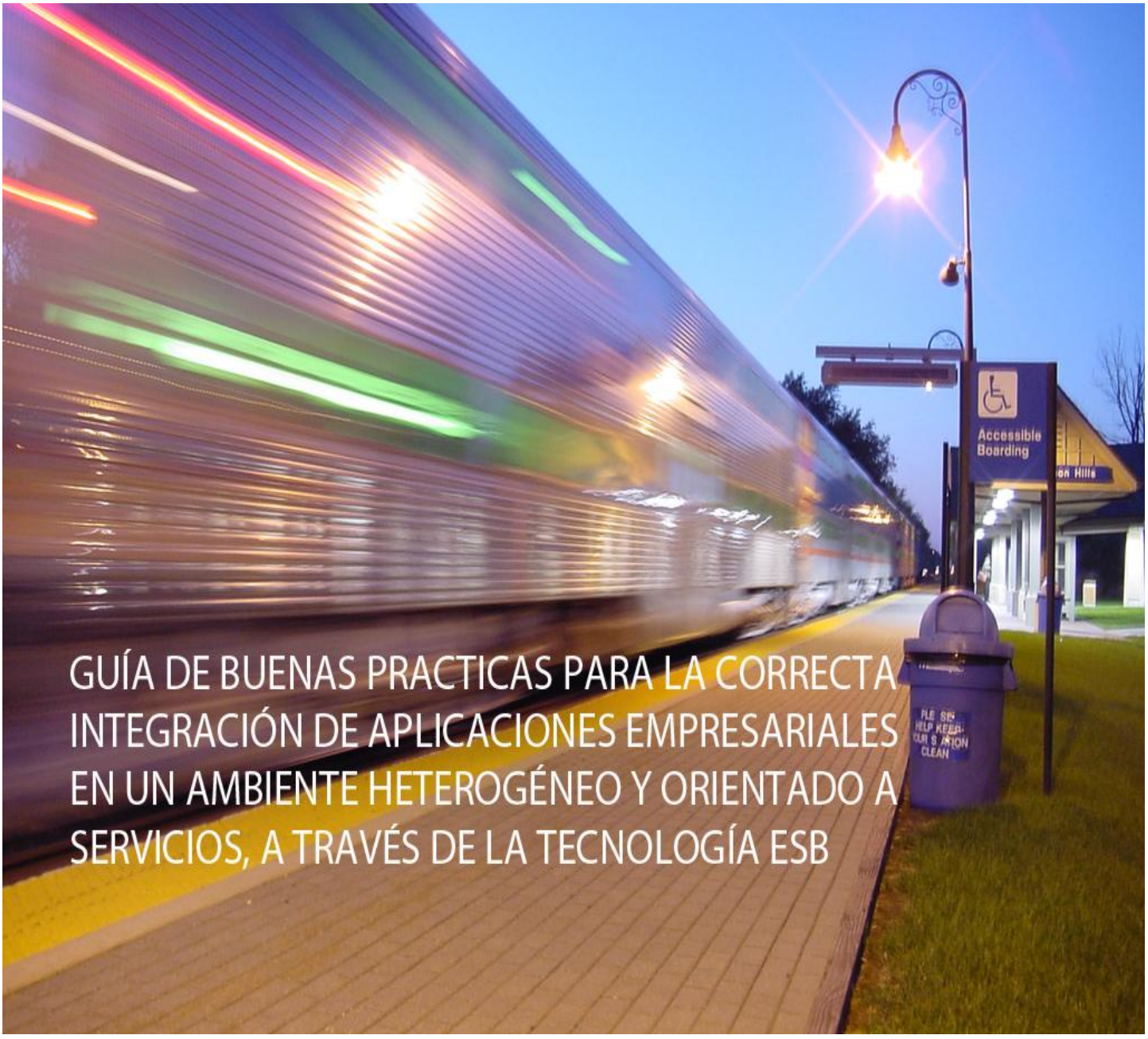


# GUÍA DE BUENAS PRACTICAS PARA LA CORRECTA INTEGRACIÓN DE APLICACIONES EMPRESARIALES EN UN AMBIENTE HETEROGÉNEO Y ORIENTADO A SERVICIOS, A TRAVÉS DE LA TECNOLOGÍA ESB

Universidad Tecnológica de Bolívar | Henry de la Hoz, Eduar Ramos

Imagen de Cubierta: Chuck Reynolds, Chicago Metra Speed
Waiting at a Metra train station in Chicago, Il. USA. Chuck Reynolds © 2003 By Chuck Reynolds Imagen tomada del sitio stock exchange:
http://www.sxc.hu/photo/10544

# GUÍA DE BUENAS PRACTICAS PARA LA CORRECTA INTEGRACIÓN DE APLICACIONES EMPRESARIALES EN UN AMBIENTE HETEROGÉNEO Y ORIENTADO A SERVICIOS, A TRAVÉS DE LA TECNOLOGÍA ESB

HENRY CHRISTOPHER DE LA HOZ ESCORCIA
EDUAR RAMOS BARRAGÁN

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
CARTAGENA DE INDIAS
2012

# GUÍA DE BUENAS PRACTICAS PARA LA CORRECTA INTEGRACIÓN DE APLICACIONES EMPRESARIALES EN UN AMBIENTE HETEROGÉNEO Y ORIENTADO A SERVICIOS, A TRAVÉS DE LA TECNOLOGÍA ESB

HENRY CHRISTOPHER DE LA HOZ ESCORCIA
EDUAR RAMOS BARRAGÁN

**Monografía Para Optar Al Título De Ingeniero de Sistemas**

**EDWIN ALEXANDER PUERTA DEL CASTILLO**
**Magister en Ingeniería**
**Director**

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
CARTAGENA DE INDIAS
2012

**CONTENIDO**

## ÍNDICE DE ILUSTRACIONES

# 1 INTRODUCCIÓN

## 1.1 Área Problémica

Las tecnologías de la información se enfrentan en la actualidad al auge de nuevos conceptos en lo concerniente a la arquitectura e ingeniería del *software*, el más notable e importante de estos es la computación distribuida que trasladó el *software* de una máquina aislada a un grupo de máquinas (no necesariamente computadoras) conectadas mediante una red. Dentro de dicha interconexión se ofrecen servicios (procesos) que son compartidos entre consumidores ampliando el espectro de posibilidades y eliminando algunos límites propios de las antiguas aplicaciones. Sin embargo, se podría decir que los desarrollos y la forma en que se piensan las aplicaciones actuales siguen estando plagadas de antiguas concepciones, es decir, percibidas como sistemas aislados. A esta situación se le puede adicionar el *boom* de la red de redes: el internet, para la cual es necesario también pensar aplicaciones y formas de diseñar e implementar *software*.

Estos nuevos conceptos traen consigo nuevos requerimientos y antiguos requerimientos reformulados, por esto, se hará indispensable que una aplicación desarrollada en el ambiente actual posea las siguientes características: distribución de recursos, desacoplamiento, escalabilidad, interoperabilidad, respuesta a cambios, seguridad; en cuanto al sistema en conjunto; y en cuanto a sus partes: atomicidad, granularidad, reusabilidad.

Inicialmente se satisfacen algunos de estos requerimientos parcialmente, con soluciones de gran aplicabilidad y de mucho auge en el mercado como son: la arquitectura cliente servidor, las aplicaciones web, los *sockets*, los objetos distribuidos (basados en la orientación a objetos, de gran éxito en cuanto a

aplicaciones de escritorio, pero, cuestionable para un sistema conectado e interoperable), Corba, *Remoting*, RPC, etc.

El producto de estas respuestas parciales y del conjunto de ensayos realizados en busca de una solución han sido nuevos enfoques que rescatan lo útil de estas tecnologías y proponen una nueva forma de concebir el software. SOA o Arquitectura Orientada a Servicios responde a la mayoría de los requisitos necesarios para las nuevas aplicaciones, enfocándose principalmente en la respuesta a cambios que se ha vuelto indispensable para las empresas, cuyos procesos de negocio evolucionan con rapidez. Esta propuesta de investigación está enmarcada dentro de la Arquitectura Orientada a Servicios, enfatizando en un constructo de la arquitectura de *software* llamado ESB (*Enterprise Service Bus*) y que se ha considerado por la industria y la academia como una herramienta muy útil para realizar una implementación SOA en un ambiente de aplicaciones heterogéneo.

## 1.2 Propósito

La implementación de SOA es un proceso complejo, que posee todos los obstáculos de cualquier implementación tecnológica, pero, su mayor problema está en que plantea un cambio estructural (arquitectónico), es decir, que no solamente se trata de una migración de plataforma tecnológica, sino, que propone remover lo construido y volver hacerlo bajo un nuevo paradigma, guiándose por los principios de diseño que esta postula, para que, en el futuro los nuevos desarrollos puedan usar los recursos funcionales actualmente construidos y cualquier cambio sea de fácil implementación. Sin embargo, la anterior visión es un ideal que en la práctica resulta utópico y de un riesgo económico y funcional muy alto, por tanto, un sector de la ingeniería de software ha apostado por tecnologías de integración que permitan que la implementación de SOA sea

gradual, de esta forma, las aplicaciones construidas bajo paradigmas previos minimizan su choque con las nuevas aplicaciones. Es así como las TI de una organización pueden tomar el camino de SOA, sin sacrificar las aplicaciones que actualmente funcionan y que posiblemente no han cumplido su ciclo de vida. El ESB ha sido una de las mejores opciones en cuanto a estrategias de integración, porque, permite brindar las funcionalidades de cualquier aplicación como servicios web(Los principales productos en el mercado tanto *open source* como propietarios hacen esto).

Este proyecto de investigación tiene como propósito fundamental hacer una exploración del ESB y de su relación con SOA. Exploración que implica acercarse de forma teórica a los elementos del ESB, acercarse de manera teórica a SOA y encontrar la compatibilidad entre los dos elementos, lo cual, es un propósito por sí mismo. Sin embargo, lo anterior tiene implicaciones prácticas; puesto que nos señala hasta qué puntos el ESB colabora con la implementación de la arquitectura SOA y en qué puntos no, pero, se requiere corroborar esta compatibilidad en la práctica, lo cual se convierte en otro propósito.

Pero esta investigación no solo tiene un interés académico, sino, que busca realizar lo que se conoce como trasferencia tecnológica, es decir, realizar un acercamiento a una tecnología que ha sido desarrollada e investigada en otros países, con el fin de familiarizarse con esta y facilitar su acceso a ambientes académicos, empresariales e industriales donde puede ofrecer beneficios.

## 1.3 Importancia

Sin querer adentrarnos en las condiciones locales, se debe reconocer que el desarrollo de *software* en Colombia posee muchos problemas relacionados principalmente con la poca divulgación(o quizás poca implementación) que existe

de los logros de la ingeniería de software. SOA es uno de estos avances y podría ver frenado su implementación, sino, se realizan esfuerzos por aclarar lo que significa y como debe ser implementado.

Esta investigación resulta importante, puesto que, intenta apoyar en este aspecto, puntualmente facilitar la implementación de SOA mediante ESB, que es –desde un punto de vista hipotético que se comprueba mediante este trabajo- la estrategia que menos impacto presenta a nivel funcional, así como también esclarecer la gran cantidad de información relacionada con el tema y que puede presentarse como un obstáculo. A su vez, se realiza un aporte teórico necesario para SOA, puesto que, por tratarse de un paradigma arquitectónico se corre el riesgo de que los encargados de los proyectos de implementación solo tengan en cuenta los aspectos técnicos, lo cual es muy común en estos procesos de asimilación tecnológica. Ejemplos de este problema se presentan en muchos proyectos de software donde se utilizan lenguajes orientados a objetos como Java o C# con el fin de tener en cuenta este paradigma, pero, no se tienen en cuenta aspectos que no son controlados por las herramientas de desarrollo, por lo cual, el producto final no posee diferencias de un proyecto construido con el paradigma procedimental. En el caso de SOA, el riesgo es aún mayor, porque las “promesas” son bastante ambiciosas y muchas de estas dependen más del seguimiento de los principios que propiamente de la restricción técnica de un producto u otro, el peor de los escenarios es que solamente se implementen servicios web como aplicaciones que procesan datos y los retornan mediante la web, manteniendo esquemas de arquitecturas tradicionales, que no representan ninguna ventaja significativa.

# 2 DEFINICIÓN DEL PROBLEMA

## 2.1 Descripción del Problema

De la mano de SOA, han surgido y evolucionado tecnologías y desarrollos que proponen cambios, los cuales de una u otra forma buscan contribuir a la migración de sistemas empresariales centralizados y estáticos a un sistema empresarial basado en la arquitectura en cuestión. Un sistema que sea flexible, distribuido y ofrezca mayor competitividad mediante el apalancamiento de los procesos de negocios valiéndose de la tecnología.

Sin embargo, no es tarea fácil la implementación de SOA, en especial si se trata de empresas que poseen una infraestructura (*software* y *hardware*) implementada y funcional, puesto que, en este caso cualquier intento de reestructurar muy probablemente sería de por si un obstáculo que traería más problemas de los que pretende solucionar. Se hace fundamental un elemento integrador que permita conservar antiguas implementaciones y que además facilite la posibilidad de futuras modificaciones de cualquiera de los elementos, garantizando de esta forma que cualquier desarrollo sea posible. Hablamos, de una herramienta de *software* conocida como ESB, la cual minimiza estos problemas, separando las necesidades del negocio de la forma en cómo están implementados y realizando lo que puede considerarse una traducción de cualquier tipo de tecnología a una orientación a servicios.

Como es de esperarse cuando se trata de tecnologías emergentes, que están en proceso de consolidación y conjugación de ideas variadas, aparecen muchas preguntas para intentar comprender: ¿qué alcance se le busca dar al ESB (entiéndase alcance como todo el conjunto de responsabilidades que puede recibir el Bus de Servicio Empresarial)? ¿De qué maneras aporta en el proceso de diseño

e implementación de sistemas SOA? ¿Cuál es el futuro o porvenir del ESB, a medida que se establece y esclarece el tema de las arquitecturas orientadas a servicios? y en un medio práctico ¿Qué sugerencias o recomendaciones puede seguir la empresa que desee valerse del ESB como herramienta de apoyo para apropiarse de SOA como su tecnología base para su desarrollo empresarial? ¿Cómo puede el ESB integrar aplicaciones basadas en SOA con aplicaciones heredadas? Estas preguntas nos ubican en el ámbito de nuestro problema, el cual se resumen en los siguientes aspectos:

- Aclaración del concepto de ESB.
- Relación ESB con SOA.
- Integración de aplicaciones heterogéneas, mediante ESB.
- Recomendaciones o mejores prácticas, para los encargados de la implementación de SOA.

Estos problemas guiaran esta investigación y serán resueltos principalmente mediante un proceso de revisión bibliográfica, que luego, será sometido a un análisis, con el cual se planteara un caso de estudio que evidencie la posibilidad de integración. Por último, los datos recogidos permitirán la construcción de un conjunto de mejores prácticas.

# 3 JUSTIFICACIÓN

Con el objetivo de aumentar su competitividad, empresas de diversos sectores buscan implementar sistemas basados en una arquitectura orientada a servicios, sin embargo, esta no es tarea fácil, puesto que ante una implementación SOA se presentan muchos obstáculos y retos. El primero de estos es la ausencia de un conocimiento apropiado de la tecnología, le siguen la muy poca referencia a implementaciones anteriores, la gran cantidad de productos en el mercado (que antes de brindar variedad y soluciones reales ofrecen productos de los que se desconoce su verdadera utilidad) y por último la necesidad de realizar implementaciones frente a aplicaciones con otro tipo de arquitectura. Entre estos retos sobresale el ultimo, porque se convierte en el más difícil de solventar debido a sus implicaciones.

Este último punto proviene de entender que SOA debe ser implementado gradualmente, puesto que, pretender cambiar toda la infraestructura tecnológica (IT, sistemas *legacy*, etc.) de una empresa puede simplemente terminar en fracaso, debido en gran parte a que existen aplicaciones que se encuentran en ejecución y algunas con procesos críticos. Ante esto se ha propuesto el ESB como tecnología, puesto que, permite que se acople cualquier tipo de tecnología con una Arquitectura Orientada a Servicios.

ESB pretende superar estos obstáculos y permitir que la implementación de SOA sea posible; sin embargo, antes de esto es necesario que se verifique la aplicación real y el verdadero aporte del ESB a la hora de su implementación, es decir, se hace necesario constatar como el Bus de Servicios puede funcionar como integrador que permita la adopción de esta arquitectura en una organización. Este puede considerarse el primer paso, y posteriormente se deben plantear las siguientes preguntas: ¿qué debería incluir (funcionalidades)?, ¿por qué debería

tomarse como una alternativa importante en la implementación de SOA?, ¿se puede considerar al ESB como eje fundamental de SOA (como muchos autores lo consideran)?

Resolver o dar luz en las cuestiones anteriores son motivos que impulsan a la realización de una investigación que aporte a la comunidad y al sector académico e industrial información referente. Por lo cual, se plantea una guía práctica con fundamentos teóricos sólidos, bien definidos y conceptualizados; contrastado mediante un informe de pruebas y prácticas sobre el ESB y ambientes SOA heterogéneos.

# 4 OBJETIVOS

## 4.1 Objetivo General

Creación de una guía de buenas prácticas que aporte conocimiento al sector académico y empresarial para la correcta integración de aplicaciones en un ambiente heterogéneo y orientado a servicios a través de ESB, mediante revisión bibliográfica y la implementación de un caso de estudio relacionado con dicha tecnología.

## 4.2 Objetivos Específicos

- Conocer y comprender la arquitectura y responsabilidades del ESB, mediante revisión conceptual, para plantear una síntesis de fácil acceso para el ámbito local que puntualice los aspectos importantes del ESB.

- Analizar la relación entre ESB con un enfoque SOA estricto, mediante una aproximación teórica entre ambos elementos y posterior contraste entre ambos enfoques, mostrando hasta qué punto el ESB colabora con la implementación de SOA.

- Aplicar un caso de estudio en el que se brinde como servicio los datos de una aplicación heredada (Informix 7.3 en Unix SCO), mediante el uso de Open ESB 2.2, para evidenciar su capacidad integradora en ambientes heterogéneos.

- Crear una guía de buenas prácticas, mediante el análisis de la información obtenida en las actividades realizadas, aplicándola a necesidades reales del desarrollo empresarial.

# 5 REVISION LITERARIA

## 5.1 Marco Teórico

Los principios teóricos sobre los cuales se encuentra fundamentado el trabajo de investigación, son principalmente conceptos relacionados con la computación distribuida, computación orientada a servicios, herramientas de integración.

La arquitectura SOA es una respuesta directa a las necesidades de las áreas de negocios que desean disponer de más flexibilidad en los sistemas de la empresa para poder modificar más rápidamente los procesos como consecuencia de peligros u oportunidades que surgen en el entorno.

SOA está basado en tecnologías desarrolladas inicialmente para realizar operaciones B2B (Negocio a Negocio) pero que se han revelado muy útiles no solo para interactuar con aplicaciones externas sino también como entes que aportan a la interacción de elementos de un mismo sistema empresarial.

En una Arquitectura Orientada a Servicios en lugar de crear aplicaciones complejas y aisladas, se desarrollan componentes reutilizables que son fáciles de mantener y probar, las aplicaciones se crean diseñando un proceso que enlaza esos componentes. Cada nueva aplicación reutiliza los componentes existentes y solo se desarrollan los que aún no existen. Estos componentes se conocen como servicios.

La forma básica de dicha arquitectura, está constituida primordialmente por un consumidor de un servicio, el cual es ofrecido por un proveedor, luego entonces para tener acceso a un contrato que relacione el consumidor con el proveedor, el

primero busca dentro de un registro de servicios, que le indica los términos del contrato de uso del servicio (ver Ilustración 1).

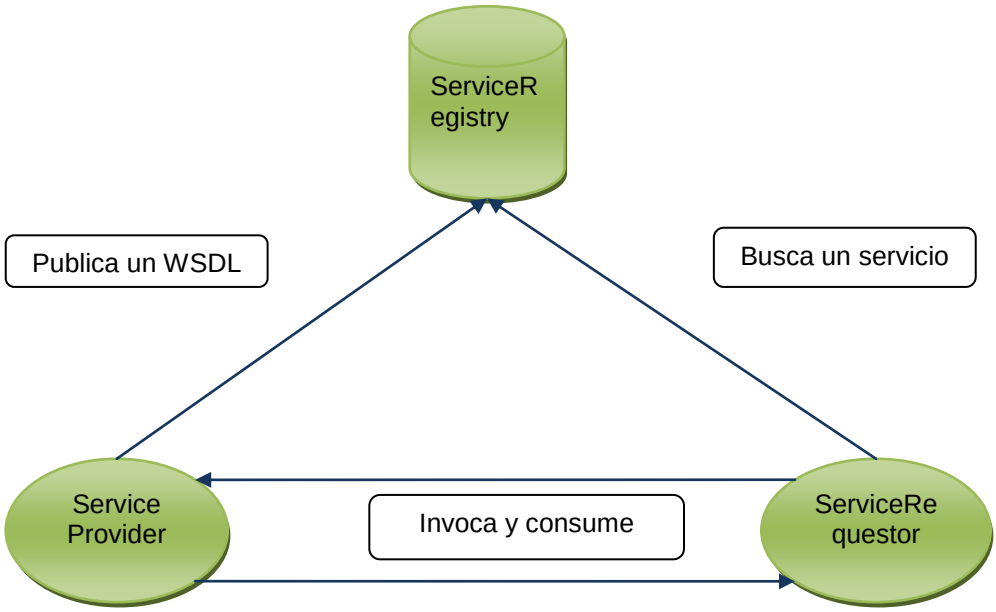


**Ilustración 1 – Arquitectura Básica SOA**

Según [1], la estrategia de orientación a servicios permite la creación de servicios y aplicaciones compuestas que pueden existir con independencia de las tecnologías subyacentes. En lugar de exigir que todos los datos y lógica de negocio residan en un mismo ordenador, el modelo de servicios facilita el acceso y consumo de los recursos de IT a través de la red. Puesto que los servicios están diseñados para ser independientes, autónomos y para interconectarse adecuadamente, pueden combinarse y recombinarse con suma facilidad en aplicaciones complejas que respondan a las necesidades de cada momento en el seno de una organización. Las aplicaciones compuestas (también llamadas “dinámicas”) son lo que permite a las empresas mejorar y automatizar sus procesos manuales, disponer de una visión consistente de sus clientes y socios

comerciales y orquestar sus procesos para que cumplan con las regulaciones legales y políticas internas.

El resultado final es que las organizaciones que adoptan la orientación a servicios pueden crear y reutilizar servicios y aplicaciones y adaptarlos ante los cambios evolutivos que se producen dentro y fuera de ellas, y con ello adquirir la agilidad necesaria para ganar ventaja competitiva. [1]

Un servicio representa una función de negocios claramente definida que puede ser invocada remotamente mediante protocolos de comunicación estándar, los servicios se definen mediante interfaces explícitos que son totalmente independientes de la implementación del servicio, los servicios deben poder ser invocados utilizando protocolos de comunicación estándar que enfatizan la interoperabilidad e independencia de ubicación.

Los servicios se pueden desarrollar en diversos lenguajes de programación (Java, C/ C++, C#, Cobol, PERL, etc.). Esto es muy práctico para modernizar sistemas legados, sin embargo, para servicios nuevos es aconsejable utilizar un lenguaje que permita instalar el servicio fácilmente en un *cluster* de servidores para ofrecer escalabilidad y alta disponibilidad, para desarrollar componentes, al igual que cualquier otro sistema es bueno adoptar una metodología de desarrollo aceptada.
Gracias a la aparición de los servicios; se dio nacimiento a lo que se conoce como SaaS (*Software as a Service*), que puede definirse como “*software* que se pone en explotación en la modalidad de servicio gestionado y que al cual se accede a través de Internet”. El concepto de SaaS suele asociarse con los proveedores de servicios de aplicación (ASP) de los años 90, que ofrecían aplicaciones “empaquetadas” a los usuarios corporativos a través de Internet.

Estos primeros intentos de poner en marcha soluciones de *Software* a través de Internet tenían más en común con las aplicaciones corporativas tradicionales (las que se instalan y utilizan dentro de la red interna de las empresas) que con las actuales aplicaciones SaaS en muchos aspectos, tales como el modelo de licencia y la arquitectura. Puesto que esas aplicaciones se crearon en principio como aplicaciones para un solo destinatario, su capacidad para compartir datos y procesos con otras aplicaciones estaba muy limitada y tendían a ser escasamente atractivas en comparación con sus equivalentes de instalación en local. Hoy día las aplicaciones SaaS pretenden aprovechar las ventajas de la centralización a partir de una arquitectura de instancia única con múltiples usuarios y ofrecer una experiencia con funcionalidades avanzadas que compitan con ventaja frente a las aplicaciones instaladas localmente. Una aplicación SaaS normalmente la ofrece un proveedor de forma directa o un intermediario (llamado “agregador”) que empaqueta ofertas SaaS de distintos proveedores y las ofrece como una plataforma unificada de aplicaciones o una suite de servicios de aplicación. A diferencia del modelo de licencias habitual del *software* que se instala en las empresas, el acceso a las aplicaciones SaaS se suele basar en un modelo de suscripción, donde los clientes pagan una tarifa por adelantado para utilizarlas. Las estructuras de precios varían de unas aplicaciones a otras: algunos proveedores aplican una tarifa plana con acceso ilimitado a diversas funcionalidades de las aplicaciones, y otros aplican tramos tarifarios que dependen del nivel de utilización.

SaaS además se posiciona como uno de los pilares del desarrollo de la orientación a servicios. A los efectos de este documento, nos vamos a referir de forma general a SOA, incluyendo en este concepto tanto los servicios implantados en local como los alojados en Internet. Consideramos que SaaS es un componente fundamental en cualquier estrategia SOA de un cliente. Debido a la tendencia de los sistemas distribuidos y desplegados a ser heterogéneos

(aplicaciones variables, tipos de servicios variables, sistemas operativos variables, protocolos de comunicación, etc.) se hace necesario una tecnología estándar, la cual sea transversal a todas las capas que interactúan en los sistemas orientados a servicios, un recurso del cual se puede valer la ingeniería de *software* para lograr la meta previamente planteada es XML.

XML es un lenguaje que tiene como fin describir información y datos de manera sistemática y estructurada. Su función principal es ayudarnos a organizar contenidos haciendo que los documentos XML sean portables entre diferentes tipos de aplicaciones, de tal manera que puedan compartirse mensajes en ambientes diversos, a través de documentos de este tipo, se separa la información de la presentación. Permitiendo a los desarrolladores crear sus propias etiquetas, que les posibilita habilitar definiciones, transmisiones, validaciones, e interpretación de los datos entre aplicaciones y entre organizaciones.
Un sencillo ejemplo de documento **XML** sería:

***<persona>***
***<nombre>****Henry****</nombre>***
***<apellidos>****de la Hoz Escorcia****</apellidos>***
***<empresa>****Akendos Agile Software* ***</empresa>***
***<teléfono>****66 76 441****</teléfono>***
***</persona>***

Principalmente XML posee una utilidad clara y es su utilización como lenguaje común para el intercambio de información entre aplicaciones, servicios, sistemas, etc. a través de Internet o cualquier tipo de protocolo de comunicación, dicho intercambio es con total y completa independencia de la plataforma, sistema operativo o lenguaje de programación en que han sido desarrolladas las aplicaciones que procesan el documento.

Este tipo de relación de interoperabilidad resulta muy valiosa en el medio comercial entre empresas y clientes (*e-commerce*), por lo cual, la información se maneja en un lenguaje estándar, de entendimiento común y la presentación es relativa o subjetiva a cada usuario final.

Un documento XML está dividido en dos partes, el prólogo, el cual contiene información acerca del documento, y el documento o elemento raíz, dentro del cual debe estar todo el contenido del documento.

