WS-BPEL Extension for Sub-processes – BPEL-SPE

A Joint White Paper by IBM and SAP
September 2005

Authors (alphabetically):
Matthias Kloppmann, IBM (matthias-kloppmann@de.ibm.com)
Dieter Koenig, IBM (dieterkoenig@de.ibm.com)
Frank Leymann, IBM (ley1@de.ibm.com)
Gerhard Pfau, IBM (gpfau@de.ibm.com)
Alan Rickayzen, SAP (alan.rickayzen@sap.com)
Claus von Riegen, SAP (claus.von.riegen@sap.com)
Patrick Schmidt, SAP (patrick.schmidt@sap.com)
Ivana Trickovic, SAP (ivana.trickovic@sap.com)

Abstract
Designing complex and large business processes requires a language that supports modularization and reuse in a portable, interoperable manner. This paper outlines an extension to WS-BPEL that allows for the definition of sub-processes that can be reused within the same or across multiple WS-BPEL processes. The paper describes different invocation scenarios and introduces a coordination protocol to be used for interoperable invocation of sub-processes across infrastructures from different vendors.

Contents

1 Introduction

1.1 Problem Statement

The Web Services Business Process Execution Language 2.0 (WS-BPEL 2.0, or BPEL for short) defines a model for services composition, which is, the aggregation of existing Web services into new Web services. The language is structured into two parts. One is the model for executable business processes that is used to specify automated business processes that orchestrate activities of multiple Web services, and which may be interpreted and executed by compliant engines. The other part is the observable behavior of Web services. Both parts require a common core of process description concepts. The BPEL common core encompasses concepts needed to describe process control flows, including compensation behavior and error handling. Those concepts are seen and used in multiple process models and are needed to build complex processes, which can be executed by underlying software.

The BPEL language currently does not support the explicit definition of business process “fragments” that can be invoked from within the same business process or from another business process. Such functionality is well-known in business process technology and subsumed by the term sub-process (activity). Sub-processes are used in practice to modularize large business process and to foster reuse of processes. Consequently, sub-processes are typically tightly coupled in terms of their lifecycle to the invoking process (called parent process). For example, when a parent process is terminated, its sub-processes have to be terminated too; when the activity invoking a sub-process is compensated, compensation is delegated to the sub-process.

The only way to approximate similar behavior in BPEL is by defining a complete business process as an independent service and invoking it via an invoke activity. The fact that this activity is really implemented as another process is completely hidden from the parent process. There is no mechanism to establish the lifecycle dependency described above. As a consequence, this approach is really only an approximation. Although the invoke activity of the parent process will be interrupted and terminated, BPEL does not provide any means to ensure that termination propagates to the invoked Web service. Therefore it cannot be ensured that the sub-process, which is exposed as a Web service, will be terminated if the parent process terminates.

As a service, a sub-process typically implements a single operation from the point of view of the parent process. Thus, to be useable as a sub-process, a process must comply with a few restrictions, which are discussed below. Combined with the ability to define sub-processes inline within another process, the concept of sub-processes provides for an easy definition and invocation of reusable process fragments.

1.2 Goals

In the following sections we describe features needed to provide sub-process capabilities in a more direct way, with the following goals:

- Allow for the invocation of a business process as a sub-process of another business process, such that its lifecycle is tightly coupled to the lifecycle of the parent process.

- Allow for the definition of a business process within the context of another business process, so it can be used (and reused) within that other process.
- Allow a sub-process defined within the context of another business process to access data from its parent process.
- Allow for the invocation of sub-processes across BPEL engines so that a process running on one BPEL engine can invoke a sub-process on another BPEL engine.

The BPEL-SPE extension is defined as a layer on top of the BPEL language so that its features can be composed with the BPEL core features whenever needed. We envisage that additional BPEL extensions may be introduced which may use BPEL-SPE.

The paper is organized as follows: Section 2 introduces the notion of sub-process in general, as well as the notion of a standalone sub-process and an inline sub-process. Details about different modes of invocations between a parent process and a sub-process are described in section 3. Interoperability aspects such as a common state model and the protocol to couple the lifecycle between a parent process and a sub-process defined and executed in different environments are detailed in section 4.

2 Definition

A sub-process should be understood as a fragment of BPEL code that can be reused within a process or across multiple processes. It may also be a long-running process, which includes interactions with other partners. However, the interaction of a sub-process with its parent process is typically limited to the initiating request message and the final reply message. A sub-process may be defined either locally within another BPEL process and reused only within that process or as a BPEL process and reused across other BPEL processes. Note that the latter kind of process can be used both as a sub-process as well as a business process on its own. It is as late as at instantiation time that an instance of such a process becomes a sub-process or not.

A sub-process is executed in the context in which it is defined. Usual lexical scoping rules apply. This means that a sub-process can access variables from the scope where it is defined. For example, a sub-process defined as a separate BPEL process can access no variables from other processes. An inline sub-process defined directly under a process element can access only global process variables and not those defined within other nested scopes. In addition, a mechanism to pass data between a calling process and a called inline sub-process is provided. This is needed in cases when an inline sub-process calls another sibling inline sub-process.

