz, no todas deben implementarse. La siguiente tabla indica su significado:

<code>CatalogImageUrl</code>	Obtiene o define la URL de una imagen que se muestra cuando el WebPart figura en el catálogo.
<code>Description</code>	Obtiene o define la descripción textual del elemento.
<code>Subtitle</code>	Subtítulo del elemento.
<code>Title</code>	Título del elemento.
<code>TitleIconImageUrl</code>	URL de una imagen que se muestra en la barra de título.
<code>TitleUrl</code>	URL hacia la que se redirige al usuario cuando hace clic en el título.

## 5.



### Crear un WebPart personalizado

La clase WebPart se programa como un componente personalizado. Respecto a los controles de usuario que implementan la interfaz WebPart, esta fórmula autoriza la creación de una biblioteca compartida entre varias aplicaciones.

La implementación es similar a la de los controles compuestos. He aquí un ejemplo capaz de realizar una búsqueda en función de un criterio especificado por el usuario:

```
#region SearchWebPart
public class SearchWebPart : WebPart
{
```

```
#region Propiedad Title

public override string Title
{
    get
    {
        if (base.Title == string.Empty)
            return "Grandes negocios en la bolsa";
        else
            return base.Title;
    }
    set
    {
        base.Title = value;
    }
}

#endregion
```

```
public SearchWebPart()
{}
```

```
#region Propiedad CompraMax

private double _CompraMax;

[WebBrowsable(true)]
[Personalizable(PersonalizationScope.User)]
public double CompraMax
{
    get
    {
        return _CompraMax;
    }
```

```
}
```

```
set
{
    _CompraMax = value;
}
```

```

}

#endregion

#region CreateChildControls

private Label lbl_titulo;
private TextBox txt_compra;
private Button cmd_buscar;

protected override void CreateChildControls()
{
    lbl_titulo = new Label();
    lbl_titulo.Text = "Compra max";

    txt_compra = new TextBox();
    txt_compra.Text = CompraMax.ToString();

    cmd_buscar = new Button();
    cmd_buscar.Text = "Buscar";
    cmd_buscar.Click += new EventHandler(cmd_buscar_Click);

    Controls.Add(lbl_titulo);
    Controls.Add(txt_compra);
    Controls.Add(cmd_buscar);
}

#endregion

```

```

#region cmd_buscar_click

private DataSet _lista;

void cmd_buscar_Click(object sender, EventArgs e)
{
    _CompraMax = double.Parse(txt_compra.Text);

    _lista = new DataSet();

    // alimentar la lista
    SqlConnection cx = new SqlConnection(@"initial catalog=banca;
data source=.\SQL2005; integrated security=true");
    SqlCommand cmd = new SqlCommand("select empresa,compra,evolucion
from bolsa where compra<" + _CompraMax, cx);

```

```

cmd.Connection = cx;

SqlDataAdapter da = new SqlDataAdapter(cmd);

da.Fill(_lista);

}

#endregion

}

#endregion

```

Este WebPart no es responsable de la representación de los resultados. Utilizaremos, para ello, un punto de comunicación con otro WebPart.

Para integrarlo, el WebPart debe registrarse como cualquier otro componente personalizado y, a continuación, instanciarse:

```

<%@ Register Namespace="lib6f_cs" TagPrefix="wp" Assembly="lib6f_cs" %>

<asp:WebPartZone ID="zonaPrincipal" runat="server">

    <ZoneTemplate>

        <wp:SearchWebPart id="searcher" runat="server" />

    </ZoneTemplate>

</asp:WebPartZone>

```

#### a. Conectar los elementos

Por último, los Web Parts agregan una nueva dimensión a los programas de Internet destinados a los usuarios. Es posible, ahora, definir el contenido que necesitan ensamblando piezas de aplicación sin estar limitados por la técnica subyacente.

Como complemento a los componentes lógicos que se comparten entre varias aplicaciones (véase el capítulo Los servicios Web WCF y REST), el desarrollador define los servicios de usuario bajo la forma de elementos Web conectables.

#### **Adaptar los elementos al diálogo**

Para alcanzar este objetivo, los Web Parts deben comunicar e intercambiar información basándose en un protocolo independiente a las páginas ASPX. Retomando el ejemplo del elemento SearchWebPart, elaboramos un componente asociado ResultWebPart. Este componente se basa en GridView, responsable de la representación de un DataSet.

Este DataSet está ubicado en el núcleo de la comunicación entre ambos elementos. Existe una interfaz para formalizar los intercambios:

```

public interface IGrandesNegocios

{

    DataSet lista { get; }
}

```

```
}
```

A continuación, el elemento SearchWebPart implementa esta interfaz y utiliza un atributo dedicado al servidor de aplicaciones, el cual carece de una vocación de proveedor de datos:

```
public DataSet lista
{
    get { return _lista; }
}

[ConnectionProvider("Lista")]
public IGrandesNegocios GetCommunicationPoint()
{
    return (IGrandesNegocios)this;
}
```

El elemento ResultWebPart define también un método que invoca al establecimiento de la comunicación y se encarga de explotar la información transmitida:

```
private IGrandesNegocios negocios;

[ConnectionConsumer("Lista")]
public void InitializeProvider(IGrandesNegocios negocios)
{
    this.negocios = negocios;
}
```

## Declarar conexiones estáticas

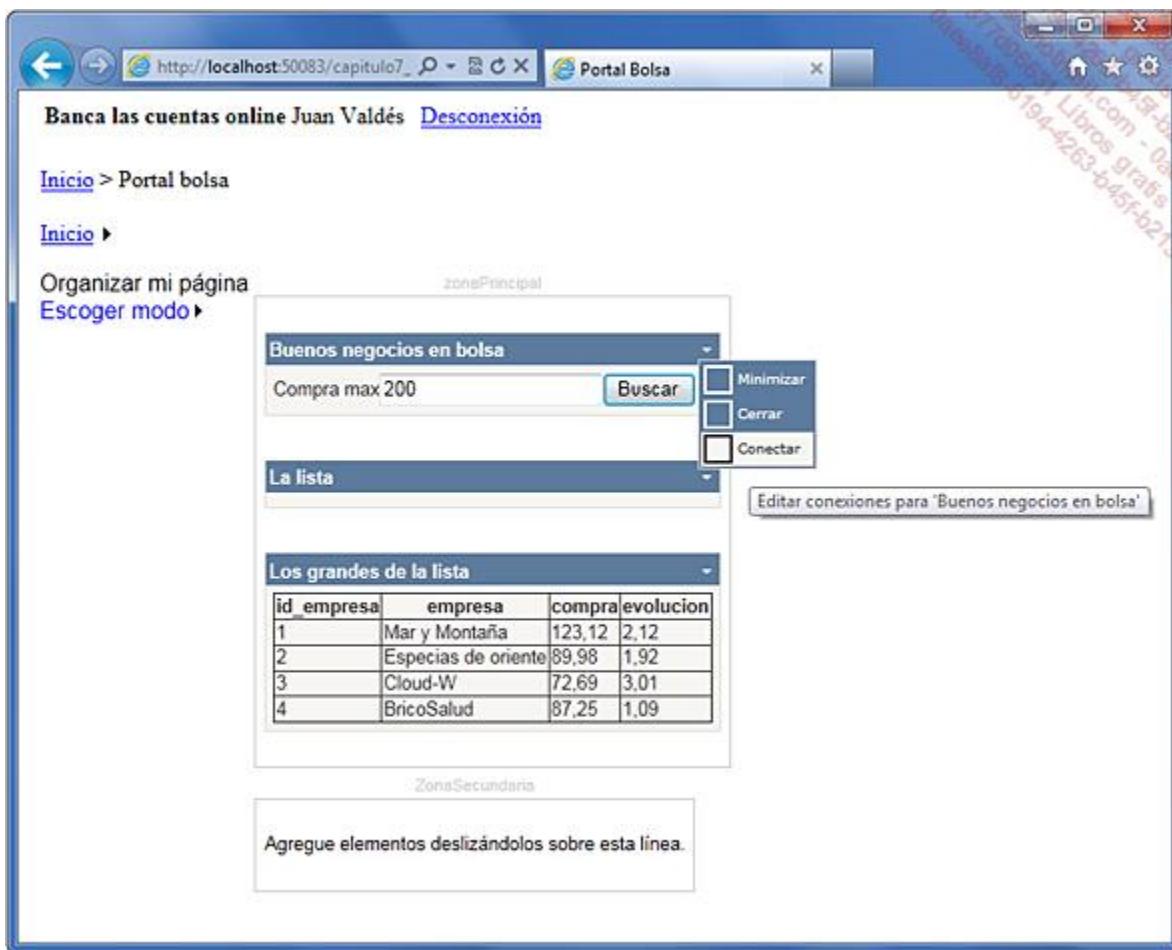
El WebPartManager puede inicializar conexiones entre elementos existentes en la página. Las conexiones correspondientes se denominan estáticas:

```
<asp:WebPartManager ID="MyPartManager" runat="server" >
    <StaticConnections>
        <asp:WebPartConnection ID="ba"
            ConsumerID="resultado" ProviderID="searcher"
            ConsumerConnectionPointID="Lista">
            </asp:WebPartConnection>
    </StaticConnections>
</asp:WebPartManager>
```

## Conexiones dinámicas

Agregando a la página un elemento ConnectionZone, es el usuario el que conecta sus elementos compatibles mediante el contrato definido a nivel de interfaz de comunicación.

Para activar el proceso de conexión, hay que pasar al modo **Connect** y, a continuación, utilizar el verbo **Conectar** sobre el elemento que consume los datos.



Tan sólo queda disfrutar de esta asociación de elementos:

[←](#) [→](#) [http://localhost:50083/capitulo7\\_](http://localhost:50083/capitulo7_) [↻](#) [✖](#) [Portal Bolsa](#) [✖](#)

Banca las cuentas online Juan Valdés [Desconexión](#)

[Inicio](#) > Portal bolsa

[Inicio](#) ▶

Organizar mi página

Escoger modo ▶

zonaPrincipal

**Buenos negocios en bolsa**

Compra max 100

**La lista**

empresa	compra	evolucion
Especias de oriente	89,98	1,92
Cloud-W	72,69	3,01
BricoSalud	87,25	1,09

**Los grandes de la lista**

id_empresa	empresa	compra	evolucion
1	Mar y Montaña	123,12	2,12
2	Especias de oriente	89,98	1,92
3	Cloud-W	72,69	3,01
4	BricoSalud	87,25	1,09

www.FreeLibros.me

# Los servicios Web WCF

Un equipo que ejecute el servidor Web IIS y esté equipado con el framework ASP.NET no tiene la única vocación de albergar sitios Web. Es, también, capaz de ofrecer servicios de aplicación distribuidos a otros equipos.

Entre los protocolos disponibles, SOAP se impone a la hora de conectar entornos heterogéneos. El uso intensivo del formato XML ha contribuido enormemente, y es uno de los puntos fuertes de la tecnología .NET, los servicios Web SOAP implementados con ASP.NET son una referencia.

Dicho esto, Microsoft apuesta con WCF (*Windows Communication Foundation*) por una unificación de sus middlewares de comunicación. El método consiste en desarrollar la interoperabilidad aumentando el número de servicios de alto nivel integrados en dicha infraestructura de comunicaciones. De este modo, .NET Remoting es capaz de "modular" SOAP, y los servicios Web WFC integran de forma estándar las extensiones WS.

## 1. El dialecto común SOAP

Los servicios Web (SOAP) utilizan un protocolo Internet para asegurar la comunicación con sus aplicaciones cliente. A menudo, se trata del protocolo HTTP, de ahí la denominación servicios Web.

Es posible aplicar otros protocolos tales como SNMP o FTP.

