Ejercicios sesión 4: Procesos BPEL síncronos y asíncronos

Índice

1. Proceso BPEL síncrono: Servicio de orden de compra.

Como ya hemos indicado, las operaciones síncronas son adecuadas en procesos de negocio que, ante una petición, requieren una respuesta inmediata. La ejecución, en la parte del consumidor del servicio puede continuar solamente después de que la respuesta sea recibida y procesada.

Con este ejercicio utilizaremos un proceso BPEL que: (a) proporciona una operación síncrona a un cliente externo; (b) consume una operación síncrona suminstradas por un servicio Web *partner*.

Cuando se diseña un proceso de negocio que incorpora interacciones con servicios Web síncronos, se necesitan las siguientes construcciones:

- `partnerLinks` que representan servicios Web *partner*
- `variables` que almacenan los datos intercambiados entre los servicios Web
- Una actividad `invoke` para **consumir** un servicio
- Un par de actividades `receive-reply` para **proporcionar** un servicio
- Mapeados de datos de entrada y salida para conseguir la lógica de negocio requerida

1.1. Lógica de negocio de los servicios

Los dos servicios Web que vamos a implementar son: (a) un servicio de orden de compra, *POService*, que es consumido por un cliente externo mediante SOAP sobre HTTP; (b) un servicio de comprobación de inventariado, *InventoryService*, consumido por el proceso BPEL que proporciona el servicio de orden de compra. Ambos servicios, *POService* e *InventoryService* se implementan como servicios BPEL.

Cuando el suministrador del servicio de orden de compra recibe una petición de un cliente, tienen lugar los siguientes eventos:

- El suministrador del servicio de orden de compra:
  - Asigna el precio de la petición
  - Llama al servicio de inventario para comprobar el estado del inventario
- La lógica de negocio del suministrador del servicio de inventario comprueba la disponibilidad de un *item*. Si el valor de *orderDescription* comienza con *OrderVal*, entonces el estado de la orden en el inventario será la de "disponible"
- Según el resultado del servicio de inventario, el suministrador del servicio de orden de compra responde con lo siguiente:
  - La orden de compra cumplimentada

- Un error (en forma de mensaje) indicando que la orden no puede ser completada

La Figura 1 ilustra el diseño de la solución que vamos a plantear:

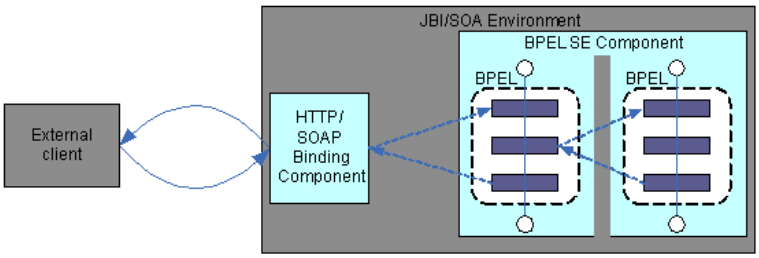

Figura 1. Interacción con un servicio Web Síncrono.

1.2. Creación del proyecto BPEL

Para este ejercicio proporcionamos un proyecto BPEL ya creado con los WSDL y los esquemas (fichero ordenCompra.zip). Creamos una carpeta en nuestro directorio de trabajo, por ejemplo la carpeta *OrdenCompra* y extraemos el contenido del fichero *OrdenCompra.zip* en el directorio *OrdenCompra*.

A continuación, y desde el menú principal de *Netbeans* elegimos *File > Open Project*. Seleccionamos el proyecto *OrdenCompraSincrona* en el directorio *OrdenCompra*.

1.3. Creación del esquema XML

En el proyecto *OrdenCompraSincrona*, encontraremos los ficheros de esquema *.xsd* ya creados (en el directorio *Process Files*). Concretamente se trata de *purchaseOrder.xsd*, e *inventory.xsd*, que contienen la definición de los tipos de datos que utilizan los servicios BPEL que vamos a implementar.

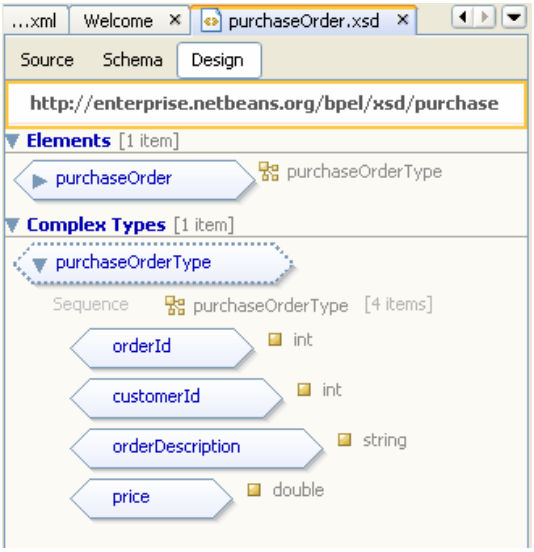

Figura 2. Esquema purchaseOrder.xsd

En el fichero ***purchaseOrder.xsd*** (ver Figura 2) hemos definido el tipo complejo ***purchaseOrderType***, que representa una orden de compra. El tipo *purchaseOrderType* está formado por los elementos locales: *orderId*, *customerId*, *orderDescription*, y *price*, que representan el identificador de la orden de compra, identificador del cliente, la descripción de la orden de compra y el precio de la compra, respectivamente.

