

Plan para la implementación exitosa de SOA

Introducción

La adopción de una arquitectura basada en servicios es actualmente una necesidad para la mayoría de las empresas. Esta necesidad no se justifica por una moda pasajera sino por la necesidad de lograr crear una infraestructura sólida que soporte la integración de los sistemas críticos de la empresa. Además, disponer de una infraestructura SOA también es un requerimiento para que una empresa pueda lograr una implementación exitosa de sistemas que permita alcanzar las metas que propone la teoría del *Business Process Management* (BPM), lo que permite responder más rápido a las necesidades del negocio.

Sin embargo, adoptar SOA no es una tarea sencilla, especialmente para aquellas organizaciones que no están acostumbradas a desarrollar sistemas distribuidos. Para aquellos que parten desde cero es usual sugerir que adopten la tecnología paulatinamente, en tres pasos, para reducir tanto la curva de aprendizaje como la probabilidad de incurrir en errores costosos. Estos son esos tres pasos:

- Planeación
- Enterprise Application Integration
- Business Process Management

Planeación

Para poder adoptar una arquitectura basada en servicios, aunque parezca obvio, lo primero que se debe hacer es determinar los sistemas que queremos exponer. Esto puede parecer sencillo, sin embargo, no lo es. Antes que nada tenemos que determinar qué es un servicio y luego debemos decidir cuales exponer y cuales desarrollar.

¿Qué es un servicio?

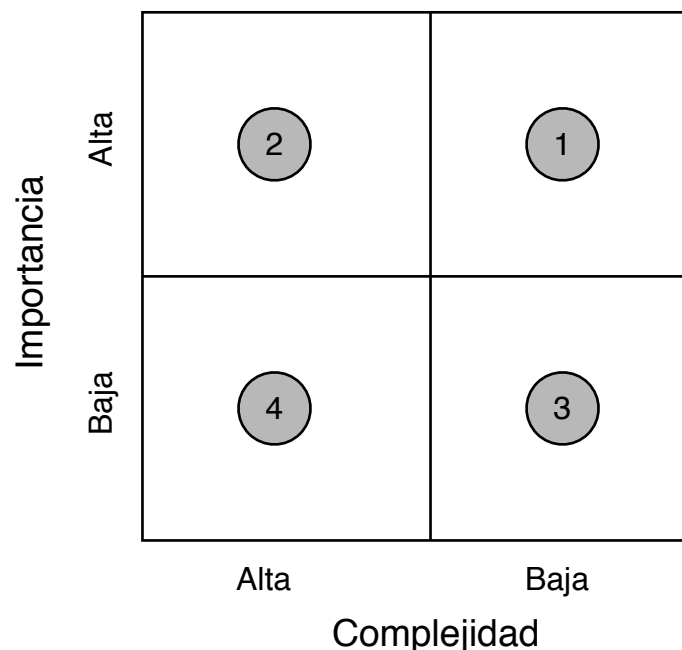
La definición de un servicio es un componente de software que puede ser invocado remotamente y que puede ser descrito de una manera estándar a través de un archivo WSDL (Web Services Definition Language). Sin embargo, si nos atenemos a la definición de manera estricta, podemos cometer graves errores. Por ejemplo, teóricamente podríamos crear un servicio que calculara una raíz cuadrada. ¿Pero, realmente es una buena idea? La respuesta es definitivamente, no. ¿Por qué? La razón es sencilla, el tiempo que tardaríamos en invocar este método sería mucho más largo que el tiempo que tardaría en ejecutarse la función. Por lo tanto, la función raíz cuadrada nunca dejará de ser una función del lenguaje de programación para convertirse en un servicio. Tomemos una función más compleja, una que dependiendo de un nivel de severidad decide imprimir un error a un log o no. ¿Es esto un servicio? En este caso, si la red es rápida y el disco no tanto, es posible que el tiempo de invocación de la función sea más bajo que el de ejecución. Sin embargo, eso no lo convierte en un servicio. ¿Por qué? Porque no es una función de negocios. El objetivo de implementar SOA es responder a las necesidades del negocio, ya sea a través de EAI (Enterprise Application Integration) o a través de BPM. Los usuarios de negocios no están interesados en funciones de TI como guardar mensajes de error en un log. Este tipo de funciones, propias de las áreas de sistemas se siguen resolviendo mejor a través de librerías compartidas utilizadas por las aplicaciones. Por lo tanto, vemos que la adopción de SOA no significa que de repente vamos a ver una proliferación incontrolada de servicios ya que en la mayoría de los casos, las empresas no tienen miles de funciones de negocio.

Generalmente, la gran mayoría de esas funciones de negocios ya están automatizadas por algún sistema y solo va a ser necesario exponerlas como servicios web. En otros casos, no y por lo tanto será necesario desarrollados desde cero.

Identificación de servicios

¿Qué servicios necesitamos? Uno de los principales errores que se pueden cometer al adoptar SOA es empezar a desarrollar nuevos servicios o incluso exponer servicios existentes sin tener la seguridad de que los vayamos a necesitar. Esto se debe a que ese proceso toma tiempo y requiere de recursos especializados. Por lo tanto, para ser exitosa, la adopción de SOA debe ser progresiva y basada en proyectos reales de EAI o de BPM. Los servicios se deben crear o exponer a medida que se requieren. Eso sí, los servicios se deben planear para que sean reutilizables y por lo tanto deben ser tan generales como sea posible. Esto significa que si lo que se requiere es un servicio para dar de alta un cliente en una región determinada, quizás sea mejor crear uno que permita crear un cliente para cualquier región.

En general, lo más recomendable es dejar que los usuarios de negocios definan los servicios, sin intervención de la gente de TI. Para lograrlo, lo mejor es dejar que utilicen una herramienta de modelado de procesos como WebSphere Business Modeler o Aris para representar tanto sus procesos existentes (as-is) como los que desearían implementar (to-be). Normalmente, de esos procesos se obtiene fácilmente los servicios que se deben implementar. Para decidir en qué orden hacerlo podemos ayudarnos de una pequeña matriz en la que un eje indicamos la complejidad del desarrollo y en la otra su importancia. De esta manera podemos empezar por los más importantes y sencillos y dejar los complejos y menos críticos para más adelante.



Exposición de servicios existentes

Cuando surge el requerimiento de un servicio existente pero que es parte de una aplicación legada, lo más conveniente es obviamente reducir al máximo la necesidad de desarrollar código. Para las aplicaciones empresariales más populares como SAP, PeopleSoft, Siebel o Ariba, el mundo J2EE tiene una solución, los conectores JCA (Java Connector Architecture, JSR 112, también conocido como J2C). Este API estándar, creado originalmente por IBM, permite que cualquier servidor de aplicaciones certificado J2EE compliant pueda utilizar conectores para comunicarse con su sistema legado. Por ejemplo, con un conector para SAP se vuelve trivial invocar BAPIs o RFCs. Por supuesto que siempre existen otras maneras de conectarse con un sistema legado, pero los

conectores simplifican la operación y suelen estar desarrollados por especialistas que logran un gran nivel de integración y de rendimiento.

Sin embargo, los conectores no resuelven todos los problemas, simplemente porque no existen conectores para todos los sistemas comerciales y mucho menos para aplicaciones desarrolladas en casa. En esos casos, hay que analizar las opciones de las que dispone el desarrollador para acceder a la aplicación.

En algunos casos, un acceso directo a la base de datos puede ser la solución, aunque no se recomienda porque generalmente esto complica la realización de cambios a la aplicación.

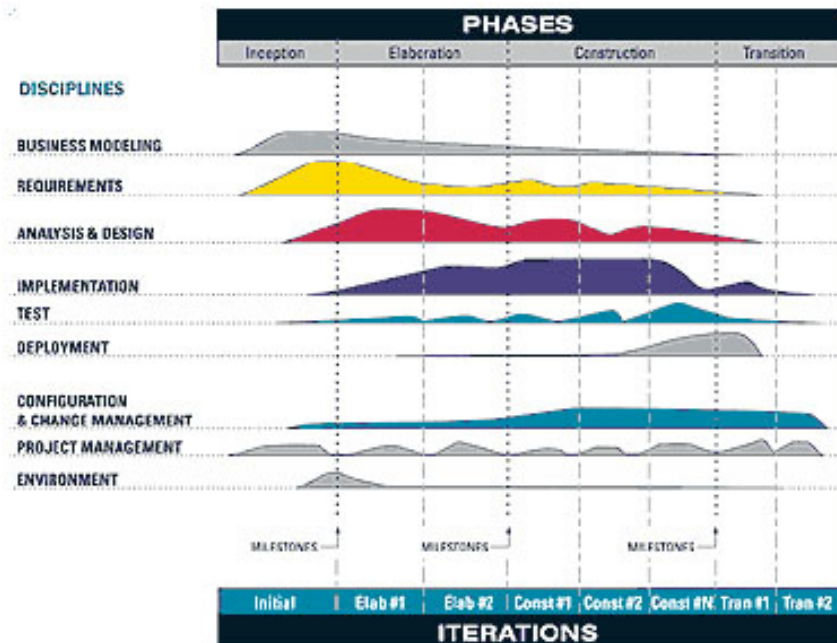
En otros casos, es posible cambiar la aplicación para exponerla como un servicio, si se dispone del código fuente y el lenguaje de programación es relativamente moderno (C, C++ o Java).

Otra alternativa más sencilla es utilizar una cola de mensajes como WebSphere MQ o Sonic MQ para integrar las aplicaciones. El uso de colas permite integrar aplicaciones fácilmente, especialmente las que se ejecutan desde la línea de comando, las que no pueden ejecutarse en paralelo o las que no tienen altos niveles de disponibilidad. Integrar aplicaciones sin utilizar colas de mensajes es posible, pero cuando se dispone de un gran número de servicios, ya no es una alternativa viable porque no es razonable pensar que en un esquema altamente distribuido todos los servicios estén siempre disponibles.

Creación de nuevos servicios

El desarrollo de nuevos servicios es un proceso muy similar al desarrollo de una aplicación. Los primeros pasos consisten en levantar los requerimientos, luego se crea una arquitectura que permita soportar la aplicación con los niveles de servicio y otros requerimientos no funcionales necesarios y finalmente se hace un análisis exhaustivo de la aplicación. Al tratarse de componentes que ofrecen funcionalidad limitada, la fase de desarrollo generalmente se puede completar en una sola iteración. Finalmente, la fase de pruebas, importantísima, se realiza normalmente de manera automatizada con herramientas ad-hoc.

Al igual que con cualquier tipo de desarrollo, el uso de una metodología de desarrollo probada como RUP (Rational Unified Process) es altamente recomendado. Si bien RUP es una metodología de desarrollo general, es fácil de adaptar a necesidades específicas, como SOA o cómputo móvil a través del Rational Method Composer. Durante la fase de análisis, herramientas de modelado como Rational Software Architect o MagicDraw UML que soportan UML 2.0 simplifican mucho el desarrollo de componentes.



Una de las características de los servicios web es que normalmente realizan solo una función. Por eso suelen ser muy fáciles de probar. Las pruebas, se deben hacer tanto para comprobar la funcionalidad del servicio como su escalabilidad. Existen herramientas que automatizan las pruebas unitarias de los servicios web.

