UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n° 0011/2011

BPEL: Principais Conceitos e Uso Prático

Arthur Ministro Pereira Moreira
Felipe Braga Carneiro Leão
Sandro Pinheiro Lopes
Flávio Faria
Leonardo Guerreiro Azevedo

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

BPEL: Principais Conceitos e Uso Prático

Arthur Ministro, Felipe Leão, Flávio Faria, Sandro Lopes, Leonardo Guerreiro Azevedo

Depto de Informática Aplicada – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{arthur.ministro, felipe.leao, flavio.faria, sandro.lopes, azevedo}@uniriotec.br

Abstract. This work presents the BPEL (Business Process Execution Language) pattern for service orchestration. It focus on presenting the descriptions of essential constructs for its use. The goal of this work is to be a tutorial that makes easy the use of BPEL pattern for developers that have not much experience in working with services. Besides, it is presented a literature review of recent works in orchestration subject.

Keywords: SOA, Service Composition, Service Orchestration, BPEL.

Resumo. Este trabalho apresenta o padrão para orquestração de serviços BPEL (*Business Proccess Execution Language*) focando na descrição dos construtos essenciais ao seu uso. O objetivo deste trabalho é ser um tutorial que facilite o uso do padrão por desenvolvedores com pouca experiência em desenvolvimento de serviços. Além disso, é realizada uma revisão bibliográfica de trabalhos recentes relacionados ao tema orquestração.

Palavras-chave: SOA, Composição de Serviço, Orquestração de Serviços, BPEL.

Sumário

1 Introdução

A Arquitetura Orientada a Serviços (do inglês Service-oriented Architecture - SOA) recebe diversas definições formais, de variados órgãos como a OASIS , o W3C e o Open Group , além de diversos autores, sendo, portanto, uma descrição livre de conteúdo de uma arquitetura de tecnologia da informação [Erickson e Siau, 2008]. Para Josuttis [2007], "SOA é um paradigma para a realização e manutenção de processos de negócio em um grande ambiente de sistemas distribuídos que são controlados por diferentes proprietários". Apesar de várias definições serem aceitas pela comunidade científica, para este trabalho, esta será considerada a definição principal. Neste paradigma arquitetural, as funcionalidades são projetadas e disponibilizadas como serviços, facilitando a interoperabilidade entre os sistemas distribuídos.

O principal conceito em uma arquitetura é o de serviços. Papazoglou *et al.*[2007] define serviços como sendo elementos computacionais auto-contidos, independentes de plataforma que suportam composição fácil, rápida e de baixo custo para aplicações distribuídas, mantendo baixo acoplamento. Serviços podem ser implementados em qualquer plataforma ou tecnologia [Josuttis, 2007]. No entanto, a principal tecnologia para implementação desses serviços é web services [Erl, 2005].

Serviços podem ser construídos considerando a invocação de outros serviços existentes. Esta abordagem é conhecida como composição de serviços, apontada como um dos principais aspectos da computação orientada a serviço que contribui para o reuso dos serviços [Erl, 2007]. Segundo Papazoglou *et al*. [2007] as técnicas mais utilizadas para composição de serviços são orquestração e coreografia.

A orquestração é uma abordagem baseada no modelo workflow, contendo nós (Web services) e o fluxo de dados entre eles. Um processo central controla os web services envolvidos e coordena a execução das diferentes operações. A Figura 1 apresenta um exemplo de orquestração onde o controlador central invoca recebe uma solicitação de um consumidor (passao 1). Em seguida, ele invoca um serviço (passo 2). Ao receber esta resposta, ele invoca outros serviços (passos 3 e 4). Ao concluir o processamento, o controlador central retorna uma resposta para o consumidor (passo 5). Apesar de facilitar a implementação, administração e monitoramento, orquestração possui limitações de escalabilidade.

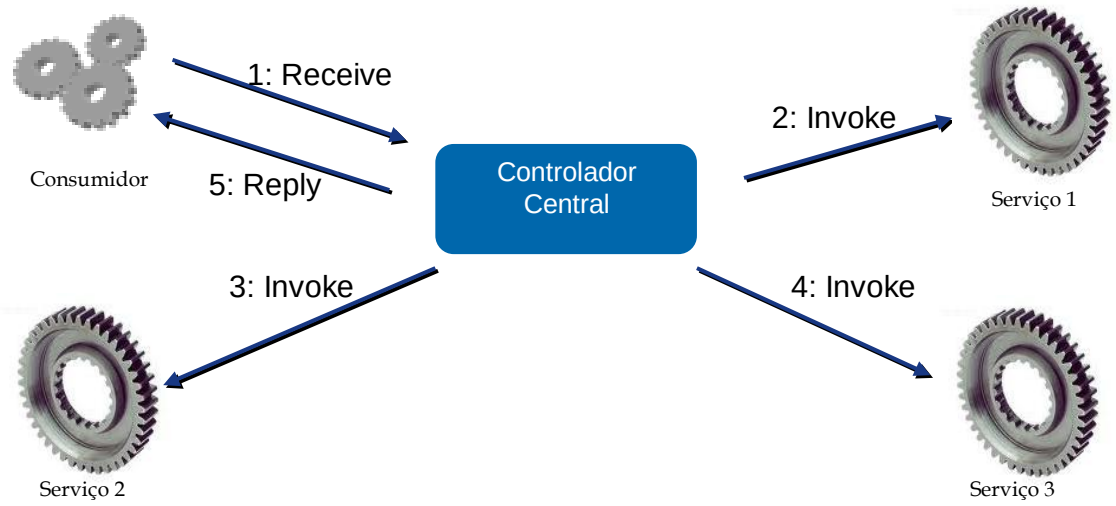

Figura 1 – Exemplo de orquestração

Na Coreografia a colaboração entre serviços é descentralizada e um conjunto de mensagens são trocadas entre os serviços. Os serviços envolvidos precisam conhecer como e quando interagir com os outros serviços, ao invés de se comunicar com uma máquina central, resultando em aumento de escalabilidade. A Figura 2 apresenta um exemplo de coreografia. O processo inicia com a invocação do serviço 1 que realiza alguma tarefa. Após a conclusão tarefa, o serviço 2 é invocado para realizar outra tarefa. Seguida os serviços 3 e 4 e assim sucessivamente. Cada serviço realiza uma tarefa ou atividade e iniciar um ou mais serviços consecutivos que realizam outras tarefas ou atividades. Comparando coreografia em relação à orquestração, existe maior dificuldade para implementação, administração e monitoramento de coreografias. Problemas decorrentes da execução são difíceis de gerenciar e a localização dinâmica de um serviço não é fácil de realizar [Pedraza e Estublier, 2009].

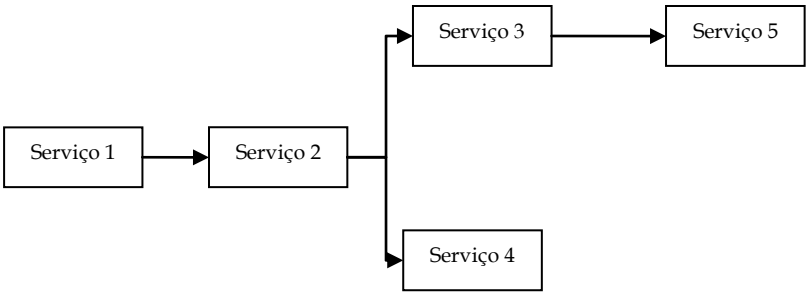

Figura 2 – Exemplo de coreografia [Josuttis, 2007]

BPEL (Business Process Execution Language) é uma linguagem utilizada para definição e execução de um processo de negócio através da orquestração de web services. Permite a realização de SOA através de uma abordagem top-down para composição, orquestração e coordenação de web services de forma simplificada.

BPEL é a junção de duas linguagens de workflow Web Services Flow Language (WSFL) e XLANG. WSFL é uma linguagem da IBM para a descrição de composição de web services. XLANG é uma linguagem da Microsoft para especificação de troca de mensagens entre web services oferecendo uma forma para orquestrar aplicações.

A primeira versão foi desenvolvida em Agosto de 2002 e em abril de 2003 foi submetida à OASIS (Organization for the Advancement of Structured Information Standards) [Alves *et. al.*, 2007].

Um processo BPEL é exposto como um serviço (web service) que compõe serviços existentes, utilizando as técnicas mencionadas. Um processo BPEL provê acesso às suas operações através de *port types* como qualquer web service. Para invocar um processo BPEL basta invocar o serviço resultante da composição.

Os processos de negócio são a principal fonte de informações das organizações. Neles estão definidas as seqüências de atividades que estão alinhadas com os objetivos estratégicos da organização. Tornar os processos executáveis com a possibilidade de utilizar parte de processos de outras organizações, com objetivo de atingir metas ainda maiores, tornou-se uma necessidade fundamental para as organizações. Com isso as principais motivação para o uso de BPEL são [Juric, 2011]:

- Simplificar a integração entre parceiros de negócio;
- Incentivar as organizações a definirem melhor os seus processos, tal melhora leva à otimização da organização;
- Facilitar a exposição das funcionalidades da organização através de serviços (web services).

Este material é um guia básico para o uso de BPEL e está estruturado da seguinte forma. A Seção 1 apresenta a motivação para o uso de BPEL. A Seção 2 apresenta com maiores detalhes os construtos e o fluxo de implementação de um processo BPEL. A Seção 3 fornece um tutorial de implementação de um processo BPEL. A Seção 4 apresenta trabalhos relacionados. Por fim, a Seção 5 apresenta a conclusão.

Os exemplos apresentados neste material são baseados no processo Cotação Viagem e foram elaborados utilizando as ferramentas *Oracle JDeveloper* BPEL *Designer* e *Oracle BPEL Process Manager*. Maiores detalhes sobre a ferramenta Oracle BPEL Process Manager podem ser obtidos em Oracle [2006] e Azevedo *et al.* [2009].

2 O Padrão WS-BPEL

Como dito anteriormente, a primeira versão do padrão foi desenvolvida em Agosto de 2002 e submetida à OASIS em abril de 2003. O processo BPEL especifica a seqüência em que os serviços serão executados nos parceiros envolvidos na orquestração. Este processo é organizado através de atividades, que podem ser utilizadas para receber uma requisição no processo, invocar parceiros ou retornar a resposta do processo ao parceiro externo [Alves *et. al.*, 2007].

O Padrão WS-BPEL comporta atividades (como *invoke*, *receive* e *reply*) que são utilizadas para acessar parceiros de negócio (identificados por *port types*). Este acesso é realizado por um canal de comunicação definido via *partnerlink* e os dados são transferidos através de variáveis. Esta seção apresenta especificações de alguns dos construtos do padrão exemplificando seus usos.

2.1 Partner Links

Um *partnerlink* define um canal de comunicação entre parceiros de negócio através do mapeamento entre o processo BPEL e o *portype* do serviço definido no WSDL (Web

Service Description Language [Christensen *et. al.*, 2001]). Basicamente o *partnerlink* é a identificação dentro de um processo referente à um parceiro, enquanto o *port type* é o portão de acesso à um serviço específico provido por este parceiro. A Figura 3 exemplifica graficamente a interação com um *partnerlink* em um processo BPEL.

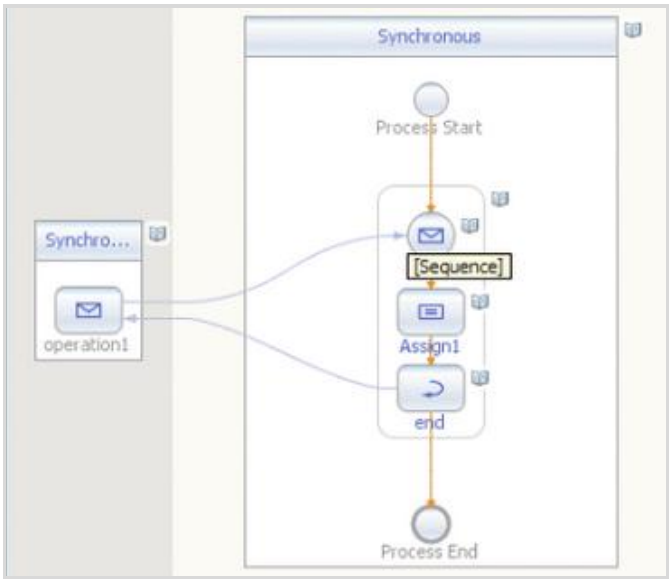

Figura 3 - Exemplo de interação com *Partner Link*

Na Figura 3, o processo é iniciado quando o parceiro realiza uma invocação enviando um determinado dado ao processo. Do ponto de vista do processo isto seria uma atividade *receive*. Uma atribuição então é feita pela atividade *assign* (as atividades serão melhor explicadas posteriormente) e então o processo envia o dado processado novamente para o parceiro através do acesso ao *partnerlink*, através de uma atividade *reply*.

A Figura 4 mostra o trecho de código que define um *partnerlink*. Neste caso, o exemplo é relacionado ao tutorial que será aplicado posteriormente neste trabalho. São definidos 3 *partnerlinks* referentes ao cliente que acessa o processo (*client*) e as duas empresas aéreas que serão consultadas (*DeltaAirline* e *AmericanAirline*).

```
<partnerLinks>
    <partnerLink name="client" partnerLinkType="client:CotacaoViagem"
                 myRole="CotacaoViagemProvider"
                 partnerRole="CotacaoViagemRequester"/>
    <partnerLink myRole="airlineCustomer" name="AmericanAirline"
                 partnerRole="airlineService"
                 partnerLinkType="nsl:flightLT"/>
    <partnerLink myRole="airlineCustomer" name="DeltaAirline"
                 partnerRole="airlineService"
                 partnerLinkType="nsl:flightLT"/>
</partnerLinks>
```

Figura 4 - Exemplo *partnerlink* do processo Cotação Viagem

No entanto, é necessário especificar a ligação entre os serviços, definindo o papel (*role*) de cada um na conversação e especificando o *portType*[1] em que cada um dos serviços receberão suas mensagens. Para isso utilizamos uma extensão do WSDL chamada *partnerLinkType* (Figura 5).

