



DIPLOMATURA EN PROGRAMACION EN .NET
Proyecto Ventas

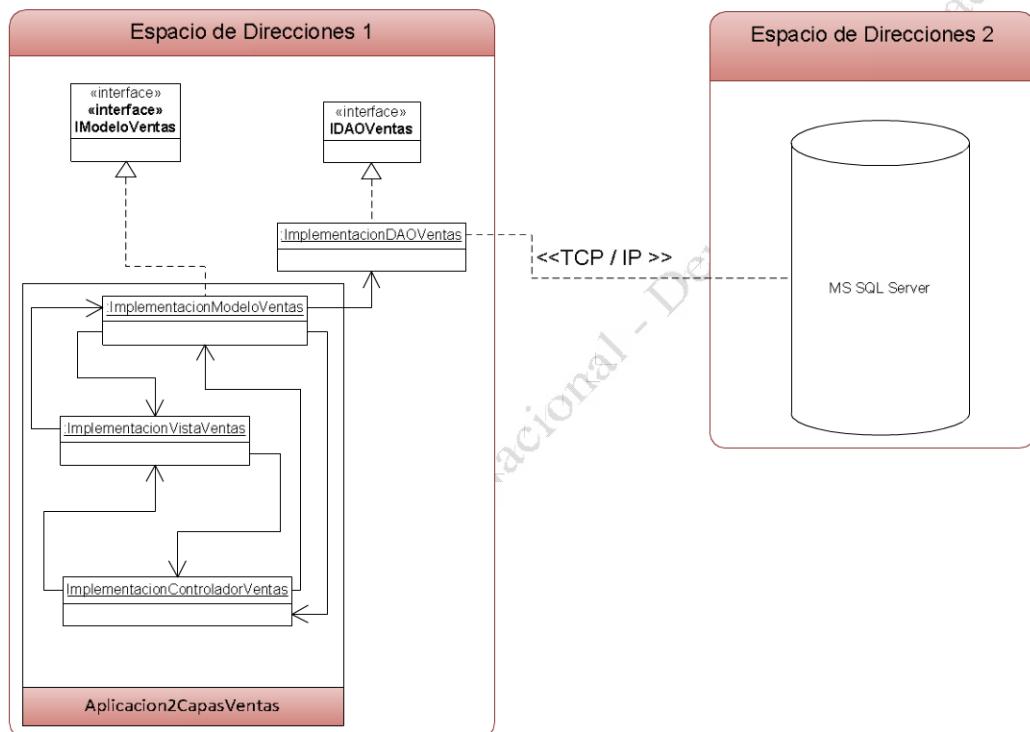
Introducción a WCF y la
programación distribuida

Comparando los diseños de dos y tres capas

En esta sección se comparan los diseños realizados en las construcciones 4 y 5 del proyecto Ventas. Para progresar desde la construcción 4 a la 5, se necesita iteraciones del ciclo de desarrollo de software para completar análisis, diseño e implementación.

Diseño de dos capas de Ventas

La siguiente figura muestra los objetos asociados en la construcción 3 destacando los distintos espacios de direcciones de memoria que ocupan cada uno en la aplicación



Los objetos alojados en los espacios de direcciones son:

- Espacio de direcciones 1:
 - Aplicacion2CapasVentas crea instancias de los objetos de las clases que implementan las interfaces IVistaVentas e IControladorVentas
 - El objeto de la clase ImplementacionModeloVentas que implementa la interfaz IModeloVentas, encapsula la gestión del sistema de almacenamiento para el resto de la aplicación a través de la instancia de la clase ImplementacionDAOVentas
- Espacio de direcciones 2:

- El sistema de almacenamientos de la empresa se ubica en el espacio de direcciones 2

La distribución de las clases entre dos espacios lógicos de direcciones a memoria clasifica a la construcción 4 como una arquitectura de dos capas

Ventajas del diseño en dos capas

Un diseño en dos capas tiene las siguientes ventajas:

- Es más extensible que un diseño de una sola capa, el cual combina la lógica de presentación, la lógica de negocio y los recursos de datos en un solo espacio de direcciones
- Por otro lado, un diseño de dos capas puede correr en cualquier máquina con la única restricción que esté conectada a una red donde se pueda acceder al servidor de datos (el cual conforma la capa de datos en el espacio de direcciones 2)
- Tiene menos puntos de fallas que un diseño de tres capas

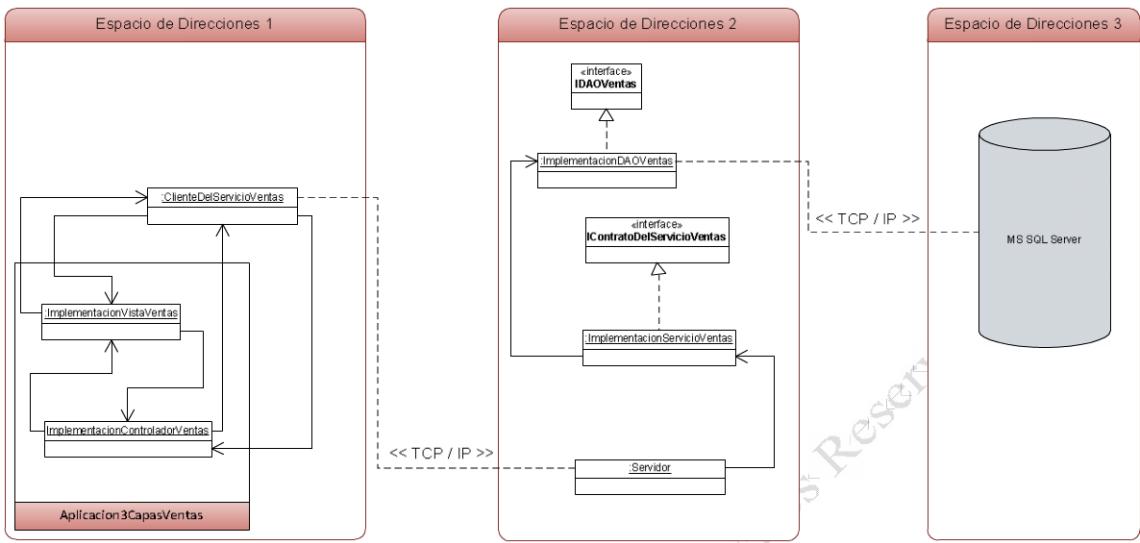
Desventajas del diseño de dos capas

Un diseño de dos capas tiene las siguientes desventajas:

- Cualquier cambio de estrategia en el negocio impactará directamente en la implementación de la lógica de negocio, la cual se encuentra en los clientes y obliga a una reconstrucción y nuevo despliegue de todos los clientes, lo cual puede ser muy costoso en tiempo y recursos.
- Cada cliente requiere manejar una conexión a la base de datos
- Este tipo de diseño restringe o complica el agregado de tecnologías en servidores como espejado (mirroring), caching, servicios de proxy o transacciones seguras con la base de datos.
- Como la lógica del negocio está en la capa cliente, todos los datos que utiliza la aplicación de la base de datos se expone a la red, con lo cual cualquiera puede accederlos.

Diseño de tres capas de Ventas

La siguiente figura muestra los objetos asociados en la construcción 5 destacando los distintos espacios de direcciones de memoria que ocupan cada uno en la aplicación



Las clases alojadas en el espacio de direcciones son:

- Espacio de direcciones 1:
 - La clase Aplicacion2CapasVentas se convierte en Aplicacion3CapasVentas y crea instancias de los objetos de las clases que implementan las interfaces IVistaVentas e IControladorVentas
 - La clase ClienteDelServicioVentas se encarga de la comunicación con la clase Servidor
 - Espacio de direcciones 2:
 - La clase servidor crea una instancia del servicio de ventas representado por la clase ImplementacionServicioVentas que implementa la interfaz IContratoDelServicioVentas para rescribir los métodos que conforman el servicio. Esta clase hace las veces del modelo del diseño de dos capas.
 - La clase ImplementacionServicioVentas a la vez crea una instancia de ImplementacionDAOVentas que se comunica con la base de datos para las operaciones de persistencia.
 - Espacio de direcciones 3:
 - El sistema de almacenamientos de la empresa se ubica en el espacio de direcciones 3

Ventajas del diseño en 3 capas

Este tipo de diseño tiene las siguientes ventajas:

- Habilita el uso eficiente de las conexiones a los recursos.
- Es posible realizar cambios en la lógica de negocio sin que esto significa volver a desplegar a los clientes
- Es arquitectónicamente mejor conformado para la escalabilidad y el balanceo de carga que otros tipos de diseño
- El escalamiento afecta primordialmente a la capa del medio solamente

Desventajas del modelo de 3 capas

Este tipo de diseño tiene las siguientes desventajas:

- Incrementa el tránsito en la red
- Tiene múltiples puntos de posibles fallas
- Los objetos de negocio tienen que ser diseñados para soportar y manejar integridad transaccional

Construcción 5

Limitación de los servidores de un solo thread

Un cliente y un servidor unidos por una conexión TCP / IP y están en una relación uno a uno. Como consecuencia, el servidor, si está disponible, siempre se encuentra esperando los requerimientos del cliente.

El problema se desencadena cuando, por ejemplo, tres clientes diferentes tratan de conectarse con el servidor. En este esquema, el primer cliente en conectarse monopoliza al servidor y nadie lo puede acceder hasta que este termine su operación con él, con lo cual, los otros dos clientes no pueden realizar ninguna operación hasta que el primero termine y se desconecte.

La solución a este problema es modificar el servidor de manera que por cada requerimiento de un cliente pueda alojar un thread para atenderlo, de manera que los requerimientos son servidos por conexiones independientes que se manejan en cada thread

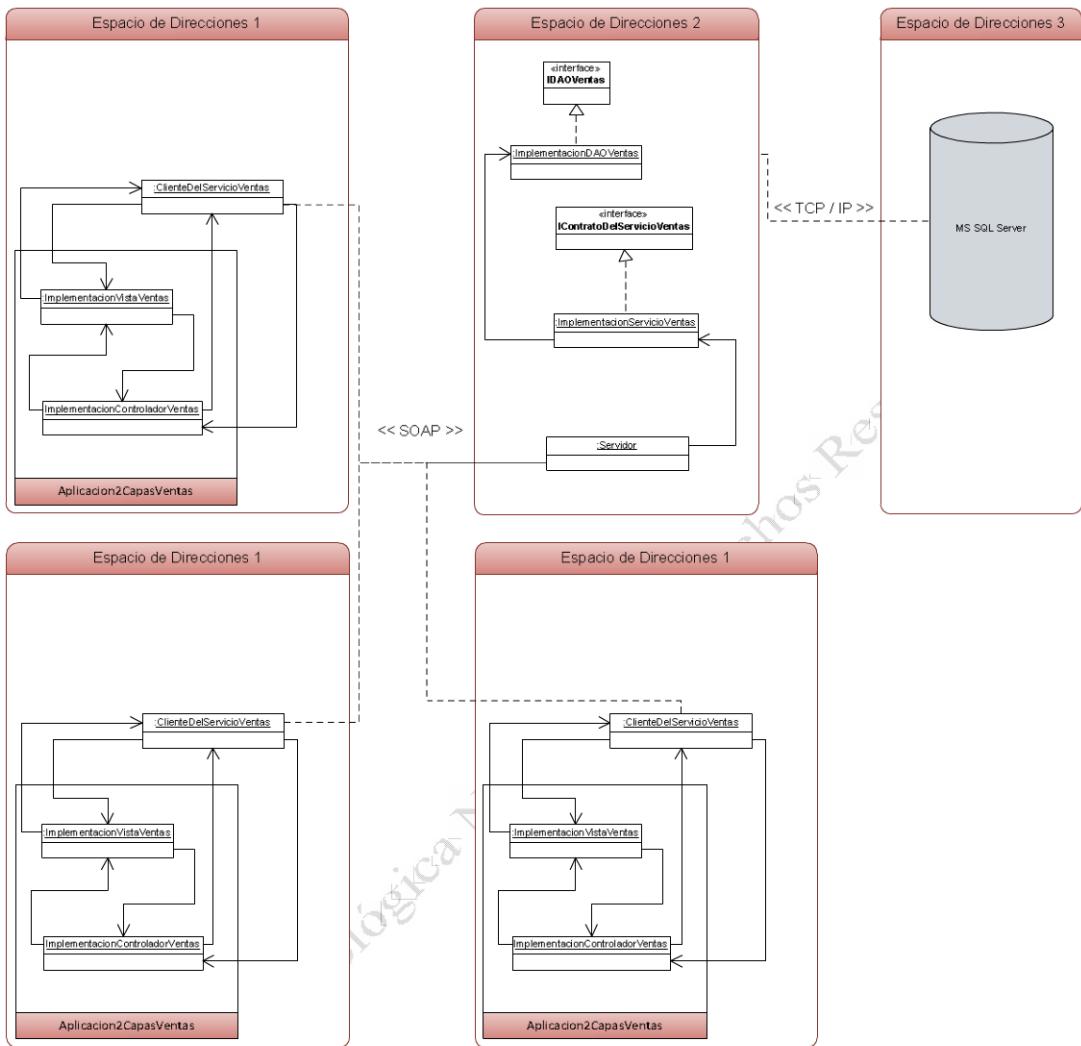
Características de un único cliente de red

En la figura que muestra el diseño en tres capas, se utiliza el protocolo de comunicación TCP / IP. Cómo estos protocolos no son de la capa de Aplicación, se deben manejar las sesiones con el cliente para el servidor no quede bloqueado cada vez que atiende un requerimiento.

La única solución para este problema es crear una reserva de threads de la cual se pueda tomar uno cuando se lo necesite, para que el sistema no tenga la sobrecarga de trabajo de crearlo cada vez que se quiera utilizar uno nuevo.

Otra posibilidad es corregir levemente la arquitectura para usar un protocolo de nivel superior como SOAP con una arquitectura orientada al servicio que maneje automáticamente las sesiones con los clientes, despreocupándose así del manejo de threads o recursos en el servidor. Esto se puede llevar a cabo mediante las tecnologías de acceso distribuido que provee Microsoft. El equipo de arquitectura debe decidir qué tipo de tecnología se debe utilizar en cada sistema cuando intervienen comunicaciones.

Una de las posibles soluciones de tecnologías remotas la provee WCF y el esquema de cómo sería un acceso de múltiples clientes se puede apreciar en la siguiente figura. De esta manera se puede obtener independencia respecto del acceso que tienen los clientes sobre el servicio y ocuparse de la lógica de negocio.



Microsoft y las tecnologías distribuidas

COM y DCOM

Microsoft desarrolló COM para que las aplicaciones puedan interactuar entre sí y promover la reutilización. COM es un conjunto de especificaciones que, cuando se sigue, permite a los componentes de software que se comuniquen entre sí. Cada componente expone su funcionalidad a través de una interfaz y se reconoce de forma única por identificadores únicos globales (GUID).

A lo largo del desarrollo del proyecto ventas se han utilizado DLLs para separar conceptualmente el desarrollo. Esta es la contrapartida moderna que ofrece .Net a COM. La diferencia fundamental

reside en la notable mejora en el manejo de interfaces para comunicarse entre sí y el uso de memoria manejada.

La ventaja de usar COM es que los diferentes componentes desarrollados en distintos lenguajes pueden escribir dichos componentes COM de software e interactuar unos con otros mediante el uso de IUnknown y otras interfaces estándar de COM. La mayoría de los productos de Microsoft, como Microsoft Office, SQL Server, e incluso de Windows, están basados en COM. Aunque COM proporciona la capacidad de reutilizar los componentes a nivel local, no fue diseñado para trabajar con componentes remotos.

Pocas especificaciones y extensiones se han hecho que se basen en COM y que interactúen con componentes remotos. Sin embargo, la necesidad de invocación de métodos remotos creció sustancialmente. Para resolver este problema, Microsoft desarrolló DCOM. En esencia, esto es una combinación de COM y un protocolo de red que le permite ejecutar un objeto COM en un equipo remoto. DCOM es un protocolo estándar para comunicación remota, propiedad de Microsoft, para extender COM de manera que pueda trabajar en entornos distribuidos. DCOM proporciona una oportunidad para distribuir componentes a través de diferentes lugares de acuerdo con los requisitos de la aplicación. Además, DCOM proporciona soporte de infraestructura básica, tales como la fiabilidad, la seguridad, la independencia de la ubicación, y la comunicación eficiente entre los objetos COM que residen a lo largo de los procesos y máquinas.

Los siguientes son los problemas de DCOM:

- DCOM y otras tecnologías distribuidas como CORBA, RMI, y así sucesivamente, se basan en varios supuestos. Una de las premisas fundamentales es que una organización se encargará de todos los componentes de los sistemas que están interactuando entre sí. Otra es que la ubicación de un componente no variará de un lugar a otro. Este escenario puede funcionar bien dentro de una organización, pero al cruzar las fronteras de la organización, las limitaciones del DCOM son más significativas.
- Microsoft ha invertido mucho en DCOM para asegurarse que llamar a un método remoto es tan sencillo como llamar al componente local mediante la simplificación de los requisitos de comunicación de red de bajo nivel. La mayoría de las veces esto dio lugar a malas prácticas de programación por los programadores, que se tradujo en un aumento del tráfico de red y cuellos de botella de rendimiento.
- DCOM, está basado en un estándar propietario, fue construido esencialmente de tomar sólo los sistemas operativos Windows en cuenta, por lo que no es adecuado para entornos heterogéneos.
- Otro problema con DCOM es que el cliente está estrechamente unido con el servidor, por lo que cualquier cambio en el cliente puede requerir una modificación en el servidor.
- DCOM, como otras tecnologías distribuidas, se basa en la arquitectura de dos niveles y sufre de algunos de los mismos defectos de la arquitectura de dos niveles.
- DCOM se presentó antes que el mundo de la informática experimente el boom de Internet. DCOM nunca se construyó con Internet en mente. Los administradores de

sistemas tienen que comprometer la seguridad del servidor que la provee con el fin de utilizar DCOM a través de firewalls y diferentes locaciones. DCOM se utiliza para comunicarse a través de los puertos que están generalmente restringidos por los firewalls porque dichos puertos son susceptibles a ataques.

.NET Remoting

Aunque COM y DCOM son capaces de proporcionar capacidad de reutilización y una plataforma distribuida, también sufren de problemas de control de versiones, recuento de referencias, etc. Microsoft .NET surgió con la visión de estar más conectados que nunca. Quería ofrecer software como un "servicio" y resolver problemas relacionados con COM. Remoting de .NET es una de las formas de crear aplicaciones distribuidas en .NET. Los desarrolladores ahora tienen opciones adicionales, como los servicios Web XML y componentes de servicio. Esencialmente, Remoting sustituye DCOM como la tecnología preferida para la construcción de aplicaciones distribuidas. Se ocupa de los problemas que han tenido las aplicaciones distribuidas durante muchos años (es decir, soporte de interoperabilidad y extensibilidad, la gestión eficiente del ciclo de vida, servidores personalizados, y un proceso de configuración fácil).

Remoting de .NET cumple con las promesas de hacer fácil la computación distribuida, proporcionando un modelo de programación simple, extensible, sin comprometer la flexibilidad, escalabilidad y robustez. Viene con una implementación predeterminada de componentes, tales como canales y protocolos, pero todos ellos son conectables y puede ser reemplazado con mejores opciones sin mucha modificación del código. Anteriormente, se utilizaron los procesos para aislar las aplicaciones unas de otras. Cada proceso tiene su propio espacio de direcciones virtuales, y el código que se ejecutaba en un proceso no podría acceder el código o los datos de otro proceso. En .NET, un proceso ahora puede ejecutar varias aplicaciones en un dominio de aplicación independiente y evitar así la comunicación entre procesos en muchos escenarios. En situaciones normales, un objeto no puede acceder a los datos fuera de su dominio de aplicación. Cualquier cosa que cruza un dominio de aplicación se le aplica Marshaling (un tipo de serialización especial que incluye referencias a memoria de manera que tengan sentido en otra locación de memoria). No sólo Remoting de .Net permite la comunicación entre dominios de aplicación, sino que también se puede extender a través de procesos, máquinas, y redes. Es flexible en los canales y formateadores que se pueden utilizar y tiene una amplia variedad de opciones para mantener el estado. A pesar que Remoting de .NET proporciona el mejor rendimiento y flexibilidad, también adolece de algunos problemas vitales.

Los siguientes son los problemas con Remoting de .Net:

- Remoting funciona mejor cuando los ensamblados que definen los tipos que se utilizan para integrar son compartidos. Remoting funciona bastante bien si existe un control total sobre los dos extremos del cable. Por lo tanto, funciona bien en una intranet en la que tiene el control completo de la implementación, control de versiones y prueba.
- Prácticamente, Remoting es propietario de .NET y funciona a la perfección para el intercambio de datos entre dos aplicaciones .NET. Está profundamente arraigado en el Common Language Runtime (CLR) y se basa en el CLR para obtener metadatos. Estos

metadatos significan que el cliente debe entender .NET con el fin de comunicarse con los criterios de valoración expuestos por Remoting.

