WS-BPEL Issue 157 – Proposal

Last modified: August 10, 2005 - 5pm PDT

Below is the proposed resolution for Issue 157 as developed in our small-group meetings in June-July 2005:

(A) Update Section 9.3, "Assignment", as follows:

- Update the second bullet item in paragraph 9 ("a sequence of one or more character information items….") to read (changes denoted by «»):
  - a sequence of «zero» or more «character information items»: this is mapped to a Text Node «or a String» in the «XPath 1.0» data model

- Update the <copy> syntax to read (changes denoted by «»):

```
<copy «keepSrcElementName="yes|no"?»>
  from-spec
  to-spec
</copy>
```

- Insert the following before paragraph 16 ("An optional `validate` attribute can be used with the assign activity….") as follows:
  "An optional `keepSrcELementName` attribute of the <copy> construct can be used to specify whether the element name of the destination (as selected by the to-spec) will be replaced by the element name of the source (as selected by the from-spec) during the copy operation. For details, please refer to Section 9.3.1, "Replacement Logic of Copy Operations"."

(B) Insert a new Section 9.3.1, "Selection Result of Copy Operations", before the existing 9.3.1, "Type Compatibility in Assignment", as follows:

- ### Selection Result of Copy Operations

  There are 11 different types of information items in the XML Infoset Information model. Most of these are not relevant in the context of XML data manipulation as performed by <copy> operation – examples include Processing Instruction Information Item, Comment Information Item, and Document Type Declaration Information Item.

  The selection result of the from-spec or to-spec used within a <copy> operation MUST be one of the following three Information Items: Element Information Item (EII), Attribute Information Item (AII), or Text Information Item (TII). Note that EII and AII are defined in [Infoset], while TII is defined to bridge the gap

between the XML Infoset Model and other common XML data models, such as XPath 1.0. TII is defined as follows:

Text Information Item (TII): This is an Information Item of which an attribute points to a sequence of zero or more Character Information Items, according to the document order. When mapped to the XPath 1.0 model, it is a generalization of String-Value (which has zero or more characters) and Text Node (which has one or more characters). An RValue of a TII MAY be mapped to a Text node, a String/Boolean/Number object in XPath 1.0, while an LValue of a TII MUST be mapped to a Text node.

If the selection result of a from-spec or a to-spec belongs to Information Items other than EII, AII or TII, a bpws:selectionFailure fault MUST be thrown. Note that if any of the unsupported Information Items are contained in the selection result, they MUST be preserved; the only restriction is that they MUST NOT be directly selected by the from-spec or the to-spec as the top-level item.

In WS-BPEL, the <copy> operation is essentially a one-to-one replacement operation. This implies that both the from-spec and to-spec MUST select exactly one information item each, which includes the case of one TII. Note that this restriction indicates that literal values (the literal variant of from-spec) MUST only contain either a TII, or a single EII, as its top-level value.

When using a partnerLink-based from-spec and to-spec, such as:

```
    <from partnerLink="partnerLinkX"
endpointReference="myRole|partnerRole" />
    <to partnerLink="partnerLinkY" />
```

with another non-partnerLink-based from-spec and to-spec in a <copy> operation, these should be treated as if they produce an LValue and RValue of an EII of which [local name] is “service-ref” and [namespace name] is the WS-BPEL namespace.

(C) Update Section 9.3.2, “Type Compatibility in Assignment”, as follows:

- Update the section title to “**Type Compatibility in Copy Operations**”

- Update the first paragraph of the section and first two bullet items following to read (changes denoted by «»):
  “For «a copy operation» to be valid, the data referred to by the from and to specifications MUST be of compatible types. The following points make this precise:
  - The «selection result of the» from-spec is a variable of a WSDL message type, and the «selection result of the» to-spec is a variable of a WSDL message

type. In this case, both variables MUST be of the same message type, where two message types are said to be equal if their qualified names are the same.
    - The «selection result of the» from-spec is a variable of a WSDL message type, and the «selection result of the» is not, or vice versa. This is not legal because parts of variables, selections of variable parts, or endpoint references cannot be assigned to/from variables of WSDL message types directly."

