Servicios Web con .NET

- **1. Introducción a los servicios Web**

Básicamente un **servicio Web** es un clase que se publica en un servidor Web con soporte para ASP.NET (actualmente el único disponible es el **Internet Information Server 5.0** de Microsoft tras ser actualizado con el .NET SDK) y a cuyos métodos es posible llamar remotamente. Estas clases pueden escribirse en la mayoría de los lenguajes de los lenguajes adaptados a .NET, como Visual Basic.NET, Jscript.NET y C#, pero nosotros nos centraremos en el desarrollo de servicios Web usando el último.

Para acceder a un servicio Web se pueden utilizar varios protocolos Web estándar, como HTTP GET ó HTTP POST, aunque el específicamente diseñado para ello por Microsoft es el **Protocolo de Acceso a Objetos Simple** (Simple Access Protocol o SOAP), que se basa en la utilización del protocolo HTTP para el transporte de mensajes y el lenguaje XML para la escritura del cuerpo de estos mensajes, y que ha sido enviado al W3C por Microsoft en mayo de 2000 para su estandarización.

Una de las grandes ventajas de los servicios Web es que pueden ser accedidos desde cualquier aplicación que sea capaz de generar mensajes e interpretar mensajes escritos en SOAP, aún cuando ésta no esté diseñada para la plataforma .NET Es más, los aplicaciones que consuman estos servicios no necesitan conocer ni cuál es la plataforma ni cuál es el modelo de objetos ni cuál es el lenguaje utilizado para implementar estos servicios. Otra gran ventaja es que son tremendamente sencillos de escribir, pues como veremos a continuación basta hacer unos retoques mínimos al código de una clase cualquiera para convertirla en un servicio Web que podrá ser accedido remotamente; por no mencionar la enorme facilidad y transparencia con la que aquellos clientes del servicio escritos bajo .NET pueden acceder al mismo, gracias a las utilidades generadoras de proxys que Microsoft proporciona.

- **2. Escritura de un servicio Web**

Para ver lo fácil que es escribir un servicio Web, en primer lugar vamos a escribir una sencilla clase en C# que simplemente consta de un único método que nos devuelve la fecha actual. Más adelante veremos cómo es trivial la conversión de la misma en servicio Web. El código de nuestra clase es:

```
using System;

class ServicioFecha
{
        public String Fecha (bool detalles)
        {
                if (detalles)
                        return DateTime.Now.ToLongDateString();
                else
                        return DateTime.Now.ToShortDateString();
         }
}
```

Como se ve, los objetos de esta clase constan de sólo un método propio: el método **Fecha()**, que devuelve la fecha actual incluyendo información detallada (día de la semana, número de mes, día del mes y año con cuatro dígitos) o no (sin día de la semana y sólo dos dígitos para el año) según indique su parámetro

Para conocer cuál es la fecha actual se utiliza la propiedad **Now** de la clase **DateTime**[1], que devuelve un objeto **DateTime** que contiene información sobre la misma; y para acceder obtener las cadenas de texto que contienen la información detallada y no detallada sobre la fecha se usan, respectivamente, los métodos **ToLongDateString()** y **ToShortDateString()** de la misma.

Para convertir esta clase en un servicio Web sólo hemos de hacerla heredar de la clase **System.Web.Services**, colocar el atributo **[WebMethod]** antes de la declaración de cada uno de los métodos de la clase que deseemos que sean accesibles remotamente e incluir una directiva **WebService** que indique al motor de **ASP.NET** (que es quien se encargará de gestionar el acceso al servicio Web) de cuál es el lenguaje en que está escrito el servicio Web y cuál es la clase que se expondrá como servicio Web

Además de estos cambios en el código del fichero fuente, también hemos de cambiar su extensión por **.asmx** y colocarlo en algún directorio virtual del Internet Information Server para que así las aplicaciones cliente puedan acceder a él. Con todo ello, el código anterior quedará así:

```
// Fichero: ServicioFecha.asmx
<% @ WebService Language="C#" Class="ServicioFecha" %>

using System;
using System.Web.Services;

class ServicioFecha: WebService // Derivamos de WebService
{
        [WebMethod] public String Fecha (bool detalles)
        {
                if (detalles)
                        return DateTime.Now.ToLongDateString();
                else
                        return DateTime.Now.ToShortDateString();
         }
}
```