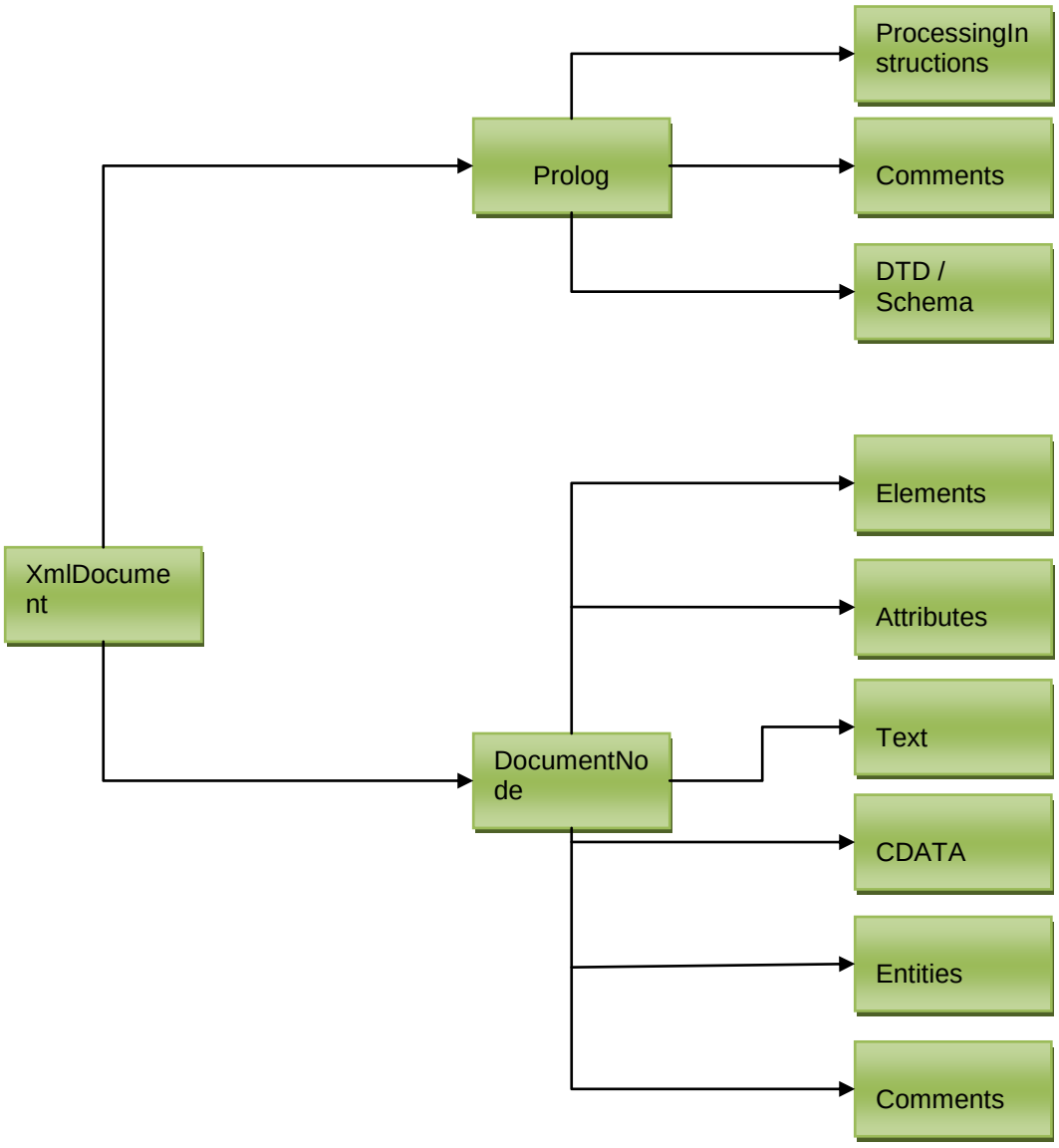


**Ilustración 2 - Estructura XML [10]**

El prólogo, es aquel que contiene meta información concerniente al documento que va más allá del contenido del mismo. Este es opcional, se ubica en la parte

superior del documento y puede contener la declaración XML, así como, instrucciones de procesamiento, comentarios y DTD s y esquemas embebidos.

Las instrucciones de procesamiento (IP s), permiten a los documentos incluir instrucciones para las aplicaciones. Estas comienzan con los caracteres '<?' y terminan con '?>'. También se encuentran: Los comentarios los cuales son pedazos de información para uso humano exclusivamente, por lo cual no afectan el procesamiento de un documento XML, estos comienzan con los caracteres '<!--' y termina con '-->'. Por último, se encuentran Los DTDS y esquemas XML, proveen reglas de cuales elementos y atributos pueden aparecer en el documento, es decir, definen que elementos y atributos son válidos, requeridos y opcionales. El prólogo puede contener referencias propias del documento, o, referenciar DTD s o esquemas externos, o ambos.

Dentro del elemento raíz, conocido como Documento se encuentra toda la información pertinente, dentro del elemento Documento se encuentran todos los otros elementos, atributos, textos, secciones CDATA, además de poder contener entidades y comentarios.

XML tiene dentro de ambientes SOA por lo menos tres utilidades puntuales y de suma importancia, se hace referencia a:

- *Simple Object Access Protocol* (SOAP), es un protocolo ligero destinado al intercambio de información estructurado en un entorno descentralizado, distribuido. Utiliza tecnologías XML para definir un *framework* extensible de mensajería proveyendo un constructor de mensajes que pueden ser intercambiados a través de una variedad de protocolos subyacentes, como http, SMTP, entre otros. El *framework* ha sido diseñado para ser

independiente de cualquier modelo particular de programación y otras implementaciones de semánticas.

Los principales objetivos de diseño de SOAP son la simplicidad y extensibilidad, para alcanzar estos objetivo la especificación omite de su arquitectura aquellos aspectos que se encuentran a menudo en sistemas distribuidos. Estas características incluyen, pero no se limitan a fiabilidad, seguridad, correlación, enrutamiento y patrones de intercambio de mensajes (MEPs). Si bien se prevé que muchas de estas características se definirán, la especificación proporciona detalles sólo para dos MEPS. Otras características pueden ser agregadas como extensiones de la especificación.

- *Web Services Description Language* (WSDL), es un lenguaje especificado en XML que se ocupa de describir los requisitos funcionales necesarios para establecer una comunicación con los servicios Web, para lo cual define un modelo que separa las funcionalidades ofrecidas por un servicio, de los detalles concretos, de cómo y dónde esta funcionalidad es ofrecida. Las definiciones del servicio WSDL proveen documentación destinada a los sistemas distribuidos y hace las veces de especificación para automatizar los detalles involucrados en las aplicaciones de comunicación.

- *Universal Description, Discovery, and Integration* (UDDI), provee un método estandarizado para la publicación y el descubrimiento de información sobre los Servicio Web. Esta iniciativa parte de la industria, que pretendía crear una plataforma independiente, un marco de trabajo abierto para la descripción de servicios, descubrimiento de organizaciones y la integración

de servicios de negocio. UDDI se centra en el proceso del descubrimiento dentro de la arquitectura orientada a servicios.

Sin embargo, la adopción de una arquitectura basada en servicios requiere de una infraestructura de comunicaciones escalable y segura entre los componentes. Esto es lo que se conoce como Bus de Servicio Empresarial.

El Bus de Servicios Empresariales es el concepto que se refiere a la infraestructura de transporte de mensajes entre el motor de procesos y los servicios de los que dispone la empresa. El bus requiere una infraestructura sólida que ofrezca a los desarrolladores la seguridad de que los mensajes sean entregados siempre, independientemente de los problemas que puedan afectar a un servicio determinado en un momento dado. El bus debe ofrecer seguridad total y conectividad hacia todo el *software* de infraestructura de la empresa, el bus debe ser totalmente monitoreable porque se convierte en la columna vertebral de todos los sistemas de la empresa [2].

Es un paradigma computacional que utiliza servicios como los elementos fundamentales para el desarrollo rápido y de bajo costo, de aplicaciones distribuidas en ambientes heterogéneos [3].

Es un paradigma que se refiere a un conjunto de conceptos, principios y métodos en los cuales el *software* es construido basándose en la composición de servicios con interfaces estándares [4].

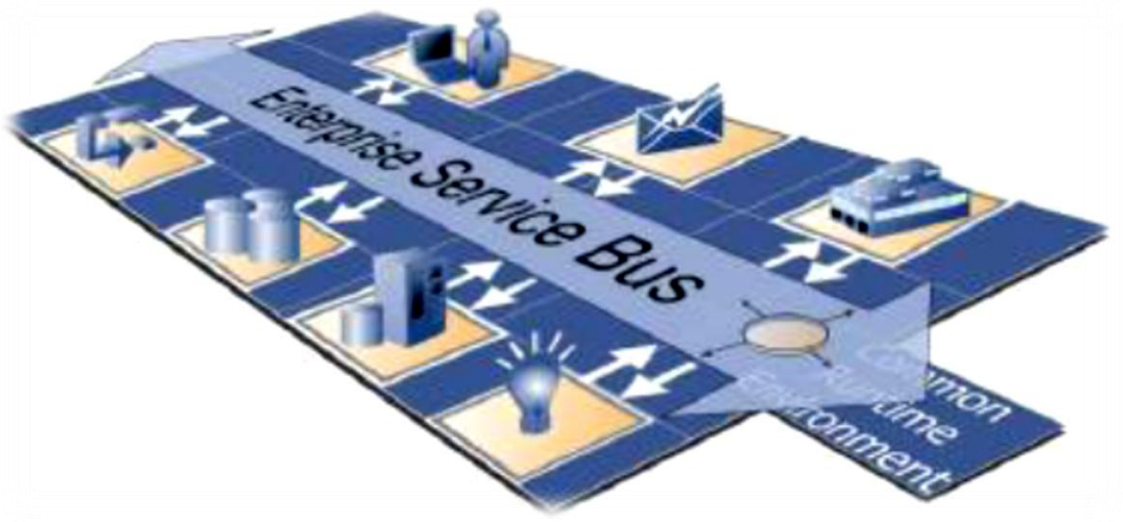


**Ilustración 3 - ESB [5]**

El ESB tiene como característica principal, servir como mediador. Dentro de sus principales generalidades están:

***Routing***

El Bus de Servicio debe ofrecer direccionamiento, de tal manera que dado un mensaje solicitud de búsqueda de servicio, este debe, a través de métodos de enrutamiento, ubicar en el registro de servicios, el repositorio contenedor del servicio deseado [6].

A pesar de que el objetivo general de *Routing* en un ESB, es el planteado anteriormente, hay diferentes formas de llevarlo a cabo. Por ejemplo: se puede acudir a métodos de *Routing* estáticos o determinísticos; también es posible que se realice enrutamiento basado en contenido, enrutamiento basado en políticas o enrutamiento basado en reglas de servicio.

**Transformación de mensajes**

Debido a la naturaleza heterogénea de los sistemas, es apenas razonable que en muchas ocasiones la solicitud de servicio de negocio, no tenga la estructura esperada por el proveedor de servicio. En estas circunstancias, el Bus de Servicio Empresarial convierte la estructura del mensaje solicitud a una estructura aceptable para el proveedor. Por ejemplo: conversión de XML a Cobol Copybook, conversión de objetos a XML, etc.

**Transformación de protocolos**

Los consumidores pueden realizar solicitudes de servicios a través del ESB utilizando cualquier tipo de protocolo; bien sea HTTP, SOAP (JMS), SOAP (MQ), etc. al llegar al proceso de comunicación con el proveedor, el Bus debe en este momento adaptarse al protocolo requerido por el contenedor del servicio. Es decir, que si un servicio ha sido expuesto a través de un proveedor de tipo JMS, el ESB debe estar apto dar acceso al servicio ofrecido, sin importar la vía de invocación que el cliente o consumidor utilice.

Es notable de igual manera que las anteriores serán las 3 primordiales funcionalidades que potencian a dicha tecnología de integración, pero las características de la misma, pueden aumentar dependiendo de las necesidades empresariales del momento.

Por su parte la computación orientada a servicios representa una plataforma de computación distribuida de última generación. Esta encapsula diferentes elementos y conceptos, incluyendo sus propios paradigmas de diseño, principios de diseño, catálogos de patrones de diseño, distintos modelos arquitecturales, así como conceptos, tecnologías y *frameworks* relacionados. La computación orientada a servicios está construida sobre plataformas de computo distribuidos

anteriores y le agrega nuevas capas de diseño, consideraciones de gobernación, y un vasto conjunto de tecnologías predilectas para su implementación. [4]

La computación orientada a Servicios, como paradigma para la construcción de *software* involucra diferentes disciplinas como:

- Sistemas distribuidos
- Arquitecturas de *software*
- *Grid Computing*
- Ingeniería de *Software*
- Lenguajes de programación
- Sistemas Multiagentes
- Sistemas de bases de datos
- Seguridad
- Calidad de Servicio
- Sistemas basados en conocimiento

Los paradigmas creados por la Computación Orientada a Servicios (SOC) y específicamente el estilo arquitectónico propuesto por una Arquitectura Orientado a Servicios (SOA), en el proceso de construcción de *software*, se ha integrado exitosamente al actual modelo empresarial, permitiendo al área de IT (tecnologías de la información), *crear software altamente reusable, y con una buena repuesta a los cambios*, como lo requieren los procesos de negocio hoy en día. Colombia y

puntualmente Cartagena no han sido ajenas a estos hechos, debido a lo cual un creciente número de empresa ha mostrado interés en estas temáticas.

La computación orientada a servicios, define un servicio como una entidad autónoma e independiente de plataforma, que puede ser descrita, publicada y descubierta, y cuyo acoplamiento es bajo, sobre la base de este concepto SOC, propone un paradigma en el cual se hagan desarrollos rápidos de aplicaciones distribuidas, de bajo costo, interoperables, encapsulables y masivas.

Existe cada vez más un número creciente de empresas que buscan implementar dentro de su sistema interno y a través de su departamento de IT una arquitectura que apunte al mayor grado posible a la Arquitectura Orientada a Servicios.

# 6 DISEÑO METODOLÓGICO

## 6.1 Metodología

Como lo señala [17](pág. 9) el diseño puede ser investigado; lo que significa que así como algunas ciencias obtienen conocimiento basándose en las experiencias recogidas en la observación a la naturaleza, otras pueden construir conocimiento basándose en la experiencia recogidas en la creación de artefactos (objetos productos de la creatividad humana).

Esto se debe principalmente a que ciencias como la ingeniería de software construye su conocimiento en la formalización de diseños que han respondido con éxito a un conjunto de necesidades en distintos ambientes; por lo que pueden ser reutilizados por otras personas con problemas similares, convirtiéndose de esa forma en conocimiento. Sin embargo, la construcción de este conocimiento no se limita a la formalización de patrones, sino, que mediante su aplicación e implementación reiterativa se producen mejoras a los patrones formalizados, así como también cambios profundos en estos. De esta forma el conocimiento en las ciencias del diseño nos muestra un ciclo de mejoramiento y cambio, que podemos resumir en que: el conocimiento se crea diseñando.

Esta investigación está enmarcada dentro de ese ciclo, por pertenecer al área de la ingeniería del software, pretende tomar diseños que han sido planteados, relacionarlos y evaluarlos con el fin de crear un conocimiento que es de gran utilidad en la implementación de SOA. Las siguientes etapas en que fue propuesta la investigación son un ejemplo de lo anteriormente planteado:

- Revisión bibliográfica

- Realización de pruebas técnicas
- Selección y conceptualización
- Planteamiento del problema
- Validación practica de hallazgos
- Conclusiones y sugerencias alternativas.

A continuación se detallará cada una de estas.

### 6.1.1 Revisión Bibliográfica.

Consiste en realizar una revisión de la literatura científica con respecto al tema. Para esto se usará en su gran mayoría recursos electrónicos, es decir, buscadores especializados, bases de datos especializadas, video conferencias e información técnica en los sitios web de las empresas y comunidades de software relacionadas con la temática de investigación, sin embargo, también se usaran los recursos físicos de las bibliotecas locales. Los principales temas de los que se busca información son:

- SOA
- Tecnologías de Integración
- ESB
- XML
- Sistemas distribuidos

Este paso está fundamentado en los patrones de búsqueda de literatura llamados *Familiarization with New Area* y *Industry and Practice Awareness* planteado en [17]

### 6.1.2 Pruebas Técnicas

Esta investigación posee un componente significativo de implementación tecnológica, por lo cual se requiere de un proceso de familiarización con la tecnología que es usada en la industria. Por tanto, se requiere un reconocimiento de los distintos productos de software presentes en el mercado. Estos productos incluyen principalmente a los lenguajes, protocolos de comunicación y bases de datos que se usan actualmente en las empresas, y a los productos ESB que se ofrecen en el mercado (entiéndase mercado como productos de índole comercial y libre). Esta etapa está fundamentada en el patrón *Industry and Practice Awareness* planteado en [17](pág. 116).

### 6.1.3 Selección y conceptualización

Proceso por el cual se discriminará toda la información recogida, buscando las fuentes que mejor explican las diferentes temáticas. Posteriormente se consignarán cada uno de los conceptos extraídos de estas fuentes y que aportan a los objetivos planteados. Este proceso tienes dos objetivos fundamentales. Por un lado busca discriminar los elementos que se enfocan en la venta de un producto específico, es decir discriminar la información que tenga un carácter publicitario más que científico, por otro lado busca formar un concepto amplio acerca del ESB, SOA y las tecnologías de integración, en el que se tengan en cuenta los elementos importantes de la literatura revisada. Esta actividad está fundamentada en el patrón de investigación *Abstracting Concepts* planteado en [17](pág. 150).

### 6.1.4 Planteamiento del problema

Si bien existe un planteamiento inicial del problema, dicho planteamiento resulta difícil de tratar, por tanto, se realizó un proceso de trasformar lo planteado en un problema de diseño de forma que su cuantificación se facilite. Se recurrirá al uso de la metodología de recolección de requerimientos, de forma que el trabajo conceptual pueda ser más fácilmente comprobado en la práctica. Dicho proceso incluye la recolección de las características de SOA como requerimientos, así como también la recolección de requerimientos de un ambiente de pruebas.

### 6.1.5 Validación practica

Una vez se ha estado familiarizado con la tecnología, se requiere que se construya un prototipo que demuestre los hallazgos realizados. Básicamente este prototipo debe demostrar las funcionalidades de cualquier software puede ser publicada como un servicio web, permitiendo así la integración de servicios heterogéneos con una implementación bajo el paradigma SOA. En resumen, consiste en la construcción de un ambiente de prueba. Esta etapa es básicamente una *demostración* como se muestra en [17] en la que se prueba bajo una situación predefinida que se pueden realizar un conjunto de cosas, en nuestro caso que se pueden integrar aplicaciones legadas con aplicaciones orientadas a servicios. También se realizará una comparación de los diferentes productos ESB con el fin de encontrar cual o cuales cumplen con las condiciones necesarias para integrar servicios heterogéneos. Ver [17].

### 6.1.6 Conclusiones y sugerencias

Todo el proceso arrojara un conjunto conclusiones que serán consignados en forma de buenas prácticas para las personas interesadas en la implementación del ESB teniendo en cuenta los principios de SOA.

## 6.2 Actividades

1. Compilar referencias bibliográficas concernientes a sistemas distribuidos, integración y primordialmente *Enterprise Service Bus* y SOA.

2. Extraer y generar un escrito en el cual se palpe información pertinente acerca del contexto de integración en cual se desenvuelve el Bus de Servicios

3. Analizar y generar un escrito en el cual se palpen las diferentes posiciones, enfoques arquitectónicos y propuestas de implementación del ESB.

4. Estudiar de qué manera puede implementarse el Bus de Servicio para apoyar un sistema basado en una Arquitectura Orientada a Servicios.

5. Implementar un ambiente de prueba donde se ponga en práctica la teoría aprendida. Y se utilicen las diferentes herramientas encontradas previamente.

6. Desarrollar un documento guía que contenga buenas prácticas y recomendaciones que contribuyan a un apropiado uso del ESB como elemento integrador en una implementación SOA en ambientes de aplicaciones heterogéneos.

7. Sustentación del trabajo realizado.

## 6.3 Hipótesis

- Es fundamental la implementación de herramientas de integración, debido a la aparición continua de aplicaciones y tecnologías que tienen gran variabilidad entre sí, en el contexto empresarial.

- Los Sistemas distribuidos y el desarrollo orientado a servicios, pueden apalancar los procesos empresariales aún más, trabajando conjuntamente con el ESB como herramienta integradora.

- El Bus de Servicios Empresarial posee la arquitectura y robustez necesaria para ofrecer una solución a la integración de ambientes de aplicaciones heterogéneas.

# 7 INTEGRACIÓN

## 7.1 Necesidad de Integración

Las grandes empresas así como las que están en proceso de evolución, tienden a presentar un crecimiento exponencial en su información, en los procesos que se hacen sobre dicha información. Lo cual genera que los sistemas monolíticos se queden cortos ante la demanda requerida para la gestión de la data contenida y toda la lógica que encierra.

Asimismo el crecimiento generado trae consigo naturalmente nuevas aplicaciones, y nuevas herramientas de desarrollo, lo que añade razones de peso para dejar a un lado los sistemas monolíticos.

Se vuelve vital por ende, un esquema que, guiado por la filosofía de distribuir las tareas, plantee una solución tecnológica viable, confiable, escalable y ante todo funcional para este problema anteriormente propuesto.

El panorama tecnológico que se presenta hoy en día, indica que aquellas empresas con mucho tiempo de existencia adquieren en su sistema (si se puede referir a uno solo) las siguientes características comunes entre compañías:

- Gran variabilidad en las aplicaciones que se utilizan (aplicaciones desarrolladas dentro de la empresa, aplicaciones desarrolladas a través de terceros y aplicaciones compradas).

- Mucha de la información que se maneja es duplicada, así como los procesos más comunes, se repiten de un programa a otro.

El hecho de que las aplicaciones hayan sido desarrolladas en instantes de tiempo diferentes, por grupos de desarrolladores diferentes, trae consigo nuevas dificultades, dentro de las cuales están:

- Diferencias en los lenguajes de programación.
- Diferencias en las metodologías de desarrollo, tales como, programación por capas cliente servidor en sus formas especiales, etc.
- Diferencias en los motores de base de datos, asimismo para esquemas de bases de datos variables (bases de datos relacionales, jerárquica, orientada a objeto, etc.)
- Diferencias entre los protocolos de comunicaciones utilizados para el envío de la información.

En conjunto, todo el entramado tecnológico del cual se habló previamente, son aquellos obstáculos con los cuales la integración debe lidiar, con la adición de que el acceso a la información y los tiempos de respuesta sean los óptimos.

Sin embargo, a pesar de que desde un punto de vista de la ingeniería de software la integración más que funcional es necesaria, cuando ubicamos el contexto de la integración en ambientes burocráticos, en ambientes ejecutivos y no en el medio del desarrollo de *software*, las circunstancias cambian notoriamente. Para los gerentes y altos puestos de las empresas, los departamentos de IT son vistos más que como generadores de ganancias, como generadores de gastos. Lo cual dificulta a los directivos entender o visualizar la verdadera necesidad de integración dentro de la empresa. Y a pesar de que el departamento IT detecte la necesidad y genere una propuesta de solución, eventualmente se quedará

el proyecto solución en una fase inconclusa debido a la falta de recursos que se le brindaron para efectuar el plan en su totalidad.

Ahora bien, para que un profesional de silla visualice la verdadera necesidad de integración, debería explicársele en términos de costo y beneficio, lo cual genera la siguiente pregunta: ¿en qué aspectos es favorable para una empresa adoptar la integración de sus aplicaciones como un medio de aumentar su competitividad así como su control sobre los procesos de negocios que lleva a cabo?

En ese orden de ideas, se puede inferir que desde un punto de vista corporativo la integración logrará una mejor administración de los movimientos que se realicen, entiéndase movimientos como aquellos procesos empresariales que varían entre sí, por ejemplo: un administrador bien pudiese interactuar con el ERP de la organización o un CRM, SRM, etc.

Pero dicha interacción puede ser más o menos compleja de acuerdo con el nivel de integración que se aplique.

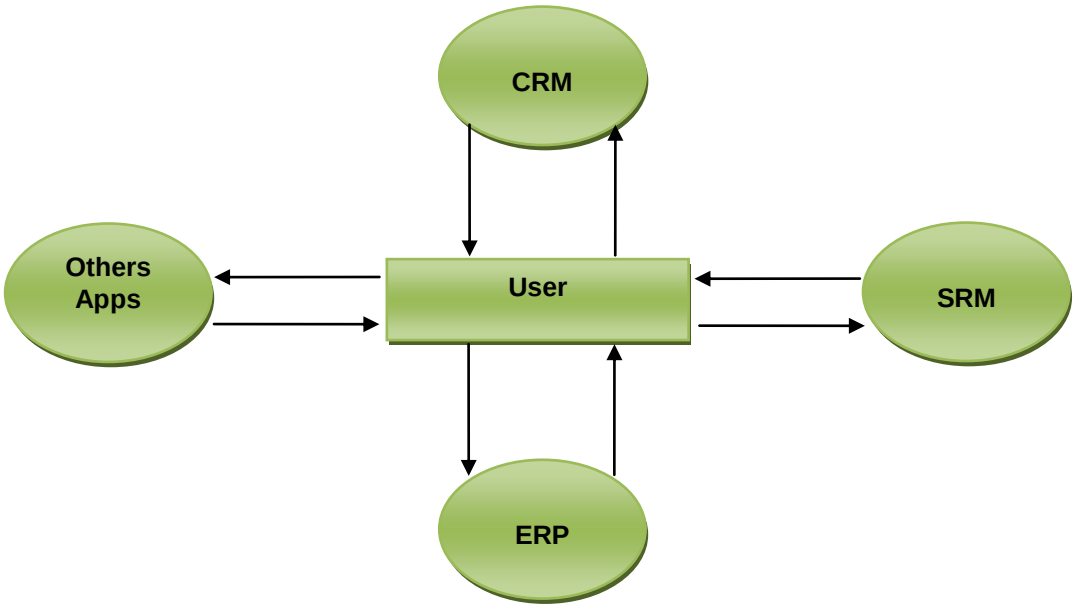


**Ilustración 4- Esquema de interacción usuario corporativo vs Aplicaciones propias empresariales sin ningún tipo de integración.**

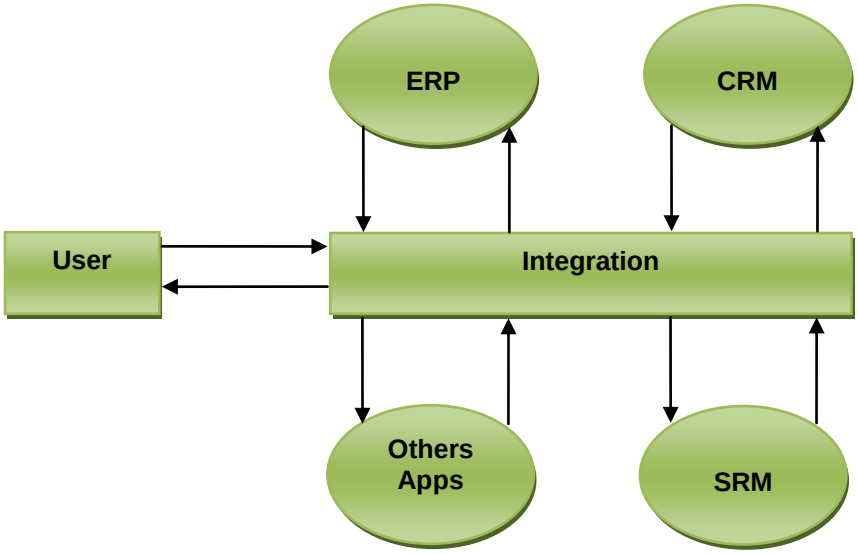


**Ilustración 5 - Esquema de interacción (mediante la utilización de integración) usuario corporativo vs Aplicaciones empresariales.**

## 7.2 Beneficios de la integración

Principalmente a través de una sólida integración empresarial, se logrará [7]:

### Mejorar la relación Cliente – Empresa

Para saber cómo mantener una buena relación con el cliente es preciso y muy recomendable entender la visualización que tiene el mismo de la compañía. Para el cliente la compañía es vista en conjunto, como una sola pieza o herramienta, la cual responde a sus necesidades y solicitudes, aunque en la realidad la empresa funciona como una división de departamentos cada uno de ellos encargado con una función específica.

