WS-BPEL 2.0 Tutorial

Author: Antony Miguel (antony.miguel@scapatech.com), Last Updated: 13th October 2005

Contents

Reader Requirements

This tutorial assumes that the reader has some basic familiarity with XML. For a tutorial on XML try here at w3schools.com.

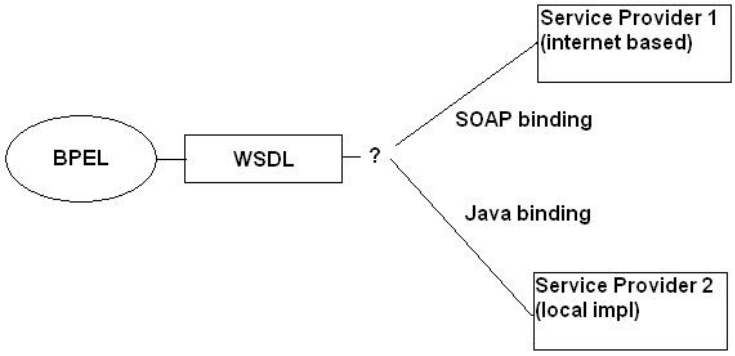
Because BPEL maps quite well to an existing set of concepts in other programming languages and technical subjects, the tutorial will be interspersed with 'Analogy' sections which will hopefully help the reader to understand BPEL in the context of some more familiar language or subject. These sections will look like this...

Analogy

(Programming) A BPEL <flow> activity is like launching multiple threads, and then waiting for them to complete.

(Real World) A BPEL <flow> activity is like asking a bunch of people to do different things for you at the same time, and then waiting for them all to respond.

The tutorial will also be interspersed with beautifully crafted images like the one below to help the reader visualise concepts.



Overview - What is BPEL?

Web Services Business Process Execution Language (WS-BPEL) is an XML based programming language to describe high level business processes. A 'business process' is a term used to describe the interaction between two businesses or two elements in some business. An example of this might be company A purchasing something from company B. BPEL allows this interaction to be described easily and thoroughly such that company B can provide a Web Service and company A can use it with a minimum of compatibility issues.

Overview - What is a Web Service?

In terms of BPEL, the term 'Web Service' means something with which you can interact. For example, you could have a web service which you can interact with to get the time of day, or a web service you can interact with to buy a fridge. The web service can have any purpose.

A Web Service is typically described using Web Service Description Language (WSDL). This is another XML based language which allows you to describe the interface to the web service.

WSDL tells you exactly how you can interact with the web service but says nothing about how the web service works. This forces you to rely on only the description of the web service rather than anything in the implementation - which helps to reduce (maybe even eradicate) compatibility problems between two web services which provide the same function, but use different implementations.

Analogy

(Programming) WSDL is like an interface or an abstract class. It tells you what the API is to the web service but it doesn't tell you anything about the implementation.

(Real World) A WSDL document is like a set of instructions for a household appliance. The instructions tell you how to use the appliance but they don't tell you how the appliance will do what it does. They tell you everything you need to know about how to use a toaster but they don't tell you the details of how the toaster toasts toast.

Overview - So whats so great about Web Services?

Web Services also have some other nice features which make them useful for describing interactions or services.

Platform independence. A web service is described in WSDL which is a well defined language which any computer or operating system can understand. This means that whether you're running a Mac or a PC or a mobile phone, everyone is speaking the same well defined language.

Distributed. Web services and WSDL are designed so that when you describe an interface, you don't say anything about where the web service is or where you are when you talk to it. So if a web service is something running on your computer or something available on the internet, the web service description looks the same, only the 'address' of the web service changes.

Overview - Where does BPEL fit in?

BPEL allows you to write programs that **use** web services, and to write programs that **are** web services.

Using BPEL you can write a program that uses a web service to fetch the weather forecast for your postcode (zip code), check whether it will be raining this weekend and use another web service to send you an SMS message telling suggesting that you go shopping (if its raining) or go to the beach (if its sunny). This might not be the most useful BPEL program ever but every time a new web service

turns up (either as something you can download onto your computer and use, or as a public web service available on the internet) the possibilities get larger and larger.

