UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

"FACULTAD 10"

TÍTULO: PROPUESTA DE UN ESB OPEN SOURCE PARA LA ARQUITECTURA DE LA INFORMATIZACIÓN DE LA UCI.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

Autores:

Yanet Martínez Rosado

Gretel Bernal Baró

Tutor:

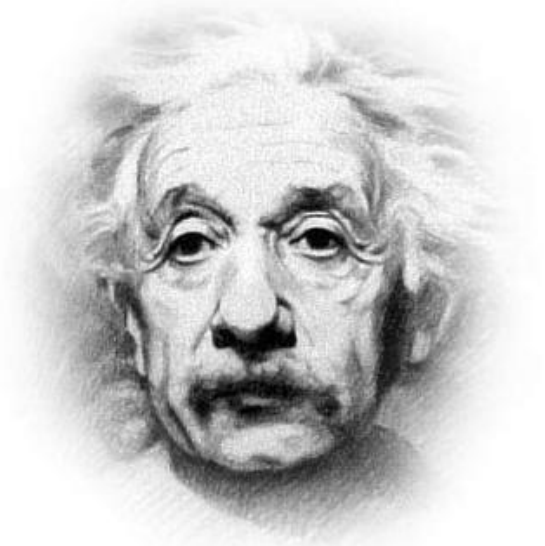
Ing. Guillermo Gómez Urquiza

Consultante:

Ing. Manuel Alejandro Gil Martin

Ciudad de La Habana, Junio 2008

"Año 50 de la Revolución"



"NUNCA CONSIDERES EL ESTUDIO COMO UNA OBLIGACIÓN SINO COMO UNA OPORTUNIDAD PARA PENETRAR EN EL BELLO Y MARAVILLOSO MUNDO DEL SABER."

ALBERT EINSTEIN

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Gretel Bernal Baró	**Yanet Martínez Rosado**	**Guillermo Gómez Urquiza**
---	---	---
Firma del Autor	Firma del Autor	Firma del tutor

DATOS DE CONTACTO

Síntesis del Tutor:

Ing. Guillermo Gómez Urquiza

Profesión: Ing. Informática

Categoría docente: Instructor recién graduado

Años de graduado: 1

Síntesis del Consultante:

Ing. Manuel Alejandro Gil Martín

Profesión: Ing. Informática

Categoría docente: Instructor

Años de graduado: 3

AGRADECIMIENTOS

A nuestras madres.

Por ser nuestras guías e inspiración, por ser nuestra razón de ser, por sus valiosos consejos para seguir el camino correcto, por estar siempre cuando las necesitamos.

A nuestros familiares.

Por formar parte de nuestras vidas y apoyarnos en todo momento.

A nuestros amigos.

Aquellos que demostraron ser amigos en las buenas y en las malas y nos brindaron su ayuda sin ningún tipo de interés.

Al tutor.

El Ing. Guillermo Gómez Urquiza por guiarnos y ayudarnos cada vez que lo necesitábamos.

Al consejero.

El Ing. Manuel Alejandro Gil Martín que nos apoyó en todo momento para que esta tesis terminara en tiempo y forma.

A Carlos.

Por brindarnos su ayuda, por estar presente en todo momento que lo necesitábamos, por ser tan buen amigo.

DEDICATORIA

Quiero dedicarle este trabajo a las personitas que más quiero en el mundo ellas son mi mamá Nelsy y mi hermanito Juampi, gracias a ellos he llegado tan lejos, quiero que sepan que los quiero mucho y que nunca los voy a decepcionar.

También quiero dedicarles este trabajo a todas las personas que me han ayudado, a mis primos, a mis tíos por parte de madre que siempre me han ayudado tanto, a mis amigos los de aquí y los de allá.

A Rody por enseñarme tantas cosas lindas, por estar siempre apoyándome en todo momento.

Yanet Martínez Rosado

En el transcurso de la vida hay siempre personas que te brindan apoyo y te sirven de sostén en el cursar de esta, por esto quiero dedicarle esta tesis a mi abuelita del alma por estar presente en todo momento, a mi madre por ser un ejemplo a seguir, por ser mi motor impulsor durante toda mi carrera, a mi hermano que tanto quiero.

A Darwin por ese optimismo que siempre me alienta a seguir adelante.

A mis familiares y amigos que tuvieron una palabra de sustento durante mis estudios.

Gretel Bernal Baró

RESUMEN

En estos tiempos donde se necesita conectar miles de aplicaciones que se ejecutan en una variedad de plataformas, escritas en una gran cantidad de lenguajes de programación, el Bus de Servicio Empresarial (ESB) surge como solución a esta problemática proporcionando una infraestructura flexible de conectividad para integrar las aplicaciones y los servicios que componen SOA(Arquitectura Orientada a Servicios).

Por estas razones los ESB se convierten en una de las opciones globalmente más utilizadas, además son productos robustos y fáciles de usar que brindan una alta fiabilidad. El ESB es la piedra angular que desde hace varios años se ha estado buscando al interior de las organizaciones como el corazón bombeador de datos e información a todo el ecosistema de aplicaciones de negocio; gracias a su modelo de publicación de servicios de negocio accesibles desde un enfoque multiprotocolo, y adaptación nativa hacia las plataformas de negocio de la organización.

Hoy en día es prácticamente imposible disponer de un modelo único de datos o plataforma única de aplicaciones. Estas dos fuerzas motivan a centrar la organización en una sólida plataforma de integración basada en servicios. Con este trabajo se persigue estudiar la trascendencia de los ESB, así como su situación en el mercado actual para proponer un modelo de integración de servicios utilizando software libre.

PALABRAS CLAVES

- Bus de Servicio Empresarial (ESB).
- Arquitectura Orientada a Servicios (SOA)

INDICE

INTRODUCCIÓN

INTRODUCCIÓN

Hoy en día con el gran desarrollo que vive la informática, la competencia en el mercado mundial provoca el perfeccionamiento cada vez más avanzado de nuevas tecnologías y software en el campo informático, promoviendo el intercambio de la información lo mismo entre usuarios conectados a la red mundial que a una red local. Por lo que se hace necesario el mejoramiento acelerado de las nuevas tecnologías para propiciar un excelente intercambio de información.

La utilización de las técnicas de computación en el mundo se ha convertido en la manera más eficaz de cualquier esfera de desarrollo de la sociedad. La habilidad para adaptarse rápidamente a las necesidades de un mercado cambiante es el objetivo clave para cualquier región del mundo. Para poder introducir estos cambios es necesario reconfigurar los recursos. Las empresas se están adaptando a marchas forzadas a los cambios continuos que se están produciendo en el mercado del software con la intención de volcarse en los servicios e integración de soluciones libres.

En la Universidad de las Ciencias Informáticas (UCI), guía del desarrollo informático en Cuba, se hace necesaria la selección de un software de conectividad e integración que ofrezca un conjunto de servicios que hagan posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.Esta Universidad tiende a buscar la mejor tecnología para crear una solución integral de intercambio de información, contar con una infraestructura común a cualquier proveedor y plataforma, lograr el envío de información en tiempo real o en línea y controlar la devolución de estado de envíos de información o devolución de errores de negocio.

Actualmente en la Universidad no existe un mecanismo capaz de resolver los problemas de integración y mensajería de los servicios Web. Se está realizando un estudio de los ESB existentes para permitir la integración, tanto de aplicaciones como de datos, principalmente, porque la información que se maneja está dispersa en distintas bases de datos, eso no permite tener la información completa en el

momento en que se necesita, implica que se deben buscar herramientas de interfaz, generar archivos y procesos, lo que alarga enormemente la toma de decisiones. La arquitectura orientada a servicios, en cambio, permite evaluar la posibilidad de integrar estas aplicaciones.

Con un ESB en la UCI como intermediario entre los diferentes servicios web se ahorra trabajo y tiempo, además se evitaría la dependencia entre los sistemas, haciendo la comunicación más confiable y segura. El ESB es capaz de guardar las peticiones en caso de que no estén disponibles en ese momento.

Si se instala un ESB se eliminaría la interconexión punto a punto de los servicios web, en este centro hay muchos servicios y este tipo de conexión a diferencia de los ESB hace que exista una gran dependencia entre los sistemas, la UCI es muy dinámica se hacen cambios en los sistemas frecuentemente con la interconexión punto a punto se afectan las aplicaciones que lo consumen. Con el uso de este tipo de servicio de mensajería se quiere en la Universidad de las Ciencias Informáticas sentar las bases para apoyar y sostener el futuro crecimiento de la misma.

La selección de un ESB Open Source para la arquitectura de la informatización de la UCI es de vital importancia para el correcto y eficaz desarrollo de la institución. La capacidad que tienen los Buses de Servicios Empresariales para cambiar el mercado informático es irreprochable y no deja de ser sorprendente.

Entre sus funciones se encuentra la traducción y transformación de formatos de mensajes, entrega y consumo transaccional de mensajes, garantía de entrega de estos, además de proporcionar una comunicación fiable entre los distintos recursos tecnológicos tales como aplicaciones, plataformas y servicios. La incorporación de una plataforma de ESB como solución va a permitir realizar tareas de una manera centralizada y versionada, garantizándose así que el catálogo de servicio que define e inspira la arquitectura SOA de la organización pueda crecer ordenadamente, y cumpla a cabalidad los principios de flexibilidad y reusabilidad que promete SOA.

Los ESB permiten que funcionalidades de negocio implementadas y residentes en sistemas legados y sistemas desarrollados sobre tecnologías cerradas puedan ser

fácilmente publicadas como servicios, y por ende ser promovidas como un potencial servicio reutilizable del portafolio de servicios empresariales.Una migración a esta arquitectura debe comenzar poco a poco, la instalación de un Enterprise Service Bus, Open Source cumple como objetivo final contar con una plataforma funcional y probada que permita extender SOA de manera directa y en mediano plazo al resto de la UCI.

Hoy por hoy con el desarrollo del software libre han ido apareciendo los buses de servicio empresarial Open Sources. En la UCI como ejemplo del desarrollo del software libre en Cuba se hace necesario el estudio y la instalación de uno de estos de manera que ofrezca y garantice confiabilidad y robustez en el intercambio de mensajes.

El **Problema Científico** del trabajo sería el siguiente:

¿Cómo resolver los problemas de integración y mensajería de los servicios Web de nuestra universidad?

Se ha planteado como **Objeto de Estudio**:

Proceso de gestión de los mecanismos capaces de resolver los problemas de integración y mensajería de los servicios Web.

El **Campo de Acción** sería el siguiente:

Proceso de gestión de los mecanismos capaces de resolver los problemas de integración y mensajería de los servicios Web en la UCI.

Para la realización de este trabajo de diploma el **Objetivo General** sería**:**

Proponer un Bus de Servicio Empresarial (ESB) que permita a la UCI la integración de todos sus servicios y que a la vez se le puedan agregar funcionalidades.

De este objetivo general se derivan los siguientes **Objetivos Específicos:**

- Estudiar los distintos tipos de ESB y sus funcionalidades.

- Recopilar documentación, y elaborar un informe acerca de su instalación y configuración.
- Listar los ESB Open Sources, con sus características fundamentales y diferencias.
- Proponer un ESB para la UCI.

Por tanto como **Hipótesis** se plantea:

Si se selecciona un Bus de Servicio Empresarial (ESB) entonces se pueden resolver los problemas de integración y mensajería de los servicios Web en la UCI.

Para dar cumplimiento a los objetivos trazados se realizan las **Tareas de la Investigación** que a continuación se presentan:

- Realizar un estudio detallado de las funcionalidades de los ESB.
- Comparar los ESB Open Source.
- Proponer una alternativa para instalarlo en los servidores de producción

Este trabajo está estructurado en tres capítulos:

Capítulo1: Fundamentación Teórica

En este capítulo se realiza un estudio de los ESB, los principales conceptos y la evolución de estos en el mercado.

Capítulo2: Comparación entre los ESB Open Sources

Con la aparición de los ESB también han ido apareciendo soluciones libres en este capítulo se hace una comparación entre muchos de estos ESB Open Sources para ver cual es el que se podría adaptar mejor a las características de la UCI.

Capitulo3: Propuesta del ESB Open Source a instalar en los procesos productivos de la UCI.

Este capítulo está dirigido a explicar el ESB Open Source que se tiene como propuesta para instalar en los procesos productivos de la universidad. Se describe

el por qué de este ESB como alternativa, sus características fundamentales y los pasos a seguir para su correcta instalación.

Resultado Esperado:

Obtención de una propuesta de un Bus de Servicio Empresarial (ESB) para la instalación en la Universidad de las Ciencias Informáticas (UCI), el cual se pueda adaptar a los servidores de la UCI y resuelva muchos de los problemas de integración y mensajería que existen hoy en día en la universidad.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción al mundo de los ESB.

1.1 Introducción

Si se hace una encuesta a todos los especialistas de integración y arquitectura sobre cual sería la plataforma de integración más usada sin lugar a dudas el término ESB sería la respuesta más predominante. No se puede hablar de los ESB sin antes haber mencionado otros términos como SOA (Arquitectura Orientada a Servicios), UDDI (Repositorio de Servicios), BMP (Administración de los Procesos de Negocio) y BAM (Actividades de monitoreo y negocio). En el presente capítulo se perfilan con más detalles, estos conceptos como fundamentación teórica base para obtener una comprensión más cabal del problema a resolver y de su entorno. También se hace una breve explicación de cada uno de ellos, de esta manera ir introduciendo hasta caer en el tema de los ESB.

1.2 Arquitectura Orientada a Servicios (SOA)

La Arquitectura Orientada a Servicios (SOA), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario. SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación. (Guinea de Salas, Alejandro y Jorrín Abellán, Sergio).

En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La complejidad de SOA podría variar mucho, dependiendo de la cantidad de servicios y las funcionalidades de estos, así, se puede tener aplicaciones que utilicen servicios y servicios de negocio que utilicen servicios, para su ejecución. Los componentes que conforman SOA son varios (Ver Anexos 1).

1.3 Repositorio de Servicios (UDDI)

UDDI son las siglas del catálogo de negocios de Internet denominado Universal Description, Discovery, and Integration. El registro en el catálogo se hace en XML (Lenguaje de marcas extensible). UDDI es una iniciativa industrial abierta (sufragada por la OASIS), en el contexto de los servicios Web. El registro de un negocio en UDDI tiene tres partes:

- Páginas blancas - dirección, contacto y otros identificadores conocidos
- Páginas amarillas - categorización industrial basada en taxonomías
- Páginas verdes - información técnica sobre los servicios que aportan las propias empresas.

UDDI es uno de los estándares básicos de los servicios Web cuyo objetivo es ser accedido por los usuarios y clientes y dar paso a documentos WSDL (Web Services Description Language), en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los servicios Web del catálogo de registros. (Informatización, 2007)

Un repositorio de servicios proporciona facilidades para descubrir servicios y adquirir la información necesaria para su uso.

- Información de contrato.
- Localización.
- Personas de contacto.
- Acuerdos a nivel de servicio (SLA)
- Descubrimiento dinámico.

1.4 Administración de los Procesos de Negocio (BPM).

Se llama Administración de los Procesos de Negocio a la metodología empresarial cuyo objetivo es mejorar la eficiencia a través de la gestión sistemática de los procesos de negocio (BPR, Reingeniería de procesos de negocios), que se deben modelar, automatizar, integrar, monitorizar y optimizar de forma continúa. Como su nombre sugiere se enfoca en la administración de los procesos del negocio.

CAPÍTULO 1

La administración de procesos de negocios permite diseñar, desplegar, ejecutar, analizar y optimizar procesos de negocios. La principal actividad es coordinar el flujo de tareas, el acceso a los recursos y el intercambio de información entre empleados, clientes, proveedores y socios de negocios. (Technology for solutions, 2007)

A través del modelado de las actividades y procesos puede lograrse un mejor entendimiento del negocio y muchas veces esto presenta la oportunidad de mejorarlos. La automatización de los procesos reduce errores, asegurando que los mismos se comporten siempre de igual manera y dando elementos que permitan visualizar el estado de ellos.

La administración de los procesos permite asegurar que los mismos se ejecuten eficientemente, y la obtención de información que luego puede ser usada para mejorarlos. Es a través de la información que se obtiene de la ejecución diaria de los procesos, que se puede identificar posibles ineficiencias en estos, y actuar sobre las mismas para optimizarlos.

