Guia de implementação do BPEL 11g

Versão 1.0.0



**Histórico de Revisões**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 18/04/2012 | 1.0.0 | Criação do Documento | Victor Fonseca |

**Sumário**

[1 Objetivo 4](#_Toc319927865)

[2 Introdução 5](#_Toc319927866)

[3 Principais componentes 6](#_Toc319927867)

[3.1 Receive 6](#_Toc319927868)

[3.2 Assign 6](#_Toc319927869)

[3.3 Transform 6](#_Toc319927870)

[3.4 Invoke 6](#_Toc319927871)

[3.5 While 6](#_Toc319927872)

[3.6 Java Embedding 7](#_Toc319927873)

[3.7 Reply 7](#_Toc319927874)

[3.8 Throw 7](#_Toc319927875)

[3.9 Scope 7](#_Toc319927876)

[3.10 Compensate 7](#_Toc319927877)

[4 Controle de transação 8](#_Toc319927878)

[4.1 Implementando o controle de transação 8](#_Toc319927879)

[5 Fluxo de compensação 10](#_Toc319927880)

[5.1 Implementando um fluxo de compensação 10](#_Toc319927881)

[5.2 Implementando o tratamento de erro 12](#_Toc319927882)

[6 Agendar inicialização 14](#_Toc319927883)

[6.1 Utilização do CONTROL-M 14](#_Toc319927884)

[6.2 Job agendado no banco de dados 14](#_Toc319927885)

[6.3 BPM chamando um BPEL 15](#_Toc319927886)

# Objetivo

Este documento tem como objetivo demonstrar as principais funcionalidades do Oracle BPEL 11g bem como formas de utilização e tratamento de erros e transações que serão utilizados pela OI.

# Introdução

Através desse documento, será possível implementar um processo em BPEL e fazer tratamentos de erros e exceções, além de controle de transação dentro do mesmo.

# Principais componentes

Para a construção de um fluxo BPEL usa-se alguns componentes que são responsáveis pelas ações dentro do fluxo. Na implementação de referência enviada juntamente com este documento encontra-se a maior parte dos componentes abaixo.

## Receive

Responsável por receber uma mensagem, seja ela enviada através de um WebService, ou uma leitura de um arquivo, ou leitura de uma fila JMS, etc. Este componente normalmente inicia um fluxo BPEL, pois após o recebimento de uma mensagem uma instância é iniciada. Ele é representado pelo símbolo abaixo:



## Assign

Responsável por atribuir valor a uma variável. Através deste componente podemos atribuir valor a uma variável, podendo ser um valor fixo ou dinâmico como o resultado de uma função BPEL por exemplo. Ele é representado pelo símbolo abaixo:



## Transform

Responsável também por atribuir valor a uma variável. Este componente tem a mesma função do assign, porém ele faz uma transformação de valores através de funções XSL e outras funções em BPEL. Ele é representado pelo símbolo abaixo:



## Invoke

Responsável pela chamada de um adapter. Este componente faz a chamada de um adapter e passa para ele os parâmetros necessários para a sua invocação. Ele é representado pelo símbolo abaixo:



## While

Responsável por fazer um loop. Este componente executa um loop baseado em uma condição configurável no próprio componente. Como em qualquer linguagem de programação, o while é executado até não atender mais a condição. Ele é representado pelo símbolo abaixo:



## Java Embedding

Responsável por executar código em Java. Através deste componente podemos escrever um código em Java para ser executado. Ele é representado pelo símbolo abaixo:



## Reply

Responsável por responder uma mensagem. Esta atividade é a última a ser executada em um fluxo síncrono ou assíncrono, porque ela quem vai responder para o cliente que chamou o BPEL. Ele é representado pelo símbolo abaixo:



## Throw

Responsável por lançar uma exceção. Quando é preciso lançar uma exceção no BPEL, este componente deve ser usado, nele pode-se configurar qual é a exceção a ser lançada. Ele é representado pelo símbolo abaixo:



## Scope

Responsável por encapsular várias atividades, definindo assim um escopo para tratamento de erros, transação, execução do fluxo compensate, etc. Ele é representado pelo símbolo abaixo:



## Compensate

Responsável por chamar os fluxos de compensação. Após implementar os fluxos de compensação, é necessário chamá-los através deste componente que normalmente fica em um fluxo de tratamento de erro. Ele é representado pelo símbolo abaixo:



# Controle de transação

Por padrão, todo processo BPEL é executado como uma única transação, ou seja, quando ocorre um erro no processo, todas as alterações são desfeitas. Mas é possível “quebrar” o processo em várias outras transações se necessário, fazemos isto finalizando a transação já iniciada e iniciando outra.

Existem alguns componentes do BPEL que finalizam uma transação e criam outra, são eles:

* ***Receive*** – A menos que ele seja o primeiro passo do processo e o processo esteja participando de outra transação (“*transaction=participate*”).
* ***checkpoint()*** – Colocado dentro do componente Java Embedding.
* ***onMessage***
* ***wait*** – A menos que este wait seja de curta duração (2 a 3 segundos).
* ***onAlarm***
* ***invoke*** – Somente se a propriedade “idempotent” estiver carregada como false.
* **final do fluxo** – A menos que o processo estiver participando de outra transação.

