Artículo de Revisión

PROPUESTA DE INTERACCIÓN ENTRE SISTEMAS

A proposed interaction between systems using automatic notification

Janneth Pardo Pinzón[1*] *y José Antonio Valero Medina*[2]†
[1] Universidad Distrital "Francisco José de Caldas", [2] Universidad Distrital "Francisco José de Caldas"
Correspondencia: [*] jpardop@udistrital.edu.co [†] jvalero@udistrital.edu.co

Recibido: 23 de junio de 2011 Aceptado: 15 de septiembre de 2011

Resumen

Uno de los retos que enfrentan las organizaciones actuales es el problema de interoperabilidad entre los diferentes sistemas de información debido a que los servicios se ofrecen a los usuarios mediante aplicaciones separadas, restringiendo de esta manera el desarrollo de consultas integrales y, por tanto, la visión y la proyección organizacional. Este hecho ha originado el desarrollo de nuevos contextos enfocados en superar la ineficiente conectividad entre sistemas heterogéneos y las dificultades de acceso transparente a los datos para poder compartir eficientemente los recursos. El interés de este artículo es hacer una breve reseña de las diferentes estrategias de conectividad entre sistemas heterogéneos, así como plantear una arquitectura prototipo que permita interconectar fuentes de datos heterogéneas y compartir entre ellas notificaciones de forma automática cada vez que se produzcan cambios en sus datos.

Palabras clave: arquitectura orientada a servicios (SOA), enterprise service bus (ESB), integración de aplicaciones, wrapper, sistemas heterogéneos.

Abstract

One of the challenges that face the current organizations is the problem of interoperability among the different information systems, since that services are offered to the users through separated applications, restricting in this way the development of integral consultations and therefore the vision and the organizational projection. This fact has led to the development of new contexts focused on overcoming the inefficient connectivity between heterogeneous systems and the difficulties of transparent data access to share the resources in an efficient way. The interest of this paper is to briefly review the different strategies of connectivity among heterogeneous systems, as well as to present a prototype architecture that allows interconnecting heterogeneous data sources and sharing notifications between them in an automatic way, whenever changes occur in their data.

Key words: Service oriented architecture (SOA), enterprise service bus (ESB), applications of integration, wrapper, heterogeneous systems.

Introducción

Las organizaciones actuales buscan continuamente estrategias e infraestructuras que les permita evolucionar y mantenerse competitivamente en el mercado (Ross *et al.*, 2006). Una de esas estrategias es lograr la articulación entre los sistemas existentes tanto al interior de las organizaciones como fuera de ellas. Generalmente, las compañías trabajan aunadamente para generar conjuntos de servicios a una colectividad (Auer, 2008), sin embargo, es posible que la toma de decisiones esté fundamentada en bases de datos desactualizadas, ya sea porque no existen los apropiados mecanismos de interconexión entre los sistemas o porque no se cuenta con una estrategia tanto de acceso transparente a los datos variables como de

notificación de cambios que suceden en un sistema y que otro necesita conocer.

La carencia de una adecuada estructura que permita conocer de forma automática el continuo cambio de información en las diversas fuentes de datos se debe a la heterogeneidad de los sistemas existentes en una organización. Muchas organizaciones están compuestas por diversas dependencias que han automatizado separadamente sus necesidades, adquiriendo para ello una serie de recursos tecnológicos desacoplados (Curl y Fertalj, 2009). Sin embargo, es necesario desarrollar nuevas tecnologías que permita interconectar, reutilizar y hacer extensibles cada uno de los sistemas sin ir en detrimento de la autonomía que disfrutan las diferentes dependencias (Mecella y Batini, 2000).

Con el propósito de corregir el problema de interoperabilidad entre los sistemas, ya sean sistemas heredados (Mecella y Batini, 2000) o nuevos desarrollos, se han diseñado una serie de mecanismos como interfaces flexibles y dinámicas que encapsulan los objetos o las clases para integrar las funcionalidades deseadas y permiten ofrecer servicios más completos (Sneed y Majnar, 1998; Kim y Bienan, s. f.). Se han desarrollo, además, diversas arquitecturas como la arquitectura orientada a servicios (SOA) la arquitectura orientada a eventos (EDA) (Minglun *et al.*, 2008), el bus de servicios empresariales (ESB) (Garcés-Erice, 2009) y la integración web, entre otros (Finkelstein, 2006). Sin embargo, no ha sido fácil de desarrollar la interacción en línea y automática tanto al interior de las organizaciones como fuera de ellas, debido a que muchos de los sistemas heredados que operan en estas no fueron implementados para este propósito.

Este artículo está dividido en tres secciones: la primera presenta una breve reseña de algunas tecnologías que se han diseñado para el proceso de intercambio de información, la segunda presenta la arquitectura propuesta para el mecanismo de notificaciones automáticas y, finalmente, se cierra con algunas conclusiones.