Para soportar esta estrategia es necesario contar con un conjunto de herramientas que den el soporte necesario para cumplir con el ciclo de vida de BPM. Este conjunto de herramientas son llamadas Business Process Management System y con ellas se construyen aplicaciones BPM.

El concepto Administración de los Procesos de Negocio se aplica tanto desde un punto de vista tecnológico como de gestión. Desde una perspectiva de gestión es un enfoque estructurado que emplea métodos, políticas y métricas para gestionar y optimizar de manera continua las actividades y procesos de una organización.

Desde el punto de vista tecnológico, agrupa una serie de herramientas de software para el modelado, ejecución y monitorización de los procesos de negocio. Permite encadenar los servicios para ganar en eficiencia y asegurar la mejora continua de los mismos. Además accede a la modificación rápida en función de la demanda cambiante y reduce los costos de mantenimiento.

1.5 Actividades de monitoreo y negocio (BAM)

Business Activity Monitoring (BAM) o Monitoreo de las Actividades del Negocio es un conjunto de tecnologías orientado a brindar acceso en tiempo real a los indicadores de desempeño críticos de la empresa, con el objetivo de mejorar la velocidad y efectividad de sus operaciones y procesos. Se trata de una solución que apunta específicamente a que los encargados de gestión dispongan de información para tomar mejores decisiones y puedan ser alertados automáticamente ante cualquier problemática que ponga en riesgo al negocio. (CIENTEC, 2007)

La mayor parte de los directivos están tan centrados en los problemas cotidianos que apenas tienen tiempo de ocuparse de los procesos que pueden mejorar para extraer mayor valor de sus negocios. Los problemas que saltan a la vista quizás se solucionen, pero muchas de las cuestiones más costosas quedan latentes bajo la superficie. Además, el tiempo de estos directivos muchas veces no deja espacio a la innovación.

BAM permite al negocio identificar y comprender los errores para emprender medidas que los resuelvan de inmediato. Lo más asombroso es que el análisis de procesos en tiempo real permite localizar las áreas más apropiadas para su mejora.

La monitorización de procesos permite obtener información de negocio y técnica, con el fin de identificar patrones de utilización, situaciones de riesgo e indicadores de desempeño. Además del almacenamiento y agregación de datos a partir de KPIs (key performance indicators), las soluciones de monitorización deben tener en cuenta:

- La usabilidad de la aplicación de consulta para expertos de negocio. El perfilado según tipos de usuarios. Los métodos de envío de alertas (email).
- Los mecanismos de captura de información, ofreciendo la capacidad de procesar datos generados por diversas fuentes, incluidas aquéllas ajenas al propio BPM.
- El rendimiento de la solución, minimizando el impacto en la escalabilidad y consumo de recursos en entornos de producción.

1.6 Bus de Servicio Empresarial (ESB).

La intersección de la arquitectura orientada a servicios con la integración de aplicaciones y el modelado de procesos de negocio, dan lugar a un nuevo producto denominado Bus de Servicio Empresarial. Es un concepto antiguo, pero que adquiere fuerza de nuevo gracias a la facilidad y bajo coste con el que se puede implantar actualmente debido a los avances tecnológicos que se han producido.

Con el auge de las tecnologías SOA y EDA (Event Driven Architecture) se perfila la necesidad de un nuevo componente en la ya compleja infraestructura de los servicios de información: el bus de servicios de empresa. Dicho componente viene a cubrir un espacio creado por la necesidad de permitir la comunicación entre componentes o servicios de la empresa.

Hasta la fecha, el papel de integrador venía dado de la mano de EAIs (Enterprise Application Integration), una tecnología que permitía comunicar distintos recursos informáticos para poder hacer uso de ellos conjuntamente. El problema que tiene esta tecnología es que los tiempos de desarrollo son largos, los proyectos conllevan un desembolso importante y existe una absoluta dependencia del fabricante.

Debido a esta difícil situación respecto a los EAIs, la industria ha evolucionado hacia lo que se ha pasado a llamar ESB. El corazón de un ESB es un MOM (Message-Oriented Middleware); lo que permite que la comunicación entre distintos componentes se haga de manera transparente, fiable y asíncrona (en el caso de que fuese necesario el sincronismo también deberá permitirlo el ESB).

Además del sistema de mensajería hacen falta conectores para comunicar los distintos recursos de la empresa con el ESB. Dichos conectores permitirán exponer los recursos de la empresa como servicios web dentro del propio ESB (de hecho la definición de los mismos se queda en el nivel de abstracta, sin necesidad de definir los puertos).

De manera que la comunicación interna del ESB se realiza con XML como formato impuesto, permitiendo con ello acceder de manera sencilla y rápida a la información

que transita por dentro. Este último hecho permite aplicar tecnologías como BAM sobre los datos que transitan por un ESB.

Es responsabilidad de un ESB el enrutamiento de los mensajes y la orquestación de los procesos. Por enrutamiento se entiende el proceso de recibir la entrada de un mensaje y decidir la salida o salidas que el mensaje debe tomar (pudiendo haber transformación del contenido, gracias al hecho de que es XML, se pueden usar tecnologías como XSLT (Lenguaje de Transformación basado en Hojas de Estilo)). La orquestación es el esqueleto de una aplicación compuesta, en la que a través de lenguajes formales se permite definir el flujo de actividades y estados por los que ha de pasar un proceso de empresa para su realización (un ejemplo bastante extendido es BPEL (Lenguaje de Ejecución de Procesos de Negocio)).

Existe la creencia de que un ESB no tiene por qué ser un producto final, sino más bien una manera de enfocar una arquitectura SOA. Sin embargo, lo cierto es que en el mercado ya existen tantas especificaciones que hacen de un ESB un contenedor de componentes estándar (JBI (Integración de negocios de Java): JSR 208(Java Specification Request)) como productos que las implementan.

A manera de resumen se puede decir que el ESB es un elemento de software, un middleware, una infraestructura basada en los mensajes y en estándares abiertos, que proporciona servicios para la construcción de arquitecturas más complejas basadas en eventos y en un motor de mensajería (el BUS).El ESB permite la integración de aplicaciones de forma rápida, directa y basada en estándares. Es una suite de productos independientes de la infraestructura que facilita el procesado, la transformación de datos, el enrutamiento y la orquestación de procesos usando Web Services. (Gómez Rubio, Álvaro, 2007)

Está integrado por aplicaciones proveedoras de servicios. Cada una de las aplicaciones puede ser independiente de las demás. Esta posibilidad de integración es independiente del modo de licencia del software, libre o propietario. Esta arquitectura permite, por ejemplo, a través del ESB utilizar con una herramienta de software propietario unos servicios desarrollados sobre plataformas de software libre.

Puede hacer uso de la infraestructura existente de servidores de aplicaciones, transportes y datos. Los ESB se están convirtiendo en un primer paso clave para el establecimiento de una SOA empresarial. Estos constituyen los cimientos de una SOA y pueden complementarse con capacidades de productividad adicionales, como la orquestación de servicios y los registros.

1.6.1 Aplicaciones y características del ESB.

Algunas de las funcionalidades básicas de un ESB.

Funcionalidades más importantes	Descripción
Servicio de registro	Un servicio cliente que invoque al proveedor de servicio solo necesita saber que el servicio existe; el cliente no necesita saber dónde se está ejecutando el servicio
Conversión de protocolos de transporte	Un ESB debe poder como una sola pieza integrar aplicaciones con diferentes protocolos de transporte como el HTTP (protocolo de transferencia de

	hipertexto) para JMS (servicio de mensajes de Java), SMTP (protocolo simple de transferencia de correo) para TCP (Protocolo de Control de Transmisión).
Transformación de mensaje	La tarea de un ESB es dirigir mensajes de un servicio al siguiente, hay ocasiones en que el formato de los datos de un servicio no satisface los requisitos del siguiente servicio. Por ese motivo, el ESB debe ser capaz de transformar los datos de un formato a otro.
Enrutamiento basado en el contenido	Existen dos tipos de enrutamiento dentro de un ESB.
Seguridad	La autenticación, la autorización y la

	funcionalidad de código deberían ser previstas por un ESB para el asegurado de los mensajes entrantes y así impedir el uso malicioso del ESB

- **Servicio de registro:**

Probablemente, la forma más elemental de mostrar el primer núcleo de una funcionalidad ESB es la idea de la Transparencia de ubicación, el servicio de comunicación de los consumidores a un prestador de servicios a través del ESB. El consumidor no necesita saber la ubicación real de los servicios del Proveedor. Esto significa que el consumidor está disociado de los servicios por parte del proveedor, de manera que de haber una nueva localización de un servidor prestador de servicios no tiene ningún impacto en el servicio al consumidor. La funcionalidad básica del ESB que proporciona esta asistencia es el servicio de registro.

La idea de un buen servicio de registro es que te brinda la posibilidad dentro del ESB para configurar dinámicamente los lugares de servicio. De esta manera si se efectúa un cambio en la ubicación del servidor no se necesita de un reinicio del ESB.

Resumiendo se puede decir que con la mediación entre servicios, un servicio cliente que invoque al proveedor de servicio solo necesita saber que el servicio existe; el cliente no necesita conocer dónde se está ejecutando el servicio. El ESB localiza el servicio cuando se invoca. Esto proporciona un cierto nivel de virtualización de los servicios y de transparencia de las ubicaciones, de forma que si un equipo falla, o si se cambia la ubicación de un proveedor de servicio, no es preciso notificar el cambio a cada uno de los clientes individuales. Esto puede contribuir significativamente a la reducción de los costes de gestión y a minimizar los riesgos.

- **Transparencia del transporte:**

En los enfoques tradicionales de la integración punto a punto todos los componentes y objetos están muy estrechamente acoplados. En SOA, los servicios están repartidos por todo el entorno y su acoplamiento es menos estricto, gracias a la transparencia de las ubicaciones. Además de apoyarse en la transparencia de las ubicaciones para conectar clientes y proveedores de servicios, el ESB también proporciona protocolo de trasporte físico para hacer posible la comunicación entre servicios utilizando transportes diferentes.

- **Soporte multiprotocolo:**

Debido a que plantea cuestiones de fiabilidad inherentes y solamente funciona bien con patrones de intercambio de mensajes (MEP) sincrónicos, el modelo de transporte HTTP no satisface los requisitos de todos los servicios y aplicaciones. Por ejemplo, el servicio de mensajes de Java (JMS) además de poseer características asincrónicas, ofrece más fiabilidad en el transporte que HTTP. Para compatibilizar el comportamiento de las aplicaciones individuales, algunos sistemas recurren a SOAP a través de JMS.

También se usan otros tipos de modelos de transporte, entre los que se cuentan los sistemas de transporte propietarios de algunos de los principales proveedores de sistemas y soluciones de planificación de recursos empresariales. El ESB necesita, por lo tanto, ser capaz de soportar muchos tipos de sistemas de transporte para integrar sistemas dispares y gestionar el transporte de comunicaciones complejas eficazmente.

- **Mensajería distribuida:**

El núcleo del ESB lo constituye una aplicación de middleware orientada al mensaje. Este núcleo proporciona un método de transporte fiable y distribuido que emplea un mecanismo de almacenamiento y reenvío gracias al cual se garantiza la entrega de los mensajes incluso en caso de anomalías en la red.

- **Calidad de servicio:**

La entrega de los mensajes y la fiabilidad de los servicios de invocación son funciones de misión crítica de cualquier sistema. Aun los servicios Web por sí solos no ofrecen un servicio de entrega garantizada. Un ESB, por otro lado, puede proporcionar un servicio de alta fiabilidad garantizando la entrega del mensaje de principio a fin que va más allá de la fiabilidad que puede ofrecer transportes como JMS.

- **Patrones de intercambio de mensajes:**

En la actualidad, la mayoría de los ESB se basan en un paradigma de solicitud/respuesta usando SOAP sobre HTTP; esto significa que el servicio cliente lanza un mensaje de solicitud al usuario y espera a recibir la respuesta. Esto se conoce como un MEP sincrónico.

Sin embargo, en el MEP de publicación/suscripción, el servicio cliente puede enviar un mensaje y suscribirse a la respuesta, en lugar de esperar a recibirla. El MEP de publicación/suscripción puede responder de forma más eficaz a eventos en un contexto empresarial, en particular cuando el ciclo de vida de una acción de servicio tiene lugar durante períodos de tiempo prolongados. Un ESB debe ser capaz de manejar ambos paradigmas.

- **Enrutamiento basado en el contenido:**

Existen dos tipos de enrutamiento dentro de un ESB. El primer servicio de enrutamiento se produce cuando la invocación de un servicio entra en el ESB y éste encamina la respuesta al proveedor de servicio apropiado, sin necesidad de que el servicio cliente conozca la ubicación del proveedor del servicio. Así es cómo se logra la transparencia de las ubicaciones que antes se ha comentado.

El otro tipo de enrutamiento es basado en el contenido, introduce una serie de reglas o lógica de negocio que se aplica al contenido del mensaje en la etapa del enrutamiento y que hacen posible que el ESB encamine los mensajes a proveedores de servicio específicos basándose en su contenido; dando prioridad,

por ejemplo, a los pedidos de determinados clientes o marcando los pedidos de gran tamaño para darles un tratamiento especial.

Esto ofrece a las empresas un servicio muy valioso, ya que puede contribuir a reducir el coste de la gestión de la información, garantiza que se respeten los acuerdos a nivel de servicio y permite a las empresas centrarse en actividades para mejorar la satisfacción de sus clientes.

- **Transformación de mensajes:**

Además del apoyo a un conjunto de protocolos de transporte, la integración entre la aplicación de un servicio consumidor y un proveedor de servicios a menudo necesita una transformación del formato del mensaje. La funcionalidad básica del ESB con la que ayuda a cambiar el formato del mensaje es la transformación de mensajes.

Una tecnología para transformar un mensaje de la fuente al formato de destino es, por ejemplo XSLT. XSLT es una recomendación de la W3C (World Wide Web Consortium) que es ampliamente adoptado en la integración de la industria, que asegura que la transformación del mensaje de un modelo a otro por medio de un fichero XSLT sea utilizable en la mayoría de los ESB disponibles en el mercado.

La mayoría de los ESB suelen incorporar motores de transformación y parseadores de mensajes que permiten y facilitan la transformación de los mismos. La manera más común suele ser por medio de un fichero XSLT de Transformación de mensajes, aunque algunas herramientas utilizan otros sistemas, incluso propietarios.

Por ejemplo, Si se dispone de un modelo de datos común de arquitectura y se quiere invocar un servicio web con un modelo de datos particular y único. El ESB permite realizar la transformación del mensaje de un modelo a otro por medio de un fichero XSLT (o cualquier otro sistema que utilice) y así desacoplar completamente la implementación de uno y otro. De esta manera, un futuro cambio de uno u otro modelo, solo debería, en principio, aceptar a la transformación del mensaje, nunca a las implementaciones de ambos modelos involucrados.

- **Configuración y no codificación:**

Una de las características más ventajosas de un ESB es que suele realizarse todo por medio de configuración y no codificación. Independientemente del producto que estemos utilizando (hoy en día la mayoría incluyen IDEs (Entorno de Desarrollo Integrado) gráficos que facilitan la configuración de los elementos del ESB; aunque algunos pueden necesitar de configuración por medio de ficheros y manualmente) suele ser suficiente con la configuración de los componentes del ESB que se vaya a utilizar. No suele necesitarse la codificación, lo que limita el número de errores y da fiabilidad al bus que se monte. No obstante, si fuese necesario, la codificación (sea por medio de Java, .NET, o cualquier otro lenguaje) suele ser posible para casos concretos en que los componentes del ESB manejado no implementen o faciliten la configuración deseada.

- **Proxy de Servicios:**

Una de las mayores ventajas del ESB es la de crear una capa de proxy por encima del servicio. Esto permite abstraer su implementación. El caso más práctico sería el de un servicio web con dos operaciones. Una de ellas por ejemplo, permite la consulta online en una base de datos de libros. La otra operación permite la compra de un libro y por tanto requiere del manejo de información confidencial como puede ser una tarjeta de crédito.

El ESB permite generar un proxy por encima de dicho servicio y dar lugar a dos nuevos servicios (que serán los que realmente se permitan invocar a través de aplicaciones cliente). Uno de los dos proxy's podría ser el servicio con la operación de consulta (sin requisito adicional). El otro servicio podría resultar en un proxy con requisito de seguridad añadido sobre la operación del servicio original. El proxy podría ser requerido para su invocación, por ejemplo, por medio de HTTPS que llamaría finalmente a la operación del servicio original.