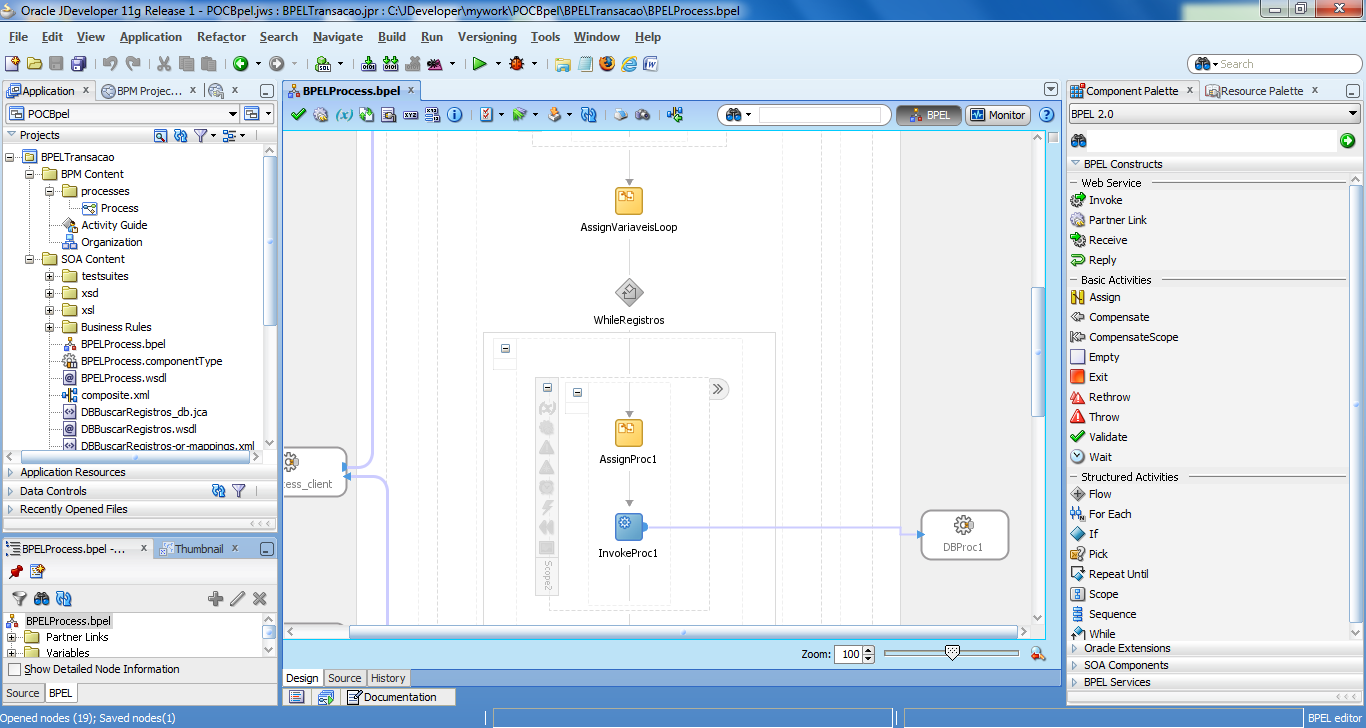
Portanto, se for necessário finalizar uma transação dentro de um fluxo BPEL, devemos usar um dos componentes acima. Lembrando que ao finalizar uma transação, o rollback não poderá mais ser efetuado e este deverá ser feito através de um fluxo de compensação.

## Implementando o controle de transação

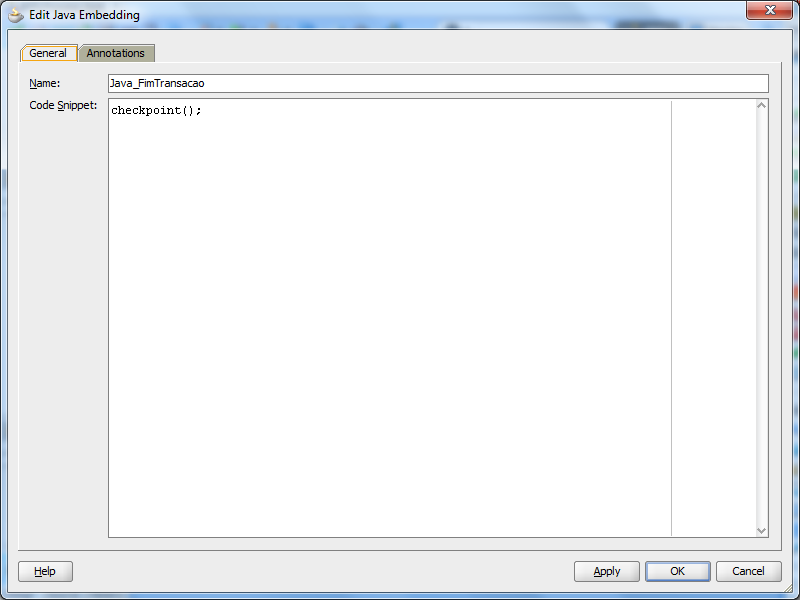
Para utilizar os recursos de “quebra” de transação no BPEL, siga os passos abaixo:

Abra o BPEL no JDeveloper.

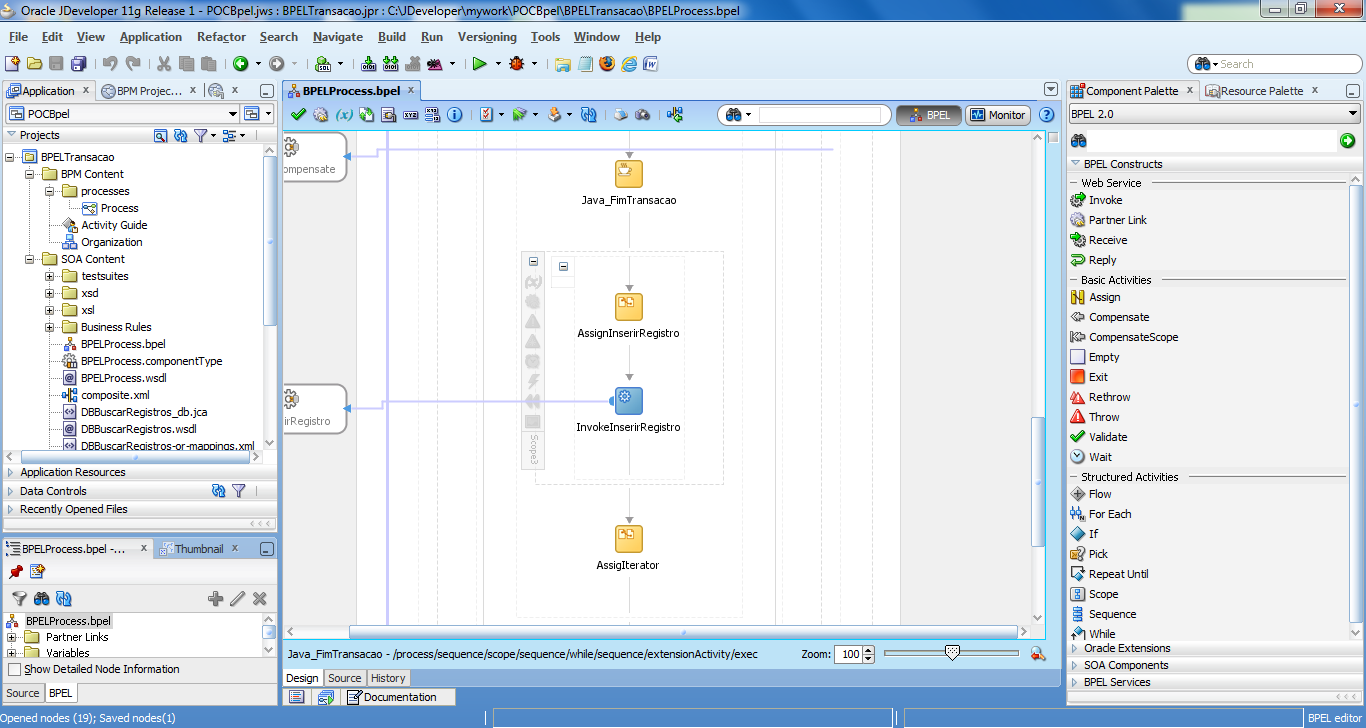
Faça a chamada a uma procedure que fará o insert em uma base de dados utilizando o DBAdapter. Para mais informações de como efetuar esta chamada, utilize o “Guia de utilização dos Adapters para SOASuite 11g”, na sessão “DataBase Adapter”.