Sin embargo, el desarrollo de nuevos componentes no es tan complicado como la administración de cambios a los mismos. En efecto, una vez que un servicio ha sido creado y es utilizado por múltiples procesos, realizar cambios al mismo puede parecer imposible. Esto no significa que una vez creado no sea posible realizar nunca cambios al mismo pero tampoco significa tener que crear otros similares pero con la funcionalidad específica que se requiere, porque eso tampoco es una opción. ¿Porqué? Porque entonces, se da la proliferación de servicios de la que hablábamos anteriormente la cual es nefasta, tanto para los programadores responsable de mantener el código como para los administradores del sistema. Por lo tanto, es necesario llevar un control de las dependencias existentes así como una comprensión precisa del impacto de los cambios. Esto es algo que se ha hecho desde hace años en proyectos tradicionales, pero que se ha vuelto aún más importante en un entorno de SOA.

Protocolo de transporte usado por los servicios

En los orígenes de los servicios web, lo importante era saber cómo invocarlos. Para eso se creó el protocolo SOAP (Simple Object Access Protocol). Simplificando, se trata básicamente de un documento XML que se utiliza para transportar los parámetros de entrada y de salida entre una aplicación cliente y un servicio web. Lo que no define es el protocolo de transporte único como HTTP o SMTP. En lugar de eso, enumera una serie de protocolos posibles y termina con puntos suspensivos indicando que cualquier protocolo es aceptable, siempre y cuando el mensaje cumpla con SOAP.

Inicialmente, la gente utilizó HTTP como el protocolo de transporte para los mensajes SOAP porque ofrece varias ventajas importantes, como bajo costo, posibilidad de utilizarlo a través de firewalls y posibilidad de utilizarlo con software de infraestructura existente (web server y application server). Sin embargo, HTTP tiene muchas limitaciones para un uso empresarial. La más importante es que HTTP no garantiza la entrega del mensaje ni la recepción de una respuesta. Esto en muchos casos no es muy importante. Por ejemplo, en EEUU, el gobierno facilita un servicio web público para que los desarrolladores puedan incluir las predicciones del tiempo en su página Web. Si este servicio se utiliza desde un portal como Yahoo! y el servicio por algún motivo no con-

testa, no pasa nada. El usuario solo necesita pulsar el botón "Reload" y el problema queda resuelto.

Sin embargo, dentro de una empresa, para aplicaciones de EAI no nos podemos dar el uso de usar el botón "Reload". Imaginemos por ejemplo que una aplicación de recursos humanos se utiliza para dar de alta a nuevos empleados. Cuando termina el proceso, la aplicación utiliza web services para conectarse con la aplicación de almacén para asignar una laptop a ese nuevo empleado. Si la llamada al servicio falla, con HTTP no sabemos si el servicio falló antes de asignar el equipo o después. Por lo tanto, no es posible simplemente volver a invocar el servicio, porque corremos el riesgo de asignar varios equipos a una misma persona. El mismo problema se presentaría con una transferencia bancaria, necesitamos la garantía de que esa transacción se va a realizar y de que solo se va a realizar una vez. En ese tipo de casos, HTTP no es una opción.

Afortunadamente existe una alternativa que cumple con los requerimientos que se han señalado. Esa alternativa consiste en utilizar colas de mensajes para servir como transporte seguro de los mensajes SOAP. Este producto garantiza la entrega de los mensajes, a pesar de fallas intermitentes en la red o en el sistema que provee el servicio. Además permite encriptar el canal de comunicación y controlar los usuarios que pueden escribir y leer a las colas, lo que permite garantizar una total seguridad en el intercambio de mensajes. El uso de servicios invocados a través de colas se conoce como SOAP/JMS debido a que JMS es el API de Java que permite interactuar con colas de mensajes.

Entonces, la pregunta obvia es ¿Qué protocolo debo usar? No existen reglas estrictas al respecto, pero si unas recetas que conviene adoptar:

Situaciones en las que se debe usar SOAP/JMS:

- Servicios asíncronos
- Servicios que representan procesos de negocios (tanto micro-flujos BPEL como macro-flujos)
- Servicios síncronos transaccionales, en los que se requiere la garantía de que se va a obtener una respuesta

Situaciones en las que se debe utilizar SOAP/HTTP:

- Cuando el lenguaje de programación utilizado por el servicio no puede interactuar con colas de mensaje
- En situaciones de B2B cuando se interactúa con un número alto o desconocido de socios de negocios
- Consultas síncronas sencillas que afectan a un solo sistema

Como se puede observar, siguen existiendo zonas grises en las que podría usar hipotéticamente ambas soluciones tecnológicas. Si bien es posible y de hecho muy fácil exponer un mismo servicio tanto por SOAP/JMS como por SOAP/HTTP, lo más recomendable es que en caso de duda se adopte SOAP/JMS ya que para el área de operaciones es mucho más sencillo monitorear una infraestructura de colas de mensajes que una basada en HTTP donde los mensajes se mezclan con el tráfico normal de acceso a Internet.

Governance

Nadie tiene problemas para administrar unos pocos servicios web, sin embargo, a medida de que aumenta el número de servicios puestos en producción, empiezan a surgir preguntas tales como ¿Qué servicios tenemos? ¿Cómo se invocan? ¿Quién garantiza la exactitud de los datos? ¿Quién los financia? ¿Cuánto nos cuesta la operación de cada uno de ellos? ¿Qué aplicaciones usan qué servicios?

