Web Services.

Un servicio Web o WebService es un servicio ofrecido por una aplicación que expone su lógica a clientes de cualquier plataforma mediante una interfaz accesible a través de la red utilizando tecnologías (protocolos) estándar de Internet.

Por ejemplo, una aplicación como Access está formada por un conjunto de componentes que ofrecen una serie de servicios, como el acceso a datos, la impresión de informes, el diseño de tablas...

La idea de los servicios es la misma, aunque éstos no tienen por qué estar en el mismo ordenador que el cliente y además son accedidos a través de un servidor Web y de un modo independiente de la plataforma, utilizando protocolos estándar (HTTP, SOAP, WSDL, UDDI).

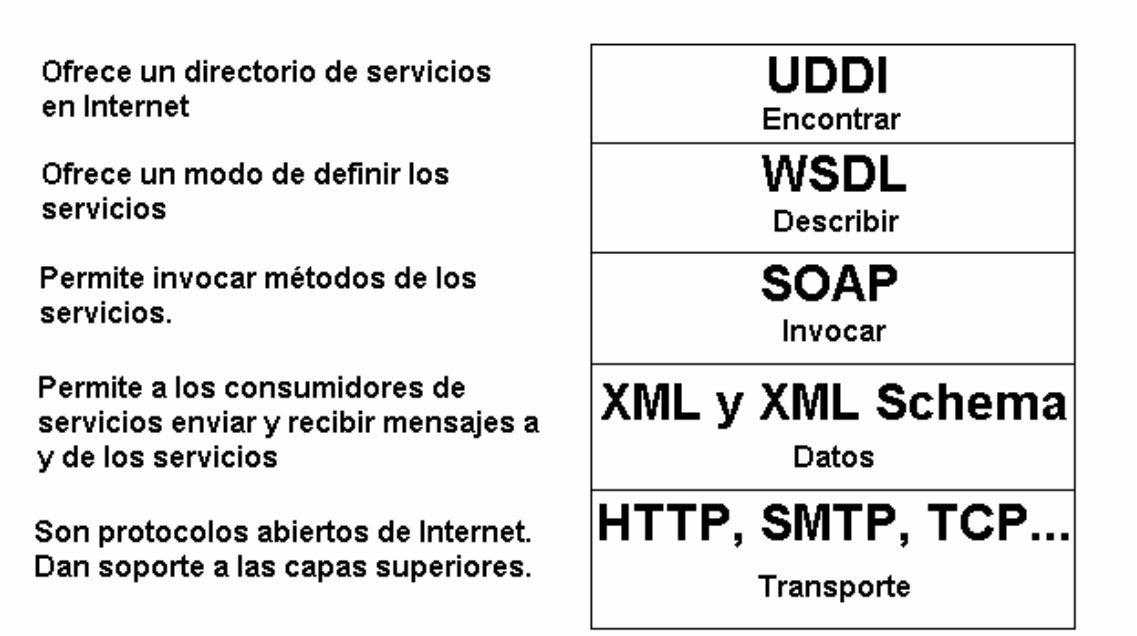

Figura 23.1 Pila de protocolos de los WebServices.

Para crear un servicio puede utilizarse cualquiera de los lenguajes disponibles en la plataforma .NET.

Una vez creado el servicio, para conseguir que sea accesible por los consumidores, es necesario describirlo utilizando un lenguaje estándar llamado WSDL (*Web Service Description Language*).

Los clientes del servicio podrán estar creados en cualquier lenguaje y ejecutarse sobre cualquier sistema operativo y hardware, lo único necesario es que sean capaces de obtener y entender la descripción WSDL de un servicio.

Un archivo WSDL es, en realidad, un archivo XML en el que se identifica el servicio y se indica el esquema para poder utilizarlo, así como el protocolo o protocolos que es posible utilizar.

Una vez dispone de esta información, el cliente puede comunicarse con el servicio utilizando protocolos como HTTP o SOAP (SOAP añade invocación de métodos a HTTP, aunque es posible hacerlo con peticiones HTTP-GET y/o HTTP-POST en lugar de SOAP).

Además de describir un servicio para que pueda ser utilizado por los clientes es importante publicar el servicio de modo que pueda ser encontrado por clientes que no conozcan necesariamente el componente que ofrece el servicio, pero que busquen un servicio de sus características. Esto se logra mediante el estándar UDDI (*Universal Description, Discovery and Integration Registry*). Realmente se trata de un servicio mundial en el que los proveedores de servicios pueden registrarlos de modo gratuito.

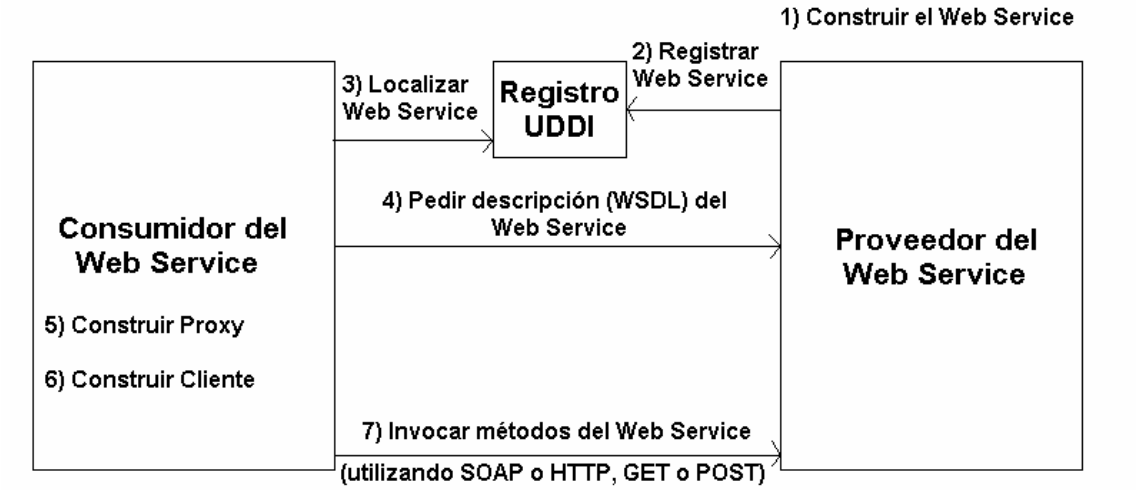

Figura 23.2 Creación, registro, búsqueda y utilización de un WebService.

¿Por qué tantos protocolos?

Actualmente, al publicar un documento en un servidor web apenas es necesario tener en cuenta las características del cliente (S.O., hardware, aplicaciones...). Esto es posible gracias a que HTML y HTTP son un estándar mundial de diseño, solicitud y transmisión de documentos. De este modo, el servicio web (www) es universal, es decir, accesible desde cualquier cliente.

Para conseguir algo similar con los WebServices, Microsoft, IBM y otras empresas han estado y están definiendo los protocolos comentados, los cuales permitirán describir un servicio, publicarlo de modo que los clientes puedan localizarlo y utilizarlo...

A continuación se comentan brevemente algunas alternativas a los protocolos comentados:

- En el caso de HTTP y SOAP otras opciones (en este caso sustitutivas y/o complementarias) son:

    - Jabber, es un protocolo asíncrono de transporte (más ligero que HTTP).
    - EbXML, está pensado para integración de servicios en soluciones B2B (*Business to Business*).
    - XML-RPC, está basado en HTTP-POST.
- En el caso de WSDL otras opciones son:
    - RDF (*Resource Description Framework*), definido por el W3C. Es más potente pero también más complejo que WSDL.
    - DAML (*DARPA Agent Markup Language*), definido por la agencia de defensa estadounidense (DARPA). Es también más potente pero más complejo que WSDL.
- En el caso de UDDI existe una propuesta alternativa realizada por Microsoft e IBM, llamada WS-Inspection Language.