[1] Através do portType uma determinada ação do partnerLink é acessada.

```
<plnk:partnerLinkType name="flightLT">
  <plnk:role name="airlineService">
    <plnk:portType name="tns:FlightAvailabilityPT" />
  </plnk:role>
  <plnk:role name="airlineCustomer">
    <plnk:portType name="tns:FlightCallbackPT" />
  </plnk:role>
</plnk:partnerLinkType>
```

Figura 5 – Exemplo *partnerLinkType* do serviço *Airline*

Para interações síncronas entre o processo e o parceiro (canal unidirecional) apenas um role é definido no *partner link type* correspondente. A Figura 6 [Nitzsche *et. al., 2007*] demonstra uma comunicação síncrona entre o processo BPEL e um serviço. Neste caso, o processo aguarda por uma resposta do serviço para continuar a execução.

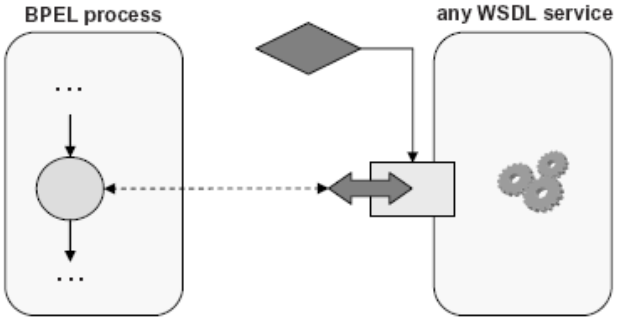

Figura 6 - Interação Síncrona

Para interações assíncronas, onde temos um canal bidirecional entre o processo e o parceiro, devemos especificar dois roles. Um para especificar o papel do processo (*myRole*) e outro para o parceiro (*partnerRole*). A Figura 7 [Nitzsche *et. al., 2007*] ilustra a comunicação assíncrona, onde o processo invoca o serviço e continua a sua execução. Ao finalizar o seu processamento, o serviço comunica o processo através de uma operação denominada *callback*.

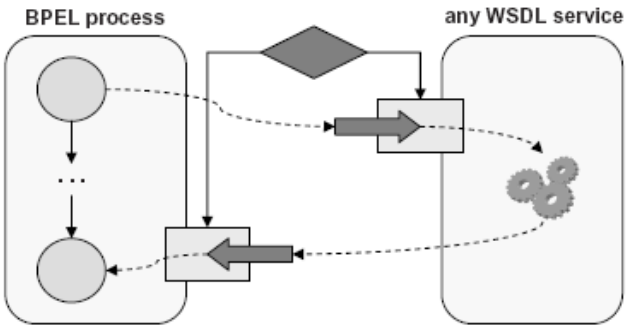

Figura 7 - Interação Assíncrona

Para invocar serviços assíncronos BPEL oferece a atividade *flow* utilizada para prover concorrência e sincronização para um conjunto de atividades. Maiores detalhes sobre a atividade *flow* serão apresentados na seção de atividades estruturadas deste material.

2.2 Variáveis

Variáveis podem ser utilizadas em um processo BPEL para armazenar, formatar e transformar mensagens que são trocadas entre os parceiros. Elas são necessárias para toda mensagem enviada a um parceiro e recebida por ele. Variáveis podem ser definidas com escopo global, neste caso estão acessíveis por todo o processo, ou podem ser definidas em um escopo específico (*scope*), disponível apenas para as atividades contidas neste escopo. Para cada variável é necessário definir um tipo: WSDL Message Type, simple type ou element.

- Simple type: armazena qualquer dado simples definido pelo XML Schema (string, integer, boolean, float)
- WSDL Message Type: armazena o conteúdo de uma mensagem WSDL enviada ou recebida pelos parceiros
- Element: armazena tanto elementos simples quanto elementos complexos definidos em um XML Schema ou em um arquivo WSDL.

A Figura 8 apresenta exemplos de algumas das variáveis declaradas no processo Cotação Viagem. É possível ver que o tipo escolhido foi WSDL Message Type (já que é descrita como *messageType*) para todas as variáveis exceto as duas últimas que são do tipo Simple Type (definida por *type*).

```
<variables>
      <variable name="inputVariable"
                messageType="client:CotacaoViagemRequestMessage"/>
    <variable name="outputVariable"
                messageType="client:CotacaoViagemResponseMessage"/>
    <variable name="FlightResponseDA"
                messageType="client:CotacaoViagemResponseMessage"/>
    <variable name="FlightResult"
                messageType="client:CotacaoViagemResponseMessage"/>
    <variable name="FlightPriceAA" type="xsd:int"/>
    <variable name="FlightPriceDA" type="xsd:int"/>
</variables>
```

Figura 8 – Variáveis do processo Cotação Viagem

2.3 Atividades

Os elementos que compõem um processo BPEL são denominados atividades. Atividades podem ser classificadas como básicas ou estruturadas. Nesta seção serão apresentadas as atividades suportadas pelo WS-BPEL.

2.3.1 Atividades Básicas

O conjunto de atividades básicas é composto pelas seguintes atividades:

- *invoke*: invoca uma operação no serviço web;
- *receive*: recebe a mensagem de uma fonte externa;
- *reply*: envia uma resposta para uma fonte externa;
- *waiting*: realiza uma pausa por um período especificado;
- *assign*: utilizada para copiar dados;
- *throw*: levantar erros na execução do processo;

- *terminate*: finaliza a execução de uma instância do serviço;
- *compensate*: desfaz alterações em caso de erro.

2.3.2 Atividades Estruturadas

O conjunto de atividades estruturadas é utilizado para agrupar atividades básicas dentro de estruturas de fluxo, resultando em estruturas complexas.

- *sequence*: define a ordem de execução;
- *switch*: para lógica condicional;
- *while*: para laços (loop);
- *pick*: possibilita o bloqueio do fluxo no processo até que um determinado evento ocorra (usualmente a recepção de uma mensagem) ou aconteça o timeout;
- *flow*: para fluxos paralelos;
- *scope*: define um escopo para agrupamento de atividades, criação de variáveis locais, tratamento pelo mesmo manipulador de erro (*fault-handler*) e levantamento de exceções.

Nesta seção, será apresentado o uso das seguintes atividades: *sequence*, *flow* e *switch.*Tal uso será apresentado através de um exemplo de processo BPEL correspondente à Figura 9. O processo exemplo envolve a cotação de uma passagem de avião entre duas companhias, o processo deve retornar a companhia que possui o melhor preço e o valor cobrado pela companhia pela passagem.

A atividade *flow* (Figura 9) <flow>, após invocada, termina apenas quando todas as atividades contidas nela estiverem finalizadas (</flow>). A atividade *sequence* define a sequência com que as atividades serão executadas em um processo BPEL. A invocação ocorre em dois passos: A atividade <*invoke*> é utilizada para invocar o serviço do parceiro. A invocação é feita de forma assíncrona. Neste caso, os serviços das companhias aéreas *American Airlines e Delta Airlines* são invocados em paralelo. A atividade <*receive*> é utilizada para receber as resposta dos serviços das companhias aéreas para o processo. O serviço realiza uma invocação *callback* do processo. Vale observar que, se tratando de uma atividade para receber um *callback*, a atividade <*receive*> só será utilizada em caso de chamadas assíncronas a serviços.

```
<flow name="Flow_Airline">
    <sequence name="Sequence_AA">
        <invoke name="Invoke_AmericanAirline"
                partnerLink="AmericanAirline"
                portType="ns1:FlightAvailabilityPT"
                operation="FlightAvailability"
                inputVariable="Invoke_AmericanAirline_FlightAvailability_InputVariable"/>
        <receive name="Receive_AmericanAirline"
                 partnerLink="AmericanAirline"
                 portType="ns1:FlightCallbackPT"
                 operation="FlightTicketCallback"
                 variable="Receive_AmericanAirline_FlightTicketCallback_InputVariable"
                 createInstance="no"/>
    </sequence>
    <sequence name="Sequence_DA">
        <invoke name="Invoke_DeltaAirline" partnerLink="DeltaAirline"
                portType="ns1:FlightAvailabilityPT"
                operation="FlightAvailability"
                inputVariable="Invoke_DeltaAirline_FlightAvailability_InputVariable"/>
        <receive name="Receive_DeltaAirline" partnerLink="DeltaAirline"
                 portType="ns1:FlightCallbackPT"
                 operation="FlightTicketCallback"
                 variable="Receive_DeltaAirline_FlightTicketCallback_InputVariable"
                 createInstance="no"/>
    </sequence>
</flow>
```

Figura 9 - Atividades flow e sequence

O bloco *sequence* define execução de atividades em sequência. Por exemplo, na Figura 9, o sequence *Sequence_AA* define a execução de uma atividade *invoke* e uma *receive.* Na atividade *invoke*, é invocada a operação *FlightAvailability* no *partnerLink AmericanAirline* através do *portType ns1:FlightAvailabilityPT* passando como argumento os dados da viagem, armazenados na variável *Invoke_AmericanAirline_FlightAvailability_InputVariable.* Como a invocação do serviço é assíncrona, a variável de saída (*output variable*) não é especificada porque a resposta não é esperada como parte da operação.

Ao final do processamento do serviço, a atividade *receive* especifica que o serviço irá invocar a operação *FlightTicketCallback* do processo, através do *portType ns1:FlightCallbackPT*. Esta invocação corresponde ao armazenamento dão valor da cotação calculado pelo serviço na variável *Receive_AmericanAirline_FlightTicketCallback_InputVariable* para ser consumido pelo processo. O elemento *createInstance="no"* da atividade *receive* indica que não deve ser criada uma nova instância do processo (Figura 9).

Após o término do bloco *flow* será necessário verificar qual a melhor cotação entre os serviços executados, neste caso os serviços das companhias *Delta Airlines* e *American Airlines.* A verificação é realizada pela estrutura condicional *switch* conforme ilustrado através do código XML apresentado na Figura 10 e descrito a seguir.

```
<switch name="Switch_FlightResult">
    <case condition="bpws:getVariableData('FlightPriceAA')&lt;bpws:getVariableData('FlightPriceDA')">
        <bpelx:annotation>
            <bpelx:pattern patternName="case">AA</bpelx:pattern>
        </bpelx:annotation>
        <assign name="Assign_AA">
            <copy>
                <from variable="FlightResponseAA" part="payload"
                      query="/client:CotacaoViagemProcessResponse"/>
                <to variable="FlightResult" part="payload"
                    query="/client:CotacaoViagemProcessResponse"/>
            </copy>
        </assign>
    </case>
    <case condition="bpws:getVariableData('FlightPriceDA')&lt;bpws:getVariableData('FlightPriceAA')">
        <bpelx:annotation>
            <bpelx:pattern patternName="case">DA</bpelx:pattern>
        </bpelx:annotation>
        <assign name="Assign_DA">
            <copy>
                <from variable="FlightResponseDA" part="payload"
                      query="/client:CotacaoViagemProcessResponse"/>
                <to variable="FlightResult" part="payload"
                    query="/client:CotacaoViagemProcessResponse"/>
            </copy>
        </assign>
    </case>
    <otherwise/>
</switch>
```

Figura 10 - Estrutura condicional switch

A atividade *switch* é composta de um ou mais blocos *case* e um bloco *otherwise*. O bloco *case* será executado caso a condição avaliada seja verdadeira (*true*). Caso a condição avaliada seja falsa, demais blocos *case (neste caso apenas um)* serão avaliados. Caso nenhum deles seja verdadeiro o bloco *otherwise* será executado. A Figura 10 ilustra o uso da atividade *switch* no processo *Cotação Viagem* para verificar o menor preço dentre as cotações realizadas pelas duas companhias.

O primeiro bloco *case* denominado *case AA* verifica se o preço do bilhete da companhia *American Airlines*, armazenado na variável *FlightPriceAA* é menor do que o preço da *Delta Ailines*, armazenado na variável *FlightPriceDA*. A expressão criada utiliza a função *getVariableData*, utilizada para recuperar valores a partir de variáveis BPEL.

O segundo bloco *case* denominado *case DA* verifica se o preço do bilhete da companhia *Delta Airlines*, armazenado na variável *FlightPriceDA* é menor do que o preço da *American Ailines*, armazenado na variável *FlightPriceAA*. A expressão é criada da mesma forma que a expressão do *case* anterior.

O bloco *otherwise* neste caso não é utilizado, pois o preço das duas companhias não pode ser igual.

A criação dos códigos XML apresentados pode ser auxiliada por ferramentas de modelagem e implementação de processos BPEL, um exemplo de criação é apresentado na Seção 3.

2.4 Correlation Sets

Essencialmente *correlation sets* permitem utilizar um valor único presente no corpo de todas as mensagens trocadas (*orderNumber*, por exemplo) para ligar esta troca de mensagens relacionadas. A idéia é semelhante ao conceito de chave primária de banco de dados. No BPEL, o valor único é utilizado para identificar uma instância do processo pois podem existir múltiplas instâncias de um processo ativas no motor de execução BPEL simultaneamente, portanto deve existir esta preocupação para que as mensagens

enviadas a um processo sejam entregues à instância do processo correta. Por exemplo, a Figura 11 apresenta quatro instâncias de processos em execução (à direita), ressaltando a execução da instância 3 e do *correlation set* correspondente.

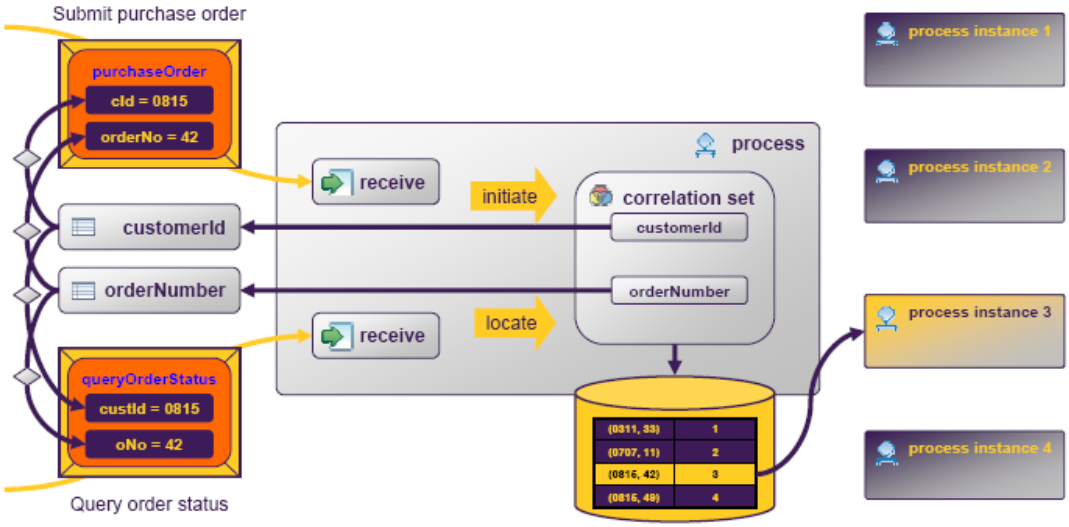

Figura 11 - Ilustração de um Correlation Set [Barreto, 2007]

2.5 Escopos

Um escopo provê um contexto que influencia o comportamento das atividades nele contidas (Figura 12), este contexto comportamental inclui variáveis, *partner links*, trocas de mensagens, *correlation sets*, *event handlers*, *fault handlers*, *compensation handler*, e *termination handler*.

