BPEL and Java

■ by Matjaz Juric

April 2005

[Discuss this Article]

Introduction

The idea and motivation behind almost each new technology and platform for enterprise application development is to provide an environment where better business applications can be developed with less effort –business applications which should closely align to the business processes, which should not be too complex, and which can be adapted to the changing nature of business processes without too much effort.

Java has provided an excellent platform for developing such applications, but business applications today cannot be isolated. Within companies, business applications have to interoperate and integrate - so they have between companies. Integrating different applications has always been a difficult task for various functional and technology related reasons.

The most recent answer to the integration challenge is the Service Oriented Architecture (SOA) and the web services technologies. The bottom-up view of the SOA sees different business applications exposing their functionalities through web services. Thus we can now access different functionalities of different legacy and new developed applications in a standard way (through web services). Such access to functionalities is important because typical companies have a large number of existing applications which have to be integrated.

Developing the web services and exposing the functionalities is not sufficient. We also need a way to compose these functionalities in the right order – a way to define business processes which will make use of the exposed functionalities. We would obviously prefer a relatively simple and straightforward way to define such processes, particularly because we know that business processes change often, therefore we would like to modify them easily.

This is where the BPEL (Business Process Execution Language for Web Services, also WS-BPEL or BPEL4WS) becomes important. BPEL allows composition of web services and is thus the top-down approach to SOA – the process oriented approach to SOA.

In this article we will discuss the role of BPEL and its relationship with Java. We will concentrate particularly on the idea of extending BPEL, to be able to compose resources other than web services (for example EJBs, JMS, etc.) on one hand, and the possibility to mix BPEL and Java code on the other hand to open up some interesting new perspectives.

Role of BPEL

The process-oriented approach to SOA requires a language for relatively simple description of how web services should be composed into business processes. Of course it would be great if such descriptions could also be executed, which would allow us not only to define abstract process definitions, but to write exact executable specifications of processes. BPEL is such a language. Actually it is the first language which:

1. Allows us to define abstract and executable processes
2. Is supported by the majority of companies
3. Software exists (from several vendors) on which such processes can be executed (BPEL servers) and developed (BPEL designers).

Before we have a more in-depth look at BPEL, let us discuss how web services can be composed. There are two ways: orchestration and choreography. In orchestration, a central process takes control over the involved web services and coordinates the execution of different operations on the web services involved in the operation. This is done as per the requirements of the orchestration. The involved web services do not know (and do not need to know) that they are involved into a composition and that they are a part of a higher business process. Only the central coordinator of the orchestration knows this, so the orchestration is centralized with explicit definitions of operations and the order of invocation of web services.

Choreography on the other hand does not rely on a central coordinator. Rather, each web service involved in the choreography knows exactly when to execute its operations and whom to interact with. Choreography is a collaborative effort focused on exchange of messages. All participants of the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.

From the perspective of composing web services to execute business processes, orchestration is the more flexible approach compared to choreography:

- We know exactly who is responsible for the execution of the whole business process.
- We can incorporate web services, even those that are not aware that they are a part of a business process.
- We can also provide alternative scenarios when faults occur.

BPEL follows the orchestration paradigm. Choreography is covered by other standards, such as WSCI (Web Services choreography Interface) and WS-CDL (Web Services Choreography Description Language). Choreography has not gained support from the industry which would be comparable to BPEL.

The first version of BPEL has been developed in August 2002 by BEA, IBM, and Microsoft. Since then the majority of vendors have joined which has resulted in several modifications and improvements and adoption of version 1.1 in March 2003. In April 2003, BPEL was submitted to OASIS (Organization for the Advancement of Structured Information Standards) for standardization purposes where the WSBPEL TC (Web Services Business Process Execution Language Technical Committee) has been formed (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel) since. This has led to even broader acceptance in industry.

BPEL can be used within and between companies. Within companies the role of BPEL is to standardize enterprise application integration and extend the integration to previously isolated systems. Between enterprises, BPEL will enable easier and more effective integration with business partners. Definitions of business processes described in BPEL do not impact existing systems thus stimulating upgrades. BPEL is the key technology in environments where functionalities already are or will be exposed via web services. With increases in the use of web service technology the importance of BPEL will rise further.

BPEL Language

Let us now have a look at the BPEL language. BPEL has been designed specifically as a language for definition of business processes. BPEL supports two different types of business processes:

- Executable processesallow us to specify the exact details of business processes. They can be executed by an orchestration engine. In most cases BPEL is used for executable processes.
- Abstract business protocols allow us to specify the public message exchange between parties only. They do not include the internal details of process flows and are not executable.

