

Servicios WCF

El servicio de mensajería entre aplicaciones se realizaba antiguamente mediante protocolos COM, DCOM o MSQM, que obligaba a los programadores a ceñirse no sólo a una forma de programación concreta, sino que también estaba atada a la plataforma y al lenguaje de programación. Los servicios web surgen con el propósito de cambiar esta filosofía, permitiendo hacer la comunicación **independiente de lenguaje de programación y plataforma** gracias a la creación de estándares de comunicación.

No importará si un servicio está codificado en **Java** o **.NET**, ni si corre en una plataforma **Windows** o **Linux**: lo importante será que respeten los estándares del protocolo sobre el cual está construido, del mismo modo que ocurre con los servidores web.

Tipos de servicios web

A día de hoy existe una gran variedad de protocolos sobre los que los servicios web pueden operar, pero son dos los protocolos *estrella* que la práctica totalidad de los servicios web utilizan hoy día:

- **SOAP**: *Simple Object Access Protocol*. Creado en 1998, se sirve de mensajes XML para el intercambio de mensajes. Puede operar sobre cualquier protocolo de transporte, aunque lo más común es que lo haga sobre HTTP o HTTPS. Es el protocolo más común en servicios web de carácter privado.
- **REST**: *REpresentational State Transfer*. Concepto surgido en el año 2000, hace uso del protocolo HTTP para el envío de mensajes, y puede utilizar lenguajes como XML o JSON.

SOAP

Este protocolo, como ya hemos dicho, utiliza XML como lenguaje de codificación, y su principal ventaja es que **puede operar sobre cualquier protocolo de la capa de transporte**, es decir, que puede ser transmitido a través de HTTP, TCP/IP, SMTP o cualquier otro, a diferencia de REST que únicamente opera sobre HTTP/HTTPS.

Su estructura recuerda vagamente a la de un documento HTML, constando de una cabecera (**header**) que implementará los metadatos del mensaje (opciones de seguridad, protocolo, transaccionalidad...) y de un cuerpo (**body**) que contendrá el mensaje en sí. Todo ello irá encapsulado dentro de un sobre (**envelope**). Un ejemplo de mensaje SOAP podría ser el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none" />
  </s:Header>
  <s:Body>
    <ObtenerProductos xmlns="http://tempuri.org/" />
  </s:Body>
</s:Envelope>
```

REST

Su filosofía se basa en la ausencia de estado y la “equivalencia” entre los cuatro verbos que pueden utilizarse en el protocolo HTTP y las cuatro operaciones CRUD (Create, Read, Update, Delete) básicas que pueden realizarse sobre una fuente de datos. Es el protocolo más utilizado en servicios web abiertos o públicos.

- **GET** ↔ **SELECT** (Obtener)
- **POST** ↔ **INSERT** (Crear)
- **PUT** ↔ **UPDATE** (Actualizar)
- **DELETE** ↔ **DELETE** (Borrar)

SOA

SOA son las siglas de *Service-Oriented Architecture*, es decir, *Arquitectura Orientada a Servicios*.

SOA se basa en organizar una aplicación en **unidades funcionales** que pueden ser gestionadas por distintos proveedores e incluso compañías de modo que puedan ser accedidas de un modo homogéneo. Por lo tanto, SOA no habla acerca de la tecnología a utilizar, como REST o SOAP, sino lo que conceptualmente debería ser un modelo de comunicaciones distribuido.

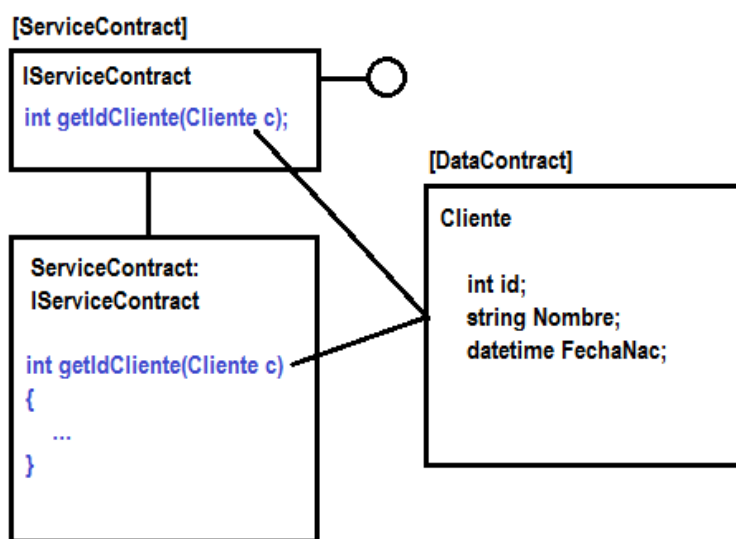
Windows Communication Foundation (WCF)

Tras una serie de acercamientos a los servicios web utilizando SOAP y servicios web de extensión .asmx, Microsoft publica finalmente *Windows Communication Foundation*. Su objetivo básico era el de **unificar** las comunicaciones. Ya no importará que nuestra aplicación distribuida se comunique a través de TCP en unos servicios, SOAP en otros y REST en otros: WCF soporta todos estos protocolos, y permitirá que nuestro código **sea independiente del protocolo que vayamos a utilizar**.

¿Cómo logra WCF esta independencia? Básicamente, mediante la separación entre **operaciones** y **datos**. Al igual que hacemos uso de interfaces en nuestro código para aislar la firma de los métodos de sus implementaciones, un servicio web WCF utiliza este mismo concepto: establece un **contrato** a través de una interfaz (que será adornado con atributos específicos) que una clase se encargará de implementar.

De este modo, un servicio WCF, se compondrá, a grandes rasgos, de:

- **Contrato de servicio** (ServiceContract): expone una operación que nuestro servicio web es capaz de ejecutar. Corresponde a una interfaz.
- **Contrato de datos** (DataContract): implementa un tipo de dato que el servicio web será capaz de manejar. Generalmente, será el tipo de dato que manejará el contrato de servicio.
- **Implementación del servicio**: implementará la interfaz correspondiente al contrato de servicio, haciendo uso del contrato de datos para intercambiar la información.



En cuanto a la comunicación, se realizará mediante la configuración de los denominados **endpoints**, a través de un simple fichero XML. Un *endpoint* no es más que la dirección de un servicio, que implementa un protocolo determinado. Así, en el siguiente ejemplo tendríamos un único servicio que puede comunicarse de tres formas distintas a través de tres *endpoints* diferentes:

El primer *endpoint* realizará su comunicación a través de HTTP utilizando REST. Los otros dos endpoints realizarán el mismo trabajo que el primero, pero usarán los protocolos TCP y MSMQ utilizando SOAP. Como podemos observar, **tan sólo ha sido necesario codificar una sola vez la funcionalidad del servicio web**, creando en un momento tres formas distintas de realizar la comunicación.

Con esto podemos entender el éxito de WCF en la plataforma .NET: permite olvidarse del transporte y centrarse en la codificación de la lógica del servicio, ahorrando una enorme cantidad de tiempo y recursos y simplificando enormemente la

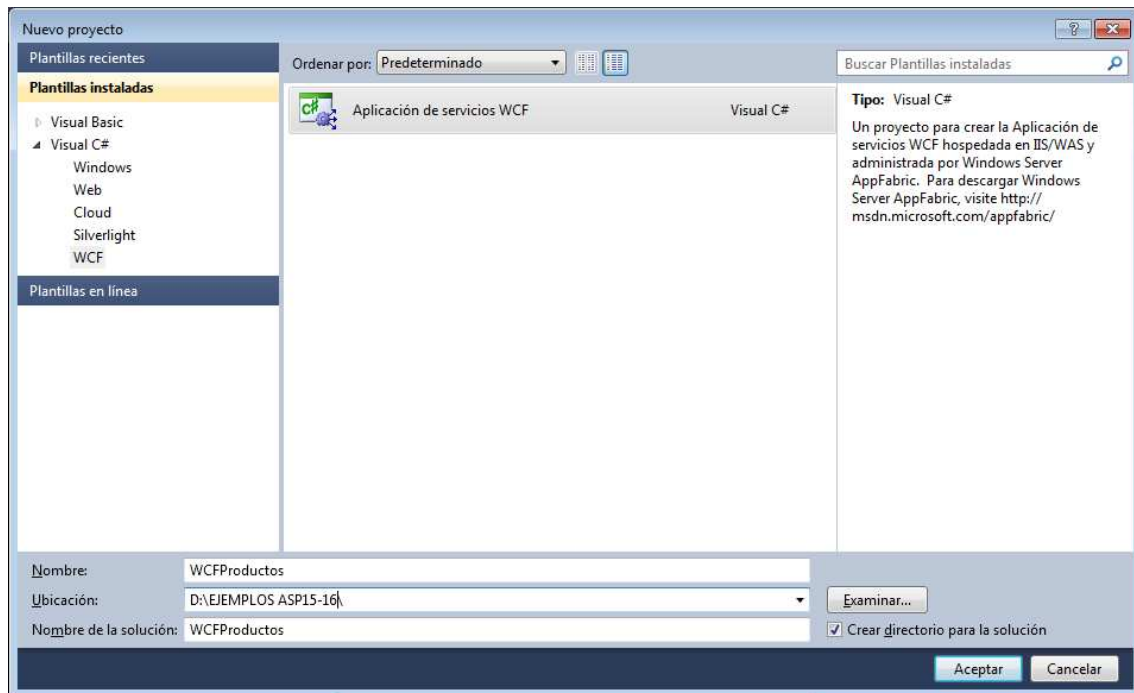
interoperabilidad, ya que, si en un determinado momento es necesario un protocolo determinado para comunicarse con un cliente preexistente, basta con configurar un endpoint para que ese cliente pueda acceder a nuestro servicio en lugar de codificar toda la lógica de comunicaciones que implicaría un servicio web convencional.

