IMPLEMENTACION DE SOLUCIÓN DE INTEGRACIÓN UTILIZANDO HERRAMIENTAS DE SOFTWARE LIBRE

JAIME EDUARDO CORTES

UNIVERSIDAD EAFIT

ESCUELA DE INGENIERIA DE SISTEMAS

MEDELLÍN

2006

IMPLEMENTACION DE SOLUCIÓN DE INTEGRACIÓN UTILIZANDO HERRAMIENTAS DE SOFTWARE LIBRE

JAIME EDUARDO CORTES

Trabajo de grado para optar el título de

Ingeniero de Sistemas

Asesor

César Augusto Suaza Arango

Aspirante a Maestría en Informática

UNIVERSIDAD EAFIT

ESCUELA DE INGENIERIA DE SISTEMAS

MEDELLÍN

2006

TABLA DE CONTENIDO

DEDICATORIA

A ti santísimo Dios refugio y fortaleza de los débiles en quien ponemos toda nuestra confianza, a ti de quien todo don procede y A santa Maria por su ayuda y protección.

AGRADECIMIENTOS

El autor expresa su agradecimiento a:
Jaime Cortes y Alba Lucia Gálvez, sus padres,
quienes con su paciencia y dedicación lograron
dar frutos de amor a través de sus hijos.

A la doctora Raquel Anaya, al profesor Cesar
Suaza y a David Konerding quienes con
Su conocimiento y ayuda lograron
motivar y orientar este proyecto.

La universidad no aprueba ni desaprueba las opiniones emitidas en los trabajos de grado, tales opiniones deben considerarse como propias del autor:

Notas de aceptación

_____

_____

_____

Presidente del jurado

_____

Jurado

_____

Jurado

Ciudad y Fecha

LISTA DE FIGURAS

GLOSARIO

TRANSPARENTE (RPC): Es un mecanismo en el cual los procedimientos locales y los procedimientos remotos son indistinguibles al programador.

STUB: Es un pequeño procedimiento de biblioteca, que representa al procedimiento del servidor en el espacio de direcciones del cliente.

CANAL: "Las aplicaciones de mensajes transmiten datos a través de canales de mensajes o tuberías virtuales que conectan un productor a un receptor. Un sistema de mensajes recién instalado típicamente no contiene ningún canal. Primero se debe determinar como las aplicaciones necesitan comunicarse y luego se crean los canales."

MENSAJE: "Un mensaje es un paquete atómico de datos que puede ser transmitido a través de un canal. Así, para transmitir datos, una aplicación debe dividir los mensajes en uno o mas paquetes, envolver cada paquete como un mensaje y luego enviar el mensaje por un canal, de la misma manera una aplicación receptora recibe un mensaje y debe extraer los datos del mensaje para procesarlos. El sistema de mensajes intentara enviar el mensaje hasta que tenga éxito."

WORKFLOW : Es un componente software que toma como entrada una descripción formal de los procesos del negocio y mantiene el estado de las ejecuciones de los procesos, de tal modo que delega actividades entre la gente y las diferentes aplicaciones de una organización u organizaciones.

REST(Representational State Transfer): pretende mostrar una imagen de cómo una aplicación Web bien diseñada se comporta: una red de páginas Web (una máquina de estados virtual). Donde el usuario progresa a través de una aplicación seleccionando enlaces (transiciones entre estados), dando como resultado la siguiente página (representando el siguiente estado de la aplicación) siendo transferido para la utilización del usuario.

PRUEBA DE CONCEPTO: permite construir un modelo funcional lo suficientemente detallado para determinar de manera específica el valor comercial y técnico de la solución ZLE para una empresa particular.[22]

ZLE(Zero Latency Enterprise): La empresa con latencia cero es una solución tecnológica acuñada por HP que le da a una organización la capacidad de capturar, consolidar, acceder y analizar información en tiempo real..[22]

JMS (Java Messaging Service): Es un API estándar para el manejo de mensaje utilizanro Java.

INTRODUCCIÓN

Los cambios en las condiciones económicas, en las condiciones de regulación del gobierno sobre la industria y la utilización de nuevas tecnologías como Radio Frequency Identification (RFID), obligan a las industrias a cambiar su visión de cómo integran sus aplicaciones y comparten datos.

La integración es un tema que está siendo estudiado ampliamente desde diferentes niveles de detalle: (a) Desde la perspectiva organizacional, la integración propone unos lineamientos organizacionales orientados a controlar y gestionar los recursos de TI de una organización de tal manera que éstos se encuentren alineados a los procesos de negocio; dichos enfoques se articulan en propuestas conocidas como Arquitecturas Empresariales (b) Desde la perspectiva de negocio, la integración es una estrategia que orienta el desarrollo de soluciones como la definición e integración de servicios que potencian los procesos de negocio; dicha aproximación se articula en la propuesta de Arquitecturas Orientadas a Servicios. (c) Desde la perspectiva tecnológica la integración estudia diversos mecanismos que permiten hacer efectiva la comunicación entre diferentes módulos software, con el objetivo de que dicha comunicación no se encuentre limitada por la tecnología sobre la cual está construido un módulo particular.

El presente proyecto está orientado a estudiar el problema de la integración desde la perspectiva tecnológica. La integración de aplicaciones se ha llevado a cabo a través de múltiples aproximaciones, muchas de ellas sincrónicas y en algunos casos enviando grandes cantidades de datos durante ciclos nocturnos. Este tipo de

integración ha conducido a una falta de conocimiento de los datos críticos de la organización en cualquier instante de tiempo debido a la falta de actualización de los datos en un tiempo cercano al real.

Por esto ante la necesidad de tener una organización que muestre información compartida de diferentes aplicaciones y plataformas, se ha desarrollado una aproximación arquitectónica basada en mensajes que facilita la interacción asincrónica y de bajo acoplamiento entre las aplicaciones, permitiendo actualizar los datos de una empresa o conjunto de empresas en tiempo cercano al real.

La mayoría de productos que implementan estas arquitecturas son productos que tienen un costo de licenciamiento altísimo, y por esto las organizaciones buscando la forma de reducir costos e integrar sus aplicaciones, están considerando la utilización de software libre.

Con este proyecto pretendemos evaluar dos herramientas de integración libres (Mule 1.1 y CBEI+ActiveMQ) e implementar una prueba de concepto en la herramienta seleccionada.

1. DESCRIPCIÓN

Las empresas para integrar sus aplicaciones y bases de datos utilizan una herramienta conocida como herramienta de orquestación[1] que permite intercambiar información utilizando XML entre aplicaciones y sin el fuerte acoplamiento del pasado en que un conector realizaba la transformación de datos de un formato a otro de forma rígida.

Dependiendo de la herramienta y de cómo se utilice. Puede ocurrir que la interacción con otras aplicaciones sea sincrónica[2], bloqueando el progreso de una solicitud a una aplicación externa en el caso en que el tiempo de respuesta sea largo o la aplicación este temporalmente fuera de servicio.

La petición sincrónica de datos implica frágiles conexiones punto a punto entre quienes envían y reciben datos volviéndose difíciles de mantener a medida que su número se incrementa, trayendo consigo problemas de escalabilidad y eficiencia.

Por otro lado si la herramienta de orquestación se implementa como un servicio que hace parte de una infraestructura de interacción asincrónica conocida como infraestructura de mensajería, entonces no se tendrán los problemas de escalabilidad y eficiencia introducidos por el modelo sincrónico.

---

[1] Existe una diferencia sutil entre Orquestar y coreografía. La primera describe como los servicios web pueden interactuar entre sí a nivel de mensajes, incluyendo la lógica de negocio y el orden de ejecución de las interacciones. Y la segunda habla del intercambio público de mensajes punto a punto entre organizaciones que colaboran entre sí.

[2]Interacción en la que dos sistemas se ponen de acuerdo al mismo tiempo para comunicarse datos. Si el sistema que espera la información no recibe ninguna respuesta, se queda bloqueado hasta recibirla o hasta que termine el tiempo de un contador. En este lapso de tiempo el sistema receptor no puede realizar ninguna tarea más que esperar.

La infraestructura de mensajería permite comunicar sistemas por medio de paquetes discretos por lo general en un formato conocido como XML. Dichos paquetes (Mensajes) requieren ser modificados y enrutados y para ello se utilizan servicios básicos de procesamiento XML.

Muchos vendedores muestran sus productos centrándose en los servicios básicos que utiliza la herramienta de orquestación y aunque estos ayudan a conocer las capacidades de una herramienta, no muestran las fortalezas y debilidades de fondo que solo se conocen a través de la arquitectura y el diseño de la herramienta de integración /orquestación.

Por ejemplo, si la solución está diseñada sobre un servidor de aplicaciones, el punto de falla es uno solo, en cambio si la herramienta está distribuida a lo largo de varios sistemas, la probabilidad de que se caiga el sistema es mucho menor pero la administración de la herramienta se vuelve complicada. "Por esto si decisiones pobres han sido tomadas de forma temprana en la arquitectura y/o el diseño, entonces es casi imposible recuperarse de ellas y en muchos casos es necesario rescribir todo el producto."[3][1]

Por otro lado, si el orquestador tiene un lenguaje de modelado de procesos complejo y rígido entonces el nivel de abstracción y la flexibilidad en el modela miento se verán comprometidos. Y por tanto la escalabilidad.

[3] CRAGGS, Steve Best of Breed ESBs Identfying best of breed characteristics in a Enterprise Service Buses. 2003 p. 4.

Las herramientas de integración propietarias basadas en mensajes tienen precios de licenciamiento que varían entre 250.000 a 800.000 dólares sin tener en cuenta los servicios de consultoría requeridos debido a su naturaleza propietaria generando el típico lock-in. Por esto es viable pensar en alternativas de código abierto basadas en estándares abiertos como por ejemplo: Mule, Celtix, serviceMix y CBEI entre otras.

1.1 FORMULACIÓN DEL PROBLEMA

¿Qué herramienta de integración de código abierto, cumple mejor los requisitos arquitectónicos de, asincronía, escalabilidad y servicios básicos de procesamiento XML?

2. OBJETIVOS

2.2 Objetivo general

Elegir y poner en ejecución una herramienta de integración libre que permita la comunicación de dos aplicaciones cuyo formato de archivo es diferente y situadas en Máquinas de diferente plataforma.

2.3 Objetivos específicos

Describir la arquitectura deseada de una solución de integración que cumpla con los requerimientos definidos utilizando patrones.

Caracterizar las herramientas de estudio de acuerdo a la arquitectura deseada.

Configurar la herramienta seleccionada, incluyendo: mensajería, orquestador y servicios requeridos.