BPEL builds on top of XML and web services. It is an XML-based language which supports the web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination and WS-Transaction. BPEL represents a convergence of two early workflow languages, WSFL (Web Services Flow Language) and XLANG. WSFL was designed by IBM and is based on the concept of directed graphs. XLANG was designed by Microsoft and is a block-structured language. BPEL combines both approaches and provides a rich vocabulary for description of business processes.

A BPEL process specifies the exact order in which participating web services should be invoked. This can be done sequentially or in parallel. With BPEL, we can express conditional behavior, for example, a web service invocation can depend on the value of a previous invocation. We can also construct loops, declare variables, copy and assign values, define fault handlers, and so on. By combining all these constructs, we can define complex business processes in an algorithmic manner.

BPEL is thus comparable to general purpose programming language such as Java, but it is not as powerful as Java. On the other hand it is simpler and better suited for business process definition. Therefore BPEL is not a replacement but rather a supplement to modern languages such as Java.

Let us have a closer look at a typical BPEL process. First, the BPEL business process receives a request. To fulfill it, the process then invokes the involved web services and finally responds to the original caller. Because the BPEL process communicates with other web services, it relies heavily on the WSDL description of the web services invoked by the composite web service.

A BPEL process consists of steps. Each step is called an activity. BPEL supports primitive and structure activities. Primitive activities represent basic constructs and are used for common tasks, such as those listed below:

- Invoking other web services, using <invoke>

- Waiting for the client to invoke the business process through sending a message, using <receive> (receiving a request)
- Generating a response for synchronous operations, using <reply>
- Manipulating data variables, using <assign>
- Indicating faults and exceptions, using <throw>
- Waiting for some time, using <wait>
- Terminating the entire process, using <terminate>, etc.

We can then combine these and other primitive activities and define complex algorithms, which exactly specify the steps of business processes. To combine primitive activities BPEL supports several structured activities. The most important are:

- Sequence (<sequence>), which allows us to define a set of activities that will be invoked in an ordered sequence
- Flow (<flow>) for defining a set of activities that will be invoked in parallel
- Case-switch construct (<switch>) for implementing branches
- While (<while>) for defining loops
- The ability to select one of a number of alternative paths, using <pick>

Each BPEL process will also declare variables, using <variable> , and define partner links, using <partnerLink> . We will say more on partner links later in this article.

A BPEL process can be synchronous or asynchronous. A synchronous BPEL process blocks the client (the one which is using the process) until the process finishes and returns a result to the client. An asynchronous process does not block the client. Rather it uses a callback to return the result (if any). Usually we use asynchronous processes for longer-lasting processes and synchronous for processes that return a result in a relatively short time. If a BPEL process uses asynchronous web services, the process itself is usually also asynchronous (although this is not necessary).

For its clients a BPEL process looks like any other web service. When we define a BPEL process, we actually define a new web service that is a composition of existing services. The interface of the new BPEL composite web service uses a set of port types, through which it provides operations like any other web service. To invoke a business process described in BPEL, we have to invoke the resulting composite web service. The figure below shows a schematic view of a BPEL process:

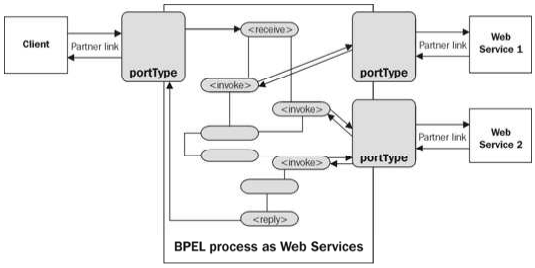

Figure: Example BPEL process

Partner Links

Earlier we have mentioned that BPEL processes also declare partner links. Let us now explain what partner links are. We have already said that BPEL processes interact with external web services in two ways:

- The BPEL process invokes operations on other web services.
- The BPEL process receives invocations from clients. One of the clients is the user of the BPEL process, which makes the initial invocation. Other clients are web services, for example, those that have been invoked by the BPEL process, but make callbacks to return replies.

BPEL calls the links to all parties it interacts with *partner links*. Partner links can be links to web services that are invoked by the BPEL process. Partner links can also be links to clients which invoke the BPEL process. Each BPEL process has at least one client partner link, because there has to be a client that invokes the BPEL process.

Usually a BPEL process will also have at least one invoked partner link, because it will most likely invoke at least one web service (usually more than one). Invoked partner links may, however, become client partner links—this is usually the case with asynchronous services, where the process invokes an operation. Later the service (partner) invokes the call-back operation on the process to return the requested data.

