Issue 280 – Proposal Draft

Last modified: July 20, 2006 – 4pm PDT

Based on rev 1.62 of WS-BPEL specification Draft

[I]
Add clarification / restriction on how SubstitutionGroup is being used in <assign><copy>

At line 2677, add the following to bullet of "RE (Replace-Element-properties)"
~~~~~~~~~~~~~~~~

- 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. [SA00042]It is only applicable when the results of both from-spec and to-spec are EIIs, and MUST NOT be explicitly set in other cases. A WS-BPEL processor MAY enforce this checking through static analysis of the expression/query language. If a violation is detected during runtime, a bpel:selectionFailure fault MUST be thrown.
        - When the `keepSrcElementName` attribute is set to "`no`", the name (i.e. [namespace name] and [local name] properties) of the original destination element is used as the name of the resulting element. This is the default value.
        - 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` is set to "`yes`" and the destination element is a Document Element EII of an element-based variable or an element-based part of a WSDL message-type-based variable, a WS-BPEL processor MUST make sure the name of the source element belongs to the substitutionGroup of the destination element used in the element variable declaration or WSDL part definition. The substitutionGroup relation is determined by XML Schemas known to the WS-Processor. A WS-BPEL processor MAY enforce this checking through static analysis of the expression/query language. If a violation is detected during runtime, a `bpel:mismatchedAssignmentFailure` fault MUST be thrown.

~~~~~~~~~~~~~~~~

Appendix A: the current description seems to be generic enough.
~~~~~~~~~~~~~~~~

`mismatchedAssignmentFailure`	Thrown when incompatible types or incompatible XML infoset structure are encountered in an `<assign>` activity.

~~~~~~~~~~~~~~~~

[II]

In section 12.5 Fault Handlers:

In the case of faults thrown with associated data the fault MUST be caught as follows:

1. If there is a `<catch>` construct with a matching `faultName` value that has a `faultVariable` whose type matches the type of the runtime fault data then the fault is passed to the identified `<catch>` construct. (See the matching criteria definition below)

2. Otherwise if the fault data is a WSDL message type where the message contains a single part defined by an element and there exists a `<catch>` construct with a matching `faultName` value that has a `faultVariable` whose associated `faultElement`'s QName matches the QName of the runtime element data of the single WSDL message part, ~~used to define the part~~ then the fault is passed to the identified `<catch>` construct with the `faultVariable` initialized to the value in the single part's element. (See the matching criteria definition below)

3. Otherwise if there is a `<catch>` construct with a matching `faultName` value that does not specify a `faultVariable` attribute then the fault is passed to the identified `<catch>` construct. Note that in this case the fault value will not be available from within the fault handler but will be available to the `<rethrow>` activity.

4. Otherwise if there is a `<catch>` construct without a `faultName` attribute that has a `faultVariable` whose type matches the type of the runtime fault data then the fault is passed to the identified `<catch>` construct. (See the matching criteria definition below)

5. Otherwise if the fault data is a WSDL message type where the message contains a single part defined by an element and there exists a `<catch>` construct without a `faultName` attribute that has a `faultVariable` whose associated `faultElement`'s QName matches the QName of the runtime element ~~used to define~~data of the single WSDL message part, ~~the part~~ then the fault is passed to the identified `<catch>` construct with the `faultVariable` initialized to the value in the single part's element. (See the matching criteria definition below)

6. Otherwise if there is a `<catchAll>` fault handler then the fault is passed to the `<catchAll>` fault handler.

7. Otherwise, the fault will be handled by the default fault handler (see section).

Matching the type of a `faultVariable` to the runtime fault data as mentioned in points #1 and #4 above is more restrictive than in points #2 and #5. In the case of #1 and #4 above, a WSDL message type variable can only match a WSDL message type fault data, while an element variable can only match element based fault data. For the case of WSDL message based fault, they match only when their QNames are identical. For cases of #1 and #4, where faultElement is used, and point #2 and #5, matching is done by comparing the runtime element-based data and the `faultElement`'s QName.

The runtime element-based data, which originates from throwing a fault with an XSD element-based variable or an XSD type-based variable or a single-part WSDL message based on an XSD element, is considered to be compatible with the globally declared element referenced by `faultElement`, when:

- the QName of the element-based data exactly matches the QName of the referenced element, or,

- the element-based data is a member of the substitutionGroup headed by the referenced element

If multiple `faultElement`-based `<catch>` constructs are compatible with -element-based fault data then their matching priority is as follows:

- A `<catch>` construct with an exact QName match takes precedence.
- If no exact match exists then the first `<catch>` construct with a substitutionGroup match handles the fault, based on the order of appearance within `<faultHandlers>`.
-

Examples:

```
---------------------------------
<scope>
  <variables>
     <variable name="y" element="foo:CustomerElem" />
  </variables>
  …
    <catch faultVariable="x" faultElement="foo:US_CustomerElem">
      …
    </catch>
    <catch faultVariable="x" faultElement="foo:CustomerElem">
      …
    </catch>
  …
  <sequence>
    <assign>
       <copy keepSrcElementName="yes">
           <from>
             <literal>
                <foo:US_CustomerElem> … </foo:US_CustomerElem>
             </literal>
           </from>
           <to variable="y">
       </copy>
    </assign>
    <throw faultName="pr:customerFault" faultVariable="y" />
  </sequence>
</scope>
---------------------------------
```

Given the above scenario, the first <catch> for “foo:US_CustomerElem” will be selected, because there is an exact match between QName used in faultElement attribute and runtime fault data. If the first <catch> is removed from the process definition, the <catch> for “foo:CustomerElem” will be selected because “foo:US_CustomerElem” is a part of substitutionGroup of “foo:CustomerElem”.

On the other hand, if the <catch> for “foo:CustomerElem” is removed and <foo:CustomerElem> is used in the assignment, the fault will NOT be caught by this particular scope.

[III] Additional clarification on fault name and fault data in the case of <reply>

It is relatively orthogonal to part [I] and [II].

Around line 3666:
~~~~~~~~~~~~~~~~~~~~~~~~

The `<reply>` activity has two potential forms. First, in a normal response, the `faultName` attribute is not used and the `variable` attribute (or its equivalent `<toPart>` elements), when present, will indicate a variable with the response message. Second, when the response indicates a fault, the `faultName` attribute is used and the `variable` attribute (or its equivalent `<toPart>` elements), when present, will indicate a variable for the corresponding fault. The `faultName` attribute SHOULD refer to a fault defined in the corresponding operation used in the `<reply>` activity and the faultVariable SHOULD match fault message type associated with fault name as well. (Note: the matching semantics here refer to the point #1 and #2 in `<catch>` related matching rules in Section 12.5 Fault Handlers.) WS-BPEL treats faults based on abstract WSDL 1.1 operation definitions, without reference to binding details. This limits the ability of a WS-BPEL process to determine the information transmitted when faults are returned over a SOAP binding.

~~~~~~~~~~~~~~~~~~~~~~~~