IBM

Business Process with BPEL4WS: Understanding BPEL4WS, Part 1

Concepts in business processes

Level: Introductory

Sanjiva Weerawarana (sanjiva@us.ibm.com), Research Staff Member, IBM TJ Watson Research Center
Francisco Curbera (curbera@us.ibm.com), Research Staff Member, IBM TJ Watson Research Center

01 Aug 2002

The recently released Business Process Execution Language for Web Services (BPEL4WS) specification is positioned to become the Web services standard for composition. It allows you to create complex processes by creating and wiring together different activities that can, for example, perform Web services invocations, manipulate data, throw faults, or terminate a process. These activities may be nested within structured activities that define how they may be run, such as in sequence, or in parallel, or depending on certain conditions. This series of articles aims to give readers an understanding of the different components of the language, and teach them how to create their own complete processes. The first part of the series will take readers through creating their first simple process. Subsequent parts will extend the example in different ways to illustrate and explain the key parts of the language, including data manipulation, correlation, fault handling, compensation, and the different structured activities in BPEL4WS.

Introduction

Today Web services can communicate with each other, advertise themselves, and be discovered and invoked using industry-wide specifications. However, until last week, linking these services together into a business process or a composition gave the user a number of conflicting specifications to choose from -- as was the case with WSFL from IBM and XLANG from Microsoft. The Business Process Execution Language for Web Services (BPEL4WS) represents the merging of WSFL and XLANG, and with luck, will become the basis of a standard for Web service composition. BPEL4WS combines the best of both WSFL (support for graph oriented processes) and XLANG (structural constructs for processes) into one cohesive package that supports the implementation of any kind of business process in a very natural manner. In addition to being an implementation language, BPEL4WS can be used to describe the interfaces of business processes as well -- using the notion of abstract processes. We will elaborate further on this is future articles.

An initial implementation of BPEL4WS called BPWS4J was also released by IBM via alphaWorks (see Resources). This implementation can serve as a testbed for users to learn and experiment with BPEL4WS.

In this article we discuss the fundamental concepts of BPEL4WS.

BPEL4WS Concepts

BPEL4WS supports two distinct usage scenarios:

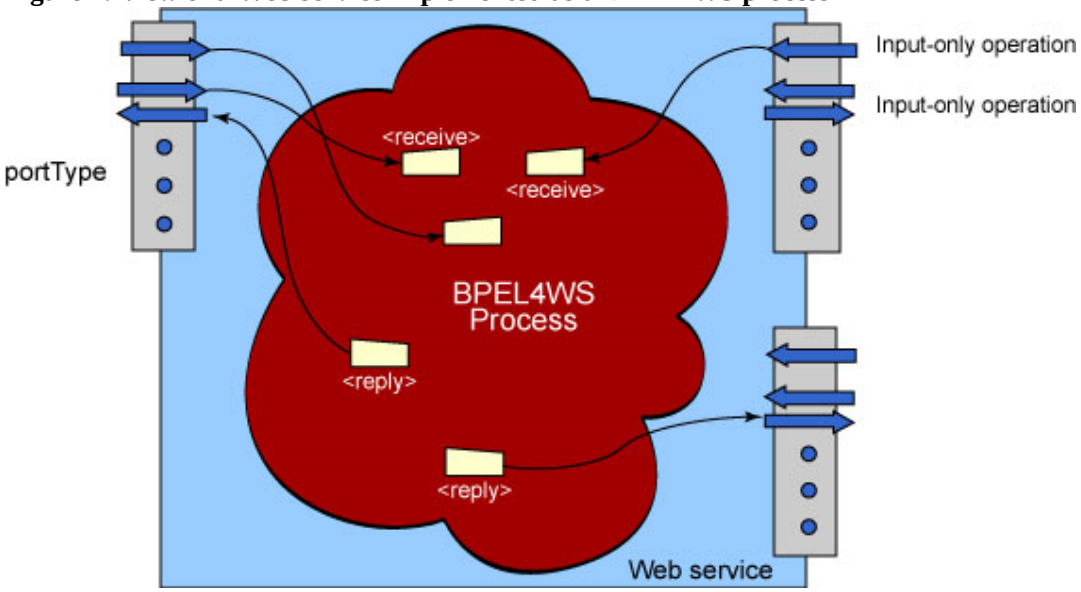
- Implementing executable business processes.
- Describing non-executable abstract processes.

This article focuses on the first scenario only; a future article in this series will consider the abstract process concepts of BPEL4WS.

