WS-BPEL Issue 82.2 – Proposal

Last modified: November 02, 2005 – 10.30 PDT

The 82.2 subissue was raised to address the definition of a process template profile for abstract BPEL. After passing the conceptual framework for this process template profile, we define below the changes to the specification to resolve Issue 82.2.

Note: This proposed language is contingent on incorporation of and consolidation with the proposed spec language for Issue 82. The language below assumes that Issue 82 language have been incorporated into the spec. The language and terminologies used in Issue 82.2 and 82.3 will be fine-tuned to be consistent.

Changes to the Specification:

A. **Section 1, Introduction**
   Move paragraph 9 before paragraph 8 and modify as follows:
   <<The basic concepts of WS-BPEL can be applied in one of two ways. It is possible to use WS-BPEL to define an executable business process.>> The logic and state….

   After the new paragraph 9, add the following:
   <<Abstract processes serve a descriptive role, with more than one possible use case. One such use case might be to describe the visible behavior of some or all of the services offered by an executable process. Another use case would be to define a process template that embodies domain-specific best practices. Such a process template would capture essential process logic in a manner compatible with a design-time representation, while excluding execution details to be completed when mapping to an executable process.
   Regardless of the specific use case and purpose, all abstract processes share a common syntactic base. They have different requirements for the level of opacity and restrictions on which parts of a process definition may be omitted or hidden. Tailored uses of abstract processes have different effects on the consistency constraints – or the constraints required for a valid executable process that are not enforceable by the XML Schema – and on the semantics of that process.
   A common base specifies the features that define the syntactic universe of abstract processes. Given this common base, a usage profile provides the necessary specializations and semantics based on executable WS-BPEL for a particular usage area of an abstract process. >>

B. **Section 6.2, The Structure of a Business Process** *(Assuming the following will be included in the text changes when applying Issue 82 to specification)*

   Remove: abstractprocess=”true|false” and add in its place:

abstractProcessProfile=”anyURI”?

Add to the bullets in paragraph 1, “The top-level attributes are as follows”:

- abstractProcessProfile: This attribute provides the URI that identifies the profile of an abstract process. It is mandatory for abstract processes.

Remove from bullets in paragraph 4, “The syntax of each of these elements….”:

- Although <exit> is permitted as an interpretation of the token activity, it is only available in executable processes and as such is defined in the section on **Extensions for Executable Processes**.

C. Section 15, Extensions:
Add the following subsection:
<<
15.4 Abstract Process Template Profile
A WS-BPEL abstract process based on the template profile may be created that is compatible with a design-time representation, by taking various inputs from technical analysts or other modeling languages. This template process may be provided at a later stage to process developers to complete execution details - for example, adding conditions and defining endpoints to template process for an executable completion.

ProfileURI:
urn:oasis:names:tc:wsbpel:2.0:abstract:simple-template/YYYY/MM

Base Language subset:
This profile restricts the common base in the following manner:

- Explicit opaque tokens – opaque activity, opaque attributes and opaque expression – MUST be used in order to denote where executable WS-BPEL constructs are added to produce an executable completion in all cases other than those listed under “Adding executable constructs without explicit opacity”.

- Implicit omission of opaque tokens MUST NOT be used in the Template Profile. For example, variables and properties used in an inbound message operation that correlates process instances to messages MUST be explicitly opaque, if not specifically defined.

- All start activities MUST be defined in a process of this Template Profile. This implies that NO new start activity is allowed to be added during executable completion (that is a message inbound activity annotated with a createInstance="yes" attribute).

(Note:
- As with executable processes, an <exit> activity MAY be used in the Template Profile.
- Explicit opaque tokens MAY be used anywhere as the common base of Abstract Process allows.)