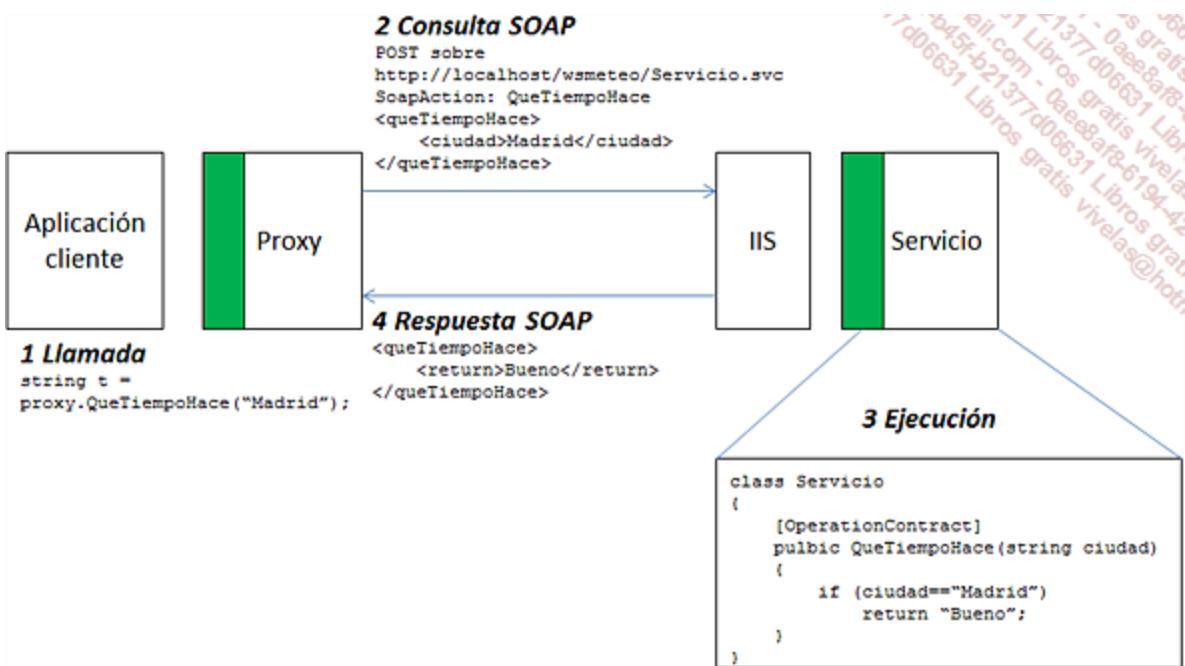
El cliente utiliza una interfaz de programación llamada proxy. Ésta incluye las funciones de extensión expuestas por el servicio Web. La tecnología SOAP prevé la definición de esta interfaz con un formato independiente de los lenguajes de programación, WSDL (*Web Service Description Language*). Es decir, el contrato WSDL de un servicio se describe, por completo, con XML.

El proxy se encarga, para la aplicación cliente, de codificar las llamadas de funciones en consultas con formato SOAP, las cuales se transmiten por la red. A la vuelta, el proxy decodifica las respuestas SOAP recibidas del servicio Web. De este modo, la programación del cliente no se realiza jamás directamente con referencia a la red, lo que facilita su evolución.



El documento WSDL se utiliza solamente para generar el proxy. La operación se realiza por línea de comandos - mediante la herramienta **svcutil.exe** - o mediante el asistente gráfico **Agregar referencia de servicio**.

El siguiente esquema detalla la cronología de intercambios entre el cliente y el servicio Web.



En primer lugar, el cliente utiliza el proxy para invocar a la función `QueTiempoHace()`.

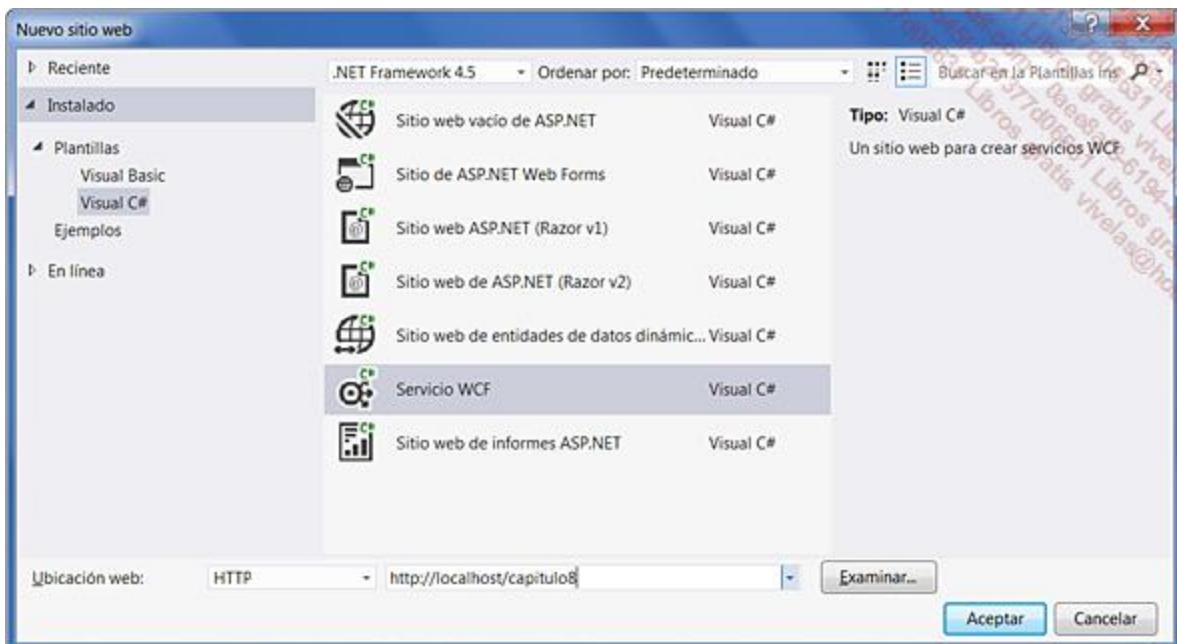
El proxy contacta al servidor Web localhost y realiza un POST HTTP en la URI `wsmeteo/Servicio.svc`. El comando `SoapAction` indica la función de destino y provee los parámetros necesarios mediante un mensaje SOAP escrito en XML.

El servidor de aplicaciones ASP.NET identifica la naturaleza de la clase `Servicio`, decodifica la consulta SOAP e invoca al método `QueTiempoHace()` pasándole como dato los valores de los parámetros.

El resultado de la ejecución del método se devuelve conforme al formato SOAP, en XML. El proxy se encarga de decodificar, para el cliente, el valor correspondiente.

## 2. Crear un servicio Web WCF

Un servicio Web se crea como cualquier otra aplicación ASP.NET, mediante el comando **Archivo - Nuevo - Sitio Web**. Un asistente específico inicializa la carpeta virtual con un archivo `Service.svc`, su archivo de código `Service.cs` ubicado en la carpeta `App_Code` y su interfaz funcional `IService.cs`.



### a. Implementación del servicio

Estudiamos, a título de ejemplo, un servicio Web que expone funciones de directorio - lista, búsqueda. Comenzamos, pues, por dos clases que describen una entrada en la agenda y su "DAO" correspondiente.

```
using System;
using System.Runtime.Serialization;

[DataContract]
public class Persona
{
    [DataMember]
    public int idPersona { get; set; }

    [DataMember]
    public string nombre { get; set; }

    [DataMember]
    public string email { get; set; }

    public Persona()
    {
```

```
        idPersona = 0;
        nombre = string.Empty;
```

```

        email = string.Empty;
    }

    public Persona(int idPersona, string nombre, string email)
    {
        this.idPersona = idPersona;
        this.nombre = nombre;
        this.email = email;
    }
}

public class PersonaDAO
{
    public PersonaDAO()
    {

    }

    public Persona[] GetData()
    {
        return new Persona[]
        {
            new Persona(1, "Juan Valdés", "juan.valdes@miserables.es"),
            new Persona(2, "Luis Antonio", "luisantonio@theclub.org")
        };
    }

    public Persona GetPersonaByID(int idPersona)
    {
        var q = from p in GetData() where p.idPersona ==
idPersona select p;
        try
        {
            return q.ToList()[0];
        }
        catch { }

        return new Persona(0, "Desconocido", "Sin email");
    }
}

```

Destacamos que la clase Persona está cualificada por atributos que controlan su serialización [DataContract] y [DataMember]. Estos nuevos atributos, introducidos por WFC, unifican los distintos atributos implicados en el proceso de serialización de los "antiguos" middleware distribuidos .NET Remoting y SOAP en versión ASMX. La clase DAO no tiene, en sí, nada de particular, salvo que utiliza LINQ por comodidad a la hora de realizar las consultas en la base de objetos.

Tras la creación del proyecto de servicio Web, Visual Studio genera una interfaz funcional IService. La renombraremos por IAgenda y completaremos su definición. Aquí, de nuevo, los atributos indican a la infraestructura WFC qué métodos son adecuados para una exposición como servicio Web:

```
[ServiceContract]
public interface Iagenda
{
    [OperationContract]
    Persona[] SelectAllPersona();

    [OperationContract]
    Persona GetPersonaByID(int idPersona);
}
```

La clase Service también se renombra por AgendaServicio, e implementa, lógicamente, la interfaz anterior:

```
public class AgendaServicio : Iagenda
{
    #region IAgenda Miembros

    public Persona[] SelectAllPersona()
    {
        return new PersonaDAO().GetData();
    }

    public Persona GetPersonaByID(int idPersona)
    {
        return new PersonaDAO().GetPersonaByID(idPersona);
    }

    #endregion
}
```

A continuación, hay que realizar ciertos cambios en el archivo Servicio.svc para hacer que el sistema generado por Visual Studio sea coherente:

```
<%@ ServiceHost Language="C#" Debug="true" Service="AgendaServicio"
CodeBehind="~/App_Code/AgendaServicio.cs" %>
```

El archivo Web.config debe, también, actualizarse, incluso si el conjunto de parámetros referidos a la definición de servicios Web podrían definirse como atributos.

```
<system.serviceModel>
    <services>
        <service name="AgendaServicio" behaviorConfiguration=
            "AgendaServicioBehavior">
            <!-- Service Endpoints -->
            <endpoint address="" binding="basicHttpBinding" contract=
                "Iagenda">
                <identity>
                    <dns value="localhost"/>
                </identity>
            </endpoint>
            <endpoint address="mex" binding="mexHttpBinding" contract=
                "IMetadataExchange"/>
        </service>
    </services>
    <behaviors>
        <serviceBehaviors>
            <behavior name="AgendaServicioBehavior">
                <!-- Para evitar la divulgación de información sobre los
                    metadatos, defina el siguiente valor a falso y suprima
                    el punto de ruptura de los metadatos que aparece antes
                    del despliegue -->
                <serviceMetadata httpGetEnabled="true"/>
            </behavior>
        </serviceBehaviors>
    </behaviors>

```

```
        <!-- Para recibir el detalle de la excepción de error
            con el objetivo de depurar, defina el siguiente valor a true.
            Definalo a falso antes del despliegue para evitar
            divulgar la información de la excepción -->
        <serviceDebug includeExceptionDetailInFaults="false"/>
    </behavior>
</serviceBehaviors>
</behaviors>
```

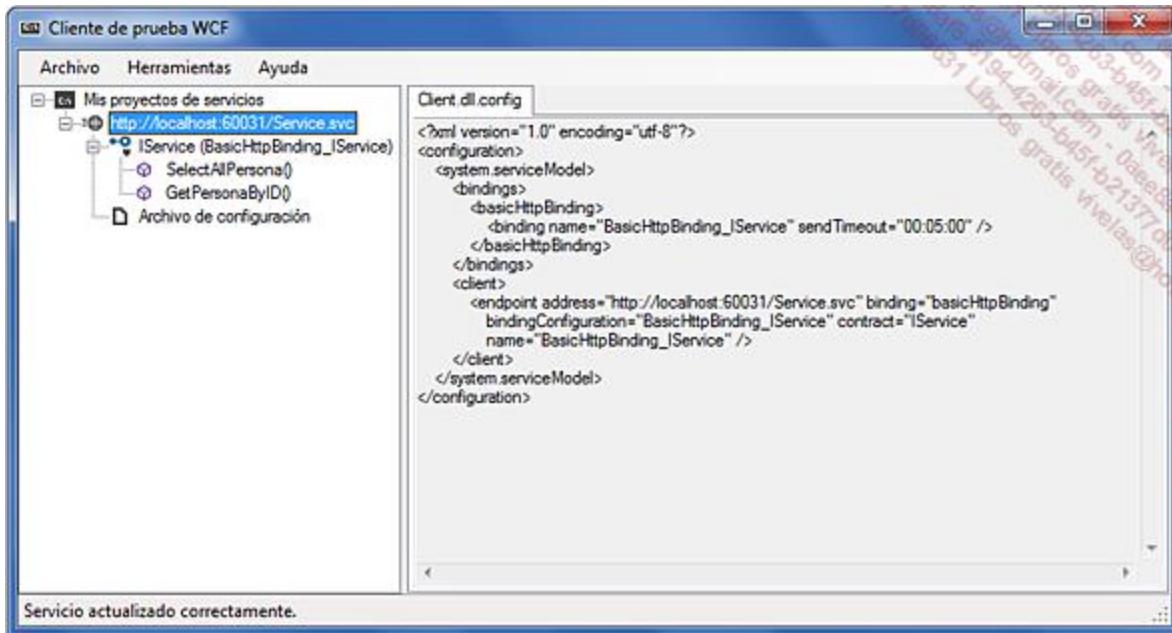
```
</system.serviceModel>
```

Entre estos parámetros, el atributo binding adquiere una importancia particular: decide el dialecto utilizado por el punto de acceso del servicio Web.