Also using BPEL you can write the same program as above, but make the decision to go to the beach or go shopping a web service itself. Other web service clients could connect to you and ask what they should do this weekend based on their postcode.

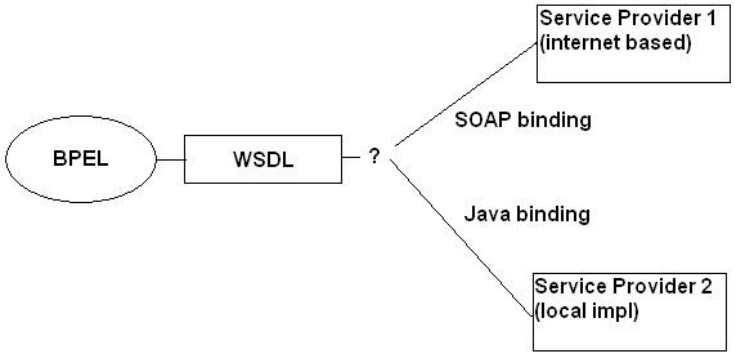
Analogy

(Programming) In a standard programming language, if you want to print 'Hello World', you use an I/O library to write 'Hello World' to the console, or to pop up a window with a message. In BPEL, you need a web service that you pass the string 'Hello World' into (which may then use some other I/O library to pop up the message or print to the console).

(Real World) Don't worry about it.

'binding'. This binding specifies what the implementation of the web service is. A common binding is SOAP/HTTP which uses the XML Simple Object Access Protocol (SOAP) over HTTP (standard web page fetching language) to speak to web services which are on the internet. Another common binding is the Java binding. This binding allows you to define local in-process java implementations which implement web services.

So if you wanted to write a web service that allowed clients to print things (e.g. 'Hello World') you could write it in Java and then expose it as a web service.



BPEL server as a nice easy to use proxy to all their legacy systems.

In either case, a common chain of events will be as follows:

- Web Service client (user) requests something from the server (perhaps via a browser or another server)
- Client request is received by the BPEL server.
- Client request is identified as a new request and a new BPEL process is used to serve that client
- Client continues to interact with the BPEL process, making requests etc, until the interaction is complete
- BPEL process disappears and client goes about it's business

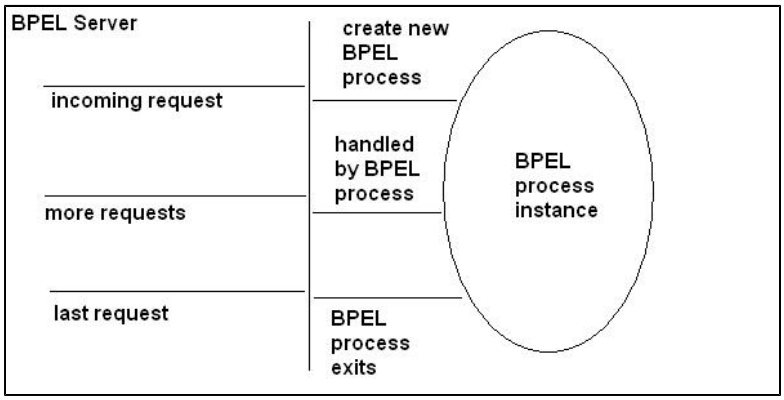
This automatic creation of BPEL process instances in response to client requests is called 'create on demand'.

