¿Qué ofrece Autentia?

autentia
real business solutions

Somos su empresa de

Soporte a Desarrollo Informático

Ese apoyo que siempre quiso tener ….

- **Desarrollo de componentes y proyectos a medida.**
- **Auditoría de código y recomendaciones de mejora.**
- **Arranque de proyectos basados en nuevas tecnologías.**
- **Curso de Formación**

  Dirección de Proyectos Informáticos.

  Gestión eficaz del Tiempo.

  Arquitecturas de desarrollo Web: Web, J2EE, SOA, WebServices, BPM, etc.

  Java/ J2EE a todos los niveles: JSPs, Servlets, EJBs, JMS, JNI, etc.

  Análisis y diseño orientado a objeto.

  UML y patrones de diseño.

  Buenas prácticas en el desarrollo de aplicaciones

  Técnicas avanzadas: Lucene, Hibernate, Spring, JSF, Struts, etc.

Nuestra mejor referencia son los conocimientos que

compartimos en nuestro web

www.adictosaltrabajo.com

Decenas de entidades cuentan ya con nosotros

Para más información visítenos en www.autentia.com

Tel. 91 675 33 06 - info@autentia.com

ESB

Cual es el problema

Una organización tiene sistemas distintos.

Necesita integrarlos.

SOA cada día más presente.

Conexión 1 a 1 = caro y complejo.

Solución: Bus integración.

Cual es el problema

¿Qué es un ESB?

Ejemplo: Agencia de Viajes → vuelo + hotel

¿Qué es un ESB?

¿Qué es un ESB?

- Permite la integración de diferentes aplicaciones o servicios preexistentes.

- Administra la intercomunicación entre ellos.

Componentes de un ESB

- Services: dos o más servicios a integrar.
- Bus: Actúa como mediador entre los diferentes servicios. Administra el intercambio y la transformación de mensajes entre ellos.
- Endpoints: puntos de enlace entre los servicios y el Bus de mensajes.

¿Por que utilizar un ESB?

Permite una rápida integración de servicios.

¿Por que utilizar un ESB?

Sólo es necesario definir como se integra cada servicio con el bus.

Permite obtener resultados más confiables.

¿Por que utilizar un ESB?

La integración de servicios al Bus se realiza mediante configuración.

La integración de servicios se basa en estándares.

¿Por que utilizar un ESB?

La plataforma provee las herramientas para realizar la integración.

Funcionalidades de un ESB

Ubicación transparente: **desacopla al consumidor de servicios de la ubicación del mismo.**

Funcionalidades de un ESB

Conversión de protocolos de transporte: integración de aplicaciones con diferentes protocolos de transporte. Http → JMS, FTP → ficheros por lotes, SMTP → TCP ...

Funcionalidades de un ESB

Transformación de mensajes: adaptación de los mensajes al formato requerido por el servicio.

Funcionalidades de un ESB

Enrutamiento de mensajes: determinar el destino del mensaje.

Funcionalidades de un ESB

Modificación de mensajes

Funcionalidades de un ESB

Seguridad: proporciona funcionalidad para la autenticación, autorización y encriptación.

Funcionalidades de un ESB

Supervisión y gestión del entorno: permite monitorizar en tiempo de ejecución el flujo de mensajes.

Mule

www.autentia.com

¿Por qué Mule?

www.autentia.com

Criterio	Mule	ServiceMix	Open ESB	Synapse	PEtALS
Soporte	+	+	+/-	+	+
Calidad de la documentación	+	+/-	+	+	+/-
Visibilidad en el mercado	++	+	+/-	+/-	+/-
Comunidad	++	+	+/-	+	+
Flexibilidad	++	+	+/-	++	+
Soporte de protocolos y conectividad	+	+	+/-	+/-	+
Integración con otros productos OpenSource	++	++	+/-	+	+
IDE	+	+	++	+/-	+

Ref. Open Source ESBs in action (Manning)

Modos de ejecución

Stand-Alone

Mediante línea de comandos especificando el fichero de configuración.

Ejemplo: /mule2/bin/mule -config ejemploMule/config/`mule-config.xml`

Servlet engine

Configurar en el web.xml del servidor un context-param con los ficheros de configuración de arranque de Mule.