También definimos el elemento global ***purchaseOrder***, del tipo *purchaseOrderType*.

Figura 3. Esquema inventory.xsd

El fichero de esquema ***inventory.xsd*** (ver Figura 3) contiene la definición del tipo complejo

inventoryType, que almacena información sobre el estado del inventario. El tipo *inventoryType* está formado por los elementos locales: *inventoryStatus* e *inventoryStatusMessage*.

También definimos el elemento global ***inventory***, del tipo *inventoryType*.

1.4. Creación del documento WSDL

En el directorio *Process Files* del proyecto *OrdenCompraSincrona* contiene también los ficheros *POService.wsdl* e *InventoryService.wsdl*.

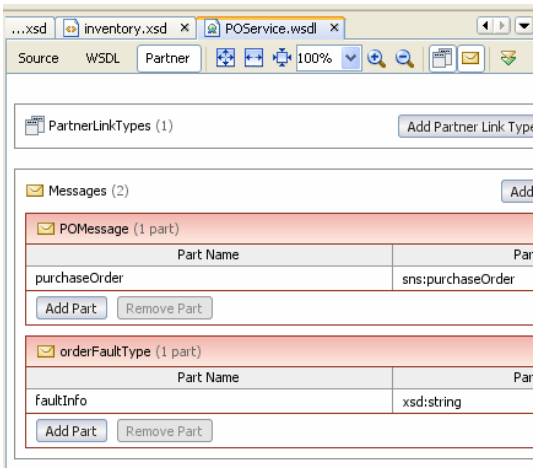

Figura 4. Fichero POService.wsdl

El fichero ***POService.wsdl*** (ver Figura 4) contiene la definición de dos mensajes: *POMessage*, que representa la orden de compra, y *orderFaultType*, que hace referencia a la información sobre el error producido (en caso de que en el inventario no queden existencias para servir la orden de compra).

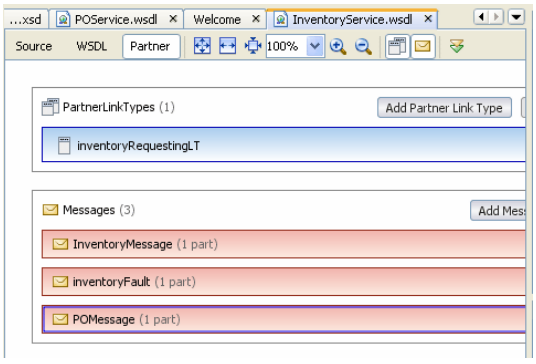

Figura 5. Fichero inventoryService.wsdl

El fichero ***inventoryService.wsdl*** (ver Figura 5) contiene la definición de los mensajes: (a) *InventoryMessage*, formado por un elemento de tipo *inventoryType*, (b) *inventoryFault*, que contiene información sobre el error producido, y (c) *POMessage*, que contiene la orden de compra.

PARTNER LINK TYPES

Como ya sabemos, un *partnerLinkType* especifica la relación entre dos servicios, definiendo el rol que cada servicio implementa. Cada rol especifica exactamente un *portType* WSDL que debe ser implementado por el servicio que implemente dicho rol.

En las Figuras 4 y 5, podemos ver que hemos definido un *partnerLinkType* en cada fichero *wsdl*. Concretamente, en *POService.wsdl* se define un *partnerLinkType* (denominado *purchasingLT*) con el rol *purchaseService*. Dicho rol hace referencia a la operación WSDL *sendPurchaseOrder* a través del *portType* denominado *purchaseOrderPT*.

```
<!-- POService.wsdl-->
<plink:partnerLinkType name="purchasingLT">
   <plink:role name="purchaseService"
               portType="tns:purchaseOrderPT">
   </plink:role>
</plink:partnerLinkType>
```

La operación *sendPurchaseOrder* define una entrada que debe enviarse al suministrador del servicio Web correspondiente, y espera una respuesta o bien un error.

```
<!-- POService.wsdl-->
<portType name="purchaseOrderPT">
   <operation name="sendPurchaseOrder">
       <input name="sendPurchaseOrderRequest"
              message="tns:POMessage"></input>
       <output name="sendPurchaseOrderReply"
              message="tns:POMessage"></output>
       <fault name="cannotCompleteOrder"
              message="tns:orderFaultType"></fault>
   </operation>
</portType>
```

En el fichero *InventoryService.wsdl* se define un *partnerLinkType* (denominado *inventoryRequestingLT*) con el rol *inventoryService*. Este rol hace referencia a la operación *inventoryService* a través del *portType* denominado *inventoryPortType*.

```
<!-- InventoryService.wsdl-->
<plink:partnerLinkType name="inventoryRequestingLT">
   <plink:role name="inventoryService"
               portType="tns:inventoryPortType">
   </plink:role>
</plink:partnerLinkType>
```

La operación *inventoryService* espera un mensaje *purchaseOrder*, y responde con un *inventoryStatus* o *inventoryFaultType*.

```
<!-- InventoryService.wsdl-->
<portType name="inventoryPortType">
    <operation name="inventoryService">
       <input name="purchaseOrder"
              message="tns:POMessage"></input>
       <output name="inventoryStatus"
               message="tns:InventoryMessage"></output>
       <fault name="inventoryFaultType"
              message="tns:inventoryFault"></fault>
    </operation>
</portType>
```

NOTA

Si el fichero WSDL de un servicio Web existente no contiene una definición para un *partnerLinkType*, podemos crear un fichero WSDL *wrapper* para importar el fichero WSDL original, y añadirle la definición de los *partnerlinktypes*. Posteriormente, podemos hacer referencia a dicho *wrapper* desde nuestro proceso BPEL.