```
1      <?xml version='1.0' ?>
2      <configuration>
3          <system.serviceModel>
4              <services>
5                  <service name='GestionPedidosDataService'>
6                      <endpoint
7                          address='http://localhost:6040/GestionPedidosHttpService'
8                          binding='webHttpBinding'
9                          contract='IServiceContract'
10                     />
11                     <endpoint
12                         address='net.tcp://localhost:9043/GestionPedidosTcpService'
13                         binding='netTcpBinding'
14                         contract='IServiceContract'
15                     />
16                     <endpoint
17                         address='net.msmq://localhost/GestionPedidosMsmqService'
18                         binding='netMsmqBinding'
19                         contract='IServiceContract'
20                     />
21                 </service>
22             </services>
23         </system.serviceModel>
24     </configuration>
```

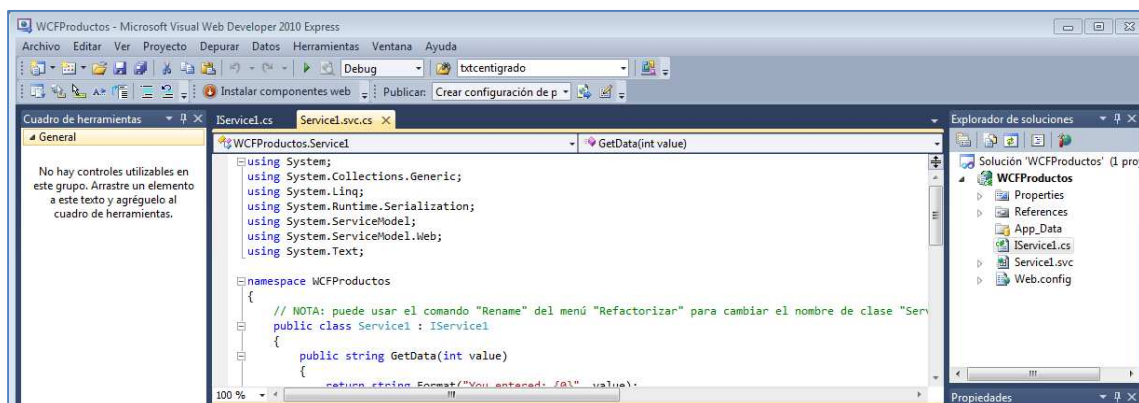
Creación de un Servicio WCF

1

..Crearemos un **Nuevo Proyecto** → **Aplicación de servicios WCF**, dado que en nuestras **versiones Express** no poseemos la opción de Biblioteca de Servicios WCF:

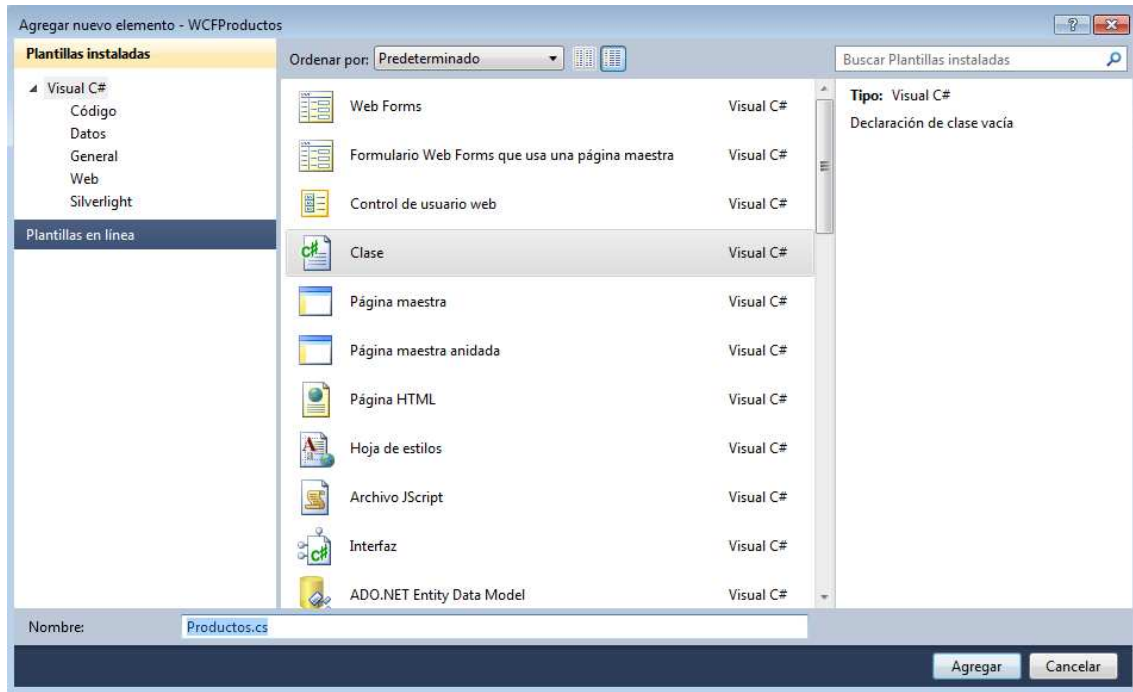


Visual Studio creará ahora un esqueleto de nuestro primer servicio web, que tendrá un aspecto similar al siguiente:



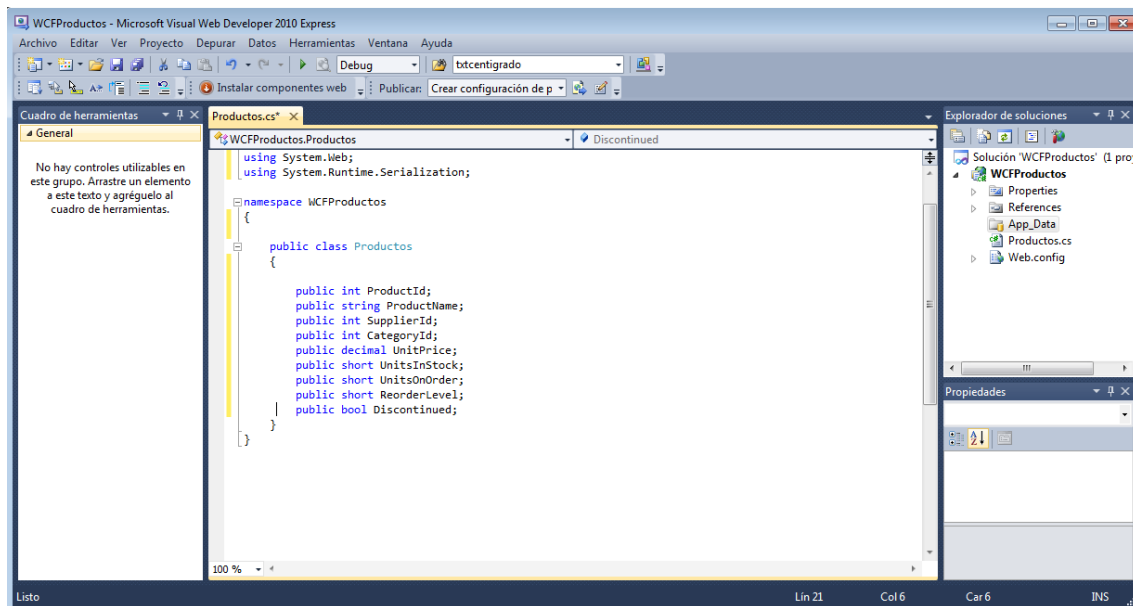
Contrato de datos (DataContract)

Si creamos una Biblioteca de Servicios Web, añadiremos una nueva clase Productos.cs, que simbolizará la estructura de datos que nuestro servicio web intercambiará con un potencial cliente: el **DataContract** o contrato de datos.

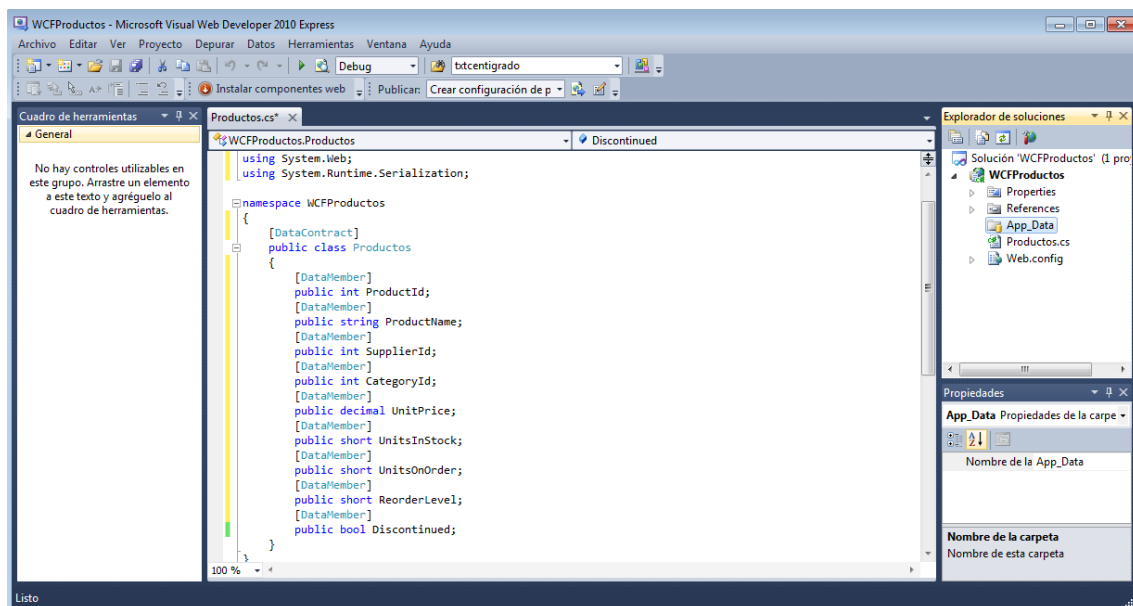


Esta información contendrá los campos que queremos enviar a través de la red. No es más que una estructura simple, sin funcionalidad, que contiene una serie de campos que normalmente simbolizará un registro en una base de datos relacional desde una perspectiva orientada a objetos.

Por tanto, haremos la clase pública y añadiremos los campos que queremos exponer, que también tendrán visibilidad pública.



Para que nuestros datos adquieran la categoría de contrato de datos, será necesario adornar la clase y sus elementos con un par de atributos heredados de *System.Runtime.Serialization*. Por lo tanto, añadimos el *using* correspondiente y asignamos el atributo *DataContract* a la clase y el atributo *DataMember* a cada uno de los elementos de la clase.



En la aplicación de Servicio WCF se crea dentro del interfaz.

