面向 Web 服务的业务流程执行语言(BPEL 或 BPEL4WS)是一种使用 Web 服务定义和执行业务流程的语言。BPEL 使您可以通过组合、编排和协调 Web 服务自上而下地实现面向服务的体系结构 (SOA)。BPEL 提供了一种相对简单易懂的方法，可将多个 Web 服务组合到一个新的复合服务(称作业务流程)中。

本文将介绍如何创建一个将一系列虚拟的、与旅行相关的 web 服务结合起来的示例业务流程，然后将其部署到 Oracle BPEL Process Manager 运行时环境。

BPEL 背景知识

首先，介绍一些背景知识。BPEL 基于 XML 和 Web 服务构建;它使用一种基于 Web 的语言，该语言支持 web 服务技术系列，包括 SOAP、WSDL、UDDI、Web 服务可靠性消息、Web 服务寻址、Web 服务协调以及 Web 服务事务。

BPEL 代表了两种早期工作流语言 - Web 服务流语言 (WSFL) 和 XLANG 的交汇。WSFL 由 IBM 基于有向图概念设计。XLANG 是一种由 Microsoft 设计的块结构化语言。BPEL 组合了这两种方法，并提供了丰富的词汇来描述业务流程。

BPEL 的第一个版本诞生于 2002 年 8 月。此后，随着许多主要供应商(包括 Oracle)的纷纷加入了，催生了多项修改和改进，并于 2003 年 3 月推出了 1.1 版。2003 年 4 月，BPEL 提交结构化信息标准促进组织 (OASIS) 以实现标准化，并组建了 Web 服务业务流程执行语言技术委员会 (WSBPEL TC)。该努力使 BPEL 在业界获得更广范围的认可。

在企业内部，BPEL 用于标准化企业应用程序集成以及将此集成扩展到先前孤立的系统。在企业之间，BPEL 使与业务合作伙伴的集成变得更容易、更高效。BPEL 激发企业进一步定义它们的业务流程，从而导致业务流程的优化、重新设计以及选择最合适的流程，进而实现了组织的进一步优化。BPEL 中描述的业务流程定义并不影响现有系统，因此对升级产生了促进作用。在已经或将要通过 Web 服务公开功能的环境中，BPEL 是一项重要的技术。随着 Web 服务的不断普及，BPEL 的重要性也随之提高。

编制与编排

Web 服务通常公开某些应用程序或信息系统的操作。因此，组合多个 Web 服务实际上涉及基础应用程序及其功能的集成。

可以用两种方式组合 Web 服务:

- 编制
- 编排

在编制(通常用于专用业务流程)中，一个中央流程(可以是另一个 Web 服务)控制相关的 Web 服务并协调对操作所涉及 Web 服务的不同操作的执行。相关的 Web 服务并不“知道”(也无需知道)它们参与了组合流程并在参与更高级别的业务流程。只有编制的中央协调员知道此目标，因此编制主要集中于操作的显式定义以及 Web 服务的调用顺序。(见图 1。)

图 1:通过编制组合 Web 服务

而编排并不依赖某个中央协调员。相反，编排所涉及的每个 Web 服务完全知道执行其操作的时间以及交互对象。编排是一种强调在公共业务流程中交换消息的协作方式。编排的所有参与者都需要知道业务流程、要执行的操作、要交换的消息以及消息交换的时间。(见图 2。)

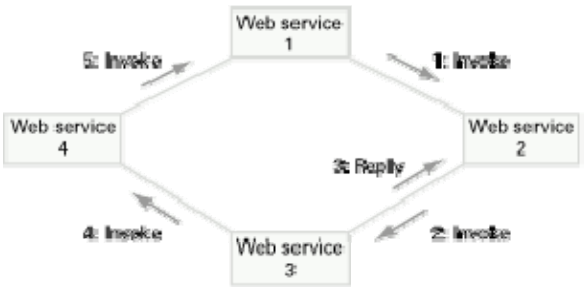

图 2:通过编排组合 Web 服务

从组合 Web 服务以执行业务流程的角度而言，编制是一个更灵活的范例，它相对于编排而言具有以下优点:

- 元件流程的协调由某个已知的协调员集中管理
- 可以组合 Web 服务而不必使它们知道它们正在参与更大的业务流程
- 可以准备其他方案以防发生故障。

BPEL 支持两种不同的业务流程描述方法(支持编制和编排):

- **可执行流程**允许指定业务流程的准确细节。它们遵循编制范例，并可由编制引擎执行。
- **抽象业务协议**允许只指定双方之间的公共消息交换。它们不包含流程的内部细节并且无法执行。它们遵循编排范例。

现在，我们来逐步演示如何创建可执行的 BPEL 业务流程;可以下载它的代码并将其部署到 Oracle BPEL Process Manager。我们将假设已经按照安装指导成功安装了 Oracle BPEL Process Manager，并假设它使用缺省端口 9700。如果在安装过程中选择了其他端口，则必须相应地修改示例。

构建业务流程

BPEL 流程指定参与的 Web 服务的确切调用顺序 - 顺序地或并行地。使用 BPEL，您可以表述条件行为。例如，某个 Web 服务的调用可以取决于上次调用的值。还可以构造循环、声明变量、复制和赋予

值、定义故障处理程序等。通过组合所有这些构造，您可以以算法的形式定义复杂业务流程。实际上，由于业务流程本质上属于活动图，因此使用统一建模语言 (UML) 活动图表示它们可能很有用。

通常情况下，BPEL 业务流程接收请求。为了满足请求，该流程调用相关的 Web 服务，然后响应原始调用方。由于 BPEL 流程与其他 Web 服务通信，因此它在很大程度上依赖于复合型 Web 服务调用的 Web 服务 的 WSDL 描述。