Muchas veces cuando un cliente tiene una dificultad o inquietud, es enviado de un departamento a otro y en la mayoría de los casos brinda la misma información una y otra vez, naturalmente el cliente se sentirá irritado, pues desea que su necesidad sea suplida prontamente. Asimismo el cliente querrá que su lealtad sea valorada y apreciada. Esta situación demostraría evidentemente que los departamentos no tienen la interacción más adecuada, lo cual genera que se solicite información de manera repetitiva. Con una correcta integración de las distintas aplicaciones así como de la información que fluye dentro de la empresa, mejora el servicio y la interacción con el cliente. Con la información que los departamentos adquieren de los clientes y junto con una adecuada integración que optimice el intercambio de información, las empresas valiéndose de técnicas de BI (*Business Intelligence*) podrían generar cambios en sus procesos, en sus productos o servicios, de tal manera que aumenten considerablemente las ganancias económicas y también impulsen a la compañía a una evolución constante que le permite abrirse campos e incluso a nuevo mercados.

**Mejorar la interacción con *Supply-Chain***

Llevando los beneficios de la integración al siguiente paso, un buen aprovechamiento de la misma permitiría a las empresas una eficiente comunicación son sus proveedores y los departamentos de administración de suministros. A través de dicha comunicación la empresa podría aprovechar la filosofía de integración B2B (*Business to Business*). A modo de ejemplo se podría pensar en el departamento de inventario, el cual se encuentre integrado con el departamento de compras y adquisiciones, encargado precisamente de realizar compra de suministros y otros elementos necesarios dentro de la empresa. Valiéndose de la integración, una vez que un artículo tenga un número de existencias menor a las estipuladas dentro del stock, la aplicación de inventario envía un mensaje directo a la aplicación utilizada en el departamento de compras quien lo recibe e inmediatamente interactúa con la aplicación de los proveedores para la recepción de pedidos. Todo esto sin la intervención humana. Una interacción entre aplicaciones a nivel de procesos de negocio (a través de estándares como XML y EDI).

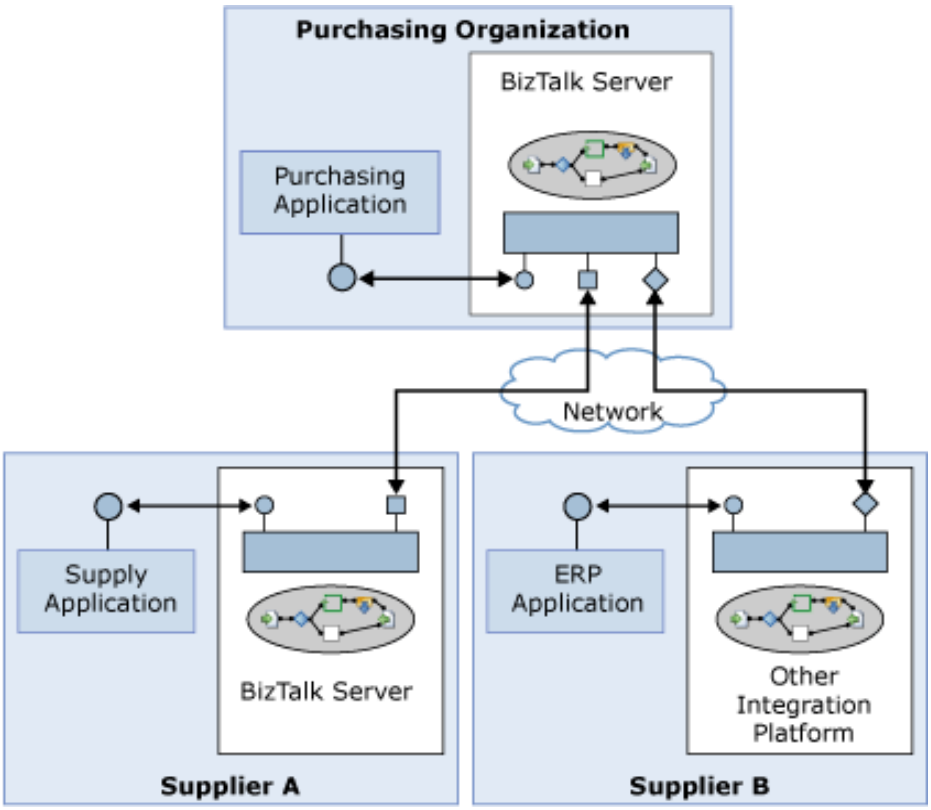


**Ilustración 6 - Diagrama de Integración Empresa a Empresa, Microsoft BizTalk Server 2009 [18]**

### Mejorar la eficiencia de los procesos empresariales internos

Gracias a que la integración facilita el flujo de información a través de los diferentes departamentos de la empresa, al pasar el tiempo la ejecución de los procesos será cada vez más eficiente, y es esta razón, una de las principales por las que en muchas organizaciones adoptan proyectos de integración de los sistemas internos.

Por otra parte, al adoptar medidas de integración también aporta en el caso de aquellas empresas que le han apostado a los *Data Warehouse,* ¿por qué? debido a que la información contenida en los *Warehouse* proviene de los diferentes

departamentos, diferentes aplicaciones (incluyendo sistemas *stovepipe*), se hace conveniente un correcto flujo y evitar la redundancia de información dentro del mismo. Adicional a esto, al valerse de la integración se pueden realizar conversiones en los formatos de los datos utilizados por las diferentes aplicaciones. Permitiendo claramente una información integra y confiable. Dándole a los niveles gerenciales, decisiones más acertadas gracias a la calidad de la data contenida.

A través de la integración, muchas de las aplicaciones se pueden expandir y ofrecer a los miembros de la organización autoservicios de tipo web. De acuerdo con los roles que desempeñan. Aprovechando de igual forma la cada vez más creciente y creciente internet.

**Reducir tiempo de espera para la adquisición de aplicaciones y mejoras**

Una indudable realidad es que el proceso de desarrollo de aplicaciones empresariales es impreciso, en especial en el ambiente tecnológico local, más aún impreciso y tardío se vuelve cuando las aplicaciones son de propósito específico. Evidentemente, son cada vez más notorios aquellos proyectos que nunca se concluyen por completo o por otra parte su costo supera por mucho el valor presupuestado o que su periodo de diseño, desarrollo, implementación y prueba está muy por fuera del cronograma estipulado previamente y al final de tiempos desfasados así como presupuestos sobrepasados, la aplicación no cumple un 100% con sus objetivos iniciales (requerimientos).

Es entendible que a pesar de que los sistemas legados, en términos de desarrollo tecnológico han quedado rezagados en el tiempo, cumplen de forma exacta su misión en la empresa y por tal motivo no han sido sustituidos (comprendiendo claramente que migrar un sistema legado es costoso e inseguro). En ese orden de ideas, se plantea que en vez de realizar un cambio sustancial de dichas

aplicaciones de misión crítica, es mejor enfocarse en buscar un mecanismo que integre los resultados que arrojen dichos sistemas con canales de comunicación que conecten los *stovepipes* con la web y demás aplicaciones. Al plantear los desarrollos de esta manera se reducirán los tiempos de codificación e implementación, por ende los costos también se verán reducidos.

Al tener una integración de las aplicaciones empresariales propias y también una intercomunicación con otras organizaciones, los ingenieros de IT se pueden enfocar más en los aspectos importantes del negocio y las políticas que en los procesos y no preocuparse en demasiado por asuntos triviales que van más ligados al proceso de codificación o aspectos técnicos de las aplicaciones. Los previamente expuestos hacen parte de una gran lista de beneficios en términos corporativos (ingresos, egresos, ganancia). Sin embargo, la integración a nivel de ingeniería de *software* también presenta un gran número de aportes a la empresa, incluso permite cerrar la brecha entre el negocio (procesos y ejecutivos) e IT (desarrolladores y aplicaciones).

Dentro de los aportes que brinda la integración al desarrollo de aplicaciones corporativas se encuentran [8]:

**Reusabilidad**

Integrar implica en términos generales que los recursos de *software* y en algunos casos de *hardware*, sean compartidos (bases de datos, procesos, servicios, procesamiento, etc.) por lo tanto gracias a que las aplicaciones a pesar de que son independientes entre sí, pueden ser vistas en conjunto, bien pudiera darse el evento de que mediante servicios, varias aplicaciones pudieran reutilizar funcionalidades que suplen necesidades comunes. Y de esa forma seria menos el tiempo de desarrollo requerido para implementar una nueva aplicación o una actualización al sistema.

**Distribución**

A medida que aumenta el despliegue y la interrelación entre las aplicaciones que son integradas, ya no serán varios sistemas empresariales interactuando entre sí, poco a poco el ambiente tecnológico tomara el matiz de un sistema altamente distribuido, dejando a un lado los vestigios de sistemas centralizados y monolíticos.

**Segmentación**

Al tener un sistema desplegado y distribuido se facilita dividir la complejidad del sistema en capaz, encapsulando las funcionalidades, procesos y servicios en cada una de las segmentaciones que se realicen (más adelante se mostrará como la integración ofrece este beneficio al desarrollo de *software* empresarial).

**Escalabilidad**

Gracias a la integración, las empresas pueden responder más rápidamente a las necesidades del cliente y potenciar sus aplicaciones de tal manera que apalanquen las soluciones ofrecidas, de igual forma la escalabilidad se hace más tangible y posible ya que no hay sistemas aislados que trabajen solos, más bien, a través de un trabajo colaborativo es más fácil alimentar con información las aplicaciones de tal manera que pueda aumentar su funcionalidad.

**Mejora del rendimiento**

Todo el potencial que puede alcanzarse con la integración puede llegar hasta el punto de mejorar el rendimiento de las aplicaciones, incluso a través de una integración a nivel de recursos de *hardware*, de tal manera que puede aprovecharse al máximo el multiprocesamiento, hilos distribuidos, multihilos, etc.

Por otra parte, la mejora del rendimiento se puede ver ejemplificada asimismo en el poco cambio de código que debe realizarse con el fin de potenciar las aplicaciones y añadirles nuevas funcionalidades.

**Continuidad operativa**

Debido a la envergadura notable que tienen las aplicaciones empresariales, la probabilidad de encontrar puntos de fallo, así como cuellos de botella, es muy alta. Sin embargo, se puede eliminar mediante la replicación y distribución. Gracias a lo cual los tiempos de inhabilitación en causa de fallos se ven reducidos. Por ende contribuye a que los servicios estén expuestos (salvo pequeños cambios sustanciales) 24/7.

**Desarrollo Rápido**

Los desarrolladores de aplicaciones pueden concentrarse en el desarrollo rápido y despliegue de la funcionalidad de negocios, sin dejar de ser transparente a la infraestructura subyacente. Los servicios pueden ser utilizados en combinaciones impredecibles para formar aplicaciones.

Al aplicarle a la integración una filosofía o enfoque a servicios (posiblemente valiéndose de la arquitectura SOA), se pueden adquirir beneficios adicionales a los estipulados previamente. Por ejemplo, siguiendo una metodología en la que se conviertan en servicio todos aquellos puntos funcionales que se detecten, se podrán componer conglomerados de servicios teniendo en cuenta los procesos de negocios que se efectúen, ya que en primera instancia, la búsqueda de mejoras sustanciales en los procesos es lo que motiva la existencia de la integración y todo lo que se propone en este documento.

Con el mismo enfoque orientado a servicios se logra obtener que aquellas diferentes implementaciones de un mismo servicio puedan ser intercambiadas de forma sencilla y fácil aun en tiempo de ejecución, y mejor aún, no impactar al usuario final, incluso sin hacer rediseños de la aplicación.

### 7.3 Tipos de integración

Es notoria la necesidad de integración que se presenta en el ambiente empresarial y tecnológico del contexto mundial actual, al ver la integración desde una posición general, percibiendo sus beneficios y contribuciones al desarrollo de *software* de gran escala, se llega a la conclusión de que es útil y aún más necesaria. Sin embargo, desde un punto de vista general, no se puede detectar en qué campos se puede apropiar de la integración para posteriormente ser aplicada. Por tal motivo el siguiente paso es entender las diferentes formas en las cuales se puede realizar integración.

Al analizar dichos tipos en los que se puede integrar se pretende dividir o segmentar en sub-problemas la problemática general de integración, de tal manera que paso a paso se resuelvan las dificultades y de ser posible tener un ambiente unificado en todos los aspectos. Las diferentes capas (a nivel empresarial) en las cuales se puede aplicar integración son [8]:

- Capa de datos
- Capa de procesos de negocios (internos)
- Capa de procesos de negocios (B2B)
- Capa de presentación
- Capa de aplicaciones

Para una definición detallada de las capas en las cuales se puede aplicar integración remitirse a [12]

**Integración de Aplicaciones**

La integración de aplicaciones se centra en la lógica de compartir la funcionalidad de negocio, integrar la lógica del negocio la cual está representada a través de lenguajes de programación. La lógica del negocio contienen las reglas necesarias para interpretar o construir apropiadamente la información.

El uso de API'*s* (*Aplication Programming Interface*) es una de las soluciones más utilizadas para lograr este tipo de integración. Las API'*s* son integración de caja negra, ya que ocultan los detalles de implementación, pero a su vez expone una interfaz, la cual es un contrato para la aplicación que vaya a consumir la API.

Dentro de los mecanismos que comunican las aplicaciones se encuentran [7]:

- ***Distributed Object Technology*:** La tecnología de objetos distribuidos fue una solución que nació después de que los principios de la orientación a objeto fueran llevados a un contexto distribuido. En el cual se tiene acceso a los métodos de objetos que se encuentran implementados remotamente. En este mecanismo de integración de aplicaciones se deben definir y eventualmente utilizar correctamente las interfaces necesarias para la invocación de los métodos del objeto distribuido y de esa forma llevar a cabo la integración. dentro de esta tecnología se encuentra CORBA, Microsoft COM+, ambiente de J2EE.

- **MOM:** Middleware Orientado a Mensajes, el cual consiste en un mecanismo de comunicación que se da en el envío de mensajes entre aplicaciones. En este caso el Middleware se encarga de recibir un mensaje que ha de ser

entregado a un receptor remoto. En este caso el emisor y el receptor son naturalmente aplicaciones que requieren interactuar y están en el mismo contexto empresarial. Dependiendo del desarrollador de este tipo de Middleware, se pueden realizar diferentes tipos de configuraciones para el envío de mensajes.

- ***Transaction processing monitors*** **(TPMs):** Este mecanismo es principalmente usado en las aplicaciones que interactúan entre sí pero que manejan información vital para el negocio. Se puede decir que es la tecnología para sistemas de misión crítica. Todo ello con el objetivo de preservar la integridad de la información que se maneja, y mitigar riesgos de seguridad en el envío de la información.

Dependiendo de las necesidades que surgen dentro de la empresa, se retoma un estilo de comunicación, o varios o combinaciones de los mismos. Señalando que como ya se expuso previamente, las empresas tienen por naturaleza misma ambientes de aplicaciones heterogéneas en muchos sentidos.

Una forma usual de combinar los beneficios de estos tipos de integración, es precisamente depositar esa lógica de mensajería y enrutamiento dentro de un Bus de Servicio Empresarial. En tal caso se convertiría al ESB en una forma de realizar la integración de aplicaciones.

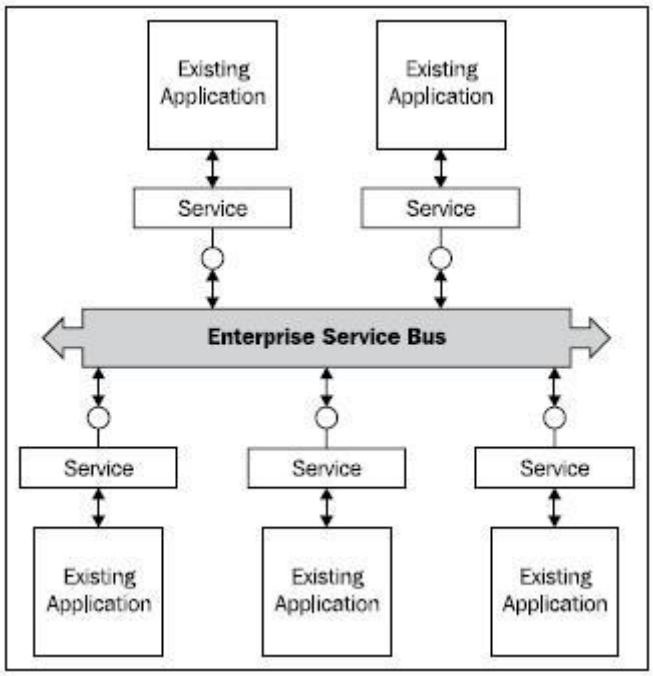


**Ilustración 7 - Integración de aplicaciones [8]**

Con un contexto de integración esclarecido y con el Bus de Servicio Empresarial en escena, se procede a seguir ahondando en detalles sobre esta tecnología, la cual puede ser vista como arquitectura o pieza de *software*.

Se analizará que características ayudan a la integración de aplicaciones y de igual forma de que maneras se pueden aprovechar dichas características para la implementación de SOA en ambientes de aplicaciones heterogéneas.

# 8 UN ENFOQUE A LA TECNOLOGÍA

## 8.1 El Bus de Servicio Empresarial (ESB)

Diseñar, implementar y desarrollar un sistema usando como guía la arquitectura orientada a servicios no es una tarea sencilla, debido a que dicha arquitectura es una filosofía de desarrollo emergente, lo cual implica que constantemente se presentan cambios y se plantean ideologías variadas respecto a los diferentes engranajes que la conforman.

Uno de los elementos más representativos que interactúa con la arquitectura en cuestión es el Bus de Servicio Empresarial. Acerca de este bus existen muchas posiciones, muchos conceptos, muchas percepciones, en fin un sin número de opiniones. Ahora bien; ¿Qué caracteriza al ESB? ¿Cuán importante es al implementar un sistema con orientación a servicios? asimismo ¿cuál es el porvenir de dicho elemento?

Cuando una empresa pretende realizar una adaptación o una migración a un sistema basado en una arquitectura orientada a servicio, los encargados deben ser conscientes de que se encararán grandes retos y obstáculos que estarán en función de la complejidad de los procesos de negocios y de la robustez de la TI de la empresa. Adicionalmente, deben tener en cuenta que este proceso de adaptación o cambio debe ser gradual.

Un punto inicial en estas migraciones es establecer un puente de comunicación que vincule diferentes departamentos de la empresa, subsistemas (los cuales funcionan con diferentes protocolos de comunicación, y ofrecen diferentes tipos de servicios) y hasta sistemas de otras compañías. Una vez, entendido que debe existir este canal, se crea una necesidad que suplir; el encargado de suplir esta necesidad es el Bus de Servicio Empresarial.

Cabe señalar que en un sistema SOA ideal solamente haría falta una integración con WS-* y SOAP, lo que indica que todos los procesos hasta la comunicación, se harían a través de los servicios web, valiéndose de todas las utilidades que dicha tecnología ofrece. Sin embargo, los sistemas actuales son variados en todos los aspectos, lo cual exige un puente más amplio, en el cual converjan diferentes tipos de comunicación provenientes de sistemas heterogéneos. Lo que muestra en primera instancia la necesidad del ESB.

## 8.2 Pertinencia del ESB

Piense por un momento en una empresa que cuenta con un número n de clientes (consumidores), y que para satisfacer las necesidades de estos, la empresa implementa sus procesos de negocio usando servicios y luego hace que interactúen entre sí, agrupándolos. Estos servicios están distribuidos en m proveedores, divididos en departamentos, a los cuales se les asignan procesos de negocios que deben suplir; es decir que si los n clientes necesitan en un momento dado de los m proveedores y las conexiones que la empresa maneja son de tipo P2P, tendríamos un entramado de n*m conexiones. Ahora aumentando la complejidad de la situación, piense en que bajo ciertas circunstancias un proveedor se convierte en consumidor y necesita de un servicio ofrecido por otro proveedor. Lo anterior aumentaría el número de conexiones, generando un caos tal que en cualquier momento ocasionaría que la red sencillamente colapsara y atentara en contra de la continuidad operativa de la empresa. En este punto podría decirse que hasta la descripción de la situación es complicada.

Para aquellas organizaciones con un número notablemente reducido de aplicaciones, las conexiones de tipo *point-to-point* son viables y no impactan en gran medida sobre la red y su *performance*, sin embargo a medida que crecen las aplicaciones que interactúan, el tipo de conexión P2P no contribuye a la escalabilidad y de no tratar esta situación con sumo cuidado y atención, podría

fácilmente incurrir en gastos para la empresa y acarrear perdidas en vez de ganancias [9].

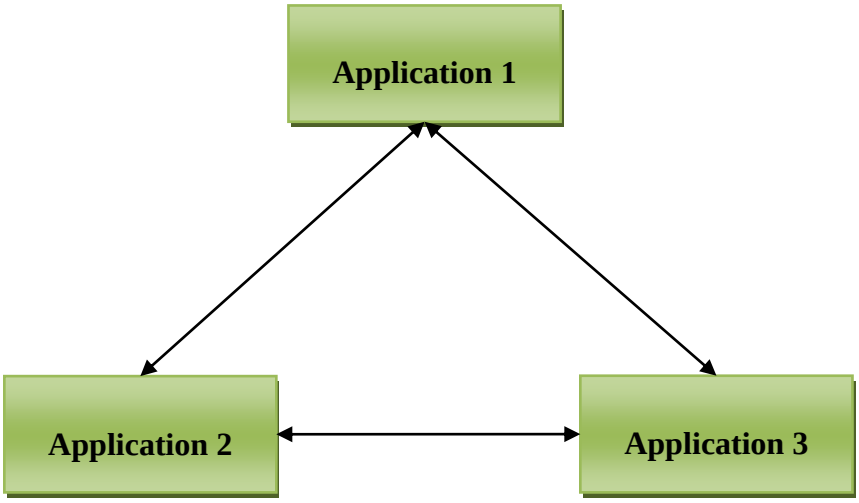


**Ilustración 8 - Integración de 3 Aplicaciones Usando P2P [9]**

En la Ilustración 8 se muestra una malla de conexiones para  aplicaciones, como es evidente hacen falta en este caso, solamente 3 conexiones para integrar 3 aplicaciones, dicho número de conexiones es obtenido utilizando la fórmula matemática:

$$c = N(N-1)/2$$

Ahora considere una empresa que requiere la integración de 6 aplicaciones, a través de conexiones P2P, la gráfica representativa para la situación actual se muestra en la Ilustración 9.

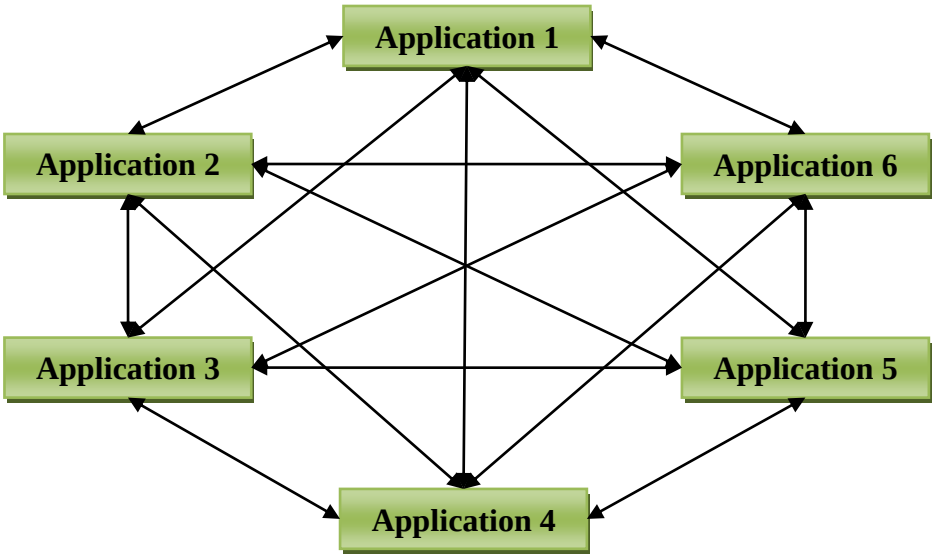


**Ilustración 9 - Integración de 6 Aplicaciones Usando Conexiones point-to-point [9]**

Es evidente que en cualquier momento la red puede colapsar o su rendimiento disminuirá considerablemente.

Cuando las aplicaciones se encuentran integradas a través de un Bus de Servicios Empresarial, no interactúan directamente entre sí. En vez de ello cada una de las cuales generan una y solo una conexión al bus de servicio y a través del mismo se relacionan con las demás aplicaciones.

Por lo tanto en solución al caos expuesto en párrafos anteriores, se puede acudir a un medio de comunicación que actúe como un mediador entre los consumidores y los proveedores. Y en este nivel ya empieza a entrar en escena el ESB.

El Bus de Servicio se encargaría de recibir la solicitud (proveniente del Consumidor), contactar con el Registro de Servicios, llamar al Proveedor contenedor del servicio, y entregar el mensaje de respuesta al Consumidor

solicitante. Este comportamiento contribuye a la transparencia en la ubicación, así como a la separación clara del Servicio de Negocio (el cual es expuesto u ofrecido al cliente a través del WSDL, definiendo entradas y salidas) y la Implementación del Servicio (la cual es codificada y al ejecutarse cumple con el servicio expuesto).

Como primer beneficio, al momento de realizar una integración valiéndose del ESB como tecnología para la relación entre aplicaciones, se considera el hecho de que para un numero de seis aplicaciones hace falta seis conexiones (en conexiones p2p serían quince), o para un numero de diez aplicaciones se necesitaría diez conexiones (en conexiones *point-to-point* las conexiones ascienden a cuarenta y cinco), lo cual optimiza el rendimiento en la red y contribuye a tiempos de latencia menores.

En ese orden de ideas también se puede destacar otro beneficio de apropiarse del ESB como tecnología de integración. Gracias a que las aplicaciones se relacionan de forma indirecta, la mantenibilidad del sistema global es más fácil y rápida. Por ejemplo, dado el caso que surgiera la necesidad de agregar una nueva aplicación a todo el sistema, solamente se haría necesaria una sola conexión (con el ESB). Con la cual sería suficiente para que la aplicación agregada interactúe con el resto de aplicaciones que ya se encuentran implementadas; asimismo no hace falta realizarle ninguna modificación a las aplicaciones existentes y que hacen parte de la estructura integrada. Contrastando con lo anterior. en el caso de una esquema p2p, donde agregar una nueva aplicación implica generar conexiones con todas y cada uno de los sistemas vigentes y en funcionamiento - haría falta realizar una modificación o edición del código fuente de dichos sistemas de software - lo previo es aplicable en el caso de necesitar remover aplicaciones[9].