- Update the third bullet item to read (changes denoted by «»):
    - In all other cases, «if the selection results of the source (from-spec) and destination (to-spec) are XML Infoset Information or XML data model items, and the XML Schema types of these are known», then the source value MUST possess «the type» associated with the destination. Note that this does not require the types associated with the source and destination to be the same. In particular, the source type MAY be a subtype of the destination type. «The required XML Schema type checking can be determined by static analysis and/or evaluated at runtime. A BPEL processor MAY perform static analysis of the expression/query language to validate compliance with this compatibility requirement, and reject a process definition if the requirement is violated. When a BPEL processor adopts an XML Schema type aware data model, it MAY perform the same analysis at runtime, where, on encountering a violation of the compatibility requirement, it MUST throw a bpws:mismatchedAssignmentFailure fault. Note that when the default XPath 1.0 expression/query language binding is used, XML Schema runtime type-compatibility checking MUST NOT be performed, as the XPath 1.0 data model is not XML Schema type aware.»

- Remove the last paragraph of the section.

Note that this will address Issue 51 (http://www.choreology.com/external/WS_BPEL_issues_list.html#Issue51) as well as Issue 157. In addition, Yaron has indicated that we may want to develop a standard approach to disable schema-type static analysis, but this is part of the discussion surrounding Issue 9 (http://www.choreology.com/external/WS_BPEL_issues_list.html#Issue9).

(D) Insert a new Section 9.4.1, "Replacement Logic of Copy Operations", before the existing Section 9.4.1, "Type Compatibility of Assignment", as follows:

- ### Replacement Logic of Copy Operations

  #### Replacement Logic for WSDL Message Variables

  When the from-spec and to-spec of a <copy> operation both select WSDL

message variables, the following replacement logic MUST be executed:

All existing message parts in the destination WSDL message variable (referenced by the to-spec) will be removed, and all existing message parts in the source WSDL message variable (referenced by the from-spec) will be copied and added to the destination WSDL message variable.

- **Replacement Table for XML Data Item:**

When the from-spec (Source) and to-spec (Destination) select one of three Information Items types, a conforming WS-BPEL processor MUST use the replacement rules for the combinations of Source and Destination Information Item types for <copy> operation, as defined in the following Replacement Logic Table:

Source\Destination	EII	AII	TII
EII	RE	RC	RC
AII	RC	RC	RC
TII	RC	RC	RC

Replacement Logic Table

Definitions

- RE (Replace-Element-properties): Replace the element at the destination with a copy of the entire element at the source, including [children] and [attribute] properties. An OPTIONAL `keepSrcElementName` attribute is provided to further refine the behavior:
  - The default value of the keepSrcElementName attribute is “no”, in which case the name (i.e. [namespace name] and [local name] properties) of the original destination element is used as the name of the resulting source element.
  - When the keepSrcElementName attribute is set to “yes”, the source element name is used as the name of the resulting destination element
  - When the keepSrcElementName attribute is explicitly set, the selection results of the from-spec and to-spec MUST be elements. A BPEL processor MAY enforce this checking through static analysis of the expression/query language. If a violation is detected during runtime, a bpws:selectionFailure fault MUST be thrown.
- RC (Replace-Content):
  - To obtain the source content:
    - The source (from-spec) MUST yield one and only one Information Item. Otherwise, a selectionFailure fault MUST be thrown.
    - Once the Information Item is yielded from the source, a TII will be computed based on the source Information Item as the

source content. The XPath "string()" function will be applied to the Information Item to obtain its string-value as the source content, if the default XPath 1.0 binding is used in the from-spec. If another expression language is used, an XPath function other than "string()" MAY be applied to the node to obtain the source content.
        - If the source is an EII with an xsi:nil="true", a selectionFailure fault MUST be thrown (where this check is performed during EII-to-AII or EII-to-TII copy).,
    - To replace the content:
        - If the destination is an EII, remove all [children] properties (if any) and add the source content TII as the child of the EII.
        - If the destination is an AII, replace the value of AII with the TII from the source. The value MUST be normalized, in accordance with the XML 1.0 Recommendation (section 3.3.3 Attribute Value Normalization: http://www.w3.org/TR/1998/REC-xml-19980210#AVNormalize).
        - If the destination is a TII, replace the TII in the destination with the TII from the source.

