Enterprise Service Bus

Integration Specialist

soa|expert

soa|expert

ÍNDICE PÁGINA

1 História do ESB **1**

1.1 Arquitetura Desacoplada 1

1.2.1 Oracla service Bus - Arquitetura Desacoplada 3

1.2.2 Transparência de localização 3

1.2.3 Mediação 3

1.3 Schema Transformation 4

1.4 Service Aggregation 4

1.5 Load Balancing 4

1.6 Reforço de Segurança 4

2 SOA e Service BUS **5**

3 Instalando e Configurando o OSB **6**

3.1 Criando um domínio OSB 11

4 Fluxo de Mensagens – Básico **19**

5 Tratamento de Erros **42**

5.1 Lidando com exceções esperadas 42

5.1.1 Lidando com exceções no OSB 43

5.1.2 Tratamento padrão de erros 47

5.1.3 Notificando os usuários 47

5.1.4 Implementação de referência - Alerts 47

5.1.5 Implementação de referência - node Error Handler 48

Confidencial – Todos os Direitos Reservados para SOAExpert.

soa|expert

ÍNDICE PÁGINA

6 Mensageria Assíncrona - JMS 52
6.1 Configurando o Weblogic Server 52
6.2 Bussiness Service Assíncrono 59

7 Bindings e camadas de transporte 62
7.1 Service Types e Transport Protocols 62
7.2 SOAP com WSDL 63
7.2.1 SOAP sem WSDL 63
7.3 XML com WSDL 67
7.3.1 XML sem WSDL 72
7.4 Definindo o Business Service 77
7.5 Criando o Proxy Service 80
7.5.1 Message Type: Text 82
7.5.2 Message Type XML 82

8 Transporte EJB 83
8.1 Criar e fazer deploy de um EJB SessionBean Statless no WLS 83
8.2 Criar um Business Service que sabe como invocar o EJB 83
8.3 Criar um Proxy Service que retorna taxa de acordo 83

Confidencial – Todos os Direitos Reservados para SOAExpert.

soa|expert

ÍNDICE PÁGINA

9 Exercícios Adicionais de Reforço 91
9.1 Definindo Contratos 93
9.2 Criando um Projeto no OSB 94
9.3 Business Service: CreditoService.biz 95
9.4 Proxy Service: EmpresaProxy.proxy 97
9.5 Proxy Service: VerificaCadastroProxy.proxy 109
9.6 Filas para os serviços de mensageria, Aprovado e Rejeitado 113
9.7 Serviços Assíncronos Aprovados e Rejeitados 114
9.8 Proxies Aprovado e Rejeitado 116
9.9 Cadastra Proxy 118
9.10 Agregação EmpresaFacade.proxy 119
9.11 ProcessEmpresa - "Orquestrador através do 122
9.12 Processamento de Lote de Arquivos - MFL Builder 127
9.13 Transformação de tipos - Xquery 130
9.14 Proxy de ETL e Disparo de processamento 131
9.15 Monitoria e Relatório | Criando um Relatório Aprova/Rejeita 134
9.16 Criando um Relatório para Verificação de Crédito 136
9.17 Criando um Alerta de Erros 137
9.18 Configurando Regras de SLA 139

Confidencial – Todos os Direitos Reservados para SOAExpert.

1 História do ESB

Nos primórdios da tecnologia, o ESB era concebido apenas para fazer os Web Services disponíveis para os *consumers*. Eles não abraçavam muitos padrões simplesmente porque muitos destes ainda não existiam.

Então de ESB eles tinham na verdade somente o nome.

A indústria amadureceu o entendimento sobre o papel que o mesmo poderia ocupar, ficou mais claro e hoje é muito mais que um simples "*service enabling*".

Soluções para o mundo contemporâneo

A indústria de tecnologia está constantemente envolvida no entendimento dos problemas que as grandes companhias podem enfrentar no seu dia-a-dia. Os ESBs modernos são simplesmente uma das últimas ferramentas destinadas a auxiliar nessa empreitada.

Os benefícios reais da arquitetura SOA estão mudando a face das organizações e se os ESBs antigos não davam conta de todos os problemas, os atuais precisam resolver a maior parte deles.

Arquitetura Desacoplada

Muitos acreditam que os Web Services proporcionam desacoplamento entre os sistemas. Isso é parcialmente verdade. Web Services podem proporcionar algum tipo de desacoplamento, pois eles formalizam um contrato entre o *service consumer* e *o service provider*; modelo ***"design by contract"*** que também possui benefícios tangíveis. Entretanto você deve tomar cuidado na hora de criar um *Schema*, para que este seja neutro entre plataformas, proporcionando uma grande reutilização.

Contudo, se você der uma olhada mais a fundo no WSDL, verificará que os endpoints são retratados na descrição:

```
<service name="HelloWorldService">

<port binding = "s1:HelloWorldServiceSoapBinding

          name = "HelloWorldPortSoapPort">

<s2:address
location=http://www.soaexpert.com.br:8001/esb/Hello World/>

</port>

</service>
```

Por especificar a máquina e porta, ou conjunto de máquinas e portas, você está acoplando fortemente esse serviço a um computador específico através de uma expressão. Você poderia até utilizar um servidor DNS para substituir partes da URL ao invés de apontar para máquinas diretamente, porém servidores DNS não são adequados para isso e não possuem habilidade para entender e administrar o status dos serviços que estão rodando nesses servidores.

Por tal motivo, arquitetura desacoplada não é totalmente alcançada pelo WSDL ou por Web Services sozinhos. Uma solução mais robusta é necessária para fazer a mediação entre *serviceclients* e *producers*, capaz de lidar com diferentes tipos de transportes e tecnologias de segurança.

Por exemplo: um serviço que pode ser invocado através do transporte HTTP, mas acionado por um serviço de baixo nível como JMS, ou e-mail , FTP, entre outros.

Esta abordagem consiste em traduzir – "**wrap**" ordens de serviço para os transportes apropriados dos clientes.

Oracle Servic Bus - Arquitetura Desacoplada

Em acordo com os benefícios da arquitetura desacoplada do WSDL e XSD, o OSB possui a habilidade para armazenar WSDL, XSD, XSLT e outros tipos que ele reconhece como "**resources**"- recursos.

Estes resources estão disponíveis através do cluster de OSB, permitindo a reutilização dos recursos sempre que necessário.

O benefício pode não estar claro ainda neste ponto do curso, mas se você considerar que muitas companhias definem seus modelos de domínio em XML – Canonical Data Model e o OSB pode guardar esses schemas XSD, você pode facilmente reutilizá-los numa gama enorme de WSDLs. Isso possibilitará a você criar um padrão de data types e message formats.

Transparência de localização

A transparência de localização é uma estratégia para ocultar a localização física dos *endpoints* dos *service clients*. Idealmente um *service client* deve saber apenas sobre uma máquina e *port name* para cada serviço. Esse cliente não precisa saber sobre os atuais *endpoints*.

Isto permite uma grande flexibilidade quando administramos nossos serviços. Você pode adicionar e remover service endpoints sem necessidade de recompilar seus service clients.

Mediação

Um Enterprise Service Bus é uma camada independente residindo entre o *service client* e os *service providers*. Essa camada adiciona um grande valor à arquitetura, já que possui a habilidade de realizar múltiplas operações: pode transformar um formato de dados para outro, de acordo com os schemas, enviar e receber inteligentemente de acordo com o conteúdo das mensagens, rotear para vários *service endpoints*, etc.

Schema Transformation

O serviço publicado no barramento pode utilizar um schema diferente do que o schema que o negócio representa. Essa é uma capacidade vital, especialmente quando fazemos agregação de serviços ou orquestração. Então o ESB permite que um service client possa receber dados usando um schema diferente do service provider.

A habilidade de transformar dados de um tipo para outro é crítica para o sucesso de qualquer ESB.

Service Aggregation

Service Aggregation atua como um *façade* e realiza uma série de chamadas aos Web Services, representando um único Web Service.

Ele atua como o pattern, fazendo múltiplas chamadas e retornando um único resultado.

A orquestração é bem parecida com a agregação, a diferença é que a orquestração inclui condições lógicas para saber qual serviço será invocado e sua ordem.

Load Balancing

Alguns produtos oferecem a capacidade de realizar o balanceamento de carga entre os múltiplos *service endpoints*. Quando se registra um serviço no OSB, você pode informar uma lista de onde os serviços estão rodando e substituir a qualquer momento, sem necessidade de restart do servidor.

Reforço de Segurança

Você deve reforçar a segurança de uma maneira centralizada sempre que possível. O OSB lhe permite um grande nível de padronização e controle.

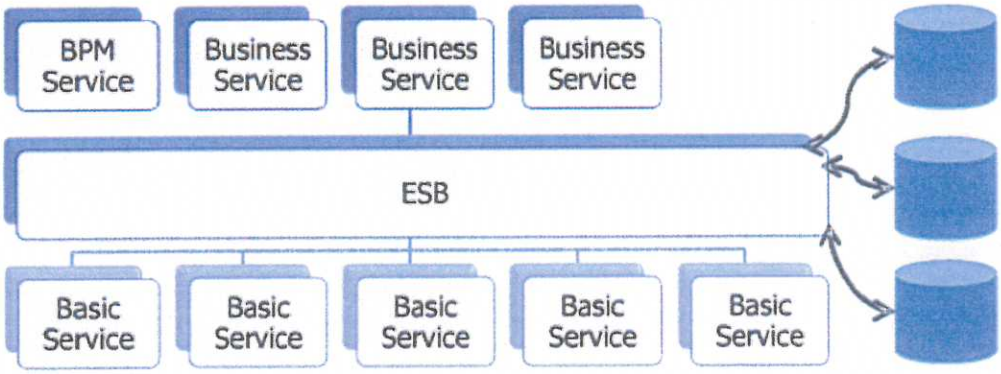
2 SOA e Service BUS

- SOA requer um serviço para conectar todos os participantes
- As características principais do bus incluem:

- Conectividade

- Suporte às tecnologias heterogêneas

- Suporte aos diferentes paradigmas de comunicação

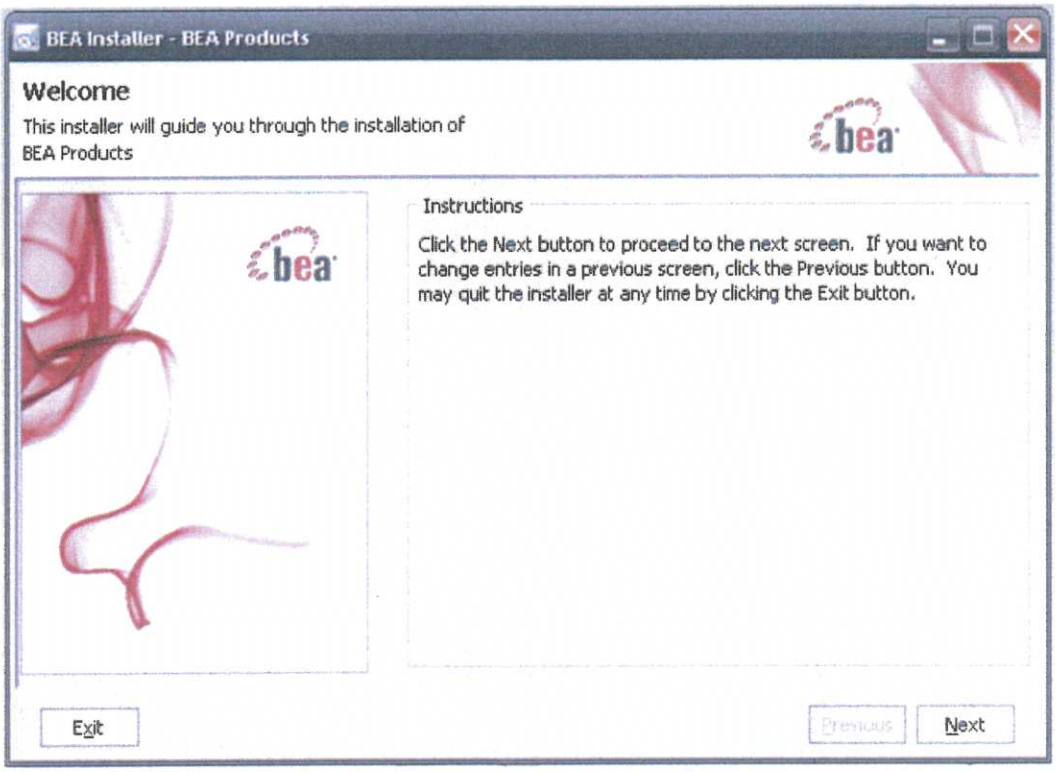


Um elemento que não faz parte do conceito SOA mas contribui fortemente para a implementação é o *Enterprise Service Bus,* o qual conecta todos os participantes numa mesma plataforma.

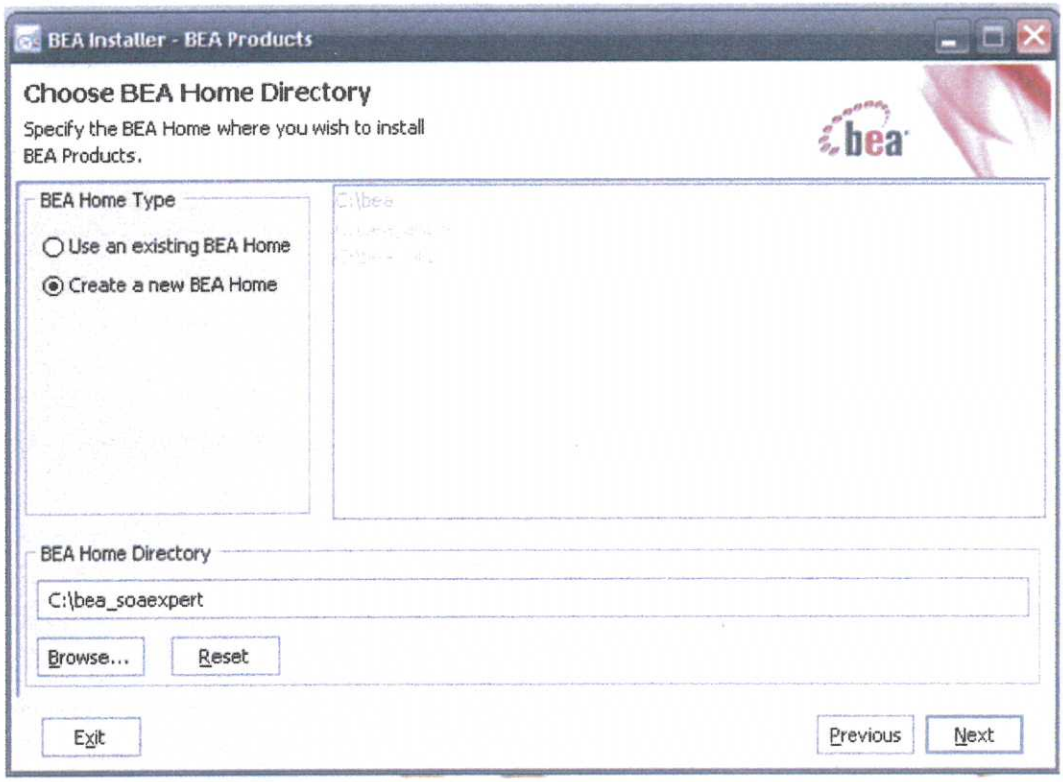
Esta plataforma incorpora diferentes tecnologias como *message-oriented middleware* (MOM) como *MQSeries* e *WebServices.* Também permeia por padrões e provisiona roteamento inteligente entre serviços e *endpoints* para estender as funcionalidades da companhia.

3. Instalando e Configurando o OSB

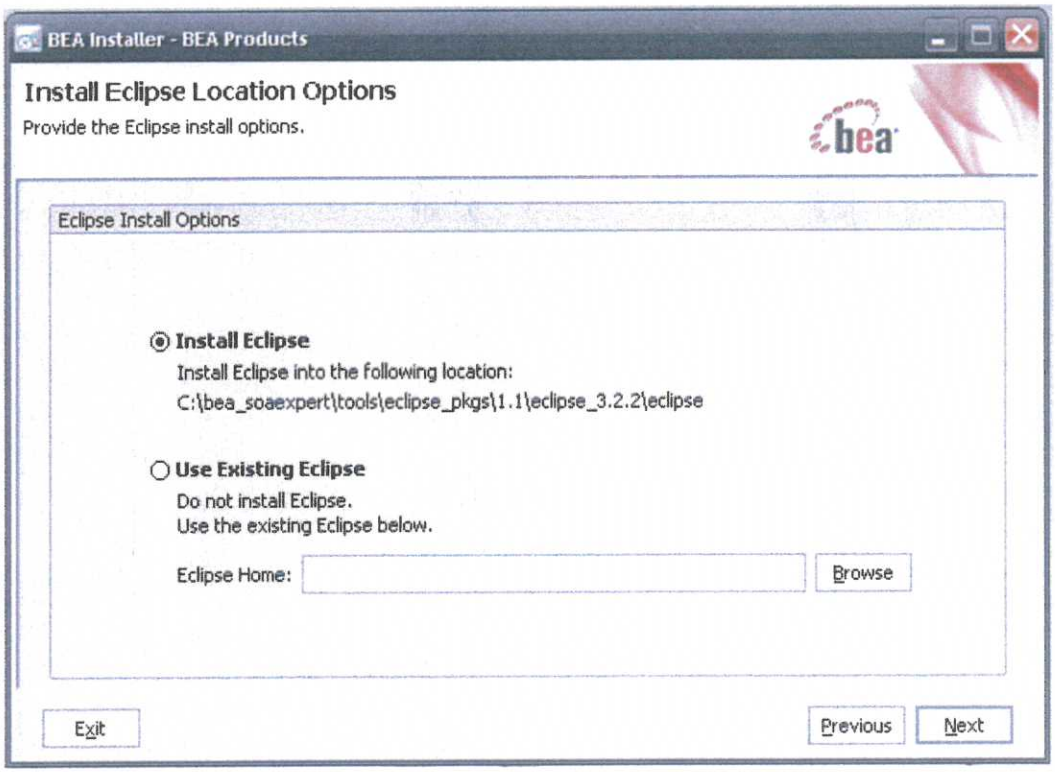
Acesse o arquivo executável da instalação do produto xxx.exe e dê dois cliques a fim de iniciar o procedimento de instalação.



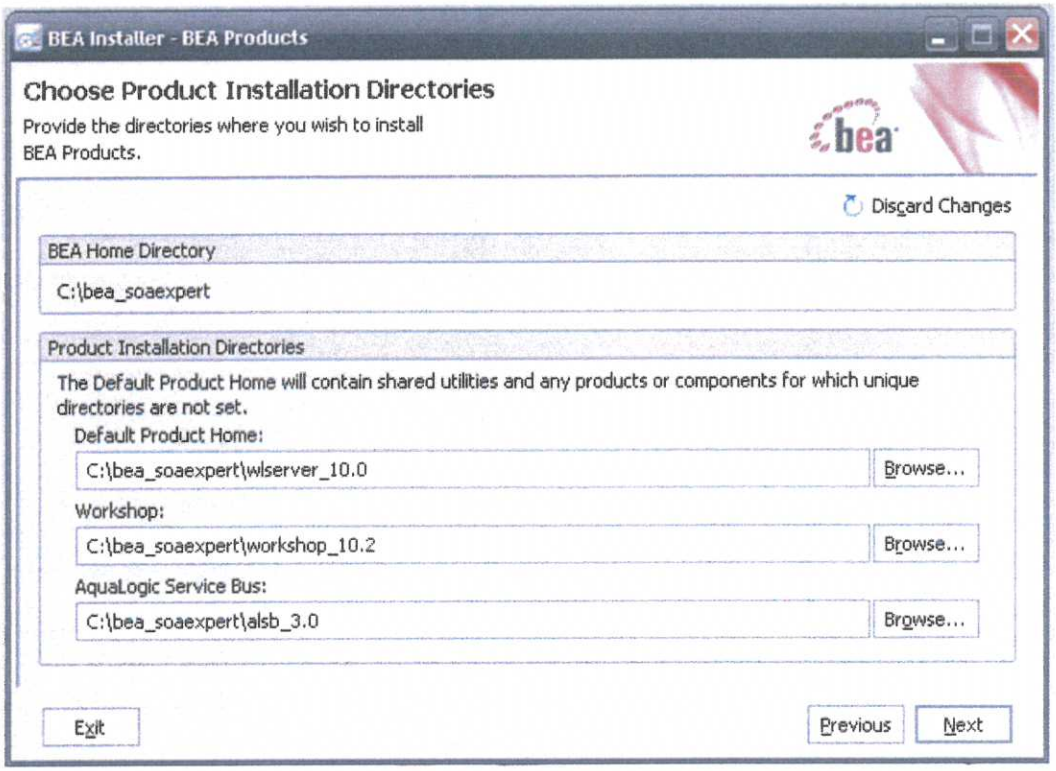
Clique em Next para continuar



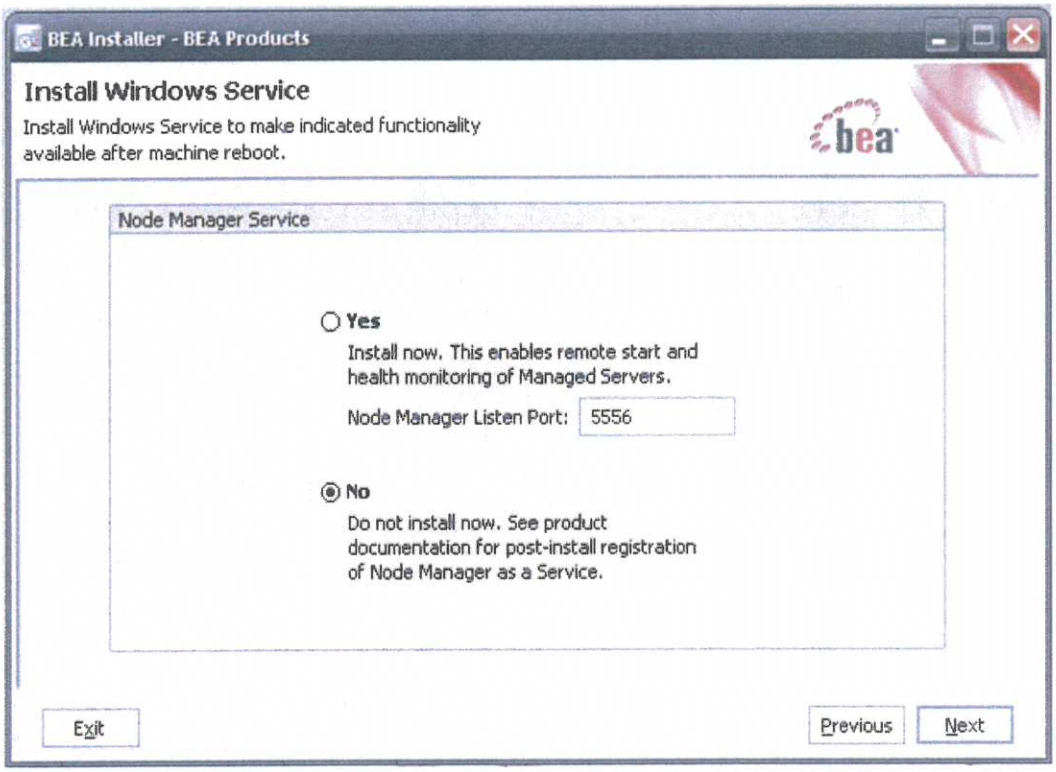
Por padrão, é sugerido o diretório bea, onde fica a instalação do produto.



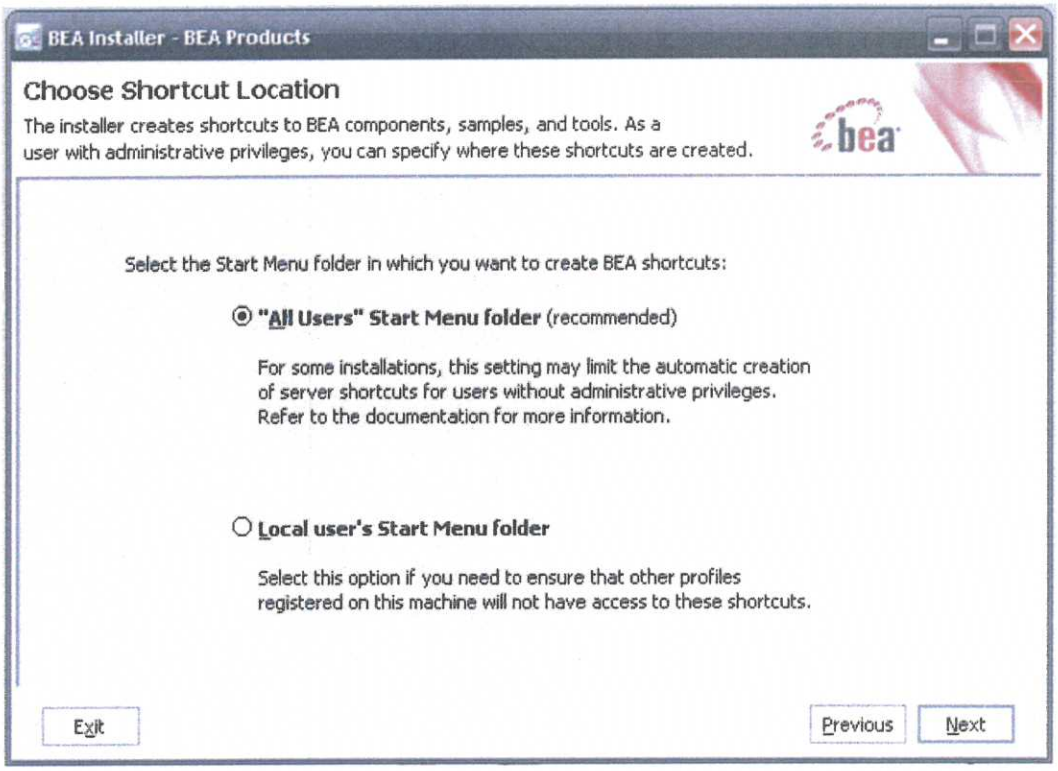
Você pode optar por instalar o plugin para o eclipse, embora a compatibilidade seja restrita.

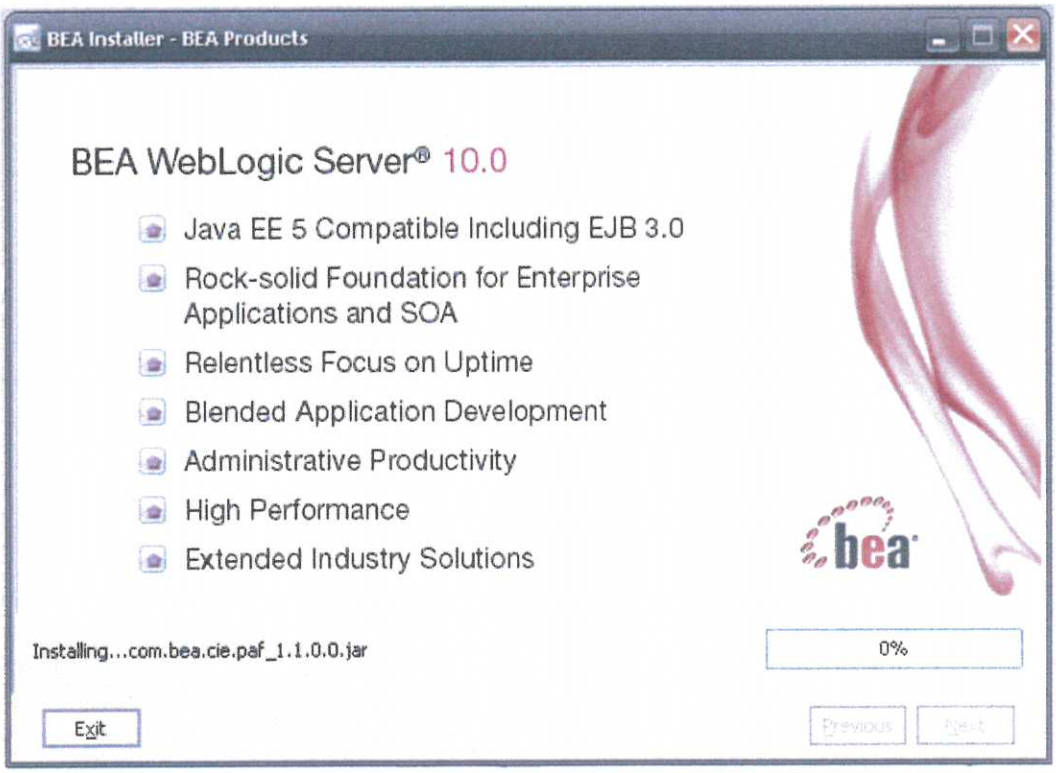


Sempre são instalados: Application Server, Workshop (Eclipse turbinado) e o OSB.



O Node Manager permite que se administre instâncias remotas de um cluster por exemplo do OSB.

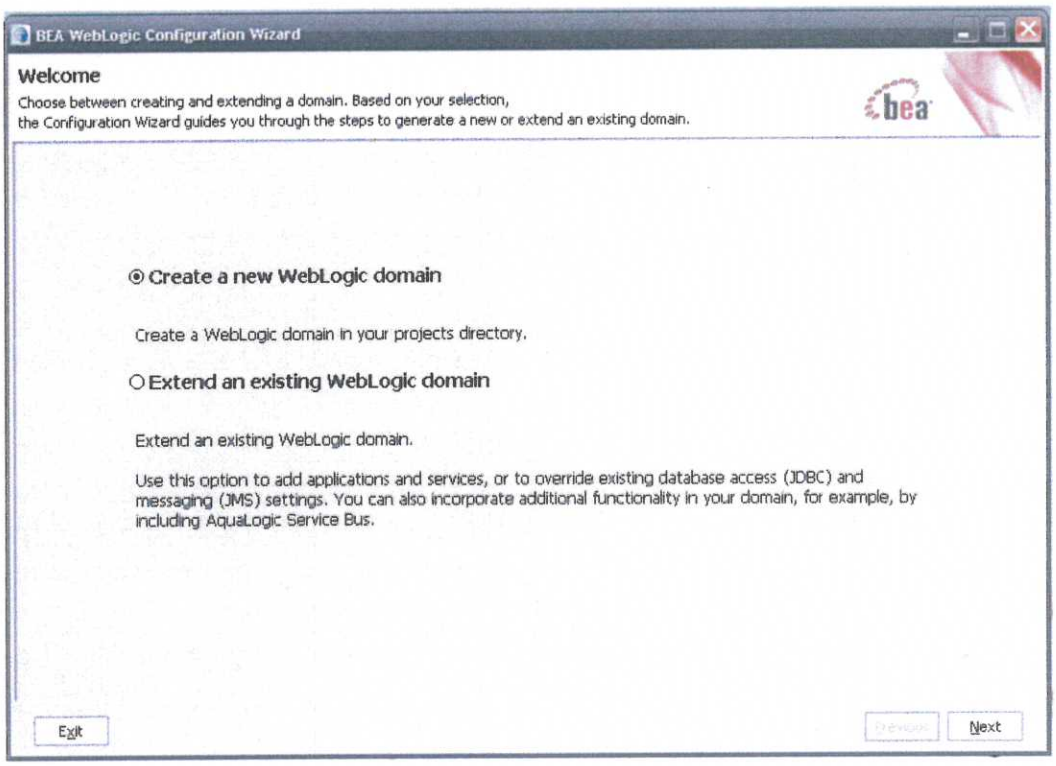




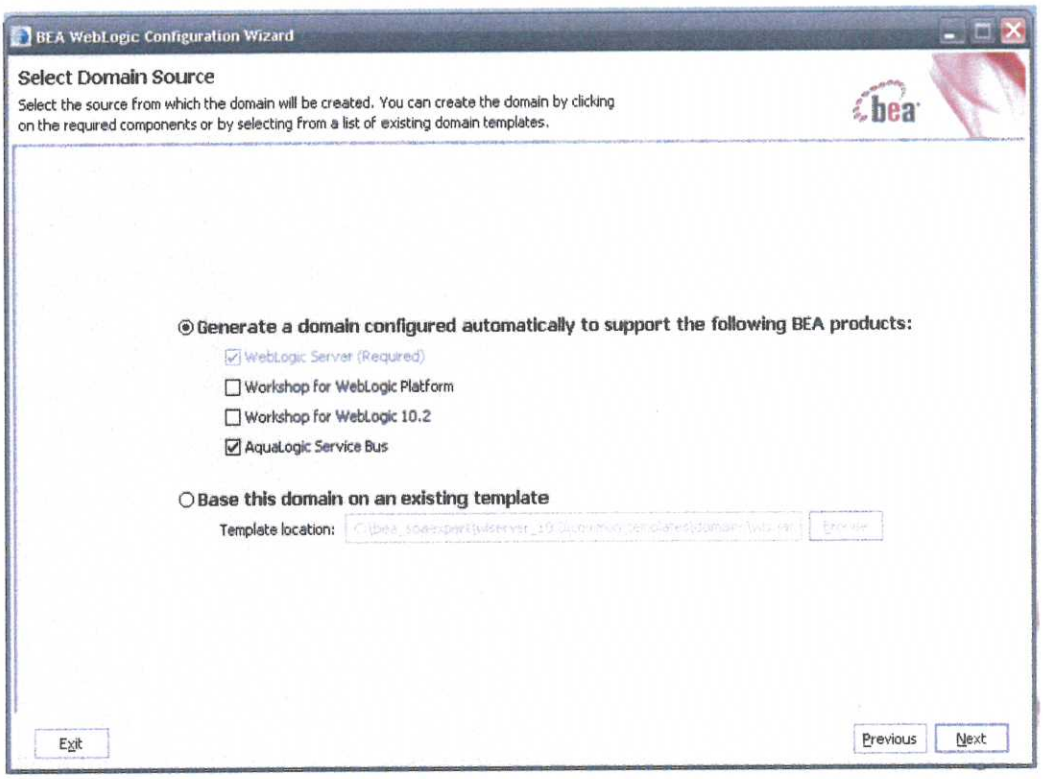
De fato começou a instalação ☺

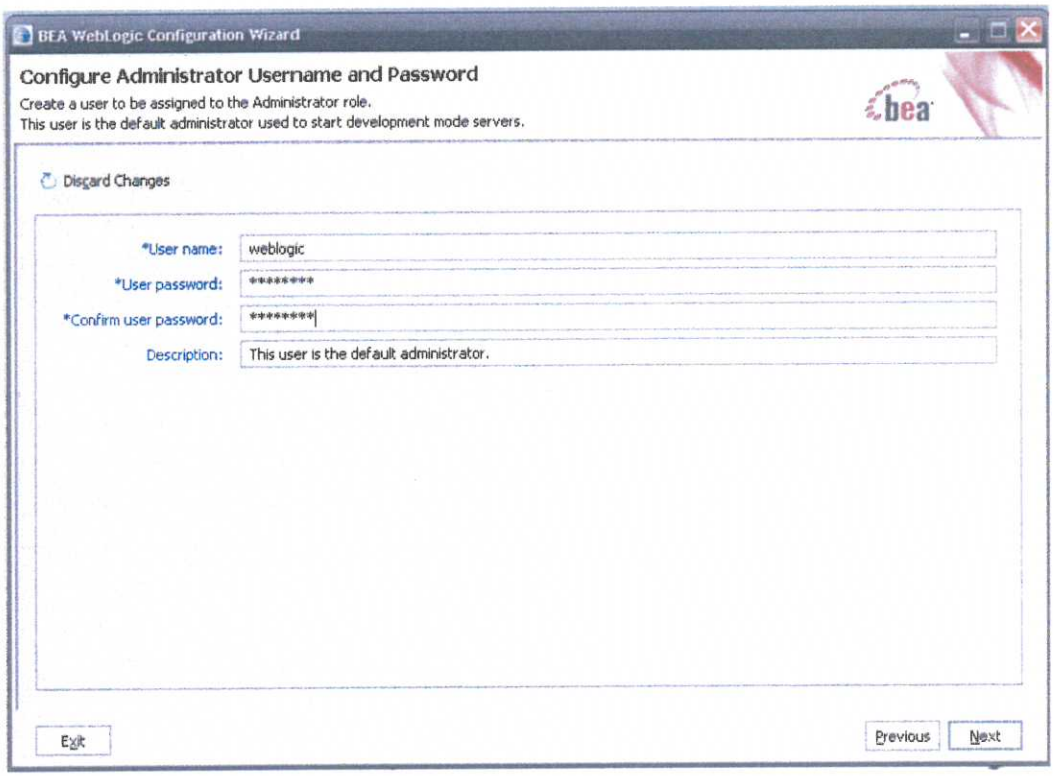
Fim da instalação, parabéns :-D !!

3.1 Criando um domínio OSB

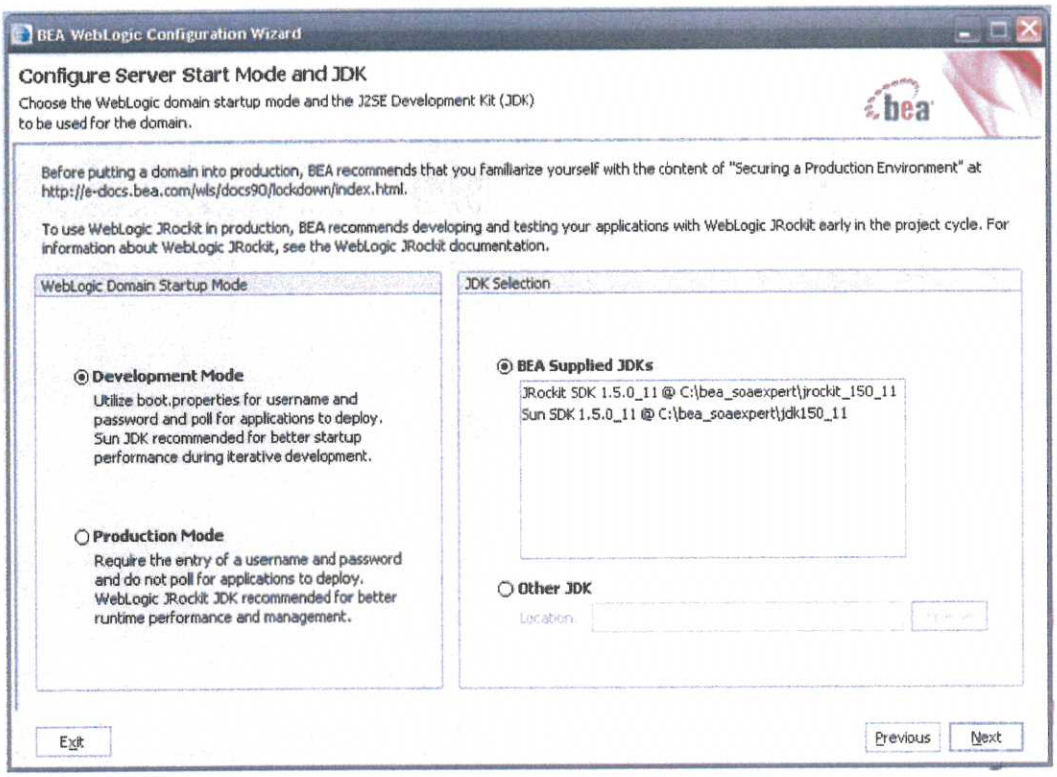


Menu Iniciar – Oracle Products – Tools – Configuration Wizard

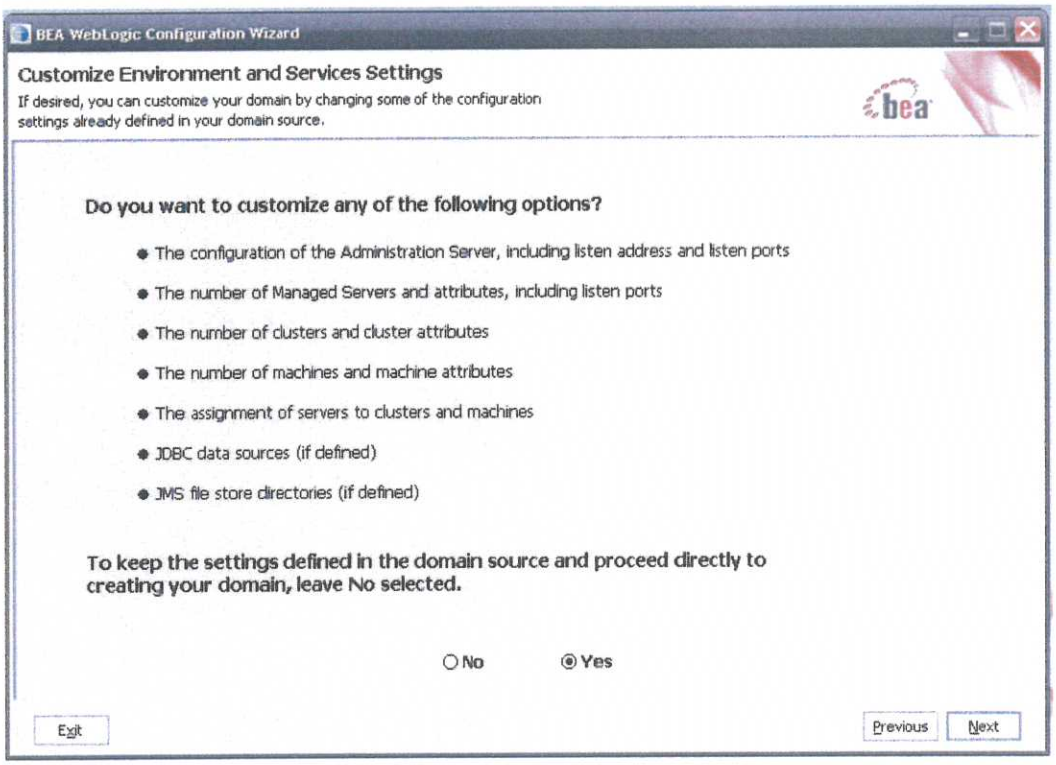




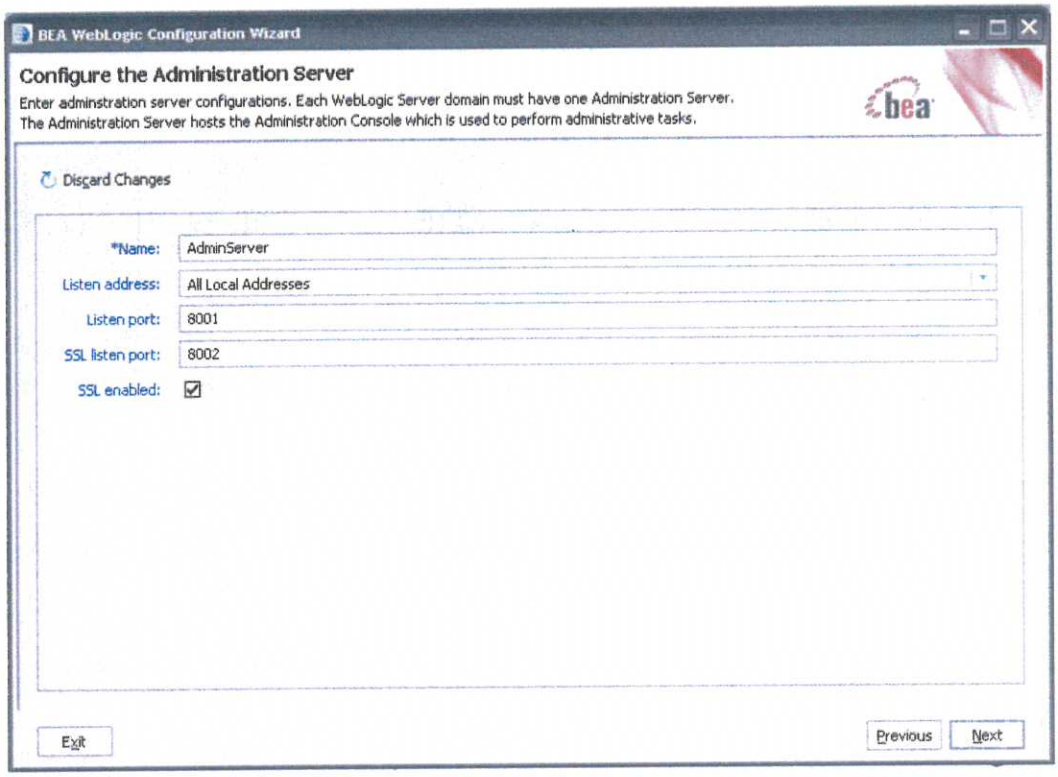
Defina usuário e senha para administração.



