

Buenas Prácticas de Desarrollo en OSB



Autor: Oracle Consulting Services (OCS)

Fecha Creación: 20 Octubre de 2010

Fecha Modificación: 17 Abril de 2011

Versión:  Draft 1.1

**Contenido**

[1 Introducción 3](#_Toc528341878)

[2 Objetivo 3](#_Toc528341879)

[3 Alcance 3](#_Toc528341880)

[4 Fuentes 3](#_Toc528341881)

[5 Supuestos 3](#_Toc528341882)

[6 Proceso de Diseño de Servicio 3](#_Toc528341883)

[6.1 ¿Cuál es el proceso de diseño de servicio? 3](#_Toc528341884)

[6.2 ¿Cuál es el proceso de diseño de servicio? 4](#_Toc528341885)

[6.3 Buenas Prácticas de diseño y desarrollo OSB 5](#_Toc528341886)

[**1** **Organización de componentes y estándares de nomenclatura** 5](#_Toc528341887)

[**2** **Diseño de proxies** 9](#_Toc528341888)

[**3** **Manejo de Errores** 16](#_Toc528341889)

[**4** **Buenas prácticas de Logging** 17](#_Toc528341890)

[**5** **Patrón de Manejo de Excepciones** 18](#_Toc528341891)

[**6** **Tuning de Proxies en Runtime** 20](#_Toc528341892)

[**7** **Uso de Work Managers** 21](#_Toc528341893)

[**8** **Xquery Tuning - Transformaciones** 22](#_Toc528341894)

[**9** **Tuning de Transporte en (WLS y OSB)** 25](#_Toc528341895)

[**10** **HTTP** 25](#_Toc528341896)

[7 Histórico de Cambios 27](#_Toc528341897)

# Introducción

Dentro de un marco SOA, Oracle Service Bus (OSB) proporciona una variedad de servicios, como ser conectividad, enrutamiento, mediación, gestión y también capacidades de orquestación de procesos.

OSB persigue la filosofía de ser un intermediario de alto rendimiento entre dos o más aplicaciones. Dada la diversidad de tamaño y funcionalidad en las implementaciones SOA, las aplicaciones OSB son sujetas a una gran variedad de patrones de uso y de requisitos de QoS.

# Objetivo

Esta guía tiene como objetivo proporcionar directrices generales de diseño, tuning y mejores prácticas en OSB para un óptimo rendimiento, escalabilidad y fiabilidad ante diferentes escenarios. Mientras intenta cubrir la mayor cantidad de áreas donde el tuning tiene un mayor impacto, este documento no cubre la totalidad de buenas prácticas asociadas a cada uno de los patrones implementados por OSB.

# Alcance

El documento abarca tópicos de diseño orientados a servicios con foco puesto en la integración. Mejores prácticas en Diseño de Proxies, Procesamiento de XML, WSDL, Performance y tuning de un entorno OSB

# Fuentes

* Definitive Guide to SOA Oracle Service Bus, Second Edition.
* Material de la base de conocimiento de Oracle

# Supuestos

Se toma como supuesto que el lector tiene conocimiento avanzado en Java y en procesamiento de XML, Query, Xpath.

El documento no considera aspectos de instalación del OSB, dando por supuesto que el lector tiene conocimientos básicos en el uso de la herramienta.

# Proceso de Diseño de Servicio

## ¿Cuál es el proceso de diseño de servicio?

Dado un caso de uso de integración particular se debe descomponer la solución en una serie de servicios. El conjunto de estos cubrirán los requisitos necesarios para cumplir con los requerimientos propuestos.  
El proceso de diseño de servicios cubre la totalidad de las actividades que se necesitan para el diseño de cada uno de estos servicios.

Por lo general, las siguientes son las entradas necesarias en el proceso de diseño de servicio para cualquier caso de uso de integración.

1. **Requerimientos funcionales:** Las entradas de datos, salidas de datos y el comportamiento requerido del caso de uso en términos de cambios de estado de todos los sistemas afectados en la empresa. Serán listados todos los sistemas involucrados, así como todas las interacciones necesarias entre ellos. La lógica condicional y el flujo de orquestación es también parte de la condición funcional.
2. **Requerimientos no funcionales**: Esto incluirá el volumen esperado, los tiempos de respuesta requeridos, los requisitos de disponibilidad, requisitos de seguridad y calidad del servicio.
3. **Documentación del producto**: Esto incluirá toda la documentación de OSB.
4. **Mejores prácticas y patrones de diseño**: Esto comprenderá todas las mejores prácticas genéricas y los patrones de diseño.
5. **Experiencias previas de solución de problemas de integración**: Esto cubre las experiencias colectivas de los miembros del equipo de diseño.
6. **Repositorio de soluciones existentes**: Esto incluirá la documentación de las soluciones existentes, incluidos los servicios reutilizables.

Al examinar los requisitos no funcionales para un caso de uso, es útil considerar los siguientes criterios SMART para sus necesidades:

* **Específicos** (Specific): sin ambigüedad, utilizando una terminología coherente, simple y en el nivel de detalle apropiado.
* **Medibles** (Measurable): es posible verificar que este requisito se ha cumplido. ¿Qué pruebas se deben realizar o qué criterios se deben cumplir para verificar que el requisito se cumple?
* **Posible** (Attainable): técnicamente viable. ¿Cuál es el criterio profesional de la factibilidad técnica del requerimiento?
* **De realización** (Realizable): realista, dados los recursos. ¿Se tienen los recursos humanos? ¿Tienen la habilidad? ¿Se tiene acceso a la infraestructura de desarrollo necesaria? ¿Se tiene acceso al ambiente de ejecución? ¿Se tiene suficiente tiempo?
* **Rastreable** (Traceable): enlaces desde su concepción a través de su especificación para su posterior diseño, implementación y prueba.

## ¿Cuál es el proceso de diseño de servicio?

Un caso de uso de integración de punta a punta puede ser dividido en uno o más servicios, cada uno de los cuales debe de tener su propio diseño. Será muy importante para el diseñador de un servicio poder clasificar las características del servicio en cada una de estas áreas.

1. **Complejidad**:
   1. Básico - definido como un servicio con una llamada a otro componente.
   2. Compuesto - definido como un servicio con más de una llamada a otros componentes.
2. **Forma de invocación**:
   1. Síncrono - los consumidores de este servicio se bloquearán esperando la respuesta, mientras que el servicio se completa.
   2. Asincrónica - los consumidores de este tipo de servicio no se bloquearán a la espera de la respuesta, mientras que el servicio se completa.
   3. Mixto - este será un servicio con una parte sincrónica seguido por una parte asíncrona.
3. **Forma de intercambio**:
   1. Dos vías - solicitud / respuesta.
   2. Una vía - fire and forget.

