

Business Process Execution Language for Web Services – BPEL4WS

The Promise of Portable Business Processes

The Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) is a XML based programming language/execution environment intended to enable portable business process definitions for WSDL based business processes. BPEL's goal is to make it possible to write a business process once in BPEL and then run it everywhere.

As a Turing complete language BPEL can do, well, anything. Although BPEL limits itself to features necessary for business processes this still requires an enormous number of capabilities. BPEL currently provides:

- Two full programming models (graph and structured),
- An exception model,
- Three types of storage,
- A threading model,
- A transaction model,
- A process life cycle management model and
- A message handling and routing model

That's a lot of functionality and this is intended to be a relatively short article. As such the article will take a breadth first approach and provide the reader with an overview of BPEL. It will therefore be necessary to leave out a number of features and simplify a number of otherwise complex issues.

The Nature of Standards

As a newly born specification BPEL has many years of work ahead of it before it is sufficiently mature to be used for building portable implementations. In the meantime it is difficult to make categorical statements about BPEL as any one of its numerous open issues could cause significant changes in BPEL's functionality. As such the reader is reminded that this article only represents the views and opinions of the author and not necessarily those of the other BPEL authors.

BPEL Execution Model

The Big Picture

The core of BPEL is the process. A BPEL process is just like any other process providing standard facilities such as storage, threads, faults, etc.

Commands given in a BPEL process are called activities and the definition of a BPEL process consists of exactly one activity. Therefore interesting BPEL processes tend to

start with either the Sequence or Flow activities, both of which are containers of other activities.

A Sequence activity contains a list of activities that are to be executed serially.

A Flow activity contains a list of activities that are to each be executed in their own threads. Threads may be executed in parallel but this isn't guaranteed, the BPEL implementation may choose to serialize execution.

Sequence and Flow can contain any arbitrary set of activities in any order including other Sequence and Flow activities.

The control flow of a BPEL process is specified using an arbitrary combination of two programming models -- structured and graph based.

Structured Programming

Structured programs are written using the Switch and While activities.

Switch defines an ordered list of Boolean conditions, each associated with an activity. The first condition that evaluates to true has its activity executed.

The While activity will loop and execute its associated activity until its Boolean expression evaluates to false.

Creative use of the Switch and While activities can create the computational equivalent of common structured constructs such as if-then-else, for-each, until, etc.

Graph Based Programming

Graph based programming is typically created with a visual programming tool that displays activities on a screen and allows those activities to be connected via links. Links have Boolean expressions assigned to them called transition conditions. Boolean expressions associated with each activity, called join conditions, decide if an activity should execute based on the value of the transition conditions of the links that point at it.

For example, a join condition on a destination activity could be written to specify that the activity should only execute if all the links that point to it evaluate to false.

BPEL supports the graph paradigm by providing for links on all activities. Any activity can be linked to any other activity. This works across threads, structures, etc.

The combination of links, transition conditions and join conditions are computationally equivalent to structured programs.

Boolean Expressions

Boolean expressions in BPEL are defined using XPATH 1.0 by default. BPEL provides a number of XPATH extension functions to enable access to information such as the value

of a particular incoming link. However the use of XPATH is a configurable option and can be overridden.

Containers & Assign

Containers are BPEL's answer to variables. Containers can only be defined at the global level and are globally visible. BPEL supports three types of containers -- message containers, partner containers and service reference containers. Partner containers are mentioned later, service reference containers are not discussed in this article.

Message containers are used to hold SOAP messages. Messages can either be created in a container by the BPEL process or they can be used to receive messages sent to the process. The type of a container is defined by a reference to a WSDL message definition.

The Assign activity is used to move XML between containers of the same type. XML fragments are moved between containers through the use of XPATH expressions with the Assign activity. The Assign activity can also move static XML into containers.

Scopes

BPEL uses scopes to manage faults and compensation handlers. A new scope is declared by using the Scope activity that then contains the various handlers and a single activity.

Faults

BPEL faults are a combination of a unique name and an optional container. The process can throw a number of pre-defined faults or faults can be thrown by the Throw activity.

A scope's fault handlers can catch faults based on the name of the fault, the type of the container associated with the fault or a combination of the two. Otherwise the BPEL fault handler provides typical features such as an optional "catch-all" fault handler, standard scope based escalation for un-caught faults and the ability of fault handlers to re-throw faults.

Compensation

Compensation handlers are a convenience mechanism that associates activities that 'undo' the actions in a scope with the scope.

For example, if a scope causes a reservation to be made at a restaurant then the compensation handler could contain activities that would send a message to cancel the reservation.

Compensation handlers are activated by a call to the Compensate activity which can only be called from inside of a fault handler or a compensation handler.