Originalmente, todo lo que se había planeado para administrar los servicios web eran los directorios UDDI (Universal Description, Discovery and Integration, un estándar de la organización OASIS). La idea es que existieran unos pocos proveedores de directorios de servicios globales que sirvieran como una “sección amarilla” en que los desarrolladores pudieran buscar por industria o por empresa los servicios B2B que ofrecieran. Un directorio UDDI es como una gran base de datos que contiene archivos WSDL y por lo tanto permite contestar a las dos primeras preguntas, ¿Qué servicios tenemos? y ¿Cómo se invocan? Sin embargo, contrariamente a las expectativas iniciales, hoy en día se utilizan mucho más los servicios web dentro de la empresa que para negocios B2B (aunque estamos viendo que poco a poco este mercado también está empezando a crecer). Por eso empezaron a surgir los directorios públicos, los cuales, a partir de la versión 3.0 de UDDI (la actual) pueden integrarse mejor con directorios públicos mediante esquemas de replicación.



Advancing Web Services
Discovery Standard

Actualmente, la utilidad de los mismos se limita a los desarrolladores. Con ellos pueden saber siempre qué servicios existen y así evitar duplicar servicios de manera innecesaria. Por eso, es recomendable que una empresa que empieza a implementar SOA tenga desde el inicio un plan para instalar este tipo de productos. Sin embargo, UDDI no responde a todas las preguntas que un director de TI puede hacer legítimamente para controlar su infraestructura. Por eso, próximamente se verá una evolución en esta categoría de productos, ya que así lo demandan las empresas que han avanzado más en la adopción de SOA y de BPM.

Normalización sintáctica

Una de las tareas de planeación más importantes que deben realizarse desde el inicio, en paralelo con el descubrimiento de los servicios, consiste en normalizar los datos que intercambiarán los servicios. Esto es muy importante porque no queremos que conceptos como “cliente” o “factura” tengan cada uno múltiples implementaciones distintas, en el peor de los casos, una para cada sistema que los utiliza. Esto no significa forzosamente que sea necesario modificar todos los sistemas para que se adecuen a la nueva definición única de estos conceptos. Para evitar ese problema, es posible crear definiciones de datos basadas en XML-Schema (un estándar controlado por la w3c) que sean flexibles, por ejemplo utilizando campos opcionales.



En algunas industrias ya existen tipos de datos estándar que se utilizan para intercambiar información (por ejemplo HL7 dentro del mundo hospitalario). Por lo tanto, antes de crear un nuevo tipo de dato es importante cerciorarse que no existe ya un tipo de dato estándar que pueda ser reutilizado.

Modelar datos de manera consistente desde el inicio permite simplificar el trabajo de los usuarios de negocios que tienen que modelar los procesos, porque así no tienen que recordar qué tipo de cliente entiende determinado servicio web. Otra ventaja importante es que de esta manera se reduce de manera importante el número de transformaciones de datos que necesitan realizar los “brokers” y así se ataca desde el principio la principal causa de problemas de una arquitectura SOA, el rendimiento.

Conclusión

En estos primeros pasos necesarios para adoptar SOA, se deben realizar dos tipos de actividades. El primero es de planeación, dentro de los cuales caen las actividades de identificación de servicios, normalización sintáctica y planeación de cómo se llevarán a cabo la administración y control de los servicios. Estas tareas se desarrollan de manera conjunta entre los usuarios de ne-

gocios que conocen los procesos y analistas técnicos del área de TI que conocen funcionalmente las aplicaciones. El segundo es de desarrollo de servicios, utilizando una metodología comprobada y establecida como RUP e involucra principalmente a personal técnico (desarrolladores, analistas, arquitectos, etc.). A pesar de que se trata de actividades claramente distintas tienen un punto en común, el hecho de que se deban realizar al inicio no significa que dejen de realizarse al pasar a la siguiente fase. En efecto, siempre es necesario desarrollar o exponer nuevos servicios o asegurarse de que no se pierda la normalización sintáctica. Por lo tanto, las tareas que se describieron en esta primera fase seguirán teniendo que aplicarse incluso mucho después de haber implementado exitosamente un Enterprise Service Bus y desarrollado sistemas que permitan implementar BPM.

Enterprise Application Integration

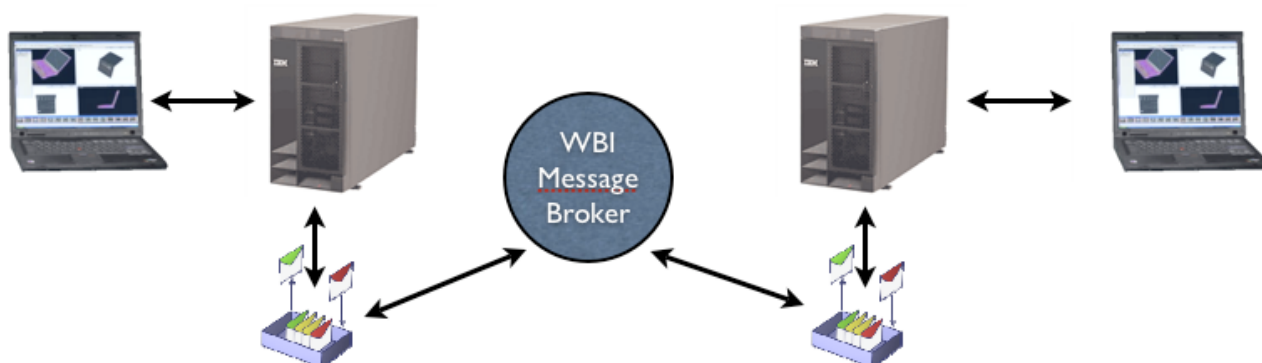
El objetivo de esta segunda fase es primordialmente lograr conectar entre sí los servicios de la empresa de manera segura. Esto es muy fácil de lograr si la primera fase se ha llevado a cabo correctamente, lo cual vamos a asumir. Con todos los servicios accesibles a través de JMS y/o HTTP, la interconexión se realiza a través de “brokers” que pueden trabajar con ambos protocolos de transporte y cuyas principales funciones son las de ruteo y transformación de mensajes.