BPEL treats clients as partner links for two reasons. The most obvious reason is support for asynchronous interactions. The second reason is based on the fact that the BPEL process can offer services. These services, offered through port types, can be used by more than one client. The process may wish to distinguish between different clients and offer them only the functionality they are authorized to use. For example, an insurance process might offer a different set of operations to car-insurance clients than to real-estate insurance clients.

To sum up, we can see that the partner links describe links to partners, where partners might be:

- Services invoked by the process
- Services that invoke the process
- Services that have both roles—they are invoked by the process and they invoke the process

BPEL Example

To get an idea how a BPEL process looks we show below a very simple BPEL process, which selects the best insurance offer. We first declare the partner links to the BPEL process client (called client) and two insurance web services (called insuranceA and insuranceB):

```
<?xml version="1.0" encoding="utf-8"?>

<process name="insuranceSelectionProcess"
       targetNamespace="http://packtpub.com/bpel/example/"
       xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
       xmlns:ins="http://packtpub.com/bpel/insurance/"
       xmlns:com="http://packtpub.com/bpel/company/" >

  <partnerLinks>
```



```
    <partnerLink name="client"
           partnerLinkType="com:selectionLT"
           myRole="insuranceSelectionService"/>

    <partnerLink name="insuranceA"
           partnerLinkType="ins:insuranceLT"
           myRole="insuranceRequester"
           partnerRole="insuranceService"/>

    <partnerLink name="insuranceB"
           partnerLinkType="ins:insuranceLT"
           myRole="insuranceRequester"
           partnerRole="insuranceService"/>

  </partnerLinks>
...
```

Next, we declare variables for the insurance request (InsuranceRequest), insurance A and B responses (InsuranceAResposne, InsuranceBResposne), and for final selection (InsuranceSelectionResponse):

```
...
  <variables>
    <!-- input for BPEL process -->
    <variable name="InsuranceRequest"
           messageType="ins:InsuranceRequestMessage"/>
    <!-- output from insurance A -->
    <variable name="InsuranceAResposne"
           messageType="ins:InsuranceResponseMessage"/>
    <!-- output from insurance B -->
    <variable name="InsuranceBResposne"
           messageType="ins:InsuranceResponseMessage"/>
    <!-- output from BPEL process -->
    <variable name="InsuranceSelectionResponse"
           messageType="ins:InsuranceResponseMessage"/>
  </variables>
...
```

Finally, we specify the process steps. First we wait for the initial request message from the client (<receive>). Then we invoke both insurance web services (<invoke>) in parallel using the <flow> activity. The insurance web services return the insurance premium. Then we select the lower amount (<switch>/ <case>) and return the result to the client (the caller of the BPEL process) using <reply> activity:

```
...
  <sequence>

    <!-- Receive the initial request from client -->
    <receive partnerLink="client"
          portType="com:InsuranceSelectionPT"
          operation="SelectInsurance"
          variable="InsuranceRequest"
          createInstance="yes" />

    <!-- Make concurrent invocations to Insurance A and B -->
    <flow>

      <!-- Invoke Insurance A web service -->
      <invoke partnerLink="insuranceA"
           portType="ins:ComputeInsurancePremiumPT"
           operation="ComputeInsurancePremium"
           inputVariable="InsuranceRequest"
           outputVariable="InsuranceAResposne" />

      <!-- Invoke Insurance B web service -->
      <invoke partnerLink="insuranceB"
           portType="ins:ComputeInsurancePremiumPT"
```



```
           operation="ComputeInsurancePremium"
           inputVariable="InsuranceRequest"
           outputVariable="InsuranceBResposne" />

    </flow>

    <!-- Select the best offer and construct the response -->
    <switch>

      <case condition="bpws:getVariableData('InsuranceAResposne',
                'confirmationData','/confirmationData/Amount')
              <= bpws:getVariableData('InsuranceBResposne',
                'confirmationData','/confirmationData/Amount')">

        <!-- Select Insurance A -->
        <assign>
          <copy>
            <from variable="InsuranceAResposne" />
            <to variable="InsuranceSelectionResponse" />
          </copy>
        </assign>
      </case>

      <otherwise>
        <!-- Select Insurance B -->
        <assign>
          <copy>
            <from variable="InsuranceBResposne" />
            <to variable="InsuranceSelectionResponse" />
          </copy>
        </assign>
      </otherwise>
    </switch>

    <!-- Send a response to the client -->
    <reply partnerLink="client"
         portType="com:InsuranceSelectionPT"
         operation="SelectInsurance"
         variable="InsuranceSelectionResponse"/>

  </sequence>

</process>
```

Because each BPEL process is a web service each BPEL process needs a WSDL document too. We will not go into further details of developing BPEL processes. More information can be found in the book Business Process Execution Language for Web Services published by Packt Publishing in October 2004.