Note that:

- Attribute values are not text nodes in XPath 1.0. Attribute nodes have a string value that corresponds to the XML normalized attribute value, which is a TII.
- Information Items referenced by the to-spec MUST be an LValue. In the XPath 1.0 data model, a TII LValue MUST be a Text Node.

Using <copy> to initialize variables

When the destination selected by the to-spec in <copy> is un-initialized, the destination variable or message part MUST first be initialized before executing the above replacement logic. The initialization details are as follows:

- For complex type and simple type variables, a skeleton structure, composed of a DII and an anonymous EII (Document Element), will be created as an integral part of the initialization of the <assign>/<copy> operation. Once this skeleton structure is created, the above "replacement" logic can be reused.
- For element based variables, a skeleton structure, composed of a DII and an EII (Document Element) with the name matching the element name used in variable declaration, will be created as an integral part of the initialization of the <assign>/<copy> operation. Once this skeleton structure is created, the above "replacement" logic can be reused.
- For an uninitialized message part, the above two blocks of logic are reused, as a message part is either of simple type, complex type, or an element.

Handling Non- XML-Infoset Data Objects in <copy>
Simple type variables and values MAY be allowed to manifest as non-XML-Infoset data objects, such as boolean, string, or float, as defined in XPath 1.0. Some expressions may yield such a non-XML-Infoset data object, for example:

<from> number($order/amt) * 0.8 </from>

To consistently apply the above replacement logic, such non-XML-Infoset data are handled as Text Information Items (TII). This logic is achieved through "to-string" data conversion, as TII resembles a string object. More specifically, when the XPath 1.0 data model is used in WS-BPEL, "string(object)" (http://www.w3.org/TR/1999/REC-xpath-19991116#function-string) coercion MUST be used to convert boolean or number objects to String/TII.

Note that this conversion is used to describe the expected result of <copy>. A WS-BPEL processor MAY skip the actual conversion for optimization if the result of <copy> remains the same, which would render the conversion redundant.

XML Namespace Preservation
In the <copy> operation, the [in-scope namespaces] properties (similar to other XML Infoset Item properties) from the source MUST be preserved in the result at the destination. For example, when variables are serialized into XML text, a WS-BPEL processor will make use of a namespace-aware XML infrastructure, which maintains the XML Namespace consistency in the XML text, where in such a case the infrastructure adds XML Namespace declarations or renames prefixes used in XML Namespaces.

Examples illustrating the replacement logic of copy operations: (pending editor group's decision on incorporating examples in the specification text)

- *EII-to-EII*

XML Schema Context

```
-------------------------------
<xs:element name="poHeader">
    <xs:complexType>
      <xs:sequence>
        <xs:choice>
          <xs:element name="shippingAddr"
type="tns:AddressType"/>
          <xs:element name="USshippingAddr"
type="tns:USAddressType"/>
        </xs:choice>
        <xs:element name="billingAddr" type="tns:AddressType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
------------------------------
"tns:USAddressType" is a type extended from "tns:AddressType".
```

