

Laboratory #4

Interoperability

Jeisson Andrés Vergara Vargas, M.Sc. (c)

Software Architecture

2017-II

Requisitos previos

- Para el desarrollo de este laboratorio se necesitara lo siguiente:
 - MS Sales, API Gateway(Ruby on Rails).
 - Aplicación Bank (Ruby on Rails) – Se proporcionará.
 - Herramienta: SoapUI - <https://www.soapui.org/>.
 - Herramienta: Open ESB SA 3.0.5 - <http://www.open-esb.net/>.

Creación y Publicación de Servicios Web (Ruby on Rails)

Se tomará como ejemplo la aplicación de Bank y se explicarán los pasos que se realizaron para la publicación del servicio web. Para esto se utilizó la gema “**wash out**”.

```
39 gem 'therubyracer', :platforms => :ruby
40 gem 'execjs'
41 gem 'wash_out'
```

Se agregó un nuevo controlador con el nombre de “**wsbanks_controller.rb**”.

```
Gemfile wsbanks_controller.rb
1 class WsbanksController < ApplicationController
2   soap_service namespace: 'urn:WashOutBank', camelize_wsdl: :lower
3   # make case
4   soap_action "make",
5     :args => { :start => :integer, :end => :integer, :user => :integer, :amount => :double},
6     :return => :boolean
7   def make
8     operation = Transaction.create(amount: params[:amount], startAccount_id: params[:start], finalAccount_id: params[:end])
9     render :soap => true
10  end
11
12  # check case
13  soap_action "check",
14    :args => { :start => :integer, :end => :integer, :amount => :double},
15    :return => :boolean
16  def check
17    amount = params[:amount]
18    validate = true
19    if !(Account.exists?(id: params[:start]))
20      validate = false
21    end
22    if !(Account.exists?(id: params[:end]))
23      validate = false
24    end
25    render :soap => validate
26  end
27 end
```

La imagen anterior muestra el código utilizado para crear un servicio web con dos operaciones: “make” y “check”.

Primero para indicar la primera configuración de que el controlador es un servicio web se debe incluir la línea “`soap_service namespace: 'urn:WashOutBank'`”.

Un servicio web puede realizar muchas acciones, por lo que para identificar cada una de ellas se utilizara la sentencia “`soap_action`” y en seguida de la palabra entre comillas, el nombre de la acción por la cual será reconocido.

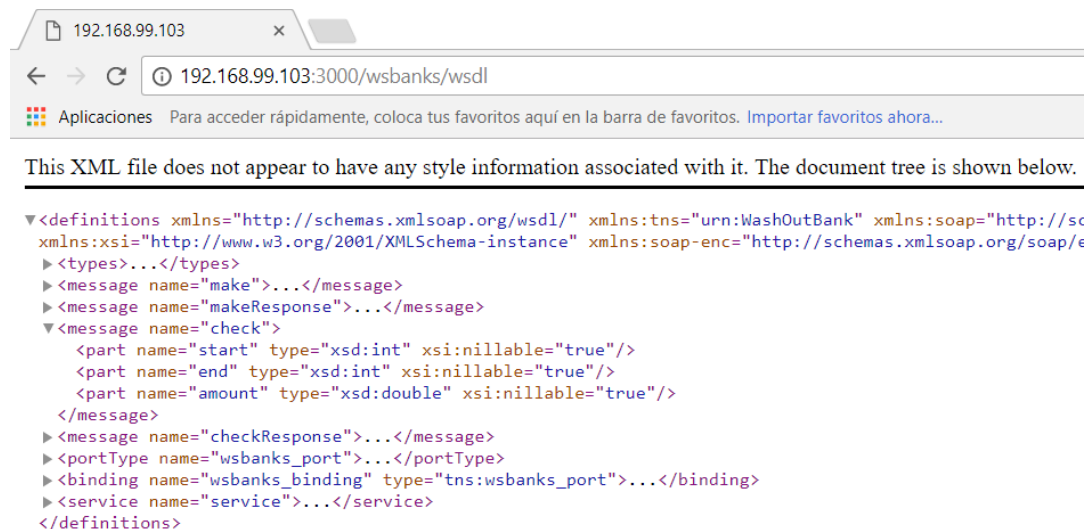
Con esto se implementa el servicio web, pero es necesario indicarle a Ruby la ejecución de este archivo para generar el documento WSDL asociado, para lo cual es necesario modificar el archivo `routes.rb`, agregando “`wash_out :controlador`”.

```

Gemfile  wsbanks_controller.rb  routes.rb
1  Rails.application.routes.draw do
2    resources :transactions
3    resources :accounts
4
5    wash_out :wsbanks
6    # For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html
7  end

```

Ya con todo esto configurado, se realiza el despliegue de la aplicación y se accederá a la siguiente ruta: “http://ip_contendor:puerto/controlador/wSDL”, donde se mostrará el documento WSDL generado:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

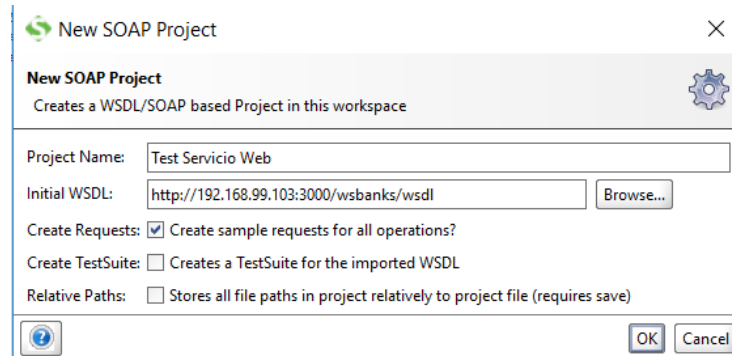
```

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:WashOutBank" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/">
  <types>...</types>
  <message name="make">...</message>
  <message name="makeResponse">...</message>
  <message name="check">
    <part name="start" type="xsd:int" xsi:nil="true"/>
    <part name="end" type="xsd:int" xsi:nil="true"/>
    <part name="amount" type="xsd:double" xsi:nil="true"/>
  </message>
  <message name="checkResponse">...</message>
  <portType name="wsbanks_port">...</portType>
  <binding name="wsbanks_binding" type="tns:wsbanks_port">...</binding>
  <service name="service">...</service>
</definitions>

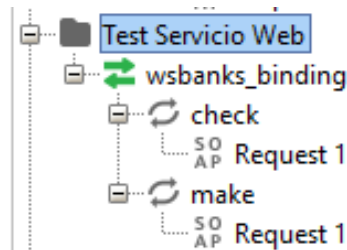
```

Verificación del servicio web

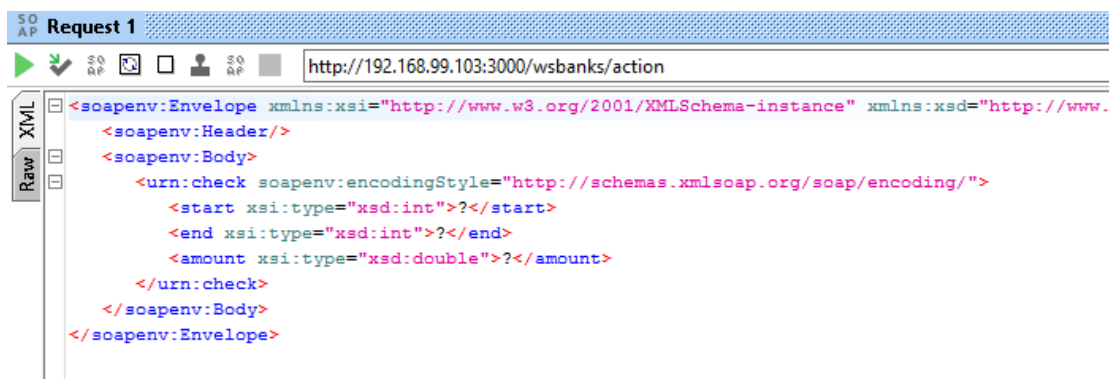
Para realizar la prueba de verificación del servicio web, se utilizará la herramienta SoapUI, en la cual se crea un nuevo proyecto SOAP. Para el caso este ejemplo, se configura de la siguiente manera:



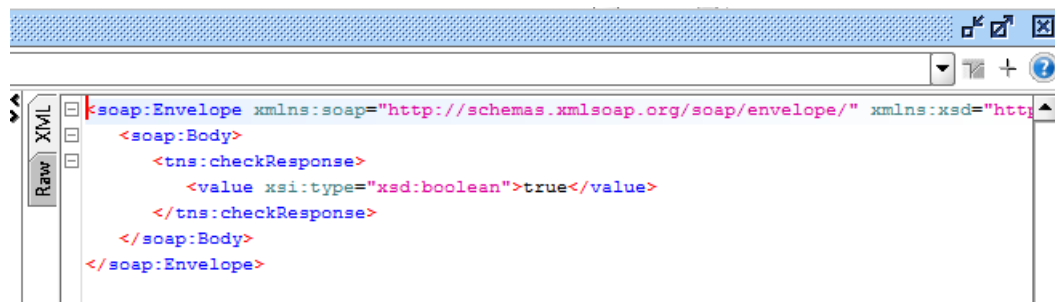
Se ingresa el nombre del proyecto para identificarlo, y se debe ingresar la ruta del documento WSDL generado en los pasos anteriores. Al dar en el botón “OK”, establecerá una conexión hacia ese servicio web para identificar el contenido del mismo y nos mostrará el siguiente resultado.



Como se muestra en la imagen anterior, se observa que se identificaron las dos acciones implementadas en este servicio web.



Al realizar doble clic veremos un código para realizar pruebas con los argumentos que se establecieron en el servicio web, donde el símbolo “?” indica el valor a ser enviado. Para este caso en específico la acción se encarga de verificar si tanto la cuenta origen “**start**” como la cuenta destino “**end**” existen en la base de datos, por lo que dará como respuesta un “**true**” o “**false**”. Una vez se coloquen los datos, se oprime en el botón que es una flecha verde para la ejecución del servicio web, mostrando el resultado obtenido.



Al igual que en el caso de la otra acción se pueden realizar las pruebas, y verificar el cambio en la base de datos.

