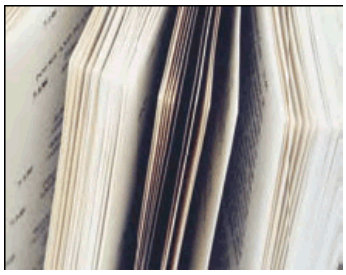
► Aprenda la disciplina, ejerza el arte y contribuya con sus ideas en www.ArchitectureJournal.net
Recursos que sirven de base.

THE ARCHITECTURE JOURNAL
Input for Better Outcomes
Journal 8

Datos a Medida

Microsoft



Contenidos

THE ARCHITECTURE JOURNAL
Input for Better Outcomes

Prólogo

Estimado arquitecto:

Bienvenido a la 8va Edición de *The Architecture Journal,* cuyo tema son los "Datos a Medida". Como arquitectos, siento que con frecuencia subestimamos la presencia de los datos en la arquitectura, en especial cuando consideramos la manera en la que los datos se utilizan en aplicaciones y sistemas que incluyen organizaciones, zonas horarias y geografías múltiples.

Una analogía que utilizo con frecuencia compara la disponibilidad de los datos dentro de un sistema con el agua que corre a través de las cañerías en una vivienda. Cuando abrimos una canilla, esperamos que instantáneamente salga agua filtrada y limpia y por lo general con una presión consistente. En esta analogía, las cañerías representan la infraestructura y el agua, los datos. Cuando pienso en datos y su relación con la arquitectura me gustaría pensar sobre el mismo enfoque –los datos que proporcionamos a los usuarios deben ser limpios, filtrados y se deben entregar sin demoras y de la forma esperada– sin importar si los datos son un simple correo electrónico, un registro de cliente o un gran conjunto de datos financieros mensuales.

Si bien no trataremos muchas técnicas de plomería en esta edición, contamos con una cantidad de artículos importantes de un grupo distinguido de autores que pone su atención en la importancia de los datos.

Comenzamos con Roger Wolter, arquitecto de soluciones para Microsoft y autor de varios artículos técnicos y libros sobre el Servidor SQL y el Agente de Servicios del Servidor SQL (SSB). Roger analiza la importancia de la confiabilidad de los datos, en especial en el contexto del diseño de sistemas conectados.

Siguen Tim Ewald y Kimberly Wolk con un artículo que explica algunos de los modelos que crearon para integrar datos en el Sistema de Publicación TechNet de MSDN, la próxima generación, un sistema basado en XML que forma las bases de MSDN2. Luego, Udi Dahan nos lleva de viaje por el uso de agregación de entidades para obtener una vista de 360 grados de nuestras entidades de datos y se concentra sobre las formas concretas de resolver las necesidades comerciales inmediatas.

Le sigue otro equipo de autores; Tom Fuller y Shawn Morgan comparten algunas de sus experiencias siendo conscientes de que la duplicación de datos puede representar un antipatrón para la Arquitectura Orientada al Servicio (SOA), en especial a la luz de las aplicaciones y los servicios autónomos. Luego, Dion Hinchcliffe, CTO en Sphere of Influence, comparte algunos patrones para la composición y consumo de datos, en especial en las áreas de aplicaciones de Web 2.0 y mushups.

Paul Nielsen finaliza nuestra serie de artículos relacionados con datos con un panorama general de Nordic, un nuevo modelo híbrido relacional/objeto que analiza una mayor flexibilidad y un mejor rendimiento sobre los modelos de datos relacionales tradicionales.

Redondeando esta edición del *Journal,* Andrew Filev comparte algunas de sus experiencias en la combinación de una metodología ágil con un modelo de tercerización y luego Arvindra Sehmi y Beat Schwegler regresan con la segunda parte de su serie Modelación para Sistemas Conectados. Si no leyó la primera parte, asegúrese de descargar la Edición 7 de *The Architecture Journal* en www.architecturejournal.net.

Bien, esto concluye nuestro tema sobre datos. Confío en que los artículos que presentamos en esta edición lo ayudarán a diseñar sistemas con la misma disponibilidad de datos con la que el agua fluye desde la canilla de su casa –por supuesto, ¡no puedo garantizarle que no se mojará las manos!

Simon Guest

Confiabilidad en sistemas conectados

Por Roger Wolter

Síntesis

Las aplicaciones de los sistemas conectados están compuestas por una cantidad de servicios con cierta relación generalmente distribuidos en una red; por lo tanto, lograr altos niveles de confiabilidad y disponibilidad para las aplicaciones de los sistemas conectados presenta un conjunto único de desafíos de arquitectura. Por ejemplo, si una aplicación deja de ejecutarse porque no está disponible cualquiera de los diez servicios que se ejecuta sobre los diez servicios diferentes, la posibilidad de falla para la aplicación es de casi diez veces la proporción de falla para un servicio individual. La falla de datos hace que este problema sea mucho más crítico porque la fuente de datos podría estar utilizando una docena de servicios. Este artículo analiza los problemas de confiabilidad que deben tenerse en cuenta al diseñar una aplicación de servicios conectados y muestra la forma en la que algunas de las nuevas características que poseen los productos de mensajería de Microsoft y Servidor SQL 2005 tratan estos problemas.

Desde las perspectivas del mantenimiento, software y hardware, la confiabilidad es costosa, por lo tanto, es muy importante comprender los requisitos de confiabilidad de la aplicación. Si bien son poco frecuente las aplicaciones que no poseen requisitos de confiabilidad, implementar una aplicación con más confiabilidad de la que se necesita podría se una pérdida de tiempo y recursos. Por esta razón, es importante comprender los problemas de confiabilidad de los sistemas conectados para que se pueda diseñar una solución que brinde el nivel necesario de confiabilidad sin desperdiciar recursos al proporcionar confiabilidad que no es necesaria.

En la arquitectura orientada al servicio (SOA) o en los sistemas conectados, los servicios se comunican entre sí a través de formatos de mensajes bien definidos, lo que significa que la confiabilidad de la aplicación de los sistemas conectados estará influenciada en gran medida por la confiabilidad de la infraestructura de la mensajería sobre la cual se basa para comunicarse entre los servicios. Utilizaremos el ejemplo de los servicios de una aplicación de un cajero automático para ilustrar los diferentes grados de confiabilidad de la mensajería y la forma en la que se logran. El manejo de mensajes entre los servicios es por lo general más complejo que la mensajería entre servidor/cliente, porque esta última puede valerse del usuario para tomar algunas decisiones sobre cómo manejar las diversas situaciones de error y desconexión, mientras que el servidor que inicia el intercambio de mensajes en una mensajería servidor-a-servidor debe tomar todas las decisiones que normalmente el usuario final realizaría en una interacción cliente/servidor.

Carga de la infraestructura

Varias aplicaciones orientadas al servicio logran un mejor rendimiento si utilizan mensajería asincrónica. Con la *mensajería asincrónica* un servicio envía un mensaje a otro sin esperar una respuesta del mensaje antes de continuar. El servicio procesa muchas otras solicitudes ya que no pierde tiempo esperando respuestas a solicitudes que realiza a otros servicios, pero pone la carga de asegurar que el mensaje se envíe y se procese sobre la infraestructura de la mensajería. La elección de gestionar un mensaje síncrono o asíncrono está por lo general determinada por los requerimientos del negocio de la aplicación.

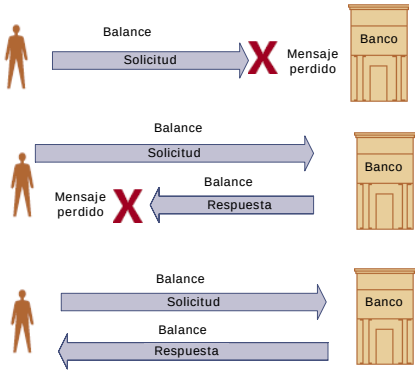
Por ejemplo, si un cliente trata de retirar dinero de un cajero automático, generalmente no es una decisión comercial sensata realizar una solicitud asincrónica para verificar el balance del cliente y continuar con el suministro del dinero sin esperar una respuesta por parte de la verificación del balance. Sin embargo, esta situación hipotética necesariamente no descarta una solicitud asincrónica. El cajero automático podría enviar una solicitud de balance tan pronto como el cliente seleccione la opción de retirar dinero y luego permitirle al cliente que ingrese una suma de dinero mientras la verificación del balance se procesa de forma asincrónica. Una vez que se ingresa la cantidad de dinero que se desea retirar y se recibe el balance, la aplicación del cajero automático puede decidir si otorga el dinero o no.

Veamos la manera en la que el cajero automático maneja el mensaje de solicitud de balance. El servicio del cajero automático envía el mensaje de solicitud de balance al servicio de cuentas para obtener el balance de la cuenta del cliente. En este punto, una de estas tres cosas sucederá: se proporcionará el balance con éxito, se proporcionará un error o la solicitud expirará su tiempo debido a que no fue devuelta de manera oportuna. Si se proporciona el balance, el servicio del cajero automático continua con la transacción. Si se devuelve un error, el cajero automático utiliza su lógica comercial para solucionar el error –podría utilizar una copia almacenada en la memoria del caché o proporcionar o no el dinero en base a la cantidad solicitada. El error más difícil de tratar es la expiración del tiempo. La expiración del tiempo tal vez significa que el mensaje se perdió en tránsito, el mensaje llegó al servicio de la cuenta pero la respuesta se demoró por algún motivo o la respuesta se envió y se perdió en el camino de regreso.

En la mayoría de los casos, la mejor acción es intentar nuevamente y esperar que funcione mejor en el próximo intento. Si la primera vez el mensaje se perdió en el camino, puede ser que llegue en el nuevo intento. Si la respuesta fuera lenta, es posible que la respuesta original llegue antes de que la segunda solicitud expire. A menos que el servicio de la cuenta sea inaccesible o que el servicio se haya perdido completamente, un nuevo intento se logrará con éxito. Dependiendo de cuál haya sido el problema, la solicitud del balance puede haber sido procesada varias veces o se pueden haber recibido múltiples resultados, pero mientras que el servicio del cajero automático esté preparado para solucionar estos errores, no son problemas serios.

Si el mensaje se envió de forma asincrónica, el servicio del cajero automático puede no contar con la información para reenviar el mensaje cuando se requiere un nuevo intento, por lo tanto, será necesario que la infraestructura de la mensajería conserve una copia del mensaje que envía y la reenvíe en caso de ser necesario. Para este tipo de mensaje de solicitud simple, probablemente sea oportuno conservar una copia del mensaje en la memoria ya que si el cajero automático se queda sin suministro eléctrico, de todas formas el cliente deberá comenzar nuevamente. Si el cajero automático requiere una respuesta aún después del corte de energía eléctrica, el mensaje deberá colocarse en algún almacenamiento persistente y el sistema del mensaje deberá reenviarlo cuando ser reinicie. La Figura 1 muestra los tres posibles resultados a una solicitud de balance.

Figura 1: Solicitud de balance



Envío del mensaje

Para la versión síncrona de esta solicitud, un simple servicio de Web SOAP proporcionará el grado necesario de confiabilidad. El manejo de errores y tiempo expirado son responsabilidad del servicio del cajero automático, por lo tanto, se debe determinar el nivel de confiabilidad para la solicitud. Para la versión asincrónica de la solicitud de balance, tanto el canal WS-RM en *Windows Communication Foundation* (WCF) como el envío expreso en MSMQ proporcionarán la confiabilidad necesaria si la solicitud no tiene que superar fallas del sistema. Si el mensaje debe superar un reinicio del sistema, la elección adecuada sería un envío recuperable MSMQ. Estos sistemas de mensajería podrán ocuparse del tiempo expirado y los reintentos necesarios para enviar el mensaje y la respuesta, entonces, el servicio del cajero automático no tiene que incluir esta lógica.

Ahora que comprendemos los problemas de confiabilidad con una solicitud de balance, continuemos con la solicitud de cambio de balance que sucede una vez que se ha otorgado el dinero. Los requisitos de confiabilidad para este mensaje son mucho mayores porque si no se envía con éxito, el banco le habrá dado dinero al cliente sin reducir su balance de cuenta. Si bien el cliente no se va a quejar, el banco no estará complacido si esta situación sucede con regularidad.

El mensaje de cambio del balance se debe enviar y procesar ante cualquier número de fallas del sistema y la red. El mecanismo de reintento basado en la memoria obviamente no es el adecuado para esta tarea ya que una falla del sistema perdería la copia del mensaje que se guarda para un nuevo intento. Conservar una copia permanente del mensaje para nuevos intentos podría también ser un problema ya que es posible que se envíen múltiples copias del mensaje a causa de los reintentos. Por ejemplo, si el mensaje disminuye del balance de cuenta del cliente USD 200.000, enviar el mensaje varias veces puede causar la disconformidad del cliente.

Los reintentos requeridos para asegurar un envío del mensaje confiable, funcionan sólo si los mensajes son idempotentes. Un mensaje *idempotente* puede enviarse cualquier cantidad de veces sin violar las restricciones comerciales de la aplicación. Algunos mensajes son idempotentes por naturaleza. Por ejemplo, un mensaje que cambia el domicilio del cliente puede ejecutarse múltiples veces sin causar efectos contraproducentes. Otros mensajes no son idempotentes por naturaleza, por lo tanto, el sistema de mensajería debe hacerlos idempotentes para prevenir el daño que pueda causar el mensaje cuando se procese más de una vez (Ver Figuras 2 y 3). Por lo general, esta transacción se logra conservando una lista de los mensajes que ya se han procesado; si el mensaje se recibe varias veces, no se procesará más de una vez.

La mayoría de los sistemas de mensajería no conservan una copia de los mensajes entrantes, pero el remitente asigna un identificador para cada mensaje y el receptor mantiene un registro de los identificadores que ha visto antes. Estos identificadores se deben conservar hasta que el destinatario del mensaje se asegure de que la fuente ha desechado el mensaje, porque si la fuente falla, el mensaje puede ser enviado nuevamente días más tarde cuando la fuente se reinicie. La lista de mensajes procesados también debe ser persistente ya que el servicio de destino puede fallar entre los reintentos. El destinatario también debe conservar un registro del mensaje enviado en respuesta al mensaje entrante ya que la fuente puede continuar enviando el mensaje original hasta que reciba una respuesta y el motivo de un nuevo intento tal vez sea que el mensaje de respuesta original se perdió.

Elegir una infraestructura de mensajes

A menos que esté dispuesto a insistir y registrar los mensajes en la lógica de su servicio, necesitará una infraestructura de mensajería confiable para brindar este nivel de confiabilidad. Microsoft cuenta con tres opciones para esta infraestructura: Mensajería recuperable MSMQ, Agente de Servicios del Servidor SQL y BizTalk. La opción que elija dependerá de los requisitos de confiabilidad y el diseño de la aplicación.

El Agente de Servicios y BizTalk proporcionan un almacenamiento de mensajes más confiable que MSMQ ya que los almacena en una base de datos del Servidor SQL mientras que MSMQ los almacena en un archivo. Si su aplicación puede funcionar con una perdida ocasional de mensajes cuando un archivo se pierde o es dañado, entonces MSMQ será el adecuado para sus necesidades. Algunas aplicaciones MSMQ evitan la pérdida potencial de mensajes almacenando una copia del mensaje en una base de datos. Si va a almacenar mensajes, es muchísimo más eficaz utilizar un Agente de Servicios que de todas formas almacena los mensajes en una base de datos.

El Agente de Servicios y el BizTalk proporcionan aproximadamente el mismo grado de confiabilidad y defensa contra la pérdida de mensajes ya que ambos almacenan mensajes en una base de datos. La gestión de mensajes en el Agente de Servicios forma parte de la base de datos de la lógica del servidor, por lo tanto, el Agente de Servicios se comunica directamente desde el proceso del Servidor SQL hacia el socket TCP/IP, lo que resulta mucho más eficaz que el método de BizTalk en el que un proceso externo llama a la base de datos para almacenar los mensajes en una tabla. Si bien el Agente de Servicios puede enviar de forma significativa muchos más mensajes por segundo que BizTalk, BizTalk ofrece una mayor cantidad de funciones –transformación del mensaje, enrutamiento de mensajes dependientes de datos, transporte de mensajes múltiples, organización y demás– que el Agente de Servicios no ofrece.

En general, si la transferencia confiable de mensajes entre las bases de datos es todo lo que necesita su aplicación, el Agente de Servicios es una mejor elección ya que es más liviano y eficaz que BizTalk en la transferencia de mensajes. Si en cambio su aplicación requiere de las funciones que

Figura 2: Reintento inocuo de petición de extracción de dinero

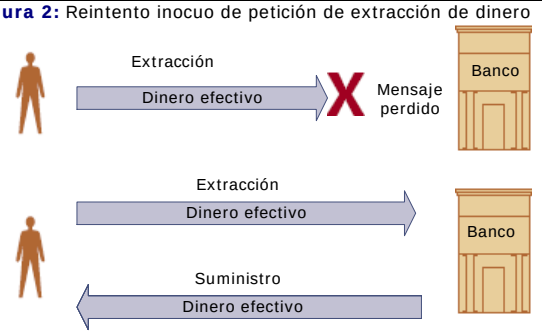
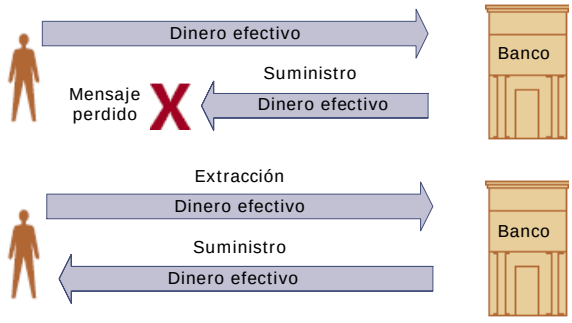


Figura 3: Reintento de petición de extracción en el que el cliente pierde dinero



proporciona el BizTalk como organización, integración de datos y manipulación de mensajes, entonces el Agente de Servicios probablemente no sea la solución adecuada.

Si bien la mensajería recuperable de MSMQ no proporciona los niveles de confiabilidad que proporcionan las opciones basadas en el Servidor SQL, posee la ventaja de no requerir el Servidor SQL en ambos puntos finales de la mensajería. Si el soporte de una base de datos en ambos puntos finales no es una opción y se pueden cumplir los requisitos de la aplicación por medio de MSMQ, entonces MSMQ es la elección correcta como infraestructura de mensajería. Si en cambio, los dos servicios de comunicación requieren el almacenamiento de datos, vale la pena la confiabilidad extra de almacenar los mensajes en una base de datos.

En nuestro ejemplo del cajero automático, el servicio del cajero automático requiere el almacenamiento local de datos para la auditoría y para las operaciones sin conexión y el almacenamiento de datos de referencia, por lo tanto, probablemente ya exista una base de datos en el cajero automático y una de las opciones basadas en el Servidor SQL es adecuada. La elección entre el Agente de Servicios y el BizTalk depende de los requisitos de la aplicación ajenos al envío de mensajes, los recursos disponibles en el cajero automático y los requisitos de acuerdo al volumen de mensajes.

Confiabilidad de ejecución

Anteriormente, analizamos la confiabilidad en el envío de mensajes desde un servicio hacia otro. Como es de esperar, descubrimos que la cantidad de confiabilidad requerida depende de lo que esté haciendo la aplicación y de lo importante que sea el mensaje de datos para la aplicación. Aquí supondremos que los mensajes se transfieren a un servicio con el grado de confiabilidad requerido y examinaremos los requisitos de confiabilidad necesarios para que el servicio procese los mensajes.

Uno de los requisitos exclusivos de un servicio que procesa mensajes asíncronos es que recibir un mensaje de una cola es un proceso de "extracción". En otras palabras, cuando un mensaje llega en cola, se situará allí hasta que la aplicación ejecute la operación "recibir" para recuperar y procesar el mensaje. Este requisito significa que un servicio asíncrono debe asegurar la ejecución cuando haya mensajes en cola para ser procesados. La forma más común de cumplir este requisito es hacer que el servicio sea un servicio de Windows administrado por el Administrados de Control de Servicios de Windows (SCM-*Windows Service Control Manager*). El SCM asegurará que el servicio se inicie cuando Windows se inicie y se puede configurar para reiniciar el servicio en caso de que falle por alguna razón.

Si bien por lo general esta configuración proporciona el nivel requerido de confiabilidad y es con frecuencia la solución preferida cuando los mensajes llegan en un ritmo constante, esto puede causar problemas si la carga del mensaje varía significativamente. Si el servicio está configurado con los recursos necesarios para administrar picos de cargas, será un desperdicio de recursos cuando la carga de mensajes sea baja; y si está configurado para administrar una carga promedio, se atrasará mucho durante las cargas pico. La mensajería de BizTalk se ejecuta como un servicio de Windows, por lo tanto, una aplicación BizTalk necesita del servicio de Windows para administrar los mensajes entrantes.

MSMQ trata el problema de carga de mensajes con desencadenadores que inician el servicio de procesamiento de mensajería cada vez que un mensaje llega a la cola. Si bien este proceso funciona bien cuando los mensaje llegan de forma ocasional, cuando la carga de mensajes es alta, la sobrecarga de iniciar miles de copias del servicio puede ser más que la lógica del servicio en sí mismo.

El Agente de Servicios proporciona una función llamada *activación* para resolver este problema. Cuando el mensaje llega a una cola vacía, el Agente de Servicios comenzará un proceso almacenado para tratar el mensaje. El proceso almacenado esperará en un bucle la llegada de más mensajes y continuará en este bucle hasta que la cola esté vacía. Si el Agente de Servicios determina que el proceso almacenado no está siguiendo los mensajes que llegan, iniciará copias adicionales del proceso almacenado hasta que haya suficientes copias para seguir los mensajes. Cuando el ritmo de llegada de mensajes disminuye, la cola se vaciará y se terminarán las copias extras. Entonces, siempre existirá aproximadamente la cantidad indicada de recursos disponibles para prestar servicio a los mensajes entrantes. Debido a que el Agente de Servicios inicia estos procesos, será notificado si uno falla y reemplazará la copia que falló. Si el servicio es una aplicación externa en vez de un proceso almacenado, el Agente de Servicios ofrece eventos a los que se puede suscribir una aplicación externa que le avisarán al servicio si se necesitan más recursos para procesar los mensajes que se encuentran en la cola.

Mensajes perdidos

El otro problema de confiabilidad que tiene que resolver la ejecución del servicio es la falla del servicio mientras se procesa el mensaje. Si el servicio borra un mensaje de la cola tan pronto como lo recibe y luego falla antes de procesar el mensaje completamente, este mensaje está realmente perdido. De igual manera, si el servicio espera hasta que haya procesado completamente el mensaje antes de eliminarlo de la cola, una falla en entre el paso de procesamiento y la eliminación del mensaje daría como resultado que el mensaje siga aún esperando en la cola cuando el servicio se reinicie y que sea procesado nuevamente.

Tal como lo mencionamos anteriormente, procesar el mensaje múltiples veces no representa un problema si el mensaje se refiere a una consulta de saldo, pero procesar un retiro de dinero puede ser molesto para el cliente implicado. La única forma de asegurar que el mensaje sea procesado "exactamente una vez" es que el procesamiento del mensaje y la eliminación de la cola sean parte de la misma transacción. Si existiera una falla en el procesamiento, tanto los cambios del procesamiento como la eliminación del mensaje serían reanudados, por lo tanto, todo volvería a la forma en la que estaba antes de que se recibiera el mensaje.

Una operación "confirmar" simple hace que se lleve a cabo tanto la acción de recibir el mensaje como la acción de procesarlo. De

igual modo, si al procesar el mensaje se genera un mensaje saliente, el "envío" del mensaje de salida debe ser parte de la transacción para evitar que el servicio se reanude pero el mensaje de respuesta aún se envíe. Este tipo de procesamiento del mensaje se denomina *mensajería transaccional*. La mayoría de las infraestructuras de mensajería confiable admiten esta mensajería.

Debido a que los mensajes se almacenan en un archivo diferente al de la base de datos, la mensajería transaccional MSMQ requiere un compromiso de dos fases para asegurar que ambas partes de la transacción estén confirmadas. Ya que los comandos ENVIAR y RECIBIR del Agente de Servicios son comandos TSQL, las operaciones de actualización de datos y mensajería en un servicio de Agente de Servicios puede ejecutarse desde la misma conexión del Servidor SQL y ser parte de la misma transacción del Servidor SQL. Por lo tanto, no es necesario un compromiso de dos fases, lo que hace que la implementación del Agente de Servicios de la mensajería transaccional sea mucho más eficaz que la implementación de MSMQ.

También, la mensajería transaccional es necesaria para hacer que un servicio no idempotente se comporte como un servicio idempotente con el fin de eliminar los problemas causados por los reintentos del mensaje. Si el servicio es idempotente por naturaleza, entonces la mensajería transaccional no es necesaria. Si la mensajería transaccional no se utiliza, el usuario del servicio debe estar preparado para recibir respuestas múltiples, a veces durante un período muy largo.

Confiabilidad de datos

Ahora veremos el impacto de los datos sobre la confiabilidad del servicio. La mayoría de los servicios acceden a datos mientras procesan los mensajes del servicio, por lo tanto, la confiabilidad de los datos está estrechamente vinculada a la confiabilidad del servicio.

Uno de los aspectos exclusivos de la ejecución de un servicio asíncrono es que en varios casos los mensajes del servicio representan objetos de negocios valiosos. Por ejemplo, en nuestro servicio del cajero automático, si se pierden los mensajes del balance debido a una falla, el balance de la cuenta no cambia y el banco pierde dinero. Por este motivo, tiene mucho sentido almacenar los mensajes en una base de datos para que puedan gozar de las mismas protecciones de disponibilidad, repetición y confiabilidad de las que gozan el resto de los datos almacenados allí. Si los mensajes de cambio de balance de nuestro ejemplo, se almacenan en la misma base de datos que las cuentas, sólo se perderían si se pierden las cuentas. Las funciones de copias de seguridad, copias de seguridad de registros y la Red de Área de Almacenamiento (SAN-*Storage Area Network*) que se utilizan para asegurar que no se pierda la información de las cuentas del banco, también se aplican para los mensajes de cambio de balance haciendo que la confiabilidad sobre el servicio sea extremadamente alta. Si sus requisitos de confiabilidad del mensaje son altos, el Agente de Servicios o el BizTalk poseen importantes ventajas de confiabilidad ya que almacenan los mensajes en una base de datos.

Una de las funciones nuevas del Servidor SQL 2005 que puede mejorar la confiabilidad del servicio es la copia espejo de la base de datos (DBM). DBM ofrece confiabilidad ya que conserva una copia secundaria de una base de datos que se mantiene consistente de forma transaccional con la base de datos primaria al aplicar a la copia secundaria cada transacción que se confirma sobre la primaria antes de devolver el control al servicio. Si la base de datos primaria falla, la copia secundaria puede convertirse en la primaria en pocos segundos.

El Agente de Servicios aprovecha la copia espejo de la base de datos para mejorar la confiabilidad de la mensajería. Si la base de datos de la cuenta es par de la base de datos DBM, la base de datos del Agente de Servicios en el cajero automático abrirá conexiones de red tanto hacia la base de datos primaria como hacia la secundaria y enviará mensajes a la primaria. Si la base de datos secundaria se convierte en la primaria, se notifica inmediatamente al Agente de Servicios y los mensajes se enrutarán a la nueva primaria sin la intervención ni interrupción del usuario.