- Example 1

```
   ------------------------------
   <assign>
     <copy>
        <from>$poHeaderVar1/tns:shippingAddr</from>
        <to>$poHeaderVar2/tns:billingAddr</to>
     </copy>
   </assign>
   ------------------------------

   The above <copy> will replace the attributes and elements
   of the billing address in "poHeaderVar2" with those of
   shipping address in "poHeaderVar1".

   poHeaderVar1
   ------------------
   <tns:poHeader>
     ...
     <tns:shippingAddr verified="true">
          <tns:street>123 Main Street</tns:street>
          <tns:city>SomeWhere City</tns:city>
          <tns:country>UK</tns:state>
     </tns:shippingAddr>
     ...
   </tns:poHeader>
   ------------------

   poHeaderVar2: (before the copy)
   ------------------
   <tns:poHeader>
     ...
     <tns:billingAddr pobox="true" />
     ...
   </tns:poHeader>
   ------------------

   poHeaderVar2: (after the copy)
   ------------------
   <tns:poHeader>
     ...
     <tns:billingAddr verified="true">
          <tns:street>123 Main Street</tns:street>
          <tns:city>SomeWhere City</tns:city>
          <tns:country>UK</tns:state>
     </tns:billingAddr>
     ...
   </tns:poHeader>
   ------------------
```

- Example 2

```
   ------------------------------
   <assign>
     <copy keepSrcElementName="yes">
```

```
      <from>$poHeaderVar3/tns:USshippingAddr</from>
      <to>$poHeaderVar2/tns:shippingAddr</to>
    </copy>
  </assign>
  -------------------------------


  poHeaderVar3
  ------------------
  <tns:poHeader>
    ...
    <tns:USshippingAddr verified="true">
         <tns:street>123 Main Street</tns:street>
         <tns:city>SomeWhere City</tns:city>
         <tns:country>USA</tns:state>
         <tns:zipcode>98765</tns:zipcode>
    </tns:USshippingAddr>
    ...
  </tns:poHeader>
  ------------------

  poHeaderVar2: (before the copy)
  ------------------
  <tns:poHeader>
    ...
    <tns:shippingAddr pobox="true" />
    ...
  </tns:poHeader>
  ------------------

  poHeaderVar2: (after the copy)
  ------------------
  <tns:poHeader>
    ...
    <tns:USshippingAddr verified="true">
         <tns:street>123 Main Street</tns:street>
         <tns:city>SomeWhere City</tns:city>
         <tns:country>USA</tns:state>
         <tns:zipcode>98765</tns:zipcode>
    </tns:USshippingAddr>
    ...
  </tns:poHeader>
  ------------------
```

- *EII-to-AII*

  XML Data Context

```
creditApprovalVar:
-------------------------------
<tns:creditApplication appId="123-456">
  <tns:approvedLimit code="AXR">4500</tns:approvedLimit>
</tns:creditApplication>
-------------------------------
```

- Example 1

```
------------------------------
<assign>
  <copy>
     <from>$creditApprovalVar/tns:approvedLimit</from>
     <to>$approvalNotice2Var/@amt</to>
  </copy>
</assign>
------------------------------

approvalNotice2Var: (before <copy>)
------------------------------
<tns2:approvalNotice amt="" />
------------------------------


approvalNotice2Var: (after <copy>)
------------------------------
<tns2:approvalNotice amt="4500" />
------------------------------
```

- *EII-to-TII*

  XML Data Context

```
creditApprovalVar:
------------------------------
<tns:creditApplication appId="123-456">
  <tns:approvedLimit code="AXR">4500</tns:approvedLimit>
</tns:creditApplication>
------------------------------
```

  - Example 1

```
------------------------------
<assign>
  <copy>
     <from>$creditApprovalVar/tns:approvedLimit</from>
     <to>$approvalNotice3Var/text()</to>
  </copy>
</assign>
------------------------------

approvalNotice3Var: (before <copy>)
------------------------------
<tns3:approvalNotice>0</tns3:approvalNotice>
------------------------------

approvalNotice3Var: (after <copy>)
------------------------------
<tns3:approvalNotice>4500</tns3:approvalNotice>
------------------------------
```