Ventajas de los Web Services.

Entre las ventajas más importantes que ofrecen los WebServices se pueden citar:

- Ofrecen una "tecnología distribuida de componentes" optimizada.
- Evitan los problemas inherentes a la existencia de firewalls, ya que SOAP utiliza HTTP como protocolo de comunicación.
- Permiten una invocación sencilla de métodos, mediante SOAP.
- Los clientes o "consumidores de servicios" pueden estar en cualquier plataforma (basta con que soporten XML/SOAP, incluso puede sustituirse SOAP por HTTP).
- Permiten centralizar los datos, independientemente de si los WebServices están distribuidos o no.

Estructura de un Servicio Web.

Todo Servicio Web ha de implementarse mediante una clase derivada de la clase Web Service, que pertenece al `namespace System.Web.Services`.

Ninguno de los métodos de esta clase, ya sean privados, protegidos o públicos será accesible para un cliente o consumidor del servicio. Si se desea que un método sea accesible desde un cliente, debe ser definido con el atributo `[WebMethod]`.

Al igual que los WebForms, un Web Service o servicio Web se compone de dos ficheros (los más importantes, aunque realmente son más):

- Un fichero con extensión `.asmx`: equivale al fichero `.aspx` de las páginas ASP.NET (WebForms). Es la página que se pedirá desde el navegador del cliente para acceder al servicio.
- Un fichero `.cs`: Contiene el código del servicio (equivale al `.cs` de las páginas ASP.NET).

Creación de un Web Service.

Es posible crear un Web Service escribiendo todo el código o bien utilizando Visual Studio. En este caso se van a mostrar los pasos para hacerlo mediante Visual Studio.

- El primer paso es crear un proyecto de tipo Web Service (se llamará `HolaInternetService`):

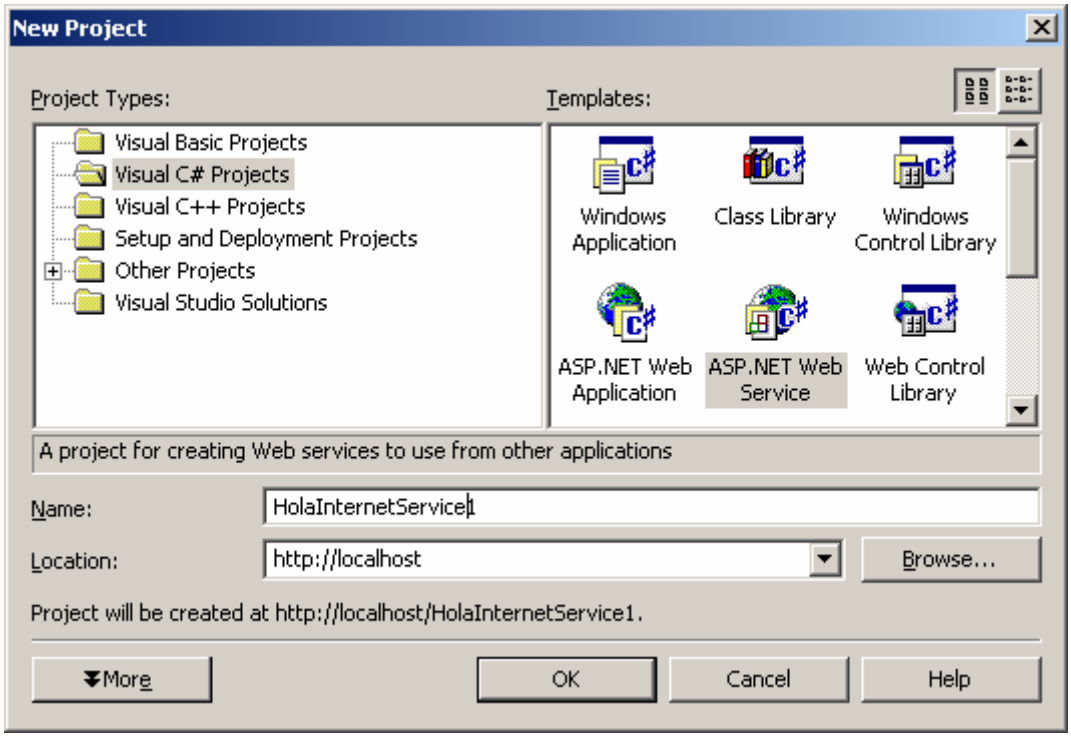

Figura 23.3. Cuadro de diálogo **Nuevo Proyecto**. Creación de un Web Service.

Al pulsar `OK` se crean 3 ficheros:

1) `Service1.asmx`

Para ver el contenido del fichero `Service1.asmx`:

- Pulsar el botón derecho del ratón sobre `Service1.asmx` en el **explorador de soluciones**.
- En el menú contextual elegir **Abrir con.**
- En el cuadro de diálogo que se muestra, elegir **Editor de código fuente (Texto)**:

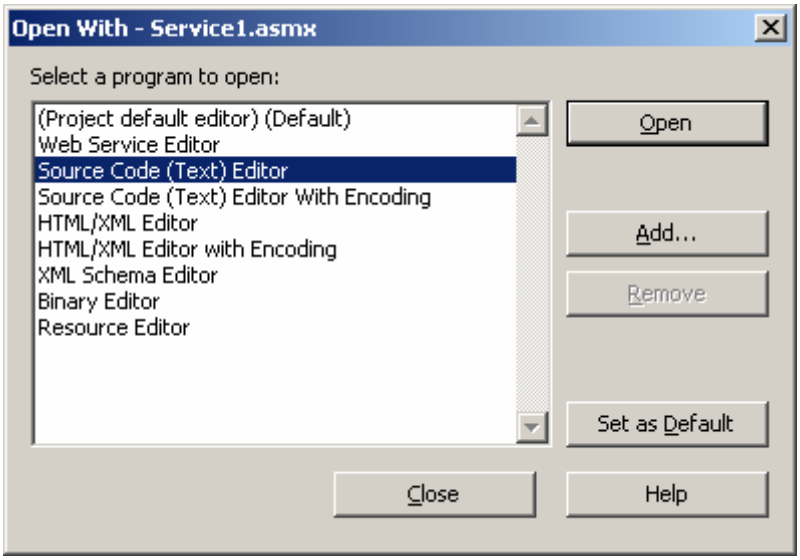

Figura 23.4. Cuadro de diálogo **Abrir con**. La opción **Editor de código fuente** permitirá ver el contenido del fichero `.asmx`.

El contenido es:

```
<%@ WebService Language="c#" Codebehind="Service1.asmx.cs"
Class="HolaInternetService.Service1" %>
```

`Service1.asmx` es la página que pedirá un usuario desde un navegador. Ésta invoca al código del servicio, que se encuentra en el fichero indicado en `Codebehind`. `Service1` es la clase que representa al servicio y `HolaInternetService` es el `namespace` en el que está.