- Remoting requiere un gran salto entre la programación de alto nivel y descender en la infraestructura. Es bastante fácil de codificar Remoting con los componentes disponibles, pero si se quiere empezar a aprender acerca de cómo agregar medios de transporte propios, el nivel de complejidad aumenta. Remoting da control más preciso de sobre cada componente arquitectónico, pero también requiere un profundo conocimiento de su arquitectura.
- .NET Remoting sufre de los problemas de equilibrio de carga, ya que no es lo suficientemente inteligente como para cambiar de una solicitud de un servidor de aplicaciones ocupado a uno que no lo está.

¿Por qué son los Servicios Web la mejor opción?

Por desgracia, con un stack de tecnología distribuida existente, se encontrará a menudo una serie de limitaciones, especialmente con la interoperabilidad entre plataformas. Por ejemplo, si se intenta implementar una aplicación COM+ para conversar a través de un servidor de seguridad o a través de routers inteligentes o límites organizacionales, a menudo se encontrará algunas diferencias significativas. La mayoría de las tecnologías de componentes distribuidos anteriormente no eran de ninguna manera construidas para hacer frente a los firewalls y routers inteligentes. Por ejemplo, si crea una aplicación utiliza Servicios de Microsoft Enterprise (un conjunto de clases que proporciona Microsoft para ser aprovechadas en aplicaciones empresariales), ¿cómo se utiliza el servicio desde un cliente Java? Teniendo en cuenta que la mayoría de las empresas están trabajando en diferentes tecnologías y diferentes plataformas, la interoperabilidad es una cuestión importante. Otras complejidades y dificultades pronto surgieron cuando estas soluciones personalizadas necesitaban ser ampliadas aún más. Los servicios Web solucionan estos problemas basándose en normas y protocolos abiertos que son ampliamente aceptados.

Los servicios Web no son más que otra forma de crear aplicaciones distribuidas. El factor distintivo de los servicios Web respecto de otras tecnologías distribuidas es que en lugar de confiar en las normas o protocolos propietarios, se basan en estándares web abiertos (como SOAP, HTTP y XML). Los estándares abiertos son ampliamente reconocidos y aceptados en la industria. Los servicios Web han cambiado la forma en la que se crean las aplicaciones distribuidas. Internet ha creado una demanda de una tecnología distribuida de bajo acoplamiento e interoperable. En concreto, antes de los servicios web, la mayoría de las tecnologías distribuidas se basaron en el paradigma orientado a objetos, pero la web ha creado la necesidad de componentes distribuidos que son autónomos e independientes de plataforma.

Los servicios Web XML están diseñados con la interoperabilidad en mente y son fáciles de llamar desde plataformas que no sean Windows. Es común confundir los servicios Web con Remoting de .NET. Los servicios Web y Remoting de .Net están relacionados, pero los primeros tienen un modelo de programación más simple. En otras palabras, ambos tienen un aspecto similar a nivel de arquitectura de alto nivel, pero difieren en su forma de trabajar. Por ejemplo, los dos tienen

diferentes formas de serializar los datos en los mensajes. Remoting admite la comunicación basada en RPC de manera predeterminada y los servicios web soportan la comunicación basada en mensajes de forma predeterminada. Los servicios Web se basan en esquemas XML para tipos de datos, y Remoting se basa en el CLR. Se puede utilizar Remoting para construir servicios Web, pero el Web Services Description Language (WSDL) generado por Remoting no está ampliamente adoptado y puede ser ignorado por algunos clientes. Aunque se puede utilizar también para la creación de componentes, es adecuado cuando se lo utiliza por una aplicación que se ejecuta en un entorno privado (por ejemplo, una intranet), en cambio los servicios web y XML crean componentes que pueden ser accesibles desde cualquier aplicación conectada a través de Internet. A través de los servicios web, Microsoft quiere conseguir lo mejor de ambos mundos: el desarrollo Web y el desarrollo basado en componentes. Los servicios Web son el primer paso hacia la orientación al servicio, que es un conjunto de principios rectores para el desarrollo de aplicaciones distribuidas débilmente acopladas. SOA es una visión de los servicios que tienen interfaces bien definidas. Estas interfaces débilmente acopladas se comunican a través de mensajes descritos por la definición de esquemas XML (XSD) y a través de los patrones de mensajes descritos por WSDL. Esto proporciona una gran arquitectura de base para la construcción de aplicaciones distribuidas. Dado que un servicio de web y sus clientes son independientes el uno del otro, sólo tienen que adherirse únicamente a los estándares de documentos WSDL y XSD con el fin de comunicarse.

La siguiente oferta de Microsoft para hacer frente a SOA es WCF. Ahora vamos a discutir la forma de servicios web WCF complementa y realza su valor.

¿Qué resuelve WCF?

WCF no es más que otra forma de crear una solución distribuida, pero ofrece una serie de ventajas con respecto a sus predecesores. Si se miran los antecedentes de WCF, se encontrará que el trabajo en WCF comenzó con el lanzamiento de .NET. En otras palabras, se ha tardado años en construirlo para llegar al mercado. WCF aborda muchas cuestiones y los siguientes son los tres objetivos de diseño principales de WCF:

- Unificación de las tecnologías existentes
- Interoperabilidad entre plataformas
- La unificación de la tecnologías distribuidas existentes

Unificación de las tecnologías existentes

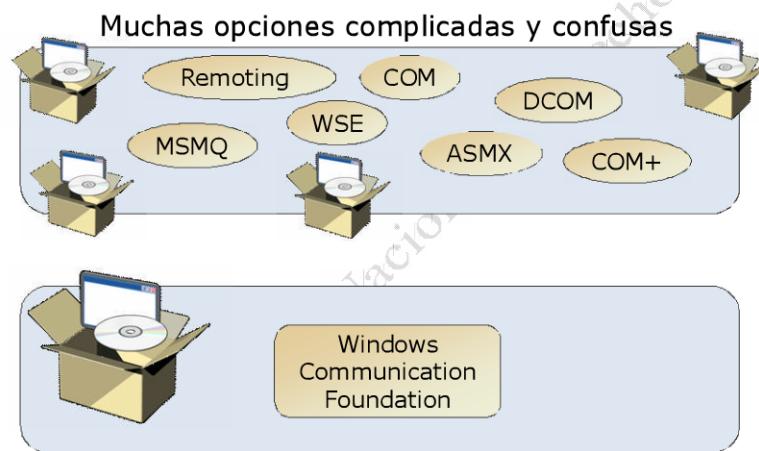
El mundo actual de la informática empresarial tiene muchas tecnologías distribuidas, cada una de las cuales realiza una tarea específica. Aparte de eso, estas tecnologías distribuidas se basan en diferentes modelos de programación (por ejemplo, si se está construyendo una aplicación que se comunica a través de HTTP, se requerirá cambiar el modelo de programación si se quiere pasar a TCP). Esto ha creado problemas para los desarrolladores, que tienen que seguir aprendiendo diferentes APIs para diferentes formas de construcción de componentes distribuidos.

Una de las cuestiones interesantes es, ¿por qué no tener una sola tecnología que se puede utilizar en todas las situaciones? WCF es la solución de Microsoft para el desarrollo de aplicaciones

distribuidas de aplicaciones empresariales. Se evita la confusión, tomando todas las capacidades de la tecnología de las pilas de los sistemas distribuidos existentes y le permite utilizar una API limpia y simple. Todo lo que se tiene que hacer como desarrollador es hacer referencia al ensamblado System.ServiceModel e importar el espacio de nombres.

WCF es un conjunto de bibliotecas de clases que viene a partir del Framework de .NET 3.0, el cual se convirtió en un API de núcleo en el sistema operativo Windows Vista. También se puede instalar en un equipo a partir de Windows XP (Service Pack 2) y Windows Server 2003.

WCF incluye lo mejor de todas las tecnologías distribuidas, reuniendo la eficiencia de ASMX, la adopción de transacciones con Enterprise Services sólo mediante el uso de atributos, la extensibilidad y flexibilidad de Remoting, MSMQ para crear aplicaciones en cola y la interoperabilidad de WSE través de WS-*. Microsoft tomó todas estas capacidades y construyó una única infraestructura estable en la forma de WCF.



WCF provee un modelo de programación unificado que ofrece toda la funcionalidad previamente existente en los ambientes de programación distribuida de Microsoft, desarrollo de componentes y servicios web como único entorno de programación, incluyendo las siguientes tecnologías:

- ASMX
- Component Object Model (COM)
- COM+
- Distributed Component Object Model (DCOM)
- Message Queuing (MSMQ)
- Remoting
- Web Service Enhancements (WSE)

WCF fue construido desde cero para incluir toda la funcionalidad precedente, de manera que los desarrolladores no tienen que preocuparse más por las diferentes tecnologías para generar

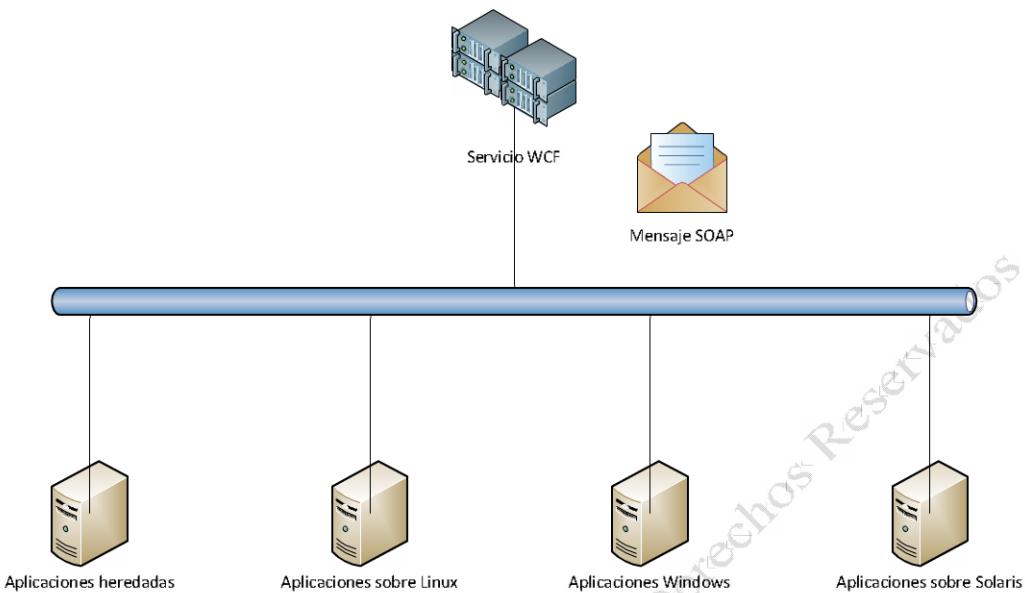
programación distribuida. Sin embargo todavía se debe entender la cómo funciona cada mecanismo individual de comunicación para asegurarse que se usará configuración correcta.

Interoperabilidad entre plataformas

La interoperabilidad ha sido un tema importante para todos los principales proveedores de software, que querían utilizar un conjunto de protocolos que fuera ampliamente aceptado y adoptado. Por lo tanto, los líderes de la industria como Microsoft, IBM, BEA y Sun formaron la organización de interoperabilidad de servicios Web (WS-I - Web Services Interoperability), que ha desarrollado un conjunto de especificaciones que permite que un software se comunique sin problemas con otro que funciona en una plataforma diferente.

Una de las grandes características de las especificaciones WS-I es que es simple, pequeña, modular y fácil de implementar. Se es libre de elegir qué especificación se necesita poner en práctica. Por ejemplo, la implementación de WS-Security no obliga a que se implemente las especificaciones de transacciones. Se divide en varias capas (por ejemplo, hay una especificación para el envío de una firma digital en un mensaje SOAP y una especificación diferente para el envío de un simple nombre de usuario y contraseña en SOAP). La arquitectura fundamental de una especificación de servicio web para todo esto es WSDL. Por lo tanto, WCF habla el lenguaje de la última serie de protocolos de servicios web para lograr interoperabilidad entre plataformas.

La siguiente figura muestra que el protocolo de mensajería nativo WCF es SOAP que como un estándar abierto proporciona la oportunidad que el servicio WCF interactúe con diferentes tecnologías que se ejecutan en distintas plataformas y sistemas operativos que no sean Windows. Dado que los servicios se basan en estándares abiertos, otras aplicaciones pueden utilizarlos sin necesidad que estos clientes posean un conocimiento detallado de cómo fueron implementados los servicios.



No sólo puede WCF interactuar con sus homólogos de otros proveedores, sino que también puede existir en paz con sus predecesores, tales como COM + y Enterprise Services. Para los desarrolladores, esto reduce drásticamente la cantidad de código de infraestructura necesaria para conseguir una interoperabilidad heterogénea.

WCF como herramienta de desarrollo orientada a servicios

WCF es el primer modelo de programación construido desde cero para proporcionar el desarrollo de aplicaciones orientadas a servicios explícitas. La orientación a servicios no es una tecnología, sino que es un concepto de diseño. La orientación al servicio utiliza las mejores prácticas para la creación de aplicaciones distribuidas de hoy.

Aunque pueda parecer sorprendente, una de las partes más interesantes del diseño de un servicio es decidir cómo se debe exponer su funcionalidad con el mundo exterior. El nivel de granularidad del servicio a menudo es uno de los temas más candentes del debate en el seno de una organización. Si el servicio es "grano fino", entonces la atención se centra generalmente en el intercambio de pequeñas cantidades de datos para realizar una tarea específica. Esto se asocia generalmente con el tipo más tradicional estilo de comunicación RPC. Las tareas adicionales, si es necesario, se invocan de manera similar. Dado que las invocaciones de servicios basados en mensajes son caras respecto de los recursos, los enfoques de granularidad fina podrían no ser prácticos en la mayoría de las situaciones, porque la sobrecarga de la transmisión y el procesamiento de muchos mensajes individuales no serían aceptable. Por otra parte, los servicios de granularidad gruesa exponen más funcionalidad dentro de la misma invocación de servicio mediante la combinación de muchas tareas pequeñas (existe un patrón de diseño que se ajusta perfectamente a esta técnica llamado "fachada"). Esto se relaciona con un menor número de mensajes transmitidos con más datos, en contraposición a muchos mensajes con menos datos.

Esto se refiere también a menos sobrecarga en ambos extremos del servicio, lo que permite un servicio de granularidad gruesa escalar mejor.

En el diseño de los servicios, es necesario extenderse más allá de los principios básicos de diseño orientado a objetos y el uso de los cuatro principios de la orientación a servicios descritos posteriormente como principios rectores. Uno de los retos en el desarrollo de WCF está cambiando la mentalidad de los desarrolladores para pensar en la construcción de sistemas distribuidos en términos de sistemas distribuidos usando servicios. WCF proporciona esta plataforma para crear la próxima generación de aplicaciones distribuidas.

¿Qué es la arquitectura orientada a servicios?

No es práctico para construir sistemas monolíticos. Estos sistemas suelen tomar muchos años para ponerlos en práctica y se dirigen normalmente a un conjunto limitado de objetivos. Hoy en día un negocio tiene que ser ágil y adaptar los procesos rápidamente, y SOA es un principio de diseño que pueden ayudar a hacer frente a esta necesidad empresarial. SOA es un conjunto de servicios bien definidos, donde cada servicio individual se puede modificar de forma independiente de otros servicios para ayudar a responder a la constante evolución de un negocio. Una implementación de SOA comprende uno o más conjunto de servicios de aplicación imprecisa e interoperables. Aunque algunos de estos aspectos puede ser similar a un desarrollo basado en componentes (que se basa en las interfaces estrictas), la diferencia clave es SOA proporciona un enfoque basado en mensajes sobre la base de estándares abiertos.

Normalmente se asocia SOA con servicios de Internet pero este no tiene por qué ser necesariamente el caso. SOA se puede utilizar correctamente como servicios de intranet. Se centrará el foco de la explicación en el desarrollo para intranet sin perder de vista los conceptos esenciales para entender el tema.

Como resultado de estar basado en estándares abiertos y el uso de mensajes que son genéricos y no representativos de cualquier plataforma o lenguaje de programación específico, se puede lograr un alto grado de acoplamiento débil y de interoperabilidad entre plataformas y tecnologías. Cada uno de estos servicios es autónomo y proporciona uno o más conjuntos de funciones de la empresa y, además, dado que los detalles de la implementación se ocultan al usuario, cualquier cambio en la aplicación no afectará el servicio, siempre y cuando el contrato no cambie. Esto permite a los sistemas basados en SOA responder de una manera efectiva más rápida y para el negocio.

Para una empresa que por lo general es más barato "consumir" un servicio que se pueda compartir en las aplicaciones, que desarrollar para cada caso lo que constituye la solución y escribir toda la funcionalidad. Si necesita que un módulo específico se actualice por alguna razón, también es benéfico que los cambios estén confinados al servicio específico.

Los servicios encapsulan los procesos de negocio en módulos de software independientes a entregar. Un servicio sólo es un bloque de construcción, no es una solución de negocio, sino que en un sistema de negocio autónomo es capaz de recibir peticiones y cuya interoperabilidad se rige

por distintas normas de la industria. Estos componentes también proporcionan la base para el aumento de las mejoras en la calidad, fiabilidad y en la disminución de los costos a largo plazo para el desarrollo y mantenimiento de software.

Para los nuevos en SOA, que es un poco difícil de entender este concepto inicialmente. Esto es, principalmente, debido a que las implementaciones destino del desarrollo para SOA son sistemas back-end (servidores y servicios que se acceden remotamente sin interacción directa del usuario). En consecuencia, desde la perspectiva de un usuario, hay pocos cambios en la interfaz gráfica (UI). Sin embargo, también se puede utilizar SOA para proporcionar implementaciones de interfaz de usuario o front-end. Se puede lograr esto mediante la combinación de la producción de servicios XML con XSL para producir HTML para el destino. Sin embargo, no es el caso del que nos ocuparemos ahora.

El paradigma SOA se aparta significativamente del modelo OO, donde se le anima a encapsular datos. Por lo tanto, un objeto va a almacenar y proteger los datos para facilitar una necesidad de negocio. Un sistema empresarial estará formado por varios objetos que están especializados para tratar "casos específicos" con los datos protegidos dentro de los objetos. SOA implica utilizar los servicios débilmente acoplados. El servicio se describe en el contrato para las entidades que lo consumen.

Sin embargo existe una solución orientada a objetos para este problema. Hay un patrón de diseño ha llamado fachada que consta de un objeto que tiene una serie de servicios que sirven de frente al toda la complejidad del sistema que se encuentra detrás de él. Este objeto permite manejar como siempre llamados a servicios funcionalidades completas de negocio que se esconden detrás de cada uno de ellos. Por lo tanto un buen diseño de un servicio WCF tendrá en su contrato una serie de métodos que cumplirán con el patrón de fachada, de manera que se pueda manejar cualquier paradigma de la programación orientada a objetos detrás de él.

Una arquitectura orientada al servicio es una filosofía de diseño independiente de las tecnologías puede ayudar en el desarrollo de servicios poderosos orientados al negocio que los clientes pueden acceder sin un conocimiento acerca de la implementación del mismo.

Beneficios de la arquitectura orientada a servicios

El objetivo primario de SOA es poder entregar más en beneficio del negocio rápidamente y permitirle al negocio responder al cambio en el requerimiento de desarrollar nuevas aplicaciones desde cero.

Sin embargo para utilizar satisfactoriamente SOA se debe tener en cuenta lo siguiente:

- No se deben realizar suposiciones acerca de cómo el servicio procesa un requerimiento y como el cliente procesa una respuesta. Los servicios y los clientes se comunican sólo cuando es necesario.
- El deben diseñar soluciones tengan acoplamiento débil y que pueda tolerar los cambios. Por ejemplo, si se utiliza un servicio web de una tercera parte en una solución, asegurarse

que ésta se puede ejecutar aún si el servicio web de la tercera parte no se encuentra disponible.

- Deben diseñar esquemas y contratos que definan la funcionalidad que el servicio expone. Esto se debe a que se pueden agregar contratos adicionales para proveer mayor nivel de funcionalidad, mientras que se implemente el contrato original, estarán capacitados de acceder a la funcionalidad original.
- Se debe mantener por separado los requerimientos no funcionales, como por ejemplo las restricciones de seguridad, de los requerimientos funcionales como la implementación del servicio

Diseño de una aplicación SOA

Los pasos de alto nivel que se utilizan para diseñar una aplicación SOA son:

- Obtener compromiso de los involucrados en el diseño debido al gran esfuerzo involucrado en la creación de la arquitectura.
- Determinar los servicios a partir de
 - Recolecta servicios de proyectos que se han desarrollado sin tener en cuenta los servicios de manera independiente
 - Construir interfaces de servicios para funcionalidad existente
 - Realizar nuevos proyectos para construir servicios a medida que se avanza

Agrupar las funciones de negocios relacionadas teniendo en cuenta la granularidad. No se debe crear una capa de acceso a datos distribuida solamente, sino que se debe desarrollar de manera que el cliente solicite servicios sin estar fuertemente enlazado a estos. La comunicación con el servicio no debe condicionar el acceso del cliente ni crear dependencias.

Los servicios pueden ser descubiertos como parte integral del diseño y arquitectura de la aplicación.

Restricciones de diseño

Cuando se diseña, mantener los métodos de negocio como funciones con un alto nivel de granularidad en lugar de ser detalladas. Cada método de negocio realiza una función completa y auto contenida que es independiente y no requiere llamados subsecuentes a otros métodos en una manera específica. Esto determina que una tarea no se realiza como una serie de llamados a servicios. Un servicio debe poder ejecutarse en su totalidad o no.