- **Conversión de protocolos:**

Como ya se ha dicho, una de las características del ESB es su funcionalidad multiprotocolo. Esto quiere decir, que permite la conexión a distintas aplicaciones o sistemas por medio de numerosos protocolos. Además, suelen incorporar la posibilidad de añadir conectores a todo tipo de aplicaciones que pueden ser desarrollados por el mismo fabricante del ESB, por terceros o incluso por el propio desarrollador a través de un API (Interfaz de Programación de Aplicaciones) estandarizada.

El ejemplo más práctico sería un servicio web por HTTP, al que se le quiere dar fiabilidad por medio de mensajería JMS. El ESB sería el encargado de exponer el servicio de manera fiable a través de un proveedor de JMS y luego realizar la conversión de protocolo necesaria para la invocación vía HTTP.

- **Auditorías y Logs de Mensajes:**

Otra característica importante del ESB es su función como auditor de mensajería y registrador de mensajes. La comunicación de información a través del ESB permite centralizar sistemas de log y persistencia de información susceptible de ser auditada. Además, luego con herramientas de BAM o incluso Business Intelligence pueden explotarse dichos datos y realizar monitorizaciones exhaustivas que permitan la optimización del ESB, la Arquitectura SOA y en definitiva el negocio de la empresa.

- **Manejo de Excepciones:**

Generalmente los servicios web tienen un comportamiento de respuesta correcta y en algunos casos pueden devolver excepciones de manera controlada (o no). El ESB permite gestionar las excepciones de una manera estandarizada y realizar las correcciones oportunas al ocurrir dichos errores.

El ejemplo más práctico consistiría en definir un modelo estándar de nuestra arquitectura para el manejo de excepciones. Por medio del ESB se puede tratar cualquier tipo de error (independientemente del protocolo o el modelo de datos de

los servicios involucrados) de una manera común a todo el entorno y así poder unificar criterios y facilitar la internacionalización de mensajes de error.

- **Seguridad:**

Un ESB debe proporcionar los medios para autenticar y autorizar a los mensajes entrantes. Deben ser capaces de proporcionar el cifrado de mensajes para protegerlos en caso que pueden ser interceptados para propósitos maliciosos. Cuando en un ESB no se aplica un modelo de seguridad para su entorno, todos los mensajes enviados al punto de partida de una corriente de integración, como una cola de mensajes o un servicio web, es capaz de comenzar este flujo, posiblemente, con mensajes maliciosos.

En la mayoría de las situaciones el ESB no es accesible desde fuera de las fronteras de la organización, debido a la configuración del cortafuego. Esto significa que los posibles mensajes maliciosos sólo pueden llegar desde el entorno de IT (Tecnología de la Información) de la organización. Pero cuando un ESB ofrece puntos de partida de la integración a las solicitudes de fuera de los límites de la organización, la seguridad es aún más importante. Hay muchas tecnologías disponibles para proporcionar la funcionalidad de seguridad de un ESB.

La mensajería infraestructura utilizada por un ESB es un punto de partida para la aplicación de la seguridad. El ESB tiene múltiples capacidades, además de la conectividad de mensajería. Por lo tanto un ESB debe prever un mecanismo de autenticación. Se puede pensar en tecnologías como Java de autenticación y autorización de servicios (JAAS) y WSSecurity o Acegi, un marco de seguridad que se integra perfectamente.

- **Acceso a Repositorio de Servicios:**

Un Repositorio de Servicios (no confundir con UDDI) permite acceder a metainformación de los servicios integrados que facilita el control de los mismos y el poder aplicar políticas y condiciones de una manera centralizada y jerarquizada

(entre muchas otras características, muy ligadas a veces con la gobernabilidad de la arquitectura).

Un ejemplo práctico consistiría en resolver de manera dinámica los endpoints de invocación de los servicios por medio del acceso a un repositorio de servicios que diera dicha información en tiempo de ejecución. De esta manera, por ejemplo, cambiar a la invocación de un servicio en pruebas u otro de respaldo, sería automático por medio de configuración del propio repositorio.

- **Validación, Enriquecimiento, Transformación y Operación de Mensajes:**

Se puede decir, que estas son las 4 características estrella de un ESB:

Validación de la información, de los mensajes entrantes, sobre los que se pueden aplicar distintas condiciones y verificaciones que hagan cumplir los requisitos establecidos.

Enriquecimiento de los mensajes, posibilidad de añadir información o metainformación adicional para cumplir las necesidades de integración en cada momento.

Transformación de los mensajes, normalmente entre los diversos modelos de datos del conjunto de servicios y aplicaciones a integrar. Operación de los mensajes o enrutación en función de directivas preestablecidas o a partir de metainformación incluso en la propia comunicación.

1.6.2 Componentes y estándares de un ESB

- **Estándares de transporte**: HTTP(S), FTP, SMTP, JMS, JMX (Java Management Extensions), JDBC (Java Database Connectivity) y RPC (Remote Procedure Call) SOAP.
- **Estándares de Conectividad:** responsables de exponer las interfaces a la red. Incluyen entre otros Simple ObjectAccess Protocol (SOAP), Universal Description, Discovery, and Integration (UDDI), y Web Services Description Language (WSDL). SOAP provee RPC, XML, UDDI proporciona un registro

de Servicios Web, WSDL es un Interface Description Language (IDL) para servicios web. Todos utilizan XML como formato estándar de datos.

- **Estándares de Portabilidad:** Web Services, J2EE y .NET.
- **Estándares de Transformación:** como XSLT, Xpath, Xquery.
- **Estándares de Seguridad:** SSL (Secure Sockets Layer), Ldap (Lightweight Directory Access Protocol), JAAS (Java Authentication and Authorization Service), JSSE (Java Secure Socket Extension), WS-Security (Seguridad en Servicios Web), Certificados.
- **Tipos de envíos**: Asíncrono, síncrono, de publicación, de suscripción.
- **Tipos de mensajes:** correo (con y sin Attachements), mensaje JMS con cabeceras, SOAP, XML.
- **Enrutamiento de peticiones** (CBR (Constant Bit Rate))

1.6.3 Características comunes de los ESB

- **Configuración por sobre programación:**

Los ESB suelen contar con una característica particular: su uso en forma declarativa. Esto reduce la aparición de errores de software (bugs) por errores de programación y simplifica la prueba de los servicios, elude los procesos de despliegue y centraliza en una sola capa la responsabilidad de la interconexión de los sistemas, lo que simplifica la administración.

- **Orquestación de servicios:**

A través de lenguajes estándares como BPEL, permiten la coordinación y organización de la invocación de los distintos servicios. De esta manera es posible definir procesos de negocio dentro de los ESB.

- **Transformación de datos:**

Es la facultad de realizar mapeos de datos entre distintos contextos de aplicación. Mediante mapeos es posible “traducir” un dato del contexto de una aplicación en particular, de tal forma que sea reconocido por otro sistema.

- **Federación de datos:**

Además de mapear datos entre distintos orígenes de datos para lectura, se incorpora el concepto de federación. Básicamente es la capacidad de realizar modificaciones en cascada de los datos mapeados. Lo que simplifica notablemente la lógica de persistencia a aplicar entre los distintos sistemas.

- **Sincrónico + Asincrónico:**

Ya que la duración de algunos de los procesos expuestos en forma de servicio puede ser considerable (desde días hasta semanas), es importante contar con la posibilidad de ejecutar tareas en forma asincrónica y en paralelo. El procesamiento sincrónico obliga a esperar el resultado de una tarea para continuar con la siguiente, que es conveniente de ser aplicada, por ejemplo, en acciones que requieren interacción con usuarios.

- **Web Services:**

Es la técnica más difundida en los últimos tiempos para exponer procesos de negocio en forma simple, a través de HTTP.

- **Basado en estándares:**

El intercambio de información normalmente se realiza en documentos basados en XML

- **Ruteo:**

La mayoría de los ESB permiten definir procesos, ofreciendo alguna lógica de flujos, como estructuras iterativas, condicionales, procesamiento en paralelo, invocación de otros servicios, entre otras.

- **Seguridad Incorporada:**

Normalmente cada ESB trae incorporada su propia seguridad referida a la autenticación y autorización de usuarios. También soportan protocolos de comunicación segura, mediante HTTPS.

- **Interfaz de usuario:**

Algunos ESB brindan la posibilidad de administrar el bus a través de una interfaz amigable. Esta interfaz está presente comúnmente en los productos pagos.

1.6.4 Características que deben cumplir las interacciones realizadas de manera interna al propio bus.

El Bus de Servicios de Empresa, es un entorno de aplicación basado en mensajes que permitirá a las aplicaciones acceder a los servicios tanto desde dentro de la propia empresa, como externamente de forma remota. En él se realiza una labor crucial en toda arquitectura SOA, posibilitando la publicación de los servicios y la normal interacción entre ellos.

Las interacciones realizadas de manera interna al propio bus, deberán cumplir las siguientes características:

- **Publicación de Servicios:**

Los servicios podrán contactar entre sí a través del intercambio de documentos definidos en un formato estándar (normalmente XML), e invocar sus respectivos modelos de objetos y clases.

- **Middleware Orientado a Mensajes:**

El transporte debe ser fiable, eficiente y robusto en todo el canal de intercambio de datos, es decir, de extremo a extremo.

- **Encaminamiento Inteligente:**

El encaminamiento de mensajes deberá estar basado en su contexto y contenido, así como en reglas de proceso de negocio.

- **Modelo Asíncrono:**

Los servicios interactuarán entre sí siguiendo un enfoque asíncrono, que asegure que aún no habiendo terminado de procesar las peticiones de servicio previas, seguirá procesando nuevas solicitudes. Esta comunicación asíncrona se logrará utilizando colas de mensajes, interacciones punto a punto entre clientes y servidores.

- **Posibilidad de Interceptación:**

Los servicios podrán interactuar entre ellos bien de forma directa, o utilizando nodos intermedios capaces de interceptar los mensajes intercambiados, inspeccionándolos, filtrando y aplicando las políticas que se consideren oportunas. También serán capaces de operar como proxy para servicios.

- **Modelo de gestión de estados:**

Los servicios podrán mantener de manera opcional información sobre el estado de su sesión, de forma temporal o persistente, recogida bien en los extremos o en los nodos intermediarios. Para ello sería necesario enviar en la cabecera de los mensajes intercambiados dicha información de estado.

- **Modelo Fiable:**

Deberá poder asegurarse un modelo fiable de entrega única de mensajes, aunque finalmente se decida optar por otros basados en sistemas de mejor esfuerzo y por tanto no seguros.

1.7 Conclusiones parciales

En este capítulo se exponen una serie de conceptos y definiciones que están relacionados con el ESB de esta manera se podrá tener una visión más clara para poder darle solución a la problemática existente, se ha plasmado la importancia del uso del ESB exponiendo sus principales funcionalidades y características, además de la las ventajas que estos proporcionan cuando se respaldan en ellos.

CAPÍTULO 2: COMPARACIÓN ENTRE LOS ESB OPEN SOURCES

2.1 Introducción

Siempre que en una compañía se piensa en dar los primeros pasos dentro del mundo SOA, vienen a la cabeza los principales fabricantes de soluciones existentes en el mercado. Tanto Microsoft, como IBM, Oracle y Sun ofrecen sus correspondientes paquetes de productos que se integran para ofrecer los estándares y tecnologías necesarios para el desarrollo de una arquitectura orientada a servicios.

Sin embargo, la adopción de una de estas soluciones suele ser costosa, ya que implica procesos de selección, montaje de proyectos piloto que permitan evaluar las características de cada producto, adquisición de licencias y firmas de contratos de soporte.

La evolución hacia una arquitectura basada en SOA es por su alto grado de penetración tanto en los sistemas, como en las áreas de negocio, un proceso complejo, largo y costoso. Las metodologías existentes para la adopción, recomiendan comenzar por un proyecto piloto que permita evaluar, tanto las características tecnológicas, como la compatibilidad con los sistemas existentes, y por encima de todo, los beneficios obtenidos mediante el uso de la arquitectura, o lo que es lo mismo, estudiar si la inversión que la compañía está planteando realizar va a proporcionar el retorno esperado tanto a nivel de IT (Tecnologías de la Información), como a nivel de negocio.

Por eso, en ocasiones es interesante plantearse la adopción usando una estrategia basada en herramientas OpenSources, que permitirá dar los primeros pasos en la tecnología sin incurrir en grandes costes iniciales, y reduciendo los riesgos.

2.2 Ventajas de utilizar Open Source

Cuando se planifica un nuevo proyecto surge siempre la duda: ¿software libre o propietario? Si se tiene en cuenta las principales ventajas de usar software Open Source entonces se irá por el camino del software libre.

Algunas de estas ventajas son:

- **Coste de propiedad reducido:**

El coste de productos Open Source con licencia GPL es significativamente inferior al de productos equivalentes con licencias propietarias, e incluso puede ser nulo.

- **Alta fiabilidad, escalabilidad y rendimiento**:

Prueba de ello es que empresas de gran porte como Yahoo lo usan.

- **Uso de estándares:**

Posibilitan la integración con otros sistemas y la evolución de los existentes. Esto es más importante de lo que se piensa, especialmente en proyectos a largo plazo. Por ejemplo, un sistema de seguimiento de expedientes desarrollado hace años en base al formato de WordPerfect -un antiguo procesador de texto- es hoy inutilizable con la consiguiente pérdida de información. El uso de formatos estándar como rtf, dif, es fundamental para quebrar la dependencia de los protocolos propietarios y asegurar la continuidad de las operaciones a largo plazo.

- **Control absoluto sobre los procesos y la información**:

El código abierto es de dominio público y puede ser auditado siempre que se quiera, asegurando así que no existan puertas traseras o bombas lógicas entre otros problemas. Sobran ejemplos de los graves inconvenientes que conlleva el uso de código propietario.

- **Independencia frente a estrategias de los fabricantes:**

¿Qué pasa cuando sale una nueva versión de Linux o de cualquier software libre? Simplemente se baja la nueva versión, se instala o se recompila y ya está. ¿Que les pasó a los usuarios de NT cuando salió 2000 Server y no actualizaron sus licencias? Hoy están sin ninguna clase de soporte. Son instalaciones muertas. El software Open Source en cambio está mantenido y respaldado por una comunidad de

desarrolladores que se extiende a todo el mundo, ellos aseguran que su aplicación esté siempre actualizada y de no ser así, podrá migrar hacia otras de prestaciones iguales o superiores sin grandes contratiempos.

- **Optimización del uso de la plataforma existente:**

Un servidor Linux puede operar como un controlador de dominio en una red Windows implementando todo los servicios en forma transparente para los usuarios.

- **Libertad de uso y redistribución:**

Las licencias de software de fuentes abiertas existentes permiten la instalación del software tantas veces y en tantas máquinas como el usuario desee.

- **Independencia tecnológica:**

El acceso al código fuente permite el desarrollo de nuevos productos sin la necesidad de desarrollar todo el proceso partiendo de cero.

- **Fomento de la libre competencia al basarse en servicios y no licencias:**

Uno de los modelos de negocio que genera el software de fuentes abiertas es la contratación de servicios de atención al cliente. Este sistema permite que las compañías que den el servicio compitan en igualdad de condiciones al no poseer la exclusividad del producto del cual dan el servicio. Esto, además, produce un cambio que redunda en una mayor atención al cliente y contratación de empleados, en contraposición a sistemas mayoritariamente sostenidos por la venta de licencias.

- **Estándares abiertos**:

Los estándares abiertos permiten una interoperatividad más alta entre sistemas, evitando incompatibilidades.

- **Sistemas sin puertas traseras y más seguros:**

El acceso al código fuente permite que tanto expertos como empresas de seguridad de todo el mundo puedan auditar los programas, por lo que la existencia de puertas traseras es ilógica, ya que se pondría en evidencia de manera casi inmediata.

- **Corrección más rápida y eficiente de fallos:**

La disponibilidad del código fuente ha demostrado solucionar más rápidamente los fallos de seguridad en el software de fuentes abiertas, posibilidad que no se da en el caso del software propietario

El mantenimiento de los productos OpenSource es mucho más activo. En muchas ocasiones, el parche a un bug está publicado a los pocos días de comunicarse (sin necesidad de contratar soporte). En cualquier caso, se tendrá mejor soporte en el mundo OpenSource en aquellos productos que dispongan de una base de usuarios más amplia.