Ruteo de mensajes

El ruteo de los mensajes en base a su contenido es una de las tareas que realiza un broker. Esto se utiliza para generar servicios de alcance general utilizando varios servicios que ofrecen un alcance más limitado. Por ejemplo, es normal utilizar un “broker” para convertir tres servicios de ventas de un tipo de seguro (vida, habitación y coche) en uno general de venta de cualquier tipo de pólizas de seguro. Normalmente un usuario de negocios solo piensa en servicios generales y no entiende porqué el área de TI diferencia entre unos y otros. El uso de un broker en ese tipo de situaciones permite reconciliar la realidad de TI con los conceptos que manejan las áreas de negocio y que quieren usar en sus modelos.

Transformación de mensajes

En la primera fase se resaltaba la importancia de realizar como parte del proceso de planeación la normalización sintáctica para evitar al máximo la transformación de mensajes. Sin embargo, esto no elimina la necesidad de transformar mensajes. Por ejemplo podemos mandar un mensaje que contiene un pedido y solo necesitar la lista de productos que contiene. Extraer la lista de productos de un mensaje XML es algo trivial utilizando XSL y XPath, estándares soportados por la mayoría de los productos de integración. Por lo tanto, es necesario con especialistas que los conozcan perfectamente.



Seguridad

Una de las preocupaciones principales que se deben tener en cuenta a la hora de implementar una arquitectura SOA es la seguridad, en particular la de los mensajes.

Los mensajes SOAP que fluyen sobre JMS son fáciles de proteger. En efecto, las colas de mensaje permiten en general encriptar el canal de comunicaciones y limitar el acceso a las colas exclusivamente a los usuarios autorizados. Si el desarrollo se hace en Java utilizando servidores de aplicaciones J2EE, los desarrolladores de la aplicación no necesitan saber el usuario/password para poder leer o escribir a una cola, solo los administradores del sistema, lo que nos da un mayor nivel de seguridad.

Si se usa SOAP sobre HTTP, en principio los riesgos son aún mayores ya que no hay manera de controlar fácilmente quién puede invocar el servicio. En cambio, al igual que con SOAP/JMS, es posible encriptar la comunicación utilizando HTTPS en lugar de HTTP.

Los estándares de web services también incluyen la normativa WS-Security que está compuesta por dos especificaciones, WS-Encryption y WS-Signature. La primera permite encriptar el mensaje, lo cual es útil especialmente en situaciones en las que el canal de comunicaciones no es seguro (generalmente en ambientes de comercio electrónico B2B). La segunda está pensada para garantizar que el contenido del mensaje no pueda ser alterado, aplicando una firma electrónica (en general a parte del documento).

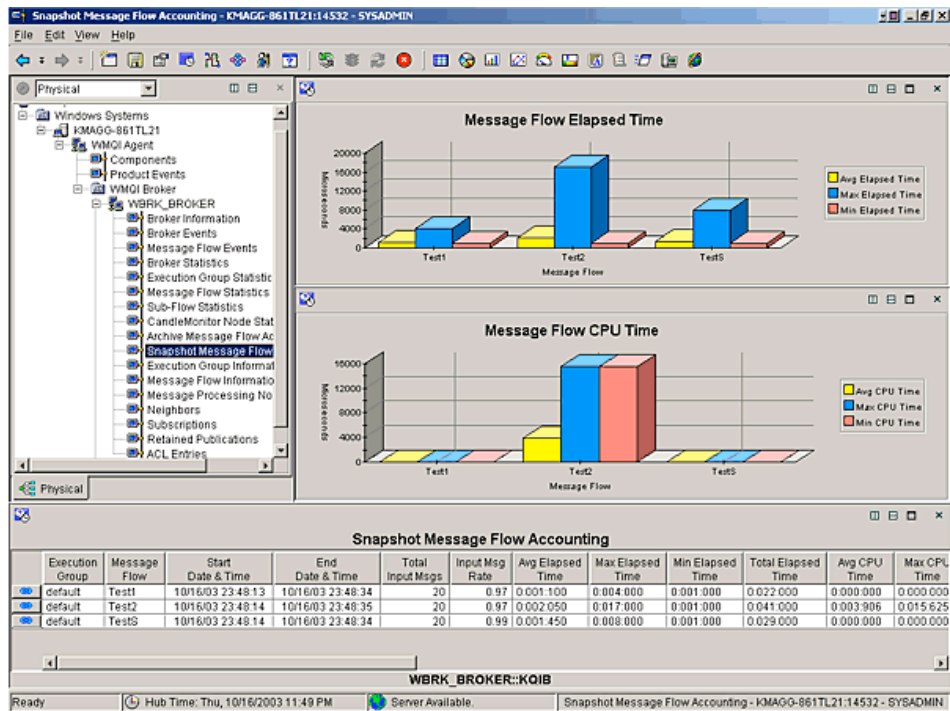
Lo importante es entender que existen los medios para crear una infraestructura segura, pero que casi siempre esto se hace a costa de rendimiento. Este costo puede llegar a ser muy alto cuando se trabaja con certificados que deben ser validados a cada llamada a un servicio web. Por lo tanto, es importante definir unas políticas de seguridad que tengan en cuenta tanto las necesidades de seguridad como de rendimiento e implementarlas de manera consistente. Es recomendable asignar esta responsabilidad a una persona que solo se encargue de esta función ya que se requieren de muchos conocimientos específicos que no es razonable exigir a programadores o administradores del sistema.

Monitoreo

Es evidente que una empresa que apuesta por SOA está adoptando una arquitectura sumamente distribuida. Si bien SOA aporta enormes beneficios para los desarrolladores y para los usuarios de sistemas, también es cierto que complica la vida del área de operaciones que se tiene que encargar de dar soporte a las aplicaciones y garantizar que operan de manera satisfactoria e ininterrumpida. Estas áreas están acostumbradas a trabajar con grandes aplicaciones monolíticas que en general solo tienen dos estados, funcionan o están caídas. Además, si están caídas, el problema generalmente es fácil de ubicar en la base de datos y las personas afectadas son exclusivamente aquellas que utilizan esa aplicación.

Cuando una empresa adopta SOA las cosas se vuelven un poco más complejas. Para empezar, si se cae un servicio, puede que no se vea afectada solo una aplicación, sino varias. Una falla de la base de datos puede requerir que se reinicialice y que por lo tanto haya que reiniciar también los servidores de aplicaciones que se conectan a esa fuente de datos y quizás ahí residan también otros servicios. Es importante que las herramientas de monitoreo que use el área de operaciones permitan conocer esas dependencias y que el área de operaciones entienda cómo todas las piezas están conectadas. No es algo complejo pero si es algo que no están acostumbrados a hacer y para lo cual necesitan entrenamiento y herramientas adecuadas. Por esa razón existen productos que permiten monitorear en una única consola todos los componentes que participan en una arquitectura SOA y entender el impacto que tiene sobre los procesos de negocios la falla de uno o varios componentes (tanto de software como de hardware).

Con SOA, nunca hay que olvidar que la arquitectura es tan sólida como lo es el eslabón más débil. Por eso y por el hecho de que normalmente las arquitecturas basadas en SOA están pensadas para trabajar en ambientes 24x7días, desde la planeación, el monitoreo de los servicios debe ser una prioridad.



Calidad de servicio

Al igual que es necesario monitorear los distintos componentes de la infraestructura SOA para poder responder rápidamente a inevitables fallas del sistema, también necesitamos herramientas que nos permitan garantizar una determinada calidad de servicio. Esa calidad se puede medir en términos de disponibilidad, pero también en términos de tiempos de respuesta.

La disponibilidad de los servicios es directamente proporcional a la calidad del código de la aplicación. Por lo tanto, una aplicación bien programada puede lograr altos niveles de disponibilidad. Sin embargo, esa aplicación no queda inmune ante problemas de hardware o picos inesperados en el uso de una aplicación. Para lograr evitar esos escollos, la única solución es el uso de clusters de servidores de aplicaciones. Un cluster permite que si falla uno de los nodos no se interrumpa la operación, ya que la carga de ese servidor se reparte entre los nodos restantes. Desde el punto de vista de operación lo ideal es tener un solo cluster con muchos nodos que albergue todos los servicios en lugar de varios clusters pequeños, cada uno de ellos con un servicio. Esto se debe a que en un cluster grande, el impacto de perder un nodo es casi nulo y si se producen picos de uso en una aplicación es más fácil distribuir esa demanda inesperada entre más nodos. Por otro lado, todas las aplicaciones y recursos se administran desde una sola consola.

Los tiempos de respuesta de una aplicación que corre sobre una infraestructura SOA son a veces fuente de misterio si no se dispone de las herramientas de monitoreo necesarias. Si un usuario se queja de que "siente la aplicación lenta", debemos poder validar si eso es cierto y si lo es determinar el componente responsable del problema. Cuando vivíamos en un mundo cliente/servidor, todo era más sencillo porque como la aplicación corría en una PC, normalmente el problema era invariablemente la base de datos. En un mundo SOA la situación es mucho más compleja porque el problema se puede dar en diversos componentes (servicios que corren en servidores de aplicaciones, colas de mensajes, brokers, adaptadores, aplicaciones legadas y bases de datos). En esta situación no se puede depender de la intuición, es necesario disponer de herramientas de diagnóstico que nos ayuden a ubicar el problema. Para resolver este tipo de situaciones complejas que enfrentan las áreas de TI que han adoptado una arquitectura basada en servicios, IBM ha creado el producto IBM Tivoli Composite Application Monitor (ITCAM) y otros proveedores de software están trabajando también en liberar productos similares. Este tipo de productos se está volviendo crítico para las áreas de operaciones que tienen que enfrentar un cambio radical en lo que se espera de ellos.