Antecedentes

Dentro de las estrategias que se han desarrollado para permitir la interoperabilidad entre los sistemas de información se pueden resaltar: los procesos para correlacionar los modelos, el desarrollo de traducción de esquemas, el hecho de ejecutar tareas conjuntas para permitir el intercambio de información y los procesos para identificar las operaciones remotas que realizan las aplicaciones que se van a integrar. El propósito ha sido siempre lograr interoperabilidad ya sea a nivel de estructuras, de modelos organizacionales, de niveles de abstracción o de sistemas operativos para compartir recursos y ponerlos a disposición de un gran número de usuarios.

Es la complejidad de los sistemas locales y su inflexibilidad lo que hace difícil generar una conexión eficaz, pues los sistemas heredados no cuentan con una interfaz que les permita el intercambio de información, por tanto, no se pueden compartir los recursos de manera eficiente. Un alto porcentaje de los procesos de comunicación e interacción que se realizan en las organizaciones se hacen de forma manual o semiautomática (De Maria, 2002), lo que conlleva un nivel de error elevado, convirtiendo los sistemas en ineficientes y obsoletos (Wang *et al.*, 2007).

Por otro lado, el aumento vertiginoso de la información ha desencadenado un masivo desarrollo de sistemas de aplicación que han entrado a formar parte de la estructura organizacional pero sin un adecuado acoplamiento; surge entonces la necesidad de integrar gran cantidad de fuentes de datos heterogéneas. En este sentido, son varias las organizaciones que han comenzado a desarrollar soluciones integradas para poder compartir la información haciendo así frente a los desafíos actuales y mejorando sus procesos y servicios.

En la siguiente sección se da una mirada a las diferentes estrategias de interoperabilidad que se han ido implementando.

Estrategias de interoperabilidad entre sistemas

Los mecanismos de interoperabilidad, es decir, la habilidad que tienen los sistemas para comunicarse entre sí o desarrollar tareas conjuntas, no necesariamente hacen referencia a una conexión completa que involucre integración de redes, de sistemas y de información, sino que se han diseñado para resolver uno o varios de estos aspectos de acoplamiento de manera que el resultado no sea una solución confusa (Teixeira *et al*., 2011).

La interoperabilidad entre sistemas se logra cuando se establece una relación entre sus componentes (Bass *et al*., 1998), básicamente utilizando interfaces flexibles para proveer servicios o reutilizar funcionalidades (Li *et al*., 2000); lo que se constituye en una poderosa herramienta para coordinar una red de servicios y obtener una adecuada elasticidad en el flujo de la información dentro de la dinámica organizacional. Es por esto que hoy en día la eficiencia de un sistema informático no solo se mide por su capacidad en la obtención de resultados, sino por la flexibilidad en los procesos de integración con otros sistemas.

El éxito para lograr una adecuada interoperabilidad requiere tanto de métodos como de una adecuada arquitectura. Los métodos están enfocados a enviar información entre aplicaciones o a definir *wrapper* para funcionalidades y servicios, y las arquitecturas se pueden categorizar en varios tipos: las primeras arquitecturas que se desarrollaron de tipo físico estaban relacionadas con la topología y los sistemas operativos, entre otros; las arquitecturas de redes que se desarrollaron estaban relacionadas con los protocolos, los *routers* y los *gateway*, mientras que las arquitecturas de aplicaciones estaban definidas con base en una serie de librerías o capas, y por último, la arquitectura lógica hacía referencia a la configuración del software (Zahavi, 2000).

No obstante, los nuevos diseños arquitecturales no solo tratan de resolver los problemas de acoplamiento entre los sistemas existentes sino que, además, contemplan la posibilidad de dejar abierta una conexión con los futuros desarrollos como una estrategia para optimizar los objetivos del negocio (Hur *et al*., 2009). Se enumeran a continuación algunas de las herramientas más eficaces para generar interacción entre sistemas heterogéneos.

Arquitectura orientada a servicios (SOA)

No existe un criterio estándar para definir la arquitectura orientada a servicios (SOA), pero se puede interpretar como una estructura de diseño flexible, desarrollada para enmascarar el problema de la heterogeneidad ente los sistemas distribuidos (Jeng y An, 2007). Es un modelo que permite reutilizar funcionalidades distribuidas generando uniformidad en los medios en donde interactúan los componentes de los servicios, de manera que permita optimizar el flujo de información (Wang *et al*., 2007; Walend, 2006).

La arquitectura SOA se basa en el hecho de que cada sistema está compuesto por una serie de servicios que son independientes de la aplicación, y que realizan funcionalidades específicas. Sin embargo, se busca que mediante la coalición de estos servicios sea posible crear nuevas aplicaciones o diseñar un servicio de orden superior, que brinde un débil y flexible acoplamiento, y que garantice un sistema que pueda adaptarse fácilmente a los cambios (Wang *et al*., 2007; Brown, 2008).

La necesidad de implementar SOA se presenta cuando existen cambios fundamentales en los negocios o por las nuevas expectativas que requieren ser resueltas. Dentro de las características de SOA se pueden mencionar la comunicación sincrónica punto a punto y el hecho de que permite implementaciones independientemente de las plataformas, del sistema operativo y del modelo de datos, y la ventaja de que los servicios pueden ser reutilizados y definidos con base en los requerimientos del negocio para lograr una adecuada respuesta empresarial (Lawer y Howell-Baber, 2007).