Adicione um componente *Java Embedding* após esta chamada, abra-o e adicione o código “checkpoint();”. Isto fará com que a transação iniciada finalize e é iniciada outra a partir deste ponto, ou seja, neste momento o *insert* invocado pela *procedure* encontra-se salvo no banco de dados.



Insira em uma outra tabela qualquer um registro que cause um erro.



Note que após a execução deste processo, o banco de dados conterá apenas o registro invocado pela primeira *procedure*, porque quando ocorreu o erro, a transação já tinha sido finalizada e o *rollback* só ocorreu para a outra transação iniciada.

# Fluxo de compensação

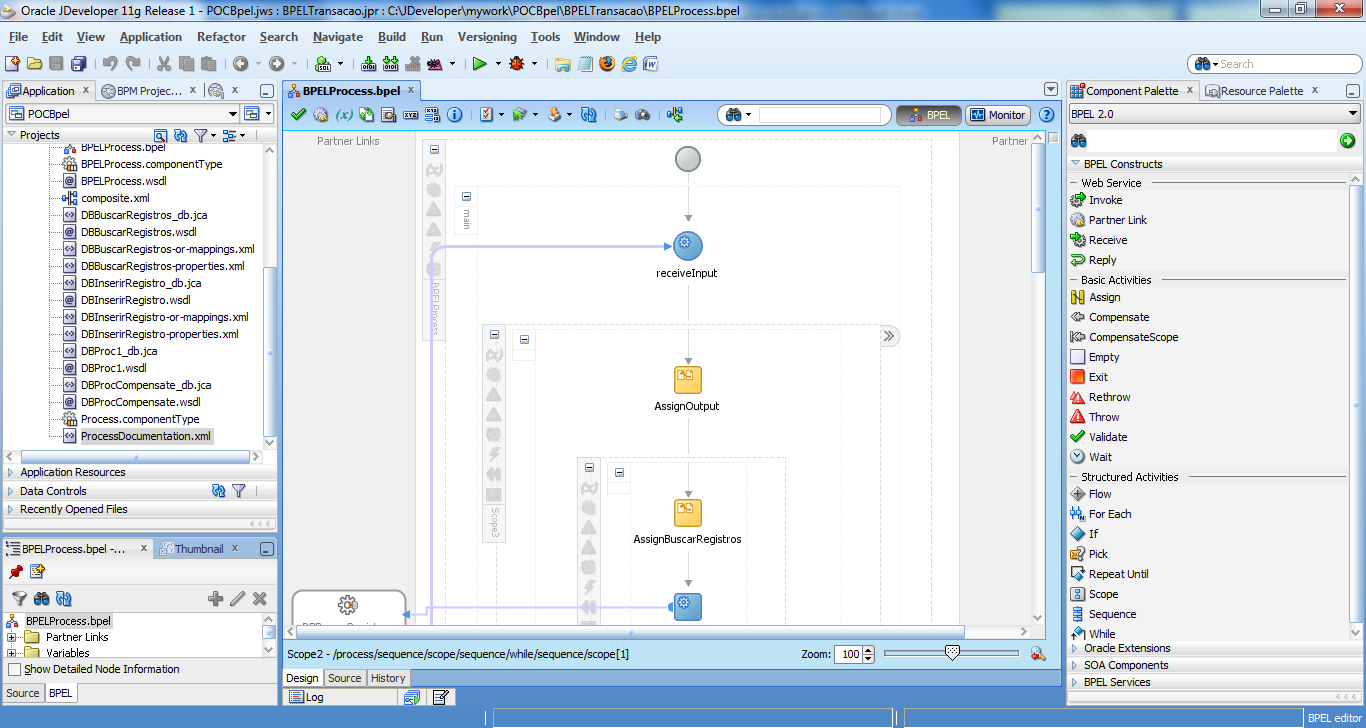
O fluxo de compensação é usado para desfazer (compensar) uma ação onde não se pode efetuar o *rollback*, seja porque o BPEL não controla a transação ou porque esta transação já foi finalizada.

