9 març
B6 - Sala Juntes

Título:	**Integración de aplicaciones de seguridad**
Volumen:	**1**
Alumno:	**Jonatan Guillen Ruiz**
Director/Ponente:	**Francisco Jordan Fernández**
Departamento:	**Arquitectura de Computadores**
Fecha:	**27/02/2009**

Agradecimientos

Me gusta que todo lo que me propongo llegue a buen termino y quiero agradecer a toda las personas que han hecho esto posible:

A mi familia, en especial a mis padres Gines y Josefa, por darme la oportunidad de empezar los estudios de Ingeniería Informática y animarme para seguir durante estos largos años.

A Ángel Toribio, por proponerme la oportunidad de acabar la carrera en esta empresa después de cursar Proyecto de Redes de Computadores en la Universidad Politécnica de Cataluña.

A Francisco Jordán, por ofrecerme un hueco en una empresa joven e internacional como Safelayer, por ser tan abierto a los proyectistas que trabajamos con él y por esa visión tan especial que tiene del mundo. Con otro director no habría sido lo mismo.

A Helena Pujol, por su ayuda como usuario de una de las arquitecturas que he montado y por sus sugerencias que han hecho que mejore el proyecto.

A Carlos Ares, por aguantarme cuando le tenía que molestar para que me aconsejara sobre algunos aspectos técnicos.

A José Manuel Rivera, por ceder a muchas de mis peticiones que complicaban su indispensable trabajo.

A todo Safelayer en general por su colaboración en el proyecto y por tratarme como uno mas dentro de este gran grupo.

A los becarios infiltrados, Carlos de Haro y Ana Ballester,
a los becarios que se han ido, Alex y Miguel Ángel,
a los becarios que se han quedado, David y Víctor,
a los becarios que me acompañan, Jorge y Julia,
y a los becarios que vendrán,
que han hecho mas ameno este año y espero que lo sigan haciendo, y por responder todas mis dudas sean del ámbito que sean.

A todos ellos y a muchos más, ***gracias.***

Resumen

Poco a poco la informática ha inundando las empresas y en estos momentos muchas de las mejoras se hacen añadiendo mas servicios a una arquitectura ya de por si muy complicada. El crecimiento de las arquitecturas ha hecho que surja un nuevo problema, la administración, comunicación e interoperabilidad entre todos los servicios.

Los ESB, Enterprise Service Bus, solucionan este problema ofreciendo una gran capacidad de integración y administración de los servicios. La tecnología de ESB parece apuntar a día de hoy como un pilar principal de las arquitecturas orientadas a servicios (SOA), y aun y así, debe entenderse ésta como una tecnología incipiente y no se puede saber hasta que punto puede llegar en un futuro.

Por otro lado, la seguridad esta siendo actualmente una de las apuestas mas importantes de las empresas. Asegurar digitalmente todo lo que anteriormente se aseguraba en cajas fuertes o en papel jurídicamente es el reto de la era digital y de la nueva sociedad de la información. Cada vez mas todo se esta dirigiendo a un mundo en el que la "confianza" debe desarrollarse de forma virtual, esto es, ni física ni presencialmente. Es necesario que esta confianza parta de la aplicación correcta de mecanismos y servicios de seguridad básicos (autenticación, integridad y confidencialidad), pero sin embargo, estos no son suficientes para el perfecto desarrollo de la confianza. Para llegar a una sociedad de la información real, son necesarios mecanismos y servicios de seguridad avanzados (evaluación de la confianza basada en terceras partes, no repudio de la información, validez legal de la información digital, conservación a lo largo del tiempo de la información de forma segura, etc.), y estos mecanismos y servicios deben introducirse en los sistemas de información corporativos de la forma menos intrusiva y más eficaz (interoperable y escalable) posible.

Los capítulos de esta memoria recogen los resultados de este trabajo final de carrera que parte i) de una investigación y estudio del arte de una nueva tecnología denominada ESB, y ii) con el propósito de integrar en cualquier organización, a través del Enterprise Service Bus, mecanismos y servicios de seguridad que permitan el desarrollo eficaz de la confianza en la información. En concreto, para ello se ha escogido un producto de seguridad avanzado de la empresa SAFELAYER denominado TrustedX. En esta memoria se incluye y documenta desde la recopilación de todas las herramientas que se han escogido, probado y comparado, hasta dos casos de uso complejos donde se aprovecha el uso de la tecnología.

Índice

Capítulo 1
Introducción

A día de hoy es difícil encontrar una empresa que no esté informatizada. Esto ha significado un cambio importante en los procesos internos y en el flujo de la información. Hasta hace poco tiempo la mayoría de los procesos se realizaban de forma manual, mientras que en la actualidad se intenta que absolutamente todo se efectúe de manera automatizada. Esto es especialmente importante teniendo en cuenta los volúmenes de información en formato digital que manejan hoy en día las compañías, en las que infraestructuras inmensas con un elevado número de aplicaciones procesan millones de unidades de datos.

Se podría definir el flujo de información como un conjunto de mensajes que se trasladan de un punto a otro. En este flujo no sólo encontramos a los mensajes, existen más elementos que serán los encargados de interaccionar con los mismos.

Cada vez hay más tipos de componentes que realizan diferentes funciones, y esta lista está condenada a aumentar. Dentro de esta serie de elementos se pueden diferenciar dos grandes grupos: los que hacen que el flujo acabe en ellos, y los que después de interaccionar con el mensaje vuelven a reactivar la circulación con uno o varios más.

Una vez llegado a este punto se puede visualizar todo este proceso de una manera semejante a esta:

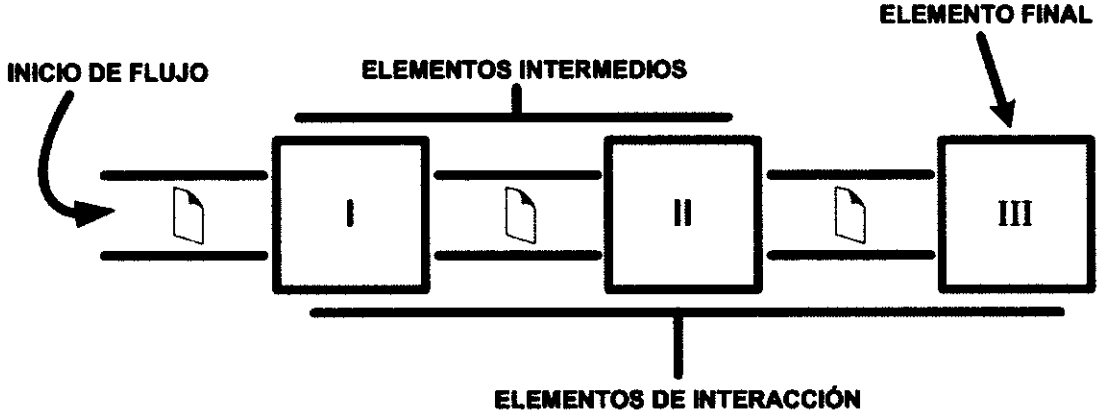


Figura 1 – Flujo de mensajes I

Los bloques de información pueden tener muchos tipos de formato. Según el componente hacia donde se dirijan necesitarán estar formateados de una manera u otra. Esto implica que el elemento anterior tiene que conocer qué elemento será el siguiente para transformar el mensaje.

Además, la estructura de las grandes empresas no es tan lineal como la mostrada en la figura, sino que tiende a ramificarse. Por ejemplo, un elemento podría alimentar a dos o más elementos. Esto implica que no sólo tendría que conocer el formato de un componente, sino el de un grupo de ellos. Es más, llegado el momento posiblemente sólo tendría que alimentar a un componente de los dos según unas determinadas condiciones.

Tradicionalmente, la inserción de un nuevo componente en el flujo en funcionamiento supondría: la seguridad que éste funciona correctamente, una implementación en el elemento anterior para que al nuevo le llegara el mensaje en el formato preciso y una

segunda implementación detrás de este nuevo componente para alimentar correctamente a los siguientes.

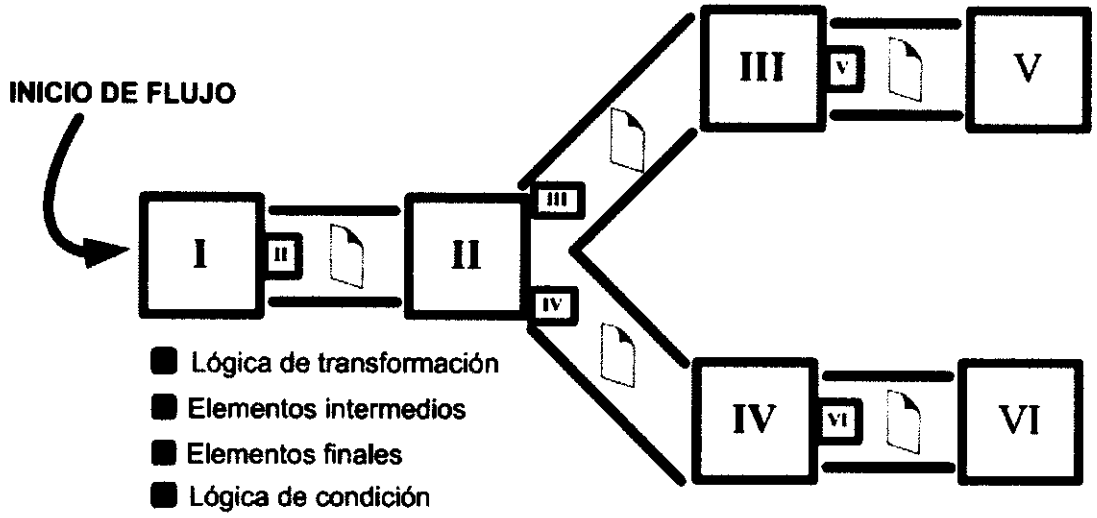


Figura 2 - Flujo de mensajes II

Como se puede observar en la Figura 2 hemos añadido la lógica de transformación después de cada elemento, incluso en algunos se han añadido varias al haber múltiples caminos. A su vez se ha tenido que añadir una lógica de condición en el elemento II, para obligar a algunos mensajes a que sólo se dirijan hacia uno de los caminos posibles.

Cada vez que se plantea una sencilla mejora al flujo, éste se complica. La lógica que se tiene que añadir cada vez es más extensa y la posibilidad de cambios posteriores se va dificultando.

En este punto aún se puede generalizar más el problema. Un flujo tiene una entrada y un final (una salida), esto hace que se pueda utilizar como un elemento en una estructura mayor, un flujo de flujos.

En conclusión, como se puede ver estamos delante de un problema. Los flujos se van haciendo grandes y van integrándose como elementos en otros flujos de mayor nivel. A su vez las lógicas de transformación y de condición van aumentando y al final la gestión de toda la estructura se complica y pasa a ser insostenible.

1.1 Enterprise Service Bus

La tecnología ESB, Enterprise Service Bus, soluciona el problema que se ha presentado anteriormente, la gestión del flujo de datos. Facilita la creación de flujos, la integración de nuevos componentes, la transformación de mensajes y soporta múltiples protocolos de entrada y salida.

Al hacer uso de un ESB la Figura 2 cambiaría a:

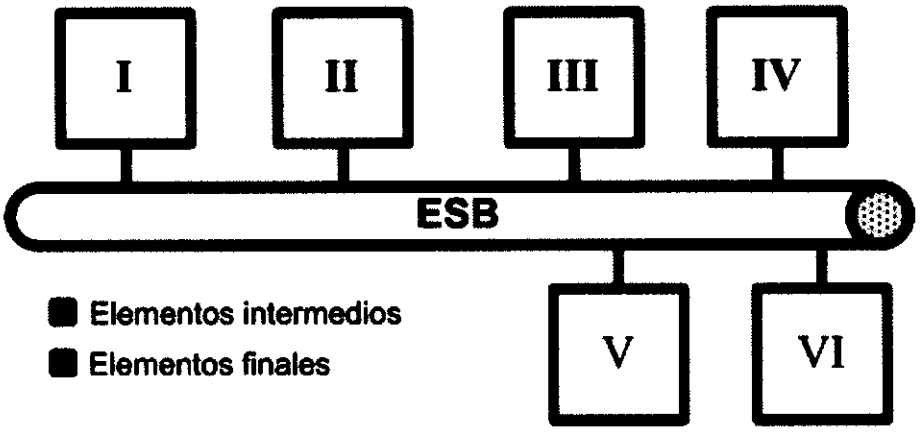


Figura 3 – Flujo Enterprise Service Bus I

Como vemos en la Figura 3 los elementos han pasado a estar conectados al ESB, no aparece el inicio de flujo, se ha eliminado toda la lógica en los componentes y no existe un camino ordenado para saber por qué elemento pasará antes el mensaje. Todo lo que ha desaparecido lo gestiona el ESB.

Con el ESB los componentes no se han de preocupar por el formato de los mensajes. El propio ESB tiene un formato interno al cual transforma todos los otros formatos que reconoce (que son muchos, xml, pdf, jpeg, etc...). De esta forma, cuando se quiere unir un componente al ESB sólo debemos asegurarnos de que éste soporta su formato. En cuanto a las transformaciones de un formato a otro, el ESB ya tiene las herramientas necesarias para realizarlas. Por otra parte, la conexión al ESB es simple ya que éste soporta múltiples protocolos de conexión (por ejemplo, ftp, file, mail, etc...).

La gestión de las soluciones ESB para crear y modificar flujos suele ser visual e intuitiva. Una vez registrados todos los componentes que formarán el flujo sólo hay que interconectarlos y definir las transformaciones necesarias. También es el lugar donde aplicaremos la lógica de condición para definir cuando tiene que pasar el mensaje por uno u otro elemento. A su vez definimos el inicio del flujo.

1.2 Seguridad, Confianza y Tecnologías Semánticas

Aunque una tecnología como el ESB sirve para la integración y gestión de flujos de aplicación en general y es la base de la concepción moderna de servicios web y las arquitecturas orientadas a servicios (SOA), en este proyecto se quiere explotar esta tecnología para facilitar la integración y gestión de la seguridad y confianza en la información dentro de la corporaciones y en sus relaciones con el exterior.

Es muy importante que la información esté protegida y para ello algunos de los mecanismos que nos interesan en este proyecto son los basados en la tecnología de infraestructura de clave pública (PKI), y sobre todo la firma y cifrado digital. La integración de estos mecanismos en los entornos de aplicación modernos puede resultar más sencilla si se utiliza una tecnología como la del ESB. Este proyecto plantea y experimenta este concepto en varios casos de uso.

Por otra parte, la explosión de información que vivimos hace que la gestión de ésta cada vez sea más compleja. En particular, la explotación de la información existente para la generación de seguridad y confianza es de gran interés en este proyecto. Para ello se trata de combinar la potencia de integración de los ESB junto con la formalización y explotación de esta información a través de las tecnologías semánticas.

1.3 Los ESB y el ROI

ROI o retorno de la inversión, como se extrae de [13], es el beneficio que obtenemos por cada unidad monetaria invertida en tecnología durante un periodo de tiempo. Suele utilizarse para analizar la viabilidad de un proyecto y medir su éxito. En épocas de crisis, se convierte en fundamental que cada céntimo invertido en tecnología regrese, a ser posible, acompañado de más.

Algunos beneficios derivados de aplicar los ESB para la integración de los productos de una empresa son:

- **Menores costes en la integración de nuevos productos.**
- **Facilidad de interacción entre aplicaciones.**
- **Menores costes de desarrollo.**
- **Menores costes de mantenimiento y soporte.**

Actualmente este es uno de los temas que mas importa a las empresas, no quieren invertir en algo sino es porque les va a retornar lo invertido en un futuro. Aquí es donde el ESB adquiere un papel importante, y es que la inversión en infraestructuras de integración asegura un retorno de la inversión a corto plazo. Tras destinar una parte del dinero a mejorar la eficiencia operativa de los sistemas tecnológicos, integrando la extensa y compleja red de aplicaciones, las inversiones se transforman en beneficios económicos.

1.4 Objetivos del proyecto

El proyecto pretende realizar un análisis del estado del arte y un estudio sobre los diferentes productos que implementan la tecnología Enterprise Service Bus. Estos productos se encuentran aún en evolución y desarrollo y en algunos casos necesitan aún mejorar pero cada uno ofrece un punto de vista diferente a la tecnología.

Para realizar el estudio se planteó hacer pruebas simples, para más tarde irlas complicando. Una vez se hayan hecho las suficientes pruebas se procederá ha hacer una comparación entre todos los ESB.

A continuación se implementara dos casos de uso complicados, uno orientado a integrar aplicaciones de seguridad y el otro a promocionar el uso de las tecnologías semánticas relacionadas con la seguridad. Los dos tendrán como objetivo mostrar el potencial de los ESB.

Para el desarrollo del proyecto se han planteado un conjunto de pautas a seguir para cumplir los objetivos. Estos objetivos se encuentran plasmados en los puntos siguientes:

1. **Estudio de la tecnología:** El primer punto se basa en el estudio preliminar de los Enterprise Service Bus. Al ser una tecnología nueva y desconocida debe empezase por un estudio del estado del arte procedente de los desarrolladores e impulsores de la tecnología y por conocer los diferentes productos que se están desarrollando. Por otra parte, también se deben estudiar las tecnologías alrededor de i) los servicios de seguridad y confianza (a través de la plataforma TrustedX) y ii) de herramientas y tecnologías de web semántica.
2. **Ensayos prácticos y comparativa:** Una vez concluido el estudio teórico de la tecnología es necesario un estudio práctico del estado del arte de dicha tecnología. Este estudio conlleva empezar la realización de prácticas, desde las más sencillas, hasta algunas más complejas. La conclusión de las pruebas dará lugar a una comparativa entre los distintos ESB.

3. **Casos de uso:** Cuando se gane la suficiente experiencia con las pruebas realizadas se diseñaran dos casos de uso relacionados con el objetivo del proyecto, integrar una aplicación de seguridad.
   3.1. **Primer caso de uso:** Este caso estará orientado a la integración de TrustedX con alguno de los ESB que se hayan probado. Tras la elección se diseñara una configuración para soportar las peticiones más variadas del producto
   3.2. **Segundo caso de uso:** En este caso se unirá el ESB a una base de datos de tripletas (base de conocimiento semántico) para convertirlo en el eje de un proyecto semántico a gran escala. Más tarde se irán añadiendo las aplicaciones para formar todo el proyecto.
4. **Redacción de la memoria** orientada a permitir al lector conocer las tecnologías estudiadas y a que sirva como guía para el uso del prototipo final.

1.4.1 Arquitectura

1.4.1.1 Integración con TrustedX

El primer caso de uso que se ha planteado es la integración de una plataforma comercial de seguridad, en este caso TrustedX, con el ESB. La idea principal es ofrecer más opciones de accesibilidad a los servicios que ofrece la plataforma.

TrustedX ofrece varios servicios de firma electrónica a través de Web Service y se quiere ofrecer la posibilidad de acceder a estos servicios a través de carpetas, sin necesidad de interacción de una tercera aplicación. Por una parte, se quiere dar total accesibilidad a estos servicios a personas que no conozcan la tecnología Web Service, a través de carpetas, y por otra parte, se quiere simplificar la integración de aplicaciones legacy (propietarias) que necesitan de servicios de firma o cifrado digital de múltiples documentos (por ejemplo, la factura electrónica) y de forma masiva, para que el ESB orqueste todo el proceso.

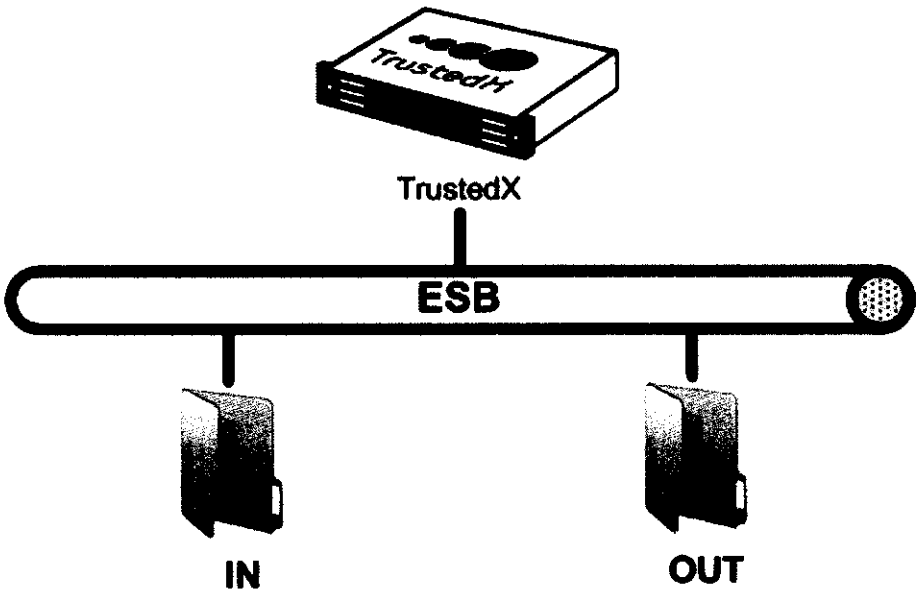


Figura 4 – Arquitectura básica del primer caso de uso

En la Figura 4 se puede ver un ejemplo de una estructura simplificada de lo que sería la integración de un servicio de TrustedX. Se pondría un archivo (por ejemplo un PDF) en la carpeta IN e instantes después lo encontraríamos en la carpeta OUT después de haber pasado por TrustedX.

1.4.1.2 Web service semántico

Por otro lado se ha realizado un servicio Web de almacenamiento de datos RDF con la ayuda de un ESB. En este caso aparte de la integración se ha implementado el servicio Web para la interacción de las aplicaciones con la base de datos, por esta razón ha supuesto una carga mayor de trabajo.

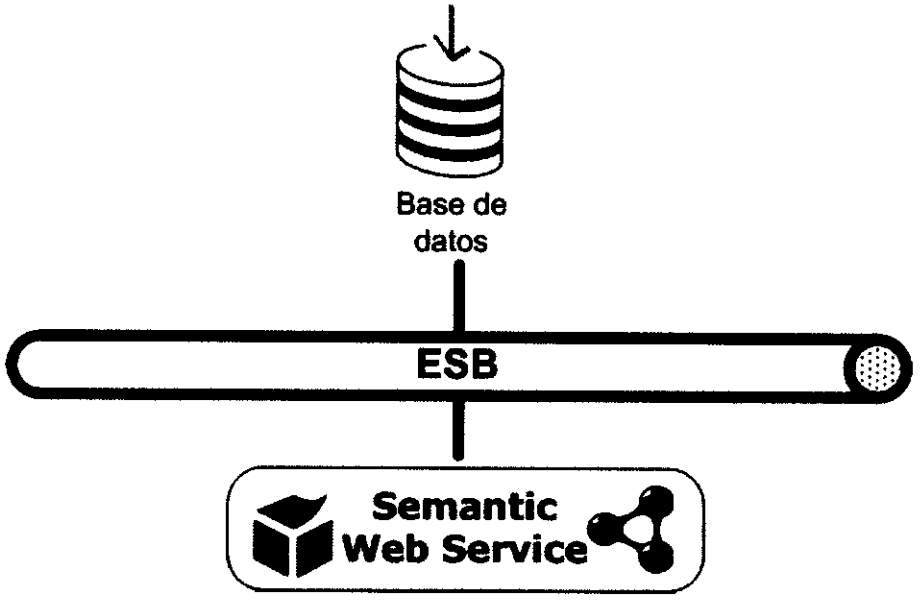


Figura 5 – Arquitectura básica del segundo caso de uso

1.5 Contexto del proyecto

Este proyecto surgió como iniciativa de la empresa ***Safelayer Secure Communications, S.A.*** en Febrero de 2008 y ha sido realizado durante los últimos meses bajo la modalidad B de proyectos de final de carrera y en régimen de beca UPC. Este trabajo ha sido realizado en el marco del proyecto Seguridad y Confianza en la Sociedad de la Información (SEGUR@), subvencionado por CDTI, Ministerio de Industria, Turismo y Comercio de España, dentro del programa CENIT[1], con referencia CENIT-2007 2004 y bajo el cual la empresa Safelayer ha subcontratado al grupo DMAG de la UPC.

El grupo DMAG (Distributed Multimedia Applications Group) dentro del Departamento de Arquitectura de Computadores de la UPC, son expertos en la distribución, gestión y distribución segura y estándar de contenido multimedia, incluyendo el uso e interoperabilidad de metadatos para facilitar la organización y búsqueda de la información asociada al contenido, así como todos los aspectos relacionados con la seguridad y gestión digital de los derechos.

Safelayer Secure Communications, S.A. es una compañía destacada en el mercado de seguridad y confianza para las TIC. Dispone de múltiples proyectos en el ámbito de la tecnología para la autenticación, autorización, integridad y confidencialidad de la identidad del ciudadano.

Entre los productos de que dispone la empresa se encuentra *TrustedX*. Éste constituye una plataforma basada en arquitectura orientada a servicios (*SOA*) que utiliza los servicios web como pasarela de acceso a sus funcionalidades. Para el desarrollo de dicha plataforma se utilizan básicamente tecnologías *J2EE* y *Open source*, así como un conjunto de librerías

[1] Las subvenciones objeto de las convocatorias CENIT (Consorcio Estratégico Nacional de Investigación Técnica) se dirigen a la financiación de grandes proyectos integrados de investigación industrial de carácter estratégico, gran dimensión y largo alcance científico-técnico.

propias de bajo nivel que disponen de las operaciones criptográficas necesarias y que son usadas en toda su gama de productos.