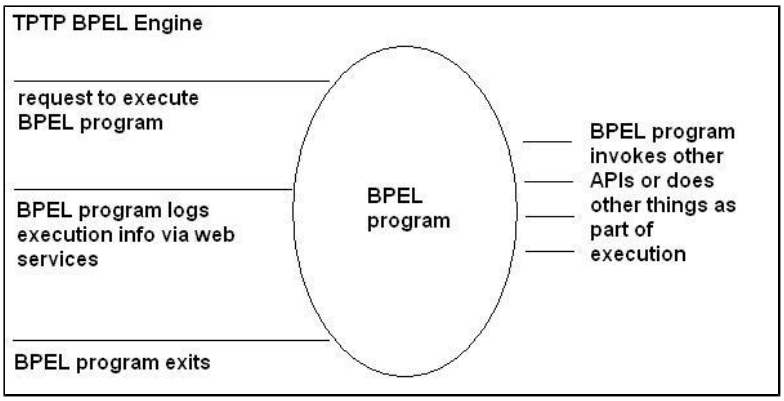
The TPTP BPEL engine has a different focus. It's primary focus is to act as an execution engine which can be programmed using BPEL.

The common chain of events for the TPTP BPEL engine is therefore slightly different:

- User writes a BPEL program as a series of steps to carry out
- User runs BPEL program
- One single copy of the BPEL program is created and immediately executed
- The running program logs information via local web services
- The program finishes and the user can look at the logs

In short, the TPTP BPEL engine doesn't do 'create on demand'.





- Data types
- Input/Output (I/O)

BPEL splits these components up in the following way:

- Programming logic - **BPEL**
- Data types - **XSD (XML Schema Definition)**
- Input/Output (I/O) - **WSDL (Web Services Description Language)**

As a simple example, lets take a Hello World program.

XSD will be used to define the types used in the program. It will be used to define a string type which will hold the 'Hello World' string.

WSDL will be used to define the web service that will actually print the string for us.

BPEL will put all these things together to create the string and print it.

Hello World BPEL program

```
<?xml version="1.0" encoding="UTF-8"?>
<process
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:print="http://www.eclipse.org/tptp/choreography/2004/engine/Print"

        <!--Hello World - my first ever BPEL program -->

        <import importType="http://schemas.xmlsoap.org/wsdl/"
                location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
                namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />

        <partnerLinks>
                <partnerLink    name="printService"
                                partnerLinkType="print:printLink"
                                partnerRole="printService"/>
        </partnerLinks>

        <variables>
                <variable       name="hello_world"
                                messageType="print:PrintMessage" />
        </variables>

        <assign>
                <copy>
                        <from><literal>Hello World</literal></from>
                        <to>$hello_world.value</to>
                </copy>
        </assign>

        <invoke partnerLink="printService" operation="print" inputVariable="hello_world" />

</process>
```

Importing WSDL and XSD files:

Importing WSDL and XSD files

```
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
        namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />
```

The 'import' statement in BPEL is a directive to import a WSDL or XSD file. Importing XSD files allows commonly used datatypes or datatypes required for a particular endpoint (thing that you speak to) to be defined in a file separate to the BPEL file. Importing WSDL files allows endpoint descriptions (definitions of things that you speak to) to be defined in files separate to the BPEL file.

The benefit here is that entire endpoints and any associated datatypes that they need can be defined outside the BPEL file and can therefore be re-used in many BPEL files.

Note that WSDL files have a similar import mechanism allowing them to import common XSD files and even other WSDL files.

In this case, a Hello World web service has been defined in the WSDL file '...tptp_EnginePrinterPort.wsdl'.

Partner Link definitions:

PartnerLink definitions

```
<partnerLinks>
        <partnerLink    name="printService"
                        partnerLinkType="print:printLink"
                        partnerRole="printService"/>
</partnerLinks>
```

Partner Links can be thought of as placeholders for things that you actually speak to. A web service is described in full by the WSDL files that specify it but Partner Links allow you to have something like an instance of the web service that you speak to. A partner link basically maps to a WSDL web service 'portType', so one partnerLink (e.g. 'printService' above) maps to a single web service.

However, partner links don't just describe what you speak to, they also can describe how other web service clients speak to you. In the partner link definition above, a 'partnerRole' attribute defines the web service that this BPEL process will speak to. Alternatively, the partner link could have a 'myRole' attribute which would define a web service that this BPEL proces implements.