## Implementando um fluxo de compensação

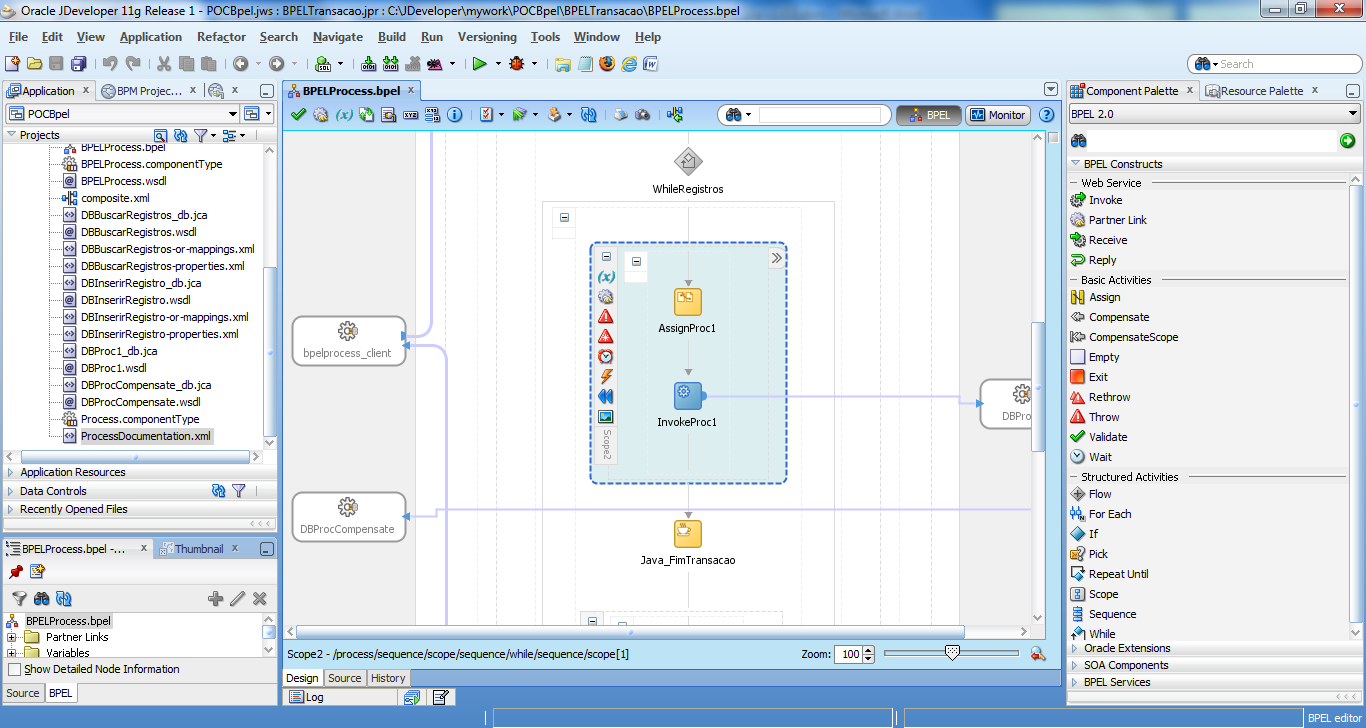
Para implementar um fluxo de compensação foi utilizada a versão 2.0 do BPEL que nos dá uma maior possibilidade de trabalhar com compensação.

Na implementação utilizada, foi adicionado um *scope* na *procedure* que faz a inserção no banco de dados antes do fim da primeira transação, se ocorrer um erro na segunda transação, este erro será capturado pelo tratamento de erro que chamará os fluxos de compensação do BPEL. Os passos para implementar um fluxo de compensação seguem abaixo.

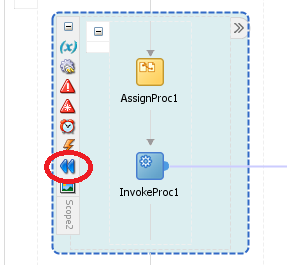
Abra o BPEL que deseja implementar a compensação.



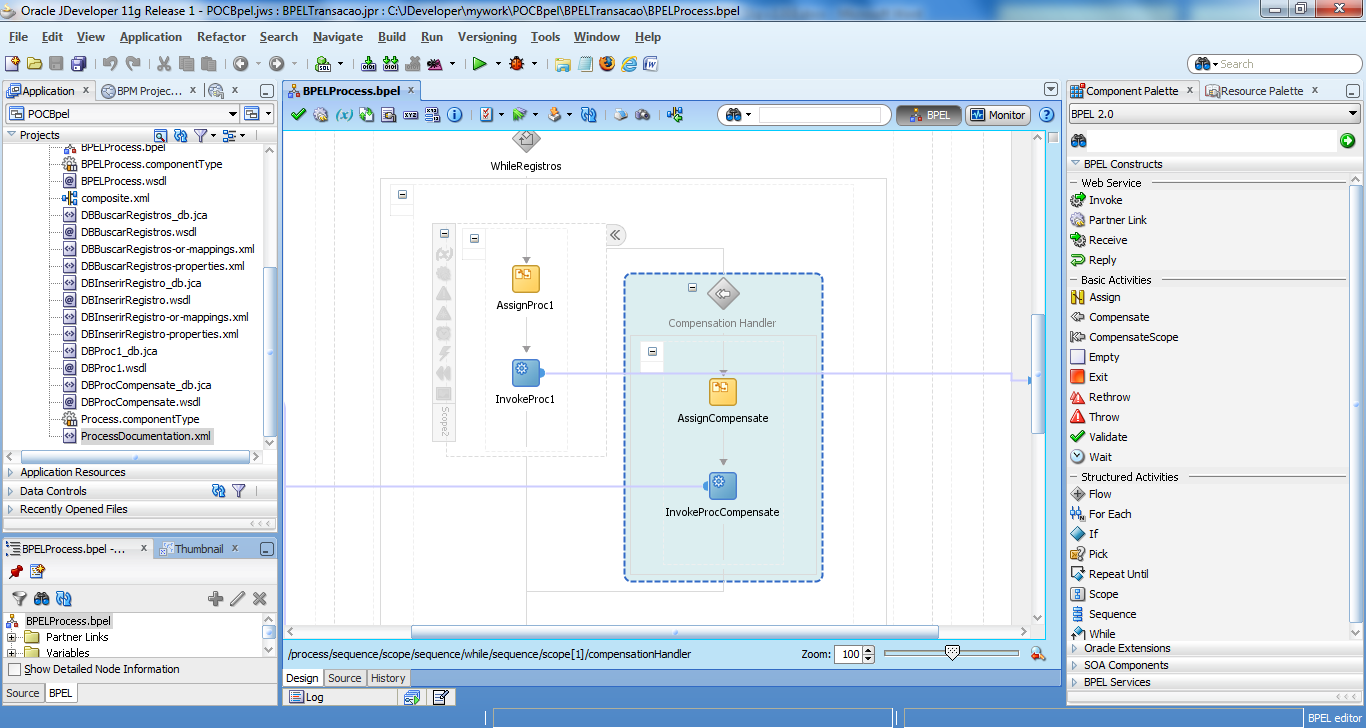
Adicione um *scope* na parte do fluxo que deve ser compensado.



Clique no penúltimo ícone do lado esquerdo do *scope* “*Add Compensation Handler*”. Este será o fluxo a ser executado para compensar a ação deste scope.