Creación y Configuración de ESB

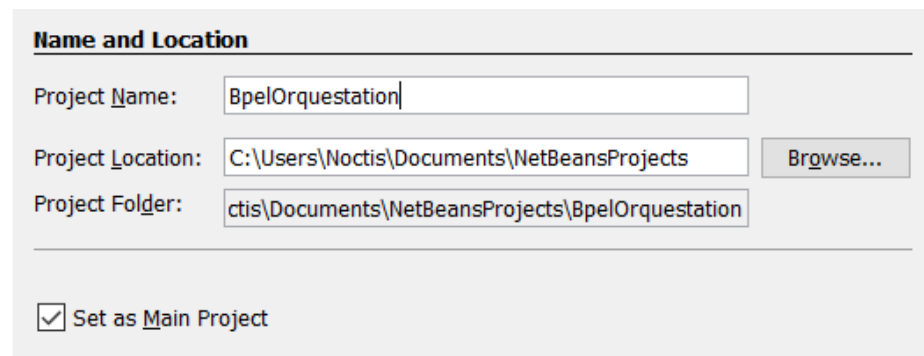
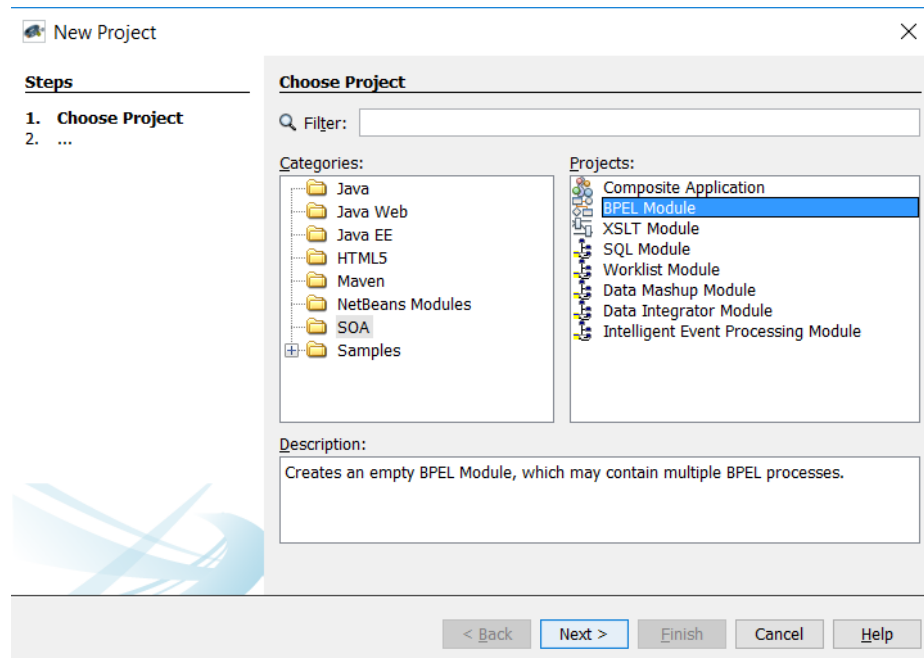
Cuando se tiene la necesidad de intercambio de información entre dos sistemas, como en el caso de ejemplo con la aplicación **Bank** y la aplicación **Store**, en donde el escenario de que al comprar un producto en la tienda, esta transacción de pago se vea reflejada en las transacciones que el banco maneja, el **Bus de Servicios (Enterprise Service Bus – ESB)**, provee una centralización de dichos servicios, en la que se pueda hacer una correcta orquestación de éstos, logrando así, un mecanismo de control que coordina y administra la secuencia de invocación de los mismos.

Para el presente laboratorio, se utilizó la herramienta **OpenESB**, que utiliza el lenguaje **BPEL** en la implementación del **ESB**.

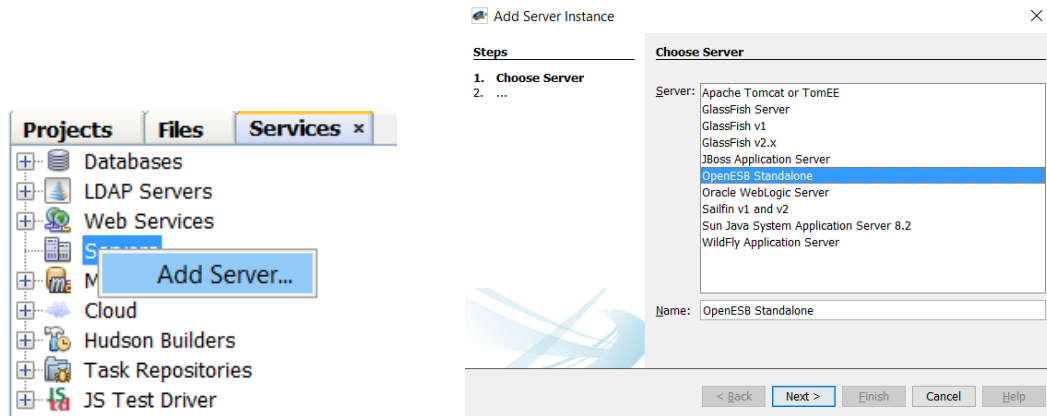
Configuración Inicial del servidor

1. Para abrir el programa OpenESB Standalone 3.0.5, los ejecutables se encuentran en la ruta “/OpenESB-SE-3.0.5/OE-Studio/netbeans/bin”, ya sea para ejecución por Linux o Windows. No olvidar dar permisos a los archivos que se encuentran en la ruta “OE-INSTANCE/bin”.

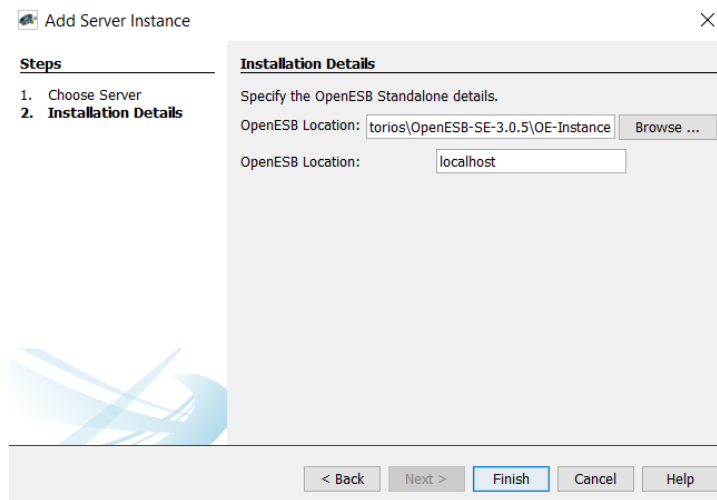
2. En el menú “File” crear un nuevo proyecto, en las categorías dadas escoger “SOA” y en el tipo de proyecto “BPEL Module”, asignando un nombre al proyecto.



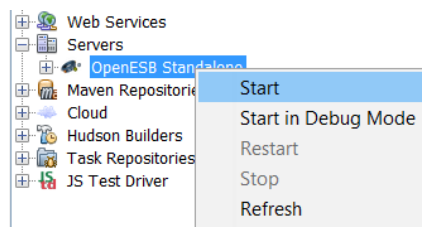
3. Una vez creado el proyecto, dirigirse a la pestaña “Services”, y en la sección de “Servers” agregar un nuevo servidor; seleccionar “OpenESB Standalone” y realizar clic en siguiente.



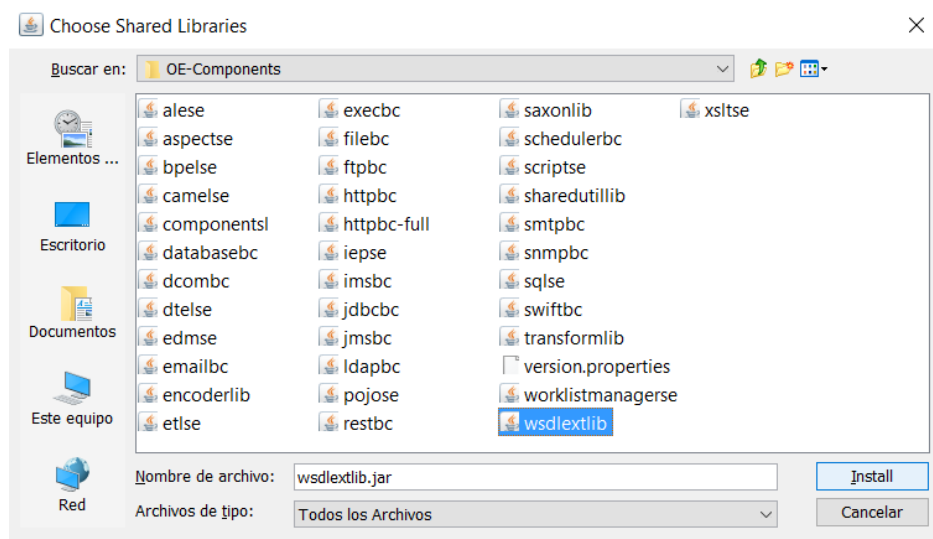
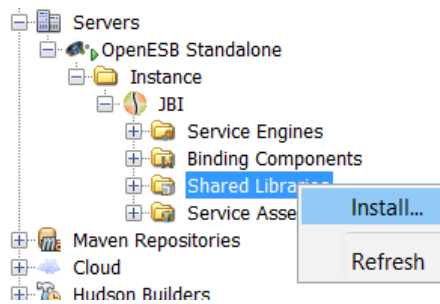
4. En la siguiente pestaña de configuración, en la localización del OpenESB ubicar la ruta “OE-Instance”, este se encuentra en la misma carpeta del directorio general descargado del OpenESB y después clic en el botón *Finish*.



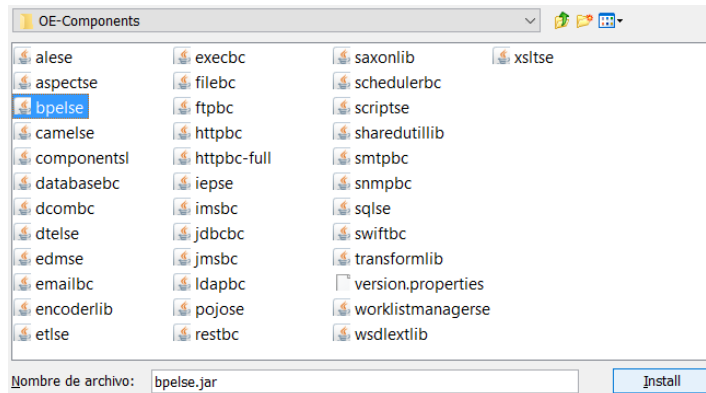
5. En la ventana que aparece escoger la opción “**OpenESB Standalone**”. Y después realizar clic derecho y seleccionar la opción “**Start**”.



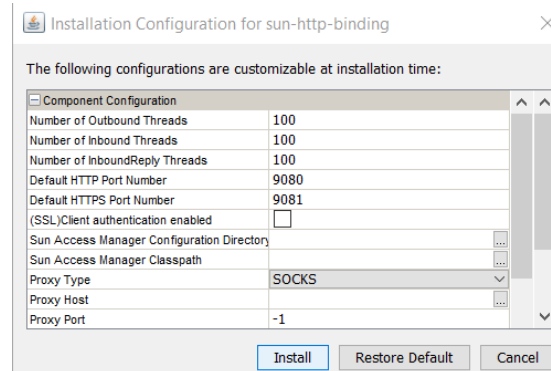
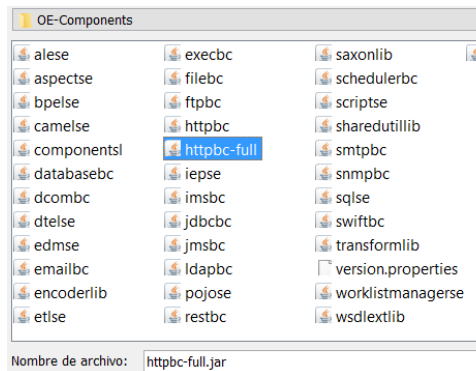
6. Una vez este iniciado el servidor, abrir las pestañas hasta encontrar las 4 carpetas raíz del servidor “OpenESB Standalone/Instance/JBI”.
7. Realizar clic derecho sobre la carpeta “Shared Library”, escoger la opción “Install...”, y buscar en la carpeta “OE-COMPONENT” el archivo que se llama “wsdlexlib.jar” y realizar clic en Install.