De este código hay que comentar varias cosas:

+ La directiva **WebService** ha de incluirse antes que cualquier otro código en el fichero **.asmx**

+ Es posible almacenar el código ya compilado en un fichero **.dll**[2] separado del **.asmx.** Este código se almacenaría en el subdirectorio **/bin** del directorio virtual de

[1] En realidad DateTime es una estructura y no una clase. Sin embargo, a efectos prácticos en esta introducción a los servicios Web en C# ello es indiferente.
[2] Para compilar en formato **.dll** se ha de especificar la opción **/t:library** al llamar al compilador.

nuestra aplicación y para referenciarlo desde la etiqueta **WebService** del **.asmx** se usaría el atributo **Codebehind** de ésta del siguiente modo:

<% @ WebService Language="C#" Codebehind="ServicioFecha" Class=ServicioFecha" %>

+ No hay que confundir el concepto de atributo en C# con los atributos de las etiqueta ASP.NET Un ejemplo de lo primero es el atributo **[WebMethod]** ya visto, y en general estos atributos sirven para indicar información sobre la clase o miembro al que preceden (**[WebMethod]** indica que el método al que precede es accesible remotamente); mientras que ejemplos de atributos de etiquetas ASP.NET son los atributos **Language** y **Codebehind** también ya comentados, siendo la utilidad de este tipo de atributos indicar información sobre la etiqueta a la que acompañan.

- **3. Página de prueba de un servicio Web**

Al abrir con Internet Explorer una página .asmx alojada en un servidor Web con soporte ASP.NET se obtiene una página con información sobre el los servicios en esta almacenados. Por ejemplo, si situamos nuestro anterior ejemplo en el directorio raíz de la jerarquía virtual de IIS, al teclear en la barra de direcciones de Internet Explorer http://localhost/ServicioFecha.asmx se obtiene:

[Imagen 7: Página de pruebas del servicio Web *ServicioFecha*]

Véase que esta página contiene información incluye datos sobre cuál es el servicio Web ofrecido, cuáles son los métodos que se ofrecen y cuál es la signatura de estos. Además, la página también incluye formularios desde los que podemos hacer llamadas de prueba a los métodos del servicio Web.

Es importante señalar que para que la página de prueba se obtenga hay que acceder al fichero **.asmx** a través de **IIS**; es decir, de modo que se haga que este se active y procese la petición. Por eso, para acceder a ella no basta hacer doble click sobre su icono en la ventana de exploración de discos de Windows, sino que hay utilizar un dirección como la arriba indicada (**URL**)

- **4. Acceso a un servicio Web mediante SOAP**

Ya se ha comentado que SOAP es el protocolo basado en HTTP y XML que Microsoft ha desarrollado para el acceso a los servicios Web. Aunque podríamos comunicarnos directamente con un servicio Web enviando mensajes escritos en el formato de este protocolo e interpretando sus repuestas, lo cierto es que ello sería bastante incómodo, por lo que Microsoft proporciona un mecanismo mucho más sencillo y transparente basado en el uso de **proxies**.

Un proxy es una clase que ofrece la misma interfaz que el servicio Web al que se pretende acceder pero que esta almacenada en la máquina local y no contiene la verdadera implementación de los métodos ofrecidos por el servicio, sino que en su lugar contiene código encargado de redirigir las llamadas que hagamos a sus métodos al verdadero servicio Web. Gracias a los proxies, conseguimos comunicarnos con el servicio remoto como si de una clase normal escrita en C# se tratase, y es el proxy el encargado de intercambiar los menajes SOAP necesarios para la comunicación remota.