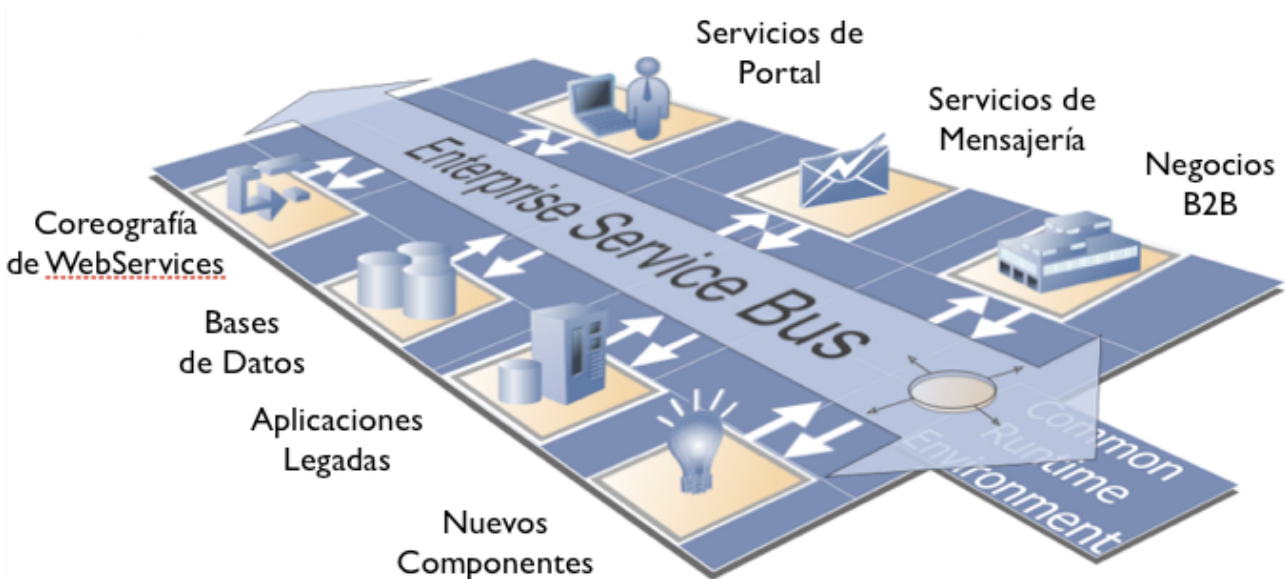
Conclusión

En esta segunda fase se instaló la infraestructura necesaria para comunicar los servicios entre sí y crear un Enterprise Service Bus que permitiera comunicar de manera segura los componentes entre sí, haciendo especialmente hincapié en los problemas que pueden surgir en materia de seguridad, calidad de servicio y monitoreo si no se planean adecuadamente. El ESB es el corazón de SOA y si no funciona adecuadamente es imposible aspirar a tener aplicaciones en las que los procesos de negocio estén separados de los servicios, como se requiere en un esquema de Business Process Management que analizaremos en la siguiente fase.

Business Process Management

En esta última etapa se pretende utilizar la infraestructura de SOA creada en la etapa anterior para comunicar aplicaciones en un esquema de EAI estándar y aprovecharla ahora para crear nuevas aplicaciones basadas en procesos de negocio. Lo que resulta novedoso es que los procesos ya no se programan sino que se definen gráficamente. Esto permite que finalmente sea más sencillo cambiar los procesos sin tener que hacer cirugía mayor a las aplicaciones tal y como estábamos acostumbrados en el pasado.

La consecuencia directa de separar servicios y procesos es que la capa de presentación queda igualmente automáticamente separada del resto del código, lo cual es beneficioso porque permite a los desarrolladores especializarse en un determinado tipo de programación, aunque todos los componentes reposen sobre las bases conjuntas de J2EE.



Capa de presentación

Al adoptar BPM, la capa de presentación se convierte simplemente en el interfaz entre el usuario y el motor de procesos. Como los procesos se exponen como servicios (tanto síncronos como asíncronos), en principio la capa de presentación podría ser desarrollada en cualquier lenguaje que permita desarrollar páginas de web dinámicas e invocar esos servicios web. La realidad es un poco más compleja porque tenemos que tener en cuenta la seguridad y administración del sistema.

Lo primero que tenemos que tomar en cuenta es que no todos los usuarios van a ser usuarios de todos los servicios. Por lo tanto, es necesario restringir el acceso al acceso de estos servicios. Esto no es tan sencillo como puede parecer ya que se puede dar el caso de que un servicio pueda ser utilizado solo por aquellos usuarios que tengan un determinado rol o aquellos usuarios que tengan alguno de los roles autorizados. Además esto puede ser muy dinámico, un día una perso-

na tiene un determinado rol y al día siguiente es ascendido y adquiere otro rol. Por lo tanto, la seguridad se debe administrar fuera del sistema. Esto se hace adoptando un estándar de la industria, utilizando un directorio LDAP.

Si bien utilizando un directorio LDAP sería factible desarrollar páginas que solo pudieran ser accedidas por usuarios con determinados roles, la realidad es que si se producen cambios a los roles autorizados, es necesario editar la página manualmente. Además, la experiencia de uso no sería agradable porque los usuarios se encontrarían constantemente con páginas que no pueden acceder. Por todo lo anterior, la mejor alternativa para desarrollar la capa de presentación es un portal basado en tecnología J2EE con soporte para portlets estándar (JSR 168), como por ejemplo WebSphere Portal. La ventaja de los portlets es que el administrador del portal puede decidir en cualquier momento qué usuarios lo pueden usar y quiénes no. Además, al momento de personalizar su página de inicio los usuarios pueden decidir qué portlets van a utilizar y cuáles no, pero de una lista en la que solo van a aparecer aquellos portlets que están autorizados a utilizar.



Por todo lo anterior, vemos que en un ambiente de BPM, la adopción de un portal es algo obligatorio para simplificar el control de acceso a los procesos y que otras consideraciones como publicación de contenido o sistema de búsqueda de información que normalmente son las principales a la hora de evaluar un portal, aquí pasan a un segundo término.