2) `Service1.asmx.cs`

Es el fichero con el código del servicio.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace HolaInternetService
{
	/// <summary>
	/// Summary description for Service1.
	/// </summary>
	public class Service1 : System.Web.Services.WebService
	{
		public Service1()
		{
			//CODEGEN: This call is required by the ASP.NET Web
			//Services Designer
			InitializeComponent();
		}
```

```
            #region Component Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent()
            {
            }
            #endregion

            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            protected override void Dispose( bool disposing )
            {
            }

            // WEB SERVICE EXAMPLE
            // The HelloWorld() example service returns the string
            //Hello World
            // To build, uncomment the following lines then save and
            //build the project
            // To test this web service, press F5

            //[WebMethod]
            //public string HelloWorld()
            //{
            //    return "Hello World";
            //}
        }
}
```

Los métodos del servicio no son accesibles desde el exterior (usuarios). Un usuario sólo puede acceder a aquellos métodos que hayan sido publicados, lo cual se hace precediéndolos del atributo `[WebMethod]`. Como se puede observar, el asistente no ha publicado método alguno, pero ha dejado un ejemplo entre comentarios (`HellowWorld`).

3) `Global.cs`

Este fichero contiene la clase `Global`, que tiene métodos de respuesta a ciertos eventos sobre objetos generales (`Application` y `Session`).

```
namespace HolaInternetService
{
    using System;
    using System.Collections;
    using System.ComponentModel;
    using System.Web;
    using System.Web.SessionState;


    /// <summary>
    ///    Summary description for Global.
    /// </summary>
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(Object sender, EventArgs e)
        {
```

```
        }

        protected void Session_Start(Object sender, EventArgs e)
        {

        }

        protected    void    Application_BeginRequest(Object    sender,
        EventArgs e)
        {

        }

        protected void Application_EndRequest(Object sender, EventArgs
        e)
        {

        }

        protected void Session_End(Object sender, EventArgs e)
        {

        }

        protected void Application_End(Object sender, EventArgs e)
        {

        }
    }
}
```

- Una vez se tiene el esqueleto del Servicio, se han de añadir los métodos de tipo `[WebMethod]` que se desee. En este caso se va a utilizar el método que por defecto añade el propio asistente como comentario.

```
//WEB SERVICE EXAMPLE
//The HelloWorld() example service returns the string Hello
//World
//To build, uncomment the following lines then save and build
//the project
//To test, right-click the Web Service's .asmx file and select
//View in Browser
//
[WebMethod]
public string HelloWorld()
{
    return "Hello World";
}
```

  Entre las propiedades más importantes del atributo `[WebMethod]` se encuentran las siguientes:
  - `CacheDuration`: permite especificar el número de segundos que la respuesta del método se mantendrá en la caché. El valor por defecto es 0, lo que quiere decir que la respuesta no se almacena en la caché.
    Ejemplo: `[WebMethod (CacheDuration = 10)]`
  - `Description`: descripción del método.

- `EnableSession`: permite activar (`true`) o desactivar (`false`) el almacenamiento del estado de la sesión.
- `MessageName`: permite cambiar el nombre con el que se expone el método hacia los consumidores.
- `TransactionOption`: Si se activa, todo el código del método se ejecuta en la misma transacción.
- `BufferResponse`: Si se activa (`true`), todos los datos se serializan y luego se envían de un golpe. Si los datos a enviar al cliente son bastantes, es mejor que su valor sea `false`, de este modo los datos serán enviados al cliente según se van serializando (poco a poco).

Utilización de un Web Service.

Para utilizar un Web Service, el modo básico es llamarlo desde un navegador a través de su página `.asmx` (en diseño se puede llamar desde la opción **Depurar**).
También puede utilizarse un Web Service desde un cliente de tipo WebForm o WinForm, pero hay que tener en cuenta algunos factores más.

En este caso, si se desea utilizar el Web Service directamente desde el navegador se ha de utilizar la siguiente URL:

```
http://localhost/HolaInternetService/Service1.asmx
```

El resultado es:

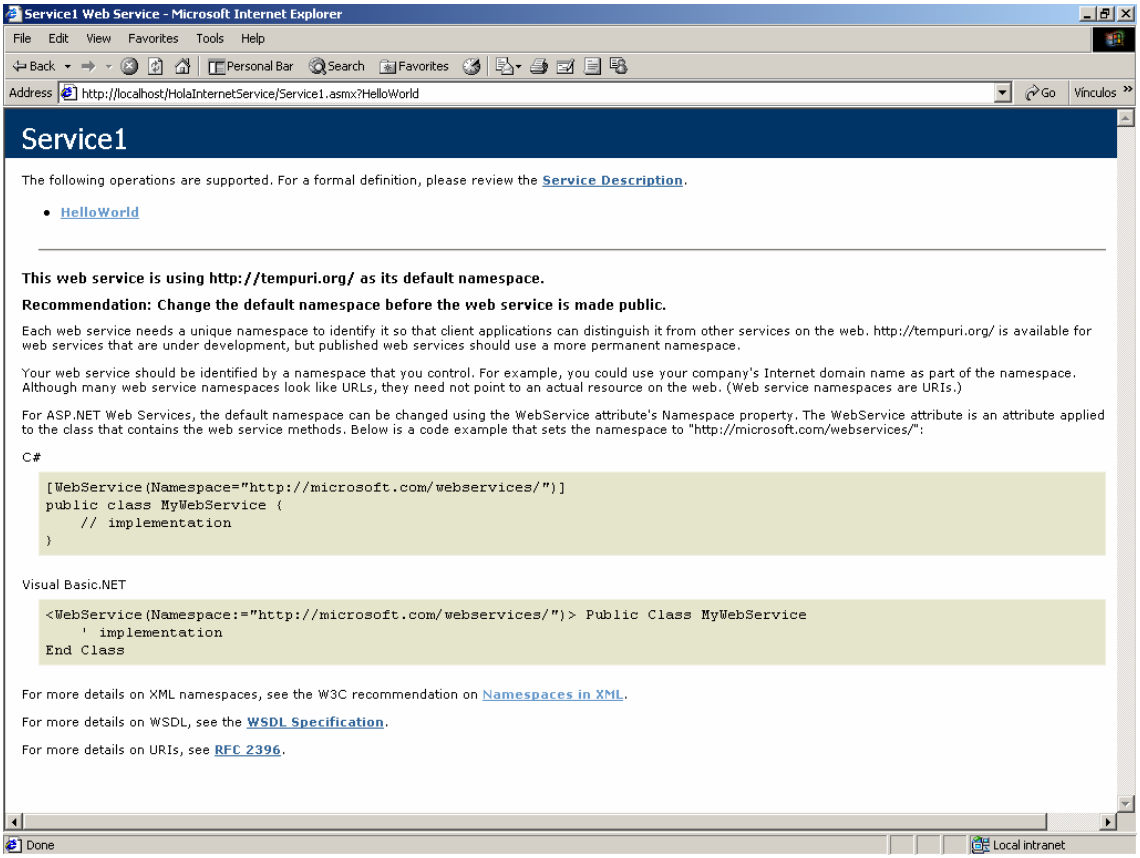

Figura 23.5. Resultado de invocar al servicio desde el Internet Explorer.

Por defecto, el Web Service devuelve una página Web encabezada con información sobre sus métodos públicos (en este caso sólo hay uno, `HelloWorld`). El resto de la información son explicaciones e información de ayuda.

Si se pulsa sobre el vínculo `HelloWorld` se muestra la página de la figura 23.6:

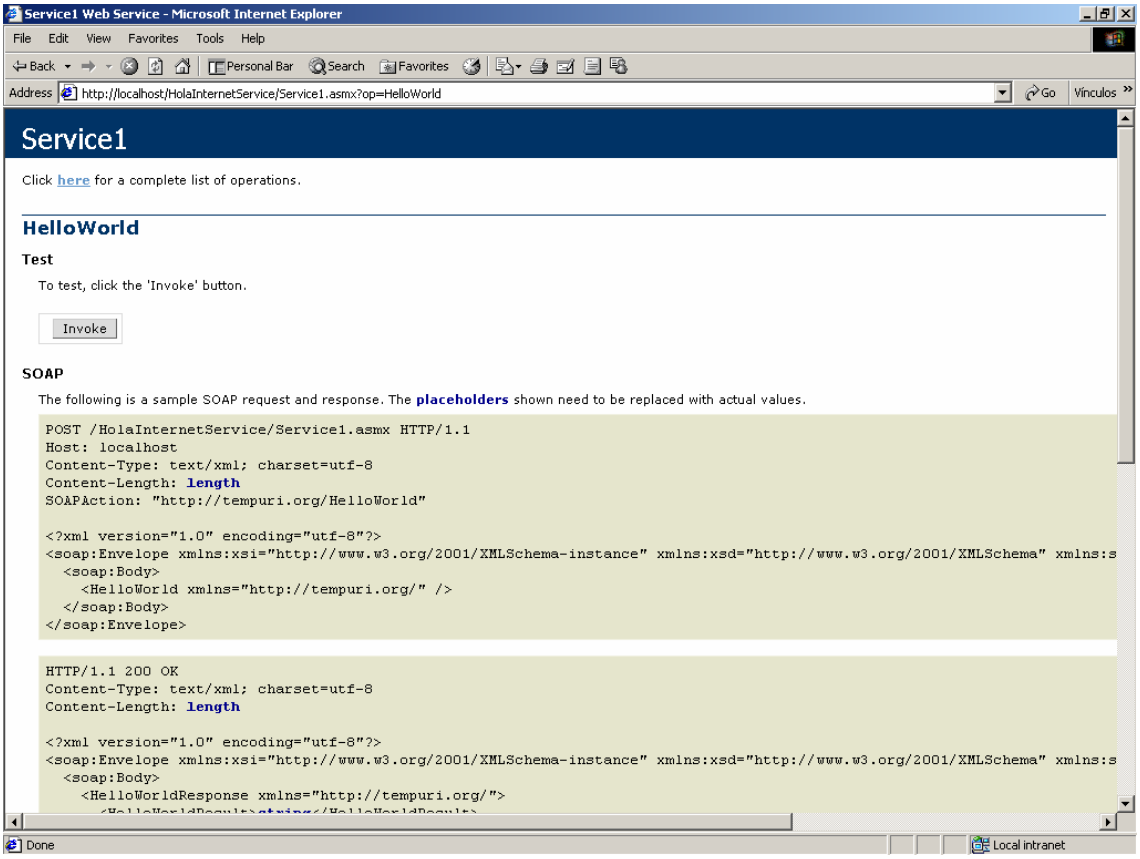

Figura 23.6. Página web de con información sobre el método `HelloWorld`. Ofrece la posibilidad de invocarlo.

Esta página se divide en tres partes fundamentales:

- Botón de comando para invocar al método `HelloWorld`.
- Ejemplo de petición (`request`) y respuesta (`response`) del método mediante SOAP.
- Ejemplo de petición (`request`) y respuesta (`response`) del método mediante HTTP GET y mediante HTTP POST.

Si se pulsa el botón `Invoque` se provoca una llamada al método `HelloWorld` y la respuesta está representada en la figura 23.7:

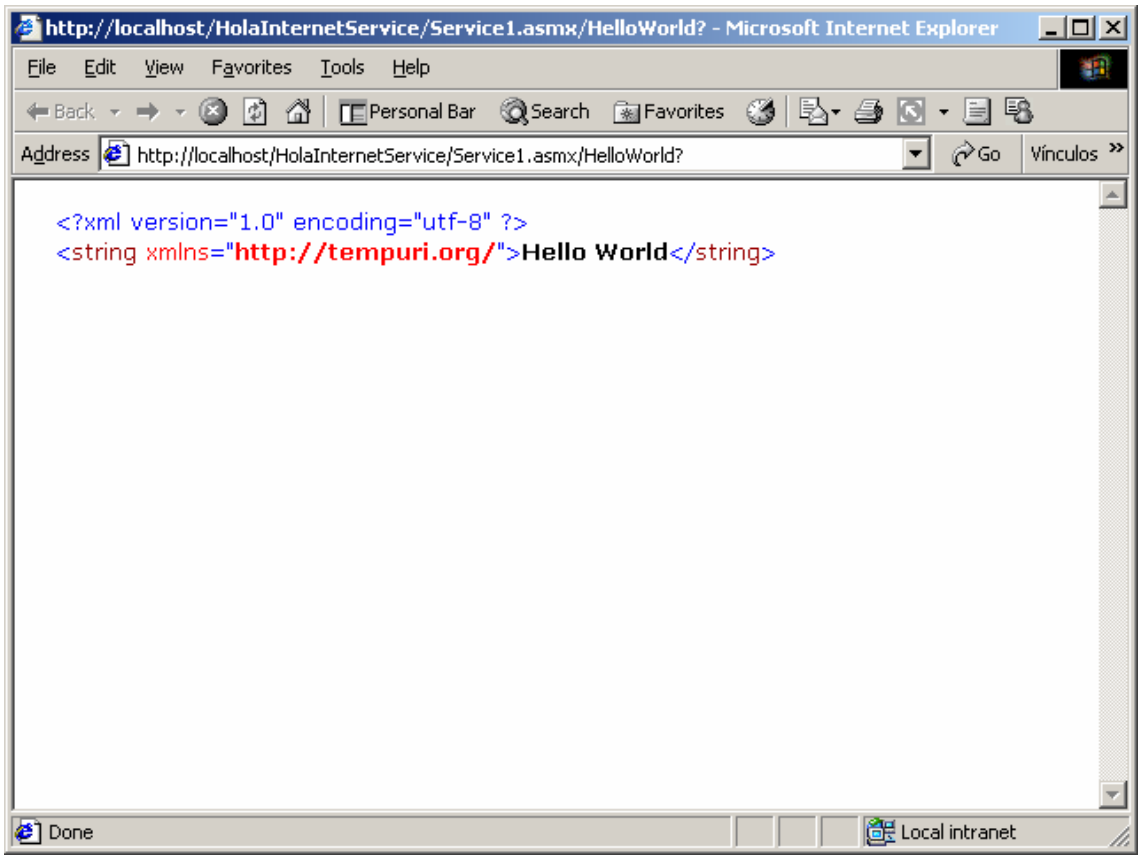

Figura 23.7. Resultado XML de invocar el método `HelloWorld` desde el Internet Explorer.

Puede verse que es la cadena `Hello World` en lenguaje XML.
Si se observa la dirección que aparece en **Dirección** se verá que es:

```
http://localhost/HolaInternetService/Service1.asmx/HelloWorld?
```

Esto es una petición http de tipo GET (generalmente se habla de método GET y método POST, pero en este caso puede llevar a confusión con los métodos reales de una clase).
Si se observa, en la página web anterior, se mostraba la estructura de una petición GET:

```
GET /HolaInternetService/Service1.asmx/HelloWorld? HTTP/1.1
Host: localhost
```

que coincide con la petición que aparece en el campo **Dirección** del navegador.

También se mostraba la estructura de la respuesta HTTP GET:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://tempuri.org/">string</string>
```

que también coincide con la estructura de la respuesta recibida en el navegador. Evidentemente, en el navegador no se ve todo el texto XML recibido.