Este PFC contribuye al proyecto Segur@ dentro de la actividad titulada "Entorno tecnológico para la gestión de la identidad digital". Uno de los objetivos de SFLY previstos en esta actividad es la implementación de un portal que reúna varias aplicaciones relacionadas con la seguridad, la identidad y la confianza tomando como base las tecnologías semánticas y la integración de los servicios mediante un ESB.

El objetivo de las aplicaciones de Semantic Trust Web Portal es mejorar la seguridad mediante nuevas tecnologías y a su vez ofrecer servicios útiles para el usuario que ayuden a comprender la potencia y ventajas de la tecnología semántica.

Capítulo 2
Tecnologías

En este capitulo se muestra una visión general sobre las tecnologías utilizadas, tanto en los casos de uso planteados como en las múltiples pruebas realizadas. En concreto se hablará sobre los diferentes Enterprise Service Bus que se han probado, tanto los open source como los comerciales, exponiendo sus características más significativas desde un punto de vista objetivo.

Debido a que en los ESB se pueden integrar un gran numero de aplicaciones, en un segundo tramo del capitulo se tratarán las más importantes. No se llegará a profundizar en ninguna de ellas, sólo se mostrara su definición y los productos relacionados con la misma, en el caso que los hay.

En este capitulo se describen varios modelos de ESB, tanto a partir de la información publicada por los fabricantes respectivos, como de la experiencia practica obtenida a partir de las pruebas con los distintos ESB.

2.1 Enterprise Service Bus

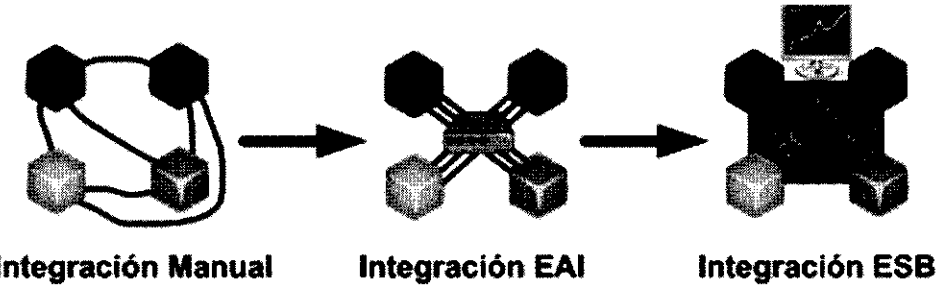


Figura 6 – Comparativa de Integración

Es posible encontrar muchas definiciones de ESB casi siempre ligadas al ámbito donde se le asignan o al punto de vista de los autores. Por esta razón se han extraído las características más importantes que reflejaban el cúmulo de definiciones que se han encontrado. Las peculiaridades que pueda tener un producto en particular normalmente no se repetirán en los demás. Las siguientes características suelen reflejarse en todos los software que se denominan ESB:

- **Plataforma de Integración basada en estándares abiertos:** En los tiempos actuales el ESB se ha convertido en la forma de integrar la multitud de aplicaciones a nivel corporativo. Es el siguiente peldaño de la generación de herramientas EAI que encontramos en el mercado. Ofrece una integración fácil, para ello soporta los protocolos más utilizados y generalmente contiene un sistema de plugins que hace más fácil integrar la aplicación. Esta es su mayor ventaja frente a otras tecnologías similares, y a día de hoy cumple su cometido.
- **Combina los paradigmas SOA y EDA:** Como bien dicen las definiciones, el ESB combina la potencia de la arquitectura orientada a servicios con la versatilidad de la arquitectura dirigida por eventos. Es posible referirse a esta unión como una arquitectura de integración de servicios dirigida por una capa de eventos.
- **Centrado en la naturaleza síncrona de los servicios y asíncrona de los eventos:** El ESB provee una capa de abstracción de bajo nivel que facilita entre otras cosas la asincronía de la que carecen las aplicaciones o servicios. Esto comporta un gran poder a la hora de pensar en el esquema de nuestros servicios.

Sólo se tendrá que esperar la petición y indicarle al ESB donde nos tiene que mandar la respuesta cuando el proceso acabe.

- **Implementa interfaces estandarizadas para proveer comunicación, conectividad, transformación, portabilidad y seguridad:** Además de los aspectos comentados anteriormente la tecnología ESB provee de un seguido de posibilidades que dan más fuerza a la arquitectura. Algunos de estos aspectos como la transformación son bastante importantes en todo el proceso del flujo de los datos.
- **Personalizables:** Cada empresa tiene servicios totalmente diferentes, y cada servicio tiene sus propias necesidades. La mayoría de ESB están enfocados a la personalización para adaptarse a cualquier cliente. Ya no sólo en la configuración de los procedimientos, sino en la posibilidad de aumentar a través de módulos parte de sus características, como por ejemplo los protocolos soportados.

En conclusión, con el ESB se construye un flujo de datos a través de una fácil integración de servicios y de una conectividad por eventos; el cual se puede administrar y guiar a través de una capa de abstracción, que normalmente viene personalizada según la compañía que lo ha desarrollado.

2.1.1 Protocolos

Durante todo el documento se verán diferentes protocolos de comunicación utilizados en todas las pruebas que se han hecho sobre los ESB. En esta sección se nombraran los más importantes para tener una idea de los que puede llegar a tener un ESB. De esta forma más tarde se podrá seguir la documentación sin ningún problema. En el caso de que se nombre alguno más se explicara en su momento.

2.1.1.1 FTP

El significado de las siglas es Protocolo de Transferencia de Archivos (File Transfer Protocol). Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo.

El Servicio FTP es ofrecido por la capa de Aplicación del modelo de capas de red TCP/IP al usuario, utilizando normalmente el puerto de red 20 y el 21. Un problema básico de FTP es que está pensado para ofrecer la máxima velocidad en la conexión, pero no la máxima seguridad, ya que todo el intercambio de información, desde el login y password del usuario en el servidor hasta la transferencia de cualquier archivo, se realiza en texto plano sin ningún tipo de cifrado, con lo que un posible atacante puede capturar este tráfico, acceder al servidor, o apropiarse de los archivos transferidos.

En el caso del ESB suele utilizarse como protocolo de entrada o salida de archivos. De esta forma los usuarios autorizados pueden dejar archivos (mensajes) que el ESB recogerá una vez detectados.

2.1.1.2 JMS

El significado de las siglas es Servicios de Mensajería de Java (Java Message Service). Permite que los componentes de aplicaciones hechos con JAVA, envíen, reciban, lean y escriban mensajes. Esto puede suceder tanto de manera síncrona como de manera

asíncrona. Este servicio de mensajería también es conocido como Middleware[2] Orientado a Mensajes (MOM).

Hay dos modelos de servicios de mensajería:

- **El modelo punto a punto:** En este caso un cliente envía un mensaje a otro cliente, si el receptor del mensaje no se encuentra activo el mensaje se encola hasta que lo este.
- **El modelo Publicador /Subscriptor**: En este otro caso tenemos un cliente que publica un tema o evento y (normalmente) más de uno que se subscriben a ese mensaje.

Tanto uno como otro pueden ser síncronos mediante el método **Receive** o asíncronos mediante el método **MessageListener.**

Administramos dos objetos diferentes en JMS:

- **ConnectionFactory:** Se usa para crear una conexión al proveedor del sistema de mensajes.
- **Destination:** Son los destinos de los mensajes que se envían y el receptor de los mensajes que se reciben.

[2] Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

Estos son algunos de los campos que podemos encontrar en el mensaje:

Campo	Tipo de Dato	Descripción
JMSMessageID	String	Un numero que identifica unívocamente al mensaje. Sólo se puede consultar una vez que esta enviado el mensaje.
JMSDestination	Destination	El destino a donde se envía el mensaje.
JMSDeliveryMode	int	Puede ser de tipo PERSISTENT, entonces se envía una única vez, o de tipo NON_PERSISTENT, de manera que se envía como mucho una vez, lo cual incluye también que no sea enviado nunca. PERSISTENT y NON_PERSISTENT están definidas como constantes.
JMSTimestamp	long	La hora a la que se envió el mensaje.
JMSExpiration	long	La hora hasta la cual el mensaje es valido, si es 0 quiere decir que no caduca nunca.
JMSPriority	int	Prioridad del mensaje de 0 a 9, siendo 0 la más baja.
JMSCorrelationID	String	Este campo se usa para relacionar una respuesta a un mensaje, se copia aquí el ID de mensaje del mensaje al que se esta respondiendo.
JMSReplyTo	Destination	Especifica el lugar a donde se deben enviar las respuestas al mensaje actual.
JMSType	String	Este campo lo puede usar el programa de mensajería para almacenar el tipo del mensaje.
JMSRedelivered	boolean	Indica que el mensaje ha sido enviado con anterioridad pero el destino no lo ha procesado, por lo que se reenvía.

Estos son algunas de las propiedades del mensaje:

Propiedad	Tipo de Dato	Descripción
JMSXUserID	String	El usuario que envía el mensaje.
JMSXApplID	String	La aplicación que envía el mensaje.
JMSXDeliveryCount	int	El numero de veces que se ha intentado enviar el mensaje
JMSXGroupID	String	identificador del grupo al que pertenece el mensaje.
JMSXGroupSeq	int	Numero de secuencia en el grupo de mensajes.
JMSXRcvTimestamp	long	La hora a la que JMS le entrego el mensaje al/los destinatario/s.
JMSXState	int	Para uso del proveedor de mensajería.
JMSX_<nombre_del_proveedor>	-	Reservado para propiedades particulares del proveedor.

Hay muchos productos que implementan este protocolo, durante las pruebas se uso un producto Open Source (ActiveMQ). JMS se puede usar como mediador entre servicios, como por ejemplo hacer el servicio asíncrono en el caso de que el propio ESB no te de una herramienta para ello. Hay algunos ESB que utilizan este protocolo internamente para implementar la asincronía.

2.1.1.3 SMTP

El significado de las siglas es Protocolo Simple de Transferencia de Correo (**S**imple **M**ail **T**ransfer **P**rotocol). Se basa en el modelo cliente-servidor, donde un cliente envía un mensaje a uno o varios receptores. La comunicación entre el cliente y el servidor consiste enteramente en líneas de texto compuestas por caracteres ASCII. El tamaño máximo permitido para estas líneas es de 1000 caracteres, aunque puede ampliarse usando las extensiones del servicio SMTP.

Las respuestas del servidor constan de un código numérico de tres dígitos, seguido de un texto explicativo. El número va dirigido a un procesado automático de la respuesta por autómata, mientras que el texto permite que un humano interprete la respuesta. En el protocolo SMTP todas las órdenes, réplicas o datos son líneas de texto, delimitadas por el carácter <CRLF>. Todas las réplicas tienen un código numérico al comienzo de la línea.

En el conjunto de protocolos TCP/IP, el SMTP va por encima del TCP, usando normalmente el puerto 25 en el servidor para establecer la conexión.

En el ESB podría ser utilizado por el administrador como una fuente de reports. Cada vez que ocurra un error categorizado muy grave en un flujo, el propio ESB le enviaría un mail de aviso.

2.1.1.4 HTTP

El significado de las siglas es Protocolo de Transferencia de Hipertexto (**H**yper**T**ext **T**ransfer **P**rotocol). Es el protocolo usado en cada transacción de la Web. HTTP fue desarrollado por el consorcio W3C y la IETF, colaboración que culminó en 1999 con la publicación de una serie de RFC, siendo el más importante de ellos el RFC 2616, que especifica la versión 1.1.

HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador o un spider) se lo conoce como "user agent" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un URL. Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Dentro de este protocolo también hace falta hacer mención de la conexión HTTPS, que se base en el http añadiéndole seguridad, para ello utiliza un cifrado basado en el SSL para crear un canal seguro.

En el ESB es muy habitual encontrar este tipo de conexiones. Actualmente la mayoría de servicios en las empresas se ofrecen a través de Web Service tanto por HTTP como por HTTPS. Por lo tanto se suele usar generalmente como punto de destino.

2.1.1.5 SSL

SSL proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. Habitualmente, sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar; la autenticación mutua requiere un despliegue de infraestructura de claves públicas (o PKI) para los clientes. Los protocolos permiten a las aplicaciones cliente-servidor comunicarse de una forma diseñada para prevenir escuchas, la falsificación de la identidad del remitente y mantener la integridad del mensaje.

Como ejemplos más claros de protocolos que usan SSL están el HTTPS o el SFTP. Es interesante que los ESB también tengan acceso por protocolos que utilicen SSL porque actualmente en las empresas casi todo se accede por seguridad.

2.1.1.6 Transporte FILE

En este caso se habla de transporte porque no es un protocolo mismamente dicho. Se esta hablando de la transferencia de archivos en un entorno local. Los ESB suelen utilizar este servicio para ir montando una estructura interna de directorios para los diferentes servicios que no tienen que ser visibles por los demás usuarios .

2.1.2 Modelos Open Source

En este apartado se van a nombrar parte de los ESB que más se han probado durante los meses de investigación. No se han hecho las mismas pruebas sobre todos, pero si se han hecho algunas pruebas comunes. Más adelante se ara referencia a esta información para completar las explicaciones.

2.1.2.1 Mule

MULE [4] ofrece una aplicación neutral que se puede integrar con las aplicaciones más convenientes, por esta razón es mucho más ligero que la mayoría de los que existen en el mercado. Estos están montados sobre aplicaciones de la misma compañía, como por ejemplos los comerciales.

Especialmente este software destaca por tener muchos proyectos a su alrededor, casi todos dedicados al desarrollo de componentes para darle más funcionalidades, entre ellos contiene una herramienta administrativa. Esta es muy útil porque la versión estándar del software no contiene consola.

Otras de las propiedades que destacan son:

- **Basado en framework de mensajería JAVA para la conexión e intercambio de datos**
- **Arquitectura orientada a servicios**
- **Integración de una gran cantidad de protocolos.**
- **Basado en las ideas de la arquitectura del Enterprise Service Bus.**
- **Facilidad de añadir nuevo componentes**
- **Posibilidad de cualquier formato en los mensajes: SOAP, image, etc...**
- **Reusabilidad de componentes**
- **Mantienen una versión Enterprise (comercial)**

La primera release que se ha encontrado data el *Abril del 2005* y la última data en *Diciembre del 2008.* El equipo que se dedica al desarrollo de MULE es de Estados Unidos, Malta y de Reino Unido. No se ha encontrado ninguna relación directa con empresas comerciales del sector, pero se conoce que hay potentes inversoras detrás.

En MULE ven el ESB así:

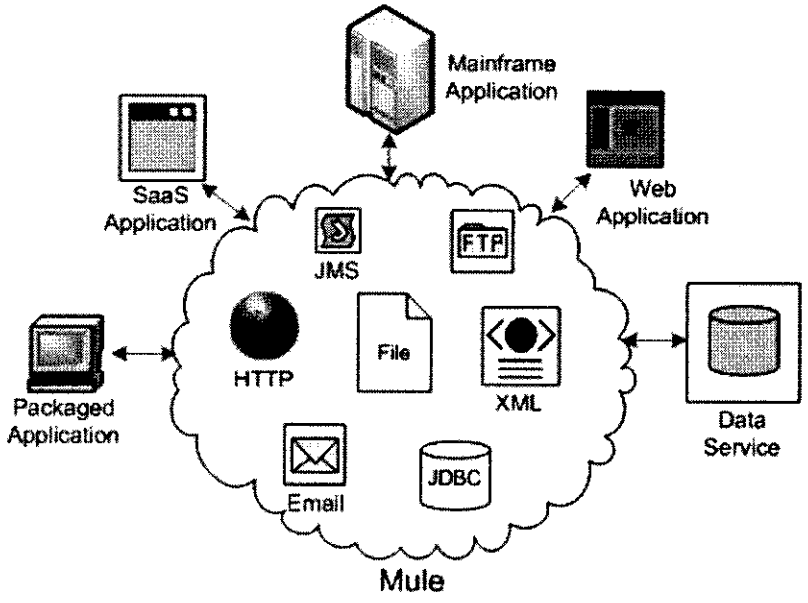


Figura 7 – ESB según MULE

2.1.2.2 ServiceMix

ServiceMix [5] es uno de los dos ESB de código libre que corren bajo la licencia Apache. Existen muchas empresas que tienen sus productos Open Source enmascarados bajo esta licencia, hay muchos grupos de desarrolladores que se han ido a desarrollar sus proyectos bajo Apache.

Lo definen como un ESB ágil, por combinar SOA con EDA. Como se ha visto antes éste tendría que ser un requisito básico, en cambio tanto éste como muchos otros lo ponen como una propiedad especial llamada ágil.

Utiliza el gestor de colas Active MQ, otro software de Apache, un gestor de colas. Con el distribuyen los errores, ofrecen fiabilidad y control remoto. Como en el caso de Mule hacen hincapié en que es una arquitectura sin ataduras y se puede montar sobre diferentes arquitecturas mayores.

La primera release que se ha encontrado data del Agosto del 2005 y la última data en Diciembre del 2008. El equipo que se dedica al desarrollo de ServiceMiX destaca la inclusión de gente que esta en proyectos de IBM e IONA. Aun y ser un proyecto Open Source de la comunidad Apache se ve como las empresas que hay detrás ya no sólo financian el proyecto sino que incluyen su personal para nutrirse del conocimiento adquirido.

Otras de las propiedades que destacan son:

- **Ha integrado SPRING**
- **Soporta EJB y POJO**
- **Se puede personalizar y así acomodarlo a tus necesidades**

En ServiceMiX ven el ESB así:

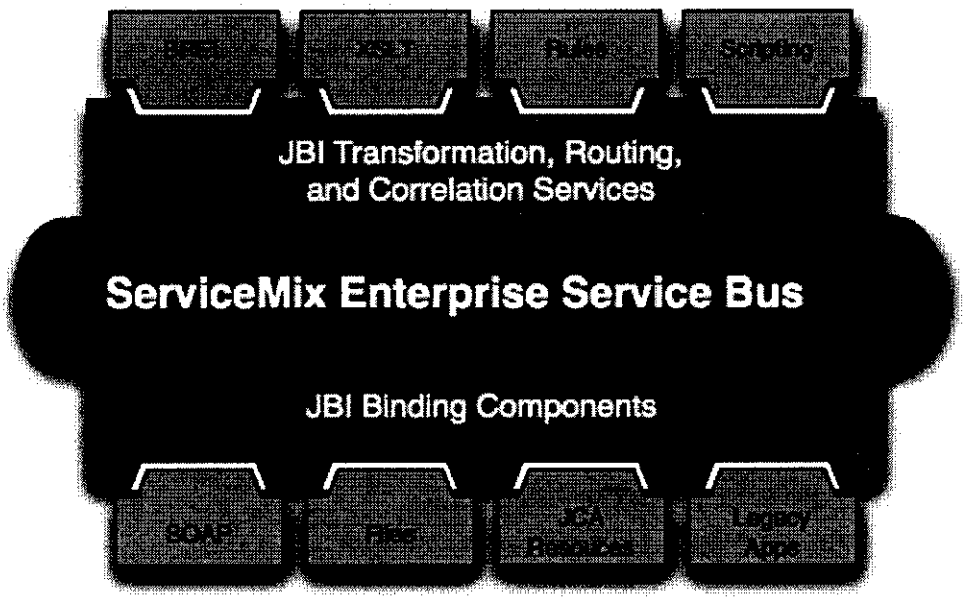


Figura 8 – ESB según ServiceMiX

2.1.2.3 Apache Synapse

Synapse [6] es un ESB simple que tiene un soporte excelente para los Web Service, XML, binario y texto. Además soporta protocolos como : http, soap, jms, ftp, file, system transport, wss, wsrm o MTOM/XOP.

Algunas otras propiedades destacables son:

- **Esta basado en un core asíncrono**
- **Es muy ligero**
- **Es uno de los más nuevos**
- **Todo el flujo se configura a través de un XML**
- **Contiene módulos ya instalados**
- **Posibilidad de creación de nuevos módulos**

La primera release que se ha encontrado data del Julio de 2007 y la última data en *Junio de 2008.* El equipo que se dedica al desarrollo de Synapse destaca la inclusión de gente que esta en el proyecto WSO2-ESB de Sri Lanka. Es otro de los proyectos Open Source de Apache que esta unido a uno externo, también Open Source en este caso, pero con una licencia diferente.

2.1.2.4 WSO2 ESB

WSO2 [7] tiene muchos productos y entre ellos destaca el ESB, el cual esta enteramente basado en synapse. Su principal característica diferencial es que WSO2 aporta una interfaz grafica y algún complemento a la core de synapse. La interfaz grafica substituye a el archivo XML de configuración dándole mucha más simplicidad y más facilidad de comprensión a la hora de ver la configuración.

En estos momentos están montando una plataforma de integración con muchos de sus productos, entre ellos el ESB. Con esta esperan competir contra las grandes empresas, ya que intentan que no sólo utilices uno de sus productos, sino que implantes todos.

Otras de las propiedades que destacan son:

- **Ultrarrápido y ligero (tiene synapse como core)**
- **Se basa en Axis.**
- **Muchas pruebas de interoperabilidad pasadas (esta basado en dos productos que las pasan)**
- **Puede conectar, administrar, trasformar e interaccionar entre servicios.**
- **Tiene bastante elementos útiles ya predefinidos**
- **Posibilidad de creación de nuevos elementos con lenguajes como: POJO, SPRING, JAVASCRIPT, RUBY, GROOVY, APACHE BSF**
- **Soporta un gran número de conexiones simultáneamente**
- **Soporta protocolos como: HTTP, HTTPS, JMS; VFS, POP3, IMAP, SMTP, AMQP, FIX, HASSIAN BINARY.**
- **Contiene herramientas para el soporta para Web Service: XML, NamesPaces, Xpath, XSLT, XQuery**
- **Se pueden definir tareas programadas**
- **Monitoriza toda lo que ocurre de una manera simple**

Cabe destacar también su consola de administración, la cual es de las primeras tan completas y más siendo un software Open Source. Resuelve la complejidad que podía ser tener que configurar un XML sin conocer a fondo el formato de la configuración. En el siguiente capitulo se hablara más profundamente del tema.

La primera release que se ha encontrado data de Junio de 2007 y la última data en *Diciembre de 2008.* El equipo que se dedica al desarrollo de WSO2 ESB destaca por ser enteramente de Sri Lanka. Éste es un proyecto basado en Synapse, por esa razón se encuentran muchos desarrolladores de wso2 en ese proyecto. Aun y estar en dos proyectos Open Source se conoce que hay empresas fuertes detrás.

En WSO2 ven el ESB así:

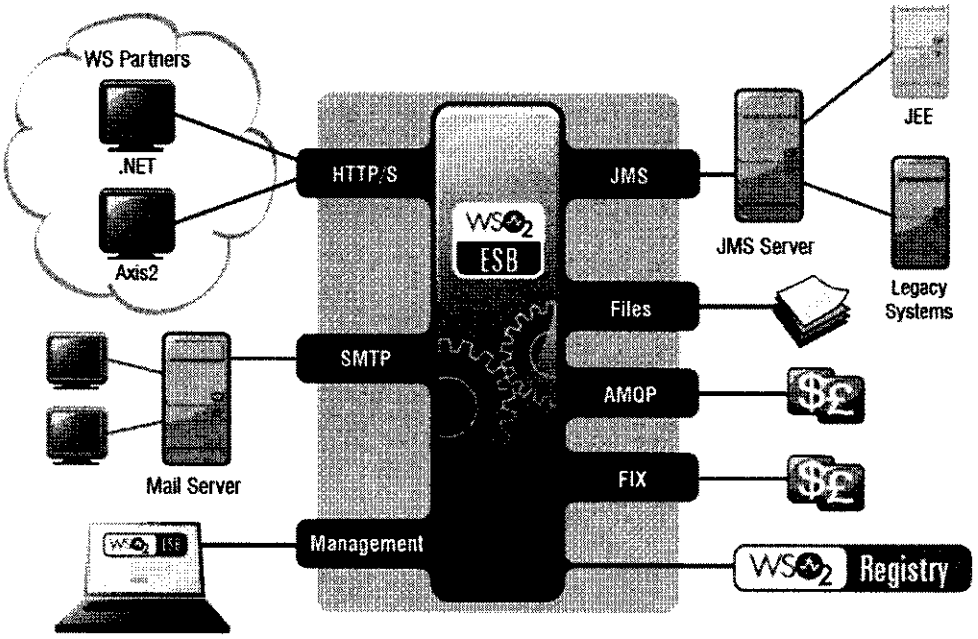


Figura 9 – ESB según WSO2

2.1.2.5 Petals Service Plataform

Igual que WSO2ESB, PETALS [8] pertenece a una empresa que tiene más de un producto relacionado con el tema. Viendo todo en conjunto se esta empezando a vislumbrar parte de una plataforma que la formarían todos los productos de la empresa.

Este es un ESB Open Source pensado para integrar grandes arquitecturas y esta preparado para trabajar en un ambiente distribuido. A su vez, como la mayoría de ESB, es fácil adaptarlo a cambios en su configuración.

Tiene capacidad para soportar muchos componentes, igualmente que los anteriores tiene la capacidad para añadir nuevos creados por el usuario. Para probarlo viene con un grupo de casos de uso interesantes.