The relationship between a parent process and a sub-process extends on the error handling model and the compensation model introduced in BPEL. That means, if a fault occurs in a sub-process and it is not caught by any locally defined fault handler, the sub-process terminates unsuccessfully and a fault is returned to the parent process. On the other side, if a fault occurred and the parent process terminates, the sub-processes are canceled. And if the scope encompassing an already completed sub-process is compensated, compensation is delegated to the sub-process.

2.1 Standalone Sub-processes

Standalone sub-processes are defined as BPEL processes. The structure of a standalone sub-process is such that it implements exactly one activity, that is, from the calling process' perspective the sub-process is like an operation of a service that consumes a message and that may return a message as response. This implies the following restrictions:

- A sub-process has a single initiating receive activity, consuming the input message of the implemented operation.
- The operation is a "logical request-response operation", which is either represented as a WSDL request-response operation, or as two one-way operations. In the case of a request-response operation, the sub-process returns its result via a corresponding reply activity (see Figure 1 in section 3). In the case of two one-way operations, the sub-process returns its result via an invoke operation to the partner from which the initiating input message originated, using the same BPEL partner link (see Figure 2 in section 3). For the remainder of this document, we do not always explicitly spell out these two representations for the logical request-response operation. Even when not explicitly stated, the final (logical) reply of the sub-process could be realized by a (physical) invoke.
- The reply activity is "the last" action of the sub-process. Different replies are allowed on conditional branches of the sub-process, but each of them must be the last action on the respective branch.
- No interaction beyond the initiating request message and the final response message is allowed between the sub-process and its parent process. Syntactically, the partner role of the partner link used for the receive/reply activities of the sub-process must be empty (in the case of a request-response operation provided via receive-reply), or there is a single one-way message for both roles (in the case of a "logical request-response operation" provided via receive-invoke).

2.2 Inline Sub-processes

Sub-processes can be defined as part of the definition of another process. They are defined within either the process scope[1] or a nested scope. Such a sub-process has the following restrictions:

- It is only visible within the scope it is defined in (or its nested scopes). In particular, an inline sub-process is never visible outside the BPEL process enclosing it.
- If data needs to be passed into the sub-process, it has the same restrictions specified above for standalone sub-processes, that is, it must implement a single logical request-response operation.
- If no data needs to be passed into the sub-process, it does not need to have any receive (or reply) activity at all.
- Names of sub-processes need to be unique within the scope of their definition.

Activities of such an inline sub-process have access to all variables, correlation sets and partner links defined in scopes enclosing the sub-process definition. Lexical scoping rules as known from structured programming languages apply. Access to variables is serialized if specified at the scope level.

[1] The process scope is the container of all elements, including variables, fault handlers, event handlers) defined on the process level.

An inline sub-process cannot be a business process on its own. As a consequence, it cannot be instantiated as a standalone process. An inline sub-process is typically not defined via a complete BPEL process model, that is, it may omit certain constructs otherwise being mandatory for a correct standalone process. For example, an inline sub-process may be defined without its own variables, without its own partner links, or without an initial receive activity. Since an inline sub-process has access to the corresponding constructs of its enclosing scope it can manipulate variables defined there – it can use the corresponding partner links, and without a receive activity it just reads the data needed from its enclosing scope. In this sense an inline sub-process can only be a “process fragment”.

Here is an example for an inline sub-process definition within a standard BPEL scope. Note that the example is only illustrative and does not propose syntax for the sub-process extension. This is why pseudo-syntax is used to illustrate the extension.

```
  <scope>

     <variables>
        <variable name="input" messageType="..."/>
        <variable name="output" messageType="..."/>
     </variables>

     [subprocesses]
       [process name="mySubProcess"]
         <!-- Standard process definition here,
              with appropriate limitations and relaxations -->
       [/process]
     [/subprocesses]

     ...
</scope>
```

3 Invocation

BPEL-SPE supports two different mechanisms by which a sub-process can return a response. It can do so either via a reply corresponding to the starting receive activity, or via a one-way invoke that is logically related to the starting receive activity. The first case is depicted in Figure 1. The parent process uses a WSDL request-response operation provided by the sub-process. The sub-process consumes the input message via its starting receive activity and returns the response message via the corresponding reply activity. The WSDL operation is defined in a port type used by the partner role within a partner link of the parent process. In this partner link, the parent process has no need to provide a port type in the corresponding `myRole`, because the sub-process does not further interact with the parent process.