2.3 Tipos de ESB Open Sources

Los grandes proveedores de software han ido incorporando los Buses de Servicio Empresarial a sus productos para crear una solución integrada. La misma filosofía han seguido los intermediarios de integración tradicionales para no perder poder competitivo, adaptando sus productos a los requisitos de los ESBs y redefiniendo su estrategia de mercado. Los proveedores especializados únicamente en soluciones ESB siguen aguantando la fuerte competencia que ejercen los proveedores líderes defendiendo un modelo de arquitectura más flexible orientado a las necesidades de los clientes.

Recientemente han ido apareciendo ESBs Open Sources muy interesantes. Algunos de estos proyectos se han ido incorporando en proyectos de mayor alcance destinados a plataformas Open Source de SOA, como es el caso de ServiceMix en LogicBlaze Fuse o Open-ESB en Open SOA de SUN. Además de los siguientes ESB Open Source: Celtix, Mule, Open-ESB, Apache ServiceMix, JBossESB.

CAPÍTULO 2

Actualmente en Internet existe una gran cantidad de ESB Open Source, esto puede
parecer abrumador a primera vista a la hora de escoger uno pero estudiando sus similitudes, y con una visión general de cada uno se hace más fácil la elección. En este capítulo no se nombran todos los ESB Open Source, sino aquellos que por su calidad, madurez, amplia base de usuarios, frecuencia de mantenimiento, y posibilidades de soporte, son aptos para ser utilizados en entornos de producción de la UCI.

2.3.1 MULE

Ross Mason fue el creador de una plataforma de integración reutilizable, de nombre Mule. Es una plataforma puntera Open Source para la integración de aplicaciones. Se trata de un ESB diseñado para soportar transacciones de alto rendimiento y multi-protocolo.

Se ha convertido en una alternativa más que viable ante las soluciones comerciales. Soporta J2EE 1.4 (Java 2 Platform, Enterprise Edition) y conectividad mediante multitud de protocolos, se integra con JBI (Java Business Integration), soporta eventos, y es capaz de realizar tareas de orquestación de servicios integrándose con jBPM como motor. Mule es muy útil por su enfoque hacia la sencillez y grandes posibilidades de integración.

Mule puede ser usado para mensajería entre sistemas de información, como middleware transaccional, o como parte de un servidor de aplicaciones. La naturaleza extensible del núcleo de Mule, así como el hecho de tener el código abierto, permite a los desarrolladores mantener el control de su infraestructura.

Mule, a diferencia de otros productos, permite a los desarrolladores y arquitectos**,** **i**ntegrar diferentes aplicaciones de un modo incremental sin afectar a la infraestructura IT subyacente ya existente en la empresa. Y como es de código abierto, los desarrolladores lo pueden adaptar a sus necesidades en lugar de ser la plataforma la que defina su entorno.

Este ESB tiene funcionalidad multi-propósito para soportar un amplio rango de tipos de integración típicos en una empresa. La instalación básica de Mule soporta más de 30 estándares, protocolos y tecnologías, incluyendo: JMS, JBI, EJB, Spring, JDBC, TCP, HTTP, Web Services y MQ Series. Consta de una licencia MuleSource Public License (MSPL), que es una modificación de la Mozilla Public License Versión 1.1**.**

Proporciona conectividad para más de 20 protocolos de transporte y se integra con un gran número de proyectos de integración, incluyendo Spring, ActiveMQ, Joram, XFire, Axis y Drools. Mule optó por no construir su arquitectura de Java Business Integration (JBI), aplicando su propia arquitectura flexible y con un ligero modelo centrado en la productividad y la facilidad de desarrollo. Después de la liberación de la versión 1.0 Mule en 2005, Mule ha recibido más atención del mercado a través de los años. Es utilizado en la actualidad por un gran número de empresas de todo el mundo, Incluyendo Walmart, HP, Sony, Deutsche Bank y CitiBank.

La visión general de la funcionalidad de Mule (Ver Anexo 2) se centra en el gran número de opciones de conectividad que proporciona para los protocolos de transporte. Mule también proporciona conectividad con contenedores JBI a través de una especie de adaptador JBI.

Algunas de las principales características de Mule:

- **Nace como un ESB**

Y no como adaptación/evolución de un producto existente previamente.

- **Es una plataforma de mensajería.**

Está diseñado para la implementación de arquitecturas orientadas a servicios. El núcleo de Mule es un contenedor de servicios basado en **SEDA** (Staged Event-Driven Architecture).

CAPÍTULO 2

Mule administra componentes llamados UMOs (Universal Message Objects), que en la práctica son plain old java objects (POJO), lo que se traduce con gran facilidad para la construcción de nuevos conectores a plataformas no incluidas inicialmente.

SEDA:

Significa “arquitectura por etapas activada por eventos”. Es utilizada en el desarrollo de sistemas que procesan grandes cantidades de mensajes en forma concurrente basándose en el diseño de la lógica de negocios en etapas independientes y que se intercomunican por medio de colas de mensajes. (Gómez Rubio, Álvaro, 2007)

- **Contenedor de aplicaciones.**

Mule actúa como un contenedor de aplicaciones sencillo y robusto, integrado de forma transparente a sistemas externos con los estándares de comunicación más conocidos. El desarrollo de componentes se basa en el estándar más sencillo de Java (plain old java objects), evitando la complejidad innecesaria de J2EE.

- **Plataforma para proceso masivo de mensajes.**

Mule provee la plataforma ideal para la construcción de aplicaciones de procesamiento masivo de mensajes, agregando inmediatamente a la solución la administración, monitoreo, log y configuración centralizadas, que son propias del producto.

- **Administración gráfica centralizada de los componentes.**

La operación de cada componente por separado puede ser controlada, por ejemplo, pausando y encolando las solicitudes para solucionar problemas eventuales en otras etapas del proceso.

- **Soporta los distintos modelos de arquitecturas**.

Centralizada, distribuida, en modo cluster, balanceada.

- **Funciona sobre una máquina virtual Java.**

Al igual que en la mayoría de los servidores de aplicaciones J2EE.

2.3.2 ServiceMix

ServiceMix es un ESB Open Source construido completamente en la semántica y APIs de las especificaciones Java Business Integration (JBI) JSR208 y dispensado bajo la licencia Apache. Es un ESB montado a partir de la especificación JBI. Combina la funcionalidad de SOA con la de EDA (Event-Driven Architecture) para facilitar y maximizar la reutilización. Utiliza ActiveMQ para proporcionar fiabilidad y tolerancia a fallos.

El JBI tiene como propósito definir un estándar para una plataforma de integración de manera que esta plataforma pueda constar con componentes de múltiples proveedores y proyectos de código abierto, para evitar el encadenamiento con proveedores. Después de que la especificación JBI fue aceptada por el JCP (Java Community Process) a finales de 2005 el proyecto ServiceMix Apache fue aceptado como una incubadora de proyectos de Apache.

El objetivo de ServiceMix es proporcionar un ESB que implemente la especificación JBI, que se centra en la flexibilidad, la fiabilidad y la amplitud de la conectividad. Después de varios años de desarrollo ServiceMix proporciona un gran conjunto de componentes JBI.

ServiceMix presenta un alto nivel en su arquitectura (Ver Anexo 3). La funcionalidad del ESB ServiceMix mostrado en el anexo 3 no es más que un pequeño subconjunto de las capacidades de este ESB en su conjunto. Pero debe quedar claro que por ahora es un ESB que se construye desde cero para la aplicación de la especificación JBI.

Al igual que OpenESB cumple con la especificación JSR 208(Java Specification Requests) e incluye un contenedor JBI completo en conjunto con una amplia variedad de componentes JBI. En particular se incluye un motor BPEL (Lenguaje de Ejecución de Procesos de Negocio) propietario basado en Apache ODE Bpe, al

mismo tiempo que es posible la integración con el motor PXE (Preboot Execution Environment).

A diferencia de OpenESB no existe integración con ningún IDE (integrated development environment) conocido. Sin embargo se cuenta con una guía para extender el plugin WTP (Web Tools Platform) de Eclipse de forma tal que sea posible el desarrollo de un proceso BPEL al contenedor JBI.

2.3.3 OpenESB

Los dos ejemplos anteriores de Open Source ESBs expuestos en el presente se encuentran en la comunidad OpenSource, en Mule Codehaus y ServiceMix en Apache. OpenESB es una iniciativa de SUN es acogido como un proyecto de Java.Net. Al igual que la aplicación ServiceMix, OpenESB también es una aplicación de la especificación JBI.

El proyecto OpenESB en realidad es un proyecto global que incluye una serie de subproyectos. OpenESB se centra en el servidor de aplicaciones y Glassfish. La principal diferenciación entre OpenESB y otras implementaciones de ESB como ServiceMix es la gama de herramientas de apoyo que posee. En el último año el IDE Netbeans se ha enriquecido de una manera impresionante como parte de la estrategia de código abierto de Sun.

Cuando se aprecia la descripción de la funcionalidad de OpenESB (Ver Anexo 4), se puede ver que los componentes pueden ser vinculados en relación con los motores de servicio, pero aún no hay posibilidad de enrutamiento o encaminamiento a varios proveedores. El número de Servicio de los motores crecerá con el tiempo y la funcionalidad de enrutamiento se añadirá en versiones futuras.

OpenESB permite integrar fácilmente aplicaciones empresariales y webservices como aplicaciones compuestas débilmente acopladas. Esto permite componer y recomponer de manera fluida y rápida aplicaciones compuestas, con todas las ventajas de una verdadera Arquitectura Orientada a Servicios.

CAPÍTULO 2

OpenESB se ejecuta sobre el servidor de aplicaciones Glassfish/Sun Application Server e incluye una gran variedad de componentes JBI, como SOAP-over-HTTP-binding, y un motor de servicio WS-BPEL 2.0. Además incluye su propio motor BPEL (BPEL SE). Debido a la arquitectura débilmente acoplada de OpenESB es posible cambiar este motor por cualquier otro que cumpla la especificación JSR 208, como por ejemplo el de ServiceMix.

Uno de los puntos fuertes de OpenESB es la integración de NetBeans 5.5 con el mismo. Esta integración incluye facilidades de diseño, compilación, desarrollo y depuración de procesos BPEL, como también facilidades de administración del entorno JBI.Aunque está basado en el SDK (kit de desarrollo de software) JBI, OpenESB amplía las capacidades de la implementación de JBI con Servicios de motores, herramientas y servicios de administración y monitorización adicionales. La integración con el entorno de desarrollo NetBeans permite el despliegue de aplicaciones de una manera rápida y eficiente, con una serie de facilidades como ayuda en el desarrollo, control de errores y pruebas.

Para muchos entornos empresariales, la capacidad de distribuir componentes y servicios se convierte en un requisito indispensable. Hasta ahora estos requisitos podían ser solucionados desplegando múltiples instancias independientes de entornos JBI, enlazados mediante protocolos estándar de comunicación. Este presenta dos importantes inconvenientes:

- No hay un único punto para administrar el sistema completo. Cada componente y cada servicio debe ser administrado por cada instancia JBI local.
- Cada operación del sistema (desplegar, instalar, iniciar, detener.) requiere un exhaustivo conocimiento de la topología del sistema. Por ejemplo, desplegar una aplicación JBI requeriría conocer la localización física en el sistema de los componentes utilizados.

Este ESB Open Source respeta las especificaciones JBI (Java Business Integration) y está compuesta por tres elementos esenciales. En el centro del dispositivo se encuentra el router de mensajes que siguen la norma NMR (Normalised Messages

Router). Sobre este modelo se han sumado dos tipos compuestos: los motores de servicios (BPM, BAM) y los componentes de conexión como los Servicios Web, el FTP, el módulo Com/DComm. El NMR asegura la unión entre estos diversos componentes, donde el adaptador ha sido escrito (en general por el editor de componentes) para una compatibilidad con la especificación JSR 208 (JBI).

Ventaja respecto a los servicios Web: es suficiente escribir un único adaptador para abrir el componente o la aplicación a la comunicación con todos los demás componentes adaptados al ESB. Con los servicios Web sin ESB hace falta escribir tanto interfaces como lugares a establecer.

A manera de resumen se puede decir que OpenESB, tiene como principales ventajas el cumplir con la especificación JBI (JSR 208), contar con un motor BPEL propio (evitando dificultades de configuración de entorno) y la existencia de un IDE totalmente integrado con el mismo (NetBeans).

A pesar de sus ventajas con respecto a otros ESBs es claro que aún se encuentra en una etapa muy temprana de desarrollo. Esto se refleja por ejemplo en las notificaciones de errores de tiempo de ejecución que resultan ser poco informativas al usuario. También presenta problemas de compatibilidad en el manejo de fallas lanzadas por webservices externos, cuyo procesamiento provoca el lanzamiento de errores fatales por parte del ESB. Por último es relevante mencionar que el mismo no soporta aún la interacción con webservices cuyo enlace sea de estilo RPC.

2.3.4 JBossESB

JBossESB proporciona una plataforma de arquitectura orientada a los servicios para integrar los componentes y servicios empresariales en procesos empresariales automatizados. Algunos de los diversos servicios incluyen: registros, servicios de transformación, enrutamiento, servicios web, servicios de reglas y orquestación de servicios.

JBoss es ampliamente conocido por su popular servidor de aplicaciones, y el éxito de proyectos de código abierto. En el área de integración empresarial, JMS proporcionó un proveedor llamado JBossMQ y un motor de reglas JBoss ESB.

Con la adición de los productos JBoss ESB se puede implementar la arquitectura JBoss sobre productos. La capa de mensajería de JBoss ESB es JBoss Messaging (el sucesor de JBossMQ), un JMS proveedor de la aplicación. La funcionalidad de enrutamiento de JBoss ESB se basa en las normas del motor, JBoss Normas y la funcionalidad de orquestación es proporcionada por el motor de proceso de jBPM.

Este ESB Open Source entre sus funcionalidades (Ver Anexo 5) proporciona opciones de conectividad incluye todos los transportes comunes utilizados en un típico proyecto de integración. JBoss es un gran jugador en el mercado de código abierto de Java, se puede esperar que JBoss abra en el mercado suficiente atención en próximos períodos. JBoss proporciona toda la funcionalidad de integración de pila. Sin embargo funcionalidades como la seguridad, el Servicio de Arquitectura de Componentes (SCA) y Objetos de servicios de datos (ODS) de apoyo aún no están disponibles.

2.3.5 SYNAPSE

Synapse es un proyecto de código abierto para desarrollar un marco interoperable para la mediación de Servicios Web. El proyecto, creado bajo los auspicios de la Fundación Apache de Software, proporcionará un componente fundamental para el software de infraestructura de Servicios Web, inclusive el ESB, de Web Services, y productos de administración de Web Services.

Synapse es una implementación de código abierto con una plataforma de mediación de servicios Web y componentes para el uso, desarrollo y despliegue de las infraestructuras de SOA. Synapse proporciona una plataforma para mediar entre dos o más servicios Web, permitiendo a los usuarios controlar los flujos entre servicios.

CAPÍTULO 2

En la gama de ESBs Open Sources que se dispone actualmente, Apache Synapse es de gran uso. Podemos cuestionarnos si Apache Synapse en realidad es un verdadero ESB, pero sobre la base de las funciones básicas, Synapse se puede catalogar como un ESB. Synapse es una web de los servicios de mediación que se basa en Apache Axis2, la web de los servicios de contenedores. Esto marca la diferencia entre Synapse y los ya expuestos ESBs, Mule, ServiceMix y OpenESB. El enfoque de Synapse es proporcionar la funcionalidad de enrutamiento, transformación, la validación y el mensaje de un registro basado en servicios web y las normas XML. Como parte de las organizaciones de normalización, como el W3C y OASIS, se está desarrollando un enorme conjunto de las normas de servicios Web .Algunos ejemplos de estos servicios web son las especificaciones WS-Abordar, WS-Security, WS-Policy fiable y WS-Mensajería.

Este ESB además de ofrecer las funcionalidades básicas, tales como enrutamiento, transformación y un registro, Synapse puede facilitar la necesaria abstracción para utilizar los servicios de las normas. Un ejemplo de esta capa de abstracción es que con 2 Líneas de configuración XML, Synapse es capaz de ejecutar un intercambio de mensajes con un WS-Confiable permitido a los servicios de mensajería Web.