Otras de las propiedades que destacan son:

- **Tiene capacidad para procesar millones de mensajes al día.**
- **Administración y monitorización remota**
- **Soporta múltiples protocolos como: JMS, FTP, HTTP**
- **Los mensajes internamente se envían en XML**

- **Es compatible con JBI y Web Service**
- **Es robusto, seguro y balanceado**
- **Es totalmente modular**
- **Tiene un mantenimiento reducido**

La primera release que se ha encontrado data en Septiembre del 2006 y la última data en *Octubre del 2008.* El equipo que se dedica al desarrollo de PETALS destaca por ser enteramente Frances, es uno de los únicos ESB con desarrolladores europeos. Detrás se encuentran empresas como CAPGEMINI y EBM WebSOURCING.

En PETALS ven el ESB así:

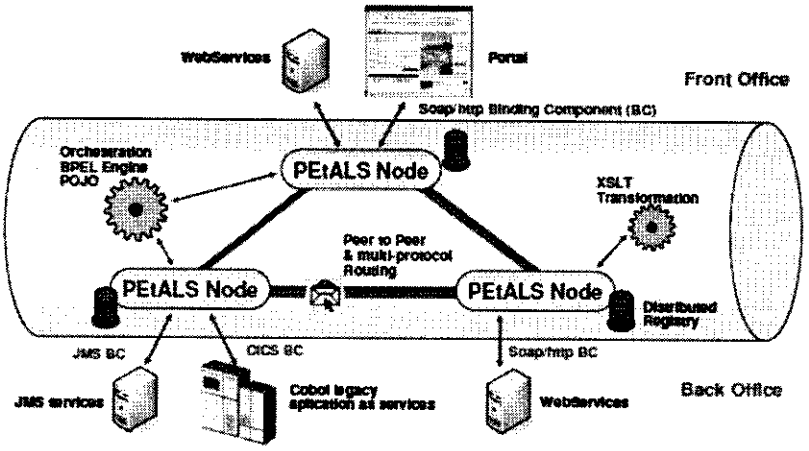


Figura 10 – ESB según PETALS

2.1.2.6 Kuali

Kuali [9] es un ESB orientado a servicios como la mayoría de los que se han visto. Hay varias formas de activar los servicios, por cola o por topic. En el primer caso, cuando llega un mensaje a la cola éste es enviado al servicio. En cada cola sólo puede estar escuchando un servicio. En el segundo caso, se utiliza la publicación, las aplicaciones están suscritas a un topic y así saben cuando les han de enviado algo.

Ofrece una posibilidad de configuración para mejorar la calidad del servicio. Así se pueden configurar tiempos de espera, números de intentos, tiempo entre intentos, etc... Incluso incluye colas de error para poder repetir los trabajos que no se hayan podido llevar a cabo.

Otras propiedades destacables son:

- **Apagado y encendido remotamente**
- **Integrado con SPRING (Remote Asyncronous Calls)**
- **Posibilidad de integración de servicios programando**
- **Capacidad de administración de mensajes**
- **Servicios descubiertos automáticamente y borrados cuando causan error**

La primera release que se ha encontrado data en Septiembre del 2007 y la última data en *Agosto del 2008.* El equipo que se dedica al desarrollo de KUALI destaca por ser enteramente formado por miembros de universidades. No se ha tenido constancia de que detrás exista alguna gran empresa.

En KUALI ven el ESB así:

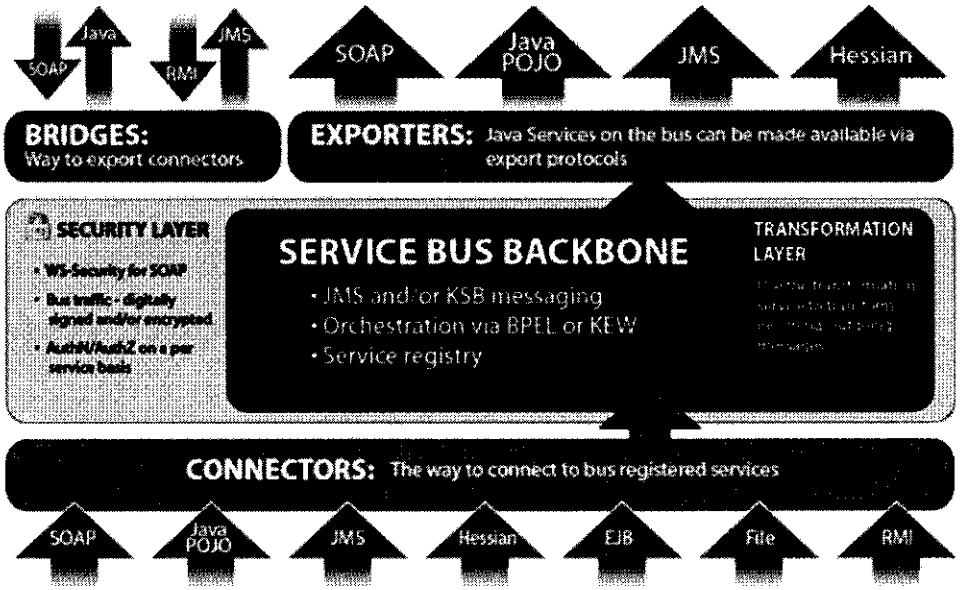


Figura 11 – ESB según KUALI

2.1.2.7 GlassFish ESB

GlassFish [10] a nacido de la unión de un ESB existente (OPEN ESB), el Aplication Server de SUN (GLASSFISH) y el entorno de desarrollo integrado de SUN (NETBEANS). Desde esta unión se ha separado una versión comercial de una versión de la comunidad, por lo cual sólo se da soporte sobre la versión comercial.

La meta de proporcionar una versión gratuita es para que la comunidad vaya testeando la versión y creando módulos nuevos. Aunque ya tiene muchos creados quieren proporcionar muchos más y aprovecharse de las ideas de la comunidad Open Source. Respecto a esto han dejado claro que sólo se dará soporte a los módulos hechos por SUN.

Algunas otras propiedades destacables son:

- **Ligero y ágil y pequeño**
- **Tiene un arquitectura Microkernel para conectar multitud de componentes**

La última release que se ha encontrado data en Noviembre del 2008. Este ESB destaca por tener un conjunto de partners, entre los cuales Bostech Corporation que tiene un ESB propio y un grupo de consultoras. Además tiene el apoyo de SUN con la reciente unión con su Aplication Server.

Por otra parte existe otro proyecto ligado a GlassFish ESB que es muy interesante, FUJI Milestone. Este será el nuevo interfaz del ESB, servirá para configurarlo más fácilmente. Aquí hay un ejemplo:

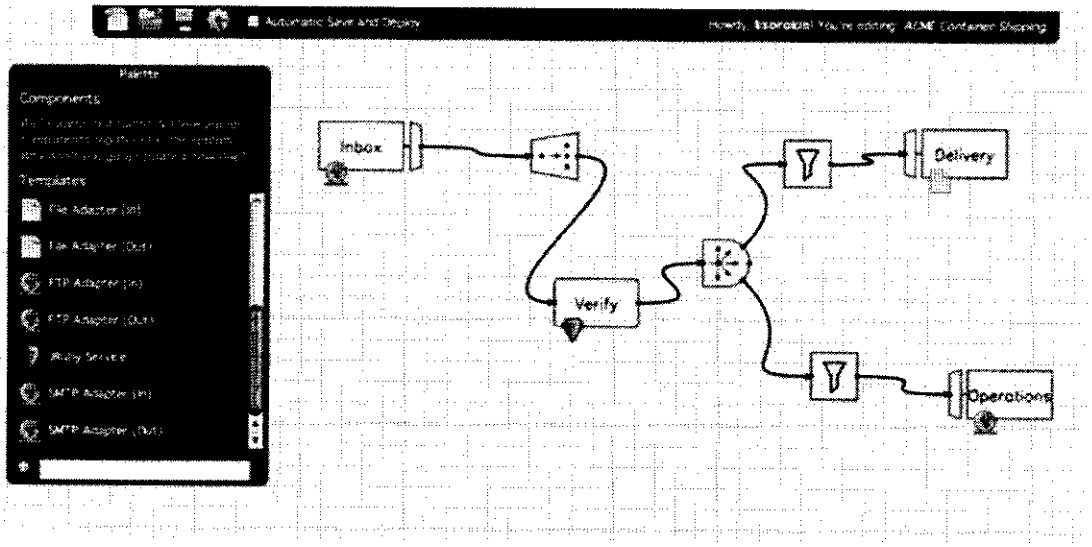


Figura 12 – Fuji Milestone (Futura consola grafica de GlassFish ESB)

Aquí se puede ver la última versión de esta interfaz que esta en fase beta. Da muy buenas sensaciones. Como proyecto paralelo es muy interesante, más adelante se comparar esta interfaz con alguna de la de los demás.

Este puede ser uno de los proyectos más prometedores y más nuevos en el mundo de los ESB, se tienen que esperar grandes cosas de un proyecto donde se ha metido SUN con mucha fuerza.

2.1.3 Modelos Comerciales

2.1.3.1 Oracle Weblogic ESB

Hace poco que ORACLE [11] compro BEA y se hizo suyos todos los productos. Oracle ya contaba con algunos productos propios, aunque casi todos los ha substituido o unido con los de BEA. El ESB que tenia Oracle era más bien simple en cambio el de BEA es muy completo. Hace un tiempo que aun se podía instalar el ESB de BEA, ahora ya esta a nombre de ORACLE y con la única peculiaridad del cambio de logos y de nombre.

A este ESB se le podrían atribuir muchas propiedades que ya se analizarán en el siguiente capitulo, las más destacable son:

- **Rápida configuración con reglas básicas sin necesidad de código.**
- **Transformaciones básicas basadas en XQuery o XSLT, aplicadas dinámicamente**
- **Publicación de servicios automatizada y dirigida por la administración**
- **Soporte de múltiples protocolos y standards**
- **Posibilidad de crear transportes propios**
- **Configuración totalmente configurable según parámetros del mensaje**

Por otra parte cabe destacar la consola de administración. Controla todos los cambios incluyendo sesiones para saber en todo momento quien esta modificando las configuraciones. Se pueden parar y encender servicios de manera totalmente dinámica. Incluso se pueden testear configuraciones desde el punto que quieras.

En ORACLE WEBLOGIC ESB ven el ESB así:

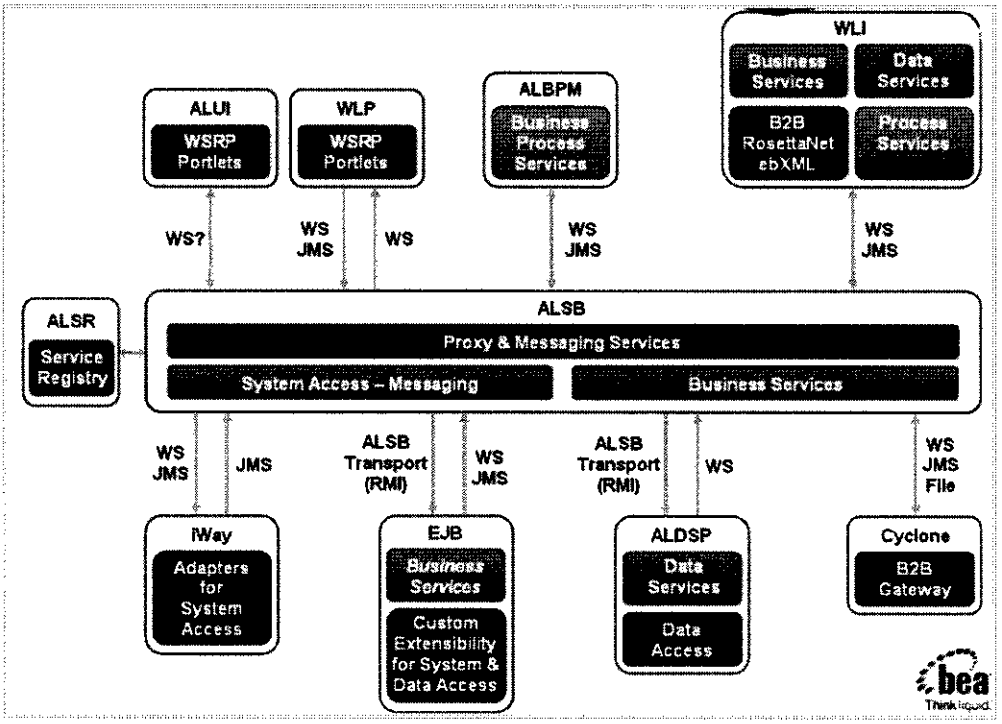


Figura 13 – ESB según ORACLE WEBLOGIC ESB

2.1.3.2 IBM WebSphere Message Broker

IBM [12] es una empresa líder en consultoría, y como tal parece que tiene cierto interés en que sus aplicaciones no sean completamente abiertas. La autentica compatibilidad del ESB es con las aplicaciones de WebSphere de la propia IBM.

Este proporciona un enfoque inteligente a la arquitectura orientada a servicios, ofreciendo conectividad basada en estándares y una solución de integración que le permite crear y desplegar rápida y fácilmente interacciones entre aplicaciones y servicios, con un número reducido de interfaces menos complicadas.

Ofrece herramientas fáciles de usar que requieren habilidades de programación mínimas y es fácil de instalar, configurar, crear y gestionar. Además da soporte a muchas soluciones ISV a través de WebSphere Adapters.

Ofrece liderazgo en estándares de arquitectura orientada a servicios para composición, mediación y alojamiento de servicios. Aumenta la agilidad y la flexibilidad empresariales pudiéndose ampliar fácilmente a un modelo ESB federado.

Se puede volver a configurar de forma dinámica para que satisfaga las cargas de procesos empresariales en constante evolución. Proporciona fáciles interacciones con cualquier aplicación JMS y HTTP.

Se integra a la perfección con la plataforma WebSphere, así como con productos de la arquitectura orientada a servicios de IBM. Su base, por ejemplo, IBM Tivoli Composite Application Manager for SOA.

Los sistemas operativos admitidos: AIX, HP Unix, familia i, Linux, Sun Solaris, Windows, z/OS.

En IBM ven el ESB así:

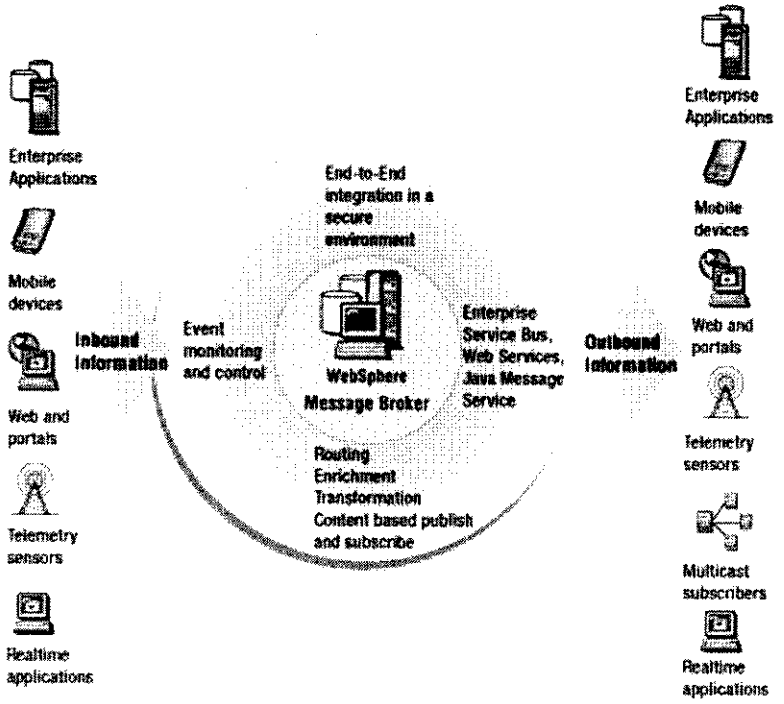


Figura 14 – ESB según IBM

2.2 Tecnologías semánticas

2.2.1 RDF

La propuesta de este modelo de estructuración de la información surgió en un primer borrador del 1997 organizado por el W3C, donde participaron diversos contribuidores de distintas organizaciones y empresas importantes en el sector informático, como por ejemplo IBM, Microsoft, Netscape, Nokia y Reuters entre otras.

Se trata de un modelo derivado de un subconjunto de la lógica de primer orden, donde se han añadido ideas sobre modelos conceptuales y restricciones como por ejemplo:

- **Identificadores de los elementos del dominio.**
- **Literales para representar todo tipo de constantes.**
- **Conceptos vistos como predicados unarios del estilo "país(Alemania)".**
- **Propiedades vistas como predicados binarios del estilo "capital(Alemania, Berlín)".**

El modelo se basa en la representación de la información en forma de grafo, lo que se denomina "RDF Graph" pudiendo estar codificado de dos maneras diferentes, por medio de notación N3 (Notation 3), o bien "RDF/XML" con su correspondiente representación en XML (eXtensible Markup Language).

Este grafo está formado por expresiones de tres elementos: sujeto, predicado y objeto. Los sujetos y objetos son los recursos que se desean describir y éstos se representan por medio de una URI (Uniform Resource Identifier). Hay que destacar, que los objetos pueden ser valores literales.

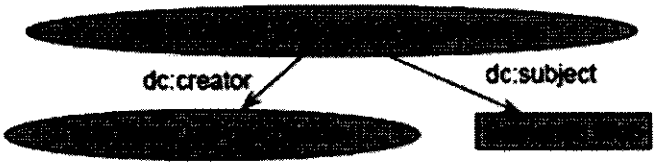


Figura 15 – Grafo RDF

En la figura anterior se puede apreciar, que un recurso "seminario" con URI "http://dbis.uni-goe.de/teaching/seminar" que hace de sujeto, tiene dos propiedades. La primera propiedad "dc:creator" expresa que el creador del recurso es otro recurso identificado por la URI "http://user.uni-goe.de/may" y la segunda propiedad "dc:subject" expresa que el contenido del seminario es "Semantic Web" que se trata de un valor literal.

La información expresada en forma de tripletas es independiente de su localización, es decir, la base de conocimiento contendrá un conjunto de tripletas cada una con valor atómico, donde el orden entre tripletas es irrelevante. Gracias a este factor es posible disponer de diferentes ficheros RDF (bases de conocimientos) en diferentes lugares, siempre que usen las mismas URIs para los recursos que describen, y los mismos nombres de propiedades para que puedan ser combinados fácilmente. Dichos nombres están definidos en ontologías, que describen los conocimientos relativos a un dominio de conocimiento. Es decir, si queremos utilizar el vocabulario de FOAF (Friend Of A Friend), que es un vocabulario pensado para describir usuarios, usaremos los nombres que estén definidos en su ontología.

Un recurso se puede describir por medio de una URL (Uniform Resource Locator) que es un identificador más concreto que una URI, es decir, una URL referencia un recurso que, en general, es accesible a través de la Web, en cambio una URI identifica cualquier tipo de recurso.

Una URI puede ser un localizador (URL), un nombre (URN) o una descripción genérica (URI) y se define como una secuencia de caracteres, que expresa una estructura jerárquica y que se basa en unos esquemas, que pueden estar registrados por una autoridad como IANA (Internet Assigned Numbers Authority). Por lo tanto estas URIs, pueden servir como identificadores únicos para describir recursos de una comunidad que lo haya así acordado previamente. Normalmente en RDF se utilizan URIs y no solamente URLs.

2.2.2 JENA

Jena es una herramienta open source, que engloba un conjunto de elementos para poder trabajar con ontologías y estándares de la Web semántica. Básicamente, destaca el razonador que implementa OWL-Lite, y un razonador de reglas genérico que permite razonar con reglas definidas por el usuario.

Hay que destacar que el razonador que implementa sobre OWL-Lite, no es técnicamente completo, ya que no permite razonar con OWL-DL. Para este tipo de razonamientos, Jena recomienda utilizar otros razonadores externos a su framework, entre los cuales figuran Pellet, Racer o FaCT++. Dicha compatibilidad se realiza de una manera sencilla, ya que Jena dispone de una interfaz DIG que conecta los razonadores de una manera estandarizada.

Debido a que Jena implementa parcialmente estos razonadores, aparecen tres tipos de configuraciones relacionadas con el nivel de expresividad con el que se desea trabajar.

Para empezar, existe la configuración Full, que por defecto permite trabajar con un amplio conjunto de construcciones de OWL. La configuración Mini no permite trabajar con restricciones de cardinalidad. Finalmente, la configuración Micro permite trabajar sobre un conjunto muy limitado de construcciones, pero garantiza una máxima rapidez para comprobaciones básicas sobre ontologías.

La alternativa al razonador OWL es el razonador de propósito general basado en reglas definidas por el usuario. Dichas reglas no están expresadas en ningún lenguaje estandarizado, como podría ser SWRL, sino que se trata de objetos propios implementados en Java, que representan reglas.

2.2.3 SPARQL

Para poder desarrollar una infraestructura capaz de explotar la información de la base de conocimientos en tripletas, es necesario incorporar un lenguaje de consultas. SPARQL es el lenguaje diseñado para consultar la información en tecnologías de Web Semántica. Se basa en consultas conjuntivas sobre las tripletas, es decir, que para devolver un resultado se deben cumplir todas y cada una de las condiciones indicadas en la petición. Además ofrece una sintaxis muy parecida al estilo de SQL, cosa que facilita su uso por tratarse de un lenguaje muy extendido.

```
SELECT result variables
FROM input
WHERE { dot-separated sequence of triple-patterns and FILTER expression }
```

Figura 16 – Estructura de una query SPARQL

La manera de expresar las condiciones en forma de tripletas, es la característica que más llama la atención, ya que hay que adaptar las condiciones a las estructuras de sujeto-predicado-objeto. Para indicar que es una conjunción de condiciones o patrones se utiliza el punto.

```
prefix : <foo://bla/names#>
select ?P ?A
from <file:john.n3>
where {?X :name ?P . ?X :age ?A . FILTER ( ?A > 30 ) }
```

Figura 17 – Ejemplo de consulta en SPARQL

Entre las diversas opciones que nos ofrece este lenguaje para realizar consultas sobre tripletas, hay que destacar las del tipo SELECT, como la de la figura anterior, que ofrece resultados en función de las variables que hayamos declarado inicialmente y de las condiciones que se satisfacen. Según la guía de referencia del lenguaje estándar publicada por el W3C, el resultado está expresado en XML y representa una tabla de registros, donde cada fila o registro hace referencia a un nodo de tripleta con sujeto, predicado y objeto.

La consulta anterior tiene como base de conocimiento el fichero "john.n3", que contiene tripletas sobre nombres de personas y sus edades. El resultado que se obtiene al ejecutar la consulta es un conjunto tripletas, donde se relaciona cada nombre de una persona, con su edad correspondiente, siempre y cuando tengan una edad superior a los treinta años.

La mayoría de software que integra consultas en SPARQL procesa de alguna que otra forma este resultado en XML, ya sea aplicando hojas de estilo para presentar el resultado

en formato HTML, más atractivo para el usuario, o bien en sistemas más avanzados, utilizando determinadas librerías Java. Por ejemplo, con Jena, se pueden manipular directamente los objetos que contienen los registros con los resultados.

Otra de las grandes opciones que ofrece SPARQL, son las consultas del tipo CONSTRUCT, que permiten realizar consultas, indicando condiciones en la forma de las tripletas, pero esta vez, el resultado en vez de ser una tabla con registros, son instancias en formato RDF con la representación de aquellas tripletas que han sido seleccionadas por el motor de resolución de consultas. Así se consigue que el resultado se pueda guardar directamente como un fichero RDF.

2.2.3.1 DB de tripletas

Tradicionalmente el conocimiento se ha podido almacenar en bases de datos en forma de tablas donde la relación era de sujeto – propiedad. Ha llegado el momento que los RDF están pasando a ser de gran importancia, y aunque la forma más usada para serializarlos es en archivos XML, cuando tienes una gran cantidad de información puede llegar a ser muy complicado. Por esa razón han empezado a surgir diferentes productos para almacenar RDF en bases de datos.

Durante estos años han aparecido varios estudios comparativos ([1], [2]) sobre los diferentes productos. El producto que hemos escogido, el SDB de Jena, en el ***primer estudio*** no lo nombran porque no había salido, y en el ***segundo estudio***[2] sólo se probo la fase beta.

En el caso del recientemente aparecido SDB de Jena, por ejemplo, permite utilizar bases de datos basadas en Oracle, Ms SQLServer, DB2, PostgreSQL, MySQL, Apache Derby, H2 y HSQLDB. Cada almacén RDF se almacena en una base de datos independiente. Cada una de estas bases de datos contiene un conjunto de tablas orientadas a almacenar la información de las tripletas: Triples para las tripletas, Quads para los grafos RDF y Nodes para los nodos de las tripletas y grafos. SDB ofrece también diferentes layouts o definiciones de campos para estas tablas, cada uno con sus ventajas e inconvenientes. Una vez que la información está almacenada en el almacén es posible acceder a ella a través del API de Jena o mediante la ejecución de consultas SPARQL.

