A Modest and Streamlined Approach to Handle Early Completion in BPEL

Last modified: March 8, 2005 – 4 pm CET

Historical Background

In September, a scope-based draft proposal was made for early completion in the context of Issue 6 (Completion Condition) and Issue 142 (Break and Continue). (http://lists.oasis-open.org/archives/wsbpel/200409/msg00168.html)

The draft proposal has a newly added <completionHandler> under scope and newly added <complete> activity. This proposal is a relatively low-level and generic mechanism, which can potentially address the need for both Issue 6 and Issue 142. It was considered as one of the most comprehensive solutions for early completion.

On the other hand, some people raised concern about the complexity and usability of this scope-based solution. It may be not that easy for people to understand the true merit and design principle of this proposal. People may use or implement this feature incorrectly. Particularly, the following areas are not trivial design decisions:

- The ordering of installing, uninstalling, interaction and execution of multiple handlers in a scope (i.e. fault handler, completionHandler, compensationHandler)
- Is the completionHandler designed for forward-works (e.g. maintain housekeeping variables)? Or backward-works (e.g. partial clean up)?
- Should we allow <compensate> activity within <completionHandler>? If not, how do we cancel a flow branch that is a sequence, which contains a scope, which is already completed?
- Is <completionHandler> really needed for majority of cases?

Also, there are different kinds of early completion targets:

- <scope>: related to the scope-based proposal in September
- <flow>: related to Issue 6
- Other activities: e.g. <sequence>, <while>, and <switch>: related to Issue 142

We think that the targets of early completion are actually <flow> activities for most important cases interesting to BPEL. By applying the 80-20 rule, we decide to focus on <flow> for BPEL 2.0 only.

Here, we attempt to distill and streamline the design principle of the scope-based proposal applied consistently to cases where <flow> is the early completion target. Also, by applying certain restrictions to its usage pattern, we believe we minimize the need of <completionHandler>.

Please NOTE this new proposal will NOT preclude any future expansion of early completion targets. It should be forward compatible with scope-based solution, if we decide to go for this scope-based solution in future iteration of BPEL specification.

Proposal Details

Part 1: <complete> activity

```
<complete target="flow-activity-name" />
```

For BPEL 2.0, the target of <complete> MUST be a <flow> activity. (In future version of the BPEL specification, the target might be expanded to other activities, e.g. <scope>.) If other kinds of activities are the targets, static code analysis MUST reports as error.

The target activity MUST be an ancestor of the <complete> activity lexically. The target activity is determined by the closest ancestor activity with the matching activity name.

All the branch activities of the target <flow> MUST be <scope>-based activity. If the target <flow> contains some nested inner <flow> activities, the same <scope>-based-only restriction applies to the branches of the inner <flow> activities. If some of branches are non-<scope> based, static code analysis MUST reports as error.

This scope-based only branches restriction does the following:

- Enforcing a clear boundary (i.e. <scope>) to encapsulate the effect and work of early completion to each branch; e.g.:
    - Enabling self clean up of branches, when upon termination triggered by early completion
    - Minimizing the need of <completionHandler>
- Allowing differentiation of branch-based completion conditions (see Part 2)

Details of semantics of <complete> activity:
For BPEL 2.0, the target flow of a <complete> activity MUST be the direct <flow> ancestor of the <complete> activity. That means, there are no other <flow> activities on BPEL construct ancestor chain between the target <flow> and <complete> activity.

[This restriction may be relaxed if we are willing to accept another version of early-completion protocol. Please see "Open Issues" below.]

The <complete> activity will perform an early completion to the target <flow> activity.

- The scope of the branch, in which a <complete> activity has been executed, is completed successful. Hence, the compensation handler will be installed.
- If there are any other sibling activities within the same parent <sequence> after the <complete> activity within the branch, those activities are unreachable BPEL code. This kind of coding combination MUST be reported as error during static

analysis. E.g.:

```
<sequence>
   ...
   <complete target="..." />
   <invoke name="invoke1" ... />
</sequence>
```

The “invoke1” activity MUST be reported as error.

- If there are any other activities within an ancestor <sequence> after the <complete> activity within the branch, those activities will NOT be executed. E.g.:

```
<sequence>
   ...
   <switch>
      <case> ...
          <sequence>
             ...
             <complete target="..." />
          </sequence>
      </case>
   </switch>
   <invoke name="invoke2" ... />
</sequence>
```

The “invoke2” activity will NOT be executed.

- The branches that either completed or faulted before the <complete> activity is executed will not be affected.
- The scopes of other branches, which are still executing, will be terminated. Therefore, the terminationHandler of the scope will be invoked to perform a self clean up.
- Other fault handling:
    - When a branch scope is being completed early by executing <complete> activity, it may result into other erroneous situations (e.g. missingReply fault).
    - When a branch scope is being terminated, it may result into other erroneous situations (e.g. missingReply fault). Those errors will NOT be propagated out of the scope, because it is already in the middle of forced termination. This semantics is consistent with forced terminations which are not triggered by early completion.

Part 2: <completionCondition>

In addition to <complete> activity, <completionCondition> construct is also available for a more declarative approach to achieve similar logic. It is noteworthy that both

<complete> activity and <completionCondition> can be used at the same time within the same <flow>.

