INTRODUCCIÓN AL MIDDLEWARE

Curso 2018

Grupo 7

Informe - Obligatorio 1

Autores:

Bruno Bradach

Bruno Rodao

Supervisor:

Prof. Guzman Llambias

Índice

1 Descripción del problema

1.1 Requerimientos

El banco ACME acaba de llevar a cabo la adquisición de dos bancos de plaza y se encuentran en proceso de transformación de sus procesos de negocio internos como consecuencia de esta operación. En ese sentido, ha decidido llevar a cabo el proyecto en tres etapas comenzando con una etapa piloto que involucra el proceso de pagos de sueldos desde su Portal Web.

La funcionalidad de pagos de sueldos permite a las empresas realizar el pago de sueldos de sus empleados hacia sus cuentas bancarias dentro del propio banco. Para ello, las empresas deben completar una planilla Excel los datos de los pagos, la cual permite generar un archivo XML que debe ser subido al sistema del banco desde su Portal Web. Las transferencias no son realizadas en el momento, sino que son llevadas a cabo por un proceso asincrónico encargado de procesarlas y almacenarlas en los sistemas internos del banco o los recientemente adquiridos.

Una vez procesados los pagos, su resultado quedará disponible desde la funcionalidad de consulta desde el Portal Web. El proceso de procesamiento de pagos está en desarrollo para incorporar los sistemas de los nuevos bancos adquiridos y se propone llevarlo a cabo mediante un middleware específico para ello.

El Portal Web se comunica con el middleware por intermedio de una cola de mensajes donde deposita el archivo XML con los pagos para su procesamiento. El middleware cuenta con un event-driven consumer para recibir los mensajes de la cola. Una vez recibido, este debe identificar qué pagos corresponden a qué sistema. Los pagos correspondientes a cuentas originarias del banco deben ser almacenados en su base de datos, mientras que los pagos asociados a las cuentas de los bancos adquiridos deben redirigirse a estos sistemas. El primero, cuenta con un viejo sistema (OldBank) que permite su comunicación mediante archivos CSV, mientras que el segundo es más moderno (NewBank) y cuenta con un Web Service SOAP de recepción de pagos. El Web Service de NewBank es de tipo request/response y requiere de autenticación http basic y TLS.

Para generar la respuesta hacia el portal Web, el middleware debe tomar la respuesta de los tres sistemas. La respuesta asociada a la base de datos es generada por el propio middleware en base a la identificación de la transacción siempre y cuando haya sido ejecutada con éxito. En caso de error se debe enviar un error genérico que no se pudo procesar la transacción. El sistema OldBank genera un archivo de respuesta que contiene el resultado de cada operación realizada, mientras que el sistema NewBank retorna en la propia respuesta del Web Service el resultado de cada operación realizada. El middleware debe ser capaz de recolectar todos los resultados, correlacionarlos con los pagos originales y generar un nuevo y único mensaje a enviar al Portal Web. La comunicación de la respuesta es también por intermedio de colas de mensajes donde el Portal Web cuenta con un polling-consumer para obtenerlas.

1.2 Requerimientos no funcionales

1. El Web Service deberá implementarse con las librerías JAX-WS. Como servidor se deberá usar Tomcat. Se deberán loggear todos los pagos procesados.
2. El Web Service debe recibir un conjunto de pagos. No se debe hacerse una invocación por pago.
3. El Portal Web se comunicará con la cola de mensajes usando librearías JMS estándares y no Spring JMS u otra tecnología. Deberá utilizarse ActiveMQ como servidor de colas de mensajes y deberá existir una cola de entrada (middleware-queue-in) y otra para recibir las respuestas (middleware-queue-out). Ningún pago deberá perderse ante una caída del servidor ActiveMQ.
4. El ruteador dentro del middleware deberá implementarse con un ruteador xpath de Spring Integration.
5. El sistema OldBank cuenta con dos carpetas donde recibir y responder el resultado del procesamiento de pagos: PAYIN y PAYOUT. Los archivos de entrada deben llamarse igual que el atributo Identificador referencia paquete, tanto los de entrada como salida.
6. La transformación de xml a csv debe hacerse con xslt. Otras transformaciones también pueden realizarse con este lenguaje, pero no es obligatorio. Se debe generar un único archivo por Identificador referencia paquete que incluya todos los pagos hacia este sistema.
7. El sistema debe tolerar el envío de pagos que incluyen los tres sistemas en un mismo paquete.
8. Se deben utilizar los EIPs aggregator, splitter y router provistos por spring integration.
9. Se debe utilizar Java 8 para todas las implementaciones.
10. Se debe utilizar PostgreSQL como motor de base de datos del sistema DataSys.
11. El sistema central debe correr sobre Tomcat 9 y estar implementado en Spring Integration 5.
12. El sistema debe soportar el procesamiento de archivos xml con hasta 1000 pagos