As an executable process implementation language, the role of BPEL4WS is to define a new Web service by composing a set of existing services.Thus, BPEL4WS is basically a language to implement such a composition. The interface of the composite service is described as a collection of WSDL portTypes, just like any other Web service. The composition (called the *process*) indicates how the service interface fits into the overall execution of the composition. Figure 1 illustrates this outer

view of a BPEL4WS process.

Figure 1. View of a Web service implemented as a BPEL4WS process



The composition primitives found in BPEL4WS come primarily from many years of experience in workflow and business process integration, hence its positioning as a business process composition language.

Implementing the service

What's in the cloud in the figure above? Unlike a traditional programming language implementation of a WSDL service, each operation of each portType does not map to a separate piece of logic in BPEL4WS. Instead, the entire type of the service (that is, the set of portTypes of the service) is implemented by one single BPEL4WS process. Thus, specific "entry-points" corresponding to external users invoking the operations of the interface are indicated within the BPEL4WS description. These entry points either consume WSDL operations' incoming messages from input-only or input-output operations. In the latter case, the process must also indicate where the output message is generated. BPEL4WS only uses and supports input-only and input-output (request-response) operations of WSDL; output-only (notification) and output-input (solicit-response) operations are not required nor supported.

The BPEL4WS process itself is basically a flow-chart like expression of an algorithm. Each step in the process is called an activity. There are a collection of primitive activities: invoking an operation on some Web service (`<invoke>`), waiting for a message to operation of the service's interface to be invoked by someone externally (`<receive>`), generating the response of an input/output operation (`<reply>`), waiting for some time (`<wait>`), copying data from one place to another (`<assign>`), indicating that something went wrong (`<throw>`), terminating the entire service instance (`<terminate>`), or doing nothing (`<empty>`).

These primitive activities can combined into more complex algorithms using any of the structure activities provided in the language. These are the ability to define an ordered sequence of steps (`<sequence>`), the ability to have branching using the now common "case-statement" approach (`<switch>`), the ability to define a loop (`<while>`), the ability to execute one of several alternative paths (`<pick>`), and finally the ability to indicate that a collection of steps should be executed in parallel (`<flow>`). Within activities executing in parallel, one can indicate execution order constraints by using the *links*.

BPEL4WS allows you to recursively combine the structured activities to express arbitrarily complex algorithms that represent the implementation of the service.

Interacting with others: Partners

As a language for composing together a set of services into a new service, BPEL4WS processes mainly consist of making invocations to other services and/or receiving invocations from *clients* (the users of the service in Figure 1). The prior is done using the `<invoke>` activity and the latter using the `<receive>` and `<reply>` activities. BPEL4WS calls these other services that interact with a process *partner*. Thus, a partner is either a service the process invokes (*invoked partners*) as an integral part of its algorithm, or those that invoke the process (*client partners*).

The first kind of partners is obvious -- the process must clearly invoke other services to do things. The `<invoke>` activity indicates the partner to invoke and what operation of which of the partner's `portTypes` to invoke on that partner. Notice, however, that invoked partners may end up being clients as well -- it may be the case that the process invokes an operation on the partner to request some service. Later on, the partner may invoke an operation on the process to provide the desired data.

The reason for treating clients of the process as partners may not be so obvious. There are actually two reasons for it: the first is that in sometimes the process may need to invoke operations on one of its client partners. This is primarily how asynchronous interaction is supported: the client invokes an operation on the process to request some service. Upon completion, the process invokes an operation on the client partner. At that point, there is no distinction between a client partner and an invoked partner.

The second reason is that the service offered by the process may be used (wholly on in parts) by more than one client. In addition, the process may wish to distinguish between these different users of the service. For example, a process representing a loan servicing system offers a single Web service, but only parts of it are accessible to the customer applying for the loan, other parts for the customer service representative and finally the entire service to the loan underwriters. Depending on whether an operation is invoked by the customer or by the underwriter, the returned behavior may be quite different. Furthermore, the approach of using partners to model clients allows the process to indicate that certain operations may only be invoked by certain clients.

So, partners are one of the following:

- services that the process invokes only.
- services that invoke the process only.
- or services that the process invokes and invoke the process (where either may occur first).

The first two are straightforward *invoked partners* and *client partners*, respectively. Consider the relationship between the process and the service for the third case when the process invokes the service first. That means that the service provides (or publishes) a portType (PT1) and the process invokes an operation of that portType. Also, the process must provide a portType (PT2) that the service invokes an operation out of. Thus, from the point of view of the process, the process requires the portType PT1 from the service and provides the portType PT2 to the service. Looking at the same relationship from the point of view of the service leads to opposite statement: The service offers the portType PT1 to the process and requires the portType PT2 from the process. The situation is the same whether the process invokes the service first or vice-versa.