Realizar una prueba de concepto en la solución definida.

3. JUSTIFICACIÓN

Muchas empresas a nivel mundial se encuentran en procesos de fusión, adquisición e integración lo cual las fuerza a cambiar constantemente. Sin embargo la tecnología que las soporta sigue siendo rígida y esto limita el crecimiento que puedan tener en el futuro.

Un problema común que afrontan las organizaciones es el de integrar sus sistemas y procesos compartiendo datos e información en tiempo "real"[4]. Lo que se ve en la realidad es la existencia de información incoherente entre bases de datos debido al retraso en la actualización de la misma.

Para poder tener información actualizada y sistemas integrados de forma flexible muchas empresas utilizan herramientas de código abierto, muchas de las cuales suelen estar a la vanguardia de la investigación a diferencia de otras tecnologías comerciales que no lo están debido a la falta de escalabilidad de sus arquitecturas.

Este proyecto forma parte de las investigaciones realizadas en el grupo de Ingeniería de Software de la Universidad EAFIT referentes al tema de Arquitecturas Orientadas a Servicios y Arquitecturas empresariales. Se espera que los resultados de esta tesis puedan ser articulados a la propuesta metodológica para el desarrollo de este tipo de aplicaciones y que está siendo definida en una tesis de maestría por el asesor de este proyecto.

[4] Los unicos sistemas que se ejecutan en tiempo real son los controladores electronicos en el piso de una planta. De aquí en adelante cuando hablamos de tiempo real nos referimos a un tiempo cercano al real.

4. LIMITACIONES DE LA INVESTIGACIÓN

Esta investigación pretende evaluar dos productos selecionados de entre mas de diez soluciones construidos para la integración, el primero se conoce como Mule 1.1 y el segundo CBEI+ActiveMQ es una propuesta de un conjunto de tecnologías que aunque no han sido construidas como un mismo proyecto, pueden utilizarse en conjunto para formar una herramienta de integración.

La evaluación estará enfocada en la arquitectura y en la tecnología y no pretenderá ser una guía exhaustiva para la selección.

El proceso a integrar es muy sencillo pues lo que nos interesa es reconocer las mejores herramientas de código abierto de entre muchas herramientas para formar una solución de integración lo mas flexible y escalable posible.

Nuestro propósito es hacer que la herramienta cumpla con los requisitos arquitectónicos de asincronía, escalabilidad y servicios básicos de procesamiento, por lo tanto solo se profundizará en estas tres características clave.

Así mismo aunque existen otros requisitos muy importantes en la evaluación de una herramienta de integración como por ejemplo el desempeño, la robustez de la aplicación, y el numero de adaptadores y conectores entre otros, estos no serán contemplados en este estudio, pero dan pie para futuras investigaciones[5].[2]

[5] Ver el estudio "PANG, Michael; MAHESHWARI Piyush. Benchmarking Message-Oriented Middleware TIB/RV vs SonicMQ University of New South Wales Sydney, Australia."

5. MARCO DE REFERENCIA

Según Gregor Hohpe "integrar aplicaciones no es una necesidad nueva, y diferentes aproximaciones han sido utilizadas para resolverla"[9]. Hohpe en su libro Enterprise Integration Patterns muestra las fortalezas y debilidades de cada una de estas aproximaciones las cuales retomamos sucintamente.

5.1 Intercambio de datos por lote (BATCH)

Esta aproximación consiste en mover un archivo de un sistema a otro sistema cargando lo datos en un archivo objetivo antes de un tiempo especificado.

Ventajas: usar archivos permite el desacople físico entre los diversos procesos, si el sistema objetivo no está inmediatamente disponible para recibir el archivo, este puede ser almacenado hasta que el sistema esté disponible, también los archivos son parte esencial de la plataforma y son independientes del lenguaje siempre y cuando utilicen el mismo conjunto de caracteres.

Desafíos: Cambios de datos en un sistema, pueden no estar disponibles en el otro sistema hasta el siguiente día, esto puede ser confuso para los usuarios, y causar problemas de integridad de los datos (si datos vencidos son actualizados en otro sistema). Adicionalmente los intercambios BATCH tienden a extraer toda la información disponible desde un sistema y replicarla en el otro. Si solo se realizan pequeños cambios en los datos, esta aproximación originará grandes cantidades de transmisión de datos innecesarios.

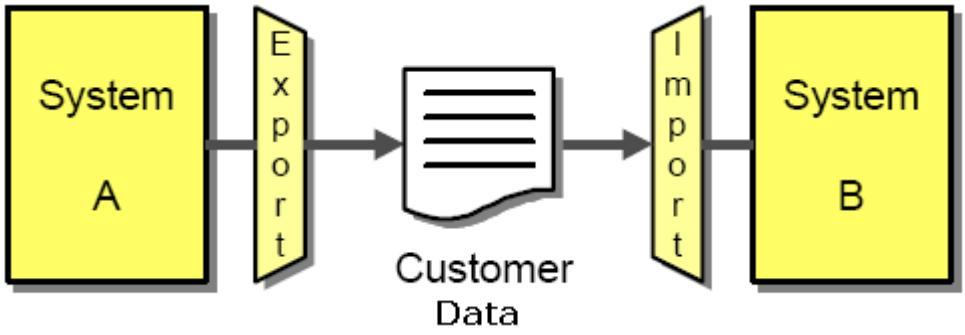


Figura 1. Intercambio BATCH (Integration Patterns Gregor Hohpe)

5.2 Base de datos compartida

Tratando de eliminar los problemas de la sincronización de datos y la replicación de grandes cantidades de datos, algunas organizaciones crearon una sola base de datos para ser compartida por todos los sistemas.

Ventajas: Teniendo todos los datos en un mismo lugar eliminaría la necesidad de duplicar datos. Los problemas de sincronización y la contención pueden ser manejados por los mecanismos de protección y bloqueo que tienen las bases de datos.

Desventajas: Dificultad para definir un modelo de datos que encaje para todas las aplicaciones, pues varias aplicaciones tienen un modelo de datos propietario y cualquier intento de transformar estos modelos en uno solo, se convierte en una tarea de nunca acabar. También el hecho de acceder a una sola base de datos se convierte en un cuello de botella.

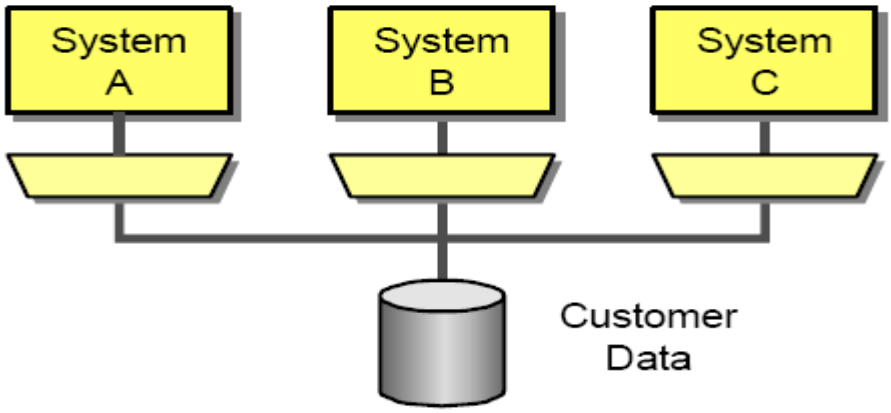


Figura 2. Base de datos compartida (Integration Patterns Gregor Hohpe)

5.3 Intercambio de datos en bruto

Consiste en el intercambio directo de datos utilizando protocolos de transferencia de datos sobre la red tales como sockets TCP/IP. Este mecanismo permite intercambio directo de datos entre dos sistemas en tiempo "real".

Ventajas: Los datos pueden ser propagados inmediatamente después de que los datos son modificados en el sistema origen. La replicación en tiempo real, reduce la posibilidad de que los datos queden sin sincronizar entre los diferentes sistemas y se evitan las replicaciones masivas durante la noche.

Desafíos: Este intercambio ocurre de forma sincrónica, es decir mientras los datos son enviados a través de sockets (o tuberías), el sistema receptor tiene que estar activamente recibiendo, así, el sistema que origina la información se queda esperando una respuesta del sistema objetivo. Si el sistema objetivo o la red no está disponible, no se pueden intercambiar datos y el sistema que origina la información se queda pegado esperando hasta que se finalice el tiempo de espera. En muchas ocasiones los desarrolladores han creado mecanismos de

almacenamiento intermedio y re comprobación para dar comunicación confiable a través de sockets.

El intercambio de datos en bruto está construido en el intercambio de byte por byte de flujo de datos no estructurados y por esto es incapaz de representar tipos de datos complejos. Por lo tanto los desarrolladores son responsables de crear el código que convierta todos los datos fuente en flujo de caracteres y reconvertirlos luego en estructuras de datos en el sistema de llegada. Sin una infraestructura de administración, esta aproximación es propensa a error y dificulta su mantenimiento.

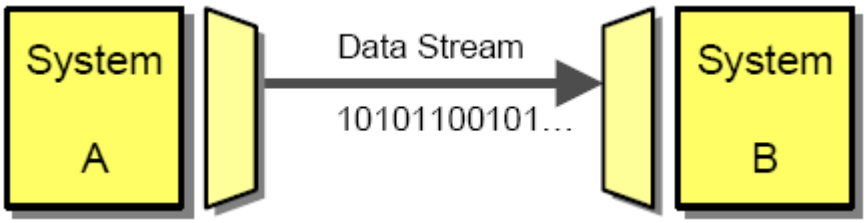


Figura 3. Intercambio en bruto (Integration Patterns Gregor Hohpe)

5.4 Llamadas a procedimientos remotos (RPC)

Este mecanismo aísla la aplicación del empaquetado e intercambio de datos en bruto agregando una capa adicional que maneja el empaquetado (Marshalling) de tipos de datos complejos en bytes para ser transferidos por la capa de transporte. RPC permite a una aplicación invocar una función de otra aplicación de forma transparente.[6][3] El empaquetado es realizado por un Stub[7][4] generado a partir de la especificación de interfaz en el lenguaje IDL.

[6] SINHA Pradeep K. Distributed Operating Systems Concepts and Design. New York: IEEE Computer Society press, 1997. p. 170. Ver Glosario

Desventajas: es un mecanismo no confiable, sincrónico y con los métodos de intercambio de datos en bruto. Además implica frágiles conexiones punto a punto entre quienes envían y reciben volviéndose difíciles de mantener a medida que su número se incrementa. Aun más, muchos vendedores proveen implementaciones RPC que no son compatibles con otros productos.

Un giro interesante es el esfuerzo puesto en SOAP y XML-RPC como el renacimiento de esta arquitectura, con todos sus desafíos.