Analogy

(Programming) ?

(Real World) a web service is like a description of how to phone a restaurant - you phone up, tell them your address and you place an order for some food. A partner link is like bit of paper which has on it a phone number for a particular restaurant - you can use the partner link to speak to a specific restaurant (web service). The actual phone number of the restaurant is called the 'endpoint address' and is defined either in the WSDL or in the BPEL (where it may be copied into an existing partnerLink to speak to a different restaurant).

Variable definitions:

Variable definitions

```
<variables>
        <variable       name="hello_world"
                        messageType="print:PrintMessage" />
</variables>
```

Variables are used to contain data in BPEL. A variable can either contain an XSD value or a WSDL message. In the example above, a variable called 'hello_world' is declared as a container for WSDL messages of type 'print:PrintMessage'. Instead of the 'messageType' attribute, the variable could have had a 'type' attribute which would specify some xsd simple or complex type like 'xsd:string' or 'xsd:integer'.

Variables are used to pass data in and out of web service endpoints.

Variable assignment:

Variable assignment

```
<assign>
        <copy>
                <from><literal>Hello World</literal></from>
                <to>$hello_world.value</to>
        </copy>
</assign>
```

Variables are manipulated in BPEL either through use via web service endpoints or by assignment. The example above shows a literal string value being assigned into the variable 'hello_world'. The variable 'hello_world' in this case is a WSDL message with a part called 'value'. The part called 'value' is an 'xsd:string' type. It can therefore have other 'xsd:string's assigned into it, including literal strings.

The '$varname' syntax used to reference the variable here is standard XPATH expression syntax. The '.' separator is used to specify the WSDL message part. If the variable or the part were an XSD complex type then a '/' separator could be used to specify the sub-element within the complex type (e.g. '$hello_world.value/subvalue').

Web Service Invocation:

Web Service Invocation

```
<invoke partnerLink="printService" operation="print" inputVariable="hello_world" />
```

The 'invoke' activity in BPEL invokes a web service endpoint. This is where the BPEL process passes the 'Hello World' data (stored in the 'hello_world' variable) to the 'print' web service. The specified partnerLink tells the BPEL engine the address of the web service you want to invoke here. The 'print' operation specifies what you actually want the web service to do and the 'inputVariable' specifies that the input WSDL message should come from the 'hello_world' variable.

What the web service actually does and exactly how the web service is implemented is not referenced in BPEL at all - all the implementation and definition information is contained within the defining WSDL file.

Analogy

(Programming) Invoking a web service is similar to invoking a function or a method on an API or an object. One important difference though is that BPEL doesn't have any reference to that object or API, it only knows how to speak to it (WSDL definition) and the address of the implementation (the endpoint reference). The mechanics of getting to the implementation specified by the endpoint reference is not dealt with by BPEL.

(Real World) Invoking the web service is akin to actually making the call to the restaurant. The variable defined previously holds the information you need to pass to them (e.g. a written list of the food you want) and the partnerLink is, as previously described, the bit of paper with the restaurant phone number. BPEL looks at the phone number, dials the restaurant and passes your order to them.

Web Service Description (WSDL file)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
               targetNamespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
               xmlns:tns="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
               xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
               xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
               >

    <!-- engine printout port -->
    <message name="PrintMessage">
        <part name="value" type="xsd:string"/>
    </message>

    <portType name="Print">
        <operation name="print">
            <input message="tns:PrintMessage"/>
        </operation>
    </portType>

    <binding name="PrintPortWsifBinding" type="tns:Print">
        <java:binding/>

        <format:typeMapping encoding="Java" style="Java">
            <format:typeMap typeName="xsd:string" formatType="java.lang.String"/>
        </format:typeMapping>

        <operation name="print">
            <java:operation methodName="print" parameterOrder="value"/>
        </operation>
    </binding>

    <service>
        <port name="JavaPrintPort" binding="tns:PrintPortWsifBinding">
            <java:address
className="org.eclipse.tptp.choreography.jengine.internal.extensions.wsdlbinding.wsif.ports.EnginePrinterPort"/>
        </port>
    </service>
```

The Target Namespace:

The Target Namespace