La forma más utilizada para lograr la implementación de esta arquitectura son los servicios web que utiliza XML para representar apropiadamente los datos y es un estándar recomendado por el World Wide Web Consortium (W3C) (2010), el Web Services Description Language (WSDL) para describir las interfaces, el Simple Object Access Protocol (SOAP) para generar intercambio de datos y el Bus de Servicios Empresariales (ESB) para generar interoperabilidad mediante mensajería o eventos (Thiran y Hainaut, 2001; Wang *et al*., 2007). Se hará referencia entonces a dos de las tecnologías más utilizadas.

Esquemas de mensajería basados en XML. La interoperabilidad entre sistemas se puede obtener haciendo uso

del lenguaje de marcado extensible (Extensible Markup Language), desarrollado por W3C (2010), y que se concibe como una forma de definir lenguajes para el intercambio de información entre diversas aplicaciones. El formato unificado XML permite enmascarar las diversas estructuras de los datos origen ya sean de tipo semiestructurado o no estructurado, por tanto, las etiquetas XML no están predefinidas sino que su diseño se puede construir a partir de los datos seleccionados (Severson y Fife, 2003). Hoy en día es uno de los formatos más utilizados para el intercambio de información estructurada entre diversas aplicaciones.

De esta manera, cuando se plantea un conjunto de solicitudes que necesitan ser resueltas, estas se registran empleando esquemas basados en XML. Es importante resaltar que la independencia de los datos XML no guarda ninguna relación con las plataformas donde están estructurados los datos, así que para consumir los servicios expuestos no es necesario ajustarse al lenguaje empleado por cada sistema.

Bus de Servicios Empresariales (ESB). Se puede definir como una capa de software basada en protocolos, que permite construir una gran arquitectura de servicios utilizando para ello un motor de mensajería o notificación de eventos. De esta manera, un ESB comienza a mostrarse como una respuesta potente y flexible para solucionar el problema de la interoperabilidad entre servicios (Wen et al., 2009). Actualmente es una de las plataformas de conectividad más utilizada.

ESB es una plataforma de integración compuesta básicamente de tres componentes adaptadores para proporcionar bidireccionalidad, acceso y lectura de datos, elementos que permiten integrar las salidas en diversos formatos, combinar los servicios web, transformar los datos, y proporciona un proceso de direccionamiento para conectar las aplicaciones (Chongshan y Shasha, 2010).

Una aplicación o servicio se conecta a un ESB mediante un *gateway*, el cual puede ser configurado en una tabla o directorio donde se definen una serie de funciones que incluyen llamada y direccionamiento a servicios, y validación del contenido de mensajes. La tabla donde se configuran las aplicaciones actúa como un controlador para definir el flujo de mensajes; cada acción es ejecutada mediante sentencias previamente definidas a partir de variables y operadores asociados a cada una de ellas.

Cuando el mensaje llega proveniente de una aplicación debe ser identificado y su esquema debe ser claramente entendido, de manera que las reglas de la aplicación o del servicio pueden ser empleadas para el procesamiento del mensaje, el cual posteriormente es enviado a la aplicación destino con base en el proceso de direccionamiento que está asociado a un servicio de nombres y su proveedor (direccionamiento: = servicio_nombre, uri).

ESB se puede entender como un híbrido entre arquitectura orientada a servicios, las tecnologías *middleware* tradicionales y el modelado de los procesos del negocio. Es una forma rápida en que las organizaciones logran integrar sus aplicaciones independientemente de su localización (Wen *et al.*, 2009).

Tecnologías middleware

El término *middleware* presenta varias definiciones, se conoce también como tecnología *plumbing* porque se utiliza para conectar componentes de aplicaciones distribuidas (Ariannejad, 2001). Zahavi en 1995, utilizó el término para describir la comunicación entre varias bases de datos, posteriormente se utilizó junto con ORB para describir su función en la integración de procesos. Hoy en día se usa como una infraestructura que permite el intercambio de información entre aplicaciones, bajo niveles funcionales como: tuplas distribuidas, llamadas a procedimientos remotos (Myerson, 2002), *middleware* orientado a mensajes (Cole-Gomolski, 1997), *middleware* de objetos distribuidos, tecnología de acceso a base de datos, *framework* orientado a componentes, servicio de directorio y servidores de aplicaciones (Sharfudeen, 2007).

Middleware es una capa de software que no solo permite la interacción entre los componentes de diversas aplicaciones, sino que además se utiliza para dar respuesta a los obstáculos de comunicación por problemas en la red. Su implementación es independiente de los lenguajes de programación y de las plataformas en que hayan sido implementados los sistemas por integrar (Li y Zhou, 2010).

Esta capa de software proporciona una plataforma de aplicaciones unificada con un esquema global de datos, su configuración puede ser bastante flexible puesto que las aplicaciones se pueden adicionar o

eliminar fácilmente; además, permite el acceso a los datos en tiempo real.