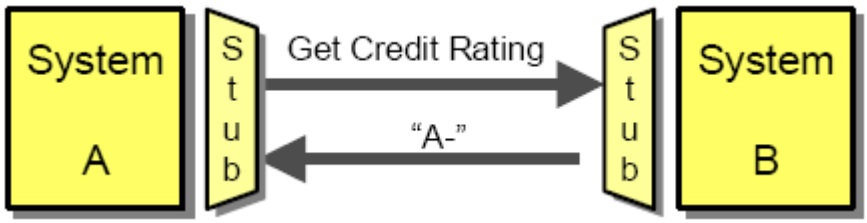


Figura 4. RPC (Integration Patterns Gregor Hohpe)

5.5 Mensajería

Este sistema trata de reparar las falencias de las soluciones previas dando un mecanismo de transferencia de datos confiable y asincrónico. Una aplicación puede publicar datos a la capa de integración y asegurar que los datos serán entregados al (los) recipiente(s). El sistema que origina la información no necesita esperar por una confirmación y puede continuar con el flujo principal de su ejecución. Los sistemas de mensajes también incorporan esquemas de direccionamiento para evitar el tener que mantener las conexiones punto a punto comunes en los sistemas RPC.

---

[7] TANENBAUM Andrew S. Redes de Computadores. México: Pearson Educación, 2003 p. 528 ver glosario

Desafíos: Los sistemas asincrónicos requieren una aproximación diferente a la arquitectura y diseño de los sistemas sincrónicos. La naturaleza asincrónica de la interacción, puede hacer las pruebas y la corrección de errores más difícil

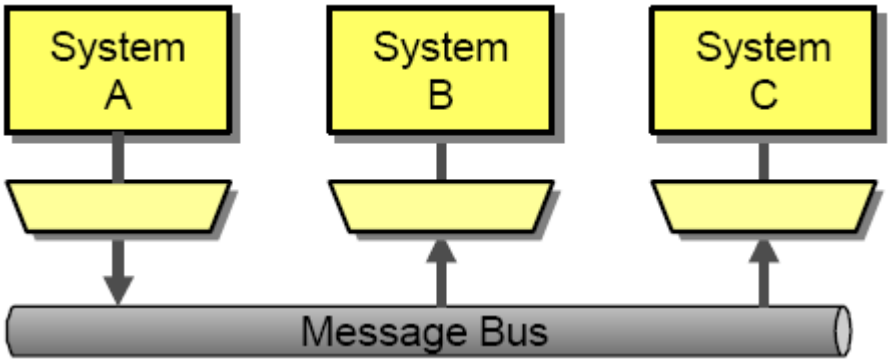


Figura 5. Bus de mensajes (Integration Patterns Gregor Hohpe)

Los sistemas de mensajes conocidos hoy como Message Oriented Middleware (MOM) pueden "enviar y entregar mensajes de forma confiable incluyendo transacciones persistentes, enrutamiento a través de diferentes participantes, propiedades de seguridad y más."

Los sistemas de mensajes asincrónicos "utilizan dos categorías de middleware":

Colas de mensajes: Una cola es un canal punto a punto a través del cual un productor envía un mensaje a un consumidor. La cola sirve de intermediario entre las dos aplicaciones desacoplandolas para que no esten directamente cnectadas entre si. Y además asegura que unicamente un receptor recivirá un mensaje particular. Ver la figura 5.1

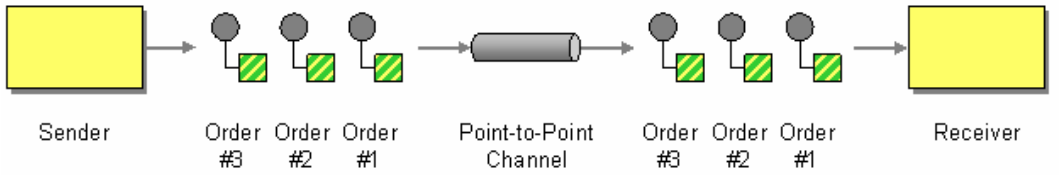


Figura 5.1 Canal Punto a Punto (Integration Patterns Gregor Hohpe)

Tópico: Un tópico es un canal punto-multipunto encargado de realizar broadcasting, es decir el tópico (originador del evento) Publica una notificación y los suscriptores u observadores interesados, sin importar cuantos, reciven la notificación.

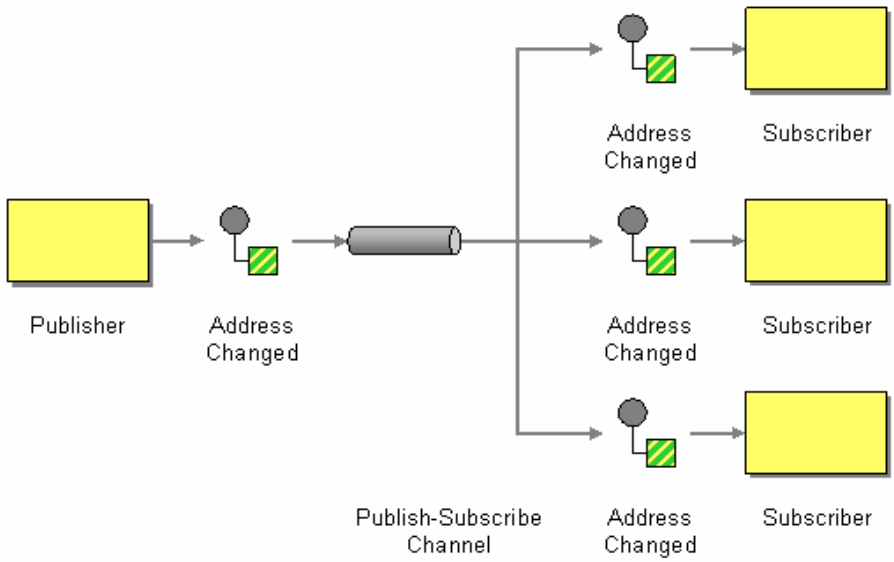


Figura 5.2 Topico (Integration Patterns Gregor Hohpe)

publicar suscribir adiciona dos conceptos importantes a la mensajería básica: "nombre conceptual y enrutamiento independiente de la infraestructura. Como enrutamos y entregamos mensajes ya no depende de la infraestructura física, sino

que depende de conceptos tales como canales nombrados o tópicos, tras los cuales uno o mas recipientes pueden estar ocultándose.”

En los sistemas publicar suscribir, el envío de los mensajes depende totalmente de las acciones de los receptores, los cuales son desconocidos por quienes envían los mensajes.

Un MOM es un punto de inicio para la integración pero no es suficiente, y para aplicaciones de integración complejas se requiere de un administrador de procesos.

5.6 Administrador de procesos o Motor de Workflow

Un administrador de procesos es una unidad central que determina el siguiente paso a ser ejecutado por otro componente o unidad de procesamiento. El flujo de proceso alternante que resulta entre la unidad central y las unidades de procesamiento es lo que se conoce como patrón Hub/spoke. Ver Figura 6.

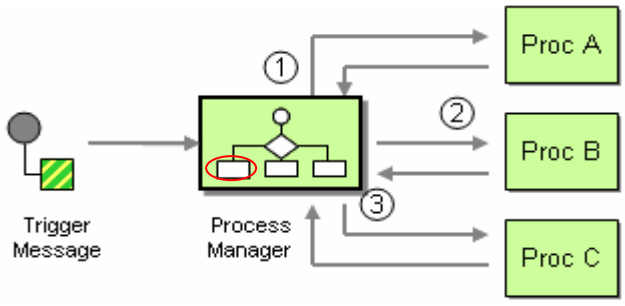


Figura 6. Uso de unidad central de procesamiento, para mantener el estado de la secuencia y determinar el paso siguiente de procesamiento con base en los resultados intermedios

Un proceso (Workflow) o modelo de proceso es una descripción de un proceso de negocio con suficiente detalle como para ser ejecutado por un sistema administrador de procesos. Un modelo de proceso esta compuesto por un numero de tareas (ver elipse roja en la gráfica) que están conectadas en forma de grafo dirigido. Una instancia de ejecución de un modelo de proceso se le llama caso o instancia de proceso, y pueden existir varios casos de un workflow particular ejecutándose simultáneamente.

Estos modelos de proceso suelen representarse por medio de lenguajes de modelado.

5.6.1 Deficiencias en los lenguajes de modelado

La presión por estandarizar varios aspectos de los sistemas distribuidos bajo el paradigma de los servicios Web, ha dado origen a una serie de lenguajes de modelado que permiten componer los servicios Web.

Estos lenguajes a su vez han sido evaluados sistemáticamente sobre sus capacidades y limitaciones y se ha encontrado que el mas estandarizado y completo en cuanto al número de patrones que soporta es el lenguaje BPEL4WS[10]. Sin embargo BPEL4WS es un lenguaje complejo debido a las superposiciones de sus construcciones y a la falta de claridad semántica[11]. Además BPEL4WS está limitado a realizar integración con aplicaciones externas únicamente a través de servicios Web[8].[12]

Unas de las debilidades importantes que tienen los diferentes administradores de proceso actuales incluidos los basados en BPEL4WS es su falta de control de la concurrencia al acceder a datos compartidos y la falta de control en el paso de datos entre instancias de tareas.[9][12]

Esta debilidad del control de la concurrencia puede ser resuelta por plataformas para el procesamiento de XML tales como 4Suite que hacen las funciones de repositorio y servidor con las capacidades de un motor de base de datos. También por bases de datos como sql server 2005 cuando almacenan los documentos XML de forma nativa.

---

[8] El ambiente operativo de las organizaciones tiene diferentes fuentes de datos y aplicaciones, y no se limita a servicios Web.

[9] Estas dos propiedades son esenciales para la interacción asincrónica del administrador del proceso con las unidades de procesamiento o aplicaciones que se quieren integrar.

Y la debilidad del paso de datos entre instancias de tareas es solucionada utilizando Workflow científicos que en lugar de estar centrados en tareas y en el control de flujo entre tareas utilizando BPEL4WS, están centrados en datos y en el flujo de datos utilizando lenguajes más flexibles. Algunos ejemplos de Workflow científicos son: Triana, Taverna y CBEI entre otros.

6. DESCRIPCIÓN DE LA ARQUITECTURA DESEADA

Las soluciones de integración están formadas por piezas diferentes y estas piezas pueden ser descritas utilizando diferentes tipos de diagramas. La diagramación definida por Gregor Hohpe, es capaz de auto describirse fácilmente y por esto la elegimos para describir la arquitectura deseada. Las piezas básicas de la solución de integración se describen en la siguiente tabla.