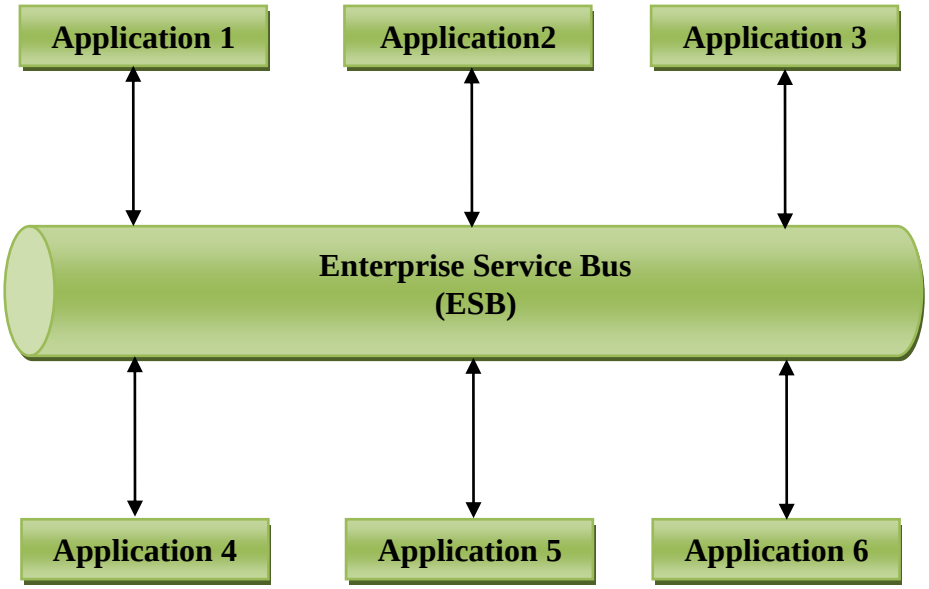


**Ilustración 10 – Integración indirecta de Aplicaciones Usando ESB [9]**

Un beneficio adicional que aporta el uso de un Bus de Servicio, es la facilidad que brinda una vez que se desee remplazar un proveedor de servicio por uno nuevo, en caso tal que el proveedor vigente ya no esté capacitado para brindar funcionalidad alguna. La razón principal de esta simplicidad es el hecho de que no es necesario especificar mediante *hard-code* la dirección de red del *end-point* de la aplicación cliente. Lo anterior es posible gracias a que la naturaleza del ESB propone un enrutamiento interno que ubique la dirección del *end-point* del proveedor del servicio, basándose naturalmente en el contenido y contexto de la petición o solicitud enviada por el consumidor.

***Routing***

A este punto surge como primera competencia solida de la tecnología ESB, la capacidad de *Routing* o enrutamiento, gracias a la cual el proveedor es

completamente inconsciente de la "identidad" de la aplicación cliente que está realizando la solicitud de consumo de servicio; de igual forma la entidad que desea acceder al servicio es indiferente a la ubicación o "identidad" de su servidor. El ESB debe ofrecer direccionamiento, de tal manera que dado un mensaje solicitud de búsqueda de servicio, este debe, a través de métodos de enrutamiento, ubicar en el registro de servicios, el repositorio contenedor del servicio deseado [6].

A pesar de que el objetivo general de *Routing* en un ESB, es el planteado anteriormente, hay diferentes formas de llevarlo a cabo. Por ejemplo: se puede acudir a métodos de *Routing* estáticos o determinísticos; también es posible que se realice enrutamiento basado en contenido, enrutamiento basado en políticas o enrutamiento basado en reglas de servicio.

Existen dos formas fundamentales en las cuales se puede realizar el enrutamiento dentro del Bus de Servicio, se trata de la tecnología ORB (*Object Request Broker*) y la tecnología de mensajes asíncronos.

**ORB**, ofrece todo un conjunto de funcionalidades que contribuyen a la comunicación y *marshalling* (de parámetros y de valores retornados) entre objetos distribuidos. Nótese que el *marshalling* realizado en el ORB es independiente de plataformas de software en las que se encuentra soportado el desarrollo, así como independiente de infraestructura de hardware. El ORB se encarga internamente de realizar las conversiones necesarias para un correcto funcionamiento y comunicación entre aplicaciones [9].

En términos generales OBR tiene tres pasos básicos que acentúan su funcionamiento:

1. Una vez que un objeto o componente desea consumir un servicio ofrecido por otro objeto o componente, primero obtiene una referencia al proveedor del servicio.

2. Cuando la referencia ha sido establecida, la instancia del ORB que se encuentra en el lado del cliente captura los parámetros y realiza un *marshalling* de los mismos, más adelante el ORB en el lado del servidor realiza un *marshalling* de los parámetros e invoca el método o servicio.

3. El ORB captura los resultados o salidas del método o servicio, realiza un marshalling de valor retornado y el ORB en el lado del cliente mediante un *marshalling* de dicho valor retornado entrega a la aplicación cliente la salida del servicio consumido.

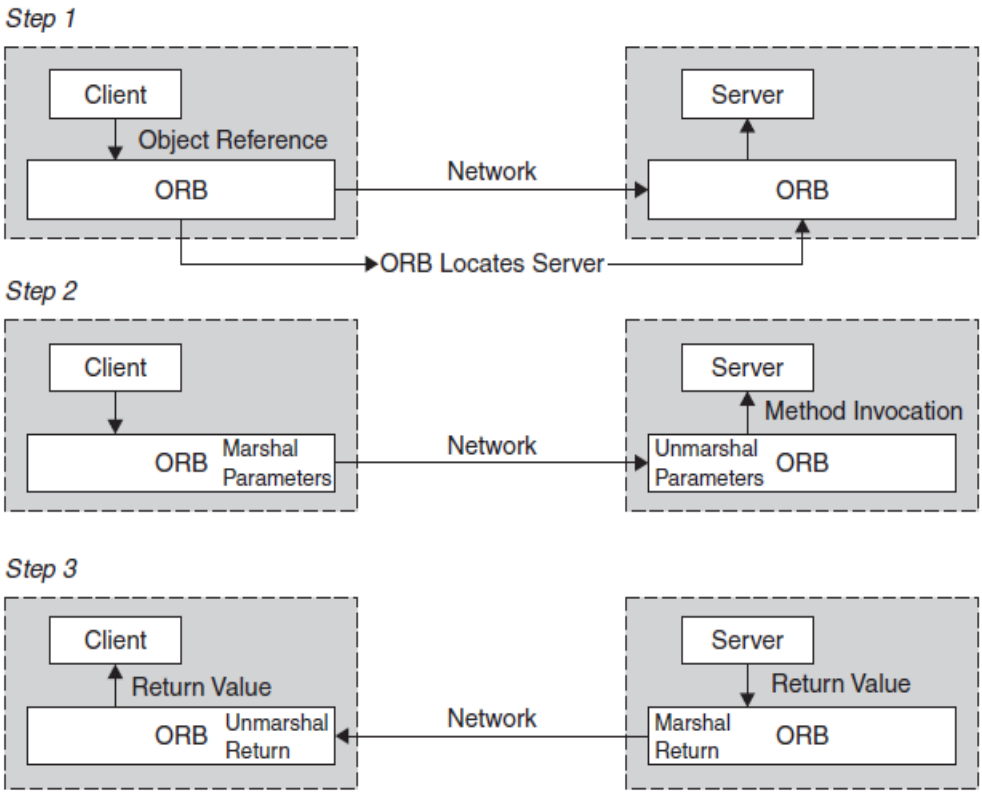


**Ilustración 11 - Pasos Básicos de Integración a través de ORB [9]**

Otra principal tecnología con la que es posible realizar *Routing*, es la de mensajes asíncronos. Este tipo de enrutamiento facilita y aporta en gran medida el bajo acoplamiento de las aplicaciones, así como aporta a una mayor independencia en cuanto al funcionamiento de las aplicaciones.

Los mensajes que comparten las aplicaciones en este tipo de tecnología de intercambio de información, comúnmente están basados en XML, debido a su flexibilidad y nivel de independencia de aplicaciones que este lenguaje ofrece. En su forma más simple y fundamental, los mensajes que a este nivel se envían y reciben, manifiestan una estructura básica formada por: la cabecera, las

propiedades y el contenido del mensaje. La cabecera es usada por el componente de software encargado de la gestión de mensajería y comunicación (MOM - *Messages Oriented Middleware*) y desarrolladores de aplicaciones (consumidores o proveedores) para proveer información relacionada y concerniente a destino(s) del mensaje, enrutamiento, tipo de mensajería y tiempo de vida y/o latencia.

En la sección de propiedades, se definen un conjunto de pares de la forma nombre/valor. Estos pares son información relevante del cuerpo del mensaje, pero que son colocadas en la sección de propiedades porque han sido apartadas para un uso especial, por ejemplo para definir constantes, etc.

Por último se encuentra la sección del cuerpo del mensaje o carga útil, que naturalmente posee la información pertinente del mensaje, como lo son inputs, outputs, descripción del servicio, etc.

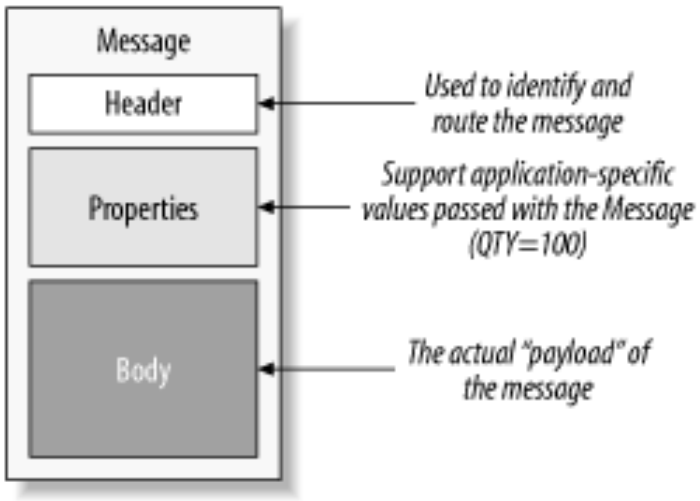


**Ilustración 12 - Estructura Básica de un Mensaje [10]**

Los tipos más comunes de mensajes que se utilizan en el desarrollo de software distribuido son [10]:

**Mensaje Texto**: Es la forma de mensaje más utilizada, en donde el cuerpo del mensaje es una cadena, como texto con significado literal o un documenta con la estructura XML. Como ejemplo claro de este tipo de mensajes se encuentran los mensajes SOAP.

**Mensaje Byte**: El cuerpo de este mensaje está formado por un arreglo simple de bytes.

**Mensaje Objeto**: La carga útil de este tipo de mensajes contiene un objeto principalmente desarrollado en Java, el cual implementa una interfaz de serialización de objetos para manejar su persistencia y portabilidad.

**Mensaje *Stream***: El contenido es un flujo de datos formado por tipos primitivos del lenguaje Java.

**Mensaje Mapa**: Conformado por parejas de valores. Con la forma clave-valor.

Cuando una aplicación cliente envía un mensaje asíncrono a un servidor, la primera no necesita de una respuesta inmediata para seguir su ejecución normal; lo cual permite que se puedan efectuar más actividades en cada una de las aplicaciones y llevar a cabo más tareas en un lapso determinado, por ende trae consigo beneficios en términos de escalabilidad del sistema.

El intercambio de mensaje en este caso se realiza sin una conexión constante entre las aplicaciones (solo cuando sea necesario el intercambio de mensajes), caso contrario a la conexión utilizando un ORB. En el evento actual, las aplicaciones interactúan a través de una cola de mensajes o canal de mensajes. Luego entonces una aplicación A envía un mensaje a la cola de mensajes, mientras continua con su ejecución normal. Más adelante la aplicación B recupera

el mensaje que se encuentra en la cola y satisface la solicitud enviada por el cliente. Es de gran ventaja este desacoplamiento, ya que no se corren riesgos de interrupción de la ejecución de las aplicaciones caso tal se pierda la conexión con la red. Una ventaja adicional y de gran validez es la poca probabilidad de presentarse un desbordamiento de solicitudes por parte de los clientes hacia el proveedor (caso comúnmente presentado en desarrollos con Objetos Distribuidos). Este hecho es posible gracias a la independencia existente entre las aplicaciones relacionadas. Una vez que la aplicación proveedora está libre de actividades y se encuentra en capacidad de procesar y suplir una solicitud, esta puede dirigirse a la cola o canal y capturar el siguiente mensaje [9].

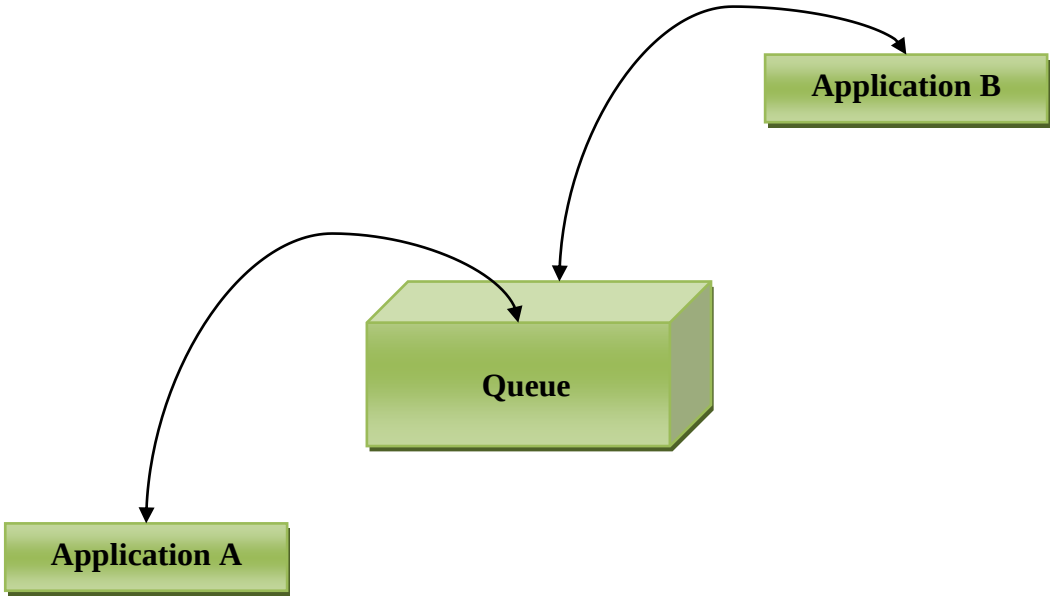


**Ilustración 13 – Conexión Indirecta de Aplicaciones a través de Mensajes Asíncronos [9]**

El tipo de mensajería que se utiliza en este tipo de integración depende del nivel de relación que se desee tener entre el consumidor y el proveedor. Cuando se requiera que un mensaje sea recibido por varias aplicaciones receptoras, se utiliza una mensajería del tipo publicar y suscribir. Sin embargo, también existe el

dominio de mensajes *point-to-point* en el cual dos aplicaciones comparten de manera privada mensajes entre sí. Cabe señalar que estos son dos modelos principales, sin embargo existen muchas otras formas especializadas de mensajería.

En el modelo de pub/sub múltiples consumidores se **suscriben** a un tópico (cola de mensajes) de su interés, mas adelante cuando el proveedor envía un mensaje a través de la **publicación** del mismo en la cola, y luego cada suscriptor recibe una copia del mensaje original enviado. En este modelo, aquellos mensajes que no poseen suscriptores al momento de ser publicados pueden ser descartados o marcados como no confiables [10].

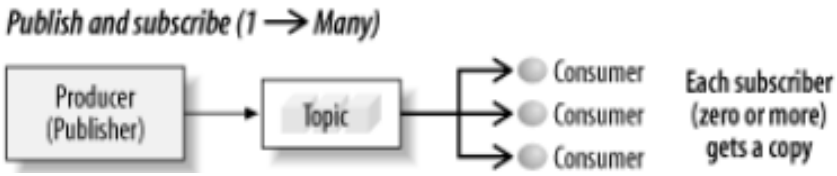


**Ilustración 14 - Modelo Publica y Suscribe [10]**

Por su parte el dominio *pont-to-point* hace referencia a un tipo especial de mensajería, en el cual solamente uno y no más que un solo consumidor puede recibir el mensaje que previamente fue enviado a la cola de mensajes por el proveedor, aun cuando varios consumidores estén vinculados a la cola de mensajes. Estos consumidores adicionales pueden ser utilizados como aplicaciones redundantes o como entidades que funcionan como equilibradores de carga (por ejemplo si la aplicación cliente se encuentra ocupada y no puede acceder al mensaje que se encuentra en la cola). En el caso de modelos *point-to-point*, los mensajes pueden esperar en la cola de mensajes hasta que acceda una aplicación cliente que los consuma.

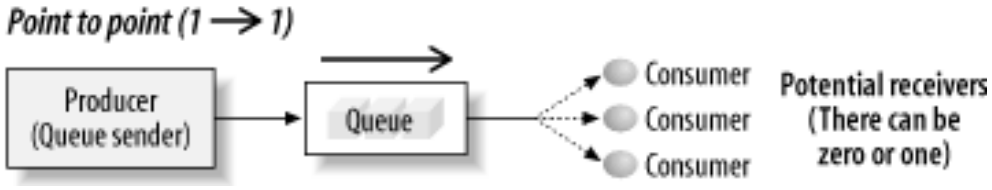


**Ilustración 15 - Modelo point-to-point [10]**

Ahora bien, ¿cuál de los dos modelos se debe utilizar? en gran medida dependerá de cuantos consumidores deben recibir una copia del mensaje enviado por el Proveedor. De igual forma es totalmente válido señalar que es posible definir un patrón propio de mensajería de acuerdo a las necesidades o requerimientos que se buscan alcanzar; probablemente sea necesario establecer un canal de mensajería dentro del ESB, así como una estructura propia o plantilla de mensajes, sin implementar ningún modelo de mensajería o definir políticas de colas de mensajes o tiempos de latencia.

Se presenta la siguiente situación como ejemplo. En una cadena de suministros, un proveedor ha decidido realizar una reducción de precios de uno de sus productos, así que mediante un modelo de pun/sub realiza un *broadcast* a todos sus compradores enviando una notificación que anuncie la reducción de precio del producto en cuestión. Sin embargo, en caso tal un comprador desee responder a la notificación de baja de precios enviada por el proveedor; pues desea comprar dicho producto, se necesitará un modelo *point-to-point* en donde únicamente se involucre a las dos partes interesadas en realizar la transacción: el comprador y el vendedor.

Por lo tanto el tipo de modelo o dominio a utilizar depende de las necesidades y la lógica del negocio sujeta a cada empresa u organización.

Estas dos grandes tecnologías de enrutamiento e integración de aplicaciones, se utilizan como columna vertebral de dos formas especializadas de ESB.

El primer tipo de Bus de servicio empresarial utiliza ORB como su eje fundamental. Este tipo de Bus tiene la ventaja de ser fácil de instalar y es más barato. Sin embargo, la funcionalidad que proporciona no aporta beneficios en términos de escalabilidad, y se acentúa la dificultad anterior cuando aumenta el número de transacciones y operaciones que se deben ejecutar. Por lo tanto, esta clase de ESB debe ser utilizado cuando se prevé un número limitado y regular de transacciones y funcionalidades con poca probabilidad de aumento en las operaciones y procesos. Este tipo de Bus se diseña generalmente para hacer frente a los servicios web, XML y Java RMI solamente.

El segundo tipo se basa en los sistemas de mensajería asincrónica. Es relativamente más caro y requiere una configuración más elaborada. Este tipo de ESB tiene tres ventajas principales sobre el primer tipo.

La primera ventaja que ofrece la utilización de la tecnología en cuestión es que proporciona una solución altamente escalable en función del volumen de las transacciones y por lo tanto es capaz de soportar una tasa mucho mayor de transacciones. El segundo beneficio es que a través de esta clase de Bus de Servicio se puede integrar un conjunto más diverso de aplicaciones. Probablemente la ventaja más importante de este es que puede garantizar la entrega de los mensajes entre aplicaciones.

Es importante señalar que los servicios web en sí no son una garantía de la entrega de mensajes entre aplicaciones. Esta garantía de la entrega de mensajes puede ser necesaria para determinadas operaciones, debido a razones contractuales o legales. También se debe tener en cuenta que, en ausencia de esta garantía, la aplicación consumidora de servicios se puede bloquear si se rompe la conexión de red o si el proveedor de servicios no se ejecuta al mismo tiempo. Los sistemas de mensajería asincrónica (MOM) pueden garantizar la

entrega de mensajes por la persistente de los mismos en ambos lados (el lado del cliente y el lado proveedor de servicios) de la red.

**Transformación de protocolos**

Con el objetivo de ofrecer una solución al problema de la integración de aplicaciones empresariales, explicado previamente; el Bus de Servicio debe ofrecer una alternativa viable en cuanto a la heterogeneidad de protocolos.

Las grandes organizaciones empresariales tienen aplicaciones que emplean una gran variedad de protocolos de comunicación; es decir, los consumidores pueden realizar solicitudes de servicios a través del ESB utilizando cualquier tipo de protocolo; bien sea HTTP, SOAP (JMS), SOAP (MQ), etc.

Eventualmente llegará el caso en el cual no coincidan el proveedor de servicios y su consumidor. Por lo tanto sin la facilidad de realizar una conversión de un tipo de protocolo a otro, el cliente o consumidor no podrá invocar o consumir el servicio que es ofrecido por el proveedor.

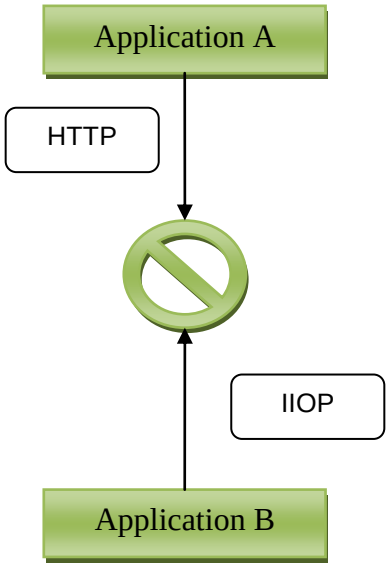


**Ilustración 16 - Incongruencia de Protocolos de Comunicaciones [9]**

En términos ideales el mecanismo de comunicación debería estar estandarizado en cualquier gran empresa y en ese orden, todas las aplicaciones se relacionarían bajo el mismo protocolo de comunicación. Por ejemplo apropiarse del protocolo HTTP como único lenguaje de comunicación, el cual es utilizado en la tecnología de servicios web.

Si bien es cierto que la integración se facilitaría en gran medida una vez que las aplicaciones utilicen el mismo protocolo, esta opción no es del todo viable, ya que incurriría en tiempos de desarrollo para la modificación de las aplicaciones existentes y más importante aún, la unificación de protocolos de comunicación no es recomendable en lo concerniente a la calidad del servicio (QoS) y seguridad del sistema empresarial.

Por lo tanto es vital que para garantizar escalabilidad e interoperabilidad dentro de la estructura del ESB se ofrezca conversión de protocolos de comunicación [9].

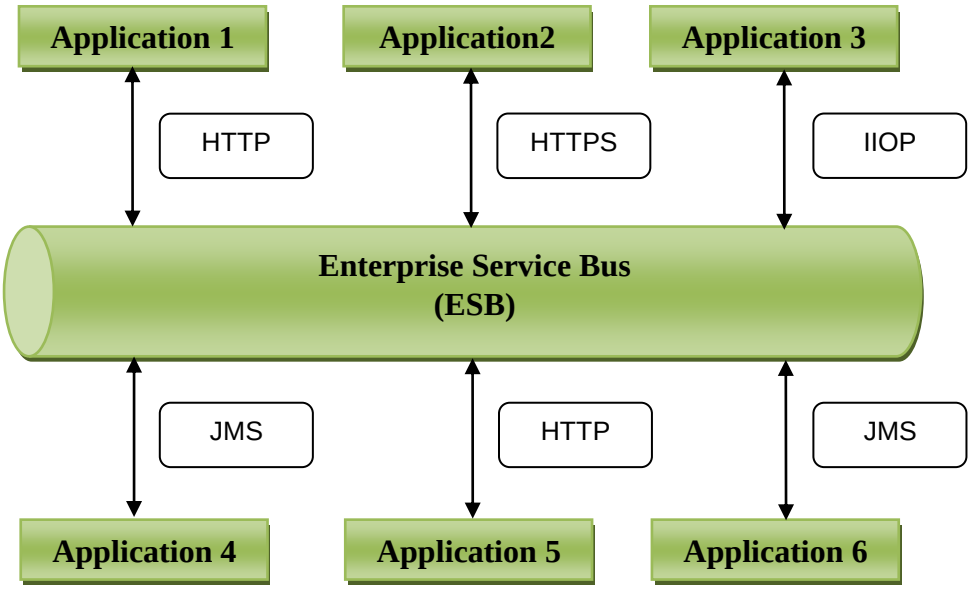


**Ilustración 17 - Diversidad de Protocolos en Aplicaciones Integradas [9]**

### Transformación de Datos o Mensajes

Debido a la naturaleza heterogénea de los sistemas, es apenas razonable que en muchas ocasiones la solicitud de servicio de negocio, no tenga la estructura esperada por el proveedor de servicio. Lo cual nuevamente evita la correcta interacción entre las aplicaciones. En estas circunstancias, el Bus de Servicio Empresarial convierte la estructura del mensaje solicitud a una estructura aceptable para el proveedor. Por ejemplo: conversión de XML a Cobol Copybook, conversión de objetos a XML, etc. o Incluso los tipos de datos en caso tal que el mensaje sea un *Stream Message* o un Mensaje Objeto.

De igual forma es importante dicha característica en caso contrario; es decir la respuesta o resultado de la operación ofrecida por el servicio proveedor no contiene una estructura, presentación o codificación entendible para el cliente. Por lo tanto el ESB mediante reglas de conversación debería transformar la

información de tal manera que sea entendible para el cliente. Una forma de aplicar dicha capacidad o competencia, es mediante Esquemas de XML, los cuales definen la estructura de los documentos.

**Modificación de mensajes**

En este nivel, el ESB debería estar apto para realizar cambios más circunstanciales al mensaje de solicitud. Es decir debe tener la habilidad de agregar o en términos generales modificar la información de tal forma que el proveedor reciba todas las entradas necesarias para desarrollar su servicio. Todas estas modificaciones lógicamente basadas en una serie de reglas de modificación.

La conversión de formato, incluso la provisión de datos no incluidos en el mensaje original y la reestructuración de los datos en sí, hacen parte de la característica del ESB analizada en esta sección [6].

**Transparencia**

Gracias a las tres capacidades fundamentales que debe tener un Bus de Servicios, - a saber: *Routing, Transformación de Mensajes y Transformación de Protocolos* - se hace posible la virtualización y transparencia en tres aspectos como mínimo:

- **Interfaces**: Gracias a la transformación de mensajes no se hace estrictamente necesario que la interface que utilice el consumidor del servicio este en completa correlación y congruencia con la exigida por el proveedor, ya que el ESB armoniza ambas interfaces a través de la conversión del mensaje solicitud. Dándole una estructura que encaje con las condiciones exigidas por el servidor.

- **Protocolos**: Para interactuar los entes que hacen parte de la relación, no están obligados a utilizar un mismo protocolo de comunicación. Por ejemplo un proveedor de servicios que recibe solicitudes de tipo SOAP sobre HTTP puede a través del ESB servir a un consumidor de servicios que solamente entiende Java RMI sobre IIOP.

- **Localización o identidad**: Gracias a que la aplicación solicitante no conoce una identidad exacta del proveedor de servicio así como el proveedor desconoce la localización del consumidor. Se hace posible que una solicitud de servicio pueda ser atendida por cualquiera de los proveedores del mismo servicio en caso de existir redundancia. Esto permite que el desarrollador del servicio pueda agregar o remover una aplicación proveedora sin la obligación de hacer *shutdown* al sistema empresarial, dando así una disponibilidad continua al servicio expuesto y publicado.