2 Arquitectura de la solución planteada

2.1 Interconexión entre Portal Web y Middleware

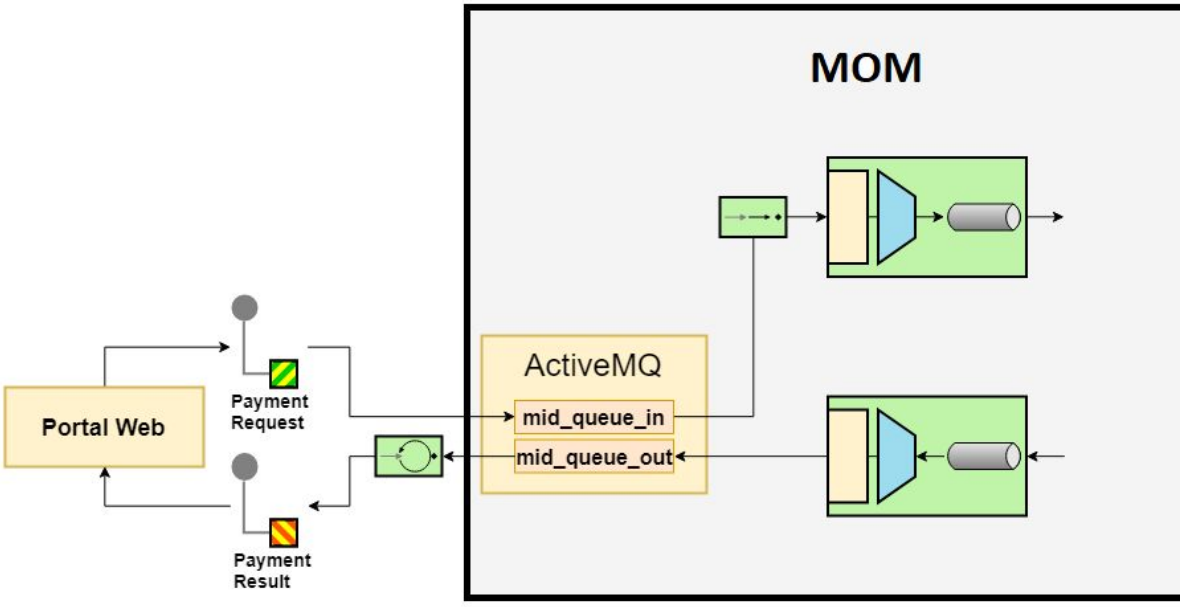

Fig 2.1.a: Interconexión Portal Web y MOM por medio de ActiveMQ

En la figura 2.1.a se muestra el modo de comunicación del Portal Web con el MOM.
El Portal Web se comunica con ActiveMQ utilizando la API JMS (Java Message Service). Desde el portal se carga un xml con los pagos, el cual es enviado a la cola 'mid_queue_in' de ActiveMQ a través de un MessageProducer de JMS. La aplicación Spring Integration (Acme-SI de ahora en más) obtiene el xml de la cola mediante un event-driven-consumer, es decir, ActiveMQ le envía los mensajes ni bien llegan a la cola siempre y cuando esté conectado.
Luego de procesar los pagos el MOM envía un xml con los resultados a la cola 'mid_queue-out' de ActiveMQ. El Portal obtiene la respuesta mediante un polling-consumer, es decir, invocando *receive()* en un MessageConsumer de JMS. Luego el resultado puede ser visto en pantalla mediante la funcionalidad de consulta.
La entrada hacia Acme-SI se realiza mediante un 'message-driven-channel-adapter' (JMS Message-Driven inbound Channel Adapter). La salida de Acme-SI hacia ActiveMQ se realiza mediante un 'outbound-channel-adapter' (outbound JMS Message-sending Channel Adapter).