PIEZAS QUE DESCRIBEN LA SOLUCIÓN DE INTEGRACIÓN

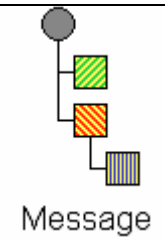
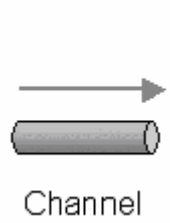
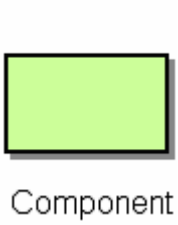
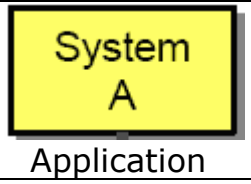
	Datos
	"Un mensaje es un paquete atómico de datos que puede ser transmitido a través de un canal. Así, para transmitir datos, una aplicación debe dividir los mensajes en uno o mas paquetes, envolver cada paquete como un mensaje y luego enviar el mensaje por un canal, de la misma manera una aplicación receptora recibe un mensaje y debe extraer los datos del mensaje para procesarlos."
	"Las aplicaciones de mensajes transmiten datos a través de canales de mensajes o tuberías virtuales que conectan un productor a un receptor. Un sistema de mensajes recién instalado típicamente no contiene ningún canal. Primero se debe determinar cómo las aplicaciones necesitan comunicarse y luego se crean los canales."
	Es una abstracción de las piezas principales que componen una solución, por ejemplo un message router, un traductor o cualquier otro componente software que haga parte de la solución.
	Representa a las aplicaciones que se quieren integrar

Tabla1. Diagramas para representar la solución de integración

Como vimos en las secciones 5.5 y 5.6 para nuestra solución de integración utilizaremos la herramienta de mensajería y el administrador de procesos, por lo tanto debe existir un puente entre estas dos aplicaciones. Veamos como sería dicho puente a través del patrón Endpoint.

6.1 Patrón punto de entrada/salida de mensajes (Message Endpoint)

Las aplicaciones y el sistema de mensajes son dos conjuntos separados de software. Las aplicaciones dan funcionalidad para un tipo de usuario, mientras que el sistema de mensajes administra los canales de mensajes para transmitir mensajes para la comunicación. Aún si el sistema de mensajes está incorporado como una parte fundamental de la aplicación, sigue estando separado y debe existir una forma de conectarse y trabajar juntos. En la figura 7 vemos un endpoint que es justamente esa comunicación entre los dos conjuntos de software.

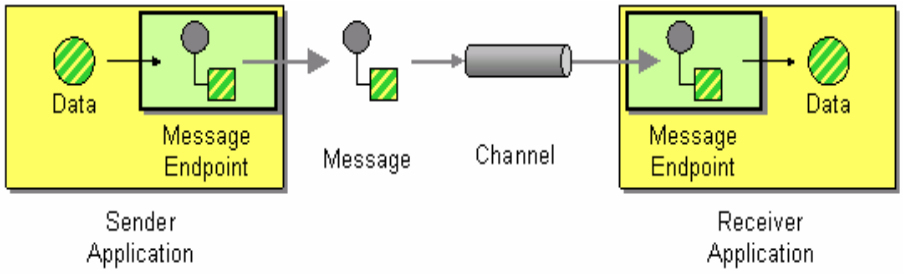


Figura 7. Diagrama de Endpoint que sirve de puente entre las aplicaciones y el sistema de mensajes

6.2 Enterprise Service Bus (ESB)

Actualmente en la empresa se ha acuñado el término bus de integración que consiste en una columna vertebral sobre la que se integran diferentes aplicaciones y servicios de manera que haya bajo acoplamiento entre ellas e interacción asincrónica. Un bus de integración reúne las siguientes herramientas:

- Message Oriented Middleware
- Servicios Web
- Enrutamiento Inteligente
- Transformación de datos XML

Veamos un diagrama de esta arquitectura de integración.

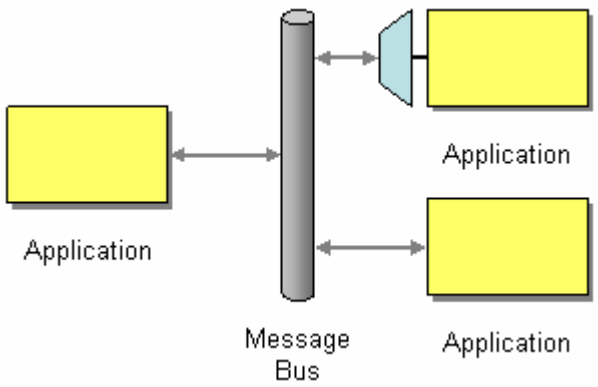


Figura 8. Arquitectura ESB (Fuente Gregor Hohpe)

"Dependiendo de la arquitectura de un ESB, es posible tener una mejor plataforma en términos de la productividad del desarrollador, la facilidad de administración y la clase de aplicaciones de integración que pueden ser desarrolladas."[13] Existen dos tipos de ESB dependiendo de cómo hayan surgido, si a partir de una herramienta de mensajes o haciendo énfasis en el concepto de los servicios Web. Veamos estos dos tipos de ESB.

6.3 ESB centrado en servicios Vs. ESB centrado en mensajes

Actualmente existen dos tipos de ESB, los que están centrados en mensajes que hacen énfasis en las capacidades de transporte y los centrados en servicios que se enfocan en las capacidades de desarrollo hospedaje y despliegue de los servicios.[10]

Las fortalezas del ESB centrado en mensajes son los tipos de interacción asincrónica, publicar/suscribir y el asegurar calidad del servicio en los mensajes entregados.

Por otro lado los ESB centrados en servicios, facilitan la creación y hospedaje de servicios basados en estándares. Veamos la siguiente tabla que nos ayuda a comparar los pros y los contras de cada tipo de enfoque arquitectónico.

ESB CENTRADO EN MENSAJES	**ESB CENTRADO EN SERVICIOS**
1. No soportan creación y hospedaje de servicios. Se supone que los servicios ya han sido implementados en una plataforma aparte, lo cual significa que el desarrollo y administración están distribuidos en dos plataformas.	**1.** Soportan la creación y hospedaje de servicios. El desarrollo y administración SOA es soportado por una sola plataforma
2. Si se quiere una comunicación	**2.** Los servicios se comunican

[10] La arquitectura de ESB descrita en la sección 6.6 es una arquitectura de ESB basada en mensajes.

administrable y a la que se le puedan aplicar políticas se tiene que establecer la comunicación únicamente a través de un MOM específico. Ya que las políticas de comunicación están atadas a una implementación de transporte específico cuando en realidad estas deben poder ser aplicadas a múltiples canales de transporte	independientemente del transporte utilizando únicamente el punto abstracto (endpoint) del servicio objetivo. El ESB luego enruta esta comunicación a través del transporte adecuado al servicio objetivo. Así, la implementación de un servicio no está limitada por el uso de un MOM específico. Y es libre de comunicarse de forma administrada sobre cualquier protocolo de transporte.
3. Requiere duplicar la infraestructura de mensajes cuando ya se tiene una infraestructura de mensajería montada. Esto debido a que la solución de integración exige un MOM específico.	**3.** Los ESB centrados en servicios puede reutilizar la infraestructura de mensajería existente en una organización, eliminando la necesidad de duplicar dicha infraestructura.
4. Las capacidades ofrecidas por el bus (por ejemplo transformación, enrutamiento y procesos) están separadas de los servicios de negocio desplegadas en la SOA. Estas capacidades del bus deben ser desplegadas como parte del bus de mensajes, o en muchos casos, como capacidades de un servidor separado en	**4.** Las capacidades ofrecidas por el bus (por ejemplo transformación, enrutamiento y procesos) son servicios Web que están desplegados en la plataforma de servicios. Estos servicios son desplegados y administrados en la misma forma que los servicios de negocio SOA. Esto significa que tanto los servicios ofrecidos nativamente por

la configuración. Estas capacidades no están definidas o desplegadas como servicios reusables en la SOA	el ESB como los servicios desplegados en el ESB pueden utilizar una infraestructura común para ser desplegados y administrados.
5. El foco del desarrollador esta en configurar el bus, poner mensajes en el bus, leer mensajes del bus y así sucesivamente. No hay un nivel de abstracción más alto de las funciones del negocio y no se provee un modelo que permita hacer servicios de una granularidad mayor. Esta abstracción debe ser provista por una plataforma de servicios y luego integrarla con el ESB centrado en mensajes.	**5.** El foco está en alcanzar un buen nivel de abstracción de la infraestructura subyacente y facilitar la construcción de servicios de negocio robustos y reutilizables tanto internamente, como por los clientes y aliados estratégicos. Para esto el ESB centrado en servicios provee un modelo de desarrollo de servicios de lo más general a lo más particular que facilita el mapeo entre interfaces centradas en negocios con interfaces técnicas de bajo nivel.

Tabla 2. Analisis comparativo entre arquitecturas ESB (Fuente: Service Centric Vs Message Centric ESBs a critical comparison of two ESB approaches [13])

En las siguientes figuras se muestran las arquitecturas de los enfoques de ESB centrados en mensajes y centrados en servicios.