Para generar automáticamente el proxy es necesario contar con una especificación del servicio (métodos ofrecidos, signatura de estos, protocolos soportados, etc.) que sea fácil de interpretar por una máquina. Para ello se utiliza un fichero XML escrito en el llamado **Lenguaje Descriptor del Servicio Web** (Web Service Descriptor Language o **WSDL**) Para obtener este fichero sólo hay que acceder al servicio Web concatenado la cadena ?WSDL al final del mismo. Así, siguiendo con nuestro ejemplo la cadena sería http://localhost/ServicioFecha.asmx?WSDL

Gracias a este tipo de ficheros, podemos generar el proxy de manera automática utilizando la utilidad **wsdl.exe** incluida en el .NET SDK. Esta utilidad se usaría así para generar un proxy para nuestro ejemplo anterior:

```
wsdl http://localhost/ServicioFecha.asmx
```

El proxy generado tendrá el mismo nombre que el fichero **.asmx** a partir del que se genera pero extensión **.cs** Es posible cambiar este nombre si así se desea indicando uno nuevo mediante la opción **/out:** de **wsdl**

A partir del proxy la escritura de una aplicación que haga uso del servicio es trivial. Sólo hay que utilizar el proxy como si de la clase remota se tratase. Por ejemplo:

```
using System;

class ClienteFecha
{
        public static void Main()
        {
                ServicioFecha s = new ServicioFecha();

                Console.WriteLine("Fecha actual: {0}", s.Fecha(false));
                Console.WriteLine("Fecha actual detallada: {0}", s.Fecha(true));
        }
}
```

De este código es fácil deducir que lo que se hace es crear un objeto de la clase remota **ServicioFecha** y mostrar dos mensajes de texto, uno para probar el funcionamiento de cada una de las formas de llamar al método **Fecha()**. Nótese la absoluta transparencia de éste código respecto a la ubicación real de la clase remota.

Para compilar este cliente hay que tener en cuenta que en la clase proxy se utilizan elementos definidos en otras librerías, por lo que hay que referenciarlas con:

```
csc ClienteFecha.cs ServicioFecha.cs
```

Así obtendremos un **ClienteFecha.exe** El resultado de ejecutarlo es:

```
Fecha actual: 07/11/2000
Fecha actual detallada: Tuesday, July 11, 2000
```

- **5. Mantenimiento del estado**

Por defecto un servicio Web carece de estado, ya que por cada llamada a un método de un servicio se crea una nuevo objeto de la clase a la que se hace la petición; objeto que la atiende y es destruido tras ello. Por consiguiente, no se guarda información sobre llamadas previas y por tanto los objetos de un servicio Web escrito así no son verdaderos objetos, ya que lo que el realmente el cliente hace son llamadas sueltas que no tienen relación entre sí.

Para conseguir almacenar información sobre llamadas previas, la clase **WebService** de la que todo servicio Web hereda proporciona unos objetos llamados **Application** y **Session**, de clase **HttpApplicationState** y **HttpSessionState** respectivamente, que actúan como depósitos de información sobre llamadas previas. El primero de estos objetos permite almacenar información que será compartida por todas las aplicaciones clientes del servicio, mientras que la información almacenada en el segundo sólo es compartida entre sucesivas llamadas de un mismo cliente

A continuación se muestra una modificación del ejemplo anterior en la que se utilizan los dos objetos antes comentados para guardar información sobre cuántos clientes han accedido al método **Fecha()** del servicio. Además, se proporcionan un

método **Accesos()** mediante el cual los clientes pueden consultar el número total de accesos que ha tenido la aplicación y un método **AccesosMíos()** mediante el cual los clientes pueden consultar cuántos accesos ellos mismos han realizado:

```
<% @ WebService Language="C#" Class="ServicioFechaEstado" %>

using System;
using System.Web.Services;

class ServicioFechaEstado:WebService
{
  [WebMethod(EnableSession=true)] public String Fecha(bool detallada)
  {
    if (Application["Accesos"] == null)
          Application["Accesos"] = 1;
    else
          Application["Accesos"] = ((int) Application["Accesos"]) + 1;

    if (Session["Accesos"] == null)
          Session["Accesos"] = 1;
    else
          Session["Accesos"] = ((int) Session["Accesos"]) + 1;

    if (detallada)
          return DateTime.Now.ToLongDateString();
    else
          return DateTime.Now.ToShortDateString();
  }

  [WebMethod] public int Accesos()
  {
    if (Application["Accesos"] == null)
          return 0;
    else
      return ((int) Application["Accesos"]);
  }

  [WebMethod(EnableSession=true)] public int AccesosMíos()
  {

    if (Session["Accesos"] == null)
          return 0;
    else
      return ((int) Session["Accesos"]);

  }

}
```