Servicios, componentes y objetos

Los servicios son procesos independientes definidos por un interfaz. Los servicios tienen un grado de granularidad mayor a la de los objetos o los componentes. Los servicios manejan sus propios recursos y también tienen una vida útil y ciclo de vida que es independiente de las aplicaciones que lo utilizan.

Los componentes son bloques de construcción a nivel de la plataforma creados para una en particular. Los componentes también son definidas por su interfaz, pero ésta depende de la plataforma.

Los objetos son bloques de construcción a nivel de lenguaje que se encuentran por lo general dentro de los componentes y los servicios. Los objetos representan conceptos de granularidad fina como ser entidades de datos o reglas de negocios.

Se pueden enumerar las diferencias entre servicios componentes y objetos de la siguiente manera:

- Los servicios son procesos independientes definidos por su interfaz
- Los componentes son bloques de construcción a nivel de plataforma como por ejemplo las dll a las que se puede referenciar en tiempo de diseño y acceder en tiempo de ejecución.
- Los objetos son bloques de construcción a bajo nivel dentro de los componentes y servicios

WCF y SOA

WFC soporta SOA proveyendo una manera fácil exponer a nivel del lenguaje contratos que son independientes de la plataforma.

La mayoría de los desarrolladores no tienen que aprender tecnologías como el lenguaje de descripción de servicios web (Web Service Description Language - WSDL) porque en tiempo de ejecución se maneja la mayoría de la funcionalidad. Las herramientas como el Visual Studio permiten una fácil creación y consumo de servicios WCF. Sin embargo, sólo porque se pone una interfaz sobre una pila de protocolos de un servicio web significa que se ha creado SOA.

WCF juega un papel fundamental en el mecanismo de distribución y es la tecnología este se debe utilizar para implementar SOA, pero sólo por utilizar WCF no significa de la solución este orientada al servicio.

Cuando se diseña o implementa un sistema se tienen objetos que representan datos de negocio, código para almacenar y manejar dichos datos y reglas de negocio que determinan cómo interactúan entre sí los mencionados objetos.

De manera alternativa, a nivel de servicio, se tiene una fachada o un controlador de negocio que referencia operaciones más granulares. Para la interfaz de negocios, la fachada provee acceso a las operaciones por completo como por ejemplo registrar un cliente, la cual en realidad conecta a operaciones mucho más granulares como puede ser realizar una verificación del crédito u obtener detalles del cliente. A nivel de servicio las operaciones del negocio sólo interactúan con las de alto nivel expuesto por las fachadas y no con métodos individuales a nivel de lenguaje.

Las desventajas de la integración de múltiples aplicaciones en redes dispares

Ahora se va a discutir algunos de los retos que se enfrentará al intentar integrar múltiples aplicaciones en la actualidad. Los siguientes son algunos de los retos fundamentales en la integración de múltiples aplicaciones que residen en redes físicas diferentes:

- Transportes: Las redes no son confiables y pueden ser lentas.

- Formato de los datos: Los dos solicitudes en cuestión se pueden ejecutar en distintas plataformas y usando diferentes lenguajes de programación, lo que hace de la interfaz con los distintos tipos de datos un reto interesante.
- Cambio: se sabe que las aplicaciones necesitan cambiar para mantenerse al día con los requisitos empresariales en constante evolución. Esto significa que cualquier solución de integración tendría que asegurarse de que podía seguir el ritmo de este cambio y minimizar las dependencias entre los sistemas.

En el pasado, los desarrolladores utilizaban varios enfoques para tratar de integrar aplicaciones en una empresa. Estos enfoques incluyen la transferencia de archivos, bases de datos compartidas, llamadas a procedimiento remoto (RPC) y mensajería. Aunque cada uno de estos enfoques podría tener sentido en algún contexto, los mensajes suelen ser más beneficiosos.

Ventajas del uso de mensajería

Las siguientes son las ventajas de utilizar mensajes, al igual que se hace en SOA:

- La integración entre plataformas: los mensajes pueden ser una especie de "traductor universal" entre distintas plataformas y lenguajes, permitiendo que cada plataforma trabaje con sus respectivos tipos de datos nativos.
- Comunicaciones asíncronas: Los mensajes suelen permitir un estilo "enviar y olvidarse" de comunicación. Esto también permite, en caso de que se lo desee, la sincronización variable ya que tanto el emisor como el receptor pueden estar corriendo a toda máquina y no ser limitados por esperar el uno del otro.
- Comunicación fiable: Mensajes inherentemente utilizan un estilo "almacenar y enviar" (store-and-forward) para la entrega, lo que les permite ser más fiable que RPC.
- Mediación: los mensajes pueden actuar como mediador cuando se utiliza el patrón de diseño mediador (Mediator) en el que una aplicación que trabaja desconectada necesita hacer un comentario sólo al sistema de mensajería y no a todas las otras aplicaciones.
- Gestión de threads: Dado que los mensajes permiten la comunicación asíncrona, esto significa que una aplicación no tiene por qué bloquear a otra hasta que finalice. Dado que esto libera muchos threads, la aplicación puede hacer otro trabajo, por lo que es más eficiente y flexible en el manejo de threads propios.
- Comunicación remota: Los mensajes reemplazan la necesidad de la serialización y deserialización que se produce cuando una aplicación realiza una llamada remota a otra aplicación. Por lo general, ya que estos pueden estar ejecutándose en diferentes procesos o incluso máquinas, las llamadas tienen pasar sus llamados a través de la red, lo que incluye parámetros y valores retornados. El proceso de la serialización de un objeto para transferir por una red se denomina Marshaling. Del mismo modo, el proceso de deserializar un objeto en el otro extremo se denomina Unmarshaling (resolución de referencias).
- Seguridad de extremo a extremo: Al contrario que en el mundo de la RPC, los mensajes pueden transferir el "contexto de seguridad completa" a los consumidores mediante una combinación de encabezados y cadenas (tokens). Esto aumenta en gran medida la

capacidad de proporcionar un control más granular incluyendo la autenticación y la autorización.

Los mensajes son los "pilares" de SOA. Estos permiten crear sistemas débilmente acoplados que pueden abarcar varios sistemas operativos. SOA se basa en mensajes no sólo para facilitar las necesidades del negocio, sino también para proporcionar el "contexto" de todo el mensaje (es decir, el contexto de seguridad, la información de enrutamiento del mensaje, si es necesario garantizar la entrega del mensaje y demás).

Entendiendo la Arquitectura Orientada a Servicios

SOA y los servicios web son las palabras de moda que prometen resolver todos los problemas de integración en el ámbito de la empresa. Aunque cualquier tipo de aplicación puede ser una aplicación SOA, por desgracia muchas implementaciones que utilizan los servicios web son comercializadas como implementaciones de SOA, cuando en realidad no lo son.

SOA puede ser definida simplemente como un concepto arquitectónico o el estilo que utiliza un conjunto de "servicios" para lograr la funcionalidad deseada. Un servicio es un sistema autónomo (de negocios) que acepta una o más solicitudes y devuelve una o varias respuestas a través de un conjunto de interfaces publicadas y bien definidas. A diferencia de las arquitecturas fuertemente acopladas tradicionales, SOA implementa un conjunto de servicios débilmente acoplados que en conjunto alcanzan los resultados deseados.

Nota: Es importante entender que aunque SOA puede parecer abstracto. SOA es más que una colección de servicios. Es una metodología que abarca las políticas, procedimientos y mejores prácticas que permitan a los servicios que se prestan y consuman efectivamente. SOA no es un "producto" que se puede comprar fuera de la plataforma, sin embargo, muchos vendedores tienen productos que pueden formar la base de una implementación de SOA.

Es importante que los servicios no queden reducidos a un conjunto de interfaces, ya que son la clave de comunicación entre el proveedor y el consumidor. Un proveedor es la entidad que presta el servicio, y el consumidor es la entidad que consume el servicio. En un mundo cliente-servidor tradicional, el proveedor será un servidor, y el consumidor será un cliente. Cuando se producen los servicios, se trata de modelar el flujo y proceso de negocios basado en los eventos del mismo reconocidos y los procesos existentes. También es necesario responder algunas preguntas para asegurar un diseño limpio de los servicios:

- ¿Qué servicios se necesitan?
- ¿Qué servicios están disponibles para consumir?
- ¿Qué servicios funcionarán juntos?
- ¿Qué servicios sustitutivos están disponibles?
- ¿Qué dependencias existen entre los servicios y otras versiones de los servicios?

El objetivo primario de SOA poder entregar más en beneficio de negocio rápidamente y permitirle al negocio responder al cambio en el requerimiento de desarrollar nuevas aplicaciones desde cero.

Sin embargo para utilizar satisfactoriamente SOA se debe tener en cuenta lo siguiente:

- No se deben realizar suposiciones acerca de cómo el servicio procesa un requerimiento y como el cliente procesa una respuesta. Los servicios y los clientes se comunican sólo cuando es necesario.
- El deben diseñar soluciones tengan acoplamiento débil y que pueda tolerar los cambios. Por ejemplo, si se utiliza un servicio web de una tercera parte en una solución, asegurarse que ésta se puede ejecutar aún si el servicio web de la tercera parte no se encuentra disponible.
- Deben diseñar esquemas y contratos que definan la funcionalidad que el servicio expone. Esto se debe a que se pueden agregar contratos adicionales para proveer mayor nivel de funcionalidad, mientras que se implemente el contrato original, estarán capacitados de acceder a la funcionalidad original.
- Se debe mantener por separado los requerimientos no funcionales, como por ejemplo las restricciones de seguridad, de los requerimientos funcionales como la implementación del servicio

Diseño de una aplicación SOA

Los pasos de alto nivel que se utilizan para diseñar una aplicación SOA son:

- Obtener compromiso de los involucrados en el diseño debido al gran esfuerzo involucrado en la creación de la arquitectura.
- Determinar los servicios a partir de
 - Recolecta servicios de proyectos que se han desarrollado sin tener en cuenta los servicios de manera independiente
 - Construir interfaces de servicios para funcionalidad existente
 - Realizar nuevos proyectos para construir servicios a medida que se avanza

Agrupar las funciones de negocios relacionadas teniendo en cuenta la granularidad. No se debe crear una capa de acceso a datos distribuida solamente, sino que se debe desarrollar de manera que el cliente solicite servicios sin estar fuertemente enlazado a estos. La comunicación con el servicio no debe condicionar el acceso del cliente ni crear dependencias.

Los servicios pueden ser descubiertos como parte integral del diseño y arquitectura de la aplicación.

Restricciones de diseño

Cuando se diseña, mantener los métodos de negocio como funciones con un alto nivel de granularidad en lugar de ser detalladas. Cada método de negocio realiza una función completa y auto contenida que es independiente y no requiere llamados subsecuentes a otros métodos en una manera específica. Esto determina que una tarea no se realiza como una serie de llamados a servicios. Un servicio debe poder ejecutarse en su totalidad o no.

¿Qué es un servicio?

Los servicios como término se han utilizado para describir todo, desde servicios web para los procesos de negocio y como para todo lo demás que haga algo. Se deben utilizar los servicios para representar las funciones del negocio y definir explícitamente los límites de lo que hace la

empresa, que en esencia sería definir lo que el servicio puede o no puede hacer. La clave es que no es un enfoque basado en la tecnología, sino, más bien, es un enfoque orientado a los negocios.

A diferencia de los servicios que ofrecen los objetos, a los servicios de SOA modelan una realidad de negocio mientras que los otros determinan la funcionalidad que puede realizar un determinado objeto.

La orientación a servicios es una "estrategia de modelado" impulsada por las empresas, que define la funcionalidad del negocio en términos de sistemas de negocio autónomos débilmente acoplados (o servicios) para el intercambio de información basada en mensajes.

Los servicios son procesos independientes definidos por un interfaz. E tienen un grado de granularidad mayor a la de los objetos o los componentes. Los servicios manejan sus propios recursos y también tienen una vida útil y ciclo de vida que es independiente de las aplicaciones que lo utilizan.

Los componentes son bloques de construcción a nivel de la plataforma creados para una en particular. Los componentes también son definidas por su interfaz, pero ésta depende de la plataforma.

Los objetos son bloques de construcción a nivel de lenguaje que se encuentran por lo general dentro de los componentes y los servicios. Los objetos representan conceptos de granularidad fina como ser entidades de datos o reglas de negocios.

Se pueden enumerar las diferencias entre servicios componentes y objetos de la siguiente manera:

- Los servicios son procesos independientes definidos por su interfaz
- Los componentes son bloque de construcción a nivel de plataforma como por ejemplo las DLL a las que se puede referenciar en tiempo de diseño y acceder en tiempo de ejecución.
- Los objetos son bloque de construcción a bajo nivel dentro de los componentes y servicios

Desarrollo orientado a servicios con WCF

El desarrollo orientado a servicios con WCF consta de dos pasos.

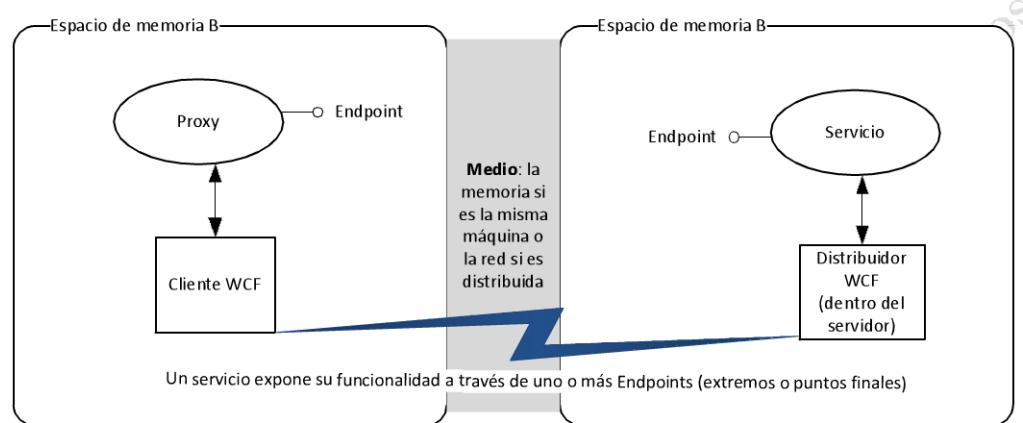
1. Crear un servicio que expone un conjunto de funcionalidades relacionadas con más de un protocolo de red.
2. Crear los clientes que puedan invocar cualquiera de los métodos expuestos por el servicio de acceso a la funcionalidad subyacente. Las aplicaciones cliente se comunican mediante el uso de mensajes de solicitud y, si la funcionalidad lo requiere, recibiendo los mensajes de respuesta.

Es importante recordar que con WCF, se puede comunicar a través de diferentes protocolos como HTTP y TCP.

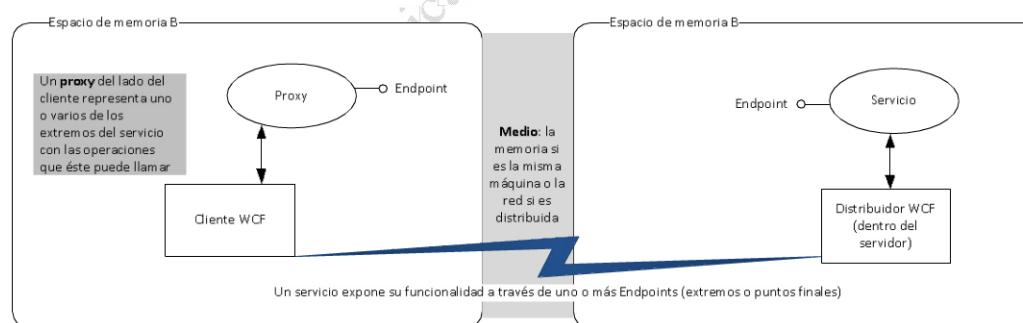
El desarrollo orientado a servicios está dirigido a clientes y servicios que no comparten el mismo espacio de direcciones en memoria. Sin embargo, el cliente y el servicio, de hecho, se pueden ejecutar en el mismo equipo y no necesariamente distribuidos en una red.

Enviando un mensaje de WCF

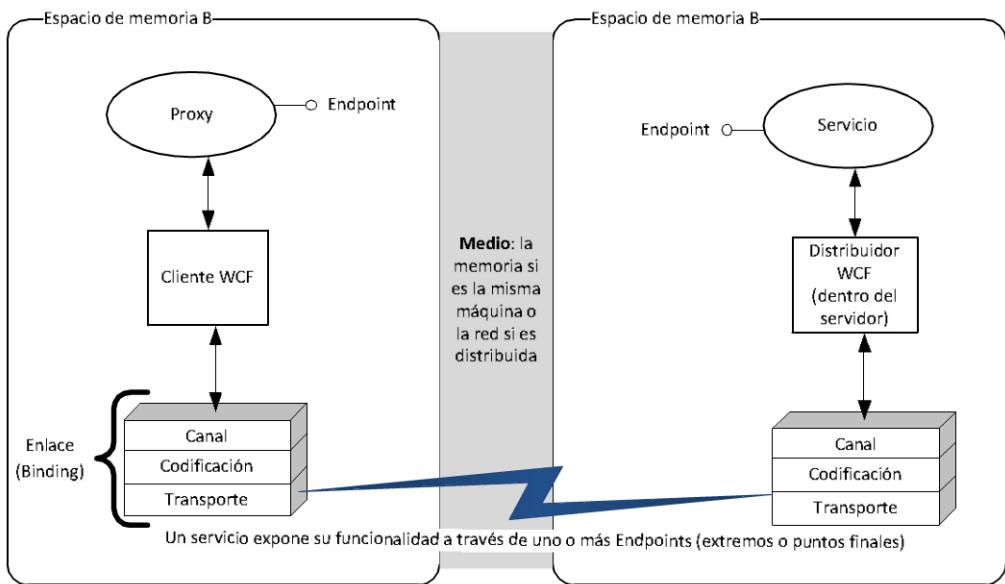
Un servicio WCF expone varios puntos finales (endpoints), que son los medios de acceso que proporcionan una forma para que otras aplicaciones se comuniquen con el servicio.



El cliente se puede comunicar con el servicio a través de un proxy que genera WCF. Este proxy abstrae las complejidades de la comunicación de manera que el cliente maneje el o los llamados a métodos fuera de su espacio de memoria como si estuvieran en el mismo.



Cada extremo de servicio contiene una dirección, un enlace y un contrato, lo que proporciona al cliente la información necesaria para comunicarse correctamente con el servicio.



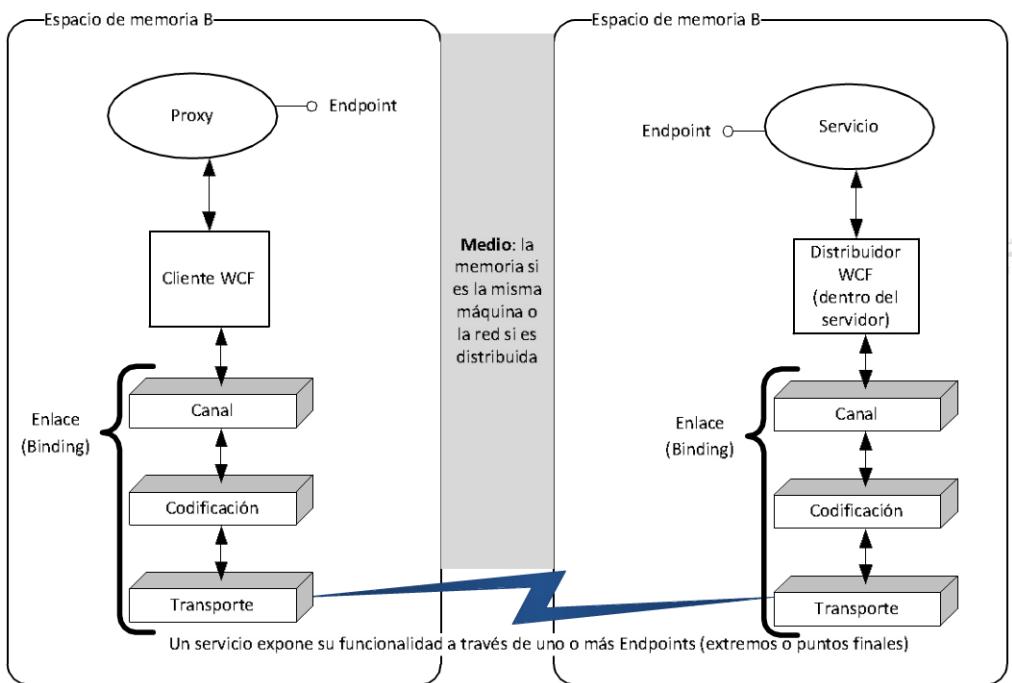
WCF provee un canal de transporte para la comunicación, codificación para dar formato a los datos y uno o más canales de alto nivel que controlan tanto seguridad como transacciones en ambos extremos de la comunicación. A este conjunto se lo denomina **enlace**.