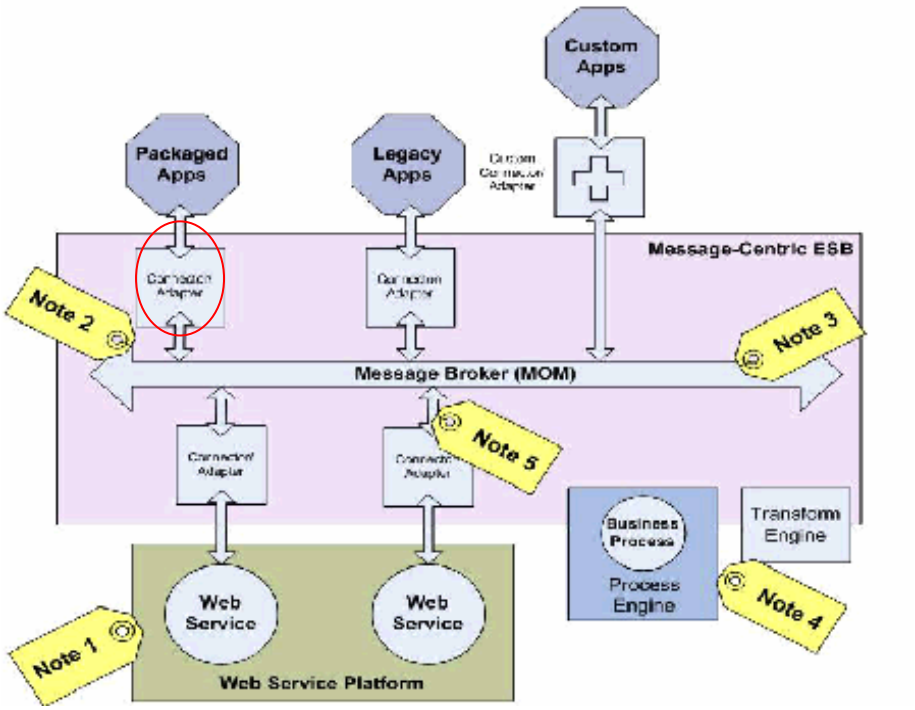


Figura 9 Arquitectura de ESB centrada en mensajes (Fuente: Service Centric Vs Message Centric ESBs a critical comparison of two ESB approaches [13])

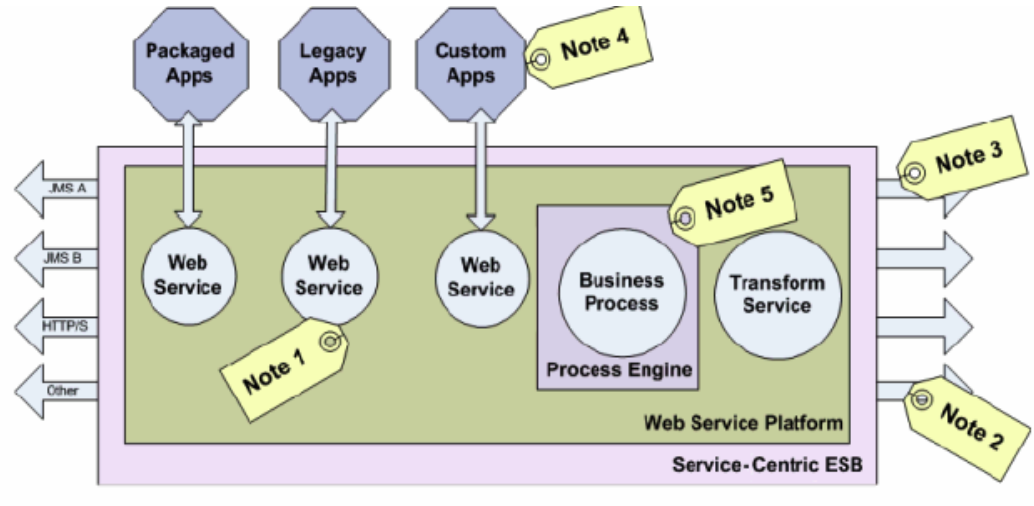


Figura 10 Arquitectura de ESB centrada en servicios (Fuente: Service Centric Vs Message Centric ESBs a critical comparison of two ESB approaches [13])

Como ejemplos concretos de estos dos tipos de arquitecturas tenemos: sonic software, centrada en mensajes, y cape clear centrada en servicios, veamos de forma rápida sus arquitecturas.

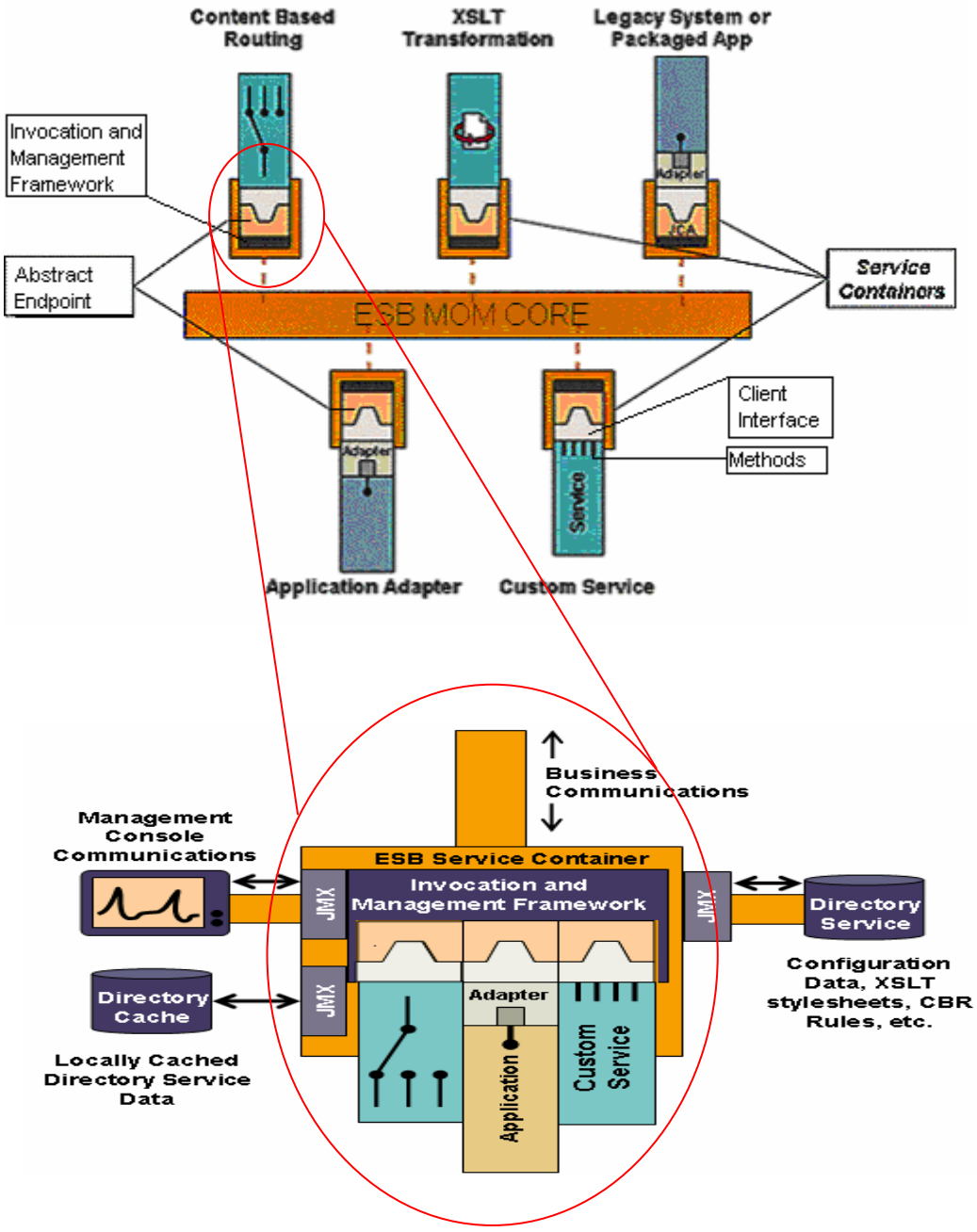


Figura 11. Arquitectura ESB centrada en mensajes con ampliación del contenedor de servicio que soporta la entrada y salida de datos de configuración, auditoria y manejo de fallas (David A Chappell)

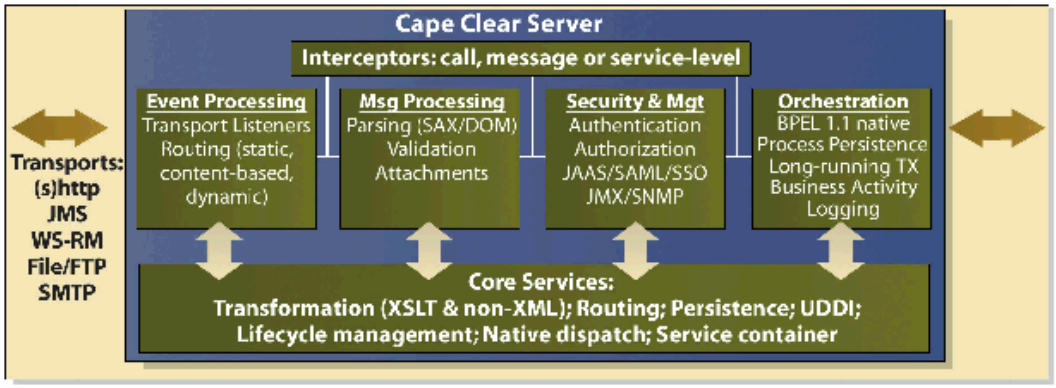


Figura 12 Servidor ESB Centrado en servicios (Cape Clear's ESB [14])

Si observamos con cuidado, los dos enfoques estudiados muestran diferentes formas de implementar la especificación JBI; Así. La arquitectura centrada en mensajes aplica JBI a cada uno de los contenedores de servicio que se conectan al MOM y la arquitectura centrada en servicios aplica JBI a una plataforma de servicios Web única(Cape Clear Server) comparar las Figuras 11 y 12 con la arquitectura de JBI que aparece en el anexo B

6.4 Refinando Criterios

En los siguientes tres numerales se concluye la información obtenida de las variables de asincronía, escalabilidad y servicios básicos de procesamiento para luego describir la arquitectura deseada

6.4.1 Asincronía (Mensajería y Actores)

La asincronía se obtiene al reemplazar la invocación directa entre aplicaciones por la comunicación indirecta entre ellas utilizando una aplicación intermedia conocida como infraestructura de mensajes. Esta infraestructura hace que las aplicaciones se desacoplen entre si y por lo tanto que puedan interactuar asincrónicamente.

Pero no solo es importante la asincronía entre aplicaciones, sino que también es importante la asincronía en el lenguaje de modelado de procesos de integración. Como se vio en la sección 5.6.1 BPEL es rígido y limitado, por esto se requiere de algun lenguaje mas flexible, que permita el acceso concurrente a datos y el paso fácil datos entre instancias de tareas.

6.4.2 Servicios Básicos y Escalabilidad (¿Arquitectura centrada en mensajes o en servicios?)

La arquitectura puede estar centrada en mensajes (usando contenedores de servicios) , o centrada en servicios(usando un servidor de aplicaciones). Si es distribuida a lo largo de la red como la aproximación de los diferentes contenedores de servicios expuestos en la sección 6.3.1 la arquitectura es más tolerante a fallos, pero es difícil de administrar. Por otro lado si la arquitectura es centrada es mucho más fácil de administrar y se gana en el nivel de abstracción SOA como se describe en la Tabla 2 sobre todo en los numerales 4 y 5.

En la arquitectura distribuida si uno quiere desplegar un nuevo servicio o simplemente cambiar un detalle en una interfaz de servicio, se debe actualizar este cambio en todos los archivos XML en todos los contenedores de servicio que están ejecutándose en la organización lo cual hace complicada la solución.

