Desarrollo de Sistemas de Software Basados en Componentes y Servicios

Máster en Ingeniería Informática

Práctica 2

Orquestación de servicios Web utilizando WS-BPEL

Autor:
Pablo Valenzuela Álvarez (pvalenzuela@correo.ugr.es)

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 14 de diciembre de 2024

Índice

Índice de figuras

1. FlightApp

1.1. Estructura del compuesto SOA

La estructura del ejercicio consta de cuatro procesos BPEL: tres para los procesos correspondientes al empleado y las compañias aéreas, y otro para la gestión del servicio (ver figura 1).

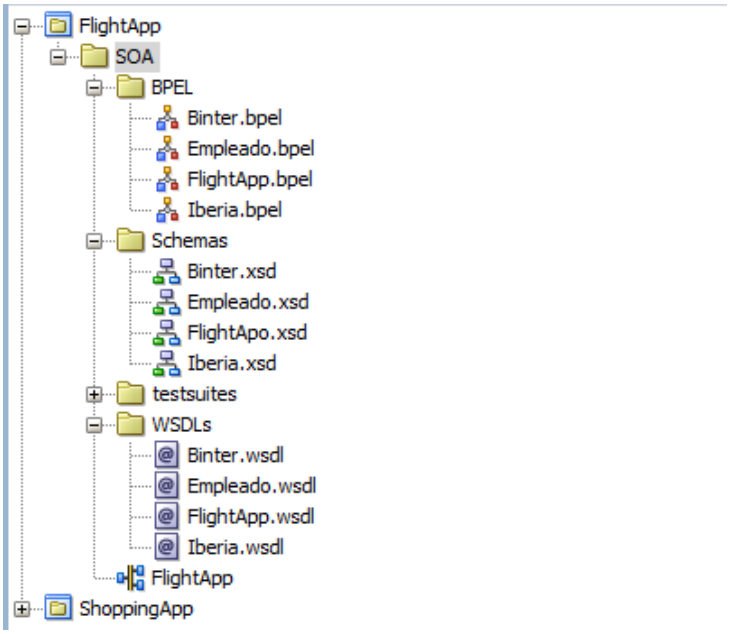

Figura 1: Estructura del proyecto FlightApp.

Los esquemas que hemos usado en los procesos contienen las siguientes entradas y salidas (IN/OUT):

- **Empleado**: nombre(IN), clase(OUT)
- **Binter**: clase_empleado(IN), aeropuerto(IN), fecha_ida(IN), fecha_vuelta(IN), precio(OUT)
- **Iberia**: clase_empleado(IN), aeropuerto(IN), fecha_ida(IN), fecha_vuelta(IN), precio(OUT)
- **FlightApp**: empleado_nombre(IN), aeropuerto(IN), fecha_ida(IN), fecha_vuelta(IN), empleado_nombre(OUT), empleado_clase(OUT), aerolinea(OUT), precio(OUT)

1.2. Roles de los procesos BPEL

A continuación, pasaremos a describir los roles de cada uno de los procesos BPEL.

1.2.1. Empleado.bpel

Este proceso asigna una clase al empleado según el nombre:

- **Pablo** → Primera clase
- **Juana** → Segunda clase
- **Otro** → Tercera clase

Se puede ver su funcionamiento en la figura 2.

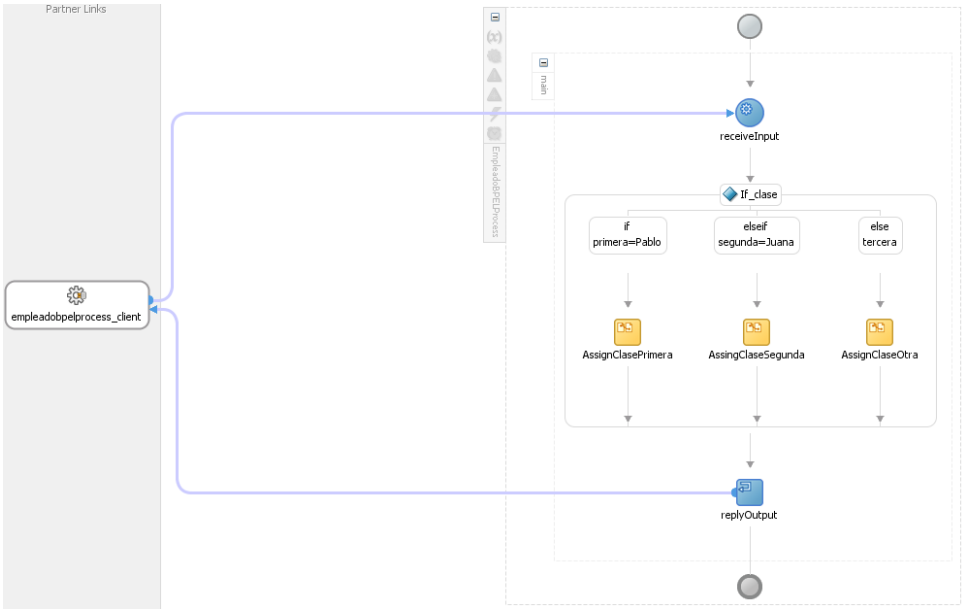

Figura 2: Proceso BPEL de empleado.

1.2.2. Binter.bpel e Iberia.bpel

Estos dos procesos se comportan de la misma manera: reciben la clase del empleado, el aeropuerto de salida y las fechas de ida y vuelta, y retornan el precio que debe pagar el empleado por volar con las especificaciones suministradas.

La figura 3 muestra el siguiente funcionamiento:

1. Comprueba que las fechas sean correctas: que la ida no sea mayor que la vuelta, o que la ida ya haya pasado.
2. Fija el precio de la clase del empleado.
3. Y por último, asigna el precio según el aeropuerto.

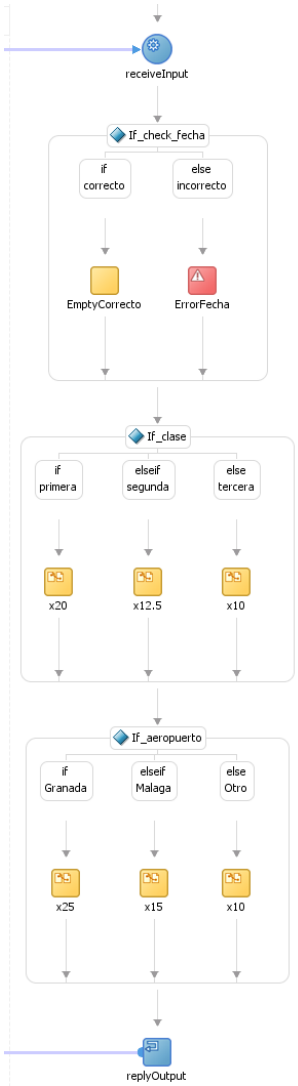

Figura 3: Proceso BPEL de Binter.

1.3. Interconexión de servicios

En la figura 4 se muestra como hemos realizado la interconexión de servicios entre los procesos BPEL. El que hemos llamado "FlightAppBPELProcess" es el encargado de invocar a los demás procesos y generar la respuesta final.

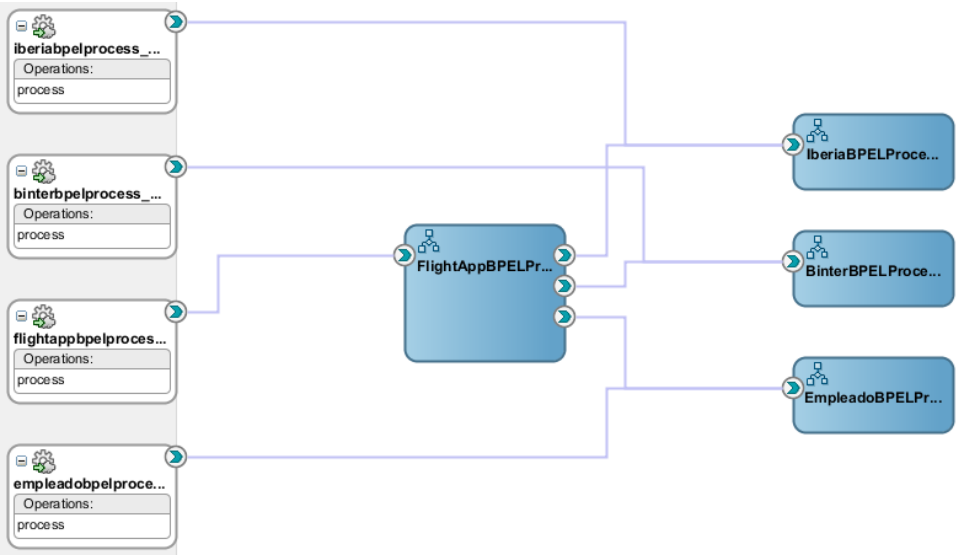

Figura 4: Interconexión de servicios del proyecto FlightApp.

En la figura 5 tenemos como se asigna la clase al empleado. Asignamos la variable nombre al proceso BPEL de empleado para después invocarlo y obtener la clase.

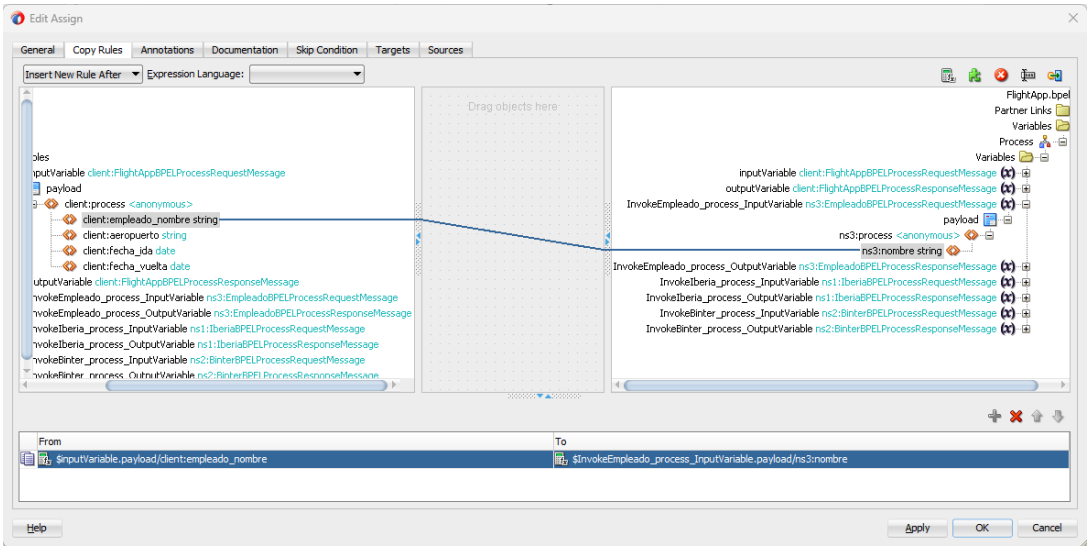

Figura 5: Obtención de la clase del empleado.

Después, como se puede ver en la figura 6, asignamos las demás variables del proceso BPEL de Iberia, para después invocarlos (el proceso para asignar las variables de Binter es el mismo).

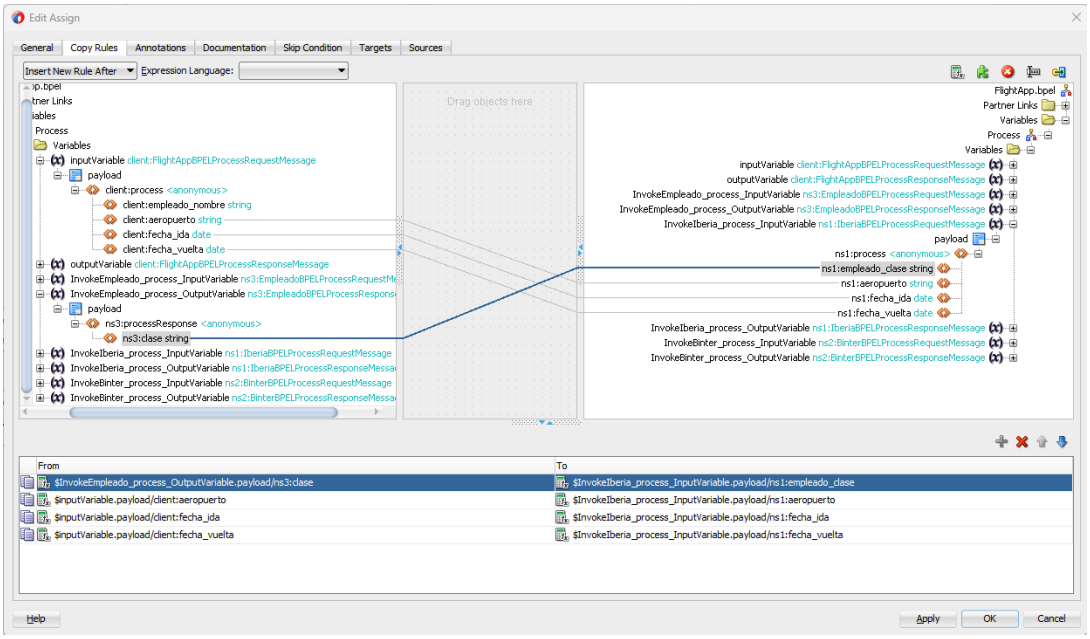

Figura 6: Obtención de las demás variables.

Y por último, comparamos ambos precios (ver figura 7). La lógica del proceso hace que se nos muestre sólo como respuesta final la mejor oferta.

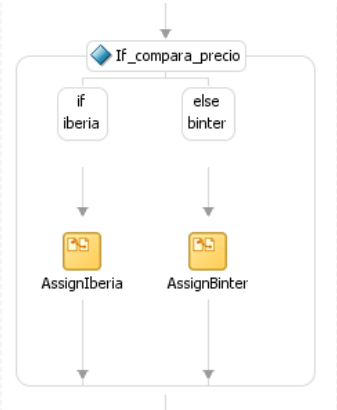

Figura 7: Comparación del mejor precio.

1.4. Prueba de servicios

Para acabar con el primer ejercicio, vamos a probar el servicio. Las siguientes figuras muestran el correcto funcionamiento del servicio, y como retorna el mejor precio según las especificaciones aportadas por el empleado. Como detección de errores (ver figura 13), hemos probado a introducir una fecha de vuelta menor que la de ida.

Figura 8: Prueba de servicio FlightApp 1.

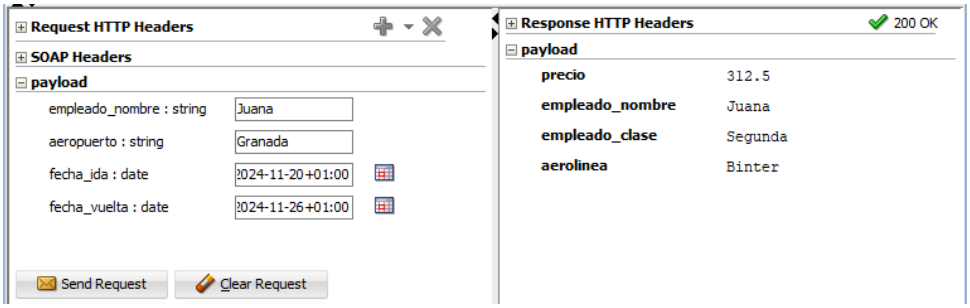

Figura 9: Prueba de servicio FlightApp 2.

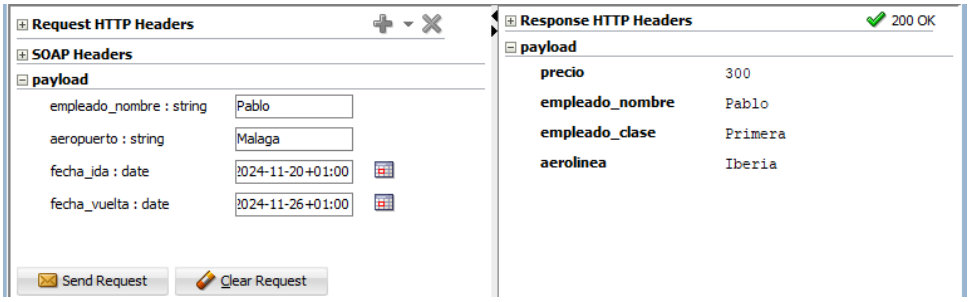

Figura 10: Prueba de servicio FlightApp 3.

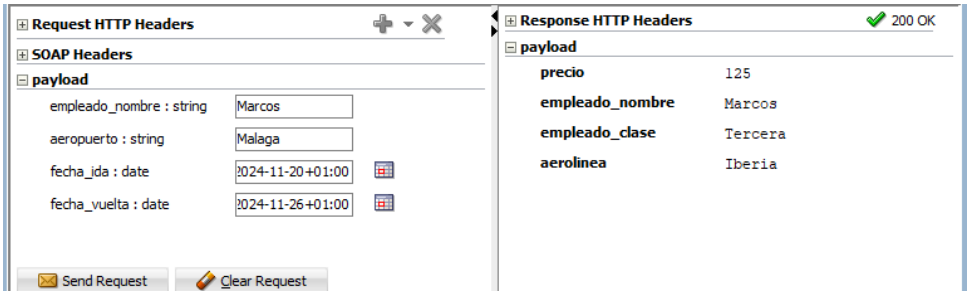

Figura 11: Prueba de servicio FlightApp 4.

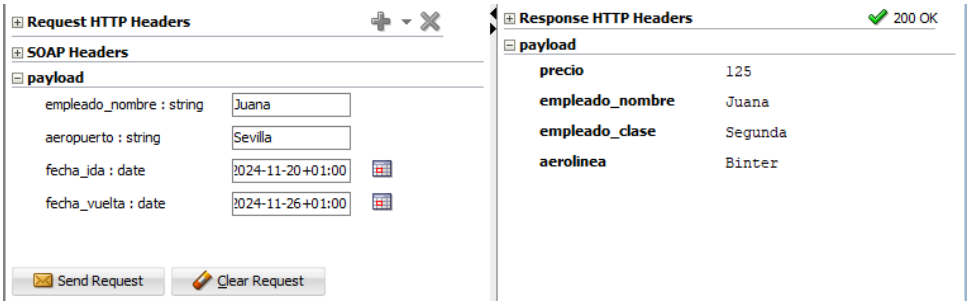

Figura 12: Prueba de servicio FlightApp 5.

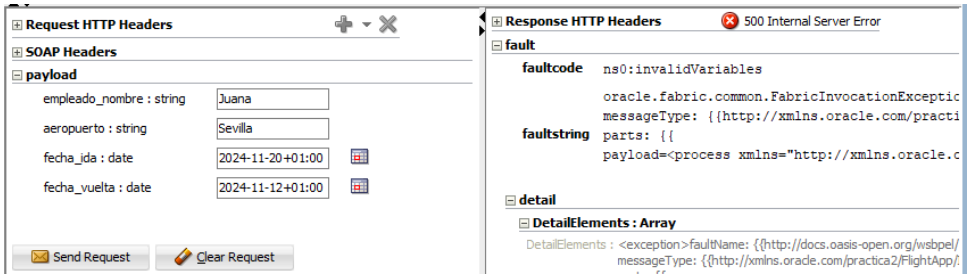

Figura 13: Prueba de servicio FlightApp 6.

2. ShoppingApp

2.1. Estructura del compuesto SOA

La estructura del segundo ejercicio consta de cuatro procesos BPEL (ver figura 14). En el proceso **VerPrecios** nos dirá si hay o no stock para un producto dado. Los procesos **Comprador** y **Vendedor** se encargarán de ir regateando precios hasta obtener un precio final. Y el último proceso (**ShoppingApp**) es el que gestionará la lógica del problema.

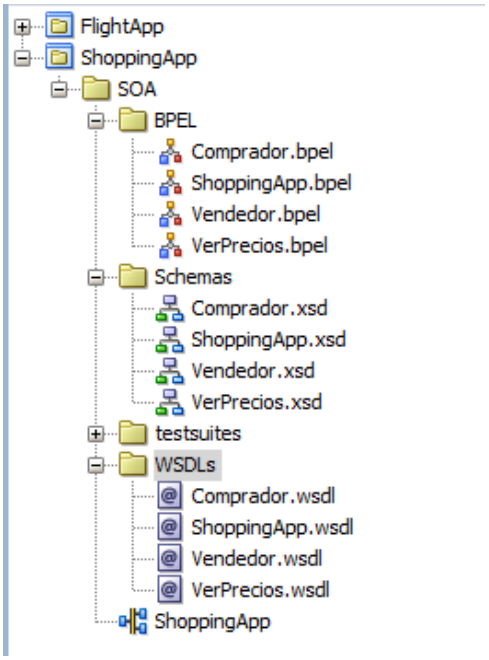

Figura 14: Estructura del proyecto ShoppingApp.

Los esquemas correspondientes a las entradas y salidas (IN/OUT) de estos procesos son:

- **VerPrecios**: producto(IN), precio(OUT), enStock(OUT)
- **Comprador**: precio_ofrecido(IN), precio_inicial(IN), comprar(OUT)
- **Vendedor**: oferta_recibida(IN), precio_ofrecido(OUT)
- **ShoppingApp**: producto(IN), producto(OUT), precio_inicial(OUT), precio_final(OUT), mensaje(OUT)

2.2. Roles de los procesos BPEL

2.2.1. VerPrecios.bpel

El proceso **VerPrecios** recibe el nombre de un producto y, mediante el if que se puede ver en la figura 15, asigna el precio y si está en stock. Si el producto buscado no esta en stock, la variable **enStock** obtendrá el valor de *false*, y el producto en cuestión no tendrá precio.

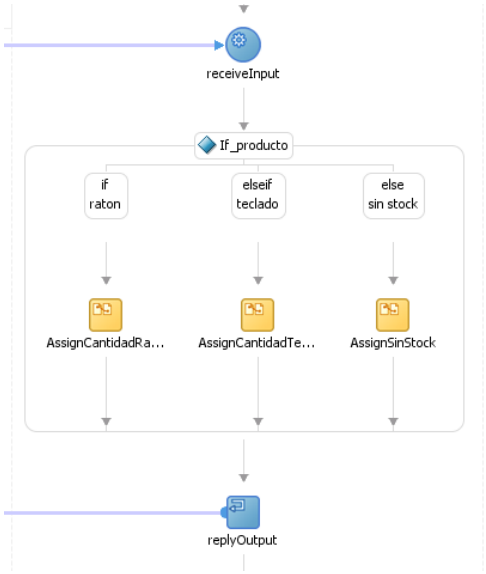

Figura 15: Proceso BPEL VerPrecios.

2.2.2. Comprador.bpel

Este proceso se limita a comparar el precio ofrecido por el vendedor. Si la condición del *if* se cumple, se efectúa la compra (ver figura 16).

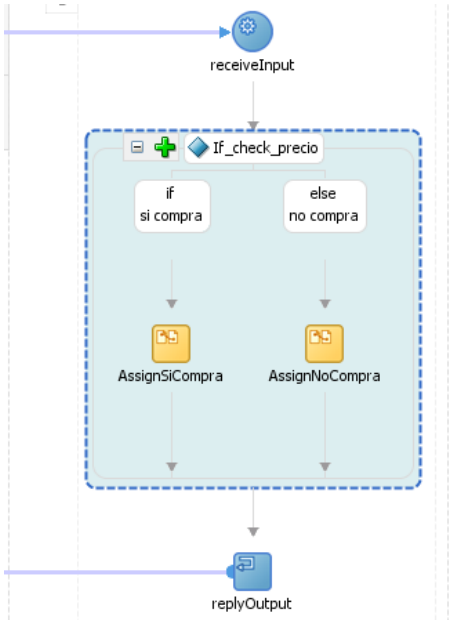

Figura 16: Proceso BPEL Comprador.

La condición de este *if* se acepta cuando el precio ofrecido por el vendedor es más bajo que el 70 % del precio original del producto.

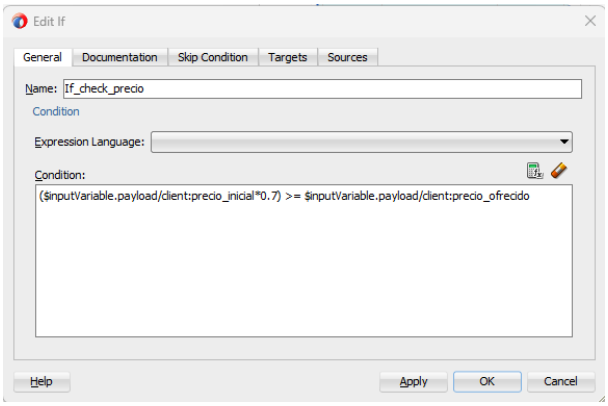

Figura 17: Funcionamiento interno del IF del proceso Comprador.

2.2.3. Vendedor.bpel

El único funcionamiento de este proceso es reasignar el precio dado por el comprador y devolverlo en forma de contraoferta (ver figura 18).

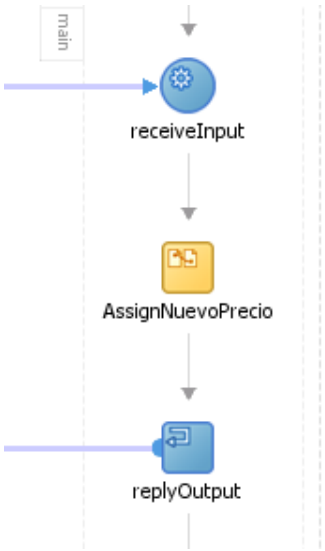

Figura 18: Proceso BPEL Vendedor.

La asignación se hace como se muestra en la figura 19. Basicamente, el vendedor hace un descuento del 10 % cada vez que recibe una oferta del comprador.

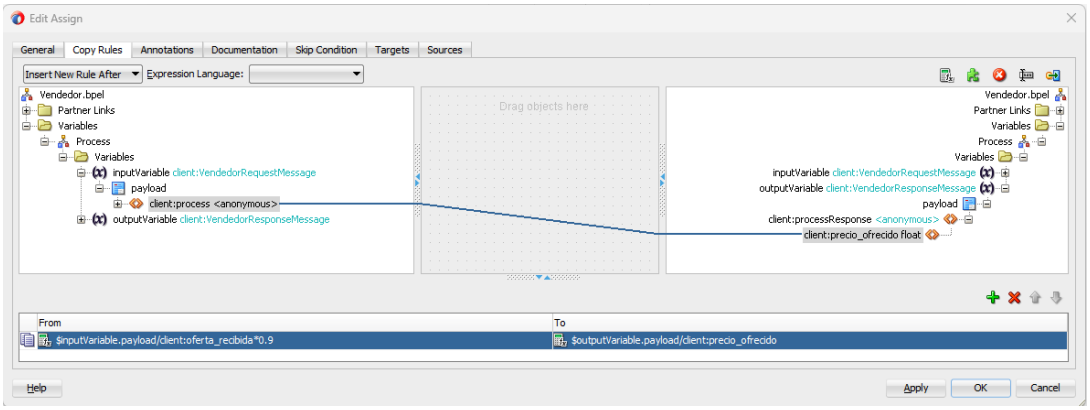

Figura 19: Funcionamiento interno del ASSING del proceso Vendedor.

2.3. Interconexión de servicios

En la figura 20 mostramos la interconexión entre los servicios del proyecto. Todos se invocan a través de *ShoppingApp*, que será el encargado de realizar la lógica del problema.

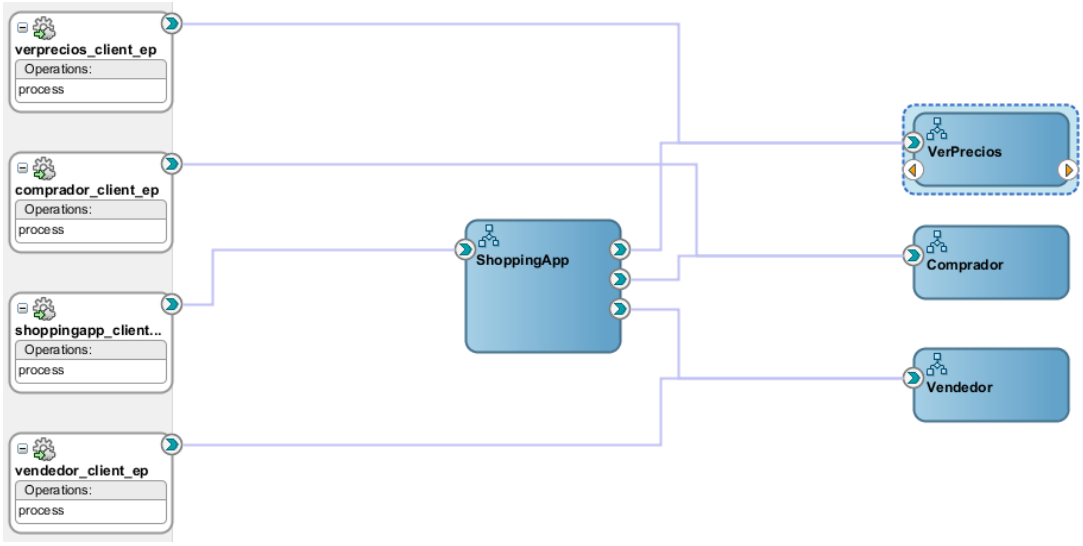

Figura 20: Interconexión de servicios del proyecto ShoppingApp.

El proceso que seguirá será el siguiente:

1. Asigna el precio y el stock del producto buscado (ver figura 21).
2. Comprueba si hay stock del producto. Si no, devuelve el mensaje “Sin Stock” (ver figura 22)
3. Realiza un bucle *while*, en el que se el comprador y el vendedor regatean precios. En el momento en el cual se cumpla la condición impuesta en el proceso **Comprador** (el producto rebajado al 70 %), se resuelve el bucle y se produce la compra (ver figura 23).

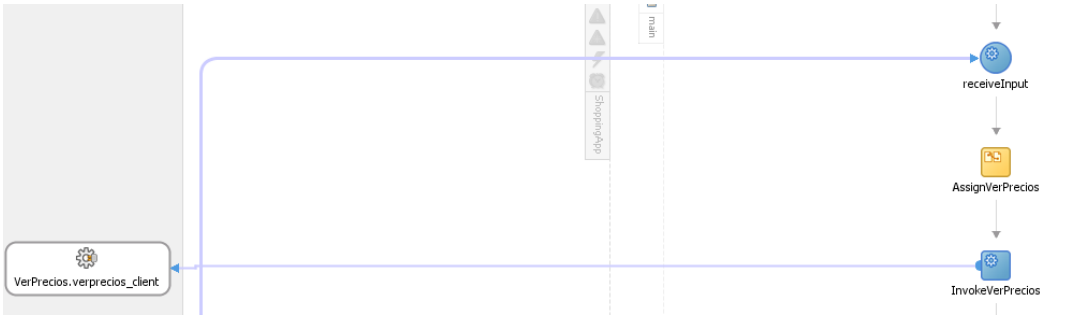

Figura 21: Primer paso de la lógica de ShoppingApp.

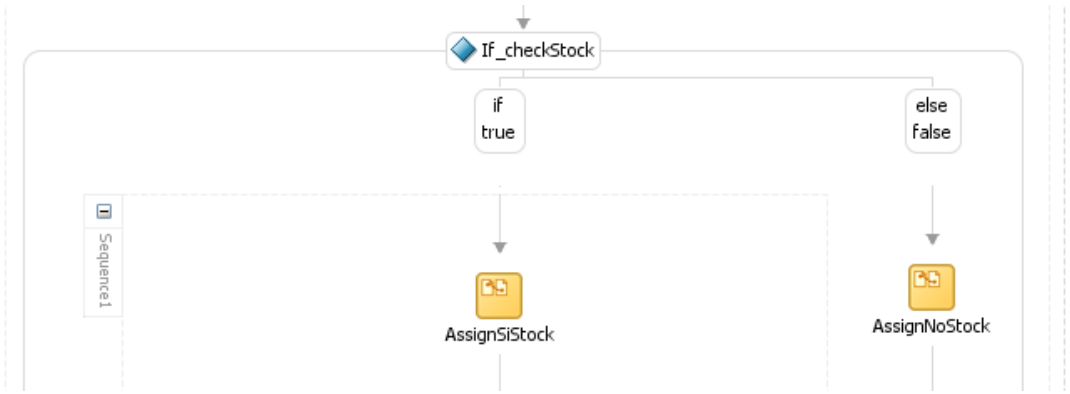

Figura 22: Segundo paso de la lógica de ShoppingApp.

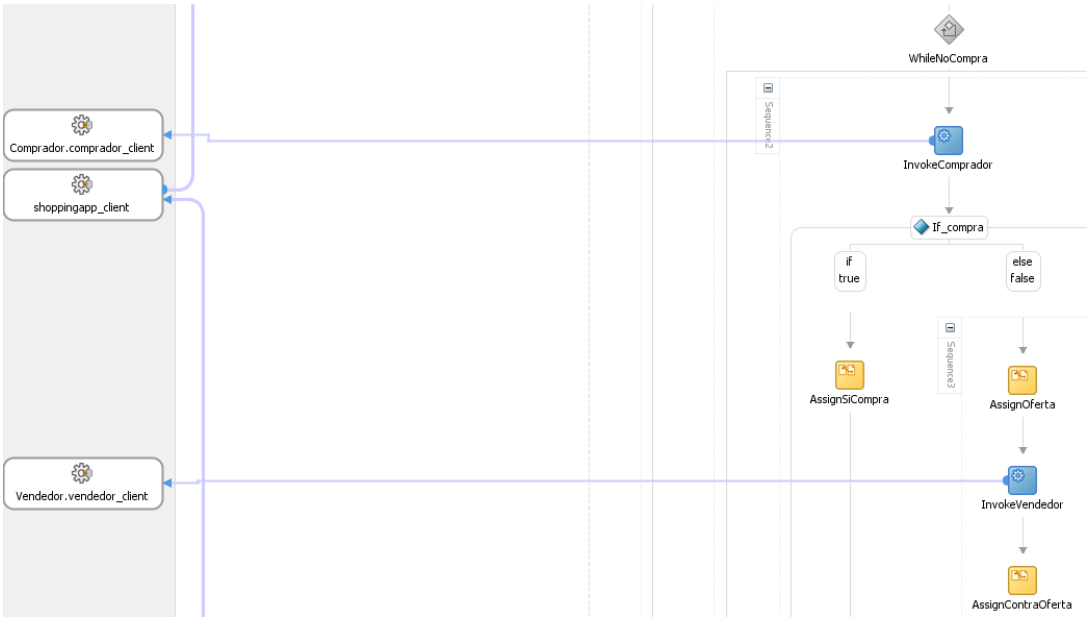

Figura 23: Tercer paso de la lógica de ShoppingApp.

2.4. Prueba de servicios

Para finalizar el ejercicio, vamos a probar el funcionamiento del servicio *ShoppingApp*. En la figura 24, vemos el precio al que el comprador compra el producto "ratón". Sabiendo que el precio inicial de este producto es de 10, el comprador ha "esperado" a que el precio estuviese por debajo de 7.

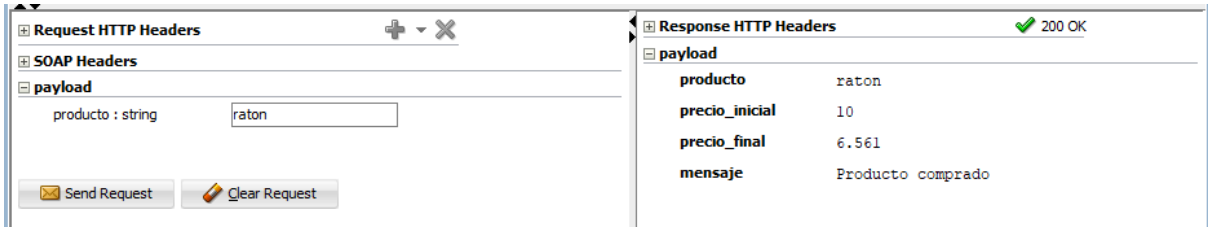

Figura 24: Prueba de servicio ShoppingApp 1.

Lo mismo ocurre con el producto "teclado" (ver figura 25). El precio inicial era de 25 y su 70 % es 17.5, por lo que el comprador ha efectuado su compra cuando el precio ha bajado de esa cantidad.

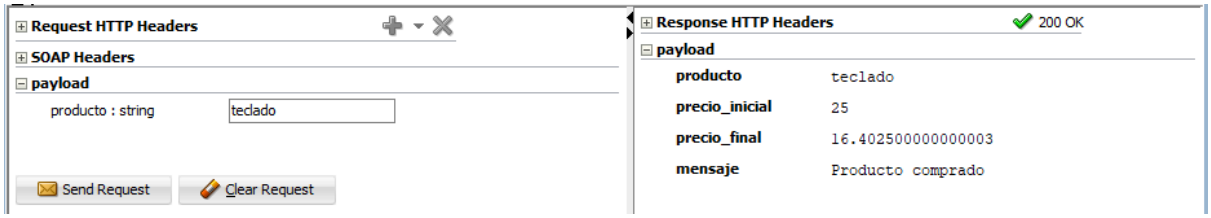

Figura 25: Prueba de servicio ShoppingApp 2.

Y en la última figura (figura 26), comprobamos que pasa cuando no hay stock de un producto.

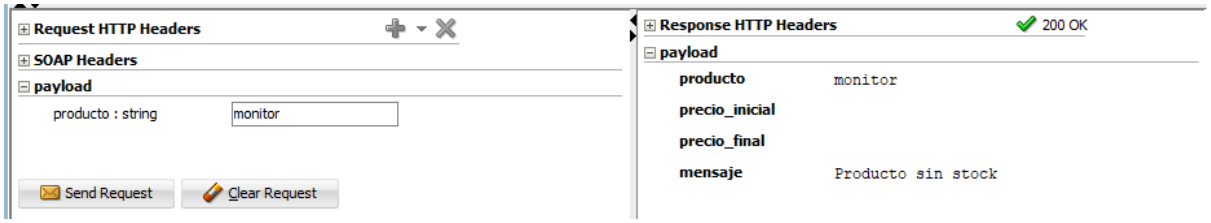

Figura 26: Prueba de servicio ShoppingApp 3.

3. Repositorio GitHub

Acceso al repositorio en GitHub de la asignatura: https://github.com/Valenz23/DSS