WS-BPEL

Web Services Business Process Execution Language

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es

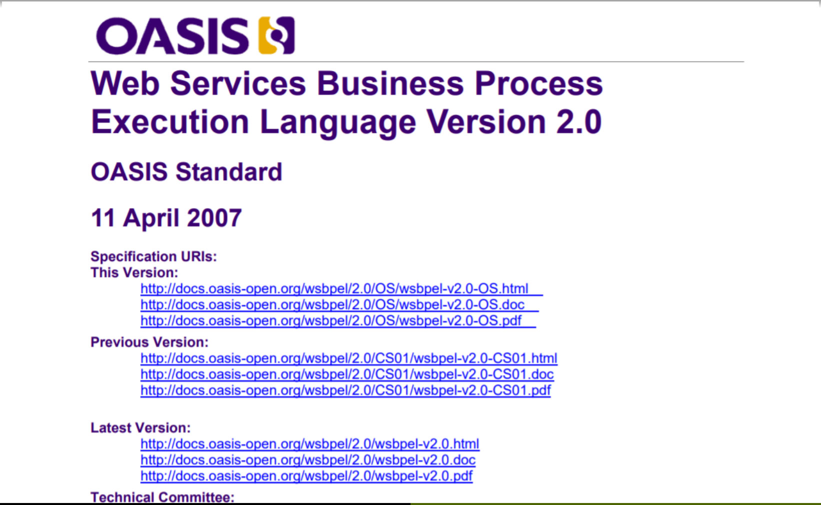
DSBCS - November, 15th 2023

Máster en Ingeniería Informática

BPEL4WS and WS_BPEL History

- Precedents: Web Services Flow Language (WSFL) of IBM, Microsoft's XLANG specification
- BPEL4WS 1.0 specification (2002, July), promoted by IBM, Microsoft and BEA Systems
- BPEL4WS 1.1 (2003, May) of SAP and Siebel Systems
- Appearance of *orchestration engines* in accordance with BPEL4WS
- Submission to the OASIS Technical Committee
- Open and official OASIS standard, which gives it a new name: WS-BPEL 2.0
- Specification: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` (2007, April)

WS-BPEL Standard



OASIS

Web Services Business Process Execution Language Version 2.0

OASIS Standard

11 April 2007

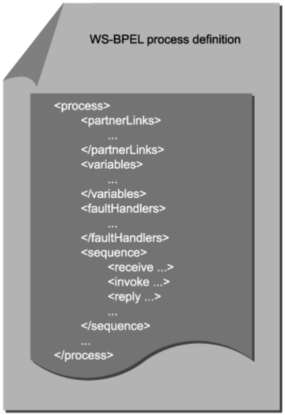
Specification URIs:
This Version:
http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.doc
http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

Previous Version:
http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html
http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.doc
http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf

Latest Version:
http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.doc
http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

Technical Committee:

Objectives



```
<process>
    <partnerLinks>
        ...
    </partnerLinks>
    <variables>
        ...
    </variables>
    <faultHandlers>
        ...
    </faultHandlers>
    <sequence>
        <receive ...>
        <invoke ...>
        <reply ...>
        ...
    </sequence>
    ...
</process>
```

To acquire a good understanding of how a BPEL process can formally be described

Definition structure of a common WS-BPEL process

The element `process`

The root element of one specification

It is assigned a name value with:

- Attribute `name`
- Establishment of namespaces, related to the definition of a process

Syntax of the element `process`

```
<bpel:process name="BookstoreABPEL"
    targetNamespace="http://packtpub.com/Bookstore/BookstoreBPEL"
    suppressJoinFailure="yes"
    xmlns:tns="http://packtpub.com/Bookstore/BookstoreABPEL"
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
         executable">
    <!-- To import the WSDL client -->
    <bpel:import location="BookstoreABPELArtifacts.wsdl"
          namespace="http://packtpub.com/Bookstore/BookstoreABPEL"
          importType="http://schemas.xmlsoap.org/wsdl/" />
    <partnerLinks> ...
    </partnerLinks>
    <variables> ...
    </variables>
    <sequence> ...
    </sequence>
    ...
</process>
```

The `partnerLinks` elements

PartnerLink

Define what services or other processes the *bpel*–process is going to be connected to

- List of participating services within this BPEL process
- A 'PartnerLink' is a substitution parameter in which we put 'the things' to which the *bpel process* is talking to