Se puede concluir que la petición realizada al pulsar el botón `Invoke` y también la respuesta han seguido la estructura HTTP GET.

La petición y respuesta podían haber sido de tipo HTTP POST ó SOAP.

Una petición HTTP POST se suele hacer cuando se deben enviar bastantes datos (un formulario, por ejemplo). Es similar a GET pero separando los datos de la cabecera de la petición.

```
POST /HolaInternetService/Service1.asmx/HelloWorld HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length
```

La respuesta HTTP POST es:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://tempuri.org/">string</string>
```

Puede verse que coincide con la respuesta GET. Esto es así porque es la petición la que es GET o POST. La respuesta tiene la misma estructura en ambos casos.

La petición y respuesta SOAP son más complejas. SOAP (*Simple Object Access Protocol*) es un protocolo pensado para representar en modo XML llamadas a métodos y también respuestas. En concreto, la estructura de una llamada SOAP al método `HelloWorld` es:

```
POST /HolaInternetService/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/HelloWorld"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope         xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

Se puede observar que una llamada SOAP es una petición HTTP POST con un añadido. En concreto, en esta llamada se indica que se invoca a un método llamado `HelloWorld` y que no tiene parámetros.

La estructura de la respuesta SOAP es:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope         xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>string</HelloWorldResult>
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>
```

En esta estructura se indica que el método `HelloWorld` devuelve un `string`.

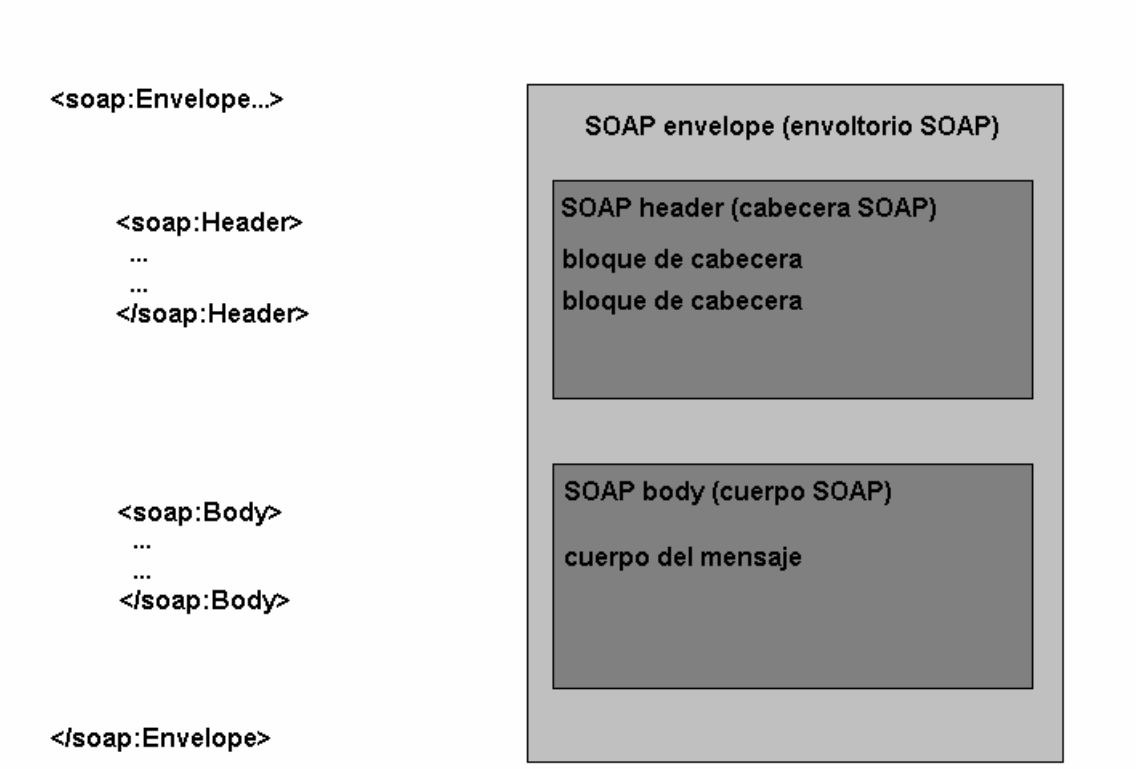

Figura 23.8. Estructura de un mensaje SOAP.

SOAP no especifica como puede ser dirigido (*enrutado*) el mensaje hacia un servicio de verificación de firma. Para resolver este problema, Microsoft ha definido el protocolo de enrutamiento SOAP, también llamado `WS-Routing`, que define un bloque de cabecera estándar que contiene la expresión con información de `routing`.

La información sobre estas estructuras y más está descrita en un fichero WSDL. La descripción WSDL del Web Service es accesible desde el hiperenlace `Service Description` de la primera página que muestra el servicio (antes del hiperenlace al método `HelloWorld`).

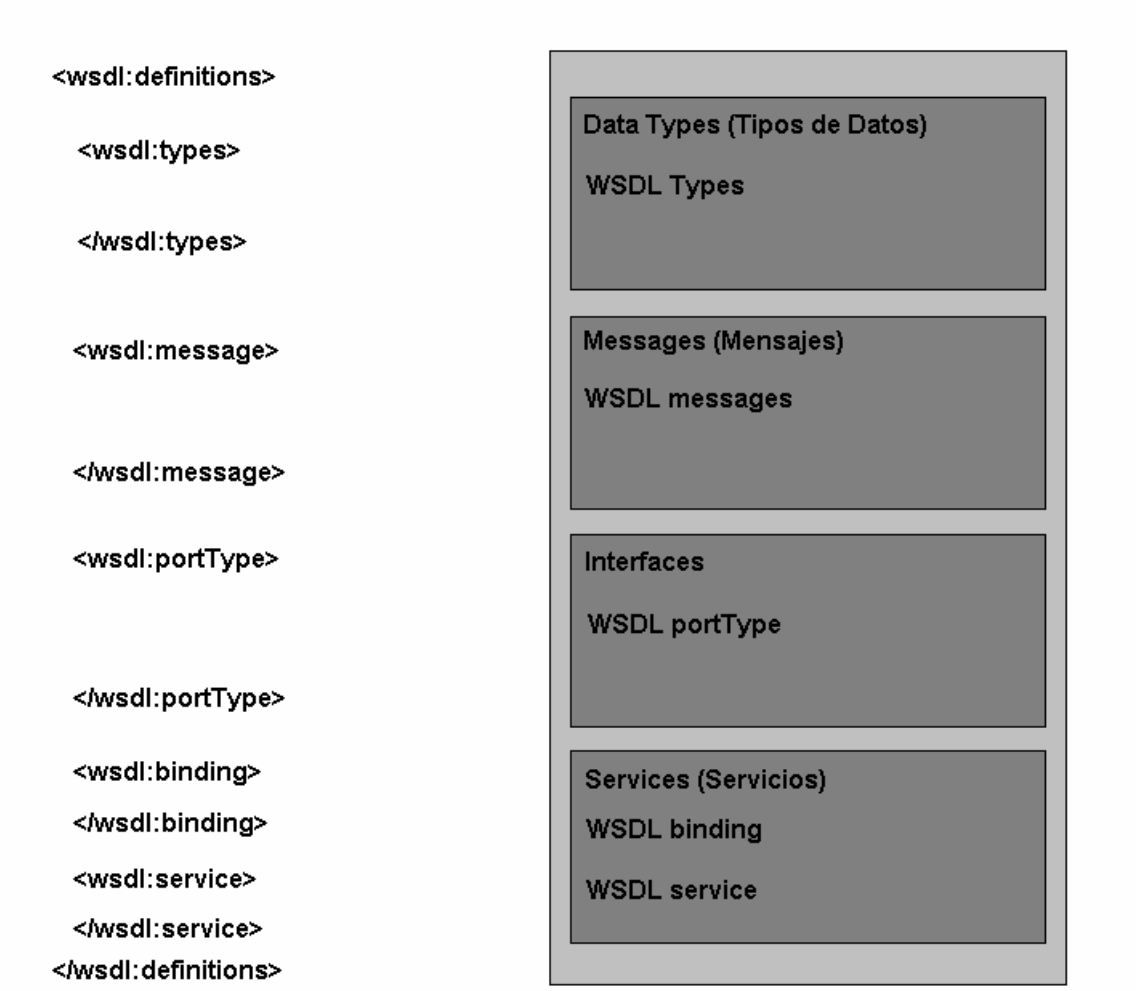

Figura 23.9. Estructura de la descripción WSDL de un Web Service (pueden ser varios).

De un modo sencillo puede decirse que un fichero WSDL describe uno o varios servicios. Un servicio es una colección de puertos (`ports`) o direcciones que implementan el servicio.

Por ejemplo:

```
<service name="Service1">
  - <port name="Service1Soap" binding="s0:Service1Soap">
                                                <soap:address
        location="http://localhost/HolaInternetService/Serv
        ice1.asmx" />
    </port>
  -              <port              name="Service1HttpGet"
      binding="s0:Service1HttpGet">
                                                <http:address
        location="http://localhost/HolaInternetService/Serv
        ice1.asmx" />
    </port>
  -              <port              name="Service1HttpPost"
      binding="s0:Service1HttpPost">
```

```
                                                    <http:address
        location="http://localhost/HolaInternetService/Serv
        ice1.asmx" />
    </port>
</service>
```

Un puerto consta de:

- Definición abstracta (`portType`): especifica el interface software del puerto, que es un conjunto de operaciones que definen los intercambios ordenados de mensajes (`messages`). Por ejemplo:

```
<portType name="Service1Soap">
        <operation name="HelloWorld">
                <input message="s0:HelloWorldSoapIn" />
                <output message="s0:HelloWorldSoapOut" />
        </operation>
</portType>
```

- Definición concreta (`binding`): especifica qué protocolos se utilizan por cada puerto. Por ejemplo:

```
<binding name="Service1Soap" type="s0:Service1Soap">
        <soap:binding
        transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
        <operation name="HelloWorld">
                <soap:operation
                soapAction="http://tempuri.org/HelloWorld"
                style="document" />
                <input>
                <soap:body use="literal" />
                </input>
                <output>
                <soap:body use="literal" />
                </output>
        </operation>
</binding>
```

Un mensaje (`message`) es una colección de partes (`part`) con nombre de un tipo particular. El tipo de una parte se define utilizando algún mecanismo de tipado de datos estándar, como por ejemplo, la especificación XML Schema.

Por ejemplo:

```
<message name="HelloWorldSoapIn">
    <part name="parameters" element="s0:HelloWorld" />
</message>
<message name="HelloWorldSoapOut">
    <part                                   name="parameters"
    element="s0:HelloWorldResponse" />
</message>
```

El fichero WSDL de este ejemplo es:

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:s0="http://tempuri.org/"
    targetNamespace="http://tempuri.org/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema attributeFormDefault="qualified"
        elementFormDefault="qualified"
        targetNamespace="http://tempuri.org/">
      - <s:element name="HelloWorld">
          <s:complexType />
        </s:element>
      - <s:element name="HelloWorldResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="1" maxOccurs="1"
                name="HelloWorldResult" nillable="true"
                type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:element name="string" nillable="true" type="s:string"
          />
      </s:schema>
    </types>
  - <message name="HelloWorldSoapIn">
      <part name="parameters" element="s0:HelloWorld" />
    </message>
  - <message name="HelloWorldSoapOut">
      <part name="parameters" element="s0:HelloWorldResponse"
        />
    </message>
    <message name="HelloWorldHttpGetIn" />
  - <message name="HelloWorldHttpGetOut">
      <part name="Body" element="s0:string" />
    </message>
    <message name="HelloWorldHttpPostIn" />
  - <message name="HelloWorldHttpPostOut">
      <part name="Body" element="s0:string" />
    </message>
  - <portType name="Service1Soap">
    - <operation name="HelloWorld">
        <input message="s0:HelloWorldSoapIn" />
        <output message="s0:HelloWorldSoapOut" />
      </operation>
    </portType>
```

```
- <portType name="Service1HttpGet">
  - <operation name="HelloWorld">
      <input message="s0:HelloWorldHttpGetIn" />
      <output message="s0:HelloWorldHttpGetOut" />
    </operation>
  </portType>
- <portType name="Service1HttpPost">
  - <operation name="HelloWorld">
      <input message="s0:HelloWorldHttpPostIn" />
      <output message="s0:HelloWorldHttpPostOut" />
    </operation>
  </portType>
- <binding name="Service1Soap" type="s0:Service1Soap">
                                                    <soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
  - <operation name="HelloWorld">
                                                <soap:operation
        soapAction="http://tempuri.org/HelloWorld"
        style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
- <binding name="Service1HttpGet" type="s0:Service1HttpGet">
    <http:binding verb="GET" />
  - <operation name="HelloWorld">
      <http:operation location="/HelloWorld" />
    - <input>
        <http:urlEncoded />
      </input>
    - <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
- <binding name="Service1HttpPost" type="s0:Service1HttpPost">
    <http:binding verb="POST" />
  - <operation name="HelloWorld">
      <http:operation location="/HelloWorld" />
    - <input>
          <mime:content    type="application/x-www-form-
          urlencoded" />
      </input>
    - <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
```

```
- <service name="Service1">
  - <port name="Service1Soap" binding="s0:Service1Soap">
                                                    <soap:address
        location="http://localhost/HolaInternetService/Serv
        ice1.asmx" />
    </port>
  -                 <port                 name="Service1HttpGet"
      binding="s0:Service1HttpGet">
                                                    <http:address
        location="http://localhost/HolaInternetService/Serv
        ice1.asmx" />
    </port>
  -                 <port                 name="Service1HttpPost"
      binding="s0:Service1HttpPost">
                                                    <http:address
        location="http://localhost/HolaInternetService/Serv
        ice1.asmx" />
    </port>
  </service>
</definitions>
```

Estas estructuras son las que determinan cómo funciona por debajo la invocación de métodos de Web Services pero no es necesario utilizarlas directamente. Cuando se quiera invocar un método de un Web Service desde una aplicación (consumidora de servicios) se utilizará una clase (`proxy`) que encapsulará el funcionamiento real de la llamada, ofreciendo a la aplicación consumidora la sensación de que el Web Service es una clase local.

Creación de una aplicación consumidora de un servicio.