Entre las funcionalidades de Synapse (Ver Anexo 6) se encuentra que puede interactuar con los servicios sobre la base de un XML, servicios web estándares y proporciona la funcionalidad de enrutamiento, la validación, transformación, la gestión y la independencia de la ubicación. Synapse es un ligero marco de la mediación que se puede extender fácilmente con la lógica personalizada.

La visión general de la funcionalidad de Synapse muestra claramente que se centra en los servicios web. Las opciones de conectividad son sobre todo SOAP sobre HTTP y SMC pero otras opciones como SMTP (Protocolo Simple de Transferencia de Correo), también son posibles. A principios del 2007, Synapse se ha graduado a la condición de ser un miembro de la Apache Web de los servicios de proyecto, como es Axis2.

2.3.6 CELTIX

Celtix un bus de servicio en tiempo de ejecución y un conjunto de APIs (Application Programming Interface) que hacen que sea fácil agregar transporte, formatos de mensaje, y características de seguridad. El objetivo es simplificar la construcción, la integración y la flexibilidad técnica y la reutilización de componentes de negocio utilizando la arquitectura orientada a los servicios.

Celtix es un ESB que se distribuye bajo una licencia libre. Inicialmente, su código formaba parte de un proyecto propietario de la empresa Iona, código que fue donado al consorcio francýs ObjectWeb. Celtix soporta JAX-WS 2.0, diversos lenguajes de script, y es compatible con los estándares JBI y SCA.La base de código Celtix se ha movido a Apache como Incubadora Apache CXF y la clave ahora es la comprobación de sus actualizaciones en Apache CXF.

Celtix está basado en Java ESB es organizada por el Consorcio de objetos y es uno de un número creciente de ESBs de código abierto, en un campo que incluye SymphonySoft del Mule y LogicBlaze del ServiceMix. Celtix funcionará con cualquier JBI de contenedores y cuenta con una aplicación de la JAX-WS (Java API para XML Web Services) para la construcción de la especificación de servicios Web basados en aplicaciones Java.

El concepto original de Celtix es para aquellos proyectos que son de carácter departamental. Su principal fortaleza es que se trata de una arquitectura altamente distribuida y no totalmente dependiente de una pila central .Celtix está disponible a través de Iona Eclipse, la GPL (Licencia Pública General de GNU), o la LGPL (Lesser GPL) licencias.

Es considerado un líder mundial encaminado a la arquitectura orientada a los servicios (SOA) para soluciones de infraestructura de rendimiento de exigentes entornos de IT. Celtix de clase empresarial proporciona capacidades ESB. Esta combinación de la tecnología y de la correspondiente consulta, la formación y el apoyo, entrega las bases esenciales requeridas por las empresas que desean desplegar SOA basadas en tecnologías de código abierto.

2.4 Lenguajes preferidos para programar código Open Source.

- C (14955 proyectos)
- C++ (15580 proyectos)
- C# (2465 proyectos)
- Delphi/Kylix (1805 proyectos)
- Java (15159 proyectos)
- JavaScript (2374 proyectos)
- Perl (5875 proyectos)
- Python (4057 proyectos)
- Visual Basic (2089 proyectos)
- Visual Basic .NET (177 proyectos)

A la vista de los resultados las tendencias mayoritarias parecen ser C/C++, JAVA y Perl/PHP.

2.5 Tabla comparativa entre algunos ESB Open Source.

ESB Open Source	Licencia	Lenguaje	Sistema Operativo	Empresa Creadora	Costo

Mule	Mule Source Public License (MSPL) (Modificación de la Mozilla Public License Versión 1.1)	Se basa en el estándar más sencillo de Java	Windows server, SUSE Linux Ubuntu/ debian Linux Mac OSX Sclaris SPARC/x86	Mule Source	Gratuito
Service- **Mix**	Licencia Apache.	Java	Windows: Windows XP SP2, Windows 2000 Unix: Cualquier Linux / Unix Que soporten Java	The Apache Software foundation	Gratuito

Open-ESB	CDDL & GPLv2.	Java	Solaris, Linux (RedHAT), Windows en todas las versiones En algunas versiones se soporta también HP-UX	Liderada por Sun	Gratuito
Synapse	Apache Software License v2.0	desarrollo inicial Java, pero en el futuro estará basado en Apache de C / C + +	Cualquier sistema operativo que corra la máquina virtual de Java.	Sonic Software conjunto al Proyecto de la Fundación Apache	Gratuito
Celtix	GPL (Licencia Pública	Java Script para XML	Cualquier sistema operativo	Creado por IONA y acogido	Gratuito

	General de GNU), o la LGPL (Lesser GPL)	(E4X)	que corra la máquina virtual de Java.	por la comunidad ObjectWeb	

2.7 Conclusiones parciales

Debido a que la integración de aplicaciones está pasando a formar parte importante de los requerimientos y el esfuerzo de las unidades de informática de las instituciones, y los ESB son la respuesta a poder racionalizar esta labor de una manera eficaz y brindando una sólida arquitectura, para que las organizaciones consigan adoptar los cambios vitales que les plantee su ámbito operacional.

A la hora de incursionar en el mundo de los ESB en ocasiones es interesante plantearse la adopción usando una estrategia basada en herramientas Open Source para ello en este capítulo se ha planteado la importancia de usar este tipo de herramienta, se ha hecho un estudio de una serie de ESB Open Source indagando de manera minuciosa en cada una de sus características, para de esta forma hacer más fácil el proceso de selección de un ESB Open Source como propuesta para instalarlo en la UCI.

CAPÍTULO 3

CAPITULO 3: PROPUESTA DEL ESB OPEN SOURCE A INSTALAR EN LOS PROCESOS PRODUCTIVOS DE LA UCI.

3.1 Introducción:

Este capítulo está dirigido a explicar el ESB Open Source que se tiene como propuesta para instalar en los procesos productivos de la universidad. Se describe el por qué de este ESB como alternativa, sus características fundamentales y los pasos a seguir para su correcta instalación.

Mule el ESB Open Source que se tiene como sugerencia está considerado como el mayor Bus de Servicio Empresarial de código abierto utilizado en el mundo. Diseñado para el apoyo de alto rendimiento, multi-protocolo de las transacciones entre sistemas heterogéneos y servicios, proporciona la base para la arquitectura orientada a servicios. El modelo de código abierto permite a las organizaciones lograr un rápido retorno de la inversión, y abrir el acceso al código fuente permitiendo a los desarrolladores personalizar fácilmente Mule para satisfacer sus necesidades de esta manera con esa serie de utilidades este ESB se convierte en la solución ideal para mejorar el proceso de mensajería en la universidad.

3.2 Superioridad de Mule frente a otros ESB Open Source.

Después de haber estudiado una serie de Open Source ESBs disponibles en la actualidad, es el momento de hacer una elección sobre el ESB que se utilizará. Para hacer una deliberación objetiva se debe definir una serie de criterios que son importantes.

Criterios de selección:

Apoyo a las funcionalidades básicas del ESB: registro de servicio, protocolo de transporte de conversión, la transformación, enrutamiento, la seguridad y la vigilancia y la gestión.

- Buena documentación escrita.
- Mercado visibilidad.
- Activa comunidad de desarrollo y apoyo.

- ❖ Flexible y fácilmente extensible con lógica personalizada.
- ❖ El apoyo a una amplia gama de protocolos de transporte y opciones de conectividad.
- ❖ Integración con otros proyectos de código abierto.
- ❖ Productividad con soporte para IDE (Entorno de desarrollo Integrado).

Tabla 3.1 Otros aspectos a comparar entre los ESB Open Source.

Criterio de selección	**Mule**	**Service Mix**	**Open ESB**	**Synapse**	**JBoss ESB**
Apoyo a las funcionalidades básicas del ESB	+	+	+/-	+	+
Buena documentación escrita	+	+/-	+	+	+/-
Mercado visibilidad	++	+	+/-	+/-	+/-
Activa comunidad de desarrollo y apoyo	++	+	+/-	+	+
Flexible y fácilmente extensible con lógica personalizada.	++	+	+/-	++	+

El apoyo a una amplia gama de protocolos de transporte y opciones de conectividad	+	+	+/-	+/-	+
Integración con otros proyectos de código abierto	++	++	+/-	+	+
Productividad con soporte para IDE(Entorno de desarrollo Integrado)	-	+/-	++	-	+/-

Leyenda:

- ++ Muy buena
- + Buena
- +/- Regular
- - Mala

El primer criterio es el apoyo a las funcionalidades básicas del ESB, todos los ESBs vistos en el capítulo anterior con excepción de OpenESB proporcionan un buen apoyo para el núcleo de funcionalidades. Hay espacio de mejora para todos los

ESBs por ejemplo en la vigilancia y la gestión. OpenESB está clasificado como promedio, ya que en algunas de las funciones básicas no es de los mejores.

El siguiente criterio es acerca de la calidad de la documentación facilitada por los proyectos de ESB Open Source. En este aspecto se puede apreciar la superioridad de Mule frente a otros ESB Open Source, por ejemplo, la documentación presentada por Open ESB dispone de bonitas capturas de pantalla y buenos ejemplos, pero la documentación de Mule es más completa que el OpenESB, debido a que también ofrece buenos ejemplos de código. Lo mismo se aplica a la Documentación de Synapse. Siendo menos complaciente la documentación de ServiceMix y JBoss ESB. Los ejemplos de ServiceMix a veces son obsoletos y no proporcionan suficiente información para comprender lo que está pasando. JBoss ESB para la estructura y el diseño de la documentación no es clara y, por lo tanto, resulta difícil de trabajar en el.

El mercado de la visibilidad del Open Source ESBs está creciendo al igual que el número de artículos en Internet, incluyendo los analistas de los informes es cada vez mayor. El número de implementaciones en empresas de todo el mundo es algo que está empezando. Mule es una excepción digna de elogio, ya que pueden contar con un gran número de implementaciones de sus productos y está recibiendo mucha atención en el mercado. ServiceMix también recibió mucha atención, porque es el más conocido de código abierto JBI y es un proyecto de Apache. Para los otros ESBs, la visibilidad en el mercado está aumentando, pero han de ponerse al día en esto con Mule y ServiceMix.

Con respecto al desarrollo y el apoyo activo de la comunidad el cual es muy importante cuando se trabaja con productos de código abierto, pues pueden ejecutarse errores, por lo tanto, es importante que un producto de código abierto pueda contar con una activa comunidad que pueda prestar apoyo en caso de dudas o errores, y que sea capaz de incluir solicitudes de mejoramiento en versiones futuras.

Las comunidades de código abierto ESBs son, en general, bastantes activos están disponibles paquetes comerciales así como foros. Mule tiene una muy activa

comunidad de desarrollo y es capaz de proporcionar apoyo a través de MuleSource. OpenESB es un poco extraño si se compara con otros ESBs, porque el equipo de desarrollo está formado solo por empleados de SUN y la disponibilidad de foros es baja.

Elegir un ESB pese a factores como la tecnología existente, del desarrollo y del despliegue, política de la compañía, sociedades estratégicas (o carencia de ellas), sistemas de la herencia entre otras. Es una evaluación reciente de todos estos productos, la decisión de una empresa para utilizar Mule están basados en estos criterios:

- Comunidad activa de código abierto y soporte comercial disponible.
- Estado del arte de la implementación y habilidad que funciona sobre Java 5/6.
- Facilidad de instalación y desarrollo.
- Facilidad de configuración y expansión.
- Integración con legalidad a terceras personas, y productos comerciales.
- Capacidad para integrar dentro/fuera sin incurrir en el bloqueo o dependencias del producto ancilares.
- Coste total bajo por parte del propietario.

3.2.1 Características especificas de Mule.

¿Qué es Mule?

Mule es una plataforma de mensajería construida en base a ideas de la arquitectura de los Bus de Servicios Empresariales. Un ESB trabaja actuando como una especie de sistema de tránsito que acarrea datos dentro o fuera de su Intranet. El ESB define una serie de paradas -endpoints- a través de las cuales las aplicaciones pueden enviar o recibir datos desde o hacia el sistema. El corazón del sistema, el Bus de mensajería, se encarga de enrutar mensajes entre los endpoints. Mule es un framework de mensajería de bajo peso que contiene un broker de objetos para manejar comunicaciones entre aplicaciones. El punto de que sea un broker de objetos es para manejar componentes de servicio. Estos componentes son llamados

Objetos Universales de Mensaje o UMOs por sus siglas en inglés (Universal Message Objects), y son básicamente objetos Java.

UMOs pueden existir dentro de una misma máquina virtual o pueden estar esparcidos a lo largo de su red y/o Internet. El broker de objetos se adhiere a una arquitectura orientada a eventos o a un patrón de diseño SEDA. Todas las comunicaciones entre UMOs y otras aplicaciones se realizan mediante endpoints de mensajería. Estos endpoints proveen una interface consistente y simple para una vasta gama de tecnologías como JMS, SMTP, JDBC, TCP. HTTP, XMPP y archivos.

Las aplicaciones Mule (Ver Anexo 7) usualmente se componen de muchas instancias de Mule a lo largo de una red. Cada instancia es un contenedor de peso ligero que hospeda uno o más componentes UMO. Cada componente UMO tendrá uno o más endpoints a través de los que enviará o recibirá eventos.

¿Mule es un ESB?

Mule fue diseñado manejando los conceptos de un ESB, pero su diseño evolucionó para proveer mayor flexibilidad en la forma de comunicación entre los servicios. Un ESB estandariza el concepto de un Bus de Mensajes mientras que los servicios Mule pueden comunicarse sobre un amplio rango de canales de comunicación. Se puede pensar en un ESB como una topología – una forma de organizar componentes y sus interacciones. Donde Mule difiere de un ESB es que soporta la topología ESB pero también soporta otras: pipeline, peer network, cliente/servidor, hub-and-spoke y embedded. Estas topologías pueden ser mezcladas y agrupadas para modelar complejos requerimientos de mensajería y servicios empresariales. Mule ha sido diseñado para proveer un modelo simple y poderoso de cablear servicios POJO entre si utilizando los endpoints y para hacer ningún tipo de asunción en referencia al mensaje o las interfases utilizadas. La meta final de Mule es adaptarse a la tecnología circundante más que prescribir cual utilizar. No existen reglas duras ni rápidas de como debe comportarse una capa de servicios de integración utilizando Mule; se puede interconectar JBI, EJB, aplicaciones de Mainframe, mensajería, servicios web, sockets y sistemas de archivo e interactuar con todos ellos de una manera sencilla y consistente.

CAPÍTULO 3

¿Por qué Mule?

Las metas del proyecto Mule y de los Objetos Universales de Mensaje (Universal Message Objects) han sido escuchadas miles de veces de muchos proveedores.

- Un framework de mensajes escalable que debe manejar las complejidades de la integración de sistemas.
- Disponer de un servidor fácil de usar, aunque poderoso, que pueda operar sobre topologías complejas.
- Desarrollo y puesta en producción de componentes de manera simple y autónoma.
- Reutilización de código. Si todos los componentes son unidades independientes y auto-contenidas, pueden ser conectados en cualquier otro sistema.
- Rápido tiempo a producción (Rapid time to market). Utilizando Mule se obtienen funcionalidades que permiten ahorrar tiempo ya que no se tiene que desarrollar en algunos casos.
- Flexibilidad. Una poderosa configuración que debe ser fácil de administrar sobre ambientes distribuidos.

Cuando comenzó el Proyecto Mule, parecía existir una manera simple y ligera de escribir componentes que hicieran algo con los datos sin la necesidad de preocuparse de quién envía o recibe esa información, el formato de los datos o la tecnología utilizada para el envío/recepción de la misma. La clave aquí era “Simple”, aunque muchas tecnologías para integración o distribución ofrecían la habilidad de conectarse a diferentes fuentes de datos, se terminaba muchas veces teniendo que colocar código adicional para hacer que se comportaran de la manera requerida y para que enviaran los datos donde se deseaba que fuese. Mule le permite rápidamente desarrollar componentes y entonces cambiar la forma en la que se comportan mediante la configuración en lugar del código. Mule no es de ninguna forma un sustituto para un framework de aplicaciones, de hecho Mule interactúa con muchos proyectos Open Source como: Acis, Spring, ActiveQ, Plexus y PicoContainer. Mule llena un espacio en el desarrollo empresarial java donde una aplicación requiere interacciones complejas con una variedad de sistemas en una

variedad de plataformas. Mule hace el trabajo ligero de interconectar esos sistemas en un ambiente desacoplado y robusto y provee el soporte necesario para direccionar, transportar y transformar datos desde y hacia esos sistemas.