Los servicios que requieren un gran volumen de datos pueden aprovechar otras funciones del Servidor SQL 2005 para mejorar la confiabilidad. La integración del Tiempo de Ejecución en Lenguaje Común (CLR) dentro del Servidor SQL significa que la lógica del servicio puede también ejecutarse dentro de la base de datos. Por lo tanto, para un servicio del Agente de Servicios la lógica, mensajes, entorno de ejecución, contexto de seguridad y datos para el servicio pueden estar en la misma base de datos.

Este almacenamiento de ubicación única posee varias ventajas en un sistema con requisitos de confiabilidad altos. Por lo general, los servidores de las bases de datos poseen funciones de software y hardware para mantener la alta confiabilidad que requiere la base de datos. Esta confiabilidad puede ahora aplicarse a todos los aspectos de la implementación del servicio. En el improbable evento de una falla del servicio, todo el entorno del servicio puede restablecerse a un estado transaccionalmente consistente con las funciones de restablecimiento de la base de datos. No sólo se guarda la base de datos, sino que también cualquiera de las operaciones en progreso se reinicia y se reanuda en el mismo entorno de seguridad y ejecución en el que se encontraba en el momento de la falla.

La naturaleza asincrónica de acoplamiento leve, de las aplicaciones orientadas al servicio imponen algunos requisitos de confiabilidad exclusivos. Al diseñar servicios, el nivel de confiabilidad requerido para el servicio debe considerarse y comprenderse. Microsoft ofrece una variedad de infraestructuras para la implementación de servicios que brindan diferentes niveles de confiabilidad. La elección de la infraestructura correcta implica hacer coincidir el nivel de confiabilidad requerido con las capacidades de la infraestructura. Las nuevas funciones del Servidor SQL 2005 proporcionan una infraestructura de alojamiento del servicio que ofrece niveles de confiabilidad sin precedentes para los servicios que requieren niveles de confiabilidad muy altos. •

Recursos

Arquitectura Orientada al Servicio
http://msdn.microsoft.com/architecture/soa

"Introducción al Servidor SQL 2005 para los desarrolladores de bases de datos", Matt Nunn (Microsoft Corporation, 2005)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/sql_ovyukondev.asp

"Construir Aplicaciones de bases de datos asincrónicas y confiables utilizando un Agente de Servicios", Roger Wolter (Microsoft Corporation, 2005)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/sql2k5_SrvBrk.asp

Revista MSDN
.NET Distribuido – "Aprenda los procedimientos básicos de la programación de Windows Communication Foundation", Aaron Skonnard (Microsoft Corporation, 2006)
http://msdn.microsoft.com/msdnmag/issues/06/02/WindowsCommunicationFoundation/

Sistema de Servidor de Windows de Microsoft – Servidor BizTalk de Microsoft
www.microsoft.com/biztalk/

Sobre el autor

Roger Wolter es arquitecto de soluciones en el equipo de Estrategia de Arquitectura de Microsoft. Roger tiene 30 años de experiencia en varios aspectos de la industria informática incluyendo trabajos en Unisys, Infospan, Fourth Shift y los últimos siete años como Administrador de Programas en Microsoft. Sus proyectos en Microsoft incluyen SQLXML, SOAP Toolkit, Agente de Servicios de Servidor SQL y el Servidor SQL Express. Su interés en el Agente de Servicios se produjo a partir de un sistema de fabricación basado en la mensajería sobre el cual trabajó en una etapa anterior de su vida. También escribió *The Racional Guide to Servidor SQL 2005 Service Broker Beta Preview* (Editorial Rational, 2005).

Modelo flexible para la integración de datos

Por Tim Ewald y Kimberly Wolk

Síntesis

Los arquitectos y los desarrolladores deben enfrentar muchos desafíos en la integración de los sistemas, y la industria ha puesto su atención en XML, los Servicios de Web y SOA para resolver los problemas de integración concentrándose en protocolos de comunicación, en particular en lo que respecta al agregado de características avanzadas que soportan el flujo de mensajes en las topologías de red complejas. Sin embargo, esta concentración en protocolos de comunicación ha quitado la atención de los problemas de integración de datos. Los modelos flexibles de combinación de datos entre los sistemas dispares son esenciales para la integración exitosa. Estos modelos están expresados en el esquema XML (XSD) en los sistemas basados en los servicios de Web, y los casos del modelo están representados como XML transmitido en los mensajes SOAP. En nuestro trabajo sobre la arquitectura del Sistema de Publicación TechNet MSDN (MTPS) tratamos tres dificultades. Veremos cuáles son estas dificultades y sus soluciones en el contexto de un problema más general, la integración de información del cliente.

La integración de los sistemas presenta una gran variedad de desafíos para los arquitectos y desarrolladores. A lo largo de varios años, la industria se ha concentrado en el uso de XML, Servicios de Web y arquitectura orientada al servicio (SOA) para resolver los problemas de integración. Mucho del trabajo realizado en este espacio se ha concentrado en los protocolos de comunicación, en especial sobre el agregado de características de avanzada diseñadas para soportar mensajes que fluyen en topologías de red complejas. Si bien indudablemente existe algún valor en este enfoque, todo el trabajo sobre los protocolos de comunicación ha quitado la atención del problema de integración de datos.

Contar con modelos flexibles para combinar datos entre los distintos sistemas es esencial para lograr una integración exitosa. En los sistemas basados en servicios de Web, estos modelos se expresan en XSD. Los casos del modelo se representan como un XML que se transmite entre los sistemas en los mensajes SOAP. Algunos sistemas distribuyen los datos XML dentro de las bases de datos relacionales; algunos no. Desde una perspectiva de integración, la estructura de estos modelos de bases de datos relacionales, no es importante. Lo que importa es la forma del modelo de datos XML definido en XSD.

Existen tres dificultades en las que por lo general se agrupan los proyectos de integración de datos basados en los servicios de Web. Las tres se relacionan con la forma en la que definen sus esquemas XML. Afrontamos las tres en nuestro trabajo sobre la arquitectura del Sistema de Publicación TechNet MSDN (MTPS), el sistema basado en el XML de última generación que sirve como base para MSDN2. analizaremos nuestras soluciones en el contexto de integración de información del cliente.

El problema esencial de integración de datos

Imaginemos que trabaja para una empresa grande. Su empresa posee varios sistemas orientados a terceros que los usuarios utilizan para realizar una variedad de tareas. Por ejemplo, un servicio brinda información personalizada de un producto para usuarios registrados que han expresado un interés particular. Otro sistema proporciona herramientas de administración de membresía para clientes que pertenecen al programa de socios. Un tercer sistema, realiza un seguimiento de usuarios que se han registrado para asistir a los próximos eventos. Desafortunadamente, todos los sistemas se

"CONTAR CON MODELOS FLEXIBLES PARA COMBINAR DATOS ENTRE LOS DISTINTOS SISTEMAS ES ESENCIAL PARA LOGRAR UN ESFUERZO DE INTEGRACIÓN EXITOSO."

desarrollaron de forma separada, uno de ellos fue desarrollado por una por una empresa diferente que su empresa adquirió un año atrás. Cada sistema almacena información relacionada con el cliente en diferentes formatos y ubicaciones.

Esta configuración presenta un problema crítico para el negocio: no posee un vista unificada de un cliente determinado. Este problema tiene dos efectos. Primero, la experiencia del cliente se ve afectada ya que la única empresa con la que están haciendo negocios los trata como personas diferentes cuando utilizan sistemas diferentes. Por ejemplo, un cliente que desea recibir información por correo electrónico acerca de un producto determinado cada vez que esté disponible, deberá expresar su interés nuevamente por el producto cuando se registre para charlas de un próximo evento patrocinado por la empresa. Su experiencia estaría más integrada si el sistema que lo registra para un próximo evento ya conoce sus intereses respecto de un producto en particular.

Segundo, el negocio se ve afectado porque no posee una comprensión integrada de sus clientes. ¿Cuántos clientes que son miembros del programa de socios también reciben información por correo electrónico sobre productos? En ambos casos, las divisiones entre los sistemas con los cuales trabaja el cliente están limitando la capacidad de la empresa para responder a las necesidades de sus clientes.

No tiene importancia si esta situación surgió porque los sistemas se diseñaron y desarrollaron de forma individual sin tener en cuenta el contexto mayor dentro del cual operan; o porque diferentes grupos de sistemas integrados fueron recopilados a través de fusiones y adquisiciones. El problema continua: la empresa debe integrar los sistemas tanto

para mejorar la experiencia del cliente así como también su conocimiento respecto de la identidad del cliente y la mejor forma de satisfacerlo.

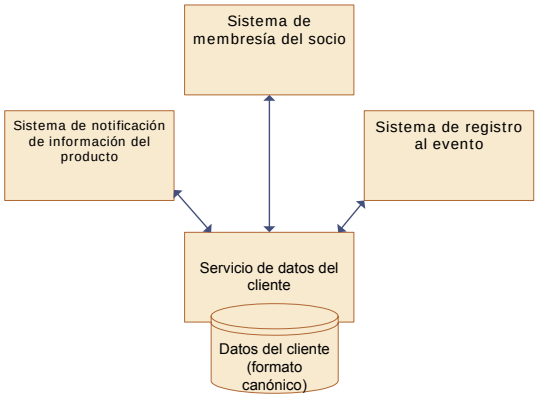
La técnica más común para resolver este problema es exigir que todos los sistemas adopten un único sistema canónico para un cliente. Un grupo de arquitectos se reúne y diseña un formato único de la empresa para representar los datos del cliente como XML. El formato se define utilizando un esquema escrito en XSD. Para permitir que los sistemas compartan los datos en el nuevo formato, un equipo central construye un nuevo almacenamiento que lo soporta. El equipo de almacenamiento y el equipo del modelo de datos XSD proporcionan su solución a todos los equipos responsables de los sistemas que interactúan de alguna forma con el cliente y exigen su adopción. El cambio esencial se muestra en las Figuras 1 y 2.

Cada sistema se modifica para utilizar el almacenamiento de datos del cliente subyacente a través de su interfaz del servicio de Web, y almacenan y recuperan la información del cliente como XML compatible con el esquema del cliente. Todos los sistemas comparten la misma instancia del servicio, el mismo modelo de datos XSD y la misma información XML.

Esta solución parece ser simple, elegante y buena, pero una implementación inexperta fallaría por alguno de estos tres motivos: demanda de demasiada información, estrategia de versionado no efectiva y sin soporte para la extensión de nivel de sistema.

Figura 1: Tres almacenamientos de datos por separado, uno por sistema

Figura 2: Formato y almacenamientos de datos único



Demanda de demasiada información

La primera posible causa de falla es un esquema y un almacenamiento que requieren mucha información. Cuando las personas construyen un servicio de Web para la integración punto-a-punto, tienden a pensar en los datos que necesita su servicio particular. Definen un contrato que requiere el suministro de estos datos particulares. Cuando se genera un contrato desde un código fuente, estos datos pueden ocurrir de forma implícita. La mayoría de las herramientas que mapean desde un código fuente hacia un contrato de servicio de Web consideran áreas de tipos de valor simple como elementos de datos requeridos, insistiendo en que el cliente los envíe. Incluso cuando se crea un contrato a mano, existe una tendencia a considerar todos los datos como necesarios. Tan pronto como el servicio determina (por medio de código o validación del esquema) que falta algún dato necesario, rechaza la solicitud. El cliente obtiene una falla del servicio.

Este enfoque para definir los contratos de servicios de Web es muy rígido y lleva a sistemas que están estrechamente acoplados. Cualquier cambio en los requisitos del servicio obliga a un cambio en el contrato y en los clientes que lo consumen. Para deshacer este vínculo es necesario separar la definición de la forma de datos que espera un servicio, de los requisitos de proceso en curso de un servicio. Más concretamente, los formatos de datos definidos por el contrato deben considerar todo como opcional más allá de los datos de identidad. La implementación del servicio debe imponer los requisitos en la ocurrencia de forma interna en el tiempo de ejecución (ya sea utilizando un código o un esquema de validación dedicado). Debe ser lo más tolerante posible cuando los datos no están presentes en una solicitud de cliente y degradan lentamente.

En el ejemplo de información del cliente, es fácil pensar en los casos en los que algunos sistemas desean trabajar con clientes pero no tienen disponible la información del cliente completa. Por ejemplo, el sistema que registra los intereses del cliente en un producto particular tal vez sólo recopile una dirección de correo electrónico preferida y un nombre del cliente. En cambio, el sistema de registro del evento también puede capturar información sobre su tarjeta de crédito y domicilio. Si un modelo de datos de cliente común requiere que cada registro de cliente válido incluya información sobre la tarjeta de crédito, domicilio, correo electrónico y nombre, el sistema no puede adoptarlo sin recopilar más datos de los necesarios ni proporcionando datos falsos. Hacer que todos los datos, más allá de los de identidad (número de ID, dirección de correo electrónico y demás) sean opcionales, facilita la adopción del modelo de datos ya que el sistema puede simplemente brindar la información que posee.

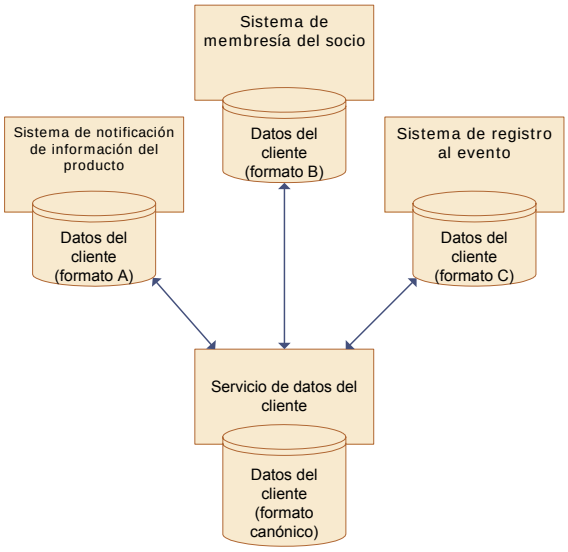
Al separar la forma de los datos de los requisitos de la ocurrencia, se facilita la gestión de cambios en la implementación de un servicio único. Esto también es fundamental cuando se define un esquema XML común para ser utilizado por múltiples servicios y clientes. Si mucha información es obligatoria, cada sistema que desee utilizar el modelo de datos puede omitir alguna información necesaria. Esto deja a cada sistema con la opción de no adoptar el almacenamiento y modelo compartido o proporcionar datos falsos (por lo general el valor predeterminado de un tipo de lenguaje de programación simple). Cualquier opción puede considerarse una falla.

Se logra mucha flexibilidad para que los sistemas adopten el modelo si se flexibilizan los requisitos de ocurrencia del esquema casi completamente. Cada sistema puede aportar tanta información como tenga disponible lo que hace que un esquema XML común sea mucho más fácil de adoptar. El costo es que los sistemas que reciben datos deben ser cuidadosos al verificar que los datos que realmente necesitan estén presentes. Si no los tienen, deben responder en consecuencia obteniendo más datos del usuario o algún otro almacenamiento, degradando su comportamiento o –en el peor de los casos– generando una falla. Lo que realmente se hace es cambiar algunas de las restricciones que por lo general uno pone en un esquema XML dentro de su código, en el que se verificarán en tiempo de ejecución. Este cambio le permite modificar esas restricciones sin revisar el esquema compartido.

Estrategia de versionado no efectiva

La segunda causa potencial de falla es la falta de una estrategia de versionado. No importa la cantidad de tiempo y esfuerzo que se invierta en definir un esquema XML desde el comienzo,

Figura 3: Combinación de almacenamientos (Ver Figuras 1 y 2)



será necesario cambiarlo con el tiempo. Si el esquema, el almacenamiento que lo soporta y cada sistema que los utiliza deben migrar a una nueva versión de una vez, no se podrá obtener éxito. Algunos sistemas deberán esperar por cambios necesarios ya que otros sistemas no se encuentran en el punto en el que pueden adoptar una revisión. Por el contrario, algunos sistemas se verán forzados a realizar un trabajo extra, inesperado, ya que otros sistemas necesitan adoptar una nueva revisión. Este enfoque es insostenible.

Para resolver este problema es necesario adoptar una estrategia de versionado que permita que el esquema y el almacenamiento avancen independientemente del ritmo al que los otros sistemas adopten sus revisiones. Esta solución parece simple y lo es, siempre y cuando se consideren los esquemas XML como el camino correcto.

Los sistemas que integran utilizando un esquema XML común ven esto como un contrato. Reducir las exigencias para los datos requeridos al hacer que los elementos sean opcionales permite que un contrato sea más fácil de concretar ya que los sistemas se comprometen menos. Para versionar, también es necesario que se permita que los sistemas realicen más *sin cambiar el espacio del nombre del esquema*. En términos prácticos esto significa que los sistemas siempre deben producir datos XML basados en la versión del esquema con el que fueron desarrollados. Siempre debe consumir datos basado en esta misma versión *con información adicional*. Esta definición es una variación de la Ley de Postel: "Sé liberal en lo que aceptas de otros, sé conservador en lo que haces". Casi indiscutiblemente, esta idea fundamenta todas las tecnologías exitosas de los sistemas distribuidos y por supuesto todas las de acoplamiento leve. Si se toma este enfoque, entonces se puede extender un sistema sin actualizar los clientes.

En el ejemplo del cliente, una actualización para el esquema y el almacenamiento puede agregar soporte para un elemento opcional adicional que captura el apellido de soltera de la madre del usuario por razones de seguridad. Si los sistemas que trabajan con versiones anteriores generan registros del cliente sin esta información, está bien porque el elemento es opcional. Si envían estos registros a otros sistemas que necesitan esta información, la solicitud puede fallar y esto también está bien. Si los sistemas nuevos envían datos del cliente a sistemas anteriores incluyendo el apellido de soltera de su madre, esto también está bien ya que están diseñados para ignorarlos.

Afortunadamente, varios juegos de herramientas de servicios de Web admiten esta función directamente en su extracción de datos en base a un esquema. Por supuesto, los mapeadores de objetos .NET XML (tanto el fiable XmlSerializer como el nuevo XmlFormatter/ DataContract) administran los datos extra correctamente. Algunos juegos de herramientas de Java también lo hacen y los sistemas que admiten las especificaciones JAXB 2.0 y JAX-WS 2.0 nuevas, también. Dado esto, adoptar este enfoque es demasiado fácil.

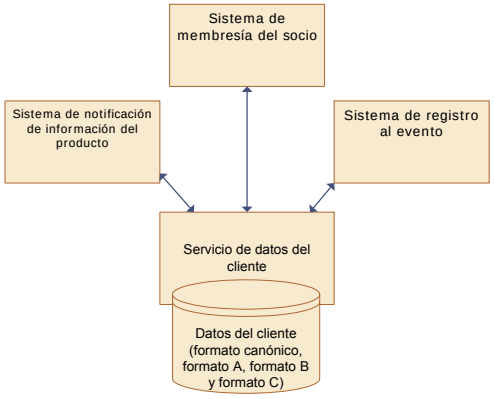
El único problema real con este modelo es que presenta definiciones múltiples de un esquema determinado, cada una representando una versión diferente. Dado un fragmento de datos –por ejemplo, un fragmento XML capturado de un mensaje enviado en línea– es imposible responder la pregunta: "¿Son válidos estos datos?". La pregunta sobre la validez sólo puede responderse en relación a una versión particular del esquema. La incapacidad para establecer definitivamente si un fragmento determinado de datos es válido presenta un problema para la depuración y posiblemente también para la seguridad. Si los modelos de datos se describen con el esquema XML, es posible responder una pregunta diferente y más interesante: "¿Son estos datos lo suficientemente válidos?".

Esta pregunta es realmente por lo que se preocupan los sistemas, y se puede responder esta pregunta usando la información de validez que proporcionan la mayoría de los validadores de esquemas basados en XML, que representa un camino razonable a seguir en los casos en los que se requiere la validez del esquema y se puede implementar con lo sistemas de validación de esquemas de la actualidad.

Sin soporte para la extensión de nivel de sistema

La tercera posible causa de falla es la falta de soporte para extensiones específicas del sistema para un esquema. La estrategia de versionado basada en la noción de que una definición del esquema cambia con el tiempo es necesaria para promover la adopción, pero no es suficiente. Si bien libera a los sistemas de tener que adoptar inmediatamente las últimas revisiones del esquema, no hace nada para ayudar a los sistemas que esperan actualizaciones específicas del esquema. Las demoras en las revisiones también pueden hacer que un esquema sea muy costoso para que lo adopte un sistema. La solución para este último problema es permitir que los sistemas adopten un esquema común para extenderlo con su propia información adicional. La información de la extensión puede almacenarse de forma local en un almacén al nivel del sistema (Ver Figura 3).

Figura 4: El mismo almacenamiento (ver figura 3) con formatos de datos en depósito compartido



En este caso se modifica cada sistema para escribir los datos del cliente tanto para su almacén dedicado utilizando su propio modelo de datos así como también para el almacén compartido usando el esquema canónico. También se modifica para leer los datos del cliente desde el almacén dedicado y desde el almacén compartido. En función de lo que encuentre, sabe si un cliente ya es conocido para la empresa y para el sistema. La Tabla 1 resume las tres posibilidades.

El sistema puede usar esta información para decidir la cantidad de información que necesita reunir sobre un cliente. Si el cliente es nuevo para la empresa, el sistema agregará la mayor cantidad de información posible al almacén canónico. La información se pone a disposición de otros sistemas que trabajan con los clientes. También puede almacenar datos en su almacén dedicado para satisfacer sus propias necesidades. Este modelo puede expandirse más para que los datos específicos de sistema se almacenen también en el almacén compartido (Ver Figura 4).

Esta solución posibilita que los sistemas se integren entre ellos utilizando datos de extensión que están más allá del alcance del esquema canónico. Para funcionar exitosamente, es necesario que el almacén y otros sistemas se dejen ver dentro de los datos de extensión. En otras palabras, no puede ser opaco. La forma más fácil de resolver este problema es hacer los mismos datos de extensión XML. El sistema que proporciona los datos define un esquema para los datos de extensión de modo que los otros sistemas puedan procesarlos de forma confiable. El almacén compartido mantiene un seguimiento de los esquemas de extensión para asegurar que los datos de extensión sean válidos, aún si no conoce de forma explícita lo que contiene la

"LA INCAPACIDAD PARA ESTABLECER DEFINITIVAMENTE SI UN FRAGMENTO DETERMINADO DE DATOS ES VÁLIDO PRESENTA UN PROBLEMA PARA LA DEPURACIÓN Y POSIBLEMENTE TAMBIÉN PARA LA SEGURIDAD."

extensión. En el caso más extremo, un sistema puede elegir un registro de cliente completo en un formato específico de sistema como datos de extensión XML. Otros sistemas que comprenden este formato pueden utilizarlo. En cambio, los sistemas que no lo comprenden dependen de la representación canónica.

Cuando los sistemas son independientes, cada uno controla su propio destino. Pueden capturar y almacenar cualquier información que necesiten en cualquier formato y en la ubicación que prefieran. El traslado a un único depósito y esquema común cambia esto. Si adoptar un formato de datos XML común limita la libertad del sistema para brindar la funcionalidad requerida, está predestinado al fracaso desde el comienzo.

Utilizar una combinación de datos de extensión de XML capturado manualmente tanto en un nivel de sistema como en un almacén compartido agrega complejidad ya que deberá mantener los datos sincronizados. Pero también proporciona una gran flexibilidad. Puede alinear los sistemas en torno a cualquier combinación de esquemas canónicos o esquemas alternativos que desee. Puede orientarse hacia un formato XML con el tiempo, pero siempre contará con la flexibilidad para desviarse de este formato y cumplir con nuevos requisitos. Esta libertad extra tiene mucho valor en el mundo incierto de la empresa.

Un beneficio sutil, adicional de este modelo es que permite que el equipo que define el esquema común disminuya su trabajo sobre revisiones. Los sistemas pueden utilizar datos de extensión para cumplir con requisitos nuevos entre revisiones. El equipo que trabajo sobre el modelo canónico puede extraer estas extensiones para insertarlas dentro del proceso de revisión. Este ciclo de retroalimentación ayuda a asegurar que los cambios del modelo estén orientados por requisitos del sistema real.

Tabla 1: Los tres casos posibles de datos del cliente

Registro en almacén compartido	Registro en almacén del sistema	Significado
No	No	El cliente es nuevo para la empresa. Recopila todos los datos comunes y específicos de sistema.
Si	No	El cliente es conocido por la empresa pero nuevo para el sistema. Recopila todos los datos específicos de sistema.
Si	Si	El cliente es conocido por la empresa y por el sistema.

Disminuir el riesgo

Muchas organizaciones están trabajando sobre la integración de sistemas usando datos XML que se describen utilizando un esquema XML y se intercambian por medio de servicios de Web. En este análisis presentamos tres causas comunes para el fracaso de estos proyectos de integración centrados en datos: demanda de demasiada información, estrategia de versionado no efectiva y sin soporte para la extensión de nivel de sistema. Para disminuir estos riesgos debe:

- Hacer los elementos del esquema opcionales y codificar los requisitos de ocurrencias específicos del sistema como parte de esta implementación del sistema.
- Construir sistemas que produzcan datos conforme a su versión de un esquema compartido pero que consuman de forma tal que los sistemas puedan adoptar revisiones del esquema a diferentes ritmos sin cambiar el espacio del nombre del esquema.
- Permitir que los sistemas extiendan esquemas compartidos con sus propios datos para satisfacer nuevos requisitos independientemente de las revisiones del modelo de datos.

Todas estas soluciones se basan en una idea fundamental: integrar exitosamente sin sacrificar la agilidad que los sistemas necesitan para ser capaces de coincidir en la menor medida posible y aún así hacer que las cosas funcionen. Entonces ¿Funciona todo esto? La respuesta es sí. Estas técnicas son fundamentales para el diseño de Sistemas de Publicación TechNet/MSDN, que es la base de MSDN2. •

Recursos

Revista MSDN
http://msdn.microsoft.com/msdnmag/05/02/insideMSDN/

Biblioteca MSDN2
http://msdn2.microsoft.com/en-us/library/

Sobre los autores

Tim Ewald es arquitecto principal en Foliage Software Systems donde ayuda a los clientes a diseñar y construir aplicaciones que abarcan desde IT empresarial hasta dispositivos médicos. Antes de unirse a Foliage, Tim trabajó en Mindreef, un proveedor líder de herramientas de diagnóstico de servicios Web y antes de esto Tim fue Gerente de Programa principal en MSDN, donde trabajó junto con Kim Wolk como co-arquitecto de MTPS, el motor de publicación basado en servicios de Web y XML detrás de MSDN2. Tim es un orador y autor internacionalmente reconocido.

Kim Wolk es gerente de desarrollo en MSDN y la fuerza orientadora detrás de MTPS, el motor de publicación basado en servicios Web y XML detrás de MSDN2. Anteriormente trabajó como desarrolladora líder y co-arquitecta de MTPS. Antes de unirse a MSDN, Kim pasó varios años como consultora, tanto en forma independiente así como también para los Servicios de Consultoría de Microsoft, donde trabajó sobre una gran variedad de sistemas empresariales de importancia crítica.

Servicios autónomos y agregación de entidades empresariales

Por Udi Dahan

Síntesis

Las empresas actualmente dependen de aplicaciones y sistemas heterogéneos para funcionar. Cada uno de estos sistemas administra sus propios datos, y generalmente no lo exponen en forma explícita para el consumo externo. Muchos de estos sistemas dependen de los mismos conceptos básicos como cliente y empleado y, como consecuencia, se definieron estas entidades en muchos lugares de forma levemente diferente. La agregación de entidades representa la necesidad comercial de obtener una visión completa de esas entidades en un solo lugar. Sin embargo, esta necesidad comercial solamente es un síntoma de un problema más importante: la coordinación informática/comercial. Las arquitecturas orientadas al servicio (SOAs) fueron aclamadas como mecanismo de unión de la informática con la actividad comercial, pero el entusiasmo ya se empieza a desvanecer. Veremos las formas concretas de utilizar los servicios autónomos para transformar el modo en que desarrollamos los sistemas a fin de que coincidan más con los procesos comerciales y para cumplir con las necesidades inmediatas de agregación de entidades.

