The SOA Magazine

Feature Article

Processes as Services:
Contract Design for Services that Encapsulate WS-BPEL Process Definitions

by Yuli Vasiliev

Published: November 6, 2007 (SOA Magazine Issue XII: November 2007, Copyright © 2007)

Download this article as a PDF document.

Abstract: Now that WS-BPEL 2.0 has become a ratified industry standard, it is making its way into the IT mainstream and is therefore becoming important to the design of SOA implementations comprised of Web services. This article is comprised of excerpts from "SOA and WS-BPEL" [REF-1] to provide an exploration of the design of WSDL definitions for Web services required to encapsulate WS-BPEL process logic. Because SOA advocates a "contract first" approach to the design of services, understanding the unique service contract design consideration raised by WS-BPEL is a key success factor to building effective process and task services.

Introduction: SOA and WS-BPEL

Service-Oriented Architecture (SOA), as an architectural platform, is adopted today by many businesses as an efficient means for integrating enterprise applications built of Web services – loosely coupled pieces of software that encapsulate their logic within a distinct context and can be easily combined into a composite solution. Although building applications that enable remote access of resources and functionality is nothing new, doing so according to the principles of service-orientation, such as loose coupling, represents a relatively new approach to building composite solutions.

While Web Services is a technology that defines a standard mechanism for exposure and consumption of data and application logic over Internet protocols such as HTTP, WS-BPEL is an orchestration language that is used to define business processes describing Web services interactions, thus providing a foundation for building service-oriented solutions based on Web services. So, to build a service-oriented solution utilizing Web services with WS-BPEL, you have to perform the following two general steps:

- Build and then publish Web services to be utilized within the solution.
- Compose the Web services into business flows with WS-BPEL.

This article explores the relationship between WS-BPEL and Web services with an emphasis on the design of WSDL definitions.

Service Composition Basics

There are actually several ways in which services can be organized to form a composite solution. For example, you might create a composite service as a PHP script exposed as a Web service, and which programmatically invokes other services as necessary. However, the most common way to build service compositions (especially more complex compositions) is by using WS-BPEL, an orchestration language that allows you to create orchestrations – a composite, controller service defining how the services being consumed will interoperate to get the job done.

Orchestration

An orchestration assembles services into an executable business process that is to be executed by an orchestration engine. Schematically, an orchestration might look like Figure 1:

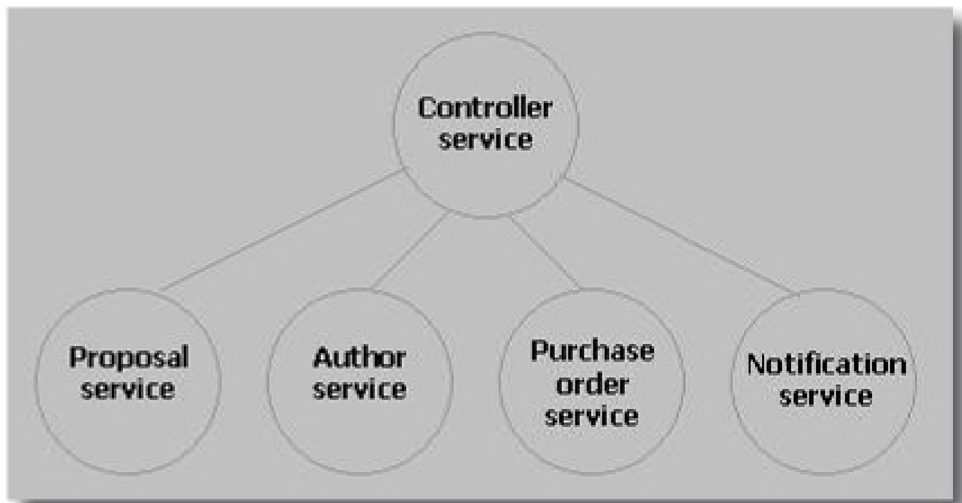

Figure 1: A controller service organizing other services into an SOA composition.

As you can see, the diagram in Figure 1 illustrates an assembly of services coordinated by the logic encapsulated in the controller service. This controller service may be a WS-BPEL business process, which, when executed against the orchestration engine, completes a certain business task.

Built with the WS-BPEL orchestration language, a controller service may also be referred to as an Orchestrated Task Service. As a Web service, it should have a corresponding WSDL document describing the service to its consumers. Building WSDL definitions for composite services built with WS-BPEL is discussed shortly.

