

## Web Services

# Describe business process activities as Web services

## Add WSBPEL to your enterprise toolbox

### Summary

This article is part of a series of short articles that introduce readers to the industry's various Web services standards. These articles provide a quick introduction to a standard, its background, underlying architecture, benefits, status, and industry adoption. As some of the content might be a depiction of the authors' viewpoints, readers are encouraged to refer to the links provided in [Resources](#) to gain a deeper understanding of a particular standard. This article focuses on the Web Services Business Process Execution Language standard being developed by OASIS. (2,800 words; **October 31, 2005**)

By **Ash Parikh, Vivek Kondur, and Premal Parikh**

---

**W**eb Services Business Process Execution Language, WSBPEL, is an XML-based process/workflow-definition execution language. WSBPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web services interfaces. WSBPEL also defines how multiple service interactions with these partners are coordinated to achieve a business goal as well as the state and the logic necessary for this coordination. WSBPEL, previously called Business Process Execution Language for Web Services (BPEL4WS), is often simply referred to as BPEL.

Business processes defined in WSBPEL can be modeled and re-engineered using numerous BPEL modeling tools to generate a BPEL file, which can then be deployed and executed in a runtime environment known as a native BPEL engine.

### What relevant standards bodies pertain to WSBPEL?

WSBPEL evolved from the convergence of ideas and concepts such as Microsoft's XLang and IBM's WSFL (Web Services Flow Language). WSBPEL is now a part of the Organization for the Advancement of Structured Information Standards (OASIS) and has industry support from many vendors such as BEA Systems, Oracle, and IBM. The WSBPEL specification mainly addresses service orchestration, where business processes are created using the services of the trading partners or stakeholders in the process.

### What is the status and industry adoption of WSBPEL?

The current version of WSBPEL is 1.1; the 2.0 version, a committee draft dated September 1, 2005, is being defined by the OASIS WSBPEL Technical Committee. WSBPEL is backed by major IT players and software vendors such as Adobe Systems, Hewlett-Packard, IBM, Microsoft, Tibco Software, Sun Microsystems, and SAP, to name a few, and has also been adopted by the industry as the popular language for defining Web services-based business processes. Currently, numerous design-time and runtime products, such as WSBPEL-compliant modeling tools, editors, and process engines, are available from major software vendors.

## What standards enable WSBPEL?

WSBPEL extends support to existing Web services standards to achieve universal interoperability between applications. As we know, Web services are built on a loosely-coupled integration model to allow the flexible integration of heterogeneous systems in a variety of domains including business-to-consumer, business-to-business, and enterprise application integration.

WSBPEL leverages several XML-based specifications pertaining to the Web services world such as:

- WSDL (Web Services Description Language) 1.1
- XML Schema 1.0
- Web Services Addressing
- XPath 1.0

WSDL messages and XML Schema type definitions provide the data model used by WSBPEL processes. XML Schema 1.0 is also used as the interface for validating all the process definitions generated using any of the available BPEL modeling tools and editors. WSBPEL's dependency on WS-Addressing avoids the invention of a private WSBPEL mechanism for Web services endpoint references—such references are obviously a general requirement in Web services usage. WSBPEL 1.1 provides extensibility to XPath 1.0, which supports data manipulation and can also accommodate future versions of these standards, specifically with regards to XPath and the related standards used in XML computation.

## How Web services enable business process orchestration

Web services is a key component that enables services-based business process orchestration. WSDL has the most influence on WSBPEL, as its process model is layered on top of the service model defined by WSDL 1.1. At the core of the WSBPEL process model is the notion of peer-to-peer interaction between the services described in WSDL; both the process and its partners are modeled as WSDL services. A business process defines how to coordinate the interactions between a process instance and its partners. In this sense, a WSBPEL process definition provides and/or uses one or more WSDL services and also provides the description of the behavior and interactions of a process instance relative to its partners and resources through Web services interfaces.

In particular, a WSBPEL process represents all partners and interactions between these partners in terms of abstract WSDL interfaces (`portTypes` and operations); no references are made to the actual services used by a process instance.