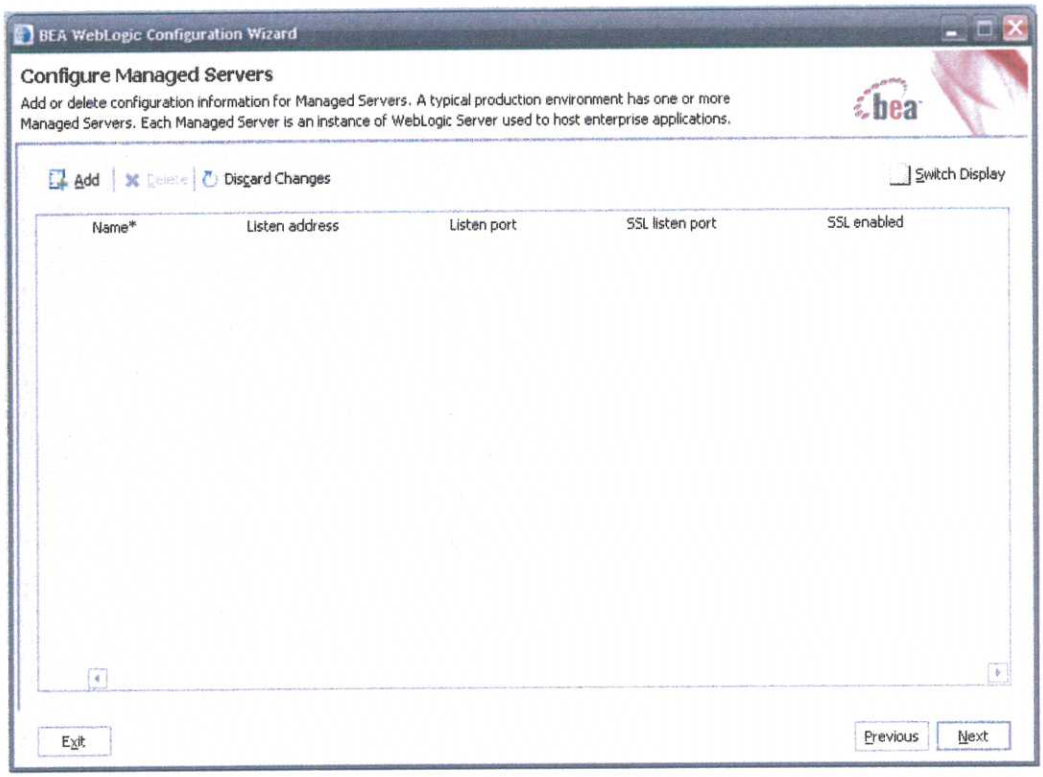
Escolha o modo de configuração, Development Mode com a JDK da Sun, pois os erros são comuns à maior parte dos desenvolvedores.



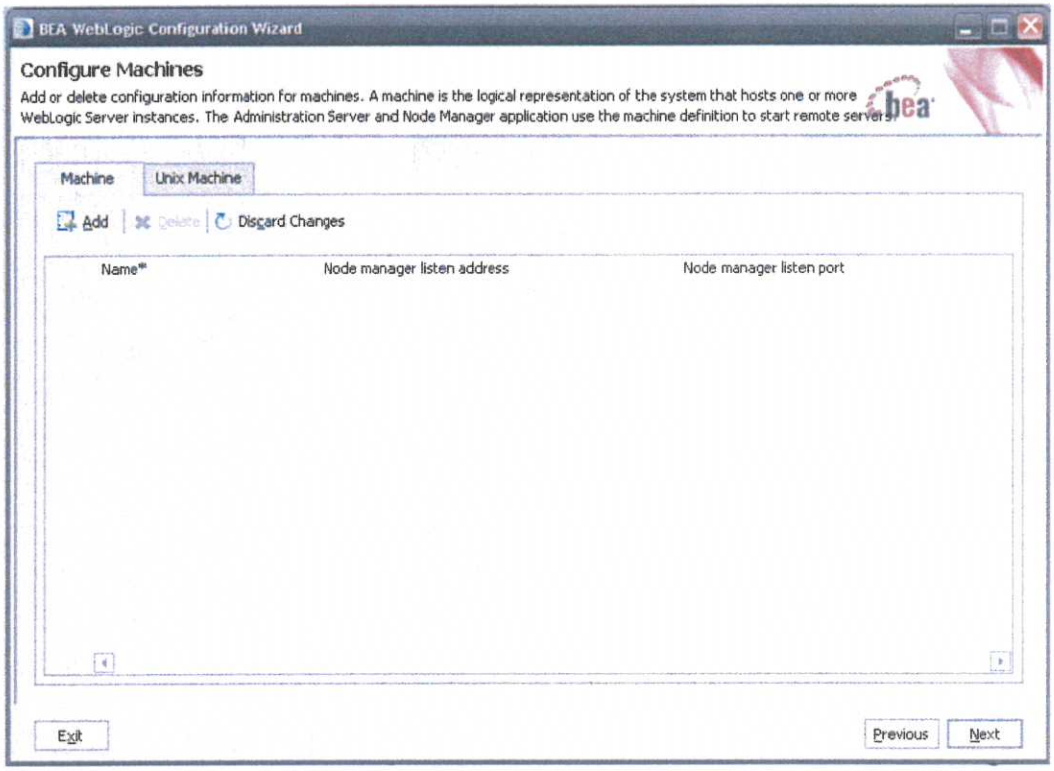
Selecione Yes para customizar as opções



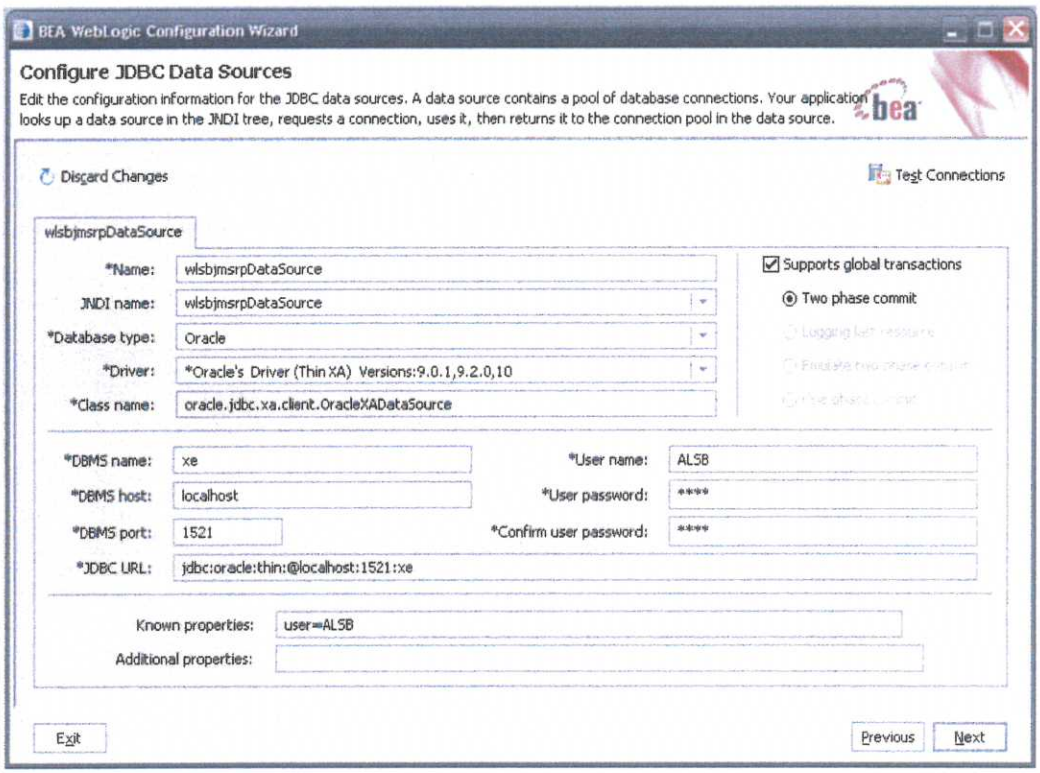
Procure respeitar o nome – AdminServer, pois em futuras manutenções iremos procurar pelo nome do processo na máquina. As portas do ESB podem ser modificadas, mantendo o padrão 8001.



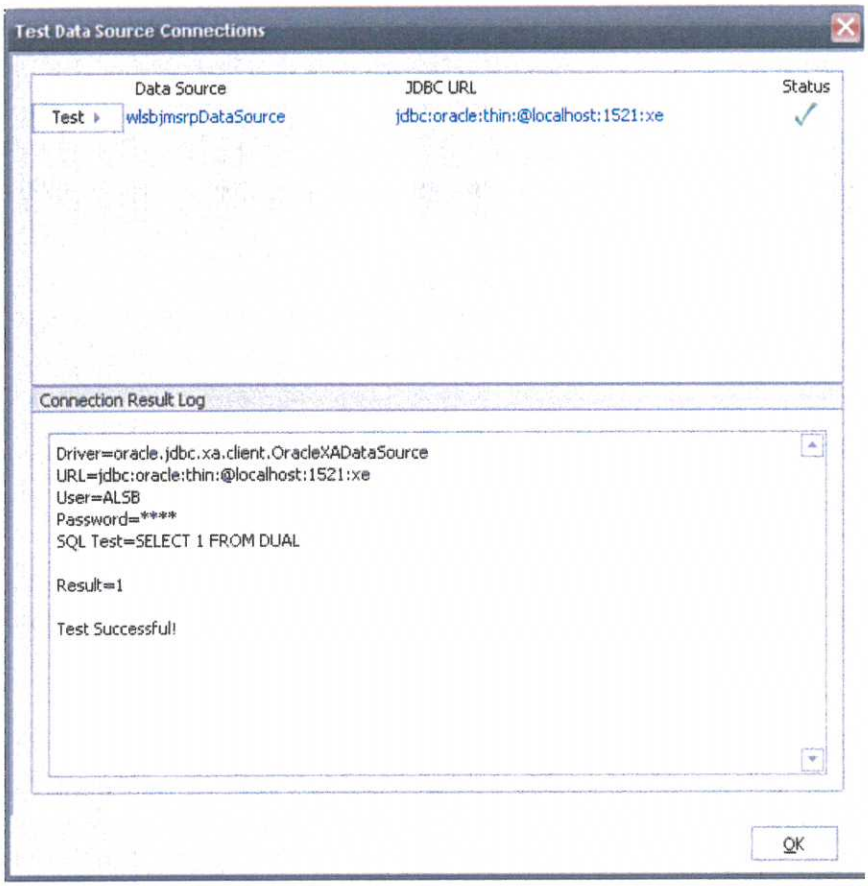
Como estamos lidando com ambiente de desenvolvimento, não adicionamos Managed Server.

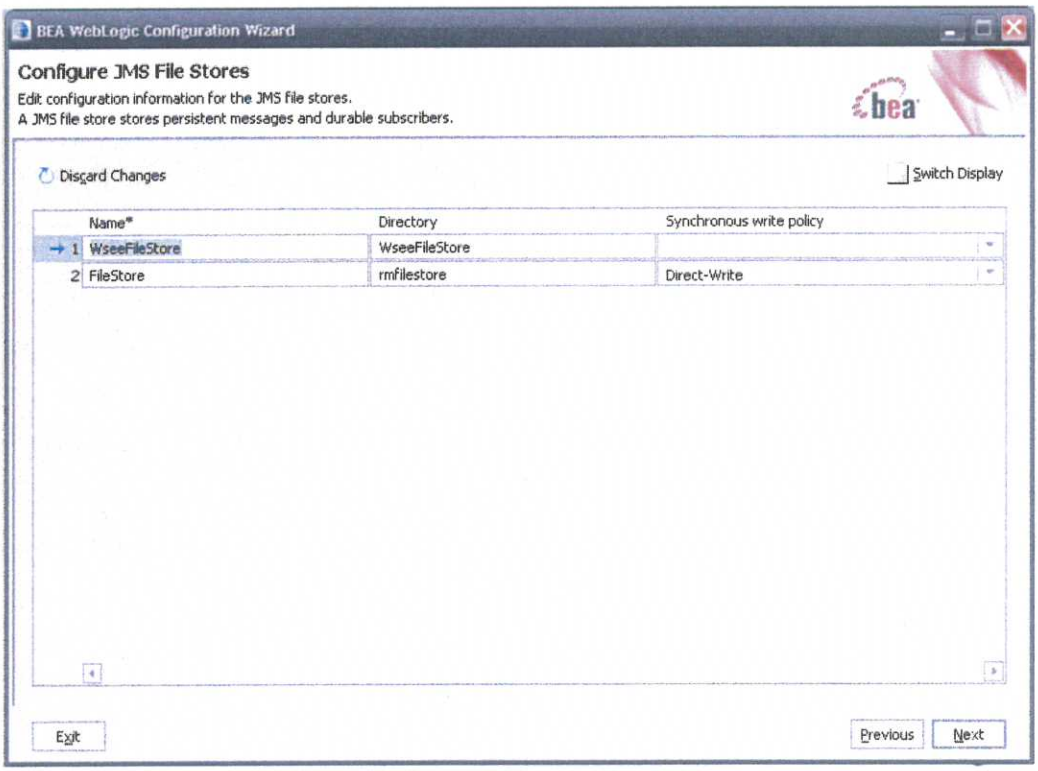


Idem.

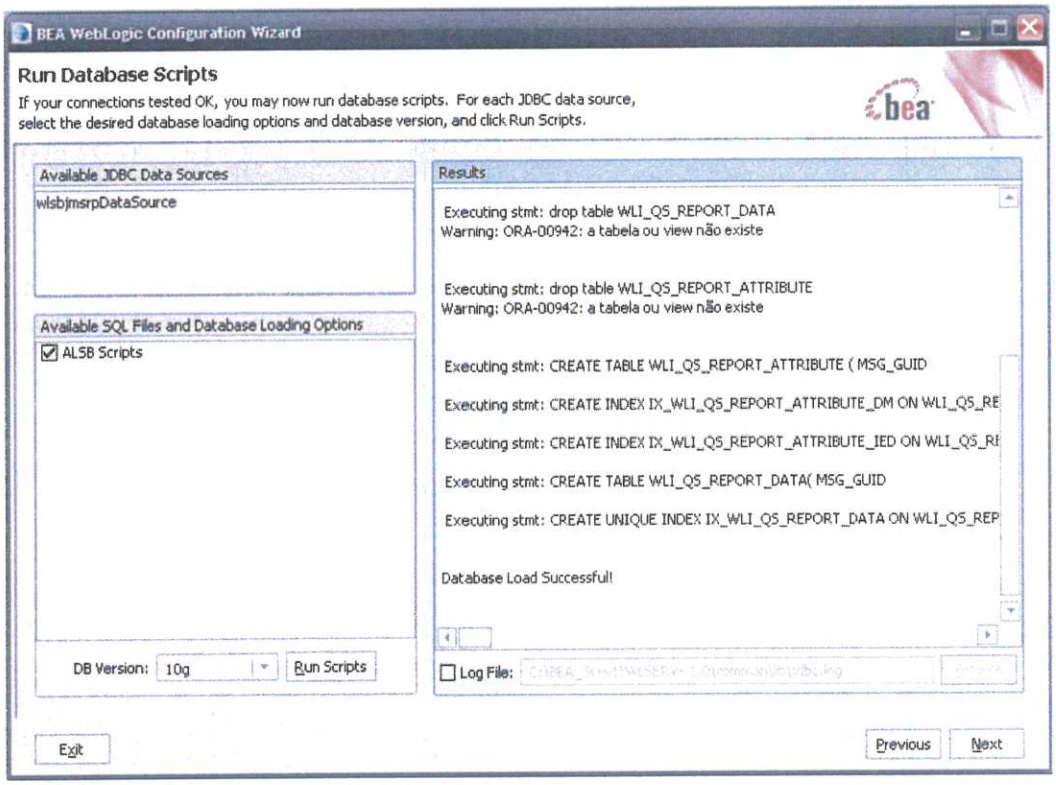


Modifique as propriedades default, para não instalar os metadados no PointBase.

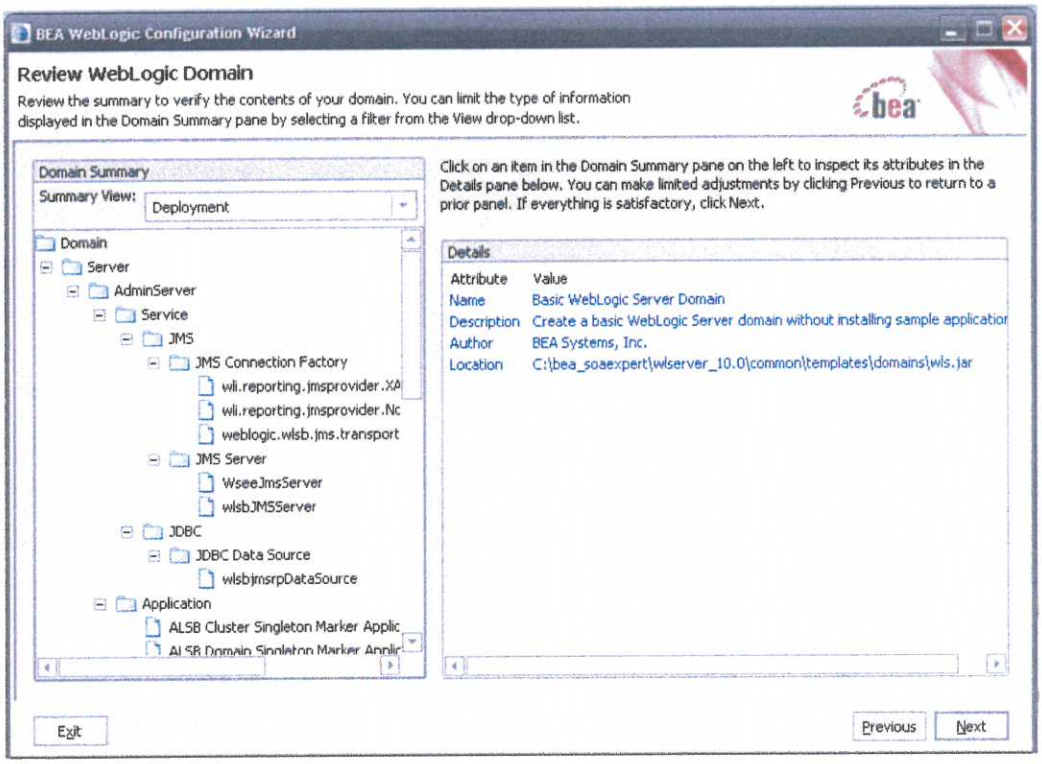




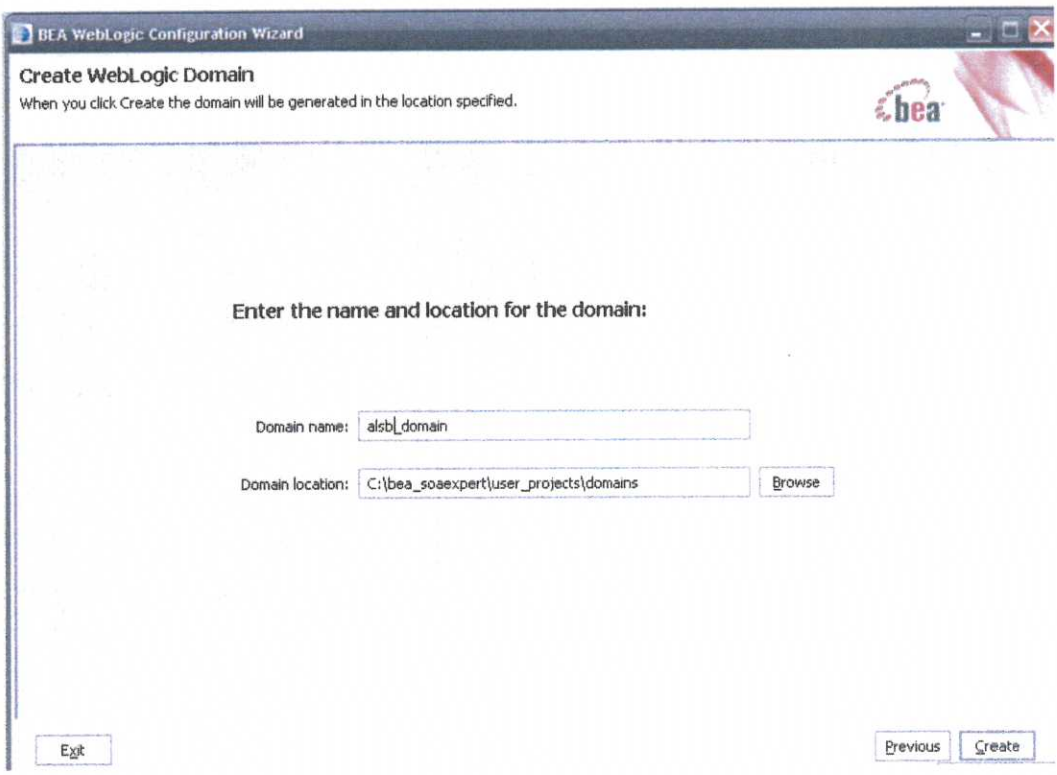
Utilizaremos os stores JMS default da plataforma.



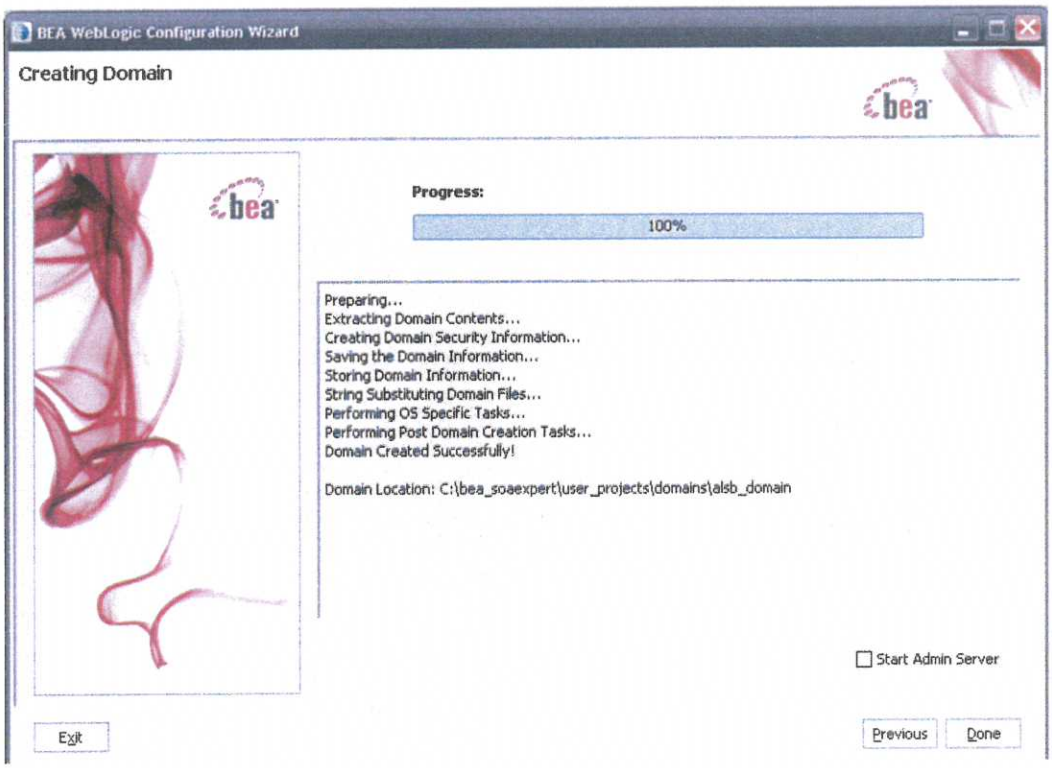
Execute os scripts para criação da estrutura necessária ao produto.



Verifique a configuração do domínio.



Crie um nome para o domínio que acabou de ser gerado.



Parabéns, você acabou de configurar o domínio de desenvolvimento do OSB ☺.

4. Fluxo de Mensagens – Básico

À partir deste capítulo vamos começar a conhecer um pouco mais de produto e entender o ciclo básico de fluxo de mensagens dentro do OSB.

Para começar, criaremos dois cenários:

- o primeiro retorna um catálogo de produtos que, em seu conteúdo, deverá elencar somente produtos que o cliente esteja apto a comprar, baseado em seu limite de crédito;
- o segundo permite ao cliente enviar uma ordem de compra. Você construirá um serviço *ordemEnvio,* que irá verificar se os produtos contidos na mesma não excedem o limite de crédito do cliente. Se o valor for excedido, você deverá retornar uma *fault message*, caso contrário, irá invocar um outro serviço *processamentoPedidos.*

❖ Nota: O serviço *processamentoPedidos* é assíncrono para melhoria de performance.

Cenário 1: O cliente requisita um catálogo de produtos

No primeiro cenário o usuário quer contatar um serviço e buscar de forma customizada o seu catálogo de produtos.

O serviço analisará o limite de crédito do usuário e construirá um catálogo apropriado ao mesmo. Para desenharmos essa operação, vamos necessitar de duas operações.

A primeira – *getLimiteCredito()* vai receber o primeiro e último nome do cliente e retornar seu limite.

A segunda operação - getCatologoProduto(), vai retornar os produtos que de fato o cliente poderá comprar.

Listando o arquivo de contrato – **MessageFlow.wsdl**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<definitions name="MessageFlowServiceDefinitions"
    targetNamespace="http://www.soaexpert.com.br/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xpt="http://www.soaexpert.com.br/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xs:schema attributeFormDefault="unqualified"
            elementFormDefault="qualified" targetNamespace="http://www.soaexpert.com.br/"
            xmlns:pedido="http://www.soaexpert.com.br/pedido/"
            xmlns:xs="http://www.w3.org/2001/XMLSchema">

            <xs:import namespace="http://www.soaexpert.com.br/pedido/" schemaLocation="./pedido.xsd" />


            <xs:element name="ProcessaPedidoSelecionado">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="pedido" type="pedido:Pedido" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element name="LimiteCredito">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="primeiroNome" type="xs:string" />
                        <xs:element name="ultimoNome" type="xs:string" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element name="LimiteCreditoResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="retorno" type="xs:int" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
<message name="CatalogoProduto">
    <part element="xpt:CatalogoProduto" name="CatalogoProduto" />
</message>

<message name="CatalogoProdutoResponse">
    <part element="xpt:CatalogoProdutoResponse" name="CatalogoProdutoResponse" />
</message>

<message name="ProcessaPedido">
    <part element="xpt:ProcessaPedido" name="ProcessaPedido" />
</message>

<portType name="MessageFlowPort">
    <operation name="processaPedidoSelecionado"
        parameterOrder="ProcessaPedidoSelecionado">
        <input message="xpt:ProcessaPedidoSelecionado" />
    </operation>
    <operation name="getLimiteCredito"
        parameterOrder="LimiteCredito">
        <input message="xpt:LimiteCredito" />
        <output message="xpt:LimiteCreditoResponse" />
    </operation>
    <operation name="getCatalogoProduto"
        parameterOrder="CatalogoProduto">
        <input message="xpt:CatalogoProduto" />
        <output message="xpt:CatalogoProdutoResponse" />
    </operation>
    <operation name="processaPedido" parameterOrder="ProcessaPedido">
        <input message="xpt:ProcessaPedido" />
    </operation>
</portType>
<binding name="MessageFlowServiceSoapBinding"
    type="xpt:MessageFlowPort">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="processaPedidoSelecionado">
        <soap:operation soapAction="" style="document" />
        <input>
            <soap:body parts="ProcessaPedidoSelecionado" use="literal" />
        </input>
    </operation>
```

```
        <xs:element name="CatalogoProduto">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="limiteCredito" type="xs:int" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:element name="CatalogoProdutoResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="listaProdutos" type="pedido:ListaProdutos" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:element name="ProcessaPedido">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="pedido" type="pedido:Pedido" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>

    </xs:schema>
</types>

<message name="ProcessaPedidoSelecionado">
    <part element="xpt:ProcessaPedidoSelecionado"
        name="ProcessaPedidoSelecionado" />
</message>

<message name="LimiteCredito">
    <part element="xpt:LimiteCredito" name="LimiteCredito" />
</message>

<message name="LimiteCreditoResponse">
    <part element="xpt:LimiteCreditoResponse" name="LimiteCreditoResponse" />
</message>

<message name="CatalogoProduto">
    <part element="xpt:CatalogoProduto" name="CatalogoProduto" />
</message>
```

```
    <binding name="MessageFlowServiceSoapBinding"
        type="xpt:MessageFlowPort">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="processaPedidoSelecionado">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body parts="ProcessaPedidoSelecionado" use="literal" />
            </input>
        </operation>
        <operation name="getLimiteCredito">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body parts="LimiteCredito" use="literal" />
            </input>
            <output>
                <soap:body parts="LimiteCreditoResponse" use="literal" />
            </output>
        </operation>
        <operation name="getCatalogoProduto">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body parts="CatalogoProduto" use="literal" />
            </input>
            <output>
                <soap:body parts="CatalogoProdutoResponse" use="literal" />
            </output>
        </operation>
        <operation name="processaPedido">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body parts="ProcessaPedido" use="literal" />
            </input>
        </operation>
    </binding>
    <service name="MessageFlowService">
        <port binding="xpt:MessageFlowServiceSoapBinding"
            name="MessageFlowSoapPort">
            <soap:address
                location="http://localhost:7001/SOAExpert/MessageFlowService" />
        </port>
    </service>
</definitions>
```

1- Crie no seu Workshop o projeto SOAExpert.

2- Na pasta do SRC do projeto, crie o package chamado br.com.soaexpert.services .

3- Crie uma pasta com o nome de resources. Esta pasta abrigará o contrato a ser criado – **MessageFlow.wsdl .**

4- Com o botão direito em cima do contrato, selecione WebServices – Generate Web Service. Procure o package onde a classe de implementação será gerada br.com.soaexpert.services.

Como você vai verificar, a classe de implementação não possui código, portanto você deverá implementar por si. Aqui consta uma implementação referência simples, para que o cenário funcione como esperado:

```
package br.com.soaexpert.services;

import java.util.ArrayList;

/**
 * MessageFlowPortImpl class implements web service endpoint interface
 * MessageFlowPort
 */

@WebService(serviceName = "MessageFlowService", targetNamespace = "http://www.soaexpert.com.br/", endpointInterface =
@WLHttpTransport(contextPath = "SOAExpert", serviceUri = "MessageFlowService", portName = "MessageFlowSoapPort")
public class MessageFlowPortImpl implements MessageFlowPort {


    public MessageFlowPortImpl() {


    }

    public void processaPedidoSelecionado(
            ProcessaPedidoSelecionado processaPedidoSelecionado) {
        final Pedido pedido = processaPedidoSelecionado.getPedido();
        System.out
                .println("Recebido, pedido selecionado pelo cliente: "
                        + pedido.getPrimeiroNome() + ""
                        + pedido.getUltimoNome() + "do produto"
                        + pedido.getLineItem()[0].getProduto().getNome());
        return;
    }



  public LimiteCreditoResponse getLimiteCredito(LimiteCredito limiteCredito) {

      final String primeiroNome = limiteCredito.getPrimeiroNome();
      final String ultimoNome = limiteCredito.getUltimoNome();
      LimiteCreditoResponse creditoResposta = new LimiteCreditoResponse();

      int credito;

      if (ultimoNome.compareToIgnoreCase("X") > 0) {
          credito = 200;
      } else if (ultimoNome.compareToIgnoreCase("S") > 0) {
          credito = 600;
      } else {
          credito = 750;
      }

      creditoResposta.setRetorno(credito);

      return creditoResposta;
  }
```

```
    public CatalogoProdutoResponse getCatalogoProduto(
            CatalogoProduto catalogoProduto) {
        //Simulando o banco de dados
        Produto[] bdprodutos = {new Produto(), new Produto(), new Produto(), new Produto()};
        bdprodutos[0].setCreditoNecessario(610);
        bdprodutos[1].setNome("IPod");
        bdprodutos[1].setCreditoNecessario(500);
        bdprodutos[2].setNome("Automovel");
        bdprodutos[2].setCreditoNecessario(710);
        bdprodutos[3].setNome("PS3");
        bdprodutos[3].setCreditoNecessario(440);

        ListaProdutos listaProdutos = new ListaProdutos();
        List<Produto> produtosCliente = new ArrayList<Produto>();
        final int credito = catalogoProduto.getLimiteCredito();
        CatalogoProdutoResponse catalogoResposta = new CatalogoProdutoResponse();

        for (int i = 0; i < bdprodutos.length; i++) {
            if (bdprodutos[i].getCreditoNecessario() <= credito) {
                // Adiciona a lista de produtos que o cliente pode comprar.
                produtosCliente.add(bdprodutos[i]);
            }
        }
        final Produto[] produtosClienteArray = produtosCliente
                .toArray(new Produto[produtosCliente.size()]);
        listaProdutos.setProduto(produtosClienteArray);

        // setando resposta
        catalogoResposta.setListaProdutos(listaProdutos);
        return catalogoResposta;
    }

    public void processaPedido(ProcessaPedido processaPedido) {
        final Pedido pedido = processaPedido.getPedido();
        System.out.println("Recebido um pedido de:" + pedido.getPrimeiroNome()
                + " " + pedido.getUltimoNome() + "para o produto "
                + pedido.getLineItem()[0].getProduto());

        return;
    }

}// The end
```

Compile e faça o *deploy* para o servidor básico – WLS.

Neste ponto estaremos atacando o cerne do problema.

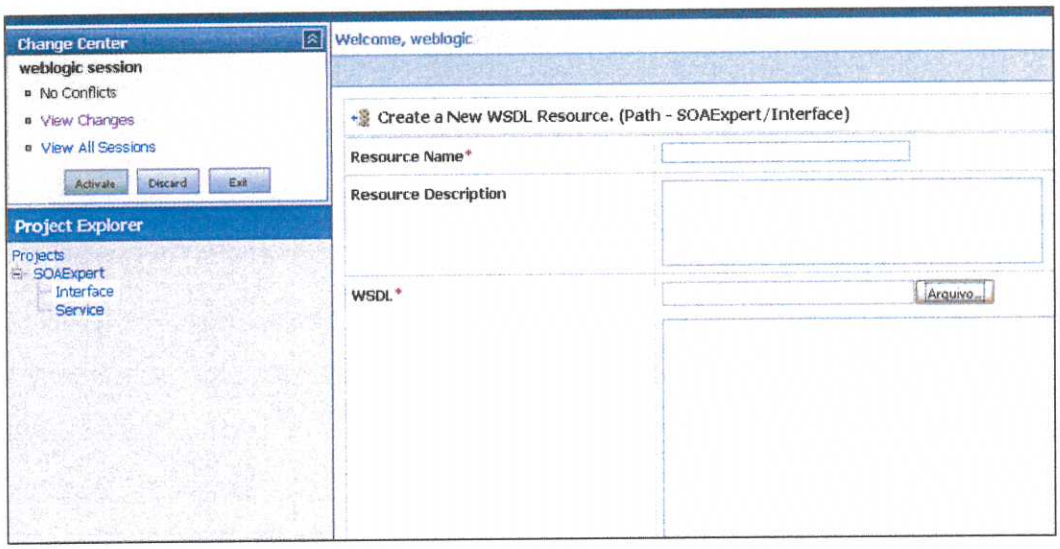
Para resolver a necessidade de implementação você deverá criar um **Proxy Service** no OSB, que faz uso dos dois *business services* que foram criados. Esse novo serviço agrega as duas operações de negócio, ocultando do usuário a complexidade de implementação (Façade Pattern). Na linguagem SOA ele é conhecido como **Service Mediation**, usando um serviço para esconder o relacionamento de baixo nível entre os demais.

Melhores Práticas : Esse exercício demonstra uma boa prática, separando as estruturas de dados do contrato WSDL.

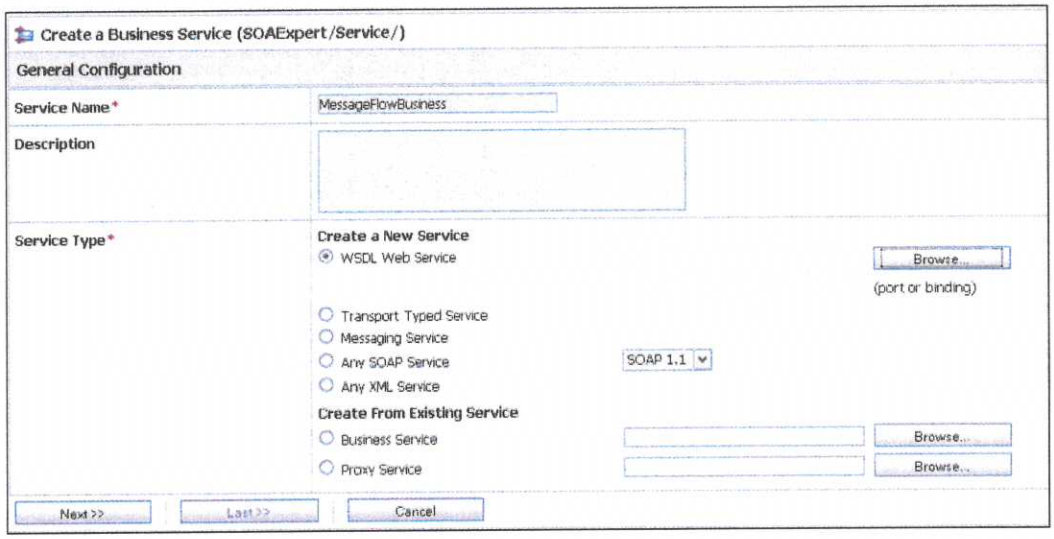
OSB

No OSB, em seu projeto SOAExpert, crie uma subpasta WSDL dentro de Interface.

Clique em *Create* e crie um recurso WSDL, apontando o arquivo para o endereço do seu WSDL criado no Workshop.



Navegando novamente na raiz do projeto SOAExpert, crie um *business service* chamado **MessageFlow.**



Note que o *endpoint* URI para o serviço inclui um **IP address** específico. Você pode, nesta aba, incluir ou remover em tempo de execução endereços.

Clique agora Last Button e aceite todos os valores padrão para seu business service.

Clique em Save e depois em **Activate**, *no Change Center*.

OBS: Não se esqueça de incluir um comentário sobre a modificação que está sendo realizada !!

Agora vem a parte boa do exercício ☺.

Iremos agora criar um **Proxy Service,** que vai pegar um nome de cliente e retornar uma lista de produtos que ele realmente possa comprar.

Será necessário criar um contrato específico para essas operações

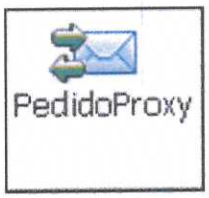
Contrato **MessageFlowProxyContract**:

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions name="MessageFlowServiceDefinitions"
    targetNamespace="http://www.soaexpert.com.br/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xpt="http://www.soaexpert.com.br/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xs:schema attributeFormDefault="unqualified"
            elementFormDefault="qualified" targetNamespace="http://www.soaexpert.com.br/"
            xmlns:pedido="http://www.soaexpert.com.br/pedido/">

            <xs:import namespace="http://www.soaexpert.com.br/pedido/"
                schemaLocation="./pedido.xsd" />

            <!-- Reutilizando os tipos que foram importados-->
            <xs:element name="Produto" type="pedido:Produto" />
            <xs:element name="Item" type="pedido:Item" />
            <xs:element name="Pedido" type="pedido:Pedido" />
            <xs:element name="ArrayOfProduct" type="pedido:ListaProdutos" />

            <xs:element name="NomeCompletoCliente">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="primeiroNome" type="xs:string" />
                        <xs:element name="ultimoNome" type="xs:string" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </types>
    <message name="Produto">
        <part name="nomeCompletoCliente" element="xpt:NomeCompletoCliente" />
    </message>
    <message name="ProdutoResponse">
        <part name="listaProdutos" element="xpt:ArrayOfProduct" />
    </message>
    <message name="OrdemPedido">
        <part name="pedido" element="xpt:Pedido" />
    </message>
```

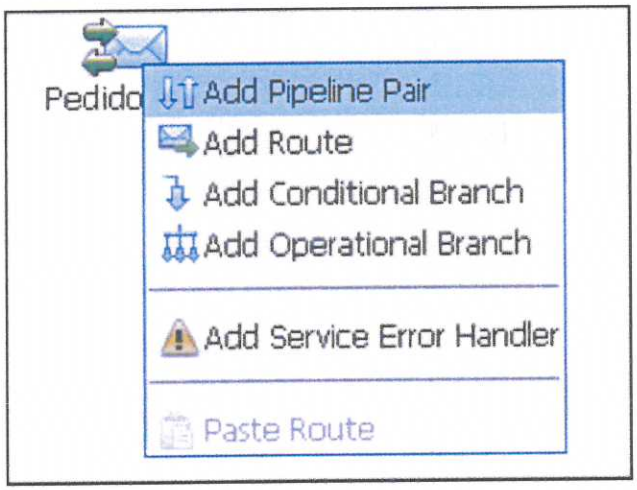
```
    <portType name="MessageFlowProxyPort">
        <operation name="getProdutos" parameterOrder="NomeCompletoCliente">
            <input message="xpt:Produto" />
            <output message="xpt:ProdutoResponse" />
        </operation>
        <operation name="ordemPedido" parameterOrder="OrdemPedido">
            <input message="xpt:OrdemPedido" />
        </operation>
    </portType>
    <binding name="MessageFlowProxyServiceSoapBinding"
        type="xpt:MessageFlowProxyPort">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="getProdutos">
            <xs:annotation>
                <xs:documentation>
                    Essa operação  busca uma lista de produtos de acordo com
                    o limite de credito do cliente.
                </xs:documentation>
            </xs:annotation>
            <soap:operation soapAction="getProdutos" style="document" />
            <input>
                <soap:body parts="nomeCompletoCliente" use="literal" />
            </input>
            <output>
                <soap:body parts="listaProdutos" use="literal" />
            </output>
        </operation>
        <operation name="ordemPedido">
            <soap:operation soapAction="ordemPedido" style="document" />
            <input>
                <soap:body parts="pedido" use="literal" />
            </input>
        </operation>
    </binding>
    <service name="MessageFlowProxyService">
        <port binding="xpt:MessageFlowProxyServiceSoapBinding"
            name="MessageFlowProxySoapPort">
            <soap:address
                location="http://localhost:7001/esb/MessageFlowService" />
        </port>
    </service>
</definitions>
```

Com o **proxy** pronto, clique no ícone que lembra um semáforo, para editar seu fluxo.

No seu estado atual, o proxy contém somente um Start node.



Clique com o botão direito sob o Start node e adicione um Pipeline Pair para esse fluxo de mensagem.

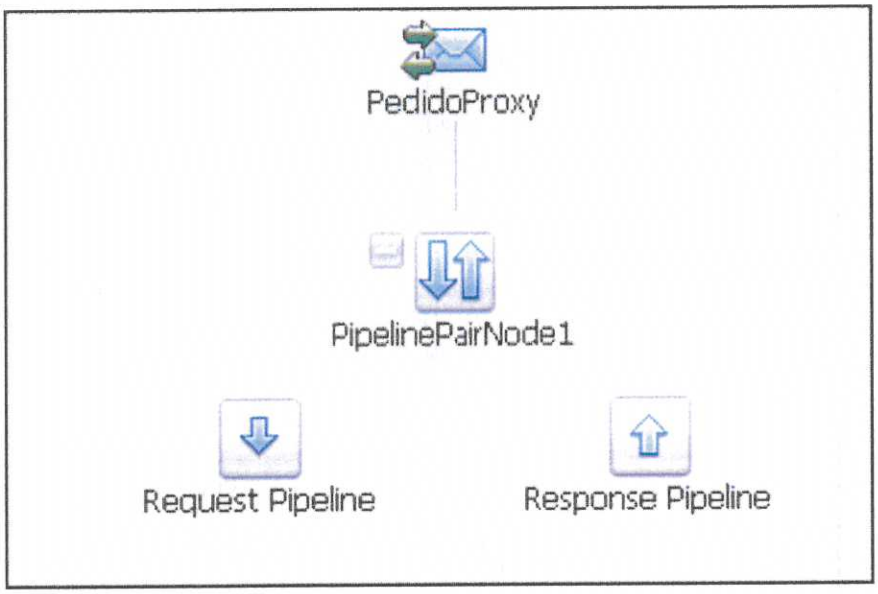


A razão por escolhermos um Pipeline é que o processo será quebrado em um série de estágios.