1.5. Definimos el proceso BPEL

En nuestro caso tendremos que crear dos documentos *bpel*, correspondientes a los dos servicios BPEL de la solución propuesta, con los nombres *POService.bpel*, y *InventoryService.bpel*.

Creamos *POService.bpel*

Nos situamos en el nodo *Process Files* del proyecto (*OrdenCompraSincrona*) y con el botón derecho seleccionamos *New > BPEL Process*. En el campo *File Name* de la ventana que aparece, ponemos el nombre del fichero: *POService* y pulsamos el botón *Finish*. A continuación se abrirá el fichero *POService.bpel* creado, con la vista de Diseño abierta.

Añadimos los *Partner Links del proceso*

Recordemos que los elementos *partnerLink* especifican los *partners* con los que interactúa el proceso BPEL. Cada *partnerLink* se corresponde con un *partnerLinkType* específico definido en el correspondiente fichero WSDL.

Un elemento *partnerLink* puede contener uno o dos roles:

- `myRole`: especifica el rol del proceso BPEL. Si definimos solamente *myRole* en un *partnerLink*, hacemos posible que cualquier *partner* o cliente interactúa con el proceso BPEL sin ningún otro requerimiento adicional sobre los *partners*.
- `partnerRole`: especifica el rol del *partner*. Si solamente definimos *partnerRole* en el *partnerLink*, permitimos interacciones con un *partner* sin imponer restricciones sobre el servicio que realice la llamada.

El proceso bpel *POService* define un *partnerLink* que denominaremos *Cliente* en el que indicaremos su rol como suministrador de la orden *sendPurchaseOrder*. Para ello realizaremos las siguientes acciones:

- Seleccionamos el fichero *POService.wsdl* y lo arrastramos a la vista de diseño de *POService.bpel*.
- En la ventana de propiedades añadimos como nombre del *partnerLink*: *Cliente*; indicamos el fichero wsdl: *POService.wsdl* de la lista desplegable; usaremos un *partnerLinkType* existente, concretamemente: *purchasingLT*, con el valor de *myRole*: *purchaseService*

Como resultado, si pasamos a la vista de código, podemos comprobar que hemos generado el siguiente código BPEL:

```
<!-- POService.bpel-->
<partnerLink name="POServicePLink"
        partnerLinkType="ns1:purchasingLT"
        myRole="purchaseService"/>
```

Puesto que *POService* consume el servicio *InventoryService*, definimos otro *partnerLink* al que llamaremos *requestInventoryPLink*. En este caso el fichero wsdl asociado es: *InventoryService.wsdl*. El nombre del *partnerLinkType ya existente es: inventoryRequestingLT. Y finalmenente definimos un partnerRole con el nombre inventoryService (para ello tendremos que pulsar el botón* Swap Roles).

```
<!-- POService.bpel-->
<partnerLink name="requestInventoryPLink"
        partnerLinkType="ns2:inventoryRequestingLT"
        partnerRole="inventoryService"/>
```

Definir variables globales del proceso

Para añadir variables, seleccionamos el proceso bpel, y nos aparecerán cuatro iconos en la parte superior izquierda de dicho proceso. Pulsamos el primero de ellos (el situado más hacia la izquierda), y nos aparecerá un editor de variables; a cada una de ellas vamos a asignarle un tipo (que seleccionaremos de los tipos que nos muestra el editor). Concretamente, las variables a crear y sus tipos son:

- Nombre: *purchaseOrderRequest*; tipo: *POMessage* (desde POService.wsdl)
- Nombre: *purchaseOrderReply*; tipo: *POMessage* (desde POService.wsdl)
- Nombre: *purchaseOrderFault*; tipo: *orderFaultType* (desde POService.wsdl)
- Nombre: *inventoryServiceRequest*; tipo: *POMessage* (desde InventoryService.wsdl)
- Nombre: *inventoryServiceReply*; tipo: *InventoryMessage* (desde InventoryService.wsdl)
- Nombre: *inventoryServiceFault*; tipo: *inventoryFault* (desde InventoryService.wsdl)

El código BPEL generado es el siguiente:

```
<!-- POService.bpel-->
  <variables>
    <variable name="purchaseOrderRequest"
              messageType="ns1:POMessage"></variable>
    <variable name="purchaseOrderReply"
              messageType="ns1:POMessage"></variable>
    <variable name="purchaseOrderFault"
              messageType="ns1:orderFaultType"></variable>
    <variable name="inventoryServiceRequest"
              messageType="ns2:POMessage"></variable>
    <variable name="inventoryServiceReply"
              messageType="ns2:InventoryMessage"></variable>
    <variable name="inventoryServiceFault"
              messageType="ns2:inventoryFault"></variable>
  </variables>
```

Añadimos una actividad *receive*

En la sección *Web Service* de la paleta de elementos, seleccionamos el icono *Receive* y lo arrastramos al área de diseño entre las actividades *Start* y *End* del proceso. El IDE de *Netbeans* muestra mediante marcas visuales dónde podemos "soltar" el elemento que estamos arrastrando.