Adding executable constructs without explicit opacity:
For this Template Profile, in these following exceptional cases, executable constructs MAY be added to the process definition during Execution Completion without any explicit opacity in the abstract process:

- Activities: For <assign> activity, the validate attribute MAY be added within an <assign> activity in an abstract process.
- Message Correlation: One or more <correlation> elements MAY be added to a message activity and <onEvent>, where no <correlation> or <correlations>is used.
- Process/Scope Declaration (including scope-equivalent constructs, such as a <invoke> macro with a compensation handler and fault handler):
    - **a.** Adding new data and resource declarations at a scope or top-level process, such as partnerLinks, variables and/or correlationSets at a scope or top-level process.
    - **b.** Adding a fault handler declaration at a scope or top-level process.
    - **c.** Adding terminationHandler declaration at a scope.
    - **d.** Adding an event handler declaration at a scope or top-level process.
- Extensions
    - **a.** Adding new general extension elements and attributes.

Extensions and document usage:
Extension attributes and elements are generally allowed in both Abstract and Executable processes; information could be added in extensions, or by natural language documentation in the business process, to signal the intention of the designer or extra semantics where needed. This is used to cover cases where the opacity causes ambiguity in other related parts of the process, such as those mentioned in Section 15.

Examples in the context of this abstract process template profile include:

- A unique identifier attribute may be added by a designer tool to uniquely identify a WS-BPEL fragment that spans the lifetime of a business process in abstract and execution completion stages - as such, the activity that replaces the "opaqueActivity" retains that unique identifier.
- WS-BPEL template designer may add natural language as documentation or extension constructs, to denote extra template information. An example of this is as follows:

```
<process name="templateExample1-HomeAppraisal" xmlns="..."
```

```
targetNamespace="http://example-bpel.org/template-example-1"
xmlns:tns="http://example-bpel.org/template-example-1"
suppressJoinFailure="yes"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ext="http://some.org/bpel/some/extension">

   <partnerLinks>
         <!-- example explanatory note: none of the 3
referenced partnerLinks have been declared -->
         <partnerLink name="homeInfoVerifier"
partnerLinkType="##opaque" myRole="##opaque"
partnerRole="##opaque">
               <documentation>
                     We have not confirmed our home
information verification partner yet.
               </documentation>
         </partnerLink>
   </partnerLinks>
   <variables>
         <variable name="commonRequestVar"
element="##opaque"/>
   </variables>

   <sequence>
         <opaqueActivity createInstance="yes">
               <documentation>
               Pick an appraisal request from one of 3
customer referral channels.
            </documentation>
         </opaqueActivity>
         <assign>
               <documentation>
                  Transform one of these 3 appraisal requesst
into our own format.
               </documentation>
               <from opaque="yes"/>
               <to variable="commonRequestVar"/>
         </assign>
         <scope>
               <faultHandlers>
                     <!-- example explanatory note: One can
add a new <catch> handler for a fault from the
"homeInfoVerifier" partnerLink of unspecified portType yet  --
>
                     <catchAll>
                           <exit />
                     </catchAll>
               </faultHandlers>
               <sequence>
                     <opaqueActivity>
                           <documentation>
                                 Extract customer and housing
info from our appraisal request into a message understood by
our home info verification partner.
                            </documentation>
                     </opaqueActivity>
                     <invoke partnerLink="homeInfoVerifier"
```

```
operation="##opaque" inputVariable="##opaque"
ext:uniqueUserFriendlyName="request verification"/>
                     <receive partnerLink="homeInfoVerifier"
operation="##opaque" variable="##opaque"
ext:uniqueUserFriendlyName="receive verification result"/>
                     <reply partnerLink="homeInfoVerifier"
operation="##opaque" variable="##opaque"
ext:uniqueUserFriendlyName="confirm receipt of verification
result">
                           <documentation>
                                  This step confirms whether
we have received the verification result. It is intended to
match the "receive verification result" step.
                            </documentation>
                     </reply>
               </sequence>
         </scope>
         <opaqueActivity>
               <documentation>
                     Relay the appraisal request and home info
verification to an appraiser, who is responsible for on-site
inspection. The appraiser may request further verification
info from the partner through this business process. We will
also will receive the results of the appraisal from this step.
               </documentation>
               <!-- example explanatory note: An unspecified
referral channel may trigger more than one unexpected fault in
this process. -->
         </opaqueActivity>
         <opaqueActivity>
               <documentation>
                     Send the appraisal result back to the
corresponding referral channel.
               </documentation>
               <!-- example explanatory note: An unspecified
referral channel may trigger more than one unexpected fault in
this process. -->
         </opaqueActivity>
   </sequence>
</process>
```
>>