En muchos casos puede ser también necesario disponer de ciertas capacidades de inferencia. SDB por ejemplo es únicamente un almacén de tripletas y no dispone de capacidades de inferencia incluidas, por lo que ante cualquier consulta SPARQL por ejemplo, únicamente se responderá con el conocimiento explícito de las tripletas almacenadas. Imaginemos por ejemplo que tenemos una ontología para representar las publicaciones de una biblioteca. En este caso podríamos tener una clase base "Publicación" y varias subclases como "Libro", "Revista", "Artículo", etc. Imaginemos ahora que alimentamos el almacén RDF con tripletas que representen que "El Quijote" es un Libro. Si hiciéramos una consulta solicitando los libros existentes aparecería, pero sin embargo si pidiésemos todas las publicaciones no lo obtendríamos como resultado, pese a ser Libro una subclase de Publicación. Para ello tendríamos que haber definido explícitamente que "El Quijote" es también una Publicación. Esto sucede así en SDB porque no utiliza el modelo de la ontología para realizar inferencias, ni a la hora de cargar las tripletas en el almacén ni tampoco en el momento de la consulta.

2.3 Web Service

Un servicio web es una colección de protocolos y estándares que se utilizan para el intercambio de datos entre distintas aplicaciones, que pueden estar desarrolladas en distintos lenguajes de programación y ejecutarse en distintas plataformas, estableciendo un lenguaje de comunicación común. La interoperabilidad se consigue mediante el uso de

estándares abiertos como *XML* y las organizaciones al cargo de su arquitectura y reglamentación son OASIS y W3C.

La organización W3C ofrece la siguiente definición de servicio Web:

> *"Un servicio web es un sistema software diseñado para soportar interacciones entre máquinas de manera interoperable en la red. Dispone de una interfície descrita en un formato procesable de manera automática (específicamente **WSDL**). Los sistemas interaccionan con el servicio web de la manera indicada por su descripción, usando mensajes **SOAP**, típicamente sobre un soporte de **HTTP** con serialización en formato **XML**, y siguiendo otros estándares relacionados con la web. "*

2.3.1 Arquitectura orientada a servicios (SOA)

La arquitectura orientada a servicios es un **patrón de diseño** arquitectónico de amplio uso actualmente basado en la utilización de **servicios web** para dar soporte a los requerimientos software de un usuario o aplicación. Es una arquitectura altamente flexible, desde un entorno *SOA* se puede acceder a múltiples servicios web, **independientes** entre sí, sin necesidad de tener conocimientos sobre la plataforma sobre la que se han implementado, de modo que dota al sistema de una mayor interoperabilidad.

Un entorno *SOA* **unifica** los procesos de negocio estructurando grandes aplicaciones como colecciones de módulos más pequeños, es decir, en servicios web. Los servicios se comunican entre sí intercambiándose datos o coordinando una actividad entre uno o más servicios. Estas aplicaciones se pueden usar por distintos grupos de gente, tanto internos como externos a la compañía, y también se pueden desarrollar nuevas aplicaciones a partir del conjunto de servicios descritos dando una mayor flexibilidad y uniformidad. De este modo se evita tener que introducir los mismos datos para acceder a distintos servicios, pudiendo acceder a ellos desde interfaces muy similares.

A la hora de construir un sistema *SOA* eficiente, se deben tener en cuenta los siguientes **requerimientos**:

- **Interoperabilidad entre distintos sistemas y lenguajes de programación:** La base más importante para realizar una integración sencilla entre aplicaciones de distintas plataformas es decidir el protocolo de comunicación que se usará, éste debe ser accesible por la mayoría de plataformas y lenguajes de programación.
- **Descripción del lenguaje clara y no ambigua:** Para usar uno de los servicios ofrecidos por un proveedor, no sólo es necesario ser capaz de acceder el sistema del proveedor, sino que se debe tener un conocimiento claro sobre la sintaxis a utilizar.
- **Búsqueda de servicios:** Para favorecer una buena integración, en la etapa de diseño o en tiempo de ejecución, es necesario un mecanismo de búsqueda, que sea capaz de encontrar los servicios más adecuados. Los servicios deben clasificarse, a partir de su ámbito, jerarquía o taxonomías en base a qué hacen los servicios de las distintas categorías y como pueden ser invocados.

La implementación de este tipo de arquitecturas se basa en el conjunto de estándares que se definen en servicios web.

2.3.2 JAX-WS

En concreto se hablara sobre JAX-WS, que forma parte del proyecto glassfish y agrupa entre otras a SOAP 1.2, WSDL 1.1, WSDL 2.0, JAXB 2.0. y abstrae las tareas básicas para crear un servicio web. Aporta dos buenas utilidades wsgen y wsimport que permiten generar de forma automática los artefactos necesarios para generar nuestro servicio y las clases proxy que representarán el servicio web en local para los clientes. Luego a través de

anotaciones, como viene siendo habitual, se indicaran que clase representa nuestro servicio y que propiedades se corresponden con los tags de nuestros xml.

```
@WebService(name = "AddNumbers",
        targetNamespace = "http://duke.org", name="AddNumbers")
public class AddNumbersImpl {
  /**
   * @param number1
   * @param number2
   * @return The sum
   * @throws AddNumbersException
   *         if any of the numbers to be added is negative.
   */
  public int addNumbers(int number1, int number2) throws AddNumbersException {
    if (number1 < 0 || number2 < 0) {
      throw new AddNumbersException("Negative number cant be added!",
                                    "Numbers: " + number1 + ", " + number2);
    }
    return number1 + number2;
  }
}
```

Figura 18 – Web Service con JAXWS

En la figura anterior podemos ver parte de la potencia que nos da la API JAX-WS. Una clase java normal como seria *AddNumbersImpl* la podemos convertir en un Web Service añadiendo la anotación *@WebService(„)*. Las anotaciones están teniendo un papel importante en las ultimas versiones de JAVA, simplifican mucho el código y con el conocimiento necesario ofrecen un gran potencial.

Además las utilidades wsgen y wsimport de las que se ha hablado antes hacen que todo esto pueda llegar a ser automático si montas antes el WSDL del Web Service. De esta forma la utilidad te genera todas las clases necesarias y sólo se le tendría que añadir la lógica a la clases.

2.4 TrustedX

TrustedX es un producto desarrollado por la empresa Safelayer Secure Communications, SA consistente en una plataforma basada en arquitectura SOA y servicios web. El objetivo de dicha plataforma es simplificar el uso de los servicios de seguridad de clave pública para firmas electrónicas, protección de datos y en definitiva cualquier tipo de gestión de identidad electrónica. La plataforma TrustedX incluye:

Un conjunto de **servicios** globales y estándares de seguridad basados en **PKI.**

Soporte para la gestión uniforme y centralizada de usuarios y recursos para el **control de acceso** a los servicios, aportando además, control de acceso único y federación.

La gestión uniforme centralizada de información de log y su **auditoría.**

Dada su arquitectura orientada a servicios, la integración con otras plataformas o aplicaciones resulta muy sencilla, reduciendo la complejidad que hasta la fecha suponía el dotar a cualquier aplicación de mecanismos de seguridad y PKI. Las funcionalidades que incluye la plataforma son las siguientes:

Firma electrónica: permite la verificación y generación de firmas. Se reconocen diferentes entidades de certificación y se permiten generar y custodiar evidencias electrónicas, para que las firmas puedan ser verificadas a lo largo del tiempo.

Protección de datos: permite la protección mediante cifrado y el custodio de los datos garantizando el mantenimiento a lo largo del tiempo y el acceso por parte de entidades autorizadas.

Gestión de claves: permite registrar, revocar, consultar y verificar las claves de las entidades autorizadas.

Autenticación, autorización y control de acceso: a través de un servició común, que hace posible el control de acceso único y federación en toda la plataforma (entre usuarios, servicios web y aplicaciones).

Gestión de objetos y entidades: se describen en un modelo de información uniforme basado en XML. Se ofrecen funciones de registro, consulta y modificación de la información sobre entidades, en particular, información de identidad, configuración y auditoría.

Auditoría y accounting: se gestiona de forma centralizada y uniforme toda la información de log de todos los servicios así como la información de uso y consumo de los mismos.

La plataforma TrustedX está formada por un conjunto de componentes de servicio que cubren las funcionalidades anteriormente descritas.

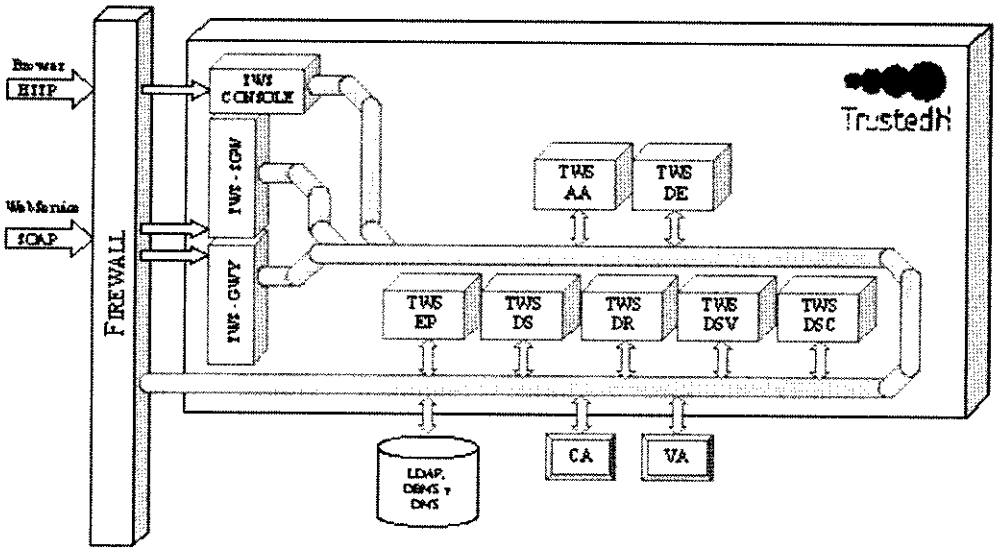


Figura 19 – TrustedX

Los principales servicios con los que se ha trabajado son los de generación y verificación de firmas electronicas (TWS-DSV). Este componente utiliza los servicios de una tercera parte confiable a través del protocolo OCSP, o puede estar conectado a una autoridad de validación que se encargue de validar el estado de los certificados involucrados de manera online.

TrustedX es un sistema basado en políticas que definen el comportamiento del sistema. Así, las políticas, junto a las reglas que se definen en ellas, rigen los servicios anteriormente descritos.

La administración de las políticas se realiza desde la consola de administración de TrustedX que ofrece una visión de todos los elementos que incluye TrustedX, así como la posibilidad de realizar todas las configuraciones necesarias. En particular se han administrado los siguientes tipos de políticas:

Política	Descripción
Autorización	Controla el acceso a los recursos. De este modo, dada una política se concede a una entidad cualquiera permiso para realizar una determinada acción sobre un recurso.
Firma digital	Incluye las reglas que determinan como se deben generar y verificar las firmas a partir del resultado de los procesos de autenticación y autorización.

El mecanismo que se ha utilizado para acceder a los servicios Web que ofrece la plataforma TrustedX es la SoapGateway. El acceso se realiza a través de un puerto Gateway, éste representa la pasarela común que sirve para procesar peticiones SOAP (Simple Object Access Protocol), ejecutar un servicio incluido en el servidor de aplicaciones y construir la respuesta adecuada en formato SOAP.

De este modo todos los servicios están ligados a este puerto de manera que una vez creada la petición SOAP, ésta se envía a TrustedX indicando el servicio que se desea consumir.

La comunicación a través del puerto Gateway está optimizada para soportar el envío de documentos XML de gran tamaño por streaming. Éste es requisito indispensable dada la naturaleza de los servicios TrustedX, que suelen tratar con documentos grandes.

Para enviar una petición se deben tener en cuenta los siguientes elementos:

Elemento	Descripción
Dirección	A la cual se debe enviar, identifica el servidor web dónde se encuentra la SoapGateway. https://trustedx-appliance:8080/trustedx-gw/SoapGateway
SOAP Action	Identifica el servicio que se quiere invocar. Puede tomar distintos valores: sign, verify, archive, etc.
Petición	La petición deberá ser un mensaje SOAP válido.

Capítulo 3
Comparativa de ESB's

En este capitulo se expone la experiencia adquirida en esta tecnología. Cuando se empieza a investigar a los ESB no se tiene conocimiento alguno sobre ellos, esto implica un trabajo extra en la primera fase. Además, al ser una herramienta de integración no sólo se ha de conocer la propia tecnología, sino se han de conocer las tecnologías que quieres integrar. Esto ha sido un impedimento a la hora de agilizar el aprendizaje.

Después del estudio previo, para familiarizarse con la tecnología y para adquirir los conocimientos requeridos, se empieza haciendo pruebas con los diferentes productos. En un principio siempre se lleva acabo las pruebas guiadas que te ofrecen los software. Posteriormente se crean nuevas con el propósito de conocer a fondo la tecnología y ver sus límites. A lo largo de este capitulo se nombran y explican algunas de estas pruebas para mostrar las carencias o atribuciones de los diferentes productos.

Se ha separado la comparativa en unos parámetros que deberían ser importantes de cara a la elección de un ESB. En cada uno se expone el nivel adquirido en ese campo en general y a su vez se explicara lo que encontramos en los productos. Al finalizar el capitulo se realizaran unas conclusiones donde se resumirá todo lo comentado en una tabla y se finalizara con la elección de un ESB Open Source y uno comercial.

Para concluir cabe destacar que durante el capitulo se encuentran muchos fragmentos de opinión personal y condicionadas por la experiencia que se ha adquirido con los diferentes productos. Hay que tener en cuenta que la conveniencia de usar uno u otro modelo de ESB dependerá en gran medida de las características de las aplicaciones a integrar.

3.1 Parámetros de evaluación

3.1.1 Requerimientos y proceso de instalación

La mayoría de los procesos de instalación que se han llevado a cabo han sido bastante sencillos. En muchos de los casos el único requerimiento era descargar el software del sitio Web y descomprimirlo en una carpeta. Una vez hecho, sólo se tenia que ejecutar para empezar con las pruebas.

Donde se han encontrado más problemas ha sido al instalarlo en diferentes sistemas operativos. En un principio se han instalado todos en Windows, pero en algunos casos de uso requería la instalación en un sistema operativo mucho más ligero, como por ejemplo Debian. En ese caso la tarea se hacia mucho más compleja y surgían bastantes incompatibilidades, además estaba mal documentada.

En el apartado de los requerimientos es donde se han encontrado más limitaciones. En general usan demasiados recursos para tareas sencillas. En la sección de pruebas de carga se podrán encontrar varios ejemplos que escenifican estas limitaciones.

Hay dos tipos de instalación:

- **Archivo ejecutable de instalación:** El mayor beneficio en este caso es que la instalación es guiada y por lo tanto fácil de ejecutar. Por otra parte los archivos que se tienen que descargar de la página web son de mayor tamaño, en algunos casos no llegan a los 100 MB, en cambio en otros son un par de imágenes de CD que sobrepasa el GB.
- **Archivos comprimidos guiados:** En este caso el tipo de instalación no es guiada y se tiene que configurar el entorno siguiendo unas directrices que te explican en la web por escrito o que se encuentran en algún archivo de texto en la propia

descarga. Ésta se encuentra en un archivo comprimido en la web. En este tipo de procesos de instalación suelen haber mas problemas que en el anterior.

En primera instancia el caso de la instalación guiada suele ser el que genera menos problemas a la hora de realizarse, pero no siempre es así. Esto es debido a que durante la instalación se suelen pedir referencias a software que no esta instalado porque no consta en los requerimientos de dicha instalación. Uno de los casos que lo escenifica perfectamente es el IBM Websphere:

- **La elección:** Tras revisar los productos que ofrece IBM semejantes al ESB, se encuentran tres: WebSphere ESB, WebSphere Message Broker y WebSphere DataPower. Los tres productos están bien documentados y de una manera u otra servirían para integrar aplicaciones. En este caso se eligió el WebSphere Message Broker ya que era el único que disponía de versión de prueba. De los dos productos que quedaban el que mas interesaba por probar era el WebSphere ESB, ya que el WebSphere DataPower era un apliance físico y para el proyecto seria imposible conseguirlo. Por esta razón se contacto con la empresa para obtener la versión de prueba, pero no fue posible. Posiblemente el hecho de no permitir la descarga de versiones de prueba de algunos de sus productos puede desembocar en una perdida de interés por el mismo.
- **La instalación:** Tras la fase de elección del producto se procede a la descarga de los CD's. Uno de ellos contiene la aplicación y los tres restantes contienen un entorno de desarrollo integrado para facilitar la configuración del mismo. Este paso no dio problemas.
- **Tras la instalación:** Cuando esta todo el entorno preparado se empieza a probar el producto. Para estas pruebas IBM adjunta unos test autoejecutables con una instalación previa. En la configuración de estos test es donde surgen los problemas mas significativos que se han encontrado para ese producto. Si un producto no puede probarse con facilidad puede ser descartado por un posible comprador que tenga varios en su cartera:
    - **Usuario de Windows:** Durante la configuración hay un momento en que demanda el nombre de usuario de Windows. Esto es debido a que los test necesitan tener unos permisos especiales tanto en la maquina como en la red para poder ejecutarse con comodidad. En este punto es donde empiezan los problemas ya que los requerimientos del nombre de usuario son hasta cierto punto exigentes, no tiene que sobrepasar los diez caracteres y tiene que tener permisos de administrador sobre el ordenador. Cuando el producto se encuentra instalado en un entorno empresarial estos dos requisitos no son fáciles de cumplir, los temas administrativos suelen estar muy restringida y cualquier nombre al que le adjuntes el apellido puede sobrepasar los 10 caracteres.
    - **Acceso a la red:** En otro paso de la instalación se demanda ciertos permisos sobre la red en que se esta ejecutando el producto. Y al igual que en el anterior punto estos temas están bastante restringidos.
    - **Búsqueda de recursos:** Por no configurar bien los apartados anteriores casi un noventa por ciento de los test no se pueden realizar. Para realizar las pruebas que aun quedan hacen falta configurar otro tipo de cosas. En este caso el problema se encuentra porque se solicita tener instalado productos de la misma compañía, tales como: la base de datos DB2 y el gestor de colas MQ series. Con productos de otra empresa no funcionaran estos test.

Todas las dificultades que se han encontrado a la hora de ejecutar los juegos de pruebas, no son el reclamo mas conveniente para atraer a nuevos clientes a que compren el producto.

El caso que se ha comentado anteriormente es uno de los más problemáticos que se ha encontrado, pero normalmente no es así. Se puede poner como ejemplo el caso de WSO2 ESB. Un software Open Source que a través de su guía de instalación explica paso a paso como se ha de instalar y los requisitos que tiene que tener el sistema:

- **Java SE Development Kit:** La distribución de Java necesaria
- **Apache Active MQ:** Un proveedor JMS (Gestor de colas), advierten que tienen compatibilidad con éste y que si quieres usar otros sólo tienes que añadir la librería correspondiente.
- **Apache Ant:** Necesario para algunos ejemplos
- **Apache Maven:** Para compilarlo en el caso de que se descargue el código
- **Navegador web:** Para acceder a la consola de administración
- **Memoria:** La memoria que se necesita
- **Espacio en el disco:** Espacio en el disco necesario
- **Sistema Operativo:** Sistemas operativos con los que funciona

Además, indica tanto la configuración de las variables de entorno, como el proceso de descompresión. La ventaja que tiene hacer todo manual es que si en algún momento falla se sabe porque ha fallado.

Otro ejemplo de buen funcionamiento, y en este caso automático, es el del software de BEA, adquirido recientemente por Oracle. Ha sido probado en ambos casos y tras los pequeños cambios introducidos por Oracle sigue funcionando igual de bien. La única molestia es el registro que implica descargarse algo de Oracle, en este caso con BEA no pasaba.

La instalación es automática, y una vez realizada se puede ejecutar alguna aplicación de prueba para ver el comportamiento del ESB sin necesidad de instalar ningún software mas. Es intuitiva y no hace faltar la lectura de ningún documento para llegar a ver en marcha el producto.

Durante esta sección se ha hablado sobre los requerimientos software de alguno de estos productos, quedaría hablar sobre los requerimientos Hardware. Este tema se va a tratar en secciones posteriores de este mismo capitulo. Por esa razón aquí sólo se resumirá por puntos los aspectos más importantes:

- **Memoria:** En la documentación de muchos de los ESB se indica que no necesitan mucha memoria. Esta afirmación es cierta en algunos escenarios, pero cuando las pruebas realizadas se personalizan de maneras que se explicaran en la sección de pruebas de carga y rendimiento, hace falta mucha más memoria. El único ESB en que se ha observado que solicita un consumo de memoria elevado (2 GB) es el de Oracle (la versión de BEA no lo advertía).
- **Espacio en disco:** Ya de por si al ser una tecnología de integración necesita una gran cantidad de espacio en disco para poder almacenar otras tecnologías, o una red distribuida para montar todo el esquema. Además hay algunos ESB, como el de IBM, que por los requerimientos software para hacerlo funcionar ocupan demasiado espacio.

3.1.2 Administración y configuración

Los ESB tienen mucha potencia y la mejor forma de sacarle provecho es con un interfaz amigable y fácil de utilizar. Sin embargo la administración de las herramientas no es siempre tan intuitiva porque aparte de lo que una empresa puede ganar con la venta del producto, una gran cantidad de ingresos provienen de los cursos de iniciación al producto. Aun y siendo producto Open Source existen esta especie de cursos, y suelen dar mas comodidades a los usuarios afiliados al proyecto.

Para que los cursos puedan existir, la configuración y administración de los ESB no tiene que ser trivial. Aunque se apueste por una configuración simple y hasta cierto punto clara, hacen dificultoso el aprendizaje para que se llegue a consumir los extras del producto. Se pueden encontrar ESB ciertamente complicados de entender, aun y estar bien documentados, como puede ser el de IBM:

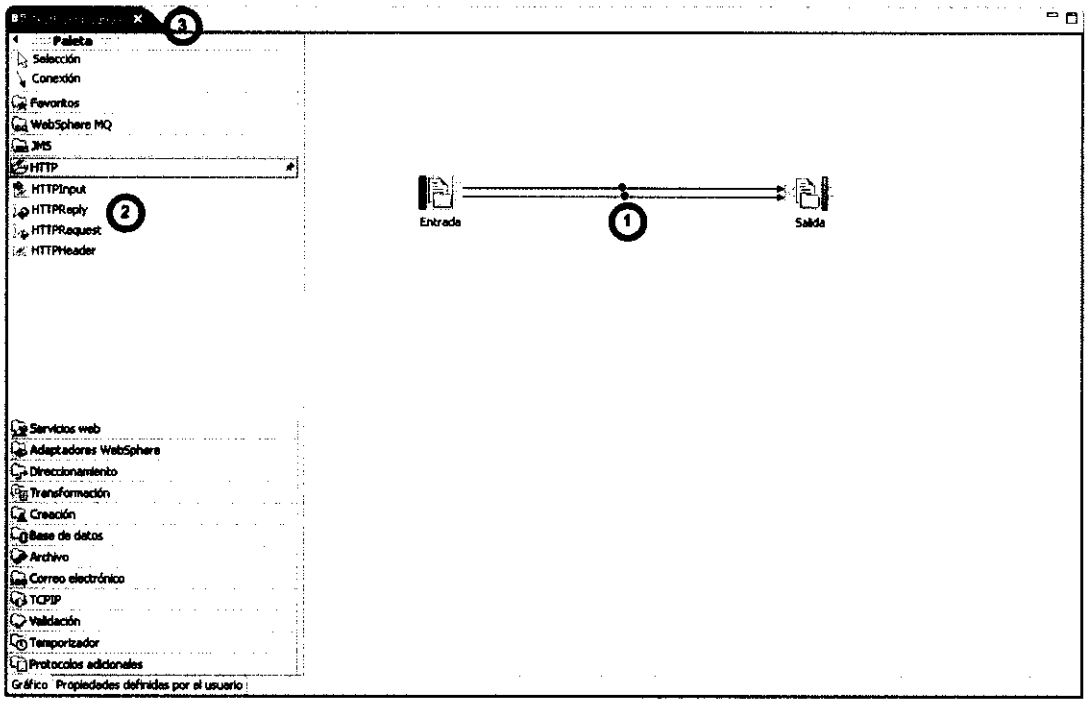


Figura 20 – Herramienta de Administración de IBM Websphere Message Broker

En la Figura 20 se puede ver el sistema que se usa para configurar los flujos el ESB de IBM. Es una modificación de la herramienta Eclipse[3] en la que se puede ver en el punto **(1)** el escenario de configuración y en el punto **(2)** un ejemplo de elementos que podemos añadir, en este caso los elementos relacionados con el protocolo HTTP.

En el punto **(3)** se esta definiendo el flujo de mensajes. Existe una entrada y una salida, y entre las dos puede existir cualquier otro elemento. En este ejemplo tenemos un modulo de entrada de protocolo FILE y otro de salida del mismo tipo. Todos los archivos del primer sitio pasaran a estar en el segundo sitio. En las pestañas de la izquierda podemos encontrar todos los elementos ordenados por protocolo.

Esta parte que se ha mostrado sólo es un escalón en el proceso de configuración del ESB de IBM. En concreto esta es la parte más fácil y que más se asemeja a la que podemos encontrar en cualquier otro ESB. Todos los otros pasos para la instalación no son tan sencillos, y como no se pudo probar los test automáticos que reflejaban todos estos pasos, el producto se descarto.

[3] Es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar aplicaciones.