El término SOA ha ganado popularidad durante el último año y se convirtió en la palabra de moda. Todo actualmente está "orientado al servicio", "activado para SOA", o "es la clave para su éxito de SOA". La industria sigue su lucha respecto a lo que define a un servicio; no obstante, diversas propiedades de servicios parecen tener buena aceptación. Los principios de Microsoft de orientación al servicio definen cuatro propiedades: los servicios son autónomos; los servicios tienen límites explícitos; los servicios comparten el contrato y esquema, no la clase o tipo; y la compatibilidad del servicio se basa en la política.

Aunque estos principios son un lineamiento de arquitectura para desarrollo de software complejo, evidentemente hay más por decir. Muchos aspectos del tiempo de ejecución de los servicios como escalabilidad, disponibilidad y fortaleza no son considerados por la orientación al servicio. Precisamente estos aspectos de tiempo de ejecución son el foco de los servicios autónomos.

Autonomía del servicio

La frase "servicios autónomos" parece a simple vista reformular con otras palabras el primer principio, pero ambos tienen en realidad significados diferentes. "Los servicios son autónomos" significa que los grupos que desarrollan servicios cooperativos pueden operar en forma independiente, a un cierto grado. Cuando se los considera juntos con el principio sobre contrato y esquema, es evidente que no es necesario que haya dependencias binarias entre esos grupos. El desarrollo de cada servicio se puede realizar en plataformas diferentes, utilizando distintos idiomas y herramientas. Un servicio autónomo, por otra parte, es un servicio cuya capacidad de funcionamiento no es controlada ni inhibida por otros servicios.

La palabra *autónomo* tiene muchas definiciones, entre ellas: autogobierno, autocontrol, independiente, autocontenido, libre de control y limitación externos. A la luz de estas definiciones de autonomía, analizaremos dos tipos de interacción de servicios: comunicación sincrónica y asincrónica.

En la Figura 1 podemos ver que el Servicio A necesita mantener activamente recursos de tiempo de ejecución (la cadena de llamada) hasta que responda el Servicio B. El tiempo que le toma al Servicio A responder a un solo pedido depende de su interacción con el Servicio B. El Servicio A se ve afectado si la red está lenta o si el Servicio B no está disponible o está lento. Por lo tanto, no parece que el Servicio A esté "libre de limitaciones externas" en este caso.

Otro tema a considerar es el acoplamiento. Aunque el Servicio A y el B puedan acoplarse en forma leve con respecto a que fueron desarrollados por separado por diferentes grupos en diferentes plataformas y compartiendo solamente un archivo WSDL, podemos ver que están acoplados en forma firme y a tiempo. Este acoplamiento temporal puede verse en el hecho de que el tiempo que le lleva al Servicio A responder a un pedido incluye el tiempo de procesamiento del Servicio B. Es precisamente este acoplamiento lo que causa fallas no deseadas en el Servicio B para repercutir en el Servicio A.

"MUCHOS ASPECTOS DE TIEMPO DE EJECUCIÓN DE LOS SERVICIOS COMO LA ESCABILIDAD, DISPONIBILIDAD, Y FORTALEZA, NO SON CONSIDERADOS POR LA ORIENTACIÓN AL SERVICIO".

Hay dos maneras en que podemos quebrar este acoplamiento temporal. Una forma es que el Servicio A sondee al Servicio B para ver los resultados. Lamentablemente, los sondeos causan a una carga errónea sobre el Servicio B (como una función de cantidad de consumidores y el intervalo del sondeo) y los consumidores obtienen la información solicitada después de que ya pasó el tiempo disponible. La Figura 2 muestra la complejidad de esta solución.

Si seguimos nuestro análisis, encontraremos que iniciar una nueva cadena, o incluso utilizar un sondeo de cadena para manejar el sondeo para cada pedido que enviamos, le quitará al Servicio A sus recursos con mucha rapidez. Una solución más potente (y aun más compleja) implicaría una cadena simple que administre el sondeo para todos los pedidos enviados. Esa cadena ordenará los resultados a medida que están disponibles para una cadena diferente, lo cual finalizaría el procesamiento del pedido

Figura 1: Comunicación sincrónica

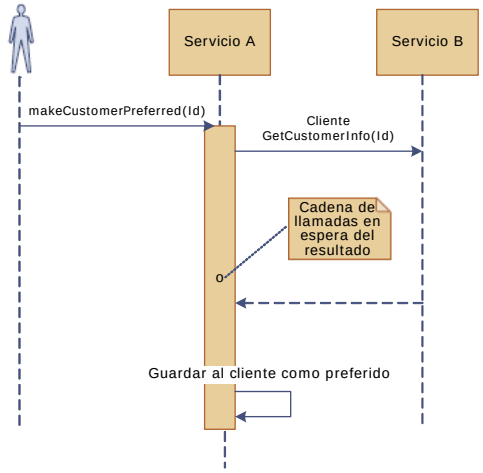
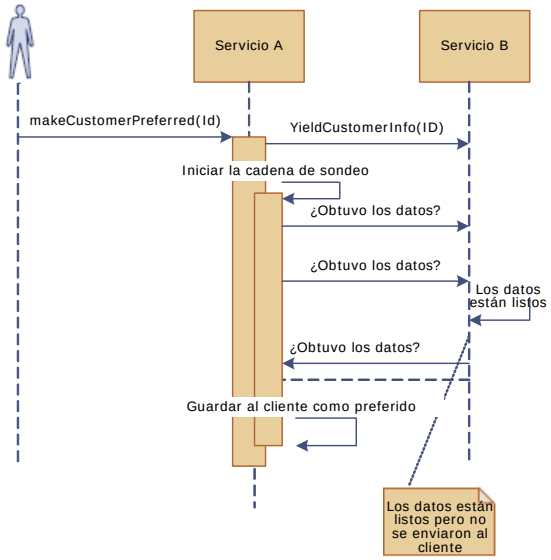


Figura 2: Comunicación sincrónica con sondeo



original. Bien podríamos reconocer a la navaja de Occam antes de continuar por este camino.

Una solución diferente para el problema del acoplamiento temporario que evita los problemas mencionados anteriormente es utilizar la comunicación asincrónica entre estos servicios (ver Figura 3). En este caso el Servicio A se suscribe a los eventos publicados por el Servicio B en cuanto a cambios en su estado interno. Cuando se producen estos eventos, el Servicio A guarda los datos que considere importantes. Entonces, cuando el Servicio A recibe un pedido no depende ya de partes externas para procesar sin afectar la disponibilidad. Note que la carga en el Servicio B es incluso menor que en el ejemplo original de comunicación sincrónica ya que no recibe más esos pedidos. La infraestructura moderna de mensajería y publicación/suscripción puede mantener la carga constante sin importar la cantidad de consumidores. La actualización de los datos también mejora con la comunicación asincrónica; el Servicio A recibe los datos más cerca del momento en que el Servicio B los dejó disponibles. No es necesario que hagamos una compensación sobre la carga (como resultado del intervalo de sondeo) con respecto a la actualización de los datos.

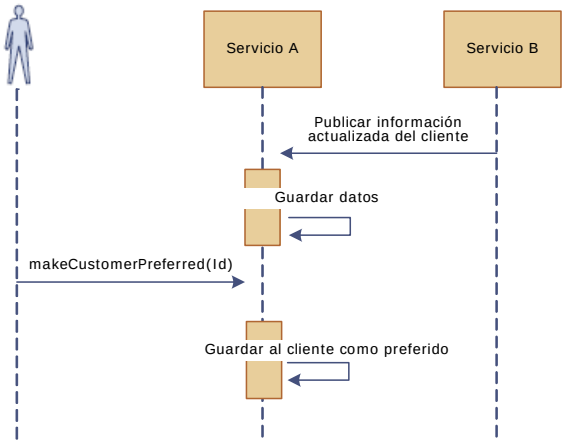
Limitaciones técnicas y duplicación de datos

Aunque el segundo principio parece claro a primera vista, la naturaleza de una limitación no es evidente en absoluto. ¿Las limitaciones del Servicio A y B en los anteriores ejemplos son diferentes en los casos de sincronía y asincronía? No parece ser así, pero existe una diferencia técnica importante: las transacciones.

Para administrar una sola solicitud en forma adecuada, en casos donde la solicitud provoca un cambio en los datos, la administración de la solicitud se debe realizar dentro del contexto de una transacción para que el estado del servicio permanezca uniforme. Si ese servicio debe interactuar con otros para administrar la solicitud, y como resultado esos servicios cambian también sus datos, ¿se deberán producir esos cambios dentro del contexto original de transacción?

Cuando la interacción de servicios es sincrónica, la división de responsabilidades entre los servicios con frecuencia puede exigir que se efectúen los cambios en el servicio dentro de una sola transacción. Cuando la interacción de servicios es asincrónica (o sincrónica con sondeo) evitamos cambiar datos en otros servicios en conjunto, y no existe necesidad de tener limitaciones de servicio cruzado de transacciones. Obviamente, si una transacción que comienza en un servicio bloqueara los recursos en otros servicios, esto exigiría altos niveles de confianza entre los servicios

Figura 3: Comunicación asincrónica

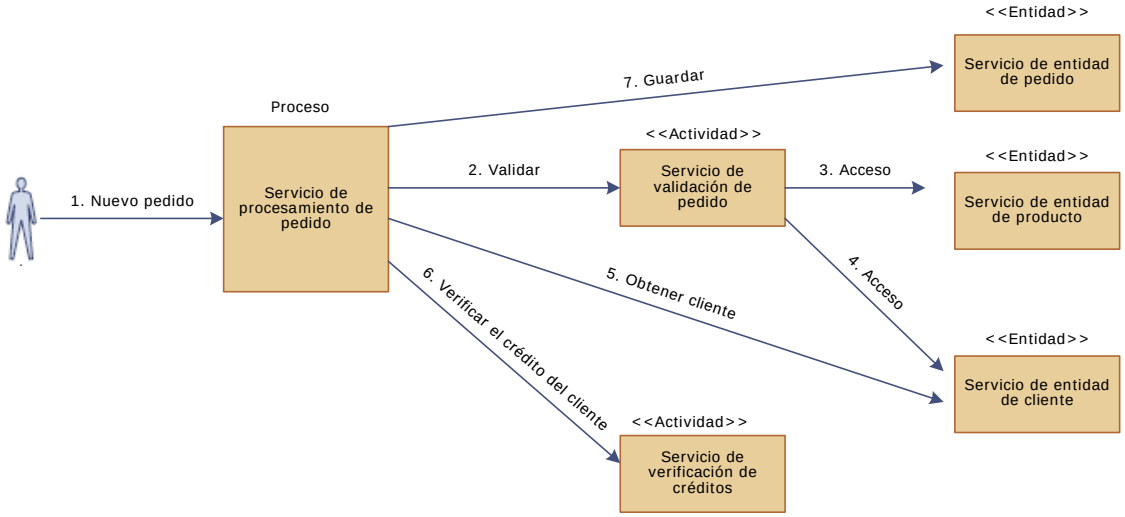


involucrados y difuminaría la diferencia entre dónde finalizó un servicio y dónde comenzó el otro.

Definamos servicios autónomos. Es evidente que los servicios autónomos cubren mucho más que las comunicaciones de bajo nivel, cubriendo muchos aspectos que incluyen la confianza y confiabilidad. No obstante, hemos visto que limitar las interacciones entre servicios a la mensajería asincrónica ha guiado nuestra arquitectura de manera tal que los servicios autónomos y acoplados en forma liviana se han cristalizado. De hecho, los servicios autónomos parecen expandirse sobre la orientación al servicio (o limitar sus grados de libertad) agregando un nuevo principio: *un servicio interactúa con otros servicios utilizando patrones de comunicación asincrónicos.*

Note que aunque un servicio pueda consumir otros servicios en forma asincrónica, este consumo no necesariamente significa que no puede exponer una interfase sincrónica. Google y Amazon hacen precisamente eso. Los servicios Web que estos exponen son sincrónicos por naturaleza pero no afectan su autonomía.

Figura 4: Interacciones entre proceso, actividad y servicios de entidad



A primera vista parece que el uso de comunicaciones de publicación/suscripción lleva a una duplicación de datos entre servicios similar a lo que ocurre cuando se realiza la duplicación de datos. Las técnicas de duplicación de datos como extracción, transformación, y carga (ETL) o transferencia de archivos manejan la transferencia de datos entre fuentes de datos de bajo nivel como bases de datos y directorios. Esta transferencia evita las

"EL AGREGACIÓN DE ENTIDADES REPRESENTA LA NECESIDAD COMERCIAL DE OBTENER UNA VISIÓN GLOBAL DE LOS DATOS EN TODA LA EMPRESA".

estructuras lógicas de más alto nivel para administrar esos datos en forma coherente, lo cual generalmente causa una duplicación de esas mismas estructuras lógicas al final de la transferencia del consumidor.

Los datos que fluyen a través de la interacción publicación/suscripción tienen conducta diferente. Cuando un servicio publica un mensaje, ese mensaje debe ser parte del contrato de servicio- ese contrato es independiente del esquema de guardado de datos subyacente. El proceso de construcción del mensaje –recuperando los datos correspondientes del almacenamiento de datos y transformándolos al esquema del mensaje- va a través de todas las capas del servicio. Los servicios que consumen estos mensajes no necesitan implementar la misma lógica. Además, la decisión sobre el momento de publicar el mensaje es una decisión lógica para el cual se ha escrito el código; no es un detalle menor del momento en que se programó la ejecución de la escritura de ETL.

La diferencia más importante entre duplicación simple de datos e interacción de servicio autónomo es que el servicio de consumidor decide guardar solamente los datos que necesita. No hay ya una "puerta trasera" de entrada a la base de datos del servicio. Cuando el servicio del consumidor recibe el mensaje que se publicó, los datos dentro del mensaje no evitan ninguna de las capas en el servicio hasta que llega a la base de datos (ver Recursos).

Cuando se utilizan estos tipos de interacciones de servicio asincrónico, generalmente descubrimos que nuestros servicios tienden a ser más importantes, con frecuencia contienen bases de datos propias y en sus propios servidores o centros de datos. Manteniendo los contextos de transacciones limitados al alcance de un solo servicio, las responsabilidades de ese servicio tienden a expandirse al nivel de una función comercial o departamento comercial- la limitación natural que se encuentra en el dominio comercial. Este efecto es bastante entendible cuando vemos los niveles de acoplamiento a diferentes niveles de negocios. Los departamentos están acoplados levemente entre sí, colaborando sin conocimiento profundo de los trabajos internos de los otros. Los grupos internos de un departamento generalmente exigen una comprensión más profunda de los trabajos de grupos paralelos para llevar a cabo el trabajo.

Podemos ver que este estilo de arquitectura no contradice de ninguna manera a ninguno de los cuatro principios, pero los tipos de servicio familiar encontrados al utilizar orientación al servicio ya no son adecuados.

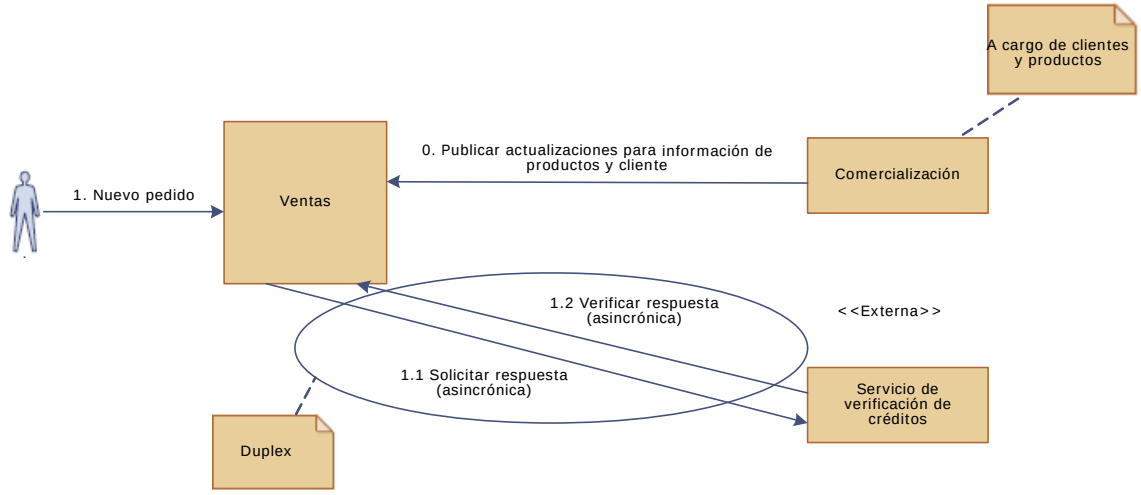
Dificultades comunes de la limitación

Los servicios de proceso, de actividad y de entidad alguna vez se propusieron como forma de realizar una orientación al servicio. Los servicios de proceso administran los procesos comerciales a largo plazo. Los servicios de actividad administran operaciones atómicas que cubren la interacción con más de un servicio de entidad. Los servicios de entidad administran la interacción con una entidad comercial. En este modelo de entidad la agregación se produce a nivel del servicio de entidad. El problema con esta opción de limitaciones de servicio se manifiesta en los flujos de interservicio sincrónico (ver Figura 4).

Aunque es muy posible modelar la misma interacción con mensajería asincrónica (específicamente con respecto a un servicio de verificación de créditos externos), el valor de separar procesamientos de pedidos en tres servicios no es claro. No es probable que un grupo diferente trabaje en cada uno de los servicios de pedidos o que funcionaran en diferentes plataformas. El acoplamiento firme entre estos servicios es inherente. Todos estos trabajan con procesamiento de pedidos; no compartir una clase común de Pedido probablemente causaría duplicación de códigos, pero esa duplicación iría contra el tercer principio de la orientación al servicio. Hay solo una conclusión: no podemos separar nuestros servicios junto con las limitaciones de proceso/actividad/entidad. Considere el resultado de modelar este proceso comercial utilizando servicios autónomos (ver Figura 5).

Los servicios autónomos en la Figura 5 representan una posible división correlacionada con una organización determinada; diferentes empresas tendrían diferentes grupos a cargo de

Figura 5: Interacciones entre servicios autónomos



diferentes procesos comerciales. Existen dos diferencias importantes que notar aquí. La primera es que el servicio de ventas guarda los datos de clientes y productos necesarios internamente para que, cuando necesite procesar un pedido, ya tenga todos los datos que necesite. La segunda diferencia es que la división de manejo de pedidos entre servicios separados no se produce en este caso. Aunque es probable que el servicio de ventas tenga capas y componentes diferentes para administrar los diversos pasos del procesamiento de pedidos, y posiblemente algún tipo de motor de flujo de trabajo para administrar esos pasos, esos son detalles de implementación del servicio.

La duplicación de códigos que surge siguiendo el principio de "esquema/clase" en el anterior caso no se produce aquí, sencillamente porque el principio no se aplica dentro del límite de servicio. (El problema aquí es que todos son la misma entidad (Pedido) probablemente con los mismos campos. Antes de que el servicio de entidad guarde el pedido, también debe realizar la validación: código que ya fue escrito para la validación del servicio de actividad). Si no podemos ya utilizar servicios de entidad, ¿cómo y dónde se produce la agregación de entidad al utilizar servicios autónomos?

Requisitos de agregación de entidad a nivel comercial

Antes de profundizar en los detalles técnicos de la solución, debemos definir mejor el problema. Como ya se dijo, la agregación de entidades representa la necesidad comercial de obtener una visión global de datos en toda la empresa. Diversos departamentos comerciales exigen diferentes elementos de datos de una entidad determinada pero rara vez exigen todos los datos para esa entidad. Este caso es uno en el cual un departamento comercial exige datos que posee otro departamento. Por ejemplo, el departamento de comercialización necesita saber el valor total de pedidos por cliente por trimestre- datos que son administrados por el departamento de ventas. En este caso, agregamos datos de dos sistemas existentes a uno de esos sistemas, que es un estilo diferente de agregación de entidades que el visto más usualmente como agregación de entidades; pero es el caso más común visto en el campo.

Agregación OLTP intersistemas. Comenzaremos por analizar un ejemplo concreto. En un escenario típico de procesamiento de pedidos tenemos dos sistemas: uno acepta pedidos de nuestro sitio Web; el otro es un sistema de administración de relaciones de clientes propios (CRM). Comercialización tiene un nuevo requisito que los "clientes preferenciales" deben recibir un 10 por ciento de descuento en todos sus pedidos. Un cliente preferido se define como cliente que viva en Estados Unidos que haya negociado por $50,000 con la empresa durante el último trimestre, pero se prevé que esta definición se modifique.

Después de analizar este requisito podemos ver que tiene dos partes: quien es un cliente preferencial, y qué hacemos con esa información. La primera decisión que se debe tomar tiene que ver con las limitaciones y responsabilidades: qué sistema estará a cargo de los *clientes preferenciales*, y qué sistema estará a cargo de lo que hacemos con los clientes preferenciales. En este caso no hay razones para que el sistema CRM no acepte la primera responsabilidad y el sistema de procesamiento de pedidos acepte la segunda. La siguiente decisión que se debe tomar es cómo interactuarán estos sistemas, si sincrónica o asincrónicamente.

Agregación OLTP sincrónica. La primera manera de manejar este requisito es agregar al sistema de procesamiento de pedidos un método de consulta que le informe a los clientes a quienes pertenecen el total de pedidos que ascienden al menos a x durante un plazo y. Este método nos da la capacidad de administrar diversas sumas y plazos a medida que el departamento de comercialización cambia sus reglas. El sistema CRM cumpliría pedidos para los cuales los clientes sean clientes preferenciales consultando el sistema de procesamiento de pedidos (ver Figura 6).

Agregación de OLTP autónoma. En el segundo enfoque, vemos a cada uno de los sistemas como servicios autónomos que notifican a consumidores externos los eventos significativos. En este caso, las interacciones entre los servicios son más impulsadas por eventos. En la figura 7, cuando el sistema de procesamiento de

"ADMINISTRAR UN SOLO SERVICIO QUE MANEJE LAS NECESIDADES DE AGREGACIÓN DE ENTIDADES A NIVEL EMPRESARIAL ES MUCHO MÁS RENTABLE QUE ADMINISTRAR MÚLTIPLES SERVICIOS DE ENTIDADES".

pedidos necesita saber si un cliente determinado es un cliente preferencial, no necesita comunicarse con el sistema CRM. Los datos de estos dos sistemas se han agregado por diseño.

Exploremos este ejemplo un poco más al cambiar el departamento de comercialización las normas que definen a los

Figura 6: Agregación de datos de diferentes sistemas sin servicios autónomos

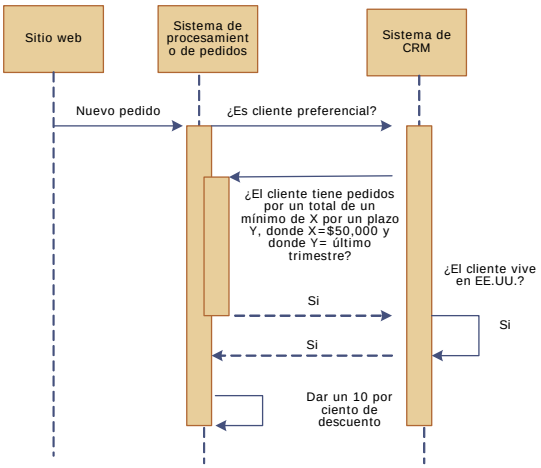


Figura 8: Cambios efectuados cuando no se utilizan los servicios autónomos

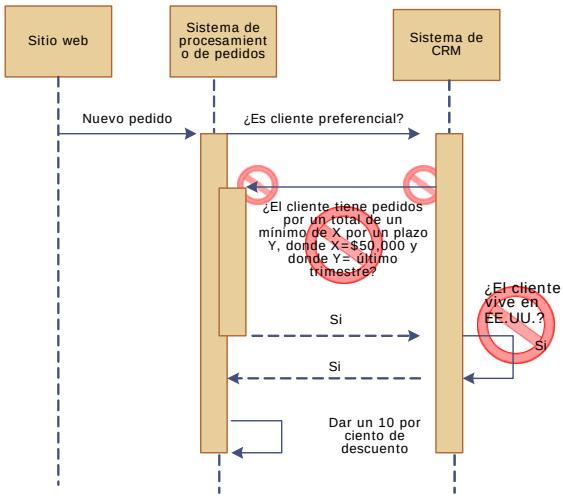


Figura 7: Agregación de datos de diferentes sistemas con servicios autónomos

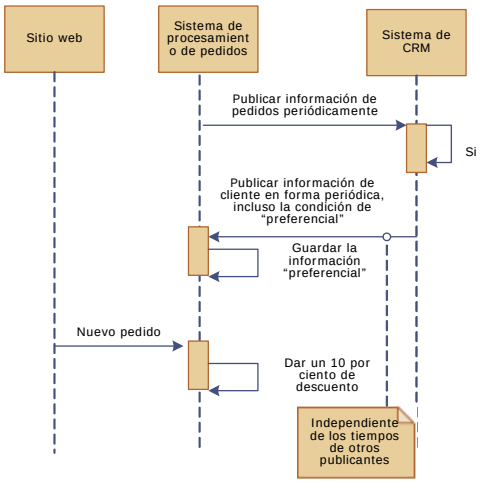
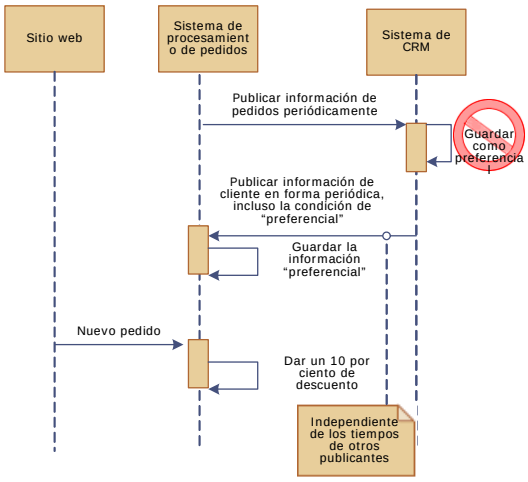


Figura 9: Cambios efectuados cuando se utilizan servicios autónomos



clientes preferenciales. Los clientes preferenciales son ahora los clientes que viven en Norteamérica que hayan realizado al menos tres pedidos en el último trimestre, cada uno por un total de $15.000 o más.

En nuestro primer enfoque, la consulta original que agregamos no es ya relevante. Ahora necesitamos dar soporte a una clase diferente de consulta (obtener clientes con un mínimo de X pedidos por un plazo Y, cada pedido por un total Z o más, donde X= 3, Y = ultimo trimestre, y Z=$15,000), cambiando la interfase del sistema de procesamiento de pedidos, algo de su implementación (para dar soporte a la nueva interfase), y el código que lo activa en el sistema CRM (ver Figura 8). En el segundo enfoque, los únicos cambios que necesitamos hacer son internos del sistema CRM. No es necesario cambiar ni la interfase ni la implementación del sistema de procesamiento de pedidos (ver Figura 9).

El término agregación de entidades recuerda un proceso activo de recolección de datos de fuentes dispares y fusión de datos para formar un todo cohesivo. Aunque la dinámica del primer enfoque se aplica muy bien a este proceso, no sucede lo mismo con el segundo enfoque. No obstante, es claro que el segundo enfoque mantiene un menor nivel de acoplamiento entre estos dos servicios, incluso al cambiar los requisitos. Recuerde esto cuando diseñe servicios; el acoplamiento leve entre servicios debe ocurrir desde el punto de vista tanto de los datos como de la lógica.