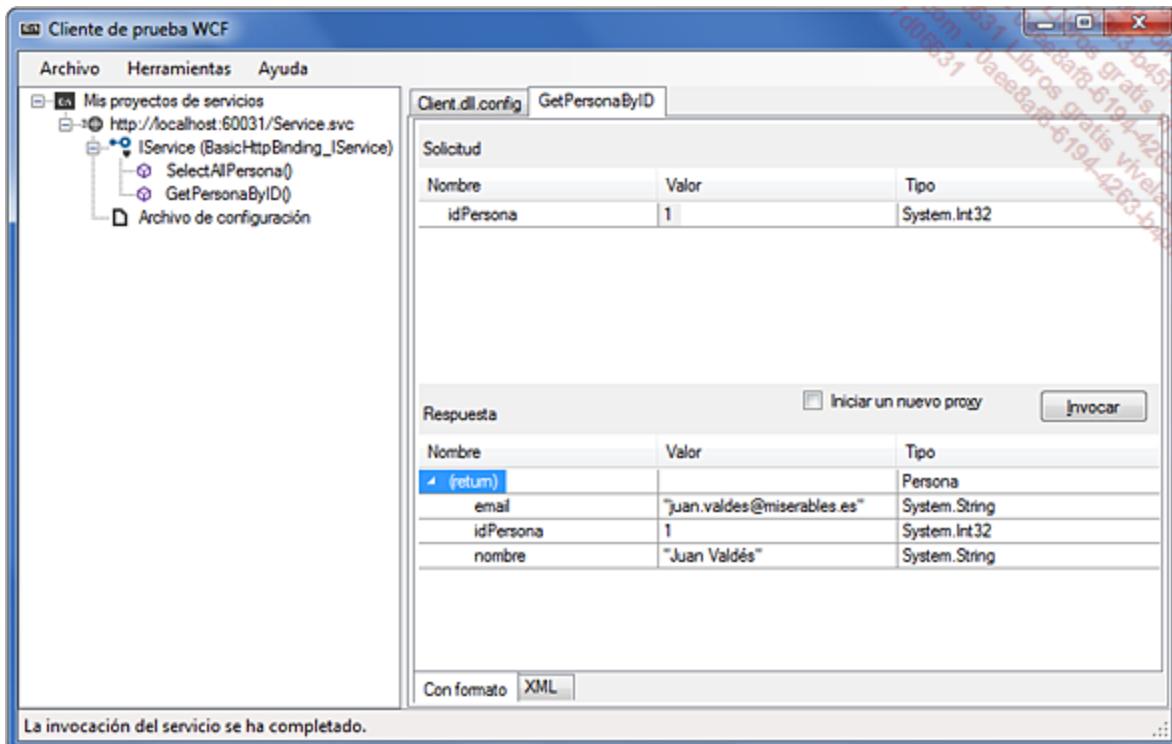
<b>basicHttpBinding</b>	El punto de acceso reconoce el dialecto SOAP.	Por defecto, Visual Studio genera un binding de tipo wsHttpBinding, reforzado con seguridad. Aplicamos deliberadamente el binding basicHttpBinding para guiar al lector, habituado a los servicios Web SOAP implementados por puntos de acceso ASMX (ASP.NET 2005).
<b>wsHttpBinding</b>	El punto de acceso reconoce el dialecto SOAP con extensión (WS-*).	
<b>mexHttpBinding</b>	El punto de acceso utiliza la extensión WS-metadata para intercambiar (publicar) sus características.	

### b. Prueba del servicio

El arranque de una sesión de depuración en el servicio Service.svc provoca la aparición de un cliente.



El cliente de prueba es capaz de invocar a un método expuesto pasándole parámetros:

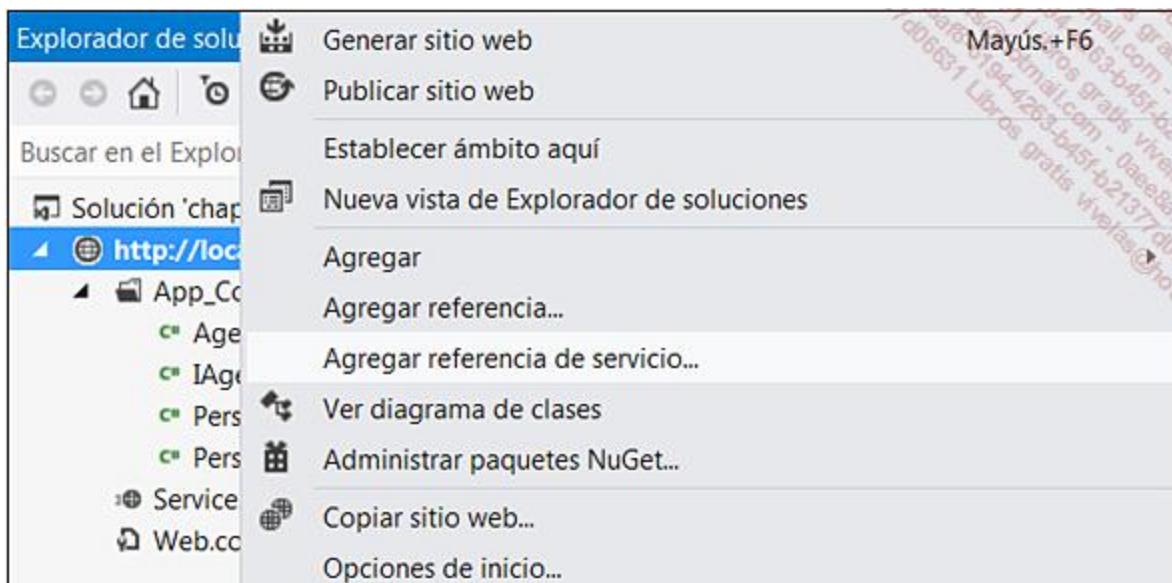


### 3. Consumir un servicio Web

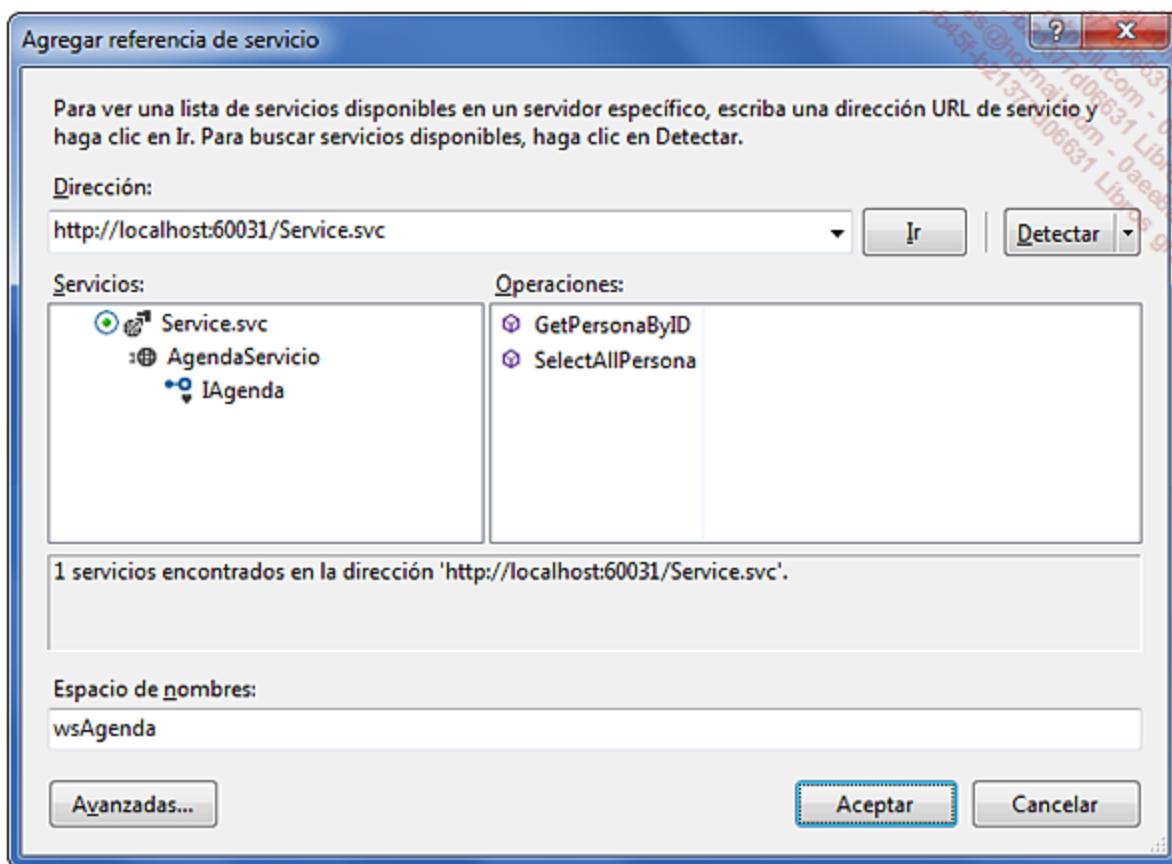
Todo tipo de software (incluido Java, PHP, C++...) es capaz de consumir un servicio Web SOAP. Vamos a crear una aplicación ASP.NET destinada a probar y consumir nuestro servicio Web agenda.

#### a. Generación del proxy

Con Visual Studio 2012, hay dos formas de generar un proxy de servicio Web. El menú contextual **Agregar referencia** se utiliza por compatibilidad con Visual Studio 2005 y, en particular, con los servicios Web ASMX. Utilizaremos, preferentemente, el menú contextual **Agregar referencia de servicio** que ofrece un mayor control sobre el proxy generado, además de tener en cuenta los servicios WFC.