Una vez que "depositemos" la actividad *Receive* en el lugar deseado, pulsamos dos veces con el botón izquierdo sobre la actividad y se abrirá el editor de propiedades, a las que daremos los siguientes valores:

- Name: *receivePOFromClient*.

- Partner Link: *Cliente*.
- Operation: *sendPurchaseOrder*.
- Input variable: *purchaseOrderRequest*.
- La casilla: *Create Instance* estará marcada.

El código BPEL resultante es:

```
<!-- POService.bpel-->
    <receive name="receivePOFromClient"
             partnerLink="Cliente"
             portType="ns1:purchaseOrderPT"
             operation="sendPurchaseOrder"
             variable="purchaseOrderRequest"
             createInstance="yes">
    </receive>
```

Añadimos una actividad *assign*

En el siguiente paso vamos a añadir una asignación a la que llamaremos *Assign 1*. Se trata de asignar un valor a la parte *price* de la orden de compra (*purchaseOrder*). El valor será: 49.98.

Para ello tendremos que arrastrar desde la paleta la actividad *Assign* en la sección de *Basic Activities*, hasta colocarla a continuación de la actividad *receive*.

Cuando "soltemos" la actividad *Assign* que estábamos arrastrando se abrirá la ventana *BPEL Mapper* en la parte inferior del IDE (si no fuese visible, tendríamos que seleccionar *Windows > BPEL Mapper* desde el menú del IDE de *Netbeans*).

En la ventana *BPEL Mapper* tenemos a ambos lados una lista de las variables que podemos utilizar (que serán las que hemos declarado como globales). Las asignaciones se harán haciendo corresponder un elemento de la izquierda (que representa el valor asignado), con un elemento de la parte derecha (que representa el valor a asignar).

En nuestro caso particular, queremos asignar a la variable que representa el precio de la orden de compra el valor 49.98. Para ello seleccionamos *Number > Number Literal* desde la barra de menús de la ventana *Mapper*, y tecleamos el valor 49.98 en el cuadro de diálogo asociado a *Number Literal*. A continuación, en la lista de variables de la parte derecha, desplegamos la variable *purchaseOrderRequest > purchaseOrder >* y dejamos visible la parte *price*. Finalmente, con el botón izquierdo pulsado lo llevamos desde un pequeño cuadrado azul que aparece a la derecha del cuadro de diálogo *Number Literal*, hasta la parte *price*, para así completar el proceso de asignar un valor a una variable.

El código BPEL generado es el siguiente:

```
<!-- POService.bpel-->
<copy>
   <from>49.98</from>
   <to>$purchaseOrderRequest.purchaseOrder/ns0:price</to>
</copy>
```

A continuación asignamos todos los valores del mensaje *purchaseOrderRequest.purchaseOrder* a los elementos correspondientes del mensaje *inventoryServiceRequest.purchaseOrder*. Para ello vammos "trazando líneas" con el botón izquierdo del ratón desde la variable de la parte inquierda del *Mapper* hasta la correspondiente variable de la parte derecha del *Mapper*. El resultado se muestra en la Figura 6.

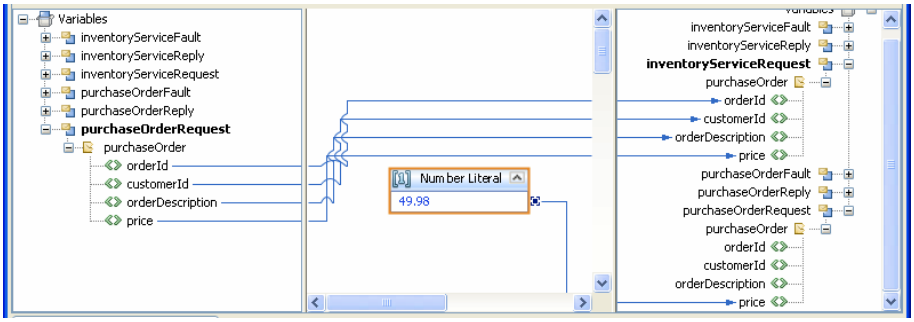

Figura 6. BPEL Mapper: asignaciones correspondientes a la actividad assign1

Añadimos la actividad *invoke*

La actividad *invoke* tiene el efecto de consumir un servicio Web. Añadiremos dicha actividad a continuación de la actividad *Assign1*. El proceso es similar al realizado para la actividad *receive*. En este caso, los valores de las propiedades para la actividad *invoke* serán los siguientes:

- Name: *CallInventoryService*
- PartnerLink: *requestInventoryPLink*
- Operation: *inventoryService*
- Input variable: *inventoryServiceRequest*
- Output variable: *inventoryServiceReply*

Añadimos la actividad *if*

La actividad *if* tiene el efecto de bifurcar la lógica del proceso BPEL en función de una condición cuyo resultado será cierto o falso. La añadiremos a continuación de la actividad *invoke*. Los valores de las propiedades de la actividad *if* serán:

- Name: *CheckAvailability*

- Condition: *$inventoryServiceReply.inventoryPart/ns3:inventoryStatus*. En la Figura 7 mostramos el mapeado a realizar en la ventana *BPEL Mapper*, asignando el valor de la parte *inventoryStatus* (de tipo *boolean*) a la variable *Result* asociada a la actividad *if*.