Cada <*scope*> tem uma atividade primária que define seu comportamento normal. A atividade primária pode ser uma atividade estruturada complexa com muitas atividades aninhadas até uma profundidade arbitrária. Todos os outros construtos sintáticos de uma atividade <*scope*> são opcionais. O contexto fornecido por um <*scope*> é compartilhado por todas as suas atividades aninhadas (Figura 13).

As definições feitas dentro de um <*scope*> que fica aninhado a um <*process*> são válidas somente dentro deste <*scope*>, não sendo válidas nos demais escopos.

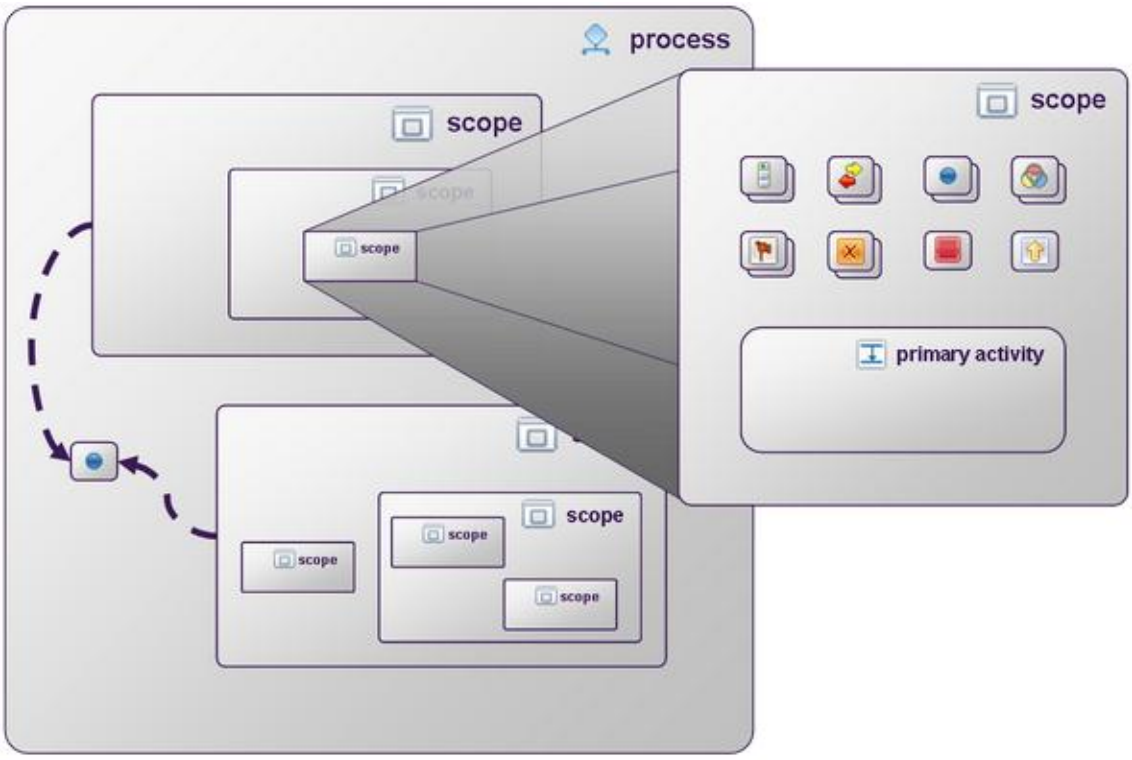

Figura 12 - Ilustração de um Escopo BPEL [Barreto, 2007]