**Características no-funcionales (QoS)**

Como es natural en una tecnología que tiene fundamentos en principios de la Ingeniería de *Software* además de tener requerimientos funcionales, el ESB tiene aspectos relacionados con la Calidad del Servicio (QoS) que también debe suplir. Los dos requerimientos principales que el ESB debe cumplir son [9]:

- **Confiabilidad y Rendimiento**: Dentro del buen rendimiento de un ESB, se puede contar que el tiempo de respuesta a la solicitud de un servicio no supere un tiempo fijado (de acuerdo a las características de la topología de la red entre ellas su velocidad, así como la complejidad del servicio a consumir), como por ejemplo un valor de tiempo de espera estándar es cincuenta milésimas de segundo. Por su parte en cuanto a la confiabilidad, se refiere al tiempo en el cual se encuentra disponible un servicio y bajo las

condiciones que ofrece el ESB un ejemplo de una métrica razonable para la disponibilidad de un servicio es de 98% del tiempo.

- **Seguridad**: la seguridad es un aspecto a tener muy en cuenta en los sistemas distribuidos. Y más importante aun cuando además de tener un sistema distribuido empresarial, se desea guardar algún tipo de relación con aplicaciones que hacen parte de organizaciones terceras. El ESB como tal no ofrece una capa de seguridad, sin embargo, aquellos Buses que proponen seguridad en su funcionalidad, en realidad ofrecen mediante un *framework* un software de seguridad que se acople a su arquitectura y permita al Bus relacionarse con redes externas sin tener que lidiar directamente con firewalls y otros sistemas que cumplan políticas de seguridad de otras organizaciones (Tivoli Suites hace parte de dichos software de gestión de la seguridad).

### 8.3 Tipos de ESB

En el ambiente comercial y *open source* existen fundamentalmente tres tipos de ESB.

- **ESB basado en Servidor de Aplicaciones**: La columna vertebral de esta especialización de Buses son servidores de aplicaciones. Principalmente se soporta en llamados síncronos a funciones o servicios, sin embargo, algunos de los mismos, tienen capacidad de soportar mensajería asíncrona. La principal ventaja que posee este tipo de ESB es su bajo costo en comparación con otros productos basados en la misma tecnología de software, asimismo, otra ventaja que posee es su facilidad de configuración en tiempo de despliegue, es decir el momento en el cual, el sistema empresarial está siendo instalado y adaptado al ambiente real de la organización.

La fortaleza que posee los ESB basados en *App Servers* es su capacidad para el tratamiento de tecnologías tales como XML y funcionalidades cimentadas en lenguajes abiertos tales como Java, pero, el tratamiento de su funcionamiento interno se hace tedioso una vez que empieza a aumentar la variedad de aplicaciones que deben interactuar en el sistema. Por lo tanto, se utiliza cuando hay un número relativamente limitado de aplicaciones a integrar, por ende no es fuerte de este tipo la escalabilidad una vez que el número de aplicaciones integradas comienza a sufrir un aumento considerable.

- **ESB basado en Sistema de Mensajería**: Soporta mensajería asíncrona y síncrona, con fuerte en la primera clase de mensajería entre aplicaciones, las tres principales ventajas que posee son:

    - Ofrece la base tecnológica que permite el mayor nivel de escalabilidad, en cuanto al problema de la solución de la integración de aplicaciones empresariales; debido a que permite el mayor número de transacciones basadas en el sistema de mensajería; así como admite un alto nivel de aumento en los procesos que dentro de él se llevan a cabo.

    - Permite realizar integración con la mayor diversidad de aplicaciones posibles, entre las cuales se encuentran aplicaciones legadas desarrolladas en lenguajes tales como C, C++ o Cobol y aplicaciones con herramientas actuales tales como Java y .net.

    - Garantiza la entrega de los mensajes enviados entre el consumidor y el proveedor, una característica únicamente ofrecida por este tipo de

ESB. Sin embargo, tienen un costo comparativamente mayor respecto a los demás.

- **Hardware ESB**: Este tercer tipo de ESB se basa en el hardware para hacer la mayoría de los procesos. Estos dispositivos ESB son fáciles de configurar, ofrecen una mayor seguridad y un procesamiento eficaz. Sin embargo, tienden a centralizar el sistema empresarial y es el menos apropiado en términos de escalabilidad de las aplicaciones. No es recomendable para aquellas organizaciones que buscan apropiarse del ESB como tecnología soporte que permite un cambio gradual a Arquitecturas Orientadas a Servicios.

# 9 ESB BASADO EN SOA

## 9.1 Características adicionales que aportan a la implementación de SOA

Es notable y claramente visible la pertinencia del Bus de Servicio Empresarial en una Arquitectura Orientada a Servicios que pretende ser acoplada a la base tecnológica de una organización -compuesta en muchos de los casos, como se explicó previamente, por sistemas legados-.

Lo cierto es que cada vez son más las organizaciones y empresas que buscan apoyar sus negocios y procesos con dicha tecnología arquitectónica, pero, sin correr riesgos de migración total, que puedan incurrir en perdida de la integridad de la información, perdidas económicas y descontrol de los procesos.

Para que el ESB se convierta en una herramienta útil en pro de lograr el objetivo anterior, debe tener además de las características desglosadas en el capítulo anterior, las siguientes funcionalidades puntuales referentes al manejo de los servicios -los cuales son las unidades funcionales de la Arquitectura Orientada a Servicios-; dichas funcionalidades puntuales se van a fundamentar en los principios básicos del Bus de Servicio Empresarial, son ellas:

**Adaptación del Transporte**

Basándose en la transformación de protocolos, la Adaptación de Transporte, en un ESB basado en SOA, hace especial énfasis en el protocolo de comunicación SOAP, debido a su importancia en las Arquitecturas Orientadas a Servicios.

Como ya se ha puntualizado, en el desarrollo de este trabajo, la heterogeneidad de las aplicaciones es habitual en las organizaciones, dicha diversidad va acompañada de un abanico de variados tipos de protocolos de comunicación, que eventualmente complican la integración a través de un middleware.

**Adaptación de Servicios**

La Adaptación de Servicios guarda relación con la transformación de protocolos y más importante aún con la transformación de la información, más concretamente a los contratos de servicios.

La AS se encarga de aplicar políticas para la relación del contratante del servicio y su proveedor, incluyendo el contrato en sí.

En la Adaptación de Servicios se recibe una solicitud de servicios proveniente de cualquier tipo de solicitante (la implementación de algún método de una clase, un componente EJB, un Objeto Distribuido, Etc.) y se aplican políticas de multiplexación de solicitudes de servicio, activación y desactivación de servicios o hasta gestión de la ejecución de los servicios, son algunas de las funciones importantes que se deben ejercer.

Un método muy útil para la adaptación de servicios es utilizar la arquitectura que se conoce como USPI, por su sigla en inglés que significa *Unified Service Providing Infrastructure*, la cual pretende contribuir a la integración de aplicaciones, mediante la utilización de *plugins* que faciliten el acoplamiento de diferentes tecnologías. El objetivo de USPI es exponer una misma lógica de servicio, vista como si estuviese implementada en varias tecnologías de desarrollo. De tal manera que mediante una semaforización y un enrutamiento especial, se encausan las solicitudes al proveedor apropiado para el mismo.

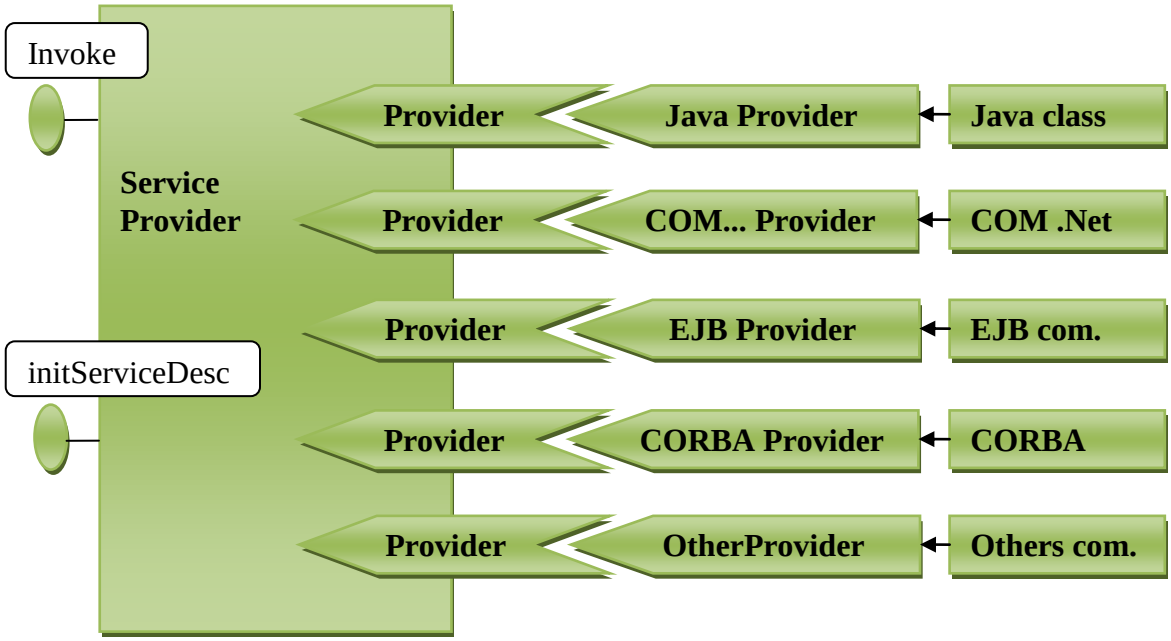


**Ilustración 18 - Unified Service Provider Infrastructure [11]**

### Servicios Comunes

En torno a las funcionalidades explicadas, se cuentan también los servicios necesarios para que dichas características tengan un corriente funcionamiento, en el ESB basado en SOA deben contarse los siguientes servicios básicos o comunes:

- Servicio de seguridad.
- Servicio de despliegue.
- Servicios de transacción.

Estos servicios mencionados son fundamentales y vitales para un correcto manejo de mensajería SOAP, que es manejada por el servicio SOAP; responsable de

dirigir los controladores de otros a su vez. Cuando se realiza una petición a un servicio particular, el motor de adaptación de servicios consulta con el servicio de seguridad y garantiza la operación en el proceso de interacción con el proveedor y el consumidor.

Es una labor importante del Servicio SOAP el desarrollo de un *Scheduling* para la dirección de diferentes solicitudes por parte de los consumidores, porque, una correcta o deficiente programación de solicitudes recibidas, puede impactar a favor o en contra del *performance* del proveedor del servicio. Igualmente, debe mantener un tiempo de vigencia para dicha programación, tal como el tiempo de latencia o espera de respuesta por parte de un servidor en el internet.

El algoritmo de programación y prioridad de solicitudes, dentro de bus de servicio empresarial basado en una Arquitectura Orientada a Servicio, es el siguiente [11]:

```
Algorithm: Priority scheduling Algorithm
 L1  while (true)
     {
L2  done=false;
L3  while (done=false)
     {
          /*  If message queue OM.Qk has some request for waiting schedule, the
          procedure goes to L7 otherwise, SOAP Engine waits request to join  some
          message queue and then sets “done=true”*/
L4        for (k=l;k>0;k--)
          {
               if (QM.Qk.size()>0 ) then
                  break;            }
L5       if (k==0) then wait;
         else done=true;
```

```
L6  }
     /*search the earliest request i from QM.Qk, and then
    extract its priority level p, choose a thread from
    thread pool to execute request */
L7 while (QM.Qk.size() >0)
    {
L8      QM.Qk[i]=QM.Qk.getearliestreq();
L9      p= QM.Qk[i] .getpriority();
L10   thread=threadpool.get(p);
        /*extract the service identifier from QM.Qk[i], and find appropriate service
          adapter to forward request*/
L11    serviceId= QM.Qk[i].getId();
L12   goalAdapter=searchSA(serviceId);
        /*invoke goal service implementation by corresponding Provider plugin,
          then wrap the response into SOAP envelop, remove request message
          QM.Qk[i] from request queue QM.Qk */
L13    If invocable(dest) Then
         {
L14         goalProvider.invoke(QM.Qk[i]);
L15         rply=goalProvider.response();
L16         QM.Qk.remove(i);
              /*construct repose message rqst and extract its priority t, and then join
                      the response message queue*/
L17         rqst=construct(rply, QM.Qk[i]);
L18         t=rqst.getpriority();
L19         QM.add(t, rqst);
L20    } }
```

Cuando se potencializa el ESB agregándole a sus característica fundamentales con las funcionalidades adicionales previamente explicadas, se puede llegar a la

conclusión de que se posee un ESB basado en los principios de la arquitectura SOA, la arquitectura en la cual interactúan las tres funcionalidades previas, dentro del contexto de un Bus de Servicio Empresarial es la siguiente:

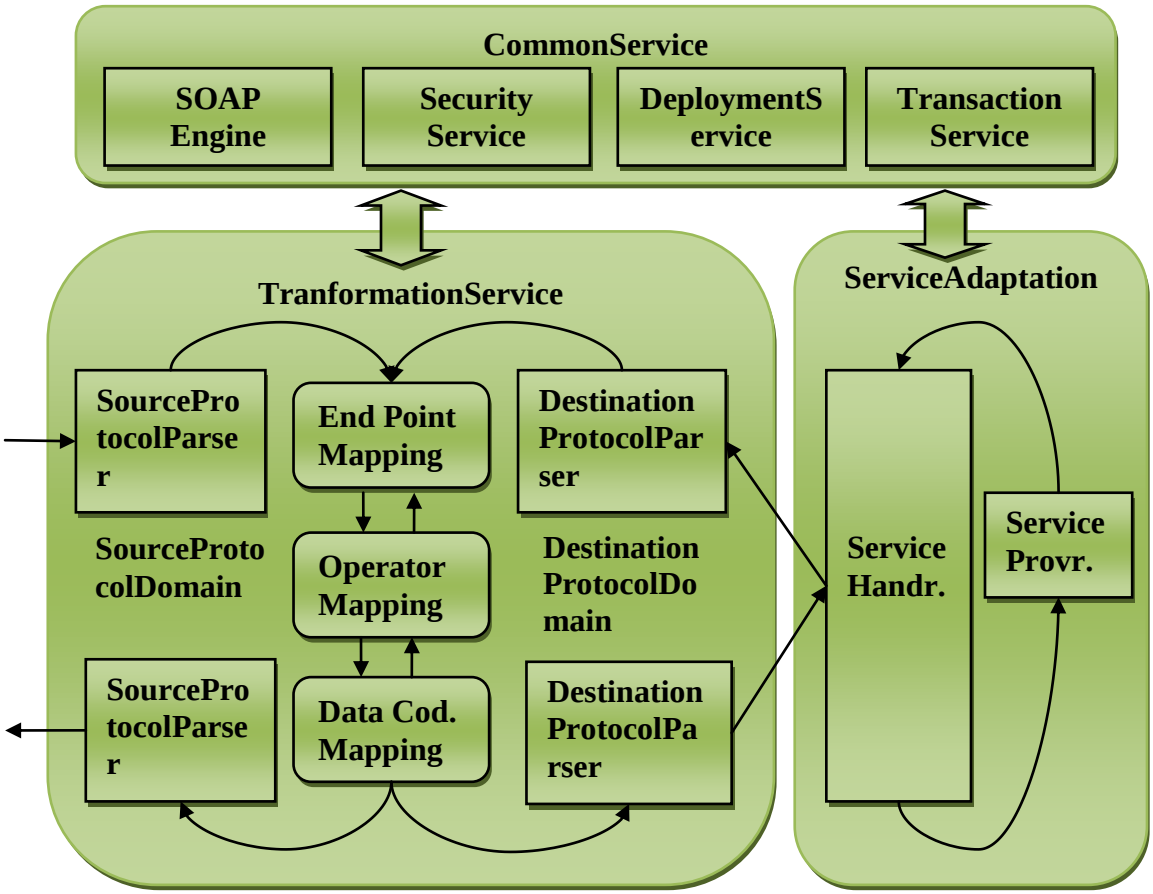


**Ilustración 19 – Una Arquitectura para un ESB basado en SOA [11]**

# 10 PRINCIPIOS DE DISEÑO DE LA ARQUITECTURA ORIENTADA A SERVICIOS VS BUS DE SERVICIO EMPRESARIAL

Como se menciono el capítulos anteriores, el bus de servicios empresarial ofrece un conjunto de ventajas que sin duda apoyan la implementación de una arquitectura orientada a servicios; sin embargo, en algunos sectores del área del desarrollo de software se ha entendido erróneamente que la implementación de un ESB garantiza la implementación de SOA, creando así la idea de que implementar un arquitectura orientada a servicios consiste principalmente en ofrecer servicios centralizados en un canal común independientemente de cómo hayan sido diseñados o desplegados.

El siguiente análisis pretende mostrar hasta qué punto el ESB colabora con la implementación de SOA, por lo cual, es necesario tener un concepto claro de SOA que sea confrontado con los aspectos del ESB.

La mejor forma de entender lo que es SOA son los principios de diseño, porque separan cada uno de los aspectos involucrados en este paradigma y ofrecen una idea clara que permite determinar si una implementación empresarial fue o no orientada a servicios. Valiéndonos de una analogía podemos entender mejor en qué consisten.

En la programación orientada a objetos se hace uso de técnicas que permiten recibir un conjunto de ventajas; de esta forma, se establece un estándar que facilita modelar la realidad, el desarrollo de aplicaciones y las comunicaciones entre los diferentes actores del proceso de desarrollo. No seguir las técnicas que se han establecido, quiere decir que no se está usando el paradigma, por tanto, no se recibirán las ventajas que esta puede ofrecer. En muchos proyectos de

*software* se cree que utilizar tecnologías de apoyo garantiza que se programa con orientación a objetos, por ejemplo, si se utiliza un lenguaje como Java o C#, no quiere decir que se está siguiendo el paradigma, puesto que, perfectamente el producto final no tendrá ninguna diferencia de una aplicación con enfoque procedimental. Es decir, que la condición necesaria para usar el paradigma es conocer sus técnicas y aplicarlas al momento del desarrollo, lo que requiere de un entrenamiento en estas de todo el equipo de desarrollo. Por su parte, los principios de diseño (que no son guías puntuales, sino enfoques que encausan el proceso de diseño: caracterizado por el uso de creatividad, experiencia y estándares) no operan a nivel de programación, sino, a nivel de arquitectura y pretenden brindar un conjunto de ventajas en la construcción de *software* a nivel empresarial; estos principios son producto de distintas propuestas, realizadas por participantes de la industria del software y expertos en el tema, que han sido probadas, discutidas y revisadas.

Encontraremos mucha literatura acerca de los principios de diseño de SOA, dada su naturaleza son un tema del que se ha hablado mucho, sin duda, su importancia se debe a que son el eje de SOA. Grupos como Gartner, autores como Tomas Erl, compañías como Microsoft, Oracle, OASIS e IBM han definido en distintos artículos y libros estos principios de diseño. Uno de los que teóricamente más ha profundizado en el tema ha sido Tomas Erl quien en su libro *SOA Principles of Services Desing* [4] realiza una explicación detallada de cada uno de los principios del diseño de servicios, además de definir dónde y cómo se cumple con cada uno de los principios de diseño. Por este motivo, la principal fuente de información con respecto a los principios será el libro antes mencionado. A continuación plantearemos la forma en que haremos el análisis propósito de este capítulo.

Confrontaremos los principios de diseño de SOA con las características y virtudes del ESB, con el fin de facilitar este análisis se hará uso de la técnica de captura de

requerimientos y se partirá del supuesto que se desea obtener los requerimientos de una aplicación que puede garantizar el cumplimiento todos estos principios, para luego, verificar como el ESB contribuye, satisface o simplemente no cubre dicho requerimiento. Por tanto, el primer paso es hacer una descripción de los principios de diseño, para luego plantearlos como requerimientos. Más adelante se realizará un compilado de los resultados obtenidos.

## 10.1 Los principios de diseño de SOA como requerimientos

Según [4] los principios de SOA son:

- Estandarización en el contrato de servicios
- Bajo acoplamiento
- Abstracción de Información
- Reutilización de servicios
- Autonomía
- Carencia de Estado
- Descubrimiento
- Composición

Existen otras concepciones respecto a los principios de diseño de SOA, donde son reducidos a cinco principios, sin embargo, Erl al ser más detallado permite enfocarse puntualmente en cada aspecto lo que resulta muy útil en esta aproximación.

**Estandarización del contrato de servicios**

El contrato de servicios tiene como principal función describir apropiadamente un servicio, permitiendo conocer que hará exactamente el servicio (predictibilidad) y que necesita para hacerlo; logrando de esta forma encapsular la lógica e implementación subyacente.

Con la descripción de un servicio se pretende que el servicio sea interoperable, lo que significa que pueda ser fácilmente entendido por otras aplicaciones que necesiten consumirlo; para esto, necesariamente se requiere de una estandarización en la que se defina la forma en que será descrito un servicio y por tanto.

El contrato de servicios por tanto contará con un conjunto de documentos que describirán el servicio en dos niveles, por un lado está el nivel técnico y por el otro el nivel no técnico.

El nivel técnico tiene como finalidad contribuir con la automatización del proceso de consumo del servicio, es decir esta documentación tiene como objetivo que sea entendida por procesos automáticos que interpreten la información que se suministre y permitan el consumo del servicio.

Por su parte la documentación no técnica facilitara a los distintos participantes del proceso de desarrollo de software entender apropiadamente el servicio y poder usarlo en la construcción de los requerimientos necesarios para modelar los procesos de negocio de la empresa.

**Como Requerimientos**

Los siguientes son los requerimientos que plantea la estandarización de servicios a una aplicación que garantice el cumplimiento de SOA.

**R.1** Se requiere que los contratos de servicios de un mismo inventario estén estandarizados permitiendo la interoperabilidad de los servicios y la predictibilidad de los mismos; permitiendo la automatización de su consumo, así como también el fácil entendimiento por parte de los participantes del proceso de negocio.

**R.1.1** Se debe permitir que se definan los documentos técnicos necesarios para que el servicio sea consumido, si bien dichos documentos podrían cumplir distintos estándares se recomienda la utilización de los estándares definidos por la industria para los servicios web: *WSDL*, *XML Schema*, *WS-Policy Description*.

**R.1.1.1** Se requiere que se puedan definir las funcionalidades que brindará un servicio web, de forma que estas puedan ser consumidas y además describir cómo se debe utilizar el servicio. Puesto que en la actualidad se ha definido como un estándar el WSDL se debe satisfacer este requerimiento haciendo uso de dicho estándar.

**R.1.1.2** Se debe permitir que se definan los documentos y esquema que permitan estandarizar los datos que serán usados en las transacciones de información de los servicios web, el lenguaje *XML Schema* ha sido posicionado por la industria como estándar para la definición de datos sobre otros lenguajes como las DTD, por tanto se requiere que la definición se realice usando este estándar.

**R.1.1.3** Se requiere que se puedan definir esquemas de datos centralizados que sean usados por los diferentes servicios que referencien un mismo tipo de dato.

**R.1.1.4** Se requiere que se permita definir para un servicio web sus políticas (seguridad, calidad del servicio, disponibilidad, etc.); preferiblemente mediante el estándar *WS-Police.*

**R.1.2** Se debe permitir que se definan los documentos no técnicos, que admitan describir el servicio, de forma que sean fácilmente entendidos por los desarrolladores.

**R.1.3.** Se debe permitir que se definan estándares específicos adecuados al inventario de servicios, de forma que no solo se cumpla con la formalización establecida para la descripción de servicios sino que además se establezca una estandarización en las descripciones realizadas, lo que incluye uso de palabras comunes para un mismo tipo de funcionalidad, sufijos dependiendo de la funcionalidad.

**Bajo acoplamiento de servicios**

Desde que se construyen aplicaciones de software se ha tomado como principio dividir el problema en partes funcionales, al hacer esto el problema se hace menos complejo y se pueden dar soluciones a cada una de estas subdivisiones; luego al acoplarlas se da solución al problema en general. Este acoplamiento compromete por tanto funcionalmente la aplicación que se está construyendo, sin embargo, no es recomendable que exista demasiada dependencia entre las partes que están constituyendo una solución, debido a que se pueden presentar problemas al momento de realizar un cambio en uno de esos componentes.

En el caso de los sistemas distribuidos el problema es aún mayor, porque cada una de las partes de la solución está distribuida en distintas computadoras, esto posiblemente incluya características diferentes en cada uno de los componentes que conforman esa solución. Si a lo anterior adicionamos los cambios tecnológicos

(lenguajes de programación, bases de datos, sistemas operativos, protocolos de comunicación) que pueden presentarse a través del tiempo; la comunicación con sistemas externos y la adición de nuevas soluciones con plataformas en bajo nivel o de sistemas heredados, el problema es aun mayor; porque, un cambio mínimo podría resultar en un gran impacto a través de toda la aplicación construida. SOA pretende solucionar esto enfatizando en el bajo acoplamiento, para esto postula principalmente que los servicios coloquen requerimientos mínimos a sus posibles consumidores y a su vez ellos se encuentren desacoplados de su entorno.

El punto clave de este desacoplamiento está en el contrato de servicios, puesto que, este es una interfaz que oculta los detalles internos de implementación y hace visible solamente las funcionalidades que se ha determinado que comprometen al mínimo al consumidor.

**Como requerimientos**

**R.2** Se requiere que los servicios sean independientes de su entorno y a su vez impongan pocos requerimientos a sus consumidores.

**R.2.1** Se requiere de un ambiente que permita desplegar servicios mínimamente desacoplados funcionalmente, de forma que se haga evidente la dependencia funcional entre cualquier de estos.

**R.2.1.1** Se requiere que no exista dependencia entre un proceso de negocio y las capacidades de un servicio, un servicio debe brindar una funcionalidad global que responda a diferentes puntos de la empresa y no exclusivamente a un proceso de negocio. Cabe señalar que posiblemente existan servicios que inicialmente solo respondan a un proceso de negocio, pero, posiblemente por los cambios que se presenten requieren ser usados en otro tipo de proceso y deben ser diseñados teniendo en cuenta este aspecto.

**R.2.1.2** Se requiere que un servicio no sea diseñado a partir de su consumidor.

**R.2.1.3** Se requiere que se soporte el diseño de servicios (tarea que cumplen funciones específicas de un cliente) y que por tanto dependan funcionalmente de otros servicios y además limiten su alcance al cumplimiento de esta tarea. Se debe poder identificar y ser validados de forma que no incluyan elementos que hagan parte necesaria del inventario de servicios principal.

**R.2.2** Se requiere que la interfaz que encapsula al servicio esta desacoplada de las características internas del servicio, de forma que solo se hagan visibles los aspectos que ofrece funcionalmente un servicio y que esta interfaz no involucre aspectos de implementación, tecnología o lógica.

**R.2.2.1** Se requiere que el contrato de servicios no incluya aspectos específicos de la tecnología (lenguaje programación, detalles de un *framework*, sistema operativo), evitando que se impongan requerimientos de una tecnología específica a un consumidor.

**R.2.2.2** Se requiere que el contrato de servicios no incluya aspectos propios de la implementación (bases de datos, APIS de sistemas legados, cuentas de usuario, nombres de archivo y rutas).

**R.2.2.3** Se requiere que la lógica resolutiva usada en el desarrollo de un servicio no impacte en las características de su contrato de servicios y que solamente se muestren las capacidades de este.

**R.2.3** Se requiere que solo se permita un tipo de acoplamiento y es el de la lógica al contrato, la lógica de implementación debe ser dependiente de lo que brinda el

servicio de forma que si se presenta un cambio en cómo opera el servicio no afecte el contrato y así reduzca el impacto.

**R.2.3.1** Se requiere que se soporte el enfoque contrato primero, de forma que exista dependencia entre de la lógica al contrato.

**R.2.3.2** Se requiere que se evite el uso de la generación del contrato de servicios a partir de la lógica.

**R.2.4** Se requiere que se soporten los estándares WS -, de forma que los requerimientos impuestos a los consumidores dependan únicamente de estos estándares.

**Abstracción**

El concepto de abstracción se basa en ocultar o hacer trasparente información que no es requerida para que un proceso sea usado. De esta forma se puede hacer uso de una funcionalidad sin conocer en detalle en qué consiste o como funciona un proceso o programa.