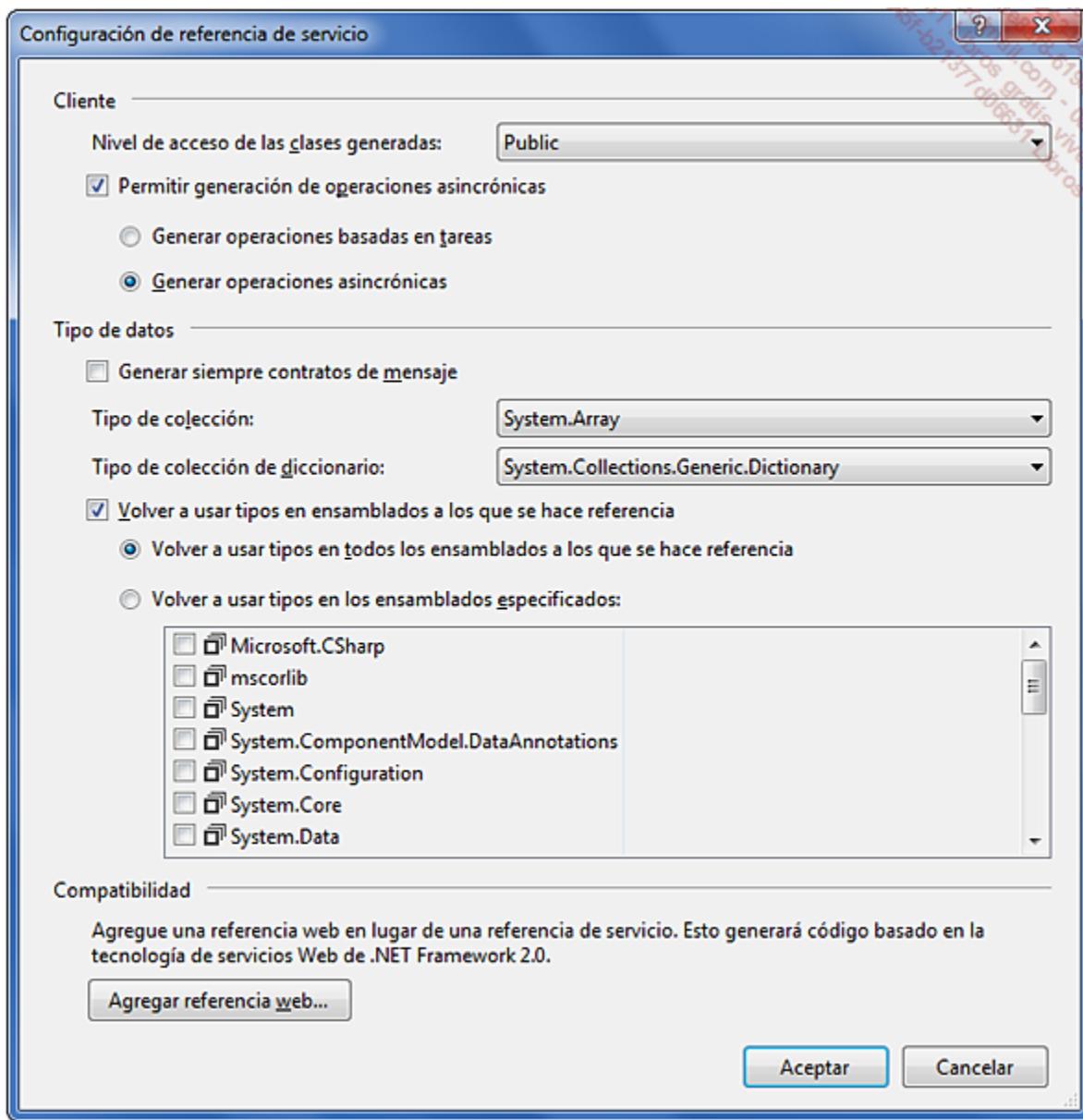


El botón **Detectar** puede ser útil si el servicio Web está ubicado en la misma solución. Esto evita tener que introducir la URL de contrato de servicio WSDL.



Escogemos generar el proxy en el espacio de nombres wsAgenda, decisión completamente arbitraria y sin efecto alguno en las funciones soportadas por el proxy.

El botón **Avanzadas** permite activar las operaciones asíncronas. En realidad, sólo el proxy es asíncrono. Un dispositivo basado en delegados y eventos deja el control al código que se invoca durante la ejecución de una consulta al servicio remoto.



La herramienta svcutil.exe se invoca, de hecho, en el servidor de aplicaciones ASP.NET para crear un proxy que responde a las características solicitadas. No hay más que instanciarlo para poder invocar a nuestro servicio remoto.

### b. Llamada síncrona

Para probar nuestro servicio de agenda, utilizamos un componente LinqDataSource cuyo evento Selecting se controla para interrogar al servicio Web:

```
protected void linq_persona_Selecting(object sender,
```

```
LinqDataSourceSelectEventArgs e)
{
    // recuperar el identificador
    int idPersona = 0;
```

```

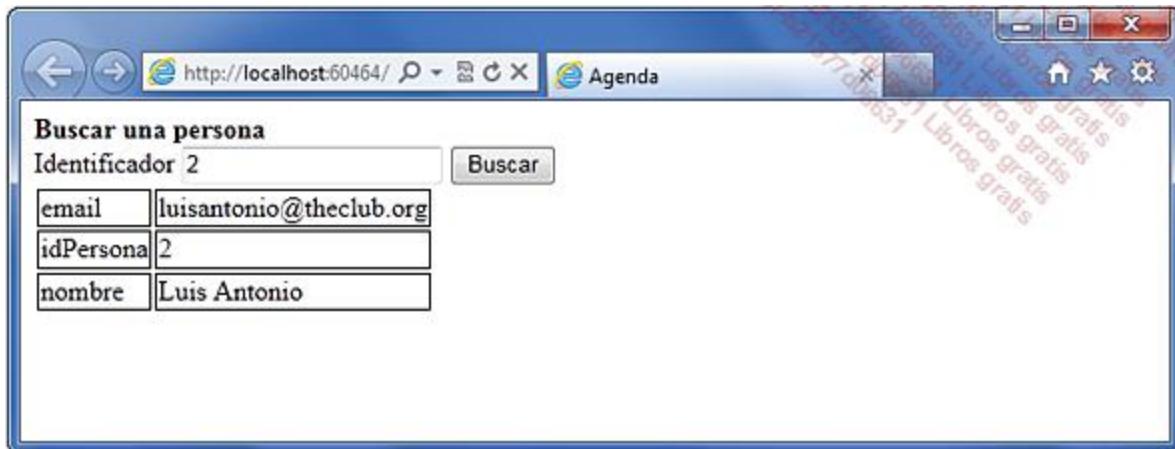
try
{
    idPersona = int.Parse(txt_id.Text);
}
catch { }

// consultar el servicio de agenda
wsAgenda.AgendaClient proxy = new wsAgenda.AgendaClient();
wsAgenda.Persona p = proxy.GetPersonaByID(idPersona);

// formatea el resultado para LinqDataSource
List<wsAgenda.Persona> sel=new List<wsAgenda.Persona>();
sel.Add(p);
e.Result = sel;
}

protected void cmd_find_Click(object sender, EventArgs e)
{
    dt_persona.DataBind();
}

```



### c. Llamada asíncrona

La consulta a un servicio remoto requiere un tiempo que puede ser largo e impactar en el rendimiento de la aplicación. La herramienta svcutil.exe genera una versión asíncrona de cada método expuesto en el servicio Web.

Tomemos, como ejemplo, una página que debe mostrar todas las entradas de la agenda en unGridView. La espera a que se procese la consulta, la recepción de la respuesta, podrían ejecutarse "como tarea de fondo". He aquí el código del DAO enlazado a un componenteObjectDataSource encargado de alimentar el GridView:

```
public class PersonaDAO
```

```

{
    public PersonaDAO()
    {

    }

    /// <summary>
    /// Este método se invoca desde ObjectDataSource
    /// </summary>
    /// <returns></returns>
    public wsAgenda.Persona[] GetData()
    {
        if (HttpContext.Current.Cache["agenda"] == null)
        {
            InitData();
            return null; // volver más tarde, sin datos de momento
        }

        // los datos se alojan en caché
        return (wsAgenda.Persona[]) HttpContext.Current.Cache["agenda"];
    }

    /// <summary>
    /// Ejecuta la consulta SOAP como tarea de fondo
    /// </summary>
    private void InitData()
    {

```

```

        wsAgenda.AgendaClient proxy = new wsAgenda.AgendaClient();
        proxy.SelectAllPersonaCompleted += new
        EventHandler<wsAgenda.SelectAllPersonaCompletedEventArgs>(proxy_
        SelectAllPersonaCompleted);
        proxy.SelectAllPersonaAsync();

    }

    /// <summary>
    /// callback invocada por el proxy cuando se ejecuta la consulta
    /// </summary>

```

```

/// <param name="sender"></param>
/// <param name="e"></param>
void proxy_SelectAllPersonaCompleted(object sender,
wsAgenda.SelectAllPersonaCompletedEventArgs e)
{
    HttpContext.Current.Cache.Add("agenda",
        e.Result,
        null,
        DateTime.Now.AddSeconds(60),
        TimeSpan.Zero,
        System.Web.Caching.CacheItemPriority.Normal, null);
}
}

```

Tras la primera llamada al método GetData(), es muy probable que los datos no existan. El método devuelve null y no se muestra ninguna fila en el GridView. Hemos tomado, no obstante, la precaución de agregar a la página un componente Ajax Timer que solicita la actualización (.DataBind) del GridView:

```

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="agenda.aspx.cs"
Inherits="agenda" Async="true" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">

```

```

    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:GridView ID="gv_personas" runat="server"
                        DataSourceID="ods_agenda" AutoGenerateColumns="False">
                        <Columns>

```

```

        <asp:BoundField DataField="email"
HeaderText="email" SortExpression="email" />

        <asp:BoundField DataField="idPersona"
HeaderText="idPersona"
SortExpression="idPersona" />

        <asp:BoundField DataField="nombre"
HeaderText="nombre" SortExpression="nombre" />

    </Columns>
</asp:GridView>

<asp:ObjectDataSource ID="ods_agenda"
runat="server" SelectMethod="GetData"
TypeName="PersonaDAO"></asp:ObjectDataSource>

<br />

<asp:Timer ID="Timer1" runat="server"
Interval="5000" ontick="Timer1_Tick">
</asp:Timer>
<br />

</ContentTemplate>

</asp:UpdatePanel>

<br />
<br />

</div>

```

```

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

```

Observe, también, la presencia del atributo de página Async, que indica al motor ASP.NET que la página realiza operaciones asíncronas.

## Los servicios Web REST

Los servicios Web SOAP (y WS) son los nuevos pilares de la programación distribuida. Intervienen entre las capas de negocio de los servidores de aplicaciones. Pero su programación encaja bastante mal con las interfaces gráficas que requieren reactividad cuando evoluciona una interfaz funcional. Dicho de otro modo, el hecho de agregar una columna no debería producir toda una retahíla de operaciones de actualización del componente distribuido, de sus interfaces, de los archivos de configuración...

WCF tiene en cuenta los servicios Web REST, basados en URI (fragmentos de URL) para controlar su ejecución.

Un servicio Web REST presenta una implementación similar a los demás servicios, con algunas particularidades:

### binding

El servicio Web REST utiliza un binding (dialecto) de tipo webHttpBinding.

```
<endpoint address="" binding="basicHttpBinding" contract="IService">
    <identity>
        <dns value="localhost" />
    </identity>
</endpoint>
```

### WebGet

El atributo WebGet indica que se ha escogido la URI para invocar un método de servicio:

```
[OperationContract]
[WebGet(UriTemplate="personas/{idPersona}",
ResponseFormat=WebMessageFormat.Json)]
    Persona GetPersonaByID(string idPersona);
```

### Formato de salida

El formato de salida es, habitualmente, JSON (*JavaScript Object Notation*), más flexible de analizar que XML para código ejecutado del lado cliente.