De manera general Mule es una plataforma de mensajería basado en arquitecturas ESB. El núcleo de Mule está basado en el servicio contenedor SEDA, que gestiona servicio de objetos, conocido como Universal Message Objects (UMOs), que son simples objetos Java. Todas las comunicaciones entre UMOs y otras aplicaciones son realizadas a través de mensajes endpoints. Estos endpoints proporcionan un simple y consistente interfaz para tecnologías tan dispares como JSM, SMTP, JDBC, TCP, HTTP, XMPP y ficheros.

Visión general de la arquitectura.

La última meta de Mule es proporcionar un método unificado de interactividad con datos de fuentes dispares, sin estorbar al desarrollador con detalles sobre cómo los datos son enviados o recibidos o protocolos implicados. El resultado es un servidor ESB, que es altamente escalable, ligero, rápido y simple. La arquitectura de Mule fue diseñada con el modelo ESB en mente y el primer enfoque es simplificar y acelerar el proceso de desarrollo de redes de servicios distribuidos. Sin embargo, como su nombre sugiere, el modelo ESB usualmente se asocia con proyectos importantes de integración donde hay un conjunto de aplicaciones de empresa dispares. Mule crea arquitecturas de servicios de nivel de empresa, posibles para proyectos más pequeños donde recursos y costo de desarrollo necesitan un mínimo de mantenimiento. El primer objetivo es posibilitar la integración entre aplicaciones estándares, protocolos abiertos y diseños bien definidos. Para alcanzar este objetivo Mule define un conjunto de componentes que pueden ser usados para realizar la mayor parte del duro trabajo necesario para conseguir unir aplicaciones dispares y comunicación de servicios.

El **canal** puede tener cualquier método de comunicación de datos entre dos puntos. Los canales se emplean en Mule para conectar componentes UMO así como para conectar diferentes Mule Nodes a través de red local o internet.

El **recibidor de mensajes** se usa para leer o recibir datos desde la aplicación. En Mule un recibidor es un elemento de un Transport provider, Mule proporciona muchos transportes, como JMS, SOAP, HTTP, TCP, XMPP, SMTP y file.

Mucha de la magia de Mule está en la capacidad para comunicar sistemas dispares sin su lógica de negocio (componentes UMO) nunca necesario para conocer la localización del sistema, el formato de los datos, el mecanismo de reparto o protocolos. Asimismo, cuando el componente UMO ha hecho su trabajo no necesita preocuparse de dónde van después los datos o en qué formato se esperan.

El **conector** entiende cómo enviar y recibir datos sobre un canal particular. Un recibidor de mensajes se une con un conector para registrar datos de interés procedentes de una fuente que entiende el conector.

El **transformador** se usa para transformar mensajes o cuerpos de eventos hacia y desde diferentes tipos. Mule no define un formato de mensaje estándar (aunque Mule puede soportar en el futuro un estándar de tipos de mensajes de definición de procesos de negocio). La transformación proporcionada hacia fuera del transformador es un tipo de transformación como Mensajes JMS-a-Objetos y transformadores XML estándares. La transformación de datos es muy subjetiva a la aplicación y Mule provee un simple y poderoso framework de transformación.

El **inbound router** se puede usar para controlar cómo y qué eventos son recibidos por un componente suscribiéndose en un canal. Los inbound routers se emplean para filtrar, dividir un conjunto y reordenar la secuencia de eventos antes de que los reciba el componente UMO.

Un **endpoint** es en realidad una configuración envoltorio que liga un conector, URI (Uniform Resource Identifier) endpoint, transformadores, filtros e información transaccional para proporcionar un adaptador de canal (Channel Adapter). El proveedor también guarda información transaccional para la instancia proveedora. El adaptador actúa como un cliente de mensajería para el sistema de mensajería e invoca funciones de aplicación a través de una interfaz de application-supplied (aplicación suministrada). Asimismo, el adaptador de canal puede escuchar un

evento de una aplicación interna e invocar al sistema de mensajería en respuesta a este evento.

Los endpoints son fundamentales para la capacidad de comunicación de Mule. Un endpoint define el canal de comunicación entre dos o más componentes, aplicaciones o repositorios. Ellos proporcionan un modo potente de permitir a los objetos hablar sobre cualquier protocolo de un modo único. Un endpoint puede ser configurado con filtros de mensaje, interceptores de seguridad e información de transacciones para controlar quién, qué y cómo son enviados o recibidos los mensajes por medio de los endpoints. Una vez que ha sido procesada la orden por el componente, un mensaje JMS puede enviarse sobre un asunto para notificar que el sistema está escuchando y responde sobre HTTP. Los endpoints son usados para conectar componentes en el servidor y sistemas externos o a otros localmente o sobre la red.

El **outbound router** se usa para publicar mensajes/eventos para diferentes proveedores, dependiendo de diferentes aspectos de eventos u otras reglas definidas en la configuración.

Las aplicaciones Mule constan usualmente de muchas instancias Mule a través de la red. Cada instancia es un contenedor light-weight que hospeda uno o más componentes UMO. Cada componente UMO tendrá uno o más endpoints que enviarán y recibirán eventos.

El contenedor proporciona un rango de servicios para componentes UMO tal como gestión de transacciones, transformación de eventos, enrutado, correlación de eventos, registro, auditoría y gestión. Que la construcción de objetos Mule esté separado de la gestión implica que los populares contenedores IoC/DI (Inversion of Control / Dependency Injection), tal como Spring, PicoContainer o Plexus, pueden usarse para construir componentes UMO.

Los componentes de un servidor Mule (Ver Anexo 8) son varios:

Mule Manager

CAPÍTULO 3

Mule Manager es la parte central de una instancia servidor Mule (también conocido como Nodo o Nodo Mule). Su primer papel es gestionar los objetos tales como conectores, endpoints y transformadores para una instancia Mule. Estos objetos son empleados para controlar flujos de mensaje hacia y desde los servicios/componentes y proporciona servicios al Mule Model y los componentes que maneja.

Mule Model

Model es el contenedor en el que los componentes son gestionados y ejecutados. También controla el flujo de mensajes hacia y desde los componentes, maneja hilos, ciclo de vida y pooling. El Mule Model por defecto está basado en SEDA, por lo que usa un modelo eficiente de encolamiento basado en eventos para maximizar el rendimiento y el ancho de banda.

El Mule Model encapsula y controla el comportamiento de una instancia de servidor Mule en tiempo de ejecución. Es responsable del mantenimiento de las instancias UMO y su configuración.

Proveedor de transporte

Un proveedor de transporte (Transport provider) es un plugin Mule que permite que componentes Mule envíen y reciban información sobre un protocolo particular, repositorio de mensajes u otra tecnología. A continuación se describe su arquitectura.

El conector proporciona la implementación para conectar con sistemas externos. El conector es responsable de enviar datos al recibidor externo y manejar un listener para recibir datos de un sistema externo.

Los conectores son responsables de controlar sesiones de la tecnología subyacente, por ejemplo, el JmsConnector proporciona una sesión Jms para el recibidor de mensajes del conector y para el repartidor de mensajes para publicar o enviar eventos sobre JMS.

Un endpoint se define por una dirección (endpoint address), y ésta define cualquier forma de destino o fuente de datos y está siempre expresada como una URI.

Los recibidores de mensajes (Message Receivers) o message listeners son responsables de recibir datos desde sistemas externos. La responsabilidad del conector es controlar el registro y baja de los recibidores. La complejidad del recibidor de mensajes variará dependiendo del sistema externo usado.

Los adaptadores de mensajes (Message Adapters) se requieren para leer de un modo común objetos de datos dispares en el formato que son recibidos desde la aplicación externa. Los adaptadores de mensajes son específicos de un conector; cuando un conector recibe datos, un adaptador de mensaje también es necesario para traducir esos datos de un tipo concreto usado por el conector en un array de bytes o en un string.

Las transacciones son controladas por el proveedor. Las transacciones se inician o perpetran cuando un recibidor de mensajes recibe un mensaje o un repartidor de mensajes envía un mensaje. Algo principal de las transacciones Mule es el TransactionCoordinator, que es responsable del mantenimiento del estado de la transacción. Para que Mule trate igual todas las transacciones hay una pequeña API que los proveedores deben definir.

Componentes UMO

Lo principal de la arquitectura Mule son componentes autónomos simples que pueden interactuar con la existencia vinculada a la fuente, transporte o entrega de datos. Estos componentes se llaman componentes UMO y se pueden organizar para trabajar con otro de varias maneras. Estos componentes puedes configurarse para aceptar datos de un número diferente de fuentes y puede enviar datos de vuelta a estas fuentes.

La implementación UMO real especificada referencia a un objeto. Este objeto puede ser cualquiera, un JavaBean o un componente de otro framework. Éste es el código cliente que hace algo con los eventos recibidos. Mule no hace sitio para ninguna

restricción en los objetos excepto que si se configura con Mule directamente, debe tener un constructor por defecto (si se configura vía Spring o Pico, Mule no impone convenciones en los objetos que se manejan).

Obviamente, con esta clase de flexibilidad, Mule necesita saber un poco más sobre los objetos. Éste es el trabajo de dos capas externas enseñadas para controlar cómo los eventos son utilizados por los objetos.

El adaptador de ciclo de vida es usado por el Mule Model para disparar métodos de ciclo de vida en los objetos (si existen). Los ciclos de vida personalizados pueden ser introducidos en la base del modelo permitiendo diferentes modelos para manejar el ciclo de vida de sus componentes de diferentes modos.

Mule define un ciclo de vida por defecto para los componentes que maneja. Los componentes pueden participar en ningún, alguno o todos estos eventos de ciclos de vida por medio de la implementación de los interfaces de ciclo de vida requeridas.

Eventos

Mule es una arquitectura basada en eventos, esto es, acciones sin una red Mule son disparadas por otros eventos ocurridos en Mule o en sistemas externos. Los eventos siempre contienen entidades como datos, el payload, que será usado y/o manipulado por uno o más componentes y un conjunto de propiedades que están asociados al procesado del evento. Estas propiedades son arbitrarias y puede establecerse en cualquier momento en que se cree el evento. Los datos del evento pueden ser accedidos en su estado original o en su estado transformado. El evento usará el transformador asociado con el endpoint que recibe el evento, y transforma su payload en un formato que el componente recibidor comprende.

En el procesamiento de eventos Mule puede enviar y recibir eventos usando tres modelos:

- **Asíncronamente**:

Muchos eventos puede procesarse por el mismo componente a la vez en hilos diferentes. Cuando el servidor Mule está ejecutando asíncronamente instancias de un componente en diferentes hilos, todos aceptan eventos de entrada, aunque el evento sólo será procesado por una instancia del componente.

- **Síncronamente**:

Cuando un componente UMO recibe un evento en este modo toda la petición es ejecutada en un simple hilo.

- **Petición-Respuesta:**

Éste permite a un componente UMO hacer una petición específica de un evento y esperar un tiempo determinado para conseguir la respuesta. Se puede controlar el sincronismo del procesamiento de eventos fijando la propiedad síncrono en el endpoint (o de forma programática en el evento). El sincronismo de la llamada puede ser propagado a lo largo de la red donde aquellos transportes que estén siendo usados soporten un canal de respuesta.

Enrutadores de mensajes

Los enrutadores de mensajes son usados para controlar cómo los eventos son enviados y recibidos por componentes en el sistema. Mule define inbound routers que se aplican a los eventos cuando son recibidos y que son invocados cuando un evento está siendo enviado.

Los inbound routers pueden usarse para controlar y manipular eventos recibidos por un componente. Típicamente, un inbound router puede usarse para filtrar eventos de llegada o un conjunto de eventos de llegada. Se puede encadenar inbound routers, en este escenario cada router se liga antes de que el evento sea enviado al componente Mule. Se puede especificar una estrategia match-all que será invocada si algún router no acepta el evento actual.

Interceptores

Los interceptores Mule son útiles para adjuntar comportamiento común a múltiples UMOs. El interceptor o el modelo de órdenes son frecuentemente eliminados como un práctico AOP (Aspect Oriented Programming), como esto permite al desarrollador interceptar el procesamiento en un objeto y potencialmente cambiar el procesamiento y el resultado. Los interceptores son muy útiles para adjuntar perfiles, permisos, inspecciones de seguridad, a un componente en Mule.

Aplicaciones externas

Las aplicaciones externas pueden ser cualquiera, desde aplicaciones de servidores hasta sistemas de nómina heredados, hasta grandes aplicaciones de comercio o una aplicación cliente. Básicamente, cualquier aplicación que tiene un modo de servicio o consumo de datos. Mule realiza toda la comunicación vía endpoints, los componentes UMO no tienen la noción de qué aplicación produce los datos, dónde está localizada ni el protocolo de transporte usado.

Desplegando MULE en un Ambiente de Negocio

La disponibilidad de un transporte común y el protocolo en un paquete de código abierto desde una simple descarga, con ningún coste, y con una comunidad rica alrededor de él es una de las razones para usar Mule. La comunidad de Mule ha demostrado la conectividad software que conectaba una amplia variedad de sistemas misión-críticos en instituciones financieras, líneas aéreas, comercio, y compañías de la tecnología. Mule y todos sus subcomponentes liados se licencian bajo variación de Mozilla Public License 1.1.

Mule se muestra también como un ESB comercial, y hay por lo menos una compañía que ofrece la ayuda 7x24 horas y la indemnización, dos requisitos que muchas corporaciones exigen antes de considerar cualquier software de código abierto para su despliegue. Es como si todos los artículos en la lista de comprobación corporativa son marcados como mínimo riesgo al basarse en tecnologías de código abierto. Esto repercute en los departamentos jurídicos y las compañías de IT, que pueden dormir tranquilas cuando Mule se convierte en parte de su arquitectura de negocio.

CAPÍTULO 3

Requisitos para trabajar con Mule.

Obtener e instalar Mule es un proceso simple. Los únicos prerrequisitos son un navegador web y una máquina virtual Java versión 1.4.2 o superior. El estándar Mule incorpora un número de paquetes, así como un JMS, dos plataformas de servicios webs, adaptadores, traductores, y todo aquello que se necesite.

3.3 Modo de instalación de Mule.

Pre-requisitos

Sistemas Operativos

Mule funciona en cualquier plataforma que soporte Java, estas incluyen:

- Windows XP SP2 y Windows 2000.
- Linux, Solaris, AIX y HP-UX.
- Mac OSX.

Entorno

- Java Developer Kit (JDK) 1.4.x o superior para el desarrollo y JDK 1.5.x
- La variable de entorno JAVA_HOME debe de ser declarada donde el directorio JDK esté instalado, por ejemplo c:\java\j2sdk1.4.2_08.
- La variable de entorno PATH debería contener también donde esté el directorio bin del JDK, por ejemplo c:\java\j2sdk1.4.2_08\bin.
- Maven 1.0.2 (requerido para desarrollar algunos ejemplos).
- Herramientas de compresión como WinZip (Windows) o GZip (Linux/Unix) para descomprimir Mule.
- Para los usuarios de Linux/Unix también pueden necesitar alguna herramienta tal como wget o ftp.

Espacio en Disco

- 40 MB de espacio en disco para Mule binary distribution.

- 4 MB de espacio en disco para Mini Mule distribution.

Instalación en Windows

Windows Binary Installation

- Descarga la distribución binaria de Mule.
- Haz click en el **.zip**, para comenzar la descarga de la distribución de Windows.
- Una vez que la distribución está descargada, extrae los archivos del zip al directorio. Por ejemplo: e.g. c:\java.

Instalación en Linux/Unix

Unix Binary Instalation

Los siguientes pasos le guiarán en la instalación de Mule bajo entornos Linux/Unix. Se debe asegurar que los archivos con nombres terminados en ***tar.gz*** tienen los permisos necesarios para su instalación.