Service link types
Modeling the third kind of partners is what gives rise to *service link types*. Instead of defining the relationship between the service and the process from the point of view of one of these participants, a service link type represents a third party declaration of a relationship between two (or more potentially) services. A service link type defines a collection of roles, where each role indicates a list of portTypes. The idea is that when two services interact with each other, the service link type is a declaration of how they interact -- essentially what each party offers.

BPEL4WS uses service link types to define partners. Basically, a partner is defined by giving it a name and then indicating the name of a service link type and identifying the role that the process will play from that service link type and the role that the partner will play. In the pure invoked partner and pure client partner cases, the service link type will have just one role in it and, hence, only one is indicated at partner definition time. The partner name is then used in `<receive>`, `<reply>` and `<invoke>` activities to indicate the desired partner.

Service references
How does a partner work at runtime? In order for it to work at runtime, the partner must resolve to an actual Web service. Thus, a partner is really eventually just a typed *service reference*, where the typing comes from the service link type and the roles. The BPEL4WS process itself does not indicate how a partner is bound to a specific service; that is considered a deployment time or runtime binding step that must be supported by the BPEL4WS implementation.

Dealing with problems

Developers need ways to handle and recover from errors in business processes. BPEL4WS has exceptions (faults) built into the language via the `<throw>` and `<catch>` constructs. The fault concept on BPEL4WS is directly related to the fault concept on WSDL and in fact builds on it.

In addition, BPEL4WS supports the notion of *compensation*, which is a technique for allowing the process designer to implement compensating actions for certain irreversible actions. For example, imagine a travel reservation process. Once a

reservation has been confirmed, one must perform some explicit operations to cancel that reservation. Those actions are called "compensating actions" for the original action.

Fault handling and compensating is supported recursively in BPEL4WS by introducing the notion of a scope, which is essentially the unit of fault handling and/or compensation. Understanding compensation and fault handling in detail is a topic on its own and will be covered in a future article.

Lifecycle of services

What about the lifecycle of these services? Web services implemented as BPEL4WS processes have an instanced life cycle model. That is, a client of these services always interacts with a specific instance of the service (process). So how does the client create an instance of the service?

Unlike traditional distributed object systems, in BPELWS instances are not created via a factory pattern. Instead, instances in BPEL4WS are created implicitly when messages arrive for the service. That is, instances are identified not by an explicit "instance ID" concept, but by some key fields within data messages. For example, if the process represents an order fulfillment system, the invoice number could be the "key" field to identify the specific instance involved with the interaction. Thus, if a matching instance isn't available when a message arrives at a "startable" point in the process, a new instance is automatically created and associated with the key data found in the message. Messages can only be accepted at non-startable points in a process after a suitable instance has been located; that is, in these cases the messages are in fact always delivered to specific instances. In BPEL4WS, the process of finding a suitable instance or creating one if necessary is called *message correlation*. Message correlation will also be covered in a future article.

Conclusion

In this article we briefly explained the main underlying concepts of BPEL4WS. We considered the overall view of what BPEL4WS is about and then about partners, faults, compensation, and lifecycle. In the future articles of this series we expect to discuss various specific aspects of BPEL4WS in detail.

[Editors note: This column will alternate with each issue between the series on concepts behind BPEL4WS, and that on how to technically implement them. The next issue will introduce technical implementation.]

Resources

- Participate in the discussion forum.
- Please note that this articles refers to version 1.0 of the BPEL4WS specification. The latest version, BPEL4WS1.1, is now available, and an article describing the key differences between the two specifications will be available shortly.
- Check out the second installment of this column, Business processes, Part 2.
- Download the Business Processes for Web Services Java Runtime from alphaWorks
- Read the Business Process Execution Language for Web services specification
- Read these related articles: Automating Business processes and transactions in Web services and Business Processes in a Web services World

About the authors

Sanjiva Weerawarana is a research staff member in the Component Systems group at IBM T.J. Watson Research Center. He is co-author of the BPEL4WS, WSDL and WSFL specifications, and co-developer of BPWS4J, Apache SOAP, WSTK, WSDL Toolkit, WSIF, and WSGW. He received a PhD in Computer Science from Purdue University. You can contact the author at

sanjiva@us.ibm.com.

Francisco Curbera is a research staff member and the manager of the Component Systems group at IBM T.J. Watson Research Center. He is co-author of the BPEL4WS, WSDL and WSFL specifications, and co-developer of BPWS4J, Apache SOAP and WSTK. He received a PhD in Applied Mathematics from Columbia University. Contact him at curbera@us.ibm.com,