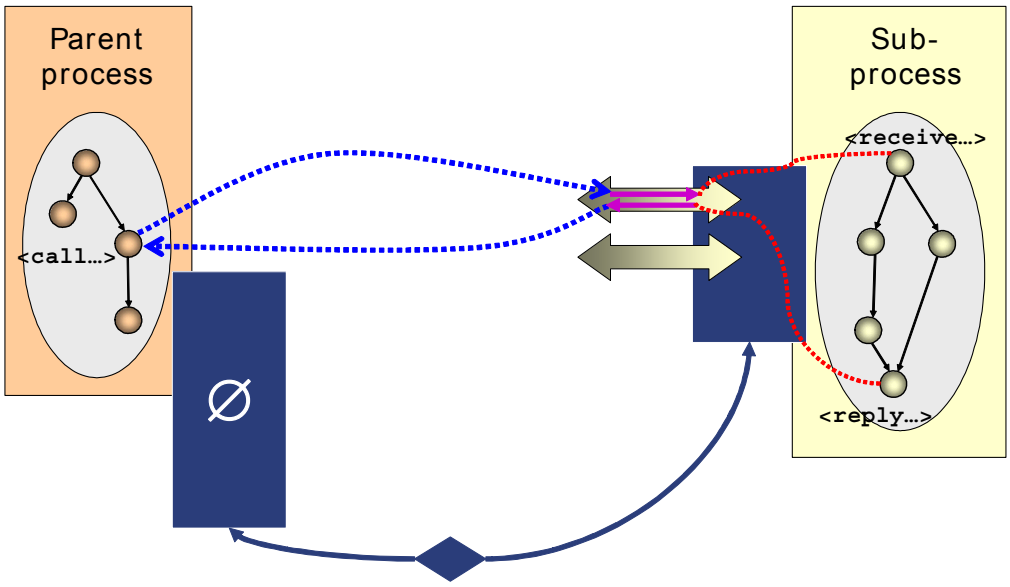


Figure 1. Invocation of a sub-process implemented via a receive-reply pattern

The second case is depicted in Figure 2. The parent process includes a partner link where the associated partner role's port type contains the operation corresponding to the initial receive of the sub-process. The `myRole` of the partner link contained within the parent process references a port type, which contains the operation to be used by the sub-process to return the response message. The port type and operation of the partner role that is to be used to send the input message to the sub-process is specified in the call activity (like in BPEL invoke or reply activities). But in addition we need to inform the process engine hosting the parent process which operation to use to receive the response message. Thus, the call activity used to call a sub-process requires the specification of the corresponding port type and operation in the `myRole` of the partner link wiring the parent process and the sub-process. This is new to BPEL which today only supports a single port type and operation in activities communicating with the outside.

The port type of the partner link that corresponds to the sub-process can be bound to the associated service endpoint at deployment time of the parent process. The analog cannot be done for sub-processes. A sub-process may be called by many different parent processes each of which may have different endpoints to which the sub-process must return its response. Thus, the endpoint of the parent process must be made known to the sub-process at runtime. More precisely, this endpoint must be known to the process engine hosting the sub-process at the time the response message must be sent back. There are various means to achieve this; for example by passing the corresponding endpoint reference "out-of-band", or by including the endpoint reference in the header of the input message sent to the sub-process.

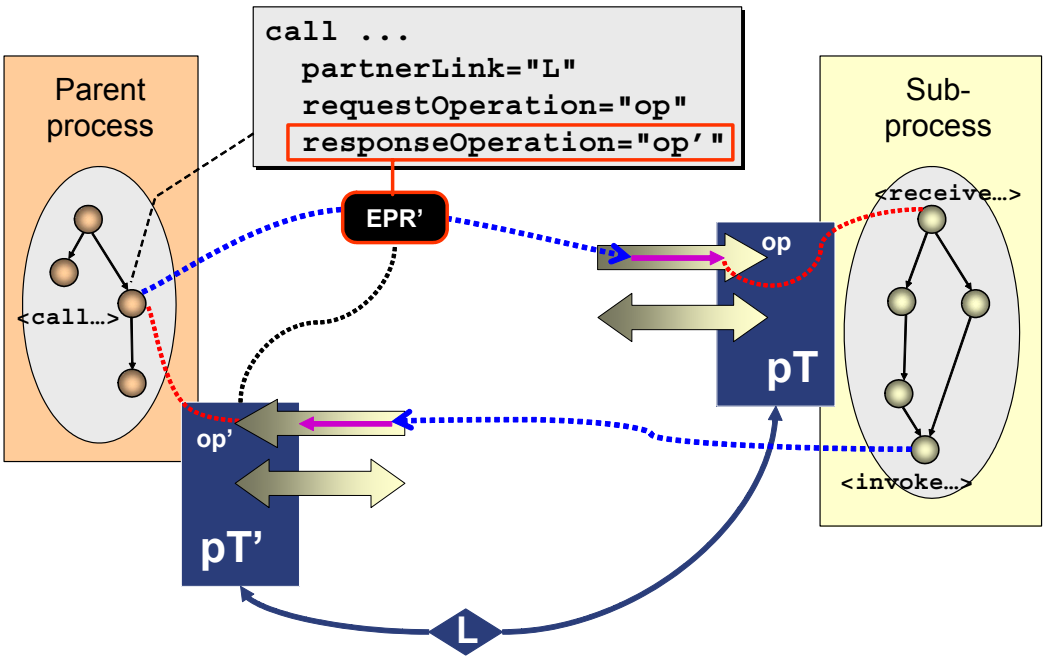


Figure 2. Invocation of a sub-process implemented via a receive-invoke pattern

3.1 Calling a Local Sub-process

Sub-processes are called via a new activity type introduced by BPEL-SPE, the "call" activity, which specifies its implementation by referring to the sub-process to call. For local sub-processes, that is sub-processes managed by the same infrastructure as the parent process, including inline sub-processes, the reference can be done via the qualified name of the sub-process. This representation is preferable to extending the invoke activity by an additional attribute that would designate the process to be called, because the referenced process already defines its signature (the operation it provides and the associated partner link) so this information does not need to be duplicated in the call activity. In addition, the call activity specifies variables for input, output and fault(s).