### Compatibilidad IIS

El punto de acceso utiliza una conexión mediante la pila (stack) de IIS:

```
<endpointBehaviors>
    <!-- Comportamiento del servicio REST -->
    <behavior name="ServicePersonaBehavior">
```

```

<webHttp />
</behavior>

</endpointBehaviors>

```

## 1. Implementación de un servicio REST

Es, entonces, la interfaz funcional la que menciona las características del servicio REST, gracias al atributo WebGet:

```

[ServiceContract]
public interface IservicePersona
{
    [OperationContract]
    Persona[] SelectAllPersona();

    [OperationContract]
    [WebGet(UriTemplate="personas/{idPersona}", ResponseFormat=
    WebMessageFormat.Json)]
    Persona GetPersonaByID(string idPersona);
}

```

La implementación indica que se requiere modo de compatibilidad con ASP.NET (dicho de otro modo, las llamadas transitan por la pila IIS).

```

[AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Required)]
public class ServicePersona : IservicePersona
{
    #region IServicePersona Membres

    Persona[] IservicePersona.SelectAllPersona()
    {
        return new PersonaDAO().GetData();
    }

    Persona IServicePersona.GetPersonaByID(string idPersona)
}

```

```

{
    int id = int.Parse(idPersona);
    return new PersonaDAO().GetPersonaByID(id);
}

```

```
#endregion  
}
```

## 2. Utilizando un servicio REST

Proponemos, como experiencia, un cliente REST basado en un proxy AJAX (XML HTTP Request).

```
<html xmlns="http://www.w3.org/1999/xhtml">  
  
<head runat="server">  
    <title></title>  
    <script>  
  
        // objeto XML HTTP Request indispensable en la llamada al servicio REST  
        var xhr = null;  
  
        function getXhr() {  
            if (window.XMLHttpRequest)  
                xhr = new XMLHttpRequest();  
            else {  
                if (window.ActiveXObject)  
                    try {  
                        xhr = new ActiveXObject("Msxml2.XMLHTTP");  
                    }  
                    catch (e) {  
                        xhr = new ActiveXObject("Microsoft.XMLHTTP");  
                    }  
                else {  
                    alert("Su navegador no soporta los objetos XMLHttpRequest");  
                    xhr = false;  
                }  
            }  
        }  
  
        // este método invoca al servicio desde su URL  
  
        function serviceAjax(url, callback) {  
            getXhr();  
            xhr.onreadystatechange =  
                function() {  
                    if (xhr.readyState == 4 && xhr.status == 200)  
                {
```

```

        f = callback + "(xhr)";
        eval(f);
    }
};

xhr.open("GET", url, true);
xhr.send(null);
}

// función callback que se invoca tras ejecutar la consulta
function getPersonaByIDCallback(xhr) {
    document.getElementById("agenda").innerHTML =
xhr.responseText;
}

// busca la ficha de la persona 1
function getPersona() {
    var idPersona = 1;
    url = "http://localhost/cap8/ServicioPersona.svc/personas/" +
idPersona;
    serviceAjax(url, "getPersonaByIDCallback");
}

```

```

</script>
</head>
<body>
    <form id="form1" runat="server">
        <input type="button" onclick="getPersona()" value="Preguntar" />
        <div id="agenda">

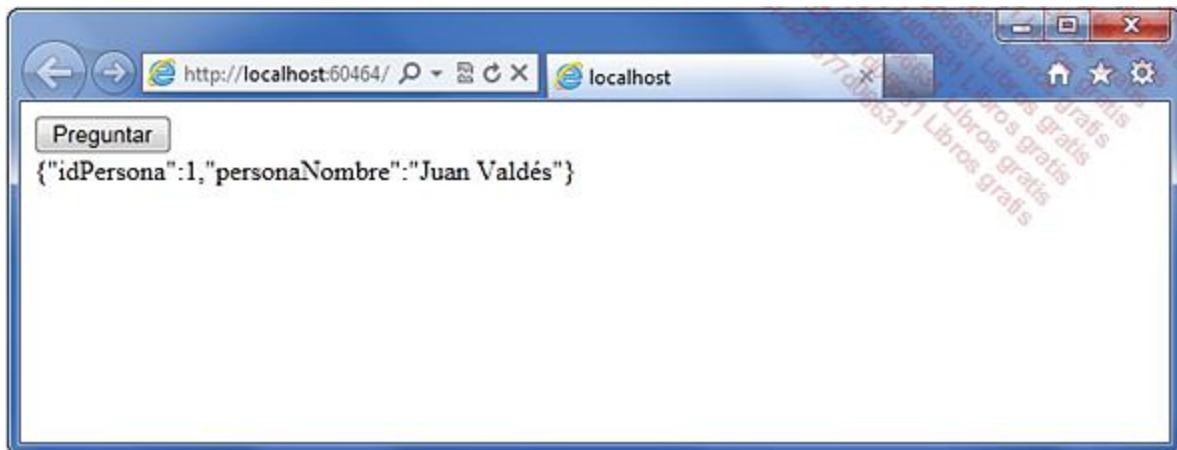
```

```

    </div>
    </form>
</body>
</html>

```

Al ejecutar la consulta, se muestran los resultados JSON sin decodificar en la sección "agenda":



## Configuración

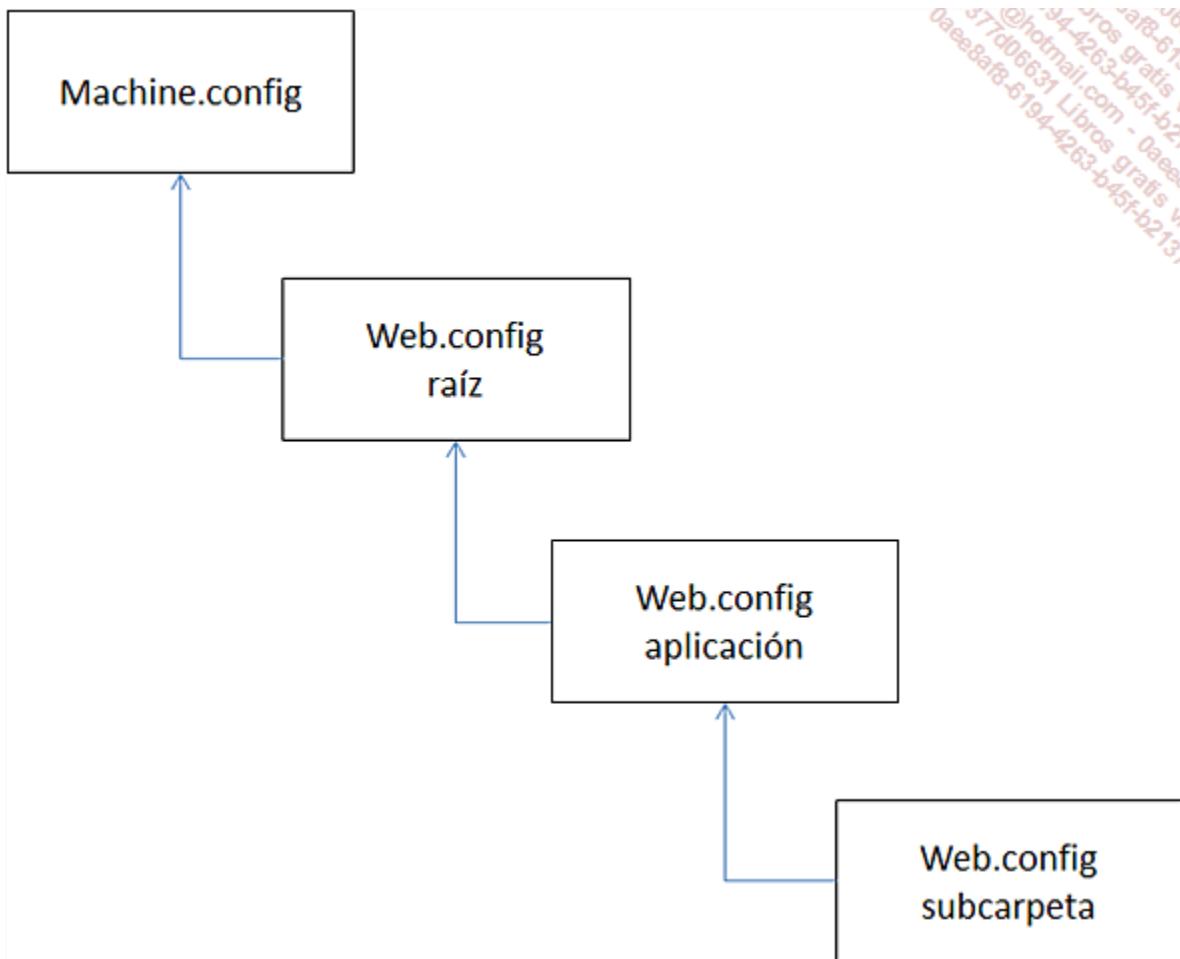
### 1. Herencia en la configuración

Cada ordenador equipado con el framework .NET dispone de un archivo central de configuración **Machine.config**. Este archivo, ubicado en la carpeta C:\Windows\Microsoft.NET\Framework\Version\Config, provee al CLR y a todos los ensamblados parámetros de todo tipo.

Las aplicaciones .NET Windows poseen su propio archivo de configuración, opcional. Se trata, en Visual Studio, del archivo **App.config** que se convierte en el archivo **NombreDelEnsamblado.exe.config** tras la compilación.

Hemos mencionado, a menudo, el archivo **Web.config** para las aplicaciones ASP.NET. En cierta medida, este archivo juega el mismo rol que App.config. Su funcionamiento es, no obstante, más fino: cada carpeta virtual puede declarar un archivo Web.config que hereda de los parámetros definidos en el Web.config de nivel superior.

Todas las aplicaciones ASP.NET heredan, también, valores provistos por el archivo raíz **Web.config** (también ubicado en la carpeta Config), que hereda, él mismo, del archivo **Machine.config**. Esto es lo que se conoce como herencia en la configuración.



Los parámetros de `Machine.config` tienen preferencia. El atributo `allowDefinition` controla la reconfiguración de una sección mediante una aplicación ASP.NET. Los posibles valores del atributo son:

Everywhere (valor por defecto)	Autoriza la configuración de la sección en: <code>Machine.config</code> , Archivo <code>Web.config</code> raíz, Archivo <code>Web.config</code> de una aplicación, Carpeta virtual y Subcarpeta física de una aplicación	2.
<code>MachineToApplication</code>	<code>Machine.config</code> , Archivo <code>Web.config</code> raíz y Archivo <code>Web.config</code> de una aplicación	
<code>MachineToWebRoot</code>	<code>Machine.config</code> y Archivo <code>Web.config</code> raíz	
<code>MachineOnly</code>	<code>Machine.config</code>	

## Configuración de pruebas y de producción

El programador debe prever varios perfiles de compilación y varios archivos de configuración para adaptarse a las distintas etapas del proceso de construcción de una aplicación.

### a. El administrador de configuración de Visual Studio

Como hemos mencionado en el primer capítulo, Visual Studio crea varios perfiles de compilación para cada proyecto, incluidos los proyectos de aplicación Web.