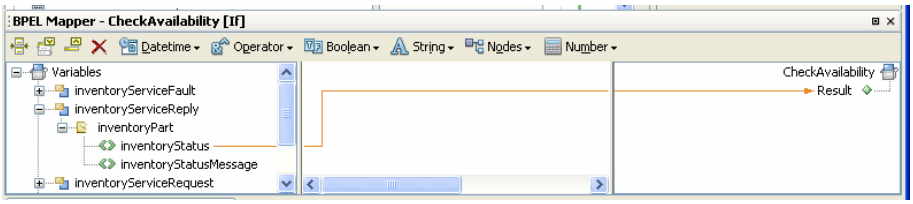

Figura 7. BPEL Mapper: asignación correspondiente a la actividad CheckAvailability

En la parte cierta de la actividad *if* (parte "then", queda justo debajo del símbolo que representa el *if*), añadimos una actividad *sequence*, en la que anidaremos las actividades *assign* y *reply*.

Para la actividad *assign* a la que llamaremos *assign 2*, haremos corresponder la variable *purchaseOrderRequest* con la variable *PurchaseOrderReply*.

Para la actividad *reply*, los valores de sus propiedades serán:

- Name: *ReplyPO*
- Partner Link: *Cliente*
- Operation: *sendPurchaseOrder*
- Normal response, Output variable: *purchaseOrderRequest*

Procederemos de la misma forma para la parte "else" de la actividad *if*, es decir, añadimos una actividad *sequence* a la que anidamos las actividades *assign* y *reply*.

Para la actividad *assign* a la que llamaremos *assign 3*, haremos corresponder la variable *inventoryServiceReply > inventoryPart > inventoryStatusMessage* con la variable *purchaseOrderFault*.

Para la actividad *reply*, los valores de sus propiedades serán:

- Name: *ReplyFault*
- Partner Link: *Cliente*
- Operation: *sendPurchaseOrder*
- Fault response, Fault Name: *ns1:cannotCompleteOrder*
- Fault response, Fault Variable: *purchaseOrderFault*

La Figura 8 muestra el resultado final de la vista de diseño del proceso POService.bpel.

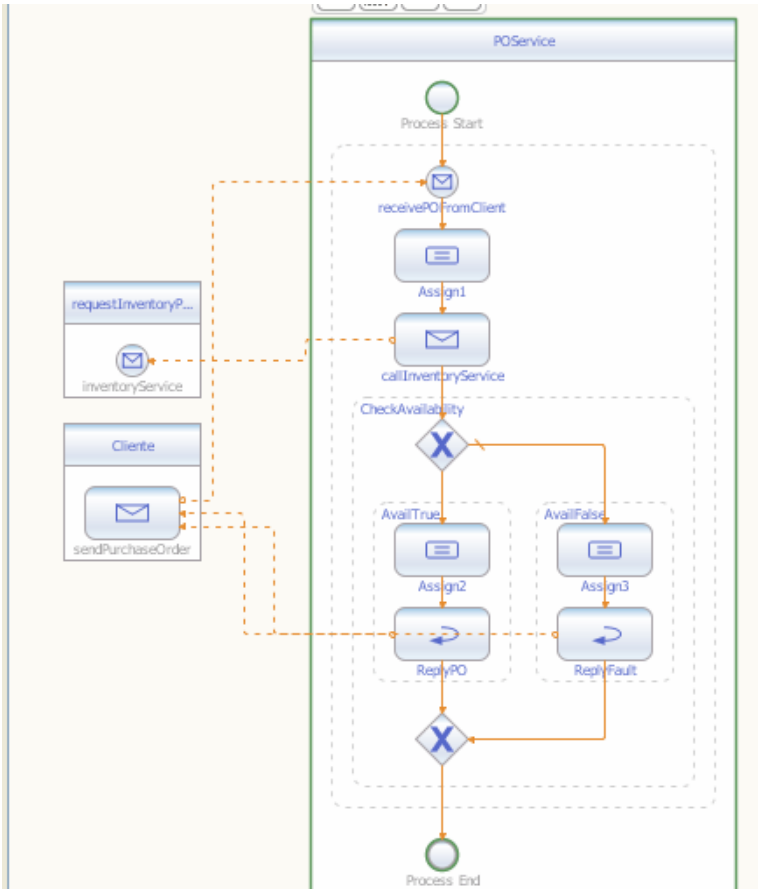

Figura 8. Vista de diseño del proceso POService.bpel

Guardamos todos los cambios con *File > Save All* (si es que no hemos ido guardando antes los cambios con *File > Save* .

De forma similar, crearemos el proceso *InventoryService.bpel*. En el fichero *inventory.zip* que se proporciona, encontraréis el fuente *inventoryService.bpel* que podéis incorporar en vuestro proyecto. Para ello no tendréis más que, desde el sistema operativo, copiar dicho fichero en el directorio *src* del proyecto.

