Integración de Aplicacións

Memoria

Carlos Franco Romero
Jesús Manuel Maneiro Figueroa

grupo ia016
Curso 2017-2018

Índice

Introducción

La práctica se divide en 2 partes, en la primera parte se diseñará e implementará un servicio web REST, y en la segunda parte se diseñará e implementará un flujo BPEL desde el que se invocará, entre otros, al servicio web desarrollado en la primera parte.

1 Parte 1: Servicio Web REST

Introducción

Para la realización de la primera parte de la práctica se ha optado por un diseño en capas, en el que la capa inferior le suministra un servicio a otra superior, facilitando así de este modo el mantenimiento, la escalabilidad y la tolerancia a fallos.

Esta aplicación está dividida en tres capas:

1. Capa Modelo:
   a) Capa de acceso a datos.
   b) Capa Lógica de Negocio.
2. Capa Servicios:
   a) Servicio Web Rest.
3. Capa Cliente: Utilizando línea de comandos que invoca parte de los servicios implementados.

La implementación de la aplicación se ha realizado bajo las especificaciones dadas en el enunciado de la práctica.

1.1 Capa Modelo

Equivalente en nuestra aplicación al módulo rs-deliveries-model. En esta capa se implementaría todo lo necesario para poder almacenar los datos en la base de datos así como acceder a dichos datos, es decir, entidades, DAOs y gestión de la persistencia pero por simplicidad se implementa el mock up MockDeliveryService que implementa DeliveryService. En esta práctica, los datos a manejar son sobre clientes (customers) y envíos (shipments). También se implementan los denominados casos de uso, que conciernen a la creación, eliminación, modificación y búsqueda de dichos customers y shipments.

1.1.1 Entidades del dominio

Las dos entidades que utilizamos en nuestra capa modelo son la siguientes: Customer y Shipment. Customer es la entidad utilizada para representar los clientes con todos sus atributos y métodos necesarios, mientras que Shipment es la que representa los envíos de dichos clientes con todos sus atributos. El estado del envío se gestiona con un enumerado llamado ShipmentState.

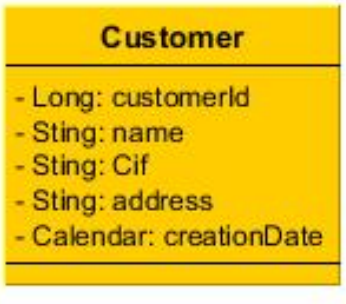

Figura 1

Figura 2

1.1.2 Interfaz del servicio

La interfaz DeliveryService que implementaremos en nuestro mock up es el siguiente:

Figura 3

1.2 Capa Servicio

Equivalente en nuestra aplicación al módulo rs-deliveries-service. Se implementan los DTOs y los conversores para CustomerDto, ShipmentDto y ShipmentStateDto.

1.2.1 DTOs

En la capa servicios utilizamos dos DTOs para el customer tres para el shipment y cuatro para excepciones con el fin de recibir y devolver datos de una forma más coherente, ocultando datos innecesarios para capas superiores.

- Dtos de Customer:

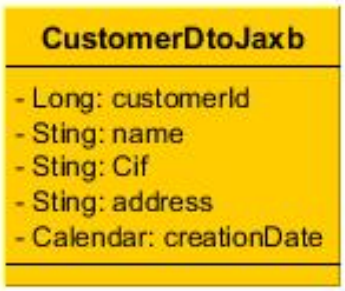

Figura 4

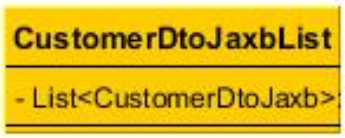

Figura 5

- Dtos de Shipment:

Figura 6

Figura 7

Figura 8

- Dtos de las Excepciones:

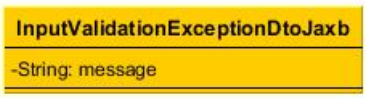

Figura 9

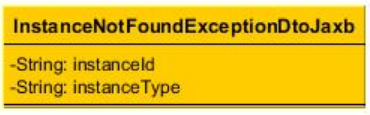

Figura 10

Figura 11

Figura 12

- Recursos definidos:

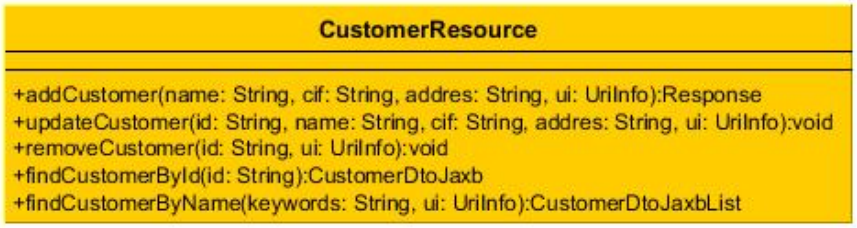

Figura 13

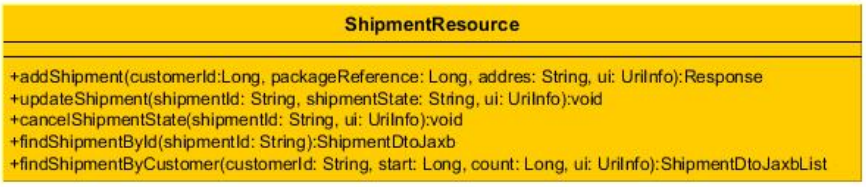

Figura 14

1.2.2 Peticiones REST

En el servicio REST se pueden invocar los casos de uso de la siguiente forma.
Casos de uso de Customer:

- **addCustomer**
  - URL: /customers
  - Método de invocación: POST.
  - Parámetros o DTO de entrada: 3 FormParam con los valores de name, cif y address y 1 Context con UriInfo.
  - DTO de respuesta: CustomerDtoJaxb.
  - Código HTTP de respuesta en caso de éxito: 201 CREATED.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST.
- **updateCustomer**
  - URL: /customers/id
  - Método de invocación: PUT.
  - Parámetros o DTO de entrada: 3 QueryParam con los valores de name, cif y address y 1 Context con UriInfo.
  - DTO de respuesta: -
  - Código HTTP de respuesta en caso de éxito: 204 NO_CONTENT.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST o 404 NOT_FOUND.
- **removeCustomer**
  - URL: /customers/id
  - Método de invocación: DELETE.
  - Parámetros o DTO de entrada: 1 Context con UriInfo.
  - DTO de respuesta: -
  - Código HTTP de respuesta en caso de éxito: 204 NO_CONTENT.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST, 404 NOT_FOUND o 409 CONFLICT.

- **findCustomerById**
  - URL: /customers/id
  - Método de invocación: GET.
  - Parámetros o DTO de entrada: -
  - DTO de respuesta: CustomerDtoJaxb.
  - Código HTTP de respuesta en caso de éxito: 200 OK.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST o 404 NOT_FOUND.
- **findCustomerByName**
  - URL: /customers
  - Método de invocación: GET.
  - Parámetros o DTO de entrada: 1 QueryParam "keywords" con el valor del nombre a buscar y 1 Context con UriInfo.
  - DTO de respuesta: CustomerDtoJaxbList.
  - Código HTTP de respuesta en caso de éxito: 200 OK.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST o 404 NOT_FOUND.

Casos de uso de Shipment:

- **addShipment**
  - URL: /shipments
  - Método de invocación: POST.
  - Parámetros o DTO de entrada: 3 QueryParam[1] con los valores de customerId, packageReference y address y 1 Context con UriInfo.
  - DTO de respuesta: ShipmentDtoJaxb.
  - Código HTTP de respuesta en caso de éxito: 201 CREATED.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST o 404 NOT_FOUND.
- **updateShipmentState**
  - URL: /shipments/shipmentId
  - Método de invocación: PUT.
  - Parámetros o DTO de entrada: 1 FormParam "state" con el valor del nuevo estado y 1 Context con UriInfo.
  - DTO de respuesta: -
  - Código HTTP de respuesta en caso de éxito: 204 NO_CONTENT.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST, 404 NOT_FOUND o 403 FORBIDDEN.
- **cancelShipmentState**
  - URL: /shipments/shipmentId
  - Método de invocación: POST.
  - Parámetros o DTO de entrada: 1 Context con UriInfo.
  - DTO de respuesta: -
  - Código HTTP de respuesta en caso de éxito: 204 NO_CONTENT.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST, 404 NOT_FOUND o 401 UNAUTHORIZED.

[1]Se utilizan QueryParam en vez de FormParam debido a las limitaciones de OpenESB en operaciones POST/PUT con parámetros en el cuerpo.

- **findShipmentById**
  - URL: /customers/id
  - Método de invocación: GET.
  - Parámetros o DTO de entrada: -
  - DTO de respuesta: ShipmentDtoJaxb.
  - Código HTTP de respuesta en caso de éxito: 200 OK.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST o 404 NOT_FOUND.
- **findShipmentByCustomer**
  - URL: /customers/id
  - Método de invocación: GET.
  - Parámetros o DTO de entrada: 3 Query Params “customerId”,“start” y “count” y 1 Context con UriInfo.
  - DTO de respuesta: ShipmentDtoJaxbList.
  - Código HTTP de respuesta en caso de éxito: 200 OK.
  - Código HTTP de respuesta en caso de error: 400 BAD_REQUEST o 404 NOT_FOUND.

Los códigos HTTP de respuesta anterirormente mencionados se corresponden a los siguientes casos:

- Mensajes de confirmación:
  - **200 - OK:** Usado en métodos GET para confirmar la obtención de forma correcta de un dato.
  - **201 - CREATED:** Usado en métodos POST porque crean tanto clientes como envíos.
  - **204 - NO_CONTENT:** Usado en métodos PUT y DELETE porque estos métodos no devuelven ningún DTO.
- Mensajes de error:
  - **400 - BAD_REQUEST:** Usado cuando se lanza la siguiente excepción:
    - **ImputValidationException:** Se lanza cuando los parámetros pasados a una función no son los correctos.
  - **401 - UNAUTHORIZED:** Usado cuando se lanza la siguiente excepción:
    - **ShipmentNotPendingException** Se lanza cuando se intenta cancelar un envío cuyo estado no es pendiente.
  - **403 - FORBIDDEN:** Usado cuando se lanza la siguiente excepción:
    - **InvalidStateException** Se lanza cuando se intenta cambiar el estado de un envío a otro al que no puede cambiar directamente.
  - **404 - NOT_FOUND:** Usado cuando se lanza la siguiente excepción:
    - **InstanceNotFoundException** Se lanza cuando se hace una petición de búsqueda que no encuentra ninguna coincidencia.
  - **409 - CONFLICT:** Usado cuando se lanza la siguiente excepción:
    - **CustomerWithShipmentsException** Se lanza cuando se intenta eliminar un cliente que tiene envíos.

1.3 Capa cliente y acceso al servicio

Equivalente en nuestra aplicación al módulo rs-deliveries-client. Se incluye el cliente de línea de comandos y se implementan las capas de acceso al servicio REST, con Xml y Json.

1.3.1 DTOs

Para el cliente utilizamos tres DTOs, ya que tiene accesible las funcionalidades de añadir y eliminar clientes, cambiar el estado de un envío y cancelar un envío.

Figura 15

Figura 16

Figura 17

1.3.2 Interfaz de la capa de acceso al servicio

Las siguientes operaciones son las funcionalidades de la aplicación que el cliente puede realizar desde la línea de comandos.

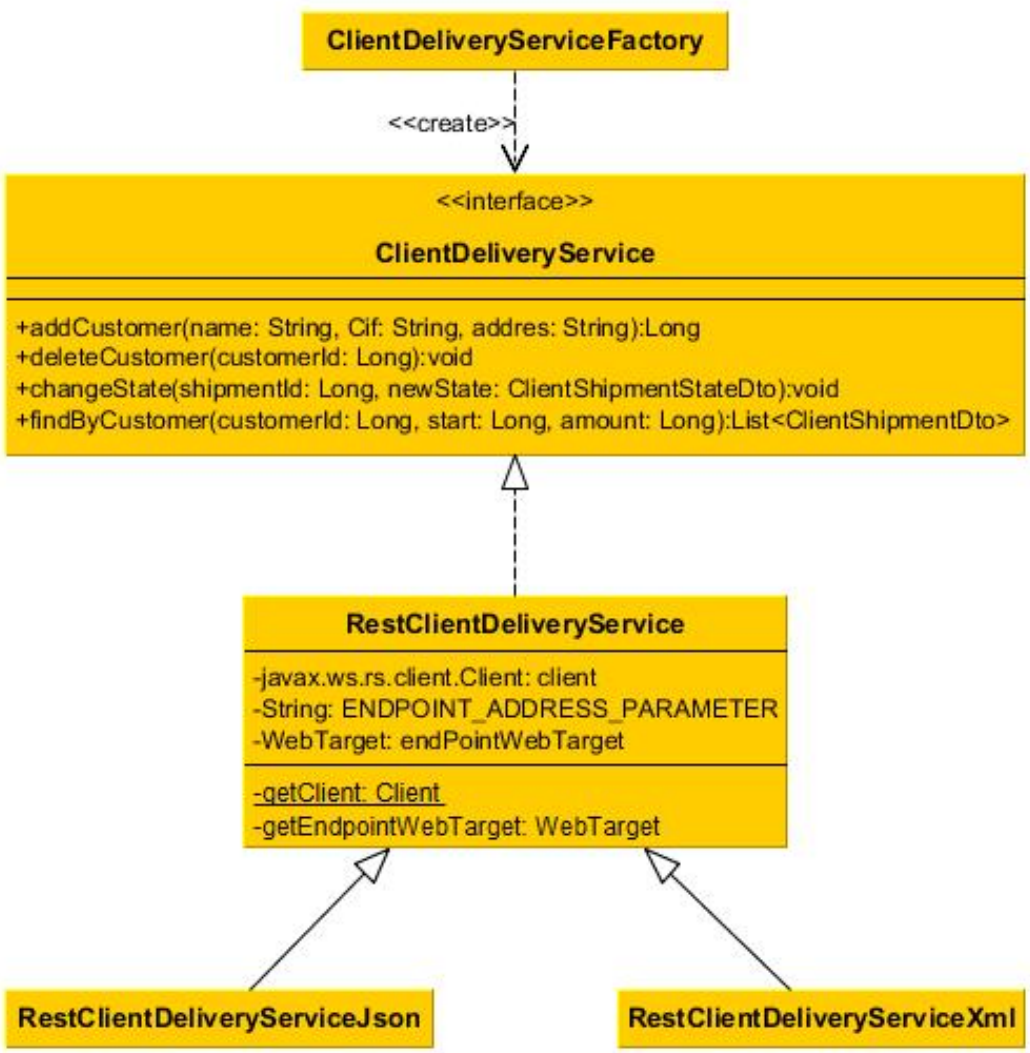

Figura 18

1.4 Parte Opcional

Parte pendiente para la iteración 3.

1.5 Problemas conocidos

Además del cambio realizado en el ShipmentResource, en addShipment debido las limitaciones de OpenESB anteriormente comentado. Se ha detectado un error con los códigos de error HTTP para las excepciones creadas, ya que no se comprueba si son o no cacheables.

2 Parte2: Flujo BPEL

Introducción

En esta parte se modela el proceso de negocio de servir pedidos utilizando el lenguaje BPEL. Implementando todas las especificaciones WSDL y BPEL necesarias utilizando OpenESB. En el flujo BPEL se invocará a varios servicios incluidos al servicio web REST de la Parte 1.

Se desarrolla para la empresa AcmeClub un servicio simplificado de un sistema automatizado de gestión de pedidos de sus clientes con un programa de puntos para conseguir ofertas y promociones en futuras compras.

La empresa dispone de los siguientes servicios:

- Servicio de CRM (CrmService). Permite obtener información de pedidos (a partir del identificador de un pedido).
- Servicio de fidelización (RewardService). Encargado de la gestión de los productos que forman parte del programa de fidelización. Este servicio dispone de las siguientes operaciones:
  - getProductReward(productId): integer.
  - isRewardMembership(customerId): boolean.
  - addMembershipReward(customerId, rewardPoints).
  - removeMembershipReward(customerId, rewardPoints).
  - getMembershipReward(customerId): long.
- Servicio de mensajeria (DeliveryService). Permite crear envíos a partir de la id del cliente, la referencia del paquete y la dirección de envío. Es el correspondiente al servicio REST desarrollado en la Parte 1.
  - addShipment(customerId, packageReference, address). Añade un envío a un cliente.

No se ha realizado la parte opcional.

2.1 Diseño

El diseño es el siguiente:

2.1.1 Flujo BPEL

El Flujo BPEL consta de 3 PartnerLink o socios de cliente (A la izquierda en la imagen del Flujo BPEL) que son servicios internos al flujo y 3 PartnerLink o socios de servicio (A la derecha en la imagen del Flujo BPEL) que son servicios externos al flujo.

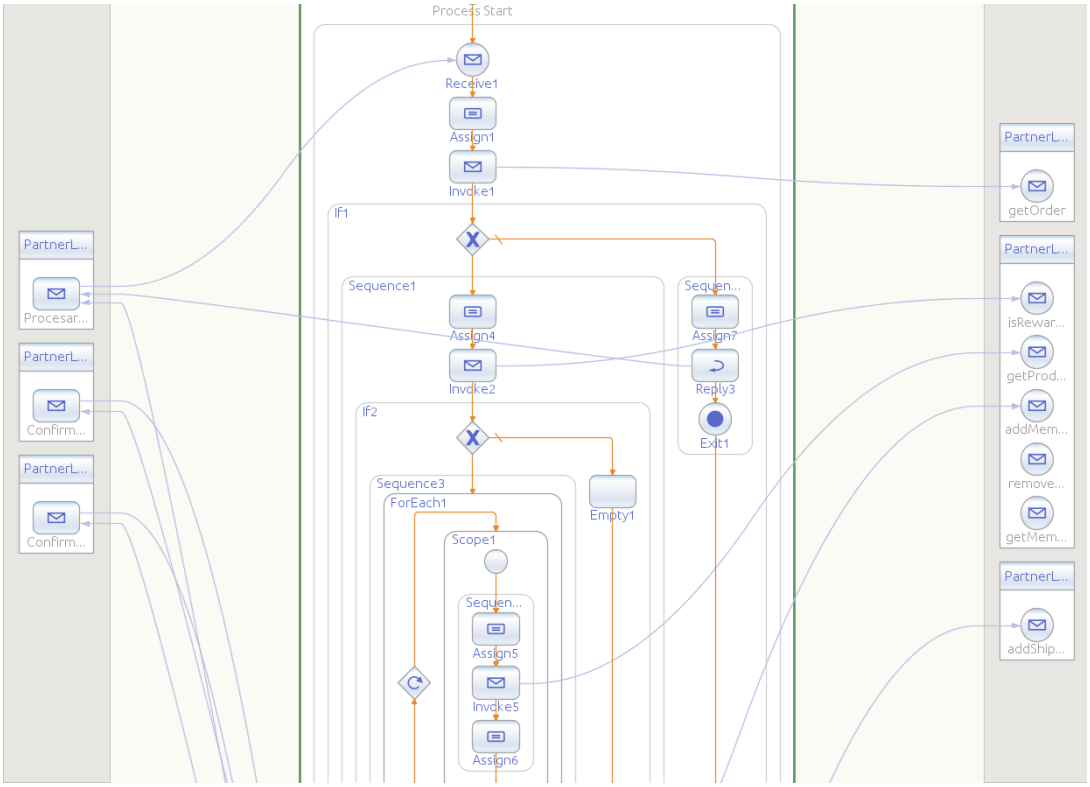

Figura 19

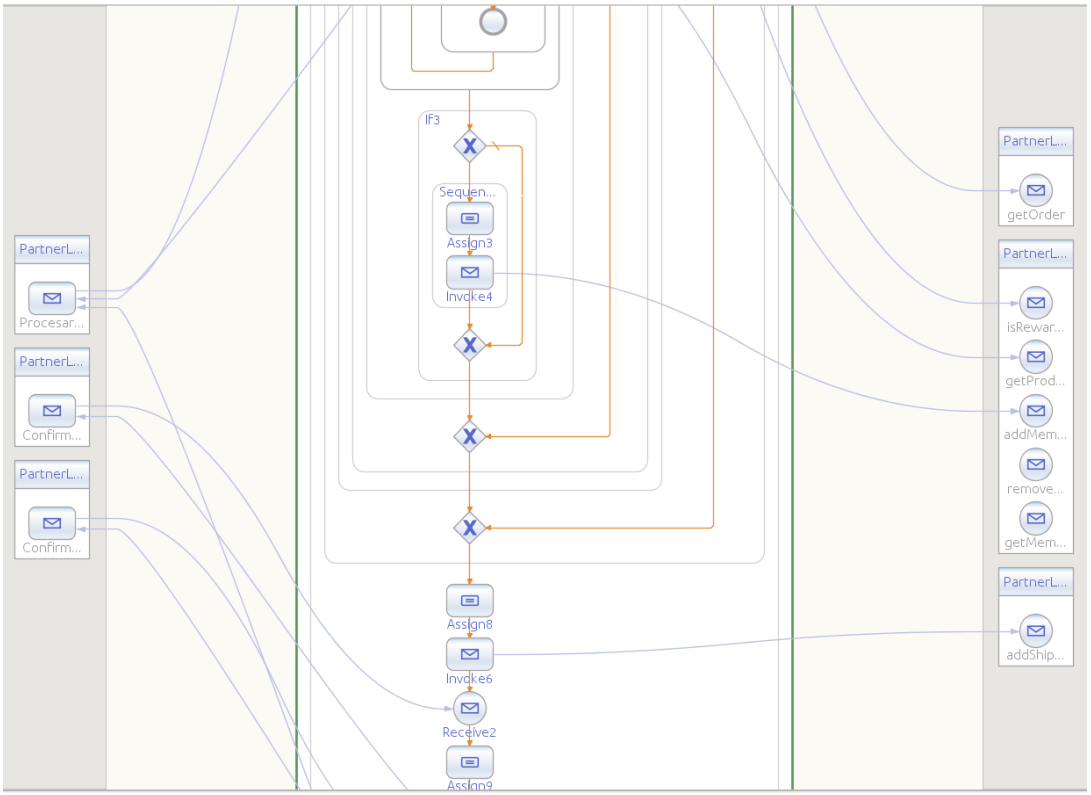
PartnerL...
Procesar...
PartnerL...
Confirm...
PartnerL...
Confirm...
If3
Sequen...
Assign3
Invoke4
Assign8
Invoke6
Receive2
Assign9
PartnerL...
getOrder
PartnerL...
isRewar...
getProd...
addMem...
remove...
getMem...
PartnerL...
addShip...

Figura 20

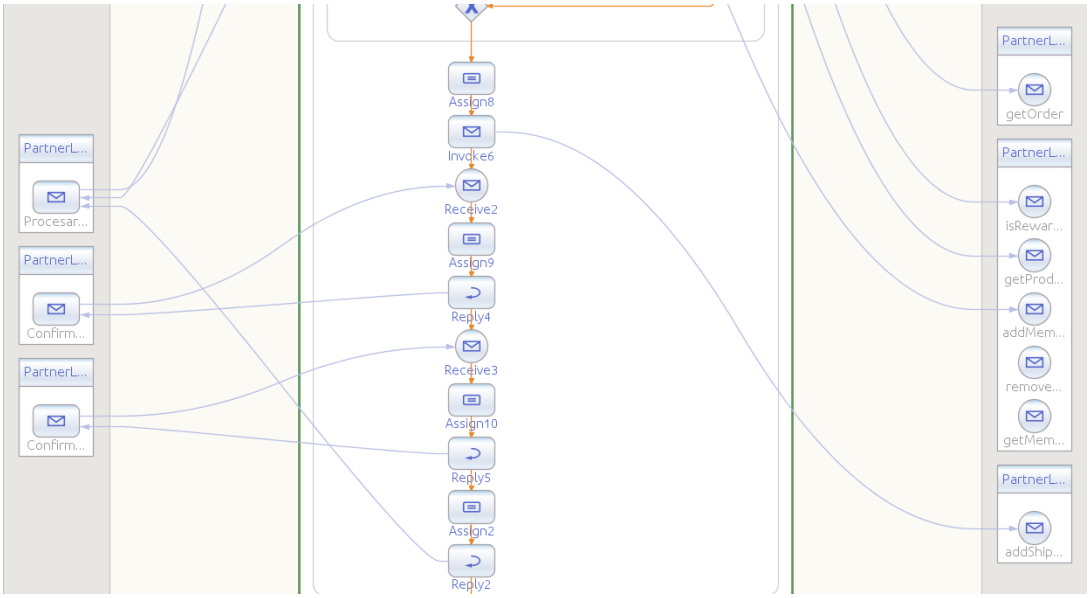
PartnerL...
Procesar...
PartnerL...
Confirm...
PartnerL...
Confirm...
Assign8
Invoke6
Receive2
Assign9
Reply4
Receive3
Assign10
Reply5
Assign2
Reply2
PartnerL...
getOrder
PartnerL...
isRewar...
getProd...
addMem...
remove...
getMem...
PartnerL...
addShip...

Figura 21

Estos socios son los siguientes:

- **PartnerLinkProcesarPedido:** Es uno de los tres socios que permite a un cliente interaccionar con el flujo Bpel. Tiene la siguiente operación:
  - ProcesarPedido: Devuelve la información final sobre un envio concreto.
    - Entrada: customerId: string, shipmentId: long.
    - Salida: delivered: boolean, customerPoints: long, delayed: boolean.
    - Fault: controlledException: string.
- **PartnerLinkConfirmarEnvío:** Es uno de los tres socios que permite a un cliente interaccionar con el flujo Bpel. Tiene la siguiente operación:
  - ConfirmarEnvío: Confirma que un pedido ha sido enviado y las horas que tardará ese envío en ser entregado
    - Entrada: shipmentId: long, estimatedHours: long.
    - Salida: part1: string.
- **PartnerLinkConfirmarEntrega:** Es uno de los tres socios que permite a un cliente interaccionar con el flujo Bpel. Tiene la siguiente operación:
  - ConfirmarEntrega: Confirma que un pedido ha sido entregado y las horas que ha tardado ese envío en ser entregado
    - Entrada: shipmentId: long, realHours: long.
    - Salida: part1: string.
- **PartnerLinkCrmProviderService:** Es el socio encargado de interactuar con el servicio SOAP externo de pedidos "Order". Los pedidos están formados por: customerId, orderId, adress, creationDate, orderLines y zipCode. Las orderLines están formadas por productos que tienen productId, name, description, prices y amount. Tiene la siguiente operación:
  - getOrder: Devuelve la información sobre un pedido concreto.
    - Entrada: orderId:long.
    - Salida: Order.

- **PartnerLinkRewardProviderService:** Es el socio encargado de interactuar con el servicio externo SOAP de recompensas. Tiene las siguientes operaciones:
  - isRewardMembership: Dice si un cliente forma parte del sistema de puntos.
    - Entrada: customerId: string.
    - Salida: return: boolean.
  - getProductReward: Devuelve los puntos de un determinado producto.
    - Entrada: productId:long.
    - Salida: return: integer.
  - addMembershipReward: Añade unos puntos a un cliente.
    - Entrada: customerId: string, rewardPoints:integer.
    - Salida: return: long.
  - removeMembershipReward, rewardPoints). Elimina unos puntos a un cliente.
    - Entrada: customerId: string, rewardPoints:integer.
    - Salida: return: long.
  - getMembershipReward: Devuelve los puntos de un determinado cliente
    - Entrada: customerId: string.
    - Salida: return: long.
- **PartnerLinkDeliveriesService:** Es el socio encargado de interactuar con el servicio REST implementado en la parte 1. Tiene la siguiente operación:
  - addShipment: Crea un nuevo envío.
    - Entrada: Se usa la petición HTTP
    - Salida: Shipment

El flujo BPEL forma parte de los siguientes componentes:

- Receive1: Recibe la información del socio ProcesarPedido
- Assign1: Asigna el valor del shipmentId del ProcesarPedidoOperationIn a orderId de GetOrderIn.
- Invoke1: Invoca a la operación GetOrderIn.
- If1: Comprueba si los customerId de ProcesarPedidoOperationIn y de GetOrderOut son los mismos.
- Assign4: Asigna el valor del customerId del ProcesarPedidoOperationIn a customerId de IsRewardMembershipIn y el valor 0 a totalPoints de GetProductRewardIn.
- Invoke2: Invoca a la operación IsRewardMembership
- If2: COmprueba el boolean resultante de IsRewardMembershipOut
- ForEach1: Recorre todas las orders de orderLines.
- Assign5: Asigna el productId de la order correspondiente de orderLine a productId de GetProductRewardIn
- Invoke5: Invoca a la operación GetProductReward.
- Assign6: Multiplica los puntos resultantes de GetProductRewardOut por el amount e la order correspondiente, se los suma el totalpoints y se asignan a totalpoints.
- If3: Compruena si totalpoints es mayor que 0.
- Assign3: Asigna la variable totalpoints y customerId de PocesarPedidoOperationIn a rewardPoints y a customerId de AddMembershipRewardIn.
- Invoke4: Invoca la operación AddMembershipReward.
- Empty1: No hace nada si no se cumple el If2.
- Assign7: Asigna un stringLiteral ala variable controlledException de Fault1FaultVar si no se cumple el If1.
- Reply3: Envia la Fault a la ProcesarPedidoOperation.
- Ext1: Termina la ejecución del flujo.

- Assign8: Concatena los parametros necesarios para realizar addShipment a un string y se los pasa a HTTP Request Parameters de AddShipmentIn.
- Invoke6: Invoca a AddShipment.
- Receive2: Recibe la información del socio ConfirmarEnvío.
- Assign9: Crea un string informando que se ha enviado un envío determinado y las horas estimadas.
- Reply4: Contesta al socio ConfirmarEnvío con el mensaje creado en Assign9.
- Receive3: Recibe la información del socio ConfirmarEntrega.
- Assign10: Crea un string informando que se ha recibido un envío determinado y las horas reales.
- Reply5: Contesta al socio ConfirmarEntrega con el mensaje creado en Assign10.
- Assign2: Comprueba si el tiempo real de entrega ha sido mayor que el tiempo estimado(realHours de ConfirmarEntregaOperationIn y estimatedHours de ConfirmarEnvioOperationIn) para asignar el resultado a la variable delayed, asigna la variable totalpoints a customerPoints y la variable delivered de ConfirmarEntregaOperationIn a delivered de ProcesarPedidoOperationOut.
- Reply2: Contesta al socio ProcesarPedido con los parámetros creados en Assign2.

2.1.2 Composite Application

En la Composite Application se crean los bindings concretos para cada Partner Link definido de manera abstracta en el flujo BPEL. Se sitúan a la izquierda de la imagen los bindings WSDLPorts y en el centro el módulo JBI (como se puede ver en la figura 22.

En la figura 22 se pueden ver además las correspondencias entre los puertos del WSDL de cada servicio con las entradas o salidas de los socios presentes en el flujo BPEL. Se ven también las relaciones de los bindings con el módulo JBI a través de flechas.

Las flechas verdes son la entradas de los datos y las moradas las salidas de los datos. Por cada entrada de los servicios externos, hay una salida para el flujo.

Se muestra a continuación el diagrama que representa la Composite Application:

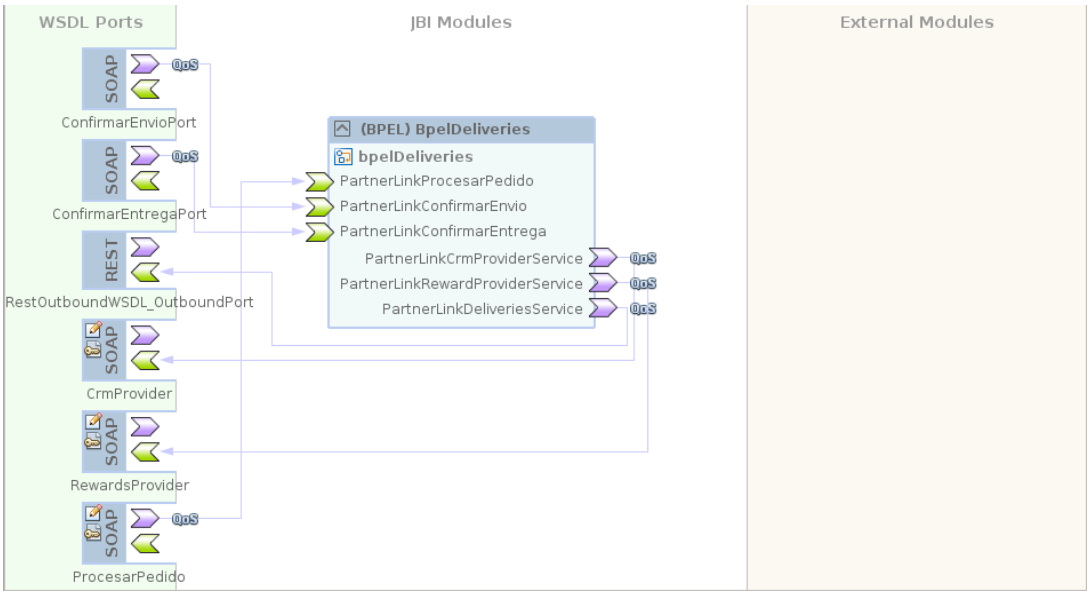

Figura 22

2.2 Parte Opcional

Pendiente para it-3.

2.3 Problemas Conocidos

Ninguno.