```
Web.config | IProductService.cs | ProductoService.svc.cs
WcfService1.IProductoService
eliminarProducto(int id)

public interface IProductoService
{
    [OperationContract]
    List<Productos> ObtenerProductos();
    [OperationContract]
    void insertarProducto(Productos p);
    [OperationContract]
    void modificarProducto(Productos p);
    [OperationContract]
    void eliminarProducto(int id);
}

[DataContract]
public class Productos
{
    [DataMember]
    public int ProductId;
    [DataMember]
    public string ProductName;
    [DataMember]
    public int SupplierId;
    [DataMember]
    public int CategoryId;
    [DataMember]
    public decimal UnitPrice;
    [DataMember]
    public short UnitsInStock;
    [DataMember]
    public short UnitsOnOrder;
    [DataMember]
    public short ReorderLevel;
    [DataMember]
    public bool Discontinued;
}
```



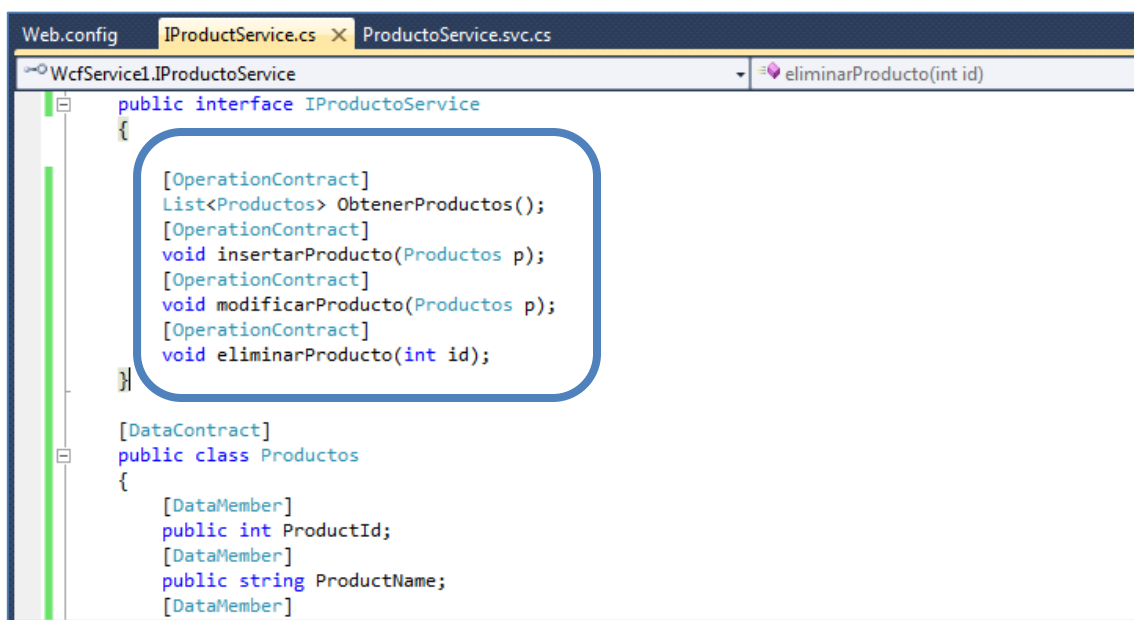
```
Web.config | IProductService.cs | ProductoService.svc.cs
WcfService1.IProductoService

public class Productos
{
    [DataMember]
    public int ProductId;
    [DataMember]
    public string ProductName;
    [DataMember]
    public int SupplierId;
    [DataMember]
    public int CategoryId;
    [DataMember]
    public decimal UnitPrice;
    [DataMember]
    public short UnitsInStock;
    [DataMember]
    public short UnitsOnOrder;
    [DataMember]
    public short ReorderLevel;
    [DataMember]
    public bool Discontinued;
}
```


Contrato de Servicio (ServiceContract)

El segundo elemento que debemos configurar es el contrato de servicio, la interfaz que contendrá la firma de los métodos que el servicio expone de forma pública. Cambiaremos de nombre a la Interfaz denominándola `IProductoService`.

Quitaremos los elementos que aparecen de prueba y añadiremos la firma de los métodos que queremos que esta incluya. Para este caso concreto, crearemos un método de recuperación, otro de inserción, otro de modificación y un último método de eliminación. Recordemos que en una interfaz no debemos indicar la visibilidad de los métodos que ésta incluye.



Por último, importaremos el espacio de nombres `System.ServiceModel` y adornaremos la interfaz con el atributo `ServiceContract`. Los métodos de la interfaz, por su parte, irán adornados con el atributo `OperationContract`, ya que establecerán el contrato de las operaciones que el servicio puede realizar.

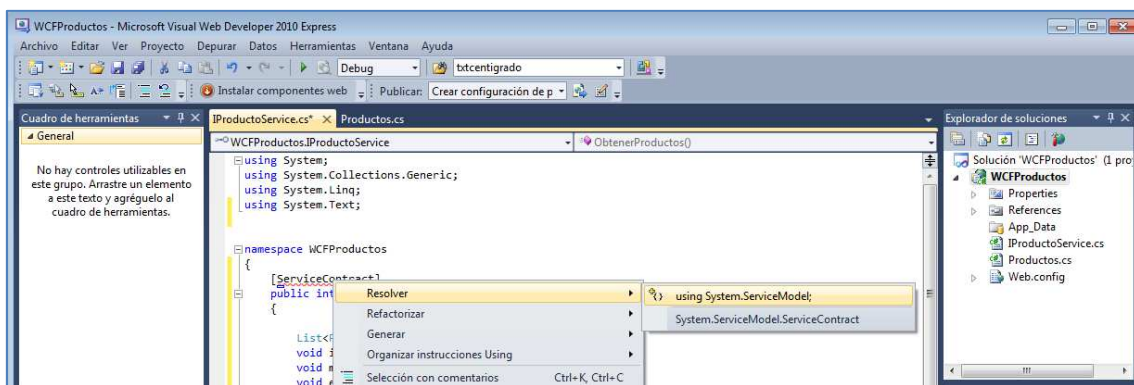
```

[ServiceContract]
public interface IProductoService
{

    [OperationContract]
    List<Productos> ObtenerProductos();
    [OperationContract]
    void insertarProducto(Productos p);
    [OperationContract]
    void modificarProducto(Productos p);
    [OperationContract]
    void eliminarProducto(int id);
}

```

Si nos posicionamos encima del *adorno* `[ServiceContract]` pulsaremos la opción Resolver del menú contextual y veremos cuál es el espacio de nombres a importar, para que el sistema reconozca los adornos.



Implementación del contrato

Contando con datos y operaciones, es el momento de codificar una clase que implemente la interfaz que define el contrato de servicio con sus correspondientes operaciones. Esta clase tiene una extensión `svc`. Le cambiaremos el nombre denominándola *ProductoService*, que implemente la interfaz *IProductoService*.

Para que implemente la Interfaz *IProductoService* se añadirá tras la clase `:IProductoService`.

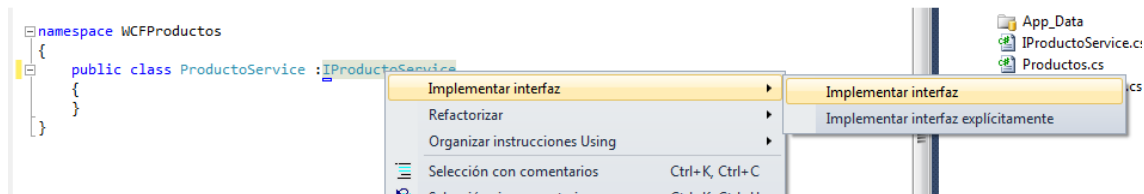
Nos fijaremos que así esté escrito en la clase:

```

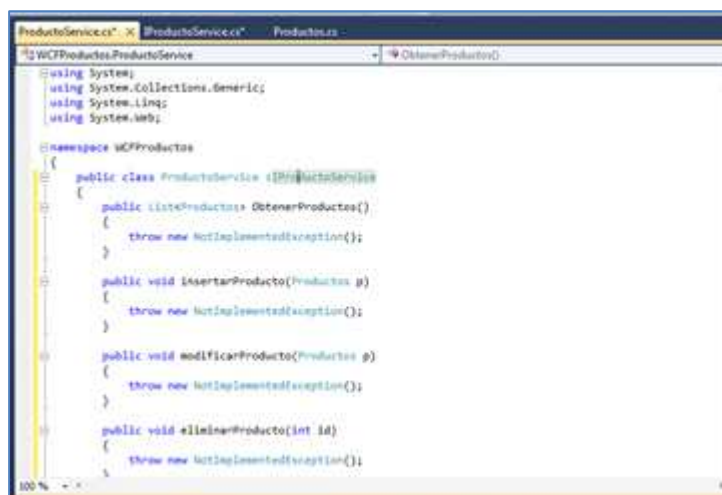
public class ProductoService : IProductoService
{

```

Para que Visual Studio genere el cuerpo de las funciones que debemos implementar, haremos clic derecho sobre el nombre de la interfaz y seleccionaremos la opción *Implement Interface*.

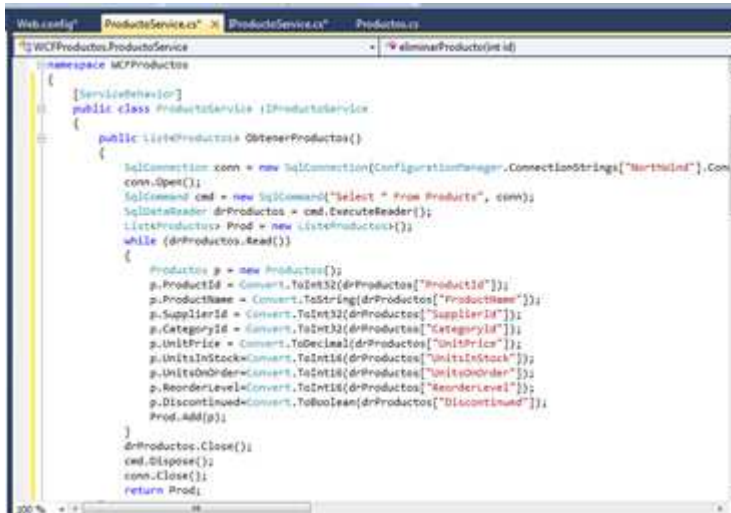


Esto hará que se genere el código correspondiente a la firma de los métodos de la interfaz, indicándonos qué es lo que debemos codificar.



Nuevamente, importaremos el espacio de nombres *System.ServiceModel* y asignaremos el atributo *ServiceBehavior* a la clase.

Y finalmente, añadimos la funcionalidad deseada a cada método. Por ejemplo, codificaríamos algo como lo siguiente:



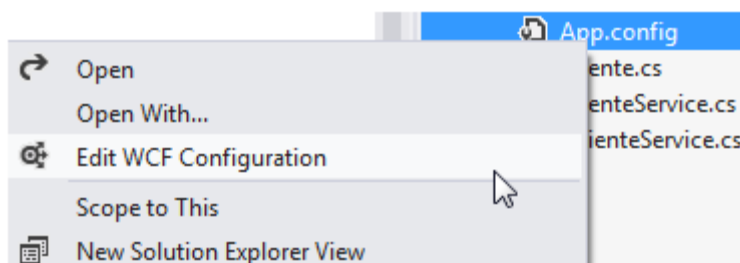
```
Web.config | ProductService.cs | ProductService.svc | Product.cs
WCFProduct.ProductService
namespace WCFProduct
{
    [ServiceBehavior]
    public class ProductService : IProductService
    {
        public IList<Product> GeteterProductos()
        {
            SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);
            conn.Open();
            SqlCommand cmd = new SqlCommand("select * from Products", conn);
            SqlDataReader drProductos = cmd.ExecuteReader();
            IList<Product> Prod = new List<Product>();