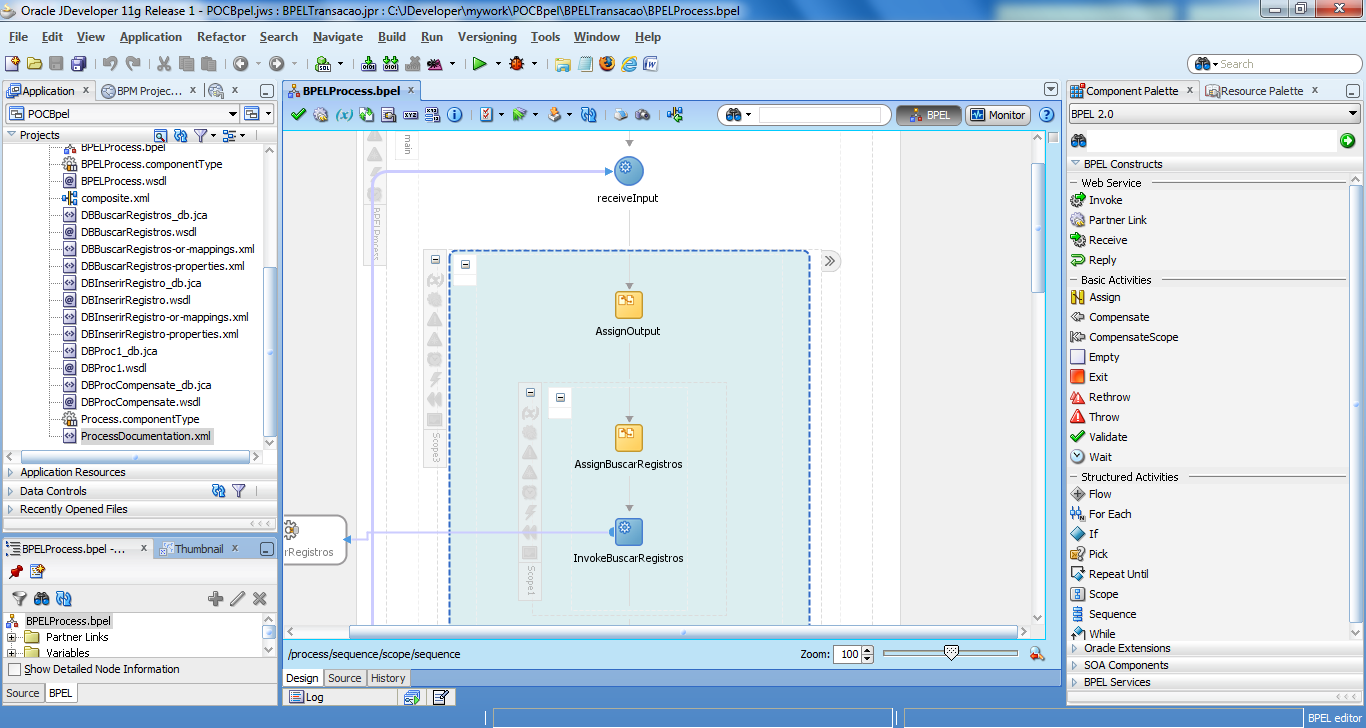
Faça um ***assign*** e um ***invoke*** a *procedure* que fará a compensação da chamada da *procedure* que faz a inserção, ou seja, crie uma *procedure* que apague os registros inseridos pela primeira *procedure*.



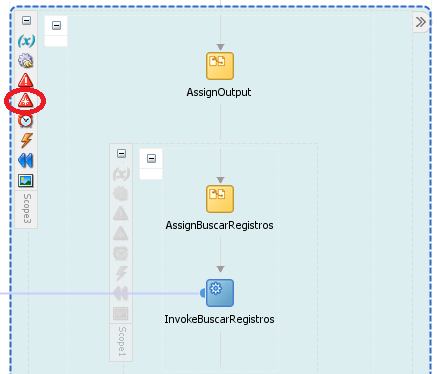
O fluxo de compensação está criado e agora precisa ser chamado pelo componente *Compensate* ou *CompensateScope*.

## Implementando o tratamento de erro

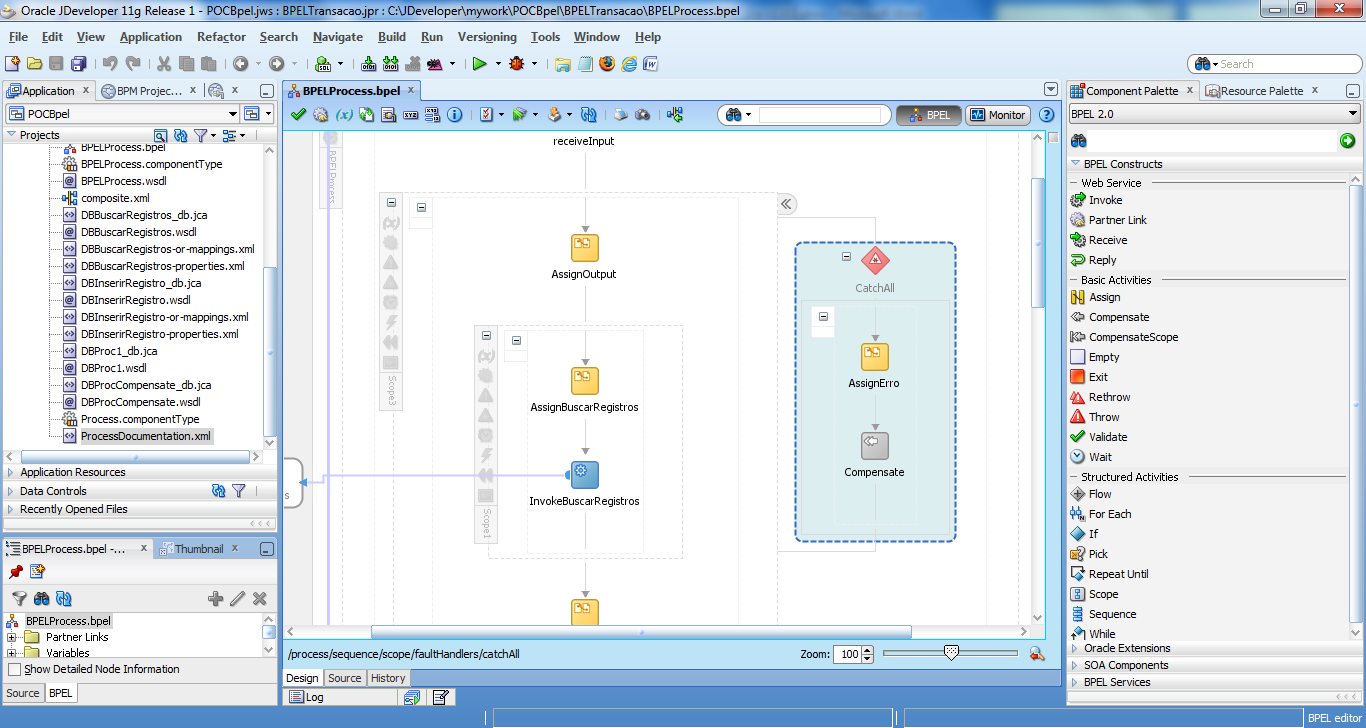
Adicione um *scope* em todo o fluxo, deixando de fora dele apenas o *receive* e o *reply*.