BPEL vs. Java

From the example BPEL process above somebody might think that such a composition could be easily done from Java too. This is correct for very simple processes. For more complex processes we can however see that BPEL provides at least two important advantages over Java.

The first advantage of BPEL over Java is that BPEL processes are portable even outside the Java platform. BPEL processes can be executed on orchestration servers based on Java platform or on any other software platform (for example .NET). This is particularly important in business-to-business interactions where different partners use different platforms.

The second important advantage of BPEL is its support for specifics of business

processes. Usually business processes are long-running, particularly if they involve interactions with partners over Internet. It can happen that such processes execute minutes, hours, even days before they finish. It can happen that they invoke a web service and need to wait for the callback a relatively long time. If we would use a Java application instead of a BPEL process we would soon have a lot of work worrying which processes have finished, which are still running. We would also need to track which Java applications (processes) we can close and which still have to run in order to receive callbacks.

BPEL also supports compensation in a relatively easy way. Compensation, or undoing steps in the business process that have already completed successfully, is one of the most important concepts in business processes. The goal of compensation is to reverse the effects of previous activities that have been carried out as part of a business process that is being abandoned.

Compensation is related to the nature of most business processes, which are long running and use asynchronous communication with loosely coupled partner web services. Business processes are often sensitive in terms of successful completion because the data they manipulate is sensitive. Because they usually span multiple partners (often multiple enterprises) special care has to be taken that business processes either fully complete their work or that the partial (not fully completed) results are undone − compensated. This is similar to ACID transactions used in enterprise information systems. BPEL supports the concept of compensation with the ability to define compensation handlers, which are specific to scopes, and calls this feature *Long-Running Transactions* (LRT).

Business processes may also have to react on certain events. Such events can be message events or alarm events. Message events are triggered by incoming messages through operation invocation on port types. Alarm events are time related and are triggered either after a certain duration or at a specific time. BPEL provides good support for managing events in business processes.

Then there are concurrent activities. In BPEL, concurrent activities are modeled using the <flow> activity. Gathering nested activities within <flow> is straightforward and very useful for expressing concurrency scenarios that are not too complicated. To express more complex concurrency scenarios, <flow> provides the ability to express synchronization dependencies between activities. In other words, we can specify which activities can start and when (depending on other activities) and define complex dependencies. For example, we will often specify that a certain activity or several activities cannot start before another activity or several activities have finished.

In contrast to web services which are a stateless model business processes require use a stateful model. When a client starts a business process, a new instance is created. This instance lives for the duration of the business process. Messages sent to the business process (using operations on port types and ports) need to be delivered to the correct instance of the business process. BPEL provides a mechanism to use specific business data to maintain references to specific business process instances and calls this feature correlation.

We could continue this discussion but it is obvious that BPEL has been designed to address the requirements of defining business processes. It is also obvious that BPEL cannot replace Java neither as a programming language nor as a platform. Actually Java platform is a platform of choice for running BPEL processes.

BPEL Servers and Development Tools

To execute BPEL executable processes we need an orchestration server. Orchestration servers provide a run-time environment for executing BPEL business processes. BPEL is strongly related to web services and to the modern software platforms that support web service development, particularly to Java 2 Enterprise Edition (J2EE) and Microsoft .NET.

BPEL servers leverage J2EE or .NET application server environments, where they can make use of the services provided by application servers, such as security, transactions, scalability, integration with databases, components such as EJBs (Enterprise Java Beans), messaging systems such as JMS (Java Message Service), etc.

BPEL orchestration servers exist for both J2EE and .NET platforms. For J2EE we can choose at least between:

- Oracle BPEL Process Manager (http://www.oracle.com/technology/products/ias/bpel/index.html)
- IBM WebSphere Business Integration Server Foundation (http://www.ibm.com/software/integration/wbisf)
- IBM alphaWorks BPWS4J (http://www.alphaworks.ibm.com/tech/bpws4j)
- OpenStorm Service Orchestrator (http://www.openstorm.com)
- Vergil VCAB Server (http://www.vergiltech.com/products_VCAB.php)
- Active Endpoints ActiveWebflow Server (http://www.active-endpoints.com/products/index.html)
- ActiveBPEL engine (http://www.activebpel.org/)
- Fivesight Process eXecution Engine (http://www.fivesight.com/pxe.shtml)

BPEL orchestration servers based on the .NET platform include:

- Microsoft BizTalk 2004 (http://www.microsoft.com/biztalk/)
- OpenStorm Service Orchestrator (http://www.openstorm.com)

OpenStorm provides solutions for both platforms J2EE and .NET. In addition to BPEL orchestration servers there are also a few BPEL design tools available. These tools enable graphical development of BPEL processes and often comprise a part of the servers:

- Oracle BPEL Designer
- IBM WebSphere Studio Application Developer, Integration Edition
- IBM BPWS4J Editor
- Vergil VCAB Composer
- Active Endpoints ActiveWebflow Designer

The Oracle, IBM, and Active Endpoints solutions are based on the Java Eclipse framework, while the Vergil VCAB Composer is built on the.NET framework. We can see that considerably more BPEL servers and designers are available for the Java platform.