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"?
   standard-attributes>
   standard-elements
   <variables>?
      ...
   </variables>
   <partnerLinks>?
      ...
   </partnerLinks>
   <messageExchanges>?
      ...
   </messageExchanges>
   <correlationSets>?
      ...
   </correlationSets>
   <eventHandlers>?
      ...
   </eventHandlers>
   <faultHandlers>?
      ...
   </faultHandlers>
   <compensationHandler>?
      ...
   </compensationHandler>
   <terminationHandler>?
      ...
   </terminationHandler>
   activity
</scope>
```

Figura 13 - Formato de um escopo em código XML [Alves *et. al.*, 2007]

2.6 Manipulação de compensação

A manipulação de compensação parte do mesmo princípio que a propriedade atomicidade de uma transação em um banco de dados. Caso ocorra algum problema durante a execução da composição e algum dos serviços não possa ser executado (por não se conseguir conexão, por exemplo) ou gere um erro, os passos que já foram executados corretamente devem ser desfeitos, começando pelo último passo executado até chegar-se ao primeiro. Em outras palavras, a compensação deve ser propagada e os passos executados com sucesso devem ser desfeitos. Isto só é possível pois o processo BPEL armazena o estado de execução do processo. É importante dizer que a compensação se aplica a um escopo do processo, este escopo deve estar definido no processo (por atividades <*scope*>), pois um construto *compensation handler* não pode ser associado diretamente a um construto <*process*>.

A Figura 14 apresenta um exemplo de compensação. O processo invoca dois serviços que executam normalmente (passo 1). Quando o terceiro serviço é invocado (passo 2), uma exceção é levantada. A exceção é interceptada (passo 3) e tratada (passo 4), e compensações são propagadas para os serviços que tinham sido executados corretamente (passos 5 e 6).

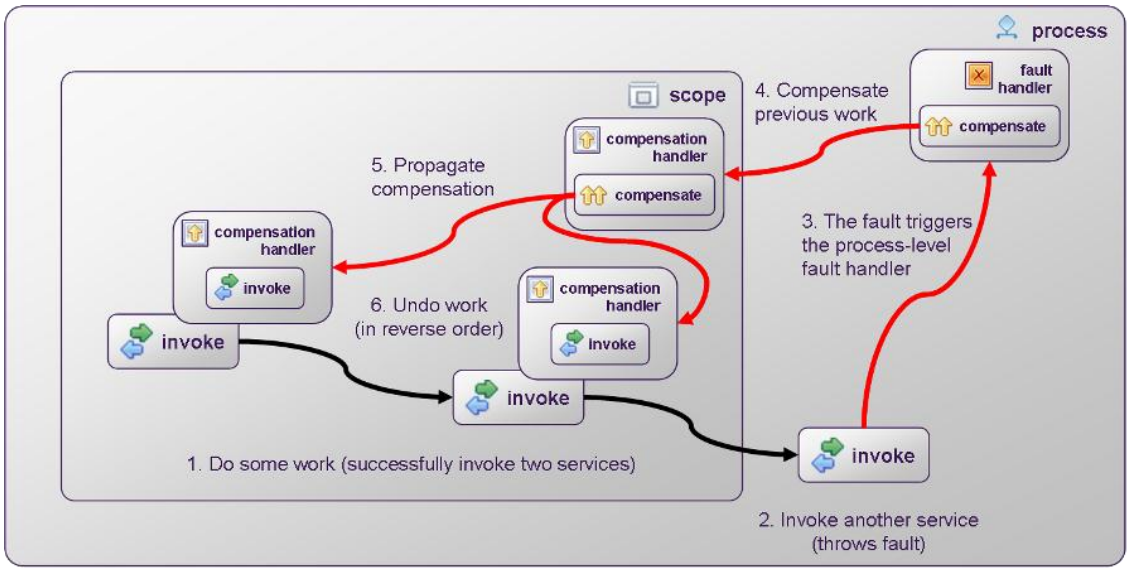

Figura 14 - Propagação de uma Compensação [Barreto, 2007]

3 Exemplo de implementação de um processo BPEL

Com base na teoria explicitada anteriormente, esta seção tem por objetivo guiar o desenvolvedor por um tutorial que exemplifique a implementação de um processo BPEL. Para tal serão utilizadas as ferramentas Oracle BPEL PM[2] e jDeveloper[3] em conjunto com serviços pré-estabelecidos.

[2] http://www.oracle.com/technetwork/middleware/bpel/overview/index.html

[3] http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html

3.1 Configuração e descrição do cenário

São utilizados quatro serviços, sendo dois deles providos pelo conjunto de exemplos do Oracle BPEL PM e dois implementados pelos autores deste relatório. Os serviços providos pelo Oracle BPEL PM estão prontos para uso e deverão ser implantados de acordo com os comandos que serão apresentados a seguir. Os serviços desenvolvidos pelos autores deste documento correspondem a duas diferentes empresas de aluguel de automóveis com filiais na Alemanha. Foi utilizada a IDE de programação Netbeans (versão 7.0) em conjunto com o servidor de aplicação Glassfish (versão 3) e o banco de dados PostgreSQL (versão 9.0.4). Para realizar os acessos ao banco de dados foi utilizado o driver JDBC compatível com esta versão.

O cenário modelo é o de um processo no qual, dada uma data de partida e uma data de retorno para uma determinada viagem, deseja-se consultar serviços de múltiplas companhias aéreas pesquisando o menor preço de passagem e, posteriormente, consultar serviços de diferentes empresas de aluguel de carros e oferecer a melhor cotação total para o usuário. O arquivo do processo básico, os projetos auxiliares e os scripts de criação para o banco de dados podem ser encontrados no arquivo "ArquivosTutorial.zip", disponível em <www.uniriotec.br/~azevedo/BPEL/FontesParaTutorialBPEL.zip>.

Para auxiliar a instalação das aplicações BPEL OC4J e jDeveloper recomendamos o uso do material de Faria e Azevedo [2010] sobre o Oracle BPEL PM e o JDeveloper que contém um passo a passo de como efetuar ambas as instalações.

Utilize os scripts disponibilizados para criar os bancos de dados das duas empresas de aluguel de carros, a Auto Vermietung e Auto Renting (Os *scripts* podem ser encontrados no arquivo "FontesParaTutorialBPEL.zip"). Após a criação dos bancos utilize os fontes disponibilizados para implantar os serviços de ambas empresas no servidor de aplicação Glassfish.

Após a implantação, os serviços devem ser publicados no BPEL PM Server através do comando *obant.* O comando *obant* é uma adaptação realizada pelo Oracle BPEL PM para o construtor de projetos Ant[4]. Para utilizar o comando, abra o "Developer Prompt", que se encontra junto ao ícone para iniciar e parar o servidor BPEL (Figura 15).

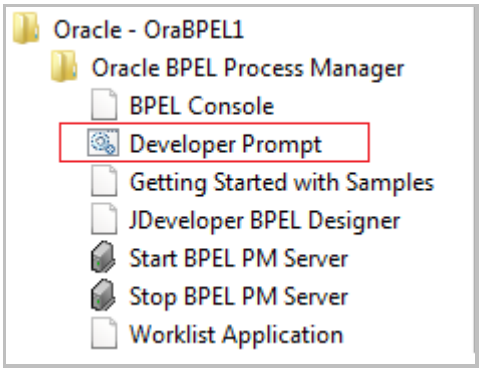

Figura 15 - Caminho para o Developer Prompt

Antes de executar este comando, inicie o BPEL PM Server, o path utilizado no comando *obant* deve seguir algumas regras: evite acentos, cedilhas ("ç") ou qualquer outro caracter não pertencente a codificação UTF-8.

[4] Ferramenta que possibilita definir a ordem de compilação de um projeto através da descrição do processo de compilação em arquivos XML. Maiores detalhes em http://ant.apache.org/

- Bom exemplo: C:\Users\labccet\Desktop\servicos_estudo_caso
- Exemplo ruim: C:\Users\labccet\Desktop\Serviços Estudo de Caso

Dirija-se até o diretório onde se encontram os serviços da American e Delta Airlines e execute o comando *obant* que fará uso do arquivo build.xml. A Figura 16 mostra a execução bem sucedida do comando, que implantará no servidor os serviços *Employee, AmericanAirline, DeltaAirline* e *TravelProcess*.

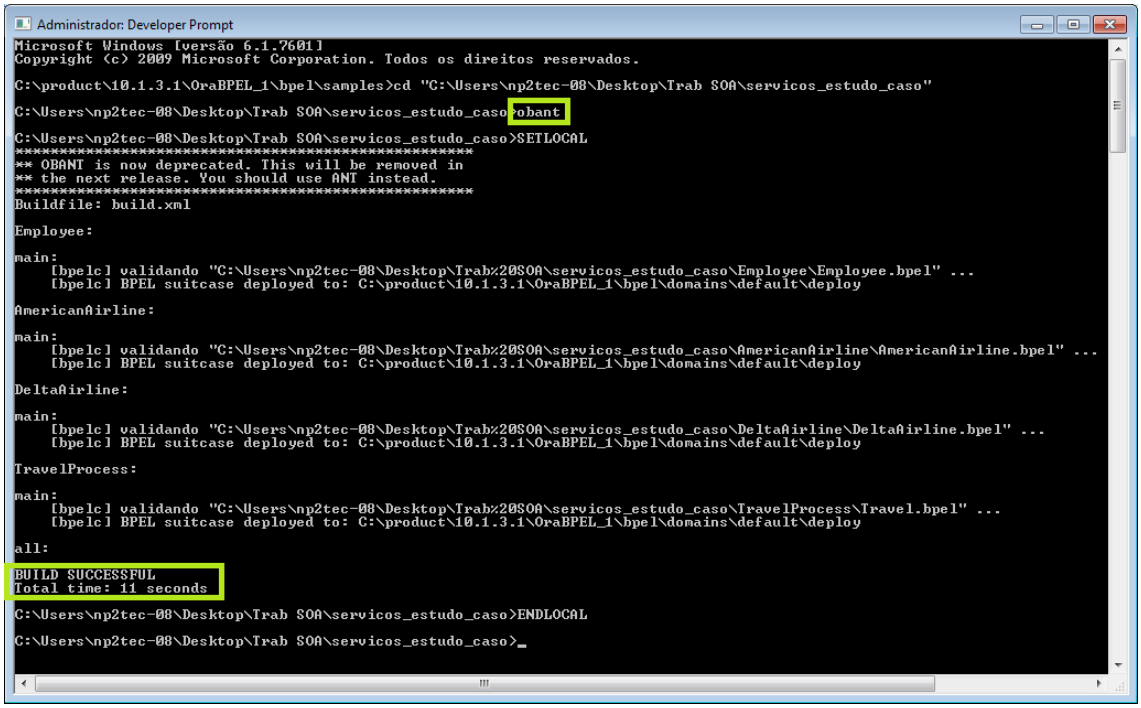

Figura 16 - Execução bem sucedida do comando obant

Após efetuar os passos listados acima inicie o Oracle JDeveloper e abra o projeto **Cotação Viagem**, para isso basta abrir o arquivo "EstudoCaso.jws". A estrutura do projeto é mostrada pela Figura 17. O processo BPEL pode ser visto graficamente através do arquivo "CotacaoViagem.bpel".

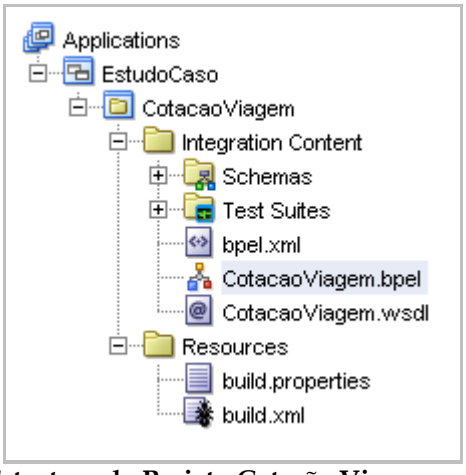

Figura 17 - Estrutura do Projeto Cotação Viagem no jDeveloper

3.2 Processo inicial

O processo BPEL deste projeto possui inicialmente 3 *partnerLinks*: DeltaAirline, AmericanAirline e Client. Os dois primeiros são os serviços que serão utilizados pelo proces-

so para realizar a cotação de passagens aéreas e o terceiro é o cliente por onde será feita a invocação do processo e para onde será enviado o resultado deste processo.

O processo utilizado neste exemplo possui os seguintes tipos de atividade:

- *Receive* – para receber os dados provenientes de um *partnerLink*, seja na invocação do processo (recebendo informações do cliente) ou após uma atividade de *Invoke* à um outro *partnerLink* (para receber informações de um *callback*);
- *Invoke* – para realizar a invocação de um serviço disponibilizado por um parceiro de negócio externo.
- *Assign* – para atribuir a uma variável um determinado valor, como, por exemplo, atribuir a uma variável o resultado proveniente de um acesso à um serviço externo.
- *Flow* – Cria um fluxo paralelo de execução no processo. No processo "CotaçãoViagem" uma atividade *flow* é criada para que os serviços das duas companhias aéreas possam ser acessados simultaneamente.
- *Switch* – Atividade que executa uma operação lógica, optando por um entre múltiplos caminhos a serem seguidos.

Sugerimos que o processo seja executado da forma que foi disponibilizado até este ponto para verificar se tudo está implantado corretamente, para só então executar as alterações propostas neste tutorial. Para executar o processo clique com o botão esquerdo em Cotação Viagem na estrutura do projeto e selecione a opção BPEL Process Deploy no submenu Deploy (Figura 18). Sugerimos também que o modelo seja estudado para entendimento dos conceitos apresentados até o momento.

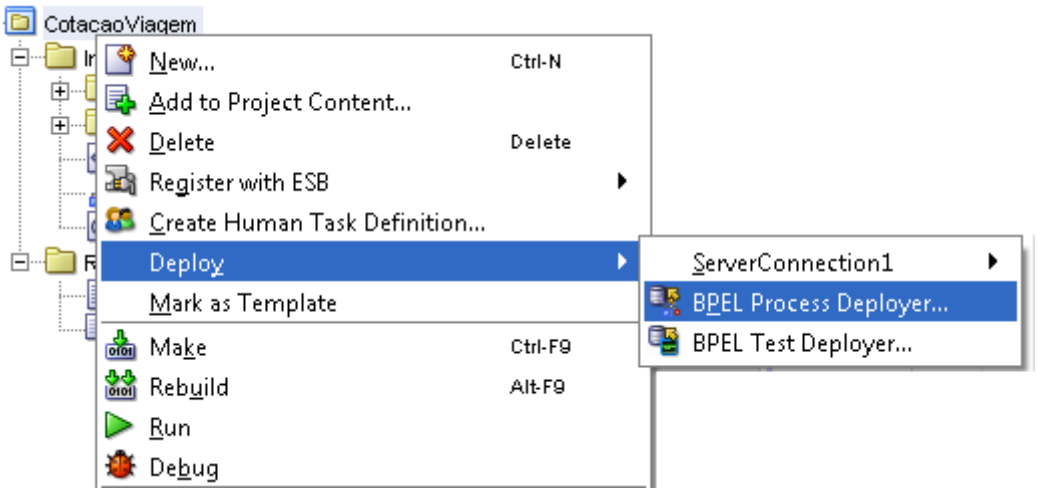

Figura 18 - Caminho para execução do Projeto

Na tela seguinte, utilize uma ServerConnection disponível. Uma ServerConnection têm a função de especificar o local em que o servidor de aplicação BPEL PMServer está disponível para ser acessado, caso não exista uma ServerConnection ela deverá ser criada através das instruções fornecidas em [Faria e Azevedo, 2010]. O console do Ant no jDeveloper irá fornecer uma mensagem de sucesso na construção do projeto (Figura 19), que será implantado no servidor BPEL e acessível através de um WSDL.

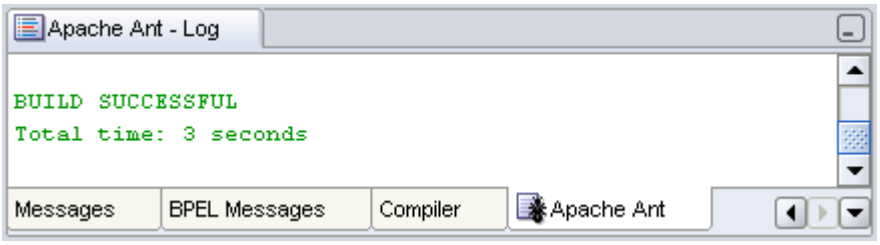

Figura 19 - Construção do Projeto BPEL

Acesso o servidor BPEL através da URL <http://localhost:9700/BPELConsole> e selecione o processo "CotacaoViagem". Na tela seguinte, exibida pela Figura 20, preencha os campos e clique no botão "Publicar Mensagem XML" para ver o processo em execução.

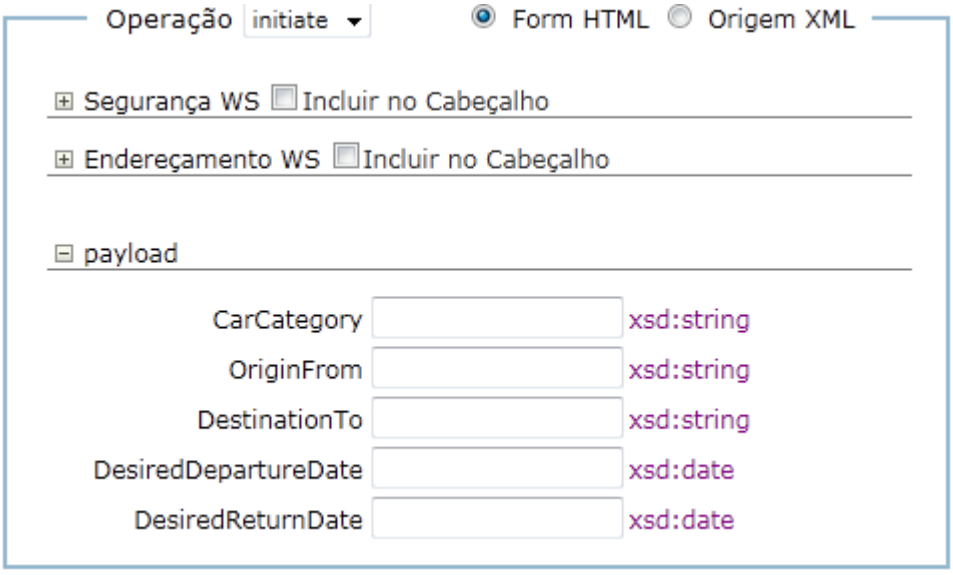

Figura 20 - Passagem de parâmetros para teste do processo CotaçãoViagem

Este processo, por enquanto, fará uso apenas dos serviços de consulta a preço de vôo das companhias American Arilines e Delta Airlines. Estes serviços estão implementados internamente no BPEL. Estes dois serviços retornam valores fixos independente dos valores passados como parâmetro; portanto, neste ponto do tutorial os valores passados como dado de entrada na tela exibida pela Figura 20 ainda não farão diferença.

Na página seguinte selecione "Fluxo Visual" para ver a execução do processo em tempo real. Ao clicar na atividade "CallbackClient" será exibida uma nova página com os dados retornados pelo processo (Figura 21). Observe que como ainda não foram inseridos os serviços para cotação de aluguel de carros, as variáveis "CarPrice", "CarBrand" e "CarModel" se encontram com valor vazio.

Figura 21 - Dados retornados pelo processo Cotação Viagem

A fase seguinte deste tutorial se concentra em alterar o processo para considerar também os serviços implantados no Glassfish para aluguel de carro.

A implantação dos serviços AutoRenting e AutoVermietung depende da criação dos respectivos bancos de dados no PostgreSQL[5]. Para criar os bancos, basta executar os scripts de criação fornecidos no pacote "FontesParaTutorialBPEL.zip" (disponibilizados em <www.uniriotec.br/~azevedo/BPEL/FontesParaTutorialBPEL.zip>).

Após a criação do banco de dados, abra a IDE de programação Netbeans e os os projetos "AutoRenting" e "AutoVermietung" fornecidos junto ao pacote "ArquivosTutorial.zip". Verifique se ambos os projetos possuem em seu *classpath* o arquivo jar do drive JDBC para acesso ao banco de dados. O driver referente à versão 9.0.4 do PostgreSQL se encontra junto aos projetos.

Com os bancos criados, os serviços implantados e BPEL configurado, é então possível realiziar as modificações necessárias ao projeto para que este realize também a cotação de aluguel de carros.

Inicialmente, deve-se considerar que poderia ser necessário adicionar novas variáveis aos parâmetros de entrada ou aos valores de retorno do processo, neste caso o arquivo CotacaoViagem.xsd deveria ser alterado para que os tipos complexos de entrada ou saída comportassem os novos valores. A Figura 22 mostra o arquivo XSD que comporta a definição dos tipos complexos de entrada e saída do processo. Este tutorial não requer que o leitor realize alterações no XSD, pois o processo fornecido no pacote "ArquivosTutorial.zip" já possui todas as variáveis de entrada (*input*) que serão utilizadas para acessar os serviços das lojas de aluguel e a variável de saída do processo já possui as variáveis que informarão os dados do aluguel cotado (CarPrice, CarBrand, CarModel).

[5] Cada serviço desenvolvido possui uma classe DAO que faz acesso ao banco de dados PostgreSQL. A senha utilizada em cada DAO deve ser igual à senha do banco PostgreSQL instalado.

```xml
<schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        targetNamespace="http://xmlns.oracle.com/CotacaoViagem"
        xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="CotacaoViagemProcessRequest">
  <complexType>
   <sequence>
    <element name="CarCategory" type="string"/>
    <element name="OriginFrom" type="string"/>
    <element name="DestinationTo" type="string"/>
    <element name="DesiredDepartureDate" type="date"/>
    <element name="DesiredReturnDate" type="date"/>
   </sequence>
  </complexType>
 </element>
 <element name="CotacaoViagemProcessResponse">
  <complexType>
   <sequence>
    <element name="result" type="string"/>
    <element name="FlightNo" type="string"/>
    <element name="FlightPrice" type="float"/>
    <element name="DepartureDateTime" type="dateTime"/>
    <element name="ReturnDateTime" type="dateTime"/>
    <element name="Approved" type="boolean"/>
    <element name="CarPrice" type="float"/>
    <element name="CarBrand" type="string"/>
    <element name="CarModel" type="string"/>
   </sequence>
  </complexType>
 </element>

