Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

21

Gráficos

Conceptos de animaciones / Dos y tres dimensiones

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Windows Communication Foundation

Reseña histórica de las comunicaciones / Comunicación entre aplicaciones / Conceptos de servidor, cliente y mensaje

ISBN 978-987-1347-43-8
00021
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Gráficos en 3D

En las siguientes páginas veremos cómo generar gráficos 3D en base al sistema de gráficos de WPF.

El sistema de gráficos de tercera dimensión de Windows Presentation Foundation se encuentra completamente integrado al sistema de gráficos 2D, y todos los elementos y primitivas visuales que son válidos en el entorno 2D también lo son en el trabajo en tercera dimensión. Por ejemplo, sabemos que para generar un gradiente utilizamos el elemento **<LinearGradientBrush>**. De igual manera, podemos usar la misma técnica para generar una textura que tiene cualquier objeto creado en tercera dimensión, sin necesidad de aprender algo nuevo. Esto representa una gran ventaja para los desarrolladores, porque no es necesario aprender un lenguaje diferente u objetos distintos para manejo de gráficos de tercera dimensión, sino que es posible reutilizar todo el conocimiento adquirido con el dibujo 2D y, simplemente, aprender algunos conceptos nuevos, propios del modelado en 3D.

Generar un objeto 3D

1) El elemento **ViewPort3D** nos brinda la capacidad de dibujar una escena tridimensional en una superficie de dos dimensiones. Este elemento expone la colección **Children**, que es una colección de objetos **ModelVisual3D** dentro de los cuales implementaremos nuestras escenas tridimensionales. El código es:

```
<Viewport3D ClipToBounds="True" Width="150"
Height="150">
   <Viewport3D.Children>
      <ModelVisual3D>
      </ModelVisual3D>
   </Viewport3D.Children>
</Viewport3D>
```

2) Dentro del elemento **ModelVisual3D** podemos especificar cualquier dibujo que deseemos. Sin embargo, a diferencia de los gráficos en dos dimensiones –en los que existen primitivas predefinidas para dibujar objetos como elipses o rectángulos–, en el mundo de 3D sólo contamos con una clase denominada **MeshGeometry3D**, en la cual es posible especificar un conjunto de vértices de triángulos mediante la propiedad **Positions**.
Esto se debe a que cualquier dibujo tridimensional puede describirse mediante triángulos. Recordemos lo que sucede en los juegos de video: cuando hacemos un acercamiento fuerte en alguna escena tridimensional, podemos apreciar que las puntas y los contornos de los objetos se tornan triangulares; esto se debe a que muchos objetos de los videojuegos, al igual que en WPF, son conjuntos de triángulos, que a través de su

FIGURA 038 | Ejemplo del código expuesto hasta el momento.

composición, dan la sensación de ser un objeto tridimensional. Regresando a nuestro ejemplo, una vez que hemos definido la superficie de dibujo, procedemos a utilizar la propiedad ModelVisual3D. Content para, dentro de ella, especificar un MeshGeometry3D. En este caso, con las posiciones establecidas estamos generando un cuadrado:

```
<ModelVisual3D.Content>
  <GeometryModel3D>

    <GeometryModel3D.Geometry>
      <MeshGeometry3D Positions="-0.5,-0.5,
      0.5 0.5,-0.5,0.5 0.5,0.5,0.5 0.5,0.5,
      0.5 -0.5,0.5,0.5 -0.5,-0.5,0.5" />
    </GeometryModel3D.Geometry>

  </GeometryModel3D>
</ModelVisual3D.Content>
```

3) A diferencia de lo que sucede con los objetos 2D, en los que el concepto de luz no existe, en 3D es indispensable especificar una fuente de luz, debido a que si no está presente, el objeto no será visible, tal como sucede en la vida real. Esto significa que debemos especificar el lugar que estará irradiando la luz y el tipo de material que compone nuestro objeto; es decir, si éste es reflejante o no. WPF especifica tres tipos de materiales: SpecularMaterial refleja la luz de tal manera que el objeto parece brillar, EmissiveMaterial refleja la luz, y DiffuseMaterial no lo hace. En nuestro ejemplo, utilizaremos el material DiffuseMaterial. Para especificar el tipo de material del cual se compone nuestro Mesh, aplicamos el siguiente código dentro de GeometryModel3D:

FIGURA 039 | La luz es reflejada en un objeto 3D en WPF.

```
<GeometryModel3D.Material>
  <DiffuseMaterial Brush="Red" >
  </DiffuseMaterial>
</GeometryModel3D.Material>
```

4) En el mundo de la tercera dimensión, los objetos cambian de apariencia desde el punto de vista de quien los observa. En WPF especificamos este punto de vista haciendo uso de los objetos que heredan de **ProjectionCamera**. Controlamos nuestras cámaras como si en verdad estuvieran posicionadas en el espacio. La propiedad **Position** es del tipo **Point3D** y especifica las coordenadas en las cuales se encuentra la cámara.

```
<Viewport3D.Camera>
  <PerspectiveCamera Position="0,0,2"/>
</Viewport3D.Camera>
```

Aquí, antes de nada, podemos ver que se generó un cuadrado derivado de los vértices especificados en el elemento **MeshGeometry3D**. Como aún no hacemos ninguna transformación de nuestro objeto, no podemos ver la profundidad que tiene; la añadiremos más adelante. Otro aspecto que podemos resaltar es que, a pesar de que el material especificado con **DiffuseMaterial** fue de color Red, vemos que nuestro programa muestra el cuadrado en color negro. Esto se

debe a que todavía no hemos indicado la manera en la que llegará la luz.

5) Para agregar una fuente de luz a la escena, dentro de **ViewPort3D.Children** especificamos el código que se indica a continuación. En la Figura 39 vemos que, al ejecutarlo, el objeto refleja la luz y, por lo tanto, se muestra del color que se estableció previamente.

```
<ModelVisual3D>
  <ModelVisual3D.Content>
    <DirectionalLight Color="White"
    Direction="-0.5,-0.1,-0.5" />
  </ModelVisual3D.Content>
</ModelVisual3D>
```

6) Procedemos a generar una transformación dentro del elemento **GeometryModel3D**, que nos permitirá ver nuestro objeto en profundidad y, de esta manera, percibir el efecto de tercera dimensión. Para lograrlo, es necesario hacer uso de la clase **RotateTransform3D**, que funciona de una manera similar a su equivalente bidimensional, pero a diferencia de éste, empleamos tres dimensiones para rotar al elemento deseado. Veamos el uso de este objeto:

XAML:

```
<GeometryModel3D.Transform>
  <RotateTransform3D>
    <RotateTransform3D.Rotation>
      <AxisAngleRotation3D Axis="0,3,0"
      Angle="40" />
    </RotateTransform3D.Rotation>
  </RotateTransform3D>
</GeometryModel3D.Transform>
```

7) Para finalizar nuestro ejemplo, vamos a animar la propiedad **Angle** de la transformación. La sintaxis para hacer animación en elementos 3D es exactamente la misma que para los objetos bidimensionales. Con esta animación veremos que el objeto se mueve de una manera circular. El código requerido se presenta a continuación, y luego veremos el código completo:

```
<Viewport3D.Triggers>
  <EventTrigger RoutedEvent="Viewport3D.
  Loaded">
    <BeginStoryboard>
      <Storyboard>
        <DoubleAnimation From="1" To="360"
        Duration="0:0:03"  Acceleration
        Ratio="1" RepeatBehavior="Forever"
        Storyboard.TargetProperty="(Viewport3D.
        Children)[1].(ModelVisual3D. Content).
        (GeometryModel3D.Transform).(Rotate
        Transform3D.Rotation).(AxisAngle
        Rotation3D.Angle)"></DoubleAnimation>
      </Storyboard>
    </BeginStoryboard>
  </EventTrigger>
</Viewport3D.Triggers>
```