            while (drProductos.Read())
            {
                Product p = new Product();
                p.ProductId = Convert.ToInt32(drProductos["ProductId"]);
                p.ProductName = Convert.ToString(drProductos["ProductName"]);
                p.SupplierId = Convert.ToInt32(drProductos["SupplierId"]);
                p.CategoryId = Convert.ToInt32(drProductos["CategoryId"]);
                p.UnitPrice = Convert.ToDecimal(drProductos["UnitPrice"]);
                p.UnitsInStock = Convert.ToInt32(drProductos["UnitsInStock"]);
                p.UnitsOnOrder = Convert.ToInt32(drProductos["UnitsOnOrder"]);
                p.ReorderLevel = Convert.ToInt32(drProductos["ReorderLevel"]);
                p.Discontinued = Convert.ToBoolean(drProductos["Discontinued"]);
                Prod.Add(p);
            }
            drProductos.Close();
            cmd.Dispose();
            conn.Close();
            return Prod;
        }
    }
}
```

Con esto concluiríamos la codificación del servicio.

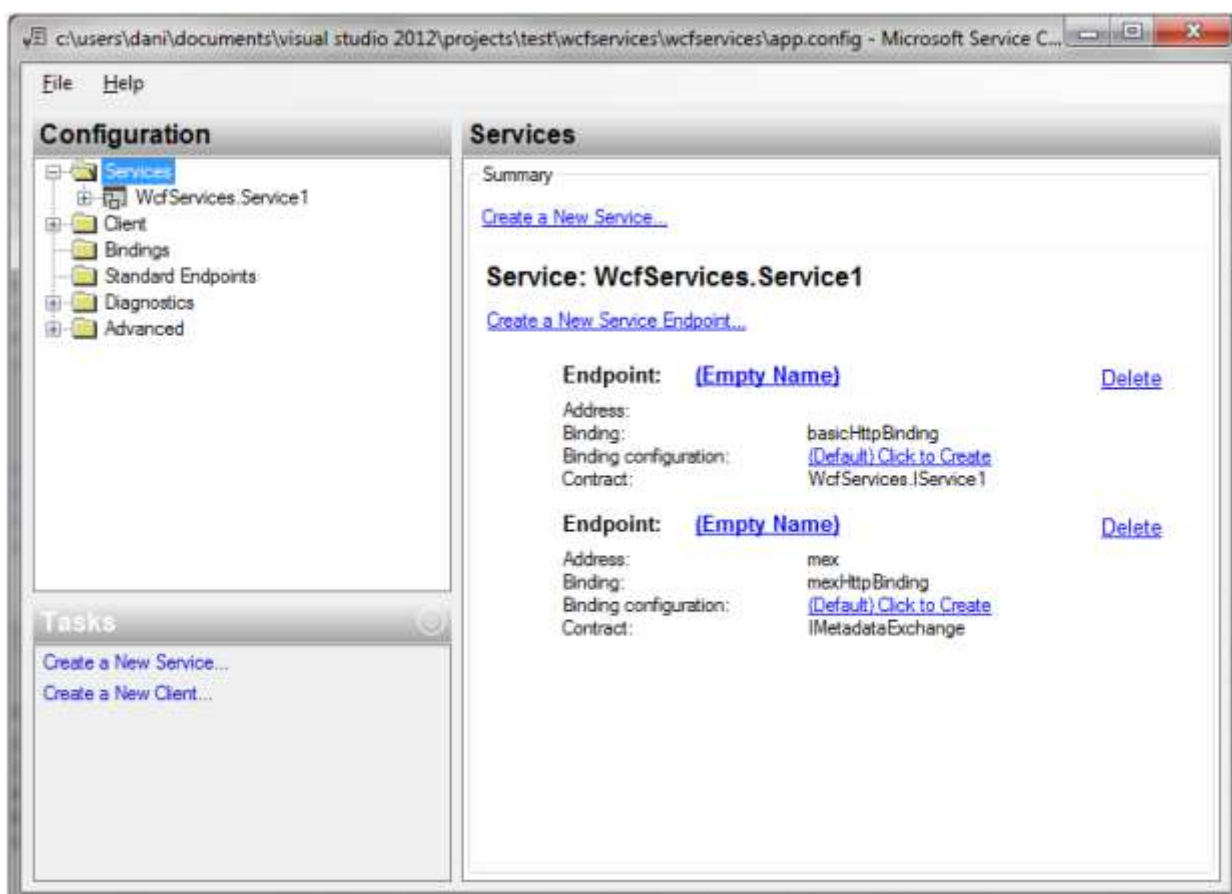
Sólo nos quedan dos pasos: configurar los *endpoints* y realizar una prueba para comprobar su funcionamiento.

Configuración si hacemos una Biblioteca de Servicio WCF

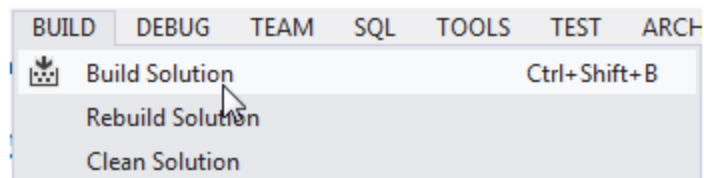