Agregación de entidades para inteligencia comercial

En el anterior caso, los datos agregados fueron esenciales para el funcionamiento del departamento(s) comercial(es) involucrado(s), participando en procesamiento de transacción día a día. No obstante, la agregación de entidades se debate más frecuentemente en el contexto administrativo- los negocios desean

obtener una visión completa de los datos empresariales. Este contexto es bastante diferente del anterior en que los datos agregados se utilizan principalmente para soporte de decisiones y para inteligencia comercial- en otras palabras, sobre todo en escenarios de uso de sólo lectura.

Una manera simple y efectiva de modelar este contexto es instituyendo un servicio de administración (ver Figura 10). Este servicio no realiza operaciones y administración de suministro para otros servicios sino que en cambio une los datos publicados por otros servicios y los guarda en un formato optimizado para sus escenarios de uso.

Otro requisito comercial común que aparece en el contexto de agregación de entidades es el análisis de antecedentes. Aunque pueda no tener sentido para el servicio de comercialización mantener los datos anteriores sobre productos pasados que ya no se ofrecen, los usuarios del servicio de administración podrían considerarlo invalorable para comparar cifras de ventas y ganancias de productos pasados y actuales. Sería responsabilidad del servicio de administración administrar estas tendencias históricas.

Un beneficio de utilizar un servicio de administración en comparación con los servicios de entidad está precisamente en el área de administración de operaciones. Administrar un solo servicio que maneja las necesidades de agregación de entidades a nivel empresarial es mucho más rentable que administrar servicios de entidad múltiple –uno por cada entidad que deba agregarse.

Figura 10: Relaciones entre el servicio de administración y otros servicios autónomos

Los cambios internos a cualquiera de estos servicios que no afectan su contrato incluso pueden no impactar al servicio de administración. Los cambios al contrato que puedan afectar a numerosas entidades causan los cambios correspondientes solamente en el servicio de administración, no en cada uno de los servicios de entidades que previamente agregaron esas entidades.

Los servicios pueden parecer la nueva unidad de reutilización en la empresa, pero esto no cumple con el principio de autonomía, sin mencionar que las limitaciones en la manera en que interactuamos con un servicio lo hacen desagradable. Por ejemplo, aunque puede parecer que el servicio de administración puede proveer fácilmente un rastreo y almacenamiento de auditoría para todos los otros servicios, esta opción puede quebrar la autonomía de esos servicios. Si el servicio de administración fracasara por cualquier motivo, otros servicios no deberían tener permitido continuar el proceso sin auditar. Además, un servicio autónomo no puede confiar en que otro servicio provea esta capacidad fundamental, que no es decir que no se puede encapsular el rastreo de auditoría (u otra funcionalidad específica) en un componente que todos los servicios reutilicen. Los temas de cumplimiento regulatorio se deben tener en cuenta dentro de cada servicio.

Al reconocer que los requisitos para agregación de entidades OLTP y OLAP son diferentes, hemos podido identificar dos soluciones diferentes, pero sencillas, utilizando un solo paradigma de comunicación. Los patrones de mensajería asincrónica permiten la creación de servicios autónomos y levemente acoplados que se parecen más a los procesos comerciales que modelan. Como consecuencia, generalmente todo lo que se necesita para responder a los cambiantes requisitos comerciales es un cambio local a un solo servicio. Estos cambios a pequeña escala no afectan los contratos de interservicio y se pueden efectuar con mayor certeza de que otros sistemas no se verán afectados y por lo tanto en menos tiempo. Alinear la informática con el comercio tiene mucho que ver con las comunicaciones interpersonales y la comprensión que la tecnología, pero no significa que la tecnología no pueda colaborar. •

Recursos

Patrones de integración empresarial: Diseño, construcción e implementación de soluciones de mensajería, Gregor Hohpe y Bobby Woolf (ADdison-Wesley Professional, 2003)

"Datos externos vs. datos internos" Pat Helland (Microsoft Corporation)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dataoutsideinside.asp

"Cómo tratar la concurrencia: diseño de interacción entre servicios y sus agentes", Maarten Mullender (Microsoft Corporation, 2004)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/concurev4M.asp

Eventos de Microsoft
"WebCast de MSDN: Porqué no puede usted hacer SOA sin mensajería (Nivel 300), Udi Dahan (Microsoft Corporation, 2006)
http://msevents.microsoft.com/cui/WebCastEventDetails.aspx?EventID=1032273610&EventCategory=5&culture=en-US&CountryCode=US

MSDN- Foros del Canal 9
"ARCast- Servicios Autónomos", Udi Dahan (Microsoft Corporation, 2006)
http://channel9.msdn.com/Showpost.aspx?postid=163201

"ARCast- Orientación al servicio y flujo de trabajo", Udi Dahan (Microsoft Corporation, 2006)
http://channel9.msdn.com/ShowPost.aspx?PostID=163471

"Desafíos de SOA: agregación de entidades", Ramkumar Kothandaraman (Microsoft Corporation, 2004)
http://msdn.microsoft.com/architecture/soa/default.aspx?pull=/library/en-us/dnbda/html/dngrfsoachallenges-entityaggregation.asp

Sobre el autor

Udi Dahan es un MPV arquitecto de soluciones de Microsoft, un reconocido experto en desarrollos .NET, y el arquitecto principal informático y gerente de línea de productos C4ISR en KorenTec. Udi es conocido como una autoridad fundamental en SOAs en Israel y asesora sobre arquitectura y diseño a gran escala, sistemas de objetivos fundamentales desarrollados en todo el país. Su experiencia cubre las tecnologías relacionadas con sistemas de control y comando, aplicaciones en tiempo real, y servicios de Internet de alta disponibilidad. Para más información, visite www.UdiDahan.com

Duplicación de datos como antipatrón de SOA empresarial

Por Tom Fuller y Shawn Morgan

Síntesis

Al irse previendo y suministrando las aplicaciones en toda una organización, el deseo de utilizar duplicación de datos como una estrategia de integración "de rápida solución" generalmente obtiene más aprobación. Este camino de menor resistencia trae redundancia e inconsistencia. Aplicar en forma periódica este antipatrón dentro de una empresa diluye las metas originales a largo plazo de una arquitectura orientada al servicio (SOA). No obstante, según el contexto en el cual aparece, la duplicación de datos se puede ver en forma positiva o negativa. Para derivar con éxito en una estrategia orientada al servicio, los arquitectos empresariales necesitan separarse de los éxitos y fracasos de otras arquitecturas. Los patrones de arquitectura, cuando se descubren y documentan, pueden brindar la mejor técnica para administrar el trabajo oneroso de influenciar el cambio en su empresa. Utilizaremos un antipatrón y un patrón para describir la forma en que la duplicación de datos puede afectar su arquitectura empresarial.

El camino hacia la orientación al servicio aún se encuentra en sus primeras etapas. Para que el SOA logre las metas nobles visualizadas por la industria, los arquitectos necesitarán basarse en las mejores prácticas documentadas. Aquí es donde los patrones y antipatrones colaborarán para construir el puente entre la conceptualización y la realidad. Los patrones representan una probada estrategia repetible para lograr resultados esperados. Los patrones y antipatrones de arquitectura de software documentaron ejemplos de éxitos y fracasos en el intento de aplicar esas estrategias en la informática. Mediante la identificación satisfactoria de abstracciones de arquitectura, el SOA comenzará a encontrar estrategias de implementación que aseguren una entrega exitosa.

Describiremos porqué la duplicación de datos, al utilizarse como patrón de base en su SOA empresarial, puede llevar a una arquitectura costosa y complicada y le brindará una estrategia de refabricación para moverse de una duplicación de datos (antipatrón) a un servicio directo (patrón). Luego le ofreceremos una breve explicación de en qué casos se puede usar la duplicación de datos para resolver con éxito algunos dominios de problemas específicos (duplicación como patrón) sin comprometer otras iniciativas de arquitectura empresarial. Los arquitectos extraerán de este debate una estrategia para facilitar el desplazamiento de la no viabilidad de arquitectura.

Para promover la consistencia, la industria del software se asentó en una plantilla para documentar los patrones de diseño. Esta plantilla, no obstante, parecía insuficiente para describir los antipatrones de arquitectura y sus soluciones refabricadas eventuales. Una colección híbrida de secciones de patrón de diferentes recursos se reunió para ayudar a definir una estructura para antipatrones y patrones descriptos en el presente (ver Recursos). La tabla 1 muestra el conjunto de secciones y si se aplican o no a patrones, antipatrones, o ambos.

Duplicación de datos: un antipatrón de arquitectura de SOA

Comenzaremos con una descripción detallada de la manera en que la duplicación de datos es un antipatrón al aplicarse en un SIA, utilizando una plantilla de antipatrón para explicar las especificaciones de contexto y las fuerzas que validen esta arquitectura. Un ejemplo ilustra los problemas cuando se aplica este antipatrón. Para cerrar, verá la solución refabricada y los beneficios que se pueden ver cuando el patrón se aplica en lugar del antipatrón.

Contexto. Este antipatrón describe un escenario común que enfrentan los arquitectos cuando intentan construir soluciones de SOA empresariales en un entorno distribuido. En el cambio de un entorno principal a un entorno de distribución, se produjo un cambio en la administración de nuestros datos importantes originales (los datos que impulsan a nuestra empresa en forma diaria). Decidimos construir nuestras nuevas aplicaciones distribuidas utilizando un modelo silo, donde una aplicación se define como conjunto de funcionalidad comercial expuesta mediante una interfase de usuario (UI), y esa funcionalidad comercial se construye dentro de su propio contexto. Esta decisión fomenta arquitecturas que favorezcan el aislamiento extremo y la agregación de entidades innecesaria.

"PARA PROMOVER LA CONSISTENCIA, LA INDUSTRIA DEL SOFTWARE SE BASÓ EN UNA PLANTILLA PARA DOCUMENTAR PATRONES DE DISEÑO. ESTA PLANTILLA, NO OBSTANTE, NO PARECIÓ SUFICIENTE PARA DESCRIBIR LOS ANTIPATRONES DE ARQUITECTURA Y SUS EVENTUALES SOLUCIONES REFABRICADAS"

Fuerzas. Este antipatrón se produce en un SOA por la comodidad que los arquitectos y diseñadores tienen con este modelo. Incluso cuando se explica como un antipatrón dentro del contexto de SOA, los arquitectos y diseñadores siguen utilizando este antipatrón por las siguientes razones:

- *El arquitecto está familiarizado con la estrategia de duplicación de datos.* Las técnicas utilizadas para duplicar datos se han afinado bien desde los primeros días del desarrollo orientado a

Tabla 1: Secciones correlacionadas con patrones y antipatrones

Sección	Descripción	Patrón	Antipatrón
Nombre	Un nombre corto útil que puede ayudar a describir rápidamente el patrón-antipatrón	X	X
Contexto	La información previa interesante donde se aplicará el patrón-antipatrón	X	X
Fuerzas	Existe un número de razones por las cuales esta solución se utiliza. Con frecuencia en un antipatrón estas razones se forman por concepciones erróneas o falta de conocimiento.	X	X
Solución	Esta sección describe la solución propuesta para el problema basada en el contexto y fuerzas documentadas y representa la estrategia o inteligencia del patrón	X	
Mala solución (antipatrón)	Esta sección describirá el antipatrón o solución a un problema que parece tener beneficios pero de hecho generará consecuencias negativas si se utilizan.		X
Consecuencias	Las consecuencias son efectos colaterales de la solución implementada. En caso de un antipatrón, las consecuencias se formarán de impactos negativos que pesarán más que el beneficio del patrón en sí.	X	X
Solución refabricada	Todo antipatrón deberá tener una inversa que de hecho sería una solución o patrón, que se pueda documentar como patrones.		X
Beneficios	Los beneficios serán los mismos para un patrón documentado y la solución refabricada de un antipatrón	X	X
Ejemplo	Un ejemplo realista de los casos en que se aplica patrón-antipatrón para resolver o enfatizar un problema.	X	x

los lotes. Esta afinación crea un nivel de comodidad para el arquitecto que ha diseñado sistemas con duplicación de datos durante años, utilizando transferencias en base a archivos o extracción, transformación y carga (herramientas ETL). Parece suficientemente sencillo seguir utilizando esta estrategia.

- *El arquitecto está preocupado por el rendimiento de un servicio expuesto por un sistema de datos originales.* Por ejemplo, acceder a un almacén de datos remotamente mediante servicios puede aminorar el ritmo del nuevo sistema sobre una solución que utiliza base de datos local, que es el caso de los datos uno-a-varios.
- *Servicios separados pueden cada uno tener visiones levemente diferentes de la entidad.* Combinar las visiones diferentes en una entidad agregada y mantener la autonomía del servicio se considera demasiado difícil, así que los datos de los servicios se duplican en un almacén, que es el caso de los datos muchos-a-uno.
- *Al arquitecto le preocupa que la durabilidad del nuevo sistema pueda estar comprometida.* Por ejemplos, si un nuevo sistema accede a los datos originales mediante un servicio que existe en la red, las fallas de red o fallas de sistema de datos originales podrían causar un período de inactividad para el nuevo sistema. Tener los datos disponibles a nivel local parece mitigar este problema.
- *Las aplicaciones se construyen usando un modelo silo.* Una aplicación se define como un conjunto de funcionalidades comerciales expuestas mediante una interfase de usuario y esa funcionalidad comercial se construye dentro de su propio contexto sin un arquitecto empresarial que ayude a guiar a quienes desarrollan la aplicación hacia la reutilización de servicios disponibles.
- *Los recursos para presentar una solución utilizando duplicación de datos son abundantes.* La habilidad necesaria para implementar una solución utilizando duplicación de datos en general es fácil de encontrar. Las aplicaciones fundamentales que impulsan servicios nuevos o existentes llevan un conjunto de habilidades menos frecuentes, y exigen que alguien con fuertes conocimientos de sistemas distribuidos, servicios Web, orientación de objetos, y otros mecanismos modernos, presenten soluciones para su negocio.

Difuminación de las líneas de control

Mala solución (antipatrón). Para lograr este antipatrón, se duplica información de datos originales desde una fuente de datos originales a una nueva base de datos creada para exponer la funcionalidad de la nueva aplicación (ver Figura 1). Esta exposición se puede lograr mediante diferentes mecanismos, como el uso de herramientas ETL o mediante mecanismos más básicos como transferencia de archivos de los datos de un sistema a otro. La información de datos originales también puede aumentarse por nuevos sistemas, agregando atributos adicionales de entidad a la entidad original.

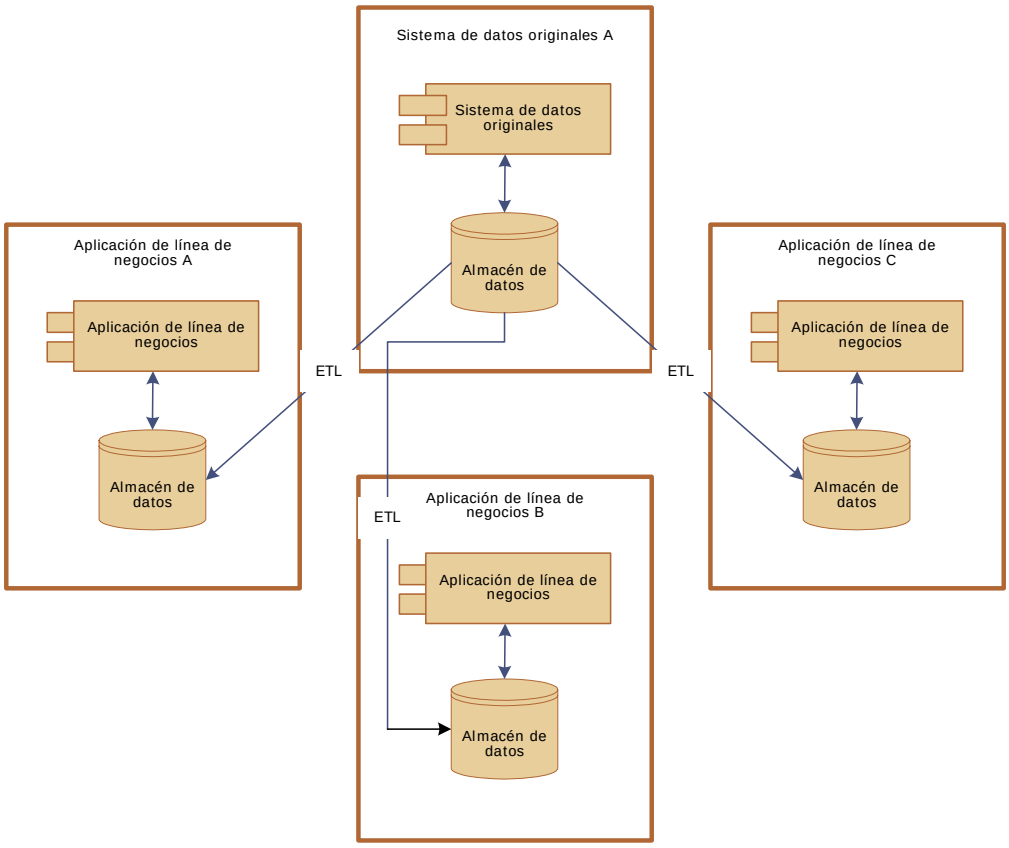
Consecuencias. Las consecuencias de este patrón si es aplicado en forma incorrecta dentro de un SOA no son visibles en forma inmediata. Cuando se duplican los datos desde el sistema de datos originales, en una forma fraccionada, a muchas bases de datos de aplicación diferentes, su orientación al servicio se vuelve difícil de administrar. Los datos duplicados se convierten en los dominantes en toda la empresa. La responsabilidad no bien identificada respecto de los datos difumina las líneas de control. Existen múltiples servicios para manipular los mismos datos (o aumentarlos levemente). "la visión comercial" de los datos se vuelve dispar.

Esta responsabilidad no identificada de los datos y pérdida de control sobre los datos originales es desconcertante para cualquier arquitecto. Como los datos se proliferan y las aplicaciones comienzan a agregar sus propios giros a los datos la complejidad comienza a aumentar. Los nuevos campos que se agregan a los datos, que pueden ser importantes para la empresa también, pueden no agregarse de vuelta al sistema de datos originales. Por supuesto, con los ajustados cronogramas y presupuestos de la mayoría de los proyectos empresariales, un equipo de proyecto raramente aceptará el esfuerzo adicional de crear nuevos elementos de datos mediante servicios que se extiendan fuera del sistema de datos originales, o crear una capa de servicios de agregación (reemplazando el servicio de datos originales inicial).

Estos pueden de hecho extender los servicios fuera de su sistema que expondrán estos nuevos datos originales. También pueden crear un nuevo servicio que imite algunas de las funcionalidades del sistema de datos originales para exponer los datos que han duplicado para su nuevo sistema. Esta exposición crea un gran problema para los arquitectos empresariales. el arquitecto ahora necesita evitar que los nuevos sistemas consuman servicios expuestos utilizando datos duplicados, que llevará a la confusión dentro de la arquitectura de empresa de la verdadera posesión del objeto comercial.

La duplicación también lleva consigo una enorme responsabilidad sobre el sistema de duplicación, de duplicar la lógica comercial del sistema de datos originales. Muchas veces los datos son manejados por el sistema de datos originales antes de exponerse como servicio a otras aplicaciones. En estos casos, el método rápido de duplicar datos al nuevo sistema lleva consigo la

Figura 1: Proliferación de datos originales que lleva a la posesión distribuida de diversas permutas



carga de también duplicar la lógica comercial utilizada para manejo de datos. Salvo que se haga un buen análisis del sistema de datos originales, esta lógica tal vez nunca se implemente, o se implemente incorrectamente. Un caso aún más perjudicial es uno en el cual la lógica comercial que rodea la recuperación de datos originales exista en forma inicial. La lógica se agrega en cambio, más tarde en un proyecto que necesita actualizar el sistema de datos originales. Este caso puede ser doloroso para una empresa que ha duplicado muchas veces los datos. Cada sistema ahora se debe analizar en cuanto a los cambios exigidos al sistema para cumplir con las nuevas demandas de los datos empresariales.

"UNA ORGANIZACIÓN SOLAMENTE PODRÁ REALIZAR COMPLETAMENTE LOS BENEFICIOS DE UN SOA SI ESTA ORGANIZACIÓN ESTÁ INFORMADA DE LOS CAMBIOS DE MENTALIDAD NECESARIOS PARA PROVEER SOLUCIONES UTILIZANDO PATRONES DE ARQUITECTURA NUEVOS"

Redundancias difundidas

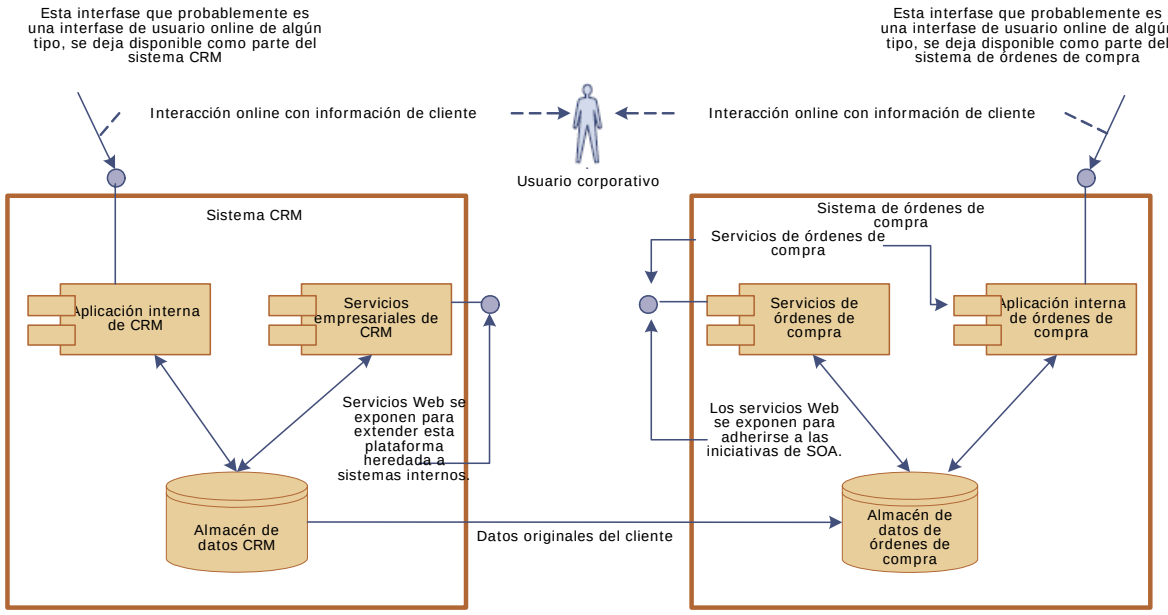
Con frecuencia es fácil descartar las consecuencias de utilizar la duplicación de datos, porque, en la superficie, parecen superfluas. Los problemas como espacio en disco, bienes reutilizables, y reducción de carga de trabajo parecen ser manejables con o sin el movimiento hacia la orientación al servicio. la consecuencia que los opaca a todos es la capacidad de disminuir la complejidad y brindar soluciones que rápidamente se adapten al cambio. La duplicación de datos agrega complejidad, fragilidad, e inflexibilidad por las redundancias difundidas que crea en su arquitectura empresarial.

Veamos un ejemplo. Una nueva aplicación se construye para manejar la administración de una orden de compra. La arquitectura demanda que los servicios comerciales se puedan crear para el nuevo sistema y que la UI, ya sea una implementación de un portal o en base a la Web, consuma esos servicios comerciales.

Los requisitos comerciales dictaminan que el sistema debería cumplir funciones de creación, modificación y visión de las órdenes de compra. Parte de la orden de compre es la información del cliente, que se almacena en un sistema de administración de relaciones de clientes (CRM). El sistema CRM expone los servicios para recuperar información de clientes, pero la decisión la toma el arquitecto respecto a duplicar los datos originales del cliente para los sistemas de orden de compra basados en fuerzas anteriormente mencionadas.

Un requisito del sistema nuevo de órdenes de compra es crear un número de cuenta para cada cliente. Estos nuevos datos se adjuntan a los datos duplicados del cliente en una base de datos

Figura 2: Redundancia de datos en el antipatrón



local del sistema de PO (órdenes de compra). La UI muestra la información del cliente mediante el uso de un nuevo servicio Web creado por un sistema de órdenes de compra vertical angosto. Otros servicios comerciales se desarrollan para la creación, modificación y vista de órdenes de compra que son consumidas generalmente por UI de sistemas de PO. Cualquier manipulación de datos o manejo que el sistema de datos originales haga cuando el usuario accede a sus servicios de datos se debe duplicar en el nuevo sistema. Cualquier cambio de datos se debe también duplicar en el nuevo sistema, como también los cambios a la lógica comercial a fin de acceder a esos datos.

El diagrama mostrado en la figura 2 aclara que la duplicación de datos resulta en servicios duplicados. Cada aplicación ahora tiene su propia visión de la entidad de cliente, con su propia manera de acceder a los datos mediante un servicio que se creó para tal fin. el sistema de órdenes de compra copia y extiende los datos de clientes a través de su propio conjunto de servicios, que crea eficazmente redundancia y confusión para cualquier sistema futuro que desee consumir servicios de cliente.

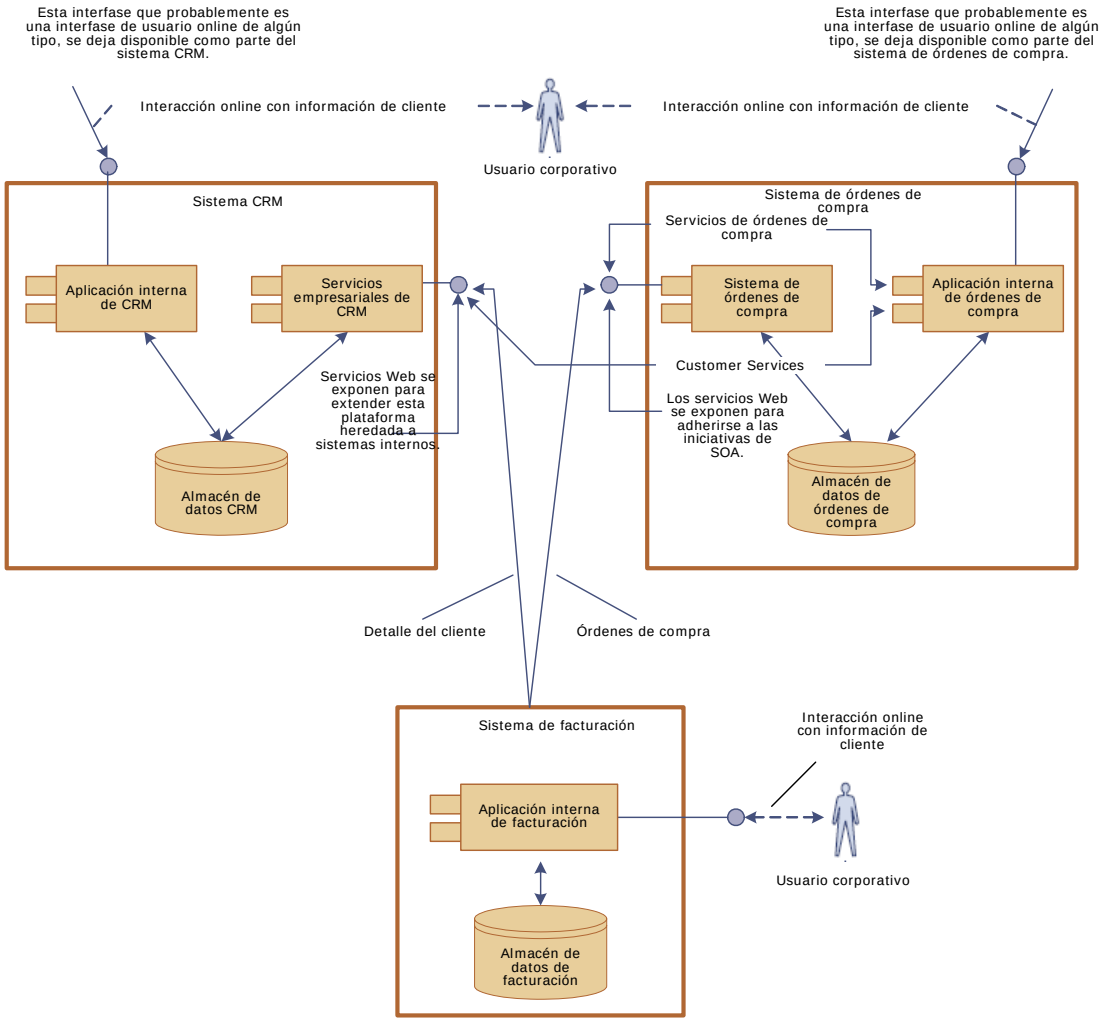
Ahora viene la siguiente aplicación, un sistema que administra la facturación y contabilidad. Este sistema también necesita los datos de clientes, incluso el número de cuenta. Hay muchas opciones actualmente que los diseñadores de sistemas deben elegir respecto a los datos de clientes. ¿Estos buscan los datos originales de clientes en el sistema de CRM y obtienen el número de cuenta del cliente en el sistema de PO, o, ya que los datos existen en el sistema de PO, solamente recurren al sistema PO para obtener toda la información del cliente? De hecho, debido a que la durabilidad es una gran inquietud (una fuerza en este antipatrón), se toma la decisión de duplicar los datos de clientes del sistema PO al sistema de facturación. Ahora, tres sistemas tienen la visión de los datos del cliente, completas en ciertos grados diferentes. Tres sistemas están obligados a construir una lógica comercial alrededor del acceso a esos datos. Cada uno expone los datos de clientes mediante servicios (porque construyen una "arquitectura orientada al servicio"), y la capacidad de la empresa de administrar datos de clientes en forma ágil sigue fracasando.

Conocimientos sobre los cambios

Solución refabricada. Los beneficios de un SOA solamente serán realizados completamente por una organización que esté informada sobre los cambios de mentalidad exigidos para brindar soluciones utilizando nuevos patrones de arquitectura. El SOA no brinda los tipos de beneficio que son posibles hasta que los arquitectos no se den cuenta de que solamente brindando un punto de control sobre los servicios y datos comerciales la organización podrá verdaderamente ser una empresa flexible, escalable y receptiva que el SOA promete. Para refabricar con éxito la solución, deberá tratar las fuerzas que crean el antipatrón:

- *El arquitecto está familiarizado con la estrategia de duplicación de datos.* Use los indicadores clave de rendimiento (KPI) que impulsan la implementación de SOA para construir un caso para no duplicar sin necesidad los datos. La capacitación en implementaciones SOA que disminuyen las necesidades de duplicar datos también pueden mitigar esta fuerza.
- *Al arquitecto le preocupa el rendimiento de un servicio expuesto por el sistema de datos originales.* Evalúe para validar el rendimiento de un servicio expuesto. Mediante la evaluación y un entendimiento del acuerdo a nivel de servicios (SLA) para la nueva aplicación, muchos encontrarán que el uso de un servicio nuevo o existente que expone datos originales rendirá dentro de los niveles de demanda de SLA.
- *Servicios separados pueden cada uno tener visiones levemente diferentes de la entidad.* Las capacidades y tecnologías para combinar las diferentes visiones en una entidad agregada y mantener autonomía del servicio ya están bien establecidas y se pueden realizar generalmente en tiempo de ejecución.
- *Al arquitecto le preocupa que la durabilidad del nuevo sistema pueda estar comprometida.* Trate el tema de la durabilidad de

Figura 3: El sistema refabricado



los servicios comerciales expuestos mediante mecanismos de infraestructura disponibles como el agrupamiento.

- *Las aplicaciones se construyen utilizando un modelo silo.* Muchos proyectos exigen la participación de un arquitecto con un ojo puesto en servicios nuevos y existentes y la funcionalidad que proveen. De hecho, las aplicaciones como las conocemos actualmente deben transformarse, de un cuadro simple, en un diagrama para un ensamble de servicios nuevos y existentes.
- *Los recursos para brindar una solución utilizando duplicación de datos son muchos.* Los patrones, plantillas, normas, y herramientas actualmente disponibles admiten al SOA verdadero con diseñadores y desarrolladores "de nivel medio".

Después de un detallado análisis de estos temas, podrá decidir su estrategia de arquitectura. El cambio hacia una orientación al servicio nos exige pensar diferente respecto de cuáles son las cuestiones fundamentales al tomar este tipo de decisiones. En vez de enfocarse tanto tiempo en la arquitectura de mi "aplicación" los arquitectos deberían enfocarse en la arquitectura de los servicios empresariales. Los servicios se deberían construir sobre el sistema que mantenga y administre los datos, o servicios de agregación se deberían construir a fin de brindar los datos, eliminando la necesidad de duplicar los datos en otra aplicación para proveer un servicio.

La visión de la empresa sobre una aplicación necesita esencialmente cambiar. Para hacer que funcione el SOA, debemos cambiar de construir aplicaciones- un sistema independiente que brinda funcionalidad de usuario- a construir productos: una colección de funciones a brindar al cliente. Deberíamos estar construyendo servicios que brinden esta funcionalidad comercial. El producto no es más que una composición u orquestación de uno o más servicios y brinda una o más piezas de funcionalidad comercial. Es la agregación de casos de uso solucionado.

Eliminación de ambigüedad

Una vez que la mentalidad se aclara, y estas fuerzas se mitigan, el acceso directo a servicios de línea de negocios (LOB) se vuelve una realidad. Las UIs de nuevos sistemas pueden solicitar servicios existentes según sea necesario, sin ninguna indirección de datos.

Después del refabricado de la arquitectura para servicios directos la ambigüedad de posesión de datos se elimina. Nuevas aplicaciones como el sistema de facturación (ver Figura 3) pueden recuperar los datos desde sistemas originales de registro.

Beneficios. Se consulta constantemente a los arquitectos para evaluar los intercambios de utilizar un cierto patrón dentro de una cierta solución. Cuando el entorno de arquitectura de la empresa intenta moverse hacia un SOA, por todas las razones que el SOA es una arquitectura importante, los intercambios de duplicar datos para mejorar la velocidad de la aplicación o durabilidad de la aplicación, se deben evaluar en comparación con los beneficios de claridad y simplicidad obtenidos de la posesión única y el régimen que una arquitectura de servicio le pueden brindar a su organización.

Los beneficios adicionales de utilizar el patrón de servicios directos se pueden encontrar en el KPI que nos lleva hacia una iniciativa de SOA en primer lugar. La orientación al servicio es el último intento de crear servicios de aplicación renovables. Los extremos ahorros de costo se pueden encontrar mediante la reutilización de servicios, tanto en el espacio de almacenamiento como en los costos laborales. Aunque la duplicación de datos puede parecer satisfacer los requisitos funcionales, las metas de la visión de arquitectura empresarial no se cumplen.

Otro beneficio se puede encontrar en la ineficiencia que se obtiene de recuperar solamente los datos que se necesitan. Las arquitecturas que impulsan la recuperación de datos por eventos están seguras de ser más eficientes que los anticuados procesos orientados por lotes. La duplicación de todos los datos durante un lote limitado por tiempo crea gastos generales costosos. Sin un mecanismo para recuperar los datos correspondientes en el momento indicado utilizando servicios, usted utilizará ciclos de procesamiento adicional para mover los datos que pueden tener poca o ninguna utilidad según las actividades que dependen de esos datos. El patrón de servicios directo asegurará que usted no realice un procesamiento inútil.

"OTRO BENEFICIO SE PUEDE ENCONTRAR EN LAS EFICIENCIAS QUE SE OBTIENEN RECUPERANDO SOLAMENTE LOS DATOS QUE SE NECESITAN. LAS ARQUITECTURAS QUE IMPULSAN LA RECUPERACIÓN DE DATOS POR EVENTOS SEGURAMENTE SON MÁS EFICACES QUE LOS ANTICUADOS PROCESOS POR LOTES"

Finalmente, la meta de cualquier servicio de datos empresariales es brindar la visión de una entidad más completa posible sin sacrificar los beneficios de las iniciativas de arquitectura empresarial. En casos donde las partes fundamentales de esa entidad se fragmentan en múltiples sistemas, el diseño del servicio puede exigir algún nivel de agregación. No obstante, esto no elimina la opción de utilizar el patrón de servicios directos para recuperar esos subcomponentes de nuestra entidad.

Una arquitectura que implementa y consume eficazmente servicios directos minimizará la fragilidad de la cartera de solución empresarial eliminando la redundancia. Mediante la centralización y reutilización, los servicios construidos que utilizan este patrón brindarán el mejor enfoque posible para continuar conservando la agilidad en un clima de cambios.

Duplicación de datos como patrón

Todas las empresas tienen sistemas y requisitos complicados que exigen a los arquitectos permanecer pragmáticos con respecto a los patrones que validan. En estos casos un arquitecto debe controlar la adopción difundida de patrones de duplicación mediante un buen régimen empresarial. Si eso no sucede, la empresa tiene un riesgo de utilizar demasiado este patrón. Debido a su baja barrera de acceso percibida para desarrolladores, diseñadores, y arquitectos, la estrategia de duplicación de datos deberá solamente considerarse como un patrón en situaciones extraordinarias.

Figura 4: Elemento fundamental de movimiento de los datos

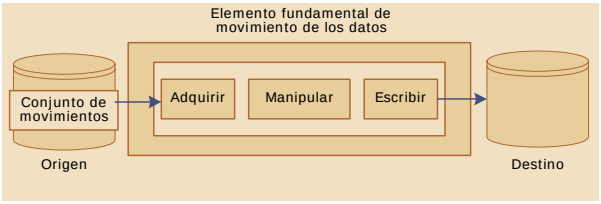
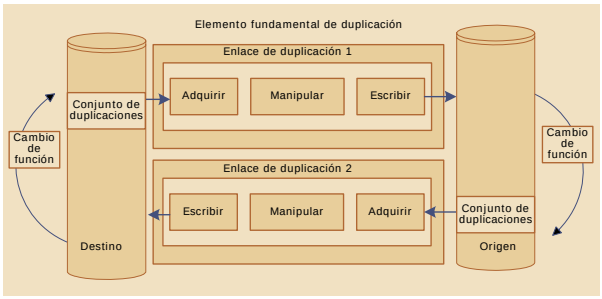


Figura 5: Un caso de duplicación de original a original



Contexto. El contexto en el cual la duplicación de datos es un patrón es levemente diferente del contexto en el cual es un antipatrón. Igualmente proviene de un modelo distribuido donde se desea obtener una copia de datos originales conservados en un sistema externo al sistema de datos originales. En algunos casos, tal vez no pueda evitar la duplicación. Algunos ejemplos incluyen cuando el período de espera de la red es un problema, como en una WAN; cuando el sistema de datos originales no es confiable o posee ventanas de mantenimiento incompatibles; cuando las capacidades offline son un requisito fundamental de la aplicación, y cuando los riesgos de no tener servicios se mitigan mediante el uso previsto de la aplicación.

Fuerzas. Las fuerzas que influyen en la decisión de realizar una copia de datos originales y pueden justificar la complejidad involucrada en duplicar datos son:

- *Disponibilidad de los datos en el sistema de datos originales no coincide con los requisitos para el nuevo sistema.* Por ejemplo, el sistema de datos originales se debe bajar para el mantenimiento entre ciertas horas donde el nuevo sistema debe estar disponible (y los datos son fundamentales para casos de utilización que brinda el nuevo sistema).
- *La red no es confiable o es demasiado lenta.* Por ejemplo, en una WAN donde la red permanentemente pierde la conexión al sistema de datos originales, copiar los datos puede solucionar el problema. Otro caso es que la red sea muy lenta para soportar el consumo directo de los datos cuando así se requiere.
- *Se prevé que el sistema tenga capacidades offline.* Esta fuerza es con frecuencia un requisito que se encuentra en sistemas que

tendrán poca o ninguna conectividad con los servicios de empresa que proveen los datos.

- *Los datos se copiarán sin ninguna lógica comercial reutilizable para los fines solamente de análisis.* Existen casos en los que los datos deben copiarse a un depósito de datos para utilizar en informes analíticos. En la mayoría de los casos, este tipo de transferencia de datos es más eficaz cuando se utiliza una herramienta ETL para sencillamente copiar los datos después de considerárselos listos para archivar.

Solución. Una solución a estas fuerzas es la duplicación de datos. La duplicación de datos se puede realizar utilizando un elemento fundamental de arquitectura conocido como *elemento de movimiento de los datos.* El elemento de movimiento de los datos consiste en un origen, enlace de movimiento, y destino.

El diagrama que se muestra en la figura 4 muestra un elemento básico de muchos otros patrones de movimiento de datos (ver Recursos). Dependiendo de la solución, podrá necesitar múltiples elementos de movimiento de datos para manejar el caso de original a original, donde los datos se pueden actualizar en el origen y también en el destino (ver Figura 5).

Beneficios. Mucho del trabajo descripto en los patrones que se mencionaron anteriormente se puede realizar con herramientas y recursos de no desarrollo. Esta función es uno de los beneficios principales de este enfoque, porque con frecuencia usted verá un costo total de propiedad inicial menor (TCO). Este beneficio de todos modos no dice todo. Con frecuencia las versiones y mantenimiento a largo plazo de estos tipos de herramientas y estrategias son muy costosos. Si el foco principal es el tiempo para comercializar y el mapa de ruta para la solución es relativamente corto, entonces este patrón puede ser viable con un grado alto de beneficios. Tenga en cuenta que intercambiará fragilidad y costos posibles de refabricación en el camino. A veces tal refabricación se puede realizar envolviendo un sistema de legado que no cumpla con los principios del SOA con una interfase que cumpla con el SOA. De este modo, la arquitectura empresarial se puede mover hacia un SOA en su totalidad, sin tener que refabricar todas las aplicaciones en su dominio.

Documentación de las mejores prácticas

Un aspecto inevitable de la innovación en la tecnología es el cambio. Los arquitectos de aplicación deben seguir concentrados en mitigar el riesgo del cambio. Utilizar patrones y antipatrones para documentar las mejores prácticas ha probado ser un éxito desde los antiguos patrones OO. Cuando se aplican a tiempo, los patrones de arquitectura son el arma más viable en la batalla contra el cambio.

Recursos

Patrones de diseño: Elementos de Software orientado al objeto reutilizable, Erich Gamma, et. Al. (Addison-Wesley Professional, 1995).

Microsoft Developer Network: "Patrones de datos- Patrones y prácticas de Microsoft", Phillip Teale, Christopher Etz, Michael Kiel, y Carsten Zeitz (Microsoft Corporation, 2003).

"Principios de Diseño de Servicio: Patrones y Antipatrones de servicio" (Microsoft Corporation, 2005).

"Antipatrones de SOA" (IBM, noviembre de 2005)

Fabricas de software: Ensamble de aplicaciones con patrones, modelos, marcos de referencia, y herramientas, Jack Greenfield, et al. (Wiley & Sons, 2004).

El patrón y antipatrón que se debaten en el presente se enfocaron en la duplicación de datos y su función en una empresa con iniciativa SOA. Utilizando una plantilla para descripción de patrones, usted podrá ver la razón por la cual esta técnica puede tener un impacto positivo o negativo según el contexto en el cual se aplica. Utilizar patrones para resolver este desajuste de arquitectura muestra precisamente cuánto éxito puede obtenerse al adoptar una metodología de arquitectura.

Al seguir evolucionando la industria del software, el rol de la arquitectura y los patrones se volverá cada vez más fundamental.

"MANTENIMIENTO Y VERSIONES A LARGO PLAZO DE ESTOS TIPOS DE HERRAMIENTAS Y ESTRATEGIAS SON MUY COSTOSOS. SI EL FOCO PRINCIPAL ESTÁ EN EL TIEMPO DE COMERCIALIZACIÓN Y EL MAPA DE RUTA PARA LA SOLUCIÓN ES RELATIVAMENTE CORTO, ESTE PATRÓN VIABLE PODRÁ TENER UN ALTO GRADO DE BENEFICIOS."

Los patrones como duplicación de datos que son comunes dentro de las arquitecturas actuales pueden ahora ser perjudiciales para el progreso de arquitectura de su empresa. Las organizaciones que pueden ver el cambio de aplicaciones a servicios serán aquellas que harán el cambio más rápidamente y verán el potencial total del SOA.

Las metas aparentemente inalcanzables de industrialización de software rápidamente se están convirtiendo en realidad. Los patrones son los elementos fundamentales de la consistencia y agilidad en la arquitectura empresarial. Considerarlos como artefactos de arquitectura de primera clase ayudará a acelerar su descubrimiento y adopción. Con el tiempo, una empresa que cree una biblioteca reutilizable de estos patrones habrá aumentado su capacidad de respuesta a sus patrocinadores comerciales y habrá obtenido una ventaja sobre la competencia. •

Sobre los autores

Tom Fuller es un CTO y consultor de SOA senior en Blue Arch Solutions Inc. (www.BlueArchSolutions.com), un proveedor de soluciones, capacitación y asesoramiento de arquitectura en Tampa, Florida. Tom recientemente recibió el premio MVP (premio de Microsoft al profesional más valioso) en la disciplina Arquitecto de Soluciones y Desarrollos Visuales. También es el actual presidente de la parte de Tampa Bay de la Asociación Internacional de Arquitectos de software (IASA), posee una certificación de MCSD.NET, y administra un sitio comunitario dedicado a SOA, servicios Web y Fundación Windows Communication (ex "Indigo"). Posee una serie de artículos publicados recientemente en la Alianza Profesional de Software Activo y en la revista de Servidor SQL Standard. Tom es orador en muchos grupos de usuario en el sudeste de EE.UU. Visite www.SOApitstop.com para más información, y contacte a Tom a tom.fuller@soapitstop.com.

Shawn Morgan es director ejecutivo y arquitecto senior en Blue Arch Solutions, Inc. (www.BlueArchSolutions.com) Shawn realizó la arquitectura para soluciones de muchas empresas pertenecientes a Fortune 500 incluso Federal Express, Beverly Enterprises, y Publix Super Markets. Shawn se enfoca en permitir que las empresas brinden soluciones informáticas mediante la arquitectura. Contáctelo a shawn.morgan@bluearchsolutions.com.

Patrones para la composición y consumo de datos de alta integridad

Por Dion Hinchcliffe

Síntesis

El desafío es consumir y administrar datos de múltiples fuentes subyacentes en diferentes formatos mientras se participa en ecosistemas de información federada y mientras se mantiene la integridad y el acoplamiento leve con el buen rendimiento. Aquí tiene algunos patrones emergentes para el creciente mundo de las aplicaciones Web híbridas (mashups) y aplicaciones de composición.

En estos días el campo del desarrollo de software se trata tanto sobre el ensamble y composición de servicios preexistentes y datos como sobre crear nuevas funcionalidades. El mundo de los mashups en la Web y aplicaciones compuestas en el mundo de la arquitectura orientada al servicio (SOA) exigen que estos datos- en diferentes formas y prácticamente desde cualquier origen- se unan, interactúen en forma limpia, y luego sigan en caminos separados. Y estos enfoques exigen que esta interacción se produzca en forma eficaz sin perder la fidelidad o la integridad.

La considerable variedad de datos estos días incluye una gama extensa de formatos de base XML, como también formatos de datos de menor peso de creciente difusión, como el JavaScript Object Notation (JSON) y microformatos. Las aplicaciones también tienen que funcionar cada vez más con formas ricas de datos, como imágenes, audio, y video, y no debemos olvidar los formatos más antiguos, usados como caballo de batalla a diario, como el texto, EDI e incluso objetos nativos. Todos estos formatos se entretejen y combinan cada vez más, al mismo tiempo que los sistemas se vuelven más interconectados e integrados en vastas cadenas de suministro, buses de servicio empresarial (ESBs), SOAs, y mashups de Web. Mantener el orden en tal caos de datos es más importante que nunca. Lo positivo es que las normas de primera orden para administrar toda esta heterogeneidad de datos comienzan a emerger.

Los servicios Web, SOA y especialmente la Web en sí, se hacen para abrir estándares que posibiliten a los sistemas comunicarse- el HTTP omnipresente y fundamental es el ejemplo principal. No obstante, esta comunicación no permite que uno asuma qué clase de datos tendrá que consumir su software en las aplicaciones del futuro. Aunque aún existe una buena posibilidad de que se encuentre con alguna forma de XML, hay igual probabilidad de que se encuentre con uno de los formatos de datos livianos que crecen en popularidad como el JSON. No obstante, cada vez más, podría ser fácilmente texto simple; un árbol de objetos nativos; o incluso una pila de servicios Web estilo WS-* de capas profundas que sea suministrado por el inminente Windows Communication Foundation (WCF).

Los desarrolladores por lo tanto enfrentan desafíos serios en cuanto a crear buenas técnicas de modelado de datos, arquitectura e integración que abarquen esta diversidad. La heterogeneidad de las formas de representación de datos puede ser bastante intimidante en entornos con integración de sistemas de alto nivel. No importa que en sistemas altamente federados y acoplados muy levemente, como se están volviendo muchas aplicaciones, la probabilidad de cambios frecuentes sea alta. Como resultado, depende de la comunidad de desarrollo de aplicaciones crear un ente de conocimientos sobre las mejores prácticas para el consumo de datos limitado desde diferentes orígenes y formatos, con técnicas coincidentes para integrarlas, fusionarlas y relacionarlas. La comunidad también necesita asegurarse de que la integridad se mantenga al mismo tiempo que sigue resistente al cambio e incluso a la evolución inevitable de los formatos de datos subyacentes.

Aunque los patrones presentados aquí no son obligatorios y pretenden ser en principio una guía, se basan en conceptos aceptados de los contratos de interfase. Los contratos de interfase son ahora comunes en el mundo de los servicios Web con el WSDL como también en muchos lenguajes de programación pero en particular en diseño por contrato. En un contrato de interfase, un proveedor y un suministrador se unen y acuerdan una serie de interacciones, en general descriptas en términos de métodos o servicios. Estas interacciones harán que los datos de interés pasen ida y vuelta a lo largo de los límites de la interfase.

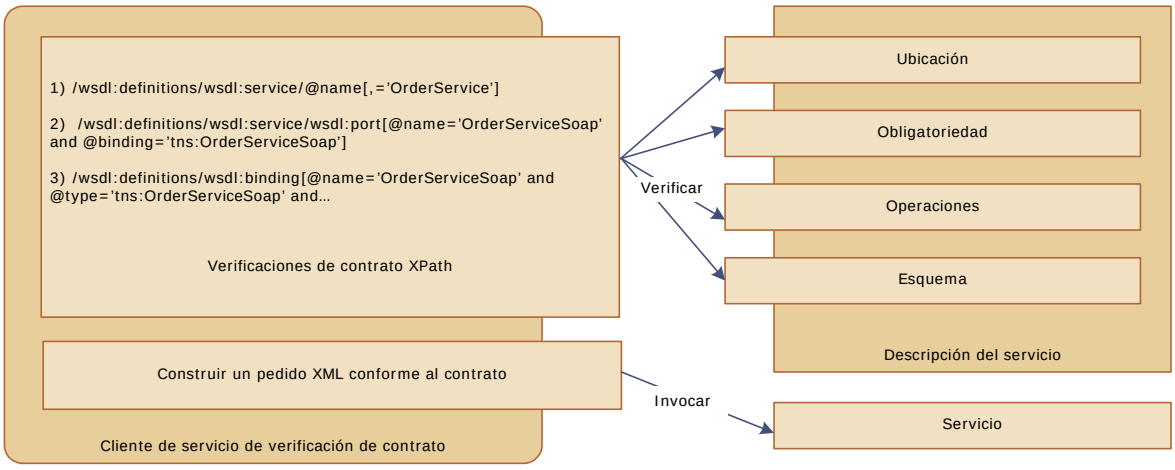
La precisa definición de las interacciones, incluso sus ubicaciones y protocolos, está justamente en el contrato, el cual es preferible que se pueda leer por máquina. De particular interés son las estructuras de datos que se pasan ida y vuelta entre suministrador y cliente como parámetros durante cada interacción. Y son estas estructuras de datos las que son realmente centro del modelo de mashup/aplicaciones compuestas. Debido a que es aquí donde estas estructuras de datos de gran diversidad se encuentran, necesitamos colocar principios de integración y recombinación de datos en una solución clara.

Fuerzas y limitaciones de consumo y composición

Junto con estas líneas, podemos obtener un sentido general de las fuerzas que dan forma al consumo y composición de datos en sistemas modernos de software distribuido. En un orden aproximado, estas son:

El contrato de interfase como impulsor de consumo y composición de datos. Al utilizar estructuras de datos de cualquier origen o servicio externo, como un servicio Web, las

Figura 1: Los esquemas integrados en un contrato de interfase son generalmente la dependencia de cliente más importante



estructuras de datos deben validarse en comparación con el contrato de interfase brindado por el servicio origen. Esta validación con frecuencia se puede realizar a tiempo de diseño para fuentes de datos que se conoce que son altamente estables y confiables. Pero en sistemas federados, especialmente aquellos no bajo el control directo del consumidor, se debe dar especial consideración a realizar una verificación del contrato en tiempo de ejecución. La verificación del contrato en tiempo de ejecución en sistemas acoplados levemente es una técnica emergente, y algunas opciones interesantes están disponibles para el consumidor del servicio a fin de evitar problemas anticipados. El asunto es que el contrato de interfase es el artefacto principal que impulsa el consumo y composición de datos.

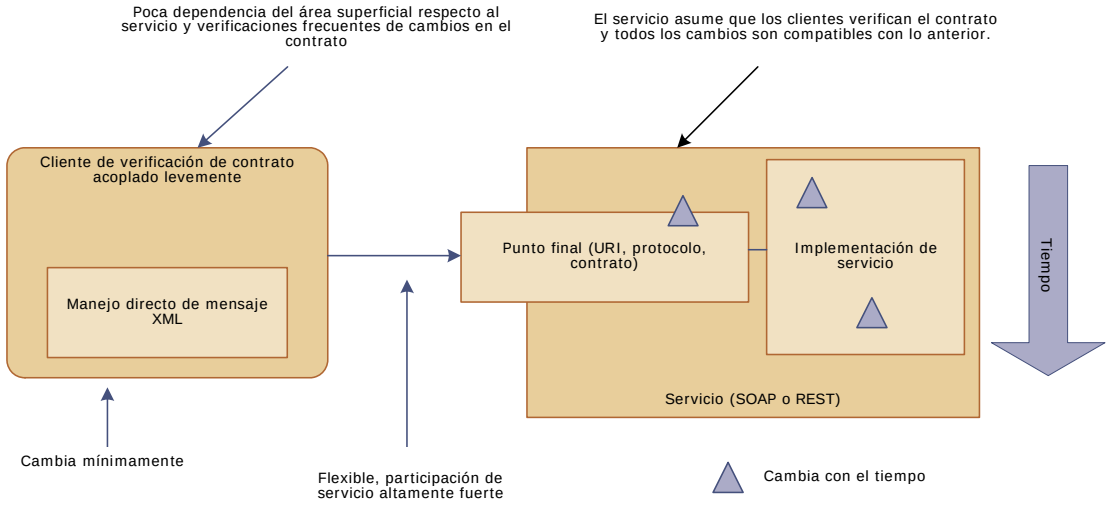
La impedancia de abstracción altera el consumo y composición de datos. El punto físico en el cual se produce la integración de datos cada vez está más fuera del control normal de las bases de datos, y las bibliotecas de acceso a datos convierten todo en una abstracción de datos unificada.

"CUANDO SE UTILIZAN ESTRUCTURAS DE DATOS DE CUALQUIER ORIGEN O SERVICIO EXTERNO, COMO SERVICIO WEB, LAS ESTRUCTURAS DE DATOS DEBEN VALIDARSE EN COMPARACIÓN CON EL CONTRATO DE INTERFASE SUMINISTRADO POR EL SERVICIO DE ORIGEN"

Los datos que se obtienen de servicios externos múltiples y en formas múltiples pueden y generalmente chocan dentro del explorador, dentro de los clientes front-end, o en el código interno del lado del servidor y fuera de la base de datos. Las estructuras de datos de estos servicios subyacentes, según la aplicación del software, van desde objetos nativos, documentos/fragmentos XML, JSON, y SOAP, a textos, datos binarios y multimedia. Aunque la impedancia de abstracción siempre ha sido un problema conocido en el software, ahora se ha exacerbado por la proliferación del número creciente de modelos disponibles para representar información. Cuando se consumen y componen estas estructuras de datos en representaciones agregadas utilizables y luego vuelven a sus servicios de origen, aparecen una cantidad de problemas serios.

- *Variabilidad del formato de los datos.* Los formatos fundamentales de las estructuras de datos subyacentes con frecuencia son altamente incompatibles, y las buenas facilidades locales para procesar todas las formas que se puedan encontrar no están generalmente disponibles. Los formatos extremadamente complejos son particularmente problemáticos ya que requieren conjuntos de protocolos sofisticados para administrar, como servicios Web de alto rango con requisitos de política WS.
- *Identidad de los datos.* Las claves e identificadores únicos de datos /objetos generalmente no están en formatos comunes y no son las facilidades disponibles para verificarlos y reforzarlos uniformemente en todas las abstracciones.
- *Relaciones de datos.* Establecer relaciones y asociaciones entre estructuras de datos de formatos muy diferentes puede ser problemático, tanto para la eficacia como para el rendimiento, por un número de razones. Convertir todos los datos a un formato común puede ser muy costoso, tanto en cuanto a mantenimiento como en costo de tiempo de ejecución. También presenta el problema de mantener el origen de los datos combinados y extraerlos de vuelta al modificarlos. Mantener las estructuras de datos en formato nativo asume mecanismos buenos para componerlos, relacionarlos y manipularlos.
- *Origen de los datos:* mantener el servicio de origen de un conjunto de datos es fundamental para mantener el contexto válido para conversaciones continuas con la fuente y para cambios a los datos en particular, como actualizaciones, agregados, y eliminaciones de datos.
- *Integridad de los datos:* modificar los datos de origen, asegurar que los cambios son válidos conforme al contrato de interfase de servicio subyacente, y no violar las definiciones del esquema es muy importante. Con documentos XML bien definidos que estén soportados por esquemas ricos XML, estas validaciones son un problema menor. No obstante, los modelos de programación de menor peso como el JSON y otros similares generalmente no poseen ningún esquema de lectura por máquina. Estos tipos de formatos de datos, aunque generalmente solo están disponibles en un servicio federado, tienen el mayor riesgo de problemas de integridad porque las normas para cambios y otro tipo de manipulación de estos no están tan bien definidas como los otros formatos.
- *Conversión de datos.* Con frecuencia, no hay conversiones bien especificadas entre formatos de datos, o, lo que es peor, las opciones múltiples se presentan para alterar los datos de manera sutil. Los datos más frecuentemente afectados son las representaciones numéricas, pero estas conversiones también afectan una fuente de datos en conjuntos de caracteres múltiples, datos de audio/video, y datos GIS con frecuencia.

Figura 2: Verificación de tiempo de ejecución del contrato de interfase en servicios Web



Áreas superficiales grandes de contrato de interfase. Los esquemas extremadamente complejos y las estructuras de interfase están aumentando para convertirse en uno de los principales problemas en la arquitectura de datos de sistemas distribuidos. La experiencia ha probado una y otra vez que lo más sencillo es más confiable y de mayor calidad. Pero los contratos de interfase de servicios distribuidos generalmente tienen esquemas XML y definiciones WSDL grandes y complejas. La probabilidad de que haya cambios, en forma inadvertida o no, aumenta con la complejidad de un contrato de interfase y los esquemas que abarca. Aunque los enfoques simples XML y REST para servicios tienden a fomentar la sencillez deseada a veces ciertos dominios de aplicaciones no son "sencillos", y estos modelos de servicio están sujetos a problemas adicionales por la falta de normas de contrato de interfase. La conclusión es que la probabilidad que cambie el contrato sin que un suministrador remoto sepa los cambios aumenta con el tamaño del contrato total, porque es más probable que los cambios a los esquemas grandes no sean detectados que en el caso de los esquemas menores.

Cambios implícitos de versión. Incluso en entornos bien controlados, los cambios a los servicios y sus contratos de interfase se pueden producir sin notificar a todas las partes dependientes. Salvo los cambios a la manera en que los servicios de datos funcionan actualmente, este patrón solamente se volverá más común en los entornos altamente federados del presente y aquellos del futuro. Sin ningún cambio, los servicios de los cuales usted depende cambiarán sin su conocimiento o previa notificación, y tendrá que aceptar este hecho como práctico, inevitable, y desafortunado. En consecuencia, desarrollar estrategias conscientes para detectar cambios de versión y manejarlos en forma efectiva es fundamental para mantener la calidad de los datos y funcionalidad de la aplicación compuesta o mashup.

Patrones de consumo y composición de datos

Los patrones descriptos en el presente pretenden presentar una postura liviana sobre el modelado de datos y arquitectura de aplicación. Estas observaciones son el resultado de analizar el mundo dinámico de mashups y aplicaciones compuestas de Web que han prosperado en la Web por varios años. Los enfoques minimalistas y ágiles utilizados por los mashups en particular son una inspiración pero en sí generalmente no son suficientes para crear software de calidad. Estos patrones pretenden capturar el espíritu de los mashups, colocarlos en contexto con buenas prácticas de ingeniería de software, y comenzar un debate en la comunidad de software sobre estos métodos difíciles para conectar los servicios y los datos.

Patrón 1- Dependencia mínima del área superficial con respecto al contrato de interfase. Este patrón es el aspecto del punto de vista de usuario de la conocida máxima de diseño de software "Sea liberal en lo que acepta, conservador en lo que envía". La mayoría de las herramientas de servicios Web, marcos de base de datos relacional, y bibliotecas multimedia fomentan la dependencia de gran parte del contrato de interfase, generalmente de todo el contrato, independientemente de lo que depende el software. Para muchos casos de consumo y composición, una pequeña dependencia del área superficial es realmente todo lo que se necesita. Una dependencia total del contrato presenta una tremenda fragilidad, porque los cambios que no se relacionan con los datos en la estructura de datos de la cual se depende, afectarán igualmente al cliente. Aunque algunas herramientas de consumo de datos ya perdonan esta dependencia, generalmente hay poco control sobre qué partes son de interés para las herramientas.

Para muchas aplicaciones, solo las dependencias directas de los elementos de datos internos que se necesitan son las adecuadas. Todas las otras dependencias deberían quitarse activamente de la relación de la dependencia dentro de los datos fuente. El resultado aquí es que los cambios al contrato de interfase que no son de importancia para el consumidor de datos no debe evitar el uso de los datos. La conversión también debe ser real; los cambios al contrato que importan deben ser aparentes de inmediato para evitar el comportamiento y uso incorrecto de los datos. Los ejemplos de dependencia mínima incluyen: XML, caminos clave a través de esquemas a elementos de datos; datos relacionales, solamente las tablas, tipos, columnas, y los índices utilizados por el cliente; y multimedia, solo las partes de la estructura de medios que se necesitan como los canales específicos, tasas de bits, partes de imagen, segmentos de video, y demás.

Patrón 2- Verificación de los cambios en el contrato de tiempo de ejecución. Los servicios de los cuales dependen los mashups y aplicaciones compuestas para los datos están sujetos a cambio imprevisto en cualquier momento. Este cambio podría ser por movimiento del punto final a una nueva ubicación, detención de uso de un protocolo previamente soportado, cambios al esquema subyacente, o incluso solamente cambios de interfase deliberados directos o fallas de servicio interno. La razón lleva a la necesidad de verificar el contrato para detectar los cambios al mismo. También exige poder manejar muchos cambios en el contrato de manera adecuada ya que estos pueden no afectar la parte del contrato que a usted le interese.

En la realidad, existen dos tipos de verificación de contrato de tiempo de ejecución. Uno es el de *verificar las especificaciones mismas del contrato*, si existe alguna. Esta especificación con frecuencia es el WSDL u otro metadato que oficialmente se publica junto con el servicio mismo. Estos se pueden verificar fácil y rápidamente con una variedad de técnicas de programación, incluso consultas XPath para esquemas de base XML u otras técnicas livianas relativamente. Las verificaciones de contratos con codificación difícil en lenguaje tradicional de programación no son fáciles de implementar o cambiar y deberían ser la segunda opción.

La segunda verificación de contrato es realizar una validación del contrato con respecto a las instancias de datos que provee el servicio. Los contratos de interfase generalmente tienen su propia impedancia de abstracción con sus mecanismos de entrega, y puede ser sorprendente la frecuencia con la cual las inconsistencias treparán entre los datos provistos y el contrato de interfase, incluso con herramientas de alta calidad.

El mayor dilema presentado por este patrón es la frecuencia de verificación "del contrato" en sí. Verificar el contrato y los datos de instancia con cada recuperación de datos de su fuente puede consumir tiempo y generalmente es innecesario. Finalmente, determinar la frecuencia de verificación de contrato depende ampliamente de los requisitos de aplicación y la naturaleza de los datos y su tolerancia para la imprecisión y la falta de corrección. Para muchas aplicaciones, verificar sincrónicamente puede no ser siquiera una opción, y puede tener sentido fijar un proceso de antecedentes periódico para identificar los cambios en el contrato, que se producirán en forma inevitable.

Patrón 3- Reducción de estructuras a formato de abstracción de datos común. En la realidad, la diversidad de las estructuras y formatos de datos con lo cual tienen que trabajar los mashups y aplicaciones compuestas solamente continuará creciendo. El software puede manipularlos en sus formatos nativos de datos, lo cual significa perder la oportunidad de mantener relaciones y cumplir las normas comerciales, o puede convertirlos a todos en una abstracción común. Esta conversión es un enfoque que las bibliotecas de datos como ADO.NET utilizan con sus DataSets y XML lo hace con otras fuentes de datos que no son XML. Subsumir las diferencias en datos fuente convirtiéndolos una y otra vez en abstracción de datos común que provee un modelo simple unificado para trabajar con él, puede ser atrayente por muchas razones. Primero, las relaciones entre las diferentes fuentes de datos subyacentes se pueden verificar y cumplir al manipularse y cambiarse los datos. Segundo, las vistas de los datos pueden tener ventajas del mecanismo de abstracción haciendo menos necesaria la construcción de mecanismos MVC (Controlador-de vista- modelo) y la utilización de bibliotecas existentes y marcos que pueden procesar la abstracción.

No todos los conjuntos de estructuras de datos formateados en forma heterogénea son buenos candidatos para este patrón, y puede tener poco sentido para algunos formatos de datos con niveles grandes de impedancia de abstracción-datos de imagen con datos de no imagen, por ejemplo. Hay también un costo potencialmente no trivial de transformación y conversión de datos fuera y dentro del formato común. Para muchas aplicaciones, sin embargo, este costo es totalmente aceptable.

Dentro de ciertos entornos de consumo de datos, como el explorador Web, hay limitadas opciones para mantener el formato de datos común, y JSON y SML DOM tienden a ser bastante populares para las soluciones basadas en el explorador. Del lado del servidor, las opciones son mejores pero dependen de la plataforma. Las estructuras XML, bibliotecas O/R como Hibernate, bases de datos relacionales, e incluso gráficos de objetos nativos con frecuencia son excelentes modelos de abstracción común según la aplicación. Pero la naturaleza diferente de datos

"LA WEB MISMA SE ESTÁ VOLVIENDO EL SUMINISTRADOR MÁS IMPORTANTE DE DATOS FEDERADOS, UNA FUENTE QUE CRECERÁ Y SE VOLVERÁ MÁS IMPORTANTE EN LOS PRÓXIMOS AÑOS".

jerárquicos como XML y gráficos de objeto es uno de los problemas clásicos de impedancia en computación, y se debe tener cuidado al utilizar este patrón.

La conclusión es: si el rendimiento no es absolutamente crítico y los formatos y esquemas de datos subyacentes son susceptibles, este patrón puede ser muy poderoso para trabajar con fuentes de datos federados. La desventaja es que el enfoque de abstracción común puede ciertamente implicar más mantenimiento y aceptar la fragilidad ya que el mapeo y los metadatos deben conservarse.

Patrón 4- Las estructuras nativas mediadas con el Controlador de vista modelo (MVC). Convertir a todos los datos fuente en un formato común no será una opción en muchos casos, porque 1) el gasto del proceso es excesivo ya que muchos de los datos podrían no utilizarse; 2) duplicarlos en el entorno local puede estar prohibido en cuanto a recursos. O podría ser porque no hay un formato común que tenga sentido para todos los tipos de datos subyacentes. En este caso, crear un MVC que medie el acceso, traducción e integridad de los datos con las estructuras nativas subyacentes puede dar las mejores opciones para el rendimiento y almacenamiento de datos.

MVC es un patrón de diseño poderoso en su propio derecho que ha probado una y otra vez ser una excelente estrategia para la separación de los problemas en software de aplicación. Su uso es particularmente adecuado cuando hay múltiples modelos de datos subyacentes en una determinada aplicación. El buen diseño de software dictamina que ofrecer una visión unificada de los datos fuente provee una interfase sencilla, clara y consistente que facilita la visión, interacción y modificación de los datos subyacentes. El acceso a los datos subyacentes con el enfoque MVC es también relativamente eficaz porque solo los datos necesarios se deben procesar para satisfacer la mayoría de las solicitudes.

Aunque hay muchas ventajas con el MVC, igualmente las desventajas son similares a las del patrón 3 en que el mantenimiento del código MVC puede ser enorme. Ciertamente hay un mayor número de bibliotecas disponibles que pueden ayudar a los diseñadores de software a construir MVC sobre el cliente y el servidor. Tenga en cuenta, igualmente: el código de mapeo es frágil y tedioso.

Patrón 5- acceso directo de estructura de datos nativos. Para muchas aplicaciones, especialmente las más simples basadas en exploradores, convertir datos fuente en formatos comunes o

Tabla 1: Abstracciones de datos comunes

Abstracción	Norma de contrato	Ventajas	Desventajas
Texto, JSON, binario	Informal, textual	Relativamente eficaz	No auto-descriptivo, no idealmente eficaz
Objetos nativos	Definición de clase	Comportamiento y datos unificados, encapsulamiento, abstracción de alto nivel, y composición	Exige la conversión de la mayoría de los datos en objetos, exigiendo una técnica de mapeo.
XML	Esquemas XML (XSD), Relax NG, WSDL y muchos otros.	Autodescriptivo, buena descripción del esquema, extensible sin anular la compatibilidad con productos anteriores.	No eficaz en tamaño, no hay manera de distribuir conducta, y la descripción del esquema es limitada incluso con XSD
Imágenes	Especificaciones para JPEG, TIFF, GIF, BMP, PNG y muchos otros.	No aplica	No aplica
Audio	Especificaciones para WAV, WMA, MP3 y AAC.	No aplica	No aplica
Video	Especificaciones para AVI, QuickTime, MPEG, y WMF	No aplica	No aplica

construir sofisticada arquitectura MVC no es una buena opción. El acceso directo a los datos tiene más sentido, y la decisión generalmente tiene sentido común, relacionada con las bibliotecas disponibles. Además, como lo mencionamos antes, lo más sencillo es generalmente de mejor calidad porque hay menos para quebrar o mantener.

En este patrón, que funciona mejor con datos menos estructurados, las estructuras de datos nativos se usan directamente sin un intermediario o conversión de datos, lo cual significa que los almacenes de datos en texto, XML, JSON y demás se manipulan a nivel nativo. Lo malo, desde luego, es que no se puede basar en las facilidades que las bibliotecas de abstracción de datos le pueden dar para reforzar la integridad o rastrear los cambios. No obstante, este patrón exige generalmente la cantidad mínima de procesamiento o aprendizaje de bibliotecas de terceros. Puede ser fácil de desarrollar, y al no haber conversión o capas de acceso de datos para atravesar, es también bastante rápido.

Patrón 6- Toda la modificación de datos es atómica. Las vistas más sofisticadas de datos y servicios prescriben propiedades conocidas como ACID, para atomicidad, concurrencia, aislamiento, y durabilidad. Generalmente adscripta a sistemas de base de datos, ACID es una regla de uso común excelente y práctica para casi cualquier sistema de acceso a datos concurrentes. Lamentablemente, casi nadie en el mundo de los servicios Web y mashups tiene la noción de las transacciones en sus protocolos, lo cual conferiría muchas de las propiedades de ACID a la modificación de datos. Lejos de ignorar el problema, los desarrolladores de aplicación compuesta y mashup deben saber con agudeza cómo trabajar sin una cuerda floja cuando trabajan con sus datos.

Este conocimiento presenta problemas significativos con casos complejos de modificaciones de datos, uno es que cualquier recuperación de datos y almacenamiento que dependa de una conversación extendida con servicios subyacentes casi ciertamente no estará protegida por la misma limitación de transacciones y propiedades de ACID relacionadas. El código de software debe prever que se produzcan los problemas de integridad de datos, especialmente en una conversación extendida que puede nunca completarse o quedarse a mitad de camino. Evitar las conversaciones prolongadas es una opción. Hacer dependiente la operación de software, lo más posible, de la interacción atómica individual con servicios subyacentes es otra forma. Cada paso en la interacción es un éxito discreto y visible que permite al software ofrecer a sus usuarios opciones claras cuando falla una conversación con un servicio de datos de suministrador. Estas dos opciones permiten a los desarrolladores de software respetar la regla ACID de uso común.

Un regreso a la simplicidad

La Web misma se está volviendo el más importante proveedor de datos federados, una fuente que crecerá y se volverá más importante en los próximos años. Aunque los formatos de datos XML y livianos probablemente sean la estructura de datos dominante con la cual la mayoría del software tendrá que trabajar en el futuro previsible, pueden aparecer sorpresas. Estas sorpresas incluyen optimizaciones necesarias en XML como XML binario, avances en microformatos, nuevos codecs multimedia que cambiarán rápidamente un paisaje audio visual íntegro, y protocolos de transporte completamente nuevos como Bittorrent, que hará a muchos de los patrones problemáticos, por decirlo de alguna manera.

Un regreso a la simplicidad en diseño de datos volvió a estar de moda, ejemplificado por el aumento de interés en los formatos como JSON, microformatos, y lenguajes dinámicos como PHP y Ruby. Esta sencillez de diseño puede hacer a los datos más maleables y fáciles de conectar entre sí. También permite menor dificultad para escribir software y administrarlo. Mientras los patrones presentados aquí están planteando unos que pueden llevar a consumo y composición de datos de alta integridad, acoplados más levemente y menos frágiles, en realidad la historia recién comienza. Al relacionarse menos la Web con páginas Web visuales y más con servicios y datos y contenidos puros, hacerse adepto a ser un ágil consumidor y proveedor del ecosistema de información será un factor de éxito cada vez más fundamental. •

Recursos

Microformatos http://microformats.org

Wikipedia
http://en.wikipedia.org/wiki/design_by_contract

"La discrepancia de impedancia entre modelos conceptuales y entornos de implementación", Scott N. Woodfield, Departamento de computación, Brigham Young University (ER 97 y Scott N. Woodfield, 1997)
http://osm7.cs.byu.edu/ER97/workshop4/sw.html

Hewlett-Packard Development Company
Informes técnicos
"Volver a pensar en el paquete SOAP de Java", Steve Loughran y Edmund Smith www.hpl.hp.com/techreports/2005/HPL-2005-83.html

Sobre el Autor

Dion Hinchcliffe es jefe de tecnología en Sphere of Influence, Inc.

Modelado de Base de Datos Relacional/Objeto Nordic

Por Paul Nielsen

Síntesis

El nuevo Modelado de Base de Datos Relacional/Objeto (Nordic), al igual muchas herramientas, no es la solución más conveniente ni la más apropiada para cada tipo de problema de base de datos. Sin embargo, el modelo híbrido relacional/objeto puede proporcionar más eficacia, mayor flexibilidad, mejor rendimiento y aún una mayor integridad de datos que los modelos relacionales tradicionales, en particular para las bases de datos que se benefician de la herencia, la extracción de datos creativa, interacciones de clase flexible o restricciones de flujo de trabajo. Descubra alguna de las innovaciones posibles cuando la tecnología orientada al objeto se modela utilizando las bases de datos relacionales experimentadas de la actualidad.

Las diferencias entre el desarrollo orientado al objeto y el modelo de base de datos relacional crea una tensión por lo general conocida como *incompatibilidad de impedancia relacional-objeto*. La herencia simplemente no se traduce bien en un esquema relacional. La incompatibilidad de impedancia técnica se agrava por la desconexión cultural entre codificadores de aplicaciones y administradores de bases de datos (DBAs). Por lo general, ninguna parte comprende totalmente ni respeta el léxico del otro. Una forma segura de ponerse en la piel de los DBAs es referirse a la base de datos como la "utilidad de persistencia de objetos". Esta relación es desafortunada porque cada lado aporta un único conjunto de ventajas al problema de arquitectura de datos.

De varias maneras el diseño orientado al objeto es superior al modelo relacional. Por ejemplo, diseñar herencias entre clases se puede lograr utilizando el patrón supertipo/subtipo relacional, pero el diseño orientado al servicio es una solución más refinada. También, casi todo el código de la aplicación está orientado al objeto y una base de datos orientada al objeto se integraría con la aplicación más fácilmente de lo que lo haría una base de datos relacional.

Tan sofisticada como es la tecnología orientada al objeto al modelar realidad, el lado relacional no lo es sin ventajas importantes. Los motores de la base de datos relacional ofrecen rendimiento, escalabilidad y alta disponibilidad de opciones y el músculo financiero para asegurar que la plataforma de base de datos aún continúe existiendo en unas pocas décadas. La tecnología relacional es bien comprendida, y las bases de datos relacionales ofrecen herramientas más eficaces de suministro de datos y consulta que las bases de datos orientadas al objeto. Las pocas empresas disponibles de bases de datos orientadas al objeto puro simplemente no poseen los recursos para competir con Microsoft, Oracle o IBM.

Entonces, el enigma de la incompatibilidad de impedancia racional/objeto es cómo adoptar de la mejor manera la elegancia de tecnologías orientadas al objeto al mismo tiempo que se mantienen la eficacia, flexibilidad y estabilidad a largo plazo de un motor de base de datos relacional experimentado. Debido a que los programadores de aplicaciones son por lo general los más interesados en resolver este problema y la naturaleza humana tiende a resolver problemas utilizando el conjunto de capacidades más cómodo, no es sorprendente que la mayoría de las soluciones sean implementadas en una capa de mapeo entre la base de datos y el código de la aplicación que traduce objetos en tablas relacionales para los objetos persistentes.

La propuesta de Nordic

Propongo que un modelo relacional, diseñado para emular funciones orientadas al objeto pueda desempeñarse perfectamente bien dentro de los motores de bases de datos relacionales de la actualidad y que la manipulación de la herencia de clase y las asociaciones complejas directamente en la base de datos, cerca de los datos, es de hecho muy efectiva. Esta efectividad no siempre fue el caso. Me contrataron para optimizar el diseño de una base de datos Objeto/Relacional, que utilizaba de forma intensiva

"TAN SOFISTICADA COMO ES LA TECNOLOGÍA ORIENTADA AL OBJETO AL MODELAR REALIDAD, EL LADO RELACIONAL NO LO ES SIN VENTAJAS IMPORTANTES.".

Transact-SQL (T-SQL) y estaba implementada con el Servidor SQL 6.5 , la cual falló. El desarrollo del Diseño de la Base de Datos Relacional/Objeto Nordic implicó un año de repeticiones, un esquema de metadatos simplificado y el desarrollo de un T-SQL del Servidor SQL más avanzado.

Al igual que con cualquier proyecto de base de datos, se necesita una capa de abstracción de datos estrictamente impuesta que encapsule la base de datos para la extensibilidad a largo plazo. Para una base de datos híbrida O/R, la capa de abstracción de datos también proporciona la fachada para las funciones orientadas al objeto. Detrás del código de la fachada se encuentran el esquema de metadatos y la generación del código para las clases, objetos y asociaciones (Ver Figura 1). Si se implementa esta solución existen pocas decisiones de diseño claves.

Gestión de clase. Dentro del esquema relacional, los metadatos de atributo y de clase se modelan con facilidad utilizando una relación común uno-a-muchos. La relación de subclase/superclase se modela como un árbol jerárquico, utilizando ya sea el patrón de la lista de adyacencia más común o el patrón de la ruta materializada más efectiva.

Navegar toda la jerarquía de la clase con funciones definidas por el usuario permite que las consultas SQL se unan fácilmente con cualquier descendiente de la clase o clases ascendentes. Estas funciones definidas por el usuario se utilizan en toda la capa de

fachada. Por ejemplo, cuando se seleccionan todas las propiedades para una clase, juntarlas con la función definida por el usuario superclass() devuelve todas las superclases, y la consulta puede entonces seleccionar todas las propiedades de una determinada clase incluyendo las propiedades heredadas de las superclases.

En el contexto de trabajo con objetos permanentes, *polimorfismo* se refiere a la capacidad del método selecto para recuperar no sólo los objetos de clase actuales sino también todos los objetos de la subclase. Por ejemplo, la selección de todos los contactos debe seleccionar no sólo los objetos de la clase del contacto sino también los objetos de las subclases del cliente y el cliente principal. Una función definida por el usuario que devuelve una variable de tabla de todas las subclases de una determinada clase hace que el procedimiento almacenado y la escritura de esta consulta sean eficaces y reutilizables.

Gestión del objeto. Los objetos se modelan mejor utilizando una tabla de objeto única que almacene los datos comunes de los objetos como el objetid único, clase del objeto, datos de auditoría y algunos atributos de búsqueda comunes a casi todas las clases como por ejemplo nombre, atributo de datos y demás. Los atributos adicionales se almacenan en tablas de clase personalizada que utilizan una clave ajena del objetid para relacionar los atributos personalizados a la tabla del objeto. El *createclass* y otros procedimientos de fachada almacenada de gestión de clase ejecutan el código del Lenguaje de Definición de Datos (DDL) para crear o modificar las tablas de clase personalizada y generar el código de fachada personalizada para los objetos de selección, inserción y actualización.

La decisión de diseño más importante para modelar la forma en la que los objetos se almacenan es cómo representar los datos de atributo personalizado. Existen tres métodos posibles: el patrón de valor par, tablas de clase personalizada concreta y tablas de clase personalizada en cascada. El *patrón de valor par*, también llamado *patrón genérico*, utiliza un patrón en forma de diamante que consiste de clase, propiedad, objeto y valor. La tabla de valor usa una columna única para almacenar todos los valores. Esta tabla estrecha y larga utiliza una fila para cada atributo. Diez millones de objetos con quince atributos usarían 150 millones de filas en la tabla de valor. El Servidor SQL es mucho más que capaz para trabajar con grandes tablas –esto no representa un problema. Este modelo parece ofrecer la mayor flexibilidad ya que los atributos pueden agregarse sin modificar el esquema relacional; sin embargo, este modelo se ve afectado de inexistencia, o en el mejor de los casos, edición de datos un poco incómoda y es difícil de consultar utilizando SQL.

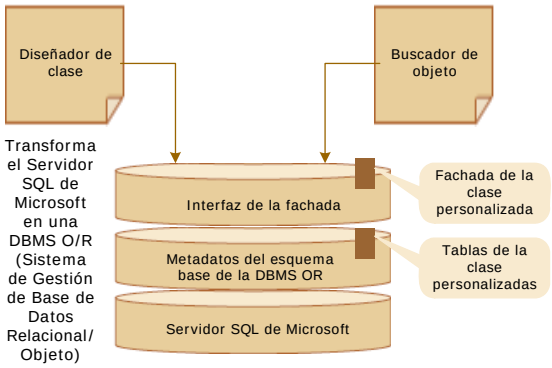
Una tabla para cada clase

El modelo de *clase personalizada concreta* usa una tabla para cada clase con columnas para cada atributo personalizado incluyendo los atributos heredados no abstractos. Un objeto existe sólo en dos tablas –la tabla del objeto y la tabla de clase personalizada concreta– mientras que los atributos se duplican en cada tabla de clase personalizada concreta de la subclase. Por lo tanto, si la clase animal posee un atributo fecha de nacimiento, y la subclase mamífero posee un atributo sexo (ya que algunos animales no poseen sexo), entonces la tabla de clase personalizada mamífero incluye las columnas de objetid, fecha de nacimiento y sexo.

Este patrón posee la ventaja que seleccionar todos los atributos para una clase determinada requiere sólo unir la tabla de metadatos del objeto con una tabla de clase personalizada única. La desventaja es implementar el polimorfismo; seleccionar todos los animales requiere realizar una unión de cada subclase y eliminar los atributos de la subclase o agregar atributos de superclase sustitutos para que todas las selecciones en la unión tengan columnas compatibles.

Una mejora en el patrón de valor par, las tablas de clase personalizada concreta, utilizan una columna relacional para cada atributo, entonces la edición de datos del atributo puede

Figura 1: El diseño híbrido O/R utiliza una fachada para encapsularla funcionalidad orientada al objeto que se ejecuta dentro de un esquema de base de datos relacional.



implementar con facilidad los tipos de datos nativos de la base de datos relacional del host.

La tercera opción implementada en el Modelado de Bases de Datos Relacional/Objeto, *tablas de clase personalizada en cascada* utiliza una tabla para cada clase al igual que la solución de clase concreta. No obstante, en vez de duplicar atributos, cada atributo se representa sólo una vez en su propia clase, y cada objeto se representa una sola vez en cada clase en cascada. Utilizando el ejemplo animal y mamífero, la tabla animal contiene el objetid y la fecha de nacimiento, y la tabla de mamífero está compuesta del objetid y el sexo. Se almacena una instancia del objeto mamífero en los metadatos del objeto, la tabla animal y la tabla mamífero.

El polimorfismo es muy fácil en esta opción, sin embargo, se necesitan más uniones para seleccionar todos los atributos de las subclases. Ya que el Servidor SQL está optimizado para las uniones, las tablas de clase personalizada en cascada funcionan muy bien. Al igual que con las tablas de clase personalizada concreta, la edición de datos masiva está soportada por la base de datos del host.

Gestión de asociación. Las asociaciones de la tecnología orientada al objeto son muy similares para las restricciones claves ajenas de la tecnología de base de datos relacional y es ciertamente posible definir asociaciones dentro de un modelo híbrido O/R agregando atributos claves ajenos y asignando restricciones de integridad referencial declarativa relacional a las tablas de clase personalizada.

Sin embargo, almacenar cada objeto en una tabla única ofrece algunas alternativas estimulantes para modelar asociaciones. En el caso en el que una base de datos relacional normalizada puede contener docenas de claves ajenas, cada una con una columna y una tabla de clave ajena diferente, y cada una hace referencia a una clave primaria distinta, una tabla de asociación híbrida O/R necesita hacer referencia sólo a una tabla única. Cada relación de clave ajena en la base de datos puede generalizarse dentro de una asociación única *objectid* hacia una clave ajena object.objectid.

La tabla de asociación puede diseñarse como una lista emparejada (ObjectA_id, ObjectB_id), pero este diseño está mucho más limitado por las recopilaciones complejas y las consultas deben identificar objectA y objectB. La alternativa más flexible utiliza una lista object-association que consta de un único objectid para referirse al objeto y un associationid para agrupar los objetos asociados. Una columna associationtypeid puede hacer referencia a los metadatos de la asociación que describen la asociación y tal vez proporcione restricciones para la asociación.

Figura 2: Una consulta de triple unión genérica ubica todos los objetos asociados sin importar la clase


```
SELECT PO.ObjectCode, SO.ObjectCode
  FROM Object (PO)
    JOIN Association (A1)
      ON PO.ObjectID = A1.ObjectID
    JOIN Association (A2)
      ON A1.AssociationID = A2.AssociationID
    JOIN Object (SO)
      ON SO.ObjectID = A2.ObjectID
```

Una sola tabla para cada objeto y otra tabla para cada asociación suena radicalmente diferente que un esquema normalizado y lo es. No obstante, la estructura física no es un problema; el Servidor SQL sobresale en lo que refiere a tablas estrechas largas, y este diseño permite la optimización de índices agrupados y no agrupados, lo que proporciona un alto rendimiento.

Encontrar todas las asociaciones

El patrón de la lista de asociación de objetos ofrece posibilidades sorprendentes para el diseño de base de datos. Primero, unir un objeto con una cantidad n de otros objetos de cualquier clase siempre utiliza la misma consulta (Ver Figura 2). Agregar tablas adicionales a una consulta relacional añade *n*-1 uniones, pero la lista de asociación de objetos utiliza de forma consistente las mismas tres uniones sin tener en cuenta la cantidad de clases implicadas. Dependiendo de la aplicación, este estilo de relacionar objetos puede escalar considerablemente mejor que un modelo relacional normalizado. Debido a que se agregan nuevas clases al modelo de datos o a la asociación, éstas se incluyen automáticamente en las consultas "encontrar todas las asociaciones" sin modificar ningún código existente.

Encontrar todas las asociaciones entre clases, o todos los objetos que no participan en ninguna asociación con otra clase, o aún alguna otra consulta creativa de extracción de datos importante, son todas consultas basadas en conjuntos reutilizables y triviales. Las funciones definidas por el usuario del Servidor SQL pueden encapsular trabajando con asociaciones y con las varias combinaciones lógicas de objetos que participan o no en las asociaciones.

Las recopilaciones complejas también son posibles con la lista de asociación de objetos. Por ejemplo, una recopilación de aulas de clase puede incluir un aula, uno o más instructores, uno o más escritorios, currículum y uno o más estudiantes tal como se define en los metadatos de asociación.

Las asociaciones generalizadas abren más posibilidades. Las páginas Web también se vinculan utilizando un método generalizado; cada hipervínculo usa una etiqueta de anclaje y un URL. Es esencialmente una lista de asociación de objetos incrustada dentro del código HTML. Es trivial mapear de forma gráfica una página Web y visualizar la navegación entre las páginas Web sin tener en cuenta el contenido de la página Web. Del mismo modo, es trivial saltar entre las tablas de asociación y el objeto e instantáneamente ubicar objetos asociados por diversos grados de separación sin importar la clase.

En la actualidad estoy desarrollando una base de datos de gestión de patrocinio de niños para ayudar a las organizaciones a combatir la pobreza. Utilizando la lista de asociación de objetos, una función única definida por el usuario que encuentra asociaciones para cualquier objeto puede encontrar que Joe patrocina un niño en Perú, tiene programado asistir a una reunión sobre pobreza, escribió tres cartas al niño, le envió un regalo el año pasado e investigó sobre un niño en Rusia.

Buscar la lista de asociación de objetos para múltiples grados de separación también manifiesta que Greg tiene programado visitar la ciudad en Perú en la que vive el niño que patrocina Joe, 14 otras personas asistieron a la reunión que asistió Joe y tres de ellos patrocinan niños en Perú. Esta flexibilidad para cualquier objeto con una consulta es imposible de lograr con un diseño relacional.

Estado de flujo de trabajo del objeto

La generalidad de la lista de asociación de objetos permite a otra innovación de base de datos –la integración del estado de flujo de trabajo del objeto dentro de la base de datos. Si bien el estado de flujo de trabajo no se aplica a todas las clases, para algunas clases, el flujo de trabajo es una dimensión de integridad de datos omitida en el modelo de base de datos relacional.

Un flujo de trabajo típico para una orden podría ser un carrito de compras, orden confirmada, pago confirmado, inventario asignado, en proceso, listo para despachar, y despachado. Una clave ajena relacional sólo restringe la tabla secundaria para hacer referencia a un valor de clave ajena primaria válida. Si se utiliza una base de datos relacional, se puede crear una fila *shipdetail* que haga referencia a la *orden* independientemente de su estado

"SI SE DEFINEN LOS ESTADOS DE FLUJO DE TRABAJO HEREDABLES COMO PARTE DE LOS METADATOS DE LA CLASE, LOS METADATOS DE LA ASOCIACIÓN PUEDEN RESTRINGIR LOS OBJETOS A ESTADO DE FLUJO DE TRABAJO Y CLASE INTEGRANDO EL FLUJO DE TRABAJO DENTRO DE LOS DATOS DEL OBJETO."

de flujo de trabajo. El código personalizado debe confirmar que la orden ha completado ciertos pasos antes del despacho.

Si se definen los estados de flujo de trabajo heredables como parte de los metadatos de la clase, los metadatos de la asociación pueden restringir los objetos a estado de flujo de trabajo y clase integrando el flujo de trabajo dentro de los datos del objeto.

He resaltado algunas de las innovaciones posibles al modelar la tecnología orientada al objeto utilizando las bases de datos relacionales experimentadas de la actualidad. Al igual que con cualquier herramienta, el Diseño de Base de Datos Relacional/Objeto Nordic no es la mejor solución para todos los problemas de base de datos; sin embargo, para las bases de datos que se benefician de la herencia, extracción de datos creativa, interacciones de clase flexibles o restricciones de flujo de trabajo, el modelo híbrido O/R puede proporcionar más eficacia, flexibilidad, rendimiento y aún más integridad de datos que los modelos relacionales tradicionales. •

Sobre el Autor

Paul Nielsen es MVP del Servidor SQL, autor de las series *SQL Server Bible* (Wiley, 2002) y está escribiendo *Nordic Object/Relational Design in Action* (Manning), que se publicará este año. Ofrece workshops sobre la optimización y el diseño de base de datos y puede contactarlo a través de su página Web: www.SQLServerBible.com

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

No importa como diga la palabra “Arquitectura”, ahora puede acceder a *The Architecture Journal* en 8 idiomas.

Nos complace anunciar que la Edición 7 de *The Architecture Journal* está disponible al público en Inglés, Español, Portugués brasilero, Francés, Alemán, Chino simplificado, Japonés y Coreano.

Para acceder a las versiones traducidas del Journal, visite **http://www.architecturejournal.net** y haga un clic sobre alguno de los idiomas que se enumeran en la barra superior para descargar el PDF

www.architecturejournal.net !

Microsoft

ARC

Adoptar y beneficiarse de procesos ágiles en el Desarrollo de Software en el exterior

Por Andrew Filev

Síntesis

En el desarrollo de software existen dos tendencias que permiten a las personas obtener más por menos: el desarrollo ágil y la tercerización en el exterior. Veamos cómo y cuándo combinar ambas de forma exitosa para incrementar la competitividad de los negocios.

Durante la etapa posterior a la burbuja, se cortaron más los presupuestos IT de lo que demandas había por sus servicios, lo que incitó a los gerentes a buscar soluciones más rentables y fortaleció la tendencia de tercerizar el desarrollo del software hacia países con mercados emergentes (desarrollo en el exterior). La fuerza orientadora económica no es la única fuerza para esta tendencia. El reciente crecimiento rápido provocado por una infraestructura de comunicaciones mejoradas, también juega un papel importante.

Con respecto a trabajar en equipos distribuidos en general y a la tercerización en el exterior en particular, el amigable software de Protocolo de Voz sobre Internet (VoIP), las mensajerías instantáneas, clientes de correo electrónico, y los wikis han hecho que las comunicaciones en línea sean más fácil. Además, en la actualidad, es por lo general preferible utilizar herramientas en línea como wikis sobre comunicaciones personales ya que estas herramientas no sólo ayudan a comunicar la información sino que también la estructuran y la almacenan. Estas herramientas también son efectivas cuando se distribuye información a muchos destinatarios.

Estas conexiones rápidas de Internet disponibles globalmente proporcionan otras herramientas, que se agregan a esta tendencia. Las herramientas de modelado ayudan a que la documentación se explique por sí misma en los grupos distribuidos. Los detectores de fallas, servidores de control de fuentes, portales Web y las herramientas de colaboración en línea colaboran todas a coordinar los proyectos distribuidos. Los servicios de terminal y las máquinas virtuales facilitan la administración y la evaluación remota.

Internet también llevo a los países con mercados emergentes a buscar altas tecnologías. Debido a que Internet pasa por alto los límites políticos, miles de jóvenes de los países en desarrollo como Rusia y China lo utilizan para aprender tecnologías de vanguardia y mejorar su dominio del inglés. Esta nueva ola de ingenieros de software que se capacitan en Internet llega justo a tiempo para reforzar la tendencia de tercerización.

El aumento de la tercerización internacional en los últimos años ha cobrado la suficiente importancia como para convertirse en objeto de debates políticos. Por lo tanto esta discusión asume que el desarrollo en el exterior es una realidad existente, y nos centraremos en maximizar los beneficios de estos proyectos de tercerización. Pasaremos por alto la política, pero puede consultar en la lista de recursos el informe de McKinsey Global Institute que cuantifica los beneficios de la subcontratación en el extranjero para la economía de los Estados Unidos de América y desacredita varios mitos sobre esto.

Tendencias de desarrollo de software ágil

Volvamos a la tierra. Aquí los corazones y las mentes de varios gerentes e ingenieros están conquistados por otra tendencia moderna, el desarrollo de software ágil. Los modelos pesados lentos no demuestran su valor en el entorno de negocios dinámicos de la actualidad. Los presupuestos escasos demandan más resultados, si bien la burocracia nunca fue la mejor elección en lo que respecta al rendimiento de la inversión (ROI). La capacidad de los métodos ágiles reside en la colaboración, la flexibilidad y la dedicación para el valor del negocio del software tal como lo expresan los principios básicos de Manifiesto Ágil: *personas e interacciones* por sobre procesos y herramientas, *software funcionando* por sobre documentación completa, *colaboración con el cliente* por sobre negociación de contratos y *respuesta frente a cambios* por sobre seguimiento de un plan (Ver Recursos).

Los métodos ágiles se adaptan mucho a la nueva ola de empresas emergentes basadas en Internet (con frecuencia denominadas Web 2.0). El desarrollo de software ágil permite que algunas de estas empresas emergentes logren más por menos y producen proyectos significativos con equipos pequeños y presupuestos minúsculos. Los principios del software funcionando y de las repeticiones cortas se reflejan en una práctica llamada *beta constante* después de que varios productos Google con la palabra "beta" se incorporaron en su logo.

"LOS MÉTODOS ÁGILES NO SON ESTÁNDAR. FUNCIONAN BIEN PARA PEQUEÑOS EQUIPOS CO-UBICADOS QUE ENFRENTAN CON CONDICIONES QUE CAMBIAN RÁPIDAMENTE."

Sin embargo, los métodos ágiles no son estándar. Funcionan bien para pequeños equipos co-ubicados que enfrentan con condiciones que cambian rápidamente. Si bien estos son casos en los que la aplicabilidad del desarrollo de software ágil está sujeta a cuestionamiento –al igual que en el desarrollo distribuido con tercerización en el exterior– mis exitosos cinco años de experiencia aplicando los principios del desarrollo ágil en equipos distribuidos demuestran que es posible y que otorgan grandes beneficios cuando se utilizan adecuadamente.

Existen otros contextos en los que el uso de procesos de desarrollo de software ágil permanece cuestionable. Algunos ejemplos son los grandes equipos de desarrollo (más de 20

Figura 1: Integración del bus del servicio y punto a punto

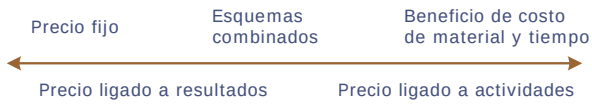


Figura 1: Tres capas de integración

personas trabajando en un proyecto independiente), sistemas en los que la predicción es primordial (aplicaciones de vida crítica) y los entornos burocráticos. No analizaremos estos contextos aquí, y además suponemos que una empresa posee cultura corporativa que favorece el desarrollo ágil y pretende aplicar las ideas que presentamos aquí para los equipos de software de menos de 20 personas (es decir, 20 personas que trabajen sobre un proyecto o equipo en particular, no en el equipo de desarrollo completo). En cambio analizaremos la aplicación de los métodos ágiles para el desarrollo distribuido en general y para la tercerización en el exterior en particular.

Combinar las tendencias

Los acuerdos de desarrollo de software en el exterior representan una gama completa de diferentes compromisos, desde contratar un desarrollador en el exterior a retacoder.com por un lado, hasta acuerdos de miles de millones de dólares con empresas norteamericanas que poseen subsidiarios en el exterior por el otro. Algunos de estos acuerdos están organizados de manera tal que impiden que las empresas utilicen un proceso de desarrollo de software ágil, aún si una de las partes lo desea.

Para implementar un proceso ágil, el modelo de tercerización seleccionado debe estimular las comunicaciones y la colaboración, asumir la flexibilidad y justificar una frecuente publicación. Si bien se pueden aplicar docenas de criterios a los acuerdos de tercerización, no muchos de ellos son tan importantes para nuestros análisis posteriores como el modelo de precios. En la Figura 1 podrá ver la asignación de los esquemas de precios más comunes.

Los resultados predecibles implican procesos predecibles. En la Figura 2 puede ver grupos de los procesos de desarrollo alineados sobre una escala predecible-adaptable. Si utilizamos un criterio predecible-adaptable, los esquemas predecibles, que se encuentran sobre el final izquierdo de la escala (Ver Figura 1), requieren más predicción desde los procesos de desarrollo de software; mientras que los procesos de desarrollo ágil están sobre el lado opuesto de la escala de procesos de desarrollo de software.

El interés de predicción está especialmente relacionado con los compromisos de *demanda fija-precio fijo*. Cuando se aplica este tipo de contrato a un acuerdo de tercerización, el cliente y el proveedor están naturalmente ligados a los procesos de desarrollo de software predecible y sus consecuencias. Por lo tanto, este contrato no es adecuado para el desarrollo de software ágil. Al diseñar un acuerdo de tercerización, recuerde que la sensibilidad a los cambios estimula el uso de modelos de precio como tiempo y material, lo que otorga flexibilidad tanto al cliente como al proveedor y es una mejor elección para el desarrollo ágil.

Al estructurar un acuerdo, considere la manera en la que el esquema seleccionado afectará las comunicaciones y la colaboración. Un proceso de desarrollo ágil requiere un entorno abierto, integración estrecha del equipo, objetivos comunes compartidos, comprensión del valor del negocio y comunicación frecuente. Cuantos más obstáculos tengamos entre ingenieros, clientes, usuarios, gerentes y otras partes interesadas, más difícil es proporcionar una base para el desarrollo de software ágil, lo que implica reducir intermediarios para permitir la máxima transparencia e integración entre los equipos.

Los grandes participantes resuelven esto estableciendo sus propias marcas en otros países, lo que tiene sentido económico si una empresa desea tener más de 100 ingenieros en su centro de desarrollo en el exterior. El número exacto depende mucho del otro país y de factores como la facilidad con la que la empresa selecciona personas talentosas allí.

Al considerar esta alternativa, no repita los errores de algunas pequeñas y medianas empresas, quienes subestimaron los gastos ocultos como tiempo record de gestión, presupuestos para viajes, honorarios de abogados locales y otros. También, el veloz crecimiento de las economías en los países en desarrollo provoca

"ELEGIR EL MODELO CORRECTO TAMBIÉN ES MUY IMPORTANTE, PERO NO GARANTIZA EL ÉXITO."

escasez de talentos, lo que no facilita la vida de pequeños nuevos valores. Si bien los participantes locales están mejor interconectados en su comunidad local, las marcas de las grandes empresas pueden compensar esto con marcas bien conocidas, mayores salarios y mejores paquetes sociales.

Tales lujos están por lo general fuera del alcance de las pequeñas empresas, que tal vez sólo desean tener 15 desarrolladores en el exterior. Los participantes pequeños y medianos exitosos reducen los costos de administrar el establecimiento de oficinas remotas utilizando equipos de desarrollo dedicados, centros de desarrollo en el exterior (ODCs), modelos construir-operar-transferir (BOT), oficinas virtuales, equipos virtuales y demás. Sin importar el nombre y los detalles, una cosa es común en estos modelos: el proveedor en el exterior pone más atención en proporcionar infraestructura HR, IT, legal y física al cliente que en participar del proceso de desarrollo. La responsabilidad para el envío del proyecto en estos acuerdos está compartida entre el cliente y el proveedor.

Para obtener los máximos beneficios de estos acuerdos, los ingenieros deben ser asignados a un cliente y la tasa de retención del equipo debe ser buena en ambas empresas. El proveedor debe proporcionar al cliente comunicaciones transparentes en todos los niveles, incluyendo las comunicaciones desarrollado-a-desarrollador. Las personas deben hablar un idioma sin un traductor.

En los compromisos exitosos a veces vemos que a pesar de que las personas en las oficinas remotas son remuneradas a través del proveedor, ellos se asocian más con el cliente y comparten la cultura corporativa y los valores de ambas empresas.

Utilizar herramientas y prácticas correctas

Existen prácticas bien conocidas utilizadas por los equipos de desarrollo de software exitosos: normas de codificación común; un servidor de control de fuentes; secuencias de comandos un-clic, construir e implementar; integración continua; evaluación de la unidad; rastreo de fallas; patrones de diseño y bloques de

aplicación. Estas prácticas deben aplicarse más estrictamente a los equipos distribuidos que a los equipos locales.

Por ejemplo, considere la integración continua. Tal vez sea extremadamente frustrante llegar a su trabajo y recibir una compilación dañada del servidor de control de fuentes, cuando la persona responsable de esto está a varias miles de millas y tal vez esté durmiendo en ese momento. Este problema no sería tan grande si el responsable es quien se encuentra en la oficina de al lado, pero puede volverse un problema importante en un contexto distribuido dañando la productividad y las comunicaciones. Puede minimizar estos riesgos siguiendo las prácticas de integración continua en todo el equipo e instalando el servidor correspondiente (como Microsoft Team Foundation Server, CruiseControl.NET y CruiseControl).

Los equipos que trabajan sobre la plataforma Microsoft.NET están en una buena posición con las funciones provistas por el *Visual Studio Team System* de Microsoft listo para usar. Puede obtener el *Microsoft Solutions Framework* prescriptivo para el Desarrollo Ágil y herramientas de soporte. Este producto es extremadamente útil para los equipos que necesitan más orientación con el desarrollo ágil en entornos distribuidos. Para los equipos con experiencia, esto es una solución integrada que proporciona un gran ROI [Rendimiento de la Inversión].

Otro producto de Microsoft que proporciona gran valor para los equipos distribuidos es *Windows SharePoint Services* (WSS). Los Wikis por naturaleza se adecuan y colaboran al desarrollo ágil en equipos distribuidos, y se estima que la próxima versión de WSS tendrá wiki entre sus mejoras. WSS también está estrechamente integrado con *Visual Studio Team System*, lo que hace que sea la mejor elección para el portal Web del equipo.

Desde el punto de vista de la infraestructura IT, recomiendo utilizar una red privada virtual (VPN) que le otorga a los equipos igual acceso a los recursos compartidos. El entorno VPN, al ser menos estricto que una red pública, permite utilizar funciones tales como las aplicaciones de *Windows Live Messenger* compartiendo llamadas de voz y video, asistencia remota y pizarra virtual.

Comunicación, Comunicación y Comunicación

Al trabajar de forma remota, los pequeños errores de entendimiento rápidamente se transforman en grandes problemas. En los equipos de desarrollo distribuido, los gerentes deben prestar atención a las prácticas de comunicación, que por lo general omiten sin consecuencias negativas en el desarrollo local. Esta atención incluye informes regulares (diarios/semanales) y reuniones de actualización del estado, lo que permite que los miembros del equipo se sincronicen, analicen los logros y comuniquen los problemas. Los gerentes también deben tratar de construir relaciones humanas personales dentro de los equipos a través de reuniones introductorias, visitas al lugar de trabajo, actividades de construcción del equipo y otros métodos.

En los acuerdos de tercerización en el extranjero, los gerentes de desarrollo deben ser conscientes de los obstáculos culturales, del idioma y de zona horaria y deben encontrar las formas de vencer estos obstáculos. La globalización lentamente pero de manera constante elimina las diferencias culturales en el entorno profesional, pero aún existen casos en los que las diferencias culturales prestan a confusión. Existen varios problemas específicos de país en este tema y están fuera del alcance de esta discusión. Los problemas de idioma son mucho más fáciles de detectar, aunque no significa que sean más fácil de superar. En los casos en los que las empresas enfrentan problemas de idioma, es

"EL PROCESO DE DESARROLLO ÁGIL REQUIERE UN ENTORNO ABIERTO, INTEGRACIÓN ESTRECHA DEL EQUIPO, OBJETIVOS COMUNES COMPARTIDOS, COMPRENSIÓN DEL VALOR DEL NEGOCIO Y COMUNICACIÓN FRECUENTE."

común y altamente recomendable contar con un entrenamiento de idiomas patrocinado por la compañía para los empleados. En la mayoría de los países de desarrollo en el exterior, los profesionales son motivados a aprender inglés, por lo tanto, son por lo general las personas en estas localidades quienes obtienen entrenamiento de idioma.

Las variaciones de zona horaria específicamente hacen que el proceso sea más difícil. Pero resulta que en los países con industrias de tercerización desarrolladas, los ingenieros por lo general están dispuestos a adaptar su horario laboral para trabajar con su contraparte en el exterior. Existen dos estrategias para manejar las diferencias de zona horaria. La primera es separar a los equipos por actividad; por ejemplo, tener gerentes de producto y de aseguramiento de calidad en la empresa y desarrolladores en el exterior. Esta organización permite implementar un ciclo en el que los desarrolladores implementan arreglos y nuevos requisitos mientras sus contrapartes están durmiendo y viceversa. Por supuesto debe haber intersecciones en los horario laborales (al comienzo/final del día laboral). El segundo enfoque es dividir los proyectos en bloques y tratar de asignar cada bloque a una ubicación, delegando la mayor cantidad de funciones posibles a esta ubicación. El segundo enfoque obliga a una mejor comunicación y por lo tanto sirve mejor al desarrollo ágil, pero ambos funcionan y muchas veces no existe otra opción.