## Using WSBPEL

Now that we have a background on WSBPEL, let's try to understand its various constructs and how we can use them to define a meaningful business process. WSBPEL resembles conventional workflow management systems and follows a similar approach. WSBPEL activities have been designed to allow for most forms of process modeling and facilitate the mapping of graphical workflow modeling tools to BPEL. Presently, the activities in WSBPEL 1.1 can be divided into basic, or unstructured, activities and structured activities. In this article, we provide only a brief overview of each activity.

**Table 1. Basic activities**

<receive>	Blocks until a message is received.
<reply>	Sends a message in response to a received message. Therefore, the

	combination of a <receive> and a <reply> forms a request-response operation on the WSDL portType for the process.
<invoke>	Invokes an operation on a portType provided by one of the collaboration partners.
<assign>	Construct can be used to update the values of variables with new data.
<throw>	Raises a fault for a fault handler to catch within the defined scope for an activity.
<terminate>	Ends the business process explicitly.
<wait>	Suspends execution for a given period of time or until certain time has passed.
<empty>	No-operation used for synchronization purposes.

**Table 2. Structured activities**

<scope>	Defines a block of activities.
<sequence>	Executes a set of activities in an order; i.e., one after another.
<flow>	Executes one or more activities concurrently; i.e., parallel processing.
<while>	Repeats an activity depending on certain conditions.
<switch>	Based on the condition, it chooses between a set of activities to execute.
<pick>	Blocks and waits for a suitable message to arrive or for a time-out alarm to go off.
<compensate>	Defines the set of activities that need to be compensated.

## How XQuery is useful in the extension and implementation of WSBPEL

WSBPEL, being described in XML syntax, can thus be treated as an XML document. An implementation of the WSBPEL specification showcases several instances of running business processes, which, in turn, necessitate management of those instances and retrieval of meaningful information. XQuery, which is the de facto query language for XML, provides a rich way of querying a WSBPEL document and retrieving information from the document—for example, finding all trading partners involved in a process and querying across partner management applications to locate Collaboration Protocol Profile and Agreement (CPPA) agreements for a request validation. We propose using an XML database platform that includes an XQuery engine to provide the technology for this logical approach of storing and managing process instance documents.

## Process lifecycle management with a native XML database

As one can imagine, in any real-world business process scenario, many process versions can be active at runtime. This means, for the purpose of maintenance and housekeeping, all production processes require persistence and querying on all copies of running processes. Also, some state management is needed to handle the interaction among activities. In addition, within a BPEL process, various versions of a Web service can be described.

A native XML database can provide persistence for all versions of processes and Web services, and also define the right processes for the right kind of applications at runtime using schema versioning and validation features of the database. The easiest way to achieve schema versioning using a native XML database is to tag the process document as the LATEST version at the time of deployment. So, whenever a new request is received for that process, the engine will create an instance from the LATEST tagged process.

To provide more flexibility, a request should carry versioning information about the process so the engine can create an instance of the process for that specific version. Also, a process can be marked as

obsolete or dead when not in use anymore. If one prefers to achieve this functionality by updating an XML document at the node level, an XQuery engine that enables node-level updates may provide a better alternative.

