Índice

Sesión 3.1 - Arquitecturas orientadas a servicios

Aplicaciones Telemáticas

1. Introducción

En esta práctica se desarrollará una aplicación distribuida donde el servidor ofrezca servicios y el cliente consuma esos servicios. Este tipo de arquitectura orientada a servicios SOA es una generalización del modelo cliente/servidor. En este tipo de arquitectura los componentes del sistema son servicios con bajo acoplamiento. La forma más habitual de implementar SOA es mediante servicios web. Típicamente, en este tipo de arquitecturas se utilizan tecnologías que solucionen el problema del transporte y la codificación de la información. En esta práctica se utilizará Windows Communication Foundation (WCF), una tecnología desarrollada por Microsoft para el desarrollo de aplicaciones orientadas a servicios que se integra con .NET Framework.

2. Construcción de un servicio con WCF

A través del Visual Studio se facilita enormemente la creación de servicios WCF. Vamos a crear un servicio mínimo para ver el proceso y analizar su estructura.

- Crea un nuevo proyecto con Visual Studio. Debes elegir, dentro de la categoría Visual C#, el grupo WCF, y a continuación WCF Service Library. El nombre del proyecto será **MathService**.

  Se genera una estructura para un servicio mínimo con dos ficheros: **IService1.cs** y **Service1.cs**.
- Borra los ficheros **IService1.cs** y **Service1.cs** del proyecto a través del explorador de soluciones.

2.1. Definición de los contratos

Antes de implementar el servicio es necesario definir un contrato, es decir, un acuerdo en el que tanto el proveedor del servicio como el consumidor estén de acuerdo. En WCF existen varios tipos de contrato, algunos de los más comunes son los siguientes:

- **ServiceContract**: atributo que indica que un tipo expone operaciones.
- **OperationContract**: atributo que se usar para indicar una operación expuesta a los clientes.
- **DataContract**: atributo que define tipos se utilizarán durante la invocación a los servicios.

Básicamente, un **ServiceContract** será una clase que proporciona servicios, **OperationContract** son los servicios dentro de esta clase implementados como métodos, y **DataContract** son los tipos de datos que se pasarán como parámetro a los servicios. Para los tipos de datos primitivos, por ejemplo enteros o reales, no es necesario especificar un contrato de datos.

- Añade una nueva interfaz al proyecto en un fichero denominado **IMath.cs**. Esta interfaz se utilizará para definir el contrato.

  El contenido de este fichero será el siguiente:

```
using System.ServiceModel;

namespace MathService
{
    [ServiceContract]
    public interface IMath
    {
        [OperationContract]
        bool Prime(int value);
    }
}
```

En el fichero se define una interfaz que proporciona servicios. La interfaz define el contrato que el proveedor debe implementar y que el cliente consumirá. Dentro de esta interfaz se declara una operación **Prime**, que determina si un número es primo.