The `partnerLinks` elements–II

Characteristics of `partnerlinks`

- It is similar to an instance of the recipient WS
- Corresponds to the WSDL `portType` syntactic element that we defined for a WS
- The partner services act as process services, which are in charge of calling the "physical"process that actually serves each one
- The `partner` services are called by the process linked to the service

The `partnerLinks` elements– II

```
<!-- PARTNERLINKS                                                        -->
<!-- List of participating services in this BPE process                  -->
<bpel:partnerLinks>
        <!-- The role 'client' represents the sollicitor of this service -->
            <bpel:partnerLink name="client"
                    partnerLinkType="tns:BookstoreABPEL"
                    myRole="BookstoreABPELProvider" />
</bpel:partnerLinks>
```

Atributo `partnerRole`

The BPEL process understands or implements the WS defined in this attribute

The `partnerLinks` elements– III

Contents of a `partnerLink`

- `myRole`: establishes the role to be played as service's provider
- `partnerRole`: associated service that the service-process will be invoking
- The attributes `myRole` and `partnerRole` can be used by the same element `partnerLink`

A `partnerLink` building example

```
<partnerLinks>
   <partnerLink name="client"
      partnerLinkType="tns:TypeWorksheetSubmission"
      myRole="ServiceProviderTypeWorkSheetSubmission"/>
   <partnerLink name="Invoice"
      partnerLinkType="inv:InvoiceType"
      partnerRole="ServiceProviderInvoices"/>
   <partnerLink name="Worksheet"
      partnerLinkType="tst:WorksheetType"
      partnerRole="ServiceProviderWorksheet"/>
   <partnerLink name="Employee"
      partnerLinkType="emp:EmployeeType"
      partnerRole="ServiceProviderEmployees"/>
   <partnerLink name="Notification"
      partnerLinkType="not:NotificationType"
      partnerRole="ServiceProviderNotification"/>
</partnerLinks>
```

The `partnerLinkType` element

Characterístics

- These constructs are embedded in WSDL documents of any associated service
- Identification of `portType` elements of WSDL for each associated service involved in a process definition
- They identify the WSDL ports referenced by the `partnerLink` elements inside a (BPEL) process

The `partnerLinkType` element– II

- Multiple `partnerLink` elements can reference the same `partnerLinkType`:
- All associated services can therefore use the same `portType` elements of the process service
- As a result, a `partnerLinkType` will have one or two descendant role-elements
- 1 role to differentiate each ïnstance"that performs the service, using one of the attributes: `myRole` or `partnerRole`:
  - `myRole` (provides) and
  - `partnerRole` (associated)

The `partnerLinkType` element– III

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DEFINITION OF THE PARTNER LINK TYPE
(in the file XXXArtifacts.wsdl)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<plnk:partnerLinkType name="BookstoreABPEL">
        <plnk:role name="BookstoreABPELProvider"
                portType="tns:BookstoreABPEL"/>
 </plnk:partnerLinkType>
```

Variables

Purpose

They are of use for holding data in BPEL processes Each variable can hold 1 *XSD value* or 1 *WSDL message*

Use

Variables are used for providing parameter passing (input/output) at the *endpoint*s of one WS

```
<xs:element name="color"> <!-- example: XSD Definition   -->
        <xs:complexType><!--Enumerated Value-->
                <xs:choice>
                        <xs:element name="green"/>
                        <xs:element name="red"/>
                        <xs:element name="blue"/>
                </xs:choice>
        </xs:complexType>
</xs:element>
```

The element `variables`

The construct `variables`

- Storage of the state information connected with the immediate work-flow logics
- Location of complete *messages* (`messageType`), of formatted *data sets* (`element`), and *types of schema* (XSD definitions)
- The information in this construction is subsequently retrieved during the completion of the process

The element `variables` – II

attribute `messageType`

It is defined for each input and output message processed by the process definition
The value associated to this attribute is the name of the message that is in the associated process definition

The element `variables` – III

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

The variable `helloWorld` is declared as one WSDL message-container of type `print:PrintMessage`

The element `variables` – IV

heir-elements `variable` used by the “BookstoreABPEL” process

```
<!-- VARIABLES: List of messages and                    -->
<!-- XML documents used in this BPEL process -->
<!-- ========================================== -->
<bpel:variables>
  <!-- References the menssage passed on as input in the
       beginning -->
  <bpel:variable name="input"
            messageType="tns:BookstoreABPELRequestMessage"/>
  <!-- References the menssage that will be returned to the
       caller -->
  <bpel:variable name="output"
            messageType="tns:BookstoreABPELResponseMessage"/>
</bpel:variables>
```

The element `variables` – V

`type` attