Issue 6.2: Updated proposal:
Includes amendments made at September F2F, plus 6.2b
Last modified: Sept 21, 2005 - 7:00pm PDT

Proposal

Syntax:

```
<forEach counterName="ncname" ...>
     standard-elements
     <iterator>
        <startCounterValue expressionLanguage="anyURI">
           ...
        </startCounterValue>?
        <finalCounterValue expressionLanguage="anyURI">
           ...
        </finalCounterValue>?
     </iterator>
     <completionCondition>?
     Scope
</foreach>
```

```
<completionCondition extension-attribute>
      extension-element
      <branches expressionLanguage="URI"?
countCompletedBranchesOnly="yes|no"?>
          an-integer-expression
      </branches>
</completionCondition>
```

Semantics:

(1) The completionCondition element is an optional element of the ~~parallel~~ forEach activity. Default behavior of the ~~parallel~~ forEach activity is they wait for all their directly enclosed activities to complete. ~~Optional <completionCondition> is applicable to parallel forEach only, and not serial forEach~~. ~~This restriction MUST be enforced by static analysis~~.

(2) <branches> is an integer value expression which is used to define condition of flavor N out of M. The actual value of the expression is calculated at the beginning of the forEach activity. It will not change as the result of the forEach execution. ~~-~~At the end of execution of each directly enclosed activity the number of completions is

checked against this value. This condition has "at least N out of M" semantics. (The exact N out of M condition semantics involves resolving racing condition among directly enclosed activities.)

(3) If the integer value is larger than the number of directly enclosed activities, then bpws:invalidBranchCondition fault MUST be thrown. 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. (For example, when the branches expression is a constant.)

(4) Element <branches> has an optional attribute "countCompletedBranchesOnly". Its default value is "no". If countCompletedBranchesOnly is "no", it means the WS-BPEL processor will count branches which have completed (either successfully or unsuccessfully). If countCompletedBranchesOnly is "yes", it means the WS-BPEL processor will count branches, which have completed successfully only.

(5) If ~~parallel~~ forEach activity specifies a completionCondition element the completion condition is evaluated each time a directly enclosed activity completes. If upon completion of a directly enclosed activity, it can be determined that the completion condition can never be true the "bpws:completionConditionFailure" MUST be thrown by the ~~parallel~~ forEach activity. If the completion condition evaluates to true the ~~parallel~~ forEach activity completes successfully.

- **For a parallel forEach activity, after completionCondition is fulfilled**, all still running directly enclosed activities will be terminated. This means that the ~~parallel~~ forEach activity begins implicitly terminating the directly enclosed activities that are still active.

- **For a serial forEach activity, when the completionCondition is being evaluated, there will be no directly enclosed activities actively running and hence no termination will be triggered. After completionCondition of a serial forEach is fulfilled, the serial forEach will just simply complete.**

(6) Standard WS-BPEL termination semantics applies to running directly enclosed activities when the

completion condition is met. The termination of running nested activities follows the termination semantics defined in the specification (see section 13.4.4 Semantics of Activity Termination).

[Chris' amendment]:
Fix language in section 13.4.4 Semantics of Activity Termination from:

"However, a termination handler cannot throw any fault. Even if an uncaught fault occurs during its behavior, it is not rethrown to the next enclosing scope. This is because the enclosing scope has already either faulted or is in the process of being terminated, which is what is causing the forced termination of the nested scope."

Add that the termination may be the outcome of early completion of parallel forEach. The new version would be:

"However, a termination handler cannot throw any fault. Even if an uncaught fault occurs during its behavior, it is not rethrown to the next enclosing scope. This is because: (a) the enclosing scope has already either faulted or is in the process of being terminated, which is what is causing the forced termination of the nested scope; (b) the scope being terminated is a branch of a parallel <forEach> and early completion mechanism triggers the termination as <completionCondition> of <forEach> is fulfilled."

(end)