Es mejor poder actualizar centralmente los servicios y no tener luego que replicar a todos los demás contenedores de servicio los cambios, puesto que la solución de integración se volvería in-administrable a largo plazo y se complicaría el control de los mensajes utilizados en la solución.[24]

6.4.3 Escalabilidad y desempeño (Concurrencia ¿hilos o eventos?)

Cuando se administran los servicios centralmente desde un contenedor J2EE, camino que suele seguirse al implementar JBI en proyectos de software libre. El contenedor crea un hilo para realizar una llamada bloqueante a cada cliente. Esta solución es compleja e impone una gran sobrecarga en la administración de los hilos cuando el número de clientes crece.

Según el editor de la especificación JBI, Ron TEN-HOVE en el foro de desarrolladores de Sun Microsystems, "la administración de hilos presenta algunas dificultades, por ejemplo es peligroso dejar que los componentes utilizados en una solución JBI manejen sus propios hilos porque puede que no finalicen los hilos adecuadamente. Y además si dichos hilos no son hilos de demonio, entonces la JVM no podrá finalizarlos." [25]

"Los limites de escalabilidad introducidos por los hilos han hecho que muchos desarrolladores se abstengan de utilizarlos y que en su lugar utilicen una aproximación orientada a eventos para administrar la concurrencia. En esta aproximación un servidor consiste de un pequeño número de hilos(uno por cada CPU) que se ejecutan continuamente, procesando eventos de diferente tipo tomados de una cola. Los eventos pueden ser generados por el sistema operativo o internamente por la aplicación, y generalmente corresponden a lecturas I/O de la red o el disco, notificaciones, contadores u otros eventos específicos de la aplicación. La aproximación orientada a eventos implementa el procesamiento de cada tarea como una máquina de estados finita, donde las transiciones entre los

estados en la FSM son disparados por eventos. De esta forma el servidor mantiene su propio estado de ejecución para cada tarea en lugar de recaer sobre el contexto de un hilo.”[17]

7. ARQUITECTURA ABSTRACTA DESEADA

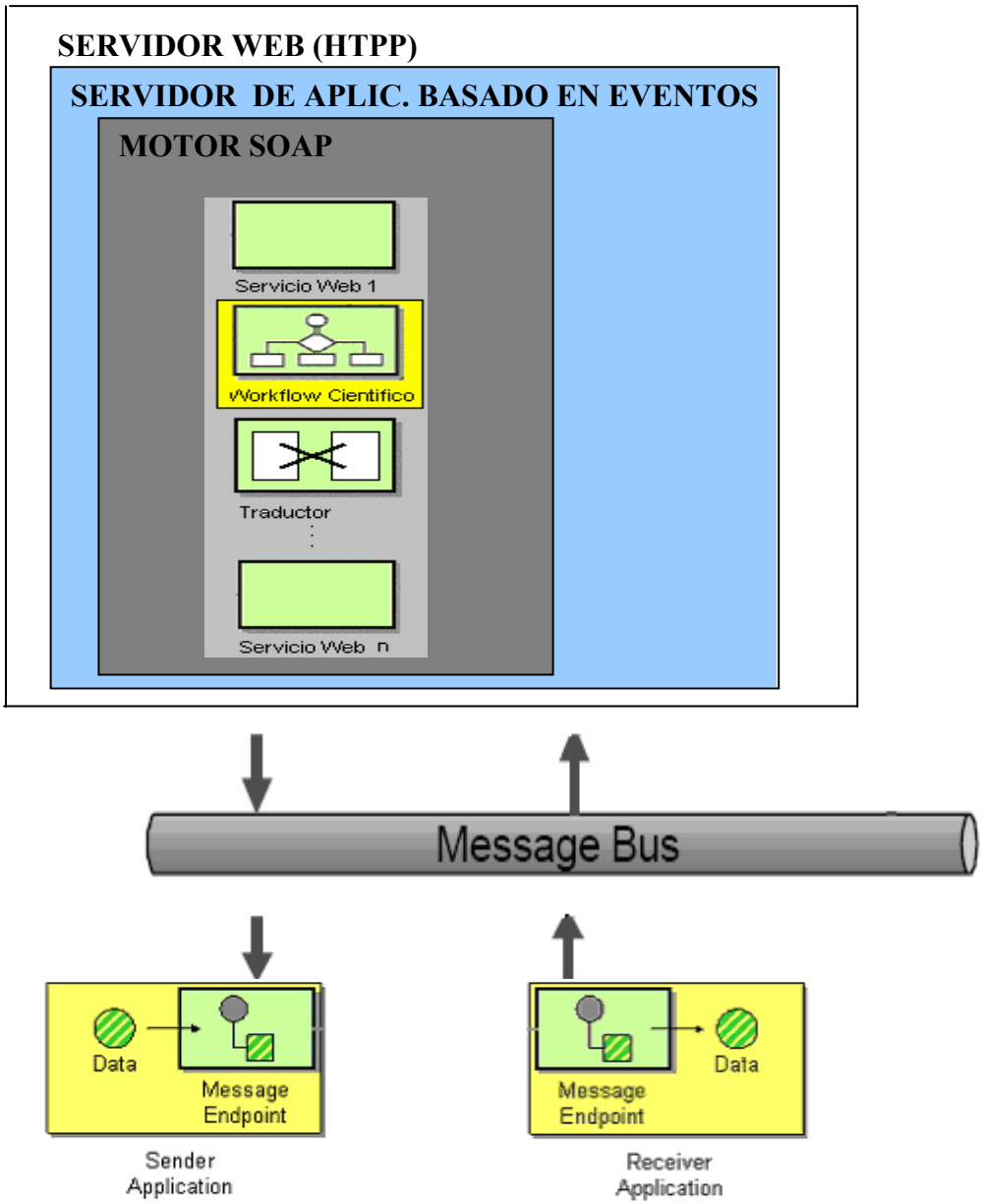


Figura 13. Arquitectura deseada (Fuente: Python WSRF Programmers' Tutorial y Enterprise Integration Patterns)

Servicio Web: Código que expone un conjunto de operaciones. Así una clase puede ser un servicio web, y los métodos de esta clase son las operaciones del servicio Web. [27] Para que los clientes puedan invocar esta clase (servicio Web) se utilizan solicitudes SOAP. Estas solicitudes son manejadas por un motor que interpreta y crea estas solicitudes SOAP.

Motor SOAP: Es una porción de código que sabe como interpretar y crear solicitudes SOAP. Del lado del servidor se utiliza un motor SOAP genérico[11] y del lado del cliente se crea un stub[12] que envia los mensajes SOAP. "Como la funcionalidad del motor está limitada a manipular SOAP. Las solicitudes recibidas por diferentes clientes deben ser manejadas por un servidor de aplicaciones."[27]

Servidor de Aplicaciones: Provee un espacio activo para aplicaciones que deben ser accedidas por diferentes clientes.[13] "El motor SOAP se ejecuta como una aplicación dentro del servidor de aplicaciones. Si el servidor de aplicaciones no incluye funcionalidad http, también requeriremos de un servidor Web."[27]

Servidor HTTP: Es conocido como servidor Web. Este software sabe manejar mensajes HTTP (Un ejemplo de este software es el servidor Apache http).

Bus de Mensajes: Es una infraestructura unificada de mensajes para aplicaciones distribuidas que provee multicast, comunicación de grupo, y comunicación punto a punto de manera confiable. (Un ejemplo de este software es SPREAD y diferentes implementaciones del estandar JMS como ActiveMQ, MantaRay entre otras)

En las arquitecturas estudiadas a la unión del motor SOAP + el servidor de aplicaciones + el servidor HTTP se le ha llamado Contenedor de servicios y en la arquitectura centrada en servicios servidor de servicios Web (Ej: servidor Cape Clear).

---

[11] Un ejemplo de motor SOAP es Apache Axis.
[12] Ver la sección 5.4 y el glosario
[13] Ejemplos: Jakarta Tomcat basado en hilos y Twisted Server basado en eventos

9. HERRAMIENTAS A EVALUAR Y MÓDULOS QUE LA COMPONEN

9.1 Mule 1.1

Mule es un framework de mensajes liviano que contiene un broker de objetos distribuible para administrar la comunicación entre aplicaciones. El papel del broker de objetos es administrar los componentes de servicio. Estos componentes de servicio son llamados Universal Message Objects o UMOs, y son básicamente objetos Java. Los UMOs pueden existir dentro de la misma Máquina Virtual o pueden ser dispersados en una red o Internet. El broker de objetos sigue el patrón de diseño SEDA. Toda la comunicación entre UMOs y otras aplicaciones es realizada a través de endpoints de mensajes.

En la figura 25 podemos ver como Mule consiste de muchas instancias de UMOs dispersas a través de la red. Cada instancia es un contenedor liviano que hospeda uno o mas componentes UMO. Cada componente UMO tendrá una o mas endpoints a través de los cuales enviará o recibirá eventos

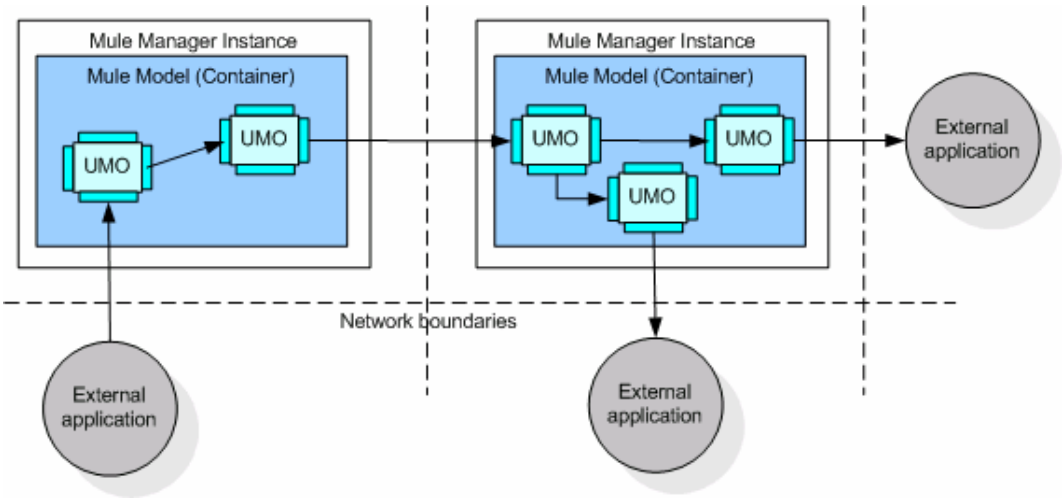


Figura 14. Arquitectura del ESB Mule

9.2 CBEI+ActiveMQ

CBEI es un framework de ejecución de workflows que “simplifica el modelo de workflow a un grafo acíclico dirigido simple de actores conectados a través de puertos nombrados.”[26]

“La ejecución del workflow sigue estrictamente la estructura de dependencia del grafo. Los actores son programados cuando los actores de los cuales depende completan su ejecución.”[26] En este lenguaje de workflow no se da soporte a estructuras condicionales, ramificaciones o variables. Aunque es posible construirlas implementando un actor utilizando el lenguaje Python.