BPEL + Java

We have seen that BPEL is an appropriate language for programming in the large - that is for composing web services (business logic) into business processes. BPEL is not (and does not try) to be a general purpose language. Therefore BPEL and Java fit

together, where Java takes the role of the programming language for web services and the platform on which web services and BPEL processes are executed.

BPEL, as proposed by the specification, can also be extended. Particularly the idea to extend the reach of BPEL beyond web services is promising. This means that we could use BPEL to compose all kinds of resources, not only web services. In J2EE this could be EJBs, JMS, RMI, JCA, and other resources. Generally there are two solutions to this question:

- Enable mixing of BPEL and Java code, which is the idea of BPELJ.
- Describe all resources (Java classes, EJBs, JMS, etc.) with WSDL, which is the idea behind the Web Services Invocation Framework.

BPELJ provides the possibility to include Java code (which is called Java snippets) in BPEL process definitions. This on one hand enables that we invoke Java resources form BPEL directly, for which BPELJ introduces Java partner links. On the other hand it gives additional power to BPELJ, because with Java snippets we can perform tasks, such as calculate values, construct XML documents, and execute other code without having to create web services. We can also use BPEL variables form Java snippet code. BPELJ is supported by IBM and BEA which have published a white paper on BPELJ.

Integration of Java code into BPEL processes to invoke Java resources is very useful. However, in some cases such approach may have disadvantages. The invocation of a Java resource differs from the invocation of a web service. The Web Services Invocation Framework (WSIF) follows another idea: use the same syntax in BPEL to invoke any resource (or service) and describe it using WSDL even if it is a Java resource that does not communicate through SOAP. WSIF also allows us to map such a service to the actual implementation and protocol.

In other words, we can bind the abstract description of the service (the port types) to a SOAP-based implementation, to a Java class, to an EJB, or any other supported resource simply by modifying the WSDL binding. No code changes in the BPEL process are necessary and no extensions to BPEL are required. The bindings supported are determined by the providers offered by the WSIF.

WSIF is an Apache technology (http://ws.apache.org/wsif/) that was originally developed by IBM alphaWorks as a part of WSTK (Web Services Toolkit). Currently some BPEL servers, for example the Oracle BPEL Process Manager already support WSIF.

Both approaches, BPELJ and WSIF are suitable for real-world scenarios and make BPEL very useful for EAI as well as for B2B.

Conclusion

BPEL is an important language for the process-oriented approach to SOA. Because BPEL has been designed specifically for definition of business processes it provides good support for various specifics of business processes such as support for long running transactions, compensation, event management, correlation, etc. BPEL is well suited for use with the J2EE platform and many BPEL servers build on top of J2EE. With ideas of combing BPEL and Java (BPELJ), and WSIF, the usability of BPEL is even increasing. We should also look at the emerging JBI (Java Business Integration) specification aka JSR 208 which will give business integration and BPEL an even better

documented position in the Java platform.

Resources

- Book “Business Process Execution Language for Web Services” by Matjaz B. Juric with Benny Mathew and Poornachandra Sarang, Packt Publishing, October 2004, ISBN 1904811183. TheServerSide.com members receive a 20% discount, when using the code: 04tssbpel20 to get the book www.PacktPub.com
- Specification “Business Process Execution Language for Web Services”, Version 1.1, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
- BPELJ: BPEL for Java, A Joint White Paper by BEA and IBM, ftp://www6.software.ibm.com/software/developer/library/ws-bpelj.pdf
- Web Services Invocation Framework, http://ws.apache.org/wsif/

Author Bio

Matjaz B. Juric holds a Ph.D. in computer and information science. He is the author of the book *Business Process Execution Language for Web Services* (Packt Publishing) http://www.packtpub.com/book/BPEL. Matjaz is also the co-author of Professional J2EE EAI, Professional EJB, J2EE Design Patterns Applied , and VB.NET Serialization Handbook , all published by Wrox Press. Matjaz has also contributed to Java Developer's Journal, Java Report, Java World , and other publications.