En la arquitectura orientada a servicios la abstracción se hace gracias a el contrato de servicios quien se encargan de mostrar solo la información necesaria para que el servicio sea efectivamente usado, evitando que se muestren detalles de la implementación y la tecnología subyacente que podrían influir en el diseño de los consumidores.

La importancia de este principio radica en que permite gracias una interfaz ocultar información que puede ser usada como un elemento a considerar al momento de realizar un diseño, conduciendo a que se tomen decisiones en la construcción de servicios que afecten la reutilización de este. Si bien no abstraer información o

detalles de implementación, plataforma tecnológica o lógica; no interfieren en el funcionamiento de una aplicación e incluso pueda otorgar ventajas a nivel de rendimiento, si colocan puntos susceptibles de generar un gran impacto en una modificación, produciendo problemas al momento de realizar cambios, lo que disminuiría la flexibilidad que precisamente es el gran diferenciador de SOA frente a paradigmas arquitectónicos tradicionales.

**Como Requerimientos**

**R.3** Se requiere que los servicios muestren solo la información necesaria y que esta sea mostrada mediante el contrato de servicios.

**R.3.1** Se requiere que el contrato de servicios oculte la información acerca de la tecnología (lenguajes, librerías, *framework*, *plugins*, lenguaje de base de datos) subyacente, detalles de implementación de un servicio (rutas, ip, bases de datos, archivos), lógica utilizada (algoritmos, estructuras de datos internas) y detalles funcionales (funcionalidades internas usadas para apoyar a las funcionalidades que son mostradas). Teniendo en cuenta las características de SOA se hace necesario que se muestre la información relacionada con los estándares WS-.

**R.3.2** El contrato de servicios debe definir los requerimientos de interacción y las restricciones para el consumo de un servicio.

**R.3.3** Se requiere que solo sea mostrada la información que se brinda mediante el contrato de servicio.

**R.3.4** La información acerca de las restricciones técnicas necesarias para consumir un servicio, deben ser especificadas mediante políticas, a través del documento *WS-Policy*, que hace parte del contrato de servicios. Se hace

necesario soportar también documentación no técnica referente a estas restricciones técnicas.

**R.3.5** Se requiere de un sistema que permita definir los niveles de abstracción de un contrato de servicios, de forma que se hagan evidentes los casos donde se requiere la intervención para disminuir el exceso de información mostrada.

**R.3.6** Se requiere de un sistema de control de acceso para la modificación de los servicios, este control de acceso se hace a nivel organizacional, de forma que solamente tengan acceso a los detalles internos de los servicios las personas encargados de estos, llevando de esta forma la abstracción al nivel del equipo de desarrollo.

**Reutilización de servicios**

Este principio es parte fundamental en la estrategia SOA, su objetivo principal es que las funcionalidades desarrollas sean útiles en distintos ámbitos de una organización, evitando la redundancia de lógica o en otras palabras centralizándola de tal forma que estas funcionalidades puedan ser usadas por distintos sectores de la misma empresa(sin que exista un servicio específico para cada ámbito) y que además sean usadas a través del tiempo gracias a los nuevos requerimientos sin que sea necesario adaptarlas o modificarlas.

Para que se cumpla con este principio de diseño se requiere en gran parte de la aplicación de los otros principios de diseño; esto se debe principalmente a la doble naturaleza de la reutilización, que es tanto un principio de diseño como una de las ventajas de SOA que se traduce directamente en menos tiempos de desarrollo, mayor flexibilidad, rápida respuesta a cambios y mayor retorno de la inversión. Es importante también señalar la importancia fundamental de una característica de los servicios web en miras de cumplir con la reutilización: los servicios deben ser

agnósticos, lo que significa que deben cumplir un objetivo por ellos mismos, sin que sea necesario el uso de otros servicios; es decir un servicio debe ofrecer un conjunto de funcionalidades concretas y diferenciadas claramente de los otros, de forma que se haga innecesario el uso de varios para cumplir un objetivo o que existan dos servicios con funcionalidades similares. Entre mayor sea el número de servicios agnósticos de un inventario de servicios mayor será la tasa de reutilización de dicho inventario.

**Como requerimientos**

**R.4** Se requiere que los servicios desarrollados sean agnósticos y puedan ser reutilizados en todos los ámbitos en los que sean necesarios, sin que sea necesario repetir lógica en para distintos casos.

**R.4.1** Se requiere que los servicios sean desarrollados sean agnósticos.

**R.4.2** Se requiere que los servicios tengan altas tasas de reutilización, por tanto se hace necesario que se mida su nivel de reutilización.

**R.4.3** Se requiere evitar que se desarrolle lógica repetida, en un inventario de servicios se debe poder encontrar servicios con lógica parecida y a su vez evitar que se desarrollen nuevos servicios que posean lógica existente.

**R.4.4** Se requiere que la lógica desarrollada se centralice en un servicio, teniendo un único punto de acceso y una sola forma de ser consumido.

**Autonomía de los servicios**

La autonomía representa la característica de los servicios para funcionar con independencia de su entorno, es decir que puede ofrecer sus funcionalidades sin la necesidad de elementos externos, su lógica resolutiva se basa por tanto en

elementos referenciales dentro del mismo ámbito del servicio, aumentando de esta forma la fiabilidad y predictibilidad de la solución. A medida que un servicio tenga menos dependencias exteriores será más fácil saber cómo funciona.

Aplicar este principio lleva en primera medida a velar por la independencia de cada servicio desarrollado, de forma que se vea reflejada en su mayor medida en los servicios que necesariamente deben compartir autonomía, a lo que se hace referencia es a los servicios diseñados como controladores, que tienen como propósito componer servicios y los cuales necesariamente dependen de otros servicios(comparten autonomía); sin embargo al garantizar la independencia de cada uno de los servicios individuales se aumenta los niveles de autonomía de estos servicios, por lo que se hace fundamental este principio entre más cercanos se esté al moldeamiento de alto nivel que corresponde a los procesos de negocio.

**Como requerimientos**

**R.5** Se requiere que los servicios desarrollados tengan un grado alto de control sobre su entorno y sus recursos.

**R.5.1** Se requiere que la lógica de los servicios desarrollados esté aislada del resto de servicios del inventario de servicios.

**R.5.2** se requiere que la lógica de los servicios desarrollado este aislada de elementos y detalles de implementación.

**R.5.3** El contrato de servicios debe expresar las funcionalidades ofrecidas por un servicio de forma tal que el alcance funcional de un servicio no se intercepte con el alcance funcional ofrecido en otro punto.

**R.5.3.1** El contrato de servicios debe definir cada uno de los servicios, sin dar lugar a intercepciones de los alcance funcionales de cada servicio, gracias a que se establece claramente sus límites y alcance.

**R.5.3.2** El contrato de servicios debe definir cada uno de las funciones ofrecidas por un servicio de forma que entre ellas no sobrepongan su alcance funcional.

**R.5.4** Se requiere que un sistema que aislé la dependencia propia de los sistemas heredados, de forma que exista una autonomía funcional entre estos, pero que no se extienda al resto del inventario de servicios.

**R.5.5** Se requiere de un mecanismo que permita identificar los servicios que serán autónomos parcialmente, puesto que deberán definir elementos que necesariamente serán compartidos y que están relacionados principalmente por los servicios que usan recursos (bases de datos, directorios).

**R.5.6** Se requiere que los servicios posean autonomía de los elementos técnicos o ambiente ejecución, de forma que se responda de forma rápida al aumento de demanda de uso, los servicios sean altamente portables en caso de que se requiera cambiar el entorno donde han sido desplegados o que se requiera modificar el tipo de tecnología sobre el que se están ejecutando.

**Carencia de estado**

Administrar información acerca del estado de un servicio pone en peligro la reutilización y escalabilidad de este. Sin embargo, es importante aclarar el concepto de estado para entonces entender el principio de diseño.

**Estado o carencia de estado**

Cuando se automatiza una tarea, se requieren del procesamiento de datos para responder efectivamente a la tarea, estos datos que son procesados se conocen

como estado o datos de estado. Cuando los datos que están siendo procesados están siendo almacenados en variables usadas en el mismo servicio se considera que el proceso "con estado", cuando no lo está haciendo se consideran sin estado o carentes de estado.

Los datos de estado pueden clasificarse en distintos tipos dependiendo de su naturaleza, es así como se tienen datos de sección, datos de contexto y datos de negocio.

Los datos de sección se refieren a elementos necesario para establecer la conexión entre un programa cliente (dentro de los cuales están incluidos los datos de un usuario) y el servidor. Por su parte los datos de contexto son los datos que posee un proceso y que son necesarios para que otro proceso funcione adecuadamente. Por último los datos de negocio son los datos que necesitan ser procesados para responder a las exigencias del negocio, generalmente se refiere a la información persistente que va ser procesada.

La orientación a servicio busca eliminar el manejo de los datos de estado, para esto toma estrategias de delegación de manejo de estado, que consiste principalmente en otorgar el manejo de los datos de estado a otros elementos (en especial el uso de persistencia) permitiendo así que la lógica solo procese los datos y no los almacene, disminuyendo así el consumo de memoria de un servicio (en especial cuando el servicio es consumido concurrentemente).

**Como requerimientos**

**R.6** Los servicios no deben mantener ningún tipo de información de estado.

**R.6.1** Los datos del negocio deben fluir a través de los servicios mediante mensajes SOAP.

**R.6.2** La información de estado no debe almacenarse en variables dentro del servicio.

**R.6.3** Se debe delegar el manejo de sesión de los servicios.

**R.6.4** Se deben definir servicios encargados de delegar el manejo de estado a entidades persistentes.

**Descubrimiento de servicios**

Dado que los servicios son un conjunto de funcionalidades que están enfocadas en satisfacer los procesos de negocio una empresa de forma que se cubran funcionalidades actuales y futuras de forma centralizada, serán necesario mecanismos que permitan encontrar, saber que hacen y cómo se consume dichas funcionalidades. Con descubrimientos se hace referencia a que se debe saber en un momento determinado si existe en el inventario de servicios una funcionalidad que satisface los requerimientos que se han planteado para una aplicación de la empresa.

Para esto principalmente se requiere del uso de metadatos que permitan: encontrar mediante un sistema de búsqueda la funcionalidad que cubre un requerimiento, interpretar el servicio (que hace y como se consume) y saber si además cumple con los requerimientos no funcionales.

**Como requerimientos:**

**R.7** Se deben brindar los mecanismos para que los servicios posean la meta información necesaria para ser descubiertos.

**R.7.1** La información suministrada debe expresar correctamente las funcionalidades que está describiendo.

**R.7.1.1** Se requiere de un conjunto de estándares de descripción, que permitan unificar conceptos de forma que se facilite la interpretación de las funcionalidades.

**R.7.1.2** Se requiere que los servicios sean descritos por personas con capacidades comunicativas, y que además sean revisados por los arquitectos o encargados de forma que certifiquen que describen apropiadamente la funcionalidad.

**R.7.2** La meta información debe estar centralizada, de forma que exista un único acceso a la información y que además sea consistente con el inventario de servicios.

**R.7.3** Debe existir un mecanismo que permita la realización de búsquedas usando como referencia la meta información.

**R.7.4** La información debe suministrar datos acerca del propósito, las capacidades y las limitaciones del recurso que están describiendo.

**R.7.5** Se debe poder encontrar recursos que cubran de forma aproximada el requerimiento por el cual se está buscando la información.

**R.7.6** Para la creación de los documentos de descripción se deben crear guías que faciliten la construcción de dicha meta información.

**Composición de servicios**

Para brindar una solución informática es ineludible recurrir a la división de un problema complejo en partes menos complejas, para luego unir funcionalmente

estas partes y resolver el problema complejo. Pero para que este principio básico sea efectivo es necesario que dichas partes puedan componerse y así brindar la solución requerida. La composición de servicios es la capacidad intrínseca que tienen los servicios para brindar soluciones trabajando en conjunto y la que tienen un conjunto de servicios específicos y conocidos como controladores para componer un conjunto de servicios, generalmente respondiendo a un proceso de negocio de la organización.

La composición a su vez está relacionada con la capacidad de reutilización, puesto que muchas de las divisiones que se realizan de un problema complejo no solo solucionan una parte de este problema sino que también puede funcionar en ambos caso; sin embargo esta posibilidad de reutilización generalmente no surge de manera espontánea, sino que es necesario identificar cuando un servicio puede ser utilizado en campos diferentes y dotarlo de los elementos necesarios para que pueda hacer parte de las distintas composiciones, y no necesariamente las actuales sino que tenga muchas posibilidades de ser utilizados sin modificaciones por los servicios que sean creados posteriormente. Por tanto, la composición principalmente apunta a un doble propósito, que los servicios puedan componerse y que puedan componerse con la mayoría de servicios posibles.

**Como requerimientos**

**R.8** Se requiere que los servicios puedan componerse y brindar soluciones para problemas complejos.

**R.8.1** Se requiere que se centralicen los servicios de forma que estén disponibles en un ambiente donde sea fácil localizarlos y referenciarlos.

**R.8.2** Se requiere que los servicios cumplan estándares de comunicación de forma que no presenten impedimentos al momento de ser compuestos.

**R.8.3** Se requiere que el contrato de servicios defina los elementos necesarios para que un servicio pueda ser usado en respuesta a diferentes requerimientos y que además funcione correctamente para todos.

**R.8.4** Se requiere de un conjunto de elemento que permitan definir como se orquestara una composición.

**R.8.5** Se necesita que se puedan definir servicios cuya función principal sea la de realizar la composición.

**R.8.5.1** Se requiere que estos servicios funcionen como controladores, es decir configurar la ejecución de servicios.

**R.8.5.2** Se requiere que los servicios controladores respondan a un proceso de negocio específico, haciendo uso de los servicios diseñados para ser compuestos.

**R.8.5.3** Se requiere los controladores sean reusables, sin embargo esta no es una característica fundamental al momento de ser diseñados, en algunos casos incluso podrían solo responder a un solo requerimientos.

**R.8.5.4** Se requiere que se puedan identificar los servicios controladores, debido a que requieren de consideraciones de diseño particulares.

A continuación luego de establecer los requerimientos, mediante una tabla se responde la pregunta ¿Ayuda ESB a cumplir con los requerimientos de los principios de diseño previamente planteados?

| | Ayuda a cumplirlo gracias a su arquitectura | Ayuda gracias a que implementa la arquitectura de SOA | No ayuda pues se trata de requerimientos propios de nivel de diseño o de metodología de desarrollo |
|---|---|---|---|
| R.1 | ✓ | | |
| R.1.1 | | ✓ | |
| R.1.1.1 | | ✓ | |
| R.1.1.2 | | ✓ | |
| R.1.1.3 | | ✓ | |
| R.1.1.4 | | | ✓ |
| R.1.2 | | ✓ | |
| R.1.3 | | | ✓ |
| R.2 | | | ✓ |
| R.2.1 | ✓ | | |
| R.2.1.1 | | | ✓ |
| R.2.1.2 | | | ✓ |
| R.2.1.3 | | | ✓ |
| R.2.2 | | | ✓ |
| R.2.2.1 | | ✓ | |
| R.2.2.2 | | ✓ | |
| R.2.2.3 | | | ✓ |
| R.2.3 | | ✓ | |
| R.2.3.1 | | ✓ | |
| R.2.3.2 | | ✓ | |
| R.2.4 | | | ✓ |
| R.3 | ✓ | | |
| R.3.1 | ✓ | | |
| R.3.2 | ✓ | | |
| R.3.3 | ✓ | | |

| | | | |
|---|---|---|---|
| R.3.4 | | | ✓ |
| R.3.5 | | | ✓ |
| R.3.6 | | | ✓ |
| R.4 | | | ✓ |
| R.4.1 | | | ✓ |
| R.4.2 | | | ✓ |
| R.4.3 | | | ✓ |
| R.4.4 | | | ✓ |
| R.5 | | | ✓ |
| R.5.1 | | | ✓ |
| R.5.2 | | | ✓ |
| R.5.3 | | ✓ | |
| R.5.3.1 | | ✓ | |
| R.5.3.2 | | ✓ | |
| R.5.4 | | | ✓ |
| R.5.5 | | | ✓ |
| R.5.6 | | | ✓ |
| R.6 | | | ✓ |
| R.6.1 | | ✓ | |
| R.6.2 | | | ✓ |
| R.6.3 | | | ✓ |
| R.6.4 | | | ✓ |
| R.7 | | ✓ | |
| R.7.1 | | ✓ | |
| R.7.1.1 | | ✓ | |
| R.7.1.2 | | | ✓ |
| R.7.2 | | | ✓ |
| R.7.3 | | | ✓ |
| R.7.4 | | | ✓ |

| | | | |
|---|---|---|---|
| R.7.5 | | | ✓ |
| R.7.6 | | ✓ | |
| R.8 | | | ✓ |
| R.8.1 | ✓ | | |
| R.8.2 | ✓ | | |
| R.8.3 | | ✓ | |
| R.8.4 | | | ✓ |
| R.8.5 | | | ✓ |
| R.8.5.1 | | | ✓ |
| R.8.5.2 | | | ✓ |
| R.8.5.3 | | | ✓ |
| R.8.5.4 | | | ✓ |

**Ilustración 20 - Tabla Requerimientos de Arquitectura SOA vs ESB**

**Compilación y breve análisis de respuestas obtenidas**

Para destacar se resalta que la comparación se realizó tomando como base las características arquitectónicas como tales del Bus de Servicio, adicionado con las características técnicas de la implementación del ESB que se escogió como la más pertinente de este trabajo, a saber el Open ESB. Lo cual pone de relieve que de realizarse la comparación previa utilizando aspectos técnicos de otra implementación del Bus de Servicio es posible obtener resultados diferentes con igual validez que los que obtuvo el grupo de trabajo.

Asimismo es importante señalar que SOA como tal parte de una filosofía arquitectónica y ESB una tecnología de implementación e integración, por lo tanto muchos de los requerimientos no pueden ser satisfechos por únicamente una sola tecnológica pues una filosofía arquitectónica abarca muchos ámbitos en lo que a desarrollo de software se refiere; en otras palabras el hecho de que el ESB no ayude en muchos de los requerimientos no significa que tiene falencias en sí mismo, sino que no es de su ámbito o pertinencia el requerimiento en cuestión.

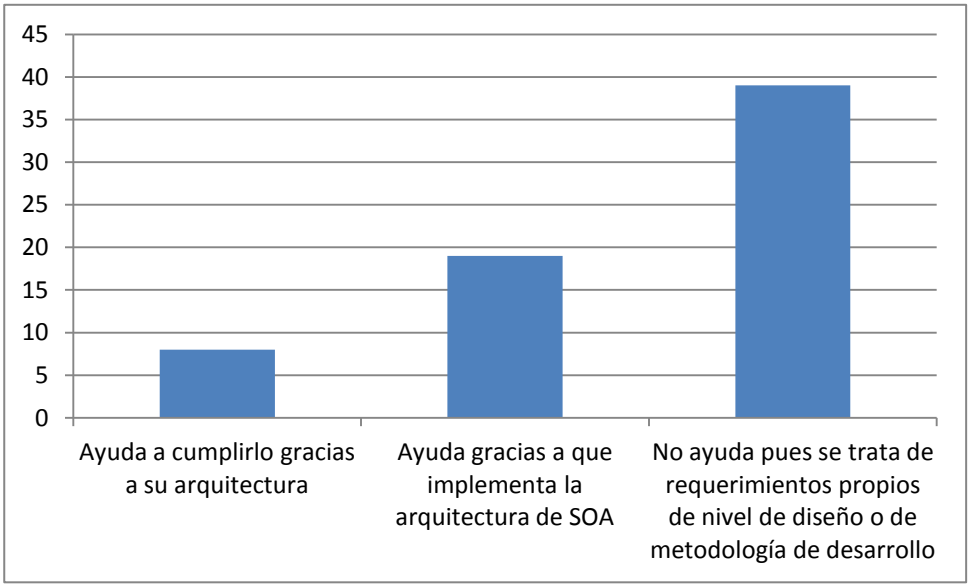


**Ilustración 21- Grafico Número de requerimientos satisfechos VS Evento encontrado**

Para ver gráficos más detallados diríjase al anexo 2.

# 11 CASO DE ESTUDIO

## 11.1 Descripción Breve de la Situación Actual

El caso aplicativo del presente trabajo se enmarca en un proceso de negocio crítico de toda organización de pequeño, mediano o gran tamaño, se hace referencia al proceso de la selección de proveedores, la organización objetivo en este caso es la Universidad Tecnológica de Bolívar.

El proceso de selección de proveedores como tal no es un conjunto de actividades que se realicen dentro de esta organización, sin embargo en búsqueda de darle transparencia a dicha selección se pretende agregar un proceso claro, especifico y bien definido que aplique un método formal de selección de proveedores, basándose en la información contenida de los mismos a través de la historia.

El diseño de estos procesos se basará o guiará por la filosofía de la Arquitectura Orientada a Servicios, mediante dicho desarrollo se busca apalancar este conjunto de actividades y de esa manera aportar al avance tecnológico y organizacional de la Universidad.

En términos generales se propone el desarrollo de los siguientes ejes funcionales:

**Evaluación De Proveedores:** El proceso de evaluación de proveedores en la Universidad Tecnológica de Bolívar inicia una vez se han realizado el proceso de compra, es necesario el desarrollo de un informe de desempeño del proveedor, por parte del jefe de adquisiciones, en conjunto con el solicitante del producto.
Para el desarrollo de ese informe de desempeño, se han definido los criterios y factores más importantes y la ponderación de acuerdo al grado de importancia de

los factores. De acuerdo a estas ponderaciones se califica al proveedor según la orden de compra, como: excelente, bueno, requiere mejorar, no cumple.

Periódicamente se realiza un consolidado de todas las evaluaciones y de acuerdo a la ponderación de las calificaciones obtenidas se obtiene una calificación general que permite determinar si el proveedor es rectificado, si se invita a mejorar o si es desertificado.

**Selección de Proveedores:** El proceso de selección de proveedores propuesto a la Universidad Tecnológica de Bolívar inicia una vez ha surgido una requisición. Previamente se han definido los criterios más importantes y sus respectivos factores y se ha es establecido una escala de calificación de factores de 1-10, en la cual se relacionan la cantidad de encuestas realizadas con la calificación obtenida en la evaluación de los factores realizada a cada proveedor y se establece una puntuación al factor para dar inicio a la aplicación del método.

**Gestión de Proveedores:** El proceso de compras requiere de una buena gestión de los proveedores, pudiendo así tener un contacto más directo con ellos, propuesto a la Universidad Tecnológica de Bolívar. Una vez se ha realizado la selección de los proveedores debe notificarse que han sido seleccionados y se invita a cotizar. Debe tenerse en cuenta el listado de proveedores que han sido seleccionados para la compra de un producto específico, y se deben contactar vía e-mail, fax, etc.

Además se debe gestionar la inscripción y actualización de datos de los proveedores en el listado maestro, un proveedor nuevo debería solicitar la inscripción y una vez sea admitida sus datos serán registrados. En caso de necesitar actualizar su información se debe indicar los datos y se actualizarán en el listado maestro.

Los ejes funcionales previamente expuestos serán desarrollados en forma de servicios valiéndose de tecnologías actuales como son las herramientas .Net para la implementación y despliegue de servicios web.

Actualmente se posee un sistema legado llamado SIFAD que gestiona la base de datos principal en el proceso de compras o adquisiciones, dicha base de datos crítica, se apoya en un motor de igualmente legado, el cual es Informix 7.3**,** la misma se encuentra alojada en un computador central que posee un sistema operativo *stovepipe*, se trata del sistema Unix SCO (Santa Cruz Operation).

Los servicios que se desarrollen requerirán información contenida en la base de datos Informix, y muy probablemente en algún momento se requiera lógica alojada en el sistema SIFAD.

Es claro que el ambiente tecnológico así como los objetivos planteados enmarcan en un claro problema de integración, integración a nivel de aplicaciones e integración a nivel de datos. ¿Por qué?, porque como ya se ha explicado en el desarrollo de este trabajo, no es una opción viable y realista transformar de golpe, toda la plataforma a una Arquitectura Orientada a Servicios.

En realidad se incurren en muchísimos riesgos (pérdidas económicas o perdidas de información) intentar migrar sistemas legados a tecnologías actuales, sin tomar en cuenta la continuidad operativa que deben tener estos sistemas.
¿Cómo se pretende entonces enfrentar esta situación problémica en la cual se desean apalancar procesos con la Orientación a Servicios, pero sin dejar a un lado el sistema legado que como se puede notar es de vital importancia en el negocio?

## 11.2 Descripción Solución General

En búsqueda de aplicar los conceptos y principios obtenidos durante la realización de este documento, y por supuesto bajo la premisa de la creación de un conjunto de buenas prácticas, que sean útiles para integrar correctamente ambientes heterogéneos en términos de aplicaciones, se toma la decisión de valerse del Bus de Servicio Empresarial como tecnología intermediaria entre las tecnologías nuevas y el sistema legado con el cual se cuenta.

Durante el desarrollo de este trabajo se puso de manifiesto que la transformación de mensaje es una características importante y fundamental en la razón de ser del Bus de Servicio, y para hacer palpable dicha importancia de la característica antes mencionada, se pretende usar la habilidad de transformar mensajes disponible en el ESB de tal manera que convierta los resultados de la consulta a la base datos Informix en formato estándar XML (Definiendo esquemas propios mediantes XML *Schema*), para posteriormente enviar la información obtenida y estandarizada a los servicios de nivel superior que ejecutan el proceso de la selección de proveedores y demás actividades.

### 11.3 Requerimientos Funcionales

| | |
|---|---|
| Nombre | R1 - Recepción de solicitudes |
| Resumen | La aplicación de integración soportada en el ESB deberá recibir solicitudes realizadas por servicios que contengan la lógica del proceso de selección de proveedores. Los cuales requerirán los datos alojados en la DB Informix 7.3 |
| Entradas | Parámetros o especificación de datos solicitados para la extracción de la información contenida en la DB. |
| Resultados | Se generan procesos de enrutamiento e invocaciones a servicios y conexiones a base de datos, que en última instancia devuelven al servicio consumidor los datos esperados por el mismo. |

| | |
|---|---|
| Nombre | R2 – Enrutamiento |
| Resumen | El sistema debe ejecutar un *workflow* que dirija la información a través de sus diferentes estaciones. |

| | |
|---|---|
| Entradas | Definición de una ruta o mapa de procesos, valiéndose del BEPL, que dirija la ejecución del enrutamiento |
| Resultados | -- |

| | |
|---|---|
| Nombre | R3 - Extracción de datos |
| Resumen | El sistema debe conectarse con la base de datos Informix y extraer de allí cualquier información pertinente para la ejecución y correcto funcionamiento de los servicios de nivel superior. |
| Entrada | - |
| Salida | Información pertinente de proveedores, evaluaciones, órdenes de compra. etc. |

| | |
|---|---|
| Nombre | R4 - Transformación de datos en formato XML |
| Resumen | El sistema debe procesar la información obtenida de la base de datos y convertirla en formato XML, dándole carácter de estándar, para ser procesado por cualesquiera aplicaciones. |
| Entrada | Conjunto de datos, resultado del *query* sobre *Informix.* |
| Salida | Datos convertidos a formato XML |

## 11.4 Requerimientos No Funcionales

- Para la integración, utilizar Bus de Servicio Empresarial, preferiblemente herramientas de código abierto.

- Los componentes que se identifiquen durante el desarrollo de la solución deben enfocarse a ser desacoplados y en gran medida reutilizables.

- La aplicación desarrollada debe estar habilitada y capaz de aumentar en conexiones con otros servicios y aplicaciones variables (escalabilidad).

Para leer el diseño solución propuesto diríjase al anexo 4.