8. Ahora en la carpeta del servidor “Service Engines” escoger la opción “Install...”, y buscar en la carpeta “OE-COMPONENT” el archivo que se llama “bpelse.jar” y lo agregarlo al servidor.



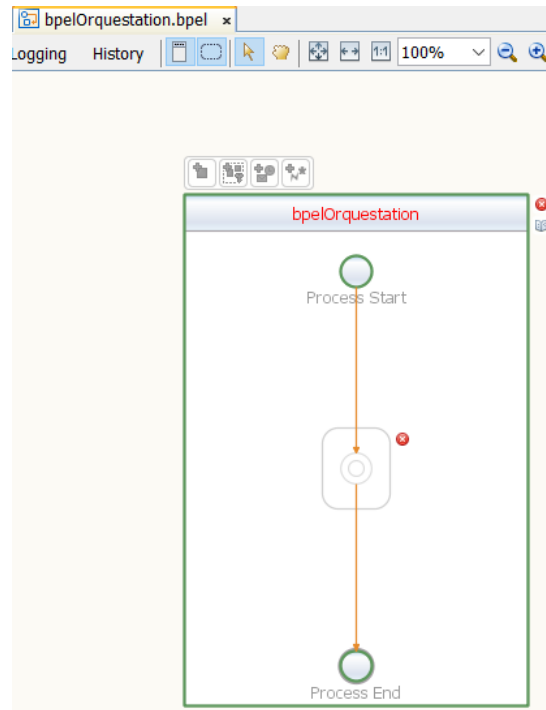
9. En la carpeta del servidor “**Binding Component**” escoger la opción “**Install...**”, y buscar en la carpeta “**OE-COMPONENT**” el archivo que se llama “**httpbc-full.jar**” y dejar las configuraciones establecidas al final.



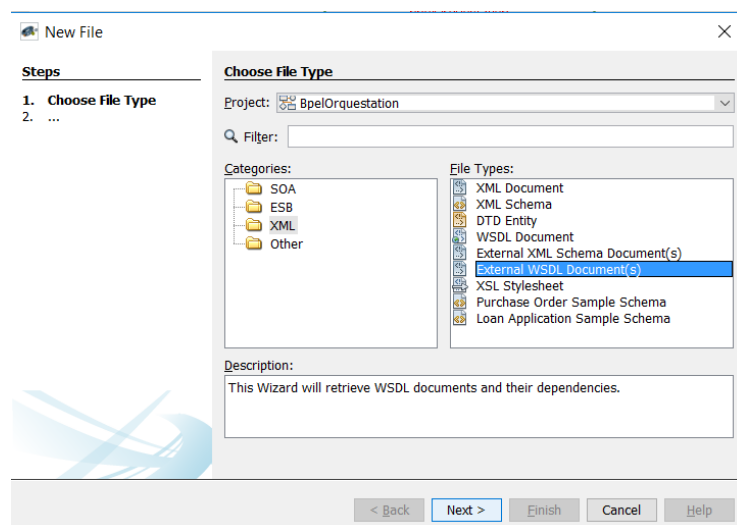
10. Por último es necesario reiniciar el servidor para que reconozca los cambios.

Diseño y Creación del Bus de Servicios

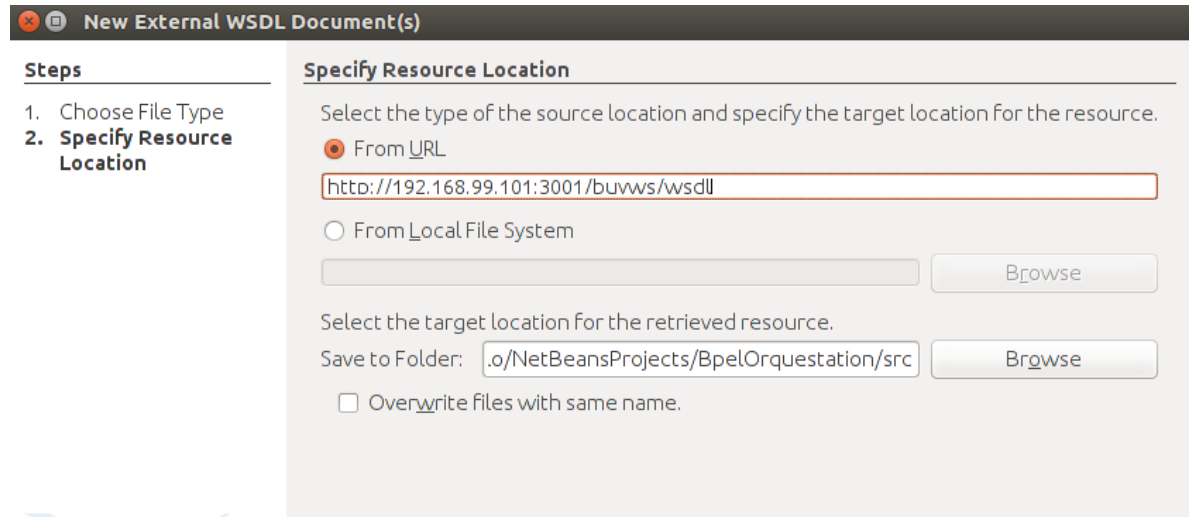
11. Después de haberse creado, en el área de trabajo del IDE se mostrará un diagrama, el cual corresponde conceptualmente a nuestro **Bus de Servicios**. Mediante este entorno gráfico se diseñará el **ruteo** y la **orquestación** de los mensajes del mismo. Por debajo lo que se hará implícitamente, será traducir lo representado en el entorno gráfico a un lenguaje llamado **BPEL** (**B**usiness **P**rocess **E**xecution **L**anguage), *Lenguaje de Ejecución de Procesos de Negocio*.



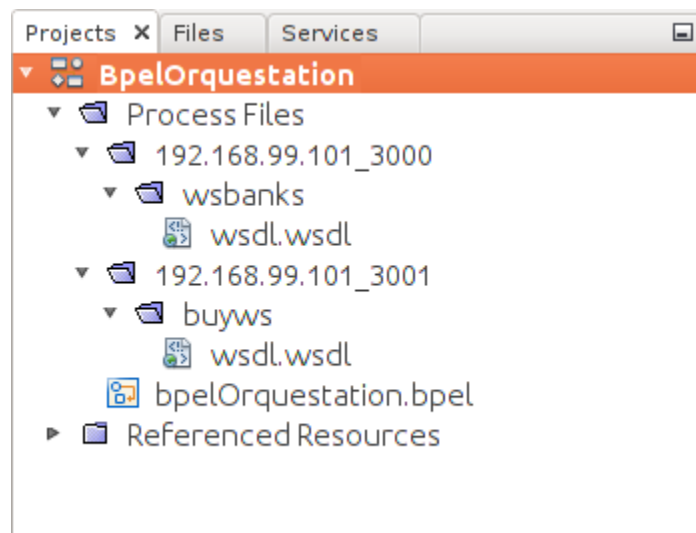
12. Agregar la **referencia** de uno de los servicios web existentes. En este caso se comenzara por el servicio expuesto por la aplicación **Store**. Para hacer esto, realizar clic derecho sobre el proyecto **BpelOrquestation** y seleccionar **External WSDL Document(s)**.



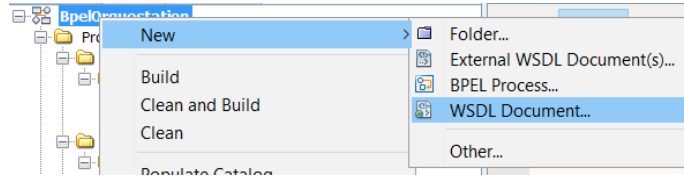
13. Agregar la referencia del servicio web expuesto por la aplicación **Store**. Seleccionar la opción “**From URL**” y en el campo referente a ésta, agregar la ruta del **WSDL** expuesta por la aplicación **Store** en rancher.



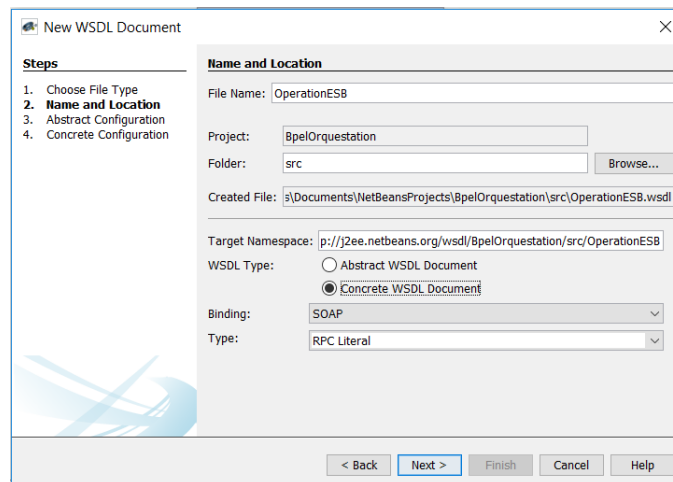
14. A continuación se realizara los mismos pasos para agregar el servicio expuesto por Bank.
15. Como resultado final se obtendrá:



16. Generar un **WSDL Document** para consumir los servicios desde la interfaz del bus.



17. Rellenar los campos de la ventana como se indica a continuación y al resto de configuración se le dejara por defecto:



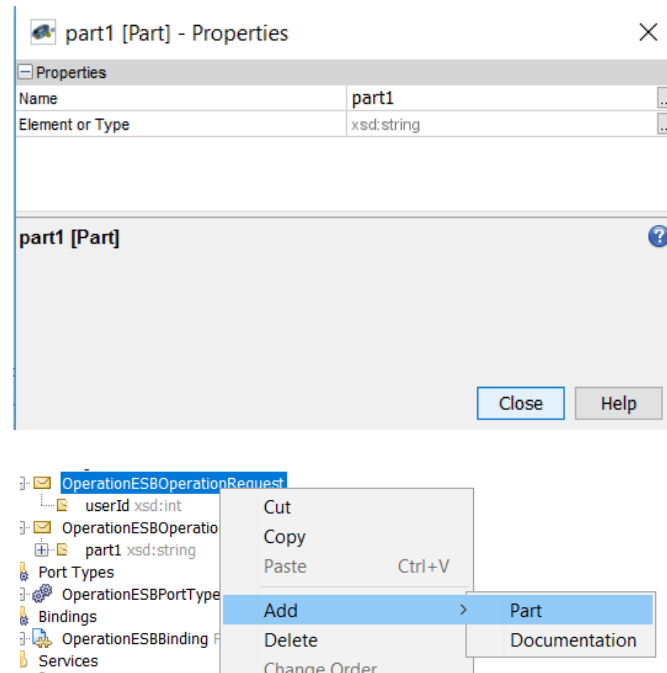
18. Al acabar la configuración se abrirá una pestaña en la mesa de trabajo, en la que se configura los parámetros de entrada y salida del **bus de servicios**. Para esto se deben agregar las variables que recibirá la interfaz del **bus de servicios**. Las cuales son las siguientes:

userId => Integer
productId => Integer
startAccount => Integer
endAccount => Integer
amount => Double

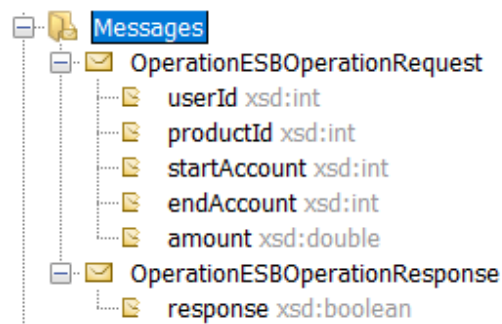
response => Boolean

19. En la nueva pestaña ubicar la siguiente opción “**OperationESBOperationRequest**” que es la encargada de las variables de entrada y “**OperationESBOperationResponse**” encargada de las variables de salida. Seleccionar la variable por defecto **part1** y editar sus

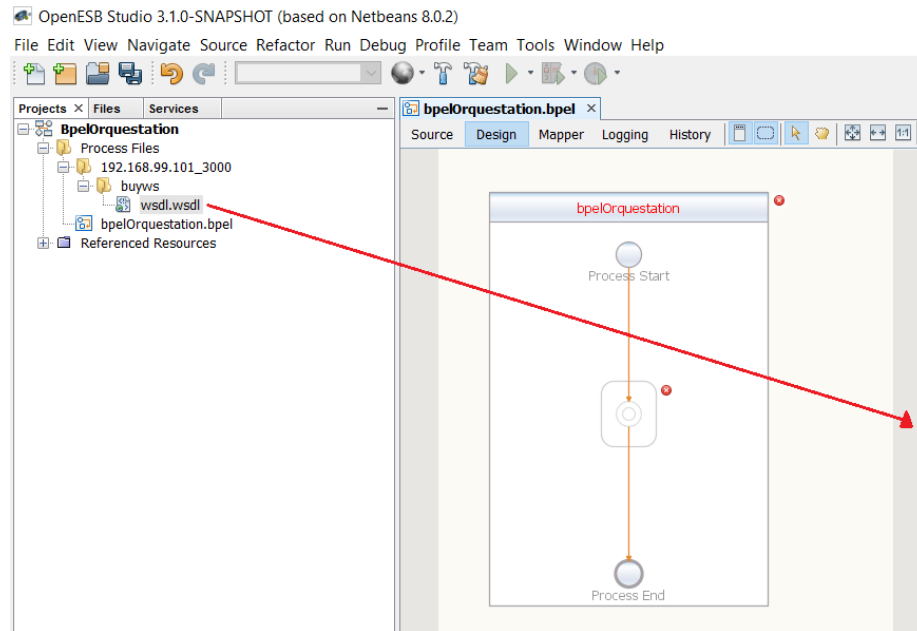
propiedades. Además, se pueden agregar más variables haciendo clic derecho sobre “OperationESBOperationRequest” y posteriormente en **Add > Part**.



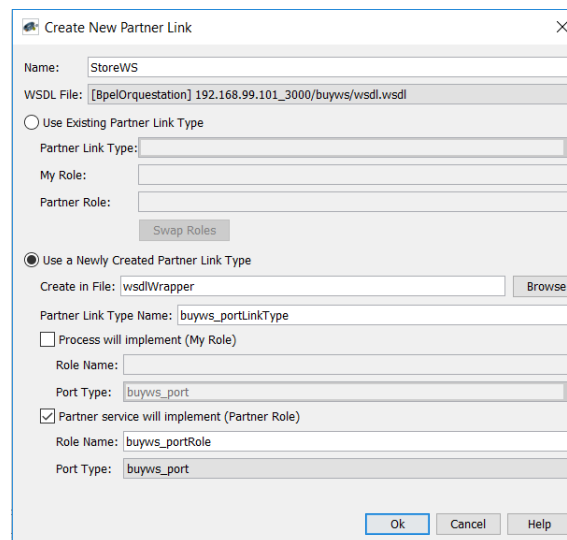
20. Agregar y configurar las variables tanto de entrada como de salida para obtener el siguiente resultado:



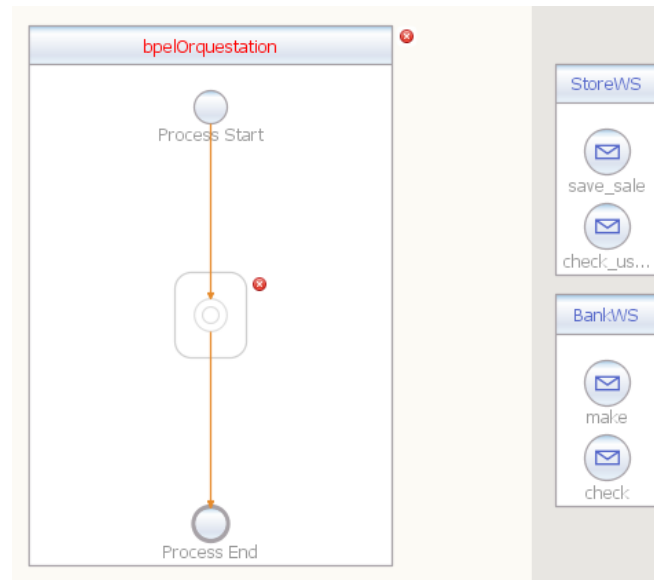
21. El paso siguiente es arrastrar los documentos WSDL que se importaron a la parte derecha de nuestra mesa de trabajo. Como se muestra a continuación.



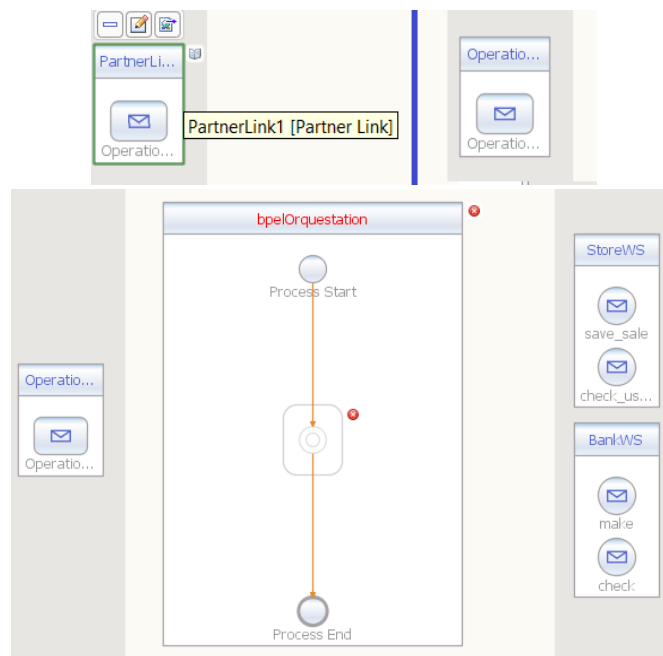
22. Lo anterior creará un nuevo **Partner Link**, el cual será la interfaz empleada por el Bus para acceder al **servicio**. Donde se le configurara con el nombre de “**StoreWS**” y finalmente clic en el botón “**Ok**”.



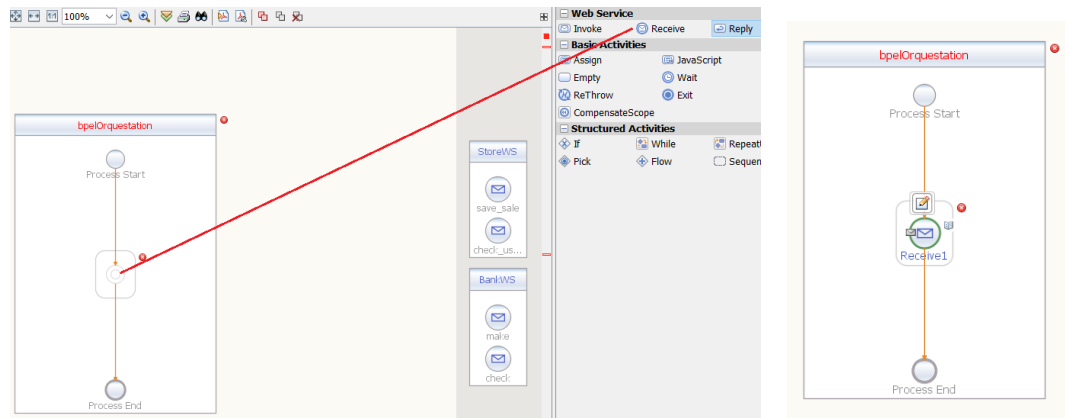
23. Realizar los mismos pasos con el WSDL de la aplicación **Bank**, y colocar el nombre de **BankWS**. Hasta este momento, el modelo del **bus de servicios** que se está elaborando tiene la siguiente forma:



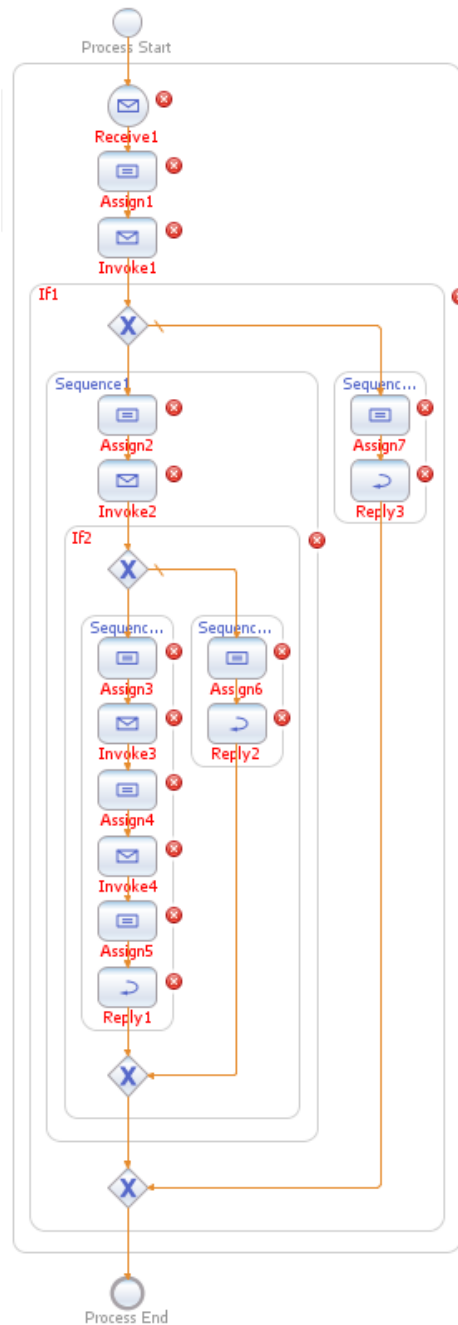
24. Arrastrar el documento WSDL creado al lado izquierdo de nuestra mesa de trabajo y cambiar el nombre de **PartnerLink** a **OperationESB**. Hasta este momento, el modelo de **Bus de Servicios** queda de la siguiente forma:



25. Para la realizar la orquestación dirigirse a la parte superior derecha y comenzar a agregar los tipos de tareas que se utilizaran en el diseño de **ruteo** y **orquestación** de los servicios. Realizar clic sostenido sobre la actividad “**Receive**” y agregarla al modelo de la siguiente forma:

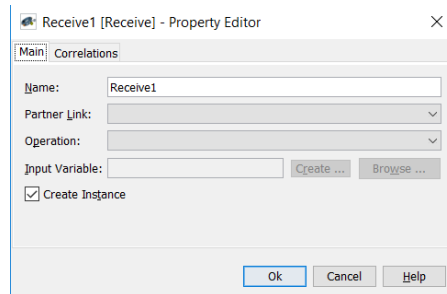


26. Dado a que la herramienta es muy intuitiva, se omitirán los pasos restantes de la construcción del **ruteo** y **orquestación** de los servicios. Y el modelo resultante y al que se debe llegar, es el siguiente:

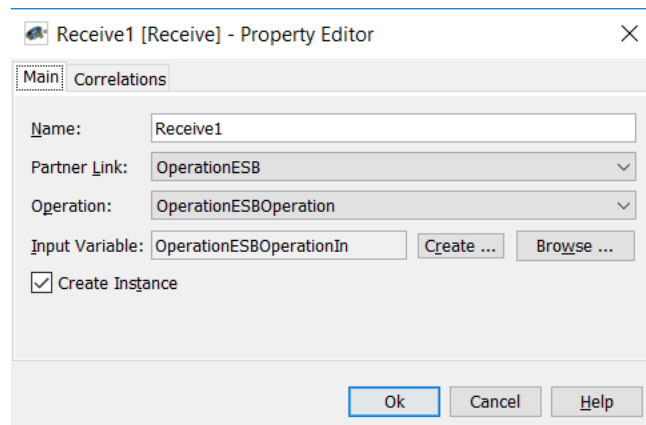
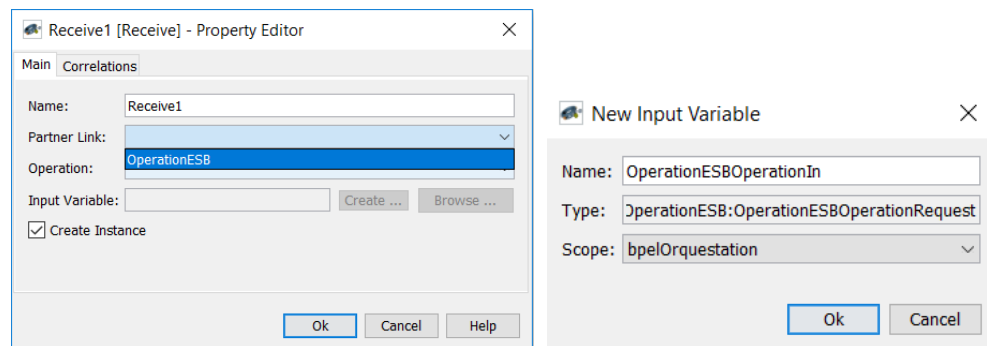


Como se puede observar en la figura, se trata de un modelo similar a un diagrama de flujo, que expresa de manera conceptual el funcionamiento del **Bus de Servicios** que genera por debajo un archivo en lenguaje **BPEL**.

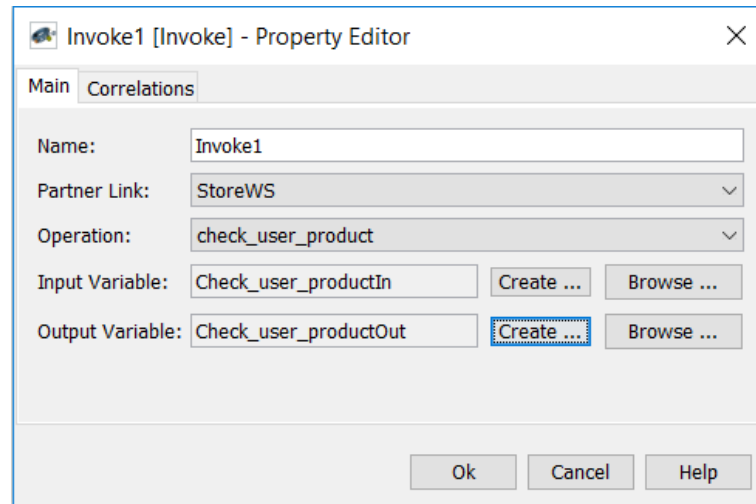
27. Una vez desarrollado el **modelo de ruteo** en la orquestación de servicios, se realizara la configuración de las variables que recibirá cada uno de los componentes del **Bus**. Para hacer dicha asignación, realizar doble clic sobre cada una de las actividades. Primero comenzar con la actividad **Receive1**. Esta actividad se encargará de recibir los datos que provengan de la aplicación que invoca al Bus de Servicios.



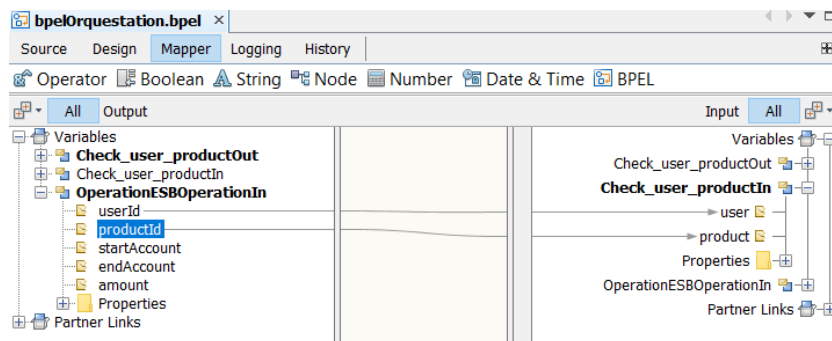
28. Asignar un nombre a la variable de entrada, la cual representa a todos los datos que se reciben en el **Bus**, se deja como esta predeterminado. Quedando la configuración final para la tarea **Receive1** de la siguiente manera:



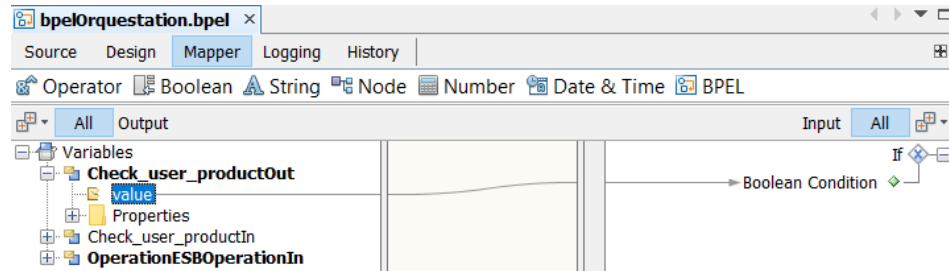
29. Configurar la actividad **Invoke1**. De la misma forma que en la configuración de la actividad Receive1, realizar doble clic sobre la actividad para configurarla.



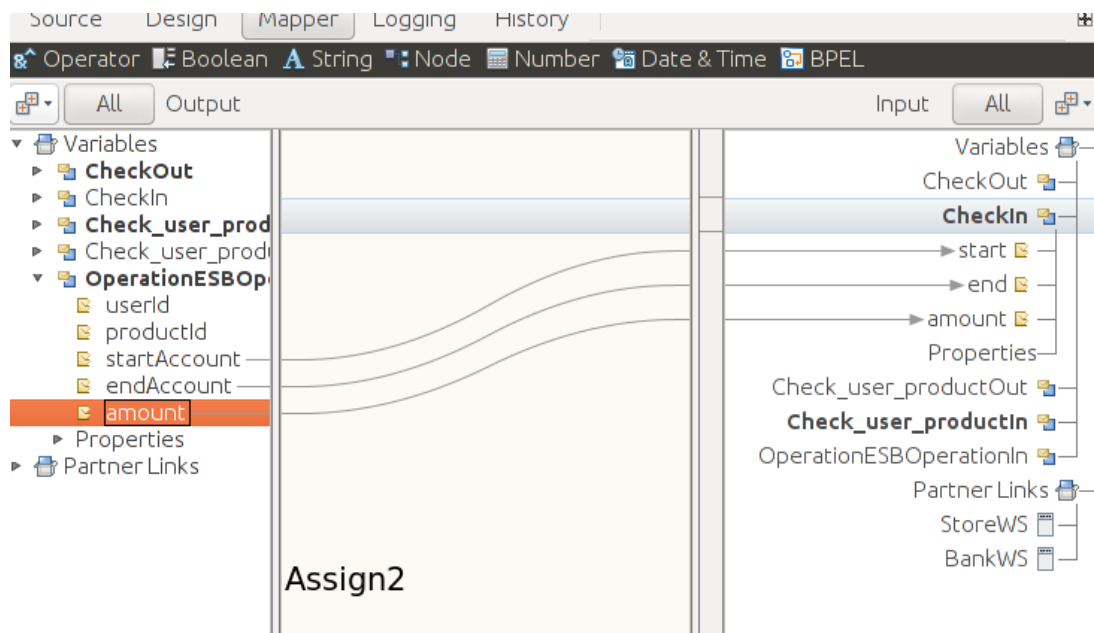
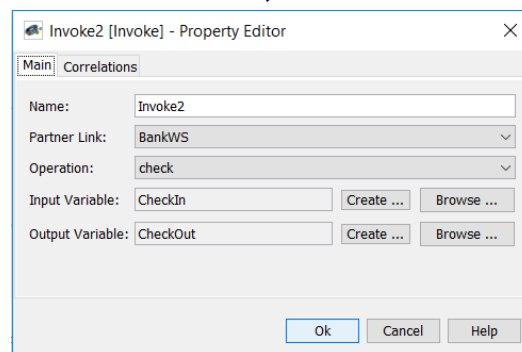
30. Configurar la actividad **Assign1**. Este tipo de actividad es la encargada de la asignación de variables a través del **Bus de Servicios**. Para hacer esto, se deben relacionar dos variables de la entrada (**OperationESBOperationIn**) al bus de servicios, con el servicio de verificar registro (**CheckIn**), ofrecido por **Store** (para relacionar las variables basta con hacer clic sostenido de una variable a otra, y soltar):



31. A continuación, configurar la actividad **if**. Realizar doble clic sobre la actividad **if**, y relacionar las variables tal como se explicó anteriormente. Lo que se está haciendo en este paso es relacionar la salida del servicio **CheckOut**, proveído por Store, a la variable de entrada del condicional:



32. De aquí en adelante se mostrarán las configuraciones de los demás componentes, en forma de pantallazos como guía. Se aclara que la orquestación establecida es la de verificación de los objetos en cada uno de los servicios web y después si se realiza las transacciones cuando todo es válido. Primero configurar la siguiente sentencia de control **IF** (Correspondiente a la validación en **Bank**).