Here is an example in pseudo-syntax, without output and faults:

```
[call xmlns:s="http://stuff-is-us.com"
      process="s:myProcess"
      inputVariable="var1"/]
```

The types of the variables specified for input and output (including faults) must be assignment compatible. An inline sub-process can access data within its parent scope(s) in addition to consuming input data and producing result data.

The semantics of the new activity, which will be part of a parent process, has the following behavior:

- The referenced sub-process is called, with the input data being passed. Control is passed from the branch of the process containing the activity to the sub-process. The data is passed by value. This decision is motivated by the fact that sharing data "by reference" introduces additional complexity, because in that case it must be assured consistent access to shared (transient) data.

- On completion of the sub-process, its result is passed back and written to the output variable in the parent process' context. Control is returned to the parent process which resumes execution on this branch.
- If a fault is raised by the sub-process, it is thrown to the scope enclosing the call activity.

If the scope containing the call activity is terminated while the activity is active, termination is signaled to the sub-process as it would be signaled into a nested scope, that is, by propagating a regular BPEL termination signal into it. Regular BPEL termination semantics applies. This means that all activities within the sub-process which are still active are terminated according to the BPEL activity termination semantics.

3.2 Calling a Remote Sub-process

To handle the remote case, where the called sub-process is not managed by the same infrastructure and hence its qualified name cannot be used to locate it, BPEL-SPE uses a different syntax for the call activity, locating the sub-process through a partner link like a standard invoked service. The syntax of the activity is based on the syntax of the invoke activity. BPEL-SPE extends this syntax to allow for the specification of the operation that is used to receive the response from the sub-process. Details of error handling and compensation behavior are discussed in sections 3.3 and 3.4.

Here is the pseudo-syntax:

```
[call name="handleOrder"
      partnerLink="myOrderProcess" requestOperation="submitOrdner"
      inputVariable="myOrder" outputVariable="orderingResult"
      receiveOperation="receiveOrder" /]
```

3.3 Fault Handling

In the case of an inline sub-process, untrapped faults from within the sub-process are propagated across the sub-process boundary into the scope enclosing the corresponding call activity. Such a fault is not distinguishable from any other fault thrown in the parent process.

In the case of a standalone sub-process, untrapped faults are transformed into a new BPEL standard fault (for example, `abnormalSubprocessTermination`), which is thrown into the parent process' scope enclosing the corresponding call activity. This fault indicates that an erroneous situation occurred within the sub-process and it might be handled in a special manner.

Application specific faults can be passed in case of a logical request-response operation provided as a WSDL request-response operation by returning a fault through the usual BPEL mechanism of returning a fault on reply. In contrast, in the case of a logical request-response operation provided as two one-way operations, returning application faults is not possible. The reason is that the BPEL-SPE extension proposed in this white paper is based on WSDL 1.1 which is broadly supported by the industry today. The WSDL 1.1 model does not allow specifying fault messages on one-way operations or notification operations. In the model proposed in this paper, one-way operations are used for consuming the response from a sub-

process by the parent process when the sub-process implements the receive-invoke pattern (see Figure 2). To support the exchange of application specific fault messages we would have had to invent mechanisms to pass faults on one-way messages, especially. But WSDL 2.0 provides such means by its in-fault and out-fault constructs. In proposing the new BPEL standard fault above we avoid legacy problems.

3.4 Compensation

BPEL-SPE treats a sub-process as a scope. Thus, BPEL-SPE introduces the ability to compensate a process instance as a whole. To do that it must be possible to specify a compensation handler at the process level. This compensation handler is invoked after completion of the process, to undo (in some cases only partially) the effects of the process execution.

The call activity represents the sub-process within the parent process – it is a proxy for that sub-process, very much like a scope, though with a separate implementation. Thus, a call activity can be the target of a compensate activity. In addition, the default compensation handler of any enclosing scope will propagate compensation not only to contained scopes, but also to contained call activities. Triggering compensation for a call activity propagates the compensation request to the associated sub-process, resulting in deep compensation. As usual, shallow compensation can be achieved by providing a custom compensation handler on a scope enclosing the call activity.

An example for a standard BPEL scope with a compensation handler containing a call activity could look as follows:

```
<scope>

   <compensationHandler>
      <sequence>
         <compensate scope="handleOrder"/>
      </sequence>
   </compensationHandler>

   [call name="handleOrder"
         partnerLink="myOrderProcess" operation="submitOrdner"
         inputVariable="myOrder" outputVariable="orderingResult" /]

</scope>
```

4 Coordination Protocol Between Process and Sub-processes

In this section, we describe invocation of remote standalone sub-processes defined and implemented in different BPEL infrastructures, possibly from different vendors. To achieve the quality of service for the invocation of sub-processes as discussed in the previous sections, status information messages in addition to standard service invocation messages must be exchanged between the two infrastructures. Examples of status messages are:

- close – upon successful completion of the parent process it sends the close message to a sub-process
- compensate – the parent process propagates compensation down to a sub-process

- terminate – termination of the parent process is propagated down to a sub-process
- exit – explicit termination of the parent process is propagated down to a sub-process
- subprocessExited – the parent process is notified that a sub-process completed unsuccessfully
- subprocessTermination – the parent process is notified that a sub-process has been explicitly terminated

This additional set of messages and the order in which the messages may be exchanged constitute the sub-process coordination protocol. The following sections briefly describe the protocol and the corresponding state diagrams.