XAML: Ejemplo completo

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/
```

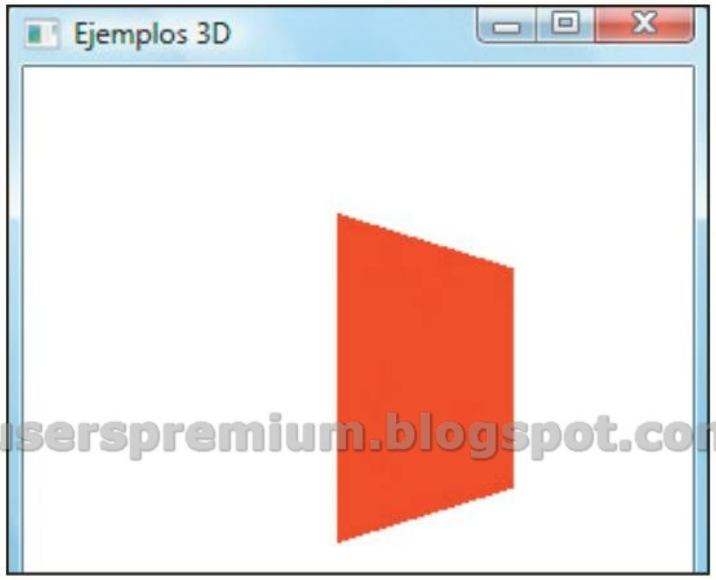

FIGURA 040 | Al aplicar una transformación 3D a un objeto, podemos hacerlo rotar y verlo en perspectiva.