You can create an orchestration to be used as a service within another, larger orchestration. For example, the orchestration depicted in Figure 1 might be a part of a WS-BPEL orchestration automating the entire editorial process – from accepting the proposal to publishing the article.

Choreography

The Web Services Choreography specification along with its corresponding Web Services Choreography Description Language (WS-CDL) provides another way to building SOA compositions. While WS-BPEL is used to orchestrate services into composite solutions usually expressing organization-specific business process flows, the Web Services Choreography Description Language (WS-CDL) allows you to describe peer-to-peer relationships between Web services and/or other participants within or across trust boundaries.

Unlike an orchestration, a choreography does not imply a centralized control mechanism, assuming control is shared between the interacting participants. What this means is that an orchestration represents an executable process to be executed by an orchestration engine in one place, whereas a choreography in essence represents a description of how to distribute control between the collaborating participants, using no single engine to get the job done.

To define a choreography, you create a WS-CDL choreography description document that will serve as the contract between the interacting participants. Specifically, a WS-CDL document describes the message exchanges between the collaborating participants, defining how these participants must be used together to achieve a common business goal. For example, there may be a choreography enabling collaboration between an orchestration, a controller service representing a WS-BPEL process and a client service interacting with that controller service.

Schematically, this might look like Figure 2:

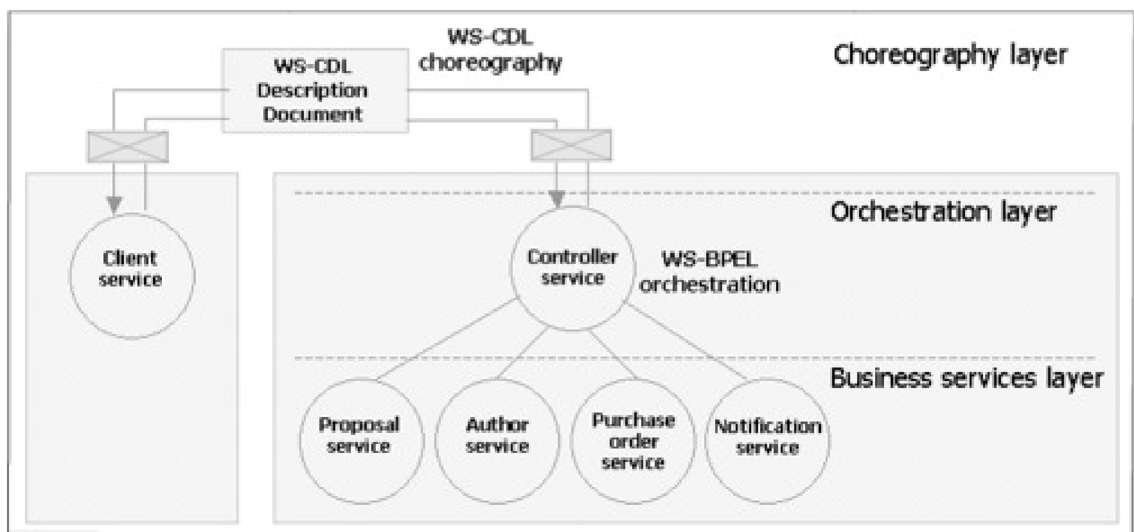

Figure 2: A choreography between a composite service based on a WS-BPEL business process and a consumer of that service.

In this scenario, the choreography layer is used to specify peer-to-peer collaborations of two services. In particular, the WS-CDL choreography document describes message exchanges between the composite service discussed in the preceding section and one of its consumers. Working with WS-BPEL

As mentioned earlier, WS-BPEL is an orchestration language used to describe execution logic of Web services applications by defining their control flows, providing a way for partner services to share a common context. To clarify, partner services are those that interact with the WS-BPEL process.

WS-BPEL is based on several specifications, such as SOAP, WSDL, and XML Schema, where WSDL perhaps is the most important one. WSDL is what makes a service usable within composite services based on WS-BPEL. WS-BPEL allows you to define business processes interacting with cooperating services through WSDL descriptions (as explained shortly).

It is interesting to note that although WS-BPEL is currently the most popular executable business process language, it is not the only way to define execution logic of an application based on Web services. There are some other specifications, such as XLANG, WSFL, XPDL, and BPML, each of which might be used as an alternative to WS-BPEL.

With WS-BPEL, you build a business process by integrating a collection of Web services into a business process flow. A WS-BPEL business process specifies how to coordinate the interactions between an instance of that process and its partner services. Figure 3 illustrates an example of a workflow diagram representing a WS-BPEL business process.