2.2. Implementación del servicio

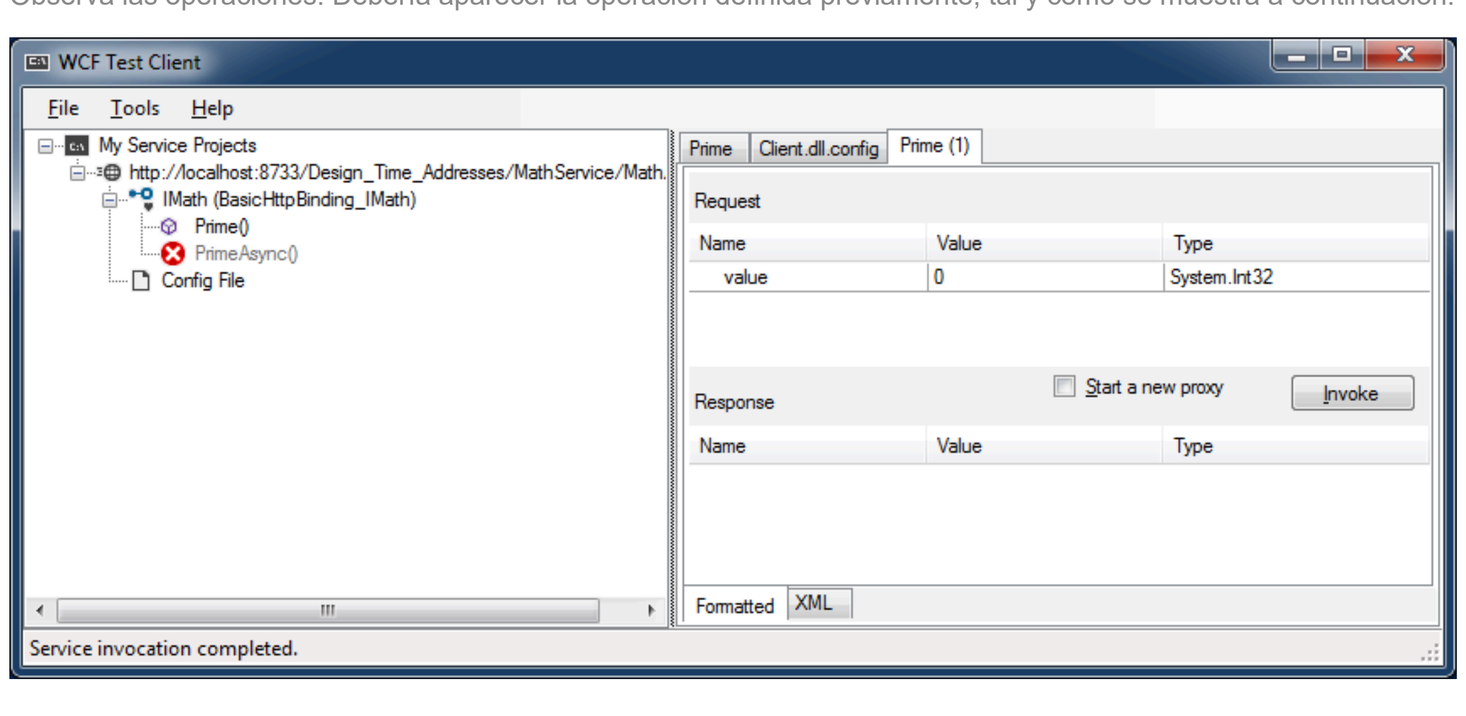
- Para implementar el servicio crea una nueva clase **Math** que herede de **IMath** e implemente la operación que se definió en el contrato.

  Cuando compiles el proyecto te aparecerán varios avisos que indican errores en la configuración del servicio WCF. Esta configuración se realiza a través del fichero **App.config**. Este fichero incluye información importante para poder exponer el servicio a los clientes, como por ejemplo la dirección y el protocolo de transporte.
- Edita el fichero **App.config**. La parte que se debe modificar está dentro de la sección **services**.
- Modifica el nombre del servicio (**service name**) para que se llame **MathService.Math**.
- Modifica el nombre del contrato (**endpoint contract**) para que haga referencia al contrato creado: **MathService.IMath**.
- Modifica la última parte de la dirección base (**baseAddress**) para en lugar de poner **Service1** ponga **Math**. No modifiques el resto de la URL, su valor permite hacer pruebas sin permisos de administrador.
- Recompila el proyecto. Los avisos deberían desaparecer.

2.3. Probando el servicio

El resultado de la compilación del proyecto es una biblioteca DLL. Existen muchas opciones para alojar el servicio. Las más comunes consisten en alojar el servicio en un servidor Web, especialmente si el servicio es un servicio Web, o alojar el servicio en un servidor específico. Para el desarrollo y las pruebas existe la opción de alojar el servicio dentro de un programa denominado WCF Service Host. Además, otro programa denominado WCF Test Client permite observar las operaciones expuestas por el servicio, e invocar las operaciones y obtener el resultado.

- Ejecuta el proyecto con la combinación de teclas habitual. Por defecto se lanzará el servicio alojado en WCF Service Host y se lanzará el cliente de pruebas.
- Observa las operaciones. Debería aparecer la operación definida previamente, tal y como se muestra a continuación:



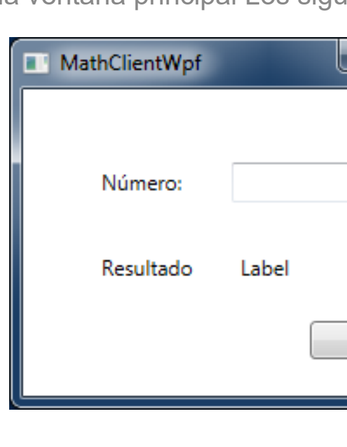
- Invoca al servicio con el cliente de prueba y verifica que el resultado es correcto.
- Fíjate en la pestaña XML. Se está utilizando SOAP para intercambiar la información con el servicio.