Neste *scope* adicionado, clique no quarto ícone a esquerda do *scope* “*Add CatchAll*” para adicionar um fluxo de tratamento de erro.



Adicione um componente do tipo Compensate. Este componente irá invocar os fluxos de compensação dos scopes internos a este.



Faça o *deploy* do projeto e teste. Quando ocorrer um erro, o *CatchAll* vai capturar este erro e chamar os fluxos de compensação.

# Agendar inicialização

Uma instancia do BPEL pode ser iniciada de várias maneiras, algumas são passivas – o BPEL recebe uma chamada e inicia uma instância, outras são ativas – o BPEL procura por um arquivo em um diretório, por exemplo, se encontrar inicia uma instância.

Esta é uma maneira intermediária de executar o BPEL, que seria agendar a inicialização de uma instância para ser executada de tempos em tempos.

Para isto temos algumas opções já que o BPEL não tem um componente específico para fazer esta inicialização.

## Utilização do CONTROL-M

O CONTROL-M é uma excelente ferramenta para agendar a inicialização de um serviço em diversos protocolos diferentes.

O CONTROL-M proporciona um método simples de programar serviços Web como os trabalhos em lote tradicionais. Além disso, permite uma interface de serviços Web bidirecional a partir de aplicativos externos no motor de programação de lotes. Um aplicativo pode chamar um serviço Web solicitando a criação de um trabalho em lote para uma finalidade específica. Outros serviços que podem ser solicitados incluem a iniciação de trabalhos, a manipulação de fluxos de trabalho, o rastreamento do status de trabalhos e o tratamento de alertas de trabalhos.

## Job agendado no banco de dados

Uma das maneiras de se inicializar uma instancia do BPEL seria a criação de um *job* agendado no banco de dados que irá chamar o BPEL através de um WebService, para fazer isto, segue os passos abaixo.

Crie uma procedure no banco de dados que faça a chamada ao endpoint do BPEL exposto como WebService, como no exemplo abaixo.

|  |
| --- |
| create or replace  PROCEDURE CALL\_WS\_PR(  P\_ID IN NUMBER ,  P\_NOME IN VARCHAR2 ,  P\_CARGO IN VARCHAR2)  AS  SOAP\_REQUEST VARCHAR2(30000);  SOAP\_RESPOND VARCHAR2(30000);  HTTP\_REQ UTL\_HTTP.REQ;  HTTP\_RESP UTL\_HTTP.RESP;  L\_LAUNCH\_URL VARCHAR2(240) := 'http://localhost:7001/soa-infra/services/default/BPELTransacao/bpelprocess\_client\_ep?WSDL';  L\_SOAP\_ACTION VARCHAR2(240) := 'http://localhost:7001/process';  BEGIN  SOAP\_REQUEST := '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:exam="http://www.example.org">  <soapenv:Header/>  <soapenv:Body>  <exam:elemento>  <exam:id>' || P\_ID || '</exam:id>  <exam:nome>' || P\_NOME || '</exam:nome>  <exam:descricao>' || P\_CARGO || '</exam:descricao>  </exam:elemento>  </soapenv:Body>  </soapenv:Envelope>';  DBMS\_OUTPUT.PUT\_LINE(SOAP\_REQUEST);  HTTP\_REQ:= UTL\_HTTP.BEGIN\_REQUEST(L\_LAUNCH\_URL, 'POST', 'HTTP/1.1');  UTL\_HTTP.SET\_HEADER(HTTP\_REQ,'Content-Type', 'text/xml'); -- since we are dealing with plain text in XML documents  UTL\_HTTP.SET\_HEADER(HTTP\_REQ,'Content-Length', LENGTH(SOAP\_REQUEST));  UTL\_HTTP.SET\_HEADER(HTTP\_REQ,'SOAPAction' , L\_SOAP\_ACTION); -- required to specify this is a SOAP communication  UTL\_HTTP.WRITE\_TEXT(HTTP\_REQ, SOAP\_REQUEST);  HTTP\_RESP := UTL\_HTTP.GET\_RESPONSE(HTTP\_REQ);  UTL\_HTTP.READ\_TEXT(HTTP\_RESP, SOAP\_RESPOND);  UTL\_HTTP.END\_RESPONSE(HTTP\_RESP);  DBMS\_OUTPUT.PUT\_LINE('-------------------------');  DBMS\_OUTPUT.PUT\_LINE(SOAP\_RESPOND); -- here is the response  END CALL\_WS\_PR; |

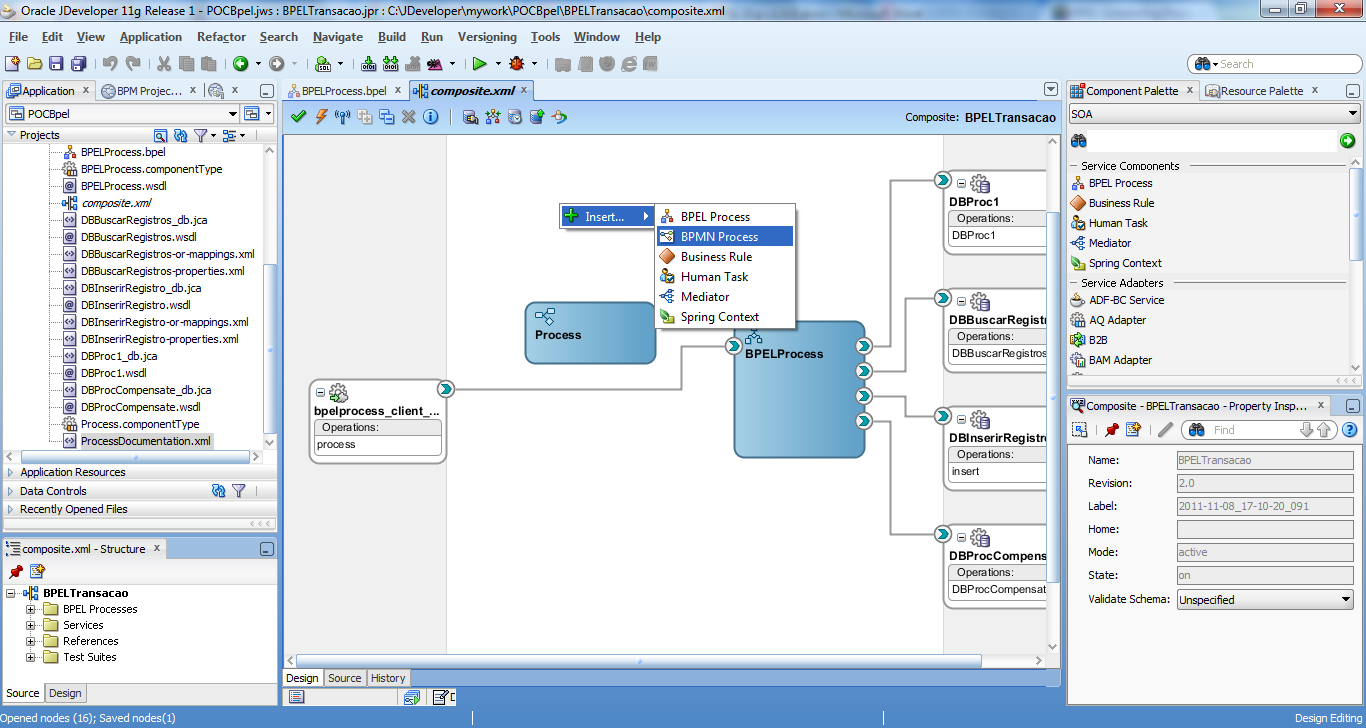
Faça o agendamento desta procedure para rodar de tempos em tempos, segue o exemplo abaixo(no exemplo foi usado o DBMS\_JOB, mas poderia ser outro agendamento de banco de dados, como o DBMS\_SCHEDULER).