33. Configurar el camino de la condición falsa en la sentencia de control IF2.

Reply2 [Reply] - Property Editor

Main Correlations

Name: Reply2

Partner Link: OperationESB

Operation: OperationESBOperation

☒ Normal Response

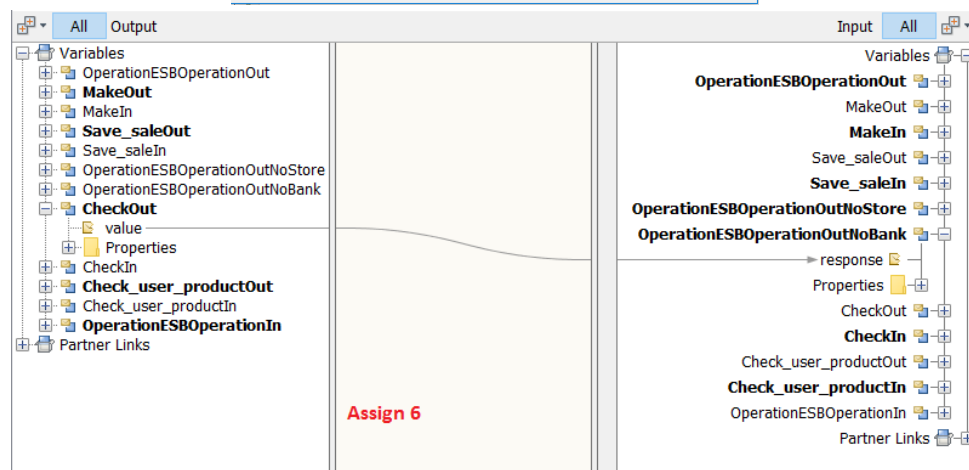
Output Variable: perationESBOperationOutNoBank Create ... Browse ...

☐ Fault Response

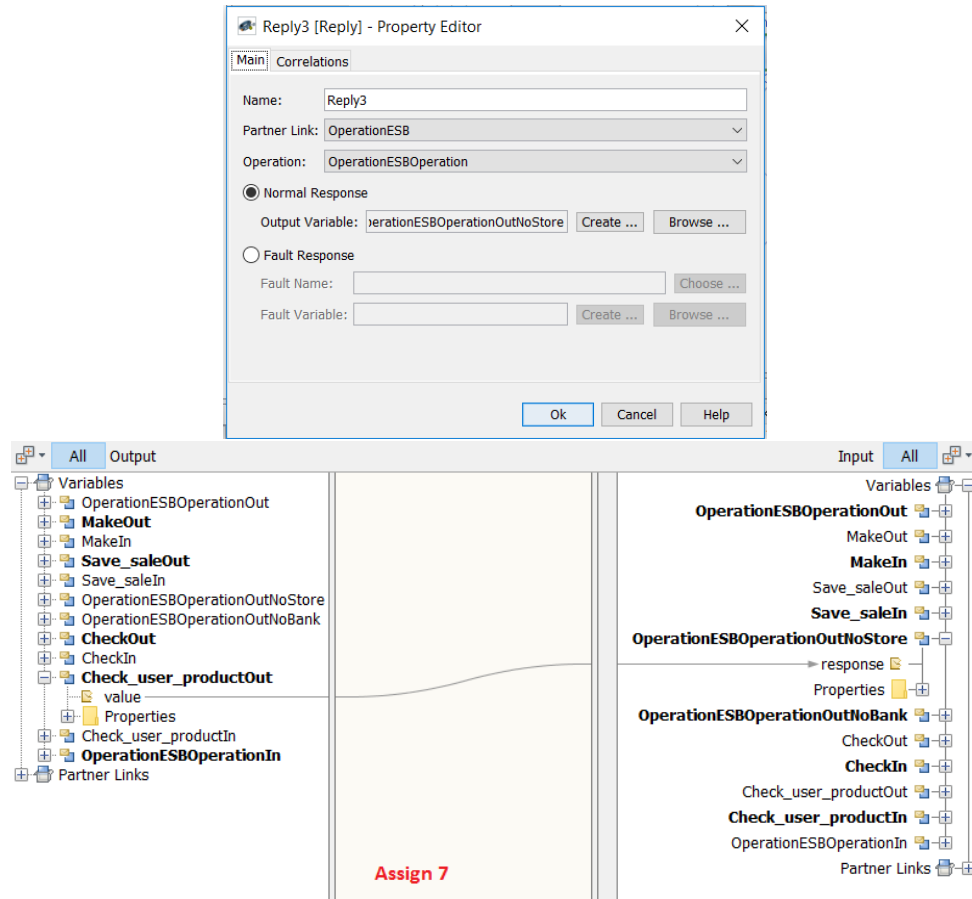
Fault Name: Choose ...

Fault Variable: Create ... Browse ...

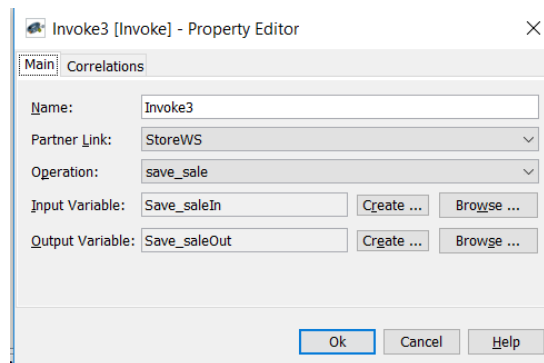
Ok Cancel Help

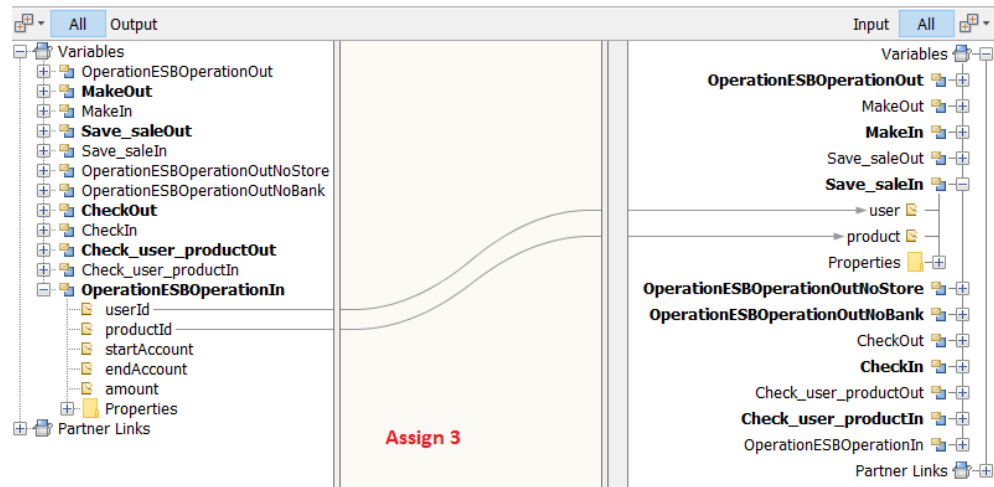


34. Configurar el camino de la condición falsa en la sentencia de control IF1



35. Como configuración final, realizar el camino faltante de las sentencias de control.





Invoke4 [Invoke] - Property Editor

Main Correlations

Name: Invoke4

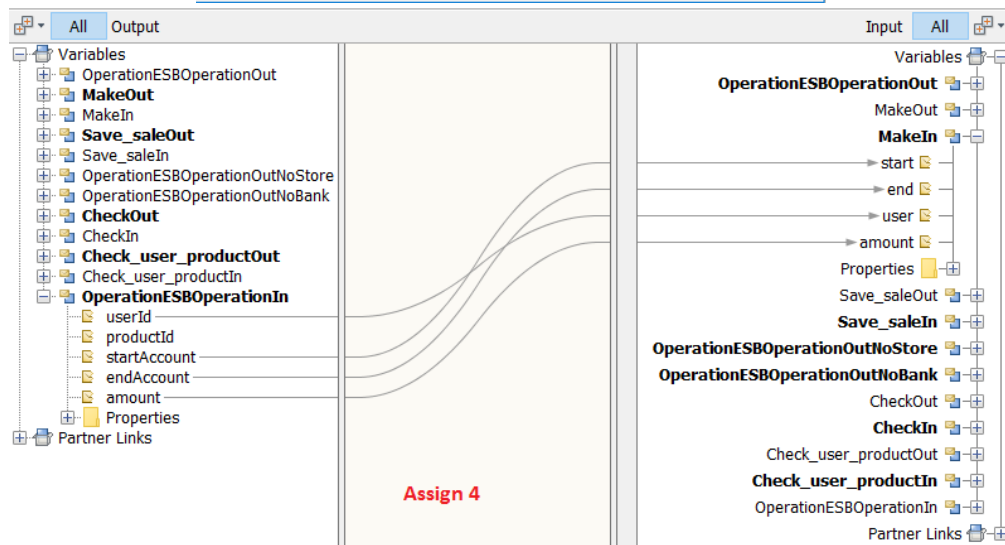
Partner Link: BankWS

Operation: make

Input Variable: MakeIn Create ... Browse ...

Output Variable: MakeOut Create ... Browse ...

Ok Cancel Help



Reply1 [Reply] - Property Editor

Main Correlations

Name: Reply1

Partner Link: OperationESB

Operation: OperationESBOperation

☒ Normal Response

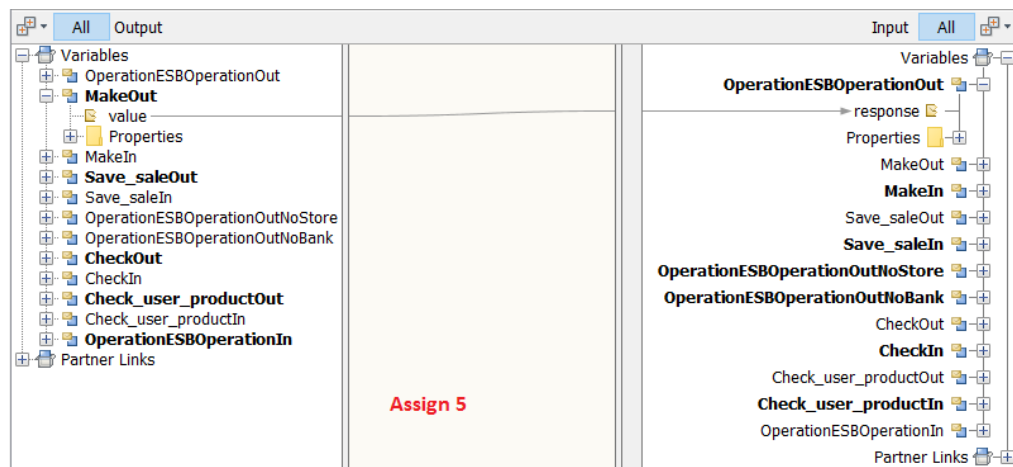
Output Variable: OperationESBOperationOut Create ... Browse ...

☐ Fault Response

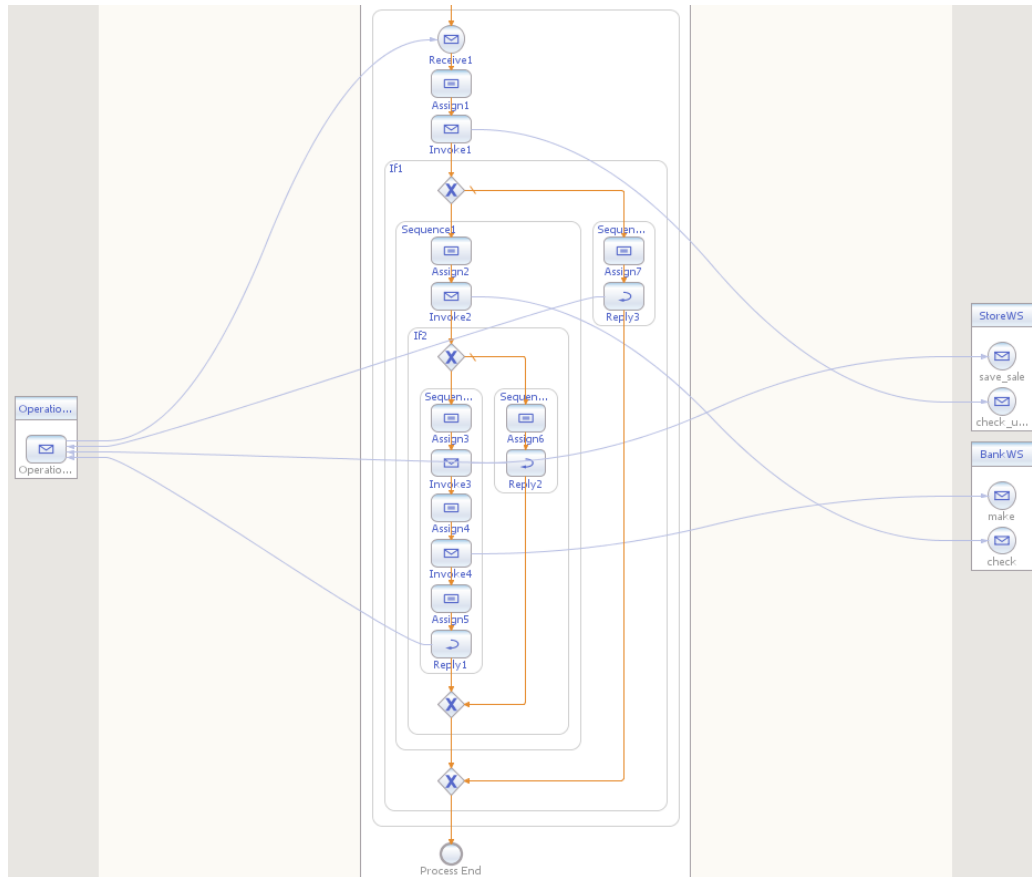
Fault Name: Choose ...

Fault Variable: Create ... Browse ...

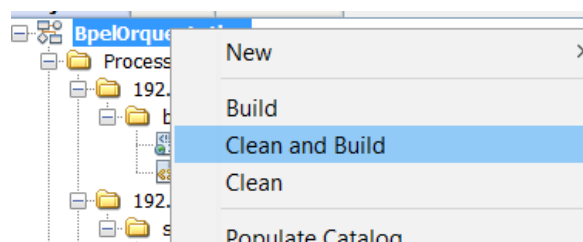
Ok Cancel Help



36. De esta manera, el modelo final del **Bus de Servicios** construido es el siguiente:



37. Realizar clic derecho sobre la aplicación **BpelOrquestation** y hacer clic en “**Clean and Build**”.



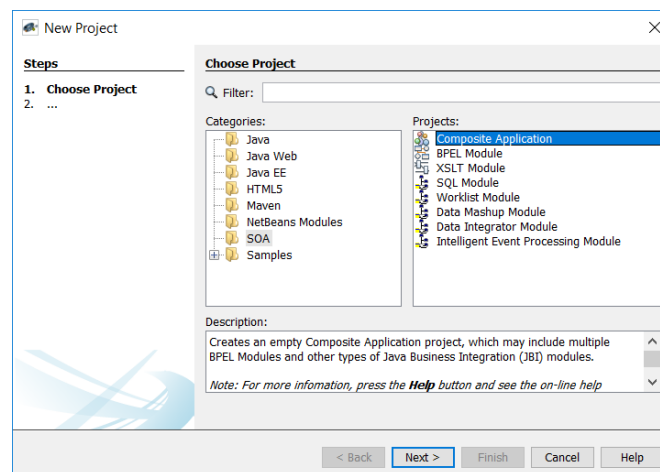
```

Notifications | Test Results | Output X
OpenESB Standalone x Retriever Output x XML check x build.xml (clean,dist_se) x
init-check:
init-taskdefs:
init:
pre-dist:
deps-jar-dist:
do-dist:
Created dir: C:\Users\Noctis\Documents\NetBeansProjects\BpelOrquestation\build
C:/Users/Noctis/Documents/NetBeansProjects/BpelOrquestation/src/bpelOrquestation.bpel:26: 8
WARNING: A part of variable "Save_saleOut.value" is not used.

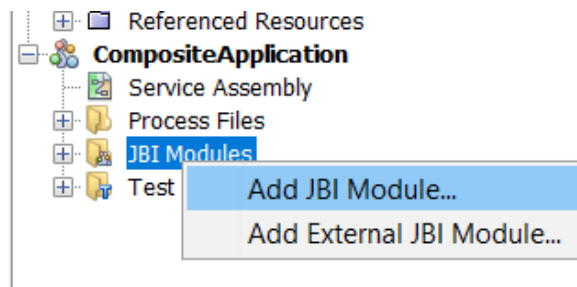
Copying 5 files to C:\Users\Noctis\Documents\NetBeansProjects\BpelOrquestation\build
Building jar: C:\Users\Noctis\Documents\NetBeansProjects\BpelOrquestation\build\SEDeployment.jar
post-dist:
dist_se:
BUILD SUCCESSFUL (total time: 0 seconds)

```

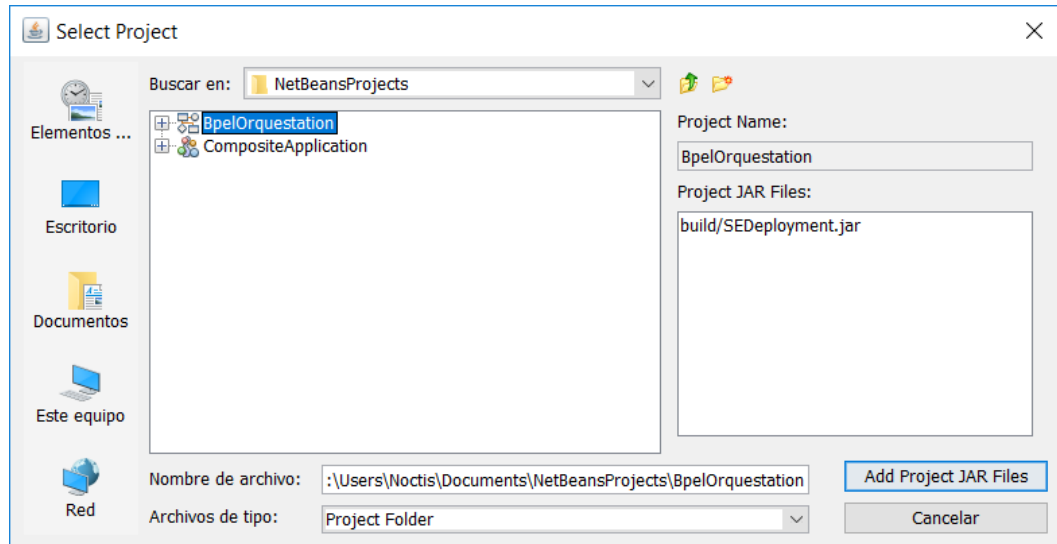
38. El siguiente paso creara una nueva **Composite Application**, en **File -> New Project**, a la cual se le asignara el nombre de “**CompositeApplication**”



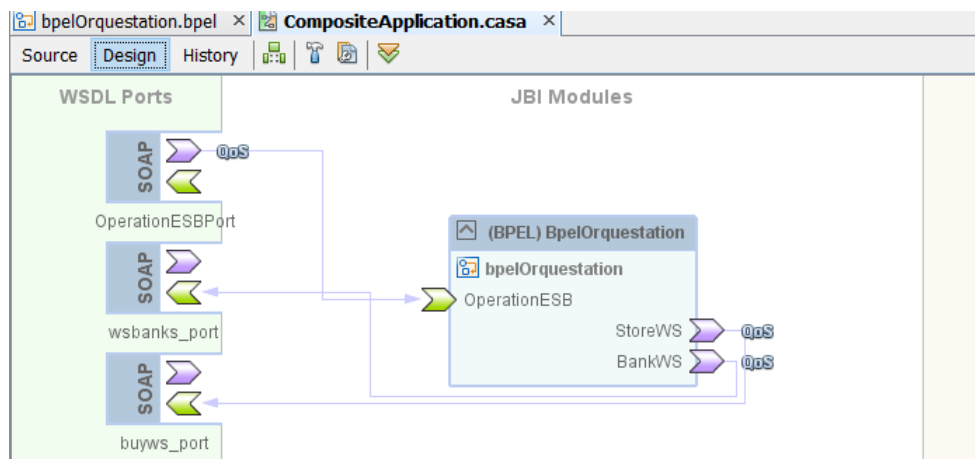
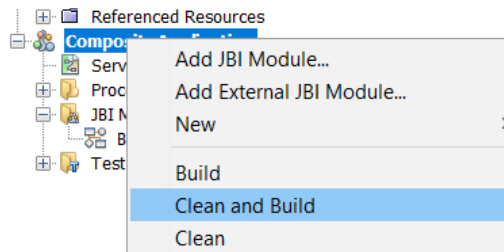
39. Una vez situados sobre la aplicación, realizar clic derecho sobre la carpeta **JBI Modules**, y seleccionar la opción “**Add JBI Module**”:



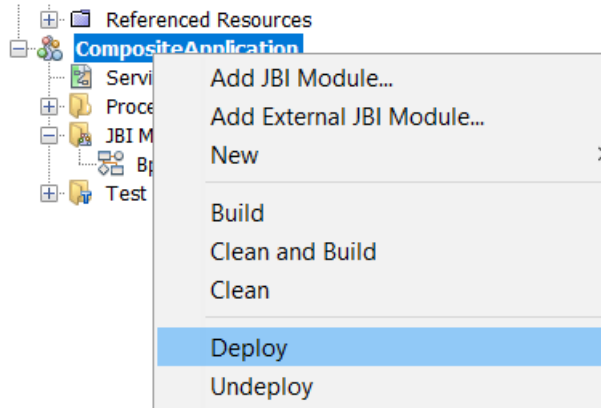
40. Seleccionar el archivo **BpelOrquestation** creado anteriormente.