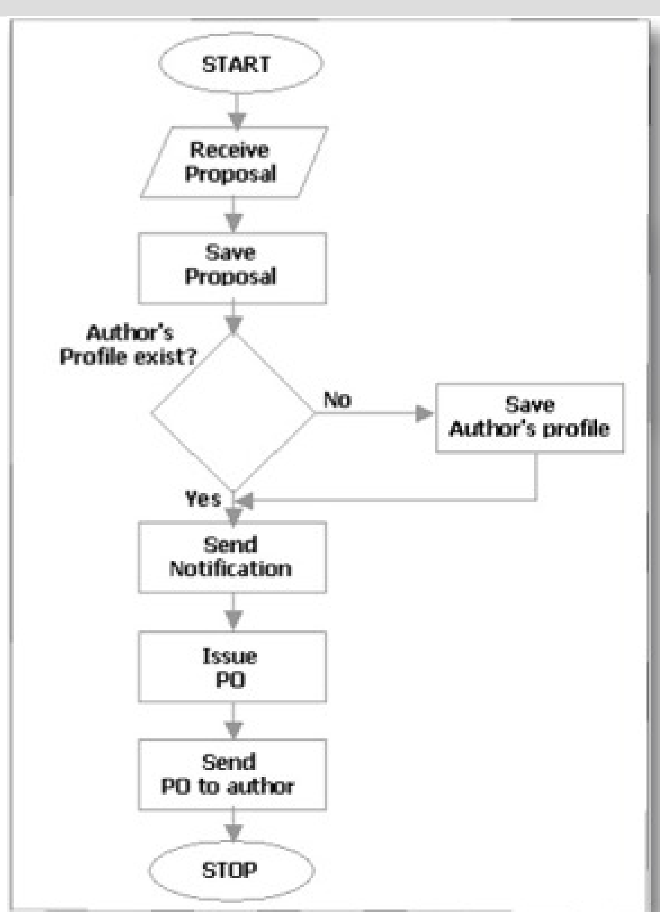

Figure 3: The workflow diagram of the WS-BPEL process encapsulated into the composite service depicted in Figure 1.

As you can see, the WS_BPEL process depicted in the figure integrates the services required to complete the steps performed after accepting a proposal into an end-to-end process. In this particular example, the process integrates four Web services, as depicted in Figure 1 earlier. As you will see in the next section, a WS-BPEL process connects to a Web service through a partner link defined in the WSDL document.

Apart from the ability to invoke multiple services, a WS-BPEL process may manipulate XML data and perform parallel execution, conditional branching, and looping to control the flow of the process. For example, in the above process you use a switch activity, setting up the two branches. If the proposal being processed has been submitted by a new author, the process will call the Author service's operation responsible for saving the information about the author in the database. Otherwise, this step is omitted.

Designing WSDL Definitions for Services that Encapsulate WS-BPEL

A WS-BPEL orchestration can be associated with a WSDL definition, thus making it possible to treat that orchestration as a service itself that can be invoked by another service or can be part of another orchestration or choreography.

To exist as a service, a WS-BPEL process should have a corresponding WSDL document, making it possible for client services to invoke the process. For example, the WS-BPEL process depicted in Figure 3 might be associated with the following WSDL definition:

Example 1

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="proposalProcessingService"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsd1="http://localhost/WebServices/schema/"
     xmlns:plnk="http://schemas.xmlsoap.org/ws/2004/03/partner-link/"
```