Existen diferentes tipos de interfaces para el manejo de *middleware* como las application program *interface* (API) y los *scripts*. Las API se pueden invocar para manejar el *middleware* directamente, solo es necesario proporcionar la información sobre la llamada al método objetivo, incluyendo el nombre del paquete y su localización, el nombre del método y sus parámetros. Por otro lado, las reglas para los *scripts* incluyen ruta de acceso y el nombre de los *scripts* (Ibrahim, 2009).

Wrapper para recursos heterogéneos

La apremiante necesidad de facilitar a los usuarios finales acceso transparente a los recursos ha generado el desarrollo de una técnica de reingeniería denominada *wrapper*, que tiene por objeto facilitar el acceso a funcionalidades a través de una interfaz predefinida. Se puede considerar como una especie de intermediario entre dos interfaces incompatibles que fueron diseñadas en diferentes plataformas y tiempos, pero cuya conexión se obtiene mediante el encapsulamiento de sus funcionalidades (Gamma *et al.*, 1999).

El término *wrapper* está asociado con los patrones estructurales (Kramek, 2007) que tratan de optimizar la forma en que se deben combinar las clases y los objetos para generar estructuras más complejas (Kuchana, 2004; Kramek, 2007). Para ello, se han diseñado interfaces que soportan los objetos y a partir de las cuales están dadas las conexiones. Los patrones estructurales adaptador, decorador y proxy se conocen también como *wrapper* porque su finalidad es encapsular una clase y modificar el comportamiento de la misma.

El patrón adaptador permite que dos clases que no están relacionadas debido a la incompatibilidad de sus interfaces puedan trabajar juntas, el proceso de adaptación es generar una forma de expresión igual para las clases y así conseguir una rápida solución, la interfaz es modificada sin ampliar la funcionalidad (Vélez *et al.*, s. f.).

El patrón decorador tiene como objetivo ampliar la funcionalidad sin intervenir con la interfaz, adiciona funcionalidades para algunos objetos, se parece a una herencia de clases (Sharfudeen, 2007; Vélez *et al.*, s. f.) pero su utilidad radica más en la flexibilidad. Se pueden adicionar funcionalidades más de una vez, aunque la sugerencia es que sea una clase abstracta y la aplicación se derive de ella.

Por su parte, el patrón proxy es un *wrapper* que tiene una interfaz y una funcionalidad idéntica a la clase que se quiere envolver, cuando se implementan los métodos de la clase proxy lo que se hace es invocar al objeto real, encapsulando y controlando el acceso al objeto dentro de una clase más cercana, generalmente se trata de un objeto remoto (Buschamann *et al.*, 1996; Elish, 2006).

Bodega de datos

La bodega de datos o Datawarehouse (DW) surge en los años noventa como una herramienta que permite integrar los datos más relevantes de una organización, que están almacenados en fuentes de datos heterogéneas. Inmon (1992) la define como "una colección de datos orientados por temas, integrados, variables en el tiempo y no volátiles para el apoyo de la toma de decisiones". El DW utiliza una capa intermedia para almacenar el conjunto de datos que no son transitorios dentro de la organización y que presentan una variabilidad temporal. Su objetivo está orientado a manipular grandes volúmenes de datos y a permitir a los usuarios finales la toma de decisiones estratégicas.

La integración de las diversas fuentes de datos se logra mediante procesos ETL (extracción, transformación y carga), que son el fundamento de las bodegas de datos; la información es extraída de las fuentes de datos, y pasa luego por una serie de transformaciones antes de ser almacenada en la bodega de datos. ETL se compone de seis procesos: seleccionar los datos que se van a extraer, mapear los datos de las fuentes de datos heterogéneas, transformar las fuentes, conectarlas, seleccionar el destino para la carga de datos, unir los atributos de las fuentes de datos origen con los atributos de las fuentes de datos destino, y cargar los datos en las bodegas de datos (Lunan, 2010; Wang, 2010).

Estas bodegas están acompañadas de los correspondientes metadatos, es decir, un tipo de datos que describen la estructura de los mismos, su ubicación, las características que identifican los recursos, los métodos de comunicación de cada uno de los datos originales, y los métodos que se establecen

en el DW para facilitar el direccionamiento adecuado de la información; una mala gestión de los metadatos puede llevar a una ineficiencia en los procesos (Wang, 2010).

Sin embargo, es de anotar que el DW presenta ciertos riesgos en su implementación como el acceso restringido a los objetos por efectos de seguridad, y la desactualización de los esquemas de datos frente a las nuevas necesidades del negocio; los metadatos deben ser continuamente actualizados de manera que los usuarios puedan contar con una descripción óptima de los recursos que desean consultar, de lo contrario se podrían estar dando falsas expectativas en cuanto a los recursos expuestos.