41. Realizar clic derecho sobre la aplicación **CompositeApplication** y seleccionar “**Clean and Build**”. Una vez hecho esto, el IDE deberá mostrar lo siguiente:

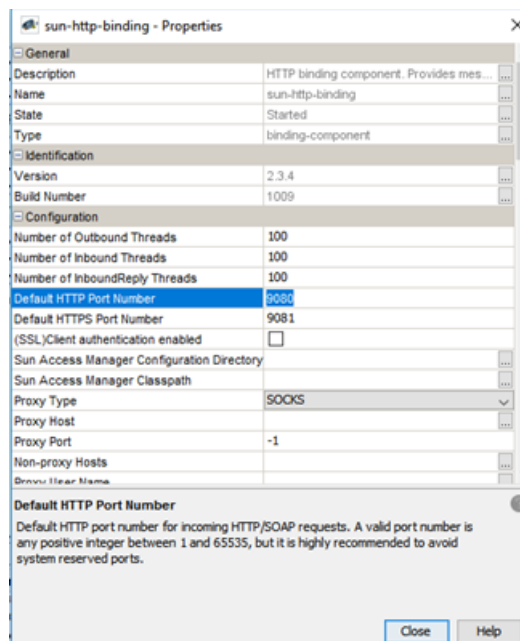


42. Desplegar la aplicación **CompositeApplication**.



43. Finalmente, se debe verificar que el **WSDL** de nuestro **Bus de Servicios** ha sido creado.

- a. Se verifica el puerto donde está desplegado el Bus de Servicios: **Servers > OpenESB Standalone > Instance > JBI > Binding Components > sun-http-binding > Properties:**



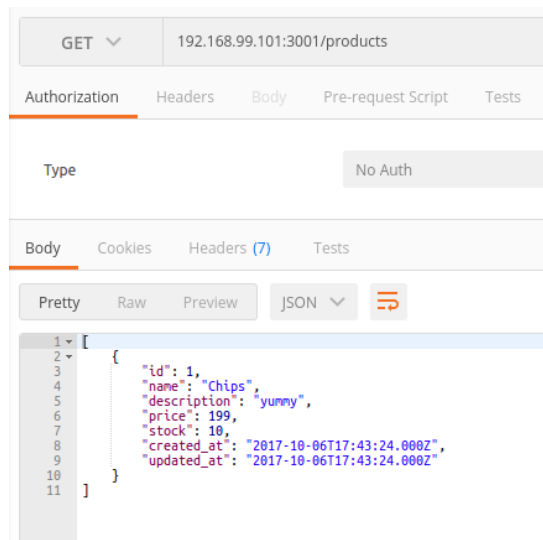
Como se puede observar, el puerto donde se encuentra desplegado el **Bus de Servicios** es el **9080**.

- b. Verificar que el **WSDL** se encuentre desplegado, ingresando a la ruta:
<http://localhost:9080/OperationESBService/OperationESBPort?wsdl>



Comprobación del bus de servicios

44. Se realizará un caso de ejemplo como se muestra a continuación, por lo cual se tendrán los siguientes datos en la base de datos.



GET 192.168.99.101:3001/users

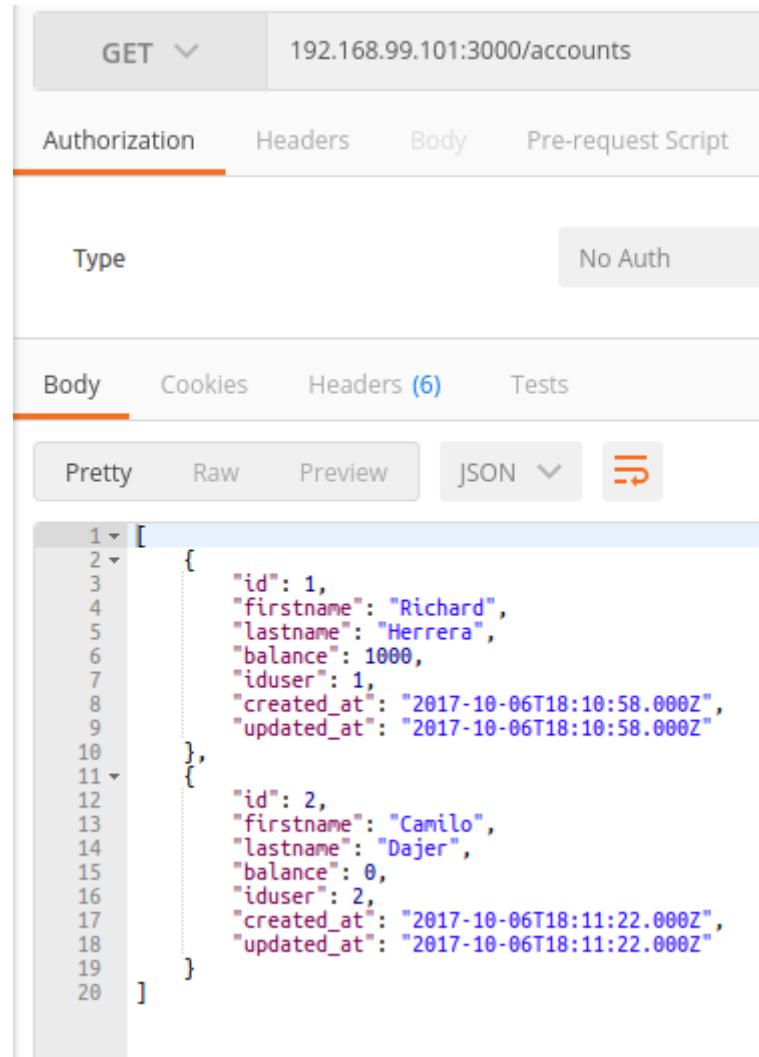
Authorization Headers Body Pre-request Script

Type No Auth

Body Cookies Headers (7) Tests

Pretty Raw Preview JSON ↕

```
1 [
2   {
3     "id": 1,
4     "firstname": "Richard",
5     "lastname": "Herrera",
6     "document": null,
7     "created_at": "2017-10-06T18:09:27.000Z",
8     "updated_at": "2017-10-06T18:09:27.000Z"
9   },
10  {
11    "id": 2,
12    "firstname": "Camilo",
13    "lastname": "Dajer",
14    "document": null,
15    "created_at": "2017-10-06T18:11:09.000Z",
16    "updated_at": "2017-10-06T18:11:09.000Z"
17  }
18 ]
```



45. Se debe tener en cuenta que el documento WSDL generado por el bus de servicio lo generará con la IP que contenga el computador que lo despliega, y en caso de no tener una IP le colocará como predeterminado "localhost", para el reconocimiento de la ubicación se asigna la IP **192.168.0.8** física el computador. Y con esto se realizara la nueva búsqueda del documento WSDL.

← → ↻ ↗ 192.168.0.8:9080/OperationESBService/OperationESBPort?wsdl

Aplicaciones Facebook Software Engine inicio [ISIS2503 A] Chartkick onWebChat Open Energy Motion Urban So

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://j2ee.netbeans.org/wsdl/Bpel0rquestation/src/OperationESB"
xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="OperationESB"
targetNamespace="http://j2ee.netbeans.org/wsdl/Bpel0rquestation/src/OperationESB">
  <types></types>
  <message name="OperationESB0perationResponse">
    <part name="response" type="xsd:boolean"/></part>
  </message>
  <message name="OperationESB0perationRequest">
    <part name="userId" type="xsd:integer"/></part>
    <part name="productId" type="xsd:integer"/></part>
    <part name="startAccount" type="xsd:integer"/></part>
    <part name="endAccount" type="xsd:integer"/></part>
    <part name="amount" type="xsd:double"/></part>
  </message>
  <portType name="OperationESBPortType">
    <operation name="OperationESB0peration">
      <input message="tns:OperationESB0perationRequest" name="input1"/></input>
      <output message="tns:OperationESB0perationResponse" name="output1"/></output>
    </operation>
  </portType>
  <binding name="OperationESBBinding" type="tns:OperationESBPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  </binding>
</definitions>
```

46. Utilizar la herramienta **SoapUI**, para verificar que el servicio expuesto por el bus de servicio funcione de la forma en la que se configuro. Por lo tanto crear un nuevo proyecto SOAP y realizar las siguientes configuraciones:

New SOAP Project

Creates a WSDL/SOAP based Project in this workspace

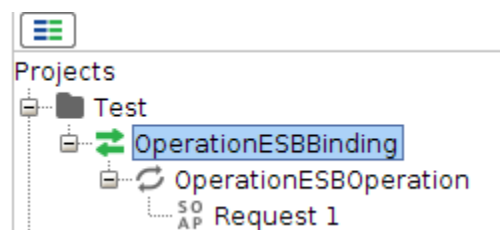
Project Name:

Initial WSDL:

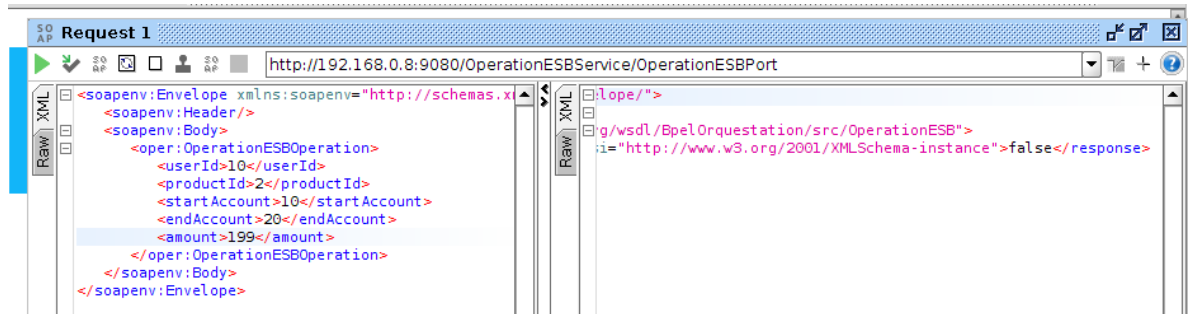
Create Requests: ☒ Create sample requests for all operations?

Create TestSuite: ☐ Creates a TestSuite for the imported WSDL

Relative Paths: ☐ Stores all file paths in project relatively to project file (requires save)



47. Para el primer caso se comprueba en que no pase la primera validación, es decir que el usuario y/o el producto no existan.



48. Realizar como **ejercicio** los otros dos casos: validar en caso de que si exista el usuario y el producto, pero no exista alguna de las dos cuentas. Y el caso final en que todo sea correcto, verificando que en las bases de datos esté reflejada la transacción.