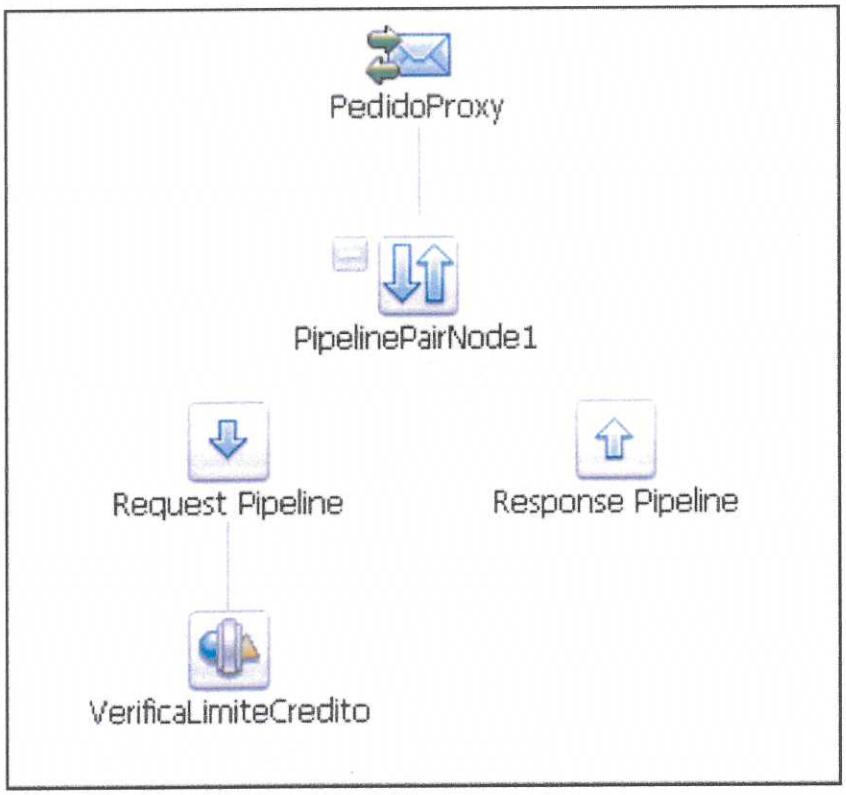
Vamos utilizar um *stage* para chamar a operação **getLimiteCredito()**e um segundo *stage* para chamar o **getCatalogoProduto().**

Você deve estar se perguntando por quê não realizar tudo numa única rota. Isso se deve ao estilo de design, a fim de garatir manutenibilidade.

Este é análogo à codificação: ao invés de criar um único método com a resolução de todos os problemas, você modulariza em métodos com escopo definido.



Adicione o primeiro *stage* para o request pipeline. Com o botão esquerdo no ícone de *request pipeline* selecione Add Stage.

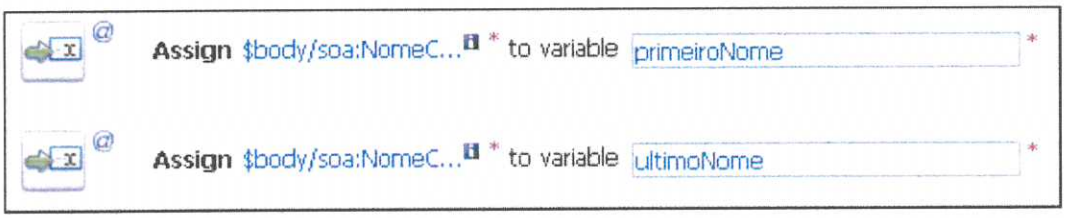


OBS: Procure colocar nomes significativos nos seus estágios, a fim de melhorar a visibilidade do processo da sua operação.

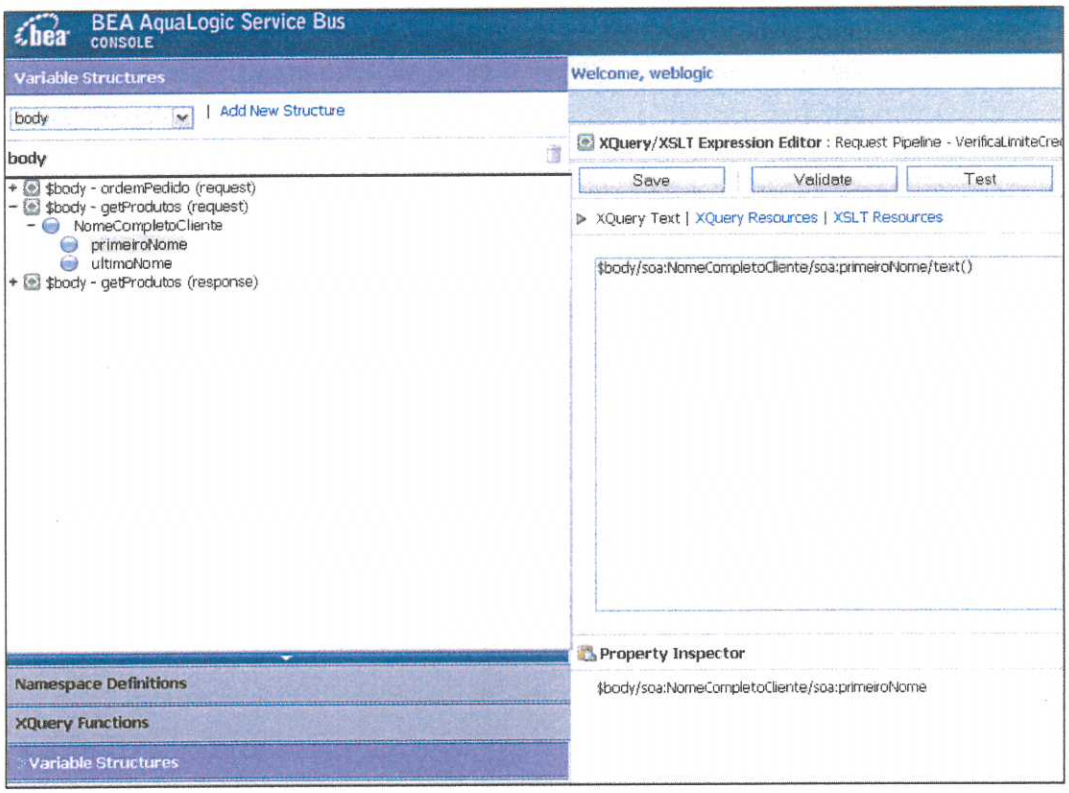
No próximo passo, clique em **edit stage**.

O *request pipeline* recebe como parâmetro de entrada as variáveis primeiroNome e últimoNome.

Agora quando passamos pelo proxy, armazenaremos esses dados a fim de utilizá-los num outro serviço – *Service Callout.*



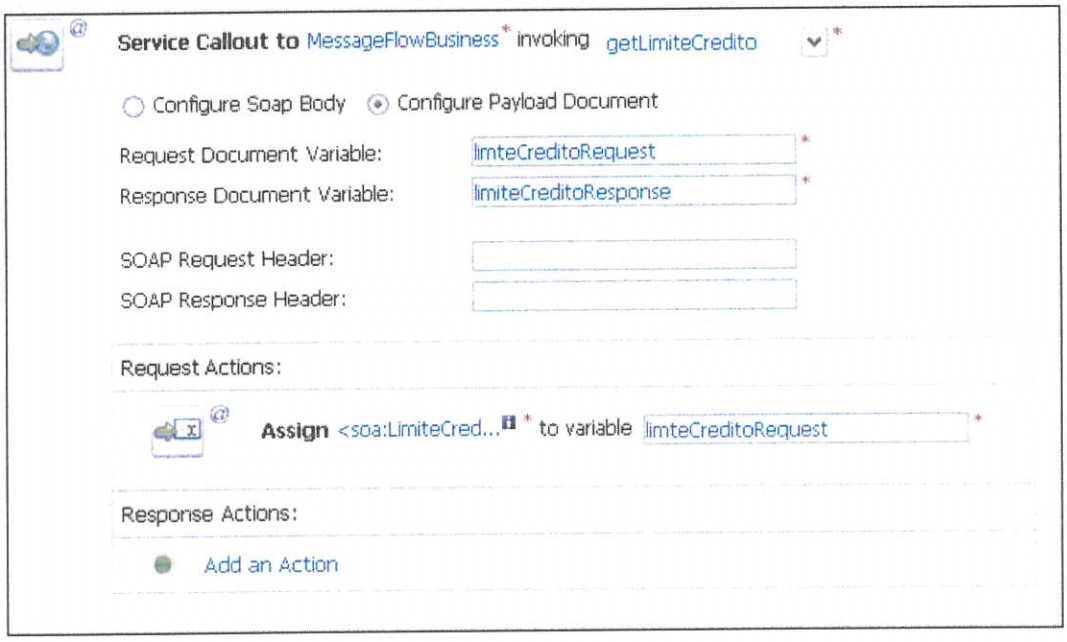
Ao clicar nas variáveis de estrutura, note que você possui um guia para auxiliá-lo a escrever a expressão XPath sem erros.



Ao final da estrutura de variável do painel **Variable Structures**, adicione um **/text()**, expressão XPath para enviar o valor da estrutura e não o nó.

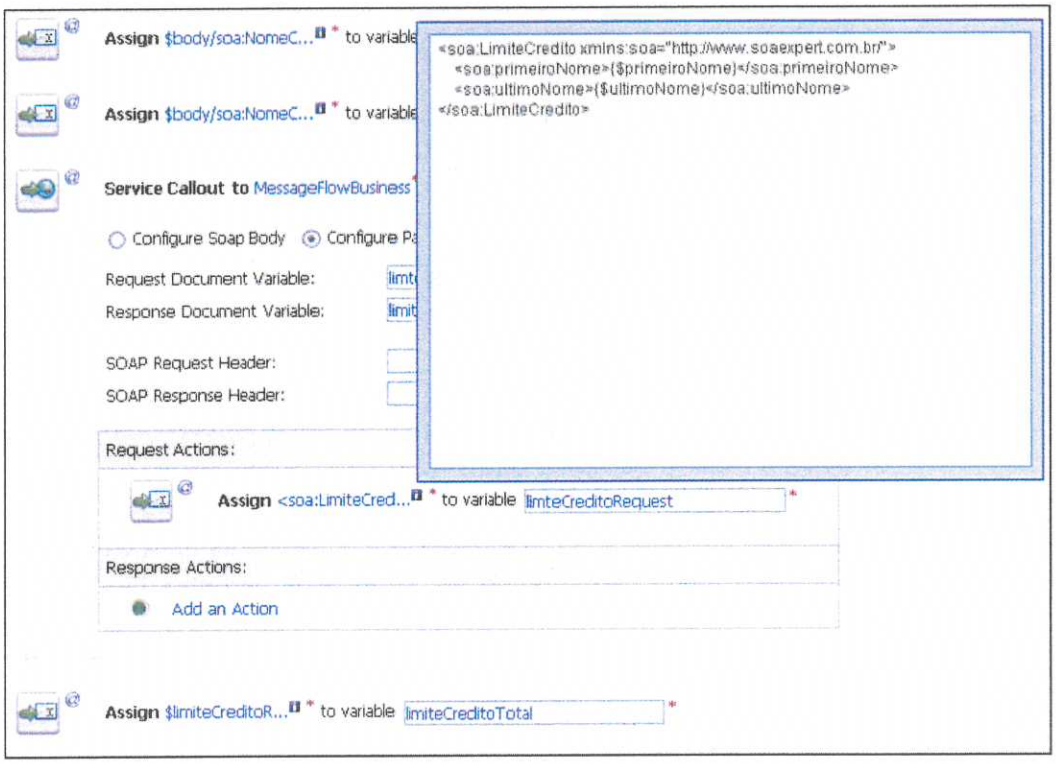
Lembre-se de validar a expressão, para evitar erros e confusões de sintaxe.

Usando essa técnica você pode economizar muito tempo de testes para saber se o payload do seu serviço está funcionando como deveria.

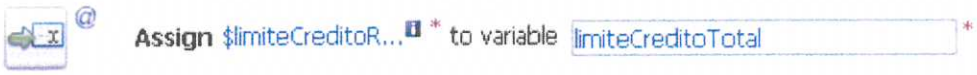


Em cima do ícone do **Assign** adicione um **Service Callout** para o MessageFlowBusiness, invocando a operação getLimiteCredito.

No Request Actions, faremos um payload com a entrada SOAP esperada pelo serviço.



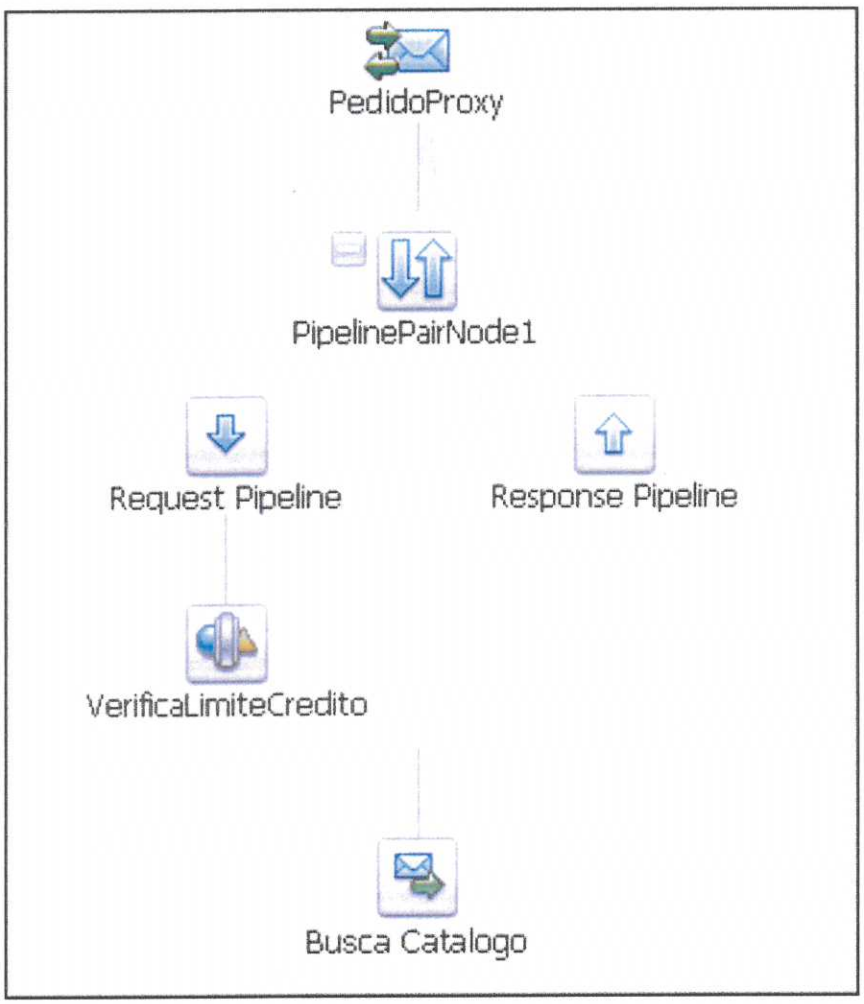
Por fim, armazenaremos a variável de retorno do serviço, numa variável local **$limiteCredito.**



OBS: Lembre-se de utilizar a expressão XPath para buscar o conteúdo.

Com o Stage concluído, vamos rotear nosso fluxo para o fim, concluindo a resposta ao serviço.

Clique com o botão direito em cima do *request pipeline* para adicionar uma *Route.*



A busca de catálogo é nossa rota final, que deverá mostrar a lista de pedidos de acordo com o cliente e seu poder de consumo.

Neste ponto, o serviço final – getCatalogoProduto - não está apto a receber as variáveis primeiroNome e ultimoNome como parâmetros. Logo, uma transformação de tipos será necessária para adequar aos parâmetros de entrada que o serviço espera.

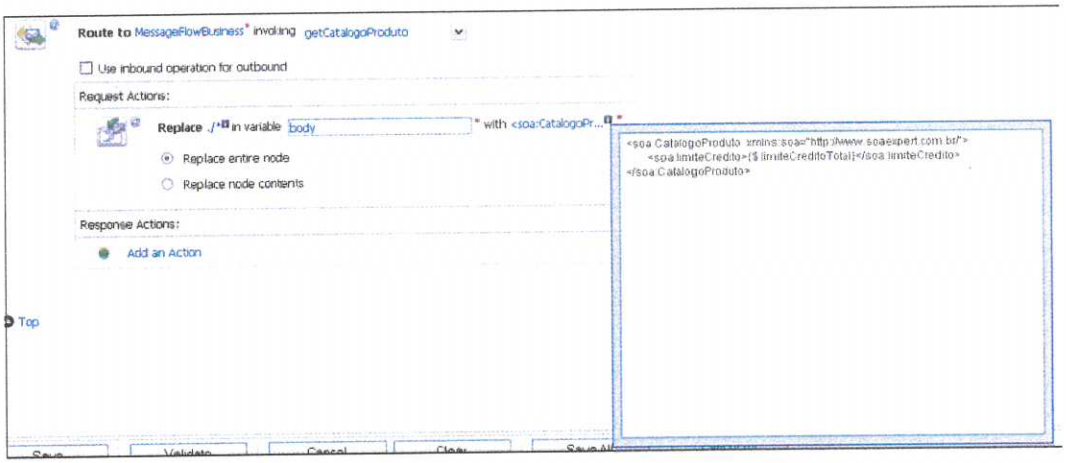
```
<soa:NomeCompletoCliente xmlns:soa="http://www.soaexpert.com.br/">
   <soa:primeiroNome>string</soa:primeiroNome>
   <soa:ultimoNome>string</soa:ultimoNome>
</soa:NomeCompletoCliente>
```

para:

```
<soa:CatalogoProduto xmlns:soa="http://www.soaexpert.com.br/">
   <soa:limiteCredito>3</soa:limiteCredito>
</soa:CatalogoProduto>
```

Escolha o serviço MessageFlowBusiness, operação getCatalogoProduto.

No Request Action adicione um replace juntamente com a expressão XPath ./* , e em variável coloque **body**.



Adicione o SOAP Body compatível.

Por fim, pelo retorno de *getCatalogoProduto* trazer um formato que o Proxy espera, não será necessária nenhuma outra transformação de tipos.

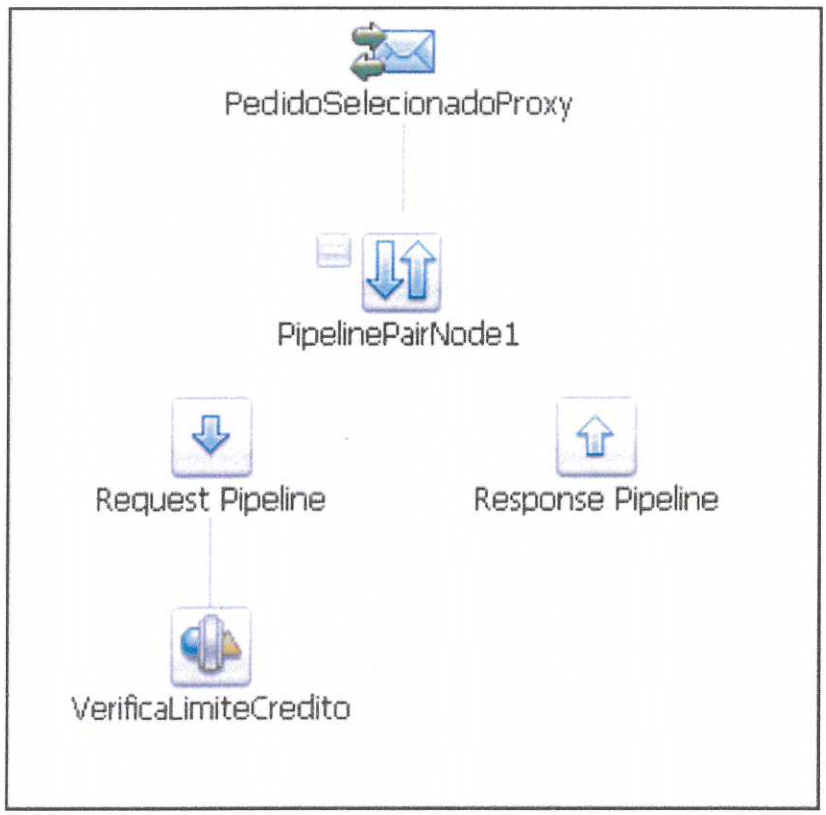
Cenário 2: O cliente compra um produto

Neste cenário o cliente submete um pedido para seu **PedidoProxyservice.** O proxy invoca o getLimteCredito e checa o limite do cliente para consumo e, se o cliente possui um limite elevado, ele é roteado para o serviço de pedido, ordemPedido. Essa decisão é realizada por um **Branch Node.**

Toda infra-estrutura será a mesma utilizada no exercício anterior.

Você irá criar esse novo Proxy baseado no mesmo WSDL do **Message Flow Proxy.**

Edite o fluxo de mensagem do serviço ordemPedido e adicione um Pipeline Pair no Start Node.

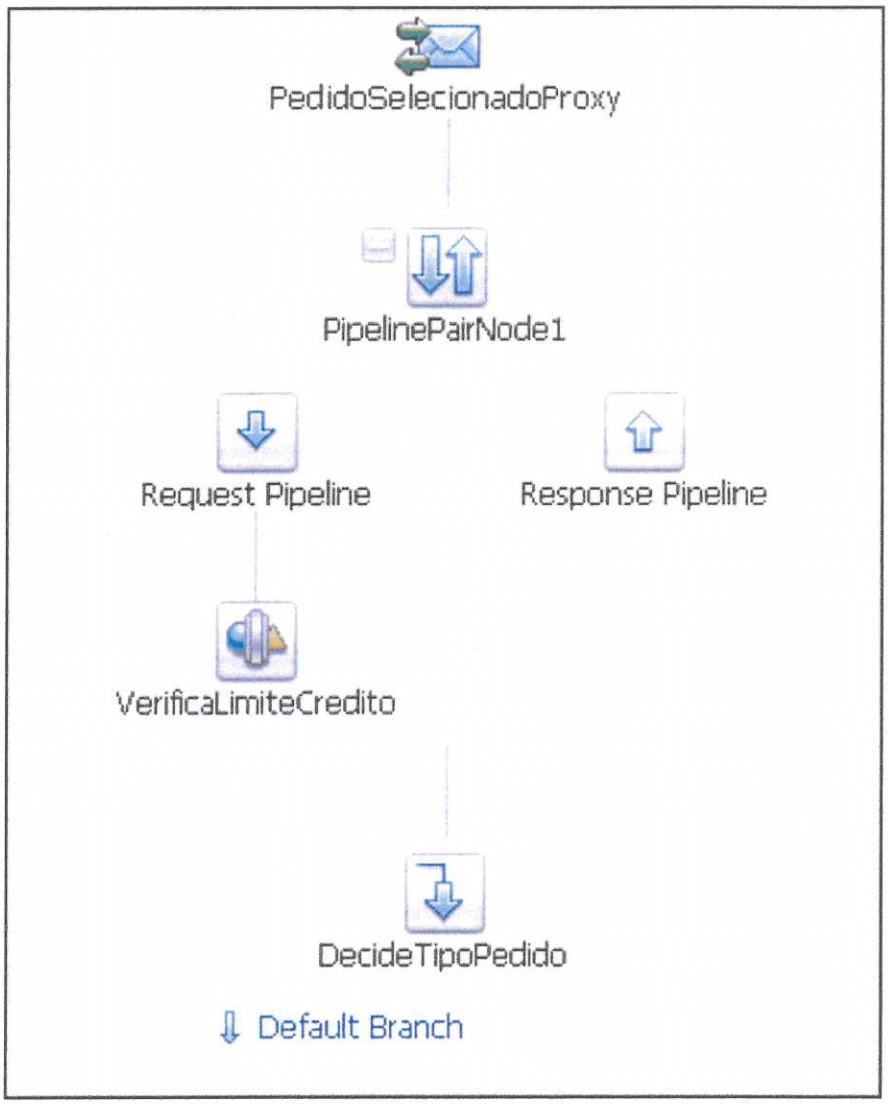


Assim como no cenário anterior, você possui dois stages e também precisará buscar o limiteCredito do seu cliente.

Como tempo é dinheiro, vamos ao proxy do cenário anterior e no stage VerificaLimiteCredito, clique com o botão direito para copiá-lo.

Sim o OSB permite que você faça um copy paste de objetos como stage ☺.

Agora adicione um conditional branch node depois do Pipeline Pair node. O nome desse node é : **DecideTipoPedido.**

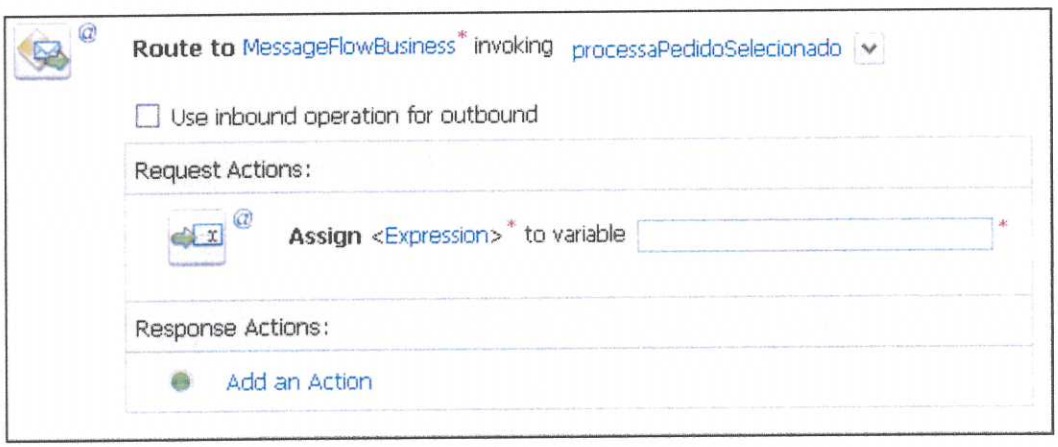


O Branch será quebrado em duas fases quando editarmos de maneira simples.

Clientes com limiteCredito maior que 700 serão classificados como "selecionado", que será processado pelo pedidoSelecionado.

Após o branch definido, adicione a rota à ele.

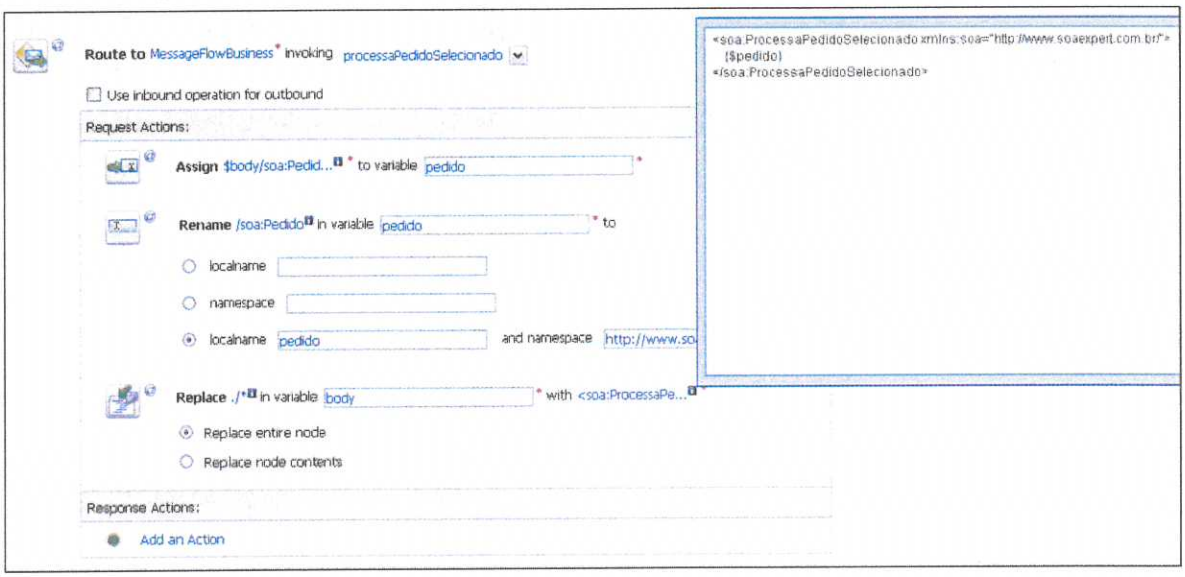
Deve-se notar que o start node é ligeiramente diferente.

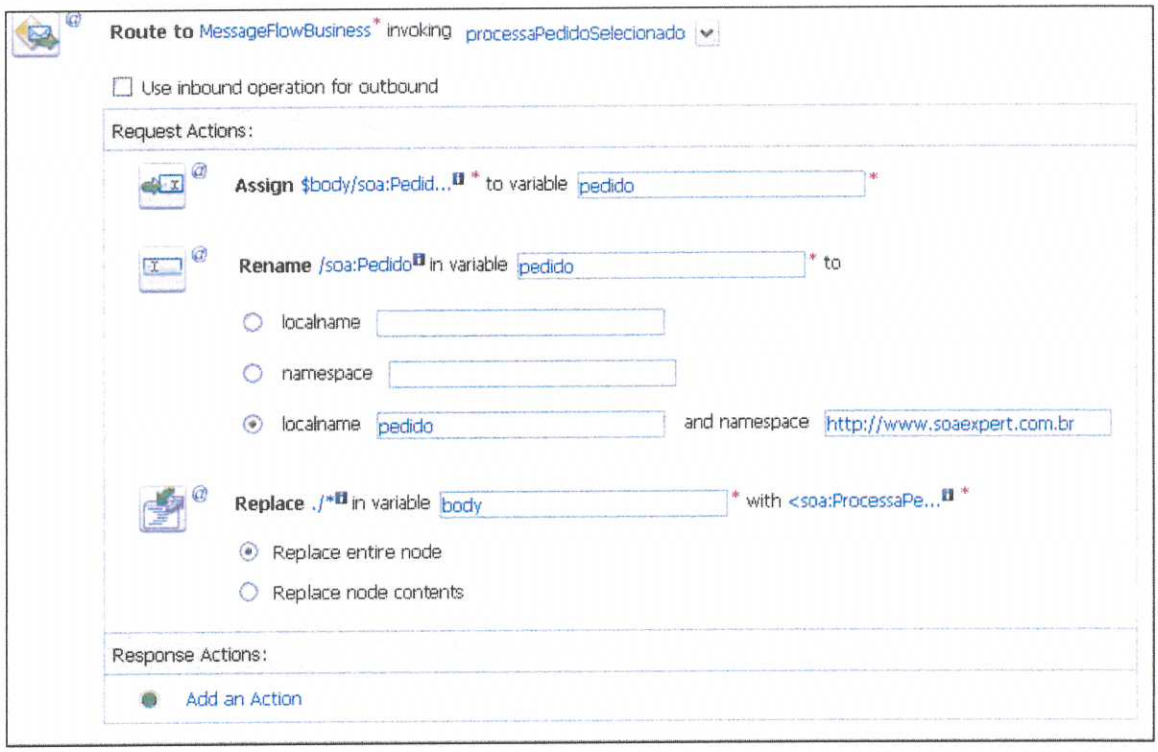


Escolha a opção processaPedidoSelecionado

Neste ponto você deverá reparar que sua estrutura de dados é ligeiramente diferente do processamento de pedidos padrão.

Por tal motivo, faça uma adequação entre a estrutura de dados, para que possa ficar condizente ao serviço.





Capítulo 5- Tratamento de Erros.

O tratamento de erro no OSB pode ser feito de forma customizada para exceções previstas e não previstas pelo provedor do serviço. No tratamento das exceções utiliza-se uma *action* do tipo *Error Handler* conforme sessão **Error! Reference source not found.**.

O objetivo do tratamento no OSB é absorver todo tipo de erro não relevante ao cliente, transformando esta ocorrência em um erro de negócio padrão conforme implementação de referência.

Para erros de negócio, apenas os membros pertencentes a este time devem ser notificados, sendo representados aqui como um *destination* que receberá mensagens JMS sobre erros de negócio conforme implementação de referência.

As equipes de infra-estrutura e desenvolvimento receberão notificações na ocorrência de erros de recurso, aqui representados por duas filas para membros das duas equipes. E, por fim, erros não tratados serão reportados apenas para a equipe de infra-estrutura através da sua fila correspondente.

Lidando com exceções esperadas

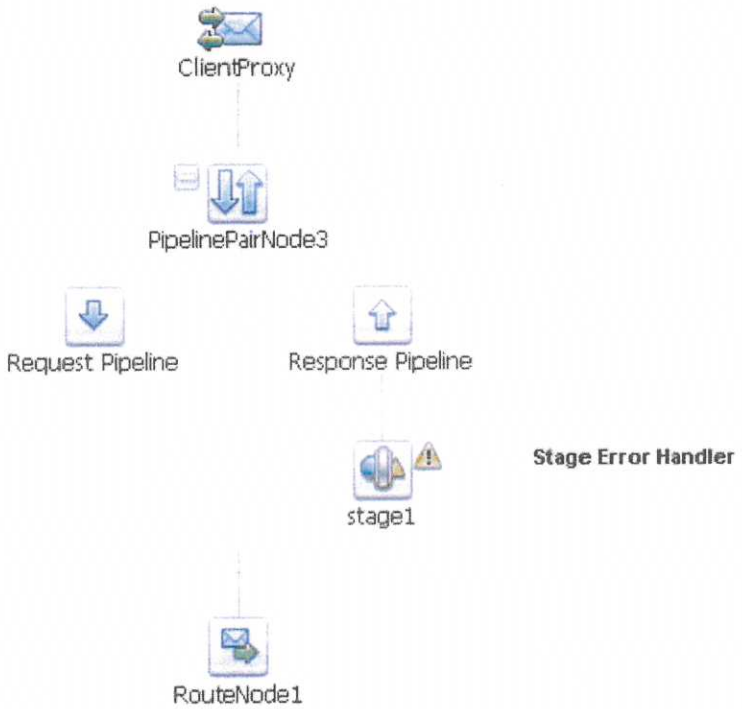
É imprescindível que cada componente de um sistema provedor de serviços possa tratar de situações de erro previstas em seu mecanismo interno, reportando o estado inconsistente com uma mensagem conforme definição do documento.

Seguindo as boas práticas, exceções devem ser tratadas o mais próximo do nó de ocorrência no *messageflow*. Ao detectar a ocorrência do erro, o fluxo deve ser desviado para uma verificação do tipo do erro – BUSINESS ou RESOURCE – que deve notificar via *alert* os destinatários interessados no tipo de erro.

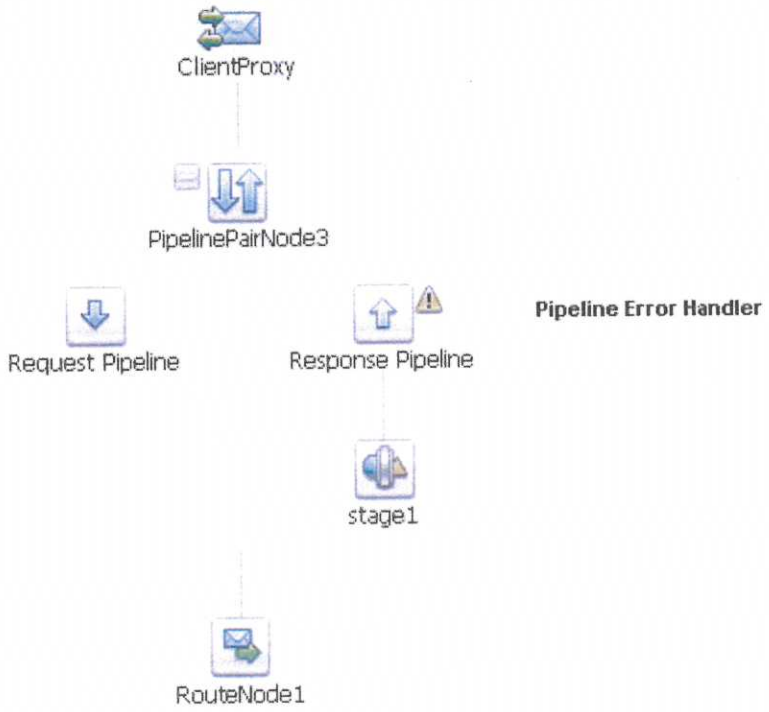
Lidando com exceções no OSB

Os node *Error Handler* podem ser aplicados em quatro níveis:

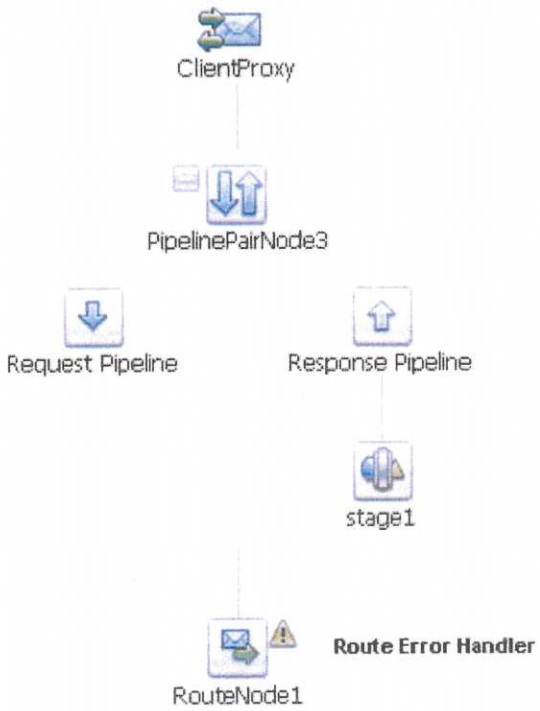
- **stage**: efetuar o tratamento padrão neste ponto dá a flexibilidade de interromper ou não o fluxo da mensagem conforme requisito do processo. Neste ponto, devem ser tratadas exceções como transformação e *service callout*;



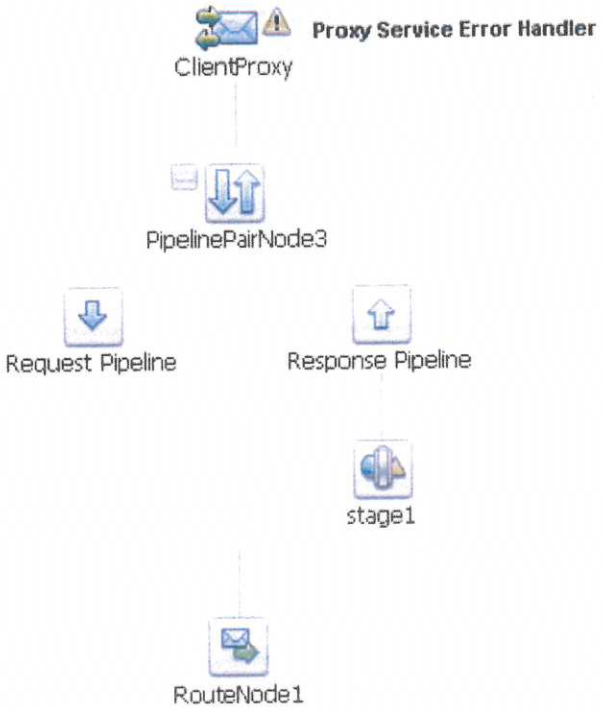
- **pipeline**: qualquer exceção relançada do *Error Handler* de *stage* será propagada para o nível de *pipeline*. Este *Error Handler* deve tratar erros decorrentes de requisições mal-formadas ou endereço de requisição inválido.



route: neste ponto devem ser tratados os erros provenientes da invocação por roteamento de um serviço. Algumas das situações que podem ocorrer neste *node* são *timeouts* no recebimento de mensagem de retorno, URLs de invocação indisponíveis, erros de segurança, erros de negócio, erros de recurso, e outros.



- **proxy**: garante o tratamento de erros não previstos pelos provedores de serviço e operações executadas dentro do *mesage flow*. É recomendado o tratamento nesse *node* apenas para erros não previstos nos níveis de *stage*, *pipeline* ou *route*. Usar esse *node* para tratar erros previstos é uma prática não recomendada.



Tratamento padrão de erros

O código para fluxo de erros tende a ser repetitivo no OSB. É uma boa prática criar um serviço especializado em tratamento de erros, isto mantém um ponto único de construção, atualização e tratamento para todos os serviços.

Notificando os usuários

A notificação do erro no OSB é feito através de alertas. É necessário criar destinatários para serem usados em cada alerta. Um destinatário pode ser uma fila JMS ou um endereço de e-mail[1]. O mecanismo de alerta deve ser utilizado para enviar notificações à equipe de negócios, desenvolvimento e infra-estrutura.

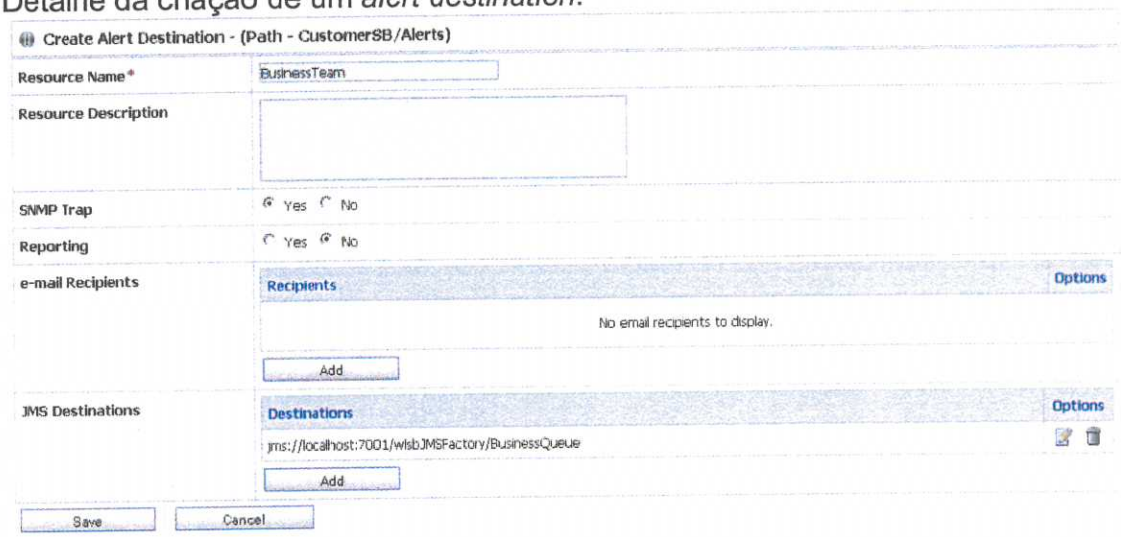
Vale observar que o alerta via e-mail pode apresentar um atraso no recebimento e, portanto, não deve ser utilizado em situações críticas.

Os alertas através de JMS fornecem mais rapidez e flexibilidade, mas implicam no desenvolvimento de uma aplicação para consumir as mensagens (ou alteração nos sistemas existentes para consumirem as mensagens).

Implementação de referência – Alerts

Assumindo a criação das filas de notificação para equipe de suporte, equipe de negócio e equipe de desenvolvimento, é gerado um *alert destination* para cada uma no console do OSB.

Detalhe da criação de um *alert destination*:



Implementação de referência – node Error Handler.

Conforme dito anteriormente, um *node Error Handler* deve tratar erros previstos e não previstos dentro de um *message flow*. Para cada nó deve ser utilizada uma ação de acordo com os requisitos de negócio do processo.

Ação *Reply*

Essa ação deve ser utilizada em situações onde o fluxo da mensagem tem de ser interrompido, então uma mensagem de erro será enviada imediatamente ao cliente do serviço. Para indicar erro, a opção *With Failure* deve ser marcada.

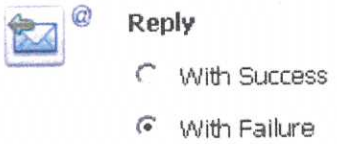


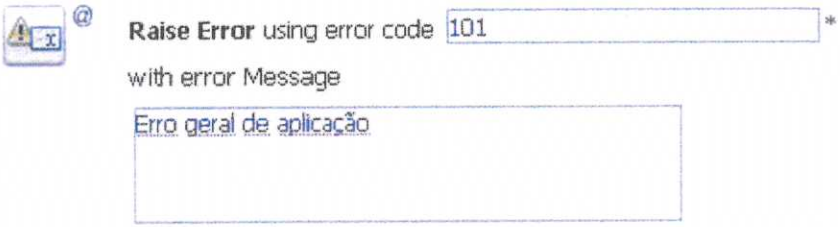
ILUSTRAÇÃO 1 – UODE REPLY PARA PROSEGUIMENTO DO FLUXO EM DIREÇÃO AO CLIENTE.

Ação *Skip*

Situações onde falhas são ignoradas após serem capturadas, permitir que o fluxo continue normalmente com o uso desta ação.

Ação *Raise error*

Lança um erro com um determinado código e descrição. O erro será tratado pelo *Error Handler* de nível superior. Transportes assíncronos, como JMS, devem conter uma ação *Raise Error* no tratamento de erro.



Ação 'Resume'

Em casos onde um erro pode ser compensado ou recuperado durante o fluxo da mensagem, esta ação deve ser utilizada para indicar que o seguimento após o nó foi o que ocasionou o erro. Seguindo os requisitos de negócio, as variáveis no contexto do *flow* podem ser alteradas visando a execução correta do restante do fluxo.