Prototipo de interacción entre sistemas mediante notificaciones automáticas

Otra forma de generar interoperabilidad entre sistemas heterogéneos es mediante el proceso de notificaciones automáticas. Un sistema de notificaciones se puede definir como un híbrido entre una red de servicios y toda la infraestructura que representa la entrega de un mensaje a múltiples usuarios o a un usuario, sobre la ocurrencia de un evento que es de su interés. El sistema de notificaciones está compuesto por tres actores: quien produce la notificación, quien la recibe o está interesado en el evento y un sistema de gestión de notificaciones.

Se desarrolla un sistema de intercambio de información que recurre al modelo publicación/suscripción y contempla tres tipos de servicios: consulta, solicitud y notificación; sin embargo, se debe resaltar que independientemente del tipo de servicio, se define inicialmente una fuente de datos que se toma como base para crear el esquema de datos que tendrá asociado el servicio. Para la consulta, los datos que se obtienen como parte de la utilización del servicio no retornan a la entidad que los solicita, sino que se direccionan a la entidad que los posee para que desde allí sean presentados al usuario; es decir, que los datos no salen de la entidad que está siendo consultada. Para el servicio de solicitud, el usuario elige la fuente de datos y específica los parámetros que requiere, con base en la estructura de datos que tiene configurada la fuente seleccionada; los parámetros seleccionados son almacenados en una estructura estándar jerárquica, se gestiona la solicitud y posteriormente se responde a los consumidores mediante el envío de un mensaje. El servicio de notificación tiene asociados dos tipos de actores: unos productores quienes se registran por temas, generan las notificaciones con la asistencia de un usuario final (operador) y las publican en el sitio donde se desarrolla el intercambio, teniendo en cuenta un estilo estándar y el filtro por tipos de notificaciones; posteriormente, estas son enviadas a los consumidores. Por otro lado están los consumidores que generan una solicitud de notificación de eventos con algunos criterios específicos, los cuales son registrados en el momento de la suscripción al sistema gestor de notificaciones. El sistema de gestión de notificaciones se puede definir como una interfaz estándar que regula las interacciones entre consumidores y proveedores de forma dinámica y anónima. El proceso de notificaciones se aprecia en la figura 1.

Es necesario resaltar que el sistema de intercambio de información soporta además fuentes de datos geográficas a través de servidores web para mapas articulados sobre ArcIMS, es decir, que este desarrollo provee las funcionalidades necesarias para conectarse a fuentes de datos gráficas y obtener consultas de mapas y datos geográficos. El usuario puede seleccionar los datos alfanuméricos y sus correspondientes datos gráficos para generar su consulta.

El inconveniente del esquema de notificaciones propuesto inicialmente consistía en una etapa asistida por un operador en la que, aunque el envío de las notificaciones se realizaba de forma automática, la generación del evento debía realizarse de manera asistida. Es decir, que para enterarse de los cambios sucedidos se estaba supeditado a que un usuario (operador) ingresara a realizar dicha actividad, con todos los riesgos que ello implicaba (que no ingresara, que indicara como cambio algo que en realidad no lo era, etc.); otras desventajas de este sistema radicaban en que las notificaciones se enviaban sin tener en cuenta la periodicidad de cambios requerida por los consumidores y en la imposibilidad que tenía el operador de particularizar la modificación de los datos, es decir, señalar el tipo de operación que se realizaba en la fuente de datos si el proceso consistía en una inserción o en la eliminación de un registro, por cuanto no había seguimiento y persistencia del tipo de operación realizada; el sistema no contaba con un historial de notificaciones que almacenara los cambios detectados en un delta de tiempo para cada una de las fuentes de datos que solicitaba el servicio de notificaciones automáticas.

A fin de contribuir con el enfoque de integración entre sistemas, se propone una red de intercambio de información mediante notificaciones automáticas, la cual se concibe como

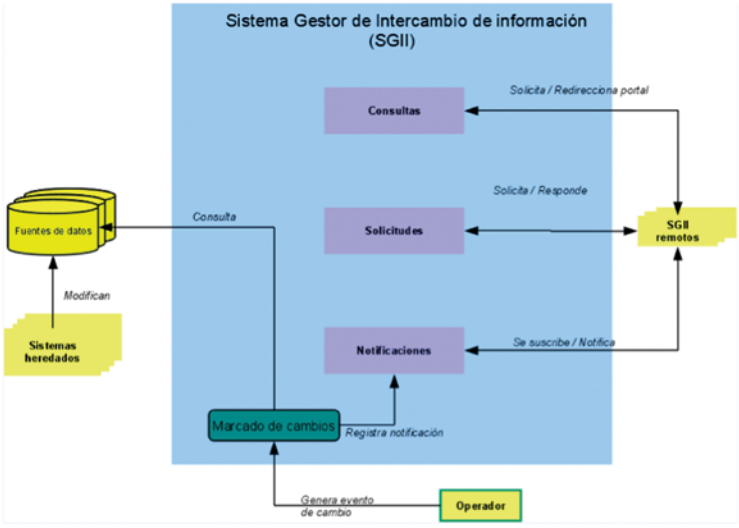

Figura 1. Proceso de notificaciones actual

un prototipo automatizado de información que permite notificar cambios de los datos entre fuentes de información heterogéneas. Entre las bondades que se pueden resaltar de este módulo están: la discriminación de los cambios realizados en una fuente de datos de interés, la configuración de una periodicidad para la cual el interesado puede recibir avisos del estado actual de los datos que le interesan, el préstamo de un servicio enmarcado en los horarios de atención que cada administrador de la fuente de datos local considere pertinentes a fin de no afectar el desempeño de cada sistema local. No obstante, aunque las notificaciones se pueden generar para sistemas en línea y fuera de línea, la propuesta únicamente contempla el segundo alcance.