```
     targetNamespace="http://localhost/WebServices/ch1/proposalProcessing/">
   <types>
    <xsd:schema targetNamespace="http://localhost/WebServices/ch1/proposalProcessing.wsdl">
    <xsd:import namespace="http://localhost/WebServices/schema/"
      schemaLocation="http://localhost/WebServices/schema/propProc.xsd"/>
    </xsd:schema>
   </types>
   <message name="receiveProposalInput">
        <part name="body" element="xsd1:proposalDocType"/>
   </message>
   <message name="receiveProposalOutput">
        <part name="body" element="xsd:string"/>
   </message>
   <portType name="proposalProcessingServicePortType">
       <operation name="receiveProposal">
          <input message="tns:receiveProposalInput"/>
          <output message="tns:receiveProposalOutput"/>
       </operation>
   </portType>
   <plink:partnerLinkType name="proposalProcessingService">
         <plink:role name="proposalProcessingServiceRole">
           <portType="tns:proposalProcessingServicePortType"/>
        </plink:role>
   </plink:partnerLinkType>
</definitions>
```

As you can see, this WSDL document doesn't contain binding or service elements. The fact is that a WSDL document of a WS-BPEL process service contains only the abstract definition of the service and partnerLinkType sections that represent the interaction between the process service and its client services. In this particular example, the WSDL definition contains only one partnerLinkType section, supporting one operation used by a client to initiate the process.

A partnerLinkType section defines up to two roles, each of which in turn is associated with a portType defined within the WSDL document earlier. WS-BPEL uses the partner links mechanism to define a relationship between a WS-BPEL process and the involved parties. As you will learn later in this section, a WS-BPEL process definition contains partnerLink elements to specify the interactions between a WS-BPEL process and its clients and partners. Each partnerLink in a WS-BPEL process definition document is associated with a partnerLinkType defined in a corresponding WSDL document. Schematically, this looks like Figure 4.

Once you have created the WSDL definition for a process service, make sure to modify WSDL documents of the services that will be invoked during the process execution as partner services. To enable a service to be part of a WS-BPEL orchestration, you might want to add a partnerLinkType element to the corresponding WSDL document. For example, to enable the Notification service to participate in the orchestration depicted in Figure 3, you would need to create the following WSDL document:

Example 2

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="notificationService"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:plink="http://schemas.xmlsoap.org/ws/2004/03/partner-link/"
     targetNamespace="http://localhost/WebServices/ch1/notification/">
   <message name="sendMessageInput">
        <part name="body" element="xsd:string"/>
   </message>
   <message name="sendMessageOutput">
        <part name="body" element="xsd:string"/>
   </message>
   <portType name="notificationServicePortType">
       <operation name="sendMessage">
          <input message="tns:sendMessageInput"/>
```

```
          <output message="tns:sendMessageOutput"/>
       </operation>
   </portType>
   <binding name="notificationServiceBinding" type="tns:notificationServicePortType">
       <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sendMessage">
         <soap:operation soapAction="http://localhost/WebServices/ch1/sendMessage"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
       </operation>
   </binding>
   <service name="notificationService">
        <port name="notificationServicePort" binding="tns:notificationServiceBinding">
           <soap:address location="http://localhost/WebServices/ch1/SOAPserver.php"/>
        </port>
   </service>
   <plink:partnerLinkType name="notificationService">
         <plink:role name="notificationServiceRole">
           <portType="tns:notificationServicePortType"/>
        </plink:role>
   </plink:partnerLinkType>
</definitions>
```

Note the partnerLinkType block highlighted in bold. By including this section in the end of the WSDL document describing the service, you enable that service as a partner link, making it possible for it to be part of an orchestration.

As mentioned, WS-BPEL uses the partner links mechanism to model the services interacting within the business process. Here is a fragment of the definition of the WS-BPEL business process depicted in Figure 3 and representing the proposalProcessingService process service, showing the use of partner links:

Example 3

```
<process name="BusinessTravelProcess"
   targetNamespace="http://localhost/WebServices/ch1/proposalProcessing/"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:prc="http://localhost/WebServices/ch1/proposalProcessing/"
         xmlns:ntf="http://localhost/WebServices/ch1/notification/"

   <partnerLinks>
      <partnerLink name="client"
                   partnerLinkType="prc:proposalProcessingService"
                   myRole="proposalProcessingServiceRole"/>
      <partnerLink name="sendingNotification"
                   partnerLinkType="ntf:notificationService"
                   partnerRole="notificationServiceRole"/>
...
   </partnerLinks>
...
</process>
```

To save space, the above snippet shows only two partnerLink sections in the process definition. The first partnerLink section specifies the interaction between the WS-BPEL process and its clients. And the other partnerLink specifies the interaction between the WS-BPEL process and the Notification service that acts as a partner service here.

Figure 4 may help you gain a better understanding of how the partner links mechanism used in WS-BPEL works.

WSDL definition (fragment)

```
<portType name="processServicePortType">
  <operation name="serviceOperation">
    ...
  </operation>
</portType>
...
<plink:partnerLinkType name="processService">
  <plink:role name="processServiceRole">
    <portType="processServicePortType"/>
  </plink:role>
</plink:partnerLinkType>
```

WS-BPEL process definition (fragment)

```
<partnerLink name="client"
    partnerLinkType="processService"
    myRole="processServiceRole"/>
...
```

Figure 4: A high-level representation of how a partnerLinkType specified in the WSDL document is associated with a partnerLink defined in the WS-BPEL process definition.