El papel del Dispatcher (distribuidor) de WCF es determinar para qué operación es el mensaje e invocar el código de dicha operación en el servicio de WCF. Hay una pila de canales entre el distribuidor (Dispatcher) WCF y el entorno externo. Esta pila contiene los siguientes componentes, cada uno con un rol específico en el proceso de mensajería.

- Protocolo de canales (como la seguridad o el control de la transacción)
- Codificador (encoder) del mensaje
- Canal de transporte

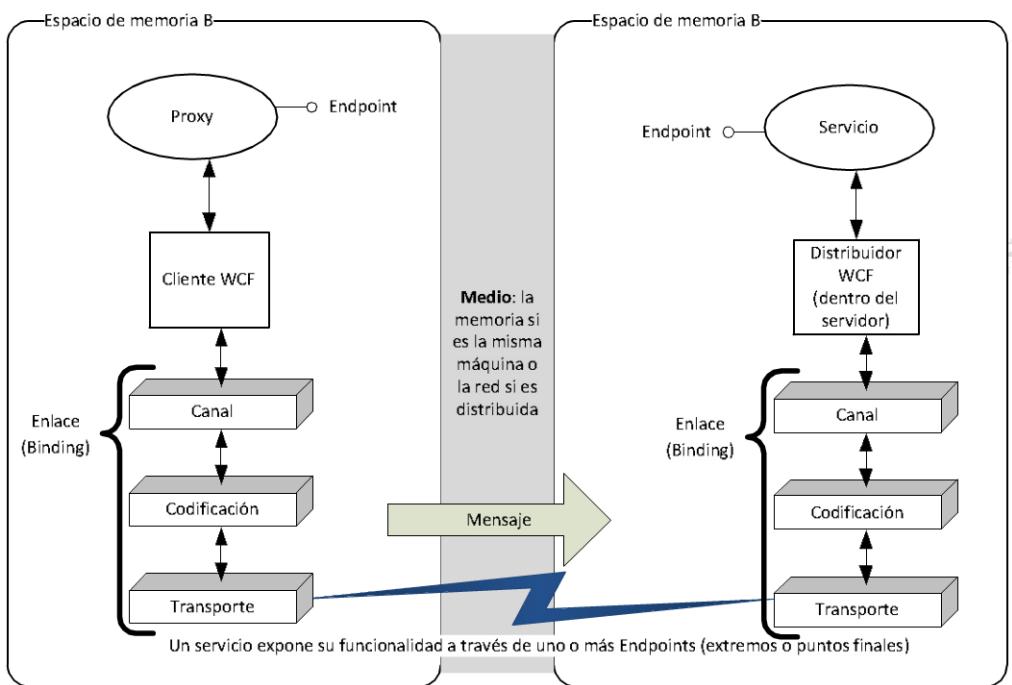
La pila de canales implementa la política (incluida la seguridad, la confiable de los mensajes y las transacciones) necesaria para comunicarse con el servicio. Es importante ordenar los canales de acuerdo a la función que realizan, por ejemplo, para un enlace dado, en la parte superior, se tendría el flujo de transacciones, seguido de la codificación, y luego el canal de transporte.

El cliente utiliza el objeto de proxy para invocar las operaciones expuestas por el servicio de WCF. El proxy oculta la complejidad del proceso de comunicación con un servicio remoto. El cliente puede llamar a las operaciones expuestas por el servicio de la misma manera que en el cliente se invocan los métodos expuestos por un objeto local.



El cliente envía un mensaje al servicio que quiere acceder por medio del proxy que maneja el mismo tipo de enlace que existe en el servidor

A través de los enlaces, el proxy genera la comunicación con el distribuidor de WCF de manera transparente. Así, una invocación es transmitida como un mensaje al extremo del servicio por medio del enlace que maneja el distribuidor



El cliente encamina el mensaje al servicio por medio del enlace para enviar un mensaje. Este es recibido por el enlace del lado del servidor que lo encamina a través del distribuidor hacia el servicio.

Finalmente, el distribuidor es quien decide la operación e instancia del servicio que atenderá al mensaje enviado por el cliente a través del proxy. Una vez resuelta la invocación, el proceso se completa realizando el camino inverso.

¿Son iguales los componentes a los servicios?

Es natural estar confundido respecto de los componentes y servicios acerca de los términos y su significado. Un componente es una pieza de código compilado que se puede montar con otros componentes para crear aplicaciones. Los componentes pueden también ser fácilmente reutilizados dentro de la misma aplicación o entre aplicaciones distintas. Esto ayuda a reducir el costo de desarrollo y mantenimiento de la aplicación una vez que los componentes maduran dentro de una organización. Los componentes se asocian generalmente con el paradigma de programación orientada a objetos.

Un servicio es implementado por uno o más componentes y es una agregación de nivel superior de un componente. La reutilización de componentes funciona bien en ambientes homogéneos; la orientación al servicio llena el vacío mediante el establecimiento de reutilización en entornos heterogéneos gracias a la agregación de uno o más componentes en un servicio y que sean accesibles a través de mensajes en base a estándares abiertos. Las definiciones de servicios se implementan con el servicio, y gobiernan la comunicación de los consumidores del mismo a través de diversos contratos y políticas, entre otras cosas.

SOA también ayuda a promover la reutilización a nivel empresa. Los servicios pueden proporcionar un beneficio significativo, ya que se puede lograr su reutilización en varios niveles de abstracción en comparación con los métodos tradicionales (en otras palabras, la orientación a objetos proporcionan sólo objetos como mecanismo de reutilización primaria). SOA puede ofrecer reutilización en múltiples niveles, incluyendo el código, el servicio, y / o la funcionalidad. Esta característica mejora la flexibilidad para diseñar aplicaciones empresariales.

WCF hace que sea más fácil para los desarrolladores crear servicios que se adhieren al principio de la orientación a servicios. Por ejemplo, en el interior, se puede utilizar la tecnología OO para implementar uno o más componentes que constituyen un servicio. En el exterior, la comunicación con el servicio se basa en mensajes. En el extremo, estas tecnologías son complementarias entre sí y colectivamente proporcionan la arquitectura general de SOA.

El ámbito de aplicación de un enfoque SOA permite incorporar sistemas de largo alcance a través de una serie de plataformas y lenguajes. Un gran ejemplo de la normalización en la empresa hoy en día son los servicios web. Estos exponen una funcionalidad que puede ser descubierta y consumido en un formato estandarizado de neutralidad tecnológica.

Los servicios son procesos independientes definidos por un interfaz. Los servicios tienen un grado de granularidad mayor a la de los objetos o los componentes. Los servicios manejan sus propios recursos y también tienen una vida útil y ciclo de vida que es independiente de las aplicaciones que lo utilizan.

Los componentes son bloques de construcción a nivel de la plataforma creados para una en particular. Los componentes también son definidos por su interfaz, pero ésta depende de la plataforma.

Los objetos son bloques de construcción a nivel de lenguaje que se encuentran por lo general dentro de los componentes y los servicios. Los objetos representan conceptos de granularidad fina como ser entidades de datos o reglas de negocios.

Se pueden enumerar las diferencias entre servicios componentes y objetos de la siguiente manera:

- Los servicios son procesos independientes definidos por su interfaz
- Los componentes son bloques de construcción a nivel de plataforma como por ejemplo las dll a las que se puede referenciar en tiempo de diseño y acceder en tiempo de ejecución.

Servicios Web como una tecnología habilitadora clave para una arquitectura orientada a servicios

Ha habido mucha discusión acerca de SOA y los servicios web en los últimos años. Podría parecer que los servicios y los servicios web son análogos en el contexto de SOA. A primera vista esto puede parecer correcto, pero la realidad es todo lo contrario. Un servicio web es sólo un tipo de implementación de un servicio en un entorno en particular (la Web). Los servicios Web son un catalizador para una implementación SOA más allá del entorno. En los últimos años, gracias a las nuevas herramientas que con la relativa facilidad permiten crear servicios Web, se ha hecho más

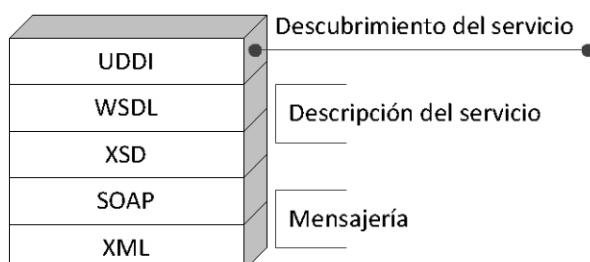
fácil entregar implementaciones SOA. El concepto de SOA no es nuevo, y algunas empresas (como IBM) lo han utilizado por más de una década.

Casi todo el mundo habla acerca de los servicios web, pero curiosamente no existe una definición universalmente aceptable. Una de las definiciones que es aceptada por algunos, es la siguiente: "Un servicio web es un componente de aplicación programables accesibles a través de protocolos web estándar." Los aspectos clave de un servicio web son los siguientes:

- **Protocolo estándar:** La funcionalidad se expone a través de interfaces usando uno de los pocos protocolos estándar de Internet como HTTP, SMTP, FTP y así sucesivamente. En la mayoría de los casos, este protocolo es HTTP.
- **Descripción del servicio:** Los servicios Web necesitan describir sus interfaces en detalle para que el cliente sepa cómo se "consume" la funcionalidad proporcionada por el servicio. Esta descripción se suele realizar a través de un documento XML llamado documento WSDL (Web Services Description Language).
- **Servicios Hallazgo:** Los usuarios necesitan saber cuáles son los servicios web y dónde encontrarlos para que los clientes puedan enlazarse a ellos y utilizar su funcionalidad. Una forma para que los usuarios sepan cuáles son los servicios es conectarse a una especie de "páginas amarillas" que lista los servicios. Estas "páginas amarillas" se implementan a través de los repositorios de integración (UDDI - Universal Discovery, Description, and Integration). Estos nodos UDDI pueden ser privados o públicos.

Nota: Un servicio web no es un modelo de objetos y no es específico del protocolo. En otras palabras, se basa en un protocolo omnipresente de Internet (HTTP) y el formato de datos (XML). Un servicio web tampoco depende de un lenguaje de programación específico. Se puede optar por utilizar cualquier lenguaje o plataforma, siempre y cuando se puede consumir y crear mensajes para el mencionado servicio web.

La siguiente figura muestra la pila de protocolos de base para los servicios web. La interacción con el servicio suele seguir una dirección de arriba hacia abajo, desde el descubrimiento del servicio a la mensajería a través de invocar a los métodos del servicio.



Para consumir un servicio web, primero se lo debe encontrar y para saber lo que ofrece (que se puede lograr mediante el uso de UDDI si es un servicio público y si no se conoce su dirección). Una

vez que haya encontrado el servicio web, es necesario comprender la interfaz: cuáles son los métodos, los parámetros aceptados y así sucesivamente. Tradicionalmente, una parte de este descubrimiento también implica los tipos de datos y el esquema que el servicio espera. Esto se puede lograr con la descripción del servicio mediante WSDL y el XML Schema Definition (XSD). Sin embargo, con WCF, la recomendación es utilizar UDDI exclusivamente para la publicación de puntos finales de intercambio de metadatos de servicios Web (WS-MetadataExchange o MEX) y pedir al servicio directamente el WSDL. Por último, el cliente debe llamar al servicio web a través de SOAP, que se basa en XML.

Los servicios WCF también siguen la pila de estándares abiertos que se mostró en la figura anterior. Por lo tanto, esta pila no sólo se refiere a los servicios web, sino que también describe la pila de protocolos para **cualquier servicio**. Por definición, los servicios no tienen que depender de servidores IIS o web para proporcionar entornos de alojamiento (servidores). WCF permite a los desarrolladores servir servicios fuera de IIS. Por lo tanto, no es necesario restringir los servicios que se originan a un servidor Web, razón por la cual no necesitan ser llamados servicios web. La pila de protocolos juega un papel importante en la comprensión de la funcionalidad SOA. Por lo tanto, hablaremos de estos protocolos, uno por uno. Vamos a empezar con SOAP.

Nota: Los servicios Web utilizan metadatos en un punto final para describir lo que otros puntos finales necesitan saber para interactuar con ellos. Esto incluye WS-Policy (que describe las capacidades, requisitos y características generales), WSDL y XML Schema. Para la comunicación de arranque con los servicios web y recuperar estos y otros tipos de metadatos, utilizar MEX, que es una especificación que define los mensajes para recuperar los metadatos y las políticas asociadas a un punto final. Esta es una especificación estándar de la industria de acuerdo en la mayoría de las principales empresas de software como Microsoft, IBM, Sun Microsystems, webMethods, SAP, etc. Las interacciones definidas por esta especificación están diseñadas sólo para la recuperación de metadatos y no se utilizan para recuperar tipos de datos, tales como estados, propiedades, etc., que podría estar asociados con el servicio.

Introducción a SOAP

En pocas palabras, SOAP es un protocolo de comunicación ligero para los servicios web basados en XML. Se utiliza para el intercambio de información estructurada y con tipo entre los sistemas. SOAP permite invocar métodos en máquinas remotas sin conocer los detalles específicos de la plataforma o software que se ejecuta en dichas máquinas. XML se utiliza para representar los datos, mientras que los datos se estructuran de acuerdo con el esquema de SOAP. El único tema es que tanto el consumidor como el proveedor tienen que estar de acuerdo en usar el esquema común definido por SOAP. En general, SOAP mantiene las cosas lo más simple posible y proporciona una funcionalidad mínima. Las características de un mensaje SOAP son los siguientes:

- Es extensible.
- Funciona a través de una serie de protocolos de red subyacentes (por debajo de él) estandarizados.
- Es independiente del lenguaje o plataforma o modelo de programación subyacente.

Nota: SOAP significa en inglés Protocolo Simple de Acceso a Objetos, pero el W3C descartó ese nombre a partir del enfoque de "acceso" versus "interoperabilidad" de objetos a través de un formato de mensajería XML generalizada como parte de SOAP 1.2.

SOAP reconoce los siguientes patrones de intercambio de mensajes: de un solo sentido, de petición-respuesta y bidireccional. Un mensaje SOAP se puede enviar a través de cualquier protocolo de comunicación y tanto el remitente como el receptor se pueden escribir en cualquier modelo de programación o puede ejecutarse en cualquier plataforma.

Extensible

Extensibilidad es el factor clave para SOAP además de la simplicidad de diseño. La extensibilidad permite varias características como confiabilidad, seguridad, etc., para ser puestas en "capas" a través de las extensiones SOAP.

Transporte

SOAP puede usar uno de los muchos protocolos de transporte estándar (tales como TCP, SMTP, FTP, MSMQ, etc.). Es necesario definir enlaces de protocolos estándar, que determinan las reglas para que el entorno pueda hacer frente a la interoperabilidad. La especificación SOAP proporciona un marco flexible para definir enlaces de protocolos arbitrarios y proporciona un enlace explícito para HTTP porque es ampliamente utilizado.

Modelo de programación

Uno de los puntos fuertes de SOAP es que no está ligado a RPC, pero se puede utilizar en cualquier modelo de programación. La mayoría de los desarrolladores se sorprenden al enterarse que el modelo SOAP es similar a un modelo de mensajería tradicional (como MSMQ) y menos a un estilo RPC. Además permite un número de patrones para comunicaciones distribuidas llamado MEP y uno de ellos es el modelo de solicitud-respuesta.

Nota: Los MEPs son fundamentalmente un conjunto de patrones de diseño para las comunicaciones distribuidas. Estos definen las plantillas para el intercambio de mensajes entre dos entidades. Algunos ejemplos son RPC, Transferencia Representativo del Estado (abreviado en inglés REST), de un solo sentido, de petición-respuesta y bidireccional. RPC es esencialmente un protocolo que permite que una aplicación ejecute otra aplicación o módulo en otro equipo conectado por medio de una red, sin que el desarrollador tenga que escribir ningún código explícito para realizar la invocación. REST, por otro lado, es un estilo arquitectónico que es diferente de RPC. Este utiliza un interfaz XML-y basada en HTTP simple, pero sin la abstracción de protocolo como SOAP. Algunos puristas ven esto como el subconjunto de las "mejores" arquitectura de la Web.

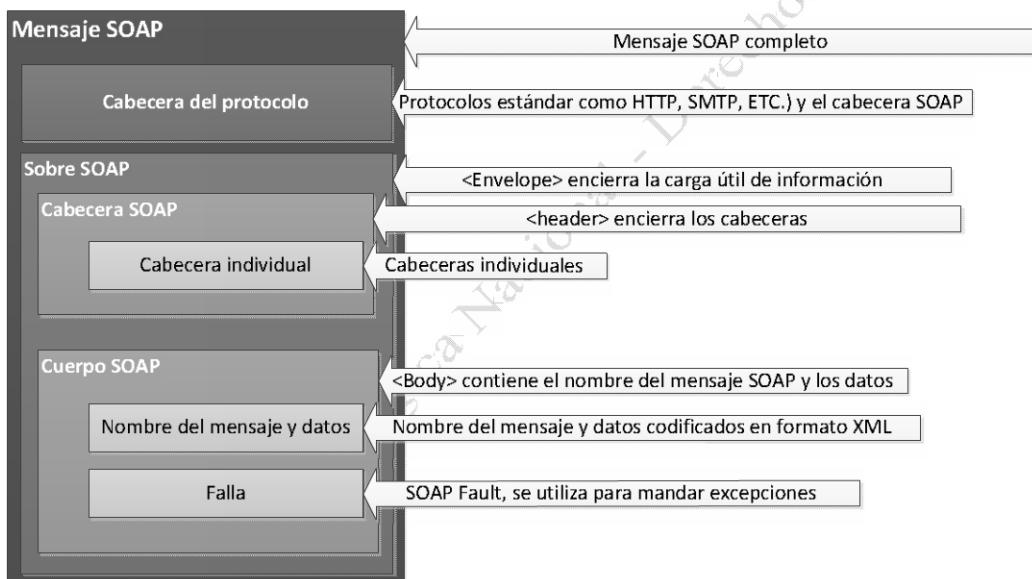
Mensajes SOAP

El bloque de construcción de SOAP es el *mensaje SOAP*, que consta de las siguientes cuatro partes:

- Un *sobre de SOAP* es un documento XML que encapsula el mensaje a comunicar. Esta es la única parte que se requiere, y el resto es opcional.

- La segunda parte del mensaje SOAP se utiliza para definir los tipos de datos personalizados que la aplicación está utilizando.
- La tercera parte del mensaje describe el patrón de RPC a ser utilizado.
- La última parte del mensaje de SOAP define cómo se une a HTTP

Un sobre de SOAP es el elemento raíz del mensaje. El *sobre* tiene dos secciones: el encabezado y el cuerpo. La cabecera tiene metadatos acerca del mensaje que puede ser necesario para un procesamiento específico, tal como la fecha y hora en que el mensaje fue enviado o un token de autenticación. El *cuerpo* contiene los detalles del mensaje o un error de SOAP. La figura 1-4 muestra la estructura de un mensaje SOAP. Un mensaje SOAP puede ser de tres tipos, a saber, solicitud de mensajes, mensajes de respuesta, y los mensajes de error.



Un mensaje SOAP del tipo requerimiento-respuesta, esencialmente tiene dos mensajes: uno es el mensaje de requerimiento enviado al servicio y el otro es un mensaje de respuesta enviado de vuelta al cliente. Aunque estos mensajes son independientes el uno del otro, el patrón de requerimiento-respuesta proporciona una correlación automática y sincronización entre los mensajes. Esto se traduce en semántica de llamada "normal" al procedimiento para el cliente.

Sintácticamente un mensaje SOAP es bastante simple, pero sigue algunas reglas como las que se muestran a continuación:

- Debe ser codificado en XML
- Se debe utilizar el espacio de nombres del sobre SOAP
- Se debe utilizar el espacio de nombres de la codificación SOAP

- No puede contener una referencia DTD
- No puede contener instrucciones de procesamiento XML

Errores SOAP (Faults)

Cuando una excepción debe ser retornada por el servicio, esto se lleva a cabo utilizando un elemento de error (Fault) en una respuesta SOAP. Un elemento de este tipo tiene que estar dentro del elemento de cuerpo del mensaje SOAP y puede aparecer sólo una vez. El elemento de error debe contener un elemento **faultcode** y un elemento **faultstring**. El mensaje de fallo contiene los detalles de la excepción, tales como código de error, la descripción, y así sucesivamente. La siguiente tabla enumera los tipos de error definidos por la especificación.

Error	Descripción
VersionMismatch	El receptor de SOAP leyó un espacio de nombres asociado con el sobre SOAP que no reconoció.
MustUnderstand	El receptor del mensaje no entendía una cabecera requerida.
Cliente	El mensaje enviado por el cliente, o bien no se formó correctamente o contenía información incorrecta.
Servidor	Un error ocurrió al servidor al procesar el mensaje.

Formato de mensaje SOAP

Existen dos tipos de formatos o estilos de mensajes SOAP: documento y RPC. El estilo documento indica que el cuerpo del mensaje contiene XML en un formato que debe ser acordado entre el emisor y el receptor. El estilo RPC indica que es una representación de una llamada a un método.