</schema>
```

Figura 22 - Arquivo XSD do porcesso BPEL

É importante entender o que o processo em seu estado atual realiza. O usuário informa um conjunto de dados ao realizar a invocação do processo, esses dados são:

- CarCategory: uma letra que identifica a classe do carro a ser cotado;
- OriginFrom: A cidade de Origem do usuário;
- DestinationTo: A cidade para a qual o usuário se dirige, ou seja a cidade onde será feito o aluguel do carro;
- DesiredDepartureDate: A data na qual o usuário sairá de sua origem; e
- DesiredReturnDate: A data na qual o usuário irá voltar para sua origem.

O processo recebe os valores passados pelo *partnerLink* "Client" como parâmetro através de uma atividade *ReceiveInput*. Esta atividade inicializa uma variável que armazena todos os dados recebidos. Logo, a seguir, o processo inicia uma atividade *Flow* (Figura 23), onde são executadas paralelamente duas atividades *Invoke* (*Invoke_AmericanAirline*) e (*Invoke_DeltaAirline*). Cada uma das atividades *Invoke* acessa o serviço de uma das companhias aéreas através dos seus respectivos *partnerLinks* e executa os métodos de cotação de passagem aérea. Sendo estes dois métodos aderentes ao padrão de troca de mensagens "assíncrono" e o processo necessitar receber o valor das cotações, é necessário que haja, logo após cada invocação, uma atividade *Receive* (*Receive_AmericanAirline*) e (*Receive_DeltaAirline*). Estas atividades *Receive* que será utilizada

pelos serviços para retornar ao processo os dados resultantes da consulta à cotação. Neste caso, os serviços fazem uma chamada *callback*.

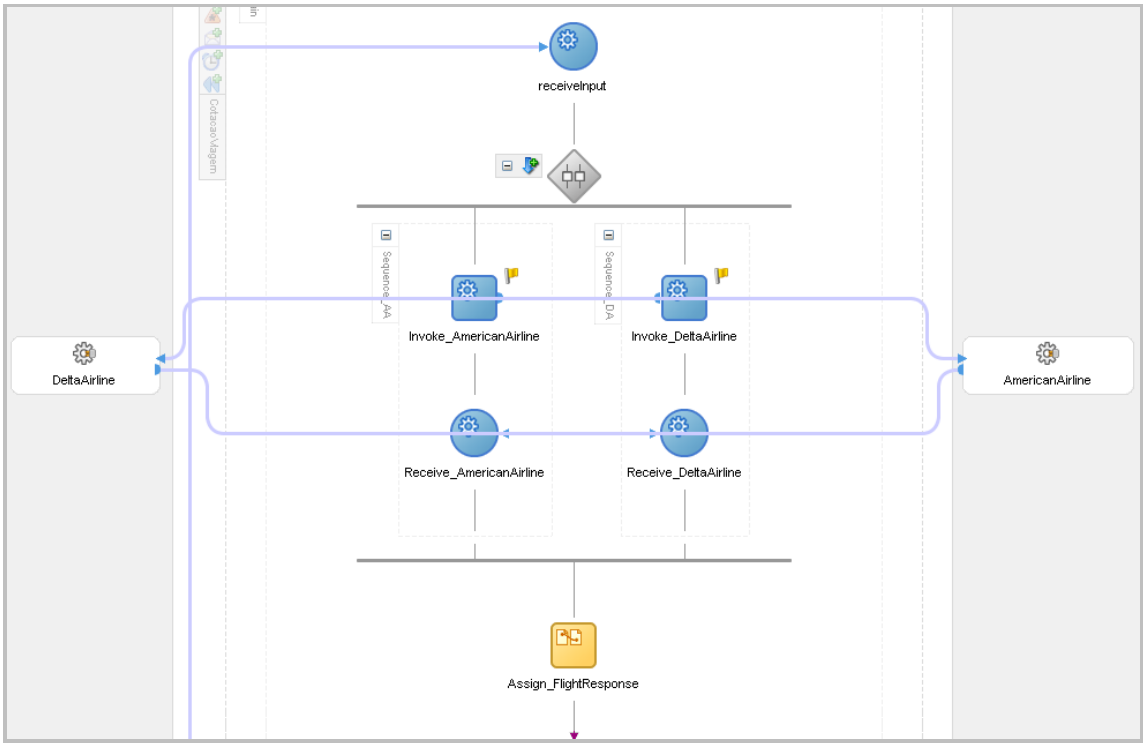

Figura 23 - Atividade Flow para invocar serviços assíncronos

Com os valores de cada companhia aérea, o processo então sai do *Flow* e segue para a atividade *Assign* (*Assign_FlightResponse*). Esta atividade realiza operações de cópia das variáveis recebidas como resposta dos serviços das companhias aéreas para variáveis que são utilizadas em comparações em uma atividade *Switch*. A atividade *Switch* tem como intenção decidir qual das duas companhias aéreas fornece o menor preço de passagem aérea e selecionar a mais barata. A Figura 24 mostra a atividade *switch*. Cada alternativa (*case*) tem uma atividade *Assign* responsável por atribuir à variável que será retornada pelo processo os dados fornecidos pela companhia aérea com menor preço de passagem.

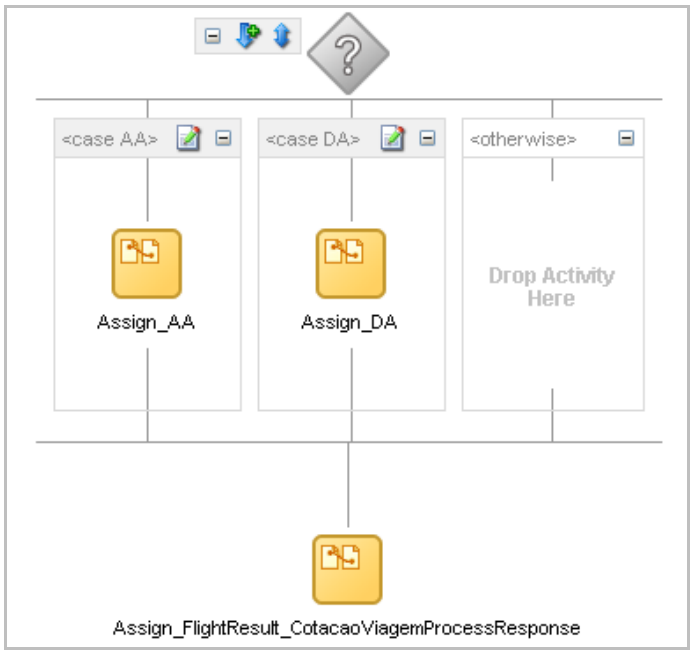

Figura 24 - Atividade Switch para decisão da melhor oferta de Vôo

O processo original tem uma atividade *Invoke* para realizar o *callback* ao *partnerLink Client* com os valores do vôo selecionado, a partir deste ponto então serão inseridas as alterações propostas por este tutorial.

3.3 Evoluções no processo inicial

Esta seção descreve os passos que devem ser executados para evoluir o processo inicial e praticar os conceitos BPEL apresentados.

Insira dois novos *partnerLinks* referentes aos serviços das lojas de aluguel de carros. Para isto clique com o botão direito do mouse na área cinza em torno do processo e selecione a opção "Create Partner Link". A janela exibida pela Figura 25 irá aparecer.

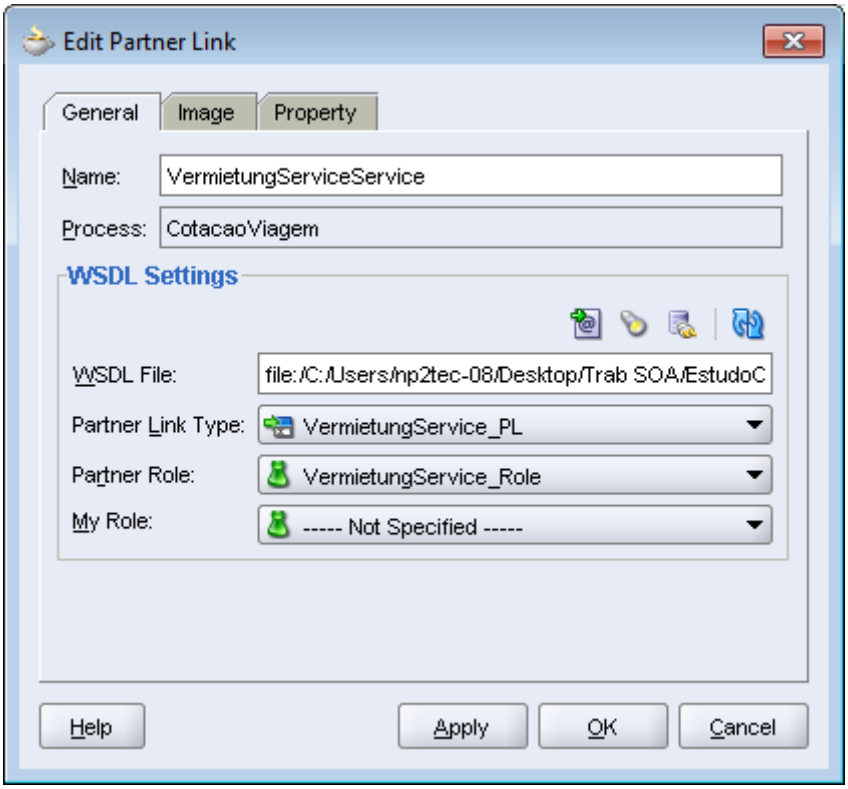

Figura 25 - Tela para criação de um Partner Link

Nesta janela insira no campo "WSDL File" o caminho para o WSDL de um dos serviços publicado no servidor GlassFish. Ao clicar "Enter" o jDeveloper irá preencher automaticamente o campo "Partner Link Type". O campo "Partner Role", entretanto, deverá ser preenchido. Para isto clique na lista e selecione a única opção disponível. Como esta atividade de *Invoke* acessa um serviço síncrono é desnecessário definir um papel no campo "My Role". Repita o mesmo procedimento para criação do Partner Link do outro serviço publicado no servidor GlassFish.

Após a atividade de *Assign* "Assign_FlightResult_CotacaoViagemProcessResponse", insira uma nova atividade *Flow*. Insira duas atividades de *Invoke*, uma em cada fluxo paralelo. Cada uma destas atividades deverá invocar o serviço de uma das lojas de aluguel de carros. Para isto, clique em uma das setas ao lado das atividades e arraste até um dos *partnerLinks*. A Janela exibida pela Figura 26 mostra a tela que aparecerá ao arrastar a seta do *Invoke* para o *partnerLink*.

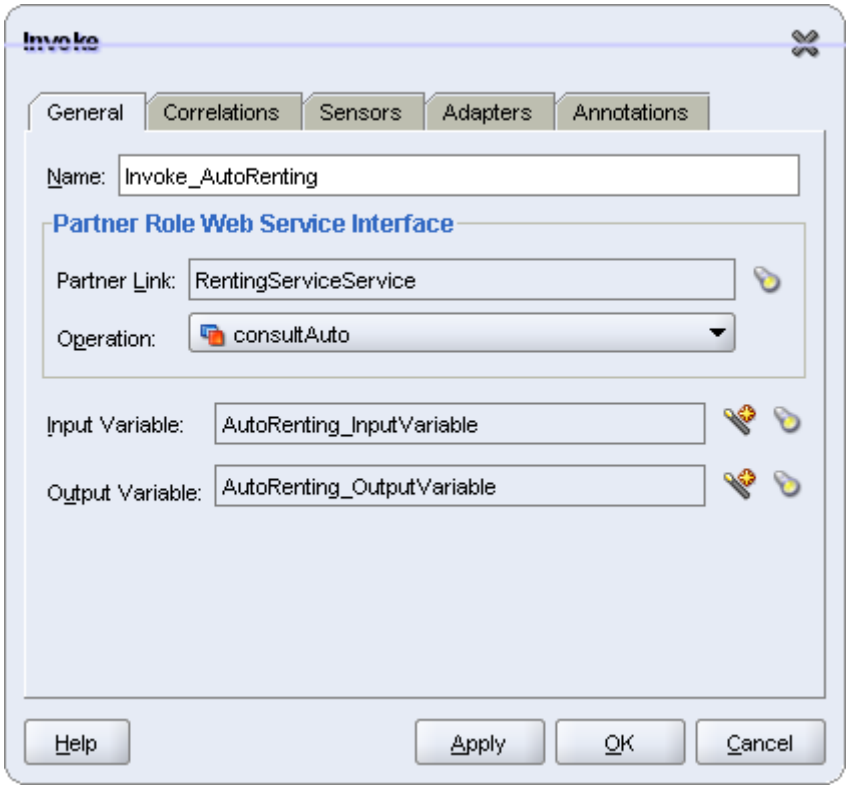

Figura 26 - Configuração de uma atividade *Invoke*

A operação será automaticamente selecionada, já que o serviço disponibiliza apenas uma operação. Duas variáveis deverão ser criadas, uma contendo os dados que serão passados ao serviço e outra para armazenar os dados de retorno da operação. Ao clicar na varinha mágica ao lado de cada campo a aplicação exibirá uma nova janela para auxiliar na criação de variáveis (Figura 27), esta janela já trará definido o tipo da variável a ser criado, dado que a atividade de *Invoke* já se encontra ligada a um *partnerLink*, de forma que a aplicação é capaz de identificar através do XSD do serviço os tipos de dados esperados.

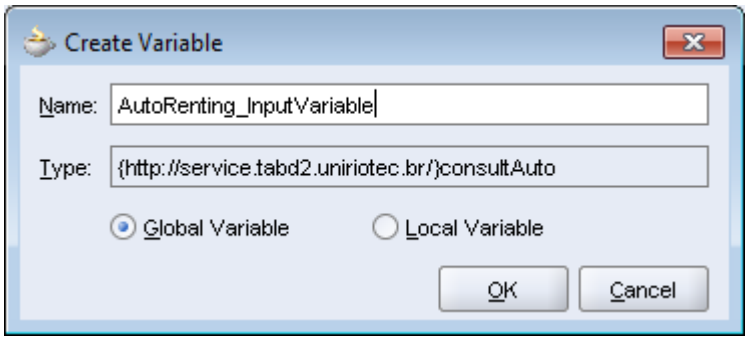

Figura 27 - Janela para criação de Variáveis