我们来看一个示例。一个 BPEL 流程由多个步骤组成，每个步骤称作“活动”。BPEL 支持基元活动和结构活动。基元活动表示基本构造，用于如下所示的常见任务:

使用 <invoke> 调用其他 Web 服务
使用 <receive>（接收请求）等待客户端通过发送消息调用业务流程
使用 <reply> 生成同步操作的响应
使用 <assign> 操作数据变量
使用 <throw> 指示故障和异常
使用 <wait> 等待一段时间
使用 <terminate> 终止整个流程。

然后，我们可以组合这些基元活动以及其他基元活动，以定义准确指定业务流程步骤的复杂算法。为组合基元活动，BPEL 支持几个结构活动。其中最重要的是:

顺序 (<sequence>)，它允许定义一组将按顺序调用的活动。
流 (<flow>)，用于定义一组将并行调用的活动
Case-switch 构造 (<switch>)，用于实现分支
While (<while>)，用于定义循环
使用 <pick> 能够选择多个替换路径之一。

每个 BPEL 业务还将使用 <partnerLink> 定义合作伙伴链接，使用 <variable> 声明变量。

为了理解 BPEL 是如何描述业务流程的，我们将定义雇员出差安排的简化业务流程:客户端调用此业务流程，指定雇员姓名、目的地、出发日期以及返回日期。此 BPEL 业务流程首先检查雇员出差状态。我们将假设存在一个可用于进行此类检查的 Web 服务。然后，此 BPEL 流程将检查以下两家航空公司的机票价格:美国航空公司和达美航空公司。我们将再次假设这两家航空公司均提供了可用于进行此类检查的 Web 服务。最后，此 BPEL 流程将选择较低的价格并将出差计划返回给客户端。

然后，我们将构建一个异步 BPEL 流程。我们将假设用于检查雇员出差状态的 Web 服务是同步的。由于可以立即获取此数据并将其返回给调用方，因此这是一个合理的方法。为了获取机票价格，我们使用异步调用。由于确认飞机航班时刻表可能需要稍长的时间，因此这也是一个合理的方法。为简化示例，我们假设以上两家航空公司均提供了 Web 服务，且这两个 Web 服务完全相同(即提供相同的端口类型和操作)。

在实际情形下，您通常无法选择 Web 服务，而是必须使用您的合作伙伴提供的服务。如果您有幸能够同时设计 Web 服务和 BPEL 流程，则应考虑用哪个接口更好。通常，您将对持续时间较长的操作使用异步服务，而对在相对较短的时间内返回结果的操作使用同步服务。如果使用异步 Web 服务，则 BPEL 流程通常也是异步的。

当您用 BPEL 定义业务流程时，您实际上定义了一个由现有服务组成的新 Web 服务。该新 BPEL 复合 Web 服务的接口使用一组端口类型来提供类似任何其他 Web 服务的操作。要调用用 BPEL 描述的业务流程，则必须调用生成的复合 Web 服务。图 3 是我们流程的示意图。

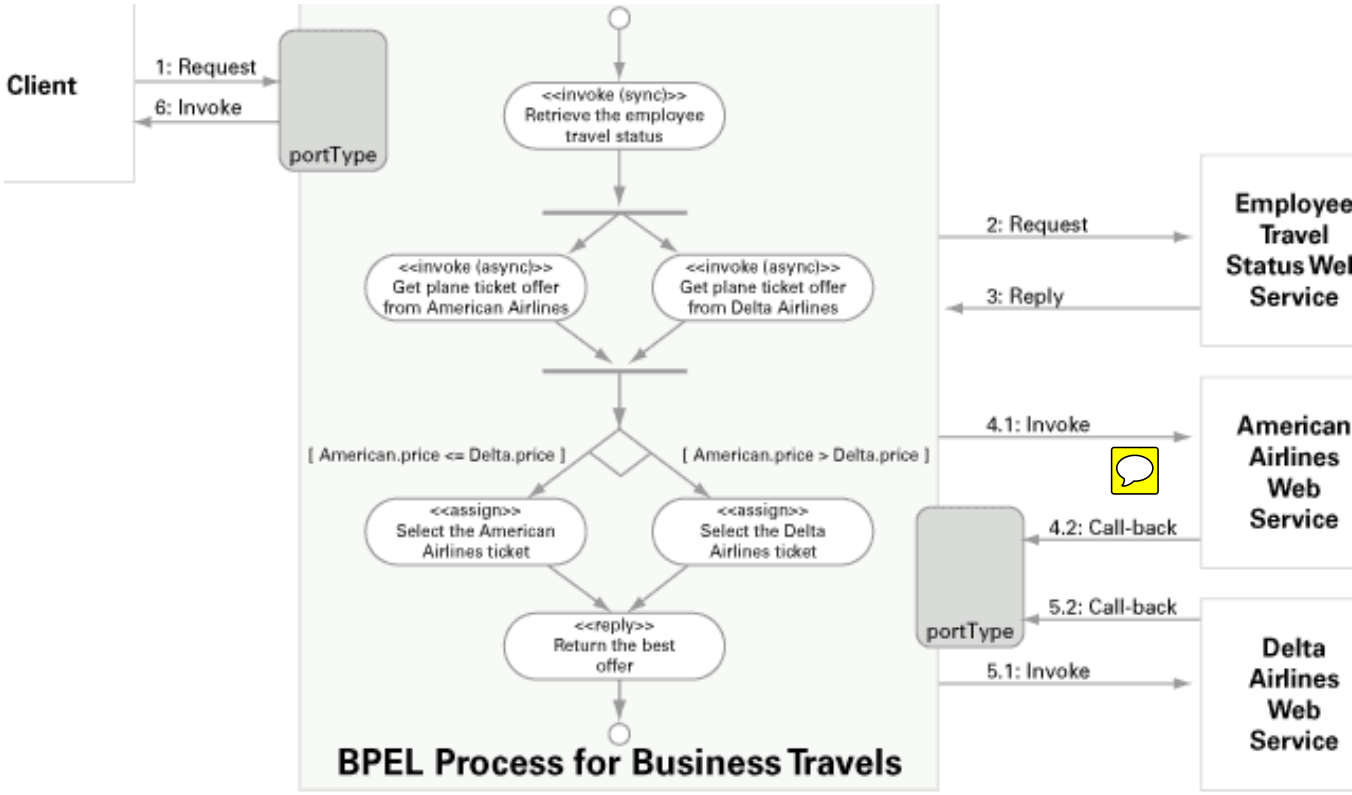

图 3:出差安排示例 BPEL 流程

在开发此示例 BPEL 流程的过程中，您将经历下列步骤:

- 熟悉相关的 Web 服务
- 为此 BPEL 流程定义 WSDL
- 定义合作伙伴链接类型
- 开发此 BPEL 流程:

定义合作伙伴链接

声明变量

编写流程逻辑定义。

第 1 步:列出相关 Web 服务的清单

在您开始编写 BPEL 流程定义之前，必须先熟悉从业务流程中调用的 Web 服务。这些服务称作合作伙伴 Web 服务。本示例使用雇员出差状态 Web 服务以及美国航空公司和达美航空公司 Web 服务(这两个 Web 服务具有相同的 WSDL 描述)。(同样，本示例中使用的 Web 服务是虚构的。)

雇员出差状态 Web 服务雇员出差状态 Web 服务提供 EmployeeTravelStatusPT 端口类型，通过它可以使用 EmployeeTravelStatus 操作检查雇员出差状态。此操作将返回雇员可以使用的乘机标准(可能为经济舱、商务舱或头等舱)。(见图 4。)

图 4:雇员出差状态 Web 服务

航空公司 Web 服务航空公司 Web 服务是异步的;因此它指定了两个端口类型:第一个端口类型 FlightAvailabilityPT 用于使用 FlightAvailability 操作检查航班可用性。为返回结果，该 Web 服务指定了第二个端口类型 FlightCallbackPT。此端口类型指定 FlightTicketCallback 操作。

尽管航空公司 Web 服务定义了两个端口类型，但它只实现 FlightAvailabilityPT。FlightCallbackPT 则由作为 Web 服务客户端的 BPEL 流程实现。图 5 是此 Web 服务体系结构的示意图:

图 5:航空公司 Web 服务

第 2 步:为 BPEL 流程定义 WSDL

接下来，我们必须将此业务出差 BPEL 公开为 Web 服务。因此，第二步是为它定义 WSDL。此流程将必须从它的客户端接收消息并返回结果。它必须公开 TravelApprovalPT 端口类型，后者将指定一个输入消息。它还必须声明 ClientCallbackPT 端口类型(用于使用回调将结果异步返回给客户端)。图 6 说明了此流程。

图 6:此 BPEL 流程的 WSDL

第 3 步:定义合作伙伴链接类型

第三步是定义合作伙伴链接类型。合作伙伴链接类型表示 BPEL 流程与相关方(包括 BPEL 流程调用的 Web 服务以及调用 BPEL 流程的客户端)之间的交互。

本示例包含三个不同的合作伙伴:客户端、雇员出差状态服务和航空公司服务。理想情况下，每个 Web 服务都应在 WSDL 中定义相应的合作伙伴链接类型。(实际情形可能不是这样的。)然后，我们可以使用 WSDL 包装合作伙伴 Web 服务(导入 Web 服务的 WSDL 并定义合作伙伴链接类型)。或者，我们可以在 BPEL 流程的 WSDL 中定义所有合作伙伴链接。但由于此方法违反了封装原则，因此不建议使用。

对于本示例，我们定义了三个合作伙伴链接类型(每个类型位于 Web 服务的相应 WSDL 中):

- travelLT:用于描述此 BPEL 流程客户端与此 BPEL 流程本身之间的交互。此交互是异步交互。此合作伙伴链接类型在此 BPEL 流程的 WSDL 中定义。
- employeeLT:用于描述此 BPEL 流程与雇员出差状态 Web 服务之间的交互。此交互是同步交互。此合作伙伴链接类型在雇员 Web 服务的 WSDL 中定义。
- flightLT:描述此 BPEL 流程与航空公司 Web 服务之间的交互。此交互是异步交互，且航空公司 Web 服务对此 BPEL 流程调用一个回调。此合作伙伴链接类型在航空公司 Web 服务的 WSDL 中定义。

每个合作伙伴链接可以拥有一个或两个角色，我们必须为每个角色指定它使用的 portType。对于同步操作，由于操作只是单向调用，因此每个合作伙伴链接类型仅有一个角色。例如，此 BPEL 流程对雇员出差状态 Web 服务调用 EmployeeTravelStatus 操作。由于它是同步操作，因此此 BPEL 流程等待完成并仅在完成操作后取得响应。

对于异步回调操作，我们必须指定两个角色。第一个角色描述客户端操作调用。第二个角色描述回调操作调用。在本示例中，BPEL 流程与航空公司 Web 服务之间存在一个异步关系。

正如我们已经指出的，我们需要三个合作伙伴链接类型:两个链接类型指定两个角色(因为它们是异步的)，一个链接类型指定一个角色(因为它是同步的)。

合作伙伴链接类型在特殊命名空间 http://schemas.xmlsoap.org/ws/2003/05/partner-link/ 的 WSDL 定义。首先，我们在客户端使用的 BPEL 流程 WSDL 中定义 travelLT 链接类型以调用此 BPEL 流程。所需的第一个角色是出差服务(即，我们的 BPEL 流程)的角色。客户端使用 TravelApprovalPT 端口类型与此 BPEL 服务通信。第二个角色 travelServiceCustomer 描述了此 BPEL 流程将在 ClientCallbackPT 端口类型中对其执行回调的客户端的特征:

```
<plnk:partnerLinkType name="travelLT">

<plnk:role name="travelService">

<plnk:portType name="tns:TravelApprovalPT" />

</plnk:role>

<plnk:role name="travelServiceCustomer">

<plnk:portType name="tns:ClientCallbackPT" />
```

```
</plnk:role>

</plnk:partnerLinkType>
```

第二个链接类型是 employeeLT。它用于描述此 BPEL 流程与雇员出差状态 Web 服务之间的通信，并在此雇员 Web 服务的 WSDL 中定义。此交互是同步交互，因此我们需要一个名为 employeeTravelStatusService 的角色。此 BPEL 流程使用雇员 Web 服务上的 EmployeeTravelStatusPT:

```
<plnk:partnerLinkType name="employeeLT">

<plnk:role name="employeeTravelStatusService">

<plnk:portType name="tns:EmployeeTravelStatusPT" />

</plnk:role>

</plnk:partnerLinkType>
```

最后一个合作伙伴链接类型 flightLT 用于描述此 BPEL 流程与航空公司 Web 服务之间的通信。此通信是异步通信。此 BPEL 流程对航空公司 Web 服务调用一个异步操作。此 Web 服务在完成请求后对此 BPEL 流程调用一个回调。因此，我们需要两个角色。第一个角色描述航空公司 Web 服务对于此 BPEL 流程服务的角色，即航空公司服务 (airlineService)。此 BPEL 流程使用 FlightAvailabilityPT 端口类型进行异步调用。第二个角色描述了此 BPEL 流程对于航空公司 Web 服务的角色。对于航空公司 Web 服务而言，此 BPEL 流程是一个航空公司客户，因此角色名称为 airlineCustomer。航空公司 Web 服务使用 FlightCallbackPT 端口类型进行回调。此合作伙伴链接类型在航空公司 Web 服务的 WSDL 中定义:

```
<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>
```

了解合作伙伴链接类型对于开发 BPEL 流程规范至关重要。有时，它可以帮助生成所有交互的图表。定义合作伙伴链接类型后，我们已经完成了准备阶段，并准备开始编写业务流程定义。

第 4 步:创建业务流程

现在，您就可以开始编写 BPEL 流程了。通常，BPEL 流程等待客户端传入的消息，以启动业务流程的执行。在本示例中，客户端通过发送输入消息 TravelRequest 启动此 BPEL 流程。然后，此 BPEL 流程通过发送 EmployeeTravelStatusRequest 消息调用雇员出差状态 Web 服务。由于此调用是同步调用，因此它等待 EmployeeTravelStatusResponse 消息。然后，此 BPEL 流程通过向上述两家航空公司 Web 服务发送 FlightTicketRequest 消息对它们进行并发异步调用。每个航空公司 Web 服务通过发送 TravelReponse 消息进行回调。然后，此 BPEL 流程选择较合适的航空公司并使用 TravelResponse 消息对客户端进行回调。

我们首先编写一个空的 BPEL 流程提纲，它展示了每个 BPEL 流程定义文档的基本结构:

```
<process name="BusinessTravelProcess" ... >

    <partnerLinks>
    <!-- The declaration of partner links -->
    </partnerLinks>

    <variables>
    <!-- The declaration of variables -->
    </variables>

    <sequence>  （序列，顺序）
    <!-- The definition of the BPEL business process main body -->
    </sequence>

</process>
```

我们首先添加所需的命名空间。此处，我们必须定义目标命名空间以及用于访问雇员和航空公司 WSDL 以及此 BPEL 流程 WSDL 的命名空间。我们还必须为所有 BPEL 活动标记声明命名空间(此处采用缺省命名空间，以便不必限定每个 BPEL 标记名)。BPEL 活动命名空间必须为 http://schemas.xmlsoap.org/ws/2003/03/business-process/:

```
<process name="BusinessTravelProcess"
```

```
targetNamespace="http://packtpub.com/bpel/travel/"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:trv="http://packtpub.com/bpel/travel/"
xmlns:emp="http://packtpub.com/service/employee/"
xmlns:aln="http://packtpub.com/service/airline/" >
...
```

合作伙伴链接接下来，我们必须定义合作伙伴链接，它们定义与此 BPEL 流程交互的不同方。每个合作伙伴链接都与描述其特性的特定 partnerLinkType 相关。每个合作伙伴链接还最多指定两个属性:

- myRole:表明业务流程本身的角色。
- partnerRole:表明合作伙伴的角色。

合作伙伴链接仅可以指定一个角色，通常同步请求/响应操作也仅能指定一个角色。对于异步操作，它指定两个角色。在本示例中，我们定义四个角色。第一个合作伙伴链接称作客户端，由 travelLT 合作伙伴链接类型描述其特性。此客户端调用该业务流程。我们需要指定 myRole 属性以描述此 BPEL 流程(travelService) 的角色。我们必须指定第二个角色:partnerRole。此处，该角色为 travelServiceCustomer，它描述 BPEL 流程客户端的特性。