Como ejemplo de administración más completo esta el WSO2 ESB, se profundizara en los pasos más importantes de cara a una configuración :

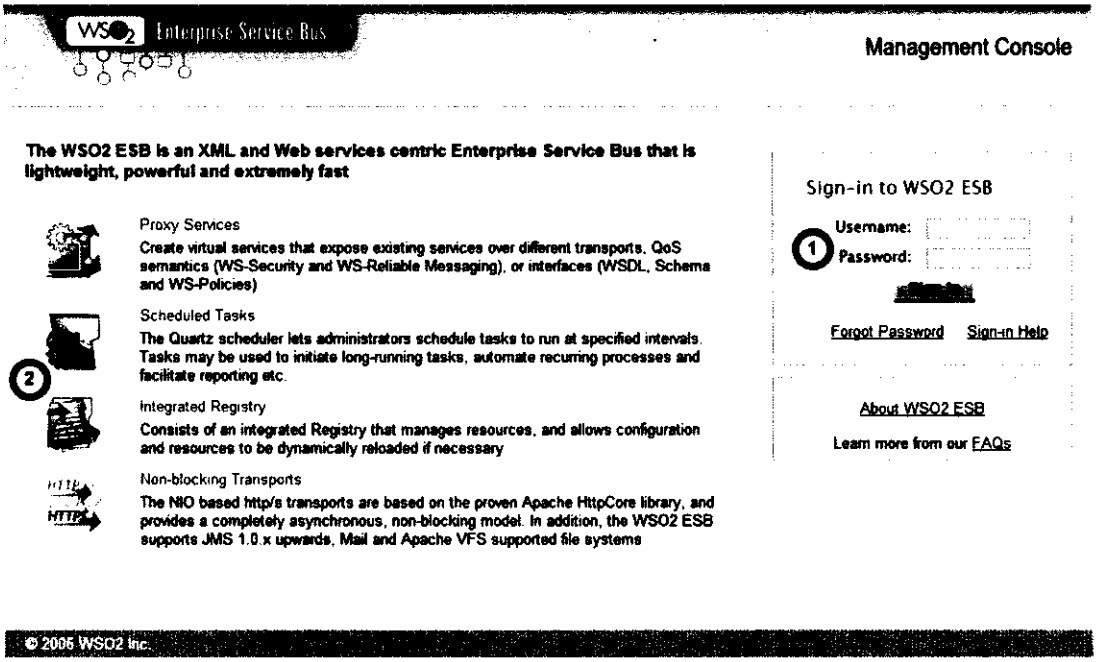


Figura 21 – WSO2 ESB – Administración – Login

Tras ejecutar el script de inicialización podemos acceder a la pantalla de login, tal y como podemos ver en la Figura 21. En esta pantalla se presenta las características del ESB en el punto **(2)** y encontramos el formulario de login en el punto **(1)**.

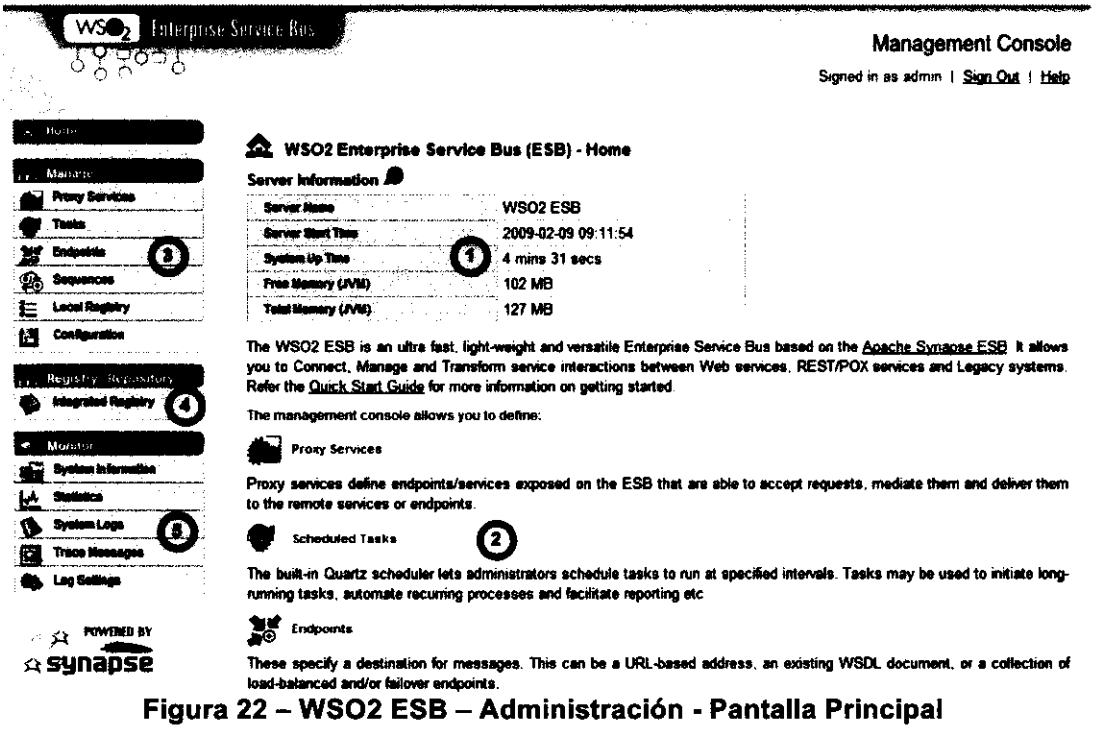


Figura 22 – WSO2 ESB – Administración - Pantalla Principal

Después de hacer login con el usuario admin se encuentra la pantalla principal de la consola de administración, Figura 22. En el punto **(1)** se observa las características del servidor, el más interesante es el dato de la memoria ocupada y libre para comprobar lo que va gastando mientras esta en producción. En el punto **(2)** ofrecen una explicación de algunas características que se pueden definir en la configuración del ESB.

En la parte izquierda se encuentra el menú con todas las opciones que ofrece el ESB. Por una parte, en el punto **(3)** se administran los elementos que se pueden crear, algunas variables de configuración y el registro local. El punto **(4)** se encuentra el registro integrado, éste es un repositorio de todos los elementos que se han creado en el punto **(3)**. En el punto **(5)** están todas las estadísticas y logs.

Al fin y al cabo en todos los ESB la configuración acaba siendo un XML en el que se va recopilando información de todo lo que hace el ESB. El repositorio lo único que hace es guardar todos los archivos XML de todos los elementos para que los se puedan usar en futuras configuraciones.

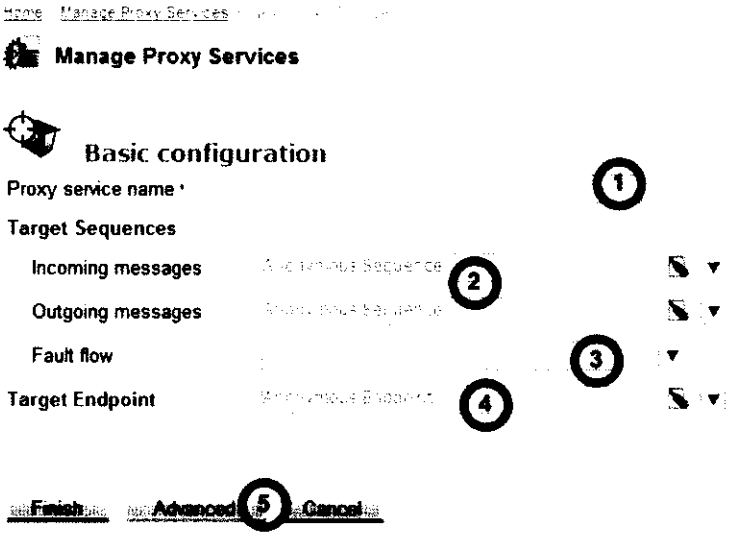


Figura 23 - WSO2 ESB – Administración – Nuevo Proxy

En este ESB los proxy services son los flujos principales. De esta manera podemos tener varios flujos principales activados a la vez. Dentro de un proxy se define un nombre que lo identifique **(1)**, tres posibles secuencias y un endpoint **(4)**.

Todos estos elementos siguen un orden lógico definidos para este ESB:

- **Incoming messages:** Cuando el flujo comienza el mensaje pasara por esta secuencia. Generalmente sirve para añadir cosas de las que le mensaje carece, o pequeñas modificaciones para llegar a un mensaje más completo.
- **Target Endpoint:** Los endpoints son servicios web o otros proxys que consumen el mensaje. En este punto puede ser que el endpoint no retorne mensaje, o que si lo haga. En el caso que lo hiciera pasaríamos al siguiente punto.
- **Outgoing messages:** Esta secuencia se espera a que el endpoint devuelva un mensaje. Si se ha configurado para que este esperando a que lo devuelva y pasa un tiempo (el tiempo que tiene configurado el sistema) y no lo hace da un error y pasa a ejecutarse la secuencia Fault-flow. Si el mensaje llega se ejecutan las acciones que hay configuradas en el flujo.

- **Fault flow:** Esta secuencia generalmente tiene configurada escrituras en logs y que se le envía un mail al administrador para comprobar el fallo. Es la secuencia que se ejecuta cuando algo falla.

Esta es la configuración simple, la cosa se complica un poco más si se entra en la configuración avanzada **(5)**. En este caso deja configurar algunas opciones más referentes a seguridad, protocolos de entrada para el proxy y algunas configuraciones extra.

Cuando se añade alguna de las tres secuencias te dan diferentes opciones, añadir una de las que has añadido anteriormente que están en la configuración del ESB, agregar una del registro (están en archivos físicos) o crearla en ese momento (anonymous). Si se escoge la última opción luego no se puede volver a utilizar.

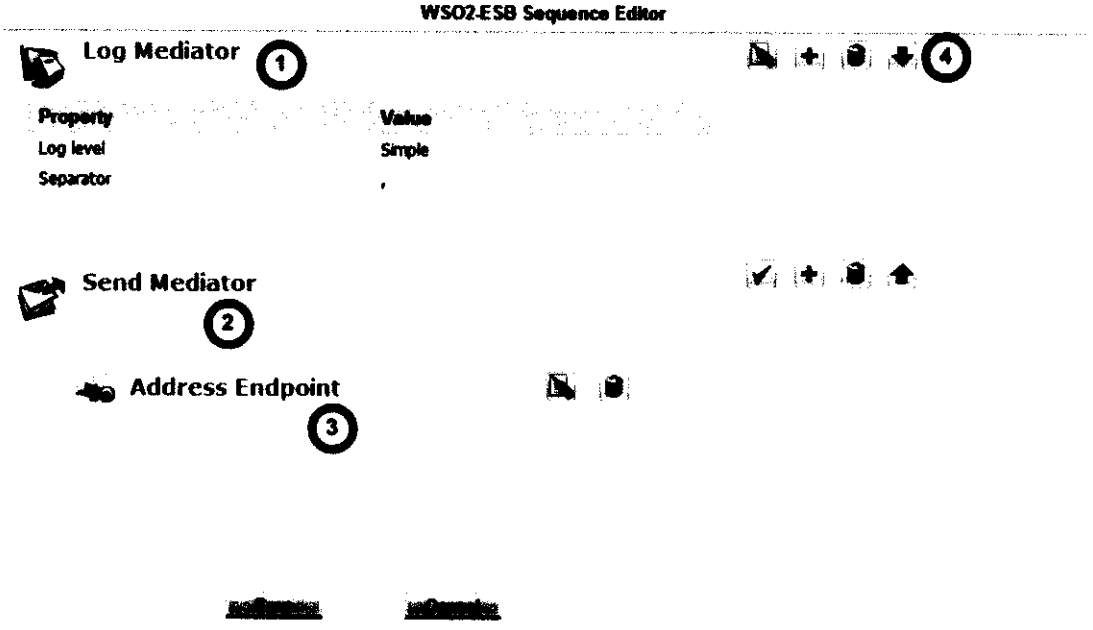


Figura 24 - WSO2 ESB – Administración – Nueva Secuencia

Una secuencia esta formado por elementos llamados mediadores. Cada elemento cumple con una función diferente. Aquí una lista de los más importantes, ordenados por categorías:

- **CORE**
  - **Log:** Imprime en el LOG el mensaje como esta actualmente o alguna de sus variables. Es verdaderamente útil para encontrar errores en la secuencia. Tiene algunos campos que indican como se mostrara el mensaje, y si se mostraran algunas variables solamente o todas. Un ejemplo en Figura 24 **(1)**
  - **Send:** éste es el mediador que da la opción de poder enviar mensajes. Sólo da la opción de definir el endpoint, donde se va a querer que vaya el mensaje.
  - **Sequence: éste** mediador sirve para añadir una secuencia que tengamos alojada en el repositorio, sólo se tiene que hacer referencia a ella.

- **Clone:** Con clone se clonan los mensaje. Los hijos de estos mensajes se procesan en paralelo. Así se pueden hacer varios procesos con un mismo mensaje sin necesidad de hacer un bucle.
- **Iterate:** Es otro de los mediadores que sirve para controlar el flujo de la secuencia del mensaje. Crea una copia del mensaje pero sólo con las características que son aceptadas por la expresión XPATH.
- **Aggregate:** Con este mediador se unen un gran número de mensajes en uno. Por ejemplo, con una pequeña configuración se consigue que coja los primeros 40 mensajes que llegan al flujo.

❑ **TRANSFORM**

- **Header:** Permite modificar, crear o eliminar una propiedad de la cabecera.
- **XSLT:** Como su nombre indica, aplica las transformaciones del XSLT pertinentes al mensaje entrante. El XSLT tiene que estar alojado en el repositorio
- **Xquery:** Se usa para realizar una transformación Xquery.

❑ **FILTER**

- **Filter:** A través de una condición decide si el mensaje sigue o no sigue. La condición se puede hacer con un Xquery o a través de una variable.
- **Switch:** éste es un mediador que permite hacer programación en la secuencia. Es como la típica instrucción de Switch de cualquier lenguaje informático. Se guía por una variable, en el caso que coincida hace lo que tenga que hacer. Si no coincide en ninguno y hay un caso default va a parar a ese. Generalmente en cada caso se define una secuencia.
- **Validate:** éste mediador mira si una sentencia XPATH es correcta. Si lo es, ejecuta la secuencia de mediadores correcta. Si no es correcta ejecuta una serie de mediadores para cuando falla.

❑ **EXTENSION**

- **Script:** WSO2 acepta varios scripts en javascript, Phyton o Ruby. Se puede añadir código inline o a través de un archivo.

En la Figura 24 se observa como es una secuencia, un conjunto de mediadores y siguen el orden en que son configurados. Aparte de ver los mediadores también se ven algunas configuraciones internas como pasa en el punto **(3)**. En este caso el envió se hace sobre una dirección. Cada mediador tiene un menú para editarlo, borrarlo, cambiarle la posición o añadir otro tras éste **(4)**.

Otra de las consolas que esta mejor montada es la del ESB de Oracle, estas imágenes son cuando aun era de BEA, no hay muchos cambios actualmente. Bea Aqualogic Service Bus tiene dos consolas, por una parte la del ESB y por otra la del Server donde va montado

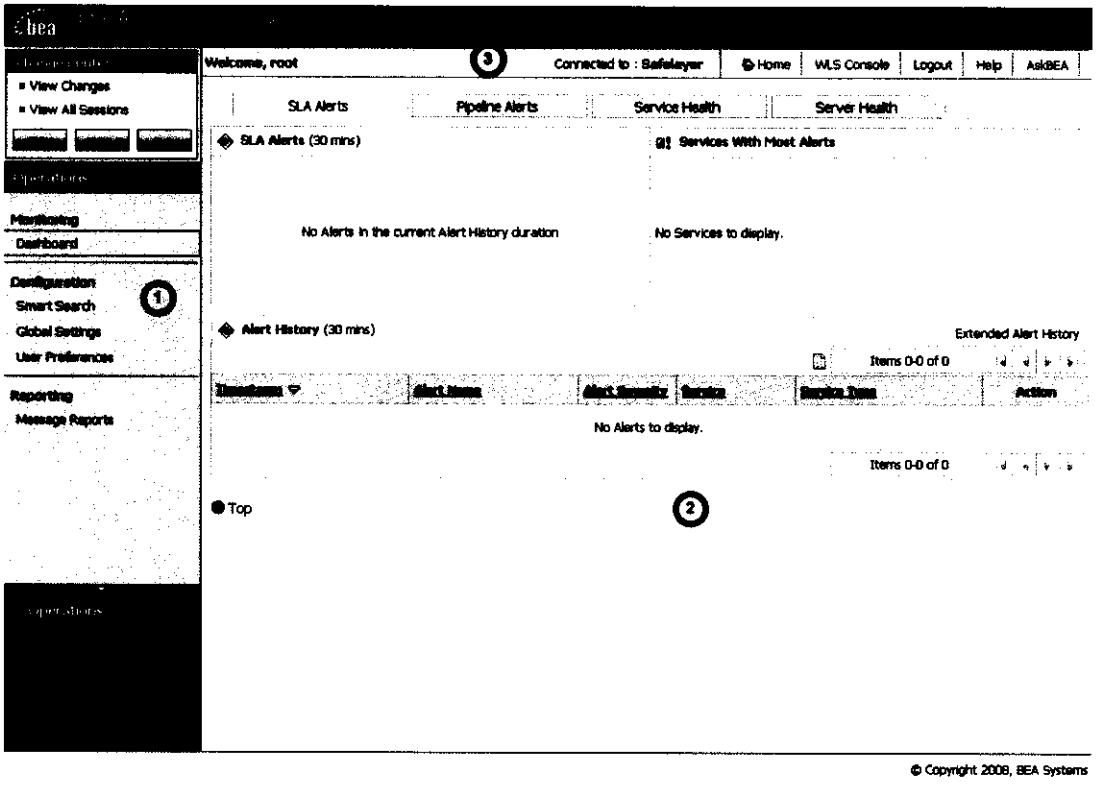


Figura 25 – Aqualogic Service Bus – Admin console

La consola de administración tiene un menú lateral **(1)**, donde se encuentran todas las opciones de configuración. En el punto **(3)** esta el menú de usuario con los comando básicos. En la parte central esta toda la parte que se puede modificar **(2)**.

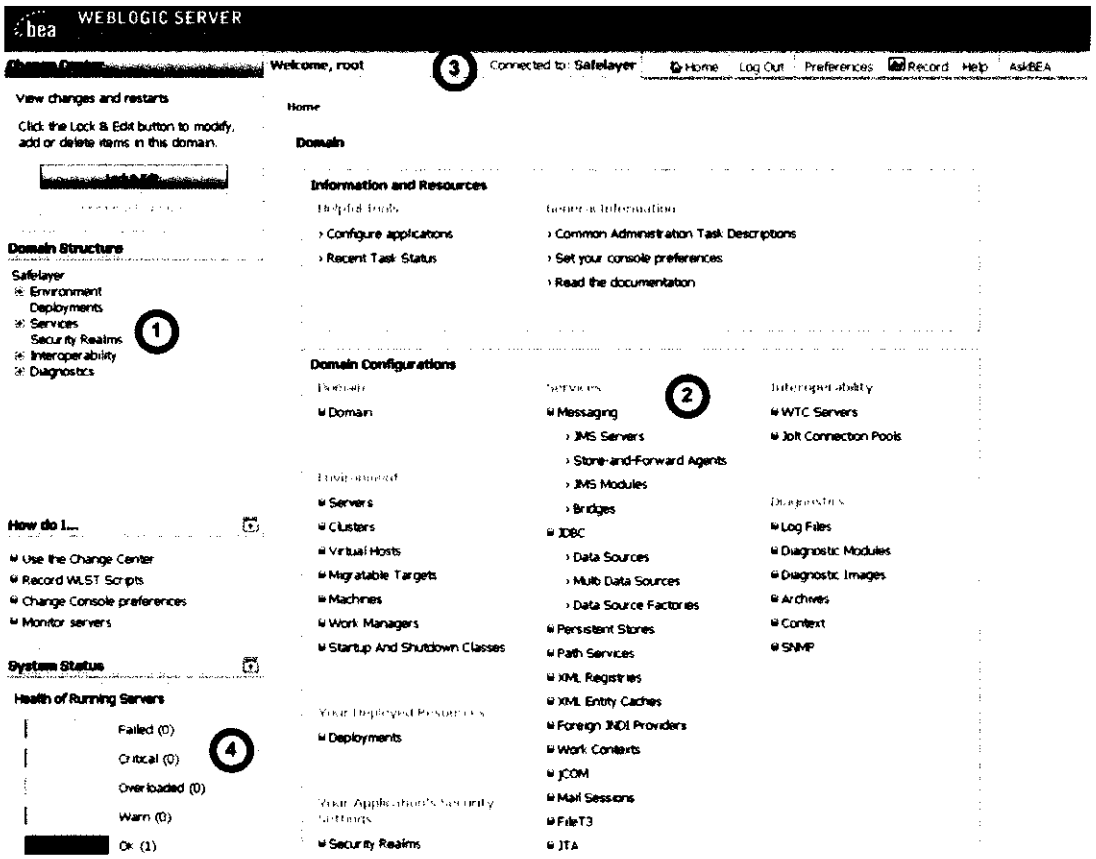


Figura 26 - BEA Aqualogic Service Bus – Server console

En la Figura 26 se muestra la consola de administración, pero en este caso del servidor en el que se ejecuta el ESB. Rápidamente se puede echar un vistazo a los errores que ha dado el servidor **(4)** y ver si su funcionamiento es correcto. Vuelve a tener el menú de usuario **(3)** y también tiene el menú **(1)** de estructura donde se guardan todas las configuraciones del servidor. Una de las veces que se tuvo que modificar algunos parámetros de esta consola fue cuando se quiso poner seguridad. La parte central es donde se modifica todo. **(2)**

Mas adelante se profundizara más a fondo en el panel de administración de este ESB en el siguiente capitulo en el caso de uso que se utiliza. Hay una peculiaridad en la consola de administración de este ESB que lo hace diferente, el control de sesiones. Para empezar a editar se ha de bloquear la configuración para que ningún otro usuario la pueda tocar y desbloquear y grabar una vez acabado. Para que no se pierdan todos los cambios que se realizan se van archivando copias del estado de la configuración y se puede volver a ese estado en cualquier momento.

Para dejar a un lado las consolas vamos a otro tipo, en este caso la visión es muy diferente. La consola sólo se usa para activar o desactivar componentes. Estamos hablando de Petals.

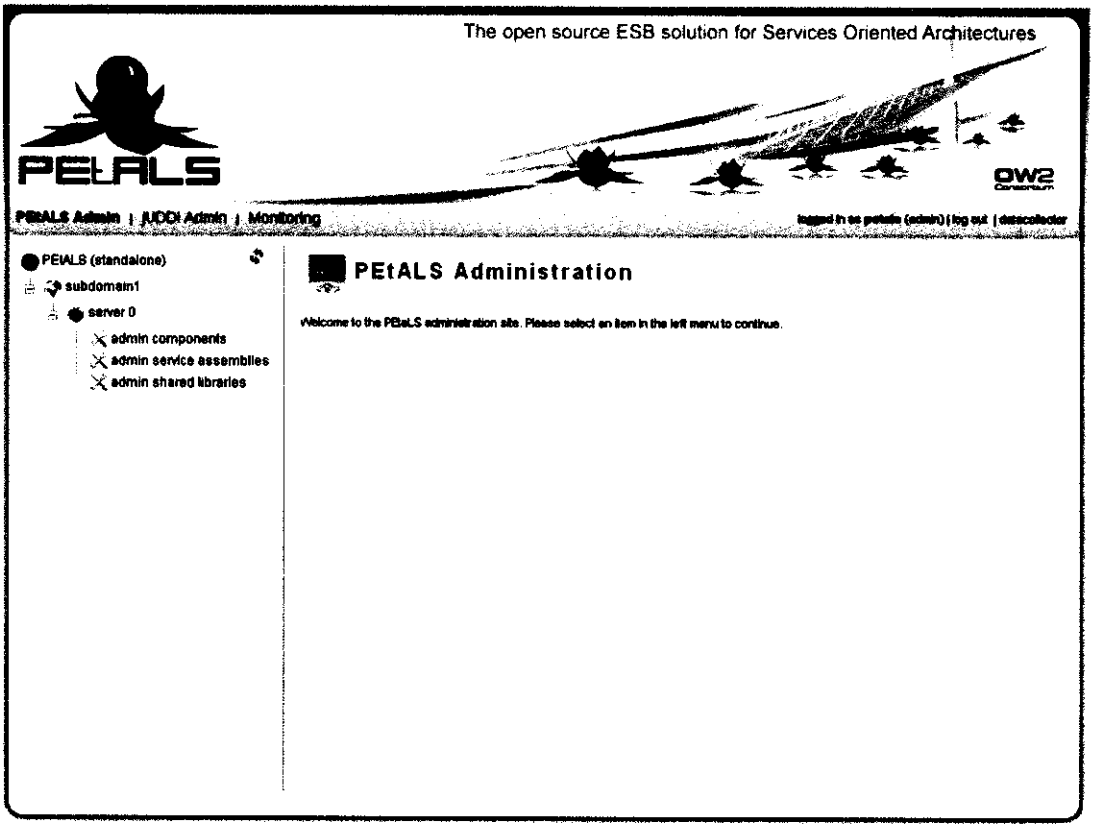


Figura 27 – Petals – Consola de Administración

Como vemos en la Figura 27 el menú es muy simple y sólo da la opción para ver información del estado de los servicios y de los componentes.