```
    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
    winfx/2006/xaml"
    Title="Ejemplo 3D" Height="300" Width=
    "300">
  <Grid>
    <Viewport3D ClipToBounds="True" Width=
    "150" Height="150">
      <Viewport3D.Camera>
        <PerspectiveCamera Position="0,0,2"/>
      </Viewport3D.Camera>
      <Viewport3D.Children>
        <ModelVisual3D>
          <ModelVisual3D.Content>
            <DirectionalLight Color="White"
            Direction="-0.5,-0.1,-0.5" />
          </ModelVisual3D.Content>
        </ModelVisual3D>
        <ModelVisual3D>
          <ModelVisual3D.Content>
            <GeometryModel3D>
              <GeometryModel3D.Geometry>
                <MeshGeometry3D Positions="-
                0.5,-0.5,0.5 0.5,-0.5,0.5
                0.5,0.5,0.5 0.5,0.5,0.5 -0.5,
                0.5,0.5 -0.5,-0.5,0.5 " />
              </GeometryModel3D.Geometry>
              <GeometryModel3D.Material>
                <DiffuseMaterial Brush="Red">
                </DiffuseMaterial>
              </GeometryModel3D.Material>
              <GeometryModel3D.Transform>
                <RotateTransform3D>
                  <RotateTransform3D.
                  Rotation>
                    <AxisAngleRotation3D
                    Axis="0,3,0" Angle="40"/>
                  </RotateTransform3D.
                  Rotation>
                </RotateTransform3D>
              </GeometryModel3D.Transform>
            </GeometryModel3D>
          </ModelVisual3D.Content>
        </ModelVisual3D>
      </Viewport3D.Children>
      <Viewport3D.Triggers>
        <EventTrigger RoutedEvent=
        "Viewport3D.Loaded">
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation From="1"
              To="360" Duration="0:0:03"
              AccelerationRatio="1"
              RepeatBehavior="Forever"
              Storyboard.TargetProperty=
              "(Viewport3D.Children)[1].
              (ModelVisual3D.Content).
              (GeometryModel3D.Transform).
              (RotateTransform3D.Rotation).
              (AxisAngleRotation3D.Angle)">
              </DoubleAnimation>
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
      </Viewport3D.Triggers>
    </Viewport3D>
  </Grid>
</Window>
```

FIGURA 041 | Secuencia de imágenes que muestran cómo se desarrolla la rotación de un objeto.

Windows Communication Foundation

Evolución del paradigma

14

Contenidos

En este capítulo, aprenderemos el concepto de aplicaciones distribuidas, conoceremos este nuevo paradigma de desarrollo y cómo implementarlo, y veremos de qué manera nace Windows Communication Foundation.

Temas tratados

- Conceptos
- Historia sobre aplicaciones distribuidas
- Contratos
- Binding y transporte de mensaje
- Seguridad en WCF

Windows Communication Foundation

Microsoft introdujo en su Framework 3.0 la solución para el desarrollo de aplicaciones distribuidas en una sola tecnología que conoceremos en profundidad.

Conceptos

Aprenderemos el concepto de aplicaciones distribuidas, las compararemos con objetos cotidianos y veremos ejemplos de este tipo de aplicaciones para, así, entenderlo claramente.

- Comunicación entre aplicaciones y aplicaciones distribuidas
- Similitudes con objetos cotidianos
- Tipos de aplicativos

Historia sobre aplicaciones distribuidas

Analizaremos la historia de la evolución del desarrollo de aplicaciones distribuidas y veremos cuál es su futuro.

- Aplicaciones cliente-servidor
- COM, COM+ y DCOM
- Remoting

Contratos

Aprenderemos cuál es la forma de generar un servicio creado con Windows Communcation Foundation, los contratos que debe respetar y para qué sirven.

- Service Contracts
- Data Contracts
- Message Contracts

Binding y transporte de mensaje

Aprenderemos qué tipo de enlaces podemos usar con servicios creados con WCF, las ventajas que tiene y ejemplos prácticos acerca de cómo implementar estos enlaces.

- Canales de transporte
- Codificación y Formateadores
- Enlaces dual-channel.
- EndPoints y Address

Seguridad en WCF

Conoceremos los nuevos modelos de seguridad que propone Microsoft para la implementación en conjunto con Windows Communication Foundation.

- Aspectos importantes
- Seguridad de transferencias
- Control de acceso
- Auditoría

Windows Communication Foundation

De aquí en adelante nos referiremos a esta nueva solución que Microsoft incorporó a partir del Framework 3.0.

Entre las nuevas tendencias, la construcción de software basado en componentes es, sin duda, uno de los mayores aciertos. ¿Por qué? ¿Alguna vez nos hemos puesto a pensar qué similitud tiene la estructura de un automóvil con la de un aplicativo informático? Si se daña una puerta del auto, nos centramos en repararla o, en el peor de los casos, es reemplazarla; no estamos pensando en ningún otro componente. Así mismo sucede en el software basado en componentes: actualizamos, modificamos, corregimos o cambiamos sólo el componente que necesitamos; el resto sigue intacto y funcionando tal como lo hacía antes. Entonces, bajo el mismo ejemplo, si queremos transportar personas en los autos, debemos preocuparnos por el canal de transporte (carretera), por la forma o el tipo de transporte (auto, camión, bicicleta) y por lo que se transporta (tipo de carga, cantidad de personas, con carga o sin carga). Entendido este ejemplo, acabamos de explicar qué es una aplicación distribuida. Estos tipos de aplicaciones basadas en

Aplicaciones punto a punto

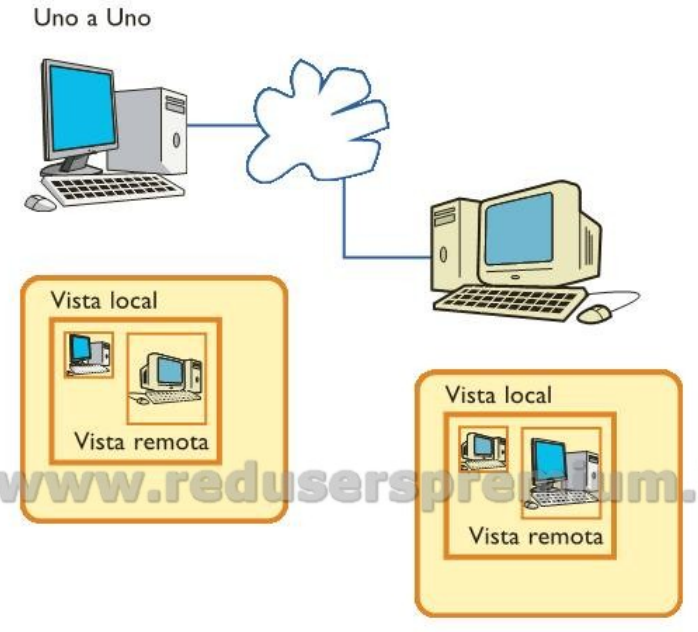

FIGURA 001 | Vemos cómo es la interacción entre aplicaciones punto a punto.

componentes distribuidos son las que en la actualidad se están desarrollando con mayor fuerza. Es frecuente encontrar un sitio Web que nos permita comprar un determinado artículo o pagar cierto servicio. De la misma manera, dentro de las empresas es común tener aplicaciones destinadas a facturación, inventarios, clientes, etcétera, en las que interviene mucha gente del personal, como cajeros, contadores, y otros. Es bajo estos escenarios que las aplicaciones distribuidas o interconectadas sacan a relucir su verdadero potencial y aporte a la vida cotidiana de personas y empresas, que se valen de la tecnología para ser más productivas y competitivas. Existen algunos aplicativos que sólo se ejecutan en una computadora y no necesitan conectividad para funcionar. Por otra lado, están las aplicaciones punto a punto, que tienen conectividad específica o direccionada hacia otra computadora; y las distribuidas propiamente dichas, en las que cada componente puede estar en una o en varias máquinas y funcionar como un todo, a la vez que puede prestar servicio a otro conjunto de computadoras.

Pero antes de continuar, hagamos una breve reseña histórica acerca de cómo fue evolucionando esta tecnología. Las aplicaciones cliente-servidor son, en realidad, las precursoras de los ambientes distribuidos, en cuyos inicios todo el procesamiento estaba centralizado en un mainframe, y las estaciones sin capacidad de procesamiento (bobas) se conectaban al servidor central para poder procesar. En lo que concierne a la construcción de aplicaciones distribuidas bajo tecnología Microsoft, sus orígenes están en el COM, DCOM y el COM+. El COM siempre será considerado como el

Aplicaciones distribuidas

FIGURA 002 | En este caso, una misma aplicación es distribuida a diferentes clientes.