También se tiene dos formas de serializar los datos a XML: el uso de definiciones literales de esquema XML y utilizando reglas de codificación SOAP. En la primera opción, la definición del esquema define el formato XML sin ninguna ambigüedad. Estas reglas se definen en la memoria descriptiva y se conoce como *codificación de la Sección 5*. Es más adecuado para el uso al estilo RPC de SOAP porque esta opción especifica cómo serializar la mayoría de los tipos de datos incluyendo matrices, objetos, estructuras, etc. Por otro lado, en la segunda opción, las diversas reglas de codificación de SOAP (por lo general se define a través de un esquema de W3C) deben analizarse en tiempo de ejecución para determinar la serialización adecuada del cuerpo SOAP. Aunque el W3C permite el uso de los formatos de documentos literales y RPC-literal, Microsoft recomienda la primera sobre la segunda, porque el mensaje no debe dictar la forma en que está siendo procesado por el servicio.

Web Services Description Language: Describir los extremos del servicio

Si no existieran normas, habría sido difícil para los servicios web y por lo tanto para SOA, ser tan ampliamente aceptados. WSDL proporciona el formato estándar para la especificación de las interfaces y permite que se las integre. Antes de entrar en los detalles de WSDL, se necesita entender el concepto de puntos finales, ya que son de suma importancia para WSDL.

¿Qué son los puntos finales?

Oficialmente, el W3C define un punto final como “una asociación entre un enlace de interfaz completamente especificada y una dirección de red, especificado por un URI que puede ser utilizado para comunicarse con una instancia de un servicio web”. En otras palabras, un *punto final* es la entidad a la que un cliente se conecta mediante la utilización de un protocolo específico y con un formato de datos cuando consume un servicio. Este punto final reside en un lugar bien conocido que es accesible para el cliente. En un nivel abstracto, esto es similar a un puerto. Al mirar desde la perspectiva de los puntos finales, un servicio también se puede definir como una colección de puntos finales.

WSDL

WSDL conforma la base de los servicios web y es el formato que los describe. WSDL describe la interfaz pública de un servicio web, incluyendo sus metadatos, como por ejemplo enlaces de protocolos, formatos de mensajes, etc. Un cliente que quiera conectarse a un servicio web puede leer el WSDL para determinar qué contratos están disponibles en dicho servicio web. Si se recuerda de antes en este capítulo, uno de los principios fundamentales de la orientación al servicio es que los contratos de compartir servicios y esquemas, no clases. Como resultado, cuando se crea un servicio, es necesario asegurarse que el contrato del servicio está bien pensado y es algo que no cambiará.

Los tipos personalizados utilizados están incrustados en el WSDL utilizando un esquema XML. El WSDL es similar al lenguaje de descripción de interfaz (IDL) para los servicios web. La información del documento WSDL se interpreta generalmente en tiempo de diseño para generar un objeto proxy. El cliente utiliza este objeto proxy en tiempo de ejecución para enviar y recibir mensajes SOAP hacia y desde el servicio.

Nota IDL es un lenguaje normalizado utilizado para describir la interfaz a un componente o rutina. IDL es especialmente útil cuando se llama a los componentes de un equipo a través de RPC, que se estén ejecutando en una plataforma diferente o hayan sido construidos usando un lenguaje diferente y puede no compartir la misma "semántica de llamada."

Un documento WSDL tiene tres partes, a saber: las definiciones, las operaciones y los enlaces de servicios, que se pueden asignar a uno de los elementos que figuran en la Tabla 1-2.

Elemento	Descripción
<portType>	Operaciones realizadas por el servicio web
<mensaje>	Mensaje usado por el servicio web
<types>	Los tipos de datos utilizados por el servicio web
<binding>	Define un punto final de comunicación (mediante el protocolo y la dirección) para acceder al servicio
<Servicio>	Agrega múltiples puertos (en combinación con el enlace y la dirección en un servicio)

Definiciones

Las definiciones se expresan en XML e incluyen tanto el tipo de datos y como las definiciones de mensajes. Estas definiciones se basan en un vocabulario de “acuerdo-acerca de” el XML que a su vez debe basarse en un conjunto de vocabulario de toda la industria.

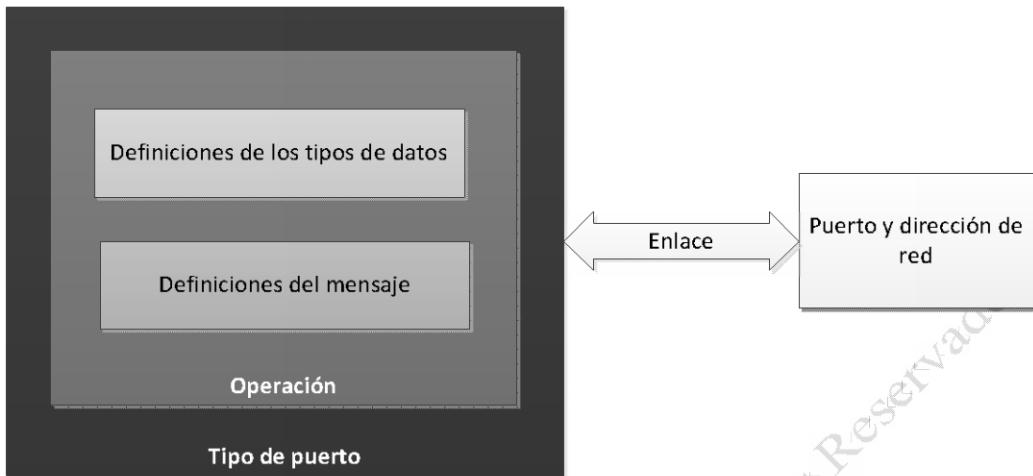
Nota Las definiciones no son obstáculos para XML y pueden expresarse en formatos que no sean XML. A modo de ejemplo, se puede utilizar el Object Management Group (OMG) IDL en lugar de XML. Si se utiliza un formato de definición diferente, al igual que con XML, tanto los remitentes y receptores tendrán que ponerse de acuerdo sobre el formato y el vocabulario. Sin embargo, según la especificación oficial del W3C WSDL, la preferencia es utilizar XSD como el sistema de tipos canónico. Cumplir con ello garantizar la máxima interoperabilidad y la neutralidad de la plataforma.

Operaciones

Las operaciones describen las acciones para el mensaje soportado por el servicio web y puede ser uno de los cuatro tipos, que se enumeran en la siguiente tabla. En una estructura de documento WSDL, las operaciones se representan mediante el elemento `<portType>`, el cual es el elemento más importante, ya que define las operaciones que se pueden realizar. En el contexto del paradigma orientado a objetos, cada operación es un método.

Error	Descripción
Una vía	El punto final de servicio recibe un mensaje.
Requerimiento-respuesta	El punto final de servicio recibe un mensaje y envía mensaje asociado
Solicitud-respuesta	El punto final de servicio envía un mensaje y recibe un mensaje asociado
Notificación	El punto final de servicio envía un mensaje.

La siguiente figura muestra la estructura de un documento WSDL, que consta de definiciones abstractas y descripciones concretas. A la izquierda está la definición abstracta donde la definición del tipo de datos es un contenedor para el uso de algún tipo de sistema como XSD. Las *definiciones de mensajes*, como su nombre indica, son las definiciones escritas de los datos que se están comunicando. Una *operación* es una descripción de una acción soportada por el servicio y contiene uno o más tipos de datos y definiciones de mensaje. Los tipos de puertos son un conjunto de operaciones con el apoyo de más de un punto final. Todos ellos están juntos en el *enlace*, que es el protocolo concreto y el formato de datos especificado para un tipo de puerto en particular. Un *puerto* es un único punto final definido como la combinación del enlace y la dirección de red. La mayoría de los desarrolladores no manejan a mano el WSDL y en su lugar utilizan el Framework de .Net para generarlo.



Enlaces de servicio

Los enlaces de servicio conectan tipos de puerto a un puerto (es decir, el formato del mensaje y detalles del protocolo para cada puerto). Un puerto se define mediante la asociación de una dirección de red a un tipo de puerto. Un servicio puede contener múltiples puertos. Este enlace se crea normalmente mediante SOAP. El elemento de enlace tiene dos atributos: un nombre que puede ser cualquier cosa, para definir el enlace y el tipo, que apunta al puerto del mencionado enlace.

Ahora que se está familiarizado con SOAP y cómo describir los servicios con WSDL. Sin embargo, ¿cómo se puede descubrir todos los servicios que están disponibles? ¿Qué estándares abiertos están disponibles para descubrir dinámicamente servicios y consumir en tiempo de ejecución? Esto se consigue mediante la implementación de UDDI.

Descubriendo dinámicamente los servicios web

UDDI es un protocolo de directorio independiente de la plataforma para la descripción de los servicios y descubrirlos e integrarlos como servicios de oficina a través de Internet. UDDI también se basa en los protocolos estándar de la industria, tales como HTTP, XML, SOAP, etc, y también describe los detalles de los servicios que utilizan WSDL y se comunica a través de SOAP. La filosofía detrás de UDDI es como la tradicional "páginas amarillas" donde se puede buscar una empresa, la buscar los servicios que ofrece e incluso ponerse en contacto con dicha empresa para obtener más información.

Una entrada de UDDI no es más que un archivo XML que detalla al negocio y los servicios que ofrece. Se compone de tres partes: las páginas blancas, amarillas y verdes. Las *páginas blancas* contienen los detalles de la compañía y su información de contacto. Las *páginas amarillas* son categorías de la industria basado en taxonomías estandarizadas, como el Sistema de Clasificación Industrial de América del Norte, y así sucesivamente. Las *páginas verdes* contienen los detalles técnicos que describen la interfaz a través de un WSDL para que un consumidor tenga suficiente información acerca de cómo utilizar el servicio y lo que se espera. Un directorio UDDI

también incluye varias formas de buscar los servicios que ofrece, incluyendo varias opciones de filtro como la ubicación geográfica, tipo de negocio, proveedor específico, etc.

Antes que UDDI se normalizara, no había una manera universal de saber qué servicios eran ofrecidos por socios u opciones comerciales. También no había manera de conseguir la integración estándar y resolver dependencias entre proveedores. Los problemas que resuelve UDDI son los siguientes:

- UDDI permite descubrir el servicio adecuado que se puede utilizar, en lugar de reinventar la rueda.
- UDDI resuelve la necesidad impulsada por el cliente para eliminar las barreras para permitir la rápida inserción en la economía global de Internet.
- UDDI describe los servicios y procesos de negocio mediante programación en un entorno único, abierto y seguro.

Nota: UDDI ofrece más que soporte en tiempo de diseño. Desempeña un papel crítico luego del descubrimiento de un servicio, ya que permite al cliente consultar mediante programación la infraestructura UDDI, que le permite a las aplicaciones cliente ser más robustas. Con una capa en tiempo de ejecución entre el cliente y el servicio web, los clientes pueden estar relacionados de forma flexible, lo cual les posibilita ser más flexibles a los cambios. Microsoft recomienda la publicación de un WS-MEX como parte de las entradas en el registro UDDI.

Envío de mensajes entre sistemas débilmente acoplados

Para lograr orientación al servicio, se necesita la capacidad de enviar mensajes de un servicio a otro. En el contexto de WCF, *la invocación de servicios* es un mecanismo general para el envío de mensajes entre la entidad que solicita un servicio y otra entidad que lo presta. Es importante entender que no importa donde existen físicamente el proveedor y el consumidor, ya que podrían estar en la misma máquina física o espaciados en los extremos opuestos del planeta. Sin embargo, desde la perspectiva de *la ejecución del servicio*, es lo que importa y WCF llena esta brecha de infraestructura.

La invocación de servicios, con independencia de la plataforma y la tecnología, sigue un patrón similar. En un nivel alto, las etapas implicadas cuando un consumidor envía un mensaje a un proveedor son los siguientes:

1. Encontrar el servicio correspondiente que expone la funcionalidad deseada.
2. Saber el tipo y formato de los mensajes que el servicio aceptaría.
3. Comprender los metadatos específicos que pudieran ser necesarios como parte del mensaje (por ejemplo, para la operación o seguridad).
4. Enviar el mensaje al proveedor con todos los datos y los metadatos correspondientes.
5. Procesar el mensaje de respuesta desde el servicio de la manera adecuada (por ejemplo, la solicitud podría haber tenido éxito, o que podría haber fallado debido a datos incorrectos o fallo en la red, y así sucesivamente).

Dado que los servicios Web son la aplicación más popular de la orientación a servicios, vamos a ver un ejemplo de cómo se puede utilizar un servicio web para enviar mensajes desde una aplicación a otra. La invocación de un "método" de servicios web es similar a llamar a un método regular, sin embargo, en el primer caso, el procedimiento se ejecuta en una máquina remota. Como el servicio se ejecuta en otro equipo, toda la información que necesita el mismo debe ser pasada a la máquina que lo aloja. Esta información luego se procesa, y el resultado se envía al cliente.

El ciclo de vida de un servicio usando XML tiene ocho pasos:

1. Un cliente se conecta a Internet o una Intranet y encuentra un servicio de directorio a usar (también se puede conectar directamente al WSDL si conoce el puerto donde se publica este documento. Si este es el caso, se arranca del paso 5).
2. El cliente se conecta al servicio de directorio con el fin de ejecutar una consulta.
3. El cliente se ejecuta la consulta correspondiente con el servicio de directorio para encontrar el servicio web que ofrece la funcionalidad deseada.
4. Se estableció contacto con el proveedor de servicios web relevantes para garantizar que el servicio sigue siendo válido y está disponible.
5. El documento con el lenguaje de descripción del servicio web (WSDL) correspondiente se recupera y se envía al cliente.
6. El cliente crea una nueva instancia de un servicio Web XML a través de la clase de proxy (recordar que el servicio se puede brindar en intranet).
7. En tiempo de ejecución desde el cliente se serializan los argumentos del método del servicio en un mensaje SOAP y se lo envía a través de la red al servicio.
8. El método solicitado se ejecuta, establece el valor de retorno e incluye los parámetros out.

Nota Aunque se ha mostrado el ejemplo de una aplicación de servicio web, WCF proporciona la unificación a través de las muchas tecnologías de computación distribuida y proporciona un modelo de programación común, independientemente del protocolo o tecnología subyacente. Además, existen diferencias en el tiempo de diseño y el tiempo de ejecución de una aplicación WCF. En tiempo de diseño, por ejemplo, una implementación de SOA por lo general en cuenta para la definición del servicio, los diferentes tipos de datos que se esperan, en qué orden, etc. Sin embargo, muchos elementos se tienen en cuenta sólo en tiempo de ejecución, tales como la ubicación física del servicio, latencia de la red y el fracaso, la detección de errores y recuperación, y así sucesivamente. Todos estos atributos no se gestionan en tiempo de diseño, pero son recogidos en tiempo de ejecución por WCF, el cual también puede cambiar los comportamientos del servicio durante la operación real.

Introducción a la arquitectura WCF

Se van a describir conceptos fundamentales incluyendo como se comunican los mensajes del servicio la noción de ABC los componentes principales que constituyen un servicio y como se considera un modelo unificado de programación.

El desarrollo orientado a los servicios con WCF

El desarrollo orientado los servicios en WCF consiste en dos pasos. Primero se crea un servicio que expone un conjunto de funcionalidades relacionadas sobre un protocolo de red. Segundo, los clientes invocan cualquiera de los métodos expuestos por el servicio para acceder a la correspondiente funcionalidad. Las aplicaciones cliente se comunican de manera orientada a los mensajes utilizándolos como de “requerimientos” y potencialmente como mensajes de respuesta.

Es importante recordar que en WCF se pueden realizar comunicaciones mediante diferentes protocolos como ser HTTP o TCP.

El desarrollo orientado a los servicios apunta a clientes y servicios que no comparten el mismo espacio de direcciones. Sin embargo el cliente y el servicio puede de hecho ejecutarse sobre la misma máquina no necesariamente en una red.

Envío de mensajes con WCF

Un servicio WCF expone múltiples puntos finales (Endpoints) los cuales son puertas de comunicación (gateways) que proveen una manera por la cual las aplicaciones se pueden comunicar con un servicio. Cada punto final de un servicio contiene una dirección, enlace y contrato (Address, Binding y Contract), el cual provee al cliente la información necesaria para comunicarse satisfactoriamente con el servicio.

El rol del despachador (Dispatcher) de WCF es determinar para qué operación es el mensaje e invocar el código de la operación en el servicio. Existe una pila de canales entre el despachador y el entorno externo. Esta pila contiene los siguientes componentes, cada uno con un rol específico en el proceso de manejar mensajes

- Canales de protocolo (como ser la seguridad o el control de transacciones)
- Codificador de mensajes
- Canales de transportes

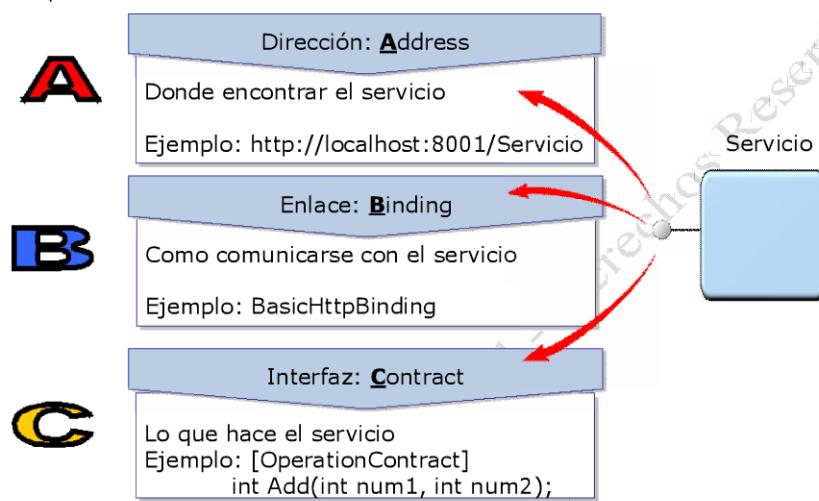
En la pila de canales implementa la política (incluyendo seguridad, confiabilidad de los mensajes y transacciones) requeridas para comunicarse con el servicio. Es importante que se ordenen los canales de acuerdo con la función que van a realizar. Por ejemplo, para un dado enlace, en la parte superior, se va a tener un flujo de transacciones seguido por una codificación y luego por el canal de transporte.

El cliente utiliza un objeto en encaminador (proxy) para invocar las operaciones expuestas por el servicio WCF. Dicho objeto ocultará la complejidad del proceso de la comunicación con un servicio remoto. El Cliente puede invocar operaciones expuesta por el servicio de la misma manera que puede invocar métodos expuestos en un objeto local.

El ABC de los puntos finales (Endpoints)

Un punto final define tres piezas de información que un cliente requiere para comunicarse satisfactoriamente con un servicio, como se muestra a continuación:

- La dirección lógica del servicio. Este es un identificador uniforme del recurso (Uniform Resource Identifier - URI) en el formato requerido por protocolo de transporte utilizado por el servicio.
- La política del enlace y los requerimientos no funcionales demandados por el servicio, según está determinado por la pila de canales. WCF provee un conjunto predefinido de enlaces, pero también se pueden definir los propios.
- La funcionalidad expuesta por el servicio. Esto incluye los nombres de los métodos de negocio, los parámetros que reciben cada método y el tipo de datos retornados. Esto debe ser especificado como un contrato de servicios.



Estructura de un servicio

Servicio posee los siguientes elementos:

- El servidor del servicio es un mecanismo para publicarlo de manera que sea accesible los clientes. Este puede ser Internet Information Services (IIS), Windows Activation (WAS) o una aplicación personalizada realizada en .NET.
- Los contratos de servicios definen la funcionalidad que este provee de manera que sea independiente de la plataforma. Por ejemplo un servicio puede proveer operaciones para consulta y entrega de datos
- El contrato IMetadataExchange permite que los clientes descubran servicios. Si un servicio lo implementa, expone un punto final para intercambio de información de Meta que lo describe. Se puede utilizar la herramienta para agregar referencias a servicios que posee el visual studio generar automáticamente un objeto encaminador. Esta característica es opcional. También se puede generar un objeto encaminador con la utilidad Svcutil.exe para no tener la necesidad poner la información de intercambio en un punto final.
- Los contratos de servicios se implementan en el Framework de .Net por medio de clases e interfaces.

Usando la interfaz a nivel de lenguaje como contrato del servicio

Frase que se pueden definir en los lenguajes de programación se utilizan como contratos de los servicios que se quiere exponer.

Ejemplo de un contrato sencillo

La marca o atributo Service Contract identificar la interfaz del lenguaje que se debe interpretar como un contrato de servicio WCF a partir de la cual se generará documento del tipo WDSL (Web Service Description Language) que posee la información de Meta que describe al servicio. El atributo Namespace debe calificar que el contrato pertenece a una organización y a un dominio funcional (este nombre por lo general se describe como un nombre de Internet). Se debe distribuir cualquier cambio en el contrato de servicio a los clientes. Caso contrario se corre el riesgo que los clientes envíen y reciban mensajes incorrectos.

La marca o atributo Operation Contract especifica la información adicional de cada operación implementada por el servicio. Los parámetros del valor retornado y las excepciones declaradas por el método con la marca o atributo cardinal información deben incluir en el protocolo utilizado para la comunicación es del tipo Simple Object Access Protocol (SOAP) para manejar los mensajes que se amplían entre el servicio y el cliente.