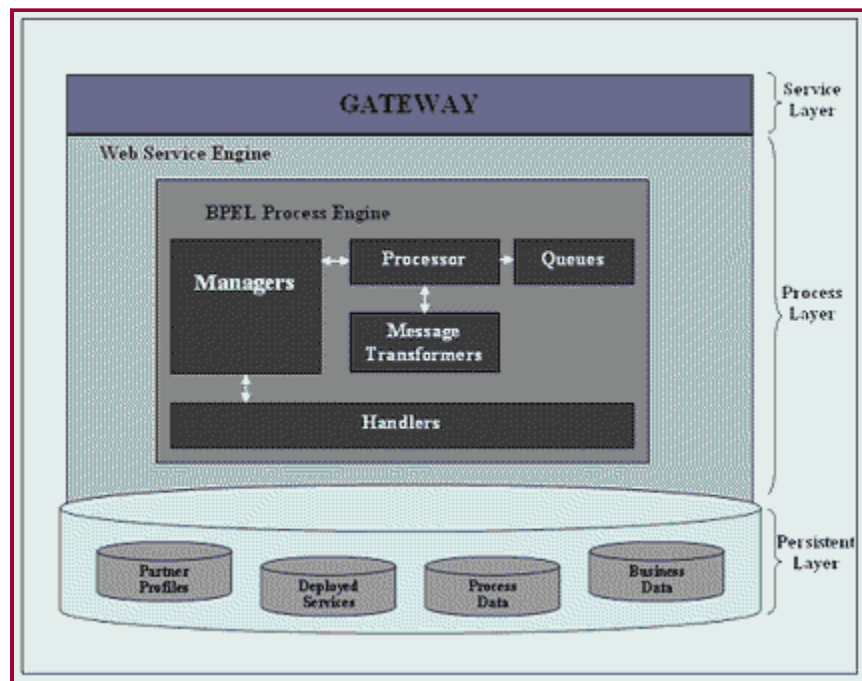
## WSBPEL process analysis

As described earlier, an XML persistence mechanism is important for process lifecycle management when combined with XQuery. Another advantage of having XML persistence for business processes is the use of an XML data management system for business process analysis, which can be important in real-world business scenarios. WSBPEL specs do not provide a clear way for any optimization mechanism based on past execution. It is thus advantageous to have historical or persisted data for the processes, either running or completed, to enable process optimization. Furthermore, various reports and key process indicators can be generated from the persisted data and metadata used during process execution, such as:

- Number of running/completed instances of processes per week
- Number of errors for specific endpoints or partner links for a process
- Execution time for a specific process over a period of time
- Size of data exchanged per endpoint
- Availability or uptime of partner links (uptime)
- Average wait time for asynchronous endpoints

## BPEL engine architecture

A typical BPEL engine from our viewpoint can be depicted as shown in the figure below:



**A typical BPEL engine architecture. Click on thumbnail to view full-sized image.**

In summary, a typical BPEL engine services requests pertaining to processes compliant to WSBPEL 1.1. The engine implementation could extend any Web services engine/SOAP engine. The architecture's key components are the gateway, process engine, message transformers, queues, handlers, and managers for various functions within the engine and a native XML database. The BPEL engine creates process instances and executes activities as per their language definitions to achieve a business result for every

request. The execution contains the business process definition, WSDL(s), and engine-specific deployment descriptor files.

## Salient features of WSBPEL

WSBPEL 1.1 clearly speaks about its capability of defining the business process, which, simply put, is the orchestration of Web services in a sequential or a structured way. WSBPEL addresses some key concepts such as:

- **Partners:** Most enterprise-scale business processes are composed of multiple partners distributed geographically. WSBPEL 1.1 defines grammar for all the interactions between the business process and its partners. Each partner interaction occurs through Web services interfaces, and the structure of the relationship at the interface level is encapsulated in what we call a partner link. The services with which a business process interacts are modeled as partner links in WSBPEL 1.1. The notion of a partner link type reflects a peer-to-peer relation between the process and each service the process calls.
- **Correlations:** Correlations are necessary for identifying a particular business process instance. In WSBPEL 1.1, correlation is achieved by a named group of properties that define a way to identify an application-level conversation within a process instance.
- **Long-lived transactions:** Most business processes run for long durations and can encompass asynchronous messaging or even involve human interactions. WSBPEL 1.1 doesn't exactly support ACID (atomicity, consistency, isolation, durability) types of transactions, but provides the ability for flexible control of reversal. Defining fault and compensation handlers help achieve support for ACID transactions in an application-specific manner, which results in a feature called long-running (business) transactions (LRTs).

LRTs, as defined in WSBPEL 1.1, describe the business processes as purely within a platform-specific implementation. There is no distributed coordination regarding an agreed-upon outcome among multiple-participant services. The achievement of distributed agreement is an orthogonal problem outside the scope of WSBPEL, to be solved using the protocols described in the Web Services Atomic Transaction, WS-Coordination, and WS Business Activity Framework specifications.