inicio de lo que denominamos aplicaciones construidas por componentes. Continuando con la historia, sigue el DCOM, que no es otra cosa que el COM distribuido; esto significa, permitir el acceso a otras aplicaciones o componentes a través de la red. Sin embargo, luego apareció COM+, que en realidad revolucionó el mundo informático con la implementación de aplicaciones distribuidas mediante un esfuerzo relativamente bajo. La incorporación de MTS tiene un papel muy importante desde entonces, ya que controla la consistencia de la información entre la base de datos y los aplicativos, tiene la capacidad de utilizar y controlar operaciones asincrónicas, y a través de él se pueden definir pools de objetos (varias instancias de objetos para atender múltiples solicitudes al mismo tiempo). Cuando nace el .NET Framework, la tecnología COM+ fue renombrada y pasó a llamarse Enterprise Services. Con el .NET Framework apareció un nuevo modelo de comunicación y distribución de objetos; la tecnología que brinda la capacidad de que un cliente remoto acceda a un componente alojado en un servidor como si estuviese trabajando localmente. Todos los modelos anteriores son válidos, pero su principal uso se da en las intranets (LAN). Por lo tanto, cuando surge la necesidad de comunicar aplicaciones a través de Internet, estas tecnologías resultan muy limitadas y es entonces cuando aparecen los servicios Web.

Implementación COM

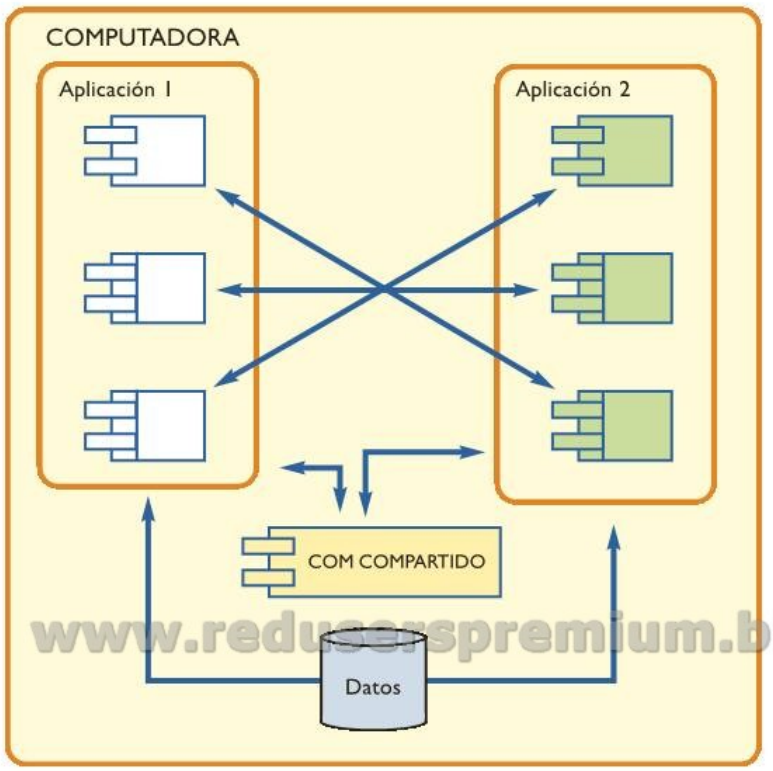

FIGURA 003 | Representación de la interacción de componentes en una implementación COM.

Aplicaciones COM+ y MTS

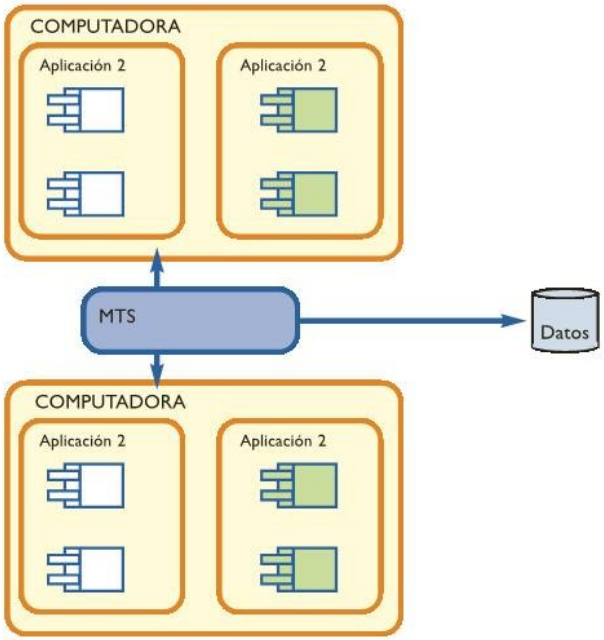

FIGURA 004 | Representación de la interacción de componentes y aplicaciones con COM+ y MTS.

Web Services

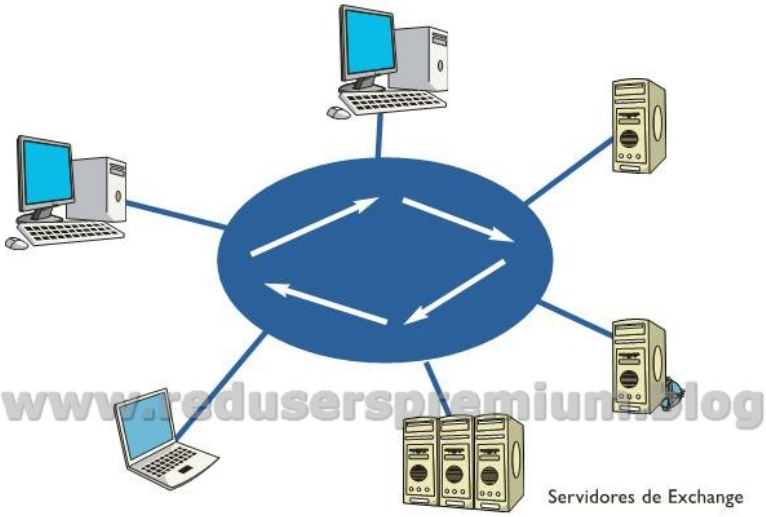

FIGURA 005 | Ésta es la manera en que interactúa un Web Service.

Concepto de servidor, cliente y mensaje

Veremos las características que WCF nos ofrece en el aspecto concreto de la comunicación.

El último modelo o tecnología provisto por Microsoft para resolver los escenarios de aplicaciones distribuidas es *Windows Communication Foundation* (WCF), cuyo nombre código durante su fase de construcción fue Indigo. WCF es uno de los cuatro componentes (WCF, WPF, WF y CardSpace) que se incluyen en el Framework 3.0 (llamado WinFX en su fase de construcción). Todos los que vienen en el .NET Framework 3.0 son una extensión del .NET Framework 2.0. Éste se encuentra incluido de forma predeterminada en Windows Vista.
WCF también está disponible en el recientemente liberado .NET Framework 3.5, que es la unión del .NET Framework 2.0, su respectivo SP1, más el .NET Framework 3.0 y su respectivo SP1, además de nuevas funcionalidades, como C# 3.0 y LINKQ.
WCF es una plataforma de comunicaciones y mensajería diseñada para unificar e implementar las tecnologías de aplicaciones distribuidas existentes. Es por eso que puede interoperar con múltiples tecnologías, tales como:

- WSE 3.0
- System.Messaging
- NET Enterprise Services
- Servicios Web ASP.NET (asmx)
- .NET Remoting
- MSMQ

Esto nos dice que WCF, casi sin un solo cambio a nivel de líneas de código, puede intercambiar información con cualquiera de las tecnologías anteriores y, así, darnos un modelo estándar, fácil y rápido de utilizar para construir aplicaciones distribuidas.
También puede interactuar con cualquier plataforma que utilice SOAP como protocolo de transporte (por ejemplo, Java2, Web Logic, etc.).
La arquitectura de WCF puede ser denominada cliente-servidor, y esto no debe asustar a los que piensan que este modelo es antiguo, caduco, etc. WCF siempre publicará algo, y esperará que un cliente se conecte para consumirlo.
Otras de las grandes premisas de implementación de WCF es que nuestros sistemas deben estar enmarcados dentro de una arquitectura SOA (arquitectura orientada a servicios).
Esta definición suele ser muy genérica, por lo que se prefiere decir que SOA es un conjunto de servicios organizados de una forma coherente, que pueden ser expuestos hacia el mundo para que cualquier aplicativo pueda consumirlos de un modo seguro y sin ambigüedades.
Los principales componentes de SOA son:

- Publicador (servidor)
- Subscriptor (cliente)
- Mensaje

SOA es definido como "un conjunto de componentes que pueden ser invocados, y cuyas descripciones de interfaz pueden ser descubiertas y publicadas".

Entonces, un servidor no es otra cosa que un proceso encargado de recibir las peticiones de múltiples clientes, solicitando acceso y ejecución de algún recurso. Los procesos en servidores son nombrados, generalmente, backends, y suelen tener los siguientes roles:

- Aceptar los requerimientos de datos que realizan los clientes.
- Procesar requerimientos de bases de datos.
- Formatear datos para entregar a los clientes.
- Procesar lógica y reglas de negocio, así como validaciones propias del negocio.
- Administrar y optimizar los accesos a las bases de datos.

El cliente, en cambio, puede ser un proceso o un aplicativo que permita estructurar requerimientos y enviarlos al servidor, también denominado front-end. Tiene la capacidad de conectarse a cualquier servicio distribuido disponible en su red, y sus principales roles son:

- Facilitar la interacción con el usuario.
- Administrar y desplegar la interfaz de usuario.
- Generar requerimientos hacia el servidor.
- Recibir los resultados del servidor.
- Mostrar de una forma amigable cualquier tipo de resultado.

Otro de los términos a nivel de estructura que debemos entender muy bien es el de mensaje. Se trata de los datos que intercambian tanto el cliente como el servidor, y puede contener los siguientes elementos:

- Información desde el cliente hacia el servidor, denominada parámetros.

Modelos distribuidos

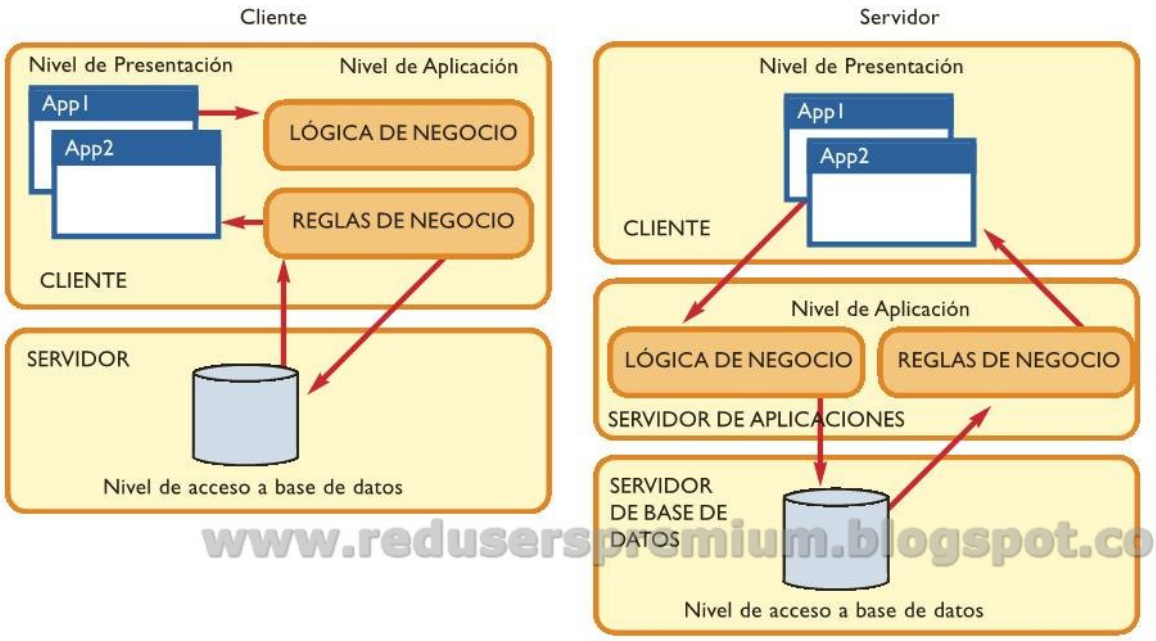

FIGURA 006 | Representación de la clasificación de modelos distribuidos cliente-servidor.

- Información del servidor al cliente, denominada objetos de retorno.
- Cabeceras de seguridad, de identificación, de compresión o encriptación.
- Metadata que ayuda a identificar la información que contiene el mensaje.

Un servicio WCF es un componente que se expone en un host, y que muestra como interfaz de entrada los denominados end-points, cada uno de los cuales es un punto de entrada al mundo exterior. Entonces, un cliente WCF es un programa que realiza intercambio de mensajes con uno o varios end-points, donde este cliente también tiene la capacidad de publicar sus respectivos end-points para recibir información del servidor. Este modelo es de tipo dúplex.
Una manera didáctica de entender los end-points es decir que sus componentes forman las siglas ABC, tal como se detalla en la Tabla 1.

Hablando de los contratos, existen algunos tipos que intervienen dentro de la implementación de WCF, como son:

- Service Contract
- Data Contract
- Message Contract

Service Contract

El Service Contract no es otra cosa que una interfaz con la descripción del método que vamos a exponer como servicio. Podrá ser identificado por cualquier cliente y, sobre la base de esta interfaz, se podrá suscribir para poder utilizar el servicio.
La forma de implementar esta interfaz es colocando como atributo de una interfaz la clase ServiceContractAttibute. Ésta se encuentra en el namespace **System.ServiceModel**, en el ensamblado System.ServiceModel.dll.
La forma de crear la interfaz de ServiceContract es la siguiente, en lenguaje C#:

```
using System;
using System.ServiceModel;

namespace WCF.Ejemplo {
    /// <summary>
    /// Este interface es lo que se conoce
    como contrato
    /// (Service Contract)
    /// </summary>
    [ServiceContract]
    public interface IEmpleado {
    }
}
```

El atributo **ServiceContract** puede inicializarse con varios modificadores, que se detallan en la Tabla 2.

Además, cada método que será mostrado hacia el mundo exterior en el servicio debe tener un atributo llamado OperationContract. La clase

Tabla 1 | Distintos tipos de end-points

Address – Dirección	Dirección de red donde se localiza el end-point.
Binding – Enlace	Define cómo se comunica el end-point con el resto de las aplicaciones. Aquí se define protocolo (TCP, SOAP, etc.), tipo de codificación (binario, texto) y directivas de seguridad (SSL, o basada en mensajes).
Contract – Contrato	Especifica qué comunica el end-point, o bajo qué patrón de intercambio de mensajes se comunica (una vía, dúplex, request, response).

OperationContractAttribute está en el namespace **System.ServiceModel**, en el ensamblado System.ServiceModel.dll.
Este atributo tiene algunos modificadores de comportamiento, detallados en la Tabla 3.

Ejemplo completo de un servicio y su implementación:

```
using System;
using System.Net.Security;
using System.ServiceModel;

namespace WCF.Ejemplo
{
   [ServiceContract(Namespace="WCF.Ejemplo")]
   public interface IEmpleado
   {
     [OperationContract(
       ProtectionLevel=ProtectionLevel.
       EncryptAndSign)]
     string Consultar(string msg);
   }

   class Empleado : IEmpleado
   {
     public string  SampleMethod(string msg)
     {
       Console.WriteLine("Llamando con:
       {0}", msg);
       return "El service proceso: " + msg;
     }
   }
}
```

Una vez que hemos entendido cómo publicar un servicio y sus operaciones, debemos comprender cómo transportar los mensajes a través de la tecnología WCF.

Data Contract

Son implementaciones de clases que pueden atravesar el canal preparado por WCF para comunicarse. Estas clases tienen la propiedad de serializarse, y de transportar información de una manera controlada y administrada por WCF.
Este tipo de contrato se aplica a las estructuras (struct) y clases (class). Se usa cuando el servicio necesita devolver más que un tipo simple.
Los contratos de datos presentan las siguientes características:

- Definen el formato de la información que se pasa hacia y desde el servicio.
- Constituyen la estructura y el tipo de los mensajes que intercambian los servicios.
- Mapean un tipo del CLR a un esquema XML.

Tabla 2 | Modificadores de inicialización de ServiceContract

Nombre	Descripción
ConfigurationName	Propiedad que especifica el nombre del elemento service en el archivo de configuración.
Name y Namespace	Nombre y namespace del elemento <portType> en el WSDL.
SessionMode	Propiedad que especifica cómo el contrato requiere conectarse para soportar las sesiones (requerida, permitida, no permitida).
CallBackContract	Propiedad que define el retorno de contrato en una conversación de dos vías (dúplex).
ProtectionLevel	Especifica el nivel de protección que tiene cada mensaje (encriptado y firmado, sólo firmado, sin protección).

- Indican cómo los tipos de datos que componen la estructura se serializan y deserializan.

La clase deberá tener su atributo **DataContract**, y cada propiedad de la clase, un atributo **DataMember**. Una clase **DataContract** podrá ser utilizada como parámetro o como retorno en un **OperationContract**.
Un ejemplo de Data Contract es el siguiente:

```
using System;
using System.ServiceModel;

namespace WCF.Ejemplo {
    [DataContract]
    public struct Contacto {
        [DataMember]
        public string Nombre;
        [DataMember]
        public string Apellidos;
        [DataMember]
        public DateTime FechaNacimiento;
        [DataMember]
        public string Email;
    }
}
```

Message Contract

Esta clase define el contenido de un mensaje SOAP; esto es, su cabecera y el cuerpo. No es muy común su uso dado que la estructura del mensaje se vuelve muy compleja.
El body o cuerpo de un **MessageContract** puede ser un **DataContract**; un ejemplo es el siguiente:

```
[MessageContract]
public class MiMensaje
{
   [MessageHeader]
    public string field1;
   [MessageBody]
    public MyClass field2;
}
```

Los atributos para los miembros de la clase **MessageContract** son el **MessageHeader**, que identifica los valores de cabecera; y el **MessageBody**, que identifica los valores del cuerpo del mensaje.

Tabla 3 | Modificadores del atributo OperationContract

Nombre	Descripción
Action	La estructura del mensaje permite enviar la acción que se va a ejecutar. En esta propiedad se puede definir si la acción del mensaje pertenece a este método.
Asyncpattern	Esta propiedad indica si esta operación puede ser invocada asincrónicamente.
IsOneWay	Indica si la operación sólo recibe un mensaje de entrada. La operación no puede ser asociada a un mensaje de salida si está en true.
IsInitiating	Indica que la operación puede ser considerada como un inicio de sesión.
IsTerminating	Si es verdadera, cuando es invocada, WCF terminará la sesión en curso, luego de que todas las operaciones se concreten.
ProtectionLevel	Especifica el nivel de protección que tiene cada mensaje (encriptado y firmado, sólo firmado, sin protección).
ReplyAction	Define la acción del mensaje de respuesta sobre esta operación.

Conceptos de binding y transporte de mensajes

Como mencionamos en uno de los temas anteriores, una definición básica y concreta de binding es: "Define cómo se comunica el end-point con el resto de las aplicaciones; aquí se define el protocolo (TCP, SOAP, etc.), tipo de codificación (binario, texto) y directivas de seguridad (SSL o basada en mensajes)".
Entonces, para comprender un poco más este tema, si los contratos definen qué hace el servicio, los bindings definen cómo acceder a él. Por lo tanto, se dividen en tres partes, detalladas en la Tabla 4.

Los bindings especifican los mecanismos de comunicación que usaremos para "hablar" con un end-point. Éstos pueden ser configurados en el archivo app.config o web.config, dependiendo de qué host estemos utilizando para alojar nuestros servicios.
Un binding tiene como componentes su nombre, un namespace (espacio de nombres de programación que lo identifica) y una colección de elementos binding compuestos.

El nombre y el namespace del binding lo identifican de forma única en los metadatos del servicio. Cada elemento del binding describe un aspecto de cómo se comunica el end-point con el exterior.

El .NET Framework 3.0 o 3.5 nos ofrece una serie de bindings probados que podemos configurar y usar sin escribir ni una sola línea de código en C#, VB.NET, etc. Con ellos definiremos cómo queremos que se codifiquen los datos, la seguridad de las comunicaciones, la compresión, el protocolo que vamos a usar, etc.
Nosotros podemos crear nuestros propios bindings si nuestro proyecto usa algún protocolo propio o algún tipo de mecanismo de comunicación diferente de los que nos da .NET Framework, para ello se debe crear una clase que herede de CustomBinding.

> Los bindings especifican los mecanismos de comunicación que usaremos para "hablar" con un end-point.

Pero primero expliquemos los bindings que vienen construidos por defecto en WCF, los denominados System-Provided Bindings: ellos son BasicHttpBinding (Tabla 5), WSHttpBinding (Tabla 6), WS2007HttpBinding (Tabla 7), WSDualHttpBinding (Tabla 8), WSFederationHttpBinding (Tabla 9), WS2007FederationHttpBinding (Tabla 10),

Tabla 4 | Partes de un binding

Canal de transporte	Es el lenguaje con que se comunica o se accede al servicio: por ejemplo, TCP, HTTP, etc.
Codificación	Indica bajo qué formato se envían los mensajes: binario, text/XML o mecanismos de optimización para la transmisión de mensajes (*Message Transmission Optimization Mechanism*, MTOM).
Canal de protocolo	Define cómo se van a comportar los mensajes dentro de la comunicación; por ejemplo, seguridad, fiabilidad o, incluso, protocolos propios creados por el usuario, etc.

NetTcpBinding (Tabla 11), NetNamedPipeBinding (Tabla 12), NetMsmqBinding (Tabla 13), NetPeerTcpBinding (Tabla 14) y MsmqIntegrationBinding (Tabla 15).

Para entender algunos términos de los que hemos mencionado anteriormente, listaremos en la Tabla 16 algunos conceptos importantes en lo que a WCF y transporte se refiere.

La clase CustomBinding se encuentra en el namespace System.ServiceModel.Channels, en el ensamblado System.ServiceModel.dll.

Clase Binding

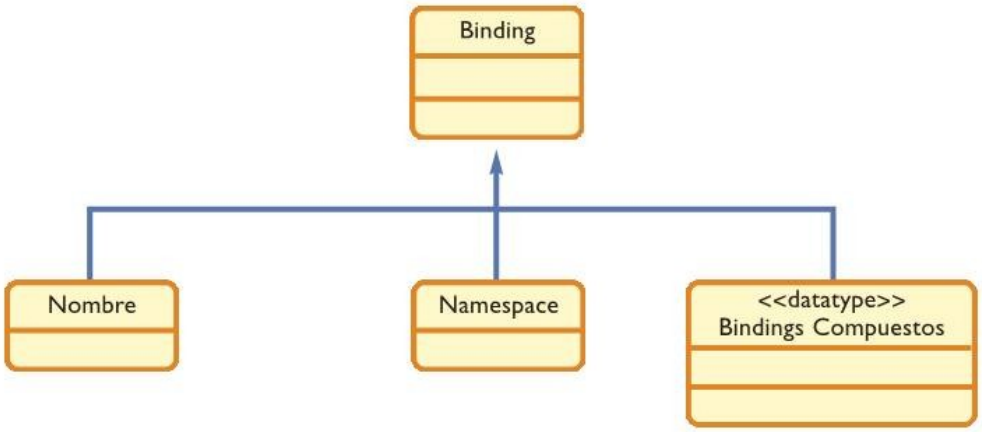

FIGURA 007 | Representación de la estructura de una clase Binding.

Tabla 5 | Detalles del binding BasicHttpBinding

Nombre	BasicHttpBinding
Elemento en la configuración	<basicHttpBinding>
Interoperabilidad	Basic Profile 1.1
Modo de seguridad (por defecto)	(None), Transport, Message, Mixed
Sesión (por defecto)	None, (None)
Transacción	None
Dúplex	n/a
Descripción	Es el modelo de comunicación más apropiado para implementar Web Services, ya que define el estándar WS-Basic Profile, que es el que implementan los Web Services asp.net (asmx). Este binding utiliza un transporte tipo HTTP con un encoding text/XML.

Un ejemplo de cómo configurar un binding es el siguiente:

```
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="WCF.MiServicio">
        <endpoint address=""
                  binding="wsHttpBinding"
                  contract="WCF.IMiServicio" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

```
<bindings>
  <wsHttpBinding>
    <binding name="MyCustomBinding"
             transactionFlow="true" />
  </wsHttpBinding >
</bindings>
```

End-points y addresses

¿Recuerdan el ABC de los end-points? Es una forma muy didáctica de memorizar los componentes de un end-point address (dirección o ubicación), binding (enlace o conectividad) y contract (Contrato).

Tabla 6 | Detalles del binding WSHttpBinding

Nombre	WSHttpBinding
Elemento en la configuración	<wsHttpBinding>
Interoperabilidad	WS
Modo de seguridad (por defecto)	(None), Transport, Message, Mixed
Sesión (por defecto)	(None), Transport, Reliable Session
Transacción	(None), Yes
Dúplex	n/a
Descripción	Binding similar a basicHttpBinding, con la diferencia de que soporta transaccionalidad (sin embargo, dúplex no soporta).