Elegir el modelo correcto también es muy importante, pero no garantiza el éxito. Es muy recomendable que al menos una parte tenga experiencia en el desarrollo ágil, preferentemente en un entorno distribuido. La falta de comunicación cara-a-cara, junto con las diferencias culturales, de idioma y de tiempo requieren atención e invertir un esfuerzo adicional para obtener los resultados deseados. Los beneficios de contar con un buen colaborador en el exterior –ahorro de costos, aumento de personal sobre la demanda, tareas relacionadas con la tercerización de infraestructura– (que se puede resumir como "obtener más por menos") supera ampliamente la inversión al construir relaciones productivas. Este balance positivo sería imposible sin las herramientas modernas que posibilita la gran infraestructura de comunicaciones, ahora disponible globalmente. •

Recursos

"Explorar los mitos de la tercerización", Martin N. Baily y Diana Farrell,
Publicación trimestral McKinsey (McKinsey & Company, 2004)
www.mckinseyquarterly.com (Nota: se requiere registrarse.)

Manifiesto para Desarrollo Ágil
http://agilemanifesto.org

Sobre el Autor

Andrew Filev (MCA, MVP) es vicepresidente responsable de las operaciones en el extranjero en Murano Software. Establece centros de desarrollo en el exterior y lidera y motiva a los equipos. Un comunicador excelente, Andrew llena el vacío entre las diferentes culturas y construye alianzas duraderas con los clientes.

Modelación Orientada al Servicio para Sistemas Conectados – Segunda Parte

Por Arvindra Sehmi y Beat Schwegler

Síntesis

Como arquitectos, podemos adoptar una nueva forma de pensamiento para *forzar* esencialmente la consideración explícita en nuestros procesos de diseño de artefactos que modelan el servicio, lo que nos ayuda a identificar de forma adecuada los artefactos y en el nivel correcto de abstracción para satisfacer y alinear las necesidades comerciales de nuestras empresas. En la primera parte ofrecimos un enfoque de tres partes para la modelación de sistemas conectados orientados al servicio de manera tal que promuevan una alineación estrecha entre la solución IT y las necesidades del negocio. Examinamos la perspectiva actual del pensamiento orientado al servicio y explicamos la forma en la que el pensamiento actual y la conceptuación escasa de la orientación al servicio ha dado como resultado varios fracasos y por lo general, niveles pobres de rendimiento de la inversión (ROI). También analizamos los beneficios de insertar un modelo de servicios entre el negocio convencional y los modelos de tecnología que son conocidos por la mayoría de los arquitectos y debatimos la metodología *Motion* de Microsoft y distribución de capacidades para identificar las capacidades del negocio que pueden asignarse a los servicios. En esta segunda y última parte mostraremos la manera de implementar estos servicios distribuidos.

En la primera parte, terminamos nuestra discusión con un proceso pragmático de Diseño y Análisis Orientado al Servicio (SOAD) que se utiliza para extraer todas las piezas necesarias requeridas para construir un modelo de servicio. Esta extracción incluye contratos de servicio, acuerdos a nivel de servicio (SLAs) derivados de la expectativa de nivel de servicio (SLE) definida por cada capacidad del negocio y los requisitos de organización del servicio. Con un modelo de servicio detallado, estrechamente alineado y derivado del modelo de negocios, usted se encuentra ahora en una buena posición para asignar el modelo de servicios a un modelo de tecnología que identifique la manera en la que cada servicio se implementará, alojará y utilizará.

Al utilizar el enfoque precedente y al crear el modelo de servicios, se puede distribuir al departamento IT esquemas de datos, contratos de servicios y los requerimientos del SLA. Sin embargo, antes de construir el servicio, se debe considerar el soporte de la autonomía del servicio al nivel de la tecnología separando claramente las interfaces de la implementación de los mecanismos de transporte subyacente. Desarrollar estrategias de implementación del servicio que disocien el punto final del servicio de la implementación del servicio, lo ayudará a crear un plan de cambios. Seleccionar hosts adecuados y opciones de gestión del servicio para cumplir con los SLAs establecidos, también requiere una consideración especial. Sus elecciones en estas áreas son capturadas y definidas por el modelo de tecnología.

Crear un modelo de tecnología

El modelo de tecnología está compuesto por diversos artefactos: Analizaremos los detalles de cada uno.

La *interfaz del servicio* especifica la forma en la que se puede recibir un documento o un mensaje. La interfaz le permite especificar los transportes que se van a proporcionar, sin tener en cuenta la implementación. Más de una interfaz de servicio puede implementar un contrato de servicio, pero cada interfaz de servicio implementa un vínculo específico como por ejemplo SOAP sobre HTTP.

Si es necesario puede proporcionar interfaces de servicio múltiples utilizando diferentes transportes. Por ejemplo, puede vincular una única interfaz a un transporte de servicios de Web y también a un transporte de cola de mensajes de Windows. Cada transporte brinda diferentes características como soporte transaccional e interoperabilidad y diferentes restricciones. Por ejemplo, la cola de mensajes no admite directamente el patrón solicitud/respuesta. Se debe brindar al menos una interfaz para la interoperabilidad, asegurando que la interfaz sea compatible con la versión 1.x del Perfil Básico de Interoperabilidad de Servicios Web (WS-I BP).

La *implementación del servicio* es la implementación de una característica del negocio sin importar el host subyacente. También, no debe depender de sus interfaces. La implementación del servicio puede llamar otros servicios utilizando proxies que dependen de un vínculo y para lograr proxies independientes de un vínculo deben crearse utilizando fábricas.

El *host del servicio* proporciona un punto final para las interfaces del servicio. La elección del host debe basarse en los requisitos SLAs especificados. Por ejemplo, si el negocio espera 24/7 operaciones, esta característica posee un gran impacto sobre su elección de host y en estas situaciones por lo general se requieren tecnologías de cola como la Cola de Mensajes de Microsoft o el Agente de Servicios del Servidor SQL. En el modelo de tecnología, los hosts representan las opciones de costos. En el modelo del negocio, el SLE representa el valor. Si se puede asignar la elección del host a un SLA capturado por el modelo del servicio, y en última instancia a un SLE orientado al negocio, se puede justificar y medir el costo de un determinado host.

Un beneficio adicional al moverse hacia servicios y abandonar aplicaciones basadas en silos es que si cuenta con una capacidad específica de negocio que requiere 24/7 operaciones, puede migrar este servicio a un host que proporcione la redundancia y el rendimiento necesarios. Tal vez pueda utilizar hosts menos costosos para otras características. Con el enfoque de silos, está obligado a seleccionar un host para toda la aplicación.

Para la *gestión del servicio* se deben tomar medidas apropiadas con respecto al artefacto si no se puede satisfacer un SLA, y para soportar esta acción debe poder monitorear y administrar el servicio. Deben instrumentarse las interfaces del servicio, las implementaciones y los hosts, por ejemplo, utilizando *Windows Management Instrumentation* (WMI). Entonces se puede utilizar *Microsoft Operations Manager* (MOM) para monitorear y administrar el servicio mientras se utiliza *Microsoft Systems Management Server* (SMS) para soportar la administración de la configuración y los cambios en el ecosistema "edge" del servicio. Su elección del motor de organización también debe proporcionar soporte de monitoreo como las funciones del *Business Activity Monitoring* (BAM) que brinda el Servidor BizTalk. En el futuro, *WS-Management* jugará un papel principal en la administración de recursos independientemente de la plataforma de gestión y el hosting.

Para el *motor de organización,* el modelo del servicio debe definir los requisitos de organización del artefacto de manera independiente a la plataforma, pero en última instancia será necesario utilizar un motor de organización específica. El modelo de tecnología identifica la plataforma de destino como el Servidor BizTalk de Microsoft.

Al crear el modelo de tecnología se pueden capturar de forma explícita los artefactos anteriormente citados, pero ¿Cómo debe abordar el desarrollo del servicio? ¿Cómo asigna estos artefactos a la implementación del servicio? y ¿Cómo implementa el contrato que especifica el modelo del servicio? En breve, analizaremos un enfoque que ayuda a construir servicios de manera tal que satisfagan los preceptos del servicio y los principios de modelado analizados anteriormente.

Crear servicios en la actualidad

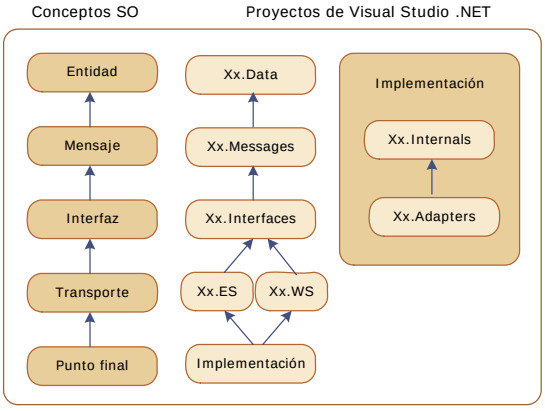
Dado un conjunto de artefactos de servicio conceptual, ¿Cómo puede definirlos en código? y ¿Cómo debe organizar el código dentro de Visual Studio 2005?. La clave que se debe tener presente es la necesidad de separación entre datos, mensajes, interfaces, intervalos de implementación y vínculos de transporte. La Figura 1 muestra un enfoque para asignar artefactos de servicio a los proyectos y soluciones de Visual Studio 2005.

"CADA TRANSPORTE BRINDA DIFERENTES CARACTERÍSTICAS COMO SOPORTE TRANSACCIONAL E INTEROPERABILIDAD Y DIFERENTES RESTRICCIONES."

El marcador de posición "xx" que se muestra en al Figura 1 representa un espacio de nombres basado en el nombre del servicio –por ejemplo, OrderService. Utilizar un único archivo de solución Visual Studio 2005 para un servicio determinado y como contenedor de los múltiples proyectos que se muestran en la Figura 1 es una práctica recomendada. Analizaremos estos proyectos.

- El proyecto *xx.Data* se utiliza para contener las definiciones del esquema de datos en línea que se obtienen de la entidad del modelo de servicio y también contiene la representación del lenguaje común en tiempo de ejecución (CRL) de este esquema.
- El proyecto *xx.Messages* se utiliza para contener las definiciones de los mensajes requeridos para comunicarse con el servicio, que incluye los mensajes entrantes y salientes. Un mensaje se representa como un elemento del esquema que contiene elementos del contrato de datos.
- El proyecto *xx.Interfaces* contiene las definiciones de la interfaz.
- Los proyectos *xx.ES* y *xx.WS* contienen los puntos finales de transporte para las interfaces. En el ejemplo que se muestra en la Figura 1, xx.ES brinda un transporte de servicios empresariales y xx.WS brinda un transporte de servicios de Web.
- Los proyectos *xx.Adapters* y *xx.Internals* contienen la implementación del servicio. Un adaptador que está contenido en el proyecto xx.Adapters brinda un nivel de indirección entre la implementación interna y la interfaz. El adaptador analiza el mensaje entrante y pasa las entidades de datos relevantes al código interno de implementación. El adaptador también empaqueta los resultados de los datos de la implementación dentro del mensaje de respuesta. Cabe observar que con este enfoque, la implementación no conoce nada acerca del mensaje que pasa a través de la interfaz, lo que le permite cambiar su infraestructura de mensajería sin impactar la implementación del servicio interno.

Figura 1 Mapeo de artefactos del servicio hacia proyectos de Visual Studio 2005



El punto final que se muestra en la Figura 1 es puramente un artefacto de implementación y depende de su elección del vínculo de transporte. Por ejemplo, con un transporte de servicios Web, podría implementar su servicio para un servidor Web de Servicios de Información en Internet (IIS) ejecutando ASP.NET.

Adaptadores. El adaptador es el artefacto clave que le permite desacoplar su implementación de la mensajería. También es un buen lugar para realizar transformaciones del mensaje en caso de ser necesario. Por ejemplo, dentro de un adaptador puede transformar una representación externa de un ID de cliente (por ejemplo, 10 caracteres) dentro de su representación interna (por ejemplo, 12 dígitos). Al crear un nuevo adaptador por versión de su servicio, el adaptador también proporciona una forma de versionar servicios sin afectar la implementación o sus interfaces del servicio publicadas anteriormente.

Versionado. Si se considera el versionado, es necesario distinguir entre versionar el contrato concreto o abstracto y versionar la implementación interna. Un objetivo clave de la arquitectura es brindar un versionado independiente entre los dos. El contrato en sí mismo está compuesto de datos, mensajes, interfaces y puntos finales. Se puede versionar cada uno de estos artefactos de forma independiente si se diseña el servicio de forma tal que cada artefacto de composición se refiera a una única versión del artefacto compuesto. Por ejemplo, el contrato del mensaje que está compuesto por el contrato de datos debe referirse precisamente a una versión del contrato de datos. Esta referencia debe ser verdadera para la representación CLR y XSD/WSDL del contrato.

Los contratos de datos y mensajes deben versionarse de acuerdo con el versionado XML y la política de extensibilidad (Ver Recursos). Se pueden versionar interfaces del servicio proporcionando una nueva interfaz que se herede de la anterior (CRL), una que esté asignada dentro de un nuevo tipo de puerto WSDL.

Seis pasos para la construcción de un servicio

Para llegar a estos proyectos de Visual Studio 2005 y construir hoy en día un servicio utilizando este enfoque, adopte este proceso de seis pasos:

1. Diseñar el contrato de mensaje y datos.
2. Diseñar el contrato del servicio.
3. Crear los adaptadores.
4. Implementar las partes internas del servicio.
5. Conectar las partes internas a los adaptadores.
6. Crear las interfaces de transporte.

Analicemos los detalles de cada paso. Para el primer paso –Diseñar el contrato de mensaje y datos– tome el esquema de datos canónico que define la representación en línea de los datos (un resultado de su diseño y análisis orientado al servicio) y defina las clases de datos y mensajes. Existen dos enfoques para crear sus esquemas de datos y clases de datos. Si se utiliza el enfoque primario esquemático, se puede crear un esquema XSD y luego utilizar una herramienta como Xsd.exe o XsdObjectGen.exe para generar las clases de datos automáticamente. Por otro lado, si prefiere un enfoque primario de codificación, puede definir las clases de datos con C# o Visual Basic y luego utilizar Xsd.exe para crear un esquema XSD equivalente. A continuación se muestra un ejemplo del contrato de datos:

```
namespace DataContracts
{
    [Serializable]
    [XmlType("Order", Namespace=
       "urn.contoso.data/order")]
    public partial class Order
    {
       [XMlElement("Customer")]
       public Customer customerField;
       [XMlElement("Items")]
       public OrderItemsList
         ordersItemsField;
       ...
    }
}
```

Una vez que ha definido las clases de datos, puede definir los mensajes de entrada y de salida, que contienen las clases de datos como su carga útil. Por ejemplo, este fragmento de código muestra un mensaje de entrada llamado OrderMessage para un servicio de pedidos hipotético que contiene una Orden como su carga útil:

```
namespace MessageContracts
{
   using System.Xml;
   using System.Xml.Serialization;
   using DataContracts;

   [Serializable]
   [XmlType(Namespace=
     "urn.contoso.msgs/orderservice"
     )]
   [XmlRoot(Namespace=
     "urn.contoso.msgs/
     orderservice"]
   public class OrderMessage
   {
     [XmlElement("Order")]
     public Order order;
   }
}
```

En el segundo paso –Diseñar el contrato del servicio– defina el contrato de servicio abstracto ya sea utilizando un enfoque primario WSDL o definiendo sus interfaces utilizando C# o Visual Basic. Las interfaces definen qué mensajes recibe su servicio y qué mensajes devuelve (si existiera alguno). Para generar la interfaz desde el WSDL es necesario utilizar el switch Wsdl.exe/si:

```
Wsdl.exe xx.wsdl /si
```

Es importante observar que este switch sólo funciona con la versión 2.0 de Microsoft.NET Framework. El siguiente ejemplo define una interfaz para el servicio de pedidos que define un único método PlaceOrder() que acepta un mensaje OrderMessage y devuelve un mensaje OrderTrakingMessage:

```
namespace Interfaces
{

   using MessageContracts;

   public interface IOrderService
   {
      OrderTrackingMessage
      PlaceOrder(OrderMessage
      placeOrderMsg);
   }
}
```

Cabe observar que en este caso se utiliza un patrón de mensaje solicitud/respuesta, y por lo tanto es necesario asegurar que el tiempo de procesamiento del mensaje (tiempo entre la solicitud y la respuesta) debe ser de sub-segundos. Si no se puede garantizar este tiempo de procesamiento, utilice otro patrón de intercambio de mensajes como un intercambio de mensajes doble que relaciona dos mensajes de ida en un patrón lógico de solicitud/respuesta.

Para generar el WSDL desde la interfaz precedente, puede crear una clase de servicio de Web ASMX que implemente la interfaz y luego llame al servicio de Web que está pasando ?WSDL –por ejemplo, http://localhost/Order-Service/OrderService.asmx?wsdl. Este servicio de Web hace que se genere el WSDL y que se devuelva desde el servicio de Web.

En el paso 3 –Crear los adaptadores– cree la clase de adaptador que proporciona la indirección entre la interfaz y los componentes internos. Esta clase garantiza que los componentes internos no conocen nada acerca del contrato del mensaje. El adaptador desempaqueta el mensaje entrante y pasa los datos de carga útil a

```
namespace Endpoints.WS
{
   using System;
   using System.Diagnostics;
   using System.Web.Services;
   using System.ComponentModel;
   using System.Web.Services.Protocols;
   using System.Web.Services.Description;
   using System.Xml.Serialization;
   using MessageContracts;
   using ServiceInterfaces;
   using Adapters;

   [WebService(Namespace=
     "urn.contoso.interfaces/orderservice")]
```

```
public class OrderService : System.Web.Services.
  WebService, IOrderService
{
  [WebMethod]
  [SoapDocumentMethod(ParameterStyle=
    SoapParameterStyle.Bare)]
  public OrderTrackingMessage PlaceOrder(
    OrderMessage placeOrderMsg)
  {
    IOrderService adapter = new OSA();
    return adapter.PlaceOrder(PlaceOrderMsg);
  }
 }
}
```

Lista 1 The IOrderService interface

la implementación interna realizando cualquier transformación del formato de los datos que sea necesaria desde el mensaje hacia el formato interno. De igual modo, en su regreso, el adaptador realiza cualquier transformación de formato que sea necesaria sobre los datos devueltos desde la implementación del servicio y los desempaqueta dentro de un mensaje de servicio de salida si existiera alguno. A continuación hay un adaptador de muestra:

```
namespace Adapters
{

   using MessageContracts;
   using ServiceInterfaces;

   public class OSA : IOrderService
   {

      public virtual
        OrderTrackingMessage
        PlaceOrder(
        OrderMessage placeOrderMsg)
      {
        // Call internals here
      ...
      }
   }
}
```

Observe que el adaptador está siempre en proceso con el solicitante y la identidad del proceso depende de su elección del host. Si la implementación interna se hostea dentro de IIS, entonces la interfaz ASMX (“edge”) inicia las partes internas del sistema. Si las partes internas se hostean dentro de los servicios empresariales pero las llamadas llegan a través de un servicio Web ASMX, la interfaz ASMX delega la llamada a la interfaz de servicios empresariales, la que luego inicia el adaptador y le pasa el mensaje. Se pueden encadenar las llamadas ya que todos los transportes implementan la misma interfaz.

En el paso 4 –Implementar las partes internas del servicio– cree la implementación interna del servicio. Existe sólo una implementación sin importar su elección de transporte o transportes. El siguiente fragmento de código muestra el código de estructura necesario para proporcionar una implementación del servicio de procesamiento de la orden:

```
namespace BusinessLogic
{

public class OSI
{

     public static string
       AcceptOrder(
       DataContracts.Order order)
     {
       // Process the order
       ...
       return "XYZ";
     }
   }
}
```

Observe cómo la implementación interna no posee conocimiento del mensaje. Sólo tiene conocimiento acerca del contrato de datos. En este caso se pasa un único objeto de Orden. Si desea ser completamente independiente del formato en línea, las partes internas no deberían aún tener acceso al contrato de datos. En este contexto, el adaptador asignaría el contrato de datos externo a los tipos de datos internos.

En el quinto paso –Conectar las partes internas a los adaptadores– llame la implementación interna desde el código del adaptador. Observe que el mensaje no se pasa a la implementación, lo que desacopla la implementación interna desde el contrato del mensaje y le permite cambiar uno sin impactar sobre el otro. El siguiente fragmento de código muestra nuevamente el código del adaptador, esta vez con una llamada a la implementación del servicio interna:

```
namespace Adapters
{
   using MessageContracts;
   using ServiceInterfaces;

   public class OSA : IOrderService
   {

     public virtual
       OrderTrackingMessage
       PlaceOrder(
       OrderMessage placeOrderMsg)
     {
       OrderTrackingMessage otm =
         new OrderTrackingMessage();
```

```
        otm.TrackingId =
          BusinessLogic.OSI.
          AcceptOrder(
          placeOrderMsg.Order);
        return otm;
      }
  }
}
```

En el paso 6 –Crear las interfaces de transporte– vincule las interfaces del servicio abstracto definidas en el paso 2 a un transporte específico. La lista 1 muestra la interfaz IOrderService que se define en el paso 2 vinculada a un transporte de servicio de Web.

Utilizar orientación automatizada del proceso

Si se sigue el proceso de seis pasos que acabamos de describir, se pueden construir servicios que sean compatibles con todos los principios y preceptos descriptos aquí. Sin embargo, debido a que todos los ítems descriptos en el proceso de seis pasos pueden describirse por medio de metadatos, es posible automatizar grandes secciones de la generación del servicio. Puede utilizar herramientas como el Juego de Herramientas para la Automatización Orientada (GAT) para ayudar a automatizar la orientación (Ver Recursos). La orientación automatizada del proceso es particularmente útil para la transformación entre artefactos CRL y XML semánticamente idénticos como las clases XSD y CRL, transformar entre el tipo de puerto WSDL y las interfaces CRL y generar diferentes tecnologías de punto final basadas en la descripción de la interfaz.

La antigua forma de pensar acerca de la orientación al servicio no está funcionando, y se necesita una nueva forma de pensar. Al adoptar este nuevo tipo de pensamiento, como arquitectos podemos *forzar* la consideración explícita de los artefactos de modelo del servicio en el proceso de diseño, lo que ayuda a identificar los artefactos de forma correcta y en el nivel adecuado de abstracción para satisfacerlos y alinearlos con las necesidades del negocio.

Desde una perspectiva de modelado, la brecha entre el negocio convencional y los modelos de tecnología es muy grande, lo que representa un factor de contribución clave para la falla de varias iniciativas con orientación al servicio. Hemos presentado un modelo de tres partes con la introducción de un modelo de servicio entre los modelos de tecnología y del negocio para fomentar un alineamiento más estrecho de los servicios con las necesidades comerciales. Si se cuenta con un modelo de servicio detallado y estrechamente alineado con el modelo del negocio y derivado de él, se encuentra en una buena posición para asignar el modelo del servicio a un modelo de tecnología que identifique la forma en la que se implementará, se alojará y se distribuirá cada servicio. La asignación de capacidades y la metodología Motion brindan una forma eficaz de identificar las capacidades del negocio y en última instancia del servicio. La descomposición del negocio dentro de las capacidades proporciona una distribución del más alto nivel para los contratos de servicios subyacentes y no al revés como suele ser en la actualidad.

Los sistemas conectados son instancias de todo el modelo de tres partes y respetan los cuatro principios de la orientación al servicio. Se pueden implementar de una forma *más completa* si se utilizan los cinco pilares de las tecnologías de plataforma de Microsoft. Recuerden que inicialmente preguntamos: ¿Cómo podemos evitar cometer con las arquitecturas orientadas al servicio (SOAs) los mismos errores en los que han resultado iniciativas prometedoras anteriores?, ¿Cómo podemos asegurarnos de que la arquitectura de implementación elegida se relaciona con el estado actual o deseado del negocio? ¿Cómo podemos asegurar una solución sostenible que reaccione a la naturaleza de cambio dinámico del negocio? –en otras palabras, ¿Cómo podemos posibilitar y mantener un negocio ágil? ¿Cómo podemos migrar a este nuevo modelo de forma elegante y a un paso que podamos controlar? y ¿Cómo podemos realizar este cambio con buen entendimiento en las situaciones en las que se puede agregar el mejor valor para el negocio desde el comienzo?.

La orientación al servicio con los servicios Web es sólo la implementación de un modelo particular. Es la calidad y el fundamento del modelo que determina las respuestas a estas preguntas.

Agradecimientos

Los autores desean agradecer a Ric Merrifield, director del Grupo Business Architecture Consulting de Microsoft, a David Ing, arquitecto de software independiente, a Christian Séller, arquitecto, Thinktecture, a Andreas Erlacher, arquitecto, Microsoft Austria y a Sam Chenaur, arquitecto de Microsoft Corporation por sus comentarios sobre las versiones preliminares de este artículo. También queremos expresar nuestro reconocimiento a Alex Mackman, tecnólogo principal de CM Group Ltd., un excelente investigador y escritor que nos ha ayudado enormemente. •

Recursos

"Designing Extensible, Versionable XML Formats," Dare Obasanjo (Microsoft Corporation, 2004)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexxml/html/xml07212004.asp

"Modelación y Mensajería para Sistemas Conectados", Arvindra Sehmi y Beat Schwegler
Un Web cast de una presentación en una Cumbre de Arquitectura Empresarial – Barcelona
(FTPOline.com, 2005) www.ftponline.com/channels/arch/reports/easbarc/2005/video/

MSDN – Juego de Herramientas para la Automatización Orientada (*Guidance Automation Toolkit*-GAT) http://msdn.microsoft.com/vstudio/teamsystem/Workshop/gat/default.aspx

Solicite un caso de estudio sobre la Metodología Motion de Microsoft enviando un correo electrónico a motion@microsoft.com.

"Modelación orientada al servicio para sistemas conectados– Primera parte," Arvindra Sehmi y Beat Schwegler, The Architecture Journal, Edición 7, Marzo 2006.
http://www.thearchitecturejournal.net

Sobre los autores

Arvindra Sehmi es jefe de arquitectura empresarial en el grupo EMEA Developer and Platform Evangelism de Microsoft. Centra su atención sobre la adopción de mejores prácticas de ingeniería de software empresarial a través de la comunidad de arquitectos y desarrolladores de EMEA y lidera el evangelismo de arquitectura en EMEA para la industria de servicios financieros. Arvindra es editor emérito de *The Architecture Journal*. Posee un doctorado en ingeniería biomédica y un master en negocios.

Beat Schwegler es arquitecto en el grupo Developer and Platform Evangelism de Microsoft EMEA. Brinda soporte y asesora a compañías empresariales sobre arquitectura del software y temas relacionados y es orador frecuente en conferencias y eventos internacionales. Posee más de 13 años de experiencia en arquitectura y desarrollo de software profesional y ha formado parte de una gran variedad de proyectos, que abarcan desde el control de la producción en tiempo real y productos de gran venta en dispositivos compactos hasta sistemas ERP y CRM de alta escala. En los últimos 4 años, el interés principal de Beat ha estado en las áreas de la orientación al servicio y los servicios Web.


ready_
ready_
ready_
Bienvenidos al negocio people ready.

En un negocio people-ready, las personas hacen las cosas realidad. Personas, preparadas con software.
Cuando usted le brinda a las personas herramientas que los conectan, los informan y los facultan, ellos están preparados. Preparados para sacar el máximo potencial de sus conocimientos, capacidades y ambiciones. Preparados para desarrollar nuevos productos, ayudar a los clientes y resolver problemas. Preparados para crear un negocio exitoso: un negocio people_ready.
Microsoft. Software para el negocio people_ready. Para obtener más información, visite microsoft.com/peopleready

Microsoft

THE ARCHITECTURE JOURNAL
Input for Better Outcomes