2.2 Procesamiento de pagos de Acme-IS

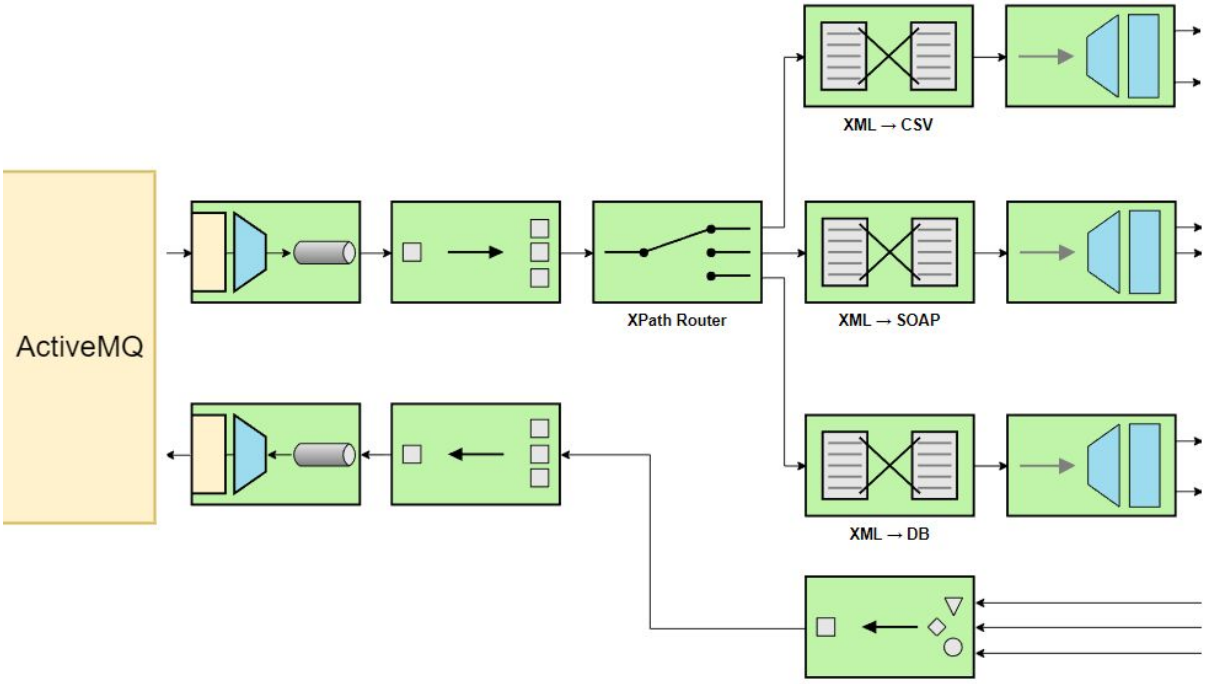

Fig 2.2.a: Procesamiento de xml de pagos en Acme-IS

Al ingresar un mensaje (xml de pagos) a Acme-IS, lo primer que se hace es dividir los pagos mediante un Splitter. El splitter divide los pagos según el banco al que deben ir (1=Acme, 2=NewBank, 3=OldBank). El splitter dividirá los pagos y reagrupará según banco. Para el caso de los bancos NewBank y OldBank la salida del splitter hacia el router será una lista de pagos. Para el caso de pagos del banco Acme, no se reagruparán sino que se enviarán de manera independiente. Esta decisión de que los pagos hacia BD de Acme no se reagruparan fue tomada en base al funcionamiento del gateway hacia BD que se verá luego. Para el splitter se utilizó XPath para obtener los conjuntos de pagos según banco. El splitter además agrega al header de cada mensaje el identificador del paquete de pagos de modo de que el aggregator al final del flujo pueda correlacionarlos.
El router XPath obtiene la salida del splitter y chequea el xml, observando el valor del elemento <bank> para dirigir los pagos hacia el Message Translator apropiado.
Hay tres translators, uno por banco. La conversión de XML a CSV se realizó mediante el uso de XSLT. En el caso de BD lo que hace el translator es generar un Map<String, Object> que será utilizado para realizar el insert por el gateway hacia BD. En el caso de Soap se optó por una conversión manual.
Luego del procesamiento de los pagos, las respuestas son enviadas a un Normalizer que transformará las distintas respuestas a xml y las enviará al Aggregator quien unirá las respuestas cuando tenga todas y las enviará al adaptador de salida hacia ActiveMQ.
El normalizer está coformado por un header-router y 3 translator. El header-router analizará el atributo 'type' y decidirá a que translator debe enviar el mensaje dependiendo de si 'type' es 'bd', 'soap' o 'csv'. Los translator se encargarán de la transformación a xml y enviarán el mensaje al Aggregator.