Ejemplo

```
C#
using System.ServiceModel;
using datos;

namespace LibreriaBanco
{
    [ServiceContract]
    public interface IBanco
    {
        [OperationContract]
        Cuenta GetBalance(int nroCuenta);

        [OperationContract]
        Cuenta Deposito(int nroCuenta, double cantidad);

        [OperationContract]
        Cuenta Retiro(int nroCuenta, double cantidad);
    }
}

VB
Imports Datos

<ServiceContract()
Public Interface IBanco

    <OperationContract()
    Function GetBalance(ByVal nroCuenta As Integer) As Cuenta

    <OperationContract()

```

```

Function Deposito(ByVal nroCuenta As Integer, cantidad As Double) As Cuenta
<OperationContract()> _
<FaultContract(GetType(ExcepcionFondosInsuficientes))> _
Function Retiro(ByVal nroCuenta As Integer, cantidad As Double) As Cuenta
End Interface

```

El atributo o marca ServiceContract

El atributo o marca ServiceContract habilita en tiempo de ejecución a WCF para generar una descripción del servicio.

Espacio de nombres por defecto para un servicio WCF es <http://tempuri.org>. Sin embargo se puede crear un espacio de nombres más específico por medio de la propiedad Namespace del atributo ServiceContract.

Si no se publican los contratos de los servicios en un punto final, dichos contratos no serán accesibles por los clientes. Los puntos finales exponen los contratos de los servicios a los clientes. Las interfaces que no estén marcados con él no serán visibles como servicios y por lo tanto no las podrán acceder los clientes.

El atributo o marca OperationContract

Este atributo es necesario para que en tiempo de ejecución se genere una descripción del servicio. Todas las operaciones expuesta del servicio deben tenerlo asociado.

Datos y mensajes

Parámetros y los valores retornados deben ser convertidos por un algoritmo al que se denomina Marshall para que la información se pueda pasar entre el cliente y el servidor. El CLR mapea sus tipos convirtiéndolos a través de una serialización a XML. WCF provee la capacidad de manejar definiciones de serializaciones personalizadas. Los datos serializados empaquetan dentro de los mensajes. El contenido y la estructura de un mensaje deben ser entendido tanto por el cliente como por el servicio en el otro fin. Es datos y los contratos de mensajes proveen control sobre estas áreas.

Contratos, Metadata y Artefactos

Para consumidor servicio, se debe saber que funcionalidad provee el mismo. Se puede utilizar la herramienta Svcutil.exe para analizar el ensamblado de servicio y extraer su metadata. Luego se puede utilizar esta metadata para crear artefactos como ser el proxy.

Demás se puede utilizar Svcutil.exe para generar un proxy analizando servicio en ejecución, sólo se puede hacer si el servicio expone metadatos de intercambio en un punto final.

Implementación de un servicio WCF

El Visual Studio provee plantillas de proyectos con la funcionalidad necesaria para crear fácilmente servicios WCF.

Definiendo el contrato del servicio y su clase

Cuando se crea un nuevo proyecto para una librería de servicio o una aplicación de servicios WCF, el Visual Studio automáticamente genera un contrato por defecto de servicio (la interfaz) y la implementación por defecto de dicho servicio.

El siguiente ejemplo de código muestra las clases generadas por defecto

Ejemplo

```
C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServiceLibrary1
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the
    // interface name "IService1" in both code and config file together.
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: Add your service operations here
    }

    // Use a data contract as illustrated in the sample below to add composite types
    // to service operations
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

```

VB
' NOTE: You can use the "Rename" command on the context menu to change the interface
' name "IService1" in both code and config file together.
<ServiceContract()
Public Interface IService1

    <OperationContract()
    Function GetData(ByVal value As Integer) As String

    <OperationContract()
    Function GetDataUsingDataContract(ByVal composite As CompositeType) _
        As CompositeType

        ' TODO: Add your service operations here

End Interface

' Use a data contract as illustrated in the sample below to add composite types to
' service operations

<DataContract()
Public Class CompositeType

    <DataMember()
    Public Property BoolValue() As Boolean

    <DataMember()
    Public Property StringValue() As String

End Class

```

También se genera el código que implementa la interfaz, la cual es una clase concreta que es el servicio en sí mismo

Ejemplo

```

C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServiceLibrary1
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the
    // class name "Service1" in both code and config file together.
    public class Service1 : IService1
    {
        public string GetData(int value)
        {
            return string.Format("You entered: {0}", value);
        }

        public CompositeType GetDataUsingDataContract(CompositeType composite)

```

```

    {
        if (composite == null)
        {
            throw new ArgumentNullException("composite");
        }
        if (composite.BoolValue)
        {
            composite.StringValue += "Suffix";
        }
        return composite;
    }
}

```

VB

```

' NOTE: You can use the "Rename" command on the context menu to change the class
' name "Service1" in both code and config file together.
Public Class Service1
    Implements IService1

    Public Function GetData(ByVal value As Integer) As String _
        Implements IService1.GetData
        Return String.Format("You entered: {0}", value)
    End Function

    Public Function GetDataUsingDataContract(ByVal composite As CompositeType) _
        As CompositeType Implements IService1.GetDataUsingDataContract
        If composite Is Nothing Then
            Throw New ArgumentNullException("composite")
        End If
        If composite.BoolValue Then
            composite.StringValue &= "Suffix"
        End If
        Return composite
    End Function

End Class

```

Se pueden modificar las clases generadas por defecto para incluir la funcionalidad personalizada que sean necesarios según los requerimientos de desarrollo.

Servidor del servicio

La manera más simple para crear un servidor que contenga al servicio es utilizando la plantilla para una aplicación de servicio (WCF Service Application template).

Nota: si bien la plantilla provee un servicio completo, es una mejor práctica definir un contrato y la implementación del servicio por medio de una plantilla de una librería de servicio (WCF Service Library) y utilizar la de la aplicación de servicio como servidor del mismo.

El Visual Studio proveer un servidor para almacenar el servicio es una versión liviana del IIS. Esto permite verificar y depurar el servicio durante el desarrollo sin explicitar que se despliegue en IIS.

Configurando el servicio

Luego de definir el contrato e implementarlo, se debe configurar el servicio de manera que se pueda comunicar con clientes remotos.

Se puede utilizar la herramienta Service Configuration Editor que viene incluida con el Visual Studio, la cual provee un editor de configuración para el servicio a través de gestionar un archivo XML.

El Service Configuration Editor provee una herramienta con una interfaz de usuario gráfica que permite modificar la configuración del archivo del servicio sin tener la necesidad de trabajar el código nativo XML.

¿Qué propiedades se deben configurar?

En poco probable dentro de que se utilice el proyecto de una aplicación de servicio WCF sin modificar el contrato por defecto o las clases de implementación, no se debe realizar cambio alguno en el archivo de configuración por defecto.

Sin embargo, si se cambia el nombre del contrato por defecto `IService1` en uno más significativo, se debe cambiar el atributo **contract** del elemento **endpoint**.

Existen muchas otras configuraciones es que se pueden realizar de manera que ajustar el comportamiento del servicio, incluyendo las siguientes:

- Agregar puntos finales adicionales como por ejemplo **mexHttpBinding** con la dirección con incidente con la que se expone la metadatos del servicio.
- Utilidades de diagnósticos como por ejemplo el registro de mensajes
- Opciones para los distintos comportamiento del servicio cuando se ejecuta si esta activada la depuración

Consumiendo servicios WCF simples

Visual Studio provee las herramientas necesarias para crear una aplicación cliente que pueda consumir un servicio.

Por ejemplo, provee una versión reducida del IIS que publica automáticamente una aplicación para que el cliente la pueda acceder. Cuando se ejecute una aplicación que tiene que cumplir la función de servidor, el Visual Studio la reconoce y la publicar adecuadamente en su IIS interno, al que se denomina Visual Studio Development Server

Importando metadatos

Para llamar a los métodos de un servicio primero se debe crear un objeto proxy (encaminador) que lo represente junto con todos sus métodos disponibles.

Agregar una referencia a servicio en Visual Studio

El entorno provee la función **Add Service Reference**, la cual genera un proxy automáticamente para un determinado servicio. Para utilizar esta característica se debe exponer metadatos en un

punto final para que pueda ser extraída la información del servicio. Si no se exponen metadatos que describan el servicio, la utilidad no será capaz de generar el proxy.

La utilidad de línea de comandos Svcutil.exe

Una alternativa a la función **Add Service Reference** es esta utilidad de línea de comandos, la cual permite generar manualmente el proxy para un servicio en un ensamblado. Por lo general luego de generar el proxy con la utilidad se agrega manualmente la clase creada a la aplicación.

Ejemplo

```
svchutil ServicioDelBanco.dll
```

Invocando el servicio usando el proxy

Cuando se genera una clase de producción se crea la clase del lado del cliente que representa al servicio y su contrato. Se pueden invocar los métodos en esta clase como si se estuviera trabajando en cualquier otro programa local. El objeto proxy oculta la complejidad de las comunicaciones entre el cliente y el servicio.

Para utilizar la clase del proxy primero se la debe tener en visibilidad a través de una referencia al espacio de nombres donde está declarado.

Ejemplo

```
C#
using ClienteServicioBanco.ReferenciaServicioBanco;
```

```
VB
Imports ClienteServicioBanco.ReferenciaServicioBanco
```

Luego se debe crear una instancia del proxy. El siguiente ejemplo muestra cómo hacerlo pasándole el punto final como parámetro.

Ejemplo

```
C#
IBanco proxy = new Cliente("WSHttpBinding_IBanco");
```

```
VB
Dim proxy As IBanco = New Cliente("WSHttpBinding_IBanco")
```

Una vez generada la estancia del proxy se pueden invocar los métodos de servicio

Ejemplo

```
C#
Cuenta c = proxy.GetBalance(1234);
```

```
VB
Dim c As Cuenta = proxy.GetBalance(1234)
```

Configurando por programación una aplicación para ser servidor de un servicio WCF

Hay una cantidad de maneras que se pueden servir los servicios WCF. Se puede utilizar Internet Information Services (IIS), Windows Activation Services (WAS) y aplicaciones auto contenidas como servidores tales como una aplicación de consola o un servicio para Windows.

Sirviendo un servicio auto contenido en una aplicación

Esto se puede ser de muchas maneras diferentes:

- Una aplicación diseñada a medida para gestionarlo
- Internet Information Services (IIS).
- Windows Activation Services (WAS)

Servir un servicio propio en una aplicación que lo gestione requiere de una cantidad adicional de codificación y configuración.

Se pueden usar plantillas de proyectos que provee el entorno para desarrollarlos y que se convierte en uno de los siguientes tipos de aplicación:

- Un programa ejecutable simple por línea de comandos.
- Un servicio de Windows que se ejecute constantemente o se atenga a un calendario previsto.
- Una aplicación de entorno gráfico que pueda incluir interacción con el cliente.

Separación de temas en un servicio WCF

Separar los temas en un servicio es una de las bases fundamentales para un buen diseño.

Teniendo en cuenta esta aproximación ayuda que los elementos de software se mantengan flexibles y de fácil manutención a medida que el sistema evolucionar. A medida que se cerró sin servicios, algunos pueden ser pequeños mientras que otros puedan ser muy grandes o empezar pequeños e ir evolucionando hasta convertirse en muy grandes a lo largo del tiempo. Además no sé sabe cómo puede ir evolucionando un servicio a medida que vayan cambiando los requerimientos a lo largo de su tiempo de vida. Por ejemplo un servicio que nació para una red local con el tiempo se puede convertir en un servicio que se exponen en Internet. En la mayoría de estos casos la implementación del servicio, el contrato y el entorno en el cual se sirve se encuentran estrechamente enlazados entre ellos de manera que realizar cambios se vuelve más complicado.

Para un servicio simple la plantilla de aplicación de servicio es suficiente. Sin embargo a causa que los componentes de servicios se almacenan juntos en el mismo proyecto, algo que sea o un poco más complicado que un servicio simple empieza a crecer y generar acoplamiento fuerte entre los componentes, por ejemplo:

- El contrato se pierde en los detalles de la implementación.
- El contrato está atado a más de una implementación.
- El servicio estaba atado a más de un entorno que nos sirve.

La mejor aproximación es mantener a los componentes y sus partes constitutivas por separado. La plantilla de aplicación de servicio muestra algunas buenas prácticas al mantener el contrato en un archivo separado de la implementación. Sin embargo en la mayoría de los proyectos de la vida real, se querrá mantener por separado al servidor del servicio en diferentes ensamblados.

Se puede utilizar la plantilla de librería de servicio para almacenar la interfaz y la implementación del mismo y luego utilizar una referencia a este proyecto para crear el servidor que lo sirve con los detalles de configuración específicos de este.

Un ejemplo de contrato

El atributo ServiceContract define el contrato e información relacionada con el servicio como por ejemplo el espacio de nombres. Por otro lado el atributo OperationContract define los métodos que el servicio expondrá cuando se implemente su contrato.

En el siguiente ejemplo se puede ver como el contrato expone tres métodos:

Ejemplo

```
C#
namespace LibreriaBanco
{
    [ServiceContract]
    public interface IBanco
    {
        [OperationContract]
        Cuenta GetBalance(int nroCuenta);

        [OperationContract]
        Cuenta Deposito(int nroCuenta, double cantidad);

        [OperationContract]
        Cuenta Retiro(int nroCuenta, double cantidad);
    }
}

VB
<ServiceContract()>
Public Interface IBanco

    <OperationContract()>
    Function GetBalance(ByVal nroCuenta As Integer) As Cuenta

    <OperationContract()>
    Function Deposito(ByVal nroCuenta As Integer, cantidad As Double) As Cuenta

    <OperationContract()>
    Function Retiro(ByVal nroCuenta As Integer, cantidad As Double) As Cuenta

End Interface
```

Ejemplo de implementación del servicio

Una clase que implementa el contrato de un servicio debe proporcionar los métodos que pertenecen al mismo. El siguiente ejemplo muestra como el contrato del punto anterior se implementa en una clase concreta.

Ejemplo

```
C#
namespace LibreriaBanco
{
    public class ServicioDelBanco : IBanco
    {
        private Dictionary<int, Cuenta> lista = new Dictionary<int, Cuenta>();

        public ServicioDelBanco()
        {
            lista.Add(1234, new Cuenta(1234, 3.5));
            lista.Add(5678, new Cuenta(5678, 55.5));
            lista.Add(9999, new Cuenta(9999, 99.5));
        }

        public Cuenta GetBalance(int nroCuenta)
        {
            Cuenta c = lista[nroCuenta];
            return c;
        }

        public Cuenta Deposito(int nroCuenta, double cantidad)
        {
            Cuenta c = lista[nroCuenta];
            c.Depositar(cantidad);
            return c;
        }

        public Cuenta Retiro(int nroCuenta, double cantidad)
        {
            Cuenta c = lista[nroCuenta];
            c.Retirar(cantidad);
            return c;
        }
    }
}
```

VB

```
Public Class ServicioDelBanco
    Implements IBanco
    Private lista As Dictionary(Of Integer, Cuenta) = _
        New Dictionary(Of Integer, Cuenta)

    Public Sub New()
        lista.Add(1234, New Cuenta(1234, 3.5))
        lista.Add(5678, New Cuenta(5678, 55.5))
        lista.Add(9999, New Cuenta(9999, 99.5))
    End Sub

    Public Function Deposito(nroCuenta As Integer, cantidad As Double) _
```

```

As Cuenta Implements IBanco.Deposito
    Dim c As Cuenta = lista(nroCuenta)
    c.Depositar(cantidad)
    Return c
End Function

Public Function GetBalance(nroCuenta As Integer) _
As Cuenta Implements IBanco.GetBalance
    Dim c As Cuenta = lista(nroCuenta)
    Return c
End Function

Public Function Retiro(nroCuenta As Integer, cantidad As Double) _
As Cuenta Implements IBanco.Retiro
    Dim c As Cuenta = lista(nroCuenta)
    c.Retirar(cantidad)
    Return c
End Function
End Class

```

Ejemplo de servidor del servicio

Se pone desarrollar una aplicación propiedad con una pequeña cantidad de código que sirva como servidor para el servicio creado. El programa más sencillo para crear un servidor que se pueda acceder para utilizar el servicio es una aplicación de consola.

El siguiente ejemplo muestra como una aplicación de consola con referencia al proyecto que contiene el servicio expone los métodos que posee para su utilización.

Ejemplo

```

C#
using System;
using System.ServiceModel;

namespace ServidorBanco
{
    class Program
    {
        static void Main(string[] args)
        {
            Uri direccionBase = new Uri("http://localhost:8888/");
            Type tipoDeLaInstancia =
                typeof(LibreriaBanco.ServicioDelBanco);

            ServiceHost servidor =
                new ServiceHost(tipoDeLaInstancia, direccionBase);
            using (servidor)
            {
                Type tipoDeContrato = typeof(LibreriaBanco.IBanco);
                string direccionRelativa = "ServicioBanco";
                servidor.AddServiceEndpoint(tipoDeContrato,
                    new WSHttpBinding(), direccionRelativa);

                servidor.Open();
            }
        }
    }
}

```

```

        Console.WriteLine("Servicio del banco " +
            "ejecutándose. Presione <ENTER> para salir.");
        Console.ReadLine();

        servidor.Close();
    }
}
}

VB
Imports System.ServiceModel

Module Module1

Sub Main()
    Dim direccionBase As New Uri("http://localhost:8888/")

    Dim tipoDeLaInstancia As Type = GetType(LibreriaBanco.ServicioDelBanco)

    Dim servidor As New ServiceHost(tipoDeLaInstancia, direccionBase)
    Using servidor
        Dim tipoDeContrato As Type = GetType(LibreriaBanco.IBanco)
        Dim direccionRelativa As String = "ServicioBanco"
        servidor.AddServiceEndpoint(tipoDeContrato, _
            New WSHttpBinding(), direccionRelativa)

        servidor.Open()
        Console.WriteLine("Servicio del banco " +
            "ejecutándose. Presione <ENTER> para salir.")
        Console.ReadLine()

        servidor.Close()
    End Using
End Sub
End Module

```

Ciclo de vida del servidor del servicio

Cuando se desarrolla una aplicación que es un servidor de servicios WCF, ya sea como un servicio de Windows o una aplicación EXE, hay una secuencia clara a seguir en el código. Las etapas del ciclo de vida de un servicio WCF manejado por una aplicación que cumple el rol de servidor son las siguientes:

1. Inicializar un objeto del tipo el ServiceHost y pasar como parámetro el tipo al que pertenece el servicio (el objeto que lo define) y la dirección base que se debe resolver para encontrarlo.
2. Abrir el objeto del tipo ServiceHost para empezar a escuchar requerimientos.
3. Detener la aplicación servidora para que no se cierre. Un servicio de Windows seguirá funcionando a través del sistema de gestión de servicios del sistema operativo, y una interfaz gráfica de usuario (GUI) de la aplicación seguirá funcionando en base al thread de la interfaz de usuario.

4. Cuando llega un mensaje para el servicio, el servidor del servicio crea una instancia de la implementación del servicio para atender el requerimiento del cliente.
5. Después que la instancia maneja el mensaje, el servidor dispone de dicha instancia de servicio.
6. Cuando se cierra la aplicación del servidor, se debe cerrar y disponer del objeto ServiceHost.

Nota: este ciclo de vida es ligeramente diferente para las diferentes políticas del ciclo de vida de servicios tales como servicios de instancia única. En este caso, se utiliza una única instancia de la implementación del servicio para iniciar el servidor. Esta instancia de la implementación del servicio solo se encarga de todas las llamadas al servicio y no se crean otras instancias del servicio.

Configuración simple por programación de una aplicación para llamar a un servicio WCF

Cuando se crea un servidor para un servicio WCF es porque se quieren crear aplicaciones clientes que puedan invocar a los métodos que éste expone. Para que un programa tenga esa posibilidad se deben realizar varios pasos que permite obtener los elementos que van a realizar la comunicación.

Requisitos previos para llamar un servicio desde una aplicación cliente

Para que un cliente pueda llamar satisfactoriamente un servicio debe cumplir los siguientes requisitos previos:

- Se debe crear una pila de canales que permite al cliente pasar mensajes al servicio. La pila de canales incluirá cada canal individual que define las características del enlace, por ejemplo, los requerimientos de restricciones de seguridad y transacciones. WCF incluye una serie de clases predefinidas como por ejemplo BasicHttpBinding, las cuales definen las características de los canales en la pila de canales.
- Una representación del lado del cliente del contrato que le permita identificar las operaciones que provee el servicio WCF.
- Un objeto proxy que oculte la complejidad de crear mensajes WCF para pasarlo al canal y que provea una manera de recuperar el contenido de las respuestas a dichos mensajes.
- Una dirección para el servicio WCF a la cual enviar los mensajes de requerimientos.

Obtener información del servicio

Una aplicación cliente puede satisfacer los requisitos previos que se mostraron en la sección anterior utilizando los metadatos que provee el servicio.

Dichos metadatos describen al mencionado servicio e incluso en información acerca de las operaciones que están expuestas y los tipos que deben ser utilizados por las mismas como parámetros o valores retornados. Se puede exponer los metadatos del servicio directamente al cliente a través de configurar el punto final Web Service Metadata Exchange que implementa el contrato IMetaDataExchange.

Si el servicio expone el mencionado punto final, se pueden utilizar herramientas del visual studio como ser Add Service Reference para crear los tipos necesarios de WCF y satisfacer los requisitos antes mencionados para la comunicación.

Otra posibilidad es satisfacer los requisitos previos obteniendo la información y los artefactos del proveedor del servicio. Esta información debe incluir lo siguiente:

- Dirección del punto final (Endpoint).
- Enlaces (Binding/s).
- Tipo de contrato común interfaz .Net

El proveedor del servicio puede entregar esto de muchas maneras diferentes. Por ejemplo puede entregar el contrato del servicio original en la forma de una interfaz de .Net junto con las instrucciones de que enlaces y puntos finales debe utilizar. De manera alternativa se puede entregar documentos de metadatos como el Web Service Description Language (WSDL) y XML Schema Definition (XSD). El cliente puede utilizar la herramienta Svcutil.exe para procesar los artefactos y generar un tipo del proxy del lado del cliente. Una tercera alternativa para el proveedor del servicio es entregar un proxy del cliente generado previamente que se puede incluir en el proyecto de programa que el mismo realiza.

Usando los Proxies de un servicio WCF

Me aplicación cliente utiliza un objeto proxy para comunicarse con un servicio. Si se utilizan herramientas como lo función Add Service Reference del Visual Studio o la de herramienta de línea de comandos Svcutil.exe, se crea una clase de tipo proxy que se puede utilizar directamente creando una estancia.

Ejemplo

C#