When a compensation handler is run it must be shown the state of all containers as they were when the associated scope successfully exited. Any changes the compensation

handler makes to the containers will only be visible to the compensation handler and will be discarded when the compensation handler exits.

The compensation handler's potentially unbounded memory requirements and isolated nature are open design issues that will be addressed in future version of BPEL.

Sub-Routines

BPEL does not define an explicit sub-routine or function call model. Rather, everything in BPEL is a web service. Re-usable functionality is packaged as a BPEL process which is then called as a web service by other BPEL processes.

Message Processing

Messages are sent and received using the Receive, Reply, Pick and Invoke activities.

Messages are received by executing the Receive activity that takes a number of arguments including - port type, operation, partner container, correlations and a message container.

Port type and operation refer to the definitions in an associated WSDL file. The partner container holds information about the web service to which the message will be sent. The definition of partners is outside the scope of this article. Correlations are groups of XPATH expressions that are used to uniquely identify a message as part of a conversation. The message container is used to store the message when it arrives.

When the Receive activity is executed the associated thread will stop executing until the specified message is received. If the operation is a request/response then the Reply activity can be used to send back the response.

The Pick activity is used when one of a number of possible incoming messages may be sent. The Pick activity is the message equivalent of a Switch except that the conditions are Receives instead of Boolean expressions. Unlike the Receive activity the Pick activity can have a time out set on it.

The Invoke activity is used to send messages and in the case of solicit/response, receive back a response. Invoke uses the same arguments as Receive but supports two message containers, one to hold the outgoing message and an optional second message container to handle the response. If Invoke should receive back a SOAP fault that fault will be thrown as a BPEL fault with the SOAP message in the container.

Programming Language or Execution Environment?

BPEL's multiple programming models, terse syntax and XML serialization do not lend themselves to direct programming. In other words, BPEL should be compared to the CLR or the JVM rather than to a programming language. It is easy to imagine writing a business process in some language/environment and then 'compiling' the resulting program down into BPEL for execution.

BPEL and Java

BPEL is a procedural language for defining a message router that will enforce the choreography of a business process. But BPEL is not intended for generic data processing, UI generation, etc. Therefore BPEL needs to be supplemented by Java, which is better suited for providing such behavior. At run time BPEL will accept incoming messages and then route them to Java backed web services that will provide the heavy lifting. Java and BPEL form a team that enables the creation of complicated business process in a platform independent manner.

The Future of BPEL

The Web Services Choreography Interface (WSCI) provides a design and functionality that is almost identical to BPEL's. The most significant difference is that WSCI doesn't support graph based programming.

It doesn't take a great leap of imagination to see WSCI and BPEL combining together to form a single standard. The author and his company believe that any official standard should be available on Royalty-Free terms through an open standards process.

There has been a great deal of discussion in the W3C Web Services Activity regarding the formation of a choreography working group that would take WSCI, BPEL and potentially other proposals as input. At the time this article was written nothing final has been decided, though there certainly seems to be momentum towards this as the Web Services Architecture Working Group recommended standardization of choreography.

Conclusion

This article has only touched on the BPEL feature set at the highest possible level of abstraction. It discussed how to:

- process activities in sequence or in parallel,
- program using a structured or graph based paradigm,
- store and manipulate XML values in containers,
- define and nest scopes,
- define, throw and catch faults,
- define and invoke compensation handlers,
- use web services as sub-routines and
- send and receive messages.

The article also touched on BPEL's similarity to the CLR/JVM, BPEL and Java's relationship and BPEL's expected convergence with WSCI to form a single open, Royalty-Free standard.

Issues not discussed in this article include: BPEL's WSDL extensions (properties, property aliases and service link types), properties in containers, serializable scopes, Wait activity, Empty activity, Terminate activity, partner definitions, service references, default compensation and fault handlers, start activities and abstract processes.

In the years to come as BPEL is more fully developed and standardized, the web community can look forward to a robust standard for writing portable business process definitions.

References

BPEL4WS: <http://dev2dev.bea.com/techtrack/BPEL4WS.jsp>

WSCI: <http://www.w3.org/TR/wsci/>

About the Author

Yaron Y. Goland is a principal technologist in the office of the CTO at BEA with primary responsibility for business process standards. Mr. Goland has spent the last eight years working on standards and related technologies. While at Microsoft Mr. Goland was the program manager for network interfaces for IE 4, representative to the IETF HTTP working group, lead author on the WebDAV specification and architect for UPnP's transport protocols. Yaron represented Openwave at the Liberty Alliance Project and is BEA's representative for WSCI and BPEL4WS.

Acknowledgements

Thanks to Bill Cox, David Orchard and Ed Cobb for their constructive comments.