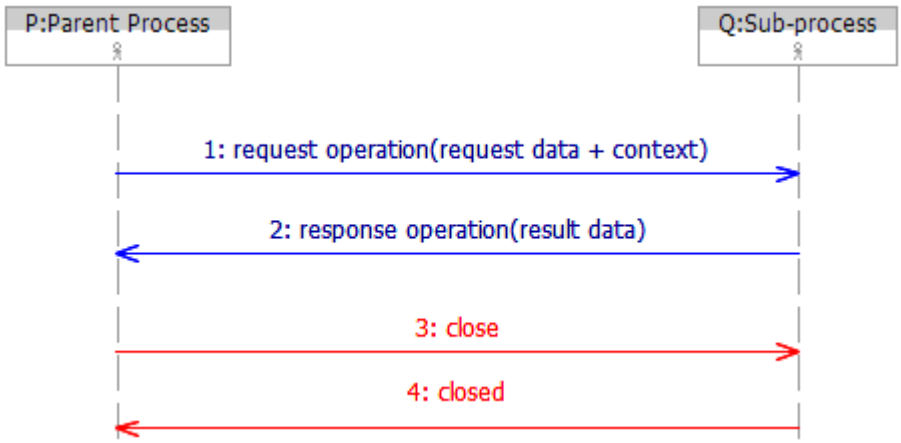
4.1 Sub-process Coordination Scenarios

In this section, we look at the different scenarios for exchange of coordination protocol messages between a parent process and a sub-process.

4.1.1 Invocation of Sub-process, without Compensation

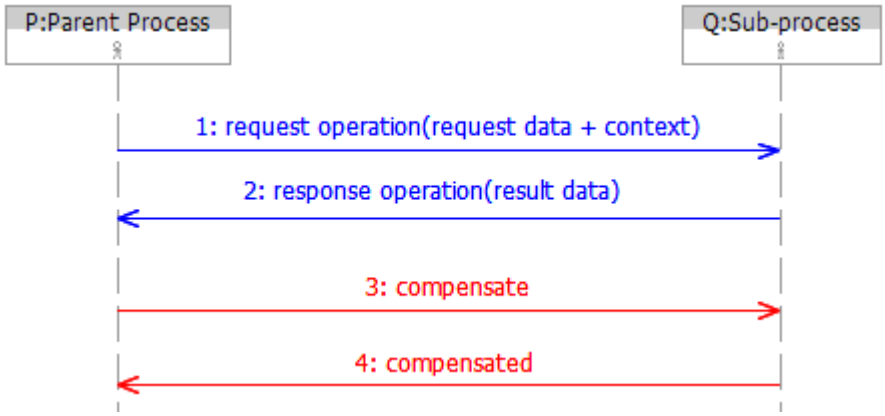
When the call activity is reached in the parent process, the associated sub-process is started by invocation of the specified Web service operation. A coordination context, which contains appropriate correlation information for subsequent exchange of sub-process coordination protocol messages, is passed. Upon completion of the sub-process, the result is sent back to the parent process.

Once the parent process completes, the sub-process is closed, so that it can free state data kept for compensation.



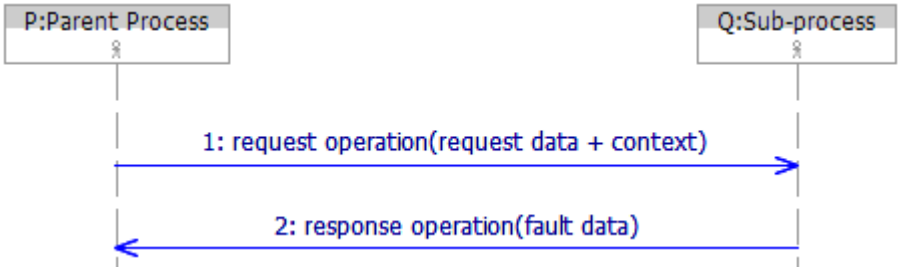
4.1.2 Invocation of Sub-process, with Compensation

As in the scenario described in section 4.1.1, the sub-process is invoked and returns a result. Eventually, compensation is triggered in the parent process as described in section 3.4, and propagated down to the sub-process.



4.1.3 Invocation of Sub-process, with Application Fault

As in the scenario described in section 4.1.1, the sub-process is invoked. In this scenario, it does not return a result, but responds with an application level fault and closes without possibility to be compensated later.