```
// Adquirir visibilidad del espacio de nombres.  
using ReferenciaServicioBanco;  
// Crear una nueva instancia de la clase del proxy.  
IBanco proxy = new ClienteServicioBanco();  
// Invocar los métodos en el objeto proxy.  
decimal balance = proxy.GetBalance("ABC123");
```

VB

```
' Adquirir visibilidad del espacio de nombres.  
Imports ReferenciaServicioBanco  
' Crear una nueva instancia de la clase del proxy.  
Dim proxy As IBanco = New ClienteServicioBanco()  
' Invocar los métodos en el objeto proxy.  
Dim balance As Decimal = proxy.GetBalance("ABC123")
```

De manera alternativa, se puede utilizar la clase ChannelFactory para proveer una instancia del proxy que maneje un determinado tipo de contrato. En este caso se debe proveer más

información que en la que se da cuando se instancia directamente al proxy, ya que cuando se lo genera automáticamente se obtiene dicha información del archivo de configuración.

Ejemplo

C#

```
EndpointAddress direccionPuntoFinal =
    new EndpointAddress("http://localhost:8888/ServicioBanco");
WSHttpBinding enlace = new WSHttpBinding();
IBanco proxy = ChannelFactory<IBanco>.CreateChannel(enlace,
    direccionPuntoFinal);
Cuenta c = proxy.GetBalance(1234);
```

VB

```
Dim direccionPuntoFinal =
    New EndpointAddress("http://localhost:8888/ServicioBanco")
Dim enlace = New WSHttpBinding()
Dim proxy As IBanco = ChannelFactory(Of IBanco).CreateChannel(_
    enlace, direccionPuntoFinal)
Dim c As Cuenta = proxy.GetBalance(1234)
```

Cuando se utiliza la clase ChannelFactory para crear al proxy, se puede observar el ABC de un punto final explícitamente, ya que se puede ver en las instancias de las clases la dirección, el enlace y el contrato.

Definiendo la configuración del cliente y el servicio usando archivos de configuración

Para ejecutar una solución en WCF tanto el servicio como el cliente requieren información de configuración. Dicha configuración se puede proveer tanto por programación como en un archivo externo.

Configuración en base a archivos

Se puede definir la configuración tanto del servicio como del cliente por programación utilizando las diferentes clases que existen en el espacio de nombres System.ServiceModel. Por ejemplo las clases EndpointAddress y ChannelFactory se han utilizado para este fin en ejemplos anteriores. La configuración por programación ofrece mucho control sobre la forma en que queda configurada una solución WCF. Sin embargo el inconveniente de tiene esto es que cualquier cambio que se requiera se debe volver a compilar el programa.

Fijar los parámetros por programación para la configuración también asume que el programador conoce las circunstancias en las cuales ejecutará tanto el cliente como el servicio.

Como alternativa se puede definir un archivo de configuración WCF tanto para el cliente como para el servicio en un archivo o estandarizado que se utilizan las aplicaciones para este fin como ser los archivos App.config, Web.config, o Machine.config. La configuración pasión archivos es más flexible es porque no es necesario compilar nuevamente los programas cuando se realizan cambios.

Cualquier cosa que se quiera implementar se puede mantener a nivel de código o contrato. Por ejemplo el manejo de thread e instancias.

Si una solución utilizada ambas opciones como la configuración por programación y a nivel de archivo, siempre tiene precedencia la que se realiza por programación.

Identificando las entradas correspondientes a WCF en un archivo de configuración

Un típico archivo de configuración WCF contendrá los siguientes elementos:

- **<system.serviceModel>**: define el comienzo de la sección donde se encontrarán todas las definiciones de configuración de todos los servicios que maneja la aplicación.
- **<services>**: es la sección que contiene todo los servicios
- **<service>**: es donde se define un servicio
- **<host>**: se encuentra dentro de **<service>** y define la dirección base que se debe acceder en forma de una URL. para interaccionar con el servicio. En este elemento también se puede definir el tiempo límite (timeout)
- **<endpoint>**: se encuentra dentro de **<service>** y es donde se define todos los elementos del punto final. Por ejemplo en él se encuentra la información de la dirección (el nombre con el que se encuentra al servicio), el enlace (que define, por ejemplo, el protocolo que se va a utilizar) y el contrato que se utiliza para acceder al servicio.

De seleccionando una opción de servidor que contenga al servicio WCF

Para que el servicio esté disponible los clientes, debe estar contenido en algún entorno servidor. Existen diferentes opciones entre las cuales se puede elegir, cada una de ellas con sus beneficios y desventajas.

Servidor para un servicio WCF

Existen tres opciones principales para servirlo un servicio entre las cuales se puede elegir. Estas son:

- Una aplicación de desarrollo propio, como por ejemplo una aplicación de consola, un servicio de Windows o una aplicación de entorno gráfico.
- Servir el servicio a través de Internet Information Services (IIS)
- Servir el servicio utilizando Windows Activation Service (WAS), el cual viene con IIS y habilita la capacidad de servir servicios sobre protocolos que no sean HTTP como por ejemplo TCP.

Todas estas opciones proveen la funcionalidad básica descripta anteriormente. Servir un servicio desde IIS y WAS ofrecen las siguientes mejoras sobre servicio que las aplicaciones desarrolladas en un lenguaje de programación:

- Activación en base al mensaje: crea una nueva instancia del servicio cuando llega un mensaje en lugar de tenerlo ejecutándose constantemente.
- Reciclado de procesos: reiniciar el proceso servidor para limpiar los efectos de los huecos en memoria.

- Apagado por falta de uso: detiene y serializa el servicio luego que transcurre un determinado tiempo para conservar los recursos del sistema.
- Monitoreo de la salud del proceso: verificar el estado de los recursos que utiliza los servicios para generar una respuesta en caso de problemas.

Implementando un servidor del servicio programado

Las aplicaciones que se desarrollan para ser servidoras de un servicio permiten personalizar la funcionalidad requerida.

Una aplicación de este tipo permite controlar manualmente la activación y como se sirve el servicio. Por ejemplo se puede tener una aplicación gráfica que permita determinar cuándo arranca la o se detiene el mismo.

Para implementar una aplicación de este tipo se debe escribir el código de la siguiente manera:

- Quedaron instancia de ServiceHost para el tipo de servicio
- Agregar uno o más puntos finales
- Abrieron el tipo ServiceHost para escuchar conexiones

De forma similar que cuando el servicio se presta por IIS, se debe proveer los cierta información de configuración como el ABC de un punto final. Se puede proveer dicha configuración tanto por programación como en un archivo de configuración externa.

Los servidores creados por medios de aplicaciones personalizadas no están restringidos a utilizar el protocolo HTTP como sucede cuando se utiliza IIS. Por ejemplo se puedes poner el servicio sobre el protocolo TCP en caso que sea necesario.

Este tipo de aplicaciones permiten agregar el código adicional para proveer un control mayor sobre el servicio WCF así como también proveer información adicional acerca del estado actual del servidor del servicio.

La clase ServiceHost maneja los eventos Opened, Closing, Closed y Faulted que permiten interaccionar con lo que ocurre en el servidor durante su ciclo de vida. Se pueden crear manejadores de eventos dentro de una clase para que no se escuche cuando sea necesario. Esto se debe realizar antes de abrir con el método Open el servidor.

Sirviendo desde un servicio de Windows

También se puede crear un servidor para un servicio WCF como un servicio de Windows. Estos servicios del sistema operativo proveen el beneficio de que se pueden activar, iniciar, y detener en el arranque, la ejecución y en el cierre del sistema operativo.

Para crear un servicio de Windows propio se debe derivar de la clase ServiceBase. Luego se pueden sobrescribir los métodos OnStart y OnStop para darle el comportamiento apropiado de comienzo y finalización al servicio WCF invocando los métodos de la clase ServiceHost.

El siguiente ejemplo muestra como realizarlo.

Ejemplo

C#

```
// Implementación del método OnStart.  
protected override void OnStart(string[] args)  
{  
    Type tipoDeLaInstancia = typeof(LibreriaBanco.ServicioDelBanco);  
    Type tipoDeLaInstancia = typeof(LibreriaBanco.ServicioDelBanco);  
  
    ServiceHost servidor = new ServiceHost(tipoDeLaInstancia);  
    servidor.Open();  
}  
// Implementación del método OnStop.  
protected override void OnStop()  
{  
    if (servidor != null)  
    {  
        servidor.Close();  
        servidor = null;  
    }  
}
```

VB

```
' Implementación del método OnStart.  
Protected Overloads Overrides Sub OnStart(ByVal args As String())  
    Dim tipoDeLaInstancia As Type = GetType(LibreriaBanco.ServicioDelBanco)  
  
    Dim servidor As New ServiceHost(tipoDeLaInstancia)  
    servidor.Open()  
End Sub  
' Implementación del método OnStop.  
Protected Overloads Overrides Sub OnStop()  
    If servidor = Nothing Then  
        servidor.Close()  
        servidor = Nothing  
    End If  
End Sub
```

Cuando se crea un servicio de Windows propio, también se debe crear un instalador para que lo instale en el sistema operativo. Visual Studio proveer herramientas para crear instaladores de este tipo a través de plantillas de proyectos.

Implementando un servicio servido por IIS

El IIS proveer un entorno servidor robusto que se puede utilizar para los servicios de WCF con la limitación que se debe utilizar el protocolo HTTP solamente.

Se puede crear un servidor de este tipo utilizando la plantilla de proyecto “WCF Service Application”. En esta plantilla tiene el archivo de configuración necesario para contener la configuración de WCF y el archivo .svc que contiene la directiva ServiceHost.

De esta manera se sisa se puede utilizar la plantilla de un proyecto del tipo “WCF Service Application” para mantener el contrato y la implementación del servicio en el mismo proyecto en un subdirectorio App_Code. Sin embargo la mejor forma para mantener un proyecto de este tipo es usar solo los archivos de configuración y el .svc, para luego agregar una referencia a un proyecto del tipo “WCF Service Library” que contenga el el contrato y la implementación del servicio.

Implementación de un servicio servido por WAS

Esta es otra alternativa de crear un servidor para el servicio con ciertas similitudes de como se hace en el IIS:

- No se tiene que escribir código específico para el servidor
- Se almacena la configuración WCF en un archivo externo
- Se debe proveer un archivo .svc que contenga la directiva ServiceHost
- Se debe colocar el servicio WCF en la carpeta de una aplicación IIS

WAS no requiere que el servicio WCF se esté ejecutando continuamente. Cuando llega un mensaje, WAS crea una instancia de servicios según sea necesario.

La principal ventaja de WAS sobre IIS es que se puede usar otros protocolos se además de HTTP como por ejemplo TCP.

Configuración de WAS

Para tener un servidor WAS para el servicio se debe activar en el sistema operativo del servidor “Non-Http Activation Components”. Esta se puede llevar a cabo con la función “**Turn Windows features on or off**” en el panel de control. También se deberá lanzar el sitio web deseado a un puerto que no es HTTP para que soporte una activación que no esté basada en este protocolo.

Desplegando un servicio WCF

Una vez que se ha construido y verificado un servicio se debe desplegar en la plataforma objetivo. Esto va a depender de las diferentes opciones tomadas para la creación del servicio.

Preparando un servicio para el despliegue

Cuando se prepara un servicio para el ambiente del servidor se deben realizar los siguientes pasos:

Preparar el código para despliegue. Por ejemplo, reunidos todos los ensamblados necesarios

Verificar que la configuración está lista para producción. Por ejemplo, deshabilitar los mensajes que se pasan remotamente cuando ocurra una excepción.

Cambiar la configuración específica del entorno como por ejemplo los puntos finales de WCF para que sean apropiados al entorno objetivo.

Desplegando servicios servidos por el IIS

Las tareas que se realizan cuando se despliega un servicio típico sobre IIS, sólo las siguientes:

1. Verificar que la configuración esta lista sólo para la producción de manera que cosas como la depuración y la propagación de detalles excepciones están deshabilitadas en el servicio.
2. Asegurarse que los ensamblados locales están desplegados en el directorio\bin y que los ensamblados compartidos globalmente se puedan acceder en el Global Assembly Cache (GAC).
3. Verificar que la configuración específica del entorno como por ejemplo los puntos finales del servicio WCF y la información de seguridad.
4. Asegurarse que se tiene la versión correcta del Framework de .Net instalada en la computadora que va a ser utilizada como servidor.
5. Crear una nueva carpeta de aplicación en el IIS que apunte a la carpeta que contiene el servicio WCF.

Desplegando servicios servidos por WAS

Desplegar un servicio WCF para un entorno que tengo un servidor WAS si lee el mismo proceso que cuando se sirve el servicio en el IIS 7.0. La principal diferencia es que se debe instalar en el panel de control “Windows Activation Components” y configurarlo para una activación sin HTTP para el servicio publicado.

Desplegando servicios en servidores propios

Si se crea un servidor para contener el servicio se debe desplegar lo tanto la aplicación servidora la librería que contienen servicio WCF.

En un ambiente de producción se puede realizar despliegue mediante:

- Los paquetes de instalación de Microsoft Installer (MSI) en la máquina objetivo.
- Realizar el despliegue utilizando mecanismos administrativos como los servicios de directorios de Active Directory o el Systems Management Service (SMS). En este último caso se debe tener a mano los archivos y carpetas en su defecto de los archivos .msi de Microsoft Windows Installers para quienes realizan la instalación.

Contratos de datos

Para que un objeto personalizado se pueda pasar entre servicios y clientes, hay que definir un contrato de datos. Los *contratos de datos* controlan la correspondencia entre los mensajes SOAP y los objetos de .NET. Para la que se pueda definir un contrato de datos, WCF tiene que saber cómo serializar el objeto. Hay dos maneras en las cuales WCF puede saber cómo hacerlo. WCF conoce contratos de datos implícitos y explícitos. Los contratos de datos implícitos son correlaciones de tipos base de .NET (Int32, String, etc.) y WCF tiene mapeos predefinidos para todos. Los tipos base de .NET a sus homólogos SOAP. Por lo tanto, no es necesario definir explícitamente los contratos de datos para los tipos simples. NET que se conocen del espacio de nombres System, incluyendo además enumeraciones, delegados, y vectores o genéricos de los tipos simples de .NET.

Cuando se crean nuevos tipos (las clases) sobre la base de los tipos base o simples de .NET, o basados en tipos que son a su vez construidos a partir de tipos simples, también se puede anotar a los tipos personalizados con el atributo [Serializable]. Esto le indica a WCF que utilice los contratos de datos implícitos. Si se utiliza este modo de serialización, no se tiene que definir un contrato de

datos. Para influir en la forma en que desea suceder la serialización, hay que definir un contrato de datos explícito para su tipo. Se puede hacer esto mediante la definición de una clase simple con todas las propiedades de tipo necesarias y anotar la clase con el atributo [DataContract]. Si se utiliza el atributo [DataContract] la forma de serialización de .NET se puede combinar con formateadores que determinen que se obtiene al serializar (propiedades públicas, propiedades privadas, etc). Por lo tanto, se tiene que anotar específicamente cada propiedad con el atributo [DataMember]. Sin embargo, es una buena práctica anotar cada clase que define objetos que se transmiten remotamente y definirlas como “entidades” de datos.

Ejemplo

```
C#
namespace datos
{
    [DataContract]
    public class Cuenta
    {
        private int nroCuenta;

        [DataMember]
        public int NroCuenta
        {
            get { return nroCuenta; }
            set { nroCuenta = value; }
        }

        private double balance;

        [DataMember]
        public double Balance
        {
            get { return balance; }
            set { balance = value; }
        }

        public Cuenta(int nroCuenta, double balanceInicial)
        {
            this.nroCuenta = nroCuenta;
            this.balance = balanceInicial;
        }

        public void Depositar(double cantidad)
        {
            balance += cantidad;
        }

        public void Retirar(double cantidad)
        {
            balance -= cantidad;
        }
    }
}

VB
Imports System.Runtime.Serialization
```

```

<DataContract()>
Public Class Cuenta
    Private _nroCuenta As Integer
    <DataMember()>
    Public Property NroCuenta() As Integer
        Get
            Return _nroCuenta
        End Get
        Set(ByVal value As Integer)
            _nroCuenta = value
        End Set
    End Property
    Private _balance As Double
    <DataMember()>
    Public Property Balance() As Double
        Get
            Return _balance
        End Get
        Set(ByVal value As Double)
            _balance = value
        End Set
    End Property

    Public Sub New(nroCuenta As Integer, balanceInicial As Double)
        Me.NroCuenta = nroCuenta
        Me.Balance = balanceInicial
    End Sub