- Example 2

```
------------------------------
<assign>
  <copy>
     <from>$creditApprovalVar/tns:approvedLimit</from>
     <to>$approvalNotice4Var/text()</to>
  </copy>
</assign>
------------------------------

approvalNotice4Var: (before <copy>)
------------------------------
<tns4:approvalNotice></tns4:approvalNotice>
------------------------------

Since there is no text node under"tns4:approvalNotice",
selectionFailure fault will be thrown. No replacment logic
will be executed.
```

- Example 3: EII-to-EII for direct comparison to EII-to-TII

```
------------------------------
<assign>
  <copy>
     <from>$creditApprovalVar/tns:approvedLimit</from>
     <to>$approvalNotice4Var</to>
  </copy>
</assign>
------------------------------

approvalNotice4Var: (before <copy>)
------------------------------
<tns4:approvalNotice></tns4:approvalNotice>
------------------------------

approvalNotice4Var: (after an EII-to-EII <copy>)
------------------------------
<tns4:approvalNotice code="AXR">4500</tns4:approvalNotice>
------------------------------
```

- *AII-to-AII*

  XML Data Context

```
orderDetailVar:
------------------------------
<tns:orderDetail amt="2299"/>
------------------------------
```

  - Example 1

```
------------------------------
<assign>
```

```
  <copy>
     <from>$orderDetailVar/@amt</from>
     <to>$billingDetailVar/@amt</to>
  </copy>
</assign>
------------------------------

billingDetailVar: (before <copy>)
------------------------------
<tns:billingDetail amt="" />
------------------------------


billingDetailVar: (after <copy>)
------------------------------
<tns:billingDetail amt="2299" />
------------------------------
```

- *AII-to-EII*

  XML Data Context

```
orderDetailVar:
------------------------------
<tns:orderDetail amt="3399" />
------------------------------
```

  - Example 1

```
------------------------------
<assign>
  <copy>
     <from>$orderDetailVar/@amt</from>
     <to>$billingDetailVar/tns1:billingAmount</to>
  </copy>
</assign>
------------------------------

billingDetailVar: (before <copy>)
------------------------------
<tns1:billingDetail id="8675309">
  <tns1:billingAmount code="F00B2R"></tns1:billingAmount>
</tns1:billingDetail>
------------------------------


billingDetailVar: (after <copy>)
------------------------------
<tns1:billingDetail id="8675309">
  <tns1:billingAmount
code="F00B2R">3399</tns1:billingAmount>
</tns1:billingDetail>
```

```
------------------------------
```

- *AII-to-TII*

```
orderDetailVar:
------------------------------
<tns:orderDetail amt="4499" />
------------------------------
```

  - Example 1

```
------------------------------
<assign>
  <copy>
     <from>$orderDetailVar/@amt</from>
     <to>$billingAmount2Var/text()</to>
  </copy>
</assign>
------------------------------

billingAmount2Var: (before <copy>)
------------------------------
<tns2:billingAmount>0</tns2:billingAmount>
------------------------------

billingAmount2Var: (after <copy>)
------------------------------
<tns2:billingAmount>4499</tns2:billingAmount>
------------------------------
```

- *TII-to-TII*

```
postalCodeVar:
------------------------------
<tns:postalCode>95110</tns:postalCode>
------------------------------
```

  - Example 1

```
------------------------------
<assign>
  <copy>
     <from>$postalCodeVar</from>
     <to>$shippingPostalCodeVar</to>
  </copy>
</assign>
------------------------------

shippingPostalCodeVar: (before <copy>)
------------------------------
<tns:shippingPostalCode>0</tns:shippingPostalCode>
------------------------------
```

```
shippingPostalCodeVar: (after <copy>)
------------------------------
<tns:shippingPostalCode>95110</tns:shippingPostalCode>
------------------------------
```

- *TII-to-AII*

  XML Data Context

  ```
  postalCodeVar:
  ------------------------------
  <tns:postalCode>94304</tns:postalCode>
  ------------------------------
  ```