2.3 Interconexión entre Acme-IS y Bancos

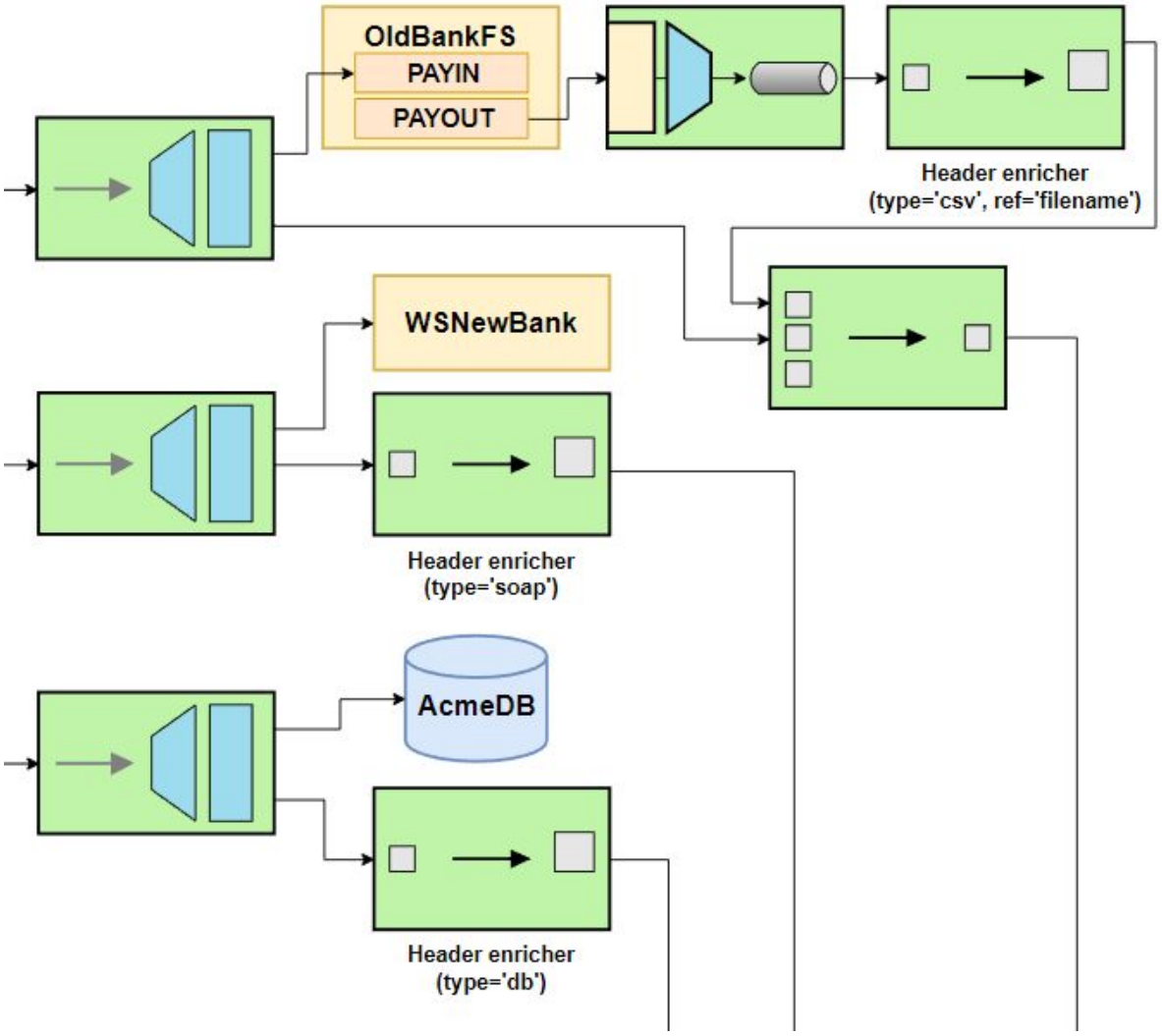

Fig 2.3.a: Interconexión entre Acme-IS, BD y Bancos externos

Luego de los translator de la entrada están los gateway de salida a los distintos sistemas (Acme BD, NewBank WS, OldBank FileSystem).
Para la comunicación con NewBank se utilizó un 'ws:outbound-gateway' (Gateway saliente de mensajería basado en Web Service) el cual hace una solicitúd al WS y luego envía en el payload un mensaje con la respuesta a la misma.
Para la inserción de pagos en la BD de Acme se utilizó un 'jdbc:outbound-gateway' (Gateway saliente de acceso a base de datos), el cual recibe un mensaje con un pago, realiza la inserción en BD y luego responde el resultado de la inserción junto con el 'id' del pago insertado.
Para la comunicación con OldBank se utilizó un 'file:outbound-gateway' (Gateway saliente que escribe el payload de mensajes a un archivo y luego responde con un mensaje que contiene el archivo escrito como payload). En el caso de OldBank, para no perder la referencia al mensaje original (el cual poseé los headers adecuados para ser correlacionado por el Aggregator), se utilizó un aggregator el cual espera a recibir 2 mensajes con el mismo 'packRefId'. El primero será la salida del gateway que escribió el archivo csv de entrada en la carpeta PAYIN y el segundo será el csv de respuesta obtenido

de la carpeta PAYOUT. Del primero nos interesan los headers y del segundo nos interesa el payload así que la respuesta del Aggregator de archivos hacia el normalizer será un nuevo mensaje con estos datos.