El sistema que se propone es una capa de software que proporciona interoperabilidad en línea entre fuentes de datos heterogéneas mediante un mecanismo de notificación de cambios y utiliza para ello elementos de acceso, lectura de datos y enrutamiento para generar las diferentes conexiones y envío de mensajes, similares al esquema planteado en un ESB.

El sistema de notificaciones automáticas presenta un servicio de configuración que se encarga de registrar un conjunto de servicios de solicitud a los cuales se les desea generar un seguimiento, teniendo en cuenta un intervalo de tiempo en el cual se desarrolla este proceso. El esquema del sistema de notificaciones propuesto se muestra en la figura 2. Dado el interés general del sistema, una de las primeras condiciones que es necesario contemplar es el acceso con cierta periodicidad a los cambios ocurridos en las fuentes de datos.

Sin embargo, la fase previa al servicio de notificaciones es la creación del perfil de los usuarios: aquí el usuario señala cuáles son sus preferencias teniendo en cuenta los criterios establecidos por cada servicio. Se crea, entonces, un repositorio donde se almacena esta información, y se desarrolla un conjunto de reglas de autenticación y privilegios lo suficientemente generales para permitir la creación de perfiles por parte del usuario. Cabe señalar que el sistema de notificaciones es una etapa posterior que se integra al de autenticación de usuarios.

Con base en las preferencias de los usuarios y el tipo de información disponible por cada uno de los sistemas adscritos al sistema de intercambio, el mecanismo de notificaciones debe filtrar aquellos eventos que sean de interés, realizando una clasificación para su posterior envío.

Cada configuración de notificaciones se registra mediante la selección de un servicio y un tiempo delta, este último representa la diferencia de tiempo en el que se comparan los cambios entre una transacción inicial y una posterior. Así, el modelo de notificaciones establece el seguimiento asociado a un servicio mediante la comparación de los cambios en las fuentes de datos en el delta de tiempo requerido.

En el sistema de notificaciones automáticas aquí propuesto se destacan tres funcionalidades gruesas: la configuración de notificaciones se comunica a través del portal web y toma los parámetros de seguimiento de cambios indicados por el usuario y los persiste en la base de datos; el modelo

de notificación gestiona los eventos que componen el seguimiento de los cambios, es decir, controla el flujo de ejecuciones por cada configuración hecha en la etapa previa y es el que establece el modelo de comparación de cambios de las fuentes de datos adscritas al sistema de gestión de intercambio de información; el historial de notificaciones muestra el listado de todos los cambios encontrados para cada servicio desde el momento que se ejecuta la primera configuración realizada por el usuario. Se genera un historial de notificaciones que se almacena en la base de datos local, el cual mantiene el registro completo de los cambios hallados en la comparación de dos transacciones, cada delta de tiempo, para cada una de las fuentes de datos que solicitaron el servicio de notificaciones automáticas. Además, se guardan los atributos que identifican el registro modificado: el nombre de la tabla, la columna, la fila y el dato modificado; los demás atributos almacenados corresponden al tipo de operación de cambio, es decir, inserción, borrado o actualización de los datos, y la fecha de registro.

El núcleo del sistema de notificaciones es el modelo de notificación que emplea un algoritmo de comparación para detectar los cambios en las fuentes de datos registradas, y controla los eventos relacionados con el monitoreo de cada servicio y el envío de notificaciones a las fuentes involucradas que solicitaron información de cambios; porque una vez que se identifiquen los cambios se activa el mecanismo de envío de notificaciones hacia cada una de las fuentes de datos, según la periodicidad y las solicitudes de cambio previamente establecidas.

Los detalles de la aplicación se pueden resumir en que existe un flujo de mensajes dividido en tres partes: un flujo de solicitud de notificaciones debidamente registrado, un proceso de comparación para detectar cambios y el registro de un evento de cambio con base en el cual el sistema queda potencialmente habilitado para enviar una notificación de cambios hacia cada uno de los sistemas registrados, que permitirá mantener a los interesados informados de las modificaciones que se detectan.

Conclusiones

En el artículo se presentaron los diferentes mecanismos que se han desarrollado para generar interoperabilidad entre sistemas; sin embargo, la mayoría de los dispositivos desarrollados se han generado de forma independiente y aunque han tenido en cuenta ciertos criterios, procedimientos y esquemas, no se vislumbra una arquitectura que soporte todas las necesidades de conexión y de consulta de forma integral, sino que se pueden reconocer algunas limitaciones porque la mayoría de las estructuras generan transformaciones punto a punto o están limitadas a la información dispuesta en la web y a la periodicidad de consulta que cada suscriptor requiera.