1.6. Compilamos el proyecto

Para compilar el proyecto BPEL, simplemente nos situamos en el nodo del proyecto *OrdenCompraSincrona* y con el botón derecho del ratón seleccionamos la opción *Build Project*

1.7. Creamos la Composite Application

Crearemos una nueva *Composite Application* para que podamos desplegar nuestro proyecto BPEL en el servidor de aplicaciones. Para ello seguiremos los siguientes pasos:

- Seleccionamos *File > New Project > Service Oriented Architecture > Composite Application*
- Asignamos el nombre *OrdenCompraSincronaApplication*
- Una vez hecho esto, nos aparecerá un nuevo proyecto con el nombre *OrdenCompraSincronaApplication*. Nos situamos en dicho nodo, y con el botón derecho del ratón seleccionamos Add JBI Module
- Seleccionamos el proyecto *OrdenCompraSincrona* y pulsamos el botón: *Add Project JAR Files*. Si desplegamos el nodo *JBI Modules* veremos que se ha añadido el nodo *OrdenCompraSincrona.jar*

A continuación debemos asegurarnos de que el servidor está en marcha para poder desplegar la *Composite Application* que hemos creado. Para ello puedes consultar el ejercicio de la sesión anterior.

Para desplegar el proceso BPEL en el servidor de aplicaciones nos situaremos en el nodo *OrdenCompraSincronaApplication* y elegiremos *Deploy Project*.

1.8. Probamos el proceso BPEL

Para probar nuestro proceso BPEL vamos a añadir un caso de prueba. Nos situamos en el nodo *Test* del proyecto *OrdenCompraSincronaApplication* y elegimos *New Test case* al que llamaremos *TestExito*.

A continuación elegimos el fichero WSDL del proceso a probar: *POService.wsld*. La operación a probar será *sendPurchaseOrder*. Vemos que se ha creado el fichero *Input.xml*, que contiene la estructura del mensaje de entrada. Como datos de entrada podemos utilizar los datos del fichero *Input.xml* que se proporciona. Para ello podemos pegar el contenido del fichero Input.xml proporcionado en el nuevo fichero *Input.xml* reemplazado así su contenido. Guardamos los cambios.

Para ejecutar el proceso con dicha entrada, nos situamos en el nodo *TestExito* y elegimos *Run*.

Recordad que la primera vez obtendremos un informe por pantalla indicándonos que no se ha "pasado la prueba" debido a que el fichero *Output.xml* estará vacío inicialmente.

Al ejecutar la prueba por segunda vez (y sucesivas) ya debemos obtener un resultado

correcto.

2. Proceso BPEL asíncrono. Uso de correlación: Hola Mundo.

Cada proceso de negocio BPEL es un servicio Web, por lo que puede parecer que solamente necesitamos conocer el puerto de destino para poder enviar un mensaje a un proceso BPEL. Sin embargo, puesto que los procesos de negocio son procesos con estado, se instancian basándose en su estado. En este ejercicio, utilizaremos los conjuntos de correlación de BPEL para dar soporte a una colaboración entre servicios Web con estado.

Para este ejercicio se proporciona el fichero *hello.zip* que contiene un proceso de negocio sencillo denominado *HelloWorldSample*, con su correspondiente *Composite Application* con el nombre *HelloWorldSampleCompApp*.

Para comenzar a trabajar con el proyecto con *Netbeans*, nos vamos al menú principal del IDE y elegimos *File > Open Project*. Seleccionamos los proyecots *HelloWorldSample* y *HelloWorldSampleCompApp*, y "pinchamos" con el botón izquierdo del ratón sobre *Open Project Folder*. Observaremos que aparece un problema de referencias (esto es normal, puesto que la aplicación compuesta tiene una dependencia del módulo JBI). Para resolver el problema nos situamos sobre el proyecto *HelloWorldSampleCompApp*, pulsamos el botón derecho y elegimos la opción *Resolve Reference Problems* del menú emergente. En el cuadro de diálogo que aparece pulsamos en el botón *Resolve*. En el cuadro de diálogo *Browse Project* seleccionamos el proyecto *HelloWorldSample* y pulsamos sobre *Open Project Folder*. Ahora el cuadro de diálogo *Resolve Reference Problems* nos indica que el problema está resuelto..

A continuación desplegamos el proyecto *HelloWorldSampleCompApp* en el servidor de aplicaciones (recuerda cómo lo hemos hecho en los ejercicios anteriores). Podemos probar el proyecto ejecutando el test *SendUserName1*.

2.1. Añadimos un estado al proceso de negocio

Para convertir al proceso *HelloWorld* en un proceso de negocio con estado, lo que haremos será añadir otra actividad *Receive* a dicho proceso. Esta actividad añade una comunicación asíncrona con el proceso y éste se convierte en un proceso con estado. El mensaje resultante devuelto por el proceso será extendido con un prefijo.

Para añadir el estado al proceso *HelloWorld* realizaremos los siguientes pasos:

- Expandimos el nodo *HelloWorldSample > Process Files* y abrimos el fichero *HelloWorlProcess*.