第二个合作伙伴链接称作 employeeTravelStatus，由 employeeLT 合作伙伴链接类型描述其特性。它是 BPEL 流程与 Web 服务之间的一个同步请求/响应关系;我们再次仅指定一个角色。此时，该角色为 partnerRole，这是因为我们描述了 Web 服务(它是此 BPEL 流程的合作伙伴)的角色:

最后两个合作伙伴链接对应于航空公司 Web 服务。由于它们使用同一类型的 Web 服务，因此我们基于一个合作伙伴链接类型 flightLT 指定两个合作伙伴链接。此处，由于我们使用异步回调通信，因此需要两个角色。此 BPEL 流程 (myRole) 对于航空公司 Web 服务的角色为 airlineCustomer，而航空公司 (partnerRole) 的角色为 airlineService:

```
<partnerLinks>
<partnerLink name="client"
partnerLinkType="trv:travelLT"
myRole="travelService"
partnerRole="travelServiceCustomer"/>

<partnerLink name="employeeTravelStatus"
partnerLinkType="emp:employeeLT"
```

```
partnerRole="employeeTravelStatusService"/>

<partnerLink name="AmericanAirlines"
partnerLinkType="aln:flightLT"
myRole="airlineCustomer"
partnerRole="airlineService"/>

<partnerLink name="DeltaAirlines"
partnerLinkType="aln:flightLT"
myRole="airlineCustomer"
partnerRole="airlineService"/>
</partnerLinks>
```

变量 BPEL 流程中的变量用于存储消息以及对这些消息进行重新格式化和转换。您通常需要为发送到合作伙伴以及从合作伙伴收到的每个消息定义一个变量。就我们的流程而言，我们需要七个变量。我们将它们命名为 TravelRequest、EmployeeTravelStatusRequest、EmployeeTravelStatusResponse、FlightDetails、FlightResponseAA、FlightResponseDA 和 TravelResponse。

我们必须为每个变量指定类型。可以使用 WSDL 消息类型、XML 模式简单类型或 XML 模式元素。在我们的示例中，我们对所有变量使用 WSDL 消息类型:

```
<variables>
<!-- input for this process -->
<variable name="TravelRequest"
messageType="trv:TravelRequestMessage"/>
<!-- input for the Employee Travel Status web service -->
<variable name="EmployeeTravelStatusRequest"
messageType="emp:EmployeeTravelStatusRequestMessage"/>
<!-- output from the Employee Travel Status web service -->
<variable name="EmployeeTravelStatusResponse"
messageType="emp:EmployeeTravelStatusResponseMessage"/>
<!-- input for American and Delta web services -->
```

```
<variable name="FlightDetails"
messageType="aln:FlightTicketRequestMessage"/>
<!-- output from American Airlines -->
<variable name="FlightResponseAA"
messageType="aln:TravelResponseMessage"/>
<!-- output from Delta Airlines -->
<variable name="FlightResponseDA"
messageType="aln:TravelResponseMessage"/>
<!-- output from BPEL process -->
<variable name="TravelResponse"
messageType="aln:TravelResponseMessage"/>
</variables>
```

BPEL 流程主体流程主体指定调用合作伙伴 Web 服务的顺序。它通常以 <sequence>（用于定义多个将按顺序执行的操作）开始。在顺序中，我们首先指定启动业务流程的输入消息。我们使用 <receive> 构造（它等待匹配消息，在本示例中为 TravelRequest 消息）实现此目的。在 <receive> 构造中，我们不直接指定消息。而是指定合作伙伴链接、端口类型、操作名称以及可选变量（用于保存收到的消息以用于随后的操作）。

我们将消息接收与客户端合作伙伴链接在一起，并等待对端口类型 TravelApprovalPT 调用 TravelApproval 操作。我们将收到的消息存储到 TravelRequest 变量中:

```
<sequence>

<!-- Receive the initial request for business travel from client -->
<receive partnerLink="client"
portType="trv:TravelApprovalPT"
operation="TravelApproval"
variable="TravelRequest"
createInstance="yes" />
...
```

<receive> 等待客户端调用 TravelApproval 操作，并将传入的消息以及有关业务出差的参数存储到 TravelRequest 变量中。此处，此变量名与消息名相同，但并不一定要相同。

接下来，我们需要调用雇员出差状态 Web 服务。但在调用之前，我们必须为此 Web 服务准备输入。查看雇员 Web 服务的 WSDL，可以看到我们必须发送由雇员部分组成的消息。我们可以通过复制客户端发送的消息的雇员部分来构造此消息。编写相应的赋值语句:

```
...
<!-- Prepare the input for the Employee Travel Status Web Service -->
<assign>
<copy>
<from variable="TravelRequest" part="employee"/>
<to variable="EmployeeTravelStatusRequest" part="employee"/>
</copy>
</assign>
...
```

现在，我们就可以调用雇员出差状态 Web 服务了。为了进行同步调用，我们使用<invoke> 活动。我们使用 employeeTravelStatus 合作伙伴链接，并对 EmployeeTravelStatusPT 端口类型调用 EmployeeTravelStatus 操作。我们已经在 EmployeeTravelStatusRequest 变量中准备了输入消息。由于它是同步调用，因此该调用等待回应并将其存储在 EmployeeTravelStatusResponse 变量中:

```
...
<!-- Synchronously invoke the Employee Travel Status Web Service -->
<invoke partnerLink="employeeTravelStatus"
portType="emp:EmployeeTravelStatusPT"
operation="EmployeeTravelStatus"
inputVariable="EmployeeTravelStatusRequest"
outputVariable="EmployeeTravelStatusResponse" />
...
```