A variável a ser utilizada como *input* na invocação do serviço foi criada, mas ainda se encontra sem dados, ou seja, ela foi apenas declarada. Para preencher os campos da variável, crie uma atividade *assign* antes de cada *Invoke*. Estas atividades têm o objetivo de copiar os valores para variáveis de *input*. A Figura 28 mostra um exemplo de operação de cópia do valor do atributo *CarCategory* da variável do processo para o atributo *classAuto* da variável de *input* para invocação do serviço AutoRenting. Esta cópia é necessária porque o tipo de dados esperado pelo serviço da AutoRenting é diferente do tipo de dados manipulado pelo processo. Neste caso, o processo considera *CarCategory*, enquanto que o serviço espera *classAuto* como entrada.

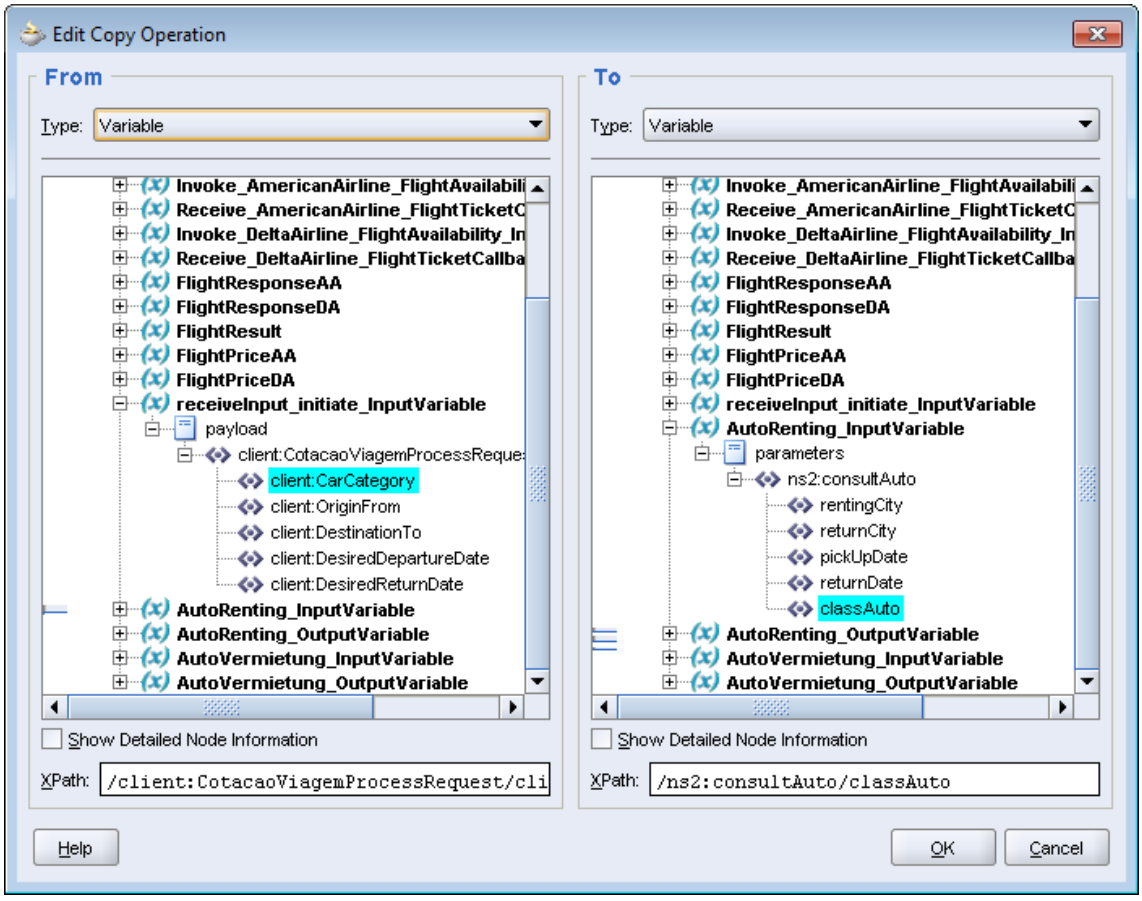

Figura 28 - Operação de cópia de valores entre variáveis

Operações de cópia para preencher todos os valores da variável "AutoRenting_InputVariable" e "AutoVermietung_InputVariable" devem ser realizadas[6]. As variáveis de output de cada um dos *Invokes* são preenchidas automaticamente pelo processo quando este recebe o retorno dos serviços invocados. A Figura 29 mostra o esquema final da atividade *Flow* comportando as duas atividades *Assign* e as duas atividades *Invoke* que acessa os dois *partnerLink* criados.

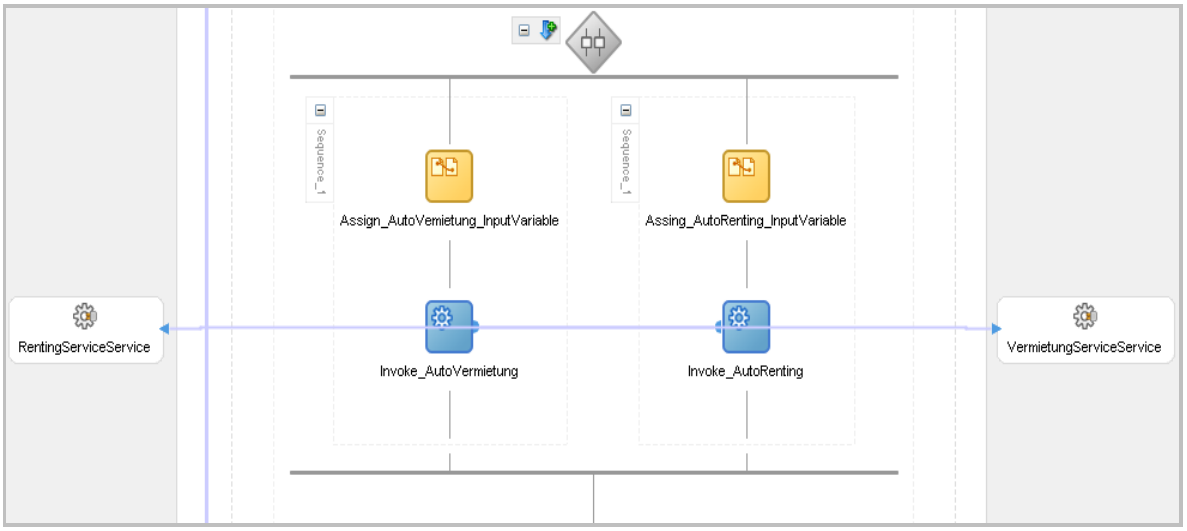

Figura 29 - Atividade *Flow* criada para invocar serviços das locadoras de automóvel

[6] A ordem das variáveis em "AutoRenting_InputVariable" e "AutoVermietung_InputVariable" é idêntica para facilitar a criação de variáveis e atribuição de valor em ambos os idiomas.

De posse das melhores ofertas das duas lojas de aluguel de carros, uma nova atividade *Switch* deve ser criada para comparar ambas e decidir qual deverá ser retornada ao usuário que invocou o processo. A atividade *Switch* deverá possuir uma alternativa além da alternativa *default*, descrita no jDeveloper como *Otherwise*. Uma alternativa da atividade *Switch* deve comportar uma atividade *Assign* e a descrição da condição que fará com que o processo siga pela alternativa escolhida. Para editar a condição clique no ícone , presente na parte superior da alternativa. Ao clicar no ícone uma nova janela será exibida. Para editar a condição clique novamente no ícone . A janela apresentada pelo sistema é exibida pela Figura 30.

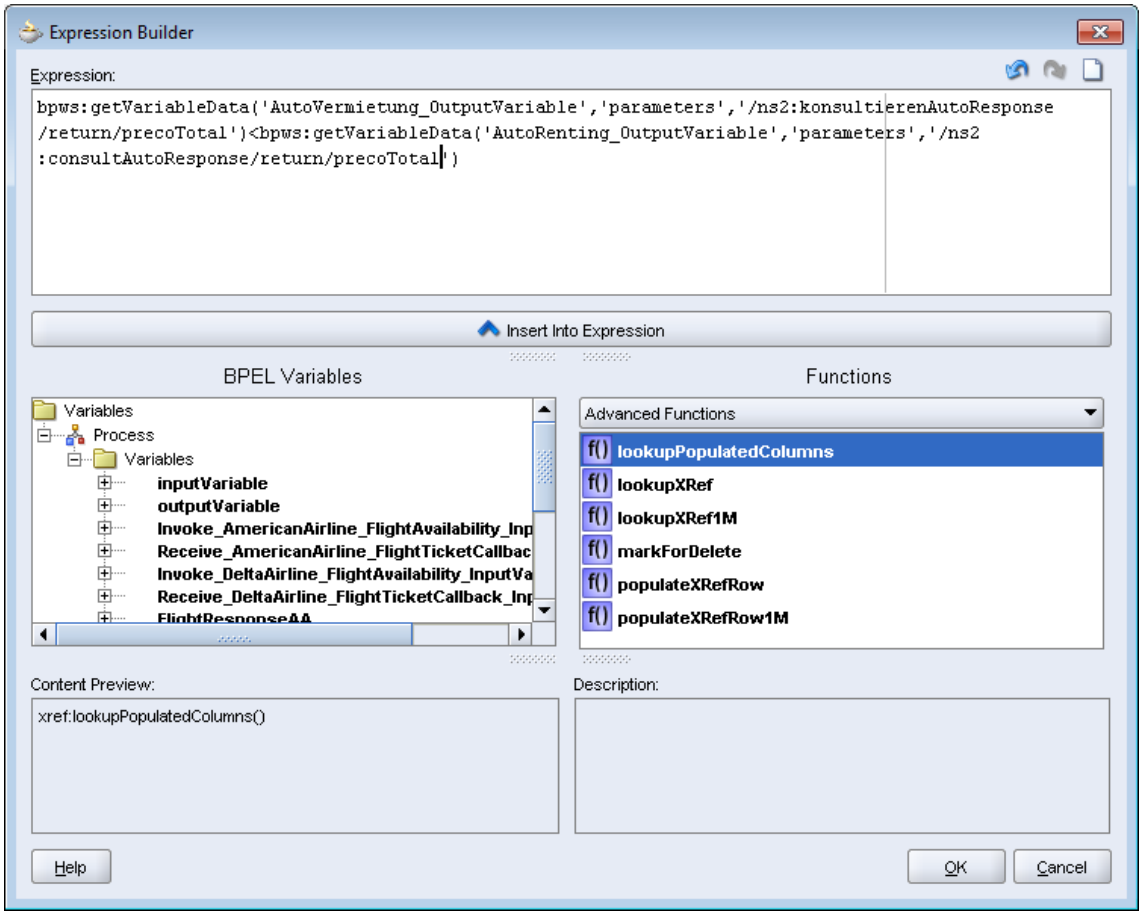

Figura 30 - Edição da condição para a entrada em uma alternativa de uma atividade *Switch*

A condição a ser testada verifica se o valor do aluguel retornado pela empresa X é inferior ao valor retornado pela empresa Y, caso X<Y seguir por uma alternativa, caso contrário o processo deve seguir pelo *Otherwise*. Para inserir atributos no teste lógico, navegue pelas variáveis do BPEL e selecione o atributo a ser utilizado (Figura 30). Com o atributo selecionado, clique no botão "*Insert Into Expression*". O atributo automaticamente é inserido no teste de condição da expressão XPath referente ao acesso à variável desejada.

Cada um dos caminhos possíveis deverá possuir uma atividade *Assign*. Desta forma o caminho a ser seguido definirá os valores que serão retornados pelo processo BPEL ao *partnerLink* que realizou a invocação do processo. A Figura 31 mostra o esquema que simboliza no jDeveloper a atividade *Switch*.

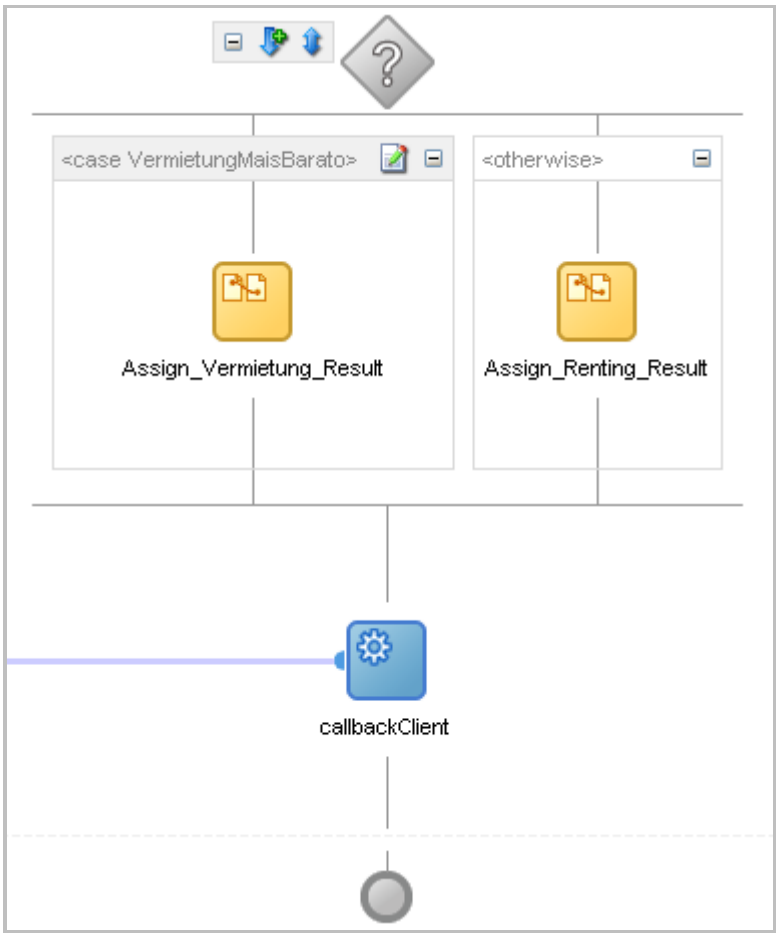

Figura 31 - Atividade *Switch* e o fim do processo BPEL

As atividades "Assign_Vermietung_Result" e "Assign_Rentig_Result" atribui à variável "outputVariable" os valores retornados do preço do aluguel, o modelo e a marca do veículo de seus respectivos serviços disponibilizados no GlassFish. A Figura 40 mostra um exemplo de operação de cópia da variável que corresponde ao preço do aluguel do serviço AutoRenting.

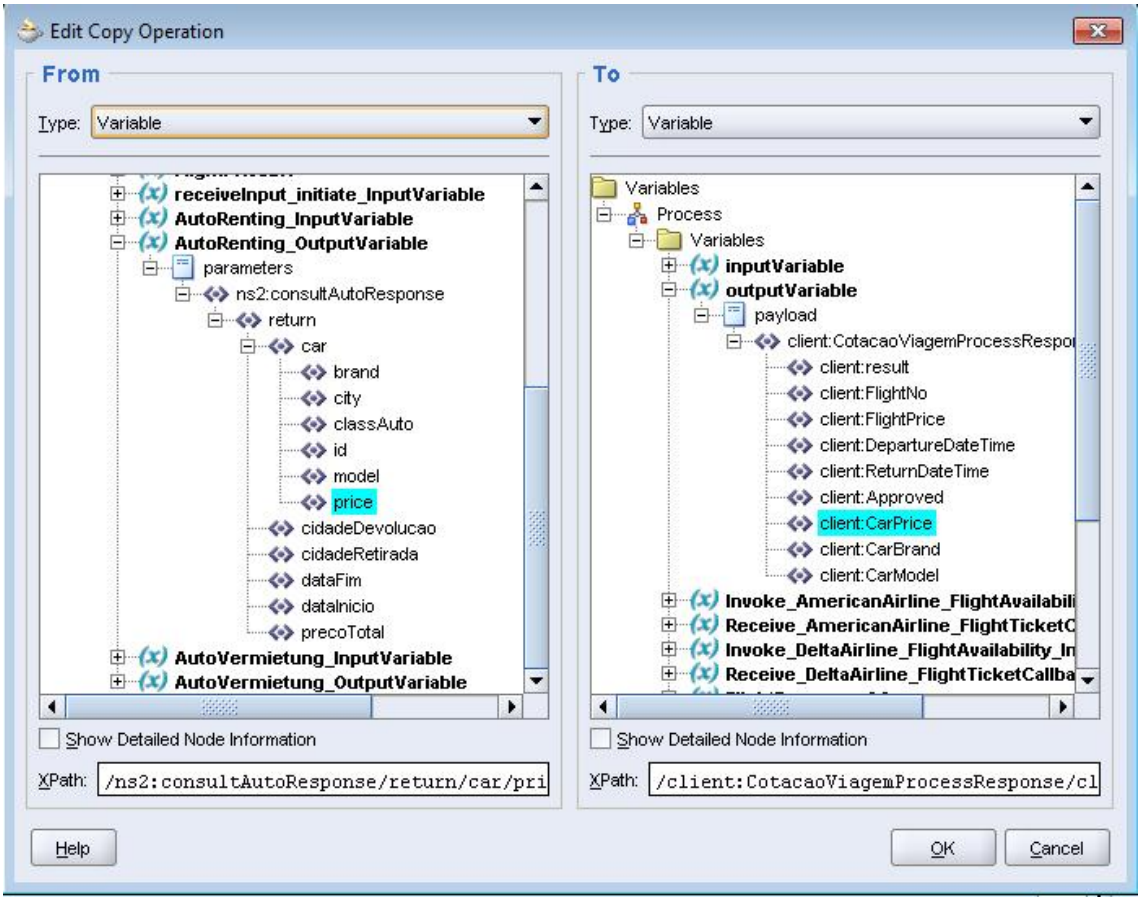

Figura 41 – Operação de cópia de "AutoRenting_OutputVariable" para "outputVariable".

O fim da atividade *Switch* está diretamente ligado à atividade de *Invoke* responsável por executar o *callback* ao *partnerLink* Client, repassando o resultado do processo com a melhor condição de compra de passagem aérea e a melhor condição de locação de veículo .

3.4 Testes do processo

Neste ponto, o tutorial se encontra concluído. Para verificar o resultado final realize o *Deploy* do processo e se dirija ao console do servidor BPEL. Selecione o processo "CotaçãoViagem" e passe variáveis como mostrado na Figura 32.

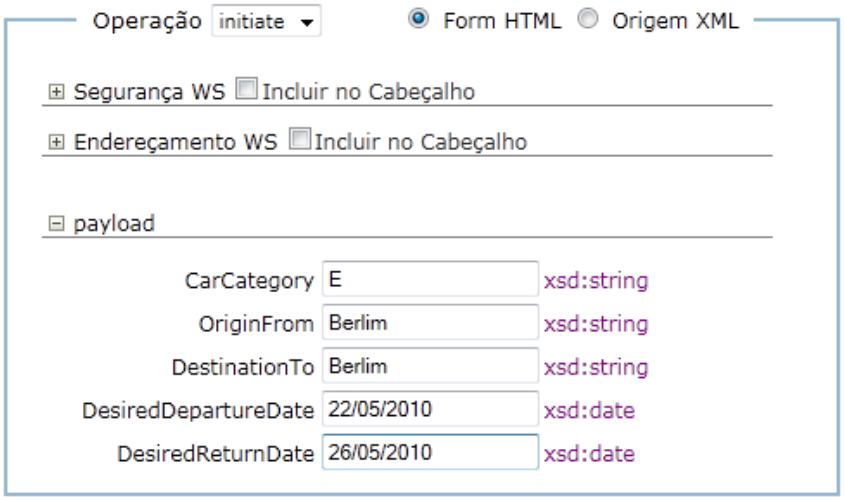

Figura 32 - Exemplo de execução do processo Cotação Viagem

Considerando que o banco de dados está ativo e os serviços AutoRenting e AutoVermietung implantados corretamente no servidor GlassFish, o servidor deverá retornar dados como os exibidos pela Figura 33.

Figura 33 - Retorno do processo Cotação Viagem

4 Trabalhos Relacionados

Nesta seção são apresentados trabalhos relacionados que apresentam problemas existentes na linguagem BPEL e propostas de melhores abordagens. Cada trabalho é resumido individualmente com a intenção de auxiliar na contextualização do leitor.