Además de las propiedades del servicio descrito anteriormente, los servicios tienen una serie de características que vienen de los requisitos no funcionales:

1. **Volumen**:
   1. Alto - más de 100 llamadas de servicio por segundo
   2. Medio - entre las llamadas de servicio 1 y 100 por segundo
   3. Bajo - menos de 1 llamada de servicio por segundo
2. **QoS** (fiabilidad):
   1. At-Most-Once - la garantía de servicio que se ejecutará no más de una vez. Normalmente esto devuelve un error al consumidor para que pueda controlar el error/reintentos.
   2. At-Least-Once - la garantía de servicio que se ejecutará al menos una vez, lo que permite la posibilidad de duplicados.
   3. Exactly-once - el servicio está garantizado para ser ejecutado una vez y sólo una vez. Esto normalmente utiliza el reconocimiento (ack), reintentos y detección de duplicados y eliminación. Cuando un servicio implica cambios a múltiples sistemas, tenemos dos opciones con respecto al momento de las actualizaciones:
      1. Todos ellos se pueden actualizar en la misma transacción distribuida - todo o nada.
      2. Se puede actualizar de forma independiente el uno del otro - JMS store-and-forward.
3. **Duración**:
   1. Muy corto - entre 0,1 y 10 segundos.
   2. Corto - entre el 1 y 60 segundos
   3. Promedio - entre 30 segundos y 1 hora
   4. Larga - entre 60 segundos y 1 mes

## Buenas Prácticas de diseño y desarrollo OSB

Las siguientes recomendaciones se denominan "mejores prácticas". Puede influir en las decisiones de diseño pero no siempre es posible ponerlas en práctica para todos los patrones de uso y los estándares corporativos. Es por esto, que sólo deben adoptarse cuando sea apropiado.

A continuación se listan tips de diseño y factores a tener en cuenta al momento de modelar un servicio en el OSB. Muchas son aclaraciones de conceptos propios del bus, que permiten tener un panorama enriquecido al momento de elegir que elemento usar.

1. **Organización de componentes y estándares de nomenclatura**

Es esencial que las normas de cómo se organizan y nombran los componentes en OSB sea acordadas la aplicación de estas normas nos permiten:

1. Mantenimiento más fácil
2. Facilidad en resolución de problemas
3. Tener un proceso mejor de creación
4. Tener un proceso mejor de transferencia de conocimiento.

A continuación se sugieren algunas normas de organización de componentes y nomenclatura dentro del OSB.

* **Estructura de Proyectos**

OSB maneja los recursos administrables bajo una estructura jerárquica denominada proyectos y carpetas que puede ser organizada de muchas maneras. Sin embargo, independiente de la estructura de proyectos que se escoja, se sugiere que las carpetas que vayan al final de jerarquía tengan una forma común.

* **Alternativas de organización de proyectos**
  + **Por proyecto:** Se utiliza cuando una necesidad del negocio abre un presupuesto que lleva a un proyecto de desarrollo con un proveedor de desarrollo de servicios que lo implemente. Esto puede llevar a la creación del proyecto OSB según el nombre del proyecto en el que éste está inmerso.
    - Ejemplo:
      * OneEasy
    - Ventajas
      * Independencia y flexibilidad entre los desarrolladores que actualizan los recursos en el Bus
    - Desventajas
      * No fomenta la visión corporativa de servicios en un esquema SOA.
  + **Por canal:** Utilizada cuando los clientes consumidores por canal son los que fijan la organización de los servicios.
    - Ejemplo:
      * Web
    - Ventajas
      * Generalmente los SLA son compartidos permitiendo un tratamiento común.
    - Desventajas
      * Se tiende a manejar proyectos OSB muy grandes.
  + **Por aplicación:** Se utiliza cuando una aplicación en particular requiere un conjunto nuevo de servicios y todos estos se agrupan en el mismo proyecto.
    - Ejemplo:
      * Siebel
    - Ventajas
      * La dependencia directa facilita actividades de debuging y baja interferencia con servicios compartidos.
    - Desventajas
      * No fomenta la visión corporativa de servicios en un esquema SOA.
  + **Por proceso**: Cuando los procesos del negocio son los que levantan los servicios agrupándolos mediante un enfoque “top down” en proyectos SOA - BPM .
    - Ejemplo:
      * Atención a Clientes
    - Ventajas
      * Permite una vista corporativa de servicios que facilita la reutilización
    - Desventajas
      * Complejo implementar
  + **Por contexto**: Cuando grupos de servicios ofrecen soporte en un contexto compartido brindando funcionalidades relacionadas.
    - Ejemplo:
      * Planes
    - Ventajas
      * Mezcla las distintas alternativas expuestas anteriormente brindando los pros de cada una.
    - Desventajas
      * Cuesta organizar y mantener ya que se requiere amplio conocimiento del negocio y Arquitectura.
* **Estructura de carpetas**

Conviene tener los proyectos bajo un standard de organización de subcarpetas. De esta manera se pueden tener los recursos organizados y tener facilidad en la búsqueda de los mismos.

Se presentan a continuación diferentes alternativas:

Alternativa 1:

Carpeta

BusinessServices

ProxyServices

Resources

Alternativa 2:

Carpeta

BusinessServices

ProxyServices

Resources

WSDL

Schemas

XQuery

JCA

Alternativa 3:

Carpeta

BS

PX

WSDL

XSD

XQuery

La carpeta BusinesServices (a veces llamada BS), alberga los servicios de negocio. De la misma forma para los Proxies (PX), ésta carpeta alberga los Proxy Services. La carpeta Resources puede contener todos los demás recursos administrables de OSB tales como WSDL, JAR, XSD, XQueries, etc. Otra alternativa es configurarle una carpeta independiente para cada recurso, de manera de separarlos especialmente cuando existe un alto número de ellos.

La carpeta BusinessServices puede contener una subcarpeta Stubs que sirve en desarrollo y testing para contener componentes que sean usados en fase de prueba. Esta carpeta debe ser removida cuando se pasa a producción.

La carpeta Resources puede tener subcarpetas que sirven para separar los recursos dependiendo su tipo.

Para la estructuración de servicios canónicos, y servicios de conectividad de aplicaciones, se utiliza otra estructura de proyectos y carpetas, cuyo objetivo es la separación de los servicios por dominio. Para cada dominio se tiene, por un lado los servicios canónicos y por otro los servicios conectores, estos ultimos divididos por aplicación. Este estándar se describe en el documento *CENCOSUD – Principios, Estándares y Lineamientos.doc.*

* **Estándares de nomenclatura de componentes**

Es útil nombrar a los Servicios Proxy con un subfijo PS o LPS (para los proxies locales) y a los Servicios Business con BS

Ejemplo:

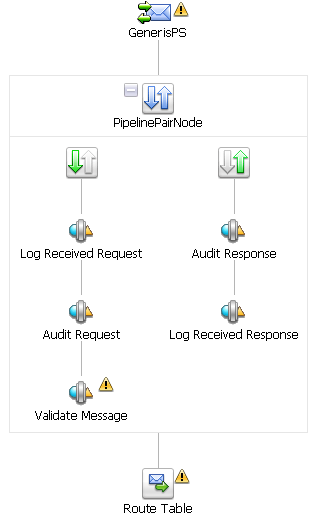
**MyProxyPS  
MyLocalProxyLPS  
MyBusinessServiceBS**

Es importante saber si un proxy es local o no. Sin embargo es una mala idea poner una referencia al transporte en el nombre del proxy (ej: MyProxyHTTP). Esto es porque el transporte puede cambiar con el tiempo. Es también importante destacar que el transporte debe ser tratado de manera independiente de la implementación. Si se requiere un servicio relacionado específicamente con un transporte, entonces se podría considerar el uso de transporte en el nombrado del proxy. Este podría ser tratado en un proxy Local.

* **Nomenclatura de Pipeline, Stages, Routing Node**

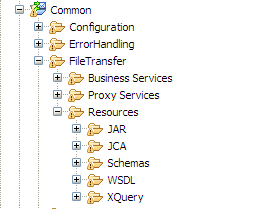
Un proxy service puede ser escrito con múltiples stages donde cada stage implemente una función específica. Ejemplo, Log Received Request.