No message flow utilizado na implementação de referência, o *node Erro Handler* do *node route* verifica a presença do ErrorInfo definido neste documento, a partir daí destina a mensagem de erro aos destinatários interessados:

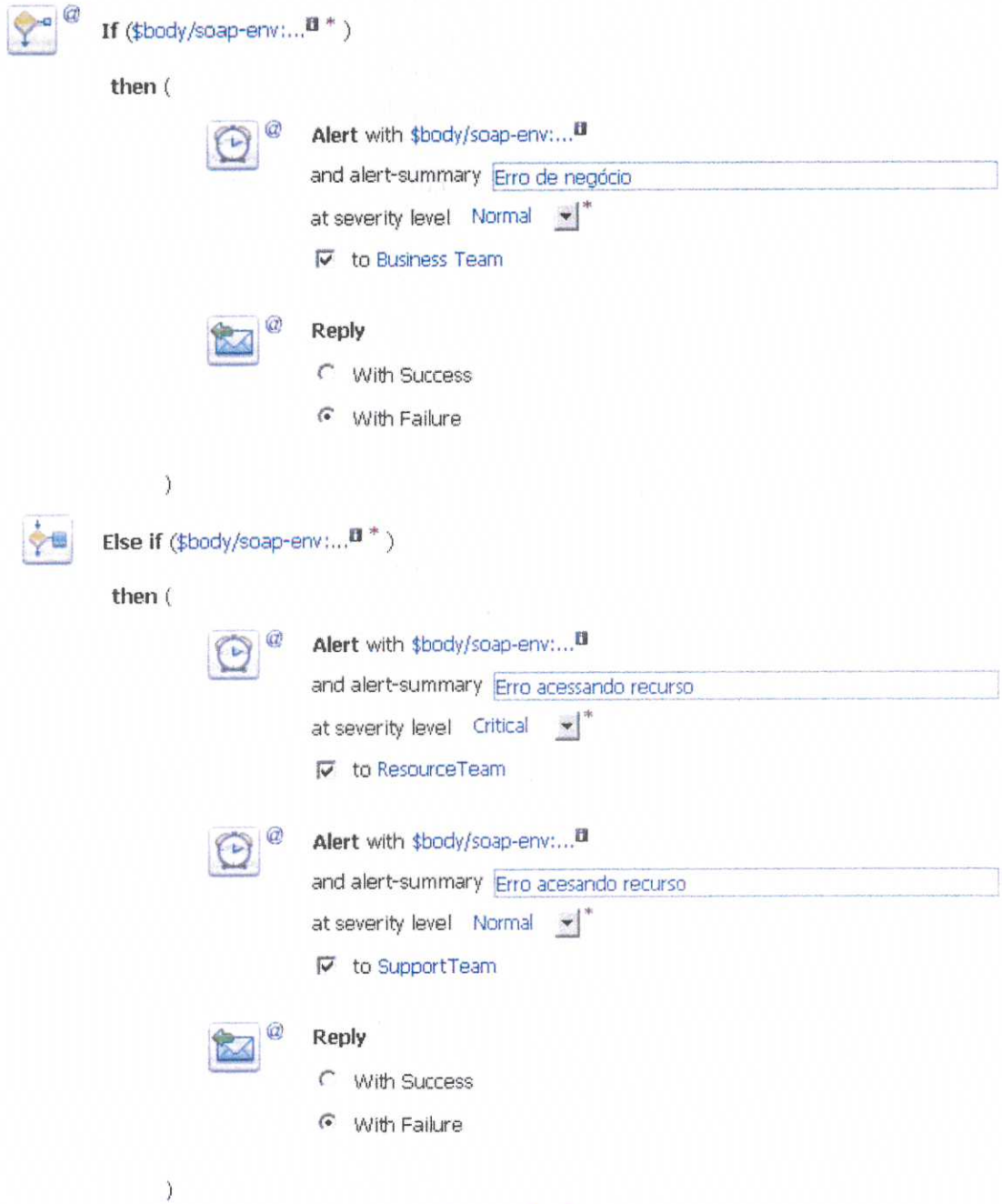
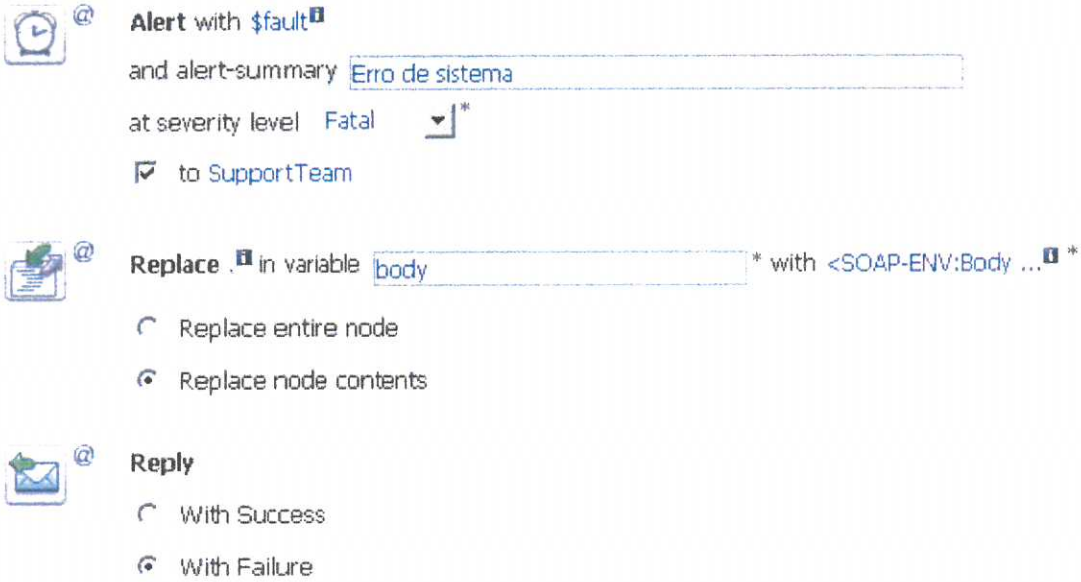


Ilustração 2 – Actions dentro de um Error Handler no nível de stage.

Para cada condição de teste executada, ao final tem-se um *reply with failure* para notificar ao cliente do serviço a ocorrência da falha. Vale observar que, neste nó, erros inesperados não são tratados, ficando a cargo dos nós em níveis superiores sanar tais tipos de erros:



Em casos de erro não tratados ou decorrentes da execução do *message flow*, a equipe de infra-estrutura deve ser notificada, no entanto a aplicação cliente deve receber um erro de alto nível. Após o alerta, o *node Error Handler* de nível mais alto que o *stage* deve executar um *replace* no conteúdo do $body com as seguinte XQuery:

```
<SOAP-ENV:Body          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <soapenv:Fault          xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cor="http://www.copr.com/corporate">
                <faultcode>soapenv:Server</faultcode>
                <faultstring>Serviço indisponível no momento.        </faultstring>
                <detail>
                        <cor:errorInfo>
                                <cor:code>-1660</cor:code>
                                <cor:message> Serviço indisponível no momento.
</cor:message>
                                <cor:time>{ fn:current-dateTime() }</cor:time>
                                <cor:additionalInfo  xsi:nil="true"/>
                                <cor:errorClass>BUSINESS</cor:errorClass>
                                <cor:source>OSB</cor:source>
                        </cor:errorInfo>
                </detail>
        </soapenv:Fault>
</SOAP-ENV:Body>
```

LISTAGEM 1 – STUERY PARA GERAR UM SOAP FAULT RETORNADO AO CLIENTE EM CASOS DE ERRO NÃO TRATADO[2]

[2] XML relevante a serviços consumidos via SOAP, um XML puro de ErrorInfo pode ser retornado em outros protocolos.

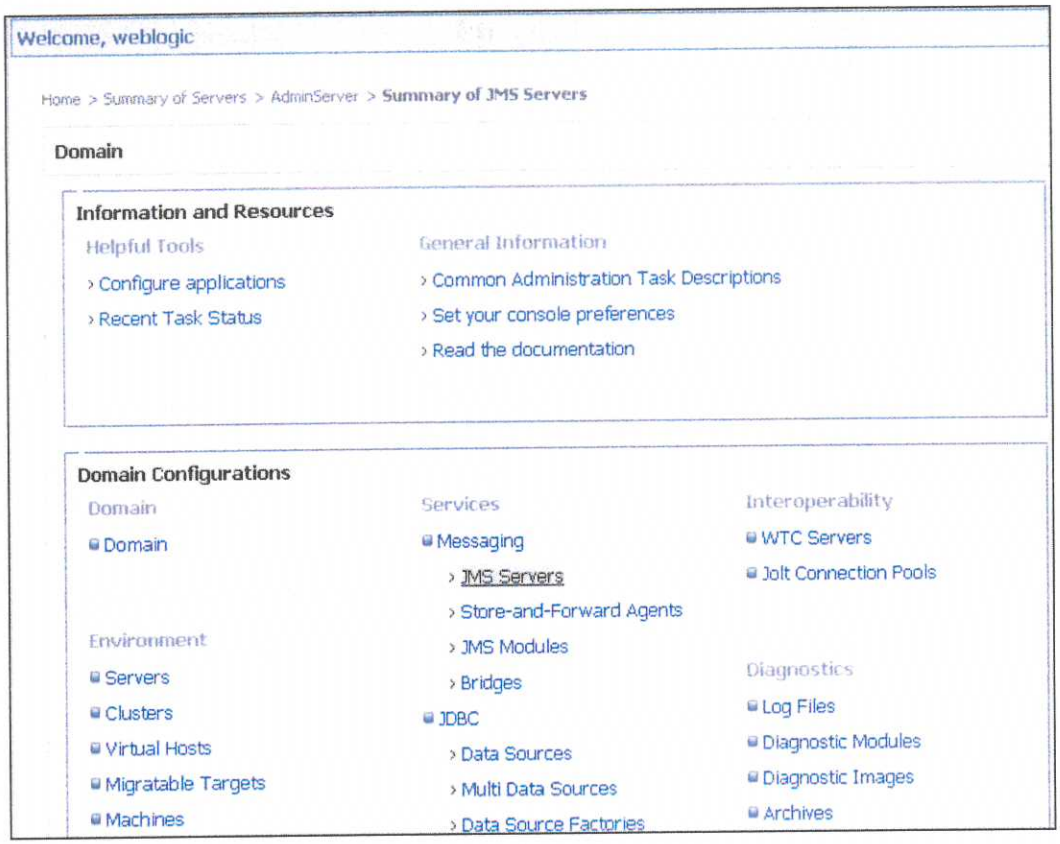
Capítulo 6– Mensageria Assíncrona - JMS.

Neste capítulo vamos dar uma olhada nos cenários mais complexos de orquestração e entender a base de todo ESB, que é o sistema de mensageria.

Aprenderemos a configurar o *Application Server*, ponto que foge um pouco ao produto OSB, mas é igualmente importante para conhecimento da solução.

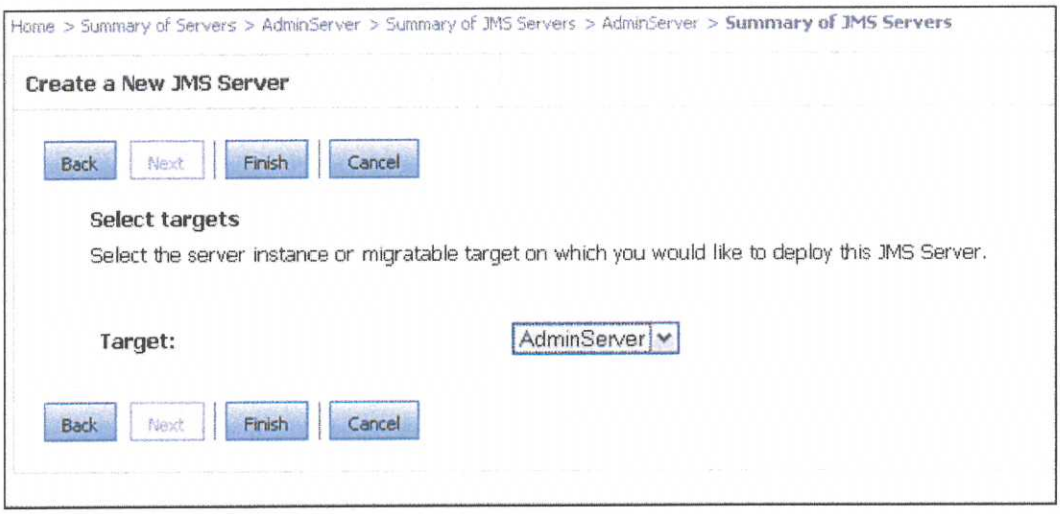
Configurando o Weblogic Server

Usando o WSL console, em Home, vá em Services | Messaging | JMS Servers.



Clique em Lock Edit e depois em New para adicionar um **novo JMS Server.**

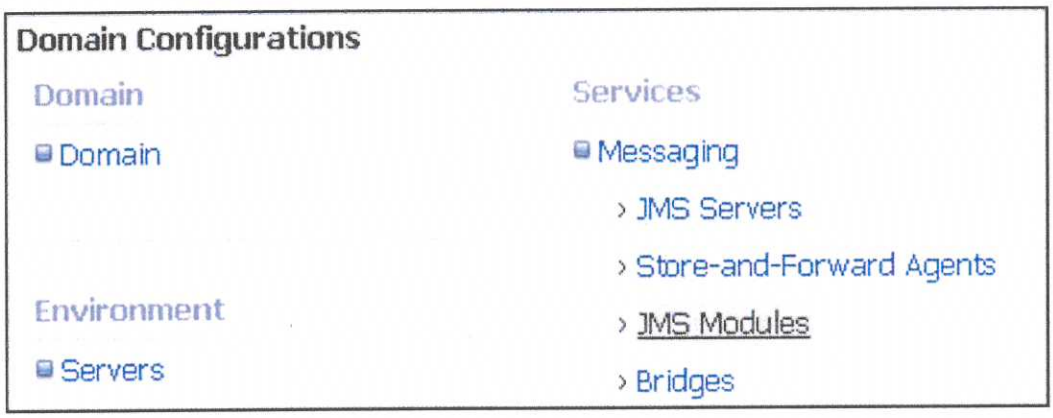
Coloque o nome de **SOAExpertJMS** e configure o tipo de persistência para **File Store**, clique em *next*.

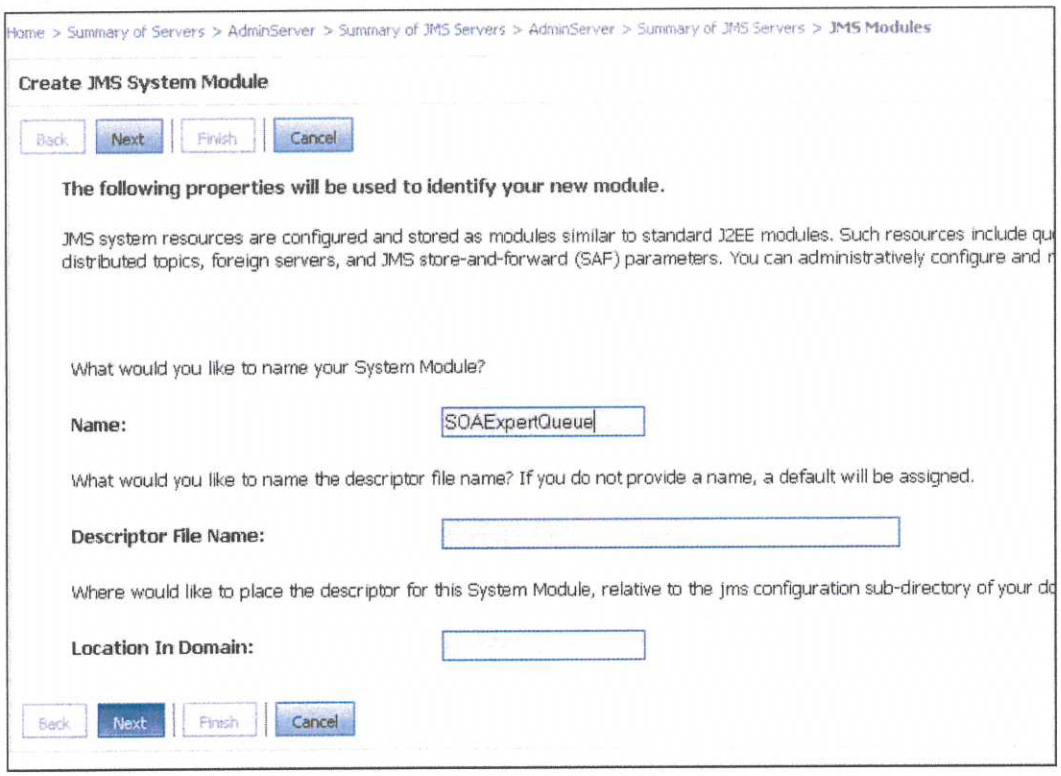


Aponte o *Target* para o **AdminServer** e *Finish*.

Agora vamos criar a fila JMS, usada para propagar os *requests* para o WebService.

Navegue na Home por Services | Messaging | **JMS Modules** e clique no botão *New*.

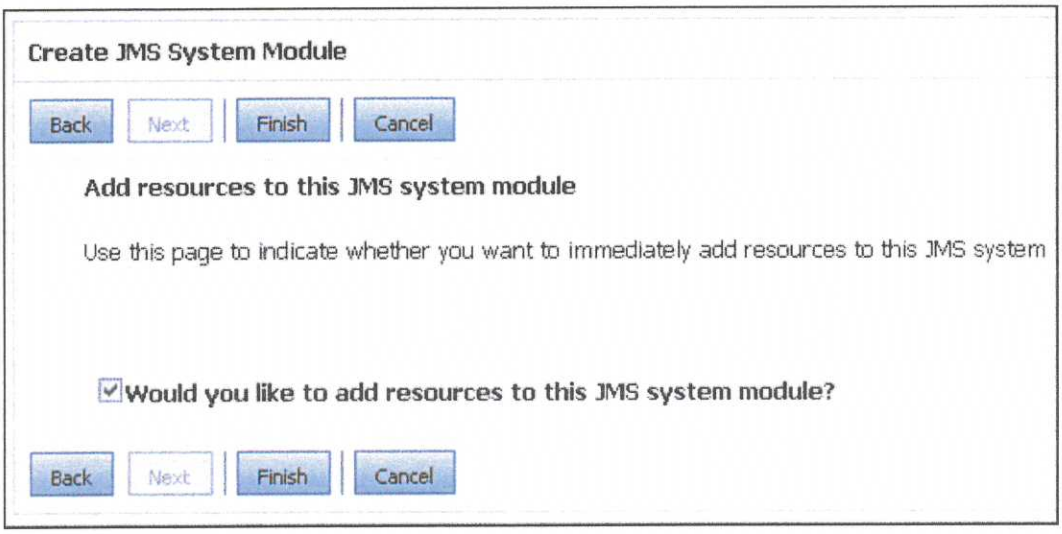




Coloque o nome da fila – **SOAExpertQueue** e deixe o *descriptor file name* e *location* em branco.

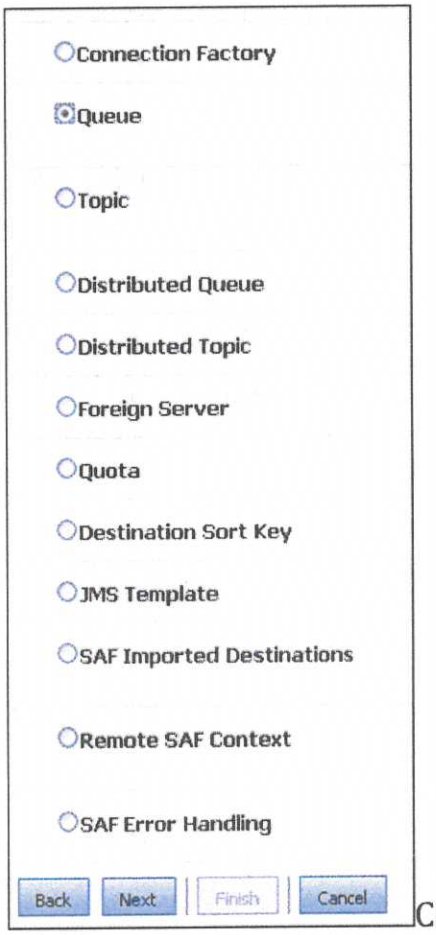
Clique em *Next,* selecione o **AdminSever** como *target* e clique em next novamente.

Agora precisamos adicionar alguns recursos ao JMS Module. Clique em "***Would you like to add resources***" e *Finish.*

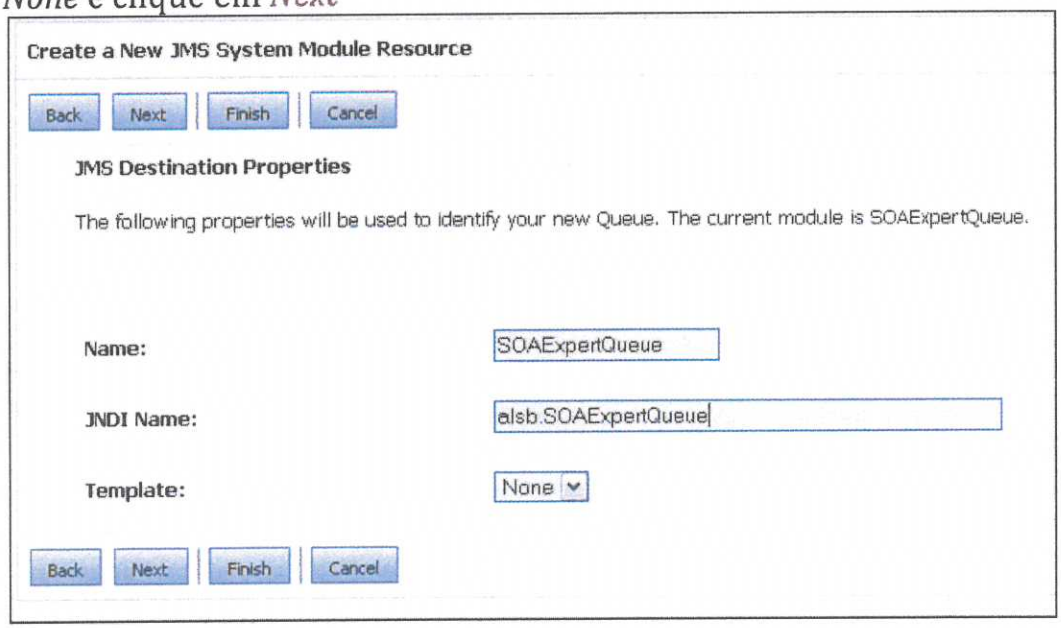


Agora vamos criar um recurso: a fila de mensagens.

Clique em New embaixo de **Summary of Resources**. Selecione **Queue** como tipo de recurso e clique Next.



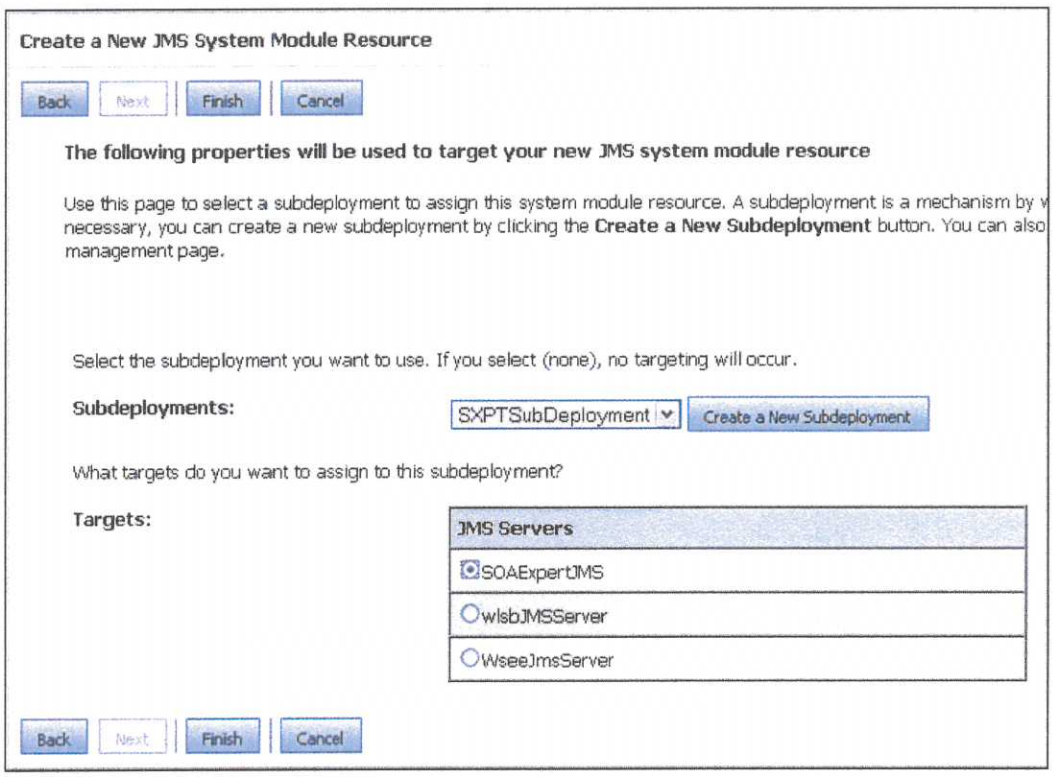
Configure o nome para **SOAExpertQueue** e coloque o nome JNDI – **OSB.SOAExpertQueue.** Em Template, coloque *None* e clique em *Next*



Ainda necessitamos criar um **Subdeployment.**

Clique em **New Subdeployment**, coloque o nome: **SXPTSubdeployment** e clique em OK.

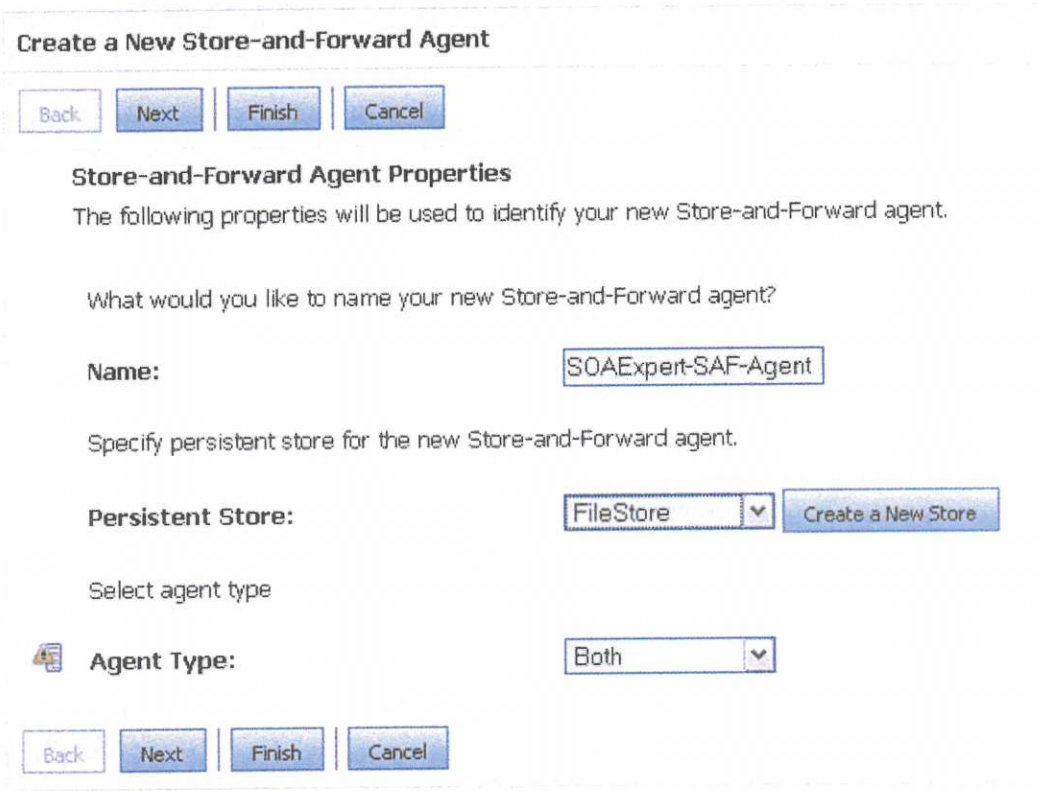
Configure o *target* do *subdeployment* para o **SOAExpertJMSServer** e clique em *Finish*.



Por fim, como último passo, vamos criar um agente **Store and Forward(SAF)** .

Ao lado, no canto esquerdo em **Domain Structure** :

Services | Messaging | Store-and-Forward-Agents.



Nomeie como **SOAExpert-SAF-Agent**, escolha **Both** para enviar e receber agentes.

Finalmente clique em **Activate Changes,** para ativar seu *JMS Server* e sua fila JMS.

Invocação Assíncrona

Nos capítulos anteriores, comentamos sobre a necessidade de utilizar serviços assíncronos em determinadas circunstâncias como, por exemplo, esperar por um longo processamento. Entretanto se seu cliente ou cenário de negócio necessita de uma resposta **imediata** o desenho do seu serviço deverá ser síncrono.

Há ainda um terceiro tipo em cima dos serviços assíncronos que é Assíncrono Request-Response, ou seja, aquele que responde ao final do processo. Ex. "**Quando você completar 18 anos, venha pode voltar e tirar a carteira de motorista**". Então o serviço responderá ao final do seu processo.

Business Service Assíncrono

Para criar um serviço assíncrono, tenha em mente algumas regras básicas:

- Uma operação assíncrona deve retornar um void como estrutura de dados.
- As operações devem conter um atributo @Oneway.
- As operações podem especificar tentativas usando o atributo @MessageBuffer()
- Os serviços assíncronos usam o atributo @BufferQueue para especificar a fila JMS que esta ouvindo.

Examinando um WSDL assíncrono, perceberá que a única diferença é que não há tipo de retorno :

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.soaexpert.com.br/AsyncBusiness/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="AsyncBusiness"
    targetNamespace="http://www.soaexpert.com.br/AsyncBusiness/">
    <types>
        <xsd:schema targetNamespace="http://www.soaexpert.com.br/AsyncBusiness/"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:ped="http://www.soaexpert.com.br/pedido/">
            <xs:import namespace="http://www.soaexpert.com.br/pedido/"
                schemaLocation="./pedido.xsd" />
            <xsd:element name="processaPedido" type="ped:Pedido" />
        </xsd:schema>
    </types>

    <message name="processaPedido">
        <part element="tns:processaPedido" name="processaPedido" />
    </message>

    <portType name="AsyncBusiness">
        <operation name="processaPedidoAssync">
            <input message="tns:processaPedido" />
        </operation>
    </portType>

    <binding name="AsyncBusinessSOAP" type="tns:AsyncBusiness">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="processaPedidoAssync">
            <soap:operation
                soapAction="http://www.soaexpert.com.br/AsyncBusiness/processaPedidoAssync" />
            <input>
                <soap:body parts="processaPedido" use="literal" />
            </input>
        </operation>
    </binding>
```


```
    <binding name="AsyncBusinessSOAP" type="tns:AsyncBusiness">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="processaPedidoAssync">
            <soap:operation
                soapAction="http://www.soaexpert.com.br/AsyncBusiness/processaPedidoAssync" />
            <input>
                <soap:body parts="processaPedido" use="literal" />
            </input>
        </operation>
    </binding>
```


```
    <service name="AsyncBusiness">
        <port binding="tns:AsyncBusinessSOAP"
            name="AsyncBusinessSOAP">
            <soap:address
                location="http://localhost:7001/AsyncBusiness" />
        </port>
    </service>
</definitions>
```

Se você examinar a classe de implementação, duas anotações adicionais são percebidas: **@BufferQueue**, que é usada para especificar a fila JMS a qual o serviço ouve, e **@MessageBuffer**, que especifica os atributos **retryCount** e **retryDelay.**

```
package br.com.soaexpert.services;

import java.util.Date;

/**
 * AsyncBusinessImpl class implements web service endpoint interface
 * AsyncBusiness
 */

@WebService(serviceName = "AsyncBusiness", targetNamespace = "http://www.soaexpert.com.br/AsyncBusiness/", endpoin
@WLHttpTransport(contextPath = "/", serviceUri = "AsyncBusiness", portName = "AsyncBusinessSOAP")
@BufferQueue(name="alsb.SOAExpertQueue")
public class AsyncBusinessImpl implements AsyncBusiness {

    public AsyncBusinessImpl() {

    }

    @MessageBuffer(retryCount=5, retryDelay="10 seconds")
    public void processaPedidoAssync(
            br.com.soaexpert.pedido.Pedido processaPedido){

        System.out.println("Inciando o processamento do pedido do cliente: " + processaPedido.getPrimeiroNome());

            try {
                Thread.sleep(30000);
            } catch (InterruptedException e) {

                e.printStackTrace();
            }finally{

                System.out.println("Completado processamento do cliente: "
                        +processaPedido.getPrimeiroNome());
            }

    }

}//The end
```

Capítulo 7– Bindings e camadas de transporte

Neste capítulo estudaremos os modelos de invocação usados pelas nossas mensagens.

Um "**service type**" representa um formato de mensagem como, por exemplo, SOAP ou XML, definidos por um contrato WSDL; SOAP sem um WSDL, XML sem um WSDL e um Messaging Type (Binary, Text, MFL ou XML).

Um protocolo de transporte é usado para transportar as mensagens de um *sender* para um *receiver*. **JMS, HTTP e HTPPS** são basicamente os protocolos do OSB.

Normalmente quando se cria um serviço a primeira etapa é definir um *service type* . O mais comum é o SOAP com WSDL, contudo você pode optar em utilizar um tipo diferente, dependendo das suas necessidades. Ex: Se você estiver provisionando um serviço a clientes que não falam SOAP (como numa arquitetura REST), você pode implementar um service type XML, ou, se seu cliente consegue se comunicar somente num formato binário proprietário, você pode escolher um Binary Messaging Type, e assim por diante.

Service Types e Trasnport Protocols

Service Type	**Transport Protocols**
SOAP ou XML com WSDL	JMS, HTTP, HTTPS
SOAP sem WSDL	JMS, HTTP,HTTPS
XML sem WSDL	JMS,HTTP,HTTPS,Email,File,FTP
Messaging Type	JMS,HTTP,HTTPS,Email,File,FTP

Obs: Todos os service types podem enviar e receber attachments usando MIME – Multipurpose Internet Mail Extensions.

SOAP com WSDL

SOAP com WSDL é o tipo de mensagem mais popular, especialmente no mundo Web Service. SOAP originalmente era um acrônimo de *Simple Object Access Protocol*, entretanto hoje é conhecido somente pelo nome e o acrônimo não é mais utilizado.

SOAP é um protocolo XML para enconding e transmissão de estrutura de dados.

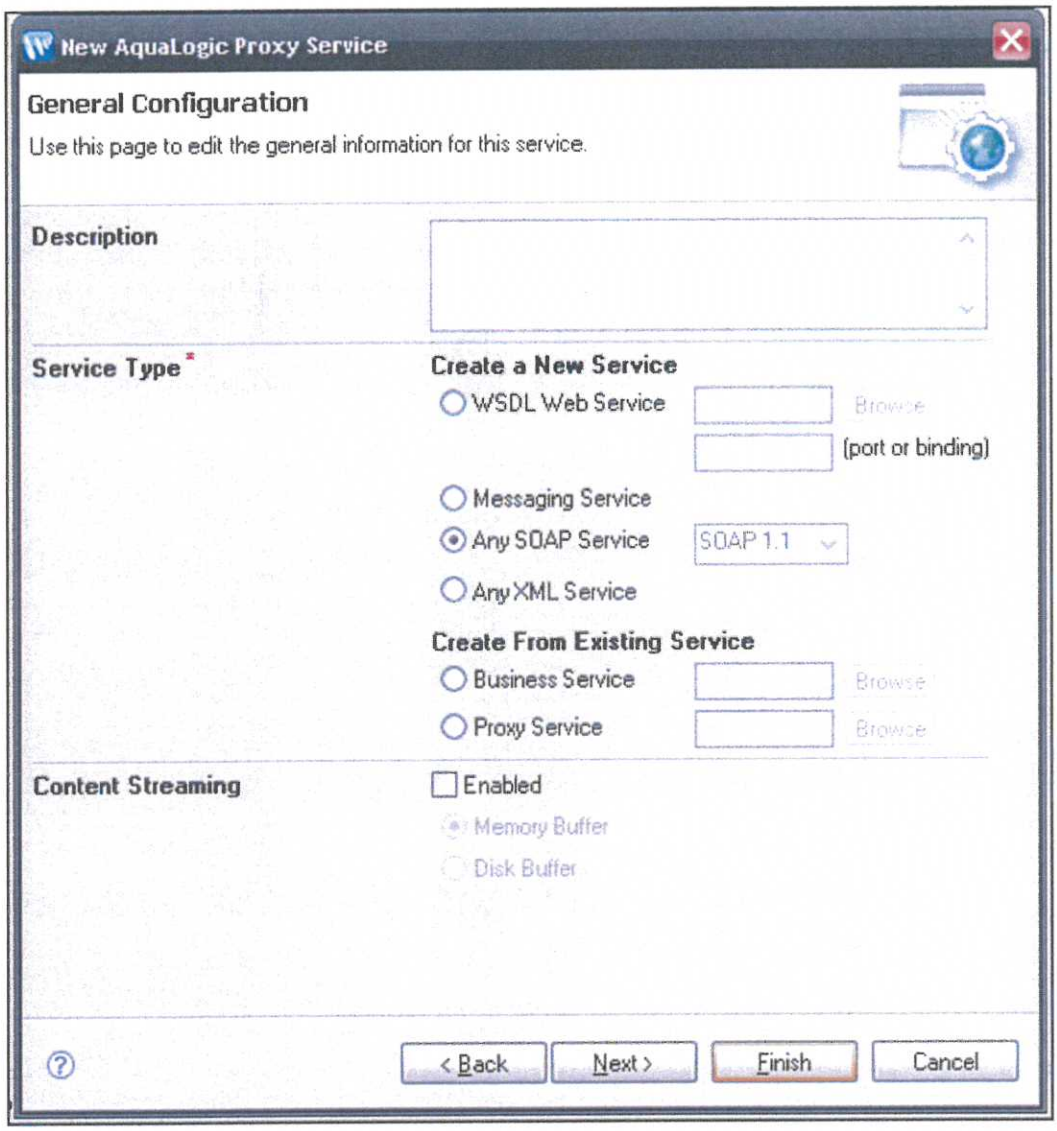
SOAP sem WSDL

Pode soar estranho quando analisamos essa opção, afinal por que eu teria um SOAP sem uma definição de estrutura e contrato? Essa abordagem é válida quando temos que lidar com estruturas legadas que utilizavam SOAP sem contratos WSDL.

Um outro uso seria ter um ponto único de entrada SOAP para todas as mensagens, então elas seriam examinadas e roteadas aos serviços apropriados.

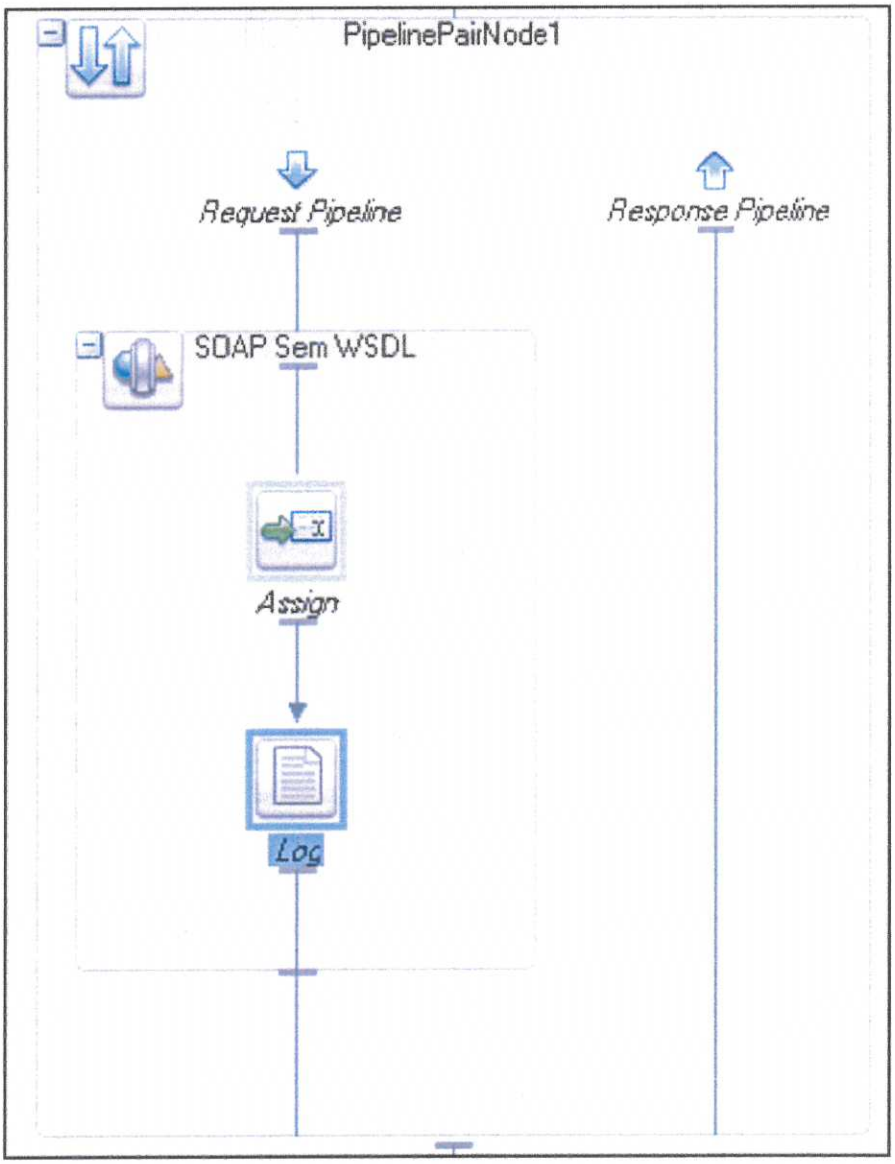
Exercício : Implementando um SOAP sem WSDL

Crie um Proxy service usando **Any SOAP Service** como serviço.



Posteriormente crie um fluxo simples de mensagens. Adicione um *Pipeline Pair* com um único *stage* no *request pipeline*. Neste *stage*, adicione duas ações.

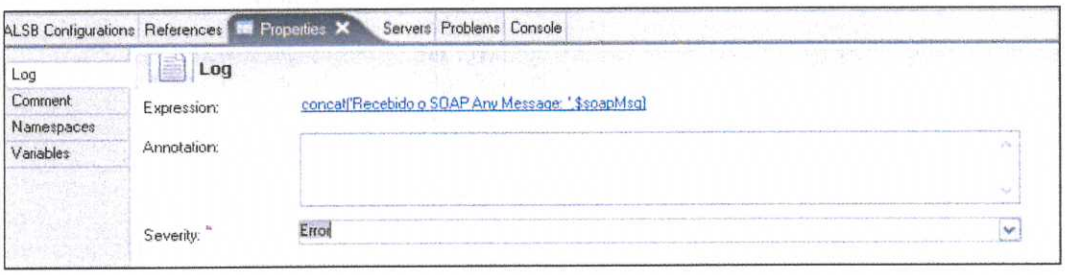
A primeira faz um *Assign* do conteúdo de `$body` para uma variável local chamada `$soapMsg`.



A segunda ação irá logar usando a seguinte expressão :

```
concat(' Recebendo SOAP Any Message: ', $soapMessage).
```

Configure a severidade do nível do log para **Error**.



Esse exemplo mostra que o serviço aceita qualquer estrutura SOAP, seria como no Java você ter um método que recebe *Collection* como parâmetro de entrada e retorna outra *Collection*.

Crie um arquivo SOAP como o exemplo e envie via TestConsole para o Proxy Service como payload.

Exemplo:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/
    <soapenv:Header/>
    <soapenv:Body>
        <sxpt:processaQualquerPedido xmlns:sxpt="http://www.soaexpert.com.br">
            <sxpt:Pedido>
                <sxpt:DataPedido>2006-04-26T13:07:41.171-07:00</sxpt:DataPedido>
                <sxpt:PedidoId>1</sxpt:PedidoId>
                <sxpt:PrimeiroNome xsi:nil="true"/>
                <sxpt:UltimoNome xsi:nil="true"/>
            </sxpt:Pedido>
        </sxpt:processaQualquerPedido>
    </soapenv:Body>
</soapenv:Envelope>
```

TestConsole:

```
(receiving request)
PipelinePairNode1
    SOAP Sem WSDL
        Message Context Changes
            added $soapMsg
                <soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
                    <soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <soapenv:Header/>
                        <soapenv:Body>
                            <sxpt:processaQualquerPedido xmlns:sxpt="http://www.soaexpert.com.br">
                                <sxpt:Pedido>
                                    <sxpt:DataPedido>2006-04-26T13:07:41.171-07:00</sxpt:DataPedido>
                                    <sxpt:PedidoId>1</sxpt:PedidoId>
                                    <sxpt:PrimeiroNome xsi:nil="true"/>
                                    <sxpt:UltimoNome xsi:nil="true"/>
                                </sxpt:Pedido>
                            </sxpt:processaQualquerPedido>
                        </soapenv:Body>
                    </soapenv:Envelope>
                </soapenv:Body>
            changed $body
            changed $inbound
(echoing request)
PipelinePairNode1
```

XML com WSDL

SOAP com WSDL é extremamente popular mas não é a única forma de enviar uma mensagem, também podemos utilizar um arquivo XML e referenciar a estrutura XML da mensagem mapeando para um contrato WSDL.