```
      targetNamespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
```

A WSDL file is defining the creation of new things. It is defining the creation of a new web service and any associated data types or WSDL Messages that are to be used with that web service. When each of these things is defined they are given a name. A WSDL message used to hold the string value to print, for example, is given the name 'printMessage'. However, 'printMessage' might be a name that many other web services want to use also. In fact 'printMessage' might be a very popular name indeed. If this were the case then a BPEL process that wanted to use two of these web services would need some way to distinguish when it was referring to web service A's printMessage and web service B's printMessage.

XML namespaces and the target namespace are how BPEL makes the distinction between web service A's messages and web service B's messages when they have the same name.

When BPEL refers to a printMessage (e.g. when the variable 'hello_world' were declared) it prefixes it with a reference to a namespace mapping 'print:'. The namespace mapping 'print:' has in turn been defined in the BPEL file to map the the target namespace where the printMessage message was defined.

In short, when the WSDL file specified a targetNamespace of 'X', all the things defined within that WSDL file are considered defined under the namespace 'X'. The BPEL process that references these things then has to reference the namespace of the thing too.

Note: the concept of namespaces is a general XML concept, and in general, names which do not have namespaces (like BPEL variable names) are called 'NCNames'. Names which DO have namespac
are called 'QNames'. The Q in QName stands for 'Qualified' as a name which contains a namespace reference is said to be 'fully qualified'.

Namespaces are defined as mappings by prefixing the attribute with 'xmlns:' (e.g. xmlns:shortnamespace="http://www.my.very.long.namespace.which.would.be.cumbersome.to.write.all.the.time/").

Analogy

(Programming) XML namespaces are like packages in Java. The WSDL messages and port type created in the WSDL file inherit the XML namespace specified by the targetNamespace attribute.

(Real World) ?

WSDL Messages:

WSDL Messages

```
<!-- engine printout port -->
<message name="PrintMessage">
    <part name="value" type="xsd:string"/>
</message>
```

WSDL messages are used to specify what the containers should be like that hold data when a WSDL operation is invoked. They are essentially lists of parts, each of which is an XSD simple or complex type.

In the example above the 'PrintMessage' is defined as having a single part 'value', which is of type 'xsd:string'.

WSDL Port Types:

WSDL Port Types

```
<portType name="Print">
    <operation name="print">
        <input message="tns:PrintMessage"/>
    </operation>
</portType>
```

WSDL Port Types represent the definition of the web service itself. They describe the API or interface to the web service. A port type is a list of operations with 'input's and 'output's. Each of the 'input's ar
'output's is a WSDL message that must have been previously defined (although it could have been imported from another WSDL file).

A WSDL port type operation can also have any number of 'fault' elements. Each of these specifies an error message which would be an alternative to the 'output' message.

Note that the name specified on the port type does not have any namespace prefix. This is because the port type is being defined here and now and it will inherit the target namespace. The 'printMessag
specified in the operation definition however DOES have a namespace prefix. This is because the printMessage has previously been defined and is being referenced. The 'tns:' prefix maps to the same namespace as the target namespace in the previously defined 'printMessage' message.

Analogy

(Programming) The WSDL port type is like an interface or abstract class. It doesn't specify any particular implementation for anything, but it does say exactly what can be done, and what goes in and comes out.

(Real World) The port type is like a description of how to phone up and order food from any restaurant. The description doesn't specify how the food is cooked or which restaurant to phone, only what the steps are to phone them and the information they will need.

WSDL Port Type Bindings:

WSDL Port Type Bindings

```
<binding name="PrintPortWsifBinding" type="tns:Print">
    <java:binding/>

    <format:typeMapping encoding="Java" style="Java">
        <format:typeMap typeName="xsd:string" formatType="java.lang.String"/>
    </format:typeMapping>