D. Section 16, Examples

Add the following Section 16.2 example as follows:

<<

16.2 Ordering Service

This example expands on the above shipping service above to illustrate the use of a WS-BPEL abstract process using the template profile. This abstract process describes a set of services to request, track, and confirm orders and their shipments, invoicing and payment. The ordering service receives orders from an order processor, sends a shipping request to the shipping service, and acknowledges shipment, pickup, invoicing and payment as each are performed.

16.2.1 Service Description

The context for the ordering service is a two-party interaction between a consumer and the service. Those that have been concretely defined are modeled in the following `partnerLinkType` definition:

```
    <plnk:partnerLinkType name="OrderingServiceLinkType">
        <plnk:role name="OrderingService">
            <plnk:portType name="tns:OrderingPortType"/>
        </plnk:role>
    </plnk:partnerLinkType>

    <plnk:partnerLinkType name="OrderingResponseLinkType">
        <plnk:role name="OrderingServiceResponse">
            <plnk:portType name="tns:OrderingResponsePortType"/>
        </plnk:role>
    </plnk:partnerLinkType>

    <plnk:partnerLinkType name="ShipperLinkType">
        <plnk:role name="ShippingService">
            <plnk:portType name="tns:Shipping"/>
        </plnk:role>
    </plnk:partnerLinkType>

    <plnk:partnerLinkType name="CompletionConfirmationLinkType">
        <plnk:role name="OrderingServiceConfirmation">
            <plnk:portType
name="tns:OrderingConfirmationPortType"/>
        </plnk:role>
    </plnk:partnerLinkType>
```

The corresponding message and portType definitions are as follows:

```
   <message name="OrderMessageType">
        <part name="OrderMessagePart"
element="tns:OrderMessage"/>
    </message>
    <message name="OrderAckMessageType">
        <part name="OrderAckMessagePart"
element="tns:OrderAckMessage"/>
    </message>
    <message name="ShipRequestMessageType">
        <part name="ShipRequestMessagePart"
element="tns:ShipRequestMessage"/>
    </message>
    <message name="ShipNoticeMessageType">
        <part name="ShipNoticeMessagePart"
element="tns:ShipNoticeMessage"/>
    </message>
    <message name="PaymentConfirmationMessageType">
        <part name="PaymentConfirmationMessagePart"
element="tns:PaymentConfirmationMessage"/>
    </message>

    <portType name="OrderingPortType">
        <operation name="PlaceOrder">
            <input  message="tns:OrderMessageType" />
        </operation>
    </portType>

    <portType name="OrderingResponsePortType">
        <operation name="GetOrderAck">
            <output message="tns:OrderAckMessageType"/>
        </operation>
    </portType>

    <portType name="OrderingConfirmationPortType">
        <operation name="GetOrderConfirmation">
            <output message="tns:OrderAckMessageType"/>
        </operation>
    </portType>

    <portType name="ShippingService">
        <operation name="shippingRequest">
            <input message="tns:ShippingRequestMessageType" />
        </operation>
    </portType>

    <portType name="ShippingServiceCustomer">
        <operation name="shippingNotice">
            <output message="tns:ShippingNoticeMessageType"/>
        </operation>
    </portType>
```

Although there are more interactions between consumer and service, not all have been modelled in this scenario, and as such, interactions as-of-yet unmodelled are by necessity opaque.