4.1 BPEL Light

BPEL Light é uma proposta feita por pesquisadores da Universidade de Stuttgart [Nitzsche *et. al.*, 2007] com o objetivo de solucionar alguns dos problemas existentes no padrão WS-BPEL. O artigo inicia contextualizando o leitor sobre Business Process Management (BPM) e Workflow, observando seus grandes sucessos na indústria e na área acadêmica. Tais tecnologias permitem a separação da lógica do processo de negócio da implementação das funções do negócio, possibilitando ao usuário trabalhar em um nível mais alto.

O Workflow comporta três dimensões: Lógica do Processo (o que é feito?), Organização (quem faz?) e Infraestrutura (quais ferramentas são usadas?). Atualmente, o BPEL não dá suporte à segunda dimensão (quem faz?), além de suportar a terceira de forma dependente de tecnologia, pois baseia-se completamente em WSDL. O BPEL Light é uma extensão do WS-BPEL 2.0 que pretende descrever a interação entre dois parceiros sem a dependência do WSDL (uma abordagem "WSDL-less"), desacoplando primeira e terceira dimensões. Abordagens WSDL-less visam atacar duas grandes limitações do BPEL: o reuso limitado e a falta de flexibilidade em termos de interface.

O reuso problemático transparece quando imaginamos um fluxo genérico, como por exemplo, um *Partner Service* que recebe uma resposta e dependendo desta resposta decide entre diferentes ramos do fluxo de negócio. A situação é muito comum. Entre-

tanto como este fragmento se encontra atrelado à WSDLs específicos, seu reuso se faz impossível, a não ser por cenários que utilizem exatamente as mesmas operações dos mesmos parceiros.

A falta de flexibilidade é percebida quando consideramos múltiplos serviços que oferecem funcionalidades similares, como por exemplo, dois diferentes serviços que ofereçam a mesma funcionalidade, mas implementem diferentes WSDLs. Neste não é possível realizar uma troca entre os serviços sem impactar na estrutura do processo. A solução indicada pelos autores para uma abordagem WSDL-less é a *Extensions to Executable BPEL* que considera o uso de qualquer IDL (*Interface Description Language*), auxiliando na geração de processos e fragmentos de processos reutilizáveis.

O BPEL ^{Light} emula o modelo de interação do BPEL tradicional, podendo expor suas funcionalidades como um web service e invocar web services convencionais. A proposta aumenta a flexibilidade, pois estão presentes a configuração, durante a implantação, e a descoberta e vínculo a interfaces apropriadas em tempo de execução.

4.2 Uma Linguagem BPEL específica de domínio independente de plataforma e legível por humanos

A vantagem em se utilizar a tecnologia de web services é o vasto conjunto de padrões aos quais se tem acesso, como segurança e manutenção de transações. Usualmente, dentro de empresas, um único produto SOA se faz suficiente, entretanto quando considerado um órgão governamental ou uma empresa de grande porte, podem ser considerados múltiplos fornecedores de soluções. De forma geral a maioria dos fornecedores de soluções relacionadas a web services e BPEL oferece soluções minimamente interoperáveis.

No mundo soa, o XML é utilizado como tecnologia que permite a criação de interfaces, descrições de processos e formatação de mensagens, tendo seu foco na interoperabilidade. O problema se encontra em lidar e transformar tais documentos XML, tarefa que possui certa complexidade. Este é o motivo pelo qual a maior parte dos desenvolvedores possui ferramentas gráficas para auxiliá-los.

Os problemas com ferramentas gráficas são sua falta de eficiência, seu uso não confiável quando se pretende gerar cenários repetidos, além de não serem suficientemente controláveis nem automatizáveis.

Os padrões por outro lado possuem muitos artefatos redundantes, dificultando a leitura e manutenção, tais artefatos são usualmente mapeados para objetos gráficos diretamente, e não escondidos do usuário. Ferramentas de desenvolvimento com freqüência não possibilitam a extensão em descrições de processos (como pré e pós-condições, autorizações, etc.) e, principalmente, apesar de BPEL ser um padrão, ferramentas diferentes podem implementar diferentes semânticas para um mesmo processo.

Objetivando resolver tais problemas os autores (Simon *et. al.*, 2010) propuseram um modelo abstrato que possa manipular conceitos do BPEL e ainda apresentar uma nova linguagem extensível chamada Service-Oriented Architecture Language (SOAL), que pode ser utilizada para descrever processos BPEL, sendo mais fácil de ser lida e manipulada por humanos. A linguagem possui sintaxe similar a C# e Java.

Podemos tomar como exemplo das facilidades fornecidas pela linguagem o mapeamento que de fato é realizado quando o desenvolvedor utiliza SOAL para os respecti-

vos construtos em XSD, WSDL e BPEL demonstrados resumidamente pela Figura 34, Figura 35 e Figura 36 respectivamente.

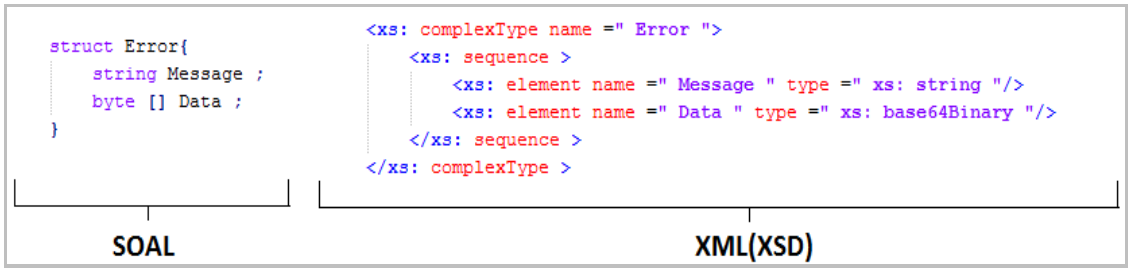

Figura 34 - Mapeamento SOAL x XSD

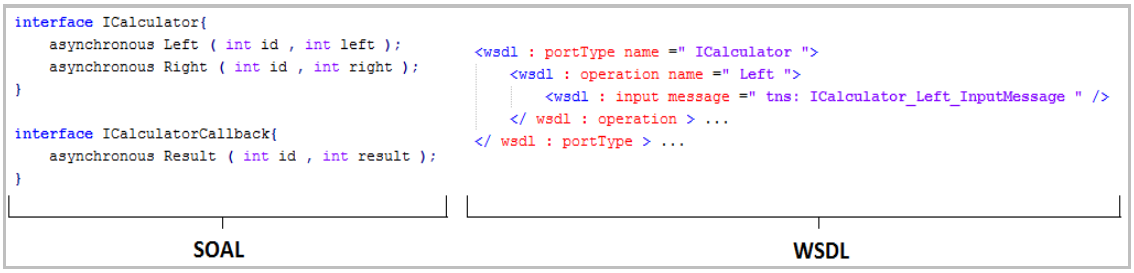

Figura 35 - Mapeamento SOAL x WSDL