De este código cabe destacar los siguientes aspectos:

+ Para acceder a la información almacenada en los objetos **Application** y **Session** basta indicar entre corchetes el nombre con el que identificaremos la información que pretendemos usar (se usa pues una sintaxis similar a la de las tablas. A ésta se le llama **indizador** en C#) Estos objetos devuelven elementos de tipo **object**, y para convertirlos a **int** usamos el operador de conversión implícita **(int)** Además en el caso de que la información que se pretende obtener no haya sido aún introducida devuelven **null**, lo que controlamos mediante las condiciones de los **if**.

+ Es necesario modificar la forma en la que usamos el atributo **[WebMethod]** dándole el valor **true** a la propiedad **EnableSession** en todos aquellos métodos que utilicen el objeto **Session** para así habilitar el uso de éste.

Una vez modificado el **.asmx** sólo queda modificar el cliente que hemos estado usando para probar el servicio para que haga uso de sus nuevas características:

```
using System;

class ClienteFechaEstado
{
  public static void Main()
  {
    ServicioFechaEstado s = new ServicioFechaEstado();
    s.CookieContainer = new System.Net.CookieContainer();

    Console.WriteLine("Accesos Totales : {0} ({1} míos)", s.Accesos(), s.AccesosMíos());
    Console.WriteLine("Fecha actual: {0}", s.Fecha(false));
    Console.WriteLine("Fecha actual detallada: {0}", s.Fecha(true));
    Console.WriteLine("Accesos Totales : {0} ({1} míos)", s.Accesos(), s.AccesosMíos());
  }
}
```

Nótese que para conseguir el mantenimiento del estado es necesario añadir al cliente una línea extra que inicialice su propiedad **CookieContainer** con un objeto del tipo **CookieContainer** definido en el espacio de nombres **System.Net**. Este objeto es el encargado de almacenar información necesaria para mantener la sesión con el servicio web, y por defecto no se crea (valor **null**) para hacer así más eficiente el funcionamiento del proxy mientras no se necesite mantener el estado. Como en nuestro caso si lo necesitamos, hemos de configurarlo a mano con la instrucción:

```
s.CookieContainer = new System.Net.CookieContainer();
```

Si ejecutamos este nuevo cliente así configurado por primera vez obtendremos:

```
Accesos totales: 0 (0 míos)
Fecha actual: 07/11/2000
Fecha actual detallada: Tuesday, July 11, 2000
Accesos totales: 2 (2 míos)
```

Y si lo volvemos a ejecutar obtendremos:

```
Accesos totales: 2 (0 míos)
Fecha actual: 07/11/2000
Fecha actual detallada: Tuesday, July 11, 2000
Accesos totales: 4 (2 míos)
```

Véase en los resultados de la ejecución del cliente cómo la información almacenada en el objeto **Session** no es compartida entre sucesivas llamadas al cliente mientras que la almacenada en el objeto **Application** sí[3]

• 6. Servicios Web con Visual Studio.NET

Visual Studio permite escribir y utilizar los servicios Web de una forma mucho más cómoda que utilizando directamente el compilador en línea de comandos **csc.exe** como se ha hecho en los ejemplo previos. A continuación veremos como escribir y utilizar el servicio Web de ejemplo anterior usando esta herramienta.

Para escribir un servicio Web hemos de arrancar Visual Studio.NET y crear un proyecto de aplicación de consola con:

- **Project Types** = *Visual C# Projects*
- **Templates** = *Web Service*
- **Name** = *ServicioFechaVS*
- **Location** = *http://josan/*

El valor dado a **Location** es el nombre del servidor donde se instalará el servicio Web. Dejaremos este campo con su valor por defecto, que coincide con el nombre la máquina donde se esté ejecutando Visual Studio (http://josan). Si nos interesa instalarlo en otra máquina sólo tendríamos que cambiar este valor por el nombre de ésta