- Arrastramos el icono *Receive* desde la paleta hasta situarla entre las actividades *ReceiveUserName* y *AddHello*. Esta acción añade una actividad *Receive1* en el diagrama.
- Vamos a cambiar las propiedades de *Receive1* de la siguiente forma:
  - Name: *SetPrefix*
  - Partner Link: *ProcessInterface*
  - Operation: *setPrefix*
  - Creamos una variable de entrada utilizando en botón *Create* con el nombre *Prefix* y cerramos el editor de propiedades
- Añadimos un prefijo al nombre de usuario en la cadena de salida de la siguiente forma:
  - Abrimos el *BPEL mapper* pinchando dos veces sobre la actividad *AddHello* y borramos el enlace existente entre la variable *UserNameRequest>part1>data* y la parte *string2* de la función *Concat* (nos situamos sobre él y pulsamos la tecla de borrado).
  - Seleccionamos la variable *Prefix>part1>data* en la parte izquierda del *mapper* y la enlazamos con la parte *string2* de la función *Concat*.
  - Hacemos lo mismo para *UsernameRequest>part1* y la parte *string3* de la función *Concat*.
  - Pulsamos *Ctrl-S* para guardar los cambios.

Con los pasos anteriores hemos añadido una comunicación asíncrona en el proceso. Ahora, después de que se reciba el primer mensaje y se inicialice el proceso, el proceso BPEL necesita esperar a otro mensaje en la actividad *SetPrefix*.

Imaginemos una situación en la que varias instancias del proceso son instanciadas y todas ellas están esperando un mensaje en la segunda actividad *Receive*.

Como ya hemos visto, la máquina BPEL utiliza conjuntos de correlación para decidir a qué instancia proceso se le ha enviado el mensaje. Como resultado de los cambios que hemos realizado, el proceso devuelve la cadena de saludo extendida con el prefijo *Mr.*.

2.2. Definimos las propiedades de correlación y los alias de las propiedades

Las propiedades se utilizan típicamente para almacenar elementos para la correlación de instancias de servicios con mensajes. Usaremos *property aliases* para especificar qué parte de los datos tiene que extraerse de los mensajes y con qué propiedad tienen que asociarse los datos extraídos. Una propiedad es un concepto abstracto, mientras que la *propertyAlias* es el aspecto concreto correspondiente. Las *property aliases* enlazan las propiedades con valores definidos en el mensaje del servicio Web utilizando una *query xpath*.

Para **crear una propiedad**, seguiremos los siguientes pasos:

- Desde el menú principal, elegimos *Window > Navigator*
- En la vista de diseño, seleccionamos el proceso *HelloWorldProcess*. La ventanta *Navigator* muestra la vista lógica de BPEL, es decir, una vista estructurada del proceso de negocio.
- En la ventana *Navigator* expandimos *Imports*. (Vamos a utilizar *HelloWorldProcessWSDLWrapper.wsdl* para añadir las propiedades y alias de las propiedades).
- Pinchamos con el botón derecho sobre *HelloWorldProcessWSDLWrapper.wsdl* y elegimos *Add Property* del menú emergente.
- Como nombre de la propiedad pondremos *MyProperty*, y elegimos *string* como tipo de la propiedad en el árbol *Built-in Types*. Finalmente pulsamos OK.

Ahora necesitamos crear una *property alias* para especificar cómo se extraen los datos de correlación a partir de los mensajes. Como tenemos dos actividades *Receive* que reciben mensajes de tipos diferentes, necesitamos definir dos *property aliases*.

Creamos la primera ***property alias***, con los siguientes pasos:

- En la ventana *Navigator* expadimos *Imports*.
- Pinchamos con el botón derecho sobre *HelloWorldProcessWSDLWrapper.wsdl* y elegimos *Add Property Alias* del menú emergente. Recuerda que necesitamos especificar la propiedad, parte del mensaje, y la *query* para crear una *property alias*. La propiedad especificada se usa para almacenar un *token* de correlación extraído, la parte del mensaje ayuda a establecer una correspondencia entre la *property alias* con algún mensaje específico y su parte, y la *query* se utiliza para especificar qué datos en particular necesitan ser extraídos.
- En el cuadro de diálogo, pulsamos sobre *Browse*.
- Expandimos el nodo *HelloWorldProcessWSDLWrapper.wsdl*, seleccionamos *MyProperty* y pulsamos OK.
- En el cuadro de diálogo, expandimos el nodo *HelloWorldProcessWSDL.wsdl* y *getHelloRequest*, y pulsamos OK.
- Elegimos *part1* como una parte del mensaje.
- Especificamos */UserData/id* en el campo de texto *Query* y pulsamos OK.

Para crear la segunda *property alias*, repetimos los pasos anteriores, teniendo en cuenta que elegiremos *HelloHelloWorldProcessWSDL.wsdl > setPrefixRequest > part1* como parte del mensaje. Para la *Query* especificaremos */Prefix/id*.

2.3. Creamos y añadimos los Correlation sets

Como ya hemos visto, un conjunto de correlación es una colección de propiedades que especifican qué datos deberían extraerse de los mensajes de entrada. Estos datos se utilizan

posteriormente para identificar la instancia del proceso destinataria del mensaje.

Para **crear un conjunto de correlación** seguimos los siguientes pasos:

- Seleccionamos el proceso *HelloWorldProcess* en la vista de diseño, pulsamos el botón derecho del ratón y elegimos *Add > Corrrelation Set* en el menú emergente.
- En el cuadro de diálogo correspondiente elegimos como nombre: *MyCorrelationSet*, y pulsamos el botón *Add*.
- Aparecerá el cuatro de diálogo *Property Chooser*, expandimos el nodo *HelloWorldProcessWSDLWrapper.wsdl*, y seleccionamos *MyProperty*. Pulsamos OK para añadir la propiedad, y OK de nuevo en el cuadro de diálogo *Add Correlation Set* para crear el conjunto de correlación.

Después de crear el conjunto de correlación, necesitamos añadirlo a las actividades que reciben/envían mensajes.

Para **añadir el conjunto de correlación a la actividad *ReceiveUserName*** tenemos que:

- Seleccionamos la actividad *ReceiveUserName* en la vista de diseño, pulsamos el botón derecho del ratón y elegimos *Edit* en el menú emergente.
- En el cuadro de diálogo correspondiente elegimos como nombre: *MyCorrelationSet*, y pulsamos el botón *Add*.
- En la pestaña *Correlations*, pulsamos en el botón *Add*, y elegimos *MyCorrelationSet* en la lista desplegable.
- A continuación fijamos el valor de *MyCorrelationSet* en la columna *Initiate* a *yes*. Finalmente pusamos sobre *OK*.

Ahora necesitamos mapear los datos de correlación con la actividad *setPrefix*. Esto permitirá a la máquina BPEL asociar datos de entrada con instancias del proceso.

Para **añadir el conjunto de correlación a la actividad *SetPrefix*** tenemos que:

- Seleccionamos la actividad *SetPrefix* en la vista de diseño
- Pulsamos con el botón derecho sobre la actividad y elegimos *Edit* en el menú emergente.
- En la pestaña *Correlations* pulsamos sobre *Add*, y elegimos *MyCorrelationSet*. Finalmente pulsamos OK. Tenemos que asegurarnos de que el valor de *MyCorrelationSet* en la columna *Inititate* tiene el valor *no*.

Finalmente grabamos todos los cambios con *File > Save All*.

2.4. Pruebas y debugging del proceso BPEL

Vamos a comprobar que el proceso que hemos diseñado funciona bien. Monitorizaremos la ejecución del proceso para verificar que se devuelve la salida esperada. Para ello vamos a

seguir los siguientes pasos:

- En la ventana de proyectos pulsamos con el botón derecho del ratón sobre el proyecto *HelloWorldSampleCompApp* y elegimos *Clean and Build Project* del menú emergente.
- Esperamos hasta que el mensaje "BUILD SUCCESSFUL" aparezca en la ventana de salida.
- Pulsamos con el botón derecho del ratón sobre el proyecto *HelloWorldSampleCompApp* y elegimos *Debug Project (BPEL)* del menú emergente para comenzar a monitorizar la ejecución del proyecto.
- Esperamos hasta que el mensaje que indica que la sesión de *debug* ha comenzado en la ventana de la consola del *debugger* de BPEL.
- En la vista de diseño, seleccionamos la actividad *ReceiveUserName*. Pulsamos sobre ella con el botón derecho del ratón y elegimos *Toggle Breakpoint*. Aparecerá un cuadrado rojo sobre la actividad con un *breakpoint*.
- Pulsamos con el botón derecho del ratón sobre la actividad *AddHello* y añadimos otro *breakpoint*.
- En la ventana de proyectos, expandimos *HelloWorldSampleCompApp > Test*.
- Pulsamos con el botón derecho sobre el test *SendUserName2* y elegimos *Run* en el menú emergente. Las actividades por las que va pasando el programa aparecen marcadas con un indicador verde.
- Tan pronto como el proceso se detiene en su primer *breakpoint* pulsamos sobre *Continue* en la barra de herramientas del *debugging* (*Run > Continue (CtrL+F5)*).
- El proceso continúa y se detiene en la actividad *SetPrefix*.
- En la ventana de proyectos, pulsamos con el botón derecho del ratón sobre el test *SendPrefix* y seleccionamos *Run*.
- El proceso continúa y se detiene en el segundo *breakpoint* de la actividad *AddHello*.
- Volvemos a pulsar sobre *Continue* en la barra de herramientas del *debugging* para continuar el proceso. (Cuando JUnit nos pregunte si queremos sobreescribir la salida responderemos afirmativamente).
- Al final del proceso, el test *SendPrefix* se ejecuta con éxito.
- En la ventana del proyectos, hacemos doble *click* sobre el nodo con la salida del test *SendUserName2*. La salida del test *SendUserName2* contiene la cadena esperada: *Hello Mr. Michael*. Lo cual significa que el proceso ha funcionado bien y que la máquina BPEL ha manejado correctamente la comunicación asíncrona utiizando el conjunto de correlación *MyCorrelationSet*
- Seleccionamos *Run > Finish Debugger Session* desde el menú principal.

Podemos comprobar que si deshabilitamos el conjunto de correlación *MyCorrelationSet*, el proceso nunca alcanza la actividad *AddHello* ya que no hay datos de correlación que puedan ayudar a la máquina de servicios BPEL para enrutar el mensaje a la instancia correcta del proceso.