Essa escolha de padrão lembrará muito a arquitetura REST, fazendo uso dos protocolos HTTP POST/GET.

EXEMPLO

Estudando o exemplo acima, você irá reparar que a sessão de SOAP binding foi reescrita para HTTP binding usando o verbo POST.

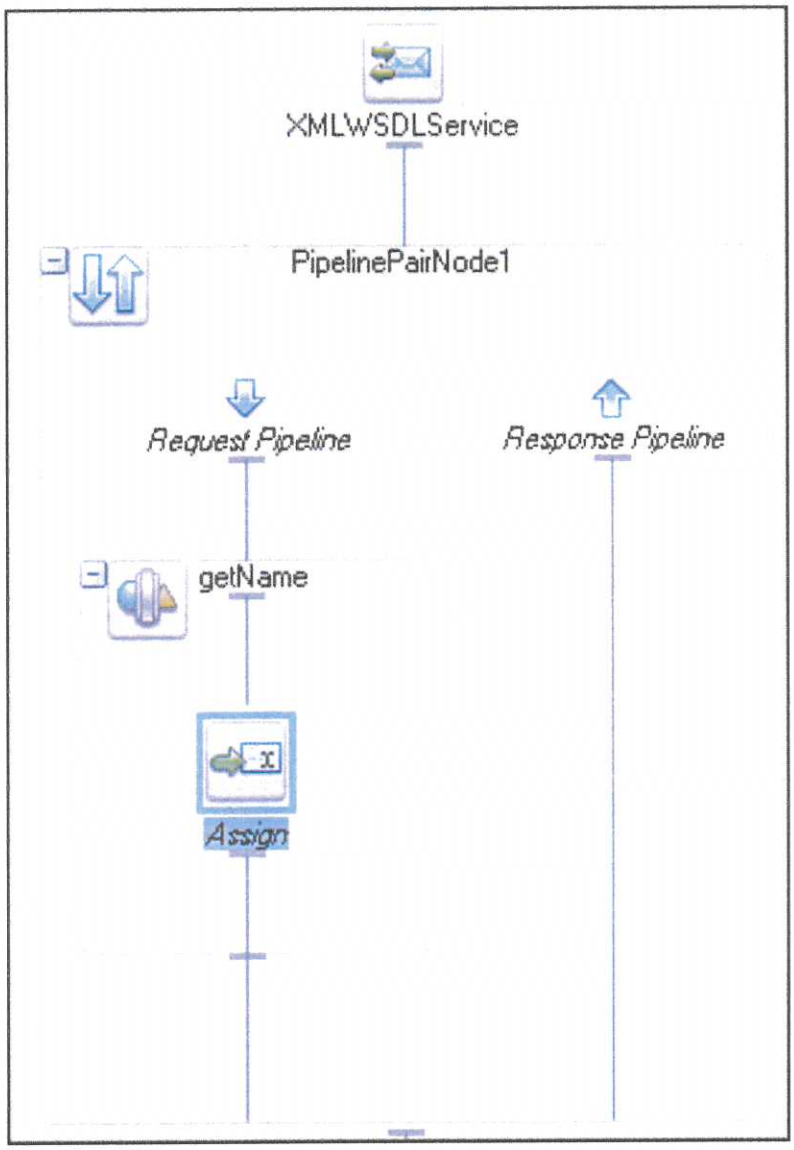
A operação **getHello** existe em /**GetHello**(caminho completo – http://localhost:7001/SOAExpert/XMLWSDLService/getHello

E o parâmetro de entrada, input, é encoded usando o HTTP POST e retorna uma estrutura XML.

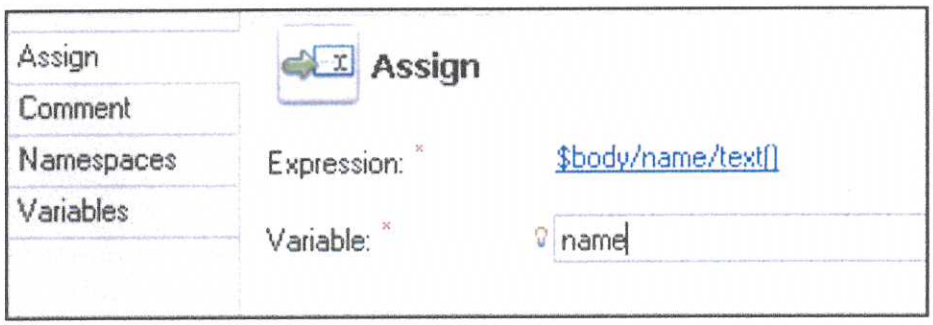
Exercício: Implementando um XML com WSDL.

Crie um novo **Proxy Service** com o nome de **XMLWSDLService,** baseado no WSDL de exemplo deste tópico. Configure o endpoint para /XMLWSDLService e selecione os defaults para o restante.

Agora crie um fluxo de mensagem simples que pega o nome do documento XML que foi enviado e retorna um XML contendo a resposta.

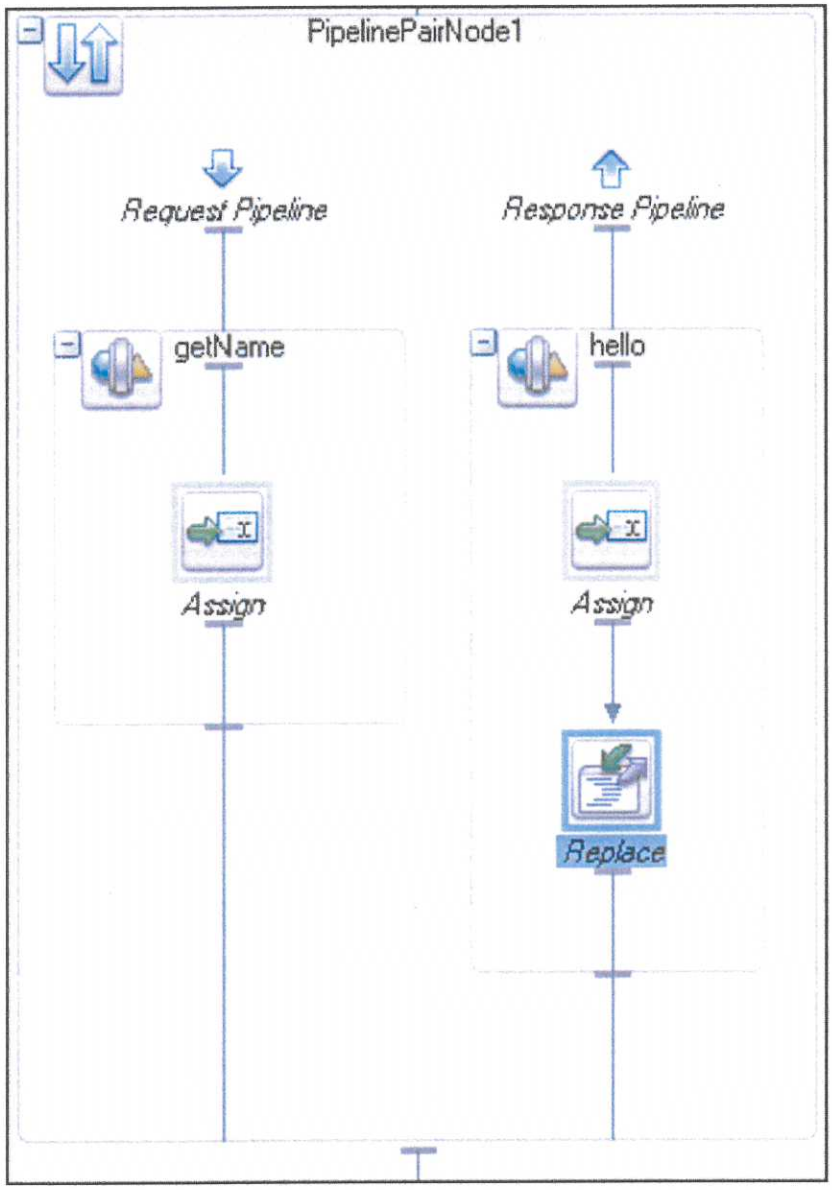


Crie agora um *stage* com o nome de getHello no Request Pipeline , que contém um *Assign* e a expressão `$body/text()`



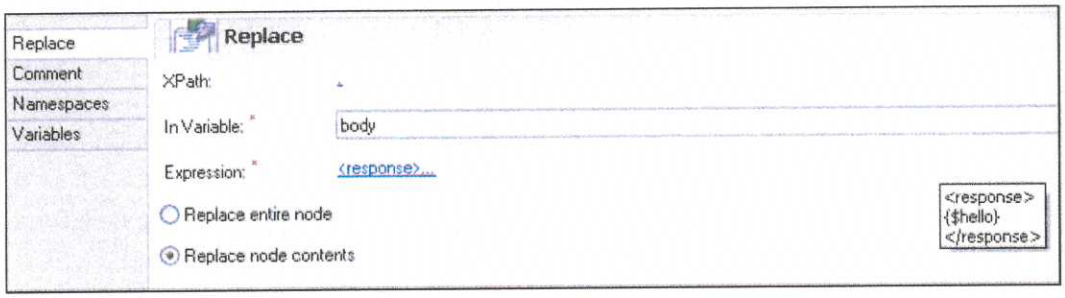
para a variável local **name.**

Crie agora o *stage***hello** no *pipeline response.* Esse *stage* possui duas ações. A primeira ação faz um **Assign** da expressão – `concat('Olah ',$name)` para a variável local **hello.**



A segunda ação irá fazer um *Replace,* que pega o conteúdo do *SOAP Body element*, representado pela variável `$body` , para a expressão

```
<response> {$hello} </response>
```



Você pode usar o test console para testar seu serviço, entretanto num Java Client, manipulando diretamente o protocolo HTTP, ficaria mais divertido estampada a resposta no console da aplicação.

Abaixo segue um código referência:

```
package br.com.soaexpert.services.client;

import java.io.BufferedReader;

public class XMLWSDLClient {

    public static void main(String[] args) {
        //Constroi o XML que serah enviado no Post
        String doc = "<?xml version='1.0'?><hello>Felipe</hello>";
        String hostname = "localhost";
        int port = 8001;
        try {
            InetAddress  addr = InetAddress.getByName(hostname);
            Socket sock = new Socket(addr, port);


            String path = "/SOAExpert/XMLWSDLService/getHello";
            BufferedWriter  wr = new BufferedWriter(new OutputStreamWriter(sock.getOutputStream(),"UTF-8"));
            wr.write("POST " + path + " HTTP/1.0\r\n");
            wr.write("Host: " + hostname + "\r\n");
            wr.write("Content-Length: " + doc.length() + "\r\n");
            wr.write("Content-Type: text/xml; charset=\"utf-8\"\r\n");
            wr.write("\r\n");

            //Envio documento
            wr.write(doc);
            wr.flush();

            // Resposta
            BufferedReader rd = new BufferedReader(new InputStreamReader(sock.getInputStream()));
            String line;
            while((line = rd.readLine()) != null)
                System.out.println(line);
        } catch(UnknownHostException ex) {
            ex.printStackTrace();
        } catch(IOException ex) {
            ex.printStackTrace();
        }
    }
}//The end
```

XML sem WSDL

O XML sem WSDL é similar ao SOAP sem WSDL e é bastante flexível e útil para escalar, entretanto muito difícil para controlar a proliferação de serviços e gerenciar a visibilidade de mudança de versões.

Messaging Types

Um ***service type*** permite uma forma de mensagem como *request* e um diferente formato de mensagem como resposta. Ex: Você tem um serviço que aceita um XML como *request*, mas retorna um arquivo binário.

Obs: Message Type service também suporta os protocolos de transporte File, FTP e Email. Contudo esses são "*one-way*" e não "*request-response*", logo você precisa definir o **response message type** como **None.**

Message Type : Binary

Como o nome diz, ele permite que você envie mensagens binárias. Você pode usar um serviço como este para se comunicar com aplicações legadas ou que usam uma estrutura proprietária de dados.

Exercício - Enviar uma mensagem binária via JMS.

Nosso exemplo envia uma mensagem binária para um MDB – *message-driven bean* em cima do protocolo JMS.

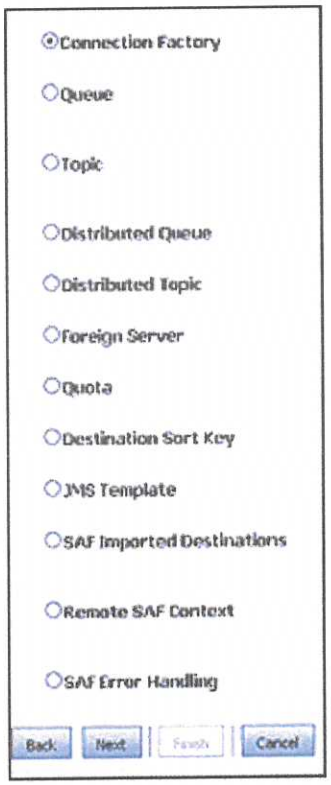
1- Como no capítulo 6, crie sua estrutura JMS até a sessão de Modules.

2- Crie um **JMS topic** nomeado *binaryFile* e um **connection factory** nomeado ***OSBConnectionFactory***.

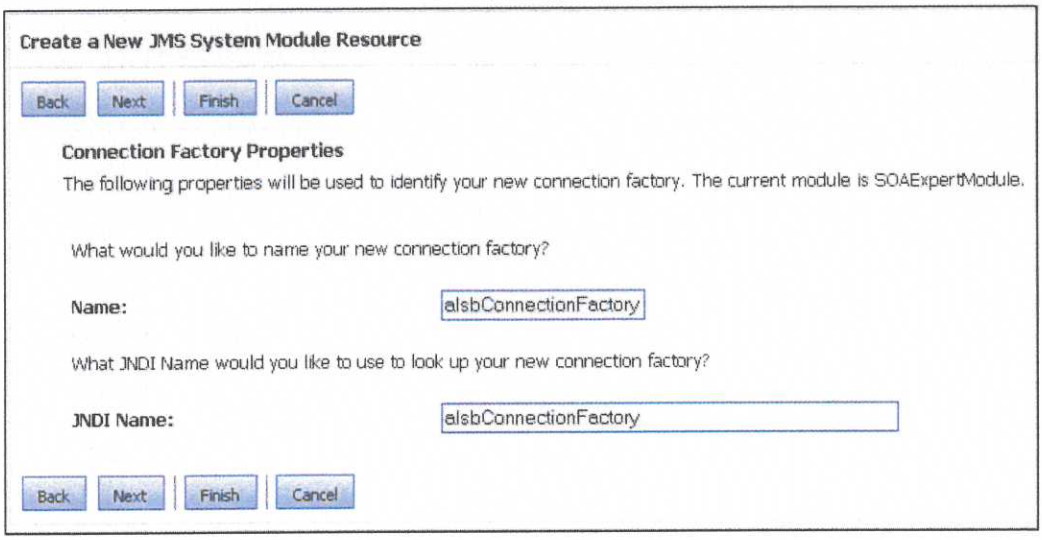
3- Defina um *business service* chamado **BinaryBusinessService** que roteará as mensagens binárias para o *JMS Topic.*

4- Defina um *Proxy Service* chamado **BinaryService**, que aceitará um formato binário de mensagem e roteará para o *business service.*

2- Dentro de JMS Modules Resources, crie uma JMS Connection Factory chamada **OSBConnectionFactory**

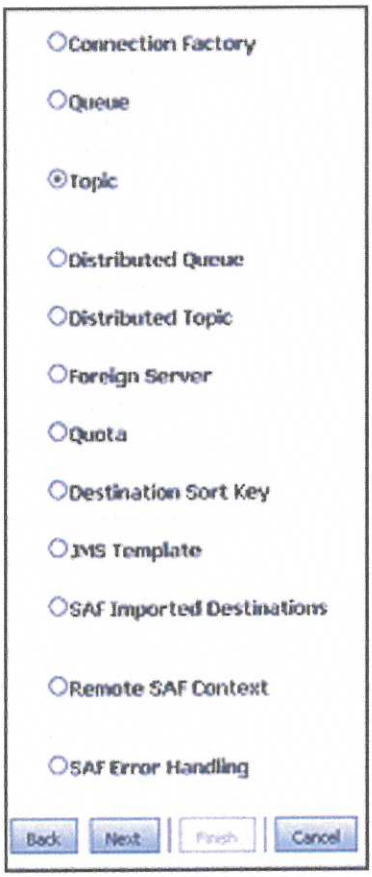


Configure o JNI name para o mesmo nome da *connection factory.*

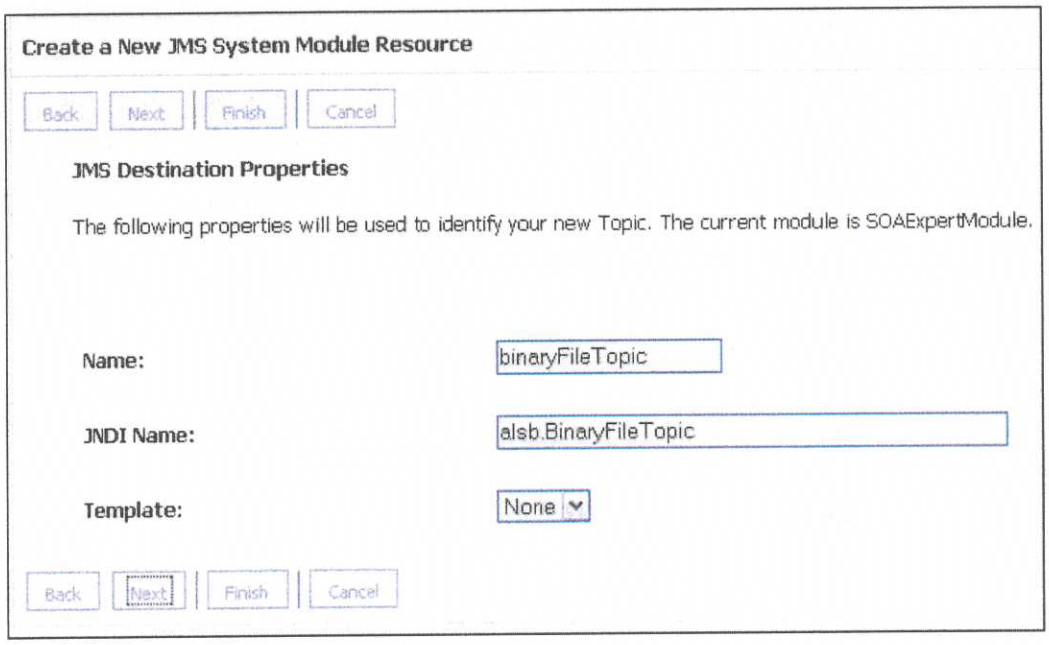


Selecione Next e target para o AdminServer.

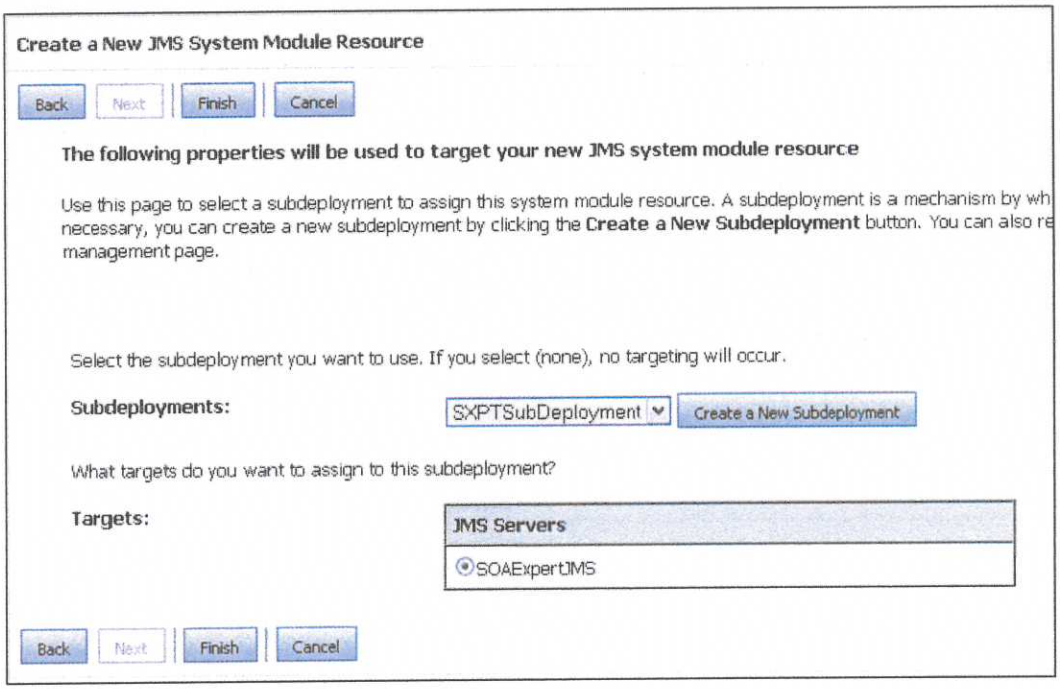
Agora você irá fazer o mesmo processo para criar sua JMS Topic. Volte em **Summary of Resources** e clique em new, selecione **Topic** :



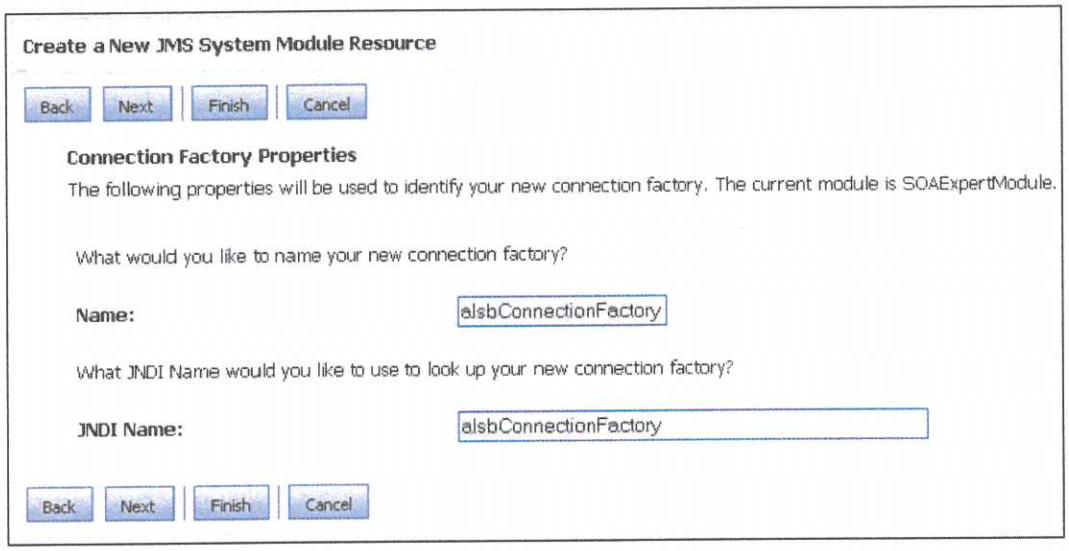
Configure o nome para **binaryFileTopic** e o JNI para **OSB.BinaryFileTopic.**



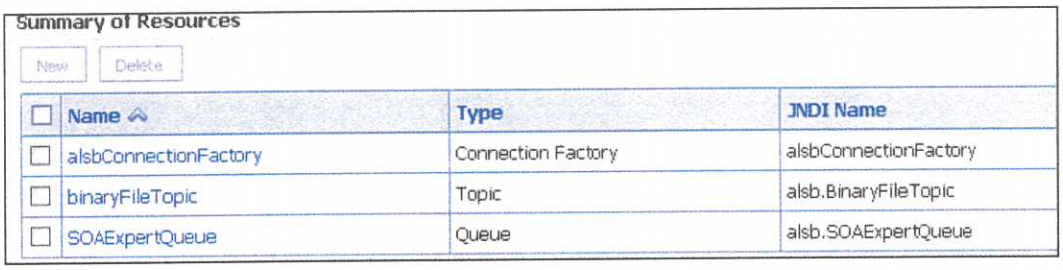
Relacione com o **Subdeployment SXPTSubDeployment** criado no exercício anterior e target para SOAExpertJMS server.



Por fim, crie uma Connection Factory com o nome de **OSBConnectionFactory**, usando o mesmo procedimento.



Ative todas as modificações via Change Center.



Name	Type	JNDI Name
alsbConnectionFactory	Connection Factory	alsbConnectionFactory
binaryFileTopic	Topic	alsb.BinaryFileTopic
SOAExpertQueue	Queue	alsb.SOAExpertQueue

Confira os recursos criados!!

3- Antes de entrarmos no Business Service, crie primeiramente um **Message-Driven Bean** , o qual irá consumir a fila **binaryFileTopic** JMS, criada para mensagens binárias, e gravar as mensagens binárias no local system.

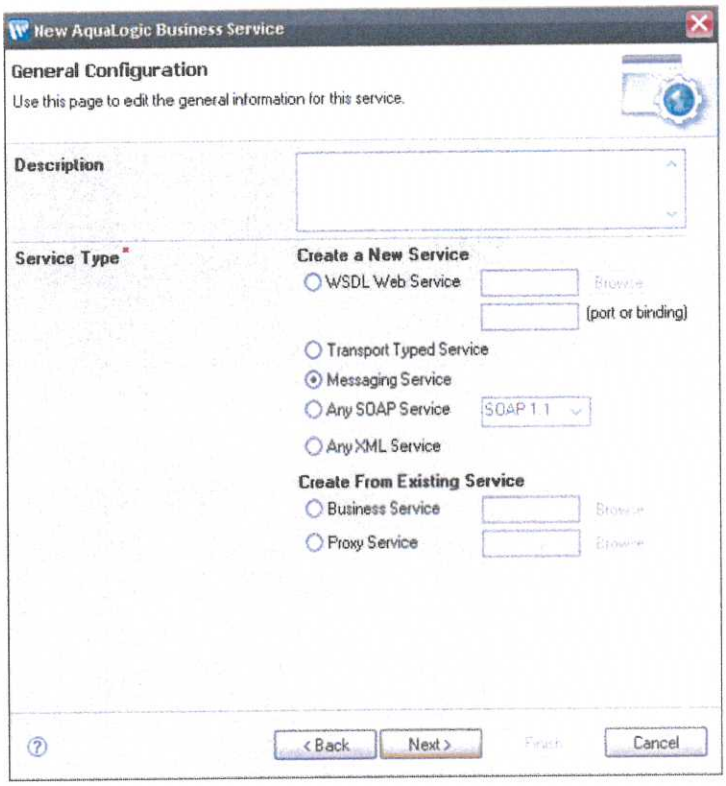
Exemplo:

```
@MessageDriven(maxBeansInFreePool = "200", destinationType = "javax.jms.Topic", initialBeansInFreePool = "20",
        transTimeoutSeconds = "0", defaultTransaction = MessageDriven.DefaultTransaction.REQUIRED,
        durable = Constants.Bool.FALSE, ejbName = "binaryMDB", destinationJndiName = "alsb.BinaryFileTopic")
public class BinaryMDB implements MessageDrivenBean, MessageListener {
    public final static long serialVersionUID = 1L;

    private static final boolean VERBOSE = true;

    private MessageDrivenContext m_context;

    public void setMessageDrivenContext(MessageDrivenContext ctx) {
        m_context = ctx;
    }

    public void onMessage(Message msg) {
        BytesMessage bm = (BytesMessage) msg;
        try {
            long length = bm.getBodyLength();
            byte[] binaryData = new byte[(int) length];
            int numRead = bm.readBytes(binaryData);
            // Cria um arquivo temporario.
            File outputFile = File.createTempFile("mdb_", "xxx");
            log("Criado o arquivo: " + outputFile.getAbsolutePath()
                    + " com tamanho de  " + numRead + " bytes");
            FileOutputStream fos = new FileOutputStream(outputFile);
            fos.write(binaryData);
            fos.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (JMSException ex) {
            System.err.println("Uma excessao ocorreu:  " + ex.getMessage());
        }
    }
    public void ejbCreate() {
    }
    public void ejbRemove() {
    }

    private void log(String s) {
        if (VERBOSE)
            System.out.println(s);
    }
}// The end
```

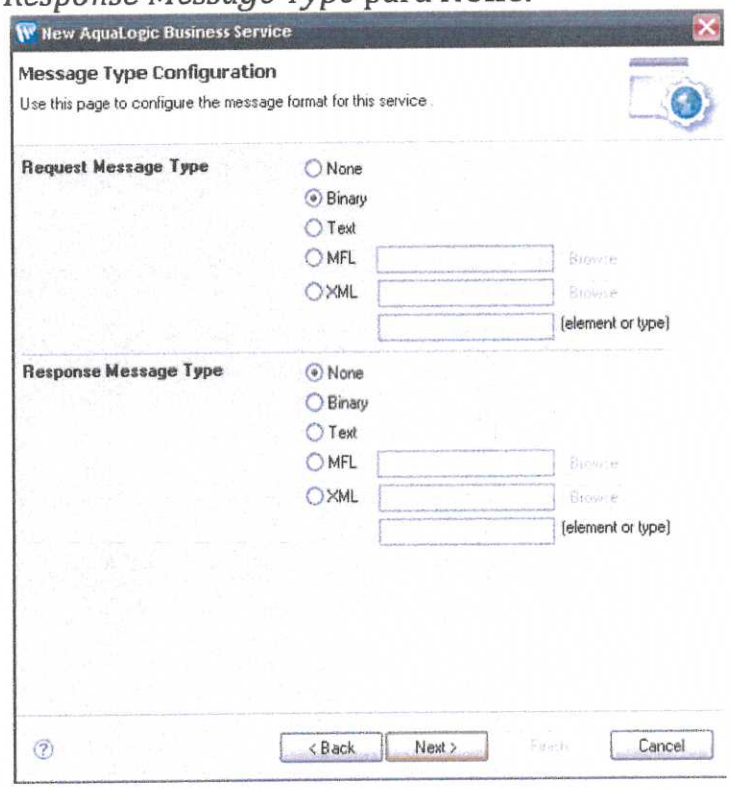
Definindo o Business Service:

Neste ponto do exercício você irá definir um *Business Service* que representa o **BinaryMDB** que acabamos de criar.

Crie um Business Service chamado BinaryBusinessService e selecione como tipo de Service Type um Message Service.

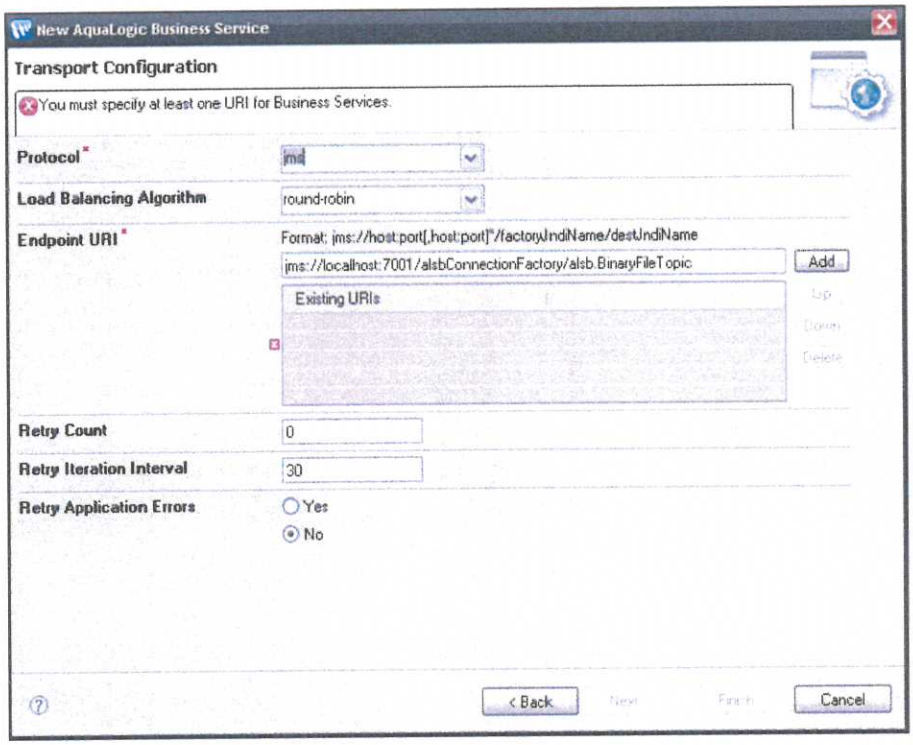


Clique em Next e selecione *o Request Message Type* para **Binary** e o *Response Message Type* para **None.**

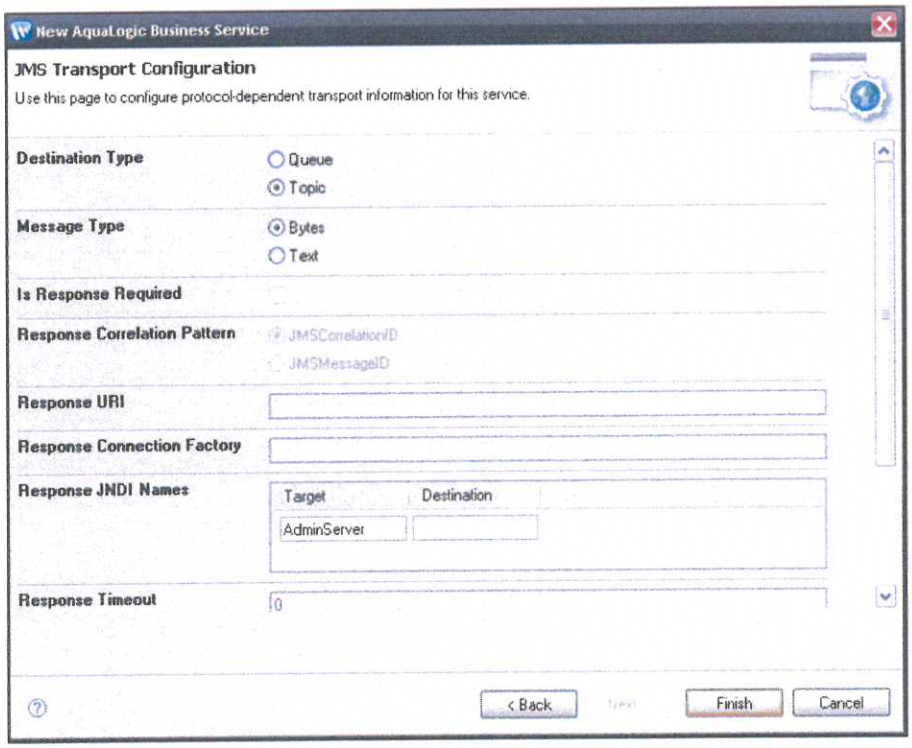


Clique em Next para continuar e sete o Endpoint URI com os nomes JNDI da sua Connection Factory e JMS Topic.

```
jms://localhost:7001/OSBConnectionFactory/OSB.BinaryFileTopic
```



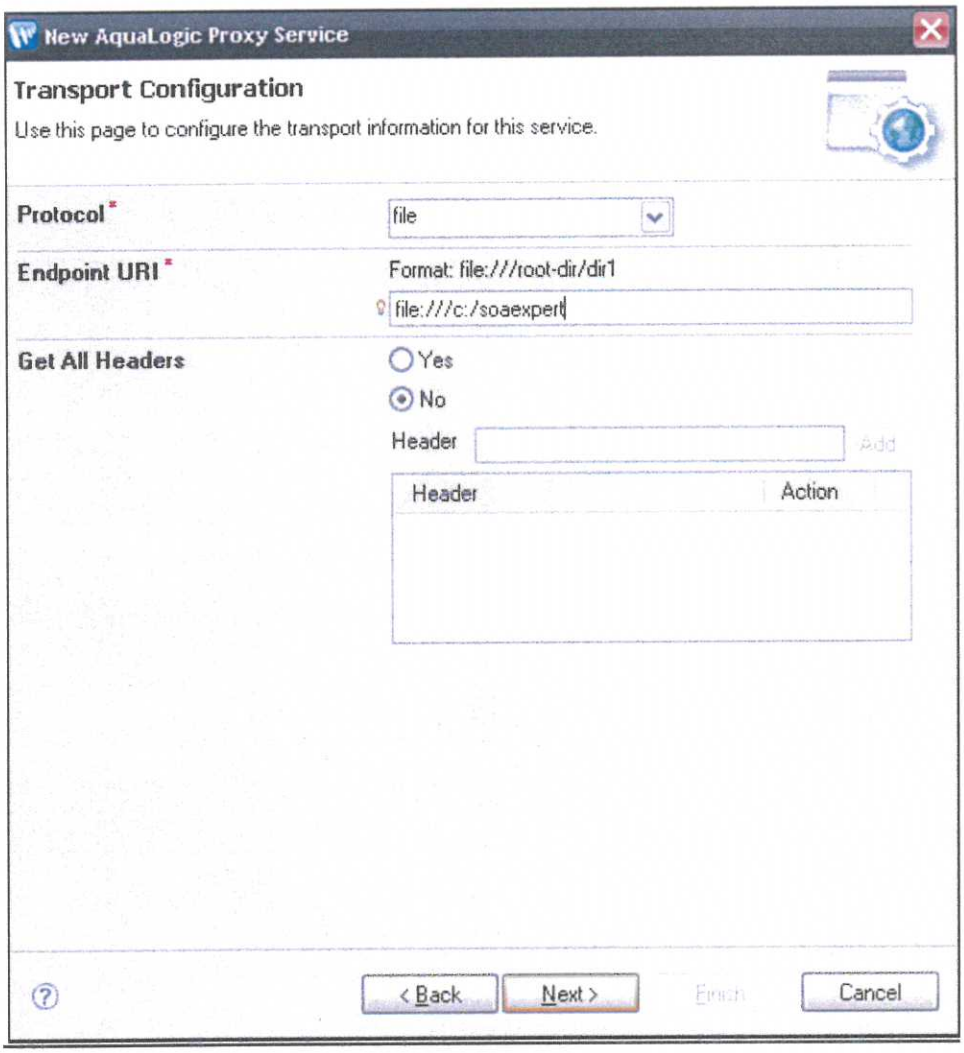
Por fim, escolha o Destination Type para Topic.



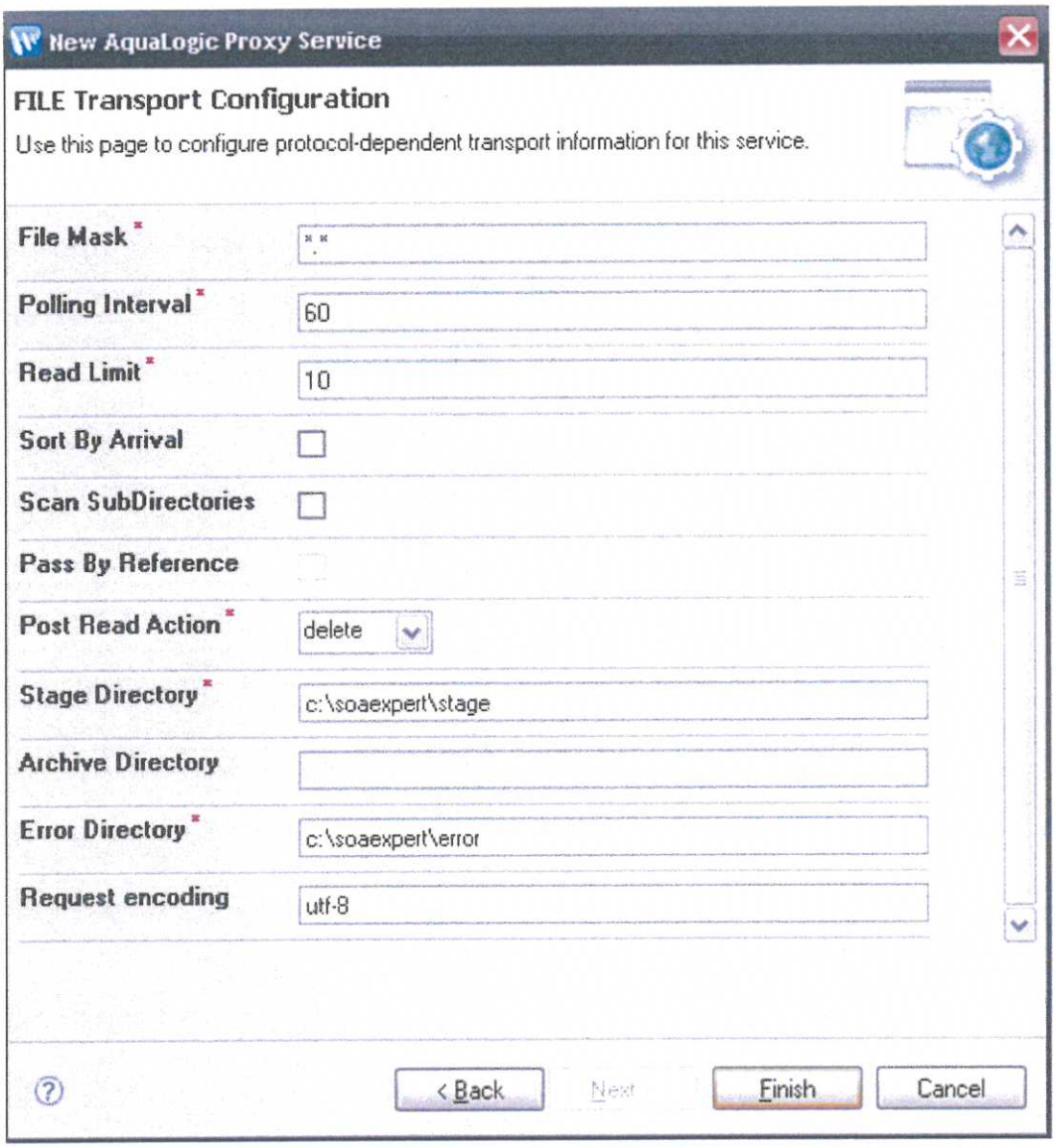
Criando o Proxy Service:

O Proxy Service que iremos criar segue o padrão do Business Service, o qual também será escolhido Messaging Service como service type e manterá os mesmos padrões para Request e Response. A diferença estará na definição do protocolo de transporte, que será **File**, setando a URI para :

```
file:///c:/soaexpert
```



Configurando o restante:



Para testar o exemplo, faça o deploy do MDB num EJB.jar e da configuração do OSB.

Copie um arquivo para o diretório c:\soaexpert e verifique as informações no console, após 60 segundos.

Message Type : Text

Mensagens de texto são úteis quando precisamos fazer o intercâmbio de estrturas não XML, como arquivos textos ou separados por valores – comma.

Message Type XML :

Basicamente não há diferenças entre o padrão XML como WSDL. Este tipo é utilizado basicamente em duas situações, quando há necessidade de definir um schema da mensagem em XML ou quando é necessário um XML como Request ou Response mas não ambos ao mesmo tempo.

Capítulo 8– Transporte EJB

Na maior parte das companhias, os EJBs acabaram se tornando software legado, por estas não possuírem uma cultura de padrões de serviço.

Raramente encontramos cenários onde Design Patterns como ***Façades***, são empregados a fim de encapsular a complexidade técnica e modularizar uma visão mais SOA.

Lembramos que **SessionBeans Statless** são os únicos tipos permitidos dentro do produto, e Entities Beans, que contém uma abstração da fonte de informações - dados, deverá ser feito um refactoring com uma abordagem DataServices – SDO.

EJBs também tratam de negócios de uma maneira mais física, logo o *Proxy Service* emprega um grande nível de abstração, agindo como um *adapter* para EJBs.

Para exemplificar o funcionamento de um EJB através do OSB, iremos montar um cenário onde um SessionBean calculará taxas de impostos de acordo com um preço enviado.

O exercício segue os seguintes passos:

1- Criar e fazer o deploy de um EJB SessionBean Statless no WLS.

2- Criar um Business Service que sabe como invocar o EJB.

3- Criar um Proxy Service que retorna a taxa de acordo com o preço enviado.

4- Criar um fluxo de mensagem para rotear a mensagem do Proxy para o EJB, e retornar a taxa calculada através do proxy, a fim de expor o serviço às aplicações externas.

