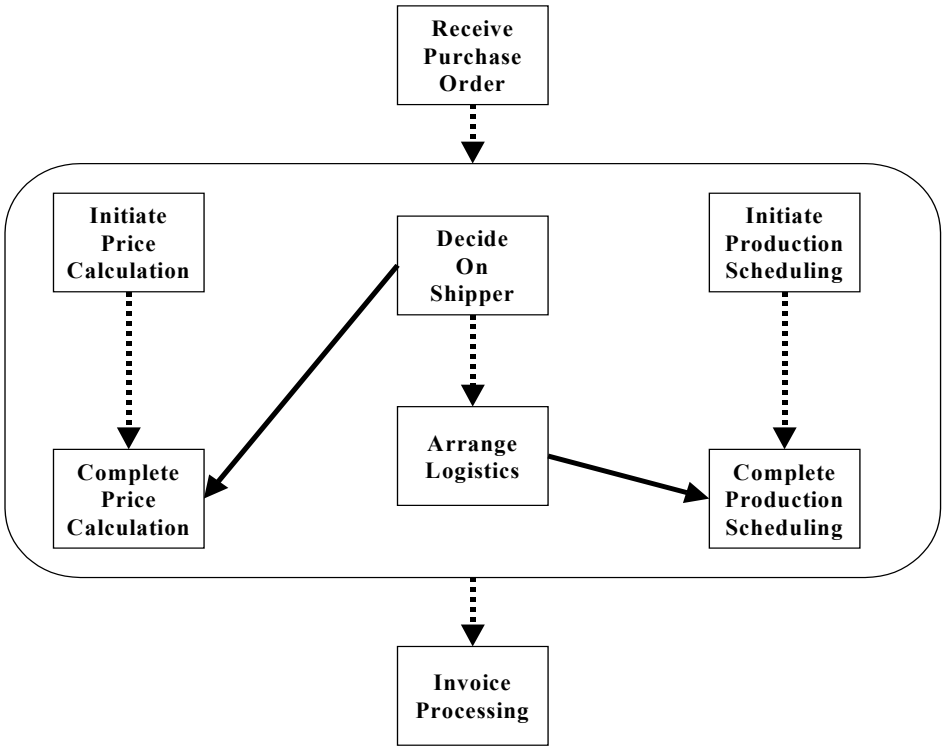

The WSDL portType offered by the service to its customers (purchaseOrderPT) is shown in the following WSDL document. Other WSDL definitions required by the business process are included in the same WSDL document for simplicity; in particular, the portTypes for the Web Services providing price calculation, shipping selection and scheduling, and production scheduling functions are also defined there. Observe that there are no bindings or service elements in the WSDL document. A BPEL4WS process is defined "in the abstract" by referencing only the portTypes of the services involved in the process, and not their possible deployments. Defining business processes in this way allows the reuse of business process definitions over multiple deployments of compatible services.

The service link types included at the bottom of the WSDL document represent the interaction between the purchase order service and each of the parties with which it interacts (see Service Linking, Partners, and Service References). Service link types can be used to represent dependencies between services, regardless of whether a BPEL4WS business process is defined for one or more of those services. Each service link type defines up to two "role" names, and lists the portTypes that each role must support for the interaction to be carried out successfully. In this example, two link types, "purchaseLT" and "schedulingLT", list a single role because, in the corresponding service interactions, one of the parties provides all the invoked operations: The "purchaseLT" service link represents the connection between the process and the requesting customer, where only the purchase order service needs to offers a service operation ("sendPurchaseOrder"); the "schedulingLT" service link represents the interaction between the purchase order service and the scheduling service, in which only operations of the latter are invoked. The two other service link types, "invoiceLT" and "shippingLT", define two roles because both the user of the invoice calculation and the user of the shipping service (the invoice or the shipping