Usando un navegador

- Descarga la distribución binaria de Mule.
- Una vez que la distribución ha sido descargada, extraiga los archivos del archivo .tar.gz en un directorio a su elección, por ejemplo /usr/local.
- Extraiga la distribución usando el comando *tar.* Por ejemplo, si se descargó la distribución Mule 1.3 :

tar -xvzf mule-1.3.tar.gz

Usando una Shell

Si no se tiene instalado un navegador o está instalando Mule en una máquina remota sin X-Windows siga estos pasos.

- Se necesitará conocer la versión de Mule Developer que va a instalar.

- El archivo URL tendrá la forma -

http://dist.codehaus.org/mule/nightly/mule-x.x-dist.tar.gz

donde *x.x* será reemplazada por la versión SNAPSHOT que se desee descargar.

- Cambie el directorio donde se desee instalar Mule; por ejemplo, /usr/local.
- Descargue la distribución Mule usando una herramienta de descarga. Por ejemplo para la descarga por defecto puede usar el wget -

  wget http://dist.codehaus.org/mule/nightly/mule-SNAPSHOT-dist.tar.gz

- Extraiga la distribución usando el comando *tar.* Por ejemplo, si se descargó la distribución Mule 1.3.

  tar -xvzf mule-1.3.tar.gz

Instalación Source

Esta es una vista del código fuente de Mule que se incluye en la distribución. Sin embargo, su objetivo está enfocado en que no contiene necesariamente archivos constructores que construyen, pero es útil para adjuntar archivos .Jar de Mule en tu IDE.

Apéndice: Distribución de Directorios

Cualquier distribución descargada de Mule, contiene está distribución de directorios.

- ./mule-1.3
- . /bin – Contiene el shell y batch scripts para arrancar Mule.
- . /conf – Archivos de envoltura y configuración de logeado.
- . /docs – Documentación API de Mule y de sus sub-proyectos. (No incluida en la versión Developer)
- . /examples – Ejemplos opcionales, si se trata de la distribución completa.

- ./lib – Directorio de Librerías
- ./lib/boot - Jars requequeridos para comenzar Mule
- lib/Mule - Core Mule Jars
- ./lib/opt – Jars opcionales requeridos por algunos transportes y componentes de Mule.
- ./lib/user - User-provided jars.
- . /licenses – Información sobre la licencia y todos los Jars que componen Mule.
- . /logs – Directorio de Logs.
- ./sbin - Java Service Wrapper ejecutables (en Mule 2.0, está bajo ./lib/boot)
- . /src - El código fuente de Mule y sus sub-proyectos.
- ./LICENSE.txt – Aceptamiento de la Licencia Mule.
- ./README.txt – Información sobre la distribución y de como comenzar.

3.4 Conclusiones parciales.

Con la realización de este capítulo se puede arribar a la conclusión de que Mule es el mejor bus de código abierto de servicio de negocio. Presenta la ventaja de ser gratis. Una comunidad de código abierto próspera alrededor de él. Mule está en la producción de muchas compañías por todo el mundo, desde las instituciones financieras hasta grandes usos en el comercio. El producto es fácil de instalar, de desplegar, de mantener, y de extender. Cualquier persona con una cierta comprensión de la integración de la empresa y de un editor de textos puede configurarlo.

Resulta interesante considerar Mule seriamente si se va a diseñar, evaluar, o desarrollar en fases de integración de negocios en cualquier institución que requiera de la integración de aplicaciones de forma rápida y directa. Mule es más completo que otros ESB de código abierto como por ejemplo OpenESB. Todos estos aspectos hacen de Mule una apuesta segura.

CONCLUSIONES

Cuando surgió la ciencia de la computación, quizás nunca se pensó en el alcance que esta iba a llegar a tener, probablemente sólo se pensó en hacer algo simple que permitiera compartir información rápidamente. Hoy en día se puede decir que el mundo se mueve gracias a que se comparte información.

El rápido avance de las tecnologías en el campo de la informática y su gran aceptación a nivel mundial hace que cada día se de más importancia al uso de las tecnologías. Es por eso que surge este proyecto con el objetivo de insertar en la UCI los últimos adelantos de la ciencia. Una vez concluido este trabajo, se cuenta con una propuesta de un Bus de Servicio Empresarial (ESB) para ser instalado en los servidores de producción de nuestra universidad.

Con esta propuesta se podrá contar con una comunicación fiable entre los distintos recursos tecnológicos tales como aplicaciones, plataformas y servicios, que están distribuidos en múltiples sistemas por toda la UCI y así de esta manera se podrán resolver muchos de los problemas de mensajería existentes. Por todo lo anterior se concluye que los objetivos propuestos para el presente proyecto han sido cumplidos satisfactoriamente.

RECOMENDACIONES

RECOMENDACIONES

Una vez cumplidos los objetivos de este trabajo, y teniendo en cuenta las experiencias obtenidas durante la realización del mismo, se recomienda:

- Tratar de poner en práctica cuanto antes el ESB en la UCI y comenzar a sentar las bases para en un futuro no muy lejano obtener resultados satisfactorios.
- Profundizar en el estudio de cómo se configura el ESB propuesto.
- Seguir investigando sobre los ESB.

REFERENCIAS BIBLIOGRÁFICAS

REFERENCIAS BIBLIOGRAFICAS

- **CIENTEC. 2007.** CIENTEC. BAM: UNA MIRADA AL NEGOCIO EN TIEMPO REAL. [Online] 2007. http://www.cientec.com/Management/Management12.asp
- **Gómez Rubio, Alvaro. 2007.** ESB...El corazón de SOA. [En línea] 12 de Julio de 2007. http://alvarogomezrubio.blogspot.com/2007/07/esbel-corazn-de-soa.html.
- **Guinea de Salas, Alejandro y Jorrín Abellán, Sergio.** Arquitectura SOA para la integracion entre software libre y software propietario en entornos mixtos. [En línea] http://www.sigte.udg.es/jornadassiglibre2007/comun/1pdf/13.pdf
- **Informatización.uci.cu. 2007.** Arquitectura para los Sistemas que Conforman la Intranet Universitaria. Ciudad Habana : s.n., 2007.
- **Technology for Solutions. 2007.** Technology for Solutions. Administración de Procesos de Negocios. [Online] 2007. http://www.tfsla.com/bpm.php

BIBLIOGRAFÍA

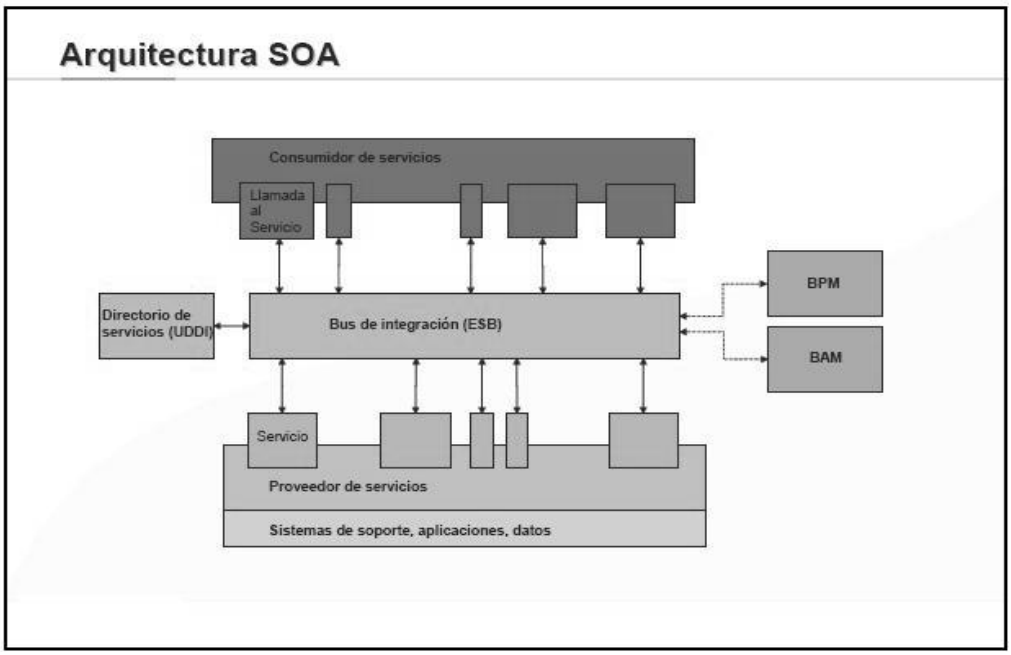
1. Información Sobre BPM. [En línea] 15 de Mayo de 2008. [Citado el: 12 de Febrero de 2008.] http://es.wikipedia.org/wiki/Business_Process_Management.
2. Introducción a la arquitectura orientada a servicios. [En línea] 2008. [Citado el: 16 de Febrero de 2008.] http://www.microsoft.com/spanish/msdn/articulos/archivo/121205/voices/SOADesign.mspx.
3. UDDI. [En línea] 7 de Mayo de 2008. [Citado el: 22 de Febrero de 2008.] http://es.wikipedia.org/wiki/UDDI.
4. Celtix. [En línea] 16 de Febrero de 2006. [Citado el: 14 de Marzo de 2008.] https://wiki.objectweb.org/celtix/.
5. ServiceMix. [En línea] 18 de Marzo de 2008. [Citado el: 13 de Abril de 2008.] http://www.servicemix.org.
6. MuleSource Inc . Mule. [En línea] 2008. [Citado el: 15 de Abril de 2008.] http://mule.mulesource.org/display/MULE/Home.
7. Descarga de Mule. Mule. [En línea] 2008. [Citado el: 5 de Mayo de 2008.] http://mulesource.org/display/MULE/Download.
8. Licencia de Mule. [En línea] 2008. [Citado el: Abril de 24 de 2008.] http://opensource.org/licenses/mozilla1.0.html.
9. Configuración de Mule. [En línea] 2008. [Citado el: 7 de Mayo de 2008.] http://mule.mulesource.org/display/MULE2USER/Mule+2.0+Configuration+Guide.
10. Recena Soto, Manuel J. SOA,una Perspectiva. [En línea] 4 de Julio de 2007. [Citado el: 15 de Marzo de 2008.] http://static.scribd.com/docs/4msq2v2qjojpm.swf?INITIAL_VIEW=width.
11. Información sobre los ESB. [En línea] http://del.icio.us/swwsman/esb+opensource.

BIBLIOGRAFÍA

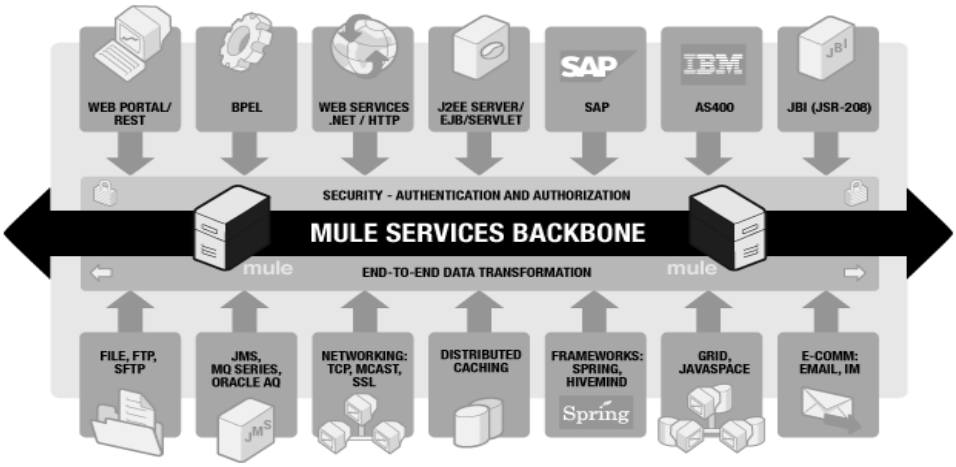
12. Soa y OpenSource. [En línea] [Citado el: 26 de Marzo de 2008.] http://opensourceesb.blogspot.com/.
13. Foro. Mule. [En línea] Nabble. http://www.nabble.com/Mule-f2725.html.
14. Muñoz, Daniel. Instalación de Mule. [En línea] 28 de Junio de 2007. [Citado el: 9 de Mayo de 2008.] http://proyectomule.wordpress.com/.
15. Rademakers, Tijs y Dirksen, Jos. Libro de los ESB. [En línea] [Citado el: 5 de Marzo de 2008.] http://www.manning.com/rademakers/rademakers_meapch1.pdf.
16. Muñoz, Daniel. Aprendiendo a trabajar con Mule. [En línea] [Citado el: 10 de Mayo de 2008.] http://proyectomule.files.wordpress.com/2007/03/mule-a-case-study-v11.pdf.
17. Ciurana, Eugene. Mule un caso de estudio. [En línea] Enero de 2007. [Citado el: 11 de Mayo de 2008.] http://www.theserverside.com/tt/articles/article.tss?l=CaseStudyMule

ANEXOS

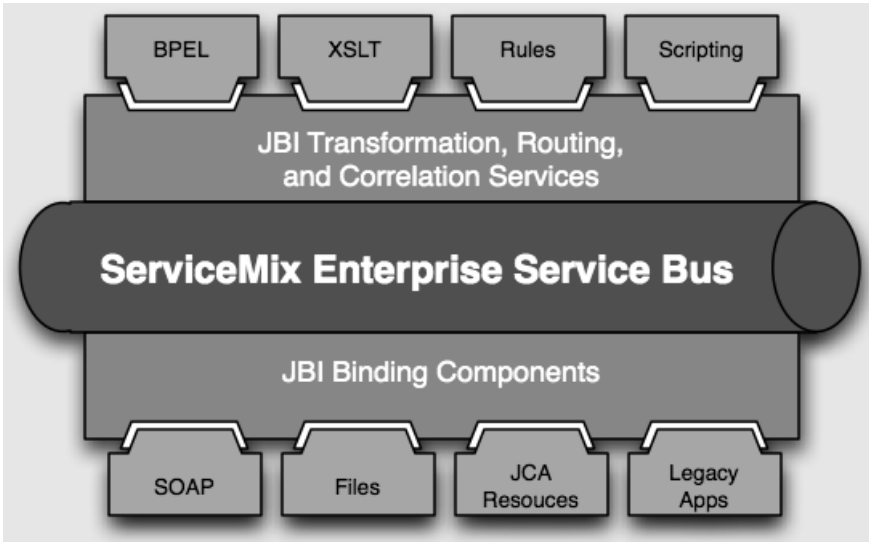
- **Anexo 1** Componentes que conforman a SOA.



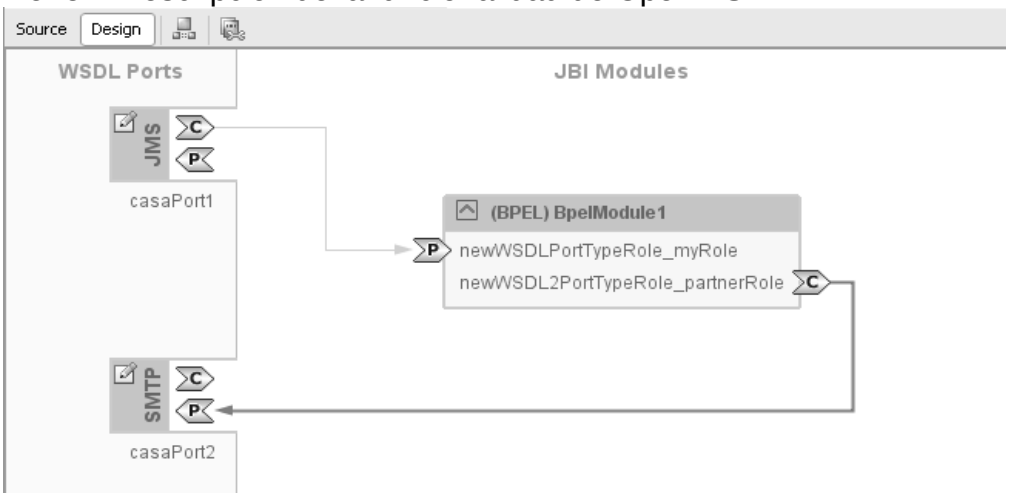
- **Anexo 2** Descripción de la funcionalidad de MULE mostrado en la página web.