Todos los stages deben ser nombrados de acuerdo a la función que están implementando y este nombre debe ser fácilmente asociado a la función. Es considerada una mala práctica el dejar el valor por defecto del nombre de un stage. Por ejemplo, stage1



**Organización del Proyecto Servicios Comunes de Cencosud**

La estructura propuesta para los servicios de apoyo es la siguiente:



Es decir, dentro de un Proyecto OSB denominado Common, tenemos direrentes carpetas que agrupan componentes por funcionalidad, como ser Configuration, ErrorHanding, FileTransfer, Logging, Retry, etc.

Dentro de cada carpeta se adopta la alternativa 2, mencionada más arriba en este mismo documento.

Dentro de la carpeta Resources se presenta la siguiente separación:

* JAR
* JCA
* Schemas
* WSDL
* XQuery

El nombre de los servicios hacen referencia a la utilidad que el servicio brinda, por ejemplo *RecordTransaction*, registra una transacción, por ejemplo un archivo procesado por el middleware.

* Los servicios de tipo BS con transporte *JCA DBAdapter* tienen el sufijo “\_db”, por ejemplo *RecordTransaction\_db*
* Los Proxy Services tienen el sufijo “*PS*”, por ejemplo *RecordTransactionPS*.
* Los demás Business Services no incluidos en los casos anteriores utilizan el sufijo “*BS”*. En particular, a los *BS* con transporte JMS se los nombra con el nombre de la cola destino con el sufijo *“BS”.*

1. **Diseño de proxies**

* **El patrón de diseño VETO**.

Un flujo de mensaje debe estar estructurado según el patrón de diseño VETO

1. Validar
2. Enriquecer
3. Transformar
4. Operar

El patrón VETO debe ser usado para asegurar la consistencia de la data que pasa por el OSB. Sirve también para asegurar que las aplicaciones externas se acoplen a la arquitectura del bus sin necesidad de conocerse unas a otras. Para que esto suceda el OSB necesita ser la entidad responsable de hacer que los servicios y sus consumidores funcionen juntos correctamente. El patrón representa el conjunto de actividades que deben ejecutarse para garantizar la integridad del OSB.

1. **Validar**

La primera actividad en el patrón es validar la información de entrada. La forma más fácil de hacerlo es a través de schemas y hacer que el OSB valide la información de entrada contra ese schema. Usar la acción “Validate” en un solo stage dedicado exclusivamente a esto.

1. **Enriquecer**

Enriquecer es definido como el proceso de obtener información adicional de un recurso externo y agregarlo al mensaje actual. Esto puede ser realizado a través de un Service Callout seguido por una acción de modificación como “Insert” o “Replace” para modificar el documento de entrada.

Tener en cuenta que los Serivce Callout deben ser implementados teniendo en cuenta que el servicio puede estar no disponible. Podría tratarse el Service Callout en un solo stage pudiendo manejar errores en este nivel.

1. **Transformar**

La transformación es un paso donde se debe formatear el mensaje actual con el formato de mensaje que espera el servicio destino. Si el servicio tiene un stage de validación, la transformación debería dejar el mensaje con el formato necesario para que pase dicha validación.

Notar que si se usan transformaciones XQueries el tamaño del XQuery debería ser el mínimo. En caso de que un XQuery sea complejo y largo, es recomendable modularizarlo en muchos XQueries y hacer la transformación en múltiples pasos. Esto promueve el mantenimiento y el soporte. Otra forma de hacer transformaciones es a través de Java Callout y modelar en un POJO Java las transformaciones. Esto va a favor de poder testear dichas transformaciones con test de unidad.

1. **Operar**

Este paso es usado para enviar el mensaje a un servicio. Con OSB esto puede ser realizado con un nodo “Publish”, “Publish Table”, “Route Node” o “Routing Table”. Cuando un mensaje es publicado a otro servicio como resultado de otras acciones de “Publish”, “Dynamic Publish” o “Publish Table”, el QoS por defecto es “Best Effort”. “Best Effort “significa que no hay mensajería confiable, sin embargo, es mas performante.

Notar que un Service Callout nunca puede ser usado para routear un mensaje. Los nodos de “Publish” y “Routing” son más performantes. Si se requiere enviar múltiples veces información a un servicio durante el flujo de un mensaje, las acciones de “Publish” son más apropiadas. Mientras que las acciones de “Routing” deben ser usadas para cuando se requiere invocar al servicio solo una vez.

* **Usando “Service Callout” sincrónico**

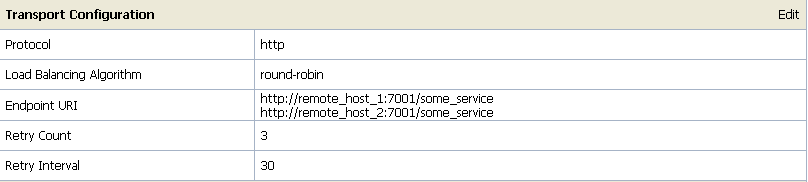
La comunicación fuera de OSB puede ser hecha a través de:

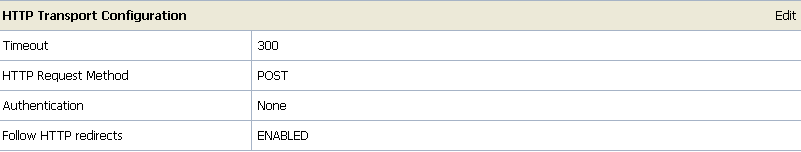
* + Route Node (mas preferible)
  + Publish Action
  + Service Callout (menos preferible)

Aunque la acción de service callout es la forma menos indicada para enviar un mensaje fuera del OSB, hay situaciones donde debe usarse. Un buen ejemplo es cuando la data de un mensaje debe ser transformada vía un servicio externo..

Uno de los mayores problemas de los Service Callouts sincrónicos es que por cualquier razón el servicio puede no estar disponible. El servicio también puede no estar manejando bien la carga y por consiguiente formar un cuello de botella en el proxy respectivo. Si se require usar service callout entonces se debe tener los siguientes puntos:

* + Si el servicio es via HTTP/S, entonces configure el sevice timeout a un valor apropiado. Esto debe ser menor que el valor de stuck thread configurado (por defecto 600) asi el proxy tiene tiempo de manejar el problema antes que el servidor mate el proceso automáticamente.
  + El reintento de servicio debe ser seteado a un valor apropiado.
  + Debe haber más de un endpoint configurado. Preferentemente deberían ser diferentes serves manejados a diferentes maquinas físicas.
  + Ubicar el service callout a un stage propio, para poder manejar errores fácilmente.
  + Agregar un manejador de errores para poder procesar la excepción si el servicio no está disponible.





Dependiendo del SLA y de los requerimientos del negocio, otra forma solución más robusta es la de routear el mensaje al servicio que corresponda de manera asincrónica con un nodo Publish o Route. Un segundo proxy podría recibir la respuesta del servicio y luego continuar con el procesamiento. Si el servicio esta caído, el mensaje quedaría en la cola hasta que el servicio volviese a estar disponible.

* **Stages: Uno o muchos**

Un solo stage en el pipeline generalmente es suficiente para la mayoría de los casos de uso. Sin embargo, hay casos donde se necesita considerar el uso de múltiples stages:

* + Multiples stages proveen una forma natural de modularización.
  + Cada stage en un pipeline request o response puede tener manejadores de errores independientes. De esta manera se evita tener un solo manejador de errores que encapsule la lógica de cada uno de las acciones que se realizan en un solo stage.