Para acceder a la configuración del servicio, podemos editar directamente el fichero de configuración *App.config* / *web.config* o bien podemos dejar que el asistente nos facilite esta tarea. Para acceder a él, haremos click derecho sobre el *App.config* y seleccionaremos la opción *Edit WCF Configuration*.



Esto abrirá un cuadro de diálogo que nos mostrará la configuración de nuestro servicio, mostrando dos *endpoints*. Sin embargo, la configuración que se nos muestra es la configuración inicial generada al crear el proyecto, por lo que no nos será de mucha utilidad.

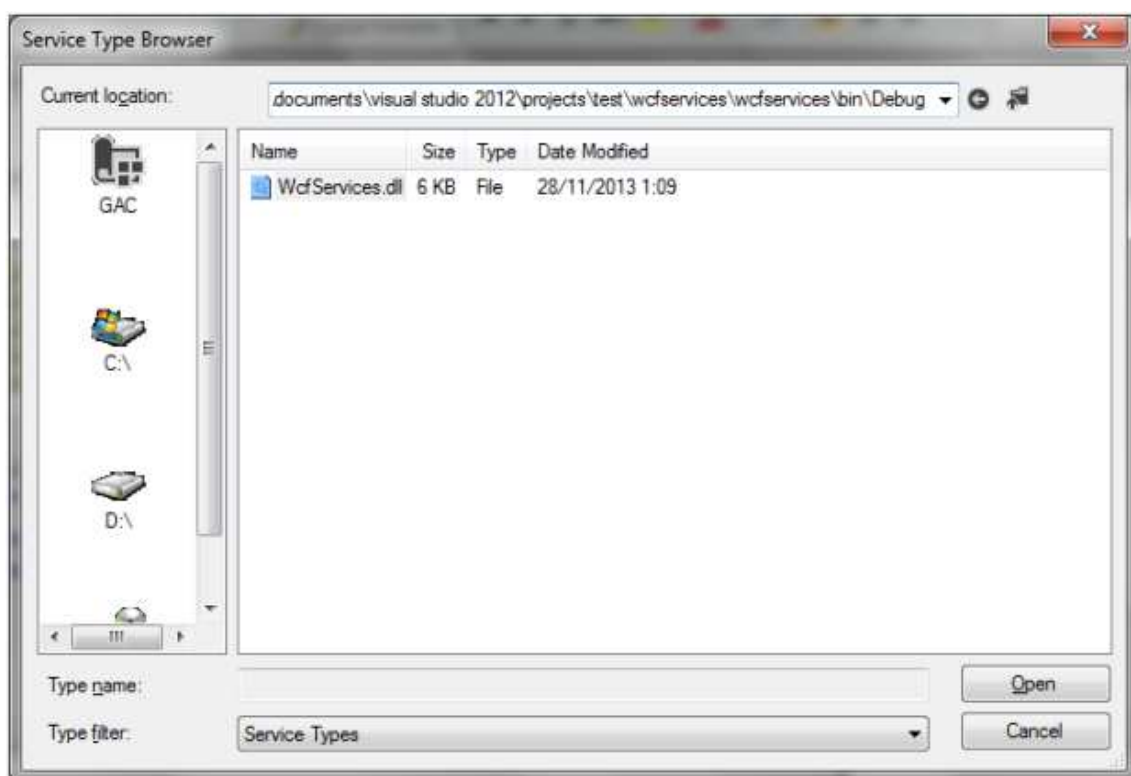


Para solucionarlo, comenzaremos por compilar el servicio web de modo que se genere el fichero *dll* objetivo.

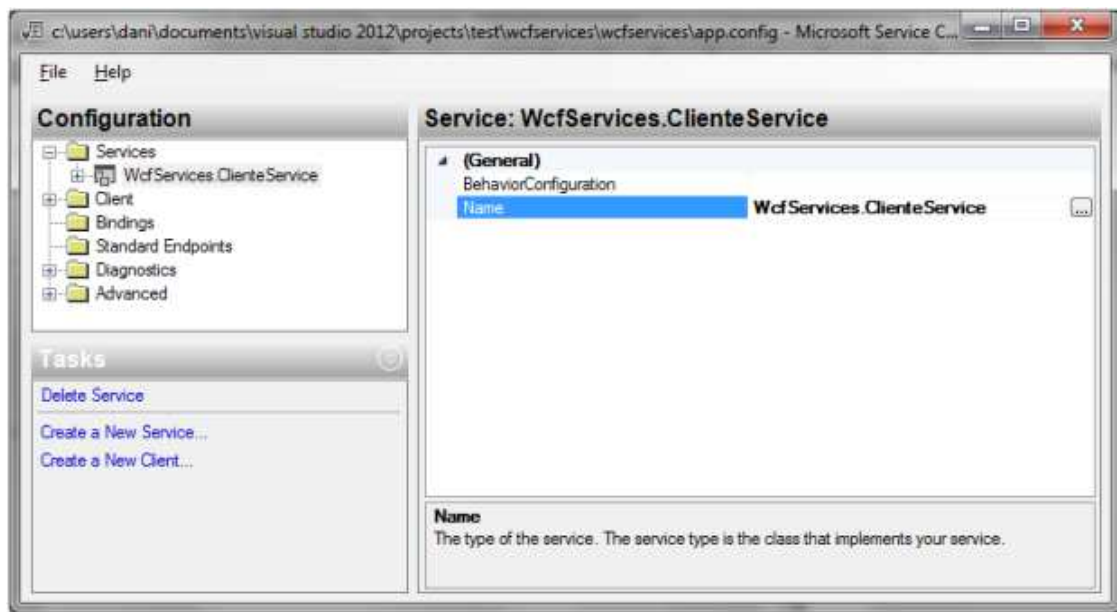


Hecho esto, seleccionaremos el servicio preconfigurado y pulsaremos el botón [...] que se muestra a la derecha del nombre del servicio cuando éste es seleccionado.

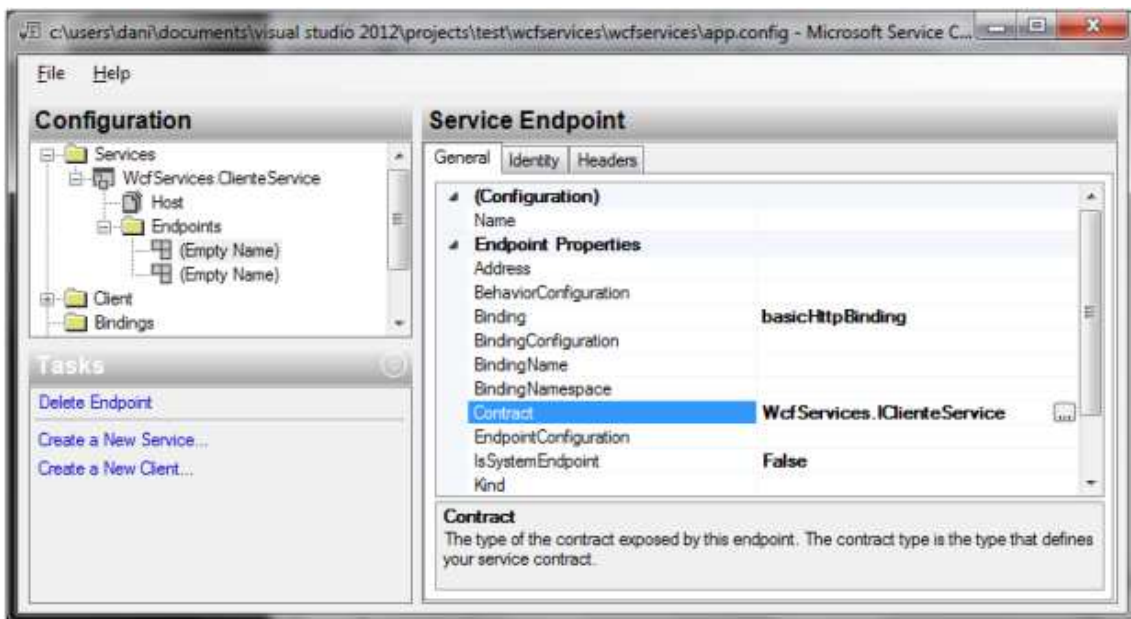
A continuación buscaremos la dll que acabamos de generar. La encontraremos en el directorio */bin/Debug* de nuestro proyecto.



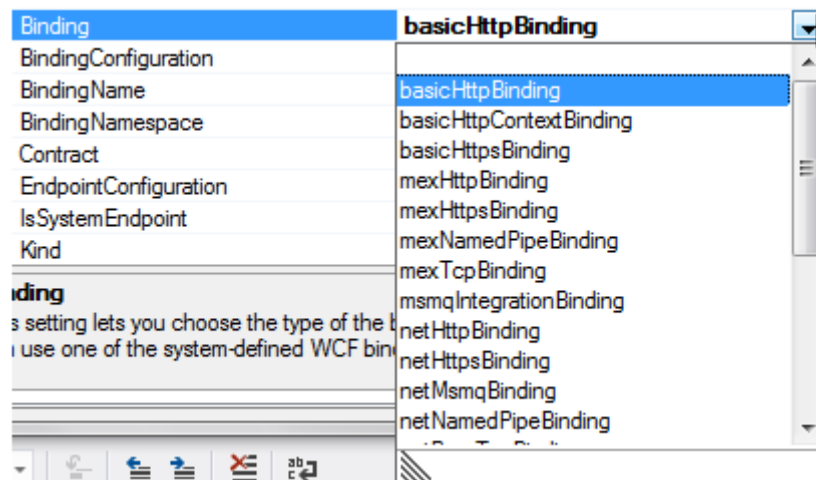
Esto hará que el nombre del servicio cambie, mostrando la configuración correspondiente al servicio que hemos codificado.