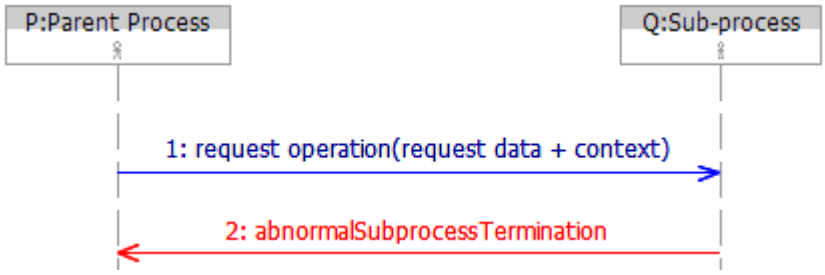
4.1.4 Abnormal Termination of Sub-process Due to Missing Response

As in the scenario described in section 4.1.1 the sub-process is invoked. In this scenario the sub-process completes but does not send a response back to the parent process (that is, the sub-process end is reached without encountering a reply activity or an invoke activity). This may happen, for example, for one of the following reasons:

- Due to a fault not being handled, that is, the fault reaches the process boundary.
- Within normal execution, the process reaches its end without having executed a response activity (because of a modeling error).

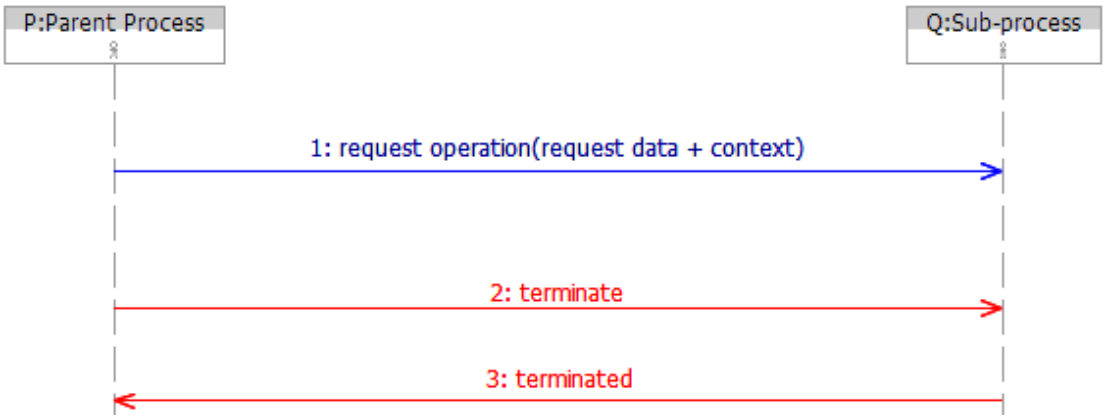
A new BPEL standard fault `abnormalSubprocessTermination` is thrown in the context of the call activity.

Note: If a process is invoked as a sub-process and completes normally but without sending a response back, this is considered a violation of the sub-process contract so the sub-process ends in state “Failed”. If the same process was invoked as a top-level process, it would have completed successfully.



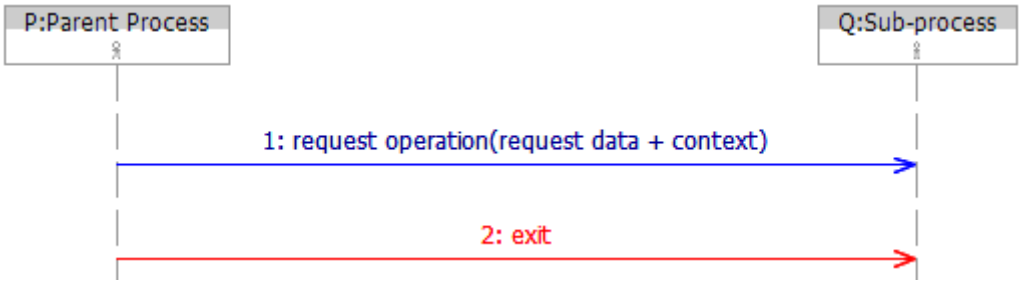
4.1.5 Termination Propagation from Parent to Sub-process

The scope enclosing the call activity in the parent process is terminated (for example, because of a fault caught at its boundary). The termination is propagated to the sub-process. Once termination has been processed (for example, termination handlers have been executed), its completion is signaled back to the parent process.



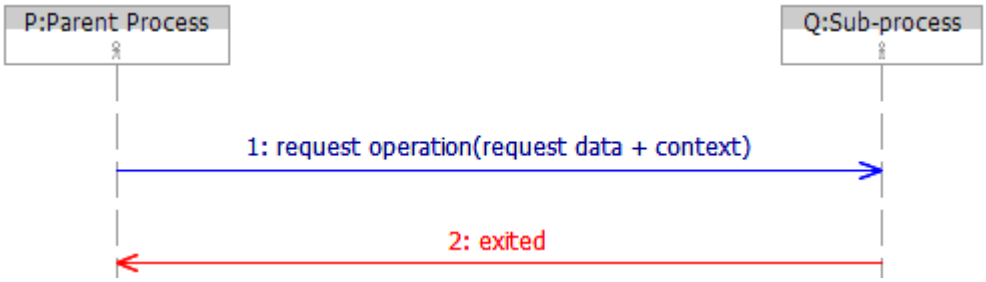
4.1.6 Propagation of Explicit Termination from Parent to Sub-process

The parent process encounters a BPEL exit activity. The exit signal is propagated to the sub-process, which exits in turn. This is a one-way notification only, as exit happens immediately and unconditionally.