16.2.2 Message Properties

The properties relevant to the service behavior are:

- The order ID, which is used to correlate the order placement with the shipping request, shipping notice, invoice confirmation, pickup confirmation and final order confirmation. In this scenario, only the shipping request, shipping notice and final confirmation have been defined. (`orderID`)
- Whether the order has been shipped or not. (`shipCompleted`)
- The shipment history. (`shipHistory`)

In this scenario, only the order ID has been defined. Below is the definition for the orderID property and its aliases:

```
<bpws:property name="orderID" type="xs:string"/>

<bpws:propertyAlias propertyName="tns:orderID"
      messageType="tns:OrderMessageType"
      part="OrderMessagePart"
      query="OrderHeader/orderID"/>
<bpws:propertyAlias propertyName="tns:orderID"
      messageType="tns:ShippingRequestMessageType"
```

```
            part="ShippingRequestMessagePart"
            query="ShippingRequestMessage/OrderHeader"/>
      <bpws:propertyAlias propertyName="tns:orderID"
            messageType="tns:ShippingNoticeMessageType"
            part="ShippingNoticeMessagePart"
            query="/ShippingNoticeMessage/orderID"/>
      <bpws:propertyAlias propertyName="tns:orderID"
            messageType="tns:PaymentConfirmationMessageType"
            part="PaymentConfirmationMessagePart"
            query="PaymentConfirmationMessage/orderID"/>
```

Although there are more messages between consumer and service, not all have been modelled in this scenario, and as such, messages as-of-yet unmodelled are by necessity opaque.

16.2.3 Process

Below is the abstract process definition per the template profile. The rough outline is as follows:

```
receive placeOrder
send shipOrder
if
      condition shipCompleted
           send orderNotice (indicating shipCompleted)
       else
           send orderNotice (indicating !shipCompleted)

receive pickupNotification
update shipHistory

receive invoice
send invoiceResponse

receive paymentConfirmation
send orderConfirmation
```