```
bpel CalculatorProcess : ICalculator{
    ICalculatorCallback callback ;
    int id , left , right , result , requestID ;
    parallel {
        instantiate receive ICalculator . Left (out id , out left )
        join correlation requestID == id;
        instantiate receive ICalculator . Right (out id , out right )
        join correlation requestID == id;
    }
    assign{
        result = left + right ;
    }
    invoke callback . Result ( requestID , result );
}
```

SOAL

```
WSDL :
    <plnk : partnerLinkType name =" ICalculatorCallbackPLT ">
        <plnk : role name =" ICalculatorCallbackRole " portType =" tns: ICalculatorCallback "/>
    </ plnk : partnerLinkType > ...
    <vprop : property name =" requestID " type =" xsd: integer " />
    <vprop : propertyAlias propertyName =" tns: requestID "
                           messageType =" tns: ICalculator_Left_InputMessage "
                           part =" id" /> ...
BPEL :
    <process name =" CalculatorProcess " ... >
        <partnerLinks >
            <partnerLink name =" ICalculatorPL " partnerLinkType =" tns: ICalculatorPLT "
                myRole =" ICalculatorRole " />
            <partnerLink name =" callback " partnerLinkType =" tns: ICalculatorCallbackPLT "
                partnerRole =" ICalculatorCallbackPLT " />
        </ partnerLinks >
        <variables > ... </ variables >
        <correlationSets >
            < correlationSet name =" requestID " properties =" tns: requestID " />
        </ correlationSets >
        <sequence >
            <flow >
                <receive name =" acceptLeft " partnerLink =" ICalculatorPL "
                         portType =" tns: ICalculator " ...
```

BPEL

Figura 36 - Mapeamento SOAL x BPEL

Definir um processo BPEL em linguagem mais genérica possibilita, como citado anteriormente, gerar processos que podem ser reutilizados em múltiplas plataformas. O que é necessário é definir o transformador do processo em SOAL para BPEL correspondente à plataforma. O diagrama da Figura 37 mostra o esquema de conversão de um processo BPEL definido em SOAL para múltiplos motores de execução BPEL.

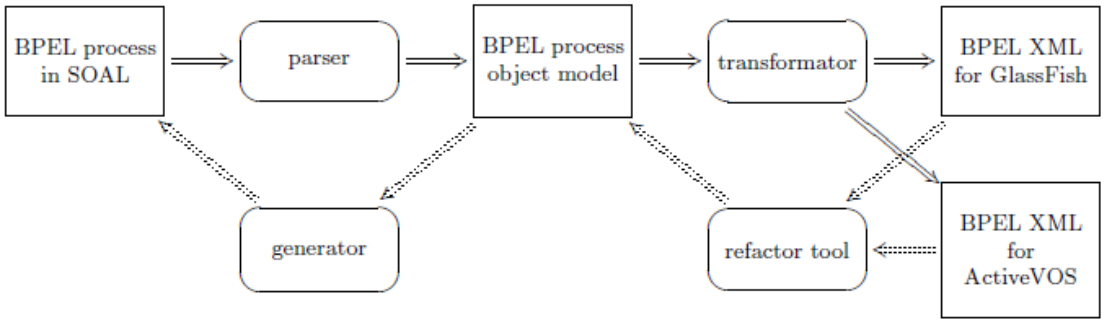

Figura 37 - A arquitetura do framework SOAL para BPEL

4.3 Life after BPEL?

O artigo “Life after BPEL?” tem o objetivo de analisar criticamente esta tecnologia, a análise passa pela linguagem em si e pelo foco do BPEL. Os autores apontam a complexidade da linguagem e a dificuldade em utilizá-la como um problema crítico pois BPEL só pode ser lido por um “olho treinado” e sua complexidade se da em função da grande gama de construtos e de formas diferentes de utilizá-los.

O artigo apresenta que BPEL suporta a modelagem de dois tipos de processos: executáveis e abstratos. Um processo abstrato é um protocolo de negócios, basicamente uma especificação do comportamento de trocas de mensagens entre vários parceiros sem revelar o comportamento interno deles, um processo abstrato vê o mundo exterior da perspectiva de uma única organização. Um processo executável vê o mundo de uma maneira similar, mas nele as especificações contêm mais detalhes tal que o processo se torna executável. Estas especificações podem ir da ordem de execução das atividades que constituem o processo, os parceiros envolvidos no processo, as mensagens trocadas entre eles até o tratamento de exceção requerido em caso de erros e exceções.

As tabelas da Figura 38 e da Figura 39 resumem os resultados da análise baseada em padrões efetuada pelos autores em relação a padrões de “control-flow” e “data-flow” de workflows. Para cada fluxo de controle e padrão de dados foi checado se é possível realizar o padrão com BPEL. Se BPEL suporta o padrão diretamente através de um de seus construtos ele é avaliado com um +. Se o padrão não é suportado diretamente ele recebe um +/-. Qualquer solução que resulte em um “construto espaguete” ou não seja possível de forma alguma recebe um -.

pattern	pattern name	BPEL
WCFP1	sequence	+
WCFP2	parallel split	+
WCFP3	synchronization	+
WCFP4	exclusive choice	+
WCFP5	simple merge	+
WCFP6	multi choice	+
WCFP7	synchronizing merge	+
WCFP8	multi merge	–
WCFP9	discriminator	–
WCFP10	arbitrary cycles	–
WCFP11	implicit termination	+
WCFP12	multiple instances no synchronization	+
WCFP13	multiple instances design time knowledge	+
WCFP14	multiple instances runtime knowledge	–
WCFP15	multiple instances without a priori knowledge	–
WCFP16	deferred choice	+
WCFP17	interleave parallel routing	+/–
WCFP18	milestone	–
WCFP19	cancel activity	+
WCFP20	cancel case	+

Figura 38 - Análise do BPEL baseada em padrões "control-flow" de Workflow

Uma situação hipotética poderia ser: Quando um contrato é finalizado ele deve ser revisado pelo diretor ou pelo gerente de operações, quem estiver disponível primeiro. Ambos serão notificados quando um contrato estiver esperando revisão. Podemos observar (na tabela) que WCFP 16 e WCFP4 são diferentes e ambos são suportados pelo BPEL, a escolha que queremos fazer aqui não é baseada em uma decisão ou em dados, é uma escolha a ser feita pelo ambiente. BPEL claramente suporta este padrão. A esco-

lha (por condições de "corrida" baseadas no tempo ou *triggers* externas) oferece a funcionalidade desejada diretamente.

pattern	pattern name	BPEL
WDP1	task data	+/–
WDP2	block data	–
WDP3	scope data	+
WDP4	folder data	–
WDP5	multiple instance data	–
WDP6	case data	+
WDP7	workflow data	–
WDP8	environment data	+
WDP9	data interaction between tasks	+
WDP10	data interaction – block task to decomposition	–
WDP11	data interaction – decomposition to block task	–
WDP12	data interaction – to multiple instance task	–
WDP13	data interaction – from multiple instance task	–
WDP14	data interaction – case to case	+/–
WDP15	data interaction – task to environment – push-oriented	+
WDP16	data interaction – environment to task – pull-oriented	+
WDP17	data interaction – environment to task – push-oriented	+/–
WDP18	data interaction – task to environment – pull-oriented	+/–
WDP19	data interaction – case to environment – push-oriented	–
WDP20	data interaction – environment to case – pull-oriented	–
WDP21	data interaction – environment to case – push-oriented	–
WDP22	data interaction – case to environment – pull-oriented	–
WDP23	data interaction – workflow to environment – push-oriented	–
WDP24	data interaction – environment to workflow – pull-oriented	–
WDP25	data interaction – environment to workflow – push-oriented	–
WDP26	data interaction – workflow to environment – pull-oriented	–
WDP27	data passing by value – incoming	+
WDP28	data passing by value – outgoing	+
WDP29	data passing – copy in/copy out	–
WDP30	data passing by reference – unlocked	+
WDP31	data passing by reference – locked	+/–
WDP32	data transformation – input	–
WDP33	data transformation – output	–
WDP34	task precondition – data existence	+/–
WDP35	task precondition – data value	+
WDP36	task postcondition – data existence	–
WDP37	task postcondition – data value	–
WDP38	event-based task trigger	+
WDP39	data-based task trigger	+/–
WDP40	data-based routing	+

Figura 39 - Análise do BPEL com base no padrão de dados do Workflow

Neste trabalho, não apresentamos as explicações em detalhes de todas as avaliações que foram feitas para chegar a estas duas tabelas. No entanto, podemos concluir que BPEL é mais poderoso do que a maioria das linguagens de processos tradicionais. De forma geral podemos dizer que BPEL é uma linguagem expressiva com algumas limitações. No entanto BPEL também é uma linguagem muito complicada com muitos conceitos.

Como já foi apresentado, BPEL pode ser utilizado para modelar tanto o comportamento de troca de mensagens tanto de processos abstratos como executáveis. No entanto, a maioria das atenções esteve voltada par o BPEL como uma linguagem para execução. Na visão dos autores BPEL é falho como linguagem para modelagem de processos abstratos.

A Figura 40 mostra os dois usos possíveis de BPEL. A figura ilustra claramente que em ambos os casos a tarefa é vista da perspectiva de um dos parceiros.

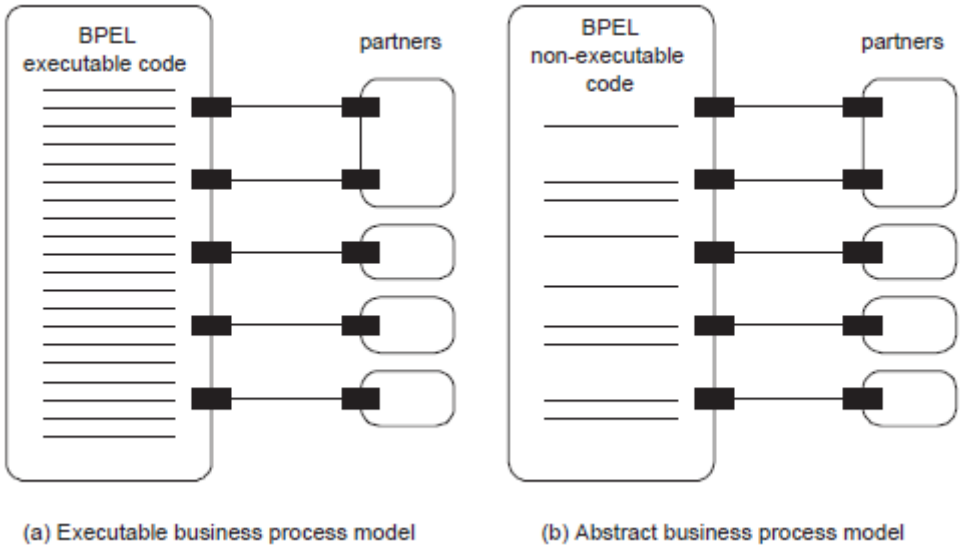

Figura 40 - As duas formas nas quais BPEL pode ser utilizado

Os autores afirmam que o padrão BPEL incorpora um número de funcionalidades especializadas para o desenvolvimento de web services, incluindo suporte direto a XML entre outras e dizem que BPEL pode facilitar a migração de um sistema para outro, pois ele abstrai a implementação interna de cada um dos parceiros participantes da composição. Os autores também afirmam que as funcionalidades providas tornam BPEL muito atraente em relação às alternativas de linguagens convencionais (orientadas a objetos) quando se trata de desenvolvimento de web services.

5 Conclusão

A utilização de BPEL possibilita a integração e execução de processos de negócios organizacionais e inter-organizacionais através de troca de informações (mensagens). Um diferencial nessa abordagem é que tanto a composição de processos quanto sua exposição é feita utilizando web services que, segundo Erl (2005), é a principal tecnologia para implementação de serviços e é, atualmente, o padrão de fato.

Neste relatório, foram apresentados definições de SOA, serviços e composição de serviços. Em seguida, foram detalhados os principais elementos utilizados na linguagem padrão WS-BPEL. Essa abordagem tornou viável a proposta principal deste relatório: implementar processos de negócio utilizando BPEL e, consecutivamente, minimizar as dificuldades relativas à aprendizagem da tecnologia. A abordagem utilizada para alcançar esses objetivos foi o desenvolvimento de um processo BPEL de Cotação de Viagem, apresentado no Seção 3.

Foi demonstrado em termos teóricos e práticos que a utilização de WS-BPEL em organizações facilita a integração de aplicações, processos de negócio e a integração entre parceiros de negócio. Entretanto, uma análise crítica do padrão foi feita com objetivo de separar a realidade – contendo os prós e contras do padrão – da propaganda divulgada pelos entusiastas e principais fornecedores de solução baseada em WS-BPEL.

A Seção 4 foi responsável por trazer à tona novos conceitos e discussões que corroboram ou levantam questões e limitações do padrão WS-BPEL. A utilização da exten-

são BPEL Light, apesar de minimizar os problemas encontrados na especificação e aumentar o poder de reuso de processos BPEL, deve ser avaliada cuidadosamente por se tratar de uma extensão. Extensões não fazem parte do padrão definido de WS-BPEL e, por isso, podem comprometer a portabilidade de processos BPEL. Também nesta seção, foi exposta a complexidade de aprendizagem da tecnologia – em grande parte devido ao uso obrigatório de XML para criação de processos BPEL interoperáveis – e apresentado alternativas como a linguagem extensível Service-oriented Architecture Language (SOAL). A proposta de utilização de linguagens de programação mais comumente utilizadas para criação de processos BPEL e posterior transformação do processo para linguagem BPEL é uma abordagem que tem como benefício utilizar as habilidades técnicas já adquiridas dos desenvolvedores.

A orquestração de serviços foi a técnica de composição utilizada nesse relatório devido a seu nível de amadurecimento e utilização. Entretanto, o estudo de coreografia e desenvolvimento de ferramentas que suportam esse tipo de composição está aumentando significativamente devido à maior escalabilidade que essa técnica oferece [Josuttis, 2007]. Assim, sugerimos como pesquisa futura, o estudo de coreografia de serviços e suas ferramentas com o intuito de relacionar o grau de amadurecimento dessa técnica em relação à orquestração, além de identificar vantagens e desvantagens na utilização dessas técnicas de composição de serviços.

6 Referências Bibliográficas

ALVEZ, A., ARKIN, A., ASKARI, S., BARRETO, C. *et al.* Web Services Business Process Execution Language Version 2.0. OASIS, 2007. Disponível em <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Acessado em Julho de 2011.

AZEVEDO, L. G., SOUZA, J., BAIAO, F., SANTORO, F.. "Inspeção da Ferramenta Oracle BPEL PM, 2009. Disponível em http://www.seer.unirio.br/index.php/monografiasppgi/article/view/294/294. Acessado em Fevereiro de 2011.

CHRISTENSEN, E., CURBERA, E., MEREDITH, G., WEERAWARANA, S. Web Services Description Language (WSDL) 1.1, 2001. Disponível em <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, Acessado em 04/07/2011

BARRETO, C. , Next Generation Web Services In Practice. 2007. Disponível em <http://charltonb.typepad.com/talks/050807-cbb-ws-www2007/>. Acessado em 11/07/2011.

FARIA, F., AZEVEDO, L. Oracle BPEL PM e JDeveloper. Programa de Pós graduação em Informática, 2010. Disponível em: < www.uniriotec.br/~azevedo/BPEL/OracleBPELPM_JDeveloper.zip>, Acessado em: Julho de 2011.

JURIC, M. B. A Hands-on Introduction to BPEL. Oracle Corporation, 2011. Disponível em <http://www.oracle.com/technetwork/articles/matjaz-bpel1-090575.html>. Acessado em Julho de 2011.

MIGUEL, A. WS-BPEL 2.0 Tutorial, 2005 Disponível em <http://www.eclipse.org/tptp/platform/documents/design/choreography_html/tutorials/wsbpel_tut.html>. Acessado em Julho de 2011.

NITZSCHE, J., VAN LESSEN, T., KARASTOYANOVA, D., & LEYMANN, F. (2007). BPEL light. Proceedings of the 5th International Conference on Business Process Management BPM, 2007 (Vol. 4714, pp. 214-229).

ORACLE (2006). Oracle BPEL Process Manager 10.1.2.0.x Quick Start Tutorial, Oracle Corporation, 2006. Disponível em <http://download.oracle.com/otndocs/products/bpel/quickstart.pdf>. Acessado em Julho de 2011.

SIMON, B., GOLDSCHMIDT, B. e KONDOROSI, K. A Human Readable Platform Independent Domain Specifc Language for BPEL. In Proceedings of Networked Digital Tecnologies, 2010.