- **Compensating:** As mentioned earlier, a business process runs for long durations, and multiple transactions must be handled since locks and isolations cannot be maintained for long periods during which technical or business errors or faults occur within the business process instance. As a result, the overall business transactions can fail or be cancelled after many ACID transactions have been committed during its progress. For the process to reach a global stabilized state, failure or cancellation must be undone within a process.

WSBPEL 1.1 enables compensation by providing application-specific *compensation handlers*. Compensation handlers are defined within the business process, which tries to reverse the effects based on faults/events of a previous activity. Compensation handlers are scoped, meaning each one is associated with various local scopes rather than with the process as a whole.

- **Fault handling:** Fault might occur anytime during the execution of the business process. A fault can be the manifestation of a system error, result from the nonavailability of a Web service due to network instability, or represent an application-specific fault or runtime fault.

WSBPEL 1.1 enables fault handling by providing *fault handlers*. WSBPEL 1.1 supports business faults and runtime faults. Business faults are application specific and occur either when the process engine executes an explicitly defined `<throw>` activity or when an `<invoke>` activity receives a fault in response. Runtime faults are not user defined and do not appear in the WSDL for a process or service. WSBPEL 1.1 defines 10 standard faults: `selectionFailure`,

conflictingReceive, conflictingRequest, mismatchedAssignmentFailure, joinFailure, forcedTermination, correlationViolation, uninitializedVariable, repeatedCompensation, and invalidReply. Fault handlers are also scoped, meaning a fault is caught by a particular activity. In case the fault is not locally caught, it is thrown to the scope that encloses the activity.

- **Event handling:** Event handling is a feature necessary in any process definition language and has been addressed in WSBPEL. Presently, WSBPEL 1.1 handles two types of events: alarm events and message events. Events can be alarms that go off after user-defined times. An alarm event occurs at most once while the corresponding scope is active. Events can also be incoming messages that correspond to a request/response or one-way operation in WSDL. A message event occurs when the appropriate message is received on the specified partner link using the specified port type and operation. When such an event occurs, the corresponding activity is carried out.

## Limitations

The following list details some of the shortcomings that we, the authors, see in WSBPEL 1.1:

1. The dynamic discovery of partners and binding at runtime is not possible, though various sophisticated BPEL engines integrate with various UDDI/ebXML registries to discover partner processes and services. According to the specification, this is intentional so as to leave the binding up to specific implementations.
2. Dynamic parallel processing is currently not supported. WSBPEL 1.1 only supports the invocation of a single Web service within an invoke activity. In many situations, a particular invoke activity must result in the creation of many activity instances, where the number of instances is unknown at design time, but calculated at runtime from the contents of a set of data or references and issue number.
3. Message mediation functionalities such as transformation, validation, and reliability are currently not addressed.
4. As discussed before, presently, support is unavailable for handling transactions within a business process that is distributed or spans multiple vendors and platforms.
5. No persistence mechanism is specified for measurement, reporting, or management.

We encourage readers to visit the monthly archives link in [Resources](#) for an up-to-date understanding of the issues being discussed in the technical committee.

## Ideas to enhance WSBPEL

We propose these answers to the above mentioned limitations in WSBPEL 1.1:


1. As we know, WSBPEL 1.1 supports business partners using its own model and grammar. We suggest the use of ebXML standards such as the Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA) for partner management within the given scope of WSBPEL. CPP is one of the specifications from ebXML that describes an organization profile. Such a profile contains various information related to the partner like:
  - The organization's name and contact information
  - Different protocols used by the organization, such as transport layer, message layer, and security layer
  - The endpoint URLs of the partner's services, timeouts, and certificates
2. As seen in the figure above, for a typical WSBPEL engine implementation, we can use native XML

databases for various types of persistence such as business-related data, ebXML Registry services for partner management, and BPEL(s) and WSDL(s), and hence repurpose the database as a service registry and repository.