    <operation name="print">
        <java:operation methodName="print" parameterOrder="value"/>
    </operation>
</binding>
```

A 'binding' in WSDL specifies how the web service is actually implemented. Everything up until this point has been abstract and has dealt with only how to speak to the web service. The binding specifies what is on the other side that you are speaking to.

A web service can be bound in many different ways. The most common bindings for a web service are:

- as a SOAP/HTTP web service - whereby some implementation would be listening on a specified port and would accept SOAP messages over an HTTP transport.
- as a Java web service - whereby some java class is mapped to the port type and is used directly as an implementation.

In the binding above, a mapping has been created which specifies that the port type operation 'print' should be mapped to a Java method called 'print'. In addition to this, the XSD type 'string' has been mapped to the Java type 'String'.

Using this mapping information a Java class can be specified later in the WSDL file as the 'address' of a concrete web service implementation. This class can be instantiated and when calls are made to 'print' operation, they will be proxied to the 'print' method of this class. In the process of proxying the 'print' operation, any XSD strings will also be converted to Java strings as specified in the binding.

WSDL Service:

WSDL Service

```
<service>
    <port name="JavaPrintPort" binding="tns:PrintPortWsifBinding">
        <java:address
className="org.eclipse.tptp.choreography.jengine.internal.extensions.wsdlbinding.wsif.ports.EnginePrinterPort"/>
    </port>
</service>
```

The WSDL 'service' element specifies a WSDL 'port'. A single port is an instance of a web service, which is implemented via a particular binding and which is available at a given address.

In the case of our printout port, we are defining a web service which is bound using the previously defined Java binding and which can be found at the address 'org.eclipse...EnginePrinterPort'.

Note that the address is binding specific. The Java binding knows to interpret the 'className' attribute as a fully qualified Java class name and understands how to instantiate the class and proxy the WSDL operations to the Java methods specified in the 'binding' element.

Partner Link Types:

Partner Link Types

```
<partnerLinkType name="printLink">
    <role name="printService" portType="tns:Print"/>
</partnerLinkType>
```

Partner Link Types are actually not a WSDL construct, but a BPEL construct. WSDL was around before BPEL and is purely designed towards describing web services. BPEL however requires that a partner link instance be associated with a particular WSDL port type. In this case there is only one end of the partner link which needs to be implemented - the 'printService' role. It is possible to have tw roles in a single partner link type, each of which can be implemented by two communicating web services.

The BPEL partnerLink definition specified a partner link type for the partner link and also either a 'myRole' or a 'partnerRole'. In our previous example the 'partnerRole' was defined as 'printService'. This because the BPEL process will be speaking TO the 'printService', rather than acting as a 'printService' which other web clients can speak to (in that case the 'myRole' part of the partner link would have been defined).

The Java Web Service Implementation:

The Java Web Service Implementation

```
package org.eclipse.tptp.choreography.jengine.internal.extensions.wsdlbinding.wsif.ports;

public class EnginePrinterPort {

        public void print(String s) {
                System.out.println(s);
        }

}
```

The Java class specified in the WSDL 'service' element is the Java class that the BPEL engine will expect to find at runtime. We therefore need to create that java class and fill it with any methods that specified mappings to in the WSDL 'binding' element (e.g. the 'print' method).

If you want to write a Java web service implementation or build up a library of useful services that you want to be able to distribute or send to other people then you can expose dependencies via the Ec Extension Point mechanism.

Just extend the extension point 'org.eclipse.tptp.choreography.internal_WSDLPortDependency' and add a reference to your JAR. The 'namespace' attributes specify which namespaces your JAR is required for. If any of the specified namespaces are used your JAR will be passed round the engine in anticipation of use by the BPEL program.

Eclipse Java Binding Dependency Extension Point

```
<extension
                point="org.eclipse.tptp.choreography.internal_WSDLPortDependency">
        <dependency jar_path="/choreography.jar">
                <namespace namespace="http://www.eclipse.org/tptp/choreography/2004/engine" />
        </dependency>
</extension>
```