1- Utilize o código referência de exemplo para construir seu SessionBean:

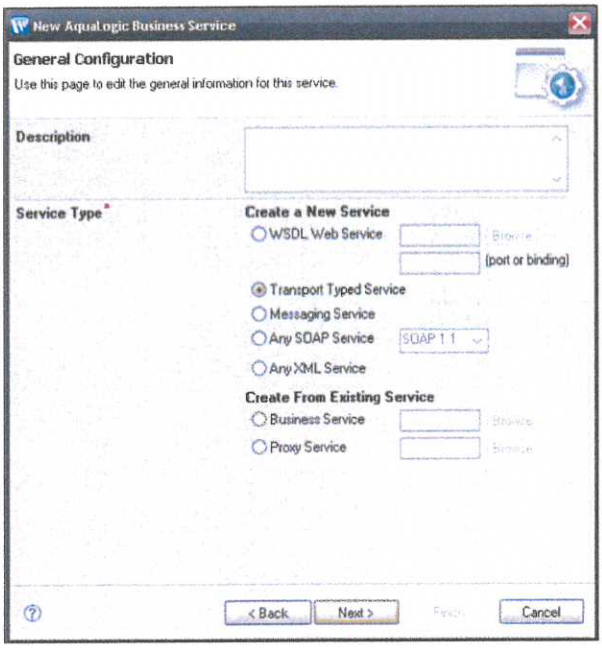
```
package br.com.soaexpert.services;

import javax.ejb.SessionBean;

@Session(ejbName = "EJBBusiness")
@JndiName(remote = "ejb.EJBBusinessRemoteHome", local = "ejb.EJBBusinessLocalHome")
@FileGeneration(remoteClass = Constants.Bool.TRUE, remoteHome = Constants.Bool.TRUE, localClass = Constants.Bool.TRUE,
public class EJBBusiness extends GenericSessionBean implements SessionBean {
    private static final long serialVersionUID = 1L;

    public void ejbCreate() {
    }

    @RemoteMethod(transactionAttribute=Constants.TransactionAttribute.SUPPORTS, isIdempotent = Constants.Bool.TRUE)
    public double calculaTaxa(double precoProduto) {
        return (precoProduto * 0.08d) + precoProduto;
    }
}//The End
```

2- O OSB possui um tipo de recurso chamado JAR file. Os JARs foram incrementados ao produto à partir da versão 2.5, exatamente para que se possa realizar chamadas a EJBs.

Como primeiro passo, adicione o client JAR como resource para que seu business service possa fazer referência.

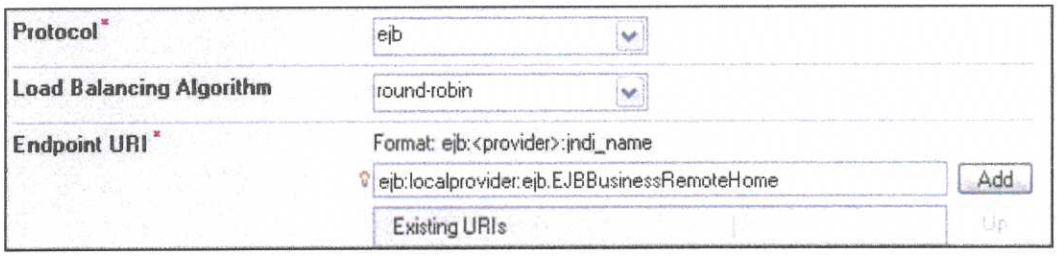
Crie uma nova pasta chamada JARs. Essa pasta conterá o client do seu EJB. Agora dentro da pasta adicione um JAR resource

Na pasta Business Service, crie um business service com o nome de EJBBusiness e escolha **Transport Typed Services.**



Selecione o protocolo ejb, com o seguinte **Endpoint**:

ejb:localProvider:ejb.EJBBusinessRemoteHome



Clique em Next para continuar e na área Client Jar, localize o recurso JAR File, que você importou para o projeto.

Service Account [ ] Browse
Supports Transaction [x]
Client Jar* SOAExpertIDE/Interface/EJB/EJBBAse Browse
Converter Jar [ ] Browse
Home Interface* br.com.soaexpert.services.EJBBusinessRemoteHome
Remote Interface* br.com.soaexpert.services.EJBBusinessRemote
TargetNamespace* http://www.openuri.
Style (•) Document Wrapped ( ) RPC
Encoding (•) Literal ( ) Encoded
Methods [x] calculaTaxa
Operation calculaTaxa
Parameters

Type	Name	Converter
double	arg0	

Return

Type	Name	Converter
double	return	

Aqui um ponto interessante, o *metadata* definido no arquivo JAR, preenche automaticamente os campos do *Wizzard*.

4-Criando um Proxy Service.

Para criar o Proxy Service você deve utilizar ao WSDL de exemplo :

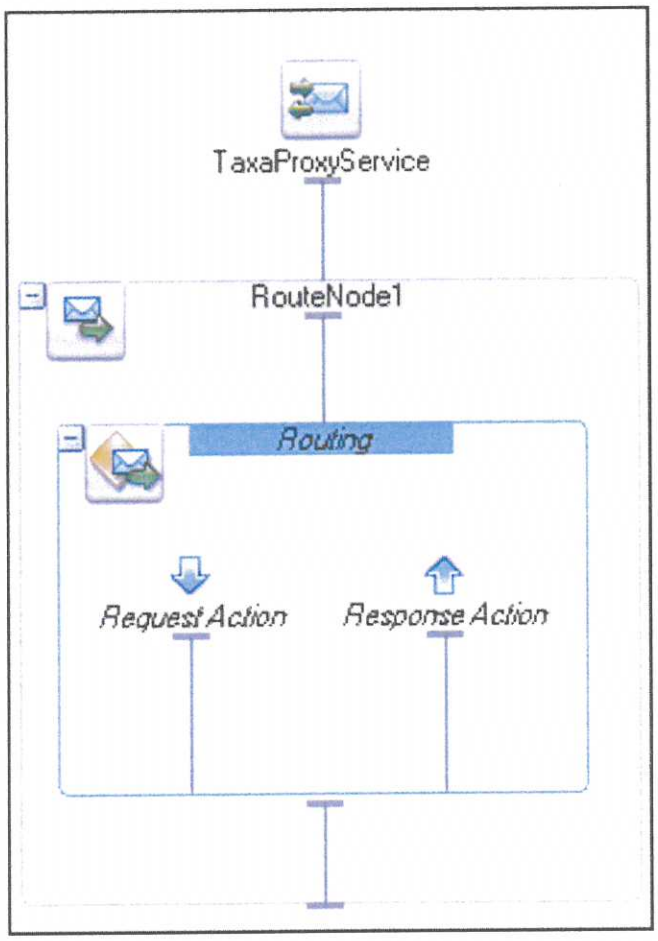
```
<?xml version='1.0' encoding='UTF-8'?>
<definitions name="EJBTaxaProxyService"
    targetNamespace="http://www.soaexpert.com.br/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:sxpt="http://www.soaexpert.com.br/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xs:schema attributeFormDefault="unqualified"
            elementFormDefault="qualified"
            targetNamespace="http://www.soaexpert.com.br/"
            xmlns:sxpt="http://www.soaexpert.com.br/"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:xs="http://www.w3.org/2001/XMLSchema">

            <xs:element name="calculaTaxaRequest">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="precoProduto"
                            type="xs:double" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="calculaTaxaResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="resultado" type="xs:double" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </types>

    <message name="calculaTaxaRequest">
        <part element="sxpt:calculaTaxaRequest" name="parameters" />
    </message>
    <message name="calculaTaxaResponse">
        <part element="sxpt:calculaTaxaResponse" name="parameters" />
    </message>
```

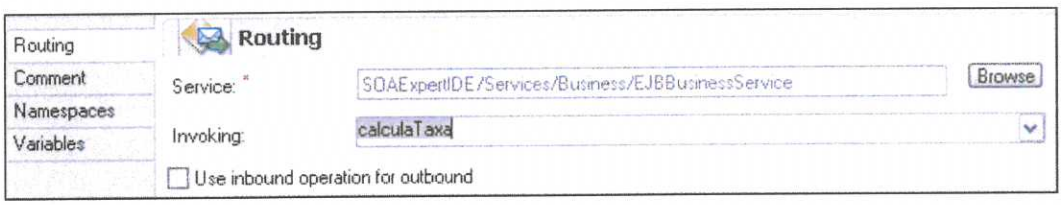
```
    <portType name="EJBTaxProxy">
        <operation name="calculaTaxa" parameterOrder="parameters">
            <input message="sxpt:calculaTaxaRequest" />
            <output message="sxpt:calculaTaxaResponse" />
        </operation>
    </portType>

    <binding name="EJBTaxProxySoapBinding" type="sxpt:EJBTaxProxy">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="calculaTaxa">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body parts="parameters" use="literal" />
            </input>
            <output>
                <soap:body parts="parameters" use="literal" />
            </output>
        </operation>
    </binding>

    <service name="calculaTaxaService">
        <port binding="sxpt:EJBTaxProxySoapBinding"
            name="EJBTaxSoapPort">
            <soap:address
                location="http://localhost:7001/SOAExpert/EJBTaxaProxyService" />
        </port>
    </service>
</definitions>
```

Uma vez selecionado o WSDL, clique em Next e selecione o protocolo HTTP. Configure o Endpoint da URI para **/EJBTaxaProxy** e clique em next e salve.

O último passo é criar um fluxo de mensagem para rotear os dados do Proxy para o BusinessService.

Abra o MessageFlow e crie uma Route. Edite a Route e configure para rotear ao EJBBusiness service, invocando a operação calculeTaxa.

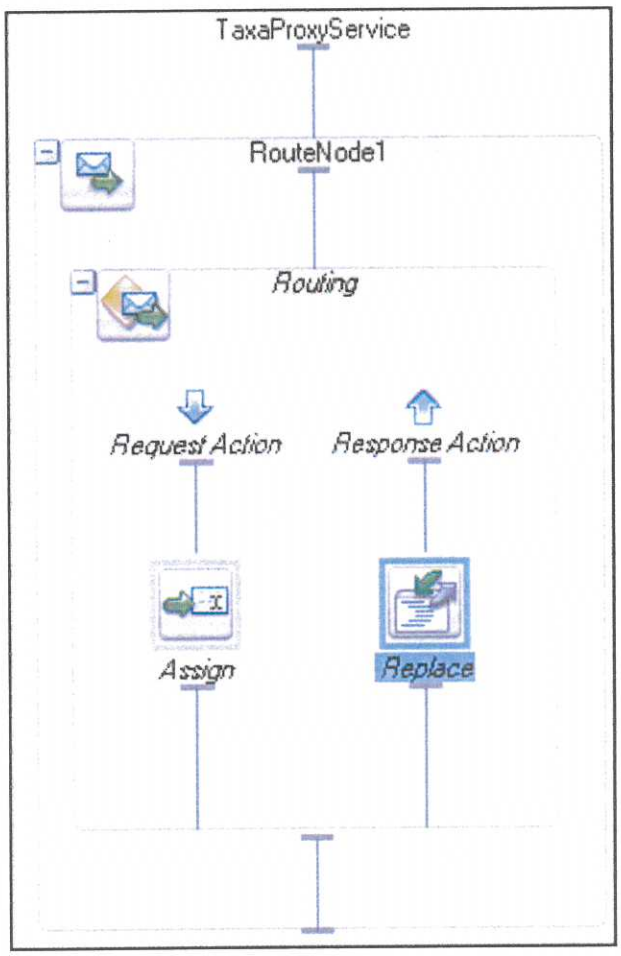




Crie uma ação Assign para o valor da expressão :

`$body/soa:calculaTaxaRequest/soa:precoProduto/text()`

para a variável **precoProdutoCorrigido**

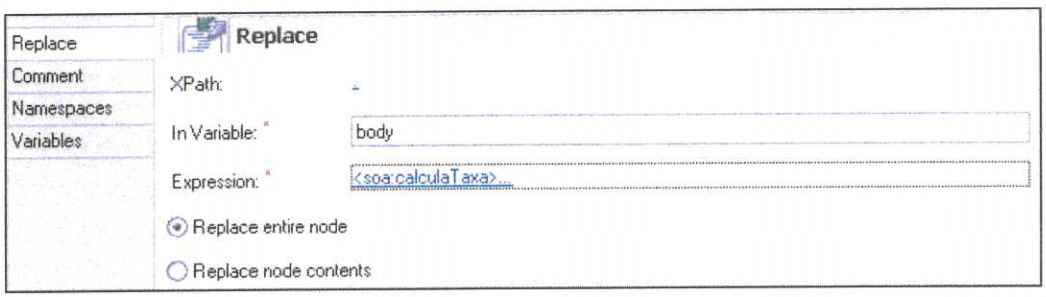


Configure a variável local como precoProduto e insira uma ação de Replace, incluindo o conteúdo de $body pela expressão:

<soa:calculaTaxa>

<soa:produto>{**$precoProdutoCorrigido**} </soa:produto>

</soa:calculaTaxa>



Exercícios Adicionais de Reforço:

Nesta etapa do curso, muitos conceitos foram passados, desde contratos, modelagem de domínios e sua contextualização utilizando a tecnologia XML – modelo canônico, tipos de invocação (Assíncrona e Síncrona) entre muitos outros, que serão explorados pelo projeto **SOA Life Cycle.**

Objetivo do exercício: Este será dividido em módulos de forma crescente, e ao fim do mesmo, teremos uma solução de processamento de Lotes Empresariais para aprovação de crédito.

Parte I – Criando o modelo de domínio da aplicação: *Canonical Data Model.*

Estrutura:

1 – Crie um novo projeto no **Eclipse – ESB**, do tipo *WebServices* em *WebLogic WebServices*, com o nome de **SOALifeCyle.**

2 – Crie as pastas para armazenar as estruturas como: contratos (WSDL), modelo canônico (XSD), na estrutura raíz do projeto.

- **Contracts**
- **Schemas**

Defina um **Domain Model:**

Namespace: www.soaexpert.com.br/domain ,

arquivo : DomainModel.xsd.

O projeto utilizará basicamente 4 tipos:

Credito, Empresa, LoteEmpresa e **Notafiscal** .

1- Empresa :

id	Integer
razaosocial	String
nomeFantasia	String
cnpj	String
telefone	Integern
notaFiscal	NotaFiscal [0...*]

2 – Credito

empresa	Empresa
serasa	boolean
spc	boolean
casasBahia	boolean
aprovado	boolean

3 - NotaFiscal

id	Integer
descricao	String
empresa	Empresa
valor	double

4 – LoteEmpresa

id	Integer
empresa	Empresa [1...*]

Parte II – Definindo contratos.

1. CreditoService.WSDL

- Namespace: http://www.soaexpert.com.br/CreditoService/

Operações Síncronas

- verificaCreditoSerasa - Request: in (empresa Element)
  - Response: out (credito Element)
- verificaCreditoSPC - Request: in (empresa Element)
  - Repsonse: out (credito Element)
- verficaCasasBahia - Request: in (empresa Element)
  - Response: out (credito Element)

Após o contrato finalizado, gere a implementação Java, clicando com o botão direito em cim do arquivo, escolha a opção WebServices – Generate.

OBS: Um código fonte Java referência está disponibilizado para a implementação.

2. EmpresaService.WSDL

O contrato EmpresaService.WSDL, será disponibilizado junto com o laboratório da aplicação, importe-o devidamente na estrutura adequada (contratos).

Repare nas diferenças para as chamadas assíncronas, vez que o elemento "out" – retorno é inexistente.

Parte III – Criando um projeto no OSB

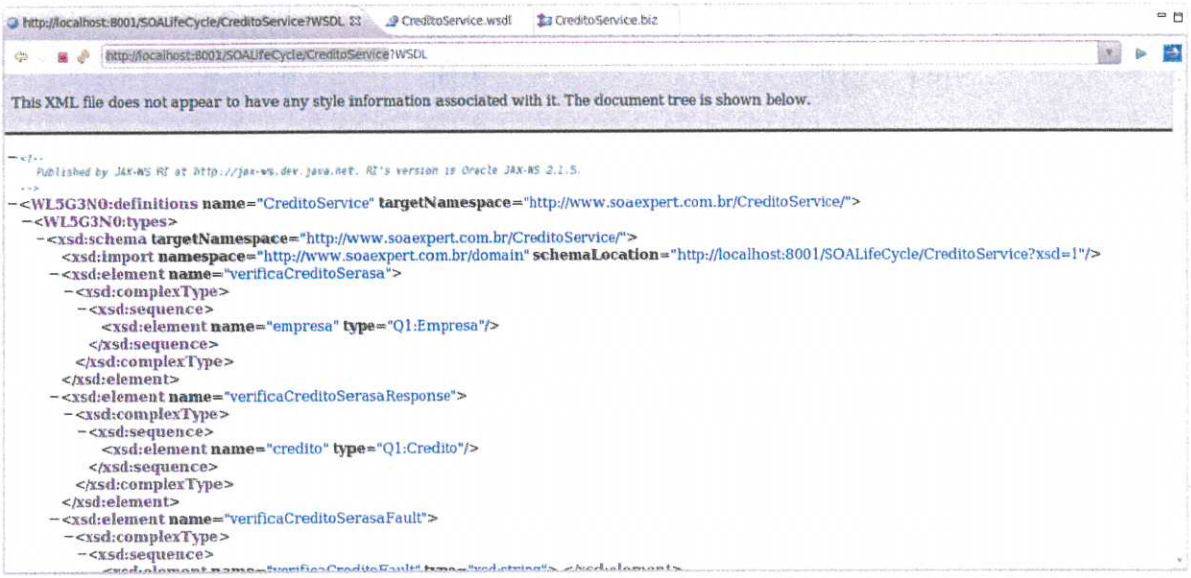
1. Crie um projeto novo do tipo **configuração OSB,** nomeado como **OSBSOALifeCycleConf.** Após esta etapa, crie um novo **projeto de ESB** chamado **SOALifeCycleESB,** que utiliza a configuração criada anteriormente.

2. Crie a Taxonomia de pastas padrão dentro do ambiente de desenvolvimento:

- Interfaces
  - Contracts
  - Schemas
- Resources
  - Transformations
- Services
  - Business
  - Proxies

Parte IV – Business Service: CreditoService.biz

1. Crie um Business Service, **CreditoService**, à partir do contrato *CreditoService.WSDL*



passo1

2. Na aba **Transport**, inclua o *endpoint* para sua implementação, exemplo: **http://localhost:8001/SOALifeCycle/CreditoService**

#dica, lembre-se de rodar sua implementação Java, a fim de buscar o endereço correto.

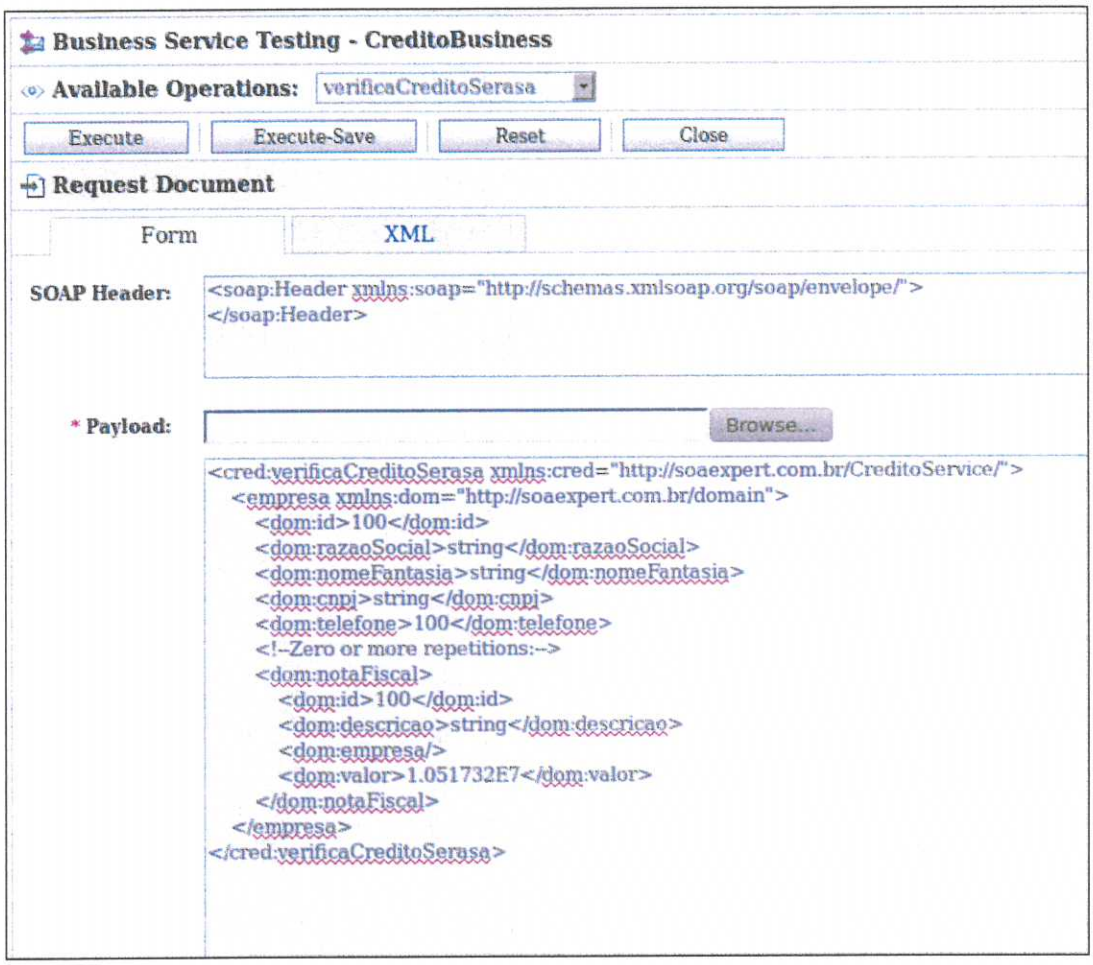
3. Na aba **HTTP Transport,** inclua os parâmetros de encoding :

- UTF-8.

Request Encoding	UTF-8
Response Encoding	UTF-8

passo2

3. Teste o **CreditoService** clicando com o botão direito e *Run As*, a fim de saber se o **OSB** está se comunicando perfeitamente com a implementação (endpoint) java.



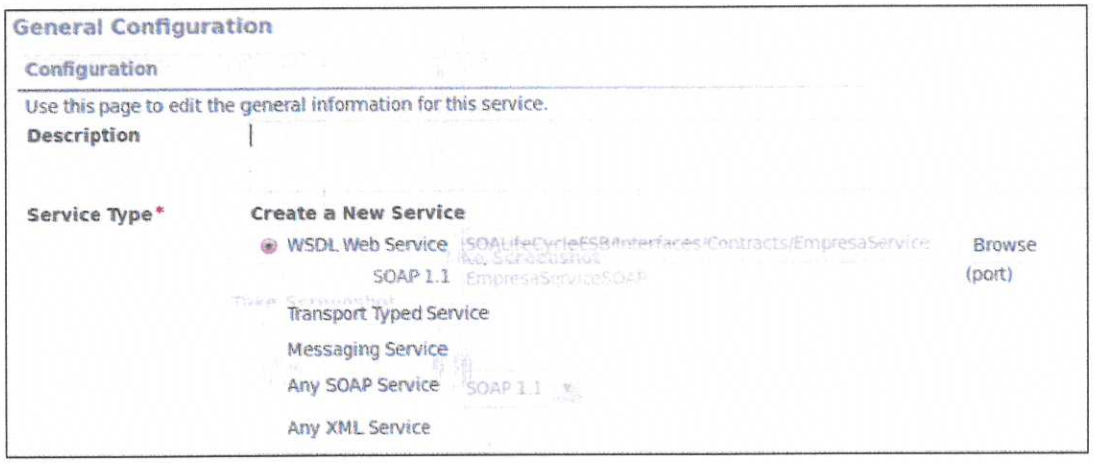
passo3

3 -Crie um ProxyService à partir do contrato de EmpresaService.WSDL, disponibilizado no lab do exercício.

4. Realize exatamente o mesmo processo para o contrato EmpresaService.wsdl e gere o **EmpresaBusiness.**

Parte V – Proxy Service: EmpresaProxy.proxy

1. Crie um proxy service à partir do contrato **EmpresaService.wsdl**

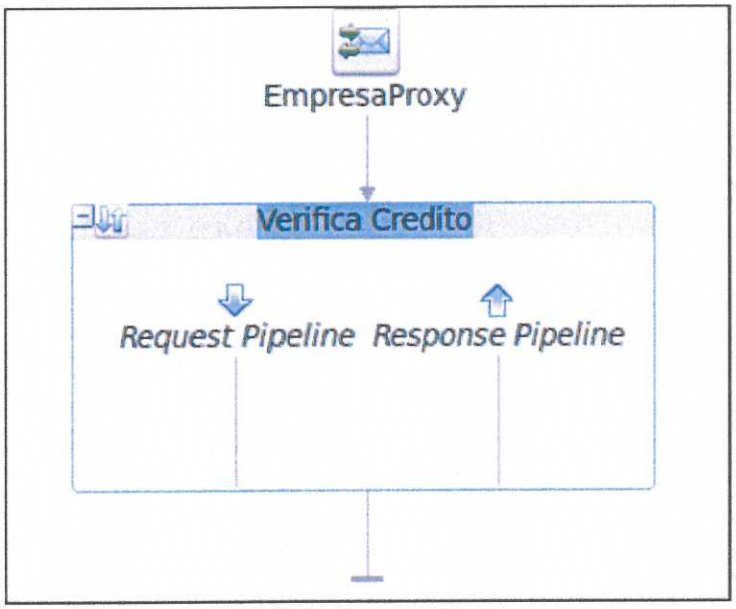


passo1.

2. Na aba **Transport,** selecione a opção: **Get All Headers – Yes**

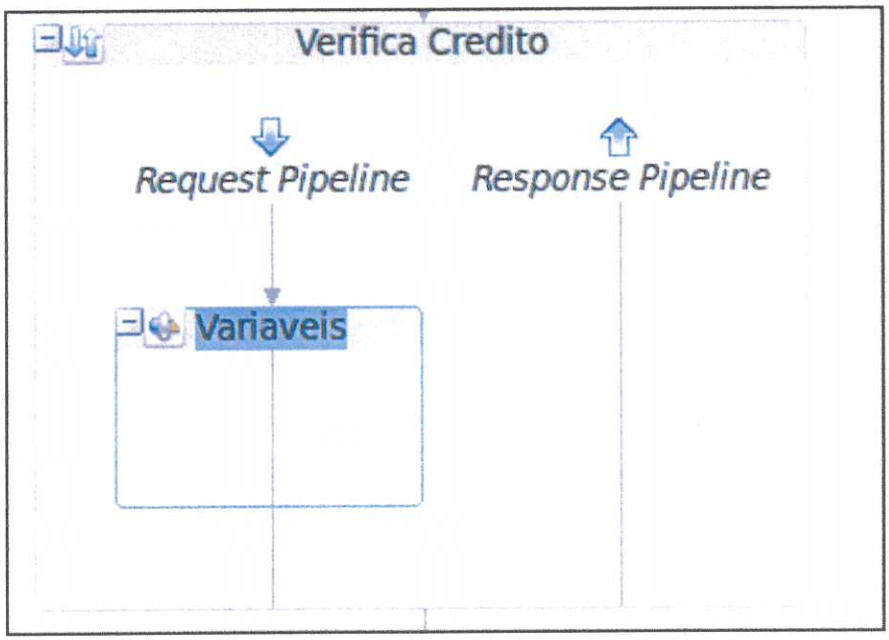
3. Na aba **HTTP Transport,** defina os encodings para Request e Response: UTF-8.

4. Na aba Message Flow, inicialize o fluxo de processamento de verfiicação de crédito, adicionando um *Pipeline Pair*, abaixo do start node chamado: **Verifica Crédito**



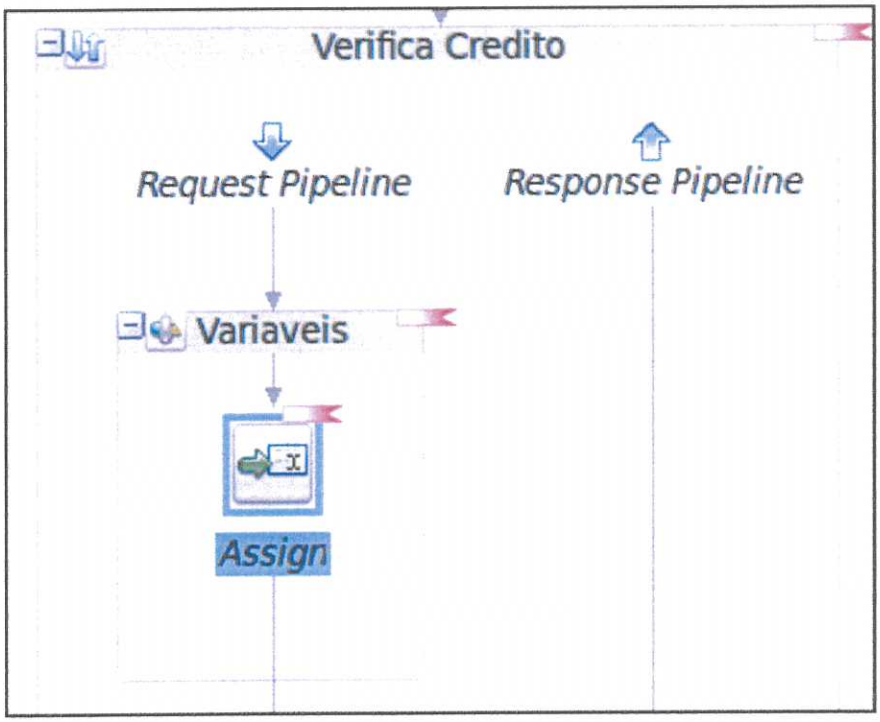
passo4.

5. Adicione um estágio chamado **Variáveis**, dentro do *Pipeline Pair.*



passo5

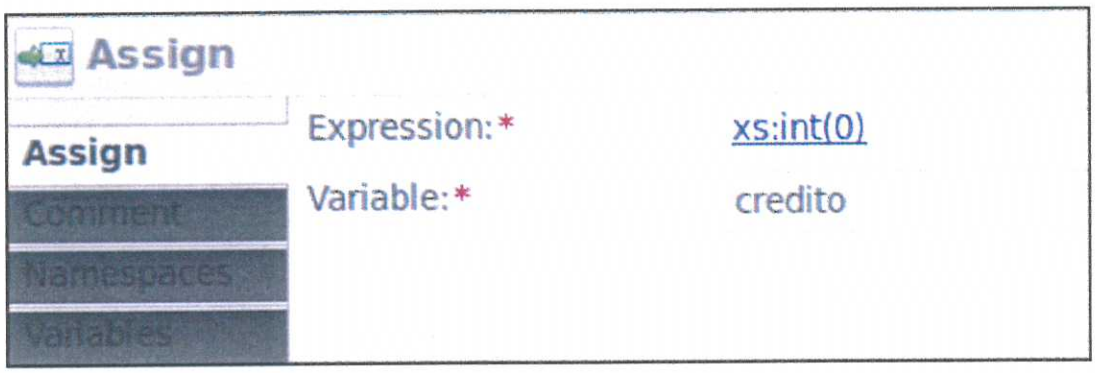
6. Adicione um **Assign** dentro do estágio Variáveis, para instanciarmos a variável **$credito** :



passo6

7. Declare a variável **$credito**, com um tipo construtor **INT** XPath:

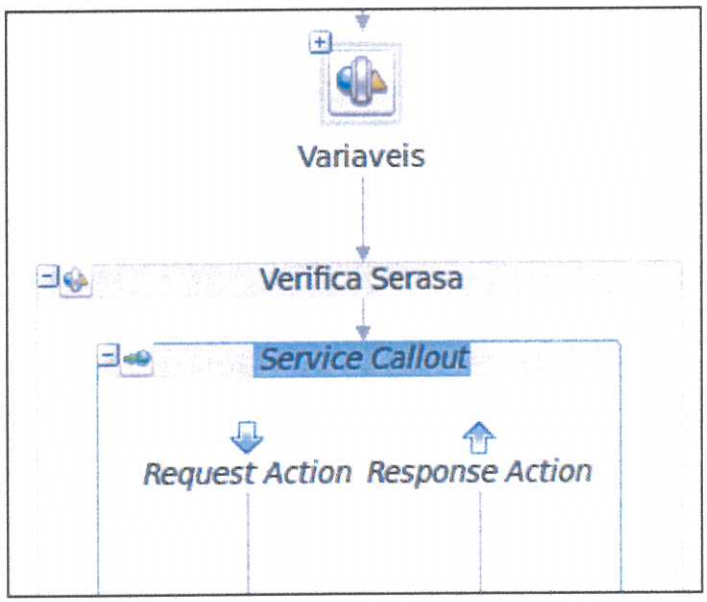
OBS: Essa variável é do tipo numérica, para operações matemáticas.



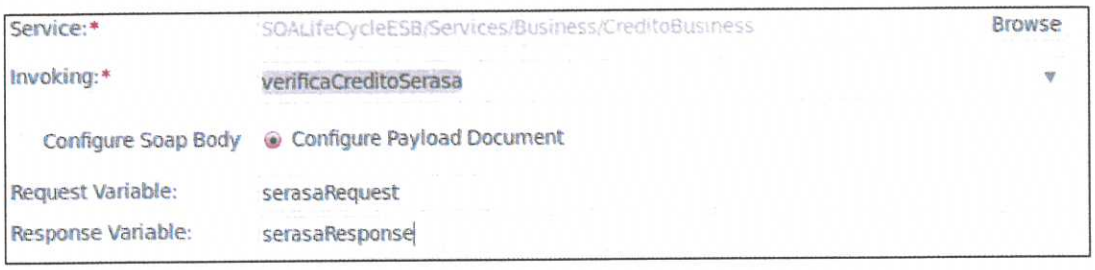
passo7

8. Adicione um segundo estágio dentro do *Pipeline-pair* chamado **Verifica Serasa.**

9. Dentro do estágio Verifica Serasa, adicione um **Service Callout,** invocando o serviço **CreditoBusiness e** a operação correspondente, verificaCreditoSerasa.

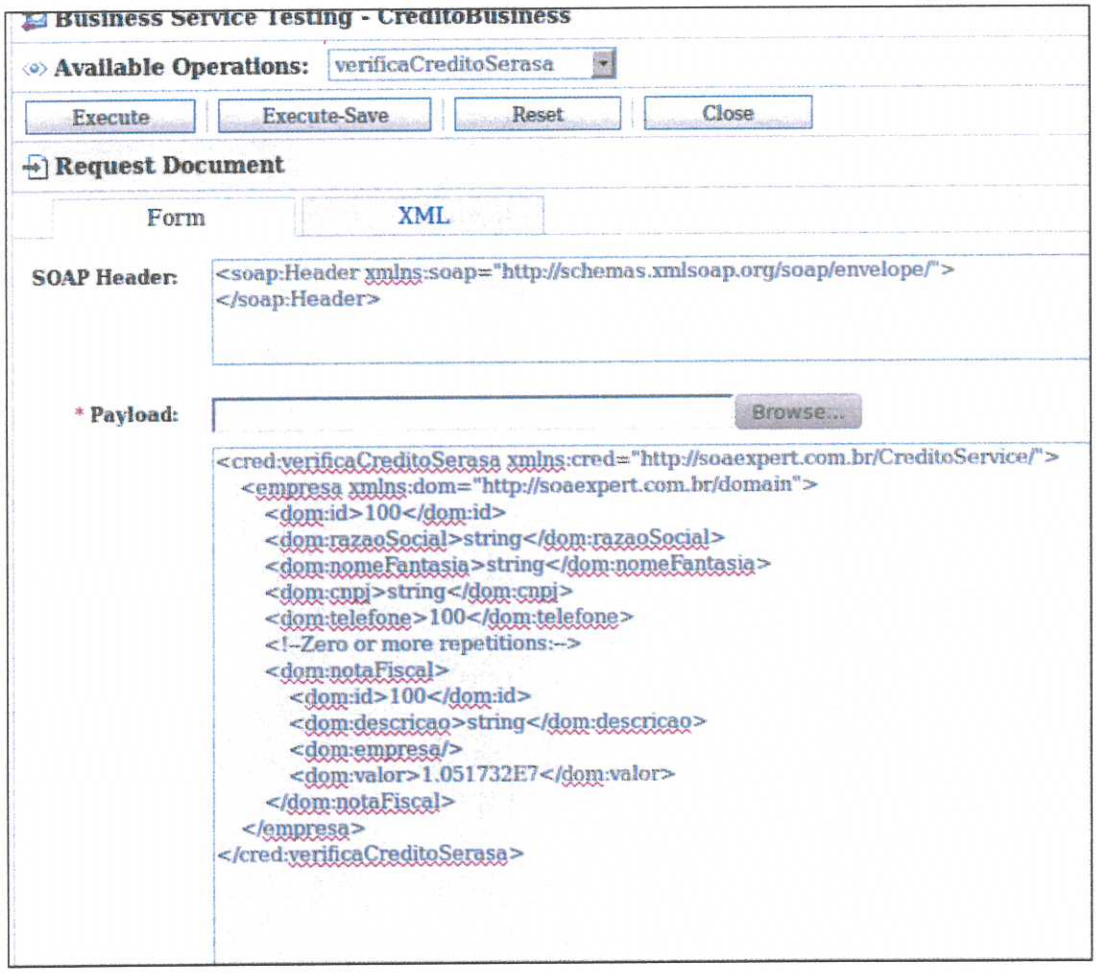


passo9a



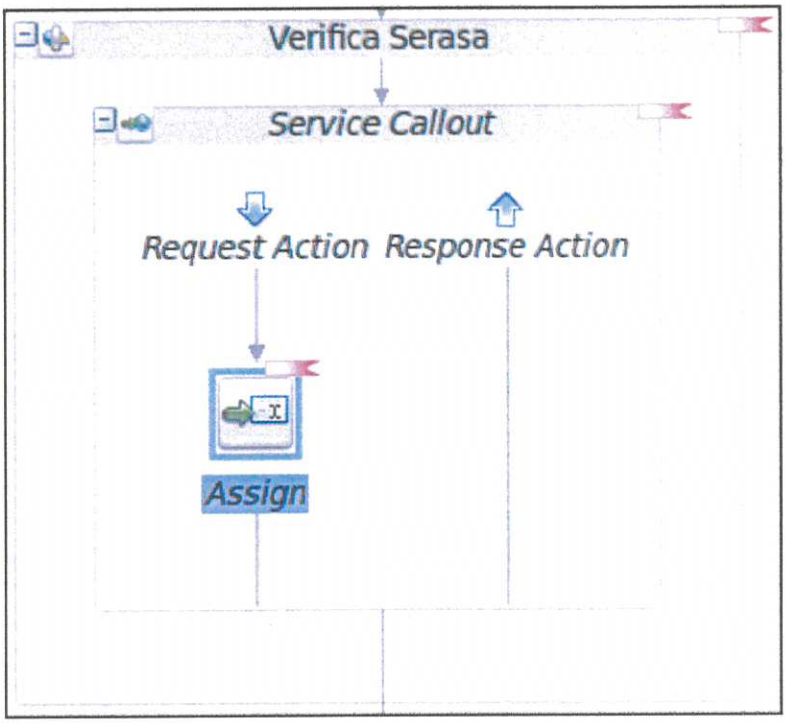
passo9b

10. Salve o projeto e execute o CreditoService Business, para que se possa copiar o payload de entrada a ser atribuído ao Request do **Service Callout.**

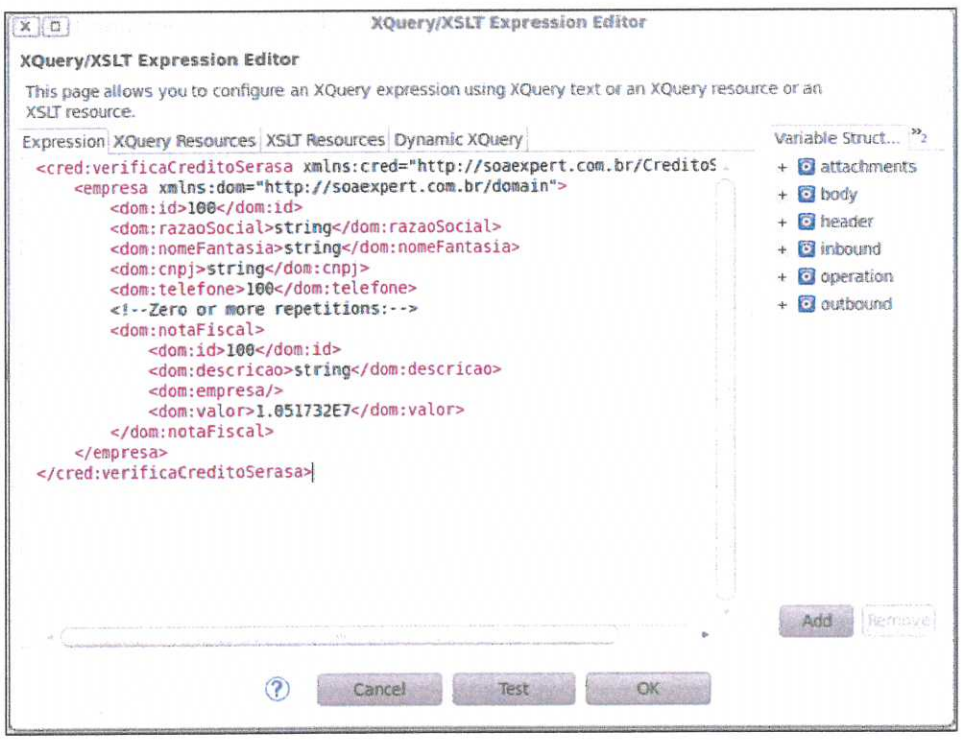


passo10

11. Atribua o payload de entrada à variável **serasaRequest** do *Service Callout*, fazendo o **Assign** do payload de entrada para mesma:



passo11a



passo 11b

OBS: Lembre-se de copiar os nós da estrutura do Request, para preencher os dados do payload de entrada e utilizar a função Xpath **text()**, a fim de obter somente o valor do nó.

```
<cred:verificaCreditoSerasa xmlns:cred="http://soaexpert.com.br/CreditoService/">
    <empresa xmlns:dom="http://soaexpert.com.br/domain">
        <dom:id>{$body/emp:verificaCadastro/empresa/dom:id/text()}</dom:id>
```

passo11c

Após a atribuição do payload:

Expression:*	<cred:verificaCreditoSerasa xmlns:cred="http://soaexpert.com.br/CreditoService/">
Variable:*	serasaRequest

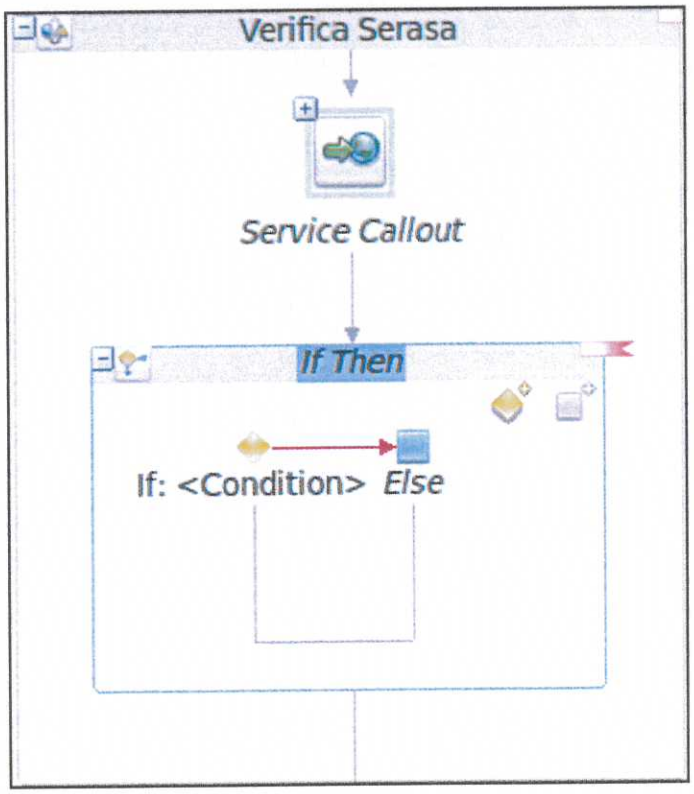
passo11d

12. Salve o projeto e teste o Proxy, para saber se o Service Callout está fazendo a chamada corretamente através do business service.

```
Invocation Trace
  (receiving request)
  Verifica Credito
    Variaveis
    Verifica Serasa
      Invoked Services
        Service Callout to: "CreditoBusiness"
      Message Context Changes
        added $serasaRequest
        added $serasaResponse
          <ns2:verificaCreditoSerasaResponse xmlns:ns3="http://soaexpert.
                                             /CreditoService/" xmlns:S="h
            <credito>
              <ns3:serasa>false</ns3:serasa>
              <ns3:spc>false</ns3:spc>
              <ns3:casasBahia>false</ns3:casasBahia>
            </credito>
          </ns2:verificaCreditoSerasaResponse>
        changed $body
        changed $inbound
  (echoing request)
  Verifica Credito
```

passo12

13. Adicione um **If Then (Flow Control)** ao estágio Verifica Serasa, abaixo do *Service Callout.*



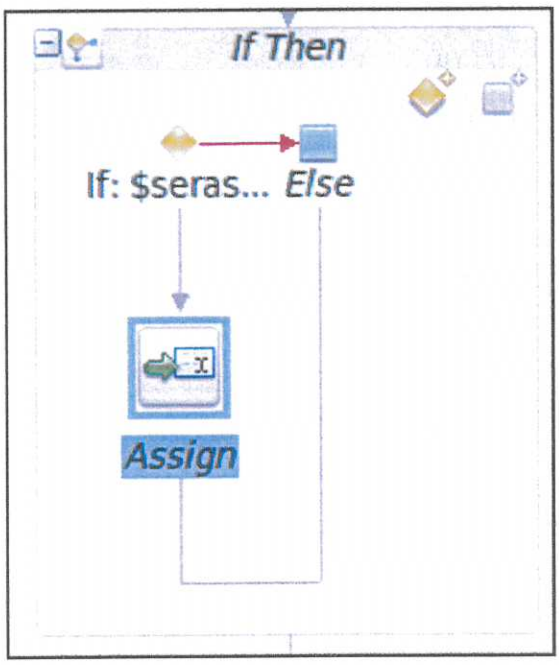
passo13

14. Clique em <Condition> para editar a condição, e no *condition builder*, inclua a expressão:

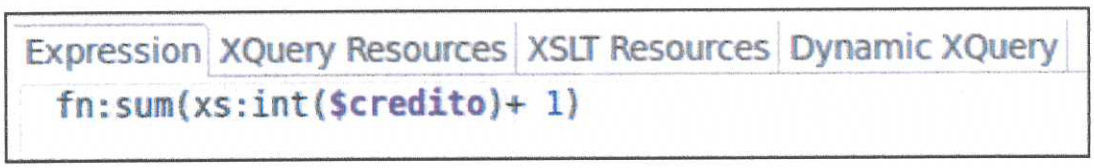
```
$serasaResponse/credito/dom:serasa/text() = "true"
```

passo14

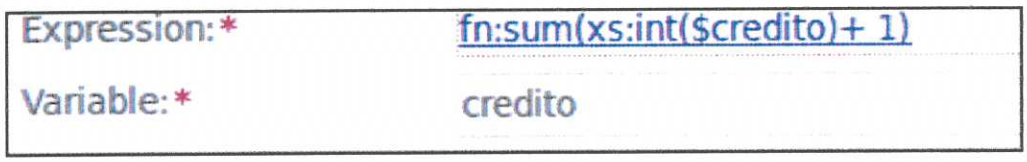
15. Na condição verdadeira, atribua 1 à variável de contagem **$credito.**



passo15a

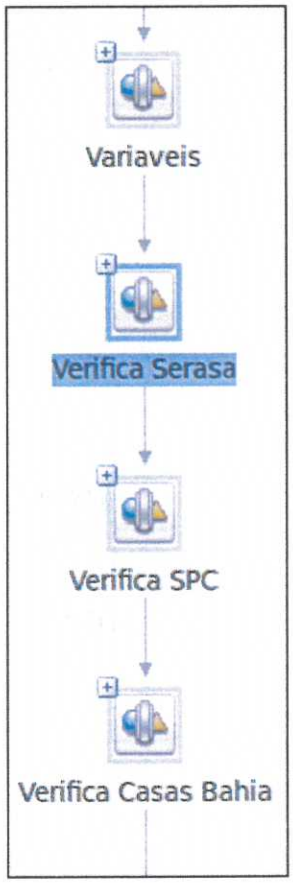


passo15b



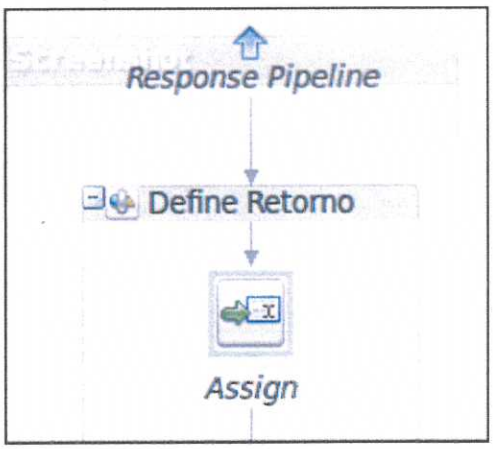
passo15c

16 - Repita os passos 8 à 15, para as operações similares : verificaSPC e verificaCasasBahia.