Below is the complete version:

```
<process name="OrderingServiceProcess"
targetNamespace="http://example-bpel.org/template-example-2"
xmlns:tns="http://example-bpel.org/template-example-2"
suppressJoinFailure="yes"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ext="http://some.org/bpel/some/extension">
      <partnerLinks>
            <partnerLink name="ordering"
partnerLinkType="tns:OrderingServiceLinkType"
myRole="OrderingService"/>
            <partnerLink name="orderingResponse"
partnerLinkType="tns:OrderingResponseLinkType"
partnerRole="OrderingServiceResponse"/>
```

```
            <partnerLink name="shipper"
partnerLinkType="tns:ShippingLinkType"
partnerRole="ShippingService"/>
            <partnerLink name="shipperResponse"
partnerLinkType="##opaque" myRole="##opaque"/>
            <partnerLink name="shippingRequestor"
partnerLinkType="##opaque" myRole="##opaque"/>
            <partnerLink name="invoiceProcessor"
partnerLinkType="##opaque" myRole="##opaque"/>
            <partnerLink name="invoiceResponse"
partnerLinkType="##opaque" partnerRole="##opaque"/>
            <partnerLink name="orderingConfirmation"
partnerLinkType="tns:CompletionConfirmationLinkType"
partnerRole="OrderingServiceConfirmation"/>
      </partnerLinks>

      <variables>
            <!-- Reference to the message passed as input during
initiation -->
            <variable name="order"
messageType="tns:OrderMessageType"/>
            <variable name="orderAckMsg"
messageType="tns:OrderAckMessageType"/>
            <variable name="orderShippedMsg" element="##opaque"/>
            <variable name="shippingRequestMsg"
element="##opaque"/>
            <variable name="shippingNoticeMsg"
element="##opaque"/>
            <variable name="pickupNotificationMsg"
element="##opaque"/>
            <variable name="shipStatusMsg" element="##opaque"/>
            <variable name="shipHistoryMsg"
messageType="tns:ShippingHistoryMessageType"/>
            <variable name="invoiceMsg" element="##opaque"/>
            <variable name="invoiceAckMsg"
messageType="tns:InvoiceAckMessageType"/>
            <variable name="paymentConfirmationMsg"
messageType="tns:PaymentConfirmationMessageType"/>
      </variables>

      <sequence>
            <receive partnerLink="ordering"
portType="tns:OrderingPortType" operation="placeOrder"
variable="order" createInstance="yes">
                  <correlations> <correlation
set="OrderCorrelationSet" initiate="yes"/> </correlations>
            </receive>
            <assign>
                  <copy >
                        <from variable="order"
part="OrderMessagePart"
query="/OrderMessage/ShippingInfo"></from>
                        <to variable="shippingRequestMsg"
part="ShipRequestMessagePart" query="/ShipRequest/ShippingInfo"/>
                  </copy>
            </assign>
            <invoke partnerLink="shipper"
portType="tns:ShippingService" operation="shippingRequest"
inputVariable="shippingRequestMsg"
ext:uniqueUserFriendlyName="send shipping request to shipper"/>
            <receive partnerLink="shipperResponse"
portType="tns:ShippingServiceCustomer" operation="shippingNotice"
variable="shippingNoticeMsg" ext:uniqueUserFriendlyName="receive
response from shipper"/>
            <switch>
                  <case condition="##opaque">
                  <!-- the first case would package the order
acknowledgement for a completed shipment -->
                        <assign>
                              <copy>
                                    <from
expression="'##opaque'"/>
                                    <to variable="orderAckMsg"
part="OrderAckMessagePart" query="/OrderAckMessage/Ack"/>
                              </copy>
                        </assign>
                  </case>
            <otherwise>
                  <!-- the second case would package the order
acknowledgement for an uncompleted shipment -->
                  <assign>
                        <copy>
                              <from expression="##opaque'"/>
                              <to variable="orderAckMsg"
part="OrderAckMessagePart" query="/OrderAckMessage/Ack"/>
                        </copy>
                  </assign>
            </otherwise>
            </switch>
            <reply partnerLink="orderingResponse"
portType="tns:OrderingResponsePortType" operation="getOrderAck"
variable="orderAck"/>
            <receive partnerLink="shippingRequestor"
operation="##opaque" variable="##opaque"
ext:uniqueUserFriendlyName="receive the pickup notification">
            <assign>
                  <copy>
                        <from expression="##opaque"/>
                        <to variable="shipHistoryMsg"
part="ShippingHistoryPart" query="/ShippingHistory/Event"/>
                  </copy>
            </assign>
            <opaqueActivity>
                  <documentation>
                        If we receive notice that the ship has
completed, update our ship history accordingly
                  </documentation>
            </opaqueActivity>
            <receive partnerLink="invoiceProcessor"
operation="##opaque" variable="##opaque"
ext:uniqueUserFriendlyName="receive invoice for processing"/>
            <assign>
                  <copy>
```

```
                        <from expression="##opaque"/>
                        <to variable="invoiceAckMsg"
part="InvoiceAckMessagePart"/>
                  </copy>
            </assign>
            <reply partnerLink="invoiceResponse"
operation="##opaque" variable="##opaque"
ext:uniqueUserFriendlyName="send response for the invoice"/>
            <receive partnerLink="shippingRequestor"
operation="##opaque" variable="##opaque"
ext:uniqueUserFriendlyName="receive payment confirmation"/>
            <assign>
                  <copy>
                        <from expression="'##opaque'"/>
                        <to variable="orderShippedMsg"
part="OrderAckMessagePart" query="/OrderAckMessage/Ack"/>
                  </copy>
            </assign>
            <reply partnerLink="orderingConfirmation"
partnerLinkType="tns:FinalConfirmLinkType"
portType="tns:OrderingConfirmationPortType"
operation="getOrderConfirmation" variable="orderShippedMsg"/>
      <sequence>
<process>
>>
```

Further explanation regarding "Adding executable constructs without explicit opacity":

With respect to the list of constructs that are listed in this subsection of 15.4 in the proposal above, at present we do not have opaque construct for most of those cases; rather, we only have opaque attribute and opaque activity. Realistically, the template author cannot list all of these resources when designing their template.

The only cases where we allow executable constructs without explicit opacity, and where such opaque tokens are available, are as follows:

- Valid attribute at <assign>
- Extension attributes

These attributes are optional by nature; hence, explicit opaque attributes are not always required. However, the absence of these attributes imply a certain "default" effect on the process.

The net effect of allowing the addition of these attributes during execution completion WITHOUT associated explicit opaque tokens is to allow the process developer (as opposed to the template author) to modify/tune the QoS related attributes from their default parameter values. As we all know, template authors do not think of business processes in terms of QoS, while process developers do. This applies just as aptly to extension attributes.