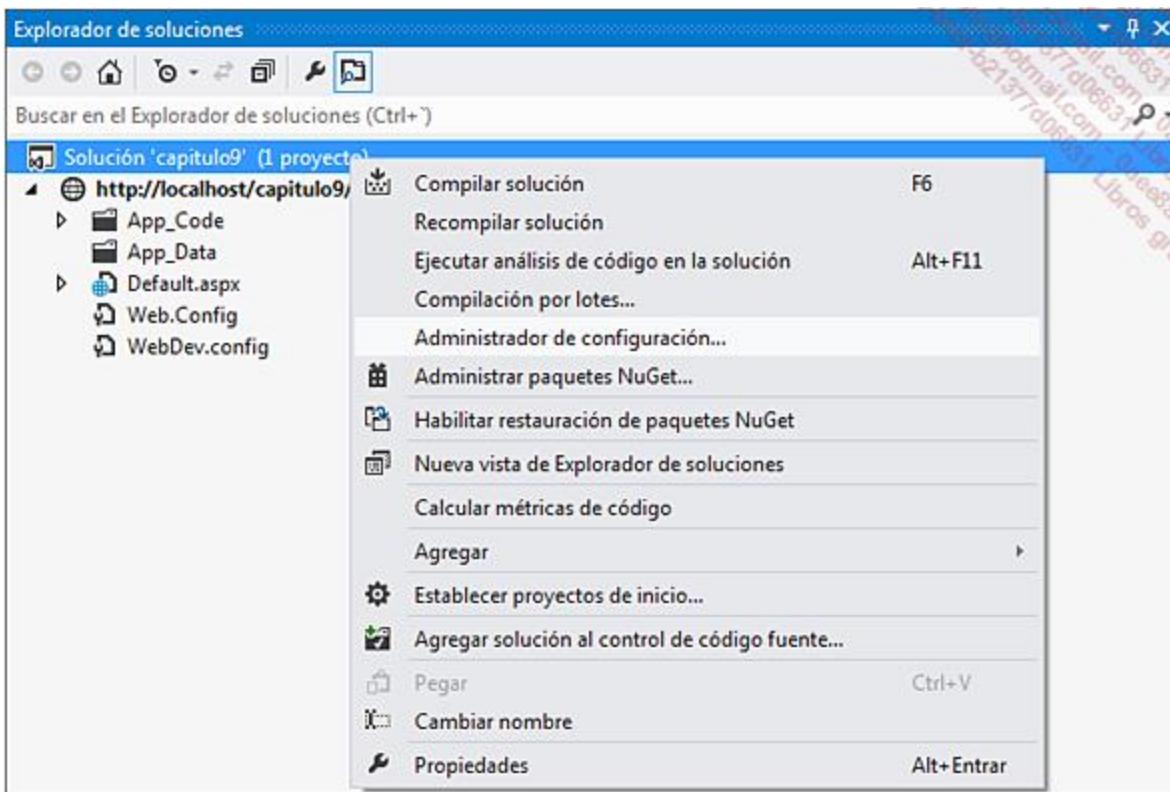
Por defecto, el compilador funciona en modo Debug, lo que significa que genera un archivo de documentación para la depuración (.pdb), que la directiva #DEBUG está definida y, de forma general, que un error de ejecución se explique con claridad al desarrollador encargado de la actualización y reparación del programa.

La autorización del depurador para una aplicación ASP.NET se activa en el archivo Web.config:

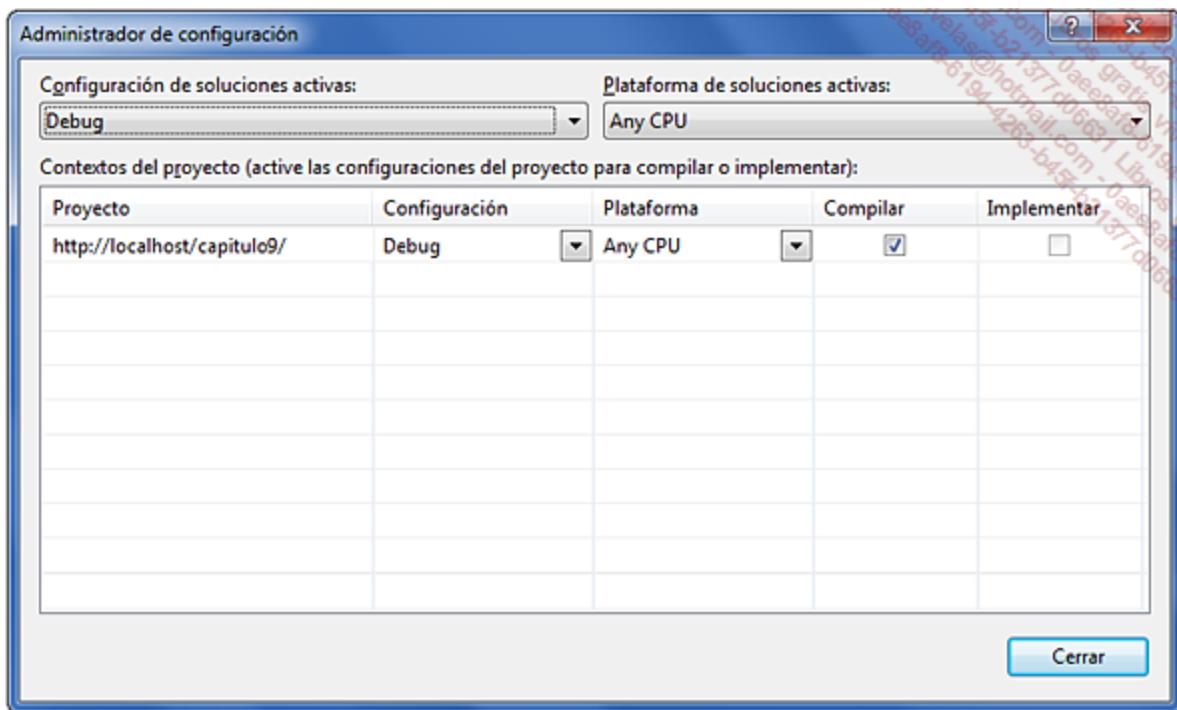
```
<compilation debug="true" />
```

El modo Release produce ensamblados más compactos - y, por tanto, mejor adaptados al despliegue - aunque los eventuales errores se reportarán sin detalle. Más allá de estar seguro de su implementación, el desarrollador habrá podido proteger su programa mediante estructuras de control de errores de tipo try... catch.

El paso de un modo de compilación a otro, o la creación de nuevos perfiles, se realiza mediante el menú contextual **Administrador de configuración** aplicado a la solución abierta.

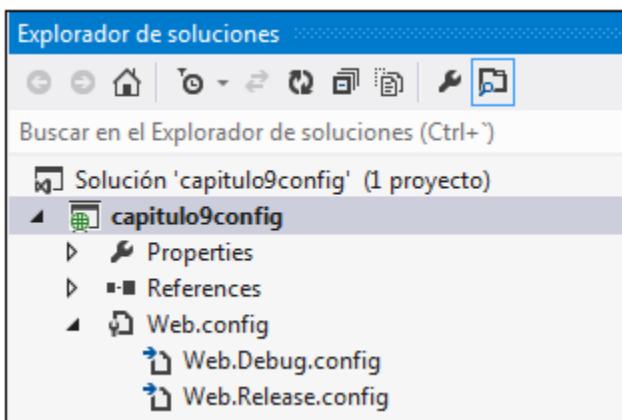


Según las ediciones de Visual Studio y los modelos de proyecto, el perfil Release no siempre existe por defecto. El programador puede crearlo fácilmente.



### b. Varios archivos de configuración Web.config

Visual Studio genera dos versiones del archivo Web.config activos para la configuración debug o release:



### c. Las páginas de error del archivo Web.config

De cara a hacer frente a los errores imprevistos, es posible definir redirecciones para cada código de estado HTTP (404, 401, 501...).

```
<?xml version="1.0"?>

<configuration>
    <!-- WebDev.config -->
```

```
    <system.web>
        <customErrors mode="On" defaultRedirect="error_general.htm">
            <error statusCode="501" redirect="error_bug.aspx"/>
```

```
</customErrors>  
</system.web>  
</configuration>
```

# Despliegue de aplicaciones ASP.NET

Una vez terminado el desarrollo de un sitio Web, éste debe transferirse sobre un entorno de pruebas o de producción. Esta operación se denomina despliegue de la aplicación.

## 1. Despliegue manual

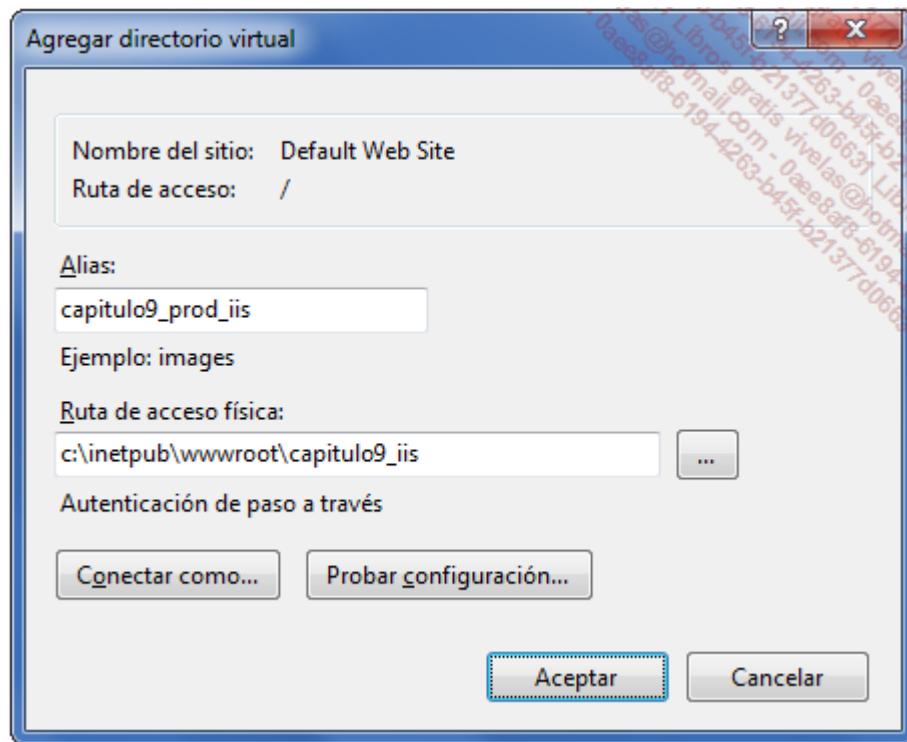
### a. Creación de una carpeta virtual

El despliegue manual consiste principalmente en copiar los archivos del sitio en la carpeta del servidor de producción. Antes de proceder a la transferencia, la carpeta de destino debe prepararse para reconocerla en las URL solicitadas por los navegadores al servidor Web.

Como existe una diferencia entre la raíz física del sistema de archivos (C:\, por ejemplo) y la raíz lógica de las URI (/ se corresponde con C:\inetpub\wwwroot), hablamos de carpeta virtual para diseñar una carpeta accesible mediante HTTP.

### Creación de una carpeta virtual por IIS

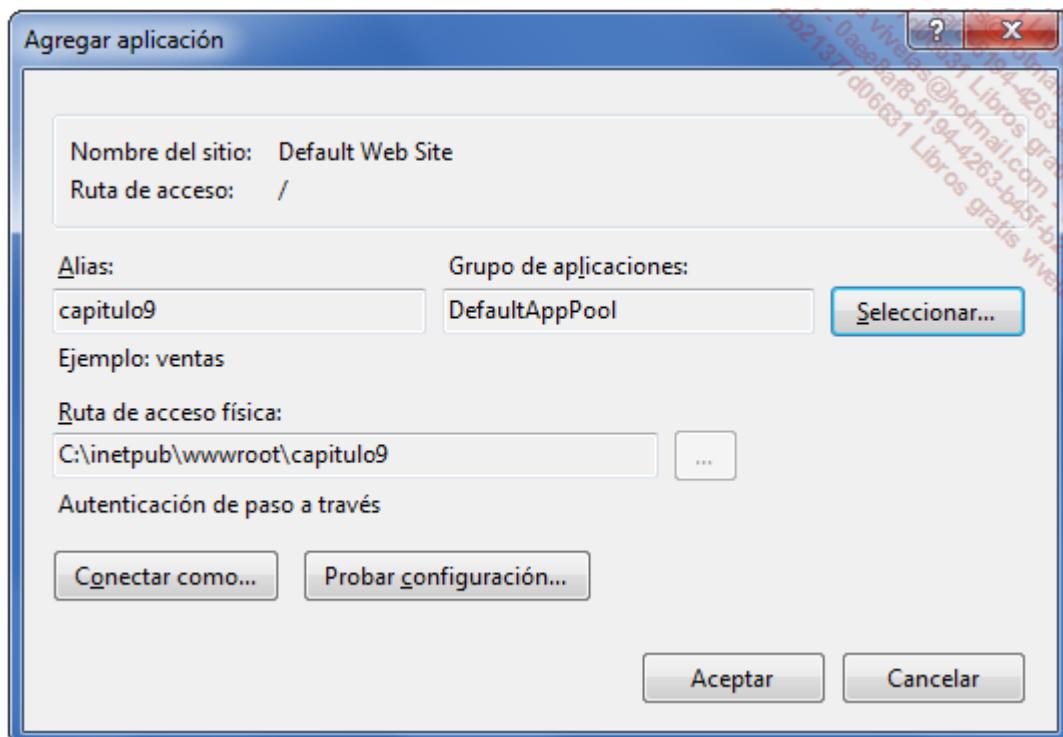
La consola de administración IIS sabe crear carpetas virtuales. Hay que utilizar el menú contextual **Agregar directorio virtual** en la entrada Sitio Web por defecto.