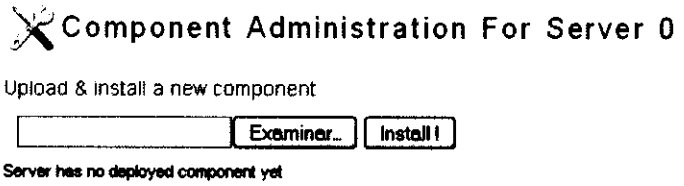


Figura 28 – Formulario para añadir un componente

La visión de Petals es diferente a lo de otros ESB, en este caso cuando se crea un servicio o un componente se crea comprimido en un JAR y esto es lo que se despliega en el ESB. En la Figura 28 se ve un formulario de despliegue de componentes.

Aunque lo verdaderamente cómodo es que los ESB tengan una consola para realizar la configuración hay otros que han apostado por enseñar al usuario el lenguaje XML que utilizan. De esta forma es el usuario quien tiene que redactar las configuraciones en XML, un ejemplo claro es MULE:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:stdio="http://www.mulesource.org/schema/mule/stdio/2.0"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
    xsi:schemaLocation="
      http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.mulesource.org/schema/mule/core/2.0 http://www.mulesource.org/schema/mule/core/2.0/mule.xsd
      http://www.mulesource.org/schema/mule/stdio/2.0 http://www.mulesource.org/schema/mule/stdio/2.0/mule-stdio.xsd
      http://www.mulesource.org/schema/mule/vm/2.0 http://www.mulesource.org/schema/mule/vm/2.0/mule-vm.xsd">

    <description>
        This is a simple component example that demostrates how to expose
        a component over multiple transports.
    </description>

    <stdio:connector name="SystemStreamConnector"
        promptMessage="Please enter something: "
        messageDelayTime="1000"/>

    <model name="echoSample">

        <service name="EchoUMO">

            <inbound>
                <stdio:inbound-endpoint system="IN"/>
                <vm:inbound-endpoint address="vm://echo"/>
            </inbound>

            <echo-component/>

            <outbound>
                <outbound-pass-through-router>
                    <stdio:outbound-endpoint system="OUT"/>
                </outbound-pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

Figura 29 – MULE – Archivo de configuración

El archivo de configuración complica mucho el aprendizaje de la tecnología, pero cuando lo conoces a fondo tiene más ventajas que la consola. Al principio **(1)**, se tienen que indicar todo lo que vas a utilizar, para que mule entienda los protocolos tiene que tener el enlace al archivo donde explica todo lo que puede hacer ese protocolo.

Antes de empezar con el modelo hay una parte donde se definen variables externas **(2)** que se usaran dentro del modelo. En este caso se ha definido un conector y la descripción. Es más cómodo definir aquí todo lo que se pueda si después se va a usar en el modelo en múltiples lugares.

Dentro del modelo se pueden definir todos los servicios que se quiera. Los servicios se asimilan a los proxy services de WSO2 ESB. Tienen un flujo de entrada **(3)**, un flujo de salida **(4)**, y en medio la llamada a un servicio o componente. En MULE llaman a la entrada y salida routers.

En este caso dentro de los routers se definen los protocolos en los que se estará escuchando (entrada), en este caso la consola (MSDOS), y donde se enviara, en este caso también a la consola. En la próxima sección veremos configuraciones más complicadas que explicaran mejor el lenguaje que maneja MULE.

3.1.3 Estado de madurez

Todo producto va evolucionando al largo de su existencia. Durante el tiempo que se ha estado investigando y probando ESB algunos han cambiado, otros han sido comprados y otros están igual que al principio. Teniendo en cuenta desde el primer momento que es una tecnología incipiente, a continuación se describirá el estado actual y como ha ido mejorando o cambiando el producto.

3.1.3.1 Mule

Mule fue uno de los primeros ESB que se probaron. Al ser el único que no tiene una consola se tenia que aprender todo el lenguaje, este lenguaje era bastante difícil de entender. Con la versión 2 de Mule el mayor adelante a sido mejorar el lenguaje y que se entienda mejor. Hubiera sido mejor que hubieran invertido en hacer una interficie grafica, pero no fue así.

```
<connector name="SystemStreamConnector" className=
"org.mule.providers.stream.SystemStreamConnector">
  <properties>
    <property name="promptMessage" value="Please enter something: "/>
    <property name="messageDelayTime" value="1000"/>
  </properties>
</connector>
```

```
<stdio:connector name="SystemStreamConnector"
        promptMessage="Please enter something: "
        messageDelayTime="1000"/>
```

Figura 30 - MULE 1 vs MULE 2

En la Figura 30 se puede ver como la simple definición de un conector antes era más tediosa. Una de las grandes diferencias es la desaparición del nombre de las clases como "org.mule.providers.stream.SystemStreamConnector" por un prefijo delante del conector para indicar de que tipo de protocolo se trata.

Mule es un ESB que lleva tiempo y tiene una gran comunidad de desarrollo, es uno de los que tienen más futuro. Además va lanzando release cada cierto tiempo y eso hace que se vayan corrigiendo todos los bugs que van surgiendo.

3.1.3.2 ServicemiX

Al inicio de la investigación se leyeron varios estudios hecho sobre ESB's que databan de años anteriores 2006-07, en esos estudios destacaban a ServiceMix por delante de Mule como el mejor ESB Open Source. Ahora mismo es uno de los que esta perdiendo popularidad, tanto en los foros como frente a Mule. ServiceMix no tiene interficie grafica, sólo dispone de una herramienta de ayuda para eclipse denominada Cimero, aun y así no consigue despegar. Hace un tiempo que están preparando la versión 4 y no la hacen despegar.

3.1.3.3 Apache Synapse y WSO2 ESB

En este caso se ha decidido que se comentaran juntos porque sus vidas se han ido uniendo poco a poco, hasta el punto de crear una Web conjunta [3]. En el momento en que se probo este ESB el foro de comunicación era bastante pobre y la comunicación con los diseñadores era difícil. Cuando se encontraba un error que era vital para seguir con las pruebas, las dos empresas se inculpaban una a la otra y ninguna lo acababa de arreglar.

Dentro de los problemas, dedicaban bastante parte de sus esfuerzos a mejorar el ESB con bastantes releases, aun y disponiendo de muchos otros productos. La última versión que

se ha probado lleva consigo novedades como la unión a un proyecto conjunto con otros productos llamado Carboon que hace más estable el producto. Esta novedad ahora que salgan más releases tras la reducción de productos, y se esperan mejoras en muchos aspectos.

Otra de las cosas que ha sorprendido es que bugs que se les dijo hace meses aun no los han corregido. Posiblemente den mas prioridad a los bugs que vengan de soporte, que los que provienen del entorno de la comunidad. Aun y así creo que será de los ESB que seguirá adelante en detrimento de ServiceMix.

3.1.3.4 Petals Service Plataform

La empresa OW2 que diseña Petals hace más de 6 meses que no sacan una release, lo más seguro es que se hayan centrado en otras cosas. Aun y ser de las únicas compañías europeas que se esta moviendo con esta tecnología no hay futuro y los otros productos le están comiendo el terreno. Además cuando se probo no dio los buenos resultados que tenían que dar, ni las pruebas que ofrecían con el producto.

3.1.3.5 Kuali

Kuali ha tenido un desarrollo parecido a Petals. Ahora mismo hace mucho que no saca una release oficial de su producto. La costumbre de no sacar release cada cierto tiempo puede llegar a afectar al producto y que este pierda atractivo. Entre los muchos productos para hacer integraciones que tuve que mirar había algunos que no se actualizaban hacia mucho, pero eran productos con renombre y muy estables.

3.1.3.6 GlassFish ESB

Anteriormente, cuando se probo, era Open ESB, hasta que SUN lo integro con su servidor GlassFish. Tras esta integración parece que el producto mejorara y mucho, pero por otro lado vuelve a ser un producto muy joven y es probable que no sea estable en sus primeras versiones.

La primera versión que ha salido de GlassFish ESB no se ha podido probar.

3.1.3.7 Comerciales

En este aspecto los ESB comerciales están en otro escalafón, son productos mas solidos y no suelen prometer cosas que en según que escenarios no pueden hacer. Aunque se ha tenido alguna mala experiencia con alguno de ellos, la imagen es buena en general. En particular cabria destacar que no se sabe que pasara con la compra de Bea por Oracle.

3.1.4 Pruebas de carga y rendimiento

Además de conocer la tecnología de los ESB, el propósito de este estudio era integrar la plataforma TrustedX con alguno de ellos. Se quería ver como se comportarían los productos de una tecnología incipiente, con un producto enfocado a la seguridad y con una solidez mayor.

Por esta razón la mayoría de casos de uso particulares que se hacían sobre los ESB han sido enfocados a integrar esta tecnología. Cuando se conocía las bases del producto se probaba el caso de uso que se puede ver en la Figura 31.

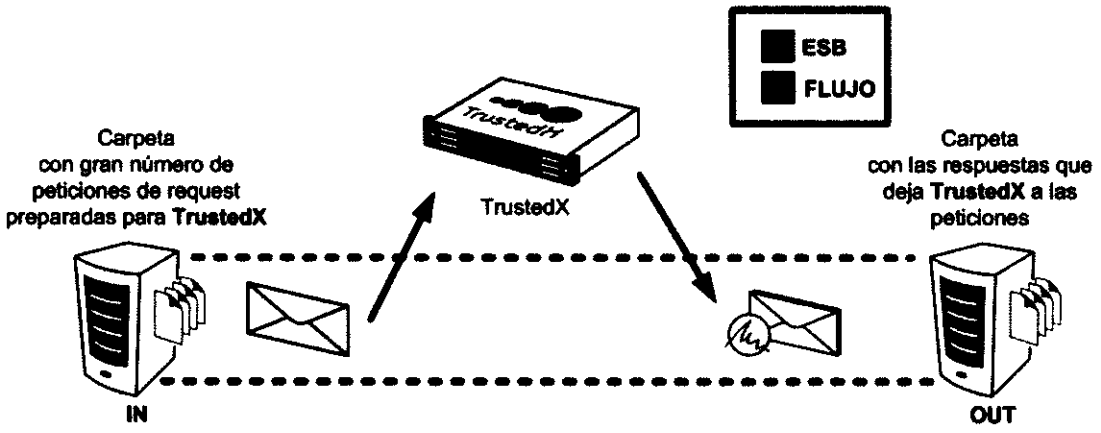


Figura 31 – Caso de Uso 1 – ESB – TrustedX

En este caso se tenia una carpeta con un gran volumen de archivos XML que estaban preparados para ser enviados a TrustedX. Las peticiones que se suelen hacer a los Web Service son de un tamaño muy reducido porque generalmente son para recoger datos específicos, como por ejemplo la cotización en bolsa. En el caso de TrustedX, las peticiones son mayores al MB. Esto es debido a que en la mayoría de operaciones que realiza TrustedX, como por ejemplo firma o verificación, entran en juego documentos y pueden llegar a ocupar bastante.

Una de las dudas que se tenían es si el ESB podría soportar tal cantidad de archivos de gran tamaño. Tenia que enviar un gran numero de peticiones, esperar la respuesta y dejar la respuesta en la otra carpeta. Todos los ESB alardean de un gasto de memoria mínimo, pero en este caso podrían con todo esta carga?

Tras las pruebas se vio que no, cuando aumentabas un poco el numero y tamaño de los documentos a firmar el ESB se quedaba sin memoria. Para este caso de uso se han tenido que configurar varios parámetros generales comunes a todos los ESB, para que dieran un rendimiento mínimo:

- **Numero de archivos a coger a la vez:** Cuando se configura una carpeta como punto de entrada se pueden configurar cuantos archivos como máximo coge el ESB en cada consulta. En teoría esto es útil para no sobrecargar el ESB.
- **Segundos entre consultas:** También se puede configurar cuantos segundos hay entre cada una de esas consultas. Hay que ser conscientes de que ha de ser un tiempo suficiente como para que de tiempo a acabar todas las peticiones que se han hecho en la consulta anterior. Sino se pueden solapar.
- **Memoria máxima:** Aumentar la memoria que el ESB coge del sistema para funcionar.

Para que se entienda un poco más el caso se explicara el comportamiento que se cree que tiene el ESB. Este se ha deducido de las múltiples pruebas que se han hecho:

I. Se detecta la petición
II. Se crea un canal de comunicación
III. Empieza el flujo
IV. Acaba todo el flujo
V. Se cierra el canal de comunicación

Cada canal de comunicación, o denominado técnicamente thread, consume memoria durante todo su proceso. Cuando el thread se cierra toda la memoria que tenia ocupada la deja libre y esta memoria es reaprovechada por otro thread. Esto va pasando hasta que se cierran todos los threads.

Generalmente la memoria que deja libre un thread no es la que suele coger el nuevo thread. Mientras tengan memoria nueva los threads van cogiendo esa en teoría. Cuando la memoria utilizada y la memoria que se ha quedado libre llegan al limite, la que ha quedado libre se limpia y pasa a ser utilizada por los nuevos threads.

En algunos de los ESB que se han probado como el WSO2 ESB la memoria nunca se limpiaba, o no se limpiaba lo eficientemente que se tendría que hacer. Aunque se pusieran pocos archivos a la vez al cabo del tiempo se acababa llenando.

Aunque ese no era el mayor problema, porque hasta cierto punto se podía controlar. La mayor preocupación era cuando se observaba que una petición de 1MB generaba un thread de 5 a 10 MB. Es decir, que el thread utilizaba más de 5 veces más el tamaño de la petición. De esta forma cuando se ponían archivos un poco más grandes la memoria se acababa en muy poco tiempo. Este error fue generalizado en todos los ESB, aunque en algunos pasaba en mayor medida que en otros. Se contacto con las diferentes compañías por foros, comunicando bugs, pero ninguna daba soluciones.

Para implementar una solución para controlar mejor el primer problema usando otro producto, se pensó en las colas. Otra de las razones por las que se pensó en esta solución fue porque algunos ESB no se les podía limitar el numero de threads, es decir cuantos quieres que lean al mismo tiempo de la carpeta, en ese caso se descontrolaba. Se cambio el formato del caso de uso y se paso al de la Figura 32. Tal y como se explica en la figura el ESB tiene a la cola de intermediario. Se quería hacer que la cola hiciera de barrera entre el ESB y el envío a TrustedX, de esta forma controlar el flujo. En este caso lo que se acabo quedando sin memoria fue la cola.

El primer proceso del ESB no paraba de añadir mensajes a la cola y se había limitado la consumición de los mismos para que el segundo proceso fuera desahogado. De este modo la cola se llenaba, y cuando la configuramos para que no dejara pasar más mensajes, el ESB en vez de parar lanzaba errores.

La primera prueba se hizo con muchos de los ESB. La segunda sólo se hizo con el WSO2 ESB. A partir de entonces se le dedico más tiempo a las pruebas de los dos casos de uso que se explican en el siguiente capitulo.

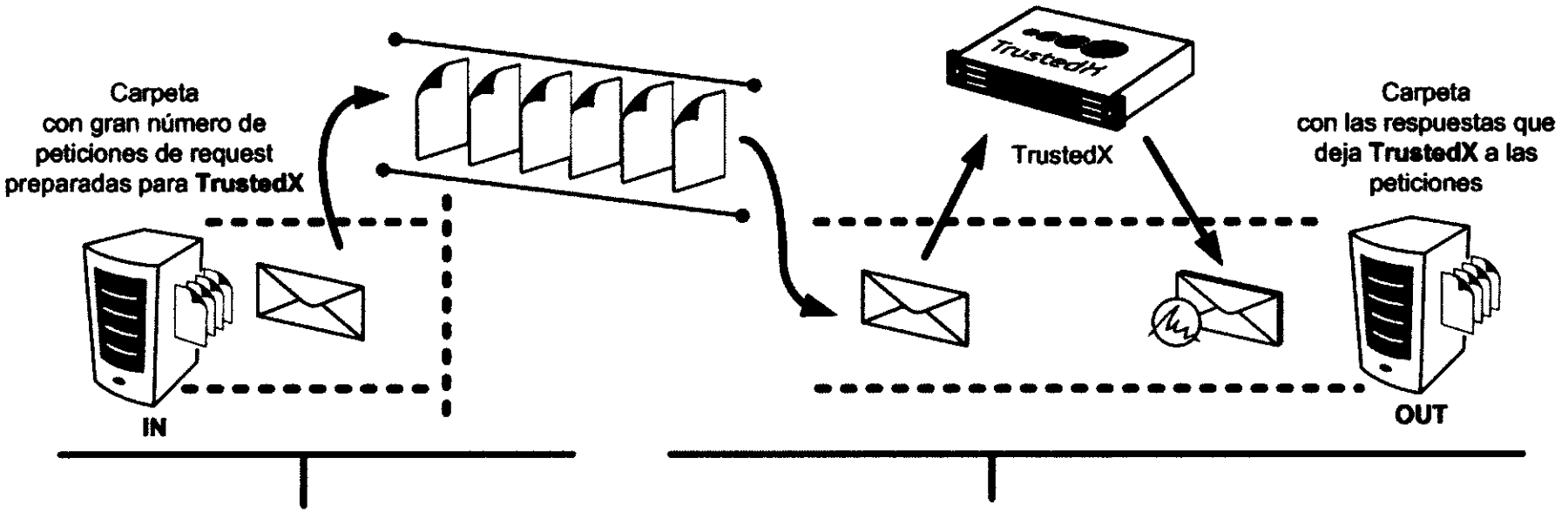


Figura 32 - Caso de Uso 2 – ESB – Cola - TrustedX

3.2 Conclusiones

La tabla de la Figura 34 resume la comparativa realizada sobre los ESB, en función de los siguientes parámetros.

- **Protocolos:** Se mide el número de protocolos que el ESB soporta. Si soporta muchos, mayor calidad.
- **Instalación:** Se mide la dificultad en la instalación, además dificultad menos nivel.
- **Consola:** Se observa si tiene o no, y además la calidad de la misma.
- **Configuración:** Se mide la dificultad al hacer la configuración, además nivel menos dificultad.
- **Madurez:** Se mide la solidez en el producto, además nivel más calidad de producto y más futuro.
- **Pruebas:** Se miden las pruebas realizadas, tanto las de rendimiento como las de carga. Además nivel mejor han ido las pruebas.
- **Comunidad:** Se observa si tiene o no, y además la calidad de la misma, número de usuarios, etc...
- **TrustedX:** Se mide la capacidad que tienen de aguantar los ESB con TrustedX dando un rendimiento máximo y con una capacidad de memoria limitada. A mas nivel, mas resistencia antes de consumir toda la memoria.
- **Total:** Una valoración global contando todos los parámetros anteriores y la percepción personal adquirida a partir de las pruebas realizadas.

SIMBOLO	DEFINICIÓN
✗	En los 3 parámetros que se utiliza significa que no tiene o es francamente pobre.
✓−−	Nivel muy bajo, es muy difícil o la calidad es muy mala.
✓−	Nivel bajo, es difícil o la calidad es mala.
✓	Nivel normal, no es destacable o esta poco probado.
✓+	Nivel bueno, es fácil o la calidad es buena.
✓++	Nivel bueno, es muy fácil o la calidad es muy buena.

Figura 33 - Leyenda

	Protocolos	Instalación	Consola	Configuración	Madurez	Pruebas	Comunidad	TrustedX	Total
Mule	✓++	✓+	✗	✓	✓+	✓	✓+	✓−	✓++
ServiceMix	✓	✓	✗	✓	✓−	✓−	✓	✓−−	✓−
Apache Synapse y WSO2 ESB	✓+	✓+	✓+	✓+	✓+	✓−	✓−	✓−−	✓+
Petals Service Plataform	✓	✓	✓−	✓	✗	✓−−	✓−	✓−−	✓−−
Kuali	✓	✓	✓−	✓	✗	✓−−	✓	✓−−	✓−−
GlassFish ESB	✓	✓	✗	✓−	✓	✓−	✓−	✓−−	✓
Weblogic ESB (Oracle)	✓+	✓++	✓++	✓+	✓+	✓	✓	✓−	✓++
IBM WebSphere Message Broker	✓+	✓−−	✗	✓−	✓+	✓−	✗	✓−−	✓

Figura 34 – Tabla comparativa

Tras todo el estudio realizado y siguiendo el resumen de la tabla de la Figura 34 se llega a la conclusión de que los ESB más óptimos para las tareas que se quieren realizar son: Mule (Open Source) y Weblogic ESB Oracle (comerciales).

Actualmente parece que ninguno de los ESB aguanta cargas de gran tamaño con archivos de más de 1 MB durante un tiempo indefinido y con una memoria limitada. Por tanto se llega a la conclusión que si se necesita este escenario se tendrá que adquirir un servidor con los requerimientos necesarios para que funcione.

Otra de las cosas que se ha considerado de gran importancia es la comunidad. Igual que cualquier empresa que se precie debe tener un gran equipo de testing, las empresas que colaboran con proyectos Open Source deben aprovechar la comunidad para que prueben el producto lo mejor posible. Por esa razón se piensa que es importante que hasta cierto punto la empresa cuide a la comunidad de usuarios que desinteresadamente este probando el producto.

Cabe destacar que de los ESB que se ven en la tabla comparativa hay ESB que se han considerado de mala calidad aun y cuando para otras tareas posiblemente serian mas adecuados. Las conclusiones que se han sacado en esta tabla están basadas en la experiencia que se ha tenido con los ESB y en la opinión personal.

Para finalizar cabe decir que todos los ESB tienen cosas en común, pero no son tan semejantes para realizar un patrón de aprendizaje. Todo esto se arreglaría si en un futuro se estandarizase un lenguaje de configuración para ESB, de esta forma hacer el paso de uno a otro no sería nada difícil.

Capítulo 4
Casos de uso

4.1 Web Service semántico con MULE

4.1.1 Objetivo y descripción

El ESB es una herramienta básica de integración, por esta razón ha sido elegida para integrar el grupo de aplicaciones que entran dentro del entorno que se describe en la Figura 35. Primero de todo se describirá cada uno de los elementos:

- **Interidy IDP:** Es un proveedor de identidad que permite a los usuarios generar tarjetas de información a partir de los datos que el proveedor custodia. Con estas tarjetas los usuarios pueden autenticarse de forma más segura y con una protección adicional contra los ataques de suplantación de identidad. Este proveedor permite obtener datos verificados a partir de ficheros FOAF firmados electrónicamente o del certificado del DNIe. Mas información en [15]
- **PKI TrustCenter:** Es un sistema capaz de ser alimentado con la información extraída del mundo de las infraestructuras de clave pública y para dar respuesta a consultas e inferir nuevo conocimiento. Por medio de una representación gráfica de la información extraída de las ontologías, se ofrecerá al usuario la capacidad de navegar o interactuar con la jerarquía de entidades de confianza. [17]
- **FOAF Manager:** Es un gestor de identidades FOAF controladas por el usuario. Es una aplicación que da soporte a la manipulación de instancias FOAF mediante herramientas semánticas, permitiendo al usuario importar atributos de identidad de un conjunto de redes sociales y, lo más importante, dotar al usuario herramientas para controlar su propia identidad mediante creación de FOAFs , edición, importación , publicación , etc.. Más información en [16]
- **TrustedX:** Es una plataforma de servicios web que provee mecanismos para tratar la autenticidad, la firma electrónica y el cifrado de datos. Basado enteramente en políticas, una vez configurado, TrustedX gestiona la confianza corporativa de forma automática.
- **CSLA:** La Autoridad Local de Sello de Contexto (CSLA), definida e implementada en [14], proporciona sellos de contexto XML que permiten mejorar los mecanismos de confianza que acompañan las operaciones telemáticas.
- **Internet:** Otro de los elementos importantes en la estructura es Internet. Mas que un elemento, es una escenificación de que el ESB puede abstraer servicios que están fuera de la red y nutrirse de ese conocimiento.