  - Example 1

    ```
    ------------------------------
    <assign>
      <copy>
         <from>$postalCodeVar/text()</from>
         <to>$shippingAddress1Var/@postCode</to>
      </copy>
    </assign>
    ------------------------------

    shippingAddress1Var: (before <copy>)
    ------------------------------
    <tns1:shippingAddress postCode="" />
    ------------------------------


    approvalNotice1Var: (after <copy>)
    ------------------------------
    <tns1:shippingAddress postCode="94304" />
    ------------------------------
    ```

- *TII-to-EII*

  XML Data Context

  ```
  postalCodeVar:
  ------------------------------
  <tns:postalCode>94107</tns:postalCode>
  ------------------------------
  ```

  - Example 1

    ```
    ------------------------------
    <assign>
      <copy>
    ```

```
      <from>$postalCodeVar</from>
      <to>$shippingAddress2Var/tns2:postalCode</to>
    </copy>
  </assign>
  ------------------------------

  shippingAddress2Var: (before <copy>)
  ------------------------------
  <tns2:shippingAddress id="9035768">
    <tns2:postalCode></tns2:postalCode>
  </tns2:shippingAddress>
  ------------------------------


  shippingAddress2Var: (after <copy>)
  ------------------------------
  <tns2:shippingAddress id="9035768">
    <tns2:postalCode>94107</tns:postalCode>
  </tns2:shippingAddress>
  ------------------------------
```

Below is an explanation of the concepts of XML Namespace Preservation as defined in the text proposed in part (D).

Explanation of the concepts of XML Namespace Preservation defined in (D) (not an addition or modification to the specification):

With the replacement logic defined above in the text to be inserted in (D), in most cases no XML namespace declaration conflicts exist between the source (i.e. the selection result of the from-spec) and destination (i.e. the self-or-parent of the selection result of the to-spec). The XML namespace mechanism is flexible enough to address such cases - for example:

```
<foo:bar xmlns:foo="http://foo.com">
   <!-- this "foo:bar" element is pointed
        by to-spec as the destination of copy -->
   <foo:abc xmlns:foo="http://foo2.com" />
   <!-- this "foo:bar" element is pointed
        by to-spec as the destination of copy -->
</foo:bar>
```

However, some cases exist where a non-trivial conflict may be encountered - for example:
v1:

```
<foo:bar xmlns:foo="http://foo.com" foo:attr="valueA" />
```

v2: (before copy)

```
<p:parent xmlns:p="http://foo.com"
xmlns:foo="http://foo.some.com">
    <p:bar foo:attrX="valueY" />
```

```
</p:parent>
```

With the following <copy> operation, we would encounter a conflict in the use of prefix "foo" in foo:attr and foo:attrX, which are associated with "http://foo.com" and "http://foo.some.com":

```
<assign>
    <copy>
       <from>$v1</from>
       <to xmlns:p="http://foo.com">$v2/p:bar</to>
    <copy>
</assign>
```

To resolve this conflict, the underlying namespace-aware infrastructure is allowed to rename prefixes to those which do not conflict, if necessary. For example, after the copy operation is completed, V2 may look like the following:

v2: (after copy)

```
<p:parent xmlns:p="http://foo.com"
xmlns:foo="http://foo.some.com">
    <p:bar xmlns:foo2="http://foo.com" foo2:attr="valueA" />
</p:parent>
```

or

```
<p:parent xmlns:p="http://foo.com"
xmlns:foo="http://foo.some.com">
    <p:bar p:attr="valueA" />
</p:parent>
```

The details of renaming prefixes are dependent on the underlying namespace-aware infrastructure, which is outside of the scope of this specification. As the above examples illustrate, there is usually more than one way to rename prefixes to handle such a conflict when producing XML namespace consistent data.

When a schema-aware data model is used at runtime in WS-BPEL, a similar prefix renaming mechanism MAY be used to handle namespace declaration conflicts, where QName values of attribute or text form are used.