2.4. Consumo del servicio

El objetivo de un servicio es exponer funcionalidad que los clientes puedan consumir y utilizar. Por tanto, una vez que el servicio está creado vamos a crear un cliente que lo consuma.

- Abre otra ventana de Visual Studio y crea un nuevo proyecto. Vamos a consumir el proyecto mediante un programa con interfaz de usuario. Debes elegir, dentro de la categoría Visual C#, el grupo Windows Desktop, y a continuación WPF Application. El nombre del proyecto será **MathClientWpf**.

  Windows Presentation Foundation (WPF) es una tecnología de Microsoft para el desarrollo de aplicaciones con interfaz de usuario en Windows. Al igual que otras tecnologías modernas incluye un lenguaje de descripción de interfaces, en este caso denominado XAML, que está basado en XML. Este fichero se puede editar como texto o también a través de un editor gráfico.
- Utiliza el editor gráfico para añadir a la ventana principal Los siguientes controles:



  Para cambiar las propiedades de los controles debes interactuar con la ventana de propiedades. Dentro de esta ventana aparecen dos opciones en la esquina superior derecha: las propiedades (con el símbolo de la herramienta) y los eventos (con el símbolo del rayo).
- Modifica las propiedades de los controles de forma similar a la figura anterior. Debes dar un nombre a cada control para luego poder acceder a él desde el código.
- Añade un manejador para el evento **Click** para el botón. Se autogenera un esqueleto al hacer doble click sobre la casilla correspondiente.
- Añade el código para convertir el contenido de campo de edición a número entero: **Int32.TryParse(Valor.Text, out iValue)**.
- Añade un resultado de prueba: **Resultado.Content = "quien sabe";**
- Verifica su funcionamiento.

Ahora vamos a añadir la funcionalidad al programa para mostrar el resultado tras invocar al servicio implementado anteriormente.

- Añade una referencia de servicio al proyecto. Esto se hace pulsando con el botón derecho sobre el proyecto, luego *Add*, y por último *Service Reference*….
- Aparece una pantalla donde indicar una dirección para descubrir el servicio, es decir, su contrato. Debes indicar la URL de dirección configurada previamente. Debería aparecer el servicio y las operaciones.
- Selecciona un espacio de nombres en la parte inferior. Por ejemplo **MathService**.

  Tras agregar la referencia de servicio se crea un proxy que se podrá utilizar para invocar al servicio como si fuera local. Por defecto al proxy se le añade **Client** al final por tanto para usarlo se realizaría la siguiente operación:

```
// Se instancia el proxy
MathService.MathClient client = new MathService.MathClient();

// Se invoca el servicio
bool result = client.Prime(Valor);
```

- Añade la funcionalidad del servicio a la aplicación y verifica su funcionamiento.

2.5. Añadiendo funcionalidad

Los servicios que utilizan tipos de datos compuestos por múltiples datos de tipo primitivo necesitan contrato de datos. Este contrato sirve para especificar el parámetro o el retorno de un servicio.

Vamos a extender el servicio matemático para poder operar con tuplas. Se va a definir una tupla como un vector de números y un nombre. Una operación aplicada sobre una tupla retornará otra, con un nombre que indique la operación realizada sobre la tupla original.

- Añade al fichero **IMath.cs** el siguiente contrato de datos para la tupla.

```
[DataContract]
public class Tuple
{
   double[] _data;
   string _name;

   [DataMember]
   public double[] Data
   {
      get { return _data; }
      set { _data = value; }
   }

   [DataMember]
   public string Name
   {
      get { return _name; }
      set { _name = value; }
   }
}
```

- Implementa un nuevo servicio que reciba una tupla y retorne otra con la suma de sus elementos y el nombre adecuado. En este caso la tupla que se retorna tendrá solo un elemento.
- Invoca al servicio desde el cliente. Previamente deberás actualizar la información del servicio para poder utilizar la nueva operación y el nuevo tipo de datos. La actualización se puede realizar pulsando con el botón derecho del ratón sobre la referencia del servicio dentro del explorador de soluciones, y luego pulsado *Update Service Reference*. El servicio debe estar en ejecución para poder realizar la actualización.
- Verifica que el resultado es correcto.