“El Workflow incluye algunos actores genéricos tales como: Emitir una constante, Ramp, Log, Sumador, Multiplexor, invocador de servicio Web y recuperador de páginas Web.”[26]

Como CBEI es implementado utilizando el Framework de programación asincrónico Twisted, los usuarios pueden escribir actores que pueden potencialmente tomar un periodo de tiempo muy largo en terminar su ejecución, sin bloquear otros actores que están actualmente ejecutándose. CBEI fue diseñado intencionalmente para eliminar características complejas de los workflow tales como estructuras de grafos cíclicas, pero debido a la flexibilidad de su framework es posible conectarlo a otro workflow que las haga por el o implementarlas como se dijo anteriormente utilizando Python.

En muchas situaciones, actores livianos que no necesitan consultar servicios externos pueden llevar a cabo rutinas, tales como procesamiento XML, reducción estadística de datos, y visualización. ver la figura 25

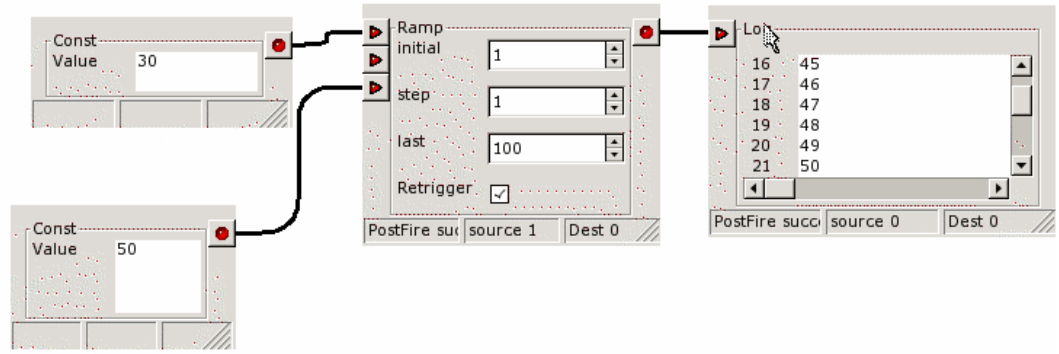


Figura 15. Actores acíclicos unidos mediante puertos (Fuente Distributed System Department Berkeley referncia [28])

En la definición general de un ESB se vio que este está compuesto por un middleware de mensajes, servicios web, enrutamiento inteligente y transformación de datos XML.

CBEI no tiene implementado el middleware de mensajes y nuestra propuesta es suministrarle uno para que la asincronía sea soportada no solo a nivel del workflow con actores asincrónicos, sino también entre las aplicaciones participantes. Se tuvieron en cuenta herramientas tales como spread, Mantaray y ActiveMQ. De estas herramientas se eligió ActiveMQ por ser superior en performance, en el manejo de la persistencia y posee varios servicios más que la hacen una implementación JMS superior a Spread y Mantaray.

Al extender CBEI con el middleware de mensajes obtenemos un ESB cuya arquitectura es la siguiente.

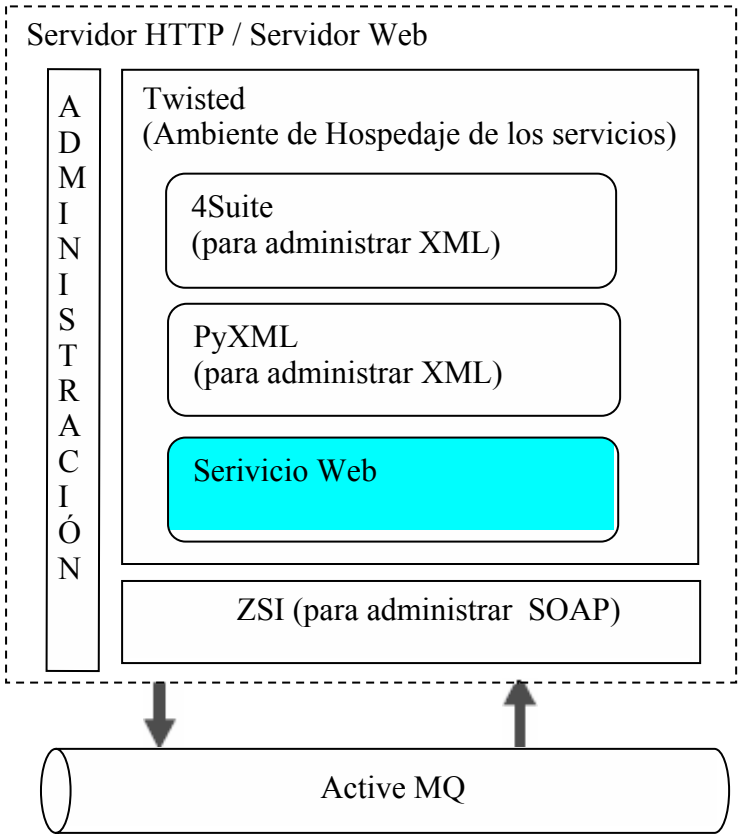


Figura 16. Arquitectura CBEI+ActiveMQ (Fuente ver Bibliografía [19] y [20])

8. CARACTERIZACION DE HERRAMIENTAS CON LA ARQUITECTURA DESEADA

Para conocer cual de las arquitecturas es mas acorde a la arquitectura abstracta deseada realizamos una comparación cualitativa de las propiedades de las arquitecturas analizadas, y una evaluación cuantitativa de acuerdo a criterios que recogen información relevante de la arquitectura.

8.1 Comparación cualitativa

En la siguiente tabla hacemos una comparación de las propiedades de las tres arquitecturas estudiadas. En la tabla el signo + indica que tiene soporte directo, +/- indica que tiene soporte parcial, - indica que no esta implementado y N/A indica que no se aplica a la arquitectura particular.

ESB / PRO-PIEDAD	ESB Distribuido (Contenedores de servicio) con Workflow de negocios	ESB central (Servidor basado en hilos) con Workflow de negocios	ESB JBI	Arquitectura Deseada (Servidor basado en eventos) con Workflow Científico
Asincronía(en el servicio de mensajería)	+	+	+	+
Asincronía (en el administrador de procesos)	-	-	N/A	+
Escalabilidad (Utilizando eventos)	-	-	-	+
Administración y control (unificada para todos los servicios)	-	+	+	+
Tolerancia a Fallos	+	+/-	N/A	+/-
Administración SOA (creación y hospedaje de servicios en una sola plataforma)	-	+	+	+
Comunicación administrada sobre cualquier protocolo de transporte	-	+	-	+
Servicios de negocio y del bus administrados y desplegados en la misma plataforma	-	+	N/A	+

Tabla 3. Comparación de las propiedades fundamentales de las arquitecturas analizadas

8.2 Comparación cuantitativa

Las variables fundamentales de asincronía, escalabilidad y servicios básicos de procesamiento que planteamos al inicio de la investigación, junto con la arquitectura propuesta nos permiten definir los criterios de evaluación, así tenemos los siguientes criterios:

Asincronía (El proceso invocador no se bloquea luego de realizar una invocación y por esto puede recibir respuesta del sistema invocado en cualquier momento)

- Infraestructura de mensajería
- Workflow asincrónico

Escalabilidad (provee una plataforma adaptable y extensible al crecimiento futuro)

- Servidor de aplicaciones basado en eventos
- Soporte al cambio dinámico (Framework de plugins)
- Adición transparente de recursos(Para el despliegue de servicios complejos no es viable apagar el ESB temporalmente mientras nuevos componentes son adicionados o se realizan cambios a las definiciones)
- Servicios priorizados (volúmenes pesados de procesamiento hacen necesario priorizar ciertas actividades respecto a otras)
- Clases de servicios (volúmenes pesados de procesamiento requieren el elegir una clase de servicio específico en lugar de otros)

Servicios básicos de procesamiento

- Administración y control de mensajes (ver tabla 2)
- Creación y hospedaje de servicios unificados(ver tabla 2)
- Comunicación administrada sobre cualquier protocolo (ver tabla 2 fila 2)

A continuación comparamos los dos productos analizados utilizando los criterios de evaluación. Cada criterio se le dará una ponderación de acuerdo a la importancia de la variable independiente (de arquitectura) a la cual pertenece.

La calificación por criterio de evaluación será así:

0 tecnología no soportada

1 soporte deficiente

2 soporte bueno

3 muy buen soporte tecnológico

VARIABLE DE ARQUIT.	CRITERIO DE EVALUACIÓN	MULE 1.1	CBEI+ActiveMQ
ASINCRONÍA 40%	Infraestructura de mensajería 15%	2	2
	Workflow asincrónico 25%	0	3
		0.3	1.05
ESCALABI-LIDAD 30%	Servidor basado en eventos	3	3
	Soporte al cambio dinámico	1	2
	Adición transparente de recursos	1	1
	Servicios Priorizados	1	1
	Clases de servicios	1	1
		0.46	0.54
SERVICIOS BASICOS DE PROCESA MIENTO 30%	Administración y control de mensajes	0	3
	Creación y hospedaje de servicios unificados	0	3
	Comunicación administrada sobre cualquier protocolo	3	3
		0.3	0.9

Tabla 4. Evaluación cuantitativa

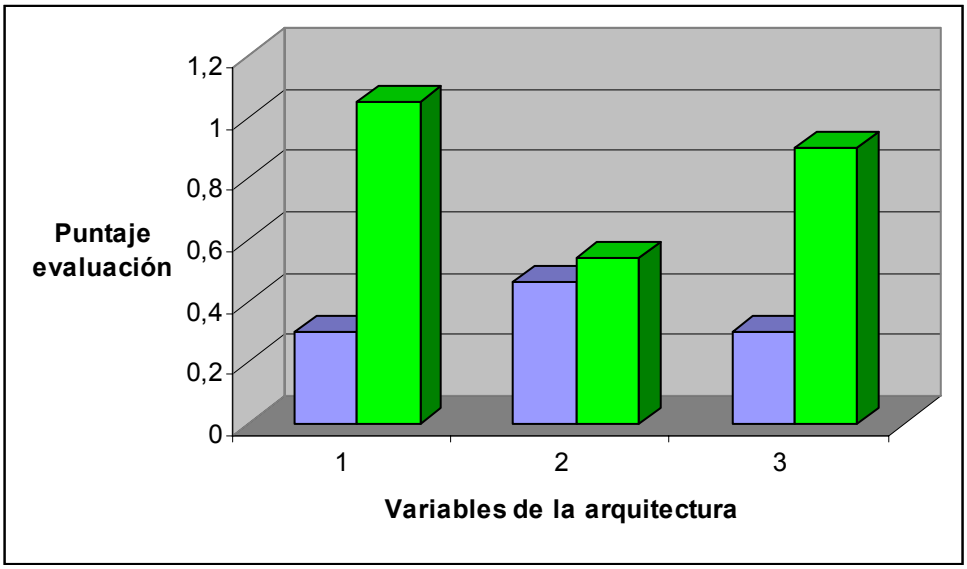


Figura 17. Puntaje de evaluación Vs Ariable de arquitectura

En el análisis podemos observar que Mule 1.1 y CBEI+ActiveMQ presentan casi la misma capacidad de escalabilidad, sin embargo en cuanto a la asincronía CBEI+ActiveMQ supera en cuatro veces el manejo de la asincronía de Mule 1.1 y supera en tres veces las herramientas básicas de procesamiento y administración de la solución total. Por tanto para nuestra solución de integración utilizaremos CBEI+ActiveMQ.