Las acciones que se pueden hacer en un stage son:

* + Transformar un mensaje.
  + Reportar un mensaje
  + Validar un mensaje y levantar errores
  + Hacer un Publish de un mensaje a un destino.
  + Hacer llamadas Callouts o de Routeo

* **Decidiendo que tipo de servicio usar**

El OSB es compatible con una variedad de tipos de servicio que va desde los considerados como servicios web (usando XML o SOAP bindings en WSDL), hasta los servicios no genéricos o no XML.   
  
El uso de WSDL para definir la Interface de un servicio tiene una serie de beneficios, pero es opcional.

* + El sistema puede proveer métricas por cada operación en el WSDL
  + Se puede hacer un branch por operación dentro del pipeline
  + Para los servicios invocados a través de un servicio proxy, el header soapaction es completado automáticamente.
  + Es fácil manipular el $body en los editores de XQuery y Xpath porque el editor provee mapeo por default del request mensaje a la variable $body.
  + Para los servicios que usan WS-Security, se requiere el uso de WSDL (las políticas son adjuntadas al WSDL).
  + El sistema soporta la sintaxis <url>?WSDL para que dinámicamente se obtenga el WSDL del proxy http. Esto es muy útil para clientes soap como el SoapUI.

OSB no valida el mensaje enviado o recibido a través de la definición de la interface del servicio automáticamente. Sin embargo es posible usar una acción de “Validate” y una expresión condicional XQuery para realizar dicha validación explícitamente en el flujo del servicio.

Cuando se utilice WSDL, elija hacer un bind al port del WSDL en vez de un binding en los siguientes casos:

* + En el WSDL generado con la sintaxis ?WSDL, el nombre del puerto se conserva. Esto puede ser importante para algunas herramientas clientes.
  + Sin embargo, la url del servicio predominante es la que se especifica en la sección transporte de la UI de definición del servicio.
  + La URL en el WSDL no se utiliza, salvo para considerarla como la URL default en la definición de un business service.
  + Utilice el tipo de servicio de mensajería si al menos uno de los mensajes (petición o respuesta) no es XML

* **Cuál es el contexto de los diferentes mensajes**

A lo largo del flujo de un servicio, hay un mensaje y las diferentes partes de este mensaje están presentes en $header, $body y $attachments. Incluso si el tipo de servicio no es SOAP, el mensaje aparece como SOAP en el contexto.

* $Header tiene el elemento encabezado SOAP.
* $body tiene el elemento Body SOAP, cuerpo principal del mensaje
  + Si el mensaje es binario, el elemento Body del $body contiene una referencia XML del elemento llamado binary-content (que contiene el mensaje binario). En el caso de los adjuntos binarios del elemento Body tiene el elemento de referencia.
  + Para los mensajes de MFL, el elemento Body en $body tiene el XML equivalente al documento MFL
  + Para los mensajes de texto, el elemento Body en $body tiene el texto completo del mensaje. Para los archivos adjuntos de texto, el elemento Body en $attachments tiene el texto.
* $attachments tiene un elemento wrapper llamado attachments con un elemento hijo por archivo adjunto. El elemento attachment tiene un cuerpo con el adjunto real
  + Para los archivos adjuntos XML, el elemento Body en $attachments tiene el XML.

A continuación se muestra los esquemas de estos elementos. El prefijo del namespace de todos estos elementos es "ctx", que es el que esta predefinido en los editores.

<element name="attachments" type="mc:AttachmentsType"/>

<!-- Each attachment in the 'attachments' variable is represented by

an instance of this element -->

<element name="attachment" type="mc:AttachmentType"/>

<!-- Element is used to represent binary payloads and pass-by reference

content -->

<element name="binary-content" type="mc:BinaryContentType"/>

<complexType name="AttachmentsType">

<sequence>

<!-- the 'attachments' variable is a series of attachment elements -->

<element ref="mc:attachment" minOccurs="0" maxOccurs="unbounded"/>

</sequence>

</complexType>

<complexType name="AttachmentType">

<sequence>

<!-- Set of MIME headers associated with attachment -->

<element name="Content-ID" type="string" minOccurs="0"/>

<element name="Content-Type" type="string" minOccurs="0"/>

<element name="Content-Transfer-Encoding" type="string" minOccurs="0"/>

<element name="Content-Description" type="string" minOccurs="0"/>

<element name="Content-Location" type="string" minOccurs="0"/>

<element name="Content-Disposition" type="string" minOccurs="0"/>

<!-- Contains the attachment content itself, either

in-lined or as <binary-content/> -->

<element name="body" type="anyType"/>

</sequence>

</complexType>

<complexType name="BinaryContentType">

<!-- URI reference to the binary or pass-by-reference payload -->

<attribute name="ref" type="anyURI" use="required"/>

</complexType>

* **Copiando propiedades JMS**

OSB asume que la interface de un servicio proxy y del servicio business pueden ser diferentes. No propaga (los headers del transporte o las JMS properties) desde el contexto inbound al contexto outbound.

El header del transporte del mensaje request y el response del servicio proxy están en el $inbound y el header del transporte del request y el response del business service esta en el $outbound.

Por ejemplo, la siguiente Xquery puede ser usada para el caso en que las propiedades JMS definidas por el usuario para un mensaje *one way* (invocación sin respuesta) necesiten ser copiadas desde el $inbound al $outbound.

**Insert** $inbound/ctx:transport/ctx:request/tp:headers/tp:user-header **as first child of** ./ctx:transport/ctx:request/tp:headers **in variable** $outbound

* **Garantía de entrega**

A menudo el usuario está interesado en cual es la garantía de entrega con la que cuenta el producto. La garantía de entrega solo está provista si el transporte del inbound es JMS con una conexión XA y el QoS del outbound es “exactly once”. Para el resto de los casos el mensaje se puede perder o puede ser entregado más de una vez depende de las circunstancias. La QoS especificada en el $outbound es solo un indicio de la garantía de entrega deseada. Para el QoS “excatly once” el sistema dará garantía de entrega de exactamente una, sino no dará garantía de ningún tipo.

Para los siguientes casos (Inbound JMS/XA and QoS = exctly once) la garantía esta provista:

* + Si el transporte oubound para una acción de Publish o Nodo Route es JMS/XA, entonces el sistema asegura que el mensaje será entregado exactamente una sola vez desde el input transport al output transport, sin riesgo de pérdida.
  + Para el resto de los transportes (email, FTP, file, HTTP/S JMS/non XA) el sistema asegura la entrega del mensaje al menos una vez desde el transporte de entrada al transporte de salida sin pérdida del mensaje.
* **Reintentos para mensajes Outbound**

Además de los intentos de los mensajes JMS de entrada, se puede configurar reintentos de salida y Load Balancing. Load Balancing, failover y reintentos trabajan en conjunto para proporcionar rendimiento y alta disponibilidad. Por cada mensaje la lista de URLs automáticamente se ordena basada en el algoritmo de load balancing en una secuencia de failover. Si el número de reintentos es N, toda la secuencia se vuelve a intentar N veces antes de detenerse. Antes de ejecutar cada ciclo de la secuencia, el sistema espera a que el intervalo de reintento especificados se cumpla antes de volver a comenzar. Después de completar los reintentos, si todavía hay un error, el manejador de errores del pipeline del nodo es invocado.

* **Content types, JMS type y encoding**

Para interoperabilidad con endpoint heterogéneos el usuario necesita controlar el content type usado, el JMS type y el encoding.