4.1.7 Propagation of Explicit Termination from Sub-process to Parent

The sub-process encounters a BPEL exit activity. The exited signal is propagated to the parent process, which throws a new BPEL standard fault `subprocessExited` in the context of the call activity.



4.2 State Diagram of the Call Activity

The call activity represents the sub-process within the parent process, which has properties of a BPEL scope; that is, a sub-process can be compensated, terminated, completed either successfully or unsuccessfully, etc. Therefore, the behavior of the call activity is consistent with the behavior of a BPEL scope. However, unlike the BPEL scope, the sub-process may be explicitly terminated, which causes the call activity to throw the new BPEL standard fault `subprocessExited` and exit.

Figure 3 shows the complete state diagram describing the behavior of the call activity in the parent process with respect to the sub-process coordination protocol.

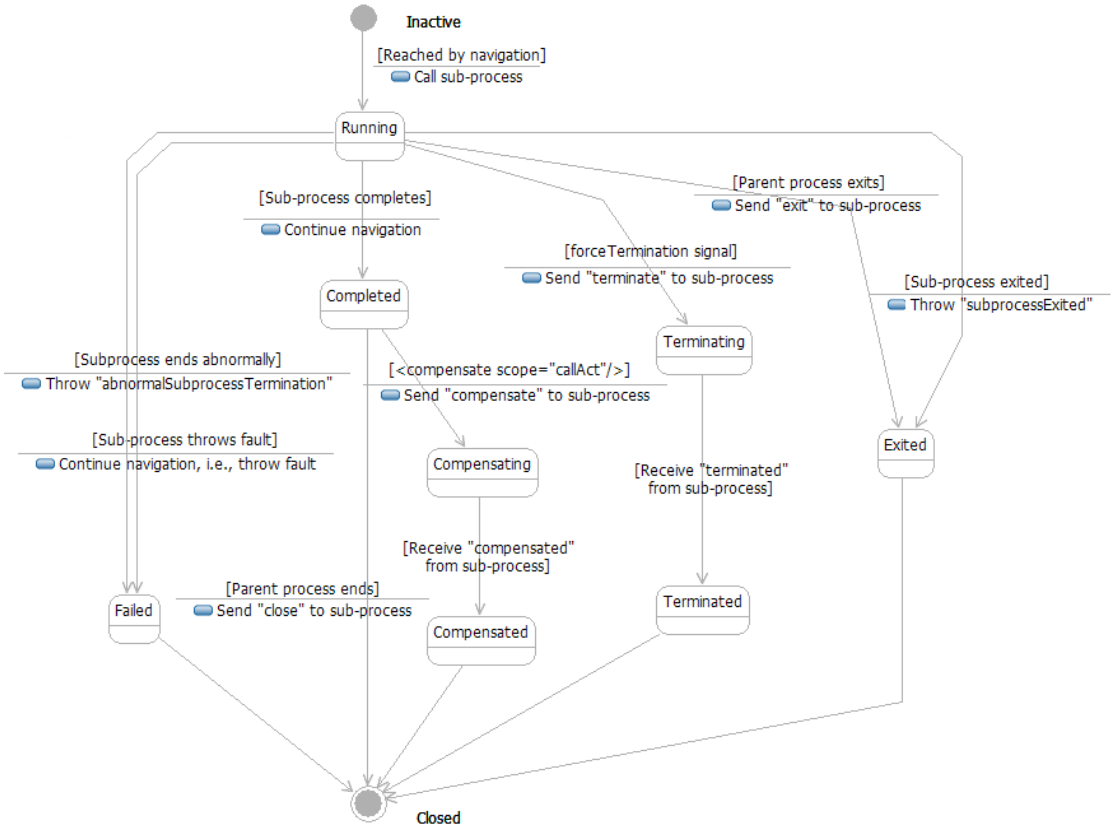


Figure 3. The behavior of the call activity

4.3 State Diagram of the Sub-process

A sub-process has many similarities to a BPEL scope. Just like a BPEL scope, a sub-process can be compensated, terminated, completed either successfully or unsuccessfully, etc. Therefore the behavior of sub-processes is defined in a way that is consistent with the behavior of BPEL scope. However, unlike a BPEL scope, if a sub-process completes without sending a response back to the parent process, it completes unsuccessfully and the new BPEL standard fault `abnormalSubprocess-Termination` is thrown in the parent process. Also, explicit termination of a sub-process is propagated to the parent process throwing the new BPEL standard fault `subprocessExited`.

Figure 4 shows the complete state diagram describing the behavior of the sub-process with respect to the sub-process coordination protocol.