Con los datos anteriores se creará en el servidor Web de la máquina indicada un directorio virtual con el nombre hayamos dado al proyecto en el que se depositaran los archivos del proyecto. Entre estos archivos, hay que resaltar que Visual Studio sigue la aproximación, ya comentada, de separar el código del servicio de su fichero **.asmx**

Una vez que se cree el proyecto, Visual Studio nos ofrecerá un pantalla en blanco donde podríamos ir colocando todos los controles que necesitemos[4] Nosotros haremos doble click sobre la misma y pasaremos directamente a editar el código fuente del servicio Web. Como se verá entonces, Visual Studio ya ha generado un esqueleto para el mismo que incluye la directiva **WebService** con los valores de sus atributos perfectamente configurados para funcionar con nuestra aplicación e incluso, dentro de un comentario, habrá colocado código de ejemplo de cómo escribir un servicio Web.

---

[3] Se supone que no hay más clientes accediendo simultáneamente al servicio Web

[4] Aunque no tiene mucho sentido colocar controles con parte visual, ya que un servicio Web no tiene componente visible, sino que es sólo lógica.

Gracias a toda esta infraestructura generada automáticamente, nosotros sólo tenemos que copiar los métodos antes escritos para *ServicioFecha* en esta ventana de código y todo estará listo para funcionar.

Una vez escrito el servicio Web, podemos pulsar **CTRL+F5** para probar nuestro servicio Web. Con ello conseguiremos que Visual Studio abra una ventana de Internet Explorer que nos mostrará la página de prueba del servicio.

Tras esto sólo queda escribir el cliente que accederá al mismo. Para ello de nuevo optaremos por realizar la aplicación cliente de consola ya creada pero con las pequeñas modificaciones derivadas de que ahora la clase remota se llama **WebService1** y de que Visual Studio evita tener que generar a mano el proxy para acceder a la misma.

Para crear una aplicación de consola seleccionamos **File -> New -> Project** en el menú principal de Visual Studio y configuramos la ventana que aparece con:

- **Project Types =** *Visual C# Projects*
- **Templates =** *Console Application*
- **Name =** *ClienteFechaVS*
- **Location =** *c:\C#*

Una vez hecho esto, pulsando el botón *OK* se creará nuestro proyecto y podremos empezar a escribir su código en un fichero **Class1.cs** Para cambiar el nombre de éste podemos abrir el Explorador de Solución (**View -> Solution Explorer)** y hacer click con el botón derecho del ratón sobre el mismo para renombrarlo; y para cambiar el de la clase creada por defecto (**Class1**) hacemos lo mismo en la Vista de Clase (**View -> Class View**). Nosotros optaremos por cambiar ambos nombres por *ClienteFechaVS*

El código *ClienteFechaVS* será el mismo que el del *ClienteFecha* ya escrito, por lo que podemos cortarlo de éste y pegarlo sobre el **Main()** de *ClienteFechaVS*.

Hecho esto sólo nos queda añadir la referencia al servicio Web necesaria para que se pueda compilar correctamente el código del cliente. Esto se hace seleccionando **Project -> Add Web Reference** en el menú principal de Visual Studio y escribiendo el la barra de direcciones de la ventana que aparecerá (**Address**) la ruta del servicio Web a referenciar, en nuestro caso *http://localhost/ServicioFechaVS/ServicioFechaVS.asmx* Nótese que hay que incluir en la ruta el subdirectorio *ServicioFechaVS*, ya que es allí donde Visual Studio colocó el servicio Web cuando se creó, que no es necesario concatenar el *?SDL* al final de la misma, y que con sólo pulsar el botón *OK* de esta ventana se añade al proyecto la referencia adecuada para poder acceder desde el mismo al servicio **ServicioFechaVS**, sin que sea necesario que nosotros usemos alguna utilidad que genere el proxy.

Sólo falta comentar que hay que añadir un *using localhost;* al comienzo del código fuente, ya que por defecto todas las referencias a servicios Web se importan organizándolas dentro de espacios de nombres llamados igual que el servidor donde están alojados los servicios. Si nos interesase modificar esto siempre podremos hacerlo a través del Explorador de Solución, seleccionando la carpeta **Web References** del mismo y cambiando el nombre *localhost* que tiene un icono con la bola del mundo al lado por el nombre deseado.