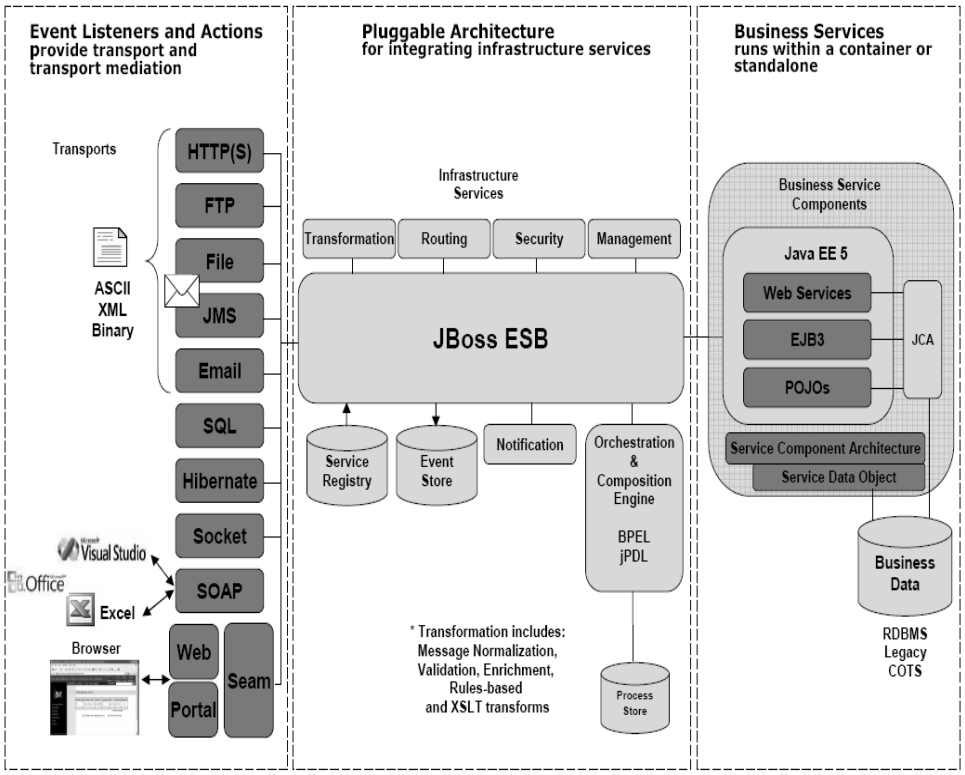
- **Anexo 3** Muestra un alto nivel de la arquitectura ServiceMix.



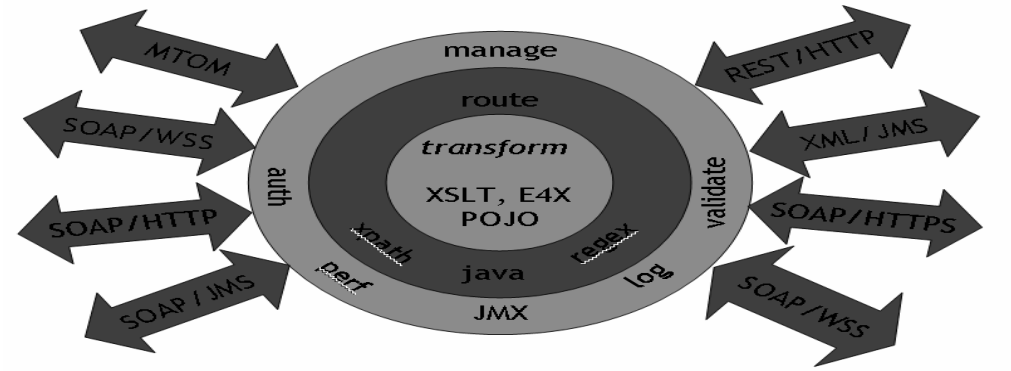
- **Anexo 4** Descripción de la funcionalidad de Open ESB



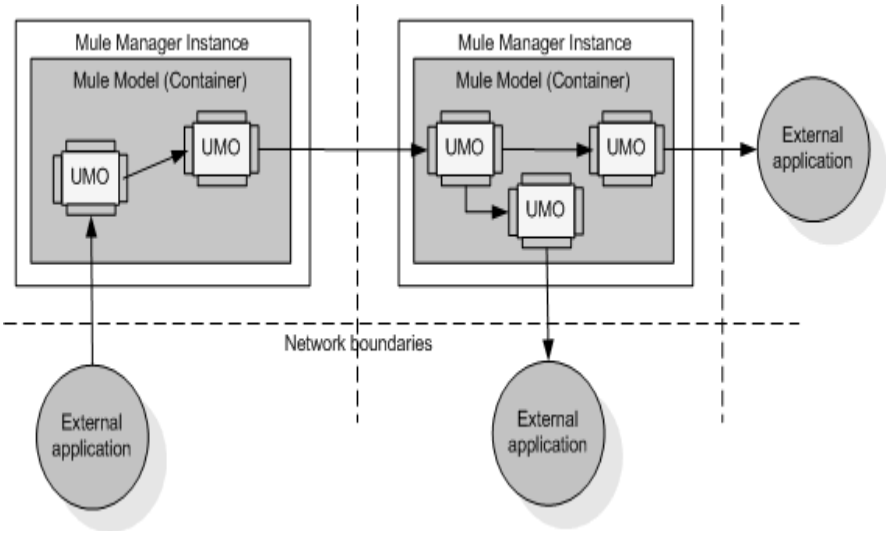
❖ **Anexo 5** Funcionalidad que puede proporcionar el JBoss ESB.



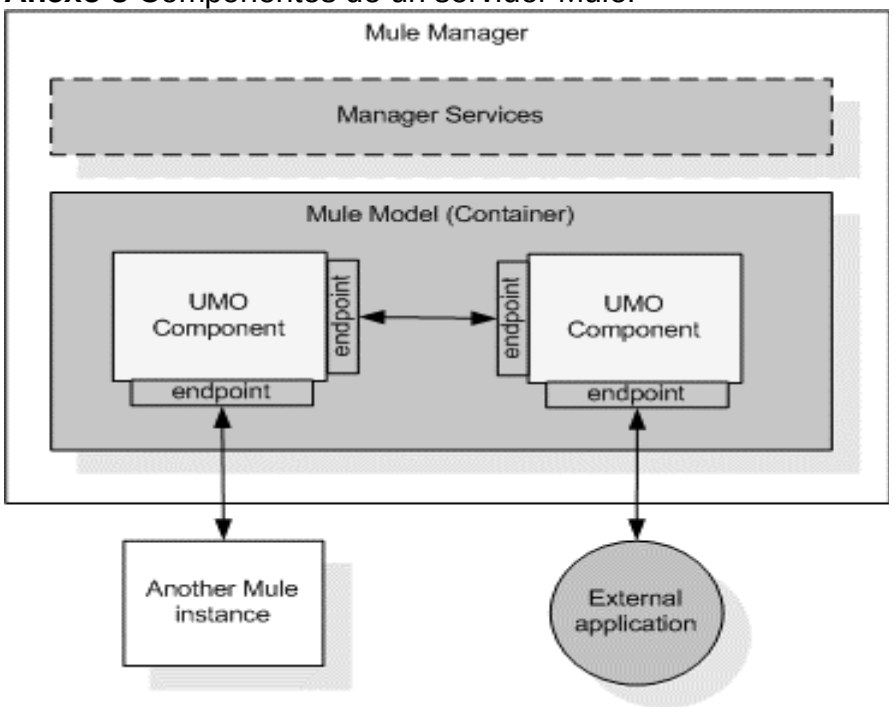
❖ **Anexo 6** Breve resumen acerca de la funcionalidad que proporciona Synapse.



❖ **Anexo 7** Escenario común utilizando Mule.



❖ **Anexo 8** Componentes de un servidor Mule.



GLOSARIO

Acegi: es un framework de seguridades open source que permite mantener la lógica de negocio libre de código de seguridad.

API (Interfaz de Programación de Aplicaciones): es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Backbone: se refiere a las principales conexiones troncales de Internet. Está compuesta de un gran número de routers comerciales, gubernamentales, universitarios y otros de gran capacidad interconectados que llevan los datos entre países, continentes y océanos del mundo.

Bombas lógicas: programa informático que se instala en un ordenador y permanece oculto hasta cumplirse una o más condiciones preprogramadas para entonces ejecutar una acción.

BPEL (Lenguaje de Ejecución de Procesos de Negocio): es un lenguaje para especificar el comportamiento de los procesos de negocio basados en servicios web. Procesos en WS-BPEL exportación e importación utilizando la funcionalidad de las interfaces de servicios Web exclusivamente.

BPR (Reingeniería de procesos de negocios): es un enfoque de gestión destinado a la mejora de los medios para elevar la eficiencia y la eficacia de los procesos que existen dentro y entre organizaciones.

Bugs: Un defecto de software (computer bug en inglés), es el resultado de un fallo o deficiencia durante el proceso de creación de programas de ordenador o computadora (software).

GLOSARIO

Cortafuego: también llamado Firewall, es un elemento de hardware o software utilizado en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red que haya definido la organización responsable de la red.

EAI (Enterprise Application Integration): EAI es la integración de nuevas aplicaciones con las ya existentes, incluyendo las aplicaciones heredadas o los paquetes de software, de forma que todas juntas proporcionen las funcionalidades necesarias para soportar los procesos de negocio de la empresa. Esta integración permite a la organización mantener el ritmo de los cambios del mercado y reaccionar a tiempo frente a ellos.

EDA (Event Driven Architecture): es un emergente estilo de arquitectura de aplicaciones que permite que ciertos eventos activen automáticamente mensajes de notificación, contribuyendo a que los usuarios tomen mejores decisiones, idealmente en tiempo real.

EJB: Los Enterprise JavaBeans (también conocidos por sus siglas EJB) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems (ahora JEE 5.0).

Endpoints: es el nombre de la entidad en un extremo de una capa de transporte, el punto final de comunicación. En arquitectura orientada al servicio, es el punto de entrada para un servicio, un proceso, o una cola de destino o tema.

Framework: En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

FTP: es un protocolo de transferencia de archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor.

Glassfish: es un servidor de aplicaciones que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta

especificación. Es gratuito y de código libre, se distribuye bajo la licencia CDDL y la GNU GPL.

HTTP: El protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web (WWW). HTTP fue desarrollado por el consorcio W3C y la IETF, colaboración que culminó en 1999.

IDE (Entorno de Desarrollo Integrado): es un programa compuesto por un conjunto de herramientas para un programador.

Interfaz: puede definirse como el conjunto de comandos y/o métodos que permiten la intercomunicación del programa con cualquier otro programa o entre partes (módulos) del propio programa o elemento interno o externo.

Java de autenticación y autorización de servicios (JAAS): es una Interfaz de Programación de Aplicaciones que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso.

Java: es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90.

JBI (Java Business Integration): es una especificación desarrollada en el marco del Java Community Process (JCP) para un enfoque de la aplicación de una arquitectura orientada a servicios (SOA). La referencia es JCP JSR 208.

JDBC (Java Database Connectivity): un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

JDBC: es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

GLOSARIO

JMS: La API de Servicios de Mensajería de Java es la solución creada por SUN para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma de Java 2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

JMX (Java Management Extensions): JMX es la tecnología que define una arquitectura de gestión, la API (Application Programming Interface), los patrones de diseño, y los servicios para la monitorización/administración de aplicaciones basadas en Java.

JSR (Java Specification Request): son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java.

Key Performance Indicators (KPIs): Indicadores clave de rendimiento, miden el nivel del desempeño de un proceso, enfocándose en el "como" e indicando como de buenos son los procesos, de forma que se pueda alcanzar el objetivo fijado.

Ldap (Lightweight Directory Access Protocol): es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.

Log: es un registro de algo. Un registro oficial de eventos durante un periodo de tiempo en particular. Para los profesionales en seguridad informática un log es usado para registrar datos o información.

Mainframe: Una computadora central o mainframe es una computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias.

MEP (Patrones de Intercambio de Mensajes): Los Message Exchange Patterns (MEP) determinan las diferentes formas de intercambiar mensajes entre el consumidor y el proveedor del servicio.

GLOSARIO

Middleware: es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

MOM (Message-Oriented Middleware): Es un tipo de Middleware, cuando los mensajes enviados al cliente se recogen y se almacenan hasta que son solicitados, mientras el cliente continúa con otros procesos.

MQ Series: Es un middleware de la familia de software de IBM.

MSMQ: es una herramienta de Microsoft para realizar el manejo de colas de mensajes.

NMR (Normalised Messages Router): El enrutador de mensajes normalizados es el componente que normaliza las rutas de los mensajes de un componente fuente a su destino utilizando algún tipo de política de enrutamiento para decidir el punto final a su uso.

OASIS: Acrónimo de (Organization for the Advancement of Structured Information Standard) es un consorcio internacional sin fines de lucro que orienta el desarrollo, la convergencia y la adopción de los estándares e-business.

Open Source: Código abierto (en inglés open source) es el término con el que se conoce al software distribuido y desarrollado libremente. Fue utilizado por primera vez en 1998 por algunos usuarios de la comunidad del software libre.

Orquestación de los procesos: es el esqueleto de una aplicación compuesta, en la que a través de lenguajes formales se permite definir el flujo de actividades y estados por los que ha de pasar un proceso de empresa para su realización (un ejemplo bastante extendido es BPEL (Lenguaje de Ejecución de Procesos de Negocio)).

Protocolo de red: conjunto de estándares que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red.

Proxy: hace referencia a un programa o dispositivo que realiza una acción en representación de otro.

Puertas traseras: es una secuencia especial dentro del código de programación mediante la cual el programador puede acceder o escapar de un programa en caso de emergencia o contingencia en algún problema. A su vez, estas puertas también pueden ser perjudiciales debido a que los crackers al descubrirlas pueden acceder a un sistema en forma ilegal y aprovecharse la falencia.

PXE (Preboot Execution Environment): es un medio para arrancar los ordenadores utilizando una interfaz de red independiente de los datos disponibles dispositivos de almacenamiento (como discos duros) o sistemas operativos instalados.

RPC (Remote Procedure Call): es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

SDK (kit de desarrollo de software): Un Software Development Kit (SDK) o kit de desarrollo de software es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema bastante concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, ordenadores, videoconsolas y sistemas operativos .

Securización: es el proceso mediante el cual se implementa una política de seguridad específica sobre una instalación de un sistema operativo. El bastionado de un equipo intenta reducir el nivel de exposición de un equipo y, por tanto, los riesgos y vulnerabilidades asociados a éste.

SMTP: Simple Mail Transfer Protocol (SMTP), o protocolo simple de transferencia de correo. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras o distintos dispositivos.

GLOSARIO

Sockets: designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada.

SSL (Secure Sockets Layer): Es un protocolo criptográfico que proporciona comunicaciones seguras en Internet.

URI (Identificador Uniforme de Recurso): Un URI es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico y enciclopedia). Normalmente estos recursos son accesibles en una red o sistema.

W3C (World Wide Web Consortium): es un consorcio internacional que produce estándares para la World Wide Web. Está dirigida por Tim Berners-Lee, el creador original de URL (Uniform Resource Locator, Localizador Uniforme de Recursos).

Web Services: Un servicio web (en inglés Web Service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Workflow: básicamente es una representación de una secuencia de operaciones, declarado como el trabajo de una persona, el trabajo de un simple o complejo mecanismo, el trabajo de un grupo de personas, el trabajo de una organización de personal o máquinas.

WS-Abordar: apoya normalizó servicio web direcciones, que permite múltiples transporte que se utilizará (además de HTTP).

WSDL (Web Services Description Language): describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

WSDL: son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web.

WS-Policy: es una especificación que permite a los servicios web utilizar XML para anunciar sus políticas (en materia de seguridad y la calidad del servicio.) y para los consumidores de servicios Web para especificar sus necesidades de la política.

WS-Security (Seguridad en Servicios Web): es un protocolo de comunicaciones que suministra un medio para aplicar seguridad a los Servicios Web.

XML (Lenguaje de marcas extensible): sigla en inglés de Extensible Markup Language, es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML).

XMPP: Extensible Messaging and Presence Protocol (Protocolo ampliable de mensajería y [comunicación de] presencia), es un protocolo abierto y ampliable basado en XML, originalmente ideado para mensajería instantánea.

Xpath: (XML Path Language) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML.

Xquery: es un lenguaje de consultas (con algunas características de lenguaje de programación) que está diseñado para consultar las colecciones de datos XML. Es semánticamente similar a SQL.

XSLT (lenguaje de transformación basado en hojas de estilo): es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.