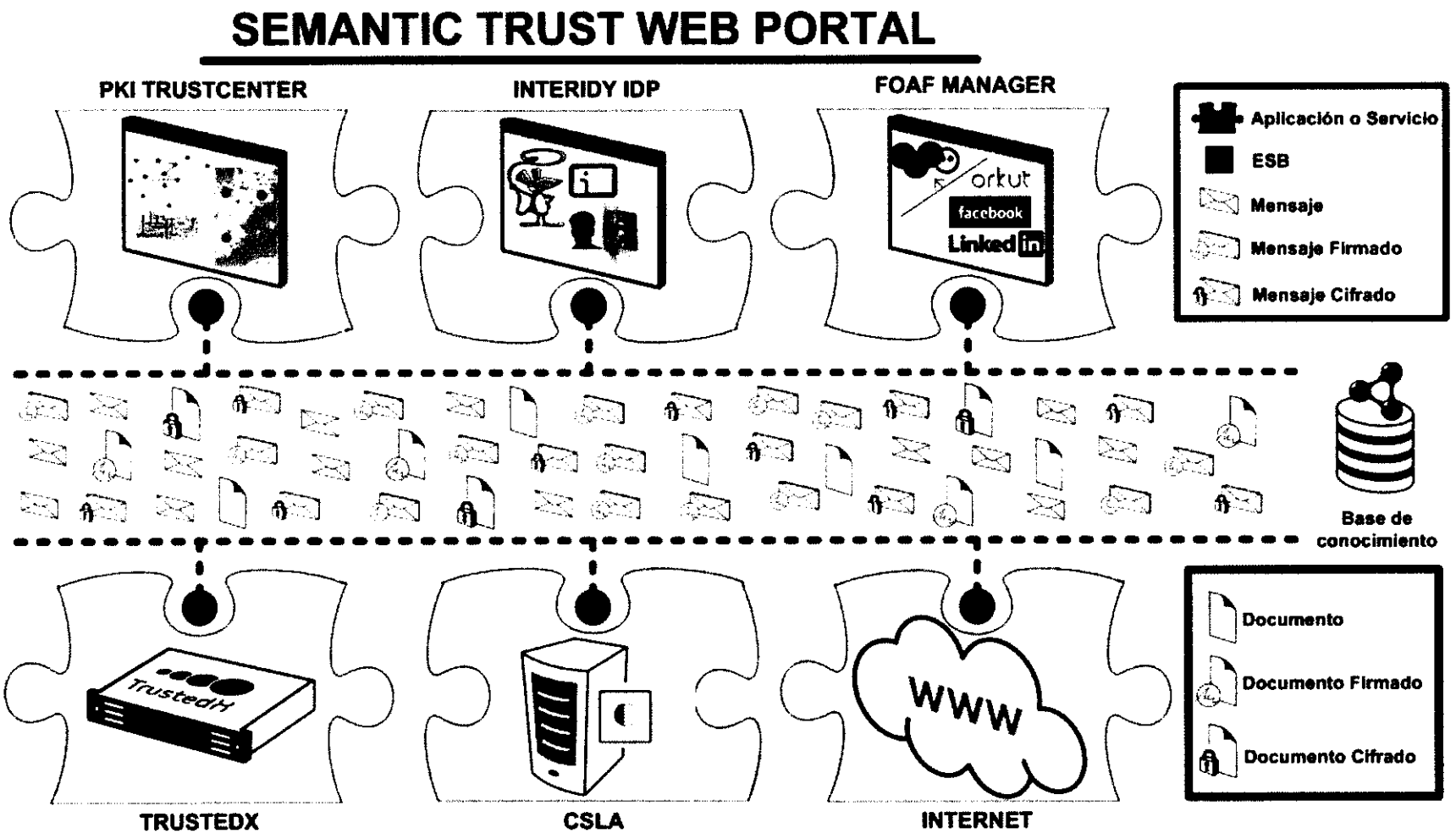


Figura 35 – Uso del ESB con MULE y la base de datos de tripletas

Por otro lado está la base de datos de conocimiento que es la encargada de guardar toda la información que usan todas las aplicaciones o elementos conectados al ESB. Su acceso esta restringido por el ESB y se ofrece a través de un Web Service.

En este caso el ESB hace de punto de unión de todas aplicaciones y distribuye los servicios que ofrece cada aplicación hacia las otras, en particular el servicio de consulta y almacenamiento de la base de datos.

El objetivo de este caso de uso es demostrar la facilidad de integración de servicios y aplicaciones y ver que beneficios se obtienen cara al futuro. Otro de los objetivos ha sido ver que tipo de aguante tiene el ESB cuando se encuentra en producción, ya que este escenario se pretende pasar a producción.

Se ha utilizado MULE para este caso de uso porque se quería probar un Enterprise Service Bus Open Source y según la comparativa del capitulo anterior este era el mejor posicionado. De esta forma completamos los dos casos de uso con dos tipos de ESB diferentes.

4.1.2 Diseño

Como se puede observar en la Figura 36 existen tres interfaces diferentes y cada una dispone de varias operaciones. El usuario se comunica a partir de ellas para acceder a la base de datos. Los tres servicios son ofrecidos por el ESB.

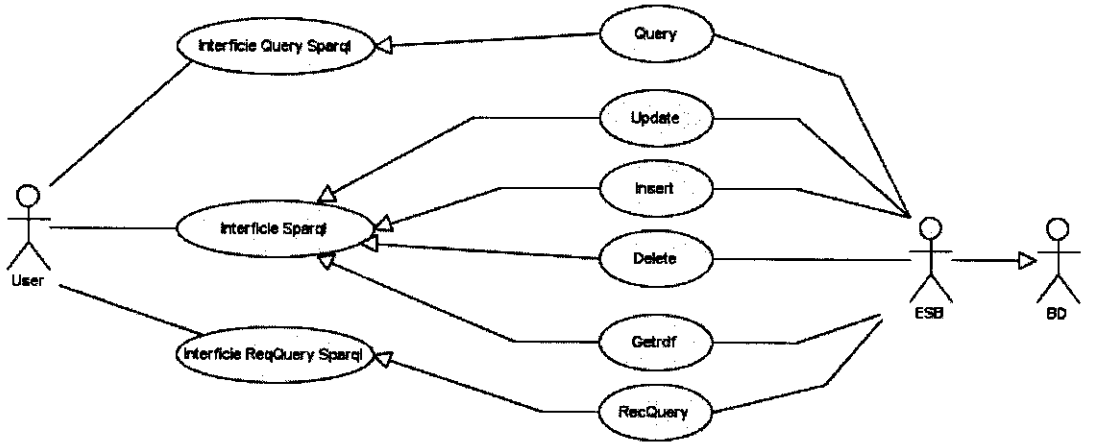


Figura 36 – Diagrama de casos de uso del Web Service Semántico

4.1.2.1 Query

Flujo básico: Query

USER	ESB	BD
1. User envía una petición SOAP por la interficie Query Sparql.		
	2. El ESB recibe la petición, mira que está bien formada y la registra en un LOG. Monta la petición SPARQL y la envía.	
		3. La base de datos recibe la petición y devuelve la respuesta.
	4. El ESB monta la respuesta de la base de datos en RDF y la envía al usuario.	
5. User recibe la respuesta del ESB.		

Flujo alternativo: Error

Existen varios errores por los cual puede fallar el servicio, por un lado que falle el ESB por algún error interno, en este caso devolvería un mensaje de sistema. Por otro lado puede fallar la consulta porque este mal formada o porque la capa de abstracción de Jena no la entienda.

4.1.2.2 Update

Flujo básico: Update

USER	ESB	BD
1. User envía una petición SOAP por la interficie Sparql.		
	2. El ESB recibe la petición, mira que esta bien formada y la registra en un LOG. Monta la petición SPARUL y la envía.	
		3. La base de datos recibe la inserción y devuelve la respuesta con el resultado si ha ido bien o mal.
	4. El ESB verifica que todo ha ido bien y envía al usuario una respuesta genérica con la validez de la transacción.	
5. User recibe la respuesta del ESB.		

Flujo alternativo: Error

El caso del Update es diferentes al del Query, en este caso se envía una query SPARUL. Este lenguaje es bastante limitado en el entorno en el que se esta usando y no nos ofrece tanta potencia como el SPARQL.

4.1.2.3 Insert

Flujo básico: Insert

USER	ESB	BD
1. User envía una petición SOAP por la interficie Sparql con el RDF que se quiere insertar.		
	2. El ESB recibe la petición, mira que esta bien formada y la registra en un LOG. Inserta el RDF en la base de datos por la API de Jena.	
		3. La base de datos recibe la inserción y devuelve la respuesta con el resultado si ha ido bien o mal.
	4. El ESB verifica que todo ha ido bien y envía al usuario una respuesta genérica con la validez de la transacción.	
5. User recibe la respuesta del ESB.		

Flujo alternativo: Error

El caso del Insert vuelve a ser diferente, aquí se esta añadiendo un RDF entero en la petición SOAP. De esta manera puede fallar porque el RDF este mal formado o no entienda alguna parte, en este caso devolvería error de inserción.

4.1.2.4 Delete

Flujo básico: Delete

USER	ESB	BD
1. User envía una petición SOAP por la interficie Sparql.		
	2. El ESB recibe la petición, mira que esta bien formada y la registra en un LOG. Borra el RDF de la base de datos por la API de Jena.	
		3. La base de datos recibe la petición de borrado y devuelve la respuesta con el resultado si ha ido bien o mal.
	4. El ESB verifica que todo ha ido bien y envía al usuario una respuesta genérica con la validez de la transacción.	
5. User recibe la respuesta del ESB.		

Flujo alternativo: Error

En este caso el único fallo que puede existir es porque no existe el ID del grafo que se quiere borrar.

4.1.2.5 Getrdf

Flujo básico: Getrdf

USER	ESB	BD
1. User envía una petición SOAP por la interficie Sparql.		
	2. El ESB recibe la petición, mira que esta bien formada y la registra en un LOG. Toma el RDF de la base de datos por la API de Jena.	
		3. La base de datos recibe la petición y devuelve la respuesta con el resultado si ha ido bien o mal.
	4. El ESB verifica que todo ha ido bien y envía al usuario una respuesta genérica con la validez de la transacción.	
5. User recibe la respuesta del ESB.		

Flujo alternativo: Error

El error es semejante que en el caso anterior, si el ID que se le envía no existe devolverá error. También puede fallar en la generación del RDF.

4.1.2.6 RecQuery

Flujo básico: RecQuery

USER	ESB	BD
1. User envía una petición SOAP por la interficie RecQuery Sparql.		
	2. El ESB recibe la petición, mira que esta bien formada y la registra en un LOG. Monta la múltiple petición SPARQL y la envía.	
		3. La base de datos recibe las peticiones y devuelve la respuesta.
	4. El ESB monta la respuesta de la base de datos en RDF y la envía al usuario.	
5. User recibe la respuesta del ESB.		

Flujo alternativo: Error

Esta consulta es mas complicada que las demás, ya que a su vez tiene un conjunto de consultas. Los fallos que pueden haber son los mismo que existen en la query, mas lo que se pueden encontrar al reutilizar resultados de querys.

4.1.3 Implementación y configuración

Por una parte se ha montado todo lo necesario para poder tener el ESB en producción dentro de un servidor, por otra parte se ha desarrollado el código necesario para ofrecer los servicios semánticos y por último se ha configurado el ESB para integrar todo.

4.1.3.1 El servidor

Se buscaba una manera ligera para montar el ESB y tras las observaciones del administrador de la empresa se decidió usar Debian como sistema operativo. A través de una maquina virtual con el Debian recién instalado se fue montando todo lo necesario.

Lo primero de todo fue montar el escenario de tal forma que se pudiera instalar el ESB sin problemas. También tuvo que instalarse la base de datos MYSQL para poder alojar la base de conocimiento y un servidor Apache para alojar aplicaciones futuras.

Los mayores problemas fueron causados por el desconocimiento del sistema operativo, tanto para configurar la red, como para instalar algunas aplicaciones. El mas significativo fue la configuración del proxy de la empresa.

4.1.3.2 Desarrollo de código

En la primera fase del desarrollo del código se tuvo que probar la conexión a la base de datos y como funcionaba Jena de manera general, antes de introducir ese código en un Web Service.

Cuando se tuvo un conocimiento suficiente y se habían probado las consultas de manera manual se implemento un Web Service con JAX-WS y las java anotations. La primera interficie que se creo fue la encargada de hacer consultas sparql. Todos los problemas de configuración fueron aquí, cuando se hicieron las otras interficies ya se tenía un conocimiento previo.

Los problemas mas complicados de solucionar estuvieron ligados a la creación o modificación del wsdl. Las librerías utilizadas tenían unas restricciones muy estrictas en cuanto a como tenia que estar estructurado el wsdl.

4.1.3.3 Configuración del ESB

Finalmente se tenía que configurar el ESB para mostrar el servicio web a las otras aplicaciones. Se configuraba una URL de entrada donde todo el mundo mandaba los mensajes, a partir de eso el ESB gestiona las peticiones. Y las envía a la petición que toca.

4.1.4 Resultados

Este escenario se puso en producción en Agosto y ha sido utilizado durante estos meses por varios usuarios. Poco a poco se han ido detectando errores y añadiendo mejoras. Aun y así ha funcionado bien. Cabe destacar que las peticiones realizadas ha estado en producción no han sido excesivamente exigentes en cuanto a número de peticiones y presencia de usuarios, pero se considera que ha dado el servicio que se le demandaba.

4.2 Integración de TrustedX con Oracle Weblogic ESB

4.2.1 Objetivo y descripción

A diferencia del caso anterior, este escenario está formado solamente por TrustedX y el ESB. En este caso el ESB no pretende integrar una gran suma de aplicaciones, sino que quiere facilitar la usabilidad del producto ofreciendo una nueva interfaz de interoperabilidad entre el usuario (humanos, aplicaciones u otros servicios web) y TrustedX.

La forma tradicional de usabilidad de los servicios que ofrece TrustedX es a partir de servicios web SOAP. Tiene un conjunto de web service a los cuales se les envía una petición que hacen un trabajo interno y devuelven un SOAP con la respuesta. Un ejemplo de entre todos los servicios que ofrece TrustedX es el de firma digital, el usuario envía un documento a TrustedX y lo firma.

Este caso de uso tiene como objetivo que el usuario puede probar diferentes servicios de TrustedX sin tener que aprender a montar un SOAP, para ello se ha pensado un proceso patrón que podría servir para la mayoría de servicios:

1) El usuario deja lo necesario para el servicio en la carpeta de entrada IN.
2) El ESB lo detecta, lo recoge y ejecuta una función java para codificarlo a Base64.
3) El ESB monta el SOAP y lo envía a TrustedX.
4) El ESB recoge la respuesta que devuelva TrustedX y ejecuta una función java para descodificarla.
5) El ESB deja la información importante en una carpeta OUT con el mismo ID de la entrada para que el usuario lo pueda reconocer.

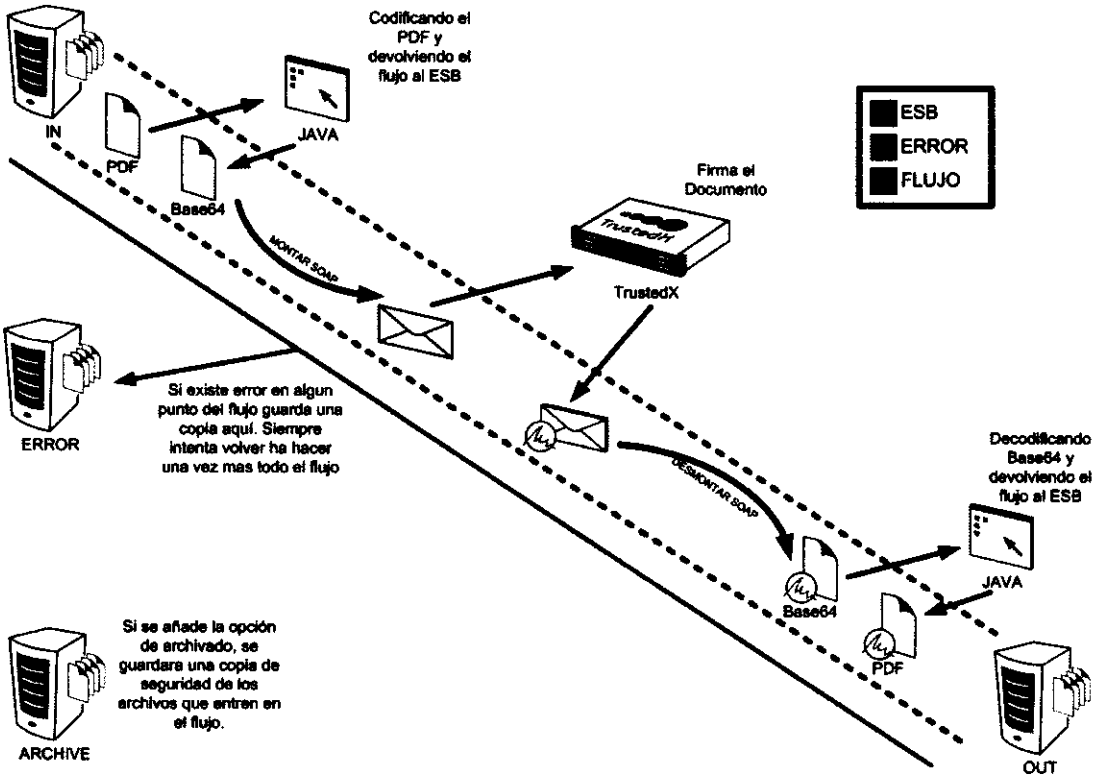


Figura 37- Oracle Weblogic ESB con TrustedX

De esta manera el usuario ha probado un servicio de TrustedX ofreciendo solo lo mínimo al ESB y recibiendo una respuesta que cualquier usuario puede comprender. Como ejemplo, en la Figura 37 podemos observar este caso, para el servicio de firma.

Para este caso de uso se escogió el ESB comercial de Oracle, que desde nuestro punto de vista es uno de los más estables en cuanto a gran número de peticiones. Aun y así, cuando se intentaba enviar archivos de un peso considerable el ESB no tenía suficiente memoria para transportar muchos archivos.

Por esta razón se decidió añadir, en la medida de lo posible, código externo en el producto para intentar corregir esta posible limitación de manera eficiente. El mayor problema se encontró a la hora de montar el SOAP, mientras en el envío el ESB mantiene todo el proceso en streaming y no aumenta la memoria utilizada, cuando quiere modificar el archivo lo tiene que guardar todo en memoria.

Para solucionar ese problema se añadieron unas clases JAVA que se ejecutan en partes estratégicas del flujo para hacer el trabajo que hacía el ESB, pero todo en streaming hasta los cambios, tanto para montar el SOAP, como para extraer la información importante que eran los dos lugares de mas gasto de memoria.

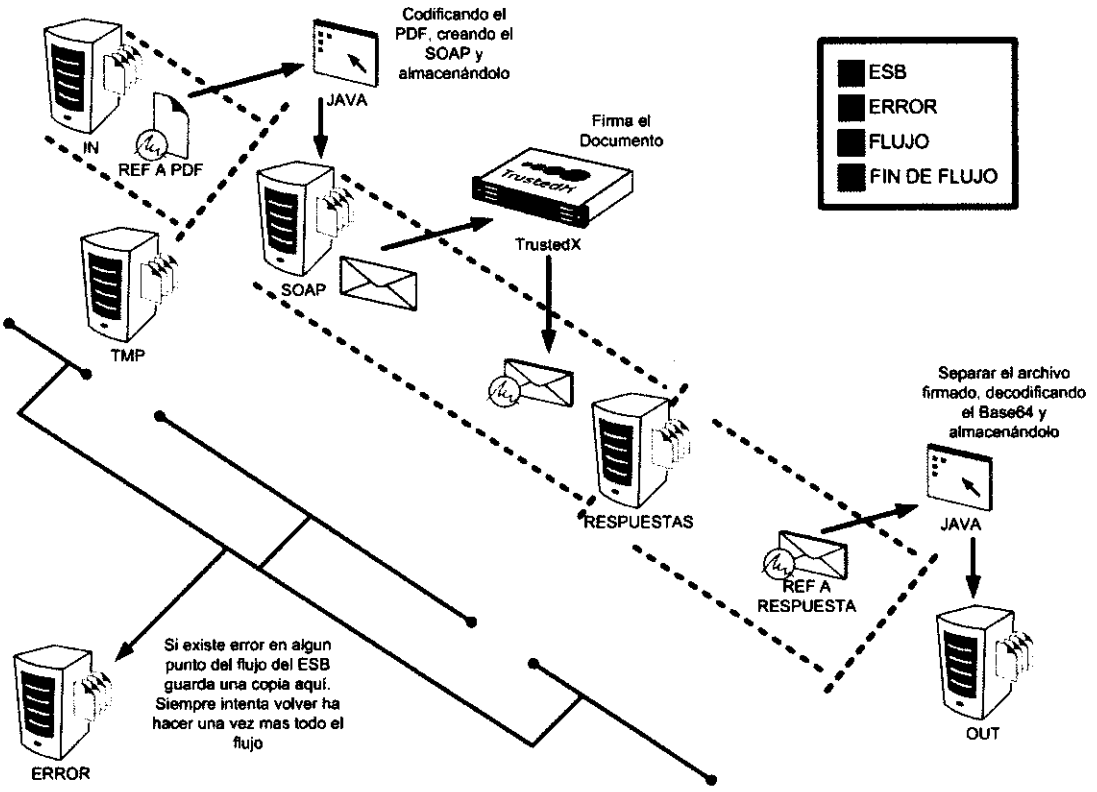


Figura 38 - Oracle Weblogic ESB con TrustedX 2

4.2.2 Diseño

Como se puede observar en la Figura 39, se han implementado solo dos casos de uso el de firma de documentos PDF y el de verificación de firmas PDF. Aunque solo se hayan implementado estos dos, el patrón a seguir sería francamente parecido para cualquier otro servicio.

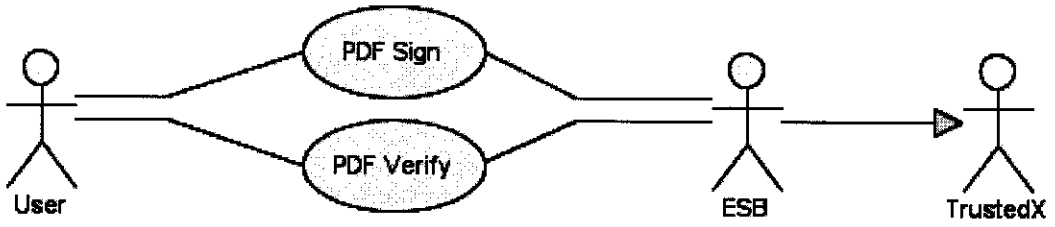


Figura 39 – Diagrama de casos de uso de Oracle Weblogic ESB con TrustedX

4.2.2.1 Firmar PDF

Flujo básico: Firmar PDF

USER	ESB	TrustedX
1. User coloca un PDF para firmar en la carpeta de entrada.		
	2. El ESB que cada cierto tiempo mira si alguien ha colocado archivos en esa carpeta lo toma, crea el SOAP necesario para enviárselo a TrustedX y lo envía a firmar.	
		3. TrustedX firma el PDF y devuelve la respuesta SOAP.
	4. El ESB verifica que todo ha ido bien, extrae el PDF firmado del mensaje SOAP y lo guarda en la carpeta de salida.	
5. User mira en la carpeta de salida y encontrara el fichero PDF con el mismo nombre que el ha puesto, pero esta vez firmado.		

Flujo alternativo: Error

En caso de error en el ESB vuelve a probar de enviar la petición. Si el fallo en una de las clases JAVA externas el flujo podría quedar detenido indefinidamente.

4.2.2.2 Verificar la firma de un PDF

Flujo básico: Firmar PDF

USER	ESB	TrustedX
1. User coloca un PDF firmado para verificar en la carpeta de entrada.		
	2. El ESB que cada cierto tiempo mira si alguien ha colocado archivos en esa carpeta lo toma, crea el SOAP necesario para enviárselo a TrustedX y lo envía a verificar.	
		3. TrustedX valida la firma del PDF y devuelve la respuesta SOAP.
	4. El ESB verifica que todo ha ido bien, extrae los datos de la verificación del mensaje SOAP y guarda un reporte en la carpeta de salida.	
5. User mira en la carpeta de salida y encontrara un fichero de texto en un lenguaje comprensible explicando como ha ido la verificación.		

Flujo alternativo: Error

En caso de error en el ESB vuelve a probar de enviar la petición. Si el fallo en una de las clases JAVA externas el flujo podría quedar detenido indefinidamente.

4.2.3 Implementación y configuración

En este caso la implementación han sido las clases externas que se utilizan para hacer los cambios por streaming. En el ejemplo de firma:

- La clase es llamada por el ESB en un punto concreto del flujo, y uno de sus parámetros es el path donde se encuentra el documento a firmar.
- La clase recoge el documento por streaming, lo codifica, y mientras lo codifica le añade lo necesario para trasformarlo en una petición SOAP.

En el último paso se ha perdido tiempo, pero se ha ganado en consumo de memoria. Ahora el proceso se alarga un tiempo imperceptible y consume mucha menos memoria, por esta razón se pueden enviar mas archivos para firmar.

En cuanto a la configuración del ESB, se ha intentado hacer lo máximo de genérica posible para que se pueda reaprovechar para otros servicios. De esta manera se ha creado un patrón para la creación de otros servicios en un futuro.

4.2.4 Resultados

Los resultados han sido mejor de lo esperados porque con ningún ESB se había conseguido llegar a tener un flujo de peticiones constante contra TrustedX y que no consumiera toda la memoria. Aunque por otra parte esto ha sido gracias al código que se ha insertado dentro del flujo.

Este proceso conlleva unas limitaciones, y es que la lógica que hay en las clases no es lo suficientemente potente para soportar según que servicios se quieran implementar. Y llega un momento que complicar la lógica puede llegar a ser demasiado costoso.

Un posible uso de este escenario seria el de la factura electrónica. Muchas empresas continúan usando factura en papel por lo costoso que es aprender todo el tema de la forma digital y de cómo usarlo. De esta manera que se ha explicado el usuario solo tendría que meter la factura en una carpeta y el ESB la firmaría y la podría enviar por mail a quien tuviera que recibirla.

Capítulo 5
Planificación

La planificación previa de todo proyecto permite analizar la carga de trabajo de cada una de las etapas, prever las necesidades de recursos humanos, de hardware y software en cada una de ellas, y proyectar los hitos intermedios de forma realista. Por otro lado, una planificación adecuada permite realizar un cálculo sobre el coste del proyecto para analizar su viabilidad de cara a su desarrollo y puesta en producción.