CONCLUSIONES

Se demostro que es posible elegir y poner en ejecución varias herramientas de software libre formando una arquitectura SOA, específicamente un Enterprise Service Bus.

En esta investigación se describió una arquitectura abstracta que cumple con los criterios de asincrónica, escalabilidad y servicios básicos de procesamiento necesarios para una organización que desea tener información actualizada en tiempo cercano al real.

A partir de la arquitectura planteada se caracterizaron dos herramientas libres Mule 1.1 y CBEI+ActiveMQ. Luego de la caracterización se tomo la decisión de implementar CBEI+ActiveMQ.

Cuando un lenguaje de patrones como el desarrollado por Gregor Hohpe es capaz De auto describirse facilita el trabajo del arquitecto. Sin embargo cuando los patrones están construidos basándose en un solo paradigma de desarrollo como por ejemplo mensajería, estos pierden capacidad de abstracción.

Se realizó una prueba de concepto en la que el orquestador a través de la mensajería envía un mensaje a una base de datos SQL Server 2005 con la orden de que ejecute un procedimiento almacenado que ha sido expuesto previamente como un servicio Web.

Los lenguajes dinámicos de scripting como Python proveen la flexibilidad necesaria para integrar módulos de software. Debido a su nivel de abstracción y facilidad de uso para el desarrollador.

En este trabajo se configuro el orquestador CBEI y la herramienta de mensajería ActiveMQ en el sistema operativo Linux Ubuntu 7.03 por la facilidad con que este sistema maneja los repositorios, pero en casi cualquier sabor de linux es posible configurar estas herramientas.

REFERENCIAS BIBLIOGRAFICAS

1. CRAGGS, Steve Best of Breed ESBs Identfying best of breed characteristics in an Enterprise Service Buses. 2003 p. 4.

2. PANG, Michael y MAHESHWARI Piyush. Benchmarking Message-Oriented Middleware TIB/RV vs SonicMQ University of New South Wales Sydney

3. SINHA Pradeep K. Distributed Operating Systems Concepts and Design. New York: IEEE Computer Society press, 1997. p. 170.

4. TANENBAUM Andrew S. Redes de Computadores. México: Pearson Education 2003 p. 528

5. MANOLESCU, Dragos - Anton, MICROWORKFLOW: A Workflow Architecture Supporting compositional object- oriented software Development. M.S, 1997 221 p. tesis. University of Illinois

6. BACON, Jean y HARRIS, Tim. Operating Systems concurrent and distributed software design. 1 ed. England: Pearson Education, 2003. p. 757-761.

7. CHAPPELL, David A. Theory in practice ENTERPRISE SERVICE BUS 1 ed. California: O'REILLY, 2004 p 77.

8. NEWCOMER, Eric y LOMOW Greg. Understanding SOA with web services 1 ed. Hagerstown: Addison Wesley 2005. p 8

9. HOHPE, Gregor y WOOLF, Bobby. ENTERPRISES INTEGRATION PATTERNS Designing, Building, and deploying messaging solutions. Boston, MA EEUU: Addison Wesley, 2004

10. Wil M.P. van der Aalst, Pattern Based Analysis of BPML(and WSCI) Australia: Queensland University of technology 2002-05

11. WOHED Petia et al. Pattern Based Analysis of BPEL4WS

12. RUSELL Nick; TER HOFSTEDE Arthur H.M.; EDMOND David. Workflow Data Patterns. Australia: Queensland University of technology p. 48

13. Cape Clear Software Inc. Service Centric Vs Message Centric ESBs a critical comparison of two ESB approaches

14. Cape Clear Software Inc. Cape Clear's ESB, How Cape Clear Software applies SOA and Web Service principles to deliver a proven ESB solution

15. TEN-HOVE, Ron y WALKER Peter. Java Business Integration (JBI) 1.0 Final Release Agosto 17, 2005 Sun Microsystems Inc.

16. HALTER, Richard. Message Broker Requirements

17. WELSH, Matt; CULLER David; BREWER, Eric. SEDA: An architecture for Well-Conditioned, scalable Internet Services. Computer Science Division, University of California, Berkeley.

18. LUDÄSCHER, Bertram y GOBLE Carole. Guest Editors' Introduction to the especial Section on Scientific Workflows. SIGMOD Record Vol.34, No. 3, Septiembre 2005

19. A Globus Primer Or, Everything you wanted to know about Globus, but Were Afraid To Ask. Describing Globus Toolkit Version 4.

20. JACKSON Keith R y BOVERHOF Joshua. PyGridWare Overview Lawrence Berkeley National Laboratory

21. LEA Doug, VINOSKY Steve y VOGELS Werner. Aschronous Middleware and Services IEEE Internet Computing. EdComputer Society South Western Collage Publishing 1999

22. http://www.compaq.com.co/servicios/aplicaciones_empresariales/enter_latencia.html

23. https://wiki.objectweb.org/celtix/Wiki.jsp?page=CeltixAndJBIDifferences

24. http://www.chwlund.com/?p=63

25. http://forum.java.sun.com/thread.jspa?forumID=512&threadID=740262

26. http://dsd.lbl.gov/gtg/projects/CBEI/

27. http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/tutorial/html/x284.html

28. http://dsd.lbl.gov/gtg/projects/vice/

ANEXO A. Java Business Integration (JBI)

JBI es una especificación que describe la manera en la que los componentes de integración, tales como servicios ESB, pueden ser conectados de forma portable estandarizada e independientemente del vendedor. JBI EG busca no requerir la totalidad de capas del servidor de aplicaciones J2EE y permitir a los vendedores, ya sean basados o no en servidores. Conectar sus componentes de integración de forma fácil e interoperable. La idea es promover un ecosistema en el que cualquier vendedor que tenga algo para ofrecer pueda hacerlo sin problemas de compatibilidad.

Aunque se diga que cualquier vendedor sea compatible, hay que reconocer que JBI esta íntimamente ligada a la arquitectura empresarial de Java. Como ellos mismos afirman en la especificación (JSR-208)[14]

Veamos la arquitectura propuesta por JBI.

[14] "This specification is considered to be complimentary to the J2EE™ Platform specification. JBI does not require the J2EE™ platform though there is a clear dependency on the J2SE™ platform. It is worth noting that the J2EE™ platform may choose to reference this specification in the future."

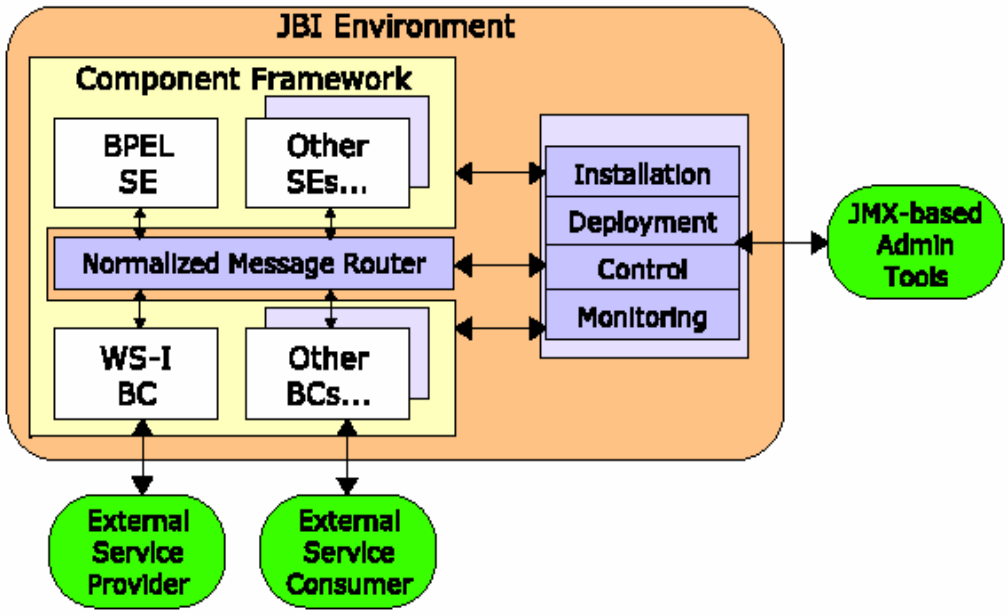


Figura 23 Vista de alto nivel de la arquitectura JBI

Servcie Engine (SE): Los SEs proveen lógica de negocio y servicios de transformación a otros componentes, así como también consumen dichos servicios. Los SEs pueden integrar aplicaciones basadas en Java así como otros recursos.

Binding Component (BC): Los BCs proveen conectividad a servicios externos al ambiente JBI. Esto puede involucrar protocolos de comunicación, o servicios suministrados por sistemas de información de la organización (recursos EIS). Los BCs pueden integrar aplicaciones (y otros recursos) que utilizan tecnologías de acceso remoto que no están disponibles directamente en Java.

Normalized Message Router(NMR): Se encarga de recibir mensajes de intercambio de todos los SEs y BCs y luego enrutarlos al componente JBI apropiado. El intercambio entre el NMR y los componentes JBI se hace a través de mensajes normalizados.

Los SEs y los BCs pueden funcionar como proveedores de servicios, consumidores o ambos. Lo importante es que se separan la lógica de negocio de la lógica de comunicaciones para reducir la complejidad.