schedule) must provide callback operations to enable asynchronous notifications to be asynchronously sent ("invoiceCallbackPT" and "shippingCallbackPT" portTypes).

```
<definitions targetNamespace="http://manufacturing.org/wsdl/purchase"
      xmlns:sns="http://manufacturing.org/xsd/purchase"
      xmlns:pos="http://manufacturing.org/wsdl/purchase"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:slnk="http://schemas.xmlsoap.org/ws/2003/03/service-link/">

<import namespace="http://manufacturing.org/xsd/purchase"
        location="http://manufacturing.org/xsd/purchase.xsd"/>

<message name="POMessage">
   <part name="customerInfo" type="sns:customerInfo"/>
   <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>
<message name="InvMessage">
   <part name="IVC" type="sns:Invoice"/>
</message>
<message name="orderFaultType">
   <part name="problemInfo" type="xsd:string"/>
</message>
<message name="shippingRequestMessage">
   <part name="customerInfo" type="sns:customerInfo"/>
</message>
<message name="shippingInfoMessage">
   <part name="shippingInfo" type="sns:shippingInfo"/>
</message>
<message name="scheduleMessage">
   <part name="schedule" type="sns:scheduleInfo"/>
</message>

<!-- portTypes supported by the purchase order process -->

<portType name="purchaseOrderPT">
   <operation name="sendPurchaseOrder">
      <input message="pos:POMessage"/>
```

```
      <output message="pos:InvMessage"/>
      <fault name="cannotCompleteOrder"
             message="pos:orderFaultType"/>
   </operation>
</portType>
<portType name="invoiceCallbackPT">
   <operation name="sendInvoice">
      <input message="pos:InvMessage"/>
   </operation>
</portType>
<portType name="shippingCallbackPT">
   <operation name="sendSchedule">
      <input message="pos:scheduleMessage"/>
   </operation>
</portType>

<!-- portType supported by the invoice services -->

<portType name="computePricePT">
   <operation name="initiatePriceCalculation">
      <input message="pos:POMessage"/>
   </operation>
   <operation name="sendShippingPrice">
      <input message="pos:shippingInfoMessage"/>
   </operation>
</portType>

<!-- portType supported by the shipping service -->

<portType name="shippingPT">
   <operation name="requestShipping">
      <input message="pos:shippingRequestMessage"/>
      <output message="pos:shippingInfoMessage"/>
      <fault name="cannotCompleteOrder"
             message="pos:orderFaultType"/>
   </operation>
</portType>
```

```
<!-- portType supported by the production scheduling process -->

<portType name="schedulingPT">
   <operation name="requestProductionScheduling">
      <input message="pos:POMessage"/>
   </operation>
   <operation name="sendShipingSchedule">
      <input message="pos:scheduleMessage"/>
   </operation>
</portType>

<slnk:serviceLinkType name="purchaseLT">
   <slnk:role name="purchaseService">
       <slnk:portType name="pos:purchaseOrderPT"/>
   </slnk:role>
</slnk:serviceLinkType>

<slnk:serviceLinkType name="invoiceLT">
   <slnk:role name="invoiceService">
       <slnk:portType name="pos:computePricePT"/>
   </slnk:role>
   <slnk:role name="invoiceRequester">
       <portType name="pos:invoiceCallbackPT"/>
   </slnk:role>
</slnk:serviceLinkType>

<slnk:serviceLinkType name="shippingLT">
   <slnk:role name="shippingService">
       <slnk:portType name="pos:shippingPT"/>
   </slnk:role>
   <slnk:role name="shippingRequester">
       <portType name="pos:shippingCallbackPT"/>
   </slnk:role>
</slnk:serviceLinkType>

<slnk:serviceLinkType name="schedulingLT">
   <slnk:role name="schedulingService">
       <slnk:portType name="pos:schedulingPT"/>
```

```
   </slnk:role>
</slnk:serviceLinkType>

</definitions>
```

The business process for the order service is defined next. There are four major sections in this process definition:

- The <variables> section defines the data variables used by the process, providing their definitions in terms of WSDL message types. Variables allow processes to maintain state data and process history based on messages exchanged.
- The <partners> section defines the different parties that interact with the business process in the course of processing the order. The four partners shown here correspond to the sender of the order (customer), as well as the providers of price (invoiceProvider), shipment (shippingProvider), and manufacturing scheduling services (schedulingProvider). Each partner is characterized by a service link type and a role name. This information identifies the functionality that must be provided by the business process and by the partner for the relationship to succeed, that is, the portTypes that the purchase order process and the partner need to implement.
- The <faultHandlers> section contains fault handlers defining the activities that must be performed in response to faults resulting from the invocation of the assessment and approval services. In BPEL4WS, all faults, whether internal or resulting from a service invocation, are identified by a qualified name. In particular, each WSDL fault is identified in BPEL4WS by a qualified name formed by the target namespace of the WSDL document in which the relevant portType and fault are defined, and the ncname of the fault. It is important to note, however, that because WSDL 1.1 does not require that fault names be unique within the namespace where the operation is defined, all faults sharing a common name and defined in the same namespace are indistinguishable. In spite of this serious WSDL limitation, BPEL4WS provides a uniform naming model for faults, in the expectation that future versions of WSDL will provide a better fault-naming model.
- The rest of the process definition contains the description of the normal behavior for handling a purchase request. The major elements of this description are explained in the section following the process definition.

```
<process name="purchaseOrderProcess"
         targetNamespace="http://acme.com/ws-bp/purchase"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:lns="http://manufacturing.org/wsdl/purchase">

   <partners>
      <partner name="customer"
               serviceLinkType="lns:purchaseLT"
               myRole="purchaseService"/>
```

```
    <partner name="invoiceProvider"
             serviceLinkType="lns:invoiceLT"
             myRole="invoiceRequester"
             partnerRole="invoiceService"/>
    <partner name="shippingProvider"
             serviceLinkType="lns:shippingLT"
             myRole="shippingRequester"
             partnerRole="shippingService"/>
    <partner name="schedulingProvider"
             serviceLinkType="lns:schedulingLT"
             partnerRole="schedulingService"/>
  </partners>

  <variables>
    <variable name="PO" messageType="lns:POMessage"/>
    <variable name="Invoice"
              messageType="lns:InvMessage"/>
    <variable name="POFault"
              messageType="lns:orderFaultType"/>
    <variable name="shippingRequest"
              messageType="lns:shippingRequestMessage"/>
    <variable name="shippingInfo"
              messageType="lns:shippingInfoMessage"/>
    <variable name="shippingSchedule"
              messageType="lns:scheduleMessage"/>
  </variables>

  <faultHandlers>
    <catch faultName="lns:cannotCompleteOrder"
           faultVariable="POFault">
       <reply   partner="customer"
                portType="lns:purchaseOrderPT"
                operation="sendPurchaseOrder"
                variable="POFault"
                faultName="cannotCompleteOrder"/>
    </catch>
  </faultHandlers>
```

```
<sequence>

   <receive partner="customer"
            portType="lns:purchaseOrderPT"
            operation="sendPurchaseOrder"
            variable="PO">
   </receive>

   <flow>

      <links>
         <link name="ship-to-invoice"/>
         <link name="ship-to-scheduling"/>
      </links>

      <sequence>
         <assign>
            <copy>
               <from variable="PO" part="customerInfo"/>
               <to variable="shippingRequest"
                   part="customerInfo"/>
            </copy>
         </assign>

         <invoke  partner="shippingProvider"
                  portType="lns:shippingPT"
                  operation="requestShipping"
                  inputVariable="shippingRequest"
                  outputVariable="shippingInfo">
            <source linkName="ship-to-invoice"/>
         </invoke>

         <receive partner="shippingProvider"
                  portType="lns:shippingCallbackPT"
                  operation="sendSchedule"
                  variable="shippingSchedule">
            <source linkName="ship-to-scheduling"/>
         </receive>
```

```
    </sequence>

    <sequence>

       <invoke  partner="invoiceProvider"
                portType="lns:computePricePT"
                operation="initiatePriceCalculation"
                inputVariable="PO">
       </invoke>
       <invoke  partner="invoiceProvider"
                portType="lns:computePricePT"
                operation="sendShippingPrice"
                inputVariable="shippingInfo">
          <target linkName="ship-to-invoice"/>
       </invoke>

       <receive partner="invoiceProvider"
                portType="lns:invoiceCallbackPT"
                operation="sendInvoice"
                variable="Invoice"/>

    </sequence>

    <sequence>
       <invoke  partner="schedulingProvider"
                portType="lns:schedulingPT"
                operation="requestProductionScheduling"
                inputVariable="PO">
       </invoke>
       <invoke  partner="schedulingProvider"
                portType="lns:schedulingPT"
                operation="sendShippingSchedule"
                inputVariable="shippingSchedule">
          <target linkName="ship-to-scheduling"/>
       </invoke>
    </sequence>
  </flow>
```

```
      <reply partner="customer"
             portType="lns:purchaseOrderPT"
             operation="sendPurchaseOrder"
             variable="Invoice"/>
   </sequence>

</process>
```

The structure of the main processing section is defined by the outer <sequence> element, which states that the three activities contained inside are performed in order. The customer request is received (<receive> element), then processed (inside a <flow> section that enables concurrent behavior), and a reply message with the final approval status of the request is sent back to the customer (<reply>). Note that the <receive> and <reply> elements are matched respectively to the <input> and <output> messages of the "sendPurchaseOrder" operation invoked by the customer, while the activities performed by the process between these elements represent the actions taken in response to the customer request, from the time the request is received to the time the response is sent back (reply).

The example makes the implicit assumption that the customer request can be processed in a reasonable amount of time, justifying the requirement that the invoker wait for a synchronous response (because this service is offered as a request-response operation). When that assumption does not hold, the interaction with the customer is better modeled as a pair of asynchronous message exchanges. In that case, the "sendPurchaseOrder" operation is a one-way operation and the asynchronous response is sent by invoking a second one-way operation on a customer "callback" interface. In addition to changing the signature of "sendPurchaseOrder" and defining a new portType to represent the customer callback interface, two modifications need to be made in the preceding example to support an asynchronous response to the customer. First, the service link type "purchaseLT" that represents the process-customer connection needs to include a second role ("customer") listing the customer callback portType. Second, the <reply> activity in the process needs to be replaced by an <invoke> on the customer callback operation.

The processing taking place inside the <flow> element consists of three <sequence> blocks running concurrently. The synchronization dependencies between activities in the three concurrent sequences are expressed by using "links" to connect them. The links are defined inside the flow and are used to connect a source activity to a target activity. (Note that each activity declares itself as the source or target of a link by using the nested <source> and <target> elements.) In the absence of links, the activities nested directly inside a flow proceed concurrently. In the example, however, the presence of two links introduces control dependencies between the activities performed inside each sequence. For example, while the price calculation can be started immediately after the request is received, shipping price can only be added to the invoice after the shipper information has been obtained; this dependency is represented by the link (named "ship-to-invoice") that connects the first call on the shipping provider ("requestShipping") with sending shipping information to the price calculation service ("sendShippingPrice"). Likewise, shipping scheduling information can only be sent to the manufacturing scheduling service after it has been received from the shipper service; thus the need for the second link ("ship-to-scheduling").