5.1 Planificación temporal

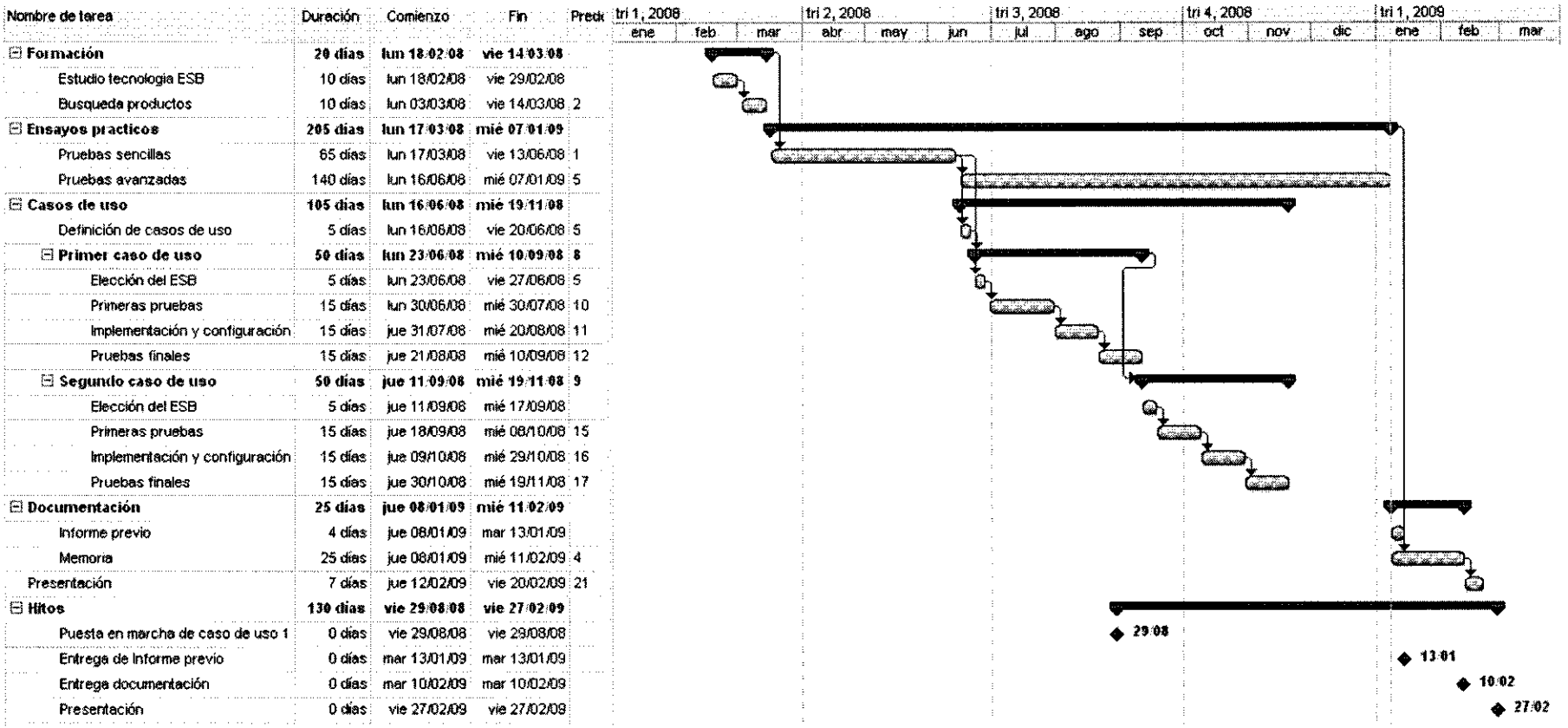
El desarrollo del proyecto fue inicialmente previsto en cuatro fases diferenciadas:

- **Formación:** Durante la primera fase se debía hacer un acercamiento a la tecnología, conocer todos los productos que hay actualmente y entender como funciona. Por otro lado, se tenía que empezar a probar.
- **Ensayos prácticos:** Los conocimientos teóricos adquiridos en la primera fase se complementan con las prácticas y ensayos de esta parte. Durante las prácticas posiblemente se tenga que estudiar otro tipo de tecnologías ligadas a los Enterprise Service Bus.
- **Casos de uso:** Mientras se siguen haciendo ensayos prácticos se diseñarán dos casos de uso complejos donde se demostrara todo lo que se ha aprendido en los dos puntos anteriores.
- **Documentación:** La fase de documentación debería empezar con el prototipo finalizado con el objetivo de documentar todo el proceso de desarrollo del mismo.

Cabe destacar que el desarrollo del proyecto contaba con tres hitos intermedios a cumplir para su correcto desarrollo. Dos de estos hitos estaban relacionados con los plazos de entrega finales definidos para el proyecto: la presentación final del proyecto (27/02/09) y la entrega del informe previo (13/01/09). Por otro lado se propuso un hito para poner en producción el primer caso de uso para el (29/08/08).

La duración real del proyecto ha sido bastante elevada puesto que se ha compaginado el PFC con un trabajo necesario dentro del proyecto de I+D Segur@ lo que ha implicado un mayor esfuerzo sobre todo en formación, ensayos prácticos y especificación y diseño de la aplicación. Al tratarse este de un proyecto 100% I+D se debían adquirir nuevos conocimientos, realizar un estudio del estado del arte de la tecnología existente y posteriormente realizar unos casos de uso complejos.

Horas dedicadas	
Tarea	**Horas**
Formación	80
Ensayos prácticos	600
Casos de uso	420
Documentación	200
TOTAL	**1.300 horas**



Nombre de tarea	Duración	Comienzo	Fin	Predec
Formación	**20 días**	**lun 18/02/08**	**vie 14/03/08**	
Estudio tecnologia ESB	10 días	lun 18/02/08	vie 29/02/08	
Busqueda productos	10 días	lun 03/03/08	vie 14/03/08	2
Ensayos practicos	**205 días**	**lun 17/03/08**	**mié 07/01/09**	
Pruebas sencillas	65 días	lun 17/03/08	vie 13/06/08	1
Pruebas avanzadas	140 días	lun 16/06/08	mié 07/01/09	5
Casos de uso	**105 días**	**lun 16/06/08**	**mié 19/11/08**	
Definición de casos de uso	5 días	lun 16/06/08	vie 20/06/08	5
Primer caso de uso	**50 días**	**lun 23/06/08**	**mié 10/09/08**	**8**
Elección del ESB	5 días	lun 23/06/08	vie 27/06/08	5
Primeras pruebas	15 días	lun 30/06/08	mié 30/07/08	10
Implementación y configuración	15 días	jue 31/07/08	mié 20/08/08	11
Pruebas finales	15 días	jue 21/08/08	mié 10/09/08	12
Segundo caso de uso	**50 días**	**jue 11/09/08**	**mié 19/11/08**	**9**
Elección del ESB	5 días	jue 11/09/08	mié 17/09/08	
Primeras pruebas	15 días	jue 18/09/08	mié 08/10/08	15
Implementación y configuración	15 días	jue 09/10/08	mié 29/10/08	16
Pruebas finales	15 días	jue 30/10/08	mié 19/11/08	17
Documentación	**25 días**	**jue 08/01/09**	**mié 11/02/09**	
Informe previo	4 días	jue 08/01/09	mar 13/01/09	
Memoria	25 días	jue 08/01/09	mié 11/02/09	4
Presentación	7 días	jue 12/02/09	vie 20/02/09	21
Hitos	**130 días**	**vie 29/08/08**	**vie 27/02/09**	
Puesta en marcha de caso de uso 1	0 días	vie 29/08/08	vie 29/08/08	
Entrega de Informe previo	0 días	mar 13/01/09	mar 13/01/09	
Entrega documentación	0 días	mar 10/02/09	mar 10/02/09	
Presentación	0 días	vie 27/02/09	vie 27/02/09	

Figura 40 – Planificación del proyecto

5.2 Planificación económica

Por otro lado hay que contabilizar el coste económico de la realización del proyecto. Para realizar este cálculo hay que considerar un desglose de todos los elementos que han sido necesarios para su realización, desde los recursos humanos hasta los recursos hardware y software (ordenador, lugar de trabajo, licencias,...). Todos estos aspectos serán analizados a continuación:

Recursos humanos

En el proyecto ha participado un becario en régimen de convenio universidad empresa (beca UPC)[4]. Para calcular su salario es necesario multiplicar el número de horas dedicadas por el salario base:

*1 becario * 1300 horas * 7€/h = 9.100 €*

Además, a este coste se deben añadir el coste adicional que supone la manutención de un empleado económicamente (luz, teléfono, etc.) que esta fijada en un 15% del coste del empleado.

9.100 € + 15% = 10.465 €

10.465 € + 1,147€ gestión del convenio = 10.466,15 €

TOTAL	***10.466*,15 €**

Mobiliario

Hay que considerar el mobiliario que ha sido necesario durante el desarrollo del proyecto. El material de oficina queda definido en el apartado 4 de los elementos comunes del impuesto de sociedades con un coeficiente lineal máximo del 10% durante un periodo máximo de 10 años.

Se considera:

Escritorio y silla de oficina valorados en 500 €.

*500 * 0,1 = 50 €*

TOTAL	**50 €**

[4] Se puede observar que no se ha considerado el trabajo del director del proyecto en el coste de recursos humanos.

Equipos informáticos

El apartado de equipos informáticos debe desglosarse en dos apartados: el equipo necesario para el desarrollo y el equipo necesario para el despliegue.

Referente al primer aspecto se ha utilizado un equipo Dell precisión 370 valorado en 1.650€ propiedad de la empresa (comprado en 2005). Teniendo en cuenta su antigüedad hay que calcular el coste de la amortización de dicho equipo mediante la tabla de coeficientes de amortización del apartado 6 del impuesto de sociedades (elementos comunes). Este coeficiente fija el coeficiente lineal máximo en un 25% durante un periodo máximo de 8 años para los equipos para el tratamiento de la información.

Año	Amortización	Valor
1	0,25 * 1.650 = 412,50 €	1650 – 412,5 = 1.237,50 €
2	0,25 * 1.650 = 412,50 €	1.237,5 – 412,5 = 825,00 €
3	0,25 * 1.650 = 412,50 €	825,00 – 412,5= 412,50 €
4	0,25 * 1.650 = 412,50 €	412,50 – 412,50 = 0€

El equipo de trabajo se encontraba en el cuarto año totalmente amortizado. Por otro lado es necesario un servidor de aplicaciones para poder desplegar el prototipo. Para este propósito se ha elegido un servidor Dell Power Edge II valorado en 1.199 €.[5]

1.199 € + 0€ = 1.199 €

TOTAL	***1.199 €***

Sistemas y programas

Para el desarrollo de la aplicación no ha sido necesario adquirir ninguna licencia puesto que las soluciones utilizadas eran en su totalidad software de libre distribución o software propietario de la empresa.

Por otro lado hay que considerar que pese a que la empresa ya disponía de los sistemas operativos instalados y de las suites ofimáticas hay que considerar también el coste de amortización de éstos. La tabla de coeficientes de amortización del impuesto de sociedades, apartado 7 de los elementos comunes especifica un coeficiente lineal máximo del 33% durante un periodo máximo de 6 años para los sistemas y programas informáticos. Hay que tener en cuenta que las licencias corporativas se renuevan anualmente por lo que la amortización puede realizarse a un solo año.

Se consideran:

Licencia Microsoft Windows XP Corporativo valorada en 150 €.

Licencia de Microsoft Office 2003 Profesional valorada en 300 €.

*(150 + 300) * 0,33 = 148,5 €*

TOTAL	**148,5 €**

[5] Destacar que este servidor servirá para alojar más de una aplicación por lo que finalmente este coste se verá distribuido.

El coste total final de todos los recursos necesarios se puede observar en la siguiente tabla:

Coste general	
Concepto	**Coste (€)**
Recursos humanos	10.466,15
Equipos informáticos	1.199
Sistemas y programas	148,5
Mobiliario	50
TOTAL	**11.863,65€**

Cabe destacar que gracias al ESB se han podido integrar varias aplicaciones desarrolladas en otros proyectos de manera muy sencilla, prácticamente sin tener que desarrollar código específico y con un proceso de aprendizaje mínimo. Esto ha hecho que el gasto sea mínimo, aunque no se ha llegado a cuantificar tratándose de un proyecto I+D.

También hay que destacar que si se quisiera poner el producto en producción habría que pagar las licencias requeridas.

Capítulo 6
Conclusiones

La realización completa de este proyecto ha permitido estudiar en profundidad el estado actual de la tecnología de los Enterprise Service Bus. Este estudio ha permitido observar que esta tecnología en particular se encuentra aún en una fase incipiente pero con unos objetivos muy ambiciosos. Esto ha permitido ver que el trabajo en este área es aún largo pero que puede ofrecer a los usuarios grandes posibilidades en el futuro.

Los principales problemas que se han encontrado con la tecnología están enfocados en el consumo de memoria. El escenario que se quiere montar tiene que mover archivos de un tamaño considerable y los ESB no pueden hacer todo por streaming para no consumir tanta memoria.

Se han escogido dos ESB, uno comercial y otro Open Source, MULE y Weblogic de Oracle respectivamente. Son los que cumplían el mayor número de requisitos para los escenarios que se han montado, y como se puede ver en la comparativa han sido los que destacan en más parámetros.

Otra de las cosas que se han encontrado sumamente importante es el papel de las comunidades dentro de los proyectos Open Source. Durante el documento se ha mencionado esta misma idea, y es que hay veces que en los foros de la comunidad encuentras muchísima más información que en la propia documentación del producto. Nos atreveríamos a decir que actualmente un producto joven sin comunidad tiene que tener las ideas muy claras y un buen equipo de testing para llegar a lo más alto.

A nivel personal, cabe destacar que era la primera vez que hacia una comparativa de productos empresariales a gran escala y no me ha sido tan sencilla como esperaba. Es complicado tener que definir unos parámetros comparativos generales que muestren la calidad de un producto cuando no hay ningún estándar definido.

Un aspecto interesante y que hasta ahora no conocía es la realidad del mundo empresarial, exactamente de las empresas Open Source. Me sorprendió que detrás de cada empresa Open Source hubiera una o mas empresas poderosas subvencionando las investigaciones para después enriquecerse de los conocimientos. Cada vez que miraba los integrantes de un proyecto Open Source de los que he probado encontraba investigadores de empresas que tienen ESB comerciales.

El trabajo en un entorno I+D ha permitido involucrarse en tecnologías novedosas, de alto nivel técnico y con grandes posibilidades de expansión futuras. Estos aspectos han tenido especial relevancia en el aprendizaje y han sido plenamente gratificantes. Durante la realización del proyecto y tras la conclusión he podido comprobar como los ESB son mas importantes de lo que pensaba y que se deberían implantar en mas entornos.

A nivel personal hay que destacar que la mayor parte de los conocimientos necesarios han sido adquiridos durante el desarrollo del proyecto puesto que las tecnologías utilizadas han sido siempre punteras y en continuo desarrollo. Esto ha comportado que se hayan podido aprovechar los conocimientos teóricos obtenidos durante la carrera relativos a la especificación y diseño del sistema y al análisis de algoritmos. En el aspecto técnico se han aprovechado los conocimientos del lenguaje Java para realizar las partes de implementación del proyecto.

Es importante el hecho de que este proyecto de fin de carrera se ha realizado en la empresa *Safelayer Secure Communications S.A.* con la colaboración del departamento DMAG de la UPC. *Safelayer* se destaca como empresa puntera a nivel nacional y con reconocimiento internacional en el ámbito de la seguridad y la tecnología PKI. El hecho de

haber realizado el proyecto en un entorno corporativo ha permitido tener una visión más amplia sobre como se planifica y desarrolla un proyecto en un entorno profesional y ha permitido adquirir conocimientos prácticos útiles para el futuro. Además de los aspectos de procedimiento, este entorno ha permitido contar con un equipo de profesionales especializados en las tecnologías de último nivel a los que se ha podido acudir en caso de necesidad. Finalmente hay que reconocer la importancia del equipo de trabajo puesto que la proximidad ha permitido la adaptación en ciertos casos del producto para resolver aspectos específicos y la mejora de calidad de dicho producto.

Probablemente, uno de los aspectos más enriquecedores ha sido el trabajo en equipo que ha sido necesario realizar puesto que el proyecto se encontraba vinculado a otros PFC y ha sido necesario tomar decisiones conjuntas, que en el caso de una aplicación individual podrían haber tomado otros caminos. Este aspecto ha permitido observar y aprender la diferencia existente entre el desarrollo de proyectos independientes y de proyectos que se entrelazan.

6.1 Usos futuros

6.1.1 Primer caso de uso

Las mejoras que se esperan en este caso están enfocadas a ampliar el número de elementos que están integrados con el ESB. El próximo en integrarse de una manera semejante a lo que se ha hecho con la base de datos, será un razonador. Algunas de las funciones más importantes de un razonador son:

- Interpretar reglas
- Determinar la consistencia de ontologías
- Resuelve consultas SPARQL

Se desea realizar un grupo de web service y dotarles de funcionalidades con el razonador. De esta manera cualquier aplicación que contacte con el ESB podrá acceder de manera sencilla a sus servicios.

Otra de las mejoras que se pretende realizar es la actualización de todas las librerías que se han usado para realizar el conjunto de clases que se comunican con la base de datos. Durante este tiempo Jena ha publicado nuevas releases y por falta de tiempo no se han podido incorporar. Seria interesante ojear las novedades y ver si estas desembocan en nuevos servicios para nuestra base de conocimiento.

Por último se quiere probar la nueva versión del ESB Open Source utilizado, MULE. Por la misma razón que con Jena no se ha tenido tiempo de probar esta nueva release. En este caso hay que tener mas cuidado porque cabe la posibilidad de que no sea compatible con el escenario que se ha montado.

6.1.2 Segundo caso de uso

La primera acción que se quiere realizar sobre este entorno es mejorar la solidez del mismo controlando mejor todos los errores que pueden suceder. Ahora se controlan, pero hasta cierto punto.

La segunda tarea a realizar es ampliar el abanico de servicios de TrustedX que se ofrecen de esta manera.

Acrónimos

EAI	Enterprise Application Integration
EDA	Event Driven Architecture
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JMS	Java Messages Service
MOM	Middleware Oriented Messages
PDF	Portable Document Format
PKI	Public Key Infrastructure
POJO	Plain Old Java Object
RDF	Resource Description Framework
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URN	Uniform Resource Name
XML	Extensible Markup Language

Referencias

[1] http://simile.mit.edu/reports/stores/

[2] http://www4.wiwiss.fu-berlin.de/benchmarks-200801/

[3] http://esbsite.org/

[4] http://www.mulesource.org/display/MULE/Home

[5] http://servicemix.apache.org/home.html

[6] http://synapse.apache.org/

[7] http://wso2.org/projects/esb/java

[8] http://petals.ow2.org/

[9] http://ksb.kuali.org/

[10] https://open-esb.dev.java.net/glassfishesb/

[11] http://www.oracle.com/appserver/weblogic/weblogic-suite.html

[12] http://www-01.ibm.com/software/integration/wbimessagebroker/

[13] http://www.alzado.org/articulo.php?id_art=30

[14] H. Pujol, "Propuesta de una arquitectura de confianza semántica para entornos ambientales seguros", Proyecto Final de Carrera, ETSETB, UPC. Pendiente de publicación.

[15] D. Campos, "Sistemas de gestión de la identidad centrados en el usuario", Proyecto Final de Carrera, FIB, UPC. Enero 2009.

[16] V. Martínez, "Tecnologías de acceso a atributos de identidad en entornos user-centric", Proyecto Final de Carrera, FIB, UPC. Enero 2009.

[17] M.A. Pareja, "Diseño e implementación de un razonador semántico sobre seguridad y confianza en PKI", Proyecto Final de Carrera, FIB, UPC. Enero 2009.

Anexo A

Archivo WSDL de la Interficie Query Sparql

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    xmlns:tns="urn:sparql.safelayer"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:st="urn:types.safelayer"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:sparql.safelayer">
  <wsdl:service name="QuerySoapService">
        <wsdl:port name="QuerySoapPort" binding="tns:QuerySoapBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </wsdl:port>
  </wsdl:service>
  <wsdl:types>
    <xs:schema targetNamespace="urn:sparql.safelayer">
      <xs:import namespace="urn:types.safelayer"
            schemaLocation="sparql-protocol-types.xsd" />
        </xs:schema>
  </wsdl:types>
  <wsdl:message name="queryRequest">
    <wsdl:part name="query" element="st:query-request"/>
  </wsdl:message>
  <wsdl:message name="queryResponse">
    <wsdl:part name="response" element="st:query-result"/>
  </wsdl:message>
  <wsdl:message name="malformedQueryFault">
      <wsdl:part name="malformedQueryFaultPart" element="st:malformed-query"/>
  </wsdl:message>
  <wsdl:message name="queryRequestRefusedFault">
      <wsdl:part name="queryRequestRefusedFaultPart" element="st:query-request-refused"/>
  </wsdl:message>
  <wsdl:portType name="SparqlQueryInterface">
    <wsdl:operation name="query">
      <wsdl:input  message="tns:queryRequest"/>
      <wsdl:output message="tns:queryResponse"/>
      <wsdl:fault  message="tns:malformedQueryFault"  name="malformedQueryFault" />
      <wsdl:fault  message="tns:queryRequestRefusedFault" name="queryRequestRefusedFault" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="QuerySoapBinding" type="tns:SparqlQueryInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="query">
    <soap:operation style="document" />
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="malformedQueryFault">
        <soap:fault name="malformedQueryFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="queryRequestRefusedFault">
        <soap:fault name="queryRequestRefusedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Archivo WSDL de la Interficie Sparql

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    xmlns:tns="urn:sparqlprotocol.safelayer"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:st="urn:types.safelayer"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:sparqlprotocol.safelayer"
    >
  <wsdl:service name="SparqlSoapService">
                    <wsdl:port name="SparqlInterface" binding="tns:SoapBinding">
                            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
                    </wsdl:port>
  </wsdl:service>
  <wsdl:types>
    <xs:schema targetNamespace="urn:sparqlprotocol.safelayer">
      <xs:import namespace="urn:types.safelayer"
                schemaLocation="sparql-protocol-types.xsd" />
          </xs:schema>
  </wsdl:types>

  <wsdl:message name="insertRequest">
    <wsdl:part name="insert" element="st:insert-request"/>
  </wsdl:message>

  <wsdl:message name="insertResponse">
    <wsdl:part name="response" element="st:insert-result"/>
  </wsdl:message>

  <wsdl:message name="deleteRequest">
    <wsdl:part name="delete" element="st:delete-request"/>
  </wsdl:message>

  <wsdl:message name="deleteResponse">
    <wsdl:part name="response" element="st:delete-result"/>
  </wsdl:message>

  <wsdl:message name="getrdfRequest">
    <wsdl:part name="getrdf" element="st:getrdf-request"/>
  </wsdl:message>

  <wsdl:message name="getrdfResponse">
    <wsdl:part name="response" element="st:getrdf-result"/>
  </wsdl:message>

  <wsdl:message name="updateRequest">
    <wsdl:part name="update" element="st:update-request"/>
  </wsdl:message>

  <wsdl:message name="updateResponse">
    <wsdl:part name="response" element="st:update-result"/>
  </wsdl:message>

  <wsdl:portType name="SparqlInterface">
    <wsdl:operation name="insert">
      <wsdl:input  message="tns:insertRequest"/>
      <wsdl:output message="tns:insertResponse"/>
    </wsdl:operation>
          <wsdl:operation name="delete">
      <wsdl:input  message="tns:deleteRequest"/>
      <wsdl:output message="tns:deleteResponse"/>
    </wsdl:operation>
          <wsdl:operation name="getrdf">
```

```
      <wsdl:input  message="tns:getrdfRequest"/>
      <wsdl:output message="tns:getrdfResponse"/>
    </wsdl:operation>
          <wsdl:operation name="update">
      <wsdl:input  message="tns:updateRequest"/>
      <wsdl:output message="tns:updateResponse"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="SoapBinding" type="tns:SparqlInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="insert">
    <soap:operation style="document" />
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
            <wsdl:operation name="delete">
    <soap:operation style="document" />
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
            <wsdl:operation name="getrdf">
    <soap:operation style="document" />
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
            <wsdl:operation name="update">
    <soap:operation style="document" />
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Archivo WSDL de la Interficie RecQuery Sparql

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    xmlns:tns="urn:recsparql.safelayer"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:st="urn:rectypes.safelayer"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:recsparql.safelayer">
  <wsdl:service name="RecQuerySoapService">
          <wsdl:port name="RecQuerySoapPort" binding="tns:RecQuerySoapBinding">
                  <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
          </wsdl:port>
  </wsdl:service>
  <wsdl:types>
    <xs:schema targetNamespace="urn:recsparql.safelayer">
      <xs:import namespace="urn:rectypes.safelayer"
                schemaLocation="sparql-rec-types.xsd" />
        </xs:schema>
  </wsdl:types>
  <wsdl:message name="recqueryRequest">
    <wsdl:part name="query" element="st:recquery-request"/>
  </wsdl:message>
  <wsdl:message name="recqueryResponse">
    <wsdl:part name="response" element="st:recquery-result"/>
  </wsdl:message>
  <wsdl:portType name="SparqlRecQueryInterface">
    <wsdl:operation name="recquery">
      <wsdl:input  message="tns:recqueryRequest"/>
      <wsdl:output message="tns:recqueryResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="RecQuerySoapBinding" type="tns:SparqlRecQueryInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="recquery">
    <soap:operation style="document" />
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```