### Creación de una aplicación a partir de una carpeta existente

La consola de administración muestra, automáticamente, las carpetas que figuran en C:\inetpub\wwwroot. Las que se hubieran creado sin declararse como carpetas virtuales no se muestran del mismo modo.

Para crear una aplicación (y, por tanto, declarar la carpeta como carpeta virtual), hay que utilizar el menú contextual **Convertir en aplicación**.



### b. Selección de archivos que se quiere copiar

Con el nuevo modelo de compilación, es mucho más sencillo seleccionar los archivos que se quiere copiar en el servidor de destino.

Archivo seleccionado	Rol
*.aspx, *.ascx, *.master	Archivos HTML, controles de usuario, páginas maestras.
*.cs	Archivos de código subyacente.
Carpetas App_Code, App_Data, App_Global_Resources...	Carpetas especiales.
Web.config	Archivo de configuración.
Web.Sitemap	Mapa del sitio.
*.gif, *.jpg, *.htm	Recursos estáticos del sitio, también llamados elementos de contenido.
Carpeta bin	Contiene los ensamblados privados referenciados mediante el archivo Web.config.

De hecho, se tienen que copiar casi todos los

archivos. Las carpetas especiales ASP.NET y los filtros IIS sobre ciertas extensiones protegen al sitio frente a la divulgación involuntaria de información o de código fuente.

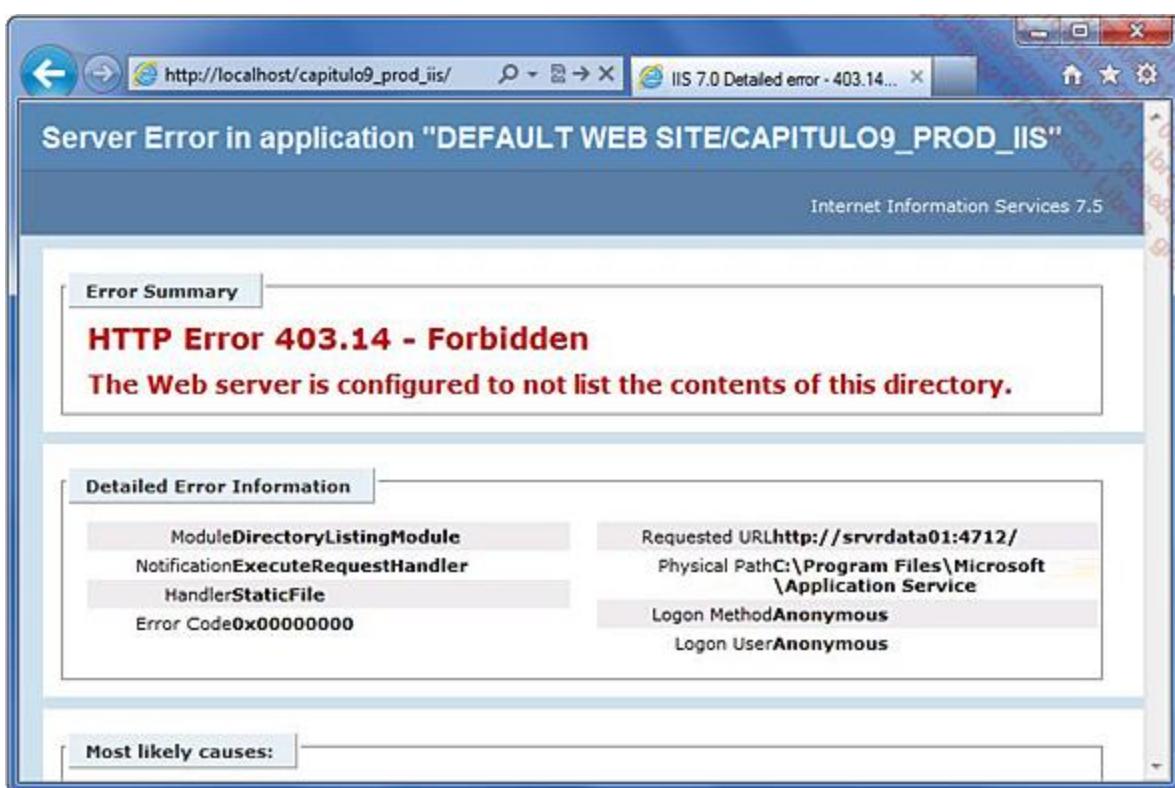
Algunos archivos pueden, si existen, seleccionarse para copiarse, aunque no están implicados en el funcionamiento de la aplicación en producción:

Archivo	Rol
*.sln, *.suo, *.csproj	Archivos de solución y de proyecto.

*.disco, *.wsdl	Archivos que permiten descubrir dinámicamente servicios Web. El WSDL no es necesario si la referencia Web no se genera en el servidor de aplicaciones.
--------------------	--

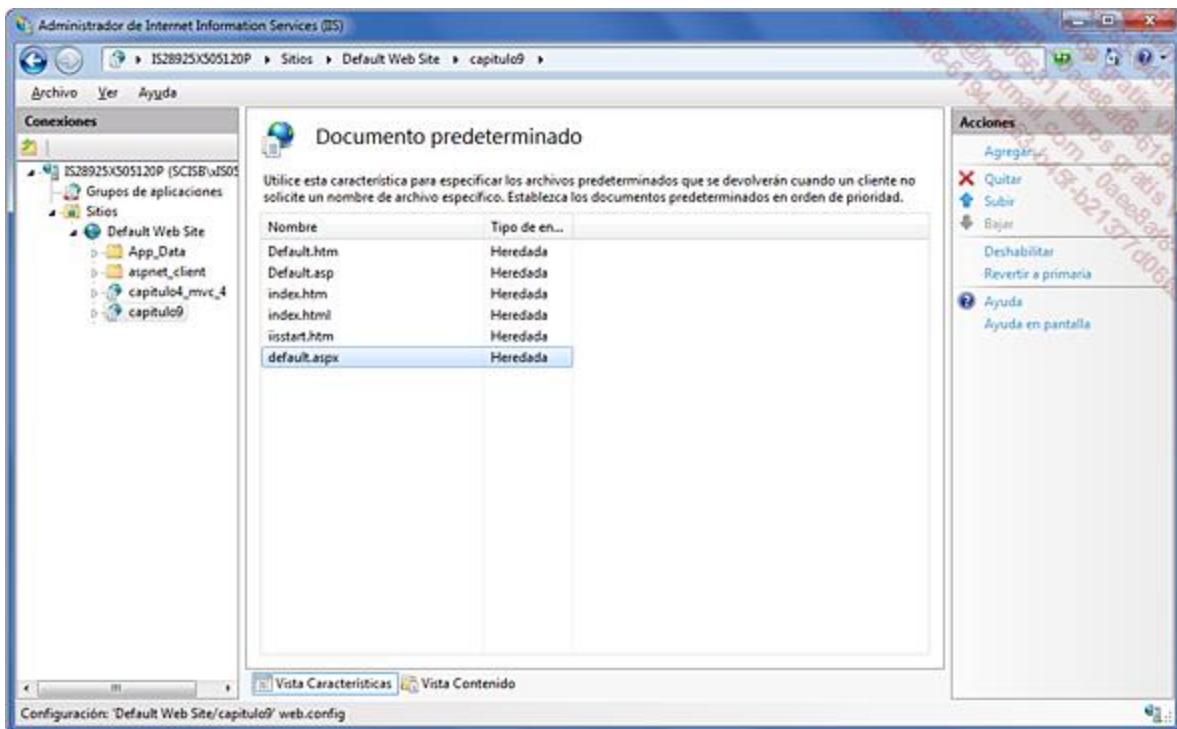
c. La página por defecto

La puesta en producción de un sitio produce, en ocasiones, sorpresas desagradables: el acceso a la URL prevista provoca un error inmediato.



No se trata, de hecho, de un problema de seguridad, sino de la página por defecto. Algunas versiones de IIS no proporcionan un estándar Default.aspx como página por defecto, sino Default.asp. Como el usuario no está autorizado a recorrer el catálogo de la carpeta mediante HTTP, se le muestra un mensaje 403.

Basta con agregar, a través de la consola IIS, la página correcta en la sección Documento predeterminado.



Destaquemos, también, que el concepto de página de inicio de Visual Studio no tiene ningún sentido en IIS.

#### d. Compilación previa

El primer usuario de una aplicación ASP.NET obtendrá, probablemente, retardos fuera de lo normal en el procesamiento de sus consultas; recordemos que el nuevo modelo prevé una compilación bajo demanda, desde que un archivo se modifica, hasta que se ejecuta por primera vez.

La herramienta **aspnet\_compiler** resuelve este problema realizando una compilación previa del conjunto del sitio Web:

```
C:\Program Files\Microsoft Visual Studio 11.0\UC>aspnet_compiler -v /capítulo9_iis
Herramienta de compilación de Microsoft (R) ASP.NET versión 4.0.30319.17929
Utilidad para precompilar una aplicación ASP.NET
(C) Microsoft Corporation. Todos los derechos reservados.

C:\Program Files\Microsoft Visual Studio 11.0\UC>_
```

Las principales opciones son:

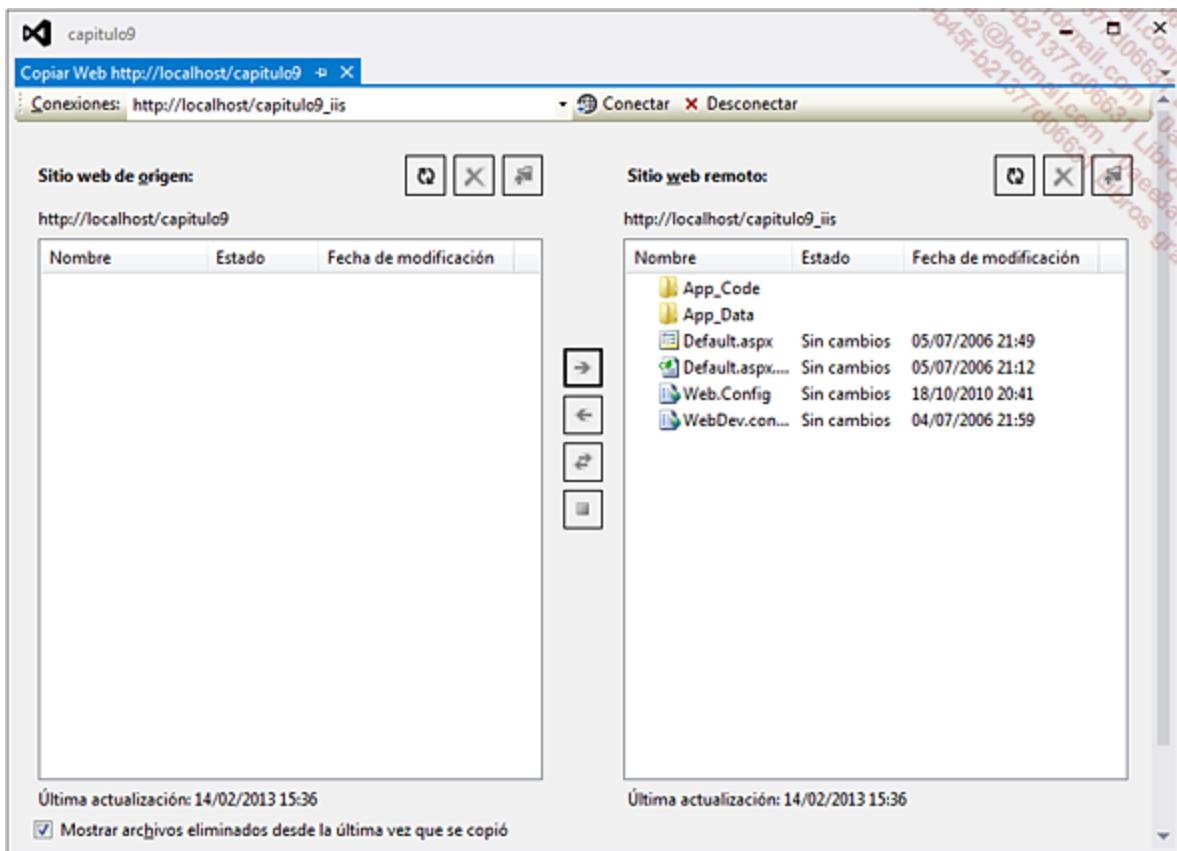
-v	Carpeta virtual que contiene a la aplicación.
----	---

-p	Carpeta física que contiene a la aplicación.	2.
-d	Compila con información de debug.	
-c	Recompilación completa de la aplicación.	
-keyfile	Clave utilizada para marcar el ensamblado.	
-aptca	Autoriza llamadas de fuentes de confianza parcial.	