El modo normal de acceder a un Web Service es desde el código de una aplicación consumidora del Web Service.

Los pasos a seguir para utilizar un Web Service desde una aplicación consumidora son:

- Crear una clase `proxy` para acceder al Web Service.
- Instanciar y utilizar la clase.

Supónga que se desea crear una aplicación Web (Web Forms) llamada `PruebaWebService1` que conste de una página web (`.aspx`) con un botón de comando y una caja de texto, de modo que al pulsar el botón de comando se invoque al método `HelloWorld` del servicio `Service1` (aplicación `HolaInternetService`) y el resultado devuelto por tal método se muestre en la caja de texto.

La página Web será algo así:

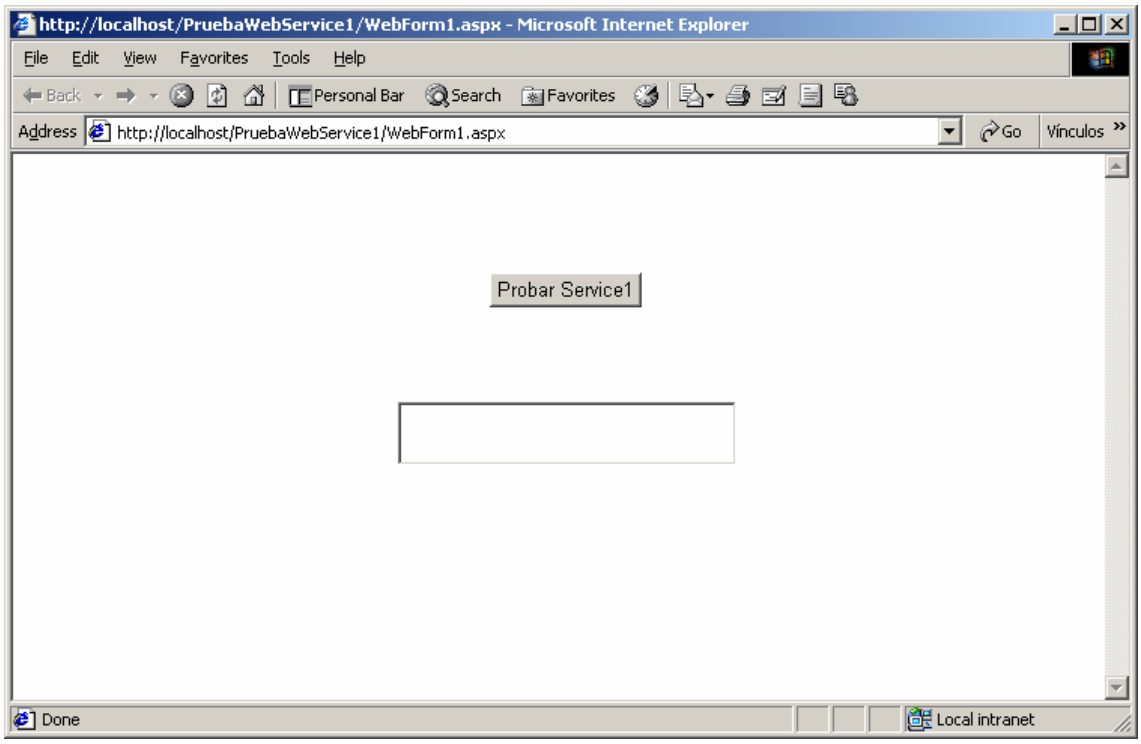

Figura 23.10. Página WebForm para invocar el método `HelloWorld` del servicio.

Cuando se pulse el botón `Probar Service1` se ejecutará el código del método manejador de su evento `Click`. Este código debe invocar al método `HelloWorld` del servicio `Service1`.

Crear una clase proxy para acceder al Web Service.

Para poder invocar a un método de un servicio, la aplicación desde la que se hace la invocación (aplicación cliente) debe disponer de un *proxy* para tal servicio.

Un *proxy* es un objeto local a la aplicación cliente cuya estructura (métodos...) es igual a la del servicio que se desea utilizar. De este modo, la aplicación cliente utilizará el *proxy* pensando que es el servicio y el *proxy* se encargará de comunicarse con el servicio e invocar sus métodos mediante SOAP o HTTP.

Para crear el *proxy* hay que registrar el servicio en la aplicación cliente, lo cual se hace mediante la opción **Añadir Referencia Web** del menú **Proyecto**. Al hacerlo se muestra un cuadro de diálogo con una caja de texto en la que se debe introducir la ruta al fichero `.vsdisco` correspondiente al servicio `Service1`.

Para este ejemplo, la ruta es:

`http://localhost/HolaInternetService1/HolaInternetService1.vsdisco.`

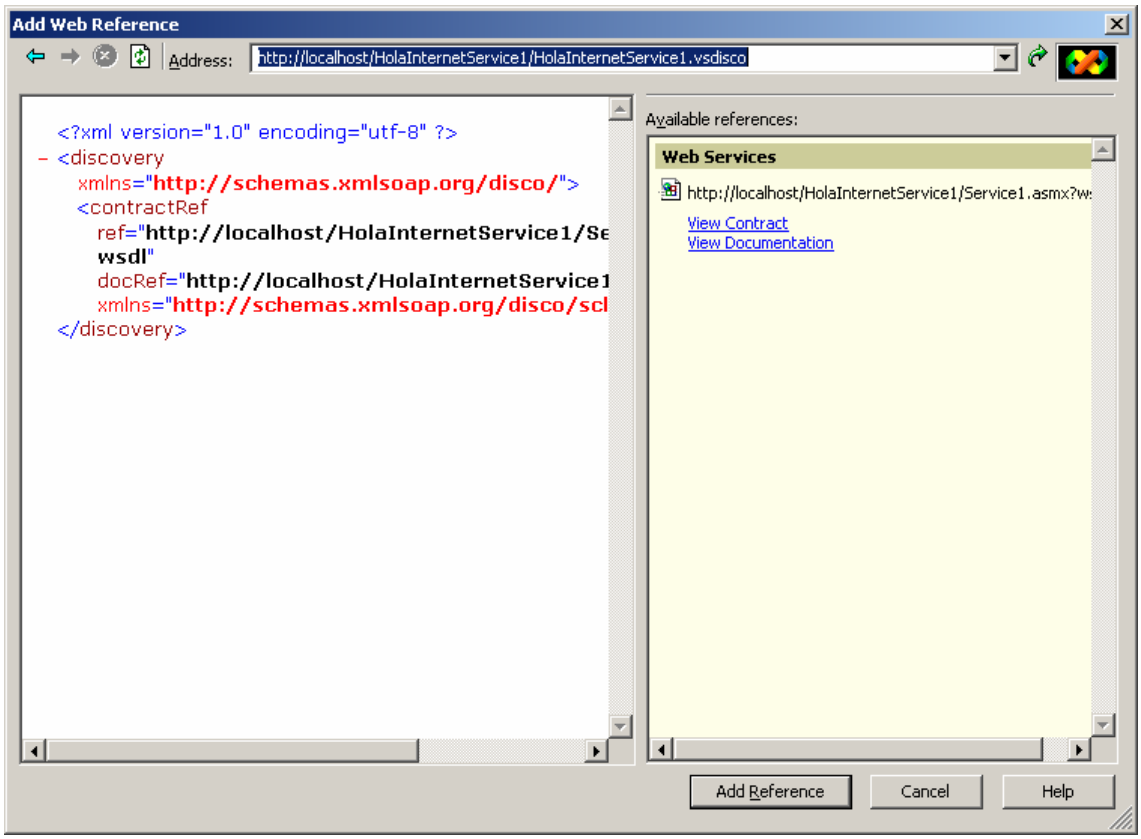

Figura 23.11. Adición a la aplicación cliente de una referencia al Web Service utilizando los servicios de descubrimiento (`discovery`).