|  |
| --- |
| DECLARE  JOB BINARY\_INTEGER;  WHAT VARCHAR2(200);  NEXT\_DATE DATE;  INTERVAL VARCHAR2(200);  NO\_PARSE BOOLEAN;  INSTANCE BINARY\_INTEGER;  FORCE BOOLEAN;  BEGIN  WHAT := 'BEGIN CALL\_WS\_PR(P\_ID => 1, P\_NOME => ''teste1'', P\_CARGO => ''12345678901''); END;';  NEXT\_DATE := SYSDATE;  INTERVAL := 'SYSDATE+1/24';  NO\_PARSE := FALSE;  INSTANCE := 0;  FORCE := FALSE;  SYS.DBMS\_JOB.SUBMIT(  JOB => JOB,  WHAT => WHAT,  NEXT\_DATE => NEXT\_DATE,  INTERVAL => INTERVAL,  NO\_PARSE => NO\_PARSE,  INSTANCE => INSTANCE,  FORCE => FORCE  );  END; |

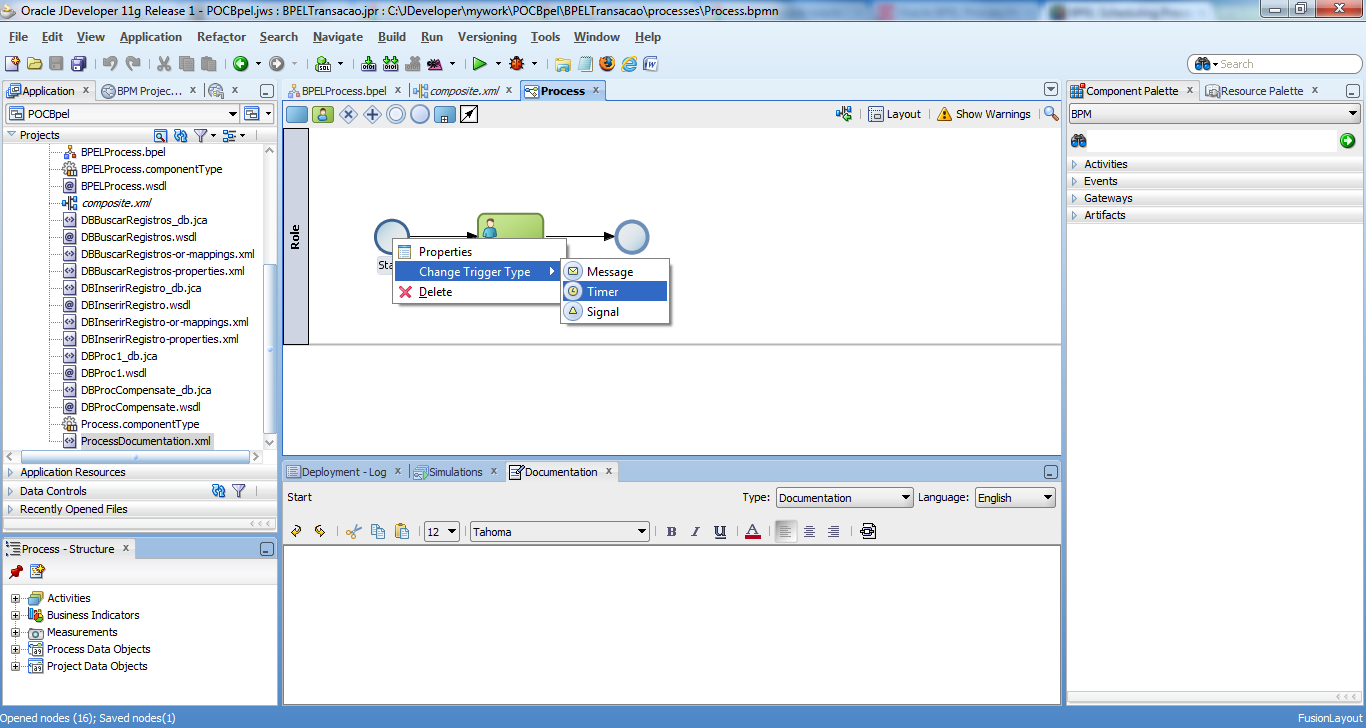
## BPM chamando um BPEL

Outra maneira de se inicializar uma instancia do BPEL é a criação de um fluxo BPM que faça a chamada ao BPEL, isto porque o no BPM temos um evento de inicialização por tempo. Segue o exemplo abaixo.

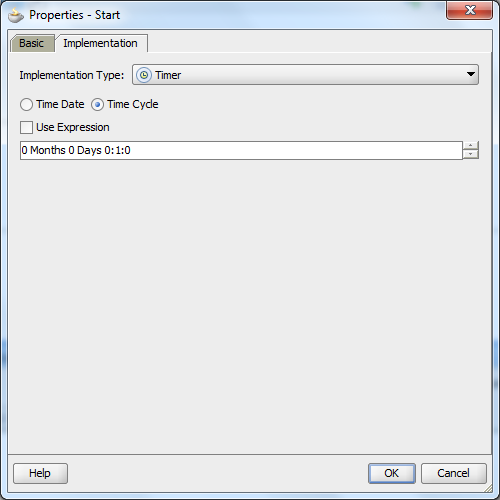
Adicione no mesmo *composite* do BPEL um fluxo BPM.