La investigación y el desarrollo en el proceso de interoperabilidad han estado enfocados en las metodologías y arquitecturas que permitan el intercambio de información, sin embargo, el desarrollo de agentes inteligentes capaces de intercambiar

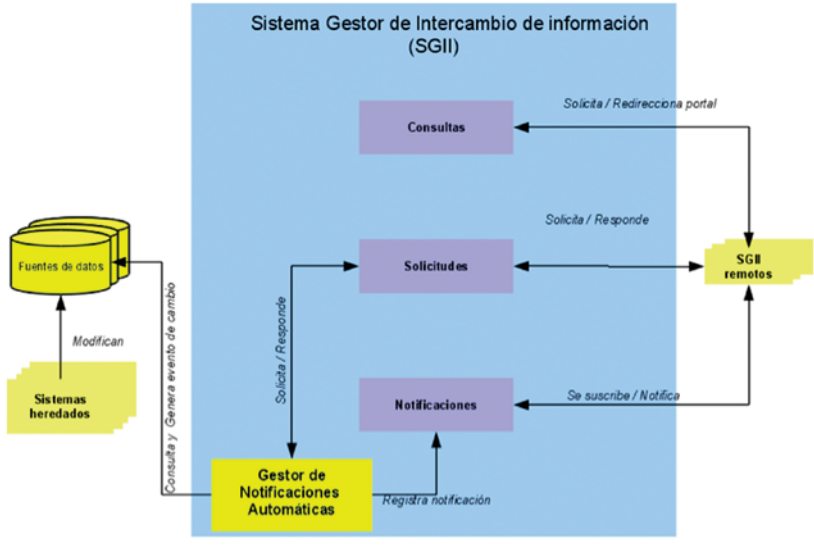

Figura 2. Proceso de notificaciones automáticas

mensajes mediante un lenguaje de comunicación para lograr la interoperabilidad entre aplicaciones a nivel semántico, por ejemplo, está aún en desarrollo. Lo ideal sería establecer mecanismos que identifiquen el comportamiento del sistema y compongan automáticamente la arquitectura adecuada para su eventual conexión con otros sistemas.

Una de las ventajas que representa el sistema de notificaciones automáticas es que reduce el acceso continuo a la red por parte de los suscriptores debido a que no necesitan acceder frecuentemente cada vez que requieran una consulta por efecto de modificación de un dato, sino que de acuerdo con los criterios de interés establecidos en el momento de la suscripción, serán notificados de los cambios que han solicitado. Por supuesto, se asume que la selección de criterios obedece a los datos que continuamente son consultados. El proceso es una manera de hacer frente a los flujos constantes y pequeños de consultas por notificación de cambios.

Referencias bibliográficas

Ariannejad, M. (2001). *Trends in Middleware Systems*. Toronto: Computer Enginnering Research Group.

Auer, L., Strauss, C., Kryvinska, N. y Zinterhof, P. (2008). *SOA as an Effective Tool for the Flexible Management of Increased Service Heterogeneity in Connverged Enterprise Networks*. Vienna: Complex, Intelligent and Software Intensive Systems (CISIS 2008).

Bass, L., Clements, P. y Kazman, R. (1998). *Software Architecture in Practice.* 2 ed. Boston: Pearson Education Inc.

Brown, P.C. (2008). *Implementing SOA: Total Architecture in Practice*. Boston-London: Addison-Wesley Professional.

Buschmann, F., Regine, M., Hans, R., Sommerland, P. y Stal, M. (1996). *Pattern Oriented Software Architecture V1*. New York: John Wiley & Sons.

Cole-Gomolski, B. (1997). Messaging middleware initiative takes a hit. *Computer world*, 31(39), S. inf.

Chongshan, R. y Shasha, W. (2010). Enterprise Information Exchange Platform Based on ESB for Research and Design. Information Science and Management Engineering (ISME), 2010 International Conference of, 484-487.

Curl, A. y Fertalj, K. (2009). A review of enterprise IT integration methods (pp. 107-112). En Luzar-Stier, V., Jarec, I. y Bekic, Z. (eds.), *Information Technology Interfases*. Cavtat-Dubrovnik: University of Zagreb, University Computing Centre.

De María, J. A. (2000). Integración de aplicaciones encapsuladas para el desarrollo de sistemas de información cooperativos (Tesis de Maestría). Uruguay: Instituto de Computación, Facultad de Ingeniería, Universidad de la República.

Elish, M. (2006). Do Structural Design Patterns Promote Design Stability? Computer Software and Applications Conference, 1, 215-220.

Finkelstein, C. (2006). *Enterprise Architecture for Integration: Rapid Delivery Methods and Technologies*. Boston, London: Artech House Mobile Communications Library.

Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1999). *Design Patterns: Elements of Reusable Object-Oriented Software*. New York: Addison Wesley.