下一步是调用上述两个航空公司 Web 服务。同样，我们先准备所需的输入消息(这两个 Web 服务的输入消息相同)。FlightTicketRequest 消息包含两部分:

- flightData:它从客户端消息 (TravelRequest) 中检索而得。
- travelClass:它从 EmployeeTravelStatusResponse 变量中检索而得。

因此，我们编写一个包含两个 copy 元素的赋值:

```
...
<!-- Prepare the input for AA and DA -->
<assign>
<copy>
<from variable="TravelRequest" part="flightData"/>
<to variable="FlightDetails" part="flightData"/>
</copy>
<copy>
<from variable="EmployeeTravelStatusResponse" part="travelClass"/>
<to variable="FlightDetails" part="travelClass"/>
</copy>
</assign>
...
```

输入数据包含需要传递给航空公司 Web 服务的数据。由于格式相同，因此我们可以使用一个简单复制直接传递它。在实际情况下，通常需要执行转换。为此，可以使用具有 <assign> 的 XPath 表达式、使用转换服务(如 XSLT 引擎)或使用由特定 BPEL 服务器提供的转换功能。

现在，我们准备调用这两个航空公司 Web 服务。我们将进行并发的异步调用。为表述并发，BPEL 提供了 活动。对每个 Web 服务的调用将包含两个步骤:

使用 <invoke> 活动进行异步调用。
使用 <receive> 活动等待回调。

我们使用 <sequence> 对这两个活动进行分组。这两个调用只在合作伙伴链接名称上存在差别。我们对一个调用使用 AmericanAirlines，对另一个调用使用 DeltaAirlines。两者均对 FlightAvailabilityPT 端口类型调用 FlightAvailability 操作，发送 FlightDetails 变量中的消息。

使用 <receive> 活动接收回调。我们再次使用这两个合作伙伴链接名。<receive> 等待对 FlightCallbackPT 端口类型调用 FlightTicketCallback 操作。我们将结果消息分别存储到 FlightResponseAA 和 FlightResponseDA 变量中:

```
...
(并发)
<!-- Make a concurrent invocation to AA in DA -->
<flow>
```

```
                                    <sequence>
<!-- Async invoke of the AA web service and wait for the callback-->

                        <invoke partnerLink="AmericanAirlines"
                          portType="aln:FlightAvailabilityPT"
                             operation="FlightAvailability"
                           inputVariable="FlightDetails" />
                        <receive partnerLink="AmericanAirlines"
                             portType="aln:FlightCallbackPT"
                             operation="FlightTicketCallback"
                              variable="FlightResponseAA" />
                                    </sequence>
                                    <sequence>
<!-- Async invoke of the DA web service and wait for the callback-->

                          <invoke partnerLink="DeltaAirlines"
                          portType="aln:FlightAvailabilityPT"
                             operation="FlightAvailability"
                           inputVariable="FlightDetails" />
                        <receive partnerLink="DeltaAirlines"
                           portType="aln:FlightCallbackPT"
                             operation="FlightTicketCallback"
                              variable="FlightResponseDA" />
                                    </sequence>
                                      </flow>
                                        ...
```

在该流程的这个阶段，我们收到两个机票报价。在下一步中，我们必须选择一个机票报价。为此，我们使用 <switch> 活动。

```
...
<!-- Select the best offer and construct the TravelResponse -->
<switch>

<case condition="bpws:getVariableData('FlightResponseAA',
'confirmationData','/confirmationData/Price')
<= bpws:getVariableData('FlightResponseDA',
'confirmationData','/confirmationData/Price')">

<!-- Select American Airlines -->
<assign>
<copy>
<from variable="FlightResponseAA" />
<to variable="TravelResponse" />
</copy>
</assign>
</case>

<otherwise>
<!-- Select Delta Airlines -->
<assign>
<copy>
<from variable="FlightResponseDA" />
<to variable="TravelResponse" />
</copy>
</assign>
</otherwise>
</switch>
...
```

在 <case> 元素中，我们检查美国航空公司的机票报价 (FlightResponseAA) 是等于还是低于达美航空公司的机票报价 (FlightResponseDA)。为此，我们使用 BPEL 函数 getVariableData 并指定变量名。价格位于 confirmationData 消息的内部，虽然它是唯一的消息部分，但我们仍必须指定它。我们还必须指定查询表达式以找到价格元素。此处，我们采用简单的 XPath 1.0 表达式。

如果美国航空公司的机票报价低于达美航空公司的机票报价，则将 FlightResponseAA 变量复制到 TravelResponse 变量（我们最终将此变量返回给客户端）。否则，我们将复制 FlightResponseDA 变量。

我们已经到达此 BPEL 业务流程的最后一步 — 使用 <invoke> 活动将回调返回给客户端。对于此回调，我们使用客户端合作伙伴链接并对 ClientCallbackPT 端口类型调用 ClientCallback 操作。保存答复消息的变量为 TravelResponse：

```
<!-- Make a callback to the client -->
<invoke partnerLink="client"
portType="trv:ClientCallbackPT"
operation="ClientCallback"
inputVariable="TravelResponse" />
</sequence>
</process>
```