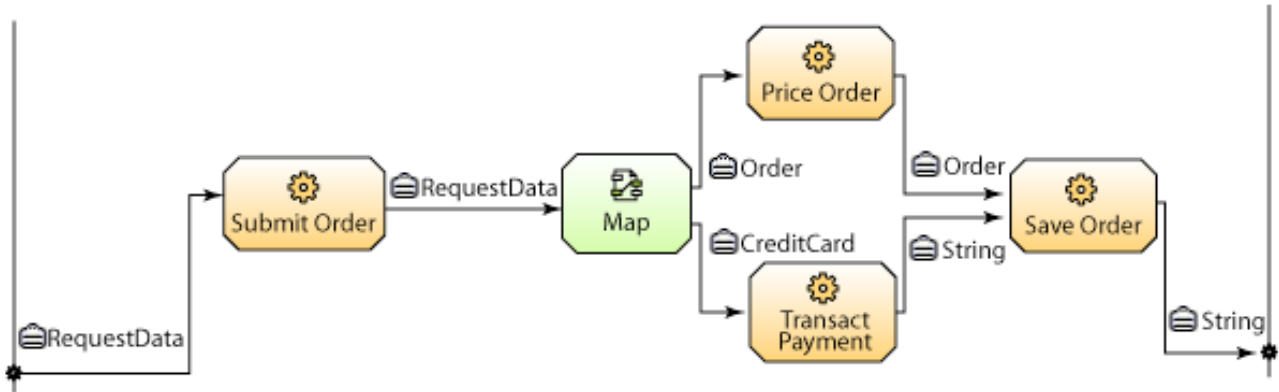
Para el desarrollo de la capa de presentación se necesitan programadores con experiencia en Java que sean especialistas en HTML, JavaScript, JSP, portlets, Java Server Faces y que por supuesto sepan crear clientes de servicios web a partir de un archivo WSDL.

Definición de procesos

El éxito de los servicios web y la adopción de SOA como herramienta de EAI hicieron que rápidamente surgiera la necesidad de poder crear nuevos servicios combinando servicios existentes dentro de un flujo automatizado. A esto se le llamó coreografía de web services. Para ello era necesario crear un nuevo estándar y entre BEA, Microsoft, IBM, SAP y Siebel se creó BPEL4WS,

que iría evolucionando hasta convertirse en WS-BPEL 2.0, ahora controlado por OASIS, la misma organización que controla UDDI.

WS-BPEL fue una pieza clave para lograr que SOA permitiera crecer más allá del mundo de EAI para el que se había creado. Con WS-BPEL se podían empezar a modelar procesos de negocio complejos sin necesidad de programar y eso fue lo que permitió que surgiera el concepto de BPM.



Sin embargo, rápidamente la industria se dió cuenta de que WS-BPEL no era la solución a todos los problemas. Con WS-BPEL era fácil crear procesos de negocios automatizados, pero la intervención de personal humano, con funciones típicas de herramientas de workflow como escalamiento automático o distribución del trabajo entre miembros de un grupo, es algo para lo que este estándar no fue diseñado.

Modificar el estándar para incorporar las funciones necesarias realmente no era una opción porque hubiera roto el diseño original del estándar. Por lo tanto, BEA e IBM decidieron que se necesitaba un nuevo estándar que permitiera ir más allá de lo que ofrece WS-BPEL. Ese nuevo estándar, ahora soportado también por SUN y Oracle entre otros, se conoce como SCA (Service Component Architecture). Con SCA es posible modelar procesos usando componentes. Uno de esos componentes puede ser un proceso WS-BPEL, pero también puede ser un componente que modela interacciones humanas o un motor de reglas. Lo importante de SCA es que no sustituye a WS-BPEL sino que lo complementa.

Tanto WS-BPEL como SCA son estándares técnicos demasiado complejos para poder ser entendidos por usuarios finales. Sin embargo, el objetivo de BPM es lograr que sean los usuarios de negocios quienes definan los procesos. La solución que ofrece IBM es un producto que se llama WebSphere Business Modeler. Este producto permite modelar procesos sin entrar en aspectos técnicos de manera a que pueda ser utilizado por usuarios finales. Los procesos se pueden exportar como archivos BPEL (incompletos) que sirven de base a los técnicos para ponerlos en producción en WebSphere Process Server, un producto que soporta tanto BPEL como SCA.

Esto significa que en el estado actual de la tecnología, para poder implementar BPM, una empresa necesita especialistas de integración que conozcan bien WS-BPEL, SCA y SDO (Service Data Objects, el estándar para representar documentos XML como un objeto Java). Es recomendable que al menos originalmente estos especialistas estén presentes con los usuarios de negocios cuando modelen sus procesos.

Los procesos que se ponen en producción, al final quedan expuestos como servicios. Por lo tanto también deben ser publicados en el registro UDDI para permitir su reutilización y facilitar lo que se conoce como "governance".

Monitoreo de los procesos

Una de las principales ventajas de modelar los procesos de manera independiente al desarrollo de los componentes es que ahora resulta trivial incluir puntos de control en el mismo para obtener medidas que sean relevantes para el negocio. Con el monitoreo de los procesos se cierra el círculo virtuoso del BPM. La información obtenida se almacena en una base de datos, lo que permite obtener agregados que permiten analizar a fondo el desempeño del proceso y la introducción de mejoras al mismo si fuera necesario.



El monitoreo de web services implica almacenar datos cada vez que se ejecuta un proceso. Esto puede significar una presión fuerte sobre la base de datos en la que se guarda la información. Por lo tanto, este repositorio debe ser monitoreado y administrado por un DBA, al igual que cualquier otra base de datos utilizada en la empresa.

Sin duda, el monitoreo de los procesos es la última fase en la adopción de BPM debido a que para ello se requiere tener una infraestructura SOA completa y que los procesos hayan sido modelados por los usuarios de negocio ya que son estos últimos los que definen las medidas que se van a monitorear. Además, el resultado de estas mediciones se observa a través del portal, para simplificar la creación de un tablero de control para los ejecutivos de la empresa, por lo cual ese producto también debe estar instalado y la seguridad configurada.

Conclusión

En esta última fase se aprovechó la infraestructura creada en las dos fases anteriores para desarrollar aplicaciones en base a servicios existentes, controlados por un proceso de negocios avanzado en el que colaboran personal humano y tareas automatizadas. Estas aplicaciones son accedidas desde un portal, lo que simplifica la administración y distribución de las aplicaciones.