# 12 BUENAS PRÁCTICAS

## 12.1 Lecciones y sugerencias

Luego de un planteamiento teórico y conceptual del ESB, que ofreciera cimientos sólidos y principios fundamentales para enmarcar el trabajo dentro de un dominio especifico y un problema a solucionar, y luego de comprobar la utilidad del Bus de Servicio, dentro de un ambiente empresarial -mediante un caso de estudio - con una necesidad de integración puntual; se dan las condiciones y requerimientos para tener validez y credibilidad al ofrecer un conjunto de buenas prácticas que sean referente y guía al momento de realizar integración de aplicaciones en un ambiente heterogéneo y con una marcada tendencia a la orientación a servicios.

**Conocer el Negocio**

Entender los procesos que se llevan a cabo en la empresa -en especial aquellos que tienen intervención tecnológica y con gran énfasis en los que guardan mayor relación con la necesidad de integración- es imprescindible y es una labor que se debe realizar antes de cualquier otra actividad.

El comprender bien el funcionamiento de los procesos empresariales permitirá una correcta obtención de requerimientos y también enfocar los esfuerzos en necesidades puntuales de tal manera que la solución diseñada y planteada se implemente en menor tiempo y se disminuye el margen de error.

De igual forma conocer el negocio aportará información respecto a la selección de la herramienta de integración más apropiada, acorde con los objetivos que se detecten.

**Identificar nivel de heterogeneidad**

Realizar desarrollo de software valiéndose del ESB como tecnología integradora, no obedece a una metodología de desarrollo de software ordinaria, por dos razones principales. Primero no se parte de un proyecto vacío donde empiece a implementarse el diseño propuesto, o donde se posea autonomía total y pueda establecerse una arquitectura personalizada. La segunda razón que guarda estrecha relación con la primera, es que ya existen aplicaciones en la vista amplia del sistema. Dichas aplicaciones son diversas entre sí, con su propia lógica y responsabilidad. Destacando que en muchos casos las mismas poseen una connotación de carácter crítico. Por lo tanto, cualquier nuevo desarrollo que se realice o cambio que se implemente, en pro de realizar integración; debe estar en la capacidad de no afectar negativamente el desempeño de las aplicaciones existentes.

En vista de lo anterior; como segunda medida es importante realizar una clara discriminación de todas y cada una de las aplicaciones que conforman el sistema que se pretende integrar, de tal manera que se tenga una visión clara y entendible de cada uno de los ítems o elementos que componen el engranaje tecnológico de la organización o empresa. Es de vital importancia entonces conocer las características de las aplicaciones, su importancia y pertinencia, información de interés para el desarrollo que se encara, etc.

La información que se recopile debe mantenerse disponible. Por lo tanto es de utilidad la creación de tablas formulario que le den persistencia a la información obtenida. (Sugerencia ver Ilustración 25)

| | |
|---|---|
| Id Aplicación | |
| Nombre Aplicación | |

| | |
|---|---|
| Función o Razón de Ser | |
| Lenguaje o Herramienta de Creación | |
| Fecha de Creación | |
| Fecha de Ultima Modificación | |
| Importancia | |
| Observaciones | |

**Ilustración 22 - Tabla Modelo para establecer Información de Aplicaciones**

**Relación Aplicación - Proceso de Negocio**

Para identificar aquellas aplicaciones que potencialmente se puedan ver afectadas o con las cuales se debe mantener una constante interacción así como aquellas con las que se deba tener cuidado y atención; es necesario entonces enmarcar las aplicaciones en cada uno de los procesos con los cuales guarde relación y sea vital para el correcto desarrollo del mismo.

De esa manera una vez que se definan los procesos críticos que requieran verse beneficiados por la integración, también se conocerán cuales aplicaciones son de especial interés en el desarrollo. Una manera de poder encontrar dicha relación es valiéndose de la información que se obtenga al aplicar la buena práctica previa. (Identificar nivel de heterogeneidad).

**Definir el objetivo y enfoque de la integración**

Gracias a las buenas prácticas previas se puede alcanzar un punto de vista más preciso y sesgado de la necesidad, proceso y bosquejo de solución de la situación o problema al cual se le hace frente. El siguiente paso sugerido es definir qué se busca integrar, es decir mediante el análisis previo comenzar a dirigir los pasos al

nivel de diseño y técnico de la solución. De tal manera que se establezca qué caminos de integración se debe seguir. Por ejemplo quizás se detecte necesidad de integrar datos, aplicaciones o procesos, etc. este paso permite saber con mayor claridad, cuál es la herramienta más apropiada a utilizar.

Probablemente se detecte que sea necesario de integrar diferentes aspectos de los sistemas de una misma organización, en tal caso se debe comenzar por la solución más pertinente y de mayor urgencia.

**Tener en cuenta aspectos de SOA no cobijados por ESB**

Teniendo como guía el capitulo numero 9 de esta investigación, se pueden establecer cuales aspectos de la arquitectura orientada a servicios no son cubierta por ESB. Estos aspectos, necesitan de un tratamiento especial que está enfocado en aspectos de diseño y de metodología de desarrollo, los cuales requieren de: capacitación del personal participante en el proceso de desarrollo en aspectos de SOA y ESB; definición de estándares de diseño, nombramiento de objetos y tipos de datos, documentación de código, documentación funcional, realización de pruebas; implementación de herramientas de apoyo (buscadores de servicios, herramientas de pruebas) y la creación de una metodología de desarrollo en la que se definan pasos y roles para los participantes, que eviten que se incurran en violaciones a los principios de diseño de SOA.

**Modelar la solución**

Con una necesidad o necesidades se inicia el camino para modelar la solución que se pretende desarrollar. A partir de dichas necesidades se obtienen una lista de requerimientos que pueden ser el punto de partida para valerse del lenguaje unificado de modelado (UML) como herramienta que permite plasmar el sistema a través de los diferentes diagramas que el lenguaje posee. De esa manera se da

un carácter formal a la solución y contribuye a un desarrollo ordenado, sistemático y en el caso eventual de presentarse fallos, fácil de detectar los mismos.

Es importante señalar que UML no debe ser tomado de manera rígida como el lenguaje para modelar la solución de integración que se afronta, más bien como una herramienta de la cual se pueden extraer diagramas o características útiles para describir algunos componentes de la solución.

Sin embargo, la debilidad de UML al modelar sistemas distribuidos o de la misma naturaleza supone un punto negativo que impide en gran manera fundamentar o cimentar la solución completa y totalmente en el lenguaje de modelado unificado. Algunos de los diagramas que fueron útiles en el presente trabajo así como aquellos que bien podrían ser de utilidad en cualquier situación de integración son:

- Requerimientos caja blanca
- Diagrama de actividades
- Diagrama de paquetes
- Diagrama de despliegue

**Validar pertinencia de ESB**

Una vez que se sabe con claridad que se desea integrar, cada una de las entidades que intervienen, que rol tienen, cuáles son sus características, además teniendo un modelo arquitectónico y una documentación que dirija el desarrollo y estandarice los procesos de creación de la solución, se hace posible detectar la pertinencia o no pertinencia de la utilización de un Bus de Servicio en el proyecto que se afronta.

Como se identificó al contrastar los principios de diseño de la Arquitectura Orientada a Servicios con el ESB, algunos requerimientos que los principios buscan suplir pueden recibir un apoyo sólido en las funcionalidades y virtudes del Bus de Servicio, sin embargo otros principios no guardan relación estrecha con el ESB así que no es pertinente en dicho caso.

Por ejemplo si en la situación problémica real que se afronta se enfoca mucho en cumplir de lleno el principio de diseño de Estandarización del Contrato de Servicio o el Bajo Acoplamiento, el Bus de Servicio se convierte en un fuerte aliado para lograrlo. ¿Por qué?; porque dicha tecnología (en especial la implementación utilizada en el caso de estudio de este trabajo) exige en primera instancia que las actores (aplicaciones, bases de datos, otros servicios, incluso servidores FTP, entre muchos otros actores) que intervengan en algún proceso y se comuniquen mediante el, intercambien información mediante contratos de servicios estándares (XML). De igual forma por la naturaleza misma del Bus de Servicio permite el bajo acoplamiento entre los consumidores, proveedores y demás entidades del sistema.

Por otra parte si por el contrario, el proyecto tiene como fin general apuntarle en gran medida al principio de diseño de Abstracción de la Información o la Reutilización de Servicios. El ESB no tiene aporte significativo así que su pertinencia es muy poca o ninguna. Criterios como los principios de diseño de mayor relevancia en el proyecto, las necesidades puntuales de integración y natural y obviamente la heterogeneidad del ambiente (aplicaciones, sistemas operativos, protocolos, etc.) darán claridad en cuanto a la pertinencia del ESB en la solución a proveer.

**Seleccionar Implementación**

Si se concluye que el Bus de Servicio tiene pertinencia en la solución, se debe a continuación definir cuál implementación del mercado es la más viable, útil y precisa para que se convierta en una herramienta eficaz y no en una carga innecesaria.

Para escoger una implementación adecuada se debe tener en cuenta factores como los siguientes:

- Presupuesto: las soluciones comerciales cuentan con un soporte técnico superior, sin embargo es de seria consideración sopesar la posibilidad de utilizar una implementación de código libre con robustez y escalabilidad.

- Características de la Implementación: dependiendo de qué característica o funcionalidad del ESB se quiere explotar al máximo, puede que se incline más por una opción que por otro. Por ejemplo algunas implementaciones traen un gran soporte en cuanto a transformación de protocolos, pero la transformación de mensajes (por ejemplo basados en XML *Schema*) no viene tan potencializada. Por lo tanto es importante determinar que característica del ESB desea explotarse más.

- Características del Despliegue y Alcance de la solución: Se debe tener en cuenta que la implementación del Bus de Servicio que se seleccione, tenga la robustez y estabilidad suficiente para contener el despliegue de la solución a través de todo el entramado tecnológico de la organización.

Aquellos ESB que son basados en servidores de aplicación son muy recomendables a este respecto, ya que para respaldar sus funcionalidades tiene el apoyo de un servidor de aplicaciones que lo cimienta. Por ejemplo para conexiones con bases de datos, administrar pool de conexiones, solicitudes de

servicio se hace útil un ESB basado en un servidor de aplicaciones (Open ESB se basa en el servidor Glassfish).

# 13 CONCLUSIONES

La existencia de aplicaciones heterogéneas en las organizaciones, hacen necesaria la existencia de herramientas de integración. En un enfoque SOA Ideal, las tecnologías asociadas a los servicios web solucionan este problema (SOAP y WSDL), cubriendo cualquier diferencia de plataforma tecnológica (sistema operativo, lenguaje de programación, servidor de aplicaciones, etc.) pero, lo normal es que no todas las aplicaciones existentes hayan sido construidas con el fin de que su funcionalidad pudiese ser consumidas por cualquier tipo de tecnología, reutilizadas y con un protocolo de comunicación estandarizado. Por lo tanto, es necesaria la utilización de un canal de interacción, el cual brinde diferentes alternativas en cuanto a comunicación y se produzca una traducción de los mensajes producidos por aplicaciones de sistemas legados a los estándares WS-*, que es la tendencia estandarizada de SOA. El ESB evitará que se cree un mecanismo de integración por cada pareja de sistemas internos y externos, además, facilita la integración con nuevas aplicaciones o la sustitución de uno de los existentes.

Las responsabilidades del ESB son enrutamiento de mensajes, transformación de protocolos, transformación de datos o mensajes, modificación de mensajes. Estas responsabilidades garantizan la trasparencia en tres aspectos que son interfaces, protocolos y localización, lo cual significa que la construcción de servicios no implique pensar en los anteriores elementos como elementos que comprometan la funcionalidad. Pero las responsabilidades no son únicamente funcionales, sino, que debido al flujo de información manejado por el ESB, es necesario que se cumplan con algunas características no funcionales.

Los principales requerimientos no funcionales que debe satisfacer un ESB son: confiabilidad, rendimiento y seguridad. Los dos primeros garantizan el funcionamiento normal de las aplicaciones tanto en disponibilidad como en tiempo de respuesta; por su parte, la seguridad pese que no se ofrece como una capa propia del ESB, puede ser ofrecida por elementos adicionales y es recomendable su implementación, puesto que, por tratarse de sistemas distribuidos una vulnerabilidad puede comprometer los datos de la organización.

Pese a lo anterior, si la intensión es que el ESB responda a la implementación de SOA bajo estándares WS-*, se requieren de un conjunto de características adicionales que son: adaptación al transporte, adaptación de servicios, servicios comunes (seguridad, despliegue, transacción, los cuales son necesarios para el funcionamiento del servicio SOAP, lo que incluye el manejo de prioridades en las respuestas a las solicitudes).

Pero, un ESB basado en SOA no garantiza que se implemente SOA, se requiere de una estrategia que esta por fuera del alcance del ESB y que apunta aspectos de la metodología de desarrollo, diseño o uso de estándares. El ESB contribuye con el cumplimiento de algunos principios de diseño de SOA, pero no con todos y no con la gran mayoría, por tanto, realizar una implementación requiere un especial conocimiento técnico y teórico del ESB, pero, también de SOA.

En el caso de estudio se comprobó que efectivamente mediante el ESB, se pueden integrar aplicaciones heterogéneas, puesto que, se comprobó que se pueden ofrecer como servicios datos y funcionalidades, permitiendo que puedan ser utilizados en otras aplicaciones, sin que sea necesario un desarrollo especifico. La estrategia de integración dependerá de las aplicaciones a integrar y el ESB ofrece distintas formas de realizarlo, reduciendo de esta forma las posibilidades de que cualquier aplicación continúe aislada.

# 14 SUGERENCIAS

Como trabajos de investigación propuestos se plantean:

- Aplicación de Gobernabilidad de SOA mediante el ESB.
- Diseño de herramienta de desarrollo que controle y valide el cumplimiento de los principios de diseño mediante ESB.
- Uso del ESB como recurso para implementar SOA 2.0 (EDA).
- Análisis ontológico de mensajes dentro del ESB para aplicar
  - *Business Intelligence* mediante minería de datos.
  - Enrutamiento inteligente de solicitudes.

## 15 REFERENCIAS


[1] MICROSOFT CORPORATION. La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real. Diciembre 2006.

[2] AALBERS, Huibert. Introducción a SOA.

[3] W.T. Tsai; YINONG Chen, Introduction to Service-Oriented Computing. Computer Science and Engineering Department, Arizona State University & Gary Bitter and Dorina Miron.

[4] ERL,Thomas. SOA Principles of Service Design. Prentice Hall. 2008.

[5] AALBERS, Huibert. Introducción a SOA.

[6] HEWIT, Eben. Java SOA Cookbook. pag 670. March 2009.

[7] RUH, William A.; MAGINNIS, Francis X.; BROWN, William J. Enterprise Application Integration A Wiley Tech Brief. Num pags 211. Wiley Computer Publishing. 2001.

[8] JURIC, Matjaz B. LOGANATHAN, Ramesh; SARANG, Poornachandra; JENNINGS, Frank. SOA Aproach to Integration XML, Web services, ESB, and BPEL in real-world SOA projects. Num. pags 366. Packt Publishing 2007.

[9] ROSHEN, Waseem. SOA-Based Enterprise Integration: A Step-by-Step Guide to Services-Based Application Integration. Num. Pags. 364. McGraw-Hill 2009.

[10] CHAPPELL, Dave. Enterprise Service Bus. O'Reilly. June 2004.

[11] HU, Jianqiang; LUO, FengE; Jun Li; TONG, Xin; LIAO, Guiping. SOA-based Enterprise Service Bus, International Symposium on Electronic Commerce and Security.

[12] PUERTA, Edwin. Método de Integración empresarial Orientada a Servicios: Pequeñas y Medianas Empresas. Octubre 2011.

[13] RADEMAKER,Tijs; DIRKSEN, Jos. Open-Source ESBs in Action. Septiembre, 2008.

[14] --------, Why ESB and SOA? Enero 2006.

[15] HASHEM, Mostafa. Handbook of Enterprise Integration. Auerbach Publications, 2010.

[16] --------, Technology Based Learning and Research. College of Education, Arizona State University.

[17] VAISHNAVI, V. K.; KUECHLER, W.J. Design Science Research Methods and Patterns. Taylor & Francis Group. 2008.

[18] MICROSOFT CORPORATION. Introducción a BizTalk Server. (Consultado Noviembre 2010) http://msdn.microsoft.com/es-es/library/aa547058(BTS.10).aspx

[19] World Wide Web Consortium. XML Schema. (Consultado Marzo 2012)http://www.w3.org/2001/XMLSchema

# ANEXOS

### Anexo 1. Comparativa de Implementaciones de ESB Libres

El siguiente esquema comparativo pretende ofrecer una vista breve de los diferentes productos *open source* que se encuentran actualmente en el medio del desarrollo de software. Como foco comparativo primario, se tomó la capacidad para ofrecer funcionalidad con respecto a las características primordiales de un ESB y posterior a ello, las funcionalidades adicionales que posee.

**Jboss**

- Ofrece soporte de múltiples protocolo de transporte: JMS, TCP/IP, email, sistema de ficheros.
- Transformación de datos XML (XSLT, Smooks).
- Enrutamiento de mensajes basado en contenido (Xpath, Drools).
- Cumplimiento del estándar JBI.
- Incluye IDE gráfico.
- Motor de orquestación de servicios – BPEL 2.0.
- Se integra de manera nativa con jBPM y con JBossAS.

**Mule**

- Transformación de datos XML (XSLT)
- Enrutamiento de mensajes basado en contenido (Xpath, JXPath)
- Múltiples protocolos de transporte soportados: JMS, HTTP, email, FTP,…
- Incluye IDE gráfico (MuleIDE, Eclipse).

**Open ESB Glassfish**

- Cumplimiento del estándar JBI
- Múltiples protocolos de transporte soportados: JMS, HTTP, SOAP, REST, FTP, email, sistema de ficheros…
- Incluye IDE gráfico (NetBeans) y una consola de administración vía web
- Motor de orquestación de servicios – BPEL
- Transformación de datos XML (XSLT, TransformXL)
- Enrutamiento de mensajes basado en contenido
- Se integra de manera nativa con Glassfish y/o con JBossAS

**Petals 2.0**

- Cumplimiento del estándar JBI
- Múltiples protocolos de transporte soportados: JMS, HTTP, FTP, sistema de ficheros…
- Incluye IDE gráfico (Eclipse).
- Motor de orquestación de servicios – BPEL.
- Transformación de datos XML (XSLT, XQuery).
- Enrutamiento de mensajes basado en contenido.
- Se integra de manera nativa con Bonita y con la solución eXoPlatform.

**ServiceMix**

- Cumplimiento del estándar JBI.

- Múltiples protocolos de transporte soportados: JMS, HTTP, FTP, sistema de ficheros…
- Dispone de una consola web para tareas de administración aunque próximamente se liberará un IDE gráfico basado en Eclipse.
- Motor de orquestación de servicios – BPEL
- Transformación de datos XML (XSLT, XSLTComponent)
- Enrutamiento de mensajes basado en contenido (Drools)
- Se integra de manera nativa con Apache Geronimo, JBOSS o JOnAS
- WSO2
- Múltiples protocolos de transporte soportados (HTTP, JMS, SMTP) incluyendo protocolos heredados tales como: EDI, CSV o registros
- Ultrarápido
- Incluye IDE gráfico
- Transformación de datos XML (XSLT, XQuery)
- Enrutamiento de mensajes basado en contenido (Xpath)

**Anexo 2. Graficas detalladas Comparación Principios de Diseño SOA vs ESB**

A continuación se muestra el listado de los principios de diseño de SOA con la respectiva grafica que se obtuvo al realizar el análisis comparativo con el Bus de Servicio.

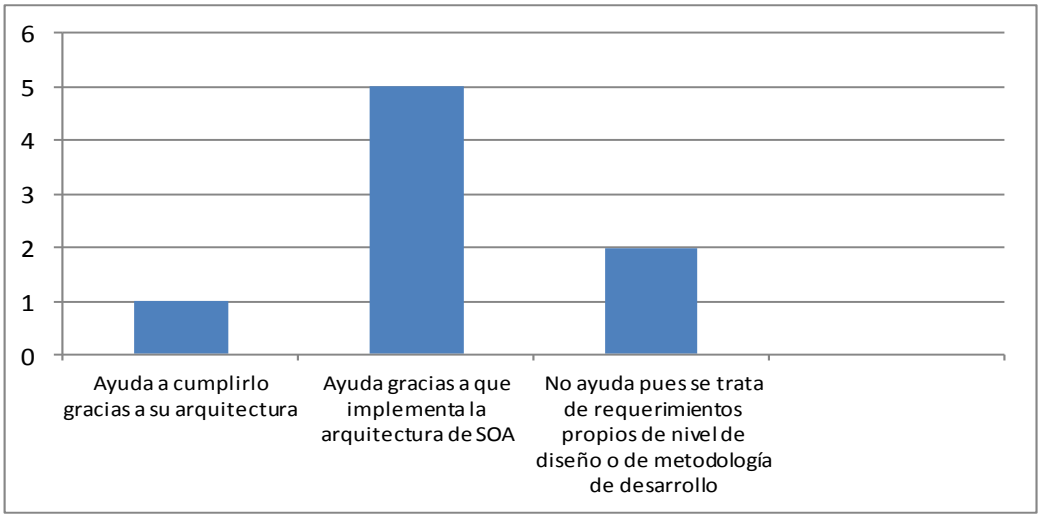


**Ilustración 23 - Grafica Numero de Requerimientos de Principio de Diseño vs ESB para Estandarización en el contrato de servicios**