    Sub Depositar(cantidad As Double)
        Balance += cantidad
    End Sub

    Sub Retirar(cantidad As Double)
        Balance -= cantidad
    End Sub
End Class

```

Laboratorio 9

Para crear el servicio WCF y el servidor que maneje la comunicación con el cliente se debe cambiar el proyecto del que se venía trabajando. En este nuevo diseño una clase dedicada a la comunicación con el servidor encaminará todas las operaciones que se deseen realizar para que estas se ejecuten en el servicio.

Por otra parte, el servicio deberá encargarse de todas las operaciones contra la base de datos. Por lo tanto éste se convierte en el modelo remoto del patrón de diseño MVC. Como se va a trabajar en una arquitectura orientada al servicio, se modifica levemente el patrón de diseño de manera que la vista no se actualizará automáticamente. Si bien la tecnología está preparada para ofrecer esto, excede de los límites de lo que se quiere mostrar como introducción al tema.

Relacionando las excepciones con los errores (Faults) a nivel de servicios

El Simple Object Access Protocol (SOAP) provee un mecanismo estándar para reportar errores a una aplicación que invoque a un servicio Web. Para incluir exitosamente información de error en las interacciones entre los clientes y los servicios se deben mapear cualquier excepción que ocurra en el servicio a un formato que se pueda trasmisir por Windows Communication Foundation (WCF) hacia el cliente y habilitarlo para que lo reconozca como una falla (Fault). Se debe recordar que se tienen que manejar estos estados porque el cliente no debe porque tener necesariamente instalado el Framework de .Net.

Relacionando las excepciones con los servicios

Una gran cantidad de aplicaciones y componentes utilizan excepciones para indicar condiciones de error. Esto pone en un problema a las aplicaciones orientadas a servicios porque las excepciones son procesos que ocurren en la memoria local de la máquina que está ejecutando el servicio. Se deben considerar que en las siguientes características de las excepciones del Framework de .Net:

- Las excepciones se proponen automáticamente para atrás en la cadena de llamados entre los métodos y hasta que de alguna manera encuentran un límite (se atrapa la excepción, su límite de proceso o una gestión en particular de la misma).
- Las excepciones contienen información local acerca del error ocurrido como puede ser un seguimiento de la pila que indica donde ocurre la excepción.
- Las excepciones contienen información que están codificadas según las especificaciones del lenguaje o la plataforma
- Las excepciones terminan en un programa si este no las atrapa de alguna manera

Los errores atrapados o causados por un servicio Web son fundamentalmente diferentes de los del Framework de .Net. Se diferencian de la siguiente manera:

- No todos los errores deben ser darrá hasta el cliente aún si el comportamiento esperado es ese.
- La información que se genera dentro del servicio dado un error puede no tener sentido en el cliente o no ser útil por lo tanto nunca debería abandonar el servidor.
- Las estructuras que son específicas de lenguaje o plataforma pueden no tener significado del lado del cliente ya que el servicio provee la capacidad de operación a través de dichas plataformas.
- Un error en una operación entre cientos o miles de operaciones no debería terminar en el servidor del servicio.

Convirtiendo las excepciones a fallas (Faults)

Especificar que los errores pueden ocurrir en un servicio es una parte importante de la aproximación del diseño por contrato para definir las operaciones de un servicio. Un caso especial es cuando el cliente no está consciente de la condición de error que se puede producir en el servidor, por lo tanto no la puede manejar.

En los servicios WCF los errores se representan en los contratos de servicios como contratos de fallas (Fault Contract). Cualquier condición de error potencial que puede ocurrir en el servicio ya

sea por razones específicas del dominio, de negocio o técnicas deberán ser mapeadas a una de las fallas definidas que una operación del servicio puede generar. Además se debe generar algún tipo de forma de falla genérica para ser utilizada cuando ocurra un error no esperado.

Si un error ocurre en tiempo de ejecución, se debe quitar la información específica de la plataforma o generar algún tipo de información del mismo que sea más significativa. Luego se puede serializar la información del error dentro de una falla. Esta falsa se propaga como una de SOAP (SOAP Fault), la cual provee una representación independiente de la plataforma del error para que se retorne al cliente.

Estructura de una falla SOAP

Es una representación en XML de la información del error que se genera cuando falla el llamado. Una falla de este tipo contiene una serie de elemento de interés que se pueden utilizar para definir una estrategia para la transmisión de errores:

Code (Código): éste es un código de falla específico de la aplicación o uno de los definidos en la especificación SOAP. Este elemento es obligatorio.

Reason (Razón): esta es una descripción en modo texto de lo que anduvo mal. Este elemento es obligatorio.

Detail (Detalle): esto puede contener datos específicos de la aplicación en formato XML de lo que se puede utilizar para pasar más información estructurada acerca del error. Este elemento es opcional.

La definición de la estructura de los errores se encuentra la especificación de SOAP. El siguiente es un ejemplo que muestra una falla SOAP de la versión 1.2.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:a ="http://www.w3.org/2005/08/addressing">
  <s:Header>
    <a:Action s:mustUnderstand ="1">
      http://www.w3.org/2005/08/addressing/soap/fault
    </a:Action>
    <a:RelatesTo>
      urn:uuid:dd129ffe-a8ff-4a70-ad6f-ad48085e94e8
    </a:RelatesTo>
    <a:To s:mustUnderstand ="1">
      http://www.w3.org/2005/08/addressing/anonymous
    </a:To>
  </s:Header>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value> s:Sender </s:Value>
      </s:Code>
      <s:Reason>
        <s:Text xml:lang ="en-US"> ERROR INFO </s:Text>
      </s:Reason>
      <s:Detail>APPLICATION-SPECIFIC INFORMATION</s:Detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

El siguiente ejemplo muestra una falla SOAP 1.1:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <s:Fault>
      <faultcode xmlns="">s:Client</faultcode>
      <faultstring xml:lang="en-US" xmlns="">ERROR
        INFO</faultstring>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

El enlace debe determinar si el mensaje de intercambio utiliza la versión de falsas de SAOP 1.1 o 1.2, ya que hubo cambios significativos entre ambas versiones. Sin entrar en mayores detalles de las diferencias se va a utilizar la versión 1.2 por simplicidad la explicación.

Por ejemplo, el enlace **basicHttpBinding** uso por defecto la versión 1.1 por cuestiones de interoperabilidad. La infraestructura de WCF serializa la representación interna de la falla ocurrida en el servicio en una forma apropiada basada la información del enlace.

Para más información, consultar en <http://www.w3.org/TR/soap12-part1/#soapfault>

Fallas en Metadatos

Luego de agregar contratos de fallos a las operaciones del servicio, las definiciones de las fallas pasan a ser parte de los metadatos del servicio desde los cuales se pueden generar artefactos para los clientes. El siguientes es un ejemplo de cómo generar un contrato de falla.

Ejemplo

C#

```
[OperationContract]
[FaultContract(typeof(LibreriaBanco.ExpcionFondosInsuficientes))]
Cuenta Retiro(int nroCuenta, double cantidad);
```

VB

```
<OperationContract()
<FaultContract(GetType(ExpcionFondosInsuficientes))> _
Function Retiro(ByVal nroCuenta As Integer, cantidad As Double) As Cuenta
```

Este contrato de fallas genera los siguientes metadatos

```
<wsdl:operation name="Retiro">

  <wsdl:input
    wsaw:Action="http://tempuri.org/MyContract/Retiro"
    message="tns:IBanco_Retiro_InputMessage"/>

  <wsdl:output
    wsaw:Action="http://tempuri.org/IBanco/RetiroResponse"
    message="tns:IBanco_Retiro_OutputMessage"/>

  <wsdl:fault
    wsaw:Action="http://tempuri.org/IBanco/ExpcionFondosInsuficientes"
    name="ExpcionFondosInsuficiente"
```

```
message="tns:IBanco_Retiro_ExcepcionFondosInsuficientes_FaultMessage
"/>

</wsdl:operation>
```

Las fallas forman parte de la información acerca de la operación. Cuando se genera un proxy para el cliente este debe incluir el código que maneja la falla y lo debe serializar en la estructura de datos apropiadas que corresponda con el Framework de .Net, incluyendo la creación de los tipos para que almacenen las fallas que contienen la información personalizada del error.

Nota: Este archivo XML no tiene que almacenarse necesariamente. WCF tiene la capacidad de manejarlo en memoria.

Usando fallas en un servicio

Es importante que servicios sean consistentes en su manejo de errores y genere información significativa de los mismos.

Diseño del manejo de errores del servicio

Se pueden clasificar los errores de la siguiente manera:

- **Errores de dominio:** estas condiciones de errores ocurren cuando se ejecuta la lógica de negocio, como por ejemplo cuando una cuenta bancaria no tiene suficiente fondos para cubrir el retiro solicitado. Se debe ser capaz de identificar todos los errores de dominio que el sistema pueda generar.
- **Errores técnicos esperados:** estos errores no están relacionados con la lógica de negocio pero pueden ocurrir como consecuencia de una razón técnica como por ejemplo el requerimiento de una conexión a una base de datos que no se puede establecer por estar el servidor ocupado. Se debe ser capaz de identificar la mayoría de estos errores técnicos que el sistema pueda generar.
- **Errores técnicos no esperados:** estos errores ocurren por causa de un fallo al manejar valores inesperados de datos, como por ejemplo los que se producen por un error en el código. Por su naturaleza es difícil anticipar estos tipos de errores pero se pueden gestionar igualmente.

El servicio debe informar el cliente acerca de los errores de dominio que puedan ocurrir. Esta información debe ser parte de la definición del contrato del servicio.

Existen dos opciones para determinar cómo el servicio puede manejar los errores técnicos:

- *Manejarlos internamente:* por ejemplo, si falla una conexión a la base de datos se puede reintentar establecerla una cantidad de veces. Sin embargo no se puede mantener eternamente esta situación por lo tanto requiere de un buen juicio de diseño establecer por cuánto tiempo se reintentará establecerla.
- *Manejarlo por medio de la interacción con el usuario:* en este caso el servicio retornará algo que le indica al usuario que ha ocurrido un error pero que no es su culpa. Esta

notificación del servicio le puede permitir al cliente tomar la decisión de reintentar operación o no.

Decidiendo qué información incluir en las fallas

Existen múltiples interesados en el sistema que pueden informar acerca de posibles errores en diferentes servicios. Para cada uno de ellos se debe determinar qué tipo de información se debe proveer. Entre las comunidades de usuarios que pueden generar reportes sobre los diferentes tipos de errores se encuentran los siguientes:

- Equipos de soporte del lado del servidor y desarrolladores: este grupo requiere la más detallada información técnica sobre la cual se pueda hacer un seguimiento del problema, como por ejemplo, ser capaz de determinar si el error proviene de una determinada línea de código o al manipular ciertos tipos de datos. Un ejemplo de este tipo de información es el volcado de pila como parte de los eventos registrados en el sistema operativo (Event Log) en la computadora que está ejecutando el servicio.
- Equipos de soporte del lado del cliente y desarrolladores: este grupo requiere menos información técnica detallada, pero sin embargo debe ser aún técnica. Un ejemplo de este tipo de información es conocer en qué servidor se ha invocado a qué servicio cuando ocurrió el error.
- Usuarios finales: este es el grupo que por lo general requiere menos cantidad información técnica. Sin embargo se puede proveer un mensaje que indique que ha ocurrido un error para que tome una determinada acción como así también un código de error que sirva para seguimiento del equipo de desarrollo.

Esto es sólo una guía para saber qué tipo información se debe transmitir entre el cliente y el servicio, sin embargo se debe analizar detalladamente para incluirla como parte del diseño.

Fallas con y sin tipo

Se pueden crear fallas con y sin tipo en WCF:

Se puede utilizar la clase FaultException para crear una falla sin tipo. Se puede configurar los elementos Code, Reason y Detail de la falla SOAP por medio de su constructor. Si el cliente fue escrito utilizando el Framework de .Net del lado del cliente se lanzará una FaultException equivalente.

Se puede utilizar la clase genérica FaultException<T> para crear una falla con un tipo determinado. En este caso se define una estructura de datos que contiene la información acerca del error y se pasa una instancia del mismo al constructor de FaultException<T>. Si el cliente se escribió utilizando el Framework de .Net se puede definir un tipo de datos para ser lanzado como una excepción FaultException<T> del lado del cliente.

Si se elige utilizar una falta con un tipo definido, se debe definir un contrato de datos para que almacene la información del error. El siguiente ejemplo muestra como realizar un contrato de este tipo.

[Visual Basic]

```

<DataContract([Namespace] := "http://myuri.org/Simple")> _
Public Class AccountOverdrawnFault

    Private m_accountNo As String

    <DataMember()> _

    Public Property AccountNo() As String

        Get
            Return m_accountNo
        End Get

        Set
            m_accountNo = value
        End Set
    End Property
    ...
End Class

```

[Visual C#]

```

[DataContract(Namespace="http://myuri.org/Simple")]
public class AccountOverdrawnFault
{
    private string accountNo;

    [DataMember]
    public string AccountNo
    {
        get { return accountNo; }
        set { accountNo = value; }
    }
    ...
}

```

Ejemplo

```
C#
using System.Runtime.Serialization;
using System.ServiceModel;
using LibreriaBanco;

namespace datos
{
    [DataContract]
    public class Cuenta
    {
        private int nroCuenta;

        [DataMember]
        public int NroCuenta
        {
            get { return nroCuenta; }
            set { nroCuenta = value; }
        }

        private double balance;

        [DataMember]
        public double Balance
        {
            get { return balance; }
            set { balance = value; }
        }

        public Cuenta(int nroCuenta, double balanceInicial)
        {
            this.nroCuenta = nroCuenta;
            this.balance = balanceInicial;
        }

        public void Depositar(double cantidad)
        {
            balance += cantidad;
        }

        public void Retirar(double cantidad)
        {

            if (balance > cantidad)
                balance -= cantidad;
            else
            {
                ExpcionFondosInsuficientes informacion =
                    new ExpcionFondosInsuficientes();
                informacion.NroCuenta = nroCuenta;
                informacion.Balance = balance;
                informacion.MensajeDeError = "Fondos Insuficientes";
                throw new FaultException<ExpcionFondosInsuficientes>(
                    informacion, "Error de operación");
            }
        }
    }
}
```

```

}

VB
Imports System.Runtime.Serialization
Imports System.ServiceModel

<DataContract()>
Public Class Cuenta
    Private _nroCuenta As Integer
    <DataMember()>
    Public Property NroCuenta() As Integer
        Get
            Return _nroCuenta
        End Get
        Set(ByVal value As Integer)
            _nroCuenta = value
        End Set
    End Property
    Private _balance As Double
    <DataMember()>
    Public Property Balance() As Double
        Get
            Return _balance
        End Get
        Set(ByVal value As Double)
            _balance = value
        End Set
    End Property

    Public Sub New(nroCuenta As Integer, balanceInicial As Double)
        Me.NroCuenta = nroCuenta
        Me.Balance = balanceInicial
    End Sub

    Sub Depositar(cantidad As Double)
        Balance += cantidad
    End Sub

    Sub Retirar(cantidad As Double)
        Dim informacion As New ExcepcionFondosInsuficientes()
        informacion.NroCuenta = NroCuenta
        informacion.Balance = Balance
        informacion.MensajeDeError = "Fondos Insuficientes"
        If _balance > cantidad Then
            _balance -= cantidad
        Else
            Throw New FaultException(Of ExcepcionFondosInsuficientes)( _
                informacion, "Error de operación")
        End If
    End Sub
End Class

```

El nivel de detalle en el error depende de quién genere la información. Dependiendo de lo sofisticado que sea sistema se pueden definir códigos de error y estructuras detalladas que se

muestran en cliente. Sin embargo la mayoría de las veces con la información del elemento Reason es suficiente.

Usando una falla con tipo

Para usar una falla con tipos se debe crear una instancia del tipo de datos que almacena la información del error en el servicio (por lo general una excepción local) y pasarlal como una instancia en el constructor para la clase FaultException definiendo el parámetro genérico con el nombre de la clase que modela la excepción.

Una vez inicializada la clase FaultException se puede lanzar para que se propague al que ha realizado la llamada.

El servidor del servicio WCF maneja el tipo FaultException y lo convierte en una falla SOAP.

El comportamiento del servicio y las excepciones

Las excepciones no manejadas en un servicio o cambian el estado del canal de comunicación a **Faulted**. Este estado previene que siga habiendo comunicaciones con una instancia particular del proxy del cliente.

El servidor del servicio se comporta diferente dependiendo del tipo de excepciones que encuentra. A menos que se atrape el error dentro del servidor, el mismo se convertirá en un error inesperado como una excepción no manejada. WCF transmite al cliente un FaultException (de manera muy similar a como lo haría para una falta sin tipo) y cambiar el estado del canal de comunicación a **Faulted**, lo cual ocasiona que los subsiguientes intentos del cliente por utilizar el mismo canal generará una excepción en su lado de la comunicación. Luego que el canal se pone **Faulted**, se debe abrir un nuevo canal creando una nueva instancia del proxy. Se puede verificar el estado de un canal examinando la propiedad State del mismo para ver si está configurada a **CommunicationState.Faulted**.

Aplicando manejo consistente de errores

Es una buena práctica agregar bloques try a todas las operaciones del servicio. Esta práctica genera diferentes beneficios:

Una excepción no manejada no se puede encontrar dentro del servidor del servicio. De esta manera se previene que el canal que se ponga en un estado de Fault y se bloquee.

Provee una locación consistente y común para realizar las excepciones y manejo de errores.

Si bien estas técnicas provocan un nivel de consistencia, puede provocar una gran cantidad de líneas de código duplicado para diferentes operaciones. El código duplicado para el manejo de errores puede generar problemas para el mantenimiento haciendo difícil emplear el código que realmente importa para el desarrollo del servicio. Otro problema es que si se realiza un cambio en el código duplicado se tiene que modificar en todos los lugares en que se encuentre.

Para mitigar estas características, si utiliza un mecanismo en común de código para el manejo de errores el cual se abstrae del código del dominio. WCF permite implementar y configurar un manejador de errores que se puede aplicar sin el requerimiento de cambiar el código de cada operación individual del servicio. Para hacer esto se debe seguir la siguiente estrategia:

1. Definir un manejador de error personalizado que implemente System.ServiceModel.Dispatcher.IErrorHandler
2. Colocar el código para el manejo del error que registra y serializa excepciones en el método IErrorHandler.HandleError
3. Implementar en el servicio la interfaz System.ServiceModel.Description.IServiceBehavior
4. En el método IServerBehavior.ApplyDispatchBehavior, agregar el manejador personalizado de error para gestionar la colección ErrorHandlers de la propiedad ServiceHost.ChannelDispatchers

Se debe tomar la decisión entre las aproximaciones del bloque try y el manejador personalizado de errores. Esta decisión se basa en un consistente sea el manejo de errores a través del servicio. Si hay mucho manejo de un error específico para diferentes operaciones es mejor utilizar el bloque try.

Diagnóstico de excepciones no esperadas

Cuando por una sección inesperada, WCF genera una FaultException y la envía al cliente. Sin embargo la información acerca de la causa de la excepción no se retorna el cliente porque la información no controlada del error es mala para la seguridad. La información acerca de los tipos de excepciones puede proveer pistas a los potenciales hackers acerca de cómo trabajan los servicios y exponer alguna otra debilidad de seguridad. Sin embargo cuando se encuentra en un entorno confiable, como uno de desarrollo o testeo, se requiere acceder a la información del error de manera que se lo puede encontrar y solucionar.

Para obtener información de la excepción incluida la falla de servicio generada por una excepción no manejada, se configura la propiedad includeExceptionDetailInFaults con un valor true en el elemento serviceDebug el archivo de configuración.

```
<behavior name="ServicioBancoBehavior">
    <serviceDebug includeExceptionDetailInFaults="true"/>
    ...
</behavior>
```

Se debe recordar configurar este valor en false cuando se despliegue de aplicación en un entorno de producción.

Manejando fallas y excepciones en los clientes

WCF propagar la falla de los servicios a los clientes y estos deben gestionarlas a través de los mensajes que reciben.

Manejando excepciones de los servicios

Un cliente debe esperar varios tipos de excepciones cuando llamado servicio:

- Tipos de fallas de dominio o técnicas que son declarados como contratos de fallas para operaciones del servicio en su contrato.
- Falsas sin tipo que maneja excepciones inesperadas dentro del servicio que el mismo convertirá en un **FaultException** sin un tipo definido.
- Excepciones relacionadas con las comunicaciones entre el cliente y el servidor. Estos pueden ser errores a nivel de transporte o pueden ser causados por fallas en el reconocimiento de los medios utilizados para la comunicación, como por ejemplo el protocolo, la seguridad el tipo de sesión.

A causa de la alta probabilidad de que ocurran errores cuando se invoca un servicio, se debe rodear el código que lo llama con un bloque try-catch.

Si el canal se encuentra en un estado de Francia por una excepción no manejada en el servicio o por una de comunicaciones, se debe volver a crear el objeto proxy. Recordar que esto se realice examinando la propiedad State del canal para verificar si su valor no está configurado en **CommunicationState.Faulted**.

Excepciones en las comunicaciones

Existen diferentes tipos de excepciones relacionadas con las comunicaciones que pueden ocurrir cuando se llama un servicio:

- **TimeoutException**: ocurren cuando se termina el tiempo estimado para la comunicación. Existen diferentes límites de tiempo para un canal abierto, al cierre de un canal y para recibir o enviar requerimientos.
- **EndpointNotFoundException**: indica que un cliente no puede encontrar el punto final del servicio. Se debe verificar la dirección del servicio que el proxy utiliza para corroborar que es una dirección válida del servicio que se esté ejecutando.
- **ProtocolException**: indicar que hay un problema en la negociación del protocolo entre el cliente y el servicio. La causa de esta excepción puede ser que el cliente y el servicio estén utilizando enlaces incompatibles.
- **CommunicationObjectFaultedException**: indicar que el canal está en un estado de Fault. Para corregir este error, se debe descartar el proxy actual y reemplazarlo por uno nuevo. Se pueden atrapar cada una de las excepciones en forma individual para manejarlas con los nombres de clases que se han indicado. Caso contrario se puede atrapar **CommunicationException** que la clase base para todos los tipos de excepciones de comunicación

Manejando falla con tipo

Si se atrapa una excepción con un tipo definido, se puede acceder al objeto que almacena la información del error leyendo la propiedad **Detail** de la clase **FaultException<T>**. Dado que la clase uso tipo genérico, se obtendrá el tipo correcto.

Ejemplo

C#

```
try
{
    c = proxy.Retiro(1234, 150);
}
catch (FaultException<ExpcionFondosInsuficientes> e)
{
    System.Console.WriteLine(e.Detail.MensajeDeError);
}
```

VB

```
Try
    c = proxy.Retiro(1234, 150)
Catch e As FaultException(Of ExpcionFondosInsuficientes)
    System.Console.WriteLine(e.Detail.MensajeDeError)
End Try
```

Manejando falla sin tipo

Se puede atrapar una falla sin tipo mediante la clase **FaultException**. Se puede acceder a la información de la falla SOAP que ésta contiene a través de las propiedades **Reason**, **Detail** y **Code**, como se muestran el siguiente ejemplo:

Para obtener más información acerca de la excepción del lado del servidor, configurar la propiedad **includeExceptionDetailInFaults** en true para el elemento **serviceDebug** en el archivo de configuración del servicio.

Laboratorio 10

El proyecto ventas no está completo sin el manejo de las condiciones de error, tanto de negocio como de infraestructura. Para poder manejarlas se debe declarar una excepción diferente a la manejada con ExpcionVentas ya que esta sirve para procesar en el mismo espacio de memoria. Por lo tanto se creará una nueva excepción que se utilizará para entregar mensajes de condiciones anómalas a los clientes con el fin de mantener separadas las ocupaciones. La clase ExpcionVentas se seguirá utilizando localmente en el espacio de memoria del servidor y brindará el detalle de lo ocurrido realizando salidas por consola. De esta manera podemos clasificar a los errores según el espacio de memoria en el cual se manejan:

- **Espacio de memoria del servidor del servicio:** maneja las excepciones que se producen en la base de datos y en el servidor
- **Espacio de memoria del cliente gráfico:** maneja las excepciones que se producen remotamente durante el procesamiento del servicio o los errores de comunicación.

Todo esto acarrea como consecuencia analizar cuándo se lanzará una excepción remota y cuando no. Las operaciones que involucran reglas de negocio fueron modificadas de manera que en lugar de lanzar ExpcionVentas, ahora lo hagan con la excepción remota.