Para la lectura de archivos del OldBank de la carpeta PAYOUT se utilizó un 'file:inbound-channel-adapter' (Inbound Channel Adapter que hace polling a un directorio y envia mensajes cuyo payload son instancias de java.io.File). Se decidió ajustar el polling en 5 segundos dado que el proceso de generación de archivos csv de salida es llevado a cabo de forma manual.
A la salida de estos 3 gateways hay un enricher el cual agregará el atributo 'type' al header para que el header-router del normalizer pueda distinguir el tipo de archivo como se explicó en 2.2.

3 Casos de prueba

3.1 Casos de prueba ejecutados sobre el sistema

Id	Descripción	XML entrada	CSV OldBank	Resultado Esperado
CP01	Ingreso de 1 pago de cada banco.	'in_01.xml'	1234567891	Se inserta un pago en BD Acme. Se generea un archivo 1234567891.in en la carpeta PAYIN. El ws NewBank loguea el pago. Al consultar resultados desde el portal se obtienen los resultados de procesamiento de los 3 pagos.
CP02	Ingreso de mil pagos del banco 1.	'in_02.xml'	N/A	Se insertan mil pagos nuevos en BD. Al consultar desde el portal se obtienen los mil resultados de estos pagos.
CP03	Ingreso de mil pagos del banco 2.	'in_03.xml'	N/A	Se loguean mil pagos en WS NewBank, al consultar desde el portal se obtienen los mil resultados de estos pagos.
CP04	Ingreso de mil pagos del banco 3.	'in_04.xml'	1234567894	Se genera un archivo csv de nombre '1234567892.in' en carpeta PAYIN con la información de los mil pagos ingresados. Al poner el archivo csv de respuesta de nombre '1234567892' en la carpeta PAYOUT éste es procesado enviando la respuesta a ActiveMQ en a lo sumo 5 segundos más tiempo procesamiento. Al consultar desde el portal se deben ber los mil resultados de estos pagos.
CP05	Ingreso de 300 pagos de cada banco.	'in_05.xml'	1234567895	Igual a CP01 pero con 300 pagos por banco.