Tabla 7 | Detalles del binding WS2007HttpBinding

Nombre	WS2007HttpBinding
Elemento en la configuración	<ws2007HttpBinding>
Interoperabilidad	WS-Security, WS-Trust, WS-SecureConversation, WS-SecurityPolicy
Modo de seguridad (por defecto)	(None), Transport, Message, Mixed
Sesión (por defecto)	(None), Transport, Reliable Session
Transacción	(None), Yes
Dúplex	n/a
Descripción	Un binding muy robusto que permite configurar las seguridades, soporta sesión e, incluso, el manejo de transaccionalidad.

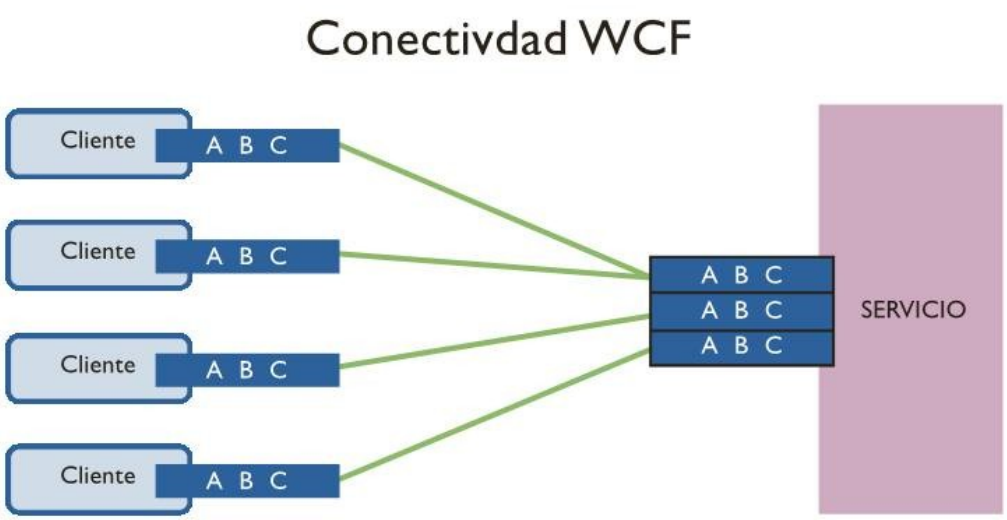

FIGURA 008 | Representación de la conectividad WCF con end-points.

Tabla 8 | Detalles del binding WSDualHttpBinding

Nombre	WSDualHttpBinding
Elemento en la configuración	<wsDualHttpBinding>
Interoperabilidad	WS
Modo de seguridad (por defecto)	None, (Message)
Sesión (por defecto)	(Reliable Session)
Transacción	(None), Yes
Dúplex	Sí
Descripción	Permite comunicación dúplex a través de un canal SOAP.

Tabla 9 | Detalles del binding WSFederationHttpBinding

Nombre	WSFederationHttpBinding
Elemento en la configuración	<wsFederationHttpBinding>
Interoperabilidad	WS-Federation
Modo de seguridad (por defecto)	None, (Message), Mixed
Sesión (por defecto)	(None), Reliable Session
Transacción	(None), Yes
Dúplex	No
Descripción	Soporta el protocolo WS-Federation, habilitando la autenticación y autorización de usuarios que pertenecen a una organización.

Los servicios pueden exponer más de un end-point, que es a donde se va a conectar cada cliente para enviar los mensajes. Pueden mostrar las descripciones de sus end-points para que los clientes puedan conectarse a ellos. Esto se realiza, generalmente, a través de WSDL y MEX, por lo que los clientes pueden hacer uso de la información facilitada para generar el código necesario de conexión para enviar los mensajes hacia el servicio.

La clase ServiceEndPoint representa un end-point en el entorno de programación de WCF, y tiene un EndPointAddress, un binding y un ContractDescription.
El EndPointAddress o la dirección no es otra cosa que un identificador más un URI y algunas cabeceras opcionales.

La identidad de seguridad generalmente es provista por el propio URI. Sin embargo, hay escenarios complejos y especiales en los cuales es necesario brindar específicamente una identidad. El AddressHeader, o las cabeceras opcionales, se utilizan para definir información adicional de direccionamiento.
Los end-points son configurados en los archivos app.config o web.config, lo que le da una verdadera flexibilidad al modelo de comunicaciones, ya que se puede cambiar de dirección únicamente cambiando el archivo de configuración, sin tener que escribir ni una sola línea de código ni recompilar la solución.
Implementar código fijo de información de end-point dentro de la aplicación host no es lo ideal, debido a que los detalles de los end-points pueden cambiar a lo largo del tiempo

Tabla 10 | Detalles del binding WS2007FederationHttpBinding

Nombre	WS2007FederationHttpBinding
Elemento en la configuración	<ws2007FederationHttpBinding>
Interoperabilidad	WS-Federation
Modo de seguridad (por defecto)	None, (Message), Mixed
Sesión (por defecto)	(None), Reliable Session
Transacción	(None), Yes
Dúplex	No
Descripción	Este binding hereda las funcionalidades de WS2007HttpBinding y soporta, además, federación.

Tabla 11 | Detalles del binding NetTcpBinding

Nombre	NetTcpBinding
Elemento en la configuración	<netTcpBinding>
Interoperabilidad	.NET
Modo de seguridad (por defecto)	None, (Transport), Message, Mixed
Sesión (por defecto)	Reliable Session, (Transport)
Transacción	(None), Yes
Dúplex	Sí
Descripción	Este binding es el más apropiado para comunicarse entre aplicaciones WCF.

Clase EndPointAddress

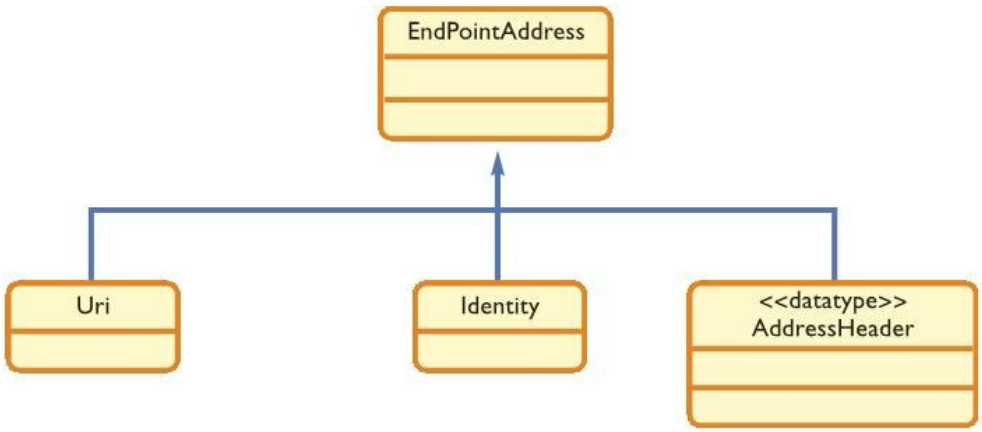

FIGURA 009 | Representación de la estructura de una clase EndPointAddress.

Tabla 12 | Detalles del binding NetNamedPipeBinding

Nombre	NetNamedPipeBinding
Elemento en la configuración	<netNamedPipeBinding>
Interoperabilidad	.NET
Modo de seguridad (por defecto)	None, (Transport), Message, Mixed
Sesión (por defecto)	Reliable Session, (Transport)
Transacción	(None), Yes
Dúplex	Sí
Descripción	Este binding es seguro, fiable y un optimizado modelo de comunicación en la misma máquina de aplicaciones WCF.