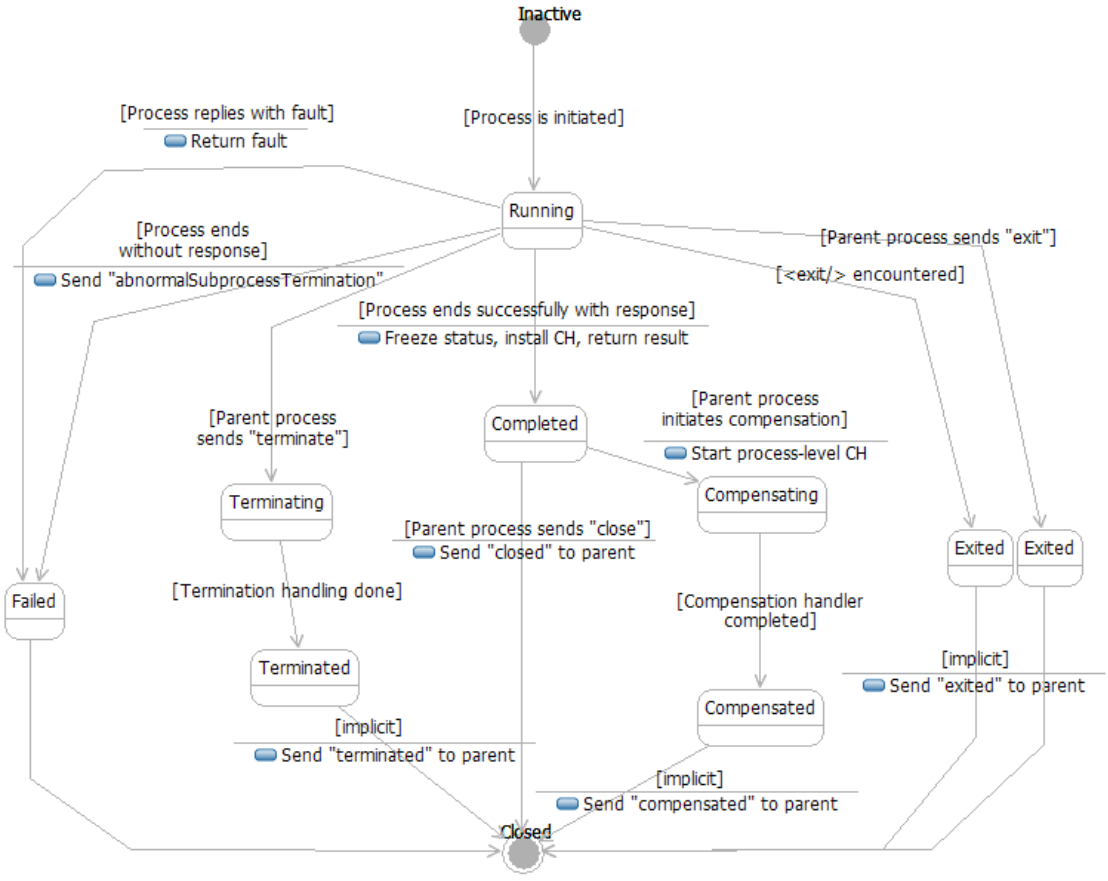


Figure 4. The behavior of the sub-process

5 Summary

The problem of modularization and reuse in the BPEL language has been intensively discussed in different contexts, including the work on the upcoming WS-BPEL standard. The outcomes of those discussions show that there is no consensus on how the problem should be resolved. This paper examines various flavors of sub-processes and different modes of invocation, and outlines the syntax and semantics of needed extension. A language specification, which defines the precise syntax and semantics of this extension, will follow.

6 Glossary

Calling process. Synonym to “parent process”.

Inline sub-process. A process that is defined within another process.

Local sub-process. Standalone sub-processes defined within the same infrastructure and inline sub-processes are subsumed under the term local sub-processes. Local sub-processes are accessed by their qualified name.

Parent process. A process that calls another process as a sub-process.

Process scope. The container of all elements, such as variables, fault handlers, etc., defined on the process level.

Remote sub-process. A standalone sub-process that is managed by another infrastructure and that is accessed as a service.

Standalone sub-process. A standard BPEL process that adheres to the restrictions introduced by the model of sub-process, and that can be called from another process as a sub-process.

Sub-process. A process that is defined either within another process, or as a BPEL process, which is called by another process (the parent process). The sub-process behaves as if it would have been defined as a scope within the parent process, regarding its compensation behavior, error handling, or lifecycle.

7 References

[BPEL4WS 1.1] Business Process Execution Language for Web Services Version 1.1, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, May 2003, available via http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, http://ifr.sap.com/bpel4ws/

[WS-BPEL 2.0] Web Service Business Process Execution Language Version 2.0, Working Draft, July 2005, OASIS Technical Committee, available via http://www.oasis-open.org/committees/wsbpel

[WSDL 1.1] Web Services Description Language (WSDL) Version 1.1, W3C Note, available via http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[XML Schema Part 1] XML Schema Part 1: Structures, W3C Recommendation, October 2004, available via http://www.w3.org/TR/xmlschema-1/

[XML Schema Part 2] XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, available via http://www.w3.org/TR/xmlschema-2/

[XMLSpec] XML Specification, W3C Recommendation, February 1998, available via http://www.w3.org/TR/1998/REC-xml-19980210

[XPATH 1.0] XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, available via http://www.w3.org/TR/1999/REC-xpath-19991116