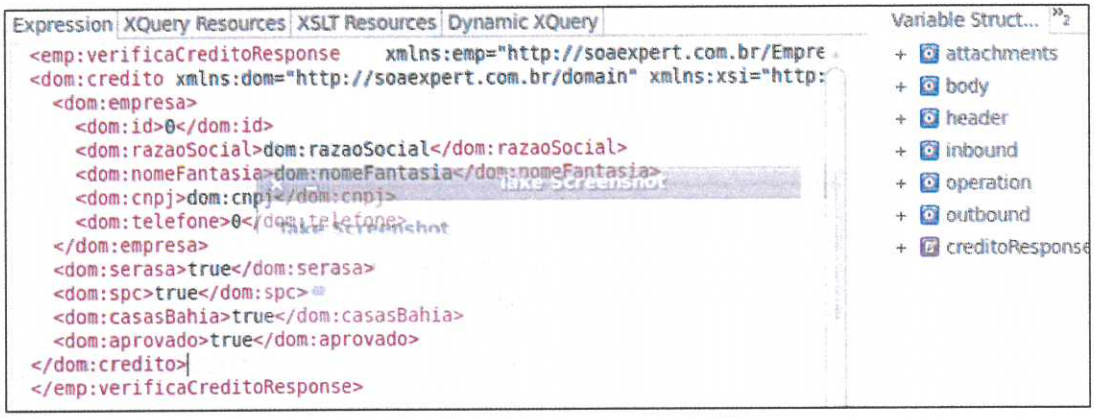
passo16

17. No Pipeline-pair do Verifica Crédito, no fluxo de Response, inclua um estágio: **Define Retorno**.



passo17

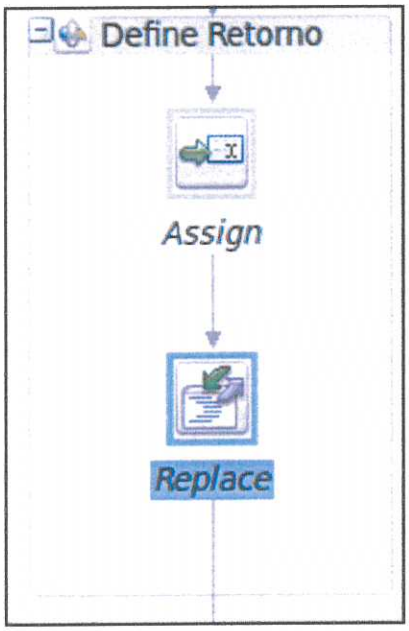
18. Dentro do Define Retorno, faça um Assign, contendo um payload de resposta de acordo com o contrato do Proxy Service, para a variável **$response.**



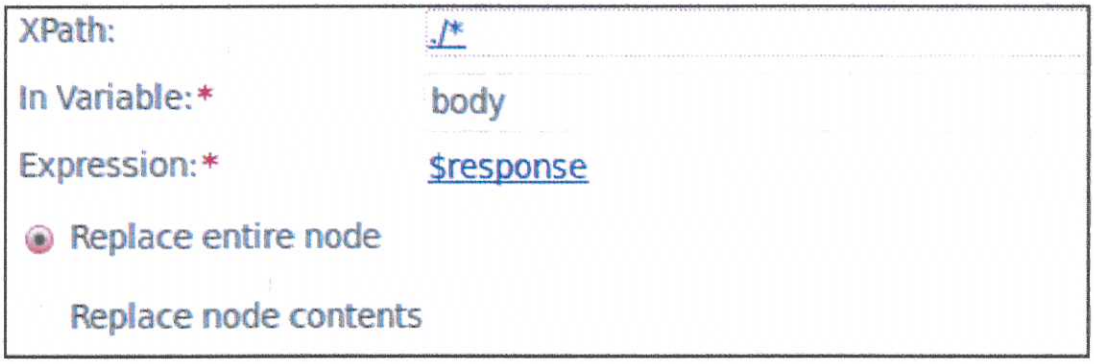
passo18

#Dica, para ficar mais simples a geração do XML de crédito, crie um elemento **credito** no Domain.xsd (schema) e gere o XML com o botão direito. Lembre-se de substituir o **Q1** por **dom** durante o wizard, namespace que estamos utilizando no ESB.

19. Inclua a manipulação Replace, para substituir o contexto **$body** que seria ecoado, pela variável de **$response.**



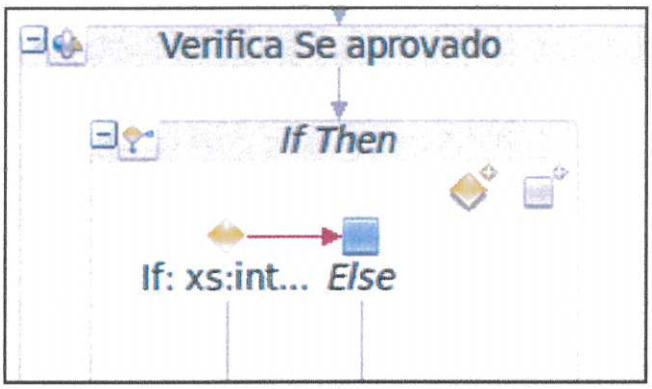
passo19a.



passo19b.

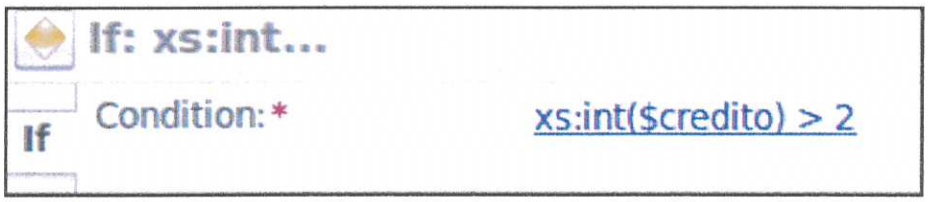
20. Inclua um estágio: **Verifica Se Aprovado**, abaixo do estágio Define Retorno.

21. Dentro do estágio Verifica Se Aprovado, coloque um controle de fluxo **If Then.**



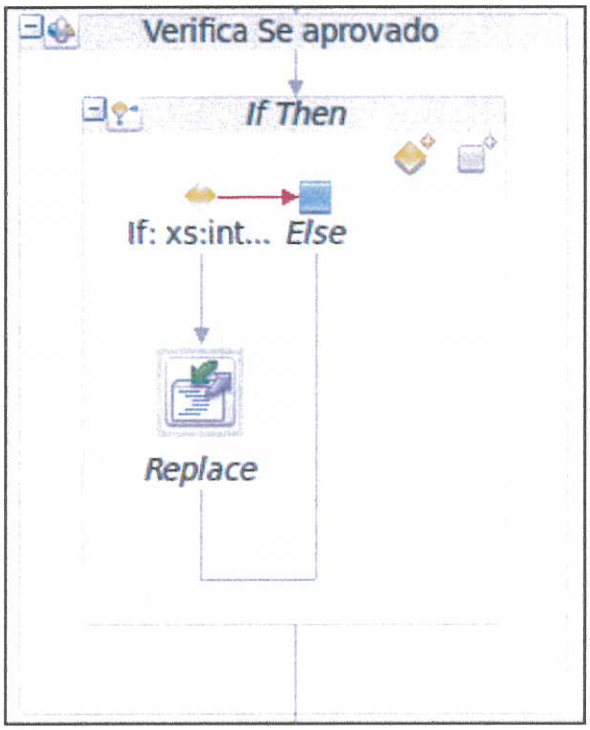
passo21.

22. Na condição, atribua a sentença:

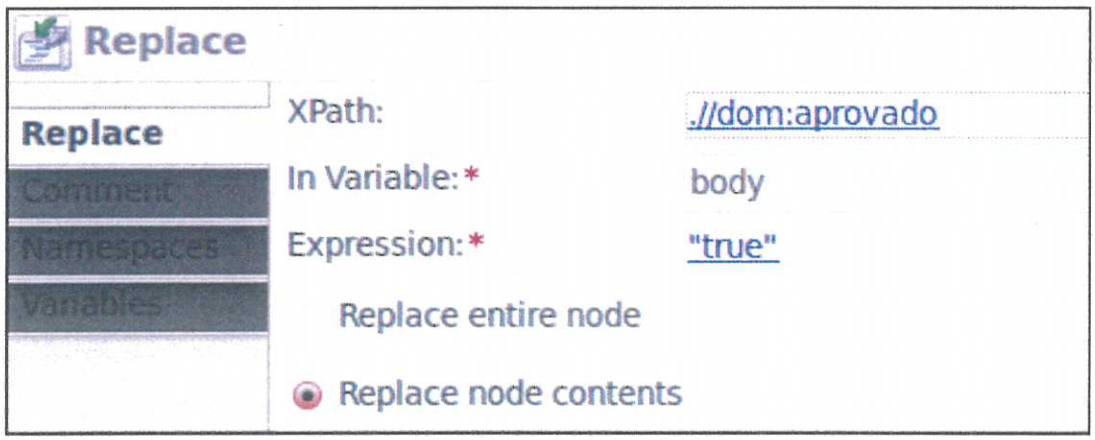


passo22.

23. Dentro da condição verdadeira, inclua um **Replace**, para manipular a variável de retorno, **$response**, com a sentença "true" para o elemento aprovado.



passo23a.

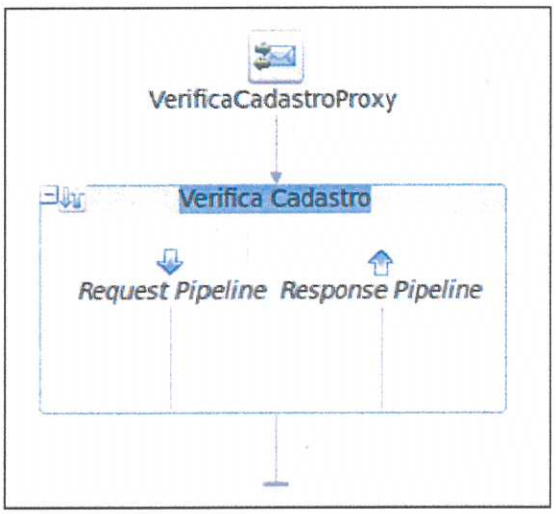


passo23b

Parte VI – Proxy Service: VerificaCadastroProxy.proxy

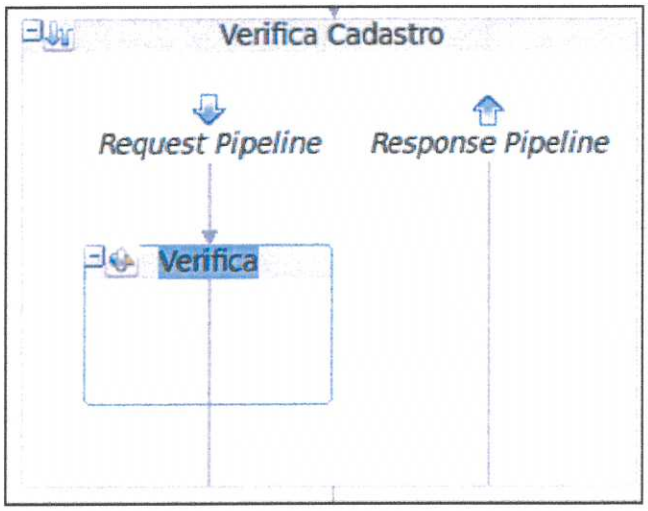
1. Crie o proxy à partir do contrato EmpresaService.wsdl, como no exercício anterior, e siga os mesmos passos até o *message flow*.

2. Inclua um *Pipeline-pair* **Verifica Cadastro.**



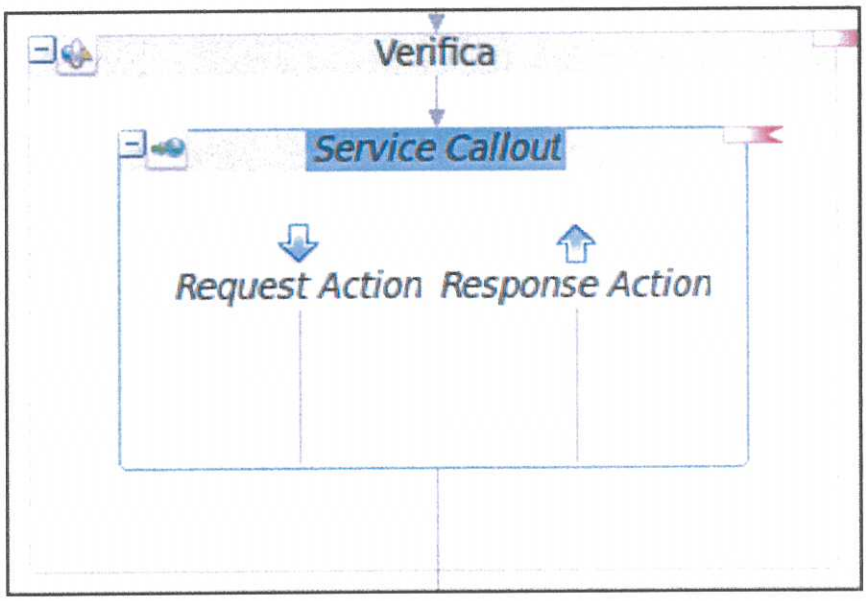
passo2.

3. Inclua um estágio chamado Verifica, no fluxo de Request do *Pipeline-pair.*



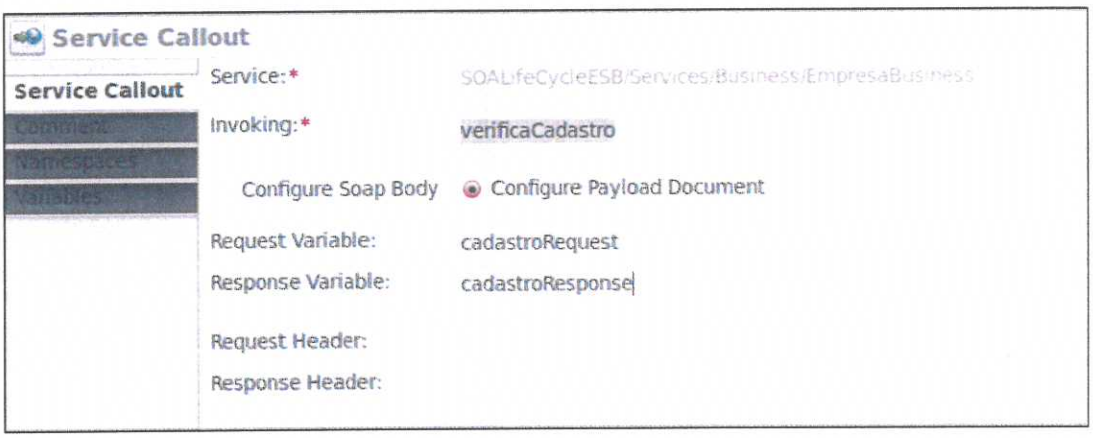
passo3.

4. Inclua um *Service Callout* para chamada Síncrona do serviço **EmpresaBusiness.**



passo4.

5. Escolha o Serviço EmpresaBusiness e invoque a operação **verificaCadastro.**



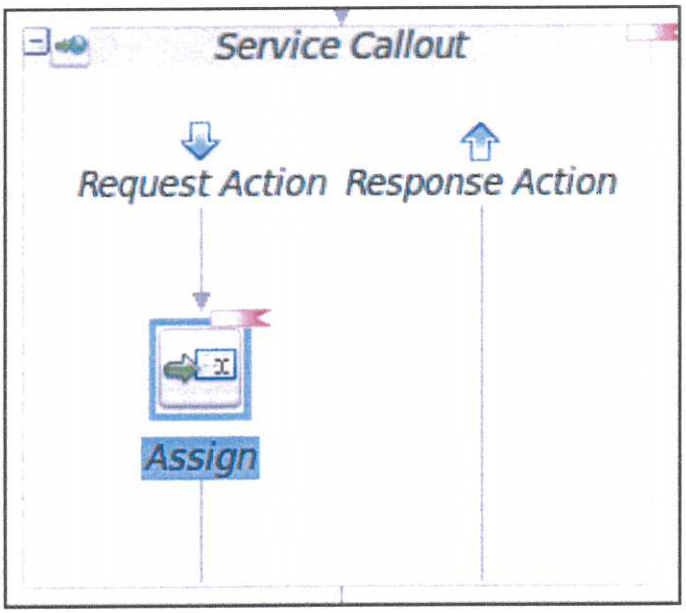
passo5.

6. Copie o Payload de entrada do Business Service, para ser utilizado no Request do Service Callout:

```
<emp:verificaCadastro xmlns:emp="http://soaexpert.com.br/EmpresaService/">
  <empresa xmlns:dom="http://soaexpert.com.br/domain">
    <dom:id>100</dom:id>
    <dom:razaoSocial>string</dom:razaoSocial>
    <dom:nomeFantasia>string</dom:nomeFantasia>
    <dom:cnpj>string</dom:cnpj>
    <dom:telefone>100</dom:telefone>
    <!--Zero or more repetitions:-->
    <dom:notaFiscal>
      <dom:id>100</dom:id>
      <dom:descricao>string</dom:descricao>
      <dom:empresa/>
      <dom:valor>1.051732E7</dom:valor>
    </dom:notaFiscal>
  </empresa>
</emp:verificaCadastro>
```

passo6.

7. Inclua um **Assign** para a manipulação do Request do *Service Callout.*



passo7

8. Lembre-se de trazer do Request, os dados para preencher o payload, de modo que o XML não fique com valores fixos:

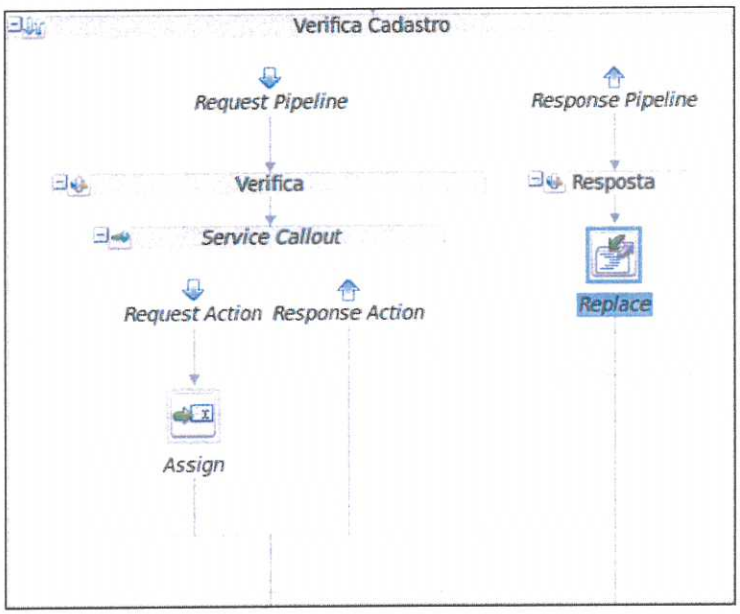
Expression | XQuery Resources | XSLT Resources | Dynamic XQuery

```
<emp:verificaCadastro xmlns:emp="http://soaexpert.com.br/EmpresaService/">
    <empresa xmlns:dom="http://soaexpert.com.br/domain">
        <dom:id>{$body/emp:verificaCadastro/empresa/dom:id/text()}</dom:id>
        <dom:razaoSocial>string</dom:razaoSocial>
        <dom:nomeFantasia>string</dom:nomeFantasia>
        <dom:cnpj>string</dom:cnpj>
        <dom:telefone>100</dom:telefone>
        <!--Zero or more repetitions:-->
        <dom:notaFiscal>
            <dom:id>100</dom:id>
            <dom:descricao>string</dom:descricao>
            <dom:empresa/>
            <dom:valor>1.051732E7</dom:valor>
        </dom:notaFiscal>
    </empresa>
</emp:verificaCadastro>
```

passo8.

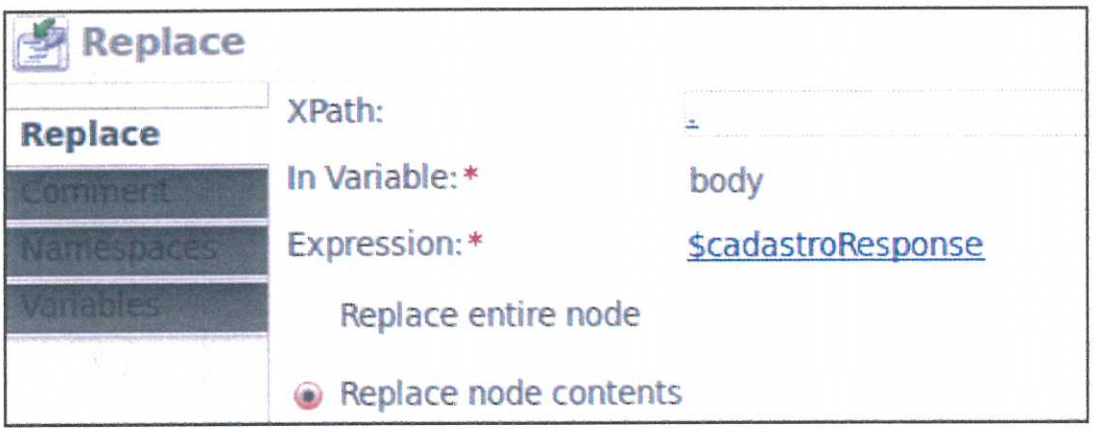
9. No fluxo de Response do Pipeline-pair, inclua um estágio de resposta chamado: **Resposta.**

10. Dentro do estágio de resposta, inclua um **Replace**, para manipulação do contexto de retorno.



passo10.

11. No Replace, substitua o conteúdo do contexto **$body** pelo conteúdo contido na variável **$cadastroResponse.**

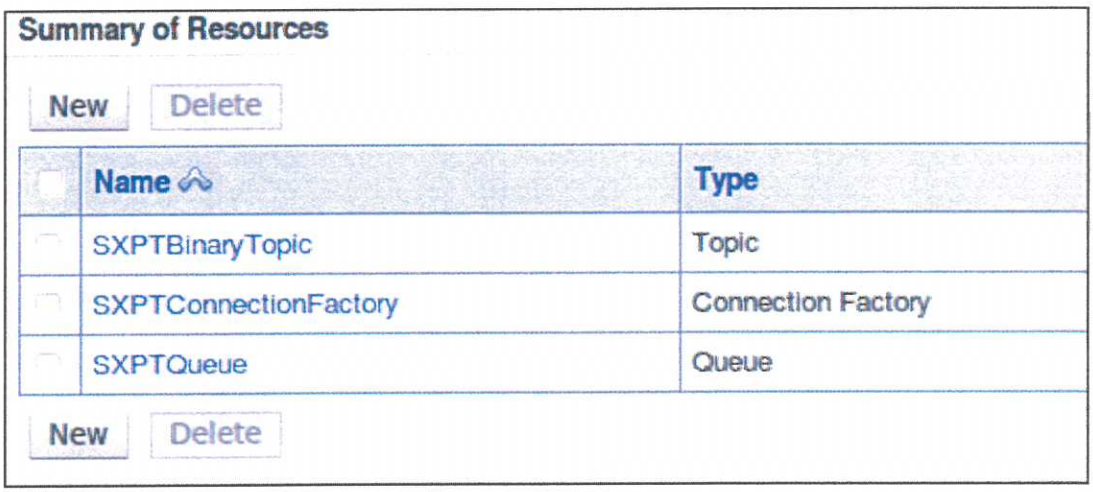


passo11.

12. Teste o Proxy , a fim de garantir o funcionamento do mesmo.

Parte VII – Filas para os serviços de mensageria Aprovado e Rejeitado.

1. Localize sua estrutura **JMSModule**, criado pelo capítulo 6.



passo1

2. Inclua uma nova fila (Queue) com o nome de **SOALifeCycleAprovado.**

*Name:	SOALifeCycleAprovado
JNDI Name:	SOALifeCycleAprovado

3. Crie a mesma estrutura para a fila: **SOALifeCycleRejeitado**

Parte VIII – Serviços Assíncronos Aprovado e Rejeitado.

1. Crie um Business Service, do tipo **Messaging Service.**

Service Type*	Create a New Service
	WSDL Web Service
	Transport Typed Service
	◉ Messaging Service
	Any SOAP Service SOAP 1.1
	Any XML Service

passo1.

2. Na aba Messaging, defina o tipo que será enviado ao Request, **Message Type**, escolhendo XML assim como o tipo complexo **Empresa.**

3. Na aba Transport, escolha o protocolo **jms** e defina na URI a ConnectionFactory criada no capítulo 6, mais o recurso Queue criado neste exercício.

Endpoint URI *	Format: jms://host:port(,host:port)*/FactoryJndiName/DestJndiName
	Existing URIs
	jms://localhost:8001/SXPTConnectionFactory/SOALifeCycleReprovado

OBS: A figura está apontando para a fila Reprovada, assuma a Aprovada para esse caso, e Reprovado para o exercício posterior.

4. Na aba **JMS Transport,** selecione o Queue como tipo destino.

5. Repita o exercício para a estrutra de Reprovado (Rejeitado).

Parte IX – Proxies Aprovado e Rejeitado.

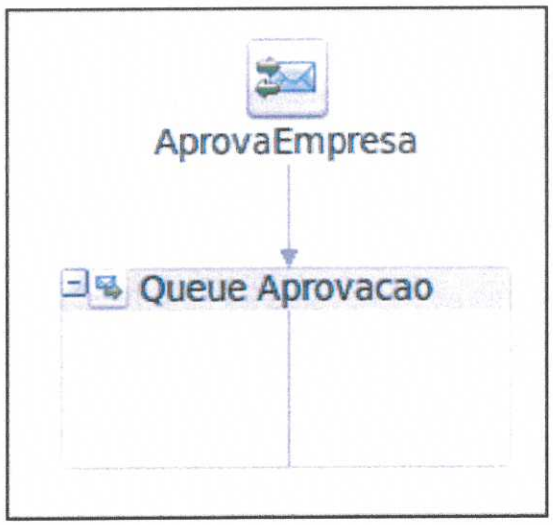
1.Crie o Aprovado Proxy à partir do contrato **EmpresaService.wsdl**

passo1

2. Na aba Transport – Get All Heders, selecione Yes.

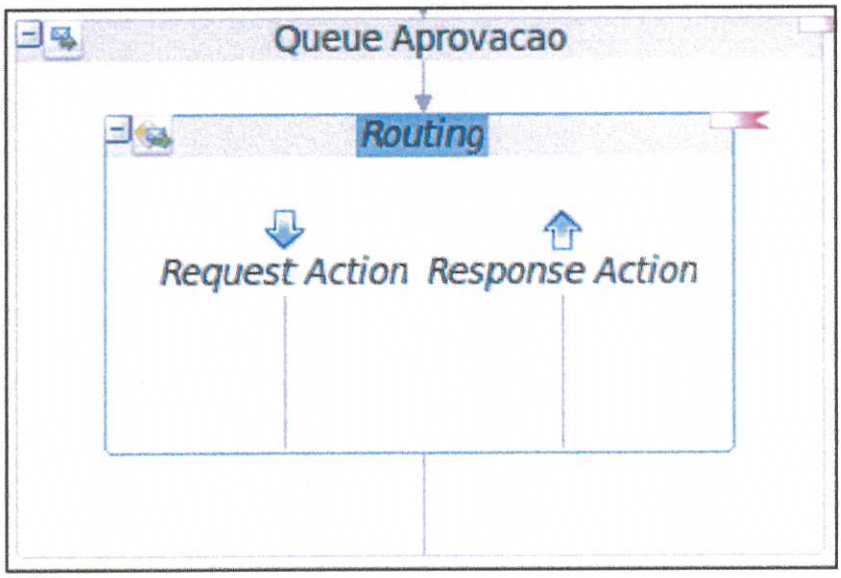
3. Siga o padrão default para todos os outros proxies criados até o momento.

4. Em MessageFlow, adicione uma **Route,** após o start node, e a nomeie Queue Aprovação ou Fila Aprovação.



passo4.

5. Adicione um **Routing,** dentro da rota Queue Aprovação.



passo5.

6. Nas propriedades do Routing, selecione o serviço Aprovado criado no exercício anterior.

7. Repita exatamente as mesmas operações para o Proxy de Rejeitado.

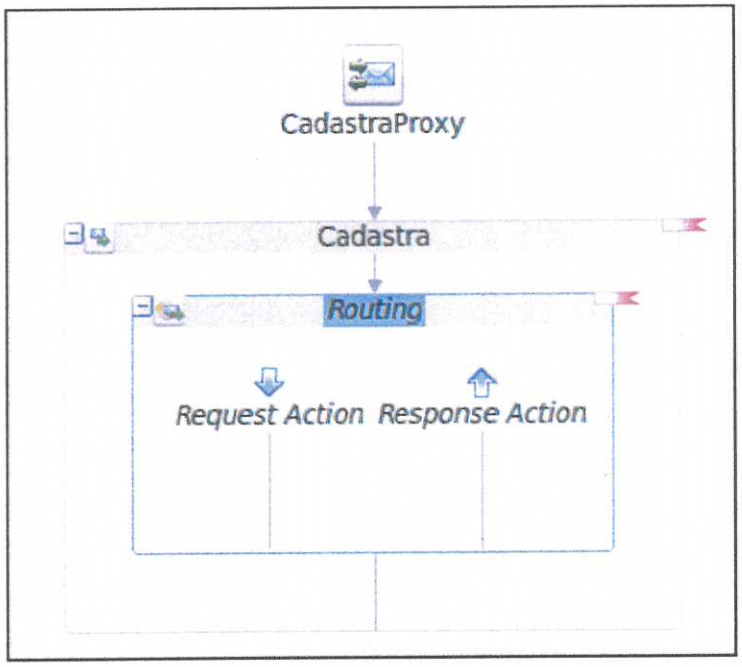
Parte X – Cadastra Proxy.

1. Crie um novo Proxy chamado: **CadastraProxy**, à partir do contrato do EmpresaService.wsdl.

2. Siga todos os passos que vimos até agora até o Message Flow.

3. Inclua um **Route,** após o Start node e o nomeie como Cadastra

4. Inclua um **Routing** dentro do Route.



passo4

5. Na aba de propriedades do Routing, selecione o serviço EmpresaBusiness e sua respectiva operação cadastra.

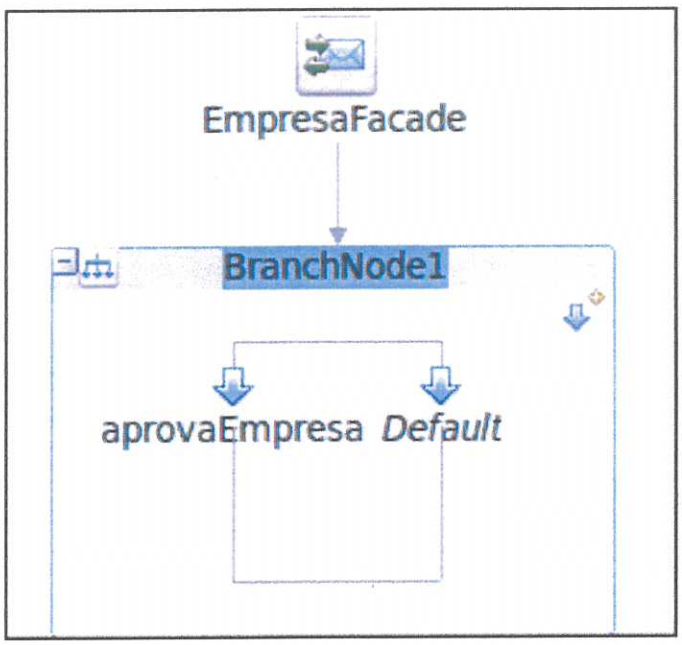
Parte XI – Agregação EmpresaFacade.proxy

A idéia dessa parte do exercício, é construir um serviço de fachada, que irá agregar todas as operações do contrato EmpresaService.wsdl, dispensando futuramente os proxies inerentes a esse contrato.

1. Crie um novo Proxy chamado **EmpresaFacade** com base no contrato EmpresaService.wsdl.

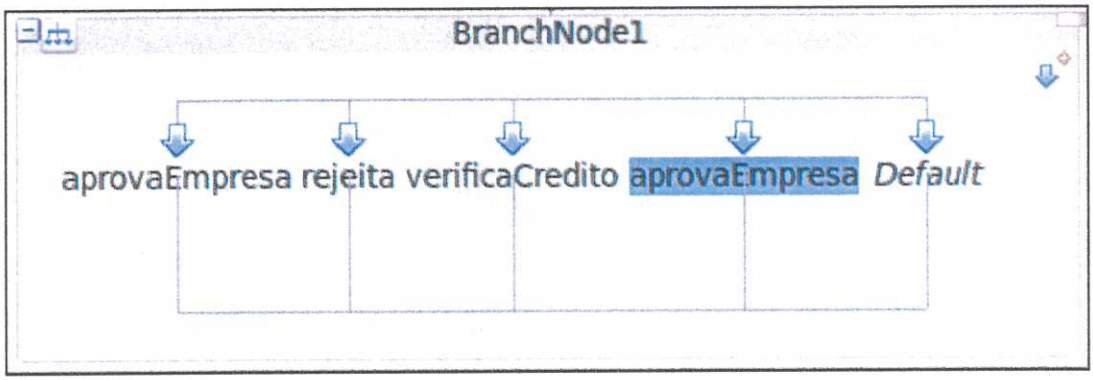
2. Siga os padrões de criação de proxies services, até a aba de MessageFlow.

3. Inclua um node **Operational Branch,** após o start node.



passo3.

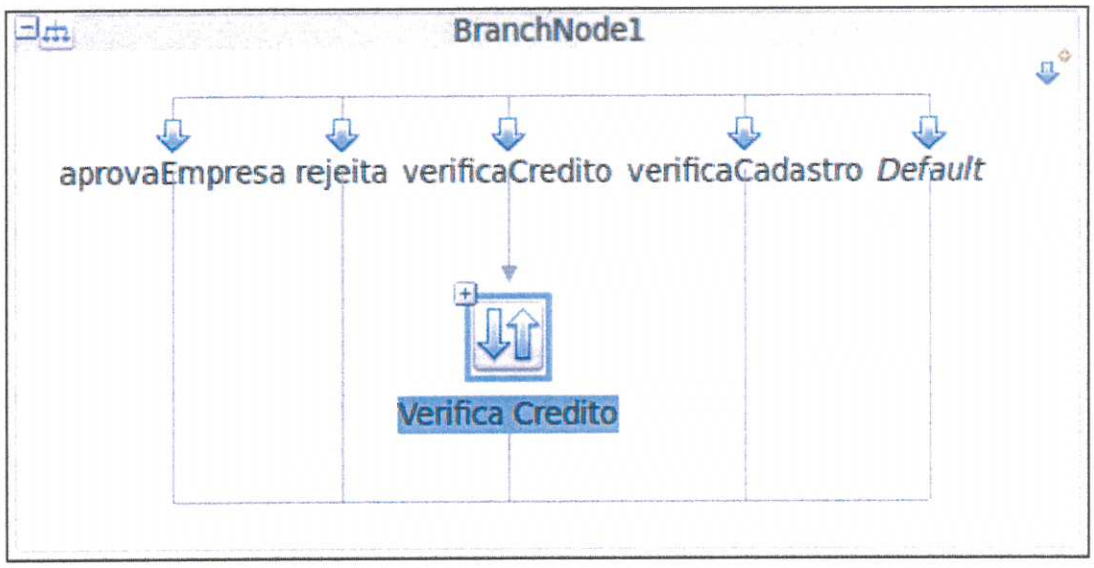
4. Clique na **seta com o sinal +,** para atribuir novas operações ao BranchNode. Atribua todas as operações contidas no contrato.



passo6.

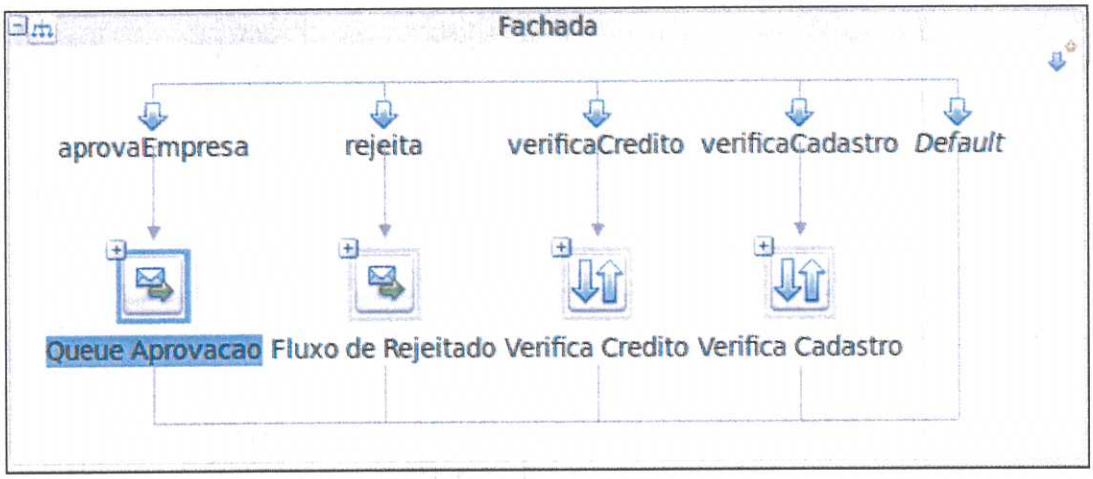
7. Vá até o **EmpresaProxy**, minimize o *Pipeline-pair*, clique com o botão direito e copie.

8. Volte ao proxy **EmpresaFacade**, e embaixo da operação verificaCredito, cole o Pipeline-pair copiado no passo anterior.



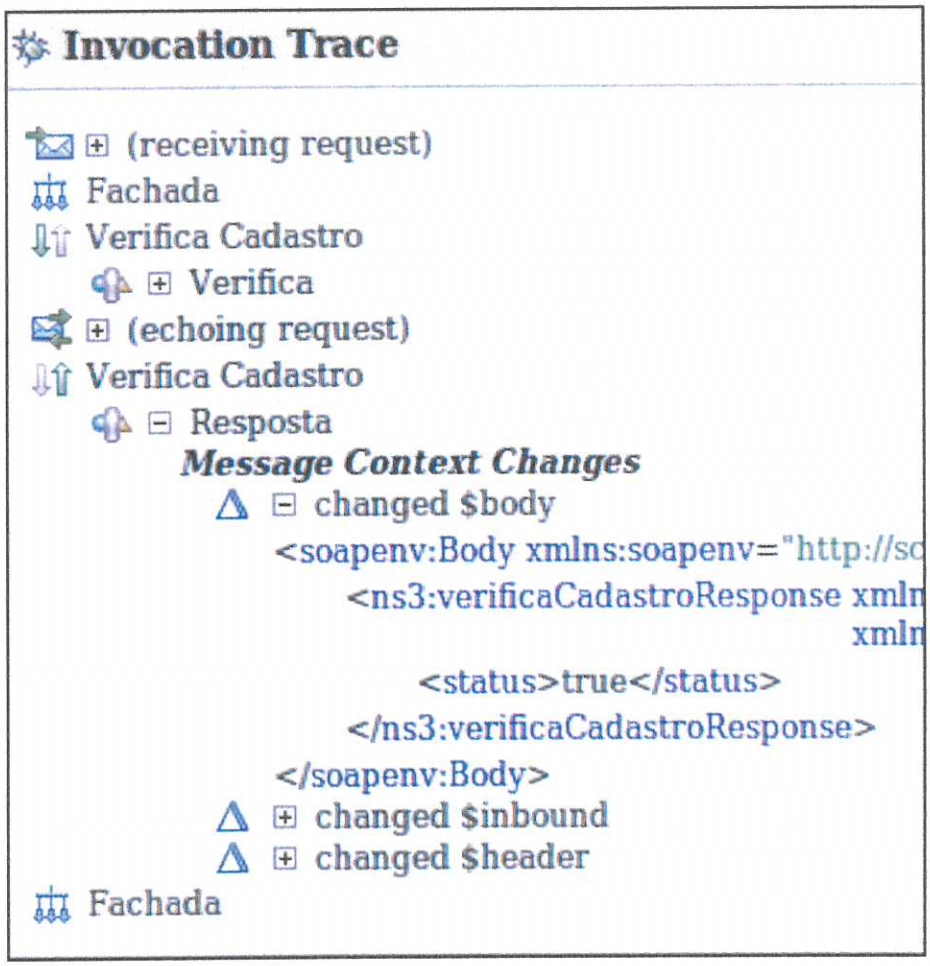
passo8.

9. Repita o passo 8 para toda a correlação entre Operação e Proxy anteriormente criados.



passo9.

10. Salve o projeto e execute o teste no mesmo, para auferir as chamadas.



passo10.

#dica, Após a conclusão com sucesso, apague os proxies antigos relacionados à empresa, deixando somente o EmpresaFacade, a fim de não confundí-lo durante o próximo exercício.

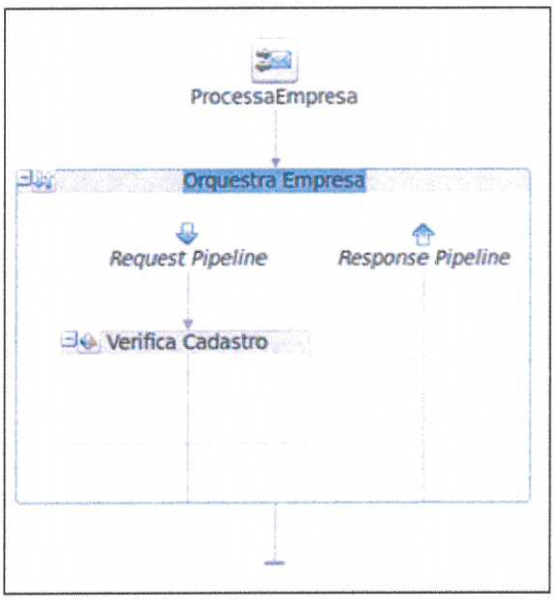
Parte XII – ProcessaEmpresa – "Orquestrador através do Façade"

1. Crie um novo Proxy, à partir do contrato disponibilizado para esta aula – **ProcessaEmpresa.wsdl**

2. Siga todos os padrões de construção de proxies, até o Message Flow.

3. Adicione um *Pipeline-pair* chamado **Orquestra Empresa**, abaixo do Start Node.

4. Dentro do Pipeline-pair, adicione um estágio de Verifica Cadastro.



passo4.

5. Dentro do estágio, adicione um **Service Callout,** para consultarmos se o cliente possui cadastro, através do nosso proxy de empresa – EmpresaFacade.

6. Nas propriedades do ServiceCallout, defina as variáveis de Request e Response.

7. Salve o projeto, e execute o EmpresaFacade, operação "verificaCadastro", para cópia do payload de entrada do Service Callout.

8. Dentro do Request Action do Service Callout, inclua um **Assign,** e o payload de entrada copiado.

OBS: Não se esqueça de substituir as variáveis do payload de entrada, utilizando expressões XPath do Request e atribuir à variável **verificaCadastroRequest.**

passo8.

9. Abaixo do Service Callout, inclua um componente **IF Then.**

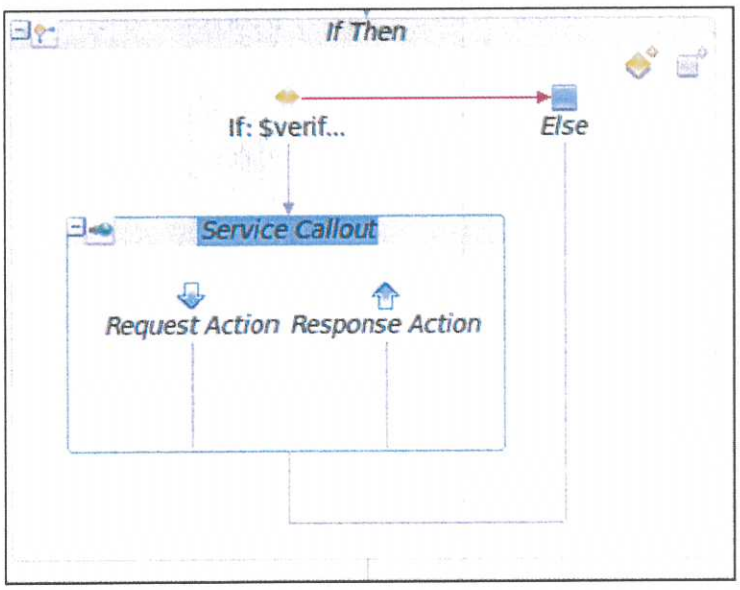
10. Em condition, forme a seguinte sentença:

Condition Expression:

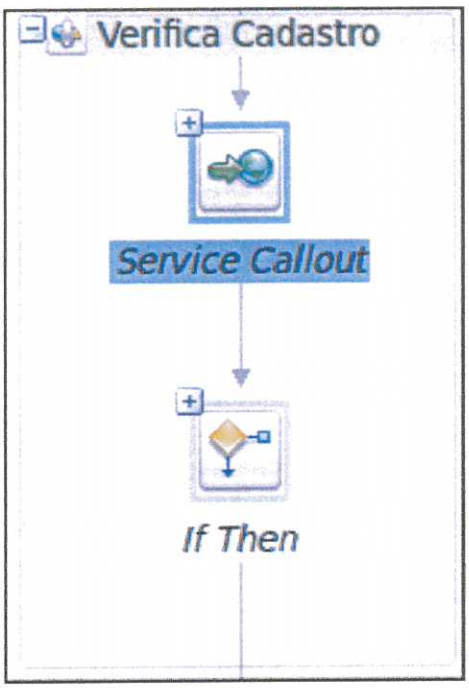
$verificaCadastroResponse/status/text() != "true"

A fim de verificar se a empresa está ou não cadastrada.

11. Na condição verdadeira, adicione um **Service Callout**, para invocarmos o serviço de cadastro, através da EmpresaFacade.



OBS: Não se esqueça de incluir um Assign para o Request, com seu respective payload de entrada.



passo11

12. Inclua um novo estágio, abaixo do Verifica Cadastro, chamado: **Verifica Crédito.**

13. Inclua um ServiceCallout, para a operação de verificação de crédito, do proxy Empresa Façade.

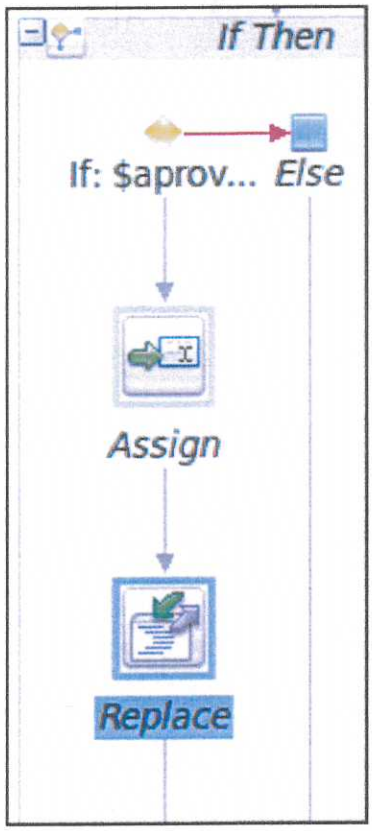
14. Após o ServiceCallout, inclua um **Assign,** com o valor da variável de retorno **status,** para a variável $aprovado.

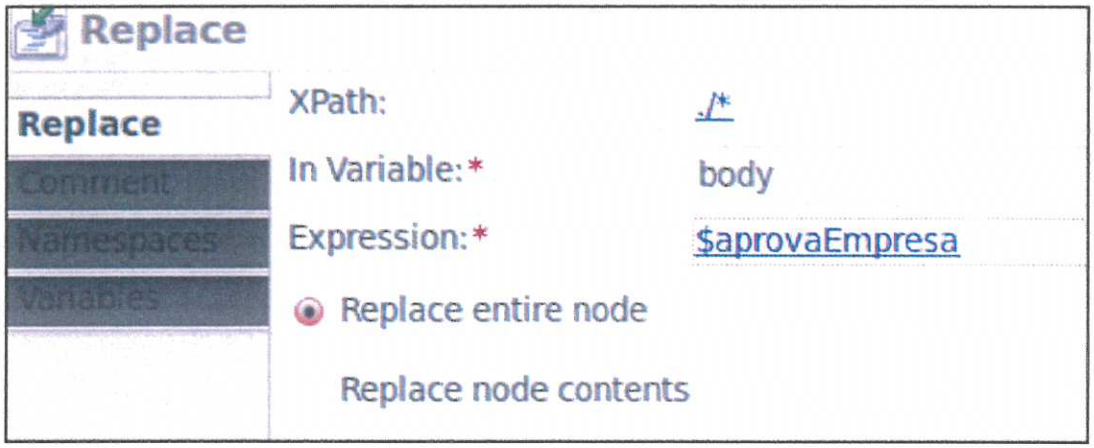
15. Inclua um novo estágio abaixo de verificação de crédito, chamado **Aprova|Rejeita.**

16. Dentro do estágio, inclua um **IF Then** , comparando o valor da variável **$aprovado** se igual à verdadeiro.

17. Na condição verdadeira, inclua um **Assign**, com o payload de entrada da operação "aprovaEmpresa", definida no proxy EmpresaFacade - **$aprovaEmpresa.**

18. Adicione um **Replace** de todo o contexto da variável $body, pelo conteúdo da variável **$aprovaEmpresa.**





19. Adicione um **Publish** , após o replace, e invoque o serviço EmpresaFacade, juntamente com a operação de aprovação.

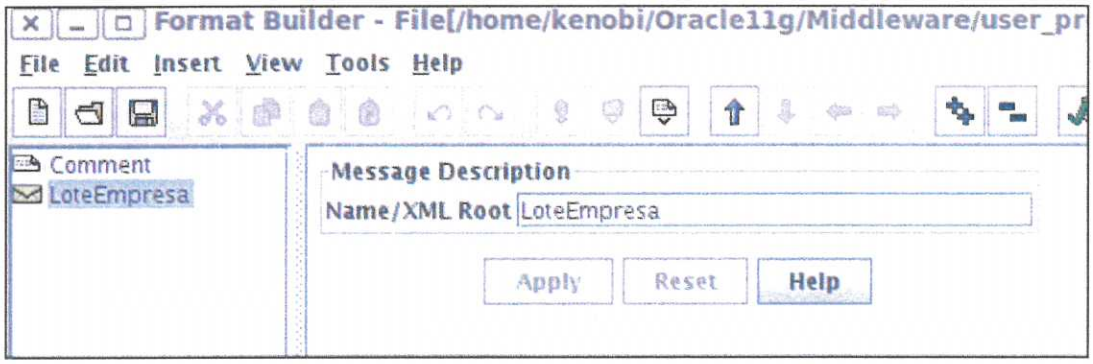
20. Repita a operação para o Else (**Rejeitado**).

Parte XIII – Processamento de Lote de Arquivos – MFL Builder

1. No Eclipse clique em File | New | e procure a opção MFL dentro de Data Transformation.

2. Crie um novo arquivo chamado **LoteEmpresa.mfl.**

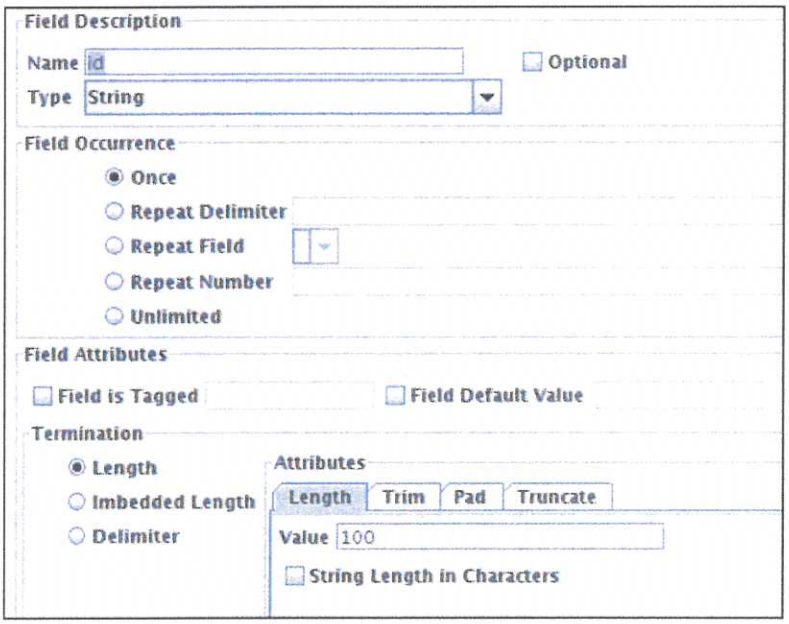
3. Crie um elemento raíz com o nome de **LoteEmpresa** .



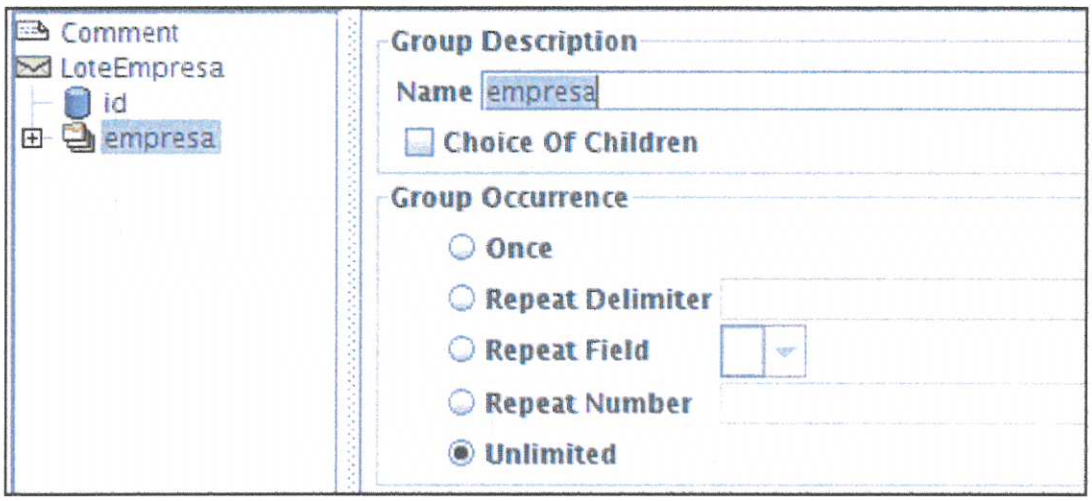
passo3

4. Clique com o botão direito em cima do LoteEmpresa e adicione um Field **id** como filho (child) do mesmo.

5. Defina o tipo do id como String, sua delimitação por tamanho (length) com o valor 8.

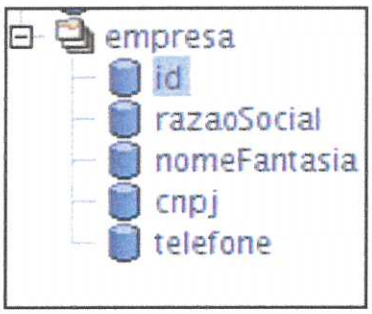


6. Adicione um Grupo (Group) ao nó raiz, chamado empresa e selecione a propriedade de ocorrência do mesmo para ilimitado.



passo6.

7. Crie as seguintes propriedades do tipo **String** embaixo do grupo empresa: id, razaoSocial, nomeFantasia, cnpj, telefone.

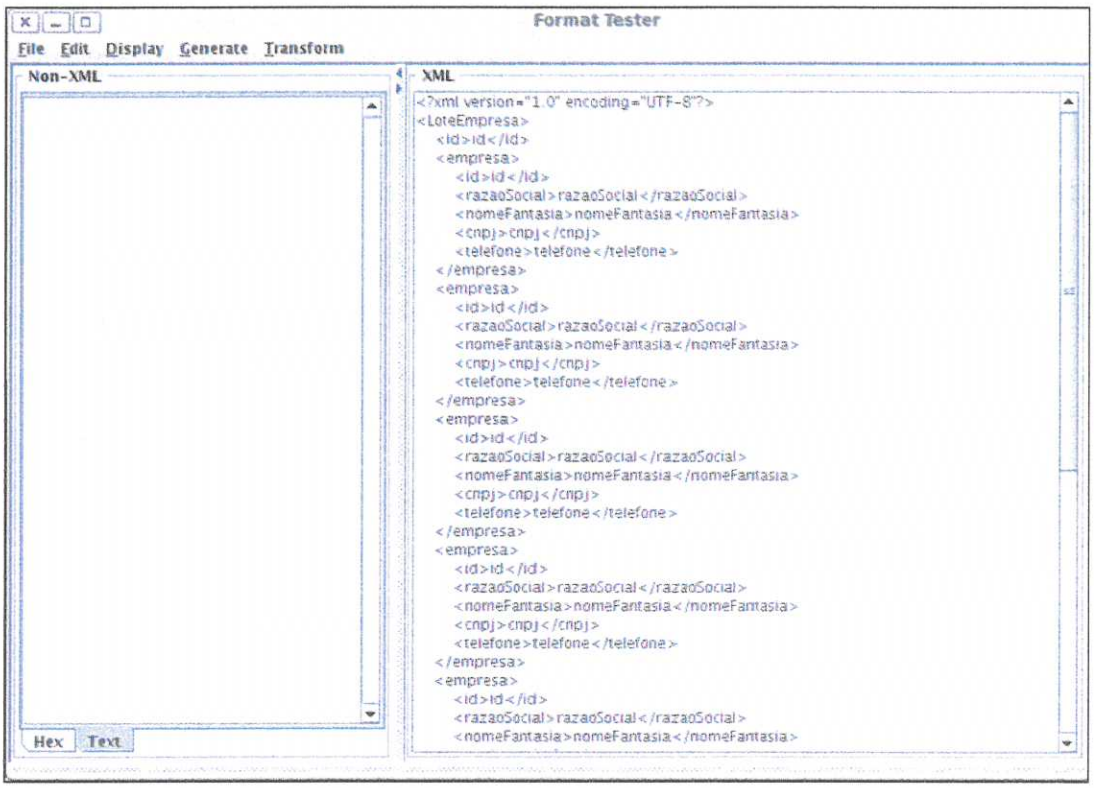


passo7

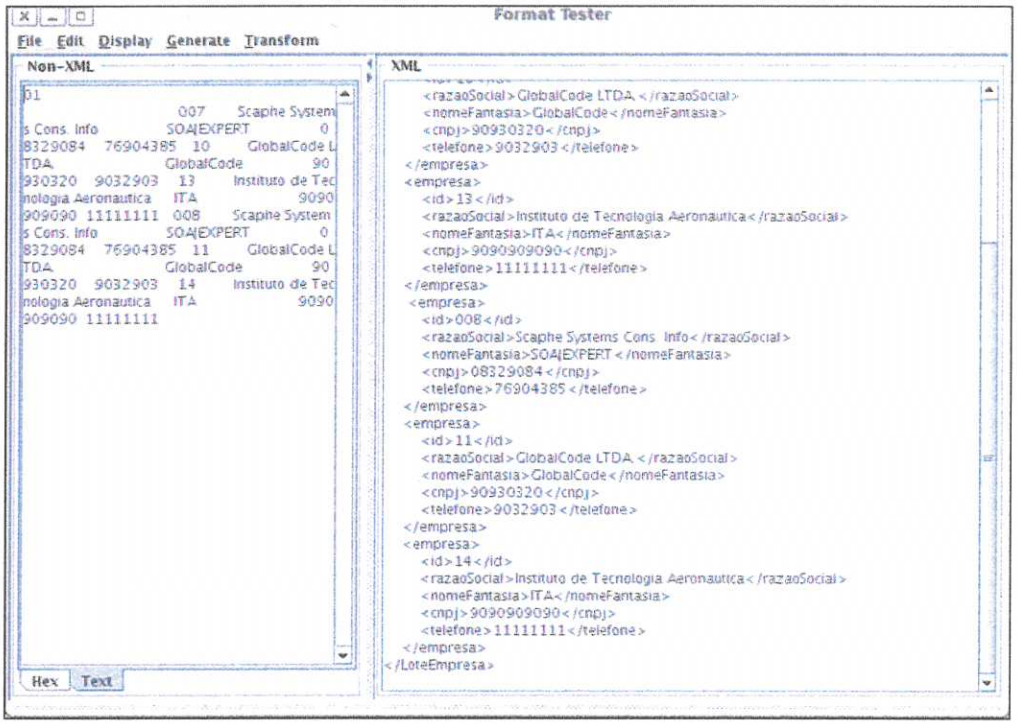
OBS: Todos os atributos serão delimitados pelo tamanho dos caracteres: Marque a opção **String Lengh.**

8. Clique em Tools – Test, para o menu de test do MFL Builder.

9. Na opção Generate, crie um XML.



10. Clique em Transform e gere o Non-XML.



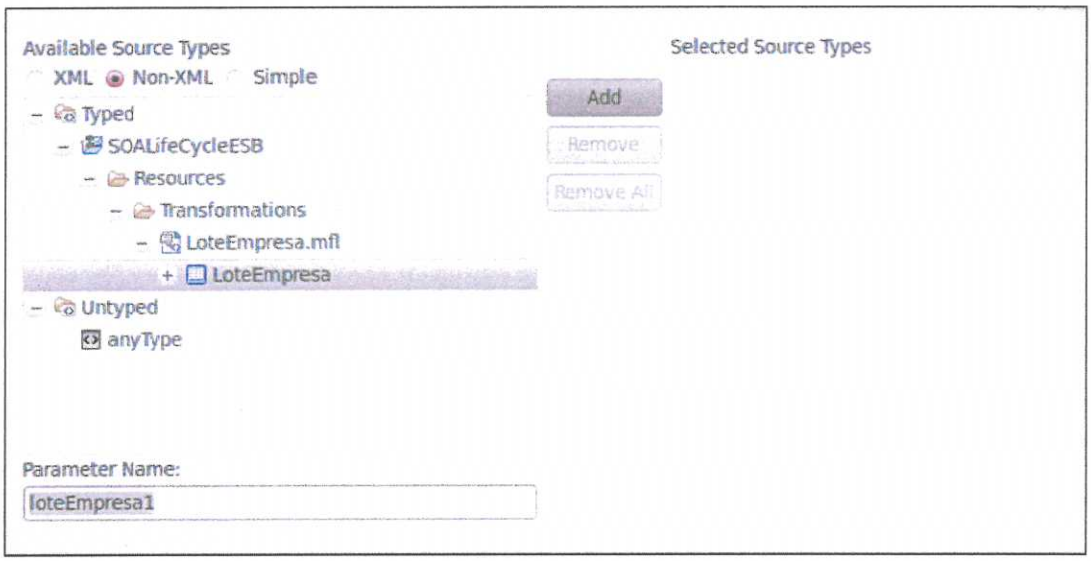
11. Salve ambos arquivos XML como loteEmpresa.xml e non-XML como, loteEmpresa.data.

Parte XIV – Tranformação de tipos – Xquery

OBS: Para o que o exercício funcione, crie um elemento chamado **loteEmpresa** do tipo LoteEmpresa, no seu DomainModel.XSD.

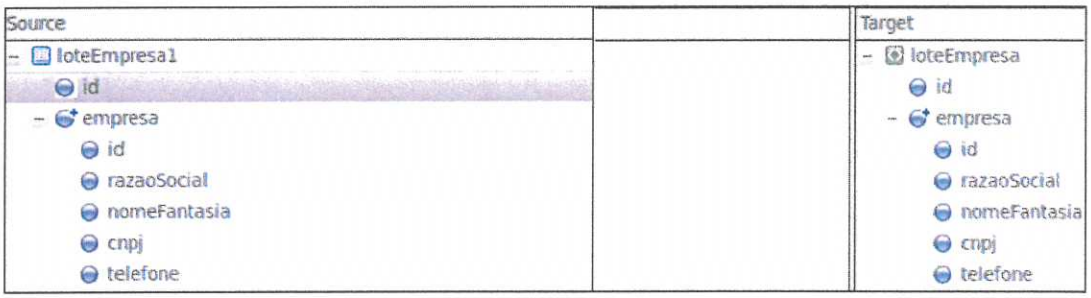
1. Crie uma Xquery no projeto OSB, na pasta transformations, chamada **LoteEmpresaMFL2XML.**

2. Em Source Types, selecione o tipo Non-XML como parâmetro de entrada e localize o **LoteEmpresa.mfl** criado no exercício anterior.

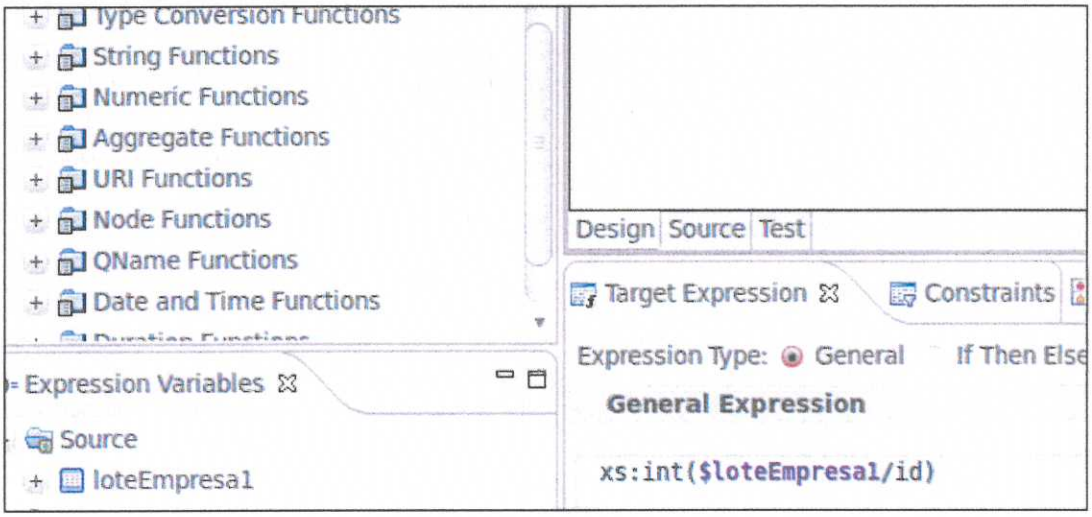


passo2.

3. Selecione o elemento **loteEmpresa** definido no DomainModel.xsd.



4. Lembre-se que existem propriedades numéricas no seu loteEmpresaXML (Target), por tanto, uma operação de Casting (conversão), sera necessária.



passo4.

5. Ligue todos os outros pontos de uma extremidade à outra.

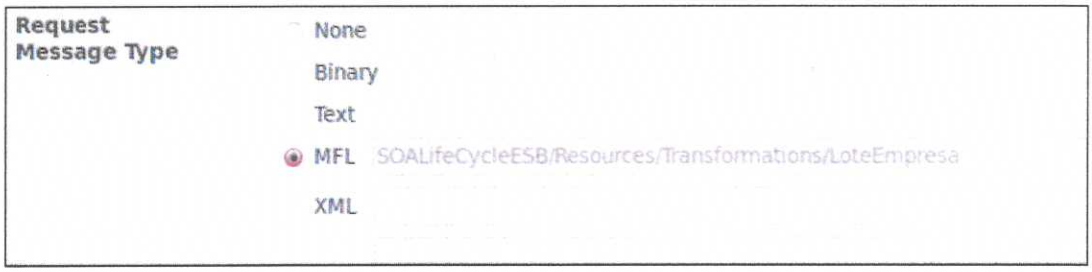
6. Verifique em source, se o XQuery realizou o laço **for** para o elemento empresa.

Parte XV – Proxy de ETL e Disparo de processamento

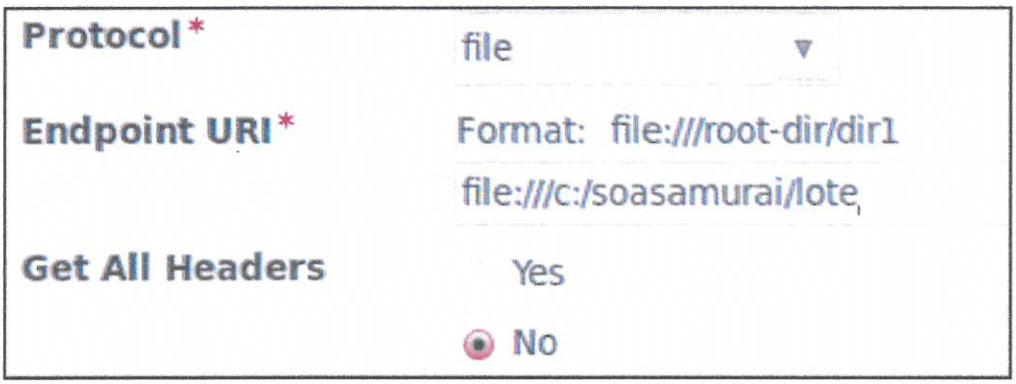
1. Crie um novo Proxy com o nome de **LoteEmpresaETL.**

2.Na aba General, escolha **Messaging Service.**

3.Na aba Messaging, na propriedade **Request Message Type**, selecione **MFL** e localize o LoteEmpresa.mfl.



4. Na aba Transport, selecione o protocolo file e defina o diretório de trabalho dos seus arquivos de processamento. Abaixo segue um exemplo referência.



passo4

5. Na aba File Transport, defina os sub-diretórios de stage e error.

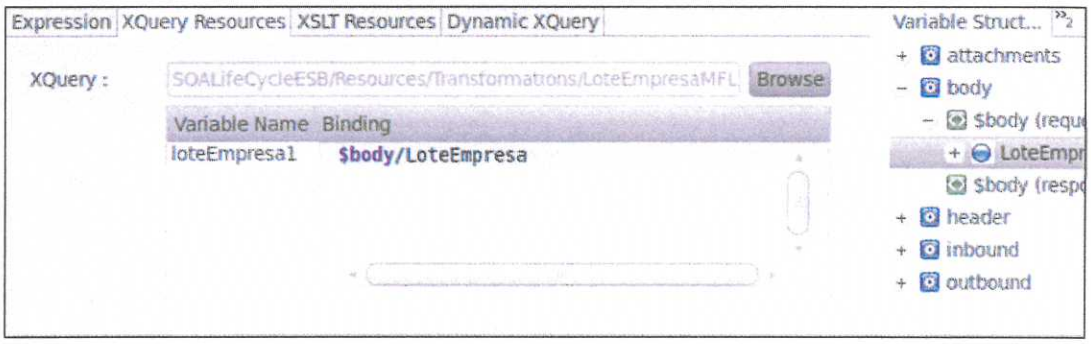
6. Na aba Message Flow, inclua um Pipeline-pair chamado: **Processa ETL**

7. Dentro do Request, adicione um estágio chamado **Carrega Lote**

8. Defina uma varíavel **loteEmpresaXML** com Assign dentro do estágio.

9. Em expression, dentro do Assign, escolha aba Xquery Resources e localize a Xquery LoteEmpresaMFL2XML, criada no exercício anterior.

10. Navegue pelo contexto $body e encontre a variável do tipo de entrada da transformação, o **MFL Lote Empresa** e arraste para o biding, a fim de formar a expressão XPATH.

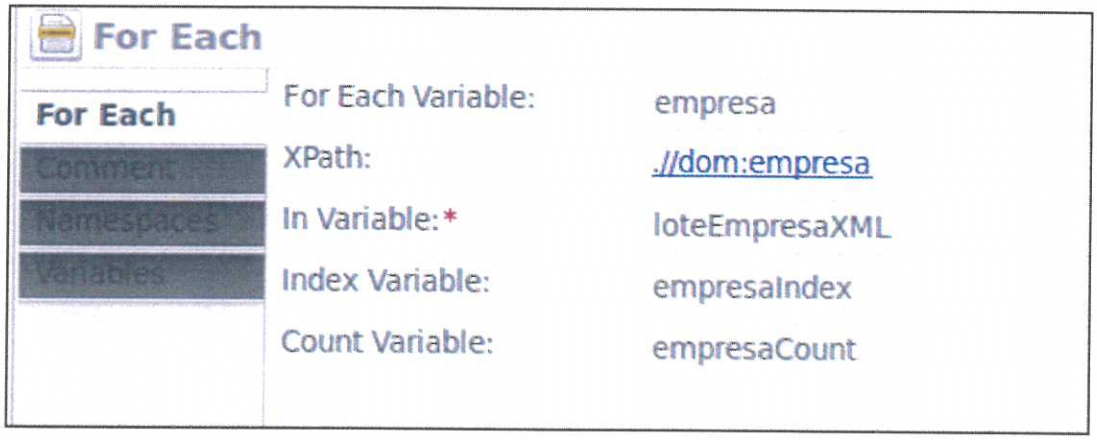


passo10.

11. Embaixo do estágio Carrega Lote, crie outro estágio chamado Dispara Processamento.

12. Inclua um elemento de **For Each** dentro do estágio criado.

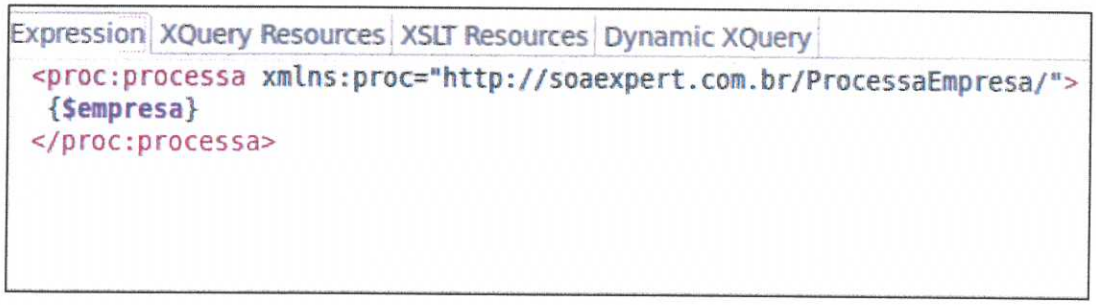
13. Nas propriedades For Each, crie uma variável $empresa, navegue pelo nó de empresas da variável **$loteEmpresaXML** e crie variáveis para índice e contagem.



passo13

14. Dentro do **For Each** inclua um Publish, para disparo Assíncrono da mensagem ao serviço **ProcessaEmpresa** (proxy).

15. Inclua um Assign com o payload de entrada do ProcessaEmpresa, levando a variável definida no For Each, empresa, como parâmetro interno do mesmo.



passo15.

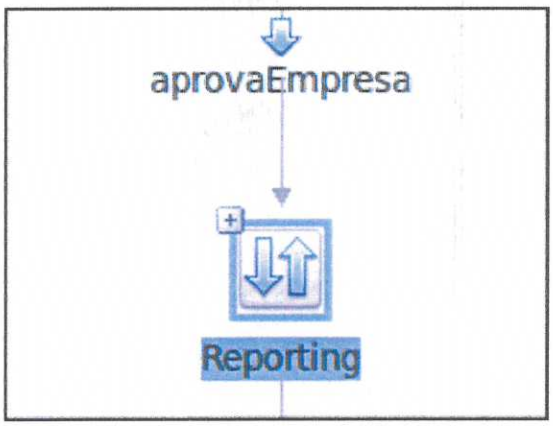
16. Teste o serviço, colocando o loteEmpresa.dat no arquivo definido definido para leitura de processamento.

Parte XVI – Monitoria e Relatório | Criando um Relatório Aprova |Rejeita

Não confunda os conceitos de monitoria com relatório, pois um serve ao propósito RealTime e o outro, para analisar o que passou e ajustar sua implementação para melhoria ou otimização.

1. Abra o proxy EmpresaFacade.

2. Embaixo da operação "aprovaEmpresa" adicione um *Pipeline-pair* com o nome de **Reporting**.



passo2.

3. Adicione um estágio dentro do Request Action do Pipeline-pair, chamado **Criando Relatório.**

4. Dentro do estágio, adicione um componente de **Report.**

passo4

5. Nas propriedades do componente Report, adicione a seguinte função XQuery: **fn-bea:serialize($body)**

Report	Expression:	fn-bea:serialize($body)
Comment:	Search keys:	Add a Key

passo5

6. Adicione uma chave, para identificação da pripriedade. Em name digite idEmpresa.

7. Na expressão XPath, localize o elemento ID da empresa no contexto de request. Expressão referência: .//dom:id.

Key Name		Key Value	
Name:*	idEmpresa	XPath:*	.//dom:id
		In Variable:*	body

passo7.

8. Dentro do elemento Routing da Queue de Aprovação, adicione um componente **Report.**

9. Em expression, coloque a sentença: **'Antes da chamada de Aprova'**

10. Adicione uma Key com o nome de **QueueAprova.**

11. No Xpath, localize o nó contendo o nome fantasia da empresa, contida na variável de contexto(request).

Key Name		Key Value	
Name:*	QueueAprova	XPath:*	.//dom:nomeFantasia
		In Variable:*	body

passo11

12. Repita o mesmo procedimento para a operação "rejeitaEmpresa" do EmpresaFacade proxy.

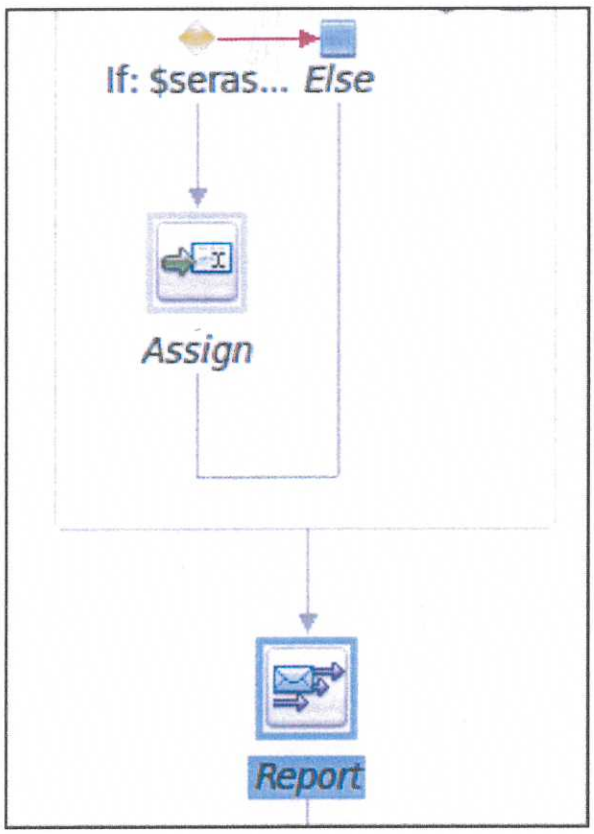
Parte XVII – Criando um Relatório para Verificação de Crédito

A idéia desse relatório é verificar em tempo real, o status de aprovação de cada entidade de crédito.

1. Inclua um componente de **Report** dentro do estágio variáveis, da operação "verificaCredito", do EmpresaFacade proxy.

2. Na propriedade expression, inclua a função XQuery similar ao exerício anterior.

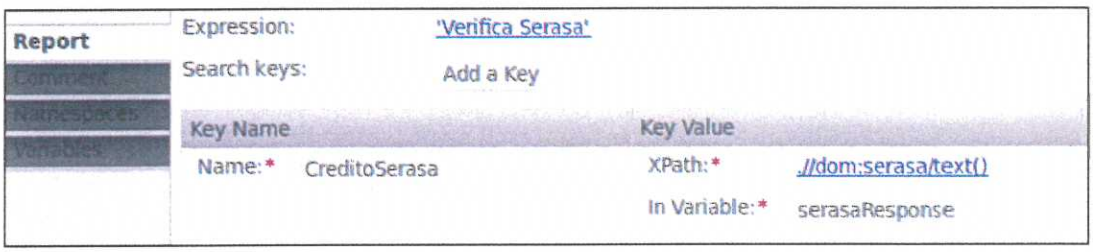
3. No *Pipeline-pair* VerificaSerasa, inclua um componente **Report** embaixo do IF Then.



passo6

4. Na propriedade Expression, coloque a sentença: 'Verifica Serasa'

5. Adicone uma Key com o nome de CreditoSerasa e localize com o Xpath o valor de retorno da variável de resposta (**ServiceCallout**) da consulta ao crédito.



passo5

6. Repita o processo de relatório, para as outras 2 verificações de crédito – SPC e CasasBahia, fazendo as devidas modificações.

Parte XVIII – Criando um Alerta de Erros

Durante nosso fluxo de negócio, erros podem acontecer e há uma maneira de notificarmos em tempo real um sistema externo, como sistemas de monitoria, e-mails às pessoas responsáveis, entre outros.

O objetivo dessa parte é criar um alterta em cima de uma operação de consulta de crédito de uma das instituições e dar continuidade no fluxo, sem que o mesmo pare por conta de uma eventual indiponibilidade.

1. Primeiramente será necessário criar uma Fila (Queue) para destinarmos nosso fluxo de erros de forma Assíncrona. Para tanto, crie uma fila como nos exercícios anterior com o nome de **EmpresaErroQueue.**

2. Dentro da pasta Resources do nosso projeto, crie um novo tipo de Componente, Alert Destination, chamado **EmpresaErrosDestination.**

3. Configure as propriedades como no quadro abaixo:

Configuration

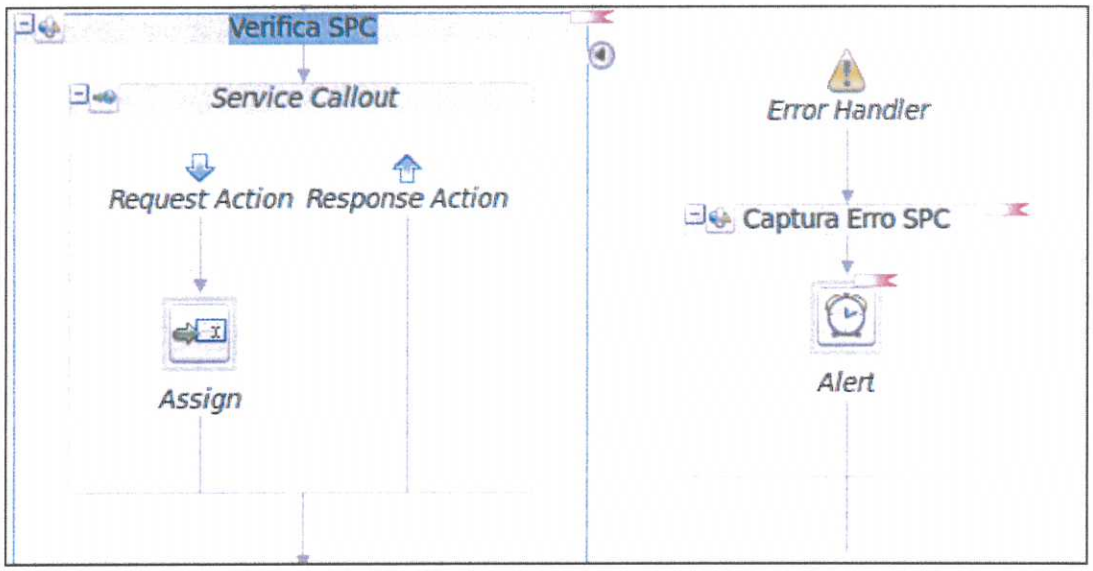
Use this page to configure the Alert Destination.

Description	Erros do Fluxo de Processamento de Empresa	
SNMP Trap	Yes	No
Reporting	Yes	No
Alert Logging	Yes	No

4. No Proxy EmpresaFacade, procure o Pipeline-pair **Verifica SPC** e adicone um **Error Handler** ao mesmo.

5. Adicione um estágio ao Error Handler chamado: Captura Erro SPC.

6. Adicione um componente de **Alert**, dentro do estágio.



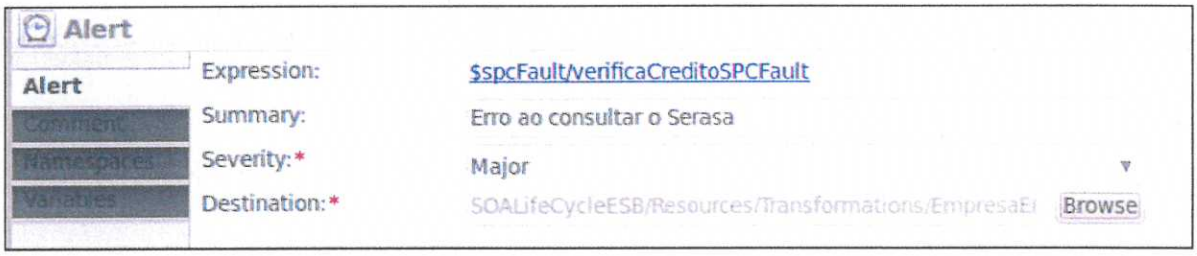
passo6

7. Na expressão do Alerta, adicione a FaultMessage da operação verificaCreditoSPC

8. Coloque um sumário explicativo como: **Erro ao consultar o SPC.**

9. Atribua um nível de severidade – Major.

10. Localize o destino criado no passo anterior.



passo10.

Parte XIX – Configurando Regras de SLA.

SLAs são inerentes ao nível operacional, e normalmente são definidas com base no conhecimento de processo de negócio, pelos arquitetos.

A idéia dessa parte, é mostrar de forma simples, como definir uma SLA Temporal, com base na performance da consulta de crédito.

Crie antes um novo destino para as regras temporais, como **EmpresaDestination**.

1. Abra o SBConsole e localize o proxy EmpresaFacade, clique sobre o mesmo.

2. Localize a aba SLA Alerts e clique na mesma.

3. Clique em Add, para adicionar um novo alerta.

4. Defina as propriedades de configuração, como no quadro abaixo:

General Configuration	
Rule Name*	TempoCredito
Alert Summary	Alerta de Tempo de Resposta do Serviço de Credito
Rule Description	
Alert Destination*	
Start Time (H:MM)	AM
End time (H:MM)	AM
Rule Expiration Date (M/D/YYYY)	
Rule Enabled	Yes No
Alert Severity	Normal
Alert Frequency	Every Time
Stop Processing More Rules	Yes No

passo14

15. Localize o Alerta Destination, definido no início do exercício.

16. Na expressão, escolha o **Minimum** e Response Time = 1000 milesegundos.