Tabla 13 | Detalles del binding NetMsmqBinding

Nombre	NetMsmqBinding
Elemento en la configuración	<netMsmqBinding>
Interoperabilidad	.NET
Modo de seguridad (por defecto)	None, Message, (Transport), Both
Sesión (por defecto)	(None)
Transacción	(None), Yes
Dúplex	No
Descripción	Es un binding tipo cola, apropiado para comunicación entre aplicaciones WCF.

(aunque también se lo puede hacer). No es difícil imaginarse cómo podría guardarse la información del end-point en un archivo de configuración o base de datos, y leerlo mientras se inicializa el ServiceHost. Debido a que es un requerimiento común, WCF proporciona esa funcionalidad de manera automática, mediante la sección de configuración predefinida llamada <system.serviceModel>.

En la siguiente configuración se definen algunos end-points de ejemplo:

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="WCF.Ejemplo.Empleado">
        <host>
          <baseAddresses>
            <add baseAddress="http://local
            host:8080/Ejemplo/"/>
          </baseAddresses>
        </host>

        <endpoint address="svc"
          binding="basicHttpBinding"
          contract="WCF.Ejemplo.IEmpleado"/>

        <endpoint address="net.tcp://local
        host:8081/Ejemplo/svc"
          binding="netTcpBinding"
          contract="WCF.Ejemplo.IEmpleado"/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Tabla 14 | Detalles del binding NetPeerTcpBinding

Nombre	NetPeerTcpBinding
Elemento en la configuración	<netPeerTcpBinding>
Interoperabilidad	Peer
Modo de seguridad (por defecto)	None, Message, (Transport), Mixed
Sesión (por defecto)	(None)
Transacción	(None)
Dúplex	Sí
Descripción	Permite la comunicación segura entre múltiples máquinas.

Tabla 15 | Detalles del binding MsmqIntegrationBinding

Nombre	MsmqIntegrationBinding
Elemento en la configuración	<msmqIntegrationBinding>
Interoperabilidad	Peer
Modo de seguridad (por defecto)	None, Message, (Transport), Mixed
Sesión (por defecto)	(None)
Transacción	(None)
Dúplex	Sí
Descripción	Es un binding que permite la comunicación entre máquinas con aplicaciones WCF y, también, con colas ya creadas en MSMQ.

Dentro del archivo de configuración, para configurar el address del end-point debemos nombrar o definir el URI de una forma adecuada. Por ejemplo, si nuestro binding es de tipo HTTP, nuestro URI comenzará con **http://**, pero si nuestro binding es TCP, nuestro URI iniciará con **net.tcp://**.

El end-point se configura tanto en el servicio como en el cliente. ¿Por qué? Simplemente, para tener una coherencia con la forma de conectarnos hacia el servicio, y si el cliente está utilizando un binding tipo dúplex en algún momento, por un instante puede convertirse en servicio.

> En un cliente podemos programar o construir un end-point, al igual que en el servidor.

El desarrollo de clientes WCF requiere del envío de mensajes a end-points de servicios de acuerdo con su dirección, binding y contrato. Para lograrlo, lo primero que debemos hacer es instanciar un objeto tipo Service-Endpoint representando el end-point objetivo. El siguiente ejemplo muestra cómo crear

Tabla 16 | Términos y conceptos referentes a WCF

Comunicación dúplex	Especifica si los contratos de comunicación de doble vía son soportados. Esta opción requiere soportar sesión en el binding.	
Sesión	Especifica si los contratos soportan sesión; esto es, permitir mantener un contexto mientras el canal esté abierto.	
Transacción	Especifica si los modelos de transaccionalidad están habilitados; esto es, para crear o soportar transacciones.	
Interoperabilidad	Es la forma en que los protocolos y tecnologías pueden asegurar la interconexión con los bindings de WCF.	
Modo de seguridad	Especifica los modelos de cómo asegura el canal:	
	None	El mensaje SOAP no está asegurado, y el cliente no ha sido autenticado.
	Transport	Los requerimientos de seguridad se determinan a nivel de transporte.
	Message	Los requerimientos de seguridad se determinan a nivel de mensaje.
	Mixed	Este modo incluye los requerimientos de seguridad en el transporte y en el mensaje.
	Both	Es el nivel de transporte y seguridad para poder implementar únicamente el netMsmqBinding.

Tabla 17 | Namespace de cada uno de los componentes

End-point	System.ServiceModel.ServiceEndpoint
Address	System.Uri
Binding	System.ServiceModel.Binding
Contract	Interfaces decorados con atributos System.ServiceModel

un ServiceEndPoint basado en el end-point HTTP del servicio, en un proceso de aplicación de consola:

C#:

```
using System;
using System.ServiceModel;
using WCF.Ejemplo;

class Program {
  static void Main(string[] args) {
    // Definimos endpoints en el cliente
    basados en el archive de configuración
    anterior
    ServiceEndpoint httpEndpoint = new
    ServiceEndpoint(ContractDescription.
    GetContract(typeof(IEmpleado)), new
    BasicHttpBinding(), new EndpointAddress
    ("http://localhost:8080/Ejemplo/svc"));
    ServiceEndpoint tcpEndpoint = new
    ServiceEndpoint(ContractDescription.
    GetContract(typeof(IEmpleado)), new
    NetTcpBinding(), new EndpointAddress
    ("net.tcp://localhost:8081/Empleado/
    svc"));

    IEmpleado svc = null;

    // creamos el channel-factory basado en
    endpoint-HTTP
    using (ChannelFactory<IEmpleado>
    httpFactory = new ChannelFactory
    <IEmpleado>(httpEndpoint)) {
      // creamos un channel-proxy para el
      endpoint
      svc = httpFactory.CreateChannel();
      // invocamos la operación del Servicio
      ...
    }
    // creamos un channel-factory basado en
    endpoint TCP
    using (ChannelFactory<IEmpleado>
    tcpFactory = new ChannelFactory
    <IEmpleado>(tcpEndpoint)) {
      // creamos el channel-proxy para el
      endpoint
      svc = tcpFactory.CreateChannel();
      // invocamos la operación del Servicio
      ...
    }
  }
}
```

Esta forma de construir un end-point no es ni debe ser la más habitual, ya que se encuentra quemado en código. Por lo tanto, lo recomendable es hacerlo a través del archivo de configuración; si generamos el proxy a través del propio Visual Studio, éste nos dará el archivo de configuración automáticamente.
Veamos el ejemplo del archivo de configuración descrito anteriormente:

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="httpEndpoint"
        address="http://localhost:8080/
        Ejemplo/svc"
        binding="basicHttpBinding"
        contract=" WCF.Ejemplo.IEmpleado"/>

      <endpoint name="tcpEndpoint"
        address="net.tcp://localhost:8081/
        echo/svc"
        binding="netTcpBinding"
        contract=" WCF.Ejemplo.IEmpleado"/>
    </client>
  </system.serviceModel>
</configuration>
```


Dado que cada end-point en el cliente tiene nombre, en el código, cuando queramos invocar uno, podemos hacerlo por el nombre, o dejar que WCF escoja el predefinido.

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

21

Gráficos

Conceptos de animaciones / Dos y tres dimensiones

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Windows Communication Foundation

Reseña histórica de las comunicaciones / Comunicación entre aplicaciones / Conceptos de servidor, cliente y mensaje

ISBN 978-987-1347-43-8
00021
9 789871 347438