OSB evita hacer supuestos sobre el cliente externo o las necesidades del servicio y usa la información configurada para este propósito en la definición del servicio. Esto maximiza la posibilidad de interoperabilidad con varios endpoints.

OSB deriva el content type para los mensajes de salida dependiendo del tipo de servicio y la interface. Content type es parte de los protocolos HTTPS(S) y email .

* Si el servicio es XML o SOAP (con o sin WSDL), el content type es text/xml.
* Si el servicio es de tipo messaging y la interface es MFL o binario, el content type es binary/octet-stream.
* Si el servicio es de tipo messaging y la interface es texto, el content type es text/plain.
* Si el servicio es de tipo messaging y la interface es XML, el content type es text/xml.
* **Request/Response asincrónicos**

Un caso de uso común es cuando el cliente llama a un servicio proxy web HTTP, pero el sistema back-end que el servicio Web invoca utiliza un request/reponse JMS. Este caso de uso se refiere a un caso "sync to async". El caso de uso inverso (async to sync) es también una posibilidad.

El uso de request/response asincrónico tiene 3 ventajas

* + El thread request no bloquea la respuesta, eliminando así un problema de manejo de threads cuando se hacen numerosos llamados.
  + El mensaje es más confiable
  + El uso de MQ Series generalmente se prefiere para la interacción con mainframes

* **Usar JMS Correlation ID**

Cuando un cliente se comunica con el OSB utilizando medios de comunicación asincrónico como colas y a su vez espera una respuesta, se deben usar mecanismos para implementar comunicación sincrónica utilizando medios asincrónicos.

Los mensajes request y response para un servicio business que se comunica con un OSB a través de JMS deben ser linekados usando JMS Correlation ID. Es decir, cuando se diseña un servicio business, se debe asegurar que se toma el getJMSCorreltationID en el mensaje de entrada se setea ese valor en el setJMSCOrrelationID del mensaje de salida antes de mandar el mensaje a la cola o topic JMS.

La interface (javax.jms.Message) es la raiz de la interface de todos los mensajes JMS. Los encabezados de mensajes incluyen el campo JMSCorrelationID, que sirve para hacer trazabilidad entre mensajes. El JMSCorrelationID puede contener un ID especifico del proveedor, un texto propio de una aplicación o un valor nativo en byte[].

* **Ruteo Dinámico**

Para el escenario en el que el diseñador de un servicio proxy no conoce el servicio concreto a invocar, pero sabe la forma de la interfaz, el servicio que se necesita invocar se especifica, directa o indirectamente en el mensaje de entrada.

En este caso, el enfoque es la detección de un servicio business con la forma correcta (la URL del transporte no importa). Una acción dinámica setea el $outbound con la URL del servicio a invocar después de seleccionar en el servicio una acción de Route o Publish. Esto permite que la URL sea obtenida en tiempo de ejecución y elimina la necesidad de saber en tiempo de diseño.

* **Información de enrutamiento en el encabezado**

Se recomienda que las decisiones de enrutamiento estén basadas en los encabezados SOAP o los encabezados de transporte (para payloads no SOAP). OSB usa un análisis parcial para procesar los encabezados SOAP, sin tener que analizar el cuerpo del mensaje.

Los encabezados de transporte son fáciles de acceder sin necesidad de análizar el payload. Así, se puede escalar con el incremento del tamaño de los mensajes mediante el almacenamiento de información en las cabeceras.

* **Usar Throttling para proteger los sistemas de back-end de picos de carga**.

Se puede implementar un mecanismo de *Throttling* mediante la creación de políticas de *Throttling* de un Business Service en OSB o mediante *Work Managers* a nivel WLS. Con esto, se puede limitar el número máximo de llamadas simultáneas hacia el servicio back-end. Si se requiere un mayor nivel de fiabilidad, se puede utilizar JMS persistente como un mecanismo seguro y transaccional para el buffer de solicitudes.

* **Utilizar paralelismo (Split-Join) para reducir la latencia cuando la CPU no está saturado**

La ejecución en paralelo es un paradigma común para reducir la latencia, aunque se sabe que consume mayores recursos del sistema. Un conjunto de tareas que no dependen una de la otra se pueden ejecutar en paralelo para poder hacer uso sensible de la latencia. Como la ejecución de múltiples tareas en paralelo aumentan el consumo de CPU, el paralelismo no proporciona beneficios cuando el CPU está ya saturado. En OSB las tareas paralelas no participan en la transacción padre y por lo tanto deben ser idempotentes.

* **Usar Seguridad de Web Services cuando se requiera seguridad a nivel de mensaje**

WS-Security tiene un overhead mayor asociados con el cifrado y el procesamiento que la seguridad a nivel de transporte como SSL + autenticación básica.

* **Usar transporte EJB principalmente cuando se desea bajo acoplamiento.**

El transporte EJB en OSB está diseñado para ofrecer un bajo acoplamiento y una interfaz orientada al mensaje utilizando WSDL. Un mensaje basado en WSDL hace la unión con Java con el estándar JAX-RPC. Esta unión tiene un costo de traducción que es aceptable cuando el EJB back-end representa un servicio reutilizables.

Sin embargo, el costo de la traducción puede ser considerado muy alto si se realizan múltiples  
llamadas cortas a uno o varios EJBs en un único proxy. Una Java Callout puede ser una mejor  
solución para acceder a un EJB de manera más acoplada, pero con un mecanismo de bajo overhead.

* **Estandarizar la representación XML cuando se cambia entre Java Callout y Acciones de manipulación de XML**

Minimizar las conversiones de tipo de datos XML a "XML como texto" y viceversa. Si una  
acción posterior requiere un tipo de datos XML, devolver un *XMLObject* de un Java Callout en lugar de un String.

1. **Manejo de Errores**

Esta sección describe las buenas prácticas en el manejo de errores en un flujo de mensaje de un proxy en OSB.

* **Manejadores de errores**

Los manejadores de errores son usados para atrapar y controlar las excepciones (de negocio y técnicas). Estos pueden ser agregados a:

* + - Stage: Cada stage en un pipeline
    - Pipeline: tanto en el request y en el response.
    - Route: en el node Route
    - Service: en el nodo global de servicio.
  + **Manejador de errores a nivel Stage**

En general es más fácil manejar errores específicos en los niveles más bajos y manejar errores generales con procesamiento por default que en los niveles más altos. Es una buena práctica manejar los errores anticipados en el pipeline y dejar para el manejador de errores a nivel servicio los errores no anticipados.   
Haciendo que cada una de las partes de un flow tenga un stage especifico, se pueden manejar errores muy específicos.

* + **Manejador de errores a nivel Pipeline**

Este tipo de manejador se ubica por sobre los manejadores stages. Cualquier excepción re lanzada en un stage es propagada a un manejador de pipeline. Los errores generales que controla este tipo de manejador son consecuencia de un mensaje mal formado o un defecto en el request path.

* + **Manejador de errores a nivel nodo Route**

La mayoría de los errores de OSB ocurren a nivel de Nodo Route. Algunos ejemplos son errores de timeout esperando mensaje respuesta, endpoints inválidos, errores de seguridad jms, etc.

Este manejador de error es responsable de poner el mensaje error en el body SOAP y enviarlo a lo largo del flujo de mensaje de respuesta. El mensaje de error también debe loguearse.

* + **Manejador de errores a nivel Servicio.**

Se recomienda siempre tener en cuenta los errores no anticipados agregando un manejador de errores a nivel de servicio en la parte superior del flujo de mensaje. Esto permite que cualquier excepción no manejada a nivel de stage, pipeline o Route sea capturada y manejada.