2.6. Hosting del servicio

Hasta el momento el servicio se ha alojado en el WCF Service Host. A continuación vamos a alojar el servicio en un programa ejecutable.

- Crea un nuevo proyecto de C# de consola denominado **MathServiceHost**.
- Añade al proyecto una referencia al *Assembly* **System.ServiceModel**.
- Añade el proyecto **MathService** a la solución.
- Añade al proyecto **MathServiceHost** una referencia al proyecto **MathService** que has agregado a la solución.
- En la configuración del proyecto **MathService** dentro de la solución, cambia las propiedades de WCF para que no inicie de forma automática.
- Modifica el fichero **App.Config** de **MathServiceHost** copiando toda la sección **system.serviceModel** del proyecto **MathService**.
- Añade el siguiente código al programa para alojar el servicio:

```
 static void Main(string[] args)
{
   using (ServiceHost host = new ServiceHost(typeof(MathService.Math)))
   {
      host.Open();

      Console.WriteLine("Servicio ejecutándose. Pulsa <enter- para terminar");
      Console.ReadLine();

      host.Close();
   }
}
```

- Ejecuta el programa **MathServiceHost**.
- Ejecuta el cliente en WPF y verifica el funcionamiento.

En este ejemplo el programa que aloja el servicio espera por la pulsación de una tecla para terminar, pero podría realizar cualquier otra tarea mientras se sirve el servicio.

2.7. *Endpoints*

WCF proporciona un API unificado para el desarrollo de servicios, pero es muy flexible a la hora de establecer la forma de acceder al servicio y el protocolo que se usa.

Además de las operaciones y del tipo de alojamiento del servicio, en WCF un servicio debe exponer un *endpoint*. Cada *endpoint* especifica tres cuestiones:

- *address*: es la dirección URL en la que se encuentra escuchando el servicio.
- *binding*: establece la forma de acceder al *endpoint*, es decir, la combinación de protocolos que se utilizan.
- *contract*: indica el contrato del servicio que se expone a través del *endpoint*.

A estos tres elementos se les denomina el ABC del *endpoint* (*Address*, *Binding* y *Contract)*.

Las direcciones son URLs de acceso y los contratos se han visto anteriormente. Un *binding* es un tema importante ya que define los protocolos que se usarán para acceder al servicio. WCF disponible de muchos tipos predefinidos y que se pueden utilizar para exponer el servicio de distinta manera. Dos de estos *bindings* son **BasicHttpBinding** y **NetTcpBinding**.

BasicHttpBinding se utiliza para exponer al servicio con WS-I también conocidos como servicios Web pesados. Este tipo de servicios utilizan SOAP sobre HTTP o HTPPS. SOAP es un protocolo que indica como intercambiar información con un servicio web. Este tipo de *binding* permite interoperabilidad con cualquier otro cliente que permita invocar a servicios web SOAP, y es el recomendado por WCF para servicios que se proporcionan a través de internet.

NetTcpBinding permite exponer al servicio mediante SOAP sobre TCP. Este tipo de *binding* no permite interoperabilidad con otros clientes que no sean específicos de WCF. Se recomienda su uso para intranets con sistemas Windows ya que es uno de los más eficientes.

Existen otros muchos tipos de *binding* en función de las necesidades concretas de la arquitectura de servicios que se desea desarrollar.

La definición de los *endpoint* de un servicio se realiza generalmente en el fichero **App.config**.

- Edita el fichero **App.config** de **MathService**.
- Identifica el ABC del servicio.

  Además del *binding* ya descrito, verás otro denominado **mexHttpBinding** que sirve para generar metadatos.

  Vamos a añadir un nuevo *endpoint* a **MathServiceHost**.
- Abre en el Visual Studio el proyecto **MathServiceHost**.
- Pulsa con el botón derecho en el fichero **App.Config** dentro del explorador de soluciones, y elige la opción *Edit WCF Configuration*.