Desplegaremos la carpeta *Endpoints* y seleccionaremos el primero de ellos, realizando el mismo proceso de seleccionar el fichero *dll* para localizar el contrato del servicio.

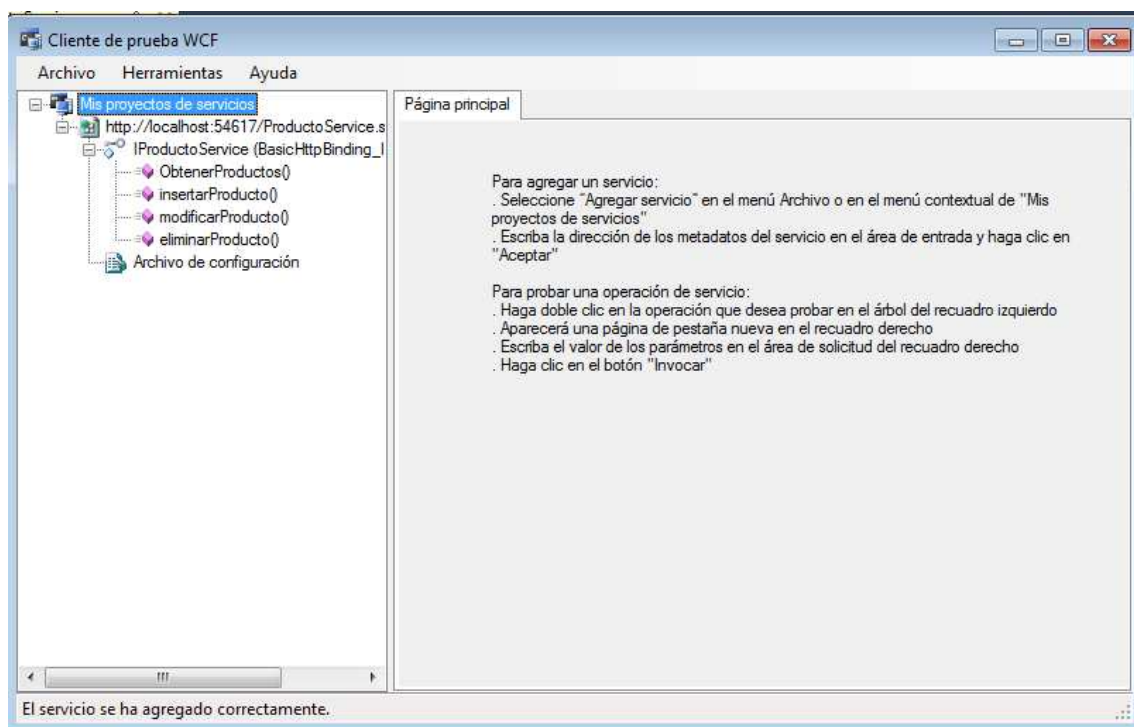


Si desplegamos el combo *Binding* podremos observar todas las posibilidades que WCF nos oferta como métodos de enlace. Seleccionaremos de momento el valor por defecto, *basicHttpBinding*.

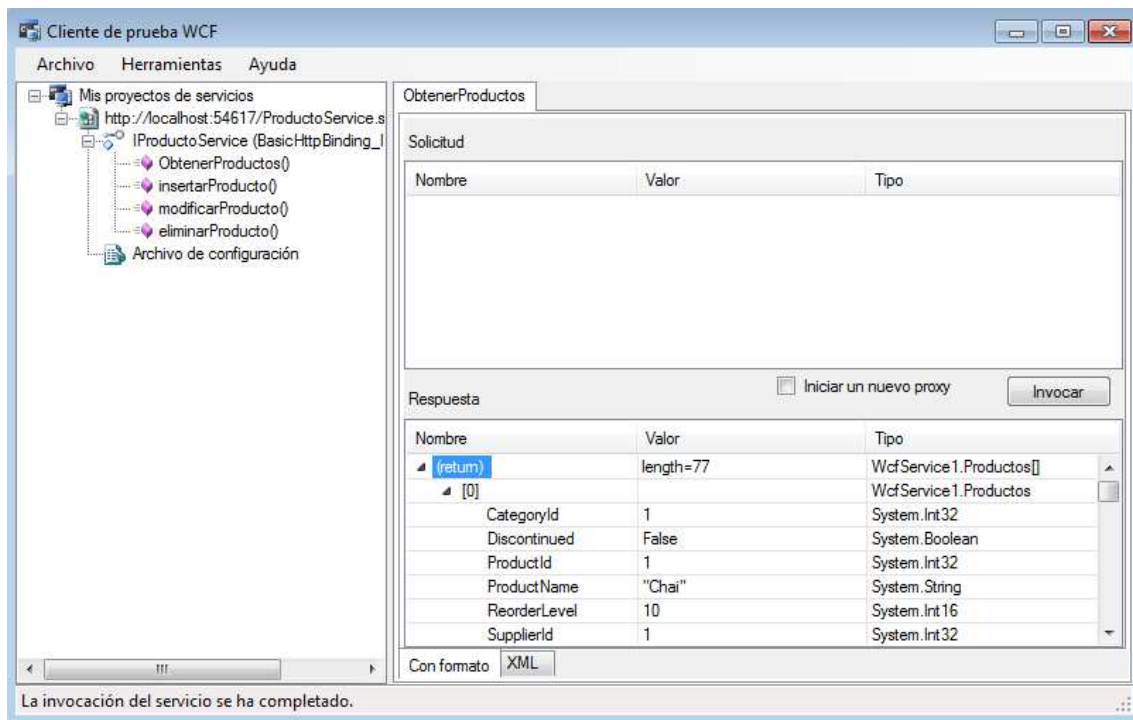


Probando el Servicio

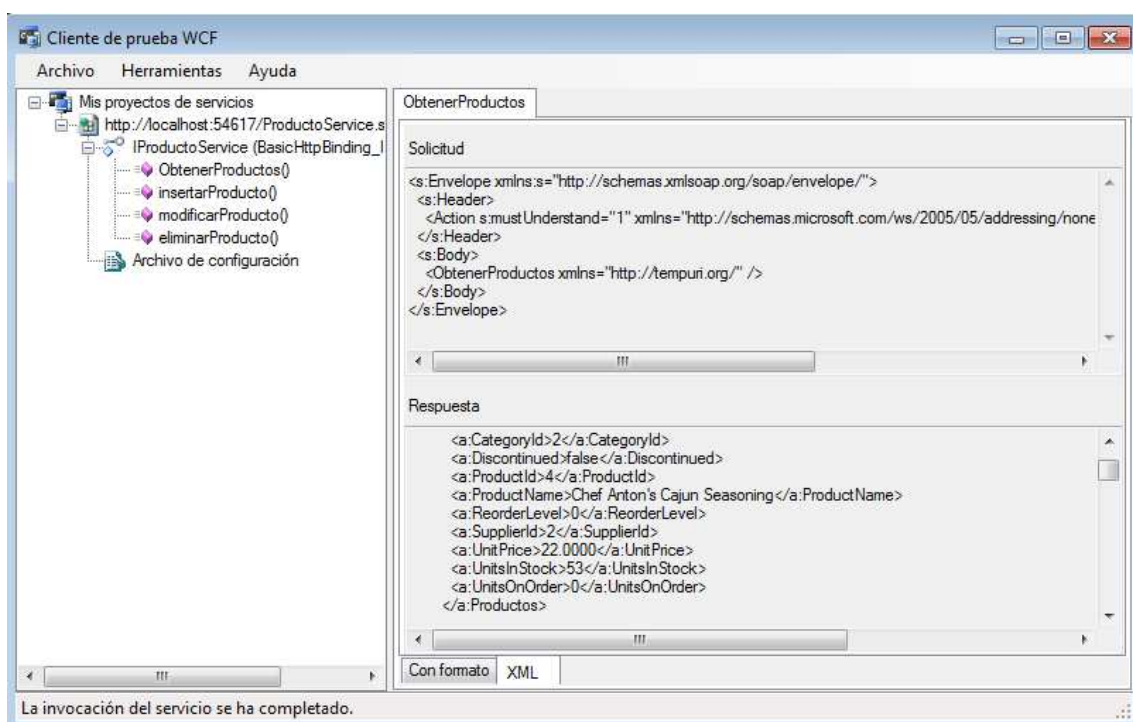
Si pulsamos F5 o pulsamos Play, se lanzará el cliente de prueba del servicio. Este cliente es generado automáticamente por Visual Studio, y nos permite invocar a los métodos expuestos por el servicio a través de una interfaz de usuario.



Haremos doble clic sobre el método *MostrarProductos* y pulsaremos el botón *Invoke*, que hará que se ejecute el método del servicio y se obtenga la respuesta correspondiente.

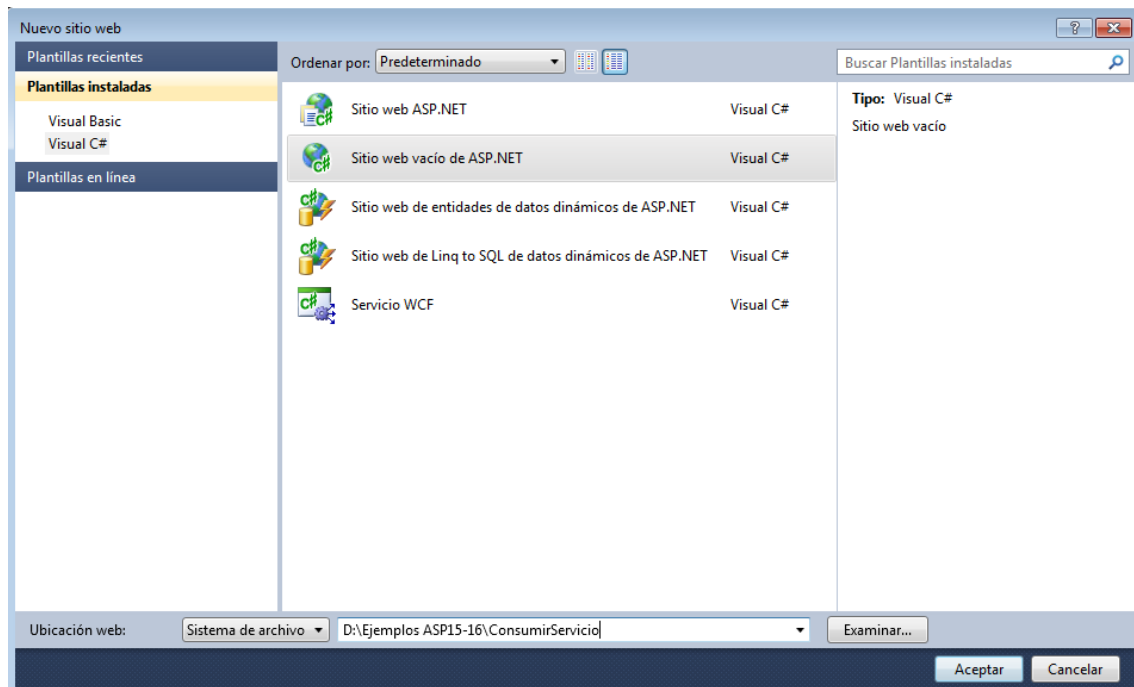


El cliente de prueba de WCF nos permite comprobar también el contenido de los mensajes. En este caso, se trata de mensajes SOAP. El mensaje SOAP se compone de un envoltorio (envelope) que contiene una cabecera (header) y un cuerpo (body). El resultado de la petición y de la respuesta será el siguiente:

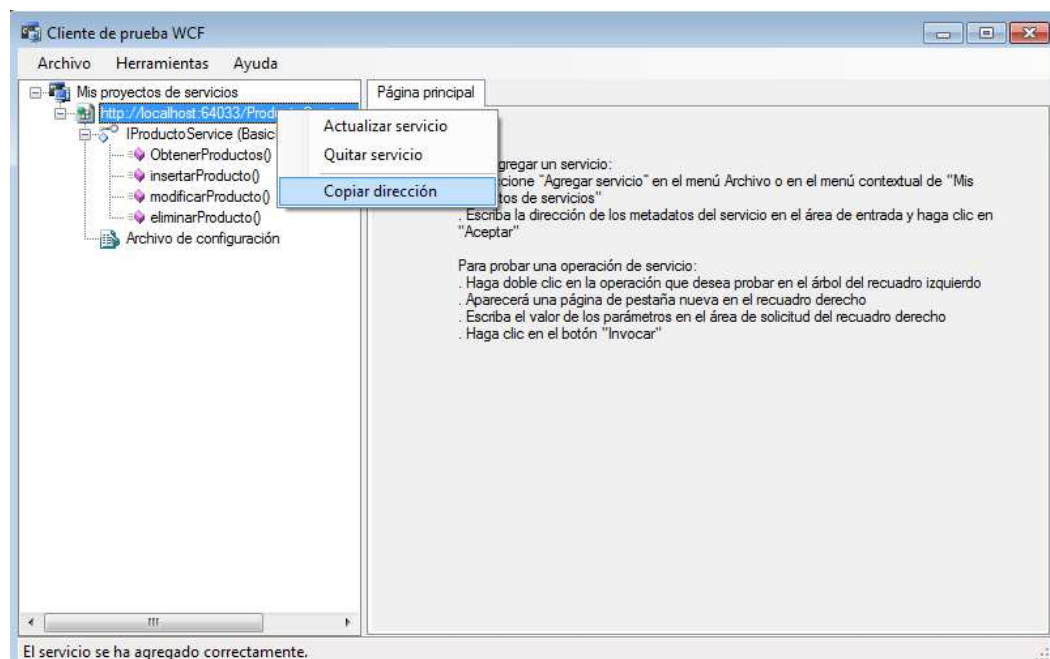


CONSUMIR UN WEB SERVICE

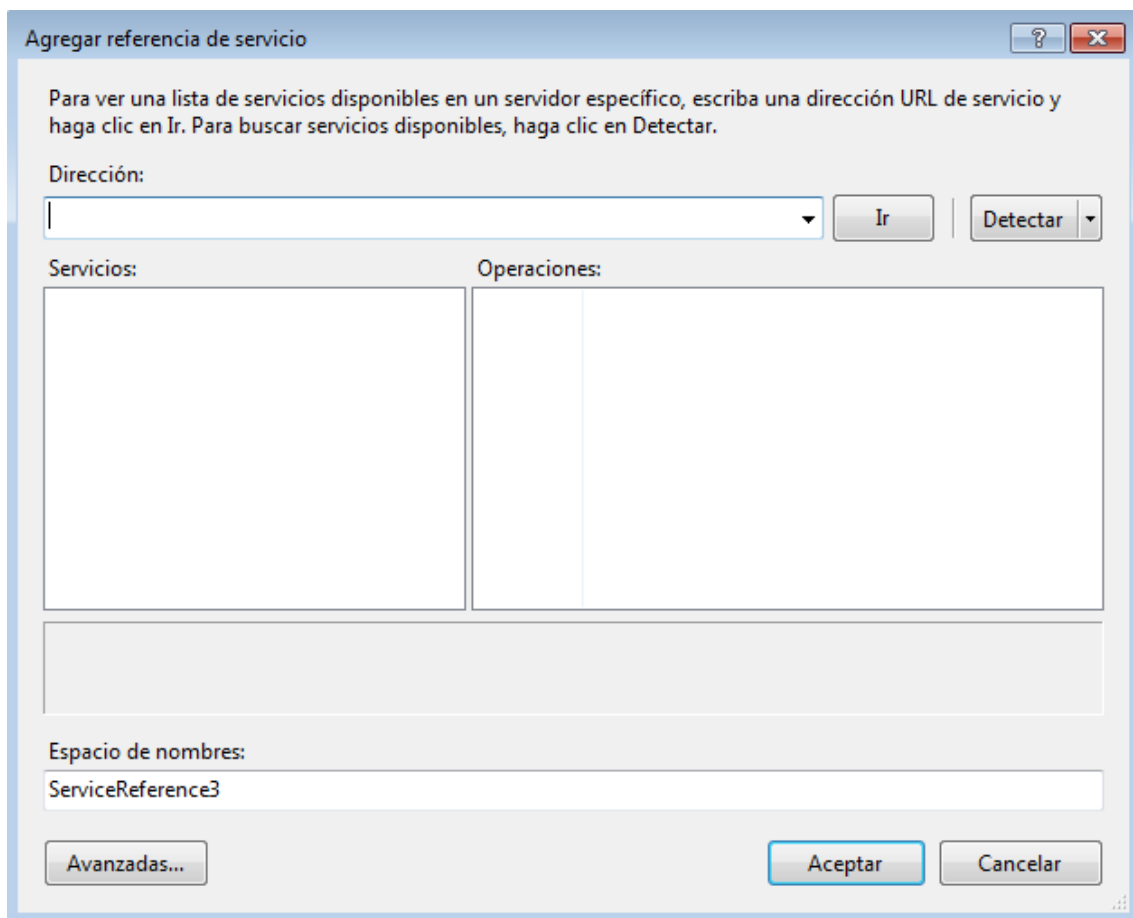
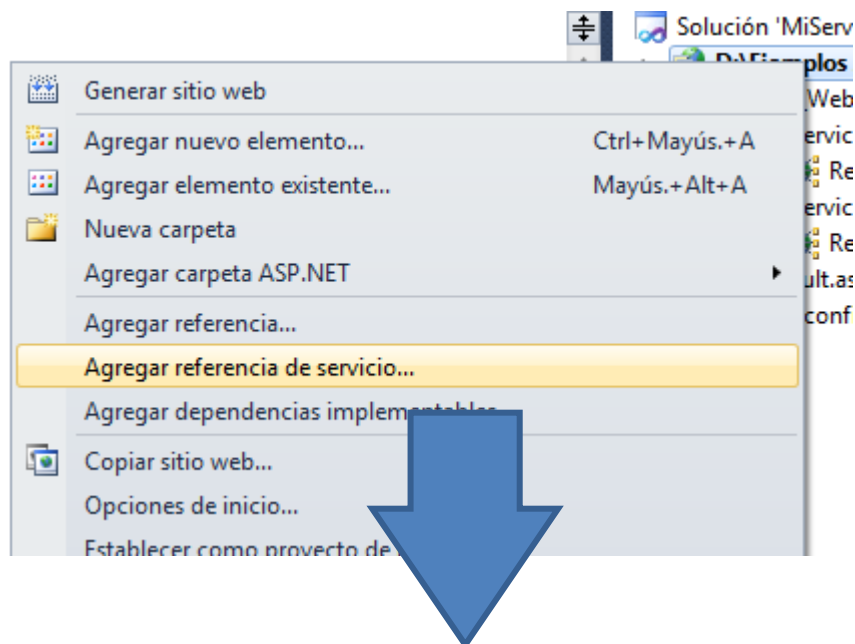
Crearemos un proyecto en donde queramos utilizar (consumir) nuestro Web Service. En nuestro caso un Sitio Web.



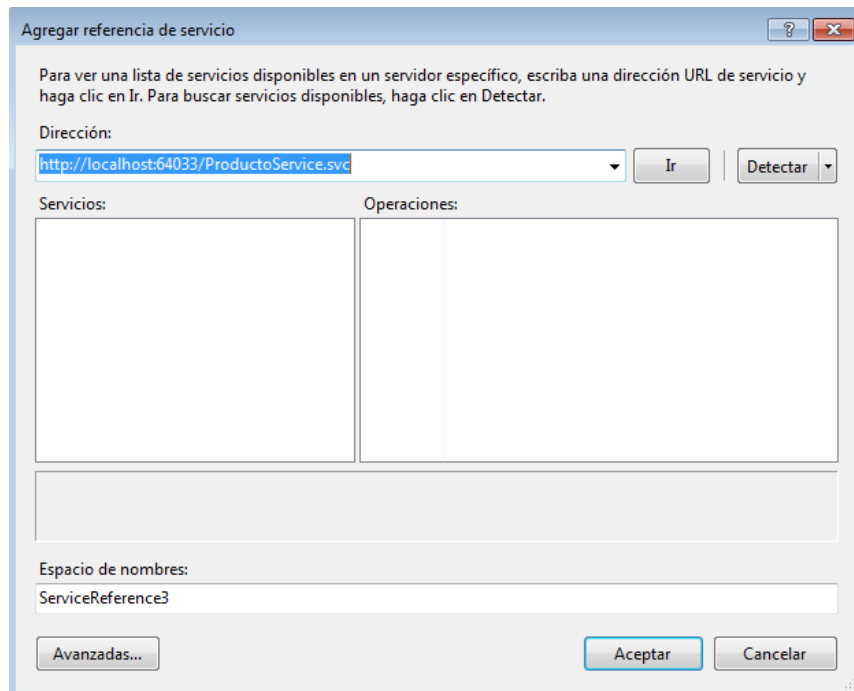
Es importante tener a mano la dirección del servicio, ya que nos hará falta para referenciarlo. Si estamos desarrollando el cliente en el mismo equipo en el que desarrollamos el web service, bastará con copiar la dirección del *WCF Test Client* que Visual Studio creará por nosotros y que hemos visto anteriormente al ejecutar el servicio.



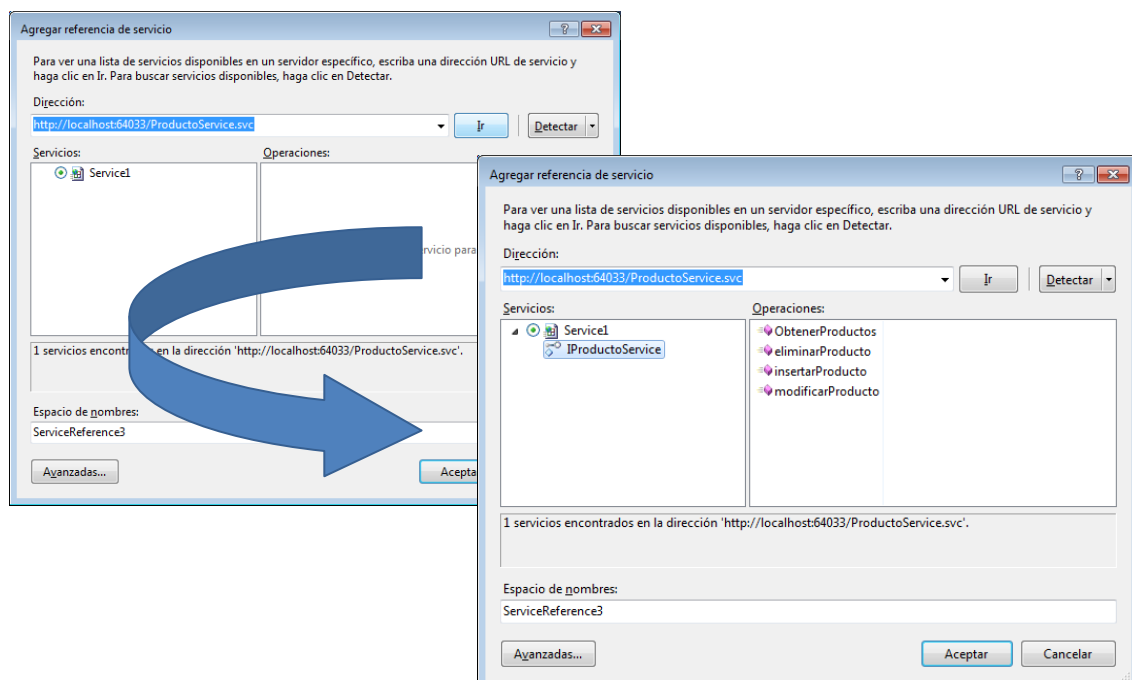
Seleccionaremos la sección **References** del proyecto y haremos clic derecho sobre ella, seleccionando la opción **Add Service Reference...** para añadir una referencia de servicio.



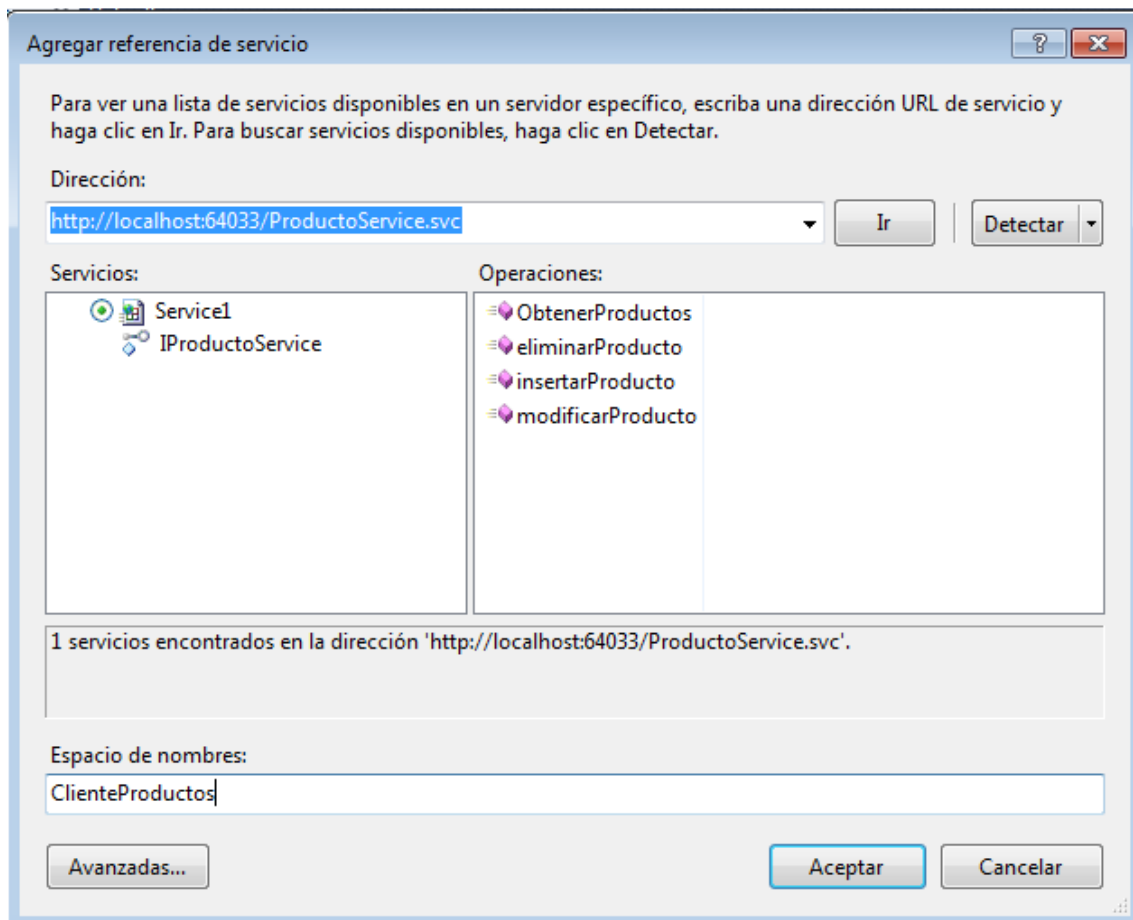
En el campo *Address* (dirección), insertaremos el valor que copiamos previamente.



Al pulsar el botón *Ir*, se mostrarán los contratos de servicio disponibles junto a las operaciones que éstos pueden ejecutar.



Asignaremos un nombre al espacio de nombres del cliente y pulsaremos el botón *Ok*.



Ahora procederemos a realizar las operaciones que realizamos de forma manual con el