Como puede observarse, en la ventana de la izquierda se muestra el contenido del fichero `HolaInternetService1.vsdisco`. En la ventana de la derecha se muestra de un modo más elegante el contenido del mismo fichero.

$$$$ (Ver la traducción de estas opciones en la versión definitiva)

Si se selecciona la opción **view contract**, se mostrará el contenido del fichero `Service1.wsdl` en la ventana izquierda. Si se selecciona la opción **view documentation** se mostrará la página correspondiente a la URL `http://localhost/HolaInternetService/Service1.asmx` (ya vista anteriormente).

Para añadir una referencia al servicio `Service1` se ha de pulsar el botón **Agregar Referencia**. Al hacerlo se añade al proyecto una *Web Reference* o referencia al web site en el que está el servicio. En este caso la referencia es `localhost` (que es el site local por defecto) y contiene a su vez información sobre `Service1`.

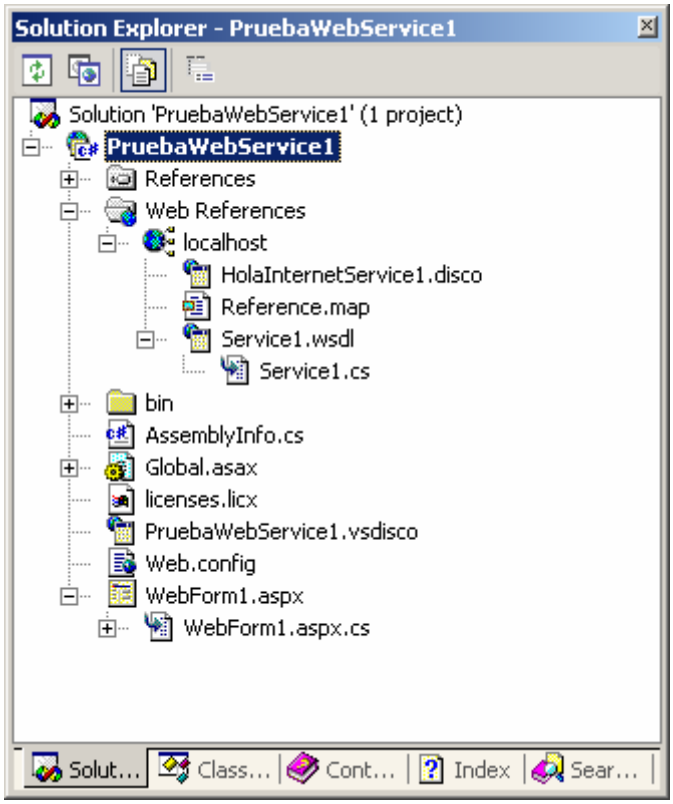

Figura 23.12. Ventana del **explorador de soluciones**. Se puede observar la referencia al servicio Web.

Lo anterior se debe entender como que "el servicio `Service1` es una clase que pertenece al `namespace localhost`".

Instanciar y utilizar la clase.

Una vez se tiene la referencia al servicio, es decir, una vez se tiene el *proxy* local, es posible instanciarlo y utilizarlo del mismo modo que se haría con cualquier otra clase. Puesto que el servicio, desde el punto de vista del cliente, pertenece al `namespace localhost`, habrá que añadir al código la línea:

```
using PruebaWebService1.localhost;
```

donde `PruebaWebService` es el `namespace` del cliente.

El siguiente paso es utilizar la clase `Service1`. Puesto que se desea que el método `HelloWorld` del servicio `Service1` sea invocado cuando se pulse el botón `ProbarService1` (`Button1`) y el resultado que devuelva aparezca en la caja de texto, ha de incluirse el siguiente código en el método `Button1_Click`:

```
private void Button1_Click(object sender, System.EventArgs e)
{
      Service1 serv1 = new Service1();
      TextBox1.Text = serv1.HelloWorld();
}
```

Puesto que `Service1` es conocido por el cliente -también en tiempo de diseño, que es cuando ha sido descubierto- el asistente muestra ayuda contextual mientras se escribe.

El resultado de ejecutar la aplicación cliente y pulsar el botón `Probar Servicio1` será invocar al Web Service `Service1`, en concreto al método `HelloWorld` y mostrar su resultado en la caja de texto (`TextBox1`).

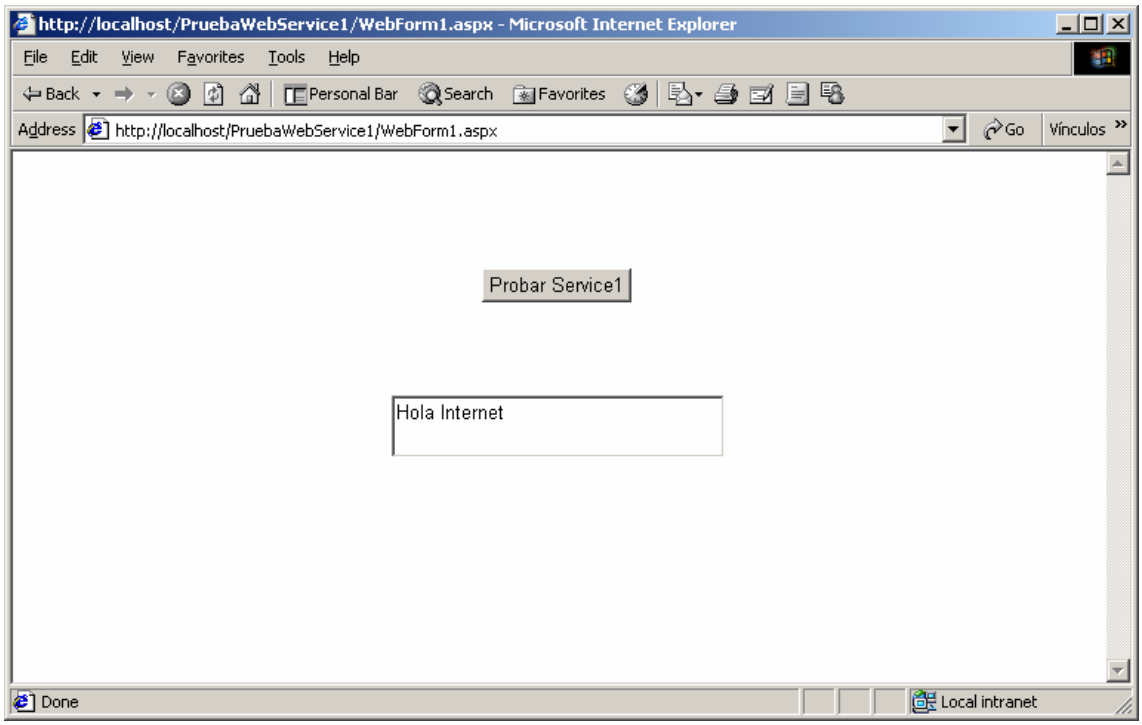

Figura 23.13. Resultado de invocar el método `HelloWorld` del Web Service desde la aplicación cliente.