到此，我们已经完成了我们的第一个 BPEL 业务流程规范。您可以看到，BPEL 并不是很复杂，并允许相对简单和自然的业务流程规范。

第 5 步:部署和测试

我们部署到 Oracle BPEL Process Manager 的每个 BPEL 流程都需要一个流程描述符。BPEL 标准不包括此流程描述符，且它特定于 BPEL 服务器。部署流程描述符是流程在给定平台上的唯一实现部分，必须重写它才能在不同 BPEL 引擎上运行该流程。Oracle 流程描述符是一个 XML 文件，它指定有关 BPEL 流程的以下细节:BPEL 源文件名、BPEL 流程名 (ID)、所有合作伙伴链接 WSDL Web 服务的 WSDL 位置以及可选的配置属性。流程描述符的默认文件名为 bpel.xml，但我们可以使用任何其他名称:

```
<BPELSuitcase>

<BPELProcess src="Travel.bpel" id="TravelProcessCh4">
<partnerLinkBindings>

<partnerLinkBinding name="client">
<property name="wsdlLocation">
```

```
Travel.wsdl

</property>

</partnerLinkBinding>


<partnerLinkBinding name="employeeTravelStatus">

<property name="wsdlLocation">

http://localhost:9700/orabpel/default/Employee/Employee?wsdl

</property>

</partnerLinkBinding>


<partnerLinkBinding name="AmericanAirlines">

<property name="wsdlLocation">

http://localhost:9700/orabpel/default/AmericanAirline/AmericanAirline?wsdl

</property>

</partnerLinkBinding>


<partnerLinkBinding name="DeltaAirlines">

<property name="wsdlLocation">

http://localhost:9700/orabpel/default/DeltaAirline/DeltaAirline?wsdl

</property>

</partnerLinkBinding>


</partnerLinkBindings>

</BPELProcess>

</BPELSuitcase>
```

我们现在准备启动 BPEL Process Manager。可以从开始菜单(如果使用 Windows)或通过执行 c:\orabpel\bin 目录中的 startOraBPEL 脚本(假设 Oracle BPEL Process Manager 已经安装到 c:\orabpel 中)来执行此操作。为便于访问，建议将此目录包含在 PATH 中。

Oracle BPEL Process Manager 包含一个名为 obant 的 Ant 实用程序，用于配置复杂的编译和部署方案。obant 只是一个围绕标准 Ant 的包装程序，用于设置环境，然后调用标准 Ant Java 任务。要使用它，我们必须准备相应的项目文件(通常名为 build.xml)。以下是出差示例流程的项目文件:

```
<?xml version="1.0"?>

<project name="TravelProcessCh4" default="main" basedir=".">


<property name="deploy" value="default"/>

<property name="rev" value="1.0"/>


<target name="main">

<bpelc home="${home}" rev="${rev}" deploy="${deploy}"/>

</target>


</project>
```

要编译和部署此 BPEL 流程，我们只需从命令行启动 obant。

既然我们已经在 Oracle BPEL 服务器上成功部署了 BPEL 流程，那我们就执行它。Oracle BPEL Process Manager 提供了一个 BPEL 控制台，通过它可以在 BPEL 服务器域中执行、监视、管理和调试 BPEL 流程。可以通过 http://localhost:9700/BPELConsole/ 访问 BPEL 控制台。我们必须单击流程名、填写以下表单并按 Post XML Message 按钮:

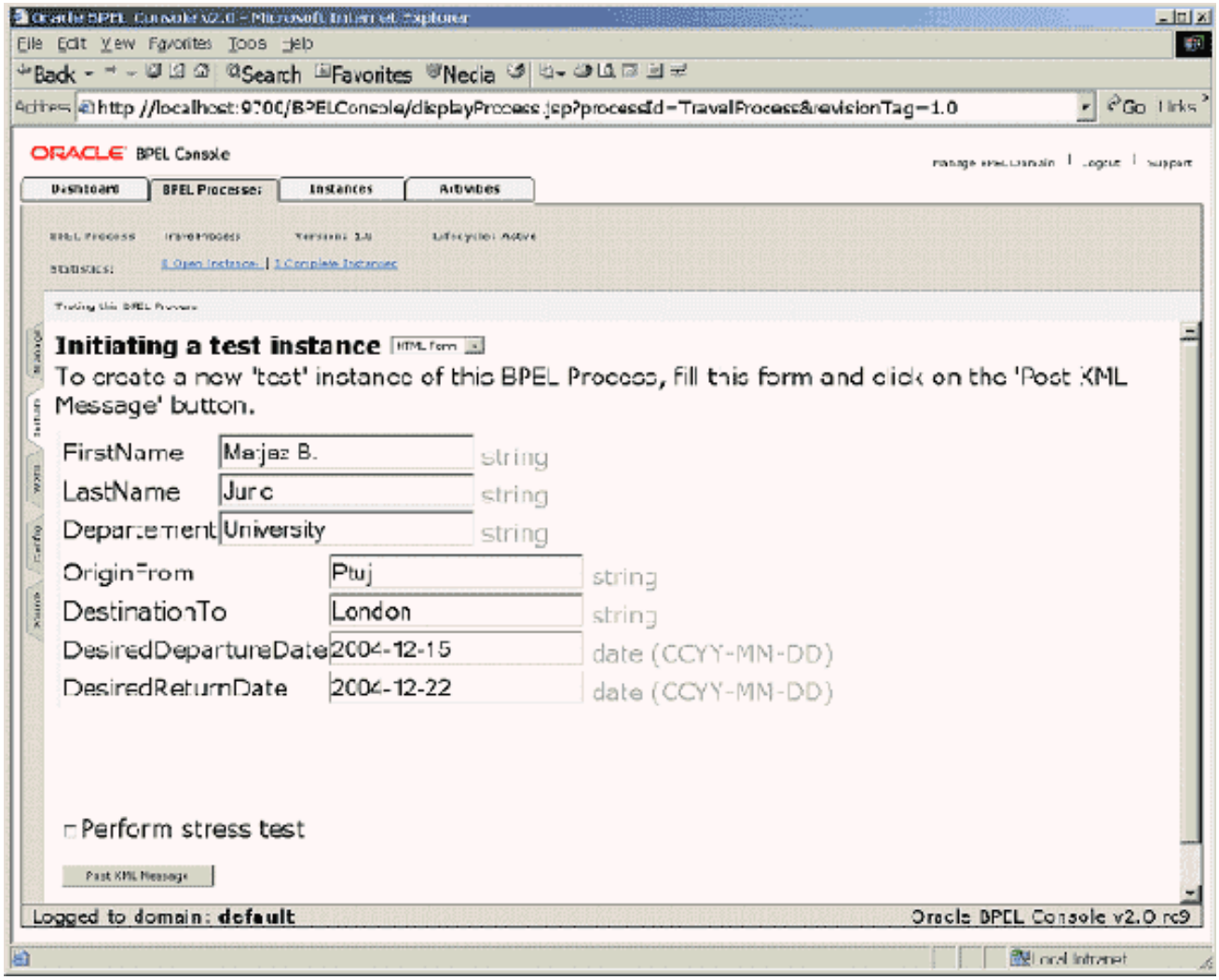

图 7:BPEL 控制台

我们现在看到一个屏幕，通知我们正在异步处理流程实例。我们可以选择执行、实例审计或实例调试的可视化流。实例的可视化流以图形方式显示了 BPEL 流程实例的执行。我们可以监视流程的执行及其状态(正在运行、已完成、已取消或过时):

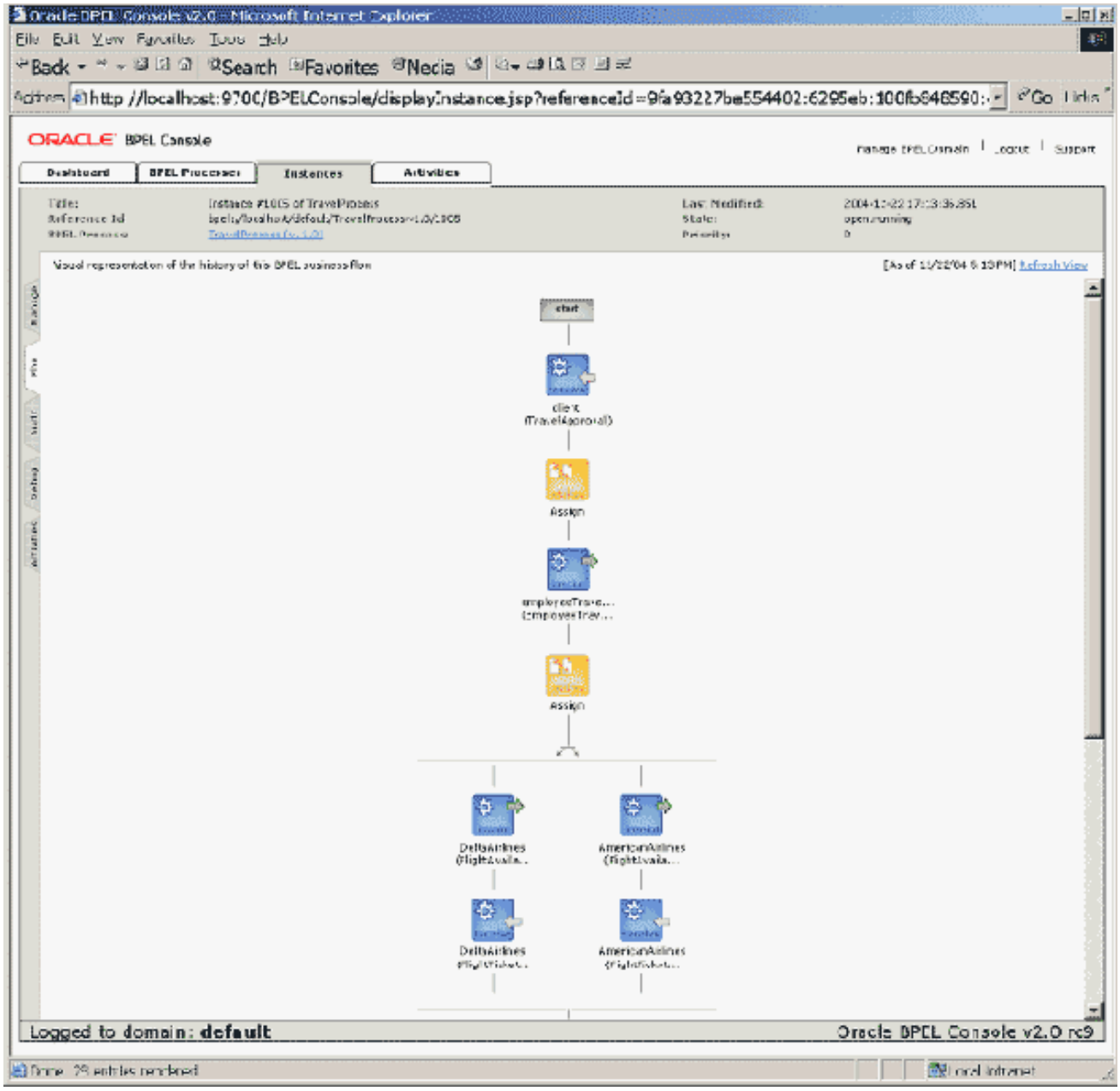

图 8:实例流的图形视图

结论

现在，您已经熟悉了使用 BPEL 组合 Web 服务的基本概念，您可以更深入地研究更高级的概念。我的下一篇文章将介绍一些高级 BPEL 特性，如故障处理、范围、补偿、并发活动以及事件处理。