  Este editor permite modificar la configuración del servicio.
- Abre la carpeta **Services**, luego **MathService.Math**, y **EndPoints**.
- Añade un nuevo *endpoint*. Como tipo elige **netTcpBinding** y como contrato **MathService.IMath**. El resto lo puedes dejar por defecto.
- Abre la carpeta **Services**, luego **MathService.Math**, y **Host**.
- Añade una nueva dirección base denominada **net.tcp://localhost:8082/Math**.
- Guarda la configuración y sal del editor. El fichero **App.config** debería ser similar al siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup-
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
  <system.serviceModel>
    <services>
      <service name="MathService.Math">
        <endpoint address="" binding="basicHttpBinding" contract="MathService.IMath">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
        <endpoint binding="netTcpBinding" bindingConfiguration=""
contract="MathService.IMath" />
        <host>
          <baseAddresses>
            <add
baseAddress="http://localhost:8733/Design_Time_Addresses/MathService/Math/" />
            <add baseAddress="net.tcp://localhost:8082/Math" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- To avoid disclosing metadata information,
          set the values below to false before deployment -->
          <serviceMetadata httpGetEnabled="True" httpsGetEnabled="True"/>
          <!-- To receive exception details in faults for debugging purposes,
          set the value below to true.  Set to false before deployment
          to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel-
</configuration>
```

- Ejecuta el programa. El servicio quedará en ejecución listo para que los clientes lo invoquen.

Ahora vamos a crear otro cliente para invocar al servicio. En este caso vamos a usar una forma alternativa de crear el proxy.

- Crea un nuevo proyecto de C# de consola denominado **MathClientConsole**.
- Añade al proyecto una referencia al *Assembly* **System.ServiceModel**.
- Añade al proyecto una referencia al *Assembly* **System.Runtime.Serialization**.

  Para agregar el proxy al proyecto vamos a utilizar una vía alternativa.
- Ejecuta el navegador de internet y abre la URL http://localhost:8733/Design_Time_Addresses/MathService/Math/.
- Aparece una pantalla de descripción del servicio que indica cómo generar el proxy y como invocar el servicio.
- Pulsa en el enlace y guárdalo en un fichero denominado **MathService.wsdl** dentro de la carpeta del proyecto **MathClientConsole**.

El fichero generado **MathService.wsdl** es un fichero que contiene la descripción del servicio en formato WSDL (Web Services Description Language). Este fichero se utilizará para generar el proxy. No es específico de Microsoft, por tanto se puede utilizar desde otros entornos, como java o python, para generar proxys de invocación específicos.

Ahora vamos a generar el proxy para C#

- Abre el intérprete de comandos de Visual Studio en el enlace que la instalación del Visual Studio deja preparado dentro del menú de Inicio. Este enlace se encuentra dentro del grupo del Visual Studio, en la carpeta *Tools*.
- Cambia al directorio donde guardaste el fichero **MathService.wsdl**.
- Ejecuta desde el intérprete el siguiente comando:

```
svcutil.exe MathService.wsdl
```

  Este proceso creará dos ficheros, uno de configuración y otro de código: **output.config** y **Math.cs**. El fichero de configuración contiene información sobre *endpoints* del servicio. El fichero de código contiene el proxy.
- Copia el contenido de **output.config** en el fichero **App.config** del proyecto **MathClientConsole**.
- Agrega el fichero **Math.cs** al proyecto **MathClientConsole**.
- Añade el siguiente código fuente al proyecto:

```
static void Main(string[] args)
{
   MathClient client = new MathClient("NetTcpBinding_IMath");

   int x = 23;

   if (client.Prime(x))
      Console.WriteLine("{0} es primo", x);
   else
      Console.WriteLine("{0} no es primo", x);
}
```

- Verifica que el funcionamiento es el esperado.
- Cambia el tipo de *endpoint* para que el cliente invoque al servicio de la otra forma posible. Te debes fijar en el nombre de *binding* que existe en el fichero **App.config**.
- Ejecuta de nuevo el programa y verifica el resultado.

3. Ampliaciones

Los ejercicios propuestos en esta práctica se valorarán con un máximo de 6 puntos. Las siguientes ampliaciones sirven para mejorar la nota.

3.1. Funcionalidad adicional

Esta ampliación consiste en añadir funcionalidad adicional al servicio matemático. En concreto se deberá añadir una operación para resolver sistemas de ecuaciones lineales. Se recomienda utilizar algún tipo de biblioteca externa. En esta ampliación se deberá implementar también un cliente que invoque el servicio y verifique el funcionamiento.

Esta ampliación se valorará con un máximo de 2 puntos.

3.2. Invocación desde otro lenguaje

Esta ampliación consiste en invocar alguno de los servicios desarrollados desde otro lenguaje de programación. Se puede usar el lenguaje que se desee.

Esta ampliación se valorará con un máximo de 2 puntos.