Garcées-Erice, L. (2009). *Building an Enterprise Service Bus for Real-Time SOA: A Messaging Middleware Stack*. Zurich: IBM-Research.

Hur, R., Youn, H. y Lee, E. (2009). Service oriented architecture implementation model for enterprise collaboration. *Communications and Information Technology*. 9th International Symposium, 495-500.

Ibrahim, N. (2009). Orthogonal Classification of Middleware Technologies. Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 46-51.

Inmon, W. (1992). *Building the Data Warehouse*. Indianapolis: Wiley Publishing.

Jeng, J. y An, L. (2007). System Dynamics Modeling for SOA Project Management [Versión electronic]. IEEE Computer Society, S. inf.

Kramek, A. (2007). Design Patterns – Adapters and Wrappers; What is an adapter and how do I use it? [versión electrónica]. Andy´s Weblog. Recuperado el 10 enero de 2010, de http://weblogs.foxite.com/andykramek/archive/2007/01/07/3096.aspx.

Kim, H. S. y Bieman, J. M. (s. f.). Migrating Legacy Software Systems to Corba based Distributed Environments through an Automatic Wrapper Generation Technique [versión electrónica]. Proceedings SCI2000 and the 6th International Conference on Information Systems Analysis and Synthesis ISAS2000. Recuperado el 10 enero de 2010 de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.9959&rep=rep1&type=pdf

Kuchana, P. (2004). *Software Architecture Design Patterns in Java*. Washington: Auerbach Publications.

Lawer, J. P. y Howell-Barber, H. (2007). *Service Oriented Architecture. SOA Strategy, Methodology and Technology*. New York: Taylot & Francis Group.

Li, M., Rana, O. y Walker, D. (2000) An XML-Based Component Model for Wrapping Legacy Codes as Java/CORBA Components. High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition, 1, 507-512.

Li, Q. y Zhou, M. (2010). The State of the Art in Middleware. Information Technology and Applications (IFITA), 2010 International Forum on Information Technology and Applications, 1-3.

Lunan, L. (2010). A framework study of ETL processes optimization based on metadata repository. International Conference o Computer Engineering and Technology (ICCET) 2nd, n6, 125-129.

Mecella, M. y Batini, C. (2000). Cooperation of Heterogeneous Legacy Information Systems: A Methodological Framework. New York: Institute of Electrical and Electronics Engineers, 216-225.

Minglun, R., Xiaoying, A. y Hongxiang, W. (2008). Service oriented architecture for inter-organizational IT resources sharing system. En IEEE (ed.), Automation and Logistics, 2008; ICAL 2008 (pp. 2169-2173). Qingdao: IEEE International Conference.

Myerson, J. M. (2002). *The Complete Book of Middleware*. Washington: Auerbach Publications.

Ross, J. W., Weill, P. y Robertson, D. C. (2006). *Enterprise Architecture as Strategy: Creating a foundation for business execution*. Harvard Business School Press.

Severson, E. y Fife, L. (2003). *XML Compenssion: Optimizing Performance of XML Applications*. Colorado: Flatirons Solutions Corporation.

Sharfudeen, A. (2007). *Transparent State Management Using the Decorator Pattern*. The Source for Java Technology Collaboration.

Sneed, H. M. y Majnar, R. (1998). *A Case Study in Software Wrapping*. IEEE Computer Society Press. 86-93.

Teixeira, T., Malo, P., Almeida, B. y Mateus, M. (2011). Towards an Interoperability Management System. Information Systems and Techonologies (CISTI), 2011 6th Iberian Conference, 1-4.

Thiran, J. y Hainaut, J-L. (2001). *Wrapper Development for Legacy Data Reuse*. Washington: Eighth Working Conference on Reverse Engineering (WCRE´01).

Vélez, J., Sánchez, A., Casado, A. y Doblas, S. (s. f.). *Técnicas avanzadas de diseño de software: Orientación a Objetos, UML, patrones de diseño y Java*. Madrid: Universidad Rey Juan Carlos-Escuela Superior de Ciencias Experimentales y Tecnología Ciencias de la Computación.

Walend, D. (2006). Understanding service oriented architecture. Developer Network. Copyright O´Reilly Media. En: http://dev.aol.com/understanding_soa.

Wang, X., Hu, S. X. K., Haq, E. y Garton, H. (2007). Integrating Legacy Systems within the Service-oriented Architecture. Power Engineering Society General Meeting-IEEE, 1-7.

Wang, Z. Y. H. (2010). An ETL Services Framework Based on Metadata. International Workshop on Intelligent Systems and Applications (ISA), 2nd, 1-4.

Wen, R., Ma, Y. y Chen, X. (2009). ESB Infrastructure´s Autonomous Mechanism of SOA. International Symposium on Intelligent Ubiquitous Computing and Education, 13-17.

W3C, (2010). W3C Web of Services. [en línea]. Disponible en: http://www.w3.org.

Zahavi, R. (2000). *Enterprise Application Integration with CORBA*. New York: John Wiley & Sons.