Es importante notar que este manejador debe ser usado para errores no anticipados. Es una mala práctica usarlo para manejar todos los errores del proceso.

* **Manejadores de errores Comunes**

Muchas veces ante errores comunes que se producen dentro de los servicios Proxy, es más eficiente contar con un manejador común de errores. Dependiendo del tipo de excepciones que pueden ser capturados, es posible escribir el código de control de errores que se repite en varios lados en un solo lugar.

Este manejador de errores debe ser un Servicio Proxy con transporte local. Al tener el transporte local, permite que su ejecución corra sobre el mismo thread.

* **Comportamiento de errores**

La manera en que los errores son manejados dependen también en como son llamados los proxy service. Por ejemplo: Queremos hacer un rollback de la transacción en un proxy asincrónico o queremos mandar un mensaje de respuesta fallido en un proxy sincrónico.

Un manejador de errores para un transporte asincrónico tipo JMS, debe contener una acción de “Raise Error”. Esto es como volver a lanzar la excepción haciendo que la transacción haga rollback. En el caso de JMS el mensaje va a ser devuelto a la cola. Si el manejador de errores no hace la acción “Raise Error” el mensaje es sacado de la cola como si se hubiera procesado de manera exitosa.

Cuando se maneja el error en el nivel de servicio, en un servicio con transporte http y se ejecuta la acción “Reply With Failure”, se está retornando al consumidor un código http 500. Por lo tanto una práctica común en la que se quiere retornar el mensaje de error como una respuesta exitosa (http 200), es reemplazar el valor del $body con el mensaje de error y ejecutar la acción “Reply With Success”.

La acción “Raise Error” va a ser capturada por un manejador de mayor nivel. Por ejemplo, si se lanza una excepción en un manejador Stage va a ser capturada en el manejador de servicio. Para hacer que la transacción haga roll back no se debe capturar la excepción en ningún manejador de nivel superior.

1. **Buenas prácticas de Logging**

Para probar y debuggear un flujo de mensaje, se debe establecer una política de logueo donde todos los proxies usen exactamente la misma manera de loguear. Este logueo no debe ser confundido con el logueo propio del negocio. Esta sección intenta explicar cómo y dónde usar la acción “Log” en un flujo de mensaje y como esto puede ser potenciado usando un log dedicado.

* **Uso de Acción de “Log”**

Algunos tips sugeridos para el uso de la acción de Log son:

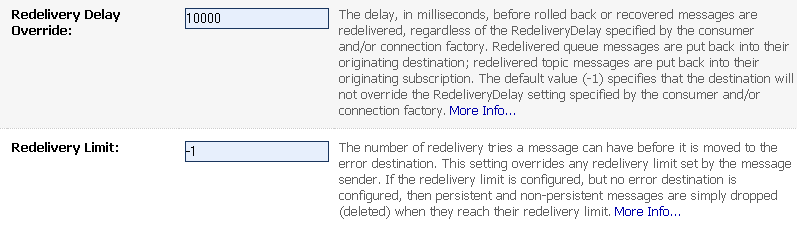
* + - El logueo no debe contener el body del mensaje completo. Comúnmente es mejor solo loguear lo básico del mensaje. Si se necesita loguear el mensaje entero, entonces asegurarse que es con nivel de “Debug”
    - El logueo de trazabilidad (Entrada y Salida de cada proxy) no debe necesariamente loguear el campo $body completo. Alcanza tan solo con un Id de mensaje. Si el logueo de trazabilidad es mínimo entonces puede dejarse en nivel “Info”
    - Los requerimientos de análisis de log deben tenerse antes de que el desarrollador comience con su trabajo. Esto permite que se apliquen estándares.
    - Cuando se usen acciones de “Publish” y “Service Callout”, las acciones de log deben estar dentro de las acciones de request y response, en vez de tenerlas dentro del flujo principal
    - Los manejadores de errores deben loguear en nivel “Error”
    - Se debe loguear únicamente en los casos donde no exista un “loop” Esto es porque esto puede causar que se llenen los archivos de logs haciendo que se produzcan problemas de sistema
    - Configurar la opción “Domain log Broadcaster buffer size” a algo mayor a 10. Esto hará que se mejores la performance si hay un gran volumen de logueo

1. **Patrón de Manejo de Excepciones**

Los Proxy Service que están suscriptos a colas de mensaje JMS deben lidiar con una cantidad excesiva de colas de errores. Si un mensaje falla en el procesamiento y no puede ser desechado, se debe devolver a la cola original. Luego de un cierto número de reintentos, el mensaje es promovido a la cola de errores configurada. Reprocesar estos mensajes requiere de múltiples proxies escuchando en las colas de errores o se requiere de un procesamiento manual.

Una forma de eliminar este porcentaje de colas de errores es la de manejar el procesamiento de reintentos antes que dejarle a WLS manejarlo a través de configuraciones JMS. Si se captura una excepción dentro del flujo de mensaje, se chequea el encabezado JMS para saber cuántas veces el mensaje fue previamente reintentado anteriormente. Si el valor alcanzó el límite predefinido, entonces el mensaje no hace roll back y se envía a una cola de manejo de errores comunes.

Usando este patrón, no dejamos que WLS mueva automáticamente el mensaje a una cola de errores propia. Entonces debemos configurar la propiedad “Redelivery Limit” en -1 y dejar en blanco la propiedad “Error Queue”



Dentro del flujo del mensaje, podemos evaluar la propiedad JMSXDeliveryCount en el encabezado JMS. Esto se puede hacer a través del $inbound ($inbound > request-jms > headers > JMSXDeliveryCount)

Una vez que obtenemos el valor, podemos chequear que se excedió al valor que tenemos predefinido. Si es el caso, el mensaje puede ser enviado a una cola de error dedicada. Antes de enviar el mensaje podemos agregar información al mensaje, como por ejemplo la excepción o información relevante al encabezado JMS.

1. **Tuning de Proxies en Runtime**

Una configuración típica de OSB abarca a un cliente invocando a un Servicio Proxy OSB que hace una o varias invocaciones a servicios backends, ruteando el mensaje y devolviendo al cliente una respuesta. En dicho escenario, el cuello de botella puede ser el cliente, la instancia de OSB, servicios intermedios o el servicio backend final. El tuning comienza con la correcta identificación de los cuellos de botella, así como también de ajustes en otros sistemas que ayuden a mejorar el rendimiento general.

Algunos de los factores que afectan la performance en OSB son:

* + - 1. Tamaño de mensaje

El tamaño de mensaje determina la latencia de la red, requerimientos de memoria y costos de transformación. Para un escenario dado, usar un mensaje grande resulta en una caída de rendimiento y un incremento de latencia.

* + - 1. Requerimientos de diseño y complejidad

El procesamiento XML es generalmente costoso en tiempo de ejecución. Las transformaciones, decisiones de ruteo, llamado a servicios, etc. tienen un impacto importante en un servicio proxy.

* + - 1. Transporte

La performance y la escalabilidad son características que derivan directamente del inbound y outbound del transporte. Transportes como HTTP, JMS (Persistente o no) tiene diferentes representaciones de red, niveles de QoS y sobrecarga de I/O

* + - 1. Requerimientos de fiabilidad

En la mayoría de los casos, cuanto más grande son los requerimientos de fiabilidad más grande es punitorio de performance para llegar a alcanzar el objetivo. La fiabilidad es alcanzada en un porxy por el soporte transaccional y la garantía de entrega.