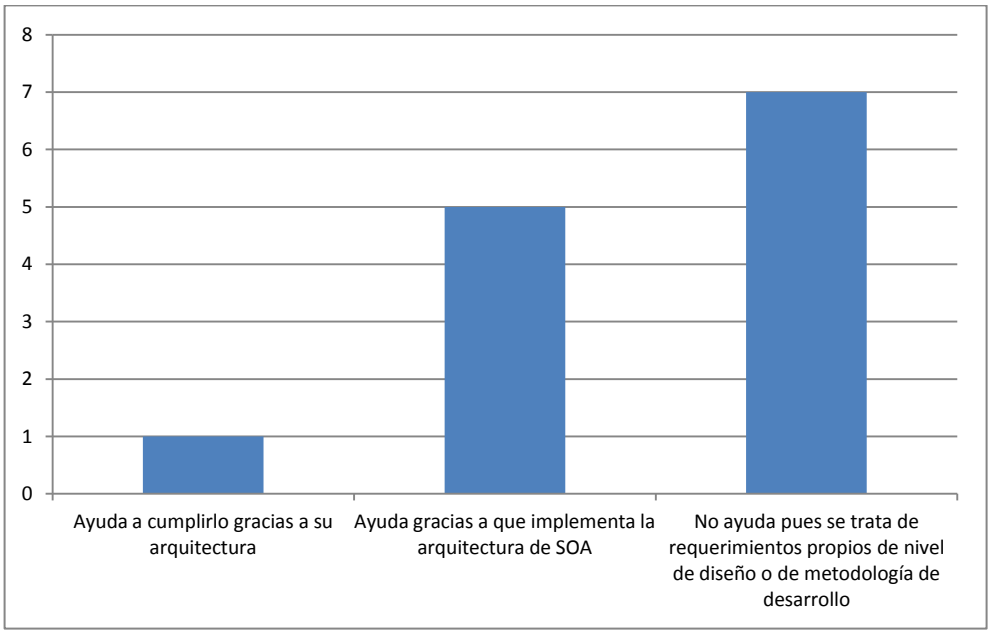


**Ilustración 24 - Grafica Numero de Requerimientos de Principio de Diseño vs ESB para Bajo acoplamiento**

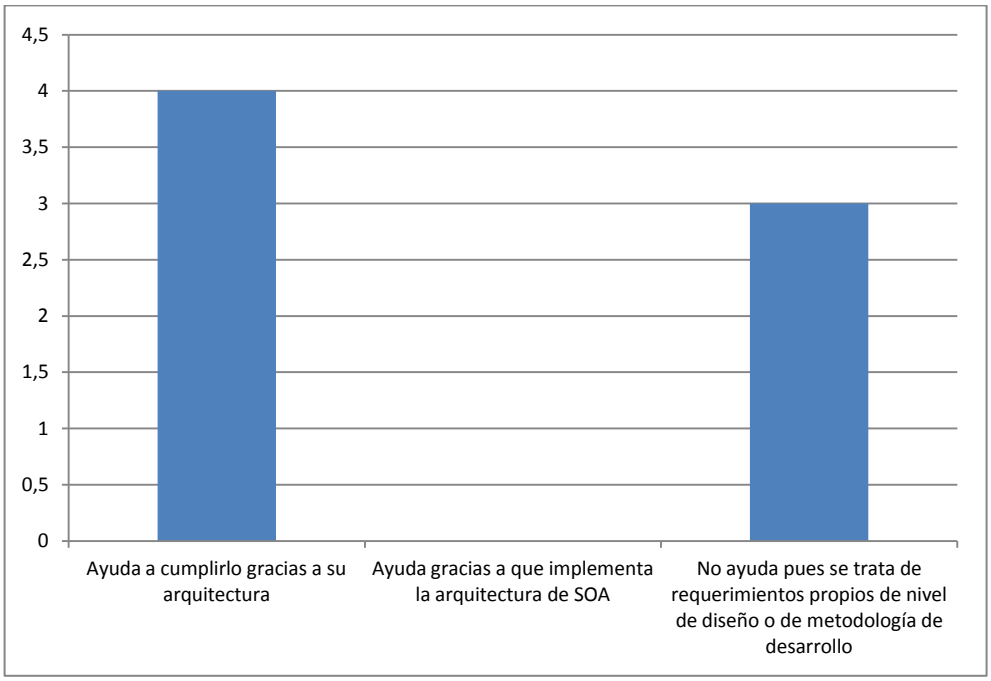


**Ilustración 25 - Grafica Numero de Requerimientos de Principio de Diseño vs ESB para Abstracción de Información**

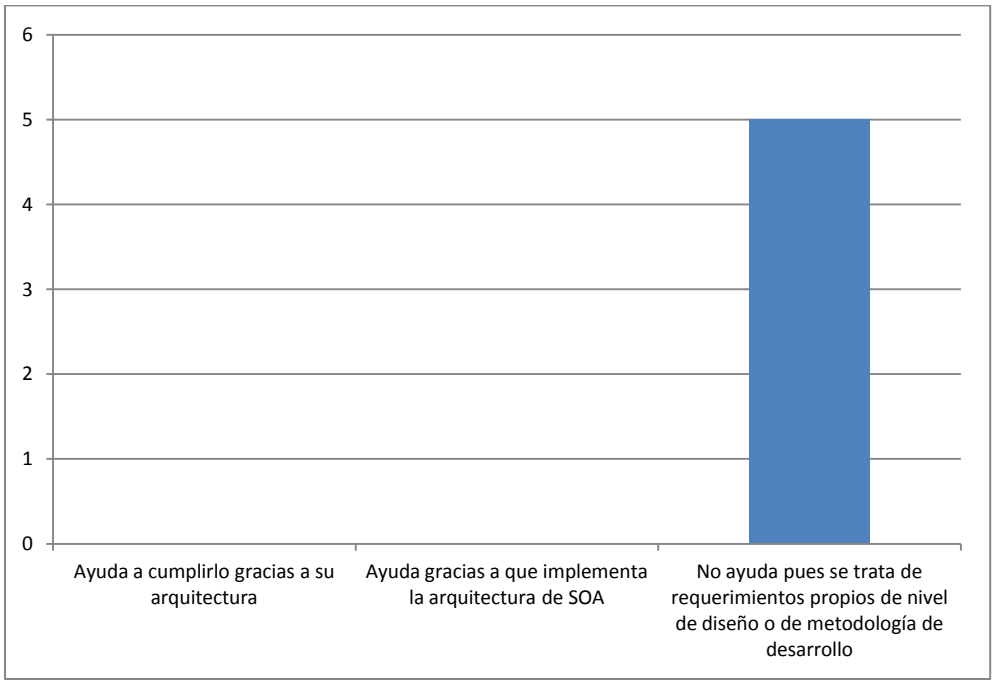


**Ilustración 26 - Grafica Numero de Requerimientos de Principio de Diseño vs ESB para Reutilización de servicios**

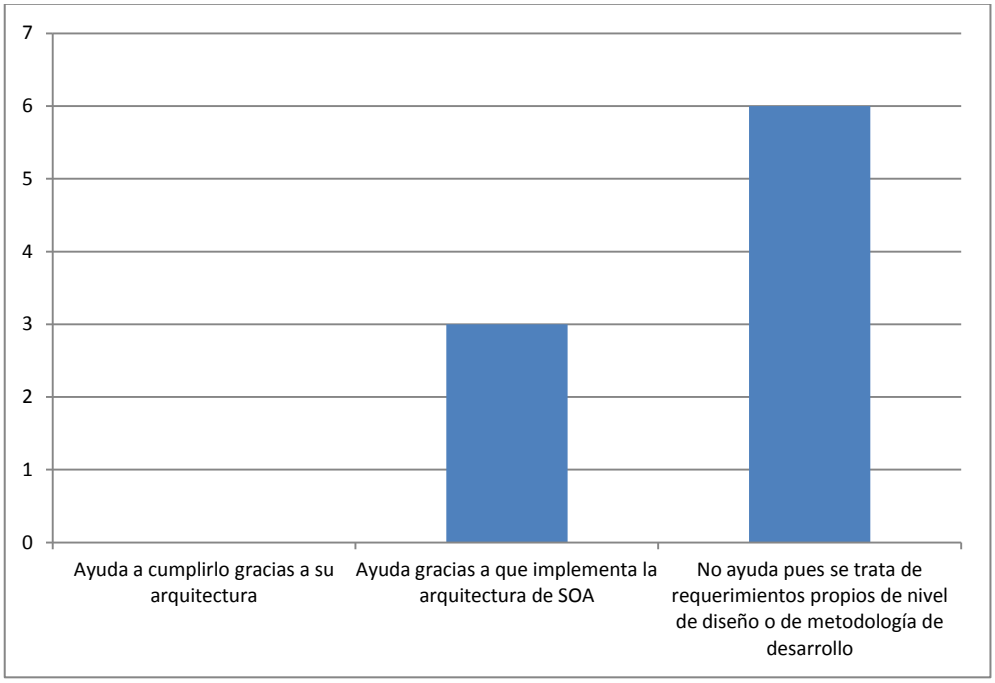


**Ilustración 27 - Grafica Numero de Requerimientos de Principio de Diseño vs ESB para Autonomía**

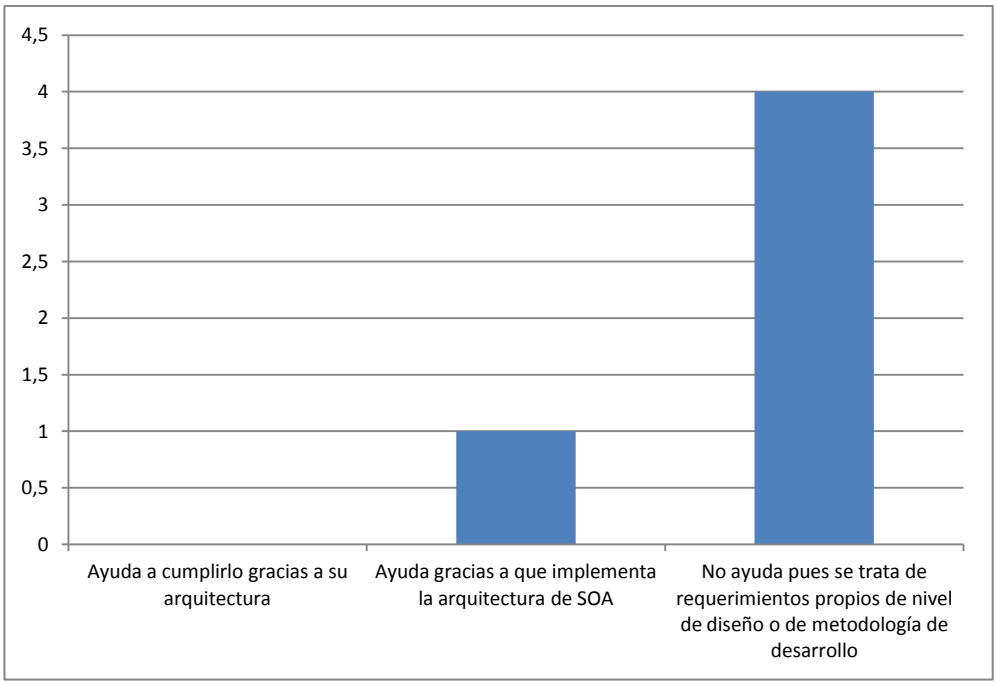


**Ilustración 28 - Grafica Número de Requerimientos de Principio de Diseño vs ESB para Carencia de Estado**

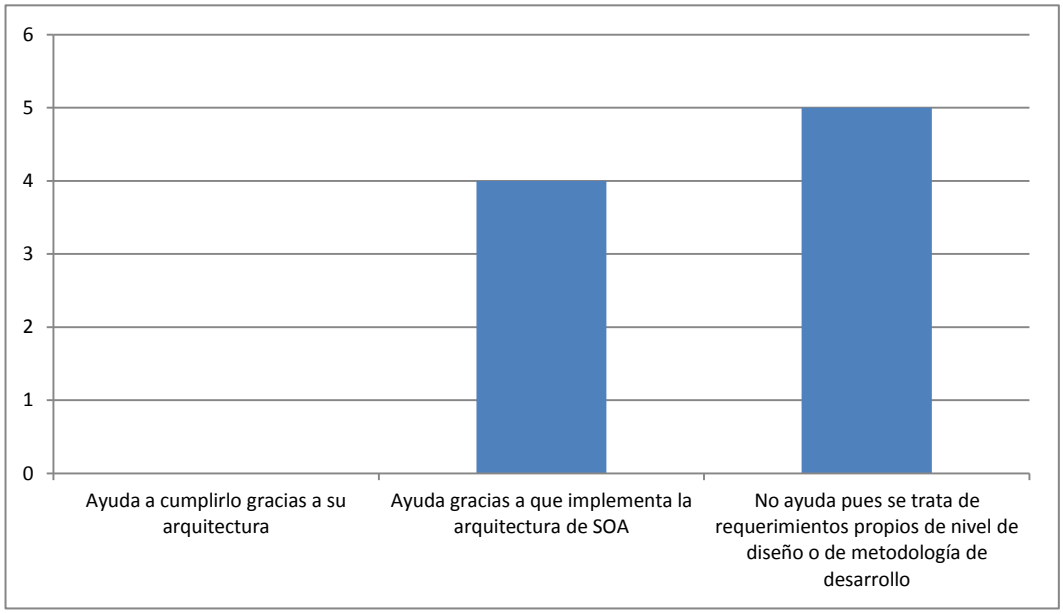


**Ilustración 29 - Grafica Número de Requerimientos de Principio de Diseño vs ESB para Descubrimiento**

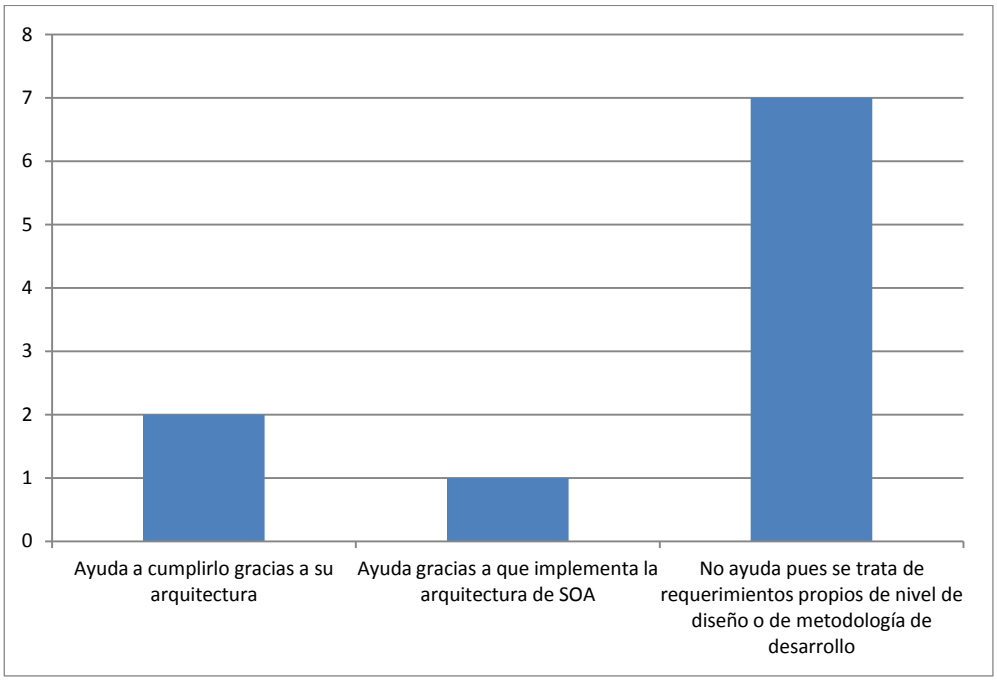


**Ilustración 30 - Grafica Número de Requerimientos de Principio de Diseño vs ESB para Composición**

## Anexo 4. Diseño Solución Caso Estudio

Los elementos que conforman la solución son:

- **Servicios Proceso de Selección de Proveedores**

  Conjunto de servicios que son los componentes del proceso de selección de proveedores, dichos servicios son desarrollados de manera agnóstica y completamente desacoplada, de tal manera que sean de fácil adaptación a cualquier tipo de organización que requiera aplicar un método matemático formal de selección de proveedores. Los servicios que aquí se encuentran enmarcados, son del tipo .Net web services.

  Estos servicios no tendrán una conexión directa con la base datos, o cualquier otro servicio o aplicación intermedia. Tendrán una conexión indirecta a través del ESB.

- **Base de Datos Informix**

  Base de datos que posee toda la información concerniente a proveedores, órdenes de compra y evaluaciones, es decir; los datos de insumo para la ejecución del proceso de la selección de proveedores, serán extraídos de esta base de datos. Por ende su importancia en la solución propuesta.

  Cabe señalar que se trata de una base de datos legada alojada en un sistema operativo rezagado en el tiempo.

- **Enterprise Service Bus**

Es el elemento en el cual se centra este documento y este caso de estudio. Como se estipuló, el Bus de Servicio Empresarial será el encargado de efectuar la integración de los diferentes componentes del macro sistema. Dentro del mismo se alojarán todas las políticas de enrutamiento, que comunicarán cada uno de los elementos involucrados en la solución propuesta. Se encargará de procesar la información obtenida proveniente de la base de datos, para ofrecerla en un formato estándar preferiblemente XML. Mediante dicha estandarización y debido a la naturaleza de la versión del Bus de Servicio utilizado, se da más heterogeneidad al mundo.

Adicional a las políticas de enrutamiento y transformación de mensajes, el ESB contendrá funcionalidades internas de seguridad y transformación de protocolos. Tendrá conexión real con los WS.Net y la base de datos Informix. Sin embargo, tomando como punto de referencia los servicios web .net o cualquier otra aplicación que posteriormente desee consumir la funcionalidad expuesta por el Bus de Servicio, la conexión que se establecerá con el Bus es igual a la conexión que se realiza con un servicio que es consumido. Es cuestión únicamente de establecer una referencia a través del WSDL y adaptarse a los términos que el contrato expone; es decir no existe ninguna diferencia entre invocar un servicio y establecer una conexión a un módulo soportado en un ESB.

La Ilustración 22 muestra la arquitectura de la solución planteada, vista desde un punto de referencia amplio a través de un **Diagrama de Paquetes.**

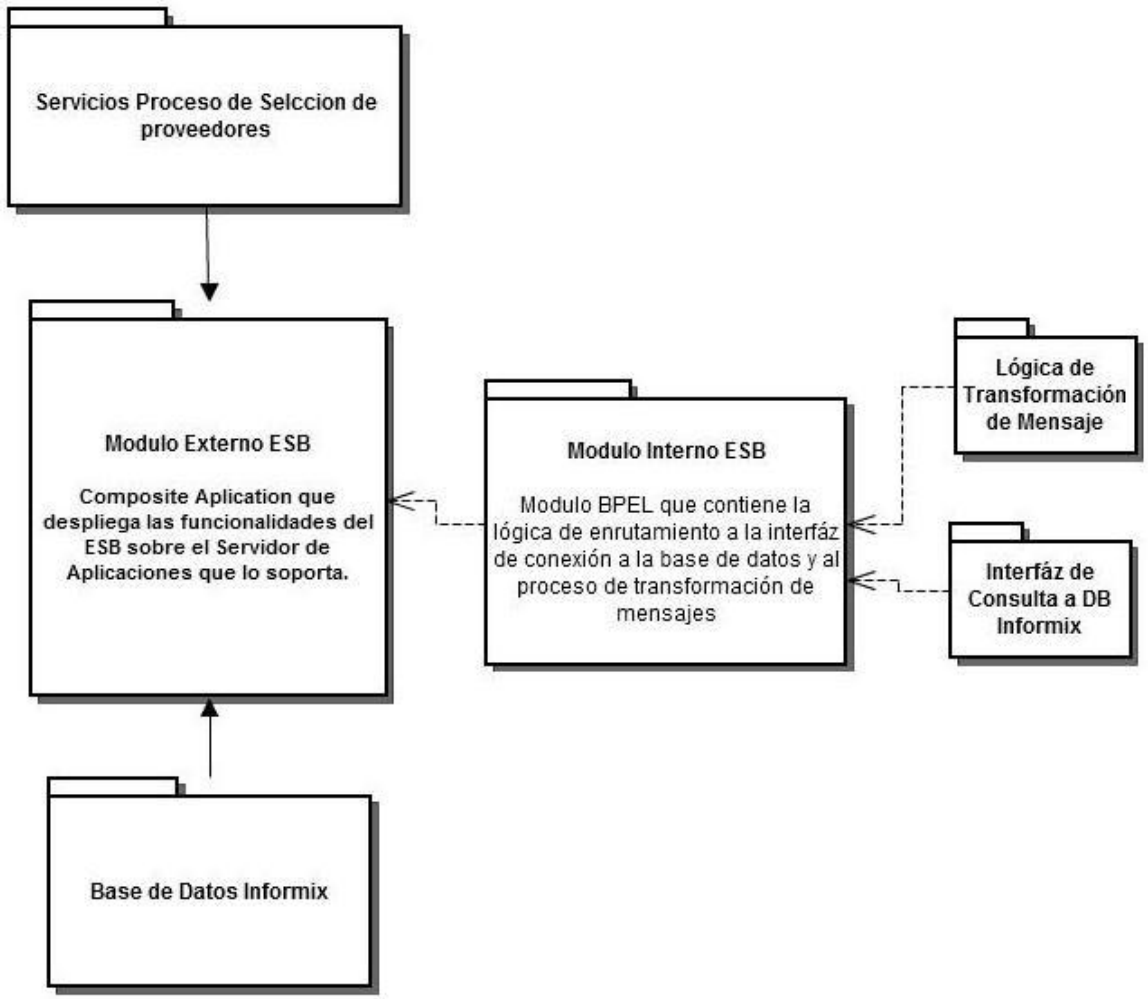


**Ilustración 34 - Diagrama de Paquetes, Arquitectura Solución Caso de Estudio**

### Diagrama de Actividad

Mediante un diagrama de actividad se pretende formalizar el proceso de la labor de enrutamiento que efectuará el ESB. Teniendo en cuenta los procesos

implicados y encerrando cada uno de los elementos relevantes en la arquitectura solución.

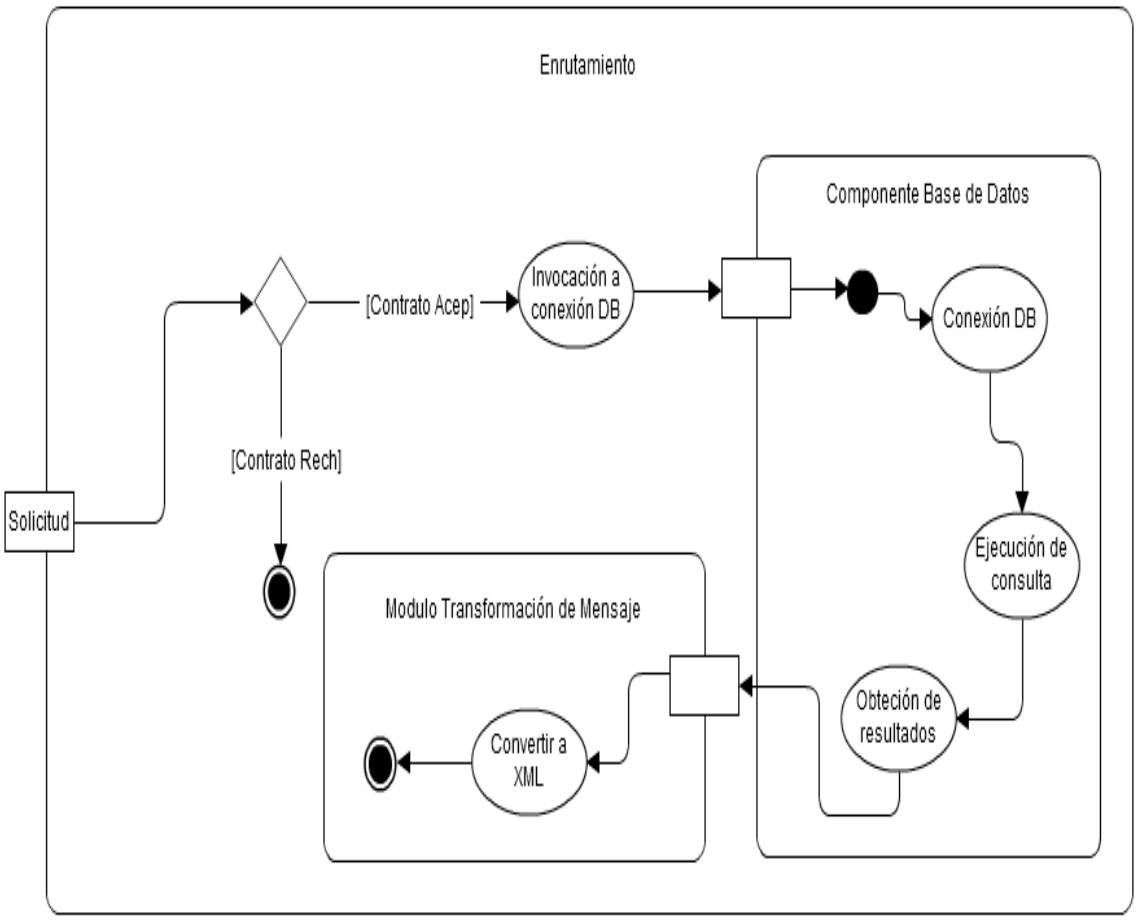


**Ilustración 35 - Diagrama de Actividad**

### Características técnicas de la solución

Se utilizarán herramientas de código libre para la implementación, desarrollo y prueba de la solución:

- Servidor de aplicaciones GlassFish 2.1.1
- IDE NetBeans 6.7.1 (para java)
- Open ESB 2.2

**Desarrollo de la solución**

Partiendo del hecho de que la base de datos Informix 7.3 es una base de datos rezagada en su versión, es entendible que el componente JDBC (el cual permite la conexión de una aplicación java con una base de datos) no esté contenido dentro del conjunto de *plugins* JDBC ofrecidos por el IDE NetBeans. Por lo tanto el primer reto enfrentado fue encontrar el *plugin* JDBC apropiado para la base de datos Informix.

Mediante una exhaustiva búsqueda se halló que el controlador apropiado para la conexión con la base de datos era el controlador de IBM Informix JDBC Driver 3.70.

Luego de tener el controlador apropiado, valiéndose de un mini puerto WAN para acceder a VPN se estableció conexión a la red en la cual se encuentra el equipo contenedor de la base de datos (equipo con sistema operativo **Unix SCO**)

Posterior a la conexión con la fuente de información remota y mediante *XML Schema* se crea la estructura base que servirá como fundamento para la creación del mensaje de respuesta que recibirá la aplicación consumidora. El *XML Schema* producido se basa en la normativa y métrica establecida por la w3c para el año 2001 [19]. Con base en dichas métricas y partiendo de la finalidad que se busca lograr al crear un esquema propio de datos, se diseñó el XML OutputXmlSchema así:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.netbeans.org/schema/OutputXmlSchema"
    xmlns:tns="http://xml.netbeans.org/schema/OutputXmlSchema"
    elementFormDefault="qualified">
    <xsd:element name="Proveedores">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Proveedor">
                    <xsd:complexType>
                        <xsd:attribute name="nit" type="xsd:string"/>
                        <xsd:attribute name="nombre" type="xsd:string"/>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

**Ilustración 36 - XML Schema**

Contando con un modelo de mensaje de respuesta, se definió el contrato mediante el cual se expondrá el vínculo con el Bus de Servicio y su lógica. Como resultado se obtiene un WSDL (basado en XML); primordialmente en el contrato de servicio se puntualizó el tipo de mensaje de entrada o solicitud que debe recibir el ESB, así como el mensaje de salida o respuesta devuelto al consumidor. Sin embargo, la estructura del contrato posee otros atributos importantes, tales como:

- Tipos de enlace (*binding*).
- Operaciones expuestas.

Los principios básicos de la Arquitectura Orientada a Servicios establecen que los servicios deben ser ofrecidos mediante un contrato de descripción de servicios (no solo *web service*). Open ESB busca mantenerse dentro de dicho principio y por lo tanto cualquier funcionalidad que tenga relación con el mismo, deberá exponerse a través de un WSDL. En este caso de estudio por lo tanto se identifica a la conexión a la base de datos y posterior ejecución de una consulta como una

funcionalidad directamente relacionada con el ESB. Luego entonces el siguiente paso realizado es definir el contrato mediante el cual el Bus de Servicio invoca o consume la conexión y consulta a la base de datos. Este tipo de WSDL contiene los siguientes atributos:

- Definición de las entradas, en este caso los parámetros de ingreso para efectuar la consulta.
- Definición del mensaje de salida, en este caso el resulset devuelto por la consulta.
- Tipo de puerto, generado a través del controlado JDBC.
- Operación, atributo que contiene la instrucción SQL.

A continuación de la definición de los contratos WSDL, a través del lenguaje de ejecución de procesos de negocio (BPEL) se creó la política de enrutamiento que debe seguir una solicitud que ingrese al Bus de Servicio (basando la política en el diagrama de actividad de la Ilustración 23). Dentro de las reglas de enrutamiento que se definen se incluye un punto o estado del proceso en el cual se establece la actividad de transformación del mensaje. La entrada que recibe esta estación del proceso es el resulset retornado por la consulta y como salida devuelve un mensaje basado en XML que utiliza como modelo o patrón de creación el *Schema* previamente definido. Por último, en el BPEL se establece que el mensaje transformado será la respuesta devuelta al consumidor, de esa forma se devuelve información en un formato estándar fácilmente procesable.

Como resultado final del proceso previamente descrito se logra exponer mediante un WSDL (lo cual le da connotación de servicio) toda la lógica y políticas

contenidas dentro del ESB. Cualquiera aplicación que tenga la capacidad de consumir servicios mediante la referencia al mismo a través de contrato de descripción de servicios, está en capacidad de obtener la información contenida en la base de datos legada Informix que a su vez se aloja en un sistema operativo muy poco común. En el caso de estudio en cuestión, la información que está contenida en la base de datos y que se expone mediante el Bus de Servicio, es información concerniente a los proveedores, calificación de proveedores y órdenes de compra. Datos administrativos de la Universidad Tecnológica de Bolívar. El módulo anteriormente diseñado y desplegado ofrece operaciones de consultas (integra datos).

## Anexo 3. Implementación de reglas de enrutamiento y transformación de mensaje, Caso de Estudio

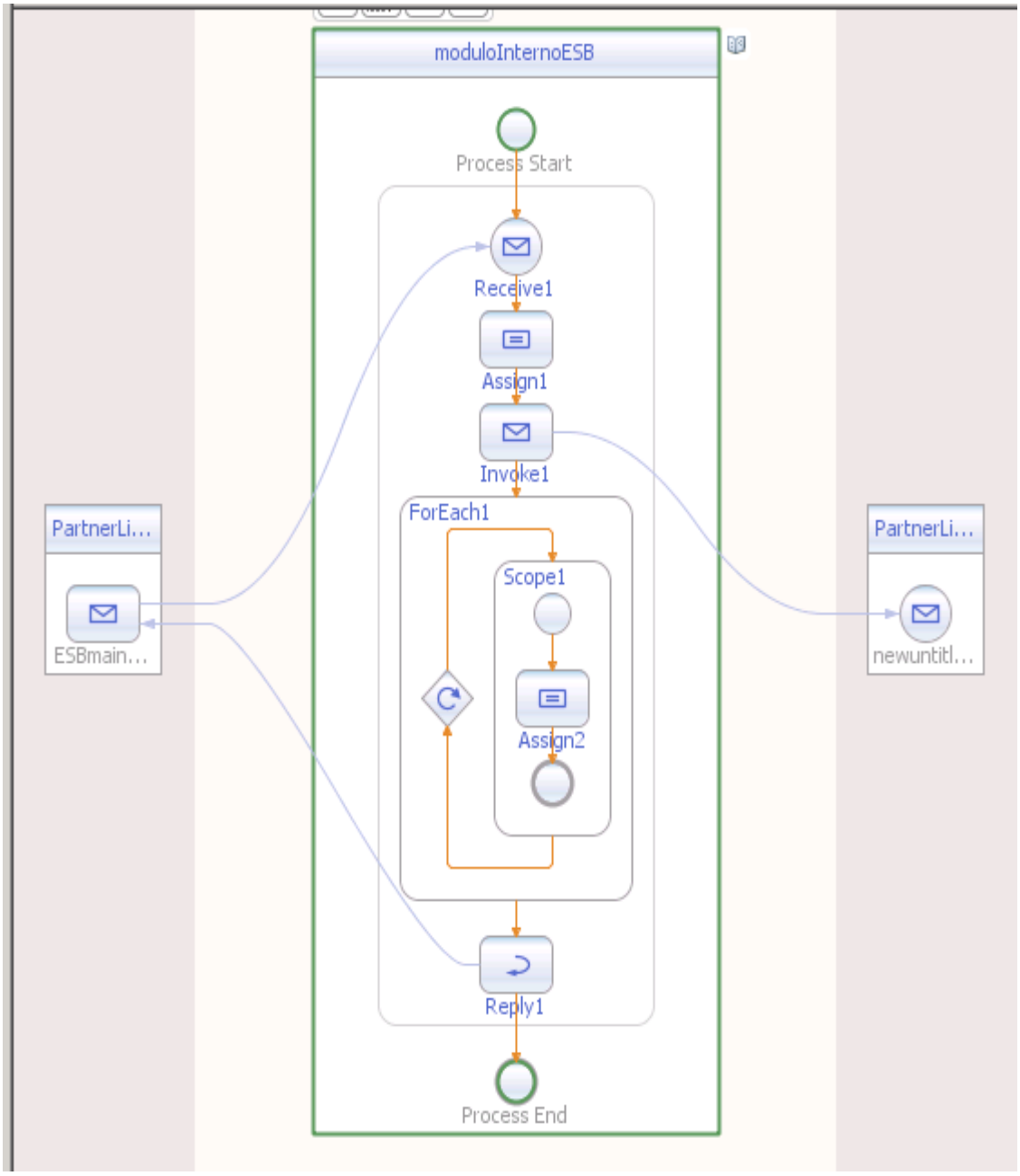


**Ilustración 37 - Pantalla Open ESB definición de Enrutamiento y Transformación de Mensaje Caso de Estudio**