3. The BPEL engine implementation could also provide a pluggable extension for XML query languages such as XQuery, in addition to existing support for XPath as the expression language in WSBPEL.
4. Since most of the business processes are stateful and long-running, a need exists for some kind of measurement, in addition to reporting and management, for forecasting business process metrics. We could achieve this by including a few more attributes such as `activityState`, `startTime`, and `endTime` under the existing "standard-attributes" list. In this way, every activity would contain metadata on state management, which could be used for metric measurements and alerts.
5. The drawbacks on transactions and messaging mentioned earlier in the limitations section could be resolved by looking into some of the WS-\* specifications such as WS-AtomicTransaction, WS-BusinessActivity, WS-Coordination, WS-Addressing, WS-ReliableMessaging, and WS-Policy.

## Conclusion

The growing adoption of Web services and service-oriented architectures in real-world business scenarios makes WSBPEL a great choice as a business process language for enterprise application integrations. To enable more rapid adoption, the authors recommend that WSBPEL tools and engines take advantage of the XML nature of WSBPEL by providing native XML persistence and querying of WSBPEL processes and services.

Once again, as some parts of this article depict the authors' ideas and perspectives, we encourage readers to get current information on the specification from the WSBPEL Technical Committee page. 

## About the author

**Ash Parikh** is the director of technology and development for the Enterprise Applications Group at Raining Data Corporation. He is a named expert in the field of SOA and distributed computing and has presented and authored abstracts for OASIS Symposium 2005, Delphi BPX Summit 2004, Delphi Enterprise On-Demand 2004, JavaOne 2004, JavaOne 2003, BEA e-World 2002, and JavaOne 2002. Parikh has more than 15 years of IT experience and is an active member on a number of Java Specification Requests in the Java Community Process and in OASIS technical committees. He is also the president of the Bay Area Chapter of the Worldwide Institute of Software Architects and the co-chair of the SDForum Web services SIG. Parikh has also authored several technical articles in journals such as *JavaWorld*, *XML-Journal*, *Java Pro*, *Web Services Journal*, *ADTmag*, *Softwaremag.com*, and *Java Skyline*.

**Vivek Kondur** is a senior developer with SourceN. He has been designing and developing SOA solutions for more than two years. His expertise and enthusiasm in the domain has made him an effective contributor to the Web services community.

**Premal Parikh** is a lead architect/team lead with the Enterprise Applications Group at Raining Data Corporation. He has more than 10 years of experience in the software industry, which includes design and architecting enterprise products, domain-specific solutions, and portals for the B2B marketplace. He has authored several articles in technical journals. Parikh is also an active member on a number of OASIS Web services standards technical committees.

## Resources

- Business Process Execution Language for Web Services Version 1.1:  
[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)

- XML Schema Part 1:  
<http://www.w3.org/TR/xmlschema-1/>
- XML Path Language (XPath) Version 1.0:  
<http://www.w3.org/TR/xpath>
- Web Services Description Language (WSDL) Version 1.1:  
<http://www.w3.org/TR/wsdl>
- Simple Object Access Protocol (SOAP) 1.2:  
<http://www.w3.org/TR/soap/>
- Transaction Specifications: WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity:  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsatspecindex.asp>
- WS-Addressing:  
<http://www.w3.org/Submission/ws-addressing/>
- WS-ReliableMessaging:  
<http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-ReliableMessaging.pdf>
- WS-Policy:  
<http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-policy.asp>
- ebXML Registry Services:  
<http://www.ebxml.org/specs/ebRS.pdf>
- Web Service Modeling Execution environment:  
<http://www.wsmx.org/>
- ActiveBPEL, an open source BPEL engine:  
<http://www.activebpel.org/>
- Apache Agila:  
<http://wiki.apache.org/agila/>
- Monthly archives for WSBPEL:  
<http://lists.oasis-open.org/archives/wsbpel/>
- For more articles on Web services, see the following resources:
  - The **Java and Web Services** section of *JavaWorld's* Topical Index:  
[http://www.javaworld.com/channel\\_content/jw-webserv-index.shtml](http://www.javaworld.com/channel_content/jw-webserv-index.shtml)
  - *JavaWorld's* **Web Services** column:  
<http://www.javaworld.com/columns/jw-web-services-index.shtml>
- Also browse the **Java and XML** section of *JavaWorld's* Topical Index:  
[http://www.javaworld.com/channel\\_content/jw-xml-index.shtml](http://www.javaworld.com/channel_content/jw-xml-index.shtml)