## Despliegue mediante un sistema de copia

Visual Studio incluye la función de copiar sitios Web, ya proporcionada por Interdev. Esta función se activa mediante el menú **Sitio Web - Copiar sitio Web** o mediante la barra de herramientas del explorador de soluciones, y se presenta bajo la forma de dos catálogos de archivos, uno local y otro remoto.

El botón **Conectar** establece una conexión de archivo, FTP, IIS local o HTTP hacia el servidor de aplicaciones de destino. Una vez establecida la conexión, Visual Studio compara los catálogos local y remoto e indica, mediante una flecha, qué archivos son los más recientes. Se copian en el sistema de destino mediante el botón con una flecha que figura entre ambas listas.



## Supervisión de aplicaciones ASP.NET

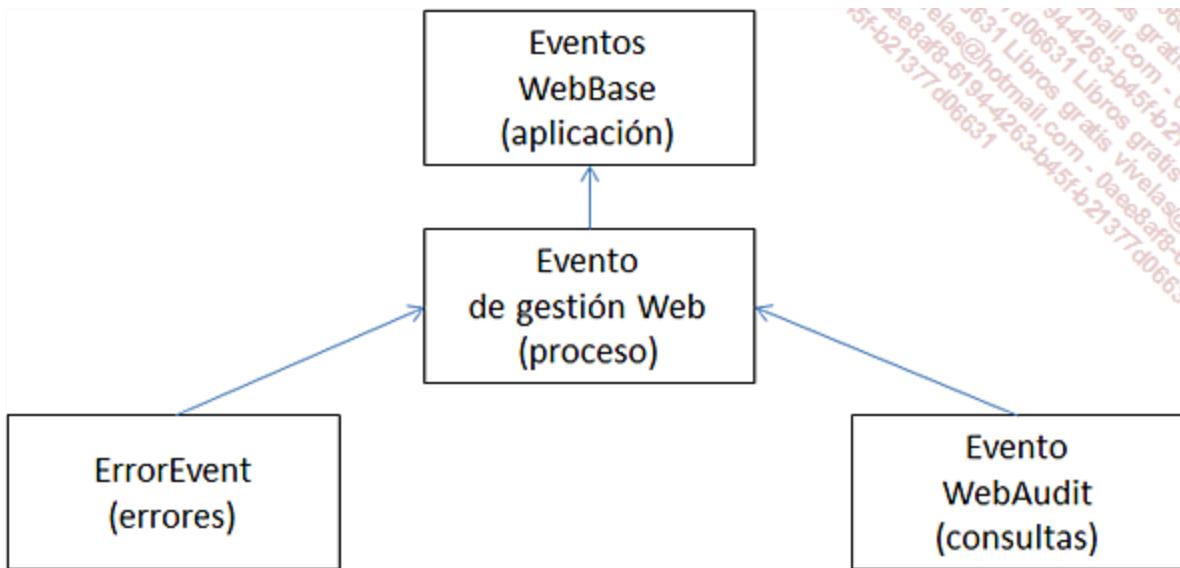
### 1. La infraestructura de supervisión Health Monitoring

Los administradores controlan el estado de las aplicaciones en producción. Éstas emiten señales, que no siempre son errores de ejecución, sino mensajes útiles para el control de la carga, el arranque de procedimientos de alerta o de detención de aplicaciones.

La infraestructura de control del estado **Health Monitoring** monitoriza estas señales y las registra en distintos soportes (registros, mensajería, indicadores de rendimiento...).

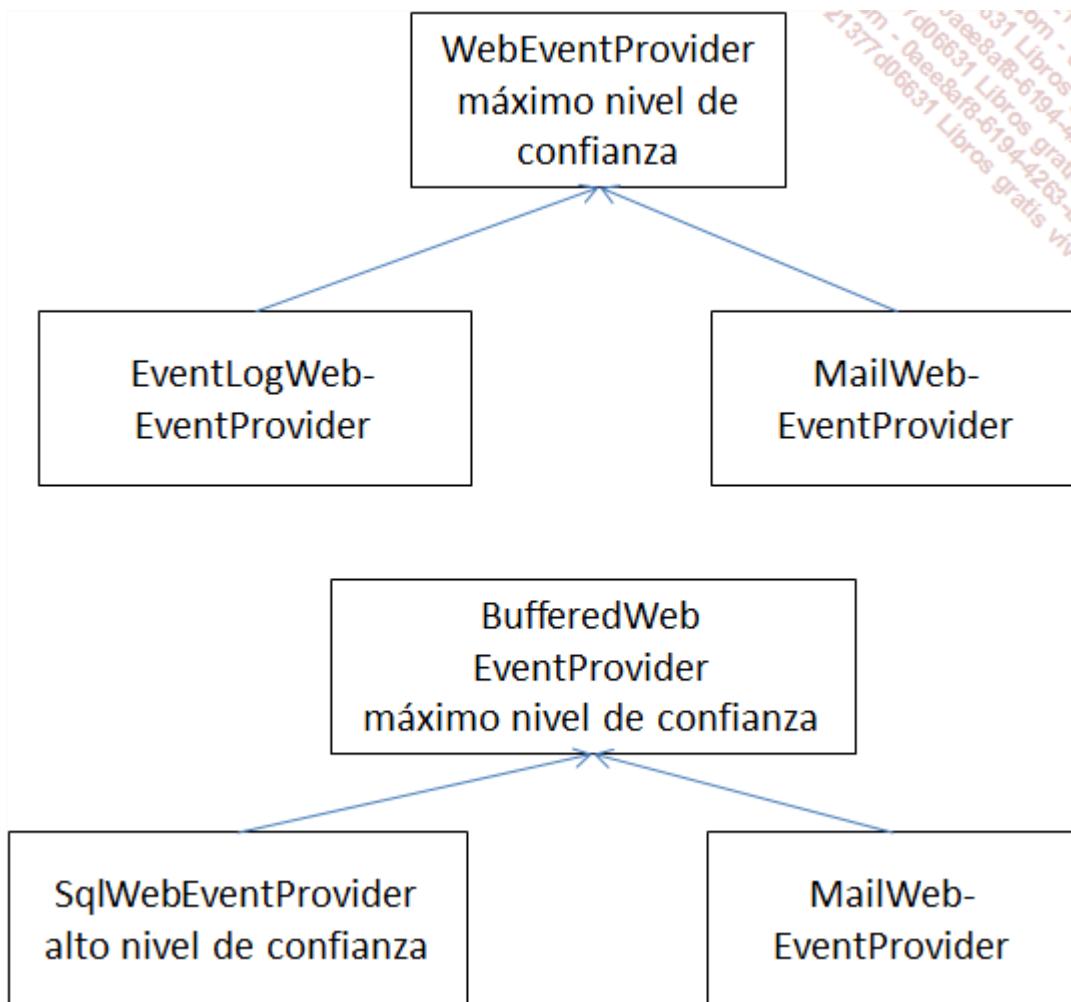
#### a. La jerarquía de eventos Web

Los eventos producidos por una aplicación Web se organizan de forma jerárquica. Cada evento se implementa mediante una clase. El siguiente esquema muestra las clases básicas de esta taxonomía:



#### b. La jerarquía de los proveedores

Los proveedores son clases capaces de procesar eventos capturados. Según el nivel de confianza de las aplicaciones, sólo son aplicables algunos proveedores:



## 2. Implementación en ASP.NET

El archivo Web.config declara las reglas de captura y procesamiento de eventos producidos por la aplicación ASP.NET. No se trata de eventos de componentes (como Button Click) sino señales de administración como, por ejemplo, **Failure Audits** (conexión bloqueada).

### a. Declarar eventos

La primera etapa consiste en declarar los eventos susceptibles de ser capturados. Estos eventos se corresponden con los tipos del espacio de nombres **System.Web.Management**. Muchos de ellos ya están declarados en el archivo Web.config raíz, también heredados. No es, por tanto, útil (ni siquiera posible) declararlos en el archivo Web.config de la aplicación.

```
<healthMonitoring>
  <eventMappings>
    <!-- ya existe en el archivo Web.config raíz
        <add name="Failure Audits" type="System.Web.Management.
WebFailureAuditEvent"/>
```

```
  -->
</eventMappings>
</healthMonitoring>
```

No obstante, la infraestructura Health Monitoring es extensible y el administrador podrá registrar sus propios eventos.

### b. Declarar proveedores de escucha

Los proveedores de escucha son canales sobre los que se inscriben los eventos capturados. El archivo de configuración raíz Web.config ya declara varios proveedores como, por ejemplo, **EventLogWebEventProvider**:

```
<providers>
  <!-- ya existe en el archivo Web.config raíz
      <add name="EventLogProvider" type="System.Web.Management.
EventLogWebEventProvider"/>
  -->
</providers>
```

### c. Agregar reglas de suscripción

Las reglas asocian eventos y proveedores:

```
<rules>
  <add name="Failure Audits default" eventName="Failure Audits" provider=
"EventLogProvider"/>
```

```
</rules>
```

Mediante esta asociación, un error de autenticación quedaría registrado en el registro de seguridad del sistema:

The screenshot shows the Windows Event Viewer interface. On the left is a navigation pane with various system management tools like Task Scheduler, Event Viewer, and Performance Monitor. The main area is a table displaying event logs. A search result for 'Error de apertura de sesión' (Authentication failure) is highlighted. The table has columns for Palabras clave (Keywords), Fecha y hora (Date and time), Origen (Source), Id. del evento (Event ID), and Categoría d... (Category). The details pane on the right shows the specific event data: Motivo: Nombre de usuario desconocido o contraseña incorrecta (Reason: Unknown user name or incorrect password), Nombre de usuario: db3admin (User name: db3admin), Dominio: DOM\_USERS (Domain: DOM\_USERS), Tipo de sesión: 2 (Session type: 2), Proceso de apertura de sesión: IIS (Session opening process: IIS), and Nombre de la estación de trabajo: EQUIPO\_LOCAL (Workstation name: EQUIPO\_LOCAL).

Palabras clave	Fecha y hora	Origen	Id. del evento	Categoría d...
🔒 Error de auditoría	14/02/2013 14:56:45	Auditoría de s...	4673	Uso de privi...
🔍 Auditoría correcta	14/02/2013 14:56:44	Auditoría de s...	4688	Creación de...
🔒 Error de auditoría	14/02/2013 14:56:37	Auditoría de s...	4673	Uso de privi...
🔒 Error de auditoría		Error de apertura de sesión:		
🔍 Auditoría correcta		Motivo: Nombre de usuario desconocido o contraseña incorrecta		
🔍 Auditoría correcta		Nombre de usuario: db3admin		
🔍 Auditoría correcta		Dominio: DOM_USERS		
🔍 Auditoría correcta		Tipo de sesión: 2		
🔍 Auditoría correcta		Proceso de apertura de sesión: IIS		
🔍 Auditoría correcta		Nombre de la estación de trabajo: EQUIPO_LOCAL		