There are two kinds of completionConditon:

- <booleanExpression>: It is a generic boolean condition operating upon process variables, which is evaluated at the end of execution of each <flow> branch. If the condition is evaluated to be true, the <flow>, which the <completionCondition> attaches to, will be completed early.
- <branches>: It is an integer expression value which is used to achieve N out of M branches logic. The integer expression will be evaluated at the beginning of the flow to yield an "N" value for the lifetime of flow. At the end of execution of each branch, the BPEL processor will count how many branches have been finished. If N or more branches (calculated at the beginning of <flow>) have been finished, the <flow>, which the <completionCondition> attaches to, will be completed early. Details of how to count branches will be discussed below.

  Please note: the branch condition is the "at least" version of N out of M semantics. (The exact N out of M condition semantics involve resolving racing condition among parallel branches and are NOT needed by the majority of users)

Both conditions (<branches> and <booleanExpression>) may be specified at the same time. They will be checked when one branch of the <flow> activity completes. If at least one condition evaluates to true, all remaining active branches of the <flow> activity will be terminated. If both conditions are specified, the <branches> will be evaluated first. And, if <branches> condition is evaluated to be true, the <booleanExpression> condition will NOT be evaluated.

Syntax:

```
<flow>
           <completionCondition> ?
             <branches expressionLanguage="URI"?
                       countCompletedScopesOnly="yes|no"?>
                    an-integer-expression
             </branches>?
             <booleanExpression expressionLanguage="URI"?>
                    a-boolean-expression
             </booleanExpression>?
     </completionCondition>
</flow>
```

A <completionCondition> is basically a special macro of <complete> activity. Picking the boolean kind of <completionCondition> as an example:

```
<flow>
     <completionCondition>
          <booleanExpression>
               a-boolean-expression
          </booleanExpression>
     </completionCondition>
     <scope name="s1"> activity-X </scope>
     <scope name="s2"> activity-Y </scope>
</flow>
```

It could be expressed as follows, high level speaking:

```
<flow name="f1">

      <scope name="s1">
          <sequence>
             activity-X
             <switch>
                 <case>
                       <condition>
                          a-boolean-expression
                       </condition>
                       <complete target="f1" />
                 </case>

             </switch>
          </sequence>
      </scope>


      <scope name="s2">
         <sequence>
            activity-Y
            <switch> … similar switch … </switch>
          </sequence>
      </scope>
</flow>
```

<u>More details of counting branches:</u>

When the integer value evaluated from the <branches> expression is larger than the number of branches in the <flow>, then bpws:invalidBranchCondition fault MUST be thrown. Please note that the number of branches may be known only during runtime in some cases. Static analysis should be encouraged to detect this erroneous situation at design time when possible. (E.g. when the branches expression is a constant.)

<branches> expression has an optional “yes”/“no” called “countCompletedScopesOnly”. Its default value is “no”. When BPEL processor counts branches for the completionCondition, there are three possibilities in terms of the state of a scope:

- executing
- finished with normal completion (i.e. compensation handler is installed)
- exited with a fault (i.e. a fault handler is executed)

If countCompletedScopesOnly is “no”, it means the BPEL processor will count scopes with normal completion AND scopes exited with a fault.

If countCompletedScopesOnly is “yes”, it means the BPEL processor will count scopes with normal completion ONLY.

Please note: there is a special case for branch condition: When N = M and countFaultedScopes is “yes”, a <flow> with that branch condition is basically the same as the one without the condition.

E.g.:

```
<flow>
      <completionCondition>
            <branches>2</branches>
      </completionConditon>
      <scope name="s1"> … </scope>
      <scope name="s2"> … </scope>
</flow>
```

is identical to:

```
<flow>
      <scope name="s1"> … </scope>
      <scope name="s2"> … </scope>
</flow>
```

After all scopes branches ends, if the completionCondition is still evaluated to false, the "bpws:completionConditionFailure" MUST be thrown.

This fault is particularly useful when we use the generic boolean completionCondition is used or countCompletedScopeOnly attribute is set to "yes" for <branches> completionCondition.

Open Issue:

*** IF we want to allow <complete> activity to perform an early completion against a <flow> activity which is not the inner most <flow> activity in the ancestor chain, the following paragraph will describe the <complete> activity semantics.

Details of semantics of <complete> activity:
A <flow> [F1] is the target of a <complete> activity, which locates within an inner <flow> activity [Fn]. The ancestor chain for <flow> activities are [Fn], [Fn-1], … [F2] and [F1]. (See example below)

The <complete> activity will perform an early completion to the inner most <flow> [Fn] activity first.

- The scope of the branch, in which a <complete> activity has been executed, is completed successful. … *[Same bullet list]*

After the early completion of <flow> [Fn] (i.e. steps listed above) is finished, the early completion signal is *cascaded* up the chain of <flow> activities by performing the steps to next innermost <flow> [Fn-1]. The cascade will end at <flow> [F1], which is the target of <complete> activity. Please note that: the early-completion signal is always sent upwards, while the termination signal is always sent downwards, if we see BPEL constructs as a tree.

Example:

```
<flow name="F1">
      <scope name="S1">
            <flow name="F2">
                  …
                  <scope name="S2">
                        <flow name="Fn">
                              …
                              <scope name="Sn">
                                    <sequence>
                                          …
                                          <complete target="F1" />
                                          …
                                    </sequence>
                              </scope>
                              …
                        </flow>
                  </scope>
                  …
            </flow>
      </scope>
      …
</flow>
```

END