The example depicted in the figure represents the relationship between a partnerLinkType defined in the WSDL document describing a WS-BPEL process service and a partnerLink specified in the process definition. When defining the partnerLinkType, you use the myRole attribute to specify the role of the WS-BPEL process. In contrast, to specify the role of a partner, you would use the partnerRole attribute, as shown in the process definition document discussed above.

Looking through the partnerLink sections in the process definition document discussed here, you may notice that they do not actually provide any information about the location of the WSDL documents containing the corresponding partner link types. This information is stored in the process deployment descriptor file. The format of this document varies depending on the WS-BPEL tool you are using.

Below are examples of two different descriptors, each created with a different tool:

Example 4

```
<?xml version="1.0" encoding="UTF-8"?>
<BPELSuitcase>
  <BPELProcess src="proposalProcess.bpel" id="proposalProcess">
    <partnerLinkBindings>
      <partnerLinkBinding name="client">
        <property name="wsdlLocation">proposalProcess.wsdl</property>
      </partnerLinkBinding>
      <partnerLinkBinding name="sendingNotification">
        <property name="wsdlLocation">http://localhost/WebServices/ch1/notification?wsdl</property>
      </partnerLinkBinding>
...
    </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>
```

Example 5

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<process ...>
 ...
   <references>
      <wsdl location="project:/proposalProcess/WSDL/proposalProcess.wsdl" namespace="http://
localhost/WebServices/ch1/proposalProcessing/"/>
      <wsdl location="project:/proposalProcess/WSDL/notification.wsdl" namespace="http://localhost/
WebServices/ch1/notification/"/>
   </references>
</process>
```

More Examples

As with regular service-oriented design, it is recommended that you begin the design of a process service with the service contract. Let's therefore follow this process in order to provide some more examples of WSDL and WS-BPEL definitions.

The Hello Service WSDL Definition (Abstract Description)

We'll starting with a new WSDL document for a simple Hello service:

Example 6

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions targetNamespace="http://localhost:8081/active-bpel/services/hello"
      xmlns:tns="http://localhost:8081/active-bpel/services/hello"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
<!-- abstract characteristics of the WS-BPEL process service -->
<message name="inputMessage">
   <part name="firstName" type="xsd:string"/>
</message>
<message name="outputMessage">
   <part name="hello" type="xsd:string"/>
</message>
<portType name="helloServicePT">
   <operation name="hello">
      <input name="inputMessage" message="tns:inputMessage"/>
      <output name="outputMessage" message="tns:outputMessage"/>
   </operation>
</portType>
<!-- partnerLinkType section representing interaction between the WS-BPEL service and its client -->
<plnk:partnerLinkType name="helloPartnerLinkType">
   <plnk:role name="helloServiceRole">
       <plnk:portType name="tns:helloServicePT"/>
   </plnk:role>
</plnk:partnerLinkType>
</definitions>
```

The highlighted block in Example 6 represents the partnerLinkType construct within which you define the relationship between the WS-BPEL process and its consumer (client service).

Note that you must specify a partnerLinkType construct for each partner service involved. In this particular case, though, the WS-BPEL process has only one partner, its consumer, and, thus, you specify only one partnerLinkType construct in the WSDL document describing the WS-BPEL process service.

The Hello Service WSDL Definition (Abstract and Concrete Descriptions)

As you may have noticed, the WSDL definition in Example 6 doesn't contain deployment information (binding and address information). The fact is that the client services will use a modified version of this document. Based on the above WSDL code, a WS-BPEL engine can dynamically generate the document containing the required binding and

address information. As a result, when we request this WSDL definition with a browser, we should receive the following output:

Example 7

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions targetNamespace="http://localhost:8081/active-bpel/services/hello"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:tns="http://localhost:8081/active-bpel/services/hello"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <message name="outputMessage">
    <part name="hello" type="xsd:string" />
   </message>
   <message name="inputMessage">
    <part name="firstName" type="xsd:string" />
   </message>
   <portType name="helloServicePT">
    <operation name="hello">
     <input message="tns:inputMessage" name="inputMessage" />
     <output message="tns:outputMessage" name="outputMessage" />
    </operation>
   </portType>
   <binding name="helloServicePTBinding" type="tns:helloServicePT">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" xmlns:soap="http://
schemas.xmlsoap.org/wsdl/soap/" />
    <operation name="hello">
     <soap:operation soapAction="" style="rpc" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
      <input>
       <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" xmlns:
soap="http://schemas.xmlsoap.org/wsdl/soap/" />
      </input>
      <output>
       <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" xmlns:
soap="http://schemas.xmlsoap.org/wsdl/soap/" />
      </output>
    </operation>
  </binding>
  <service name="helloServicePT">
   <port binding="tns:helloServicePTBinding" name="helloServicePTPort">
     <soap:address location="http://localhost:8081/active-bpel/services/helloServicePT" xmlns:
soap="http://schemas.xmlsoap.org/wsdl/soap/" />
   </port>
  </service>
  </definitions>
```

The WS-BPEL engine implicitly generated the binding and service definitions, thus making it possible for a client partner service to consume the process service discussed here.

The Hello WS-BPEL Process Definition

The next step is to implement the WS-BPEL process definition. Here are the contents of the corresponding hello.bpel file:

Example 8

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="hello"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:lns="http://localhost:8081/active-bpel/services/hello"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
         suppressJoinFailure="yes"
         targetNamespace="http://hello">
   <partnerLinks>
      <partnerLink myRole="helloServiceRole" name="customer" partnerLinkType="lns:
helloPartnerLinkType"/>
   </partnerLinks>
   <variables>
      <variable messageType="lns:inputMessage" name="inputMessage"/>
      <variable messageType="lns:outputMessage" name="outputMessage"/>
   </variables>
   <sequence>
      <receive createInstance="yes"
               operation="hello"
               partnerLink="customer"
               portType="lns:helloServicePT"
               variable="inputMessage"/>
      <assign>
         <copy>
            <from expression="concat( 'Hello, ', bpws:getVariableData('inputMessage',
'firstName'),'!' )"/>
            <to part="hello" variable="outputMessage"/>
         </copy>
      </assign>
      <reply operation="hello"
             partnerLink="customer"
             portType="lns:helloServicePT"
             variable="outputMessage"/>
   </sequence>
</process>
```

The partnerLinks section contains partnerLink sections establishing relationships with the partner service that interact with the business process during the course of its execution. In Example 8, the partnerLinks section contains only one entry – the one that represents the relationship between the process service and the client service. In this case, you use the myRole attribute in the partnerLink element because the WS-BPEL process service is the service provider to the client service.

It is important to note that the name specified in the partnerLinkType attribute of the partnerLink element is equal to the one specified in the corresponding partnerLinkType section in the hello.wsdl document discussed earlier.

The sequence of activities defined within the sequence construct is fairly straightforward. The first one is the receive activity that is used to handle a request message sent by a client service. By setting the createInstance attribute of the receive activity to yes, you instruct the WS-BPEL engine to create a process instance when this activity receives a request message.

The next activity defined within the sequence construct is assign. Here you actually generate the output message, based on the information arrived with the input message.

Finally, you define the reply activity, which is responsible for sending the output message to the client service. Now that the hello process service have been deployed and you know how to obtain the WSDL document describing this service to its clients, it's time to test it. For that, you might create and then run the following PHP script:

Example 9

```
<?php
 //File: SoapClient_hello.php
 $client = new SoapClient("http://localhost:8081/active-bpel/services/helloServicePT?wsdl");
 try {
  print($client->hello('Larry'));
 }
 catch (SoapFault $e) {
  print $e->getMessage();
 }
?>
```

When executed, the above script should invoke the hello process service discussed in the preceding sections, and output the following hello message:

Hello, Larry!

The poInfo WSDL Definition

The hello process service we justed covered is a simplified example of a WS-BPEL process service. It doesn't use partner services to get the job done, simply composing a hello message based on the data sent by the client. In contrast, a real-world WS-BPEL process may invoke a lot of partner services during its execution.

The following example describes the poInfo WS-BPEL process service that interacts with two partner services: poOrderDocService and poOrderStatusService.

Let's start with the poInfo WSDL definition:

Example 10

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="poInfo"
    targetNamespace="http://localhost:8081/active-bpel/services/poInfoService.wsdl"
    xmlns:tns="http://localhost:8081/active-bpel/services/poInfoService.wsdl"
    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ns2="http://localhost/WebServices/wsdl/poOrderDoc"
    xmlns:ns3="http://localhost/WebServices/wsdl/poOrderStatus"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import namespace="http://localhost/WebServices/wsdl/poOrderDoc"
          location="http://localhost/WebServices/wsdl/po_orderdoc.wsdl"/>
  <import namespace="http://localhost/WebServices/wsdl/poOrderStatus"
          location="http://localhost/WebServices/wsdl/po_orderstatus.wsdl"/>
  <types>
    <schema attributeFormDefault="qualified"
            elementFormDefault="qualified"
            targetNamespace="http://localhost:8081/active-bpel/services/poInfoService.wsdl"
            xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="poInfoRequest">
        <complexType>
          <sequence>
            <element name="pono" type="xsd:string"/>
            <element name="par" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
     </schema>
  </types>
  <message name="poInfoResponseMessage">
    <part name="payload" type="xsd:string"/>
  </message>
  <message name="poInfoRequestMessage">
    <part name="payload" element="tns:poInfoRequest"/>
  </message>
  <portType name="poInfoPT">
    <operation name="getInfo">
      <input message="tns:poInfoRequestMessage"/>
      <output message="tns:poInfoResponseMessage"/>
    </operation>
  </portType>
  <plnk:partnerLinkType name="poInfoLT">
   <plnk:role name="poInfoProviderRole">
      <plnk:portType name="tns:poInfoPT"/>
   </plnk:role>
  </plnk:partnerLinkType>
```

```
  <plnk:partnerLinkType name="poDocLT">
   <plnk:role name="poDocProviderRole">
      <plnk:portType name="ns2:poOrderDocServicePortType"/>
   </plnk:role>
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name="poStatusLT">
   <plnk:role name="poStatusProviderRole">
      <plnk:portType name="ns3:poOrderStatusServicePortType"/>
   </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

The poInfo process service is invoked when a client sends a request message containing two parameters: pono and par. The first one specifies the pono of the order document on which you need to get information, while the second one specifies what kind of information should be returned, meaning two possible choices: the entire document or the status of the document.

Example 10 also shows how the WSDL definition imports two definitions describing the poOrderDocService and poOrderStatusService partner services. The example assumes that you have the po_orderdoc.wsdl and po_orderstatus.wsdl documents created.

Note the use of the poInfoRequest complex type when defining the input message. This structure makes it possible for the client to send two parameters, namely pono and par within a single request message.

Another important thing to note here is the use of three partner links. The first one defines the interaction between the WS-BPEL process service and its client, while the other two define the relationships between the WS-BPEL service and the poOrderDocService and poOrderStatusService partner services respectively.

The poInfo WS-BPEL Definition

The following WS-BPEL definition is a bit more complicated than the one's shown so far. This is because the poInfo. bpel definition shown below contains conditional logic:

Example 11

```
<?xml version="1.0" encoding="UTF-8"?>
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
     xmlns:ns1="http://localhost:8081/active-bpel/services/poInfoService.wsdl"
     xmlns:ns2="http://localhost/WebServices/wsdl/poOrderDoc"
     xmlns:ns3="http://localhost/WebServices/wsdl/poOrderStatus"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     name="poInfo.bpel"
     suppressJoinFailure="yes"
     targetNamespace="http://poInfo.bpel">
   <import importType="http://schemas.xmlsoap.org/wsdl/"
           location="wsdl/poInfo.wsdl"
           namespace="http://localhost:8081/active-bpel/services/poInfoService.wsdl"/>
   <import importType="http://schemas.xmlsoap.org/wsdl/"
           location="http://localhost/WebServices/wsdl/po_orderdoc.wsdl"
           namespace="http://localhost/WebServices/wsdl/poOrderDoc"/>
   <import importType="http://schemas.xmlsoap.org/wsdl/"
           location="http://localhost/WebServices/wsdl/po_orderstatus.wsdl"
           namespace="http://localhost/WebServices/wsdl/poOrderStatus"/>

   <partnerLinks>
      <partnerLink myRole="poInfoProviderRole" name="poInfoProvider" partnerLinkType="ns1:poInfoLT"/>
      <partnerLink name="poDocRequester" partnerLinkType="ns1:poDocLT"
partnerRole="poDocProviderRole"/>
      <partnerLink name="poStatusRequester" partnerLinkType="ns1:poStatusLT"
partnerRole="poStatusProviderRole"/>
   </partnerLinks>
   <variables>
      <variable messageType="ns1:poInfoRequestMessage" name="poInfoRequestMessage"/>
```

```
      <variable messageType="ns1:poInfoResponseMessage" name="poInfoResponseMessage"/>
      <variable messageType="ns2:getOrderDocInput" name="poDocRequestMessage"/>
      <variable messageType="ns2:getOrderDocOutput" name="poDocResponseMessage"/>
      <variable messageType="ns3:getOrderStatusInput" name="poStatusRequestMessage"/>
      <variable messageType="ns3:getOrderStatusOutput" name="poStatusResponseMessage"/>
   </variables>
   <sequence>
     <receive createInstance="yes"
              operation="getInfo"
              partnerLink="poInfoProvider"
              portType="ns1:poInfoPT"
              variable="poInfoRequestMessage"/>
      <if>
       <condition>($poInfoRequestMessage.payload/ns1:par = 'doc')</condition>
       <sequence>
        <assign>
         <copy>
            <from part="payload" variable="poInfoRequestMessage">
               <query>ns1:pono</query>
            </from>
            <to part="pono" variable="poDocRequestMessage"/>
         </copy>
        </assign>
        <invoke inputVariable="poDocRequestMessage"
              outputVariable="poDocResponseMessage"
              operation="getOrderDoc"
              partnerLink="poDocRequester"
              portType="ns2:poOrderDocServicePortType">
        </invoke>
        <assign>
         <copy>
            <from>$poDocResponseMessage.doc</from>
            <to>$poInfoResponseMessage.payload</to>
         </copy>
        </assign>
       </sequence>
       <elseif>
        <condition>($poInfoRequestMessage.payload/ns1:par = 'status')</condition>
        <sequence>
        <assign>
         <copy>
            <from part="payload" variable="poInfoRequestMessage">
               <query>ns1:pono</query>
            </from>
            <to part="pono" variable="poStatusRequestMessage"/>
         </copy>
        </assign>
        <invoke inputVariable="poStatusRequestMessage"
              outputVariable="poStatusResponseMessage"
              operation="getOrderStatus"
              partnerLink="poStatusRequester"
              portType="ns3:poOrderStatusServicePortType">
        </invoke>
        <assign>
         <copy>
            <from>$poStatusResponseMessage.status</from>
            <to>$poInfoResponseMessage.payload</to>
         </copy>
        </assign>
       </sequence>
       </elseif>
       <else>
        <assign>
         <copy>
```

```
            <from>'Wrong input parameter. Should be either doc or status!'</from>
            <to>$poInfoResponseMessage.payload</to>
         </copy>
        </assign>
       </else>
      </if>
      <reply operation="getInfo"
             partnerLink="poInfoProvider"
             portType="ns1:poInfoPT"
             variable="poInfoResponseMessage"/>
   </sequence>
</process>
```

The most interesting part of the WS-BPEL process definition in Example 11 is the if/elseif/else construct. Example 12 shows how it looks schematically:

Example 12

```
   <sequence>

      activities

      <if>
       <condition>...</condition>
       <sequence>

        activities

       </sequence>
     <elseif>
       <condition>...</condition>

       <sequence>

        activities

       </sequence>
      </elseif>
      <else>
       activity
      </else>
     </if>

     activities

   </sequence>
```

Note the use of the inner sequence constructs in Example 12. The fact is that WS-BPEL doesn't allow you to use more than one activity within if or elseif or else blocks. That is why you have to enclose a set of activities within any of those blocks by sequence.

Assuming you put the remaining parts in place to make this process execute successfully, you can test it with the following PHP script:

Example 13

```
<?php
 //File: SoapClient_poInfo.php
 $client = new SoapClient("http://localhost:8081/active-bpel/services/poInfoService?wsdl");
 $xmldoc = '<wrapper><pono>108128476</pono><par>doc</par></wrapper>';
 $xmldoc = simplexml_load_string($xmldoc);
 try {
  print($client->getInfo($xmldoc));
```

```
 }
 catch (SoapFault $e) {
  print $e->getMessage();
 }
?>
```

This should output the entire po document whose pono is 108128476. When setting the $xmldoc variable in the above script as shown below:

```
$xmldoc = '<wrapper><pono>108128476</pono><par>status</par></wrapper>';
```

You should receive a short message: shipped, which indicates the status of the document.

Conclusion

Process services may appear to establish a Web service endpoint just like any other service within an SOA implementation. However, because of the specific requirements that the WS-BPEL language places on WSDL definitions, additional constructs need to be added and optimized. It is important to understand these new elements and how they position a service as one of potentially many partner services within a larger service composition. Because although any given WS-BPEL process will compose other services, by establishing its own WSDL service contract, it becomes as composable as any other service. It becomes a process that also exists as a service.

References

[REF-1] "SOA and WS-BPEL", Yuli Vasiliev, Packt Publishing (ISBN: 184719270X)

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL

Copyright © 2006-2007 SOA Systems Inc.
All Rights Reserved