Añadir un listener que arranque Mule ESB.

Mule - elementos

Component(componente): Contiene la “lógica de negocio”. (Por ejemplo, un bean de Sping, un servicio REST, un POJO, etc.)

Transport(transporte): Maneja la conectividad con una determinada tecnología o aplicación (por ejemplo, JMS, SAP, FTP, etc.)

Transformer(transformadores) :Transforma los datos al formato esperado por el siguiente componente.

Inbound Router(router de entrada): Determina qué hacer con el mensaje entrante antes de ser enviado al servicio.

Outbound Router(router de salida):Determina donde debe ser enviado un mensaje tras su procesamiento por parte del servicio.

Mule – flujo básico

1.Un transport recibe un mensaje. (Por ejemplo, un mensaje se ha puesto en una cola JMS donde el transport está escuchando.)

2. Antes de que el mensaje se envíe al router de entrada, sufre la primera transformación (si es necesario) al formato requerido.

3. El mensaje es procesado por el router de entrada. Por ejemplo, podríamos tener un "consumo selectivo", que sólo acepta los mensajes que envían las aplicaciones en las confiamos.

4. Tras pasar por el router de entrada, el mensaje se envía al componente, que aplica su lógica de negocio.

5. Después de que el servicio lo procese, la respuesta (mensaje) se envía al router de salida. Este determina dónde enviarlo. Podríamos, por ejemplo, dividir este mensaje en varias partes y enviarlos a diferentes destinos.

6. Finalmente, podemos transformar el mensaje una vez más para adaptarlo a las necesidades del receptor.

Arquitectura Mule

Channel:canal de comunicación de las aplicaciones con Mule.

Connector:

- Message Reciever: receptor de los mensajes del canal.
- Transformer: adapta la información al servicio.

Inbound Router: determina que hacer con los mensajes recibidos.

Component: implementa la lógica de integración

Outbound Router: determina a donde deben enviarse los mensajes salientes tras ser procesados por el componente.

Message Dispacher: define como deben ser enviados los mensajes al canal de salida.

Configurando Mule

Configuración de Mule → Fichero XML.

Un servicio es un sencillo componente que especifica:

- sobre que canal escucha (inbound router),
- que metodo invoca (Component).
- hacia que canal publica (outbound router).

Ejemplo – Agencia de viajes

Mule endpoints

Mule endpoints

Establece la manera en la que una aplicación se conecta a un canal.

Los canales, conectores, emisores y receptores trabajan juntos para lograrlo.

Gracias a los namespaces específicos de transporte esta configuración se realiza de forma sencilla.

Ejemplos:

```
<jms:inbound-endpoint queue="prueba.queue" />
<jms:outbound-endpoint topic="prueba.topic" />
<file:inbound-endpoint name="example-in" path="example/in" />
<file:outbound-endpoint name="example-out" path="example/out" />
```

Ya sabemos como Mule interactua con los mensajes. Veamos que puede hacer con ellos.

Mule Transformers

Una vez recibido el mensaje debe ser transformado para adaparse al formato requerido por el servicio.

Esto se realiza en dos pasos:

Definición de las transformaciones.

Transformaciones genéricas:

Mule aplica sus propias transformaciones basadas en el tipo de transporte empleado.

```
<jms:jmsmessage-to-object-transformer name="JMSToStringTransformer" />
```

```
<xml:xslt-transformer name="XSLT" xsl-file=="prueba.xslt" />
```

Ejemplo de transformaciones automáticas realizadas por Mule para mensajes JMS.

Objeto Java	Mensaje JMS
java.lang.String	javax.jms.TextMessage
byte[]	javax.jms.BytesMessage
java.util.Map	javax.jms.MapMessage
java.io.InputStream	javax.jms.StreamMessage
java.lang.Object	javax.jms.ObjectMessage

Mule Transformers

www.autentia.com

Transformaciones de usuario:

Estas transformaciones pueden sobre-escribir las genéricas o ser aplicadas conjuntamente.

En este caso las transformaciones genéricas no son ejecutadas por defecto.

```
<custom-transformer name="pruebaTransformer" class="autentia.esb.prueba.BigDecimalTransformer" />
```

Clase que implementa el transformador:

```
Package autentia.esb.prueba;

import org.mule.api.transformer.TransformerException;

import org.mule.transformer.AbstractTransformer;

public class BigDecimalTransformer extends AbstractTransformer {

	protected Object doTransform(Object src, String encoding) throws TransformerException {

		if(src instanceof String) {

			return new BigDecimal(src);

		}

		return res;

	}

}
```

Mule Transformers

www.autentia.com

Asociación de los transformadores a los endpoints.

Se realiza sobre la propia definición de los endpoints.

Ejemplo:

```
<jms:inbound-endpoint queue="query.response" />
      <transformer ref="JMSToStringTransformer" />
      <transformer ref="XSLT" />
</jms:inbound-endpoint >
```

Mule Routers

Inbound Router

Determina como son recibidos los mensaje por el componente.

Configurable mediante filtros, permite establecer la estrategia de evaluación del mensaje.

Ejemplo sencillo

```
<inbound>
    <inbound-endpoint address="cxf:http://localhost:63081/agencia/hotel/hotelSearch" />
</inbound>
```

Nombre del router	descripción
Idempotent receiver	Solo se reciben mensajes con identificador único
Aggregator	Combina en un mensaje dos o más mensajes
Resequencer	Cambiar el orden de los mensajes
Selective consumer	Evalua si enviar o no el mensaje
Wiretap router	Permite redireccionar mensajes
Forwarding consumer	Envía el mensaje al outbound sin pasar por el componente

Mule routers

Outbound Router

Determina donde serán enviados los mensajes tras su procesamiento.

```
<outbound>
    <pass-through-router>
        <vm:outbound-endpoint path="aggregatorQueue" />
    </pass-through-router>
</outbound>
```

Nombre del router	descripción
Filtering outbound router	Enruta basándose en el contenido del mensaje
Recipient list	Permite enrutar hacia múltiples endpoints
Multicasting router	Enviar el mismo mensaje a múltiples endpoints
Chaining router	Encadena varios endpoints
Message splitter	Divide en varios mensajes y envía por separado
Exception-based router	Prueba sobre varios endpoints
List message splitter	Separa mensajes de una lista

Mule Connectors

Se incluyen en el fichero de configuración de Mule.

Permiten a Mule conectarse con diversas tecnologías.

- Ficheros
- JMS (queue y topic)
- JDBC
- mail (POP3 y SMTP)
- FTP
- EJBs (RMI y capa WS)
- BPM
- HTTP
- IMAP
- QUARTZ
- RMI
- SOAP

Ejemplo:

```
<smtp:gmail-connector name="emailConnector" />
```

Component

Es invocado cuando un mensaje recibido por el “inbound router” pasa todos los filtros.

Este componente por defecto es un POJO sin dependencias con Mule.

Puede ser gestionado por Spring o implementado en otras tecnologías como Groovy, REST, etc.

Una vez terminado el proceso el resultado que devuelve al outbound router es el objeto respuesta.

En caso de devolver null el proceso se detiene.

Configuración en Mule:

```
<component class="com.autentia.ejemplo.PruebaComponentImpl" />
```

¿Como sabe Mule que método debe invocar?

- Dejar que Mule decida...
- Especificar el nombre del método en la configuración de Mule.

```
<component class="com.autentia.ejemplo.PruebaComponentImpl" methods="process"/>
```

- Mediante una interfaz que defina el “entrypoint” de la clase, como por ejemplo:

  Callable que obliga a implementar el método onCall.

Conclusión

- Mule recibe mensajes escuchando en un canal.

- Un *message receiver* recibe el mensaje utilizando la tecnología específica requerida por el canal.

- El transformador de entrada es invocado.

- Antes de llamar al componente, el mensaje pasa por el *inbound router* que decide si será procesado el mensaje.

- El componente lo procesa.

- El resultado pasa al *outbound router* que determina donde será enviado.

- El transformador de salida es invocado.

- El *message dispacher* deja el mensaje en el canal de salida.