* + - 1. Políticas de seguridad

OSB soporta una gran combinación de paradigmas de seguridad a nivel de red y a nivel de mensaje. Dependiendo el paradigma seleccionado, los costos pueden variar significativamente.

* + - 1. Hardware

Arquitectura de CPU y el medio de almacenamiento juegan un papel importante a la hora de determinar rendimiento y latencia

A continuación se proponen algunos factores a tener en cuenta realizar optimizaciones de performance en OSB:

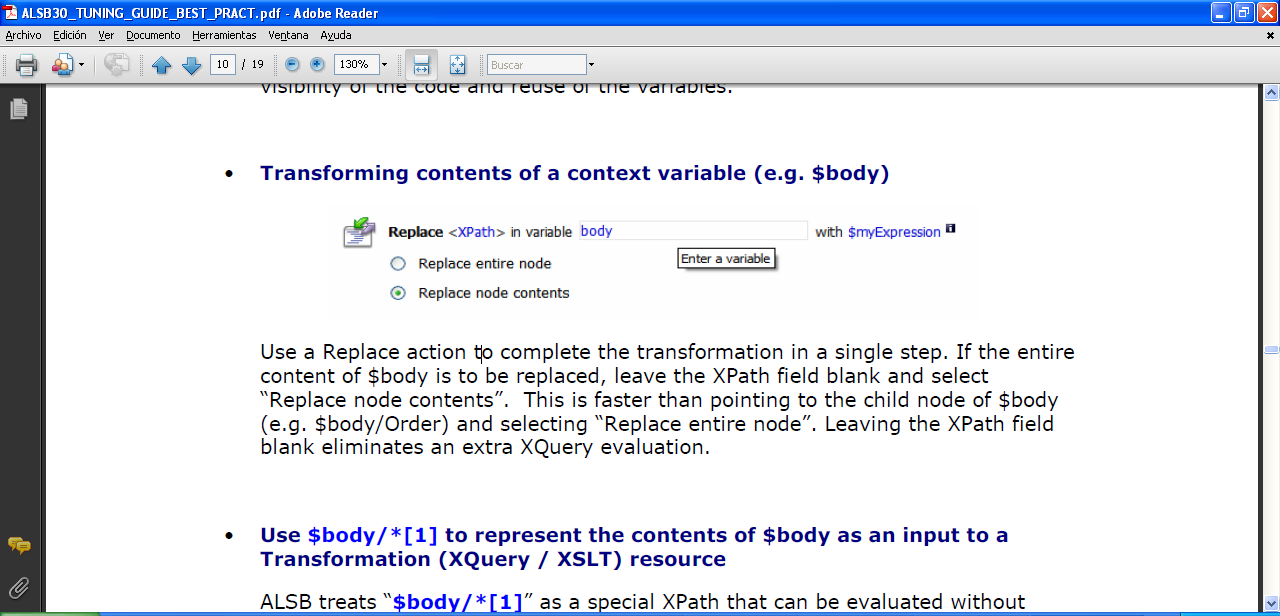
* **Evitar crear variables de contexto que solo se usen una vez dentro de otro XQuery**

Las variables de entorno que son creadas con una acción “Assign” se convierten en XMLBeans y luego son convertidas a formato nativo de XQuery para poder realizar la siguiente llamada XQuery. Múltiples acciones de “Assign” pueden ser diseñadas en una sola acción “Assign” utilizando una expresión FLOWR. Valores intermedios pueden ser creados con sentencias “let”.

Evitando redundancia en la asignación de variables de contexto, se elimina el overhead asociado a las a las conversiones internas de formatos. Este beneficio tiene que ser puesto en la balanza sobre la visibilidad del código y el re-uso de las variables

* **Transformación del contenido de una variable de contexto (Ej: $body)**

Usar la acción “Replace” para transformar, en un solo paso, el contenido completo de una variable de contexto. Si el contenido completo de la variable $body se quiere reemplazar, dejar el campo XPATH en blanco y seleccionar “Replace node contents”. Esto es más rápido que apuntar a un nodo hijo y seleccionar “Replace entire node”. Al dejar el campo Xpath en blanco se elimina una evaluación extra en Xquery.

****

* **Usar “$body/\*[1]” para representar el contenido de la variable $body como entrada a un recurso de transformación (Xquery o xslt)**

OSB trata a “$body/\*[1]” como un Xpath especial que puede ser evaluado sin tener que invocar al motor Xquery. Esto es más rápido que especificar un camino absoluto al child del $body. Un Xpath general como por ejemplo “$body/Order” debe ser evaluado por el motor Xquery antes de que la transformación principal sea ejecutada.

* **Habilitar Streaming para escenarios puros de “Content Based Routing”**

Escenarios de solo lectura, como “Content Based Routing”, se ven considerablemente favorecidos cuando se habilita streaming. OSB puede aprovechar las capacidades de análisis parcial del motor XQuery cuando se use Streaming con XPaths indexados. Así, el payload se analiza y se procesa sólo en el campo que se refiere el XPath. Un beneficio adicional para los escenarios de sólo lectura es que streaming elimina la sobrecarga asociada con el análisis y la serialización de  
XMLBeans.

* **Habilitar Streaming si hay un OOM al transformar mensajes de gran tamaño.**

La salida de una transformación se almacena en un formato de buffer comprimido, ya sea en memoria o en disco. El usar un formato buffer comprimido tiene una pena de CPU significativa. Por lo tanto, streaming debe evitarse cuando el Out Of Memory no es una preocupación.

* **Ajustar la configuración adecuada del nivel de QoS y de transacción.**

No utilice XA o “Exactly-Once” a menos que la fiabilidad de “una vez y solo una vez” sea necesario y sea posible (por ejemplo, no es posible si el cliente es un cliente HTTP).  
XA añade costos de CPU y I/O a todos los elementos que intervienen en la transacción global. Si OSB inicia una transacción, es posible sustituir XA con LLR para alcanzar el mismo nivel de fiabilidad. OSB puede invocar un servicio HTTP back-end de forma asíncrona si el QoS es "Best-Effort”. La invocación asincrónica permite a OSB escalar mejor ante servicios backend de larga duración. También permite Publish a través de HTTP para ser realmente un “fire-and-forget”.

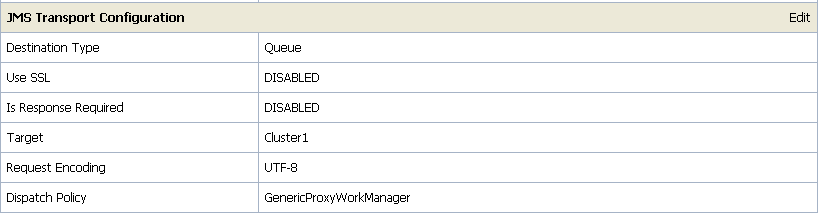
1. **Uso de Work Managers**

Un *Work Manager* define un conjunto de clases y limitaciones de threads para manejar las ejecuciones en WLS. Se puede definir Work Manager Globales de nivel de dominio. Tambien se pueden definir de nivel de aplicación o de nivel de módulo. WLS permite que se pueda configurar como tu aplicación prioriza su ejecución basado en reglas que se definen y pudiendo monitorear la performance en tiempo de ejecución. Por ejemplo: si se tiene 2 servicios, donde uno es muy usado pero poco importante y el otro es menos usado pero muy importante. Se puede configurar un *Work Manager* para acompañar la priorización del servicio.