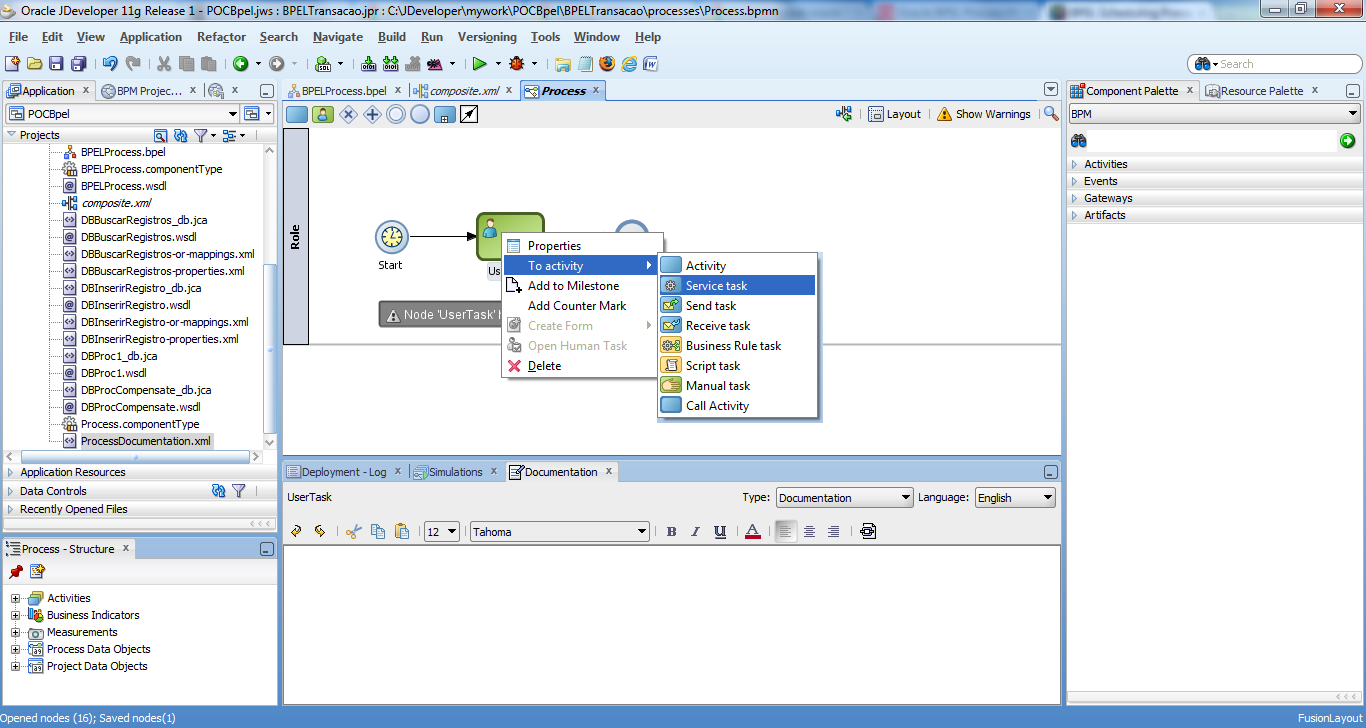
Abra o processo BPM adicionado e mude o evento de *start* para ***timer***.



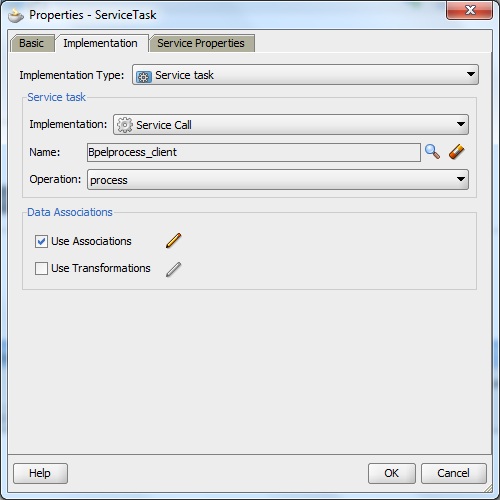
Abra o evento de start e ajuste o tempo que desejar para a inicialização do BPEL. No caso abaixo foi agendado para rodar de hora em hora.



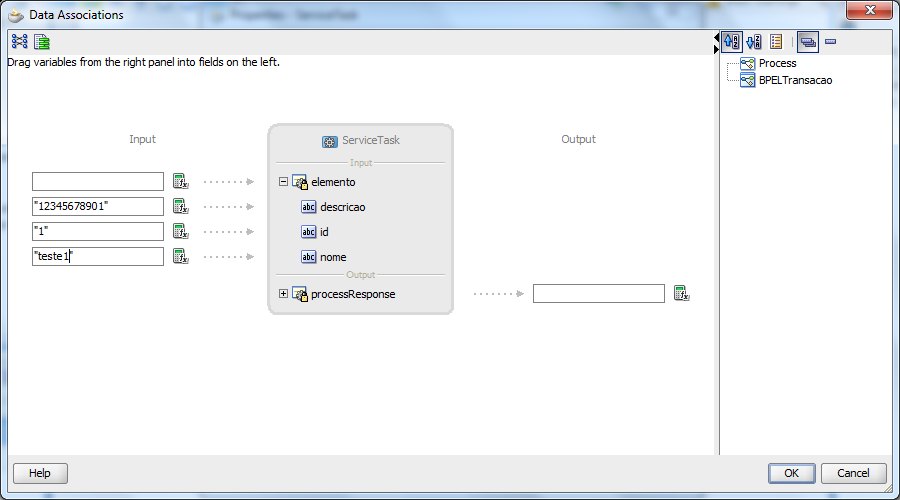
Altere a “*User Task*” para uma “***Service Task***” que ficará responsável por chamar o BPEL.



Vá para a aba *Implementation* e altere o campos para chamar o BPEL que encontra-se no composite, como na figura abaixo.



Marque a caixa ***Use Association***e clique no ícone ao lado para passar os parâmetros necessários para a chamada ao BPEL.



Clique em OK.

Salve tudo e faça *deploy* do projeto.

O BPEL será iniciado de hora em hora pelo evento do BPM.