Es una buena práctica configurar Works managers a cada proxy en ves de dejarlo con la configuración por default. Antes de poder asignar un WM a un Proxy, este debe haber sido creado por consola (Enviroments > Work Managers)



El siguiente diagrama es un ejemplo de un Work Managers y una restricción de cantidad de threads máximos. Los *Works Managers* y sus respectivos constraints deben estar en cluster, no en el Admin server.



Una vez que el Work Manager fue creado, puede ser agregado a un Politica de Dispatch en un Proxy Service. Esta configuración se encuentra bajo la sección “Transport Configuration” del Proxy Service. Notar que los WM y sus constraints deben tener el mismo nombre que el proxy con un sufijo de “WorkManager” y “MaxThread” respectivamente. Esto ayuda a que no se reusen WM en múltiples proxies.

1. **Xquery Tuning - Transformaciones**

OSB utiliza XQuery y XPath para diversas acciones como “Assign”, “Replace”, “Routing Table”. La siguiente estructura XML ($body) va a ser usada para explicar los conceptos de tuning de XQuery y XPath

<soap-env:Body>

<Order>

<CtrlArea>

<CustName> Mary </CustName>

</CtrlArea>

<ItemList>

<Item name="ACE\_Car"> 20000 </Item>

<Item name=" Ext\_Warranty"> 1500 </Item>

.... un gran número de elementos

</ItemList>

<Summary>

<Total> 70,000 </Total>

<Status> Enviado </Status>

<Shipping> Mi empresa de envío </Shipping>

</Summary>

</Order>  
</soap-env: Body>

* **Evitar el uso de la // en Xpaths.**

body$//CustName retorna el mismo valor que $body/Order/CtrlArea/CustName, pero de manera menos eficiente.

La expresión. "//" implica todas las apariciones de un nodo, independientemente de la ubicación en el árbol XML. Por lo tanto, toda la profundidad y amplitud del árbol XML debe ser recorrido para encontrar el patrón especificado. Use "//" si la ubicación exacta de un nodo no se conoce en tiempo de diseño.

* **Indexar XPaths donde sea posible.**

Un XPath puede ser indexado simplemente agregando "[1]" después de cada nodo de la ruta.  
XQuery es un lenguaje declarativo y XPath puede devolver más de un nodo, puede devolver un array de nodos. $body/Order/CtrlArea/CustName implica devolver todas las instancias de la Orden dentro de $body y todas las instancias de CtrlArea bajo Order, etc  
Por lo tanto, todo el documento tiene que ser leído con el fin de procesar correctamente los anteriores XPath.

Si sabemos que hay una instancia única de la Order dentro de $body y una sola  
instancia de CtrlArea bajo Order, podemos reescribir el anterior XPath como  
$body/Order[1]/CtrlArea[1]/CustName[1]. Este XPath implica retornar la primera instancia de los nodos hijos. Por lo tanto, sólo la parte superior del documento necesita ser procesada por el motor XQuery que resulta en un mejor rendimiento.

• Nota: La indexación no debe ser utilizada si el valor de retorno esperado es un array de  
nodos. Por ejemplo $body/Order[1]/ItemList[1]/Item devuelve todos los nodos “Items”, pero  
$body/Order[1]/ItemList[1]/Item[1] sólo devuelve el nodo del primer elemento.

* **Extraer las partes que se usan frecuentemente de un XML largo, como variables intermedias con expresiones FLOWR**

Una variable intermedia puede ser utilizada para almacenar el contexto común para múltiples valores. Un ejemplo de un contexto XPaths común:

$body/Order[1]/Summary[1]/Total, $body/Order[1]/Summary[1]/Status, $body/Order[1]/Summary[1]/Shipping  
  
Los XPaths anteriores pueden ser cambiado para usar una variable intermedia:  
let summary := $body/Order[1]/Summary[1]/  
$summary/Total, $summary/Status, $ summary/Shipping   
  
El uso de variables intermedias consume más memoria, pero minimiza la redundancia de procesamiento de XPath.

* **Usar un “hybrid Approach” en escenarios de solo lectura con Streaming**

Los beneficios de la transmisión pueden ser negativos si el payload es accedido un gran número de veces para la lectura de múltiples campos. Si todos estos campos se encuentran en una sola sub-sección del documento XML, un enfoque híbrido ofrece el mejor rendimiento.   
El enfoque híbrido incluye habilitar streaming a nivel de proxy y asignación de la sub-sección a una variable de contexto, Los campos individuales pueden ser accedidos desde esta variable de contexto.

* **Hacer Xqueries fuera de un loop “For”**

Si se necesita usar una acción “For-Each”, tratar de hacer los Xqueries fuera del loop, de lo contrario se está utilizando mucha más inicialización del motor de Xquery. Si es posible genere un único XML que contenga toda la información que se requiere y que sirva para procesar dentro del Loop

1. **Tuning de Transporte en (WLS y OSB)**
2. **HTTP**

* **Apagar el logging de acceso HTTP (habilitada por defecto).**

Si está activado, cualquier acceso HTTP resulta en 2 líneas escritas en los registros de acceso, que termina teniendo una sobrecarga de I/O. Además, el subsistema WLDF indexa los logs conllevando algunos ciclos de CPU.



**JMS**

* **Asegurar que este seteado el correcto nivel de persistencia de los destinos JMS.**
  + Para los escenarios no persistentes, de manera explícita desactivar la persistencia a nivel de servidor JMS, desmarcando la propiedad "Store Enabled". (Bajo la sección “Advanced” de la solapa General para el servidor JMS de la consola WLS).
  + También es posible sobre escribir el nivel de persistencia en el destino JMS. Para escenarios JMS persistentes, por lo general File Store funciona mejor que JDBC  
    store. Si hay varios servidores JMS involucrados, lo ideal sería crear en cada store  
    en un disco separado para una baja contención de I/O.
* **Usar Dipatch Policy para proxies con concurrencia**

Utilice una Dispatch Policy con una restricción de máximo de hilos. Establezca la restricción al mismo número de clientes. Por lo general que no se recomienda no tener más de 6 hilos por CPU. Si no se define la Dispatch policy, el número predeterminado de listeners por proxy JMS es de 16.

* **JMS Publish**

Si se necesita publicar en más de un destino diferente dentro de un flujo de servicio, se puede obtener una notable mejora si el publish se hace a través de un Java Callout a un Pojo. Esto es porque OSB solo cachea el último productor usado, así que si se publica en múltiples colas, el productor no es cacheado

**File / FTP / MQ (Transporte basado en polling)**

La latencia y el rendimiento de los transportes basados en poll dependen de la frecuencia con la  
que la fuente es polleada y el número de archivos y mensajes leídos en cada ciclo de barrido.

• Intervalo de polling: El intervalo default de polling para el transporte File y FTP se establece en 60 segundos. Use un intervalo más pequeño para escenarios de alto rendimiento, donde en  
el tamaño del mensaje no es muy grande y la CPU no está saturada.

• Límite de lectura: el límite de lectura determina el número de archivos y mensajes que se  
se leen por barrido. El valor por default es de 10. Si se configura un valor alto de limite de lectura y un valor bajo de polling interval puede producir que un gran número de mensajes sean simultáneamente leídos en memoria. Esto puede causar un OOM si los mensajes son de tamaño grande.

# Histórico de Cambios

| Versión | Fecha | Autor | Comentarios |
| --- | --- | --- | --- |
| V.Draft1 | 04/10/2010 | Ignacio Bernini | Sin documento previo |
| V.Draft1.1 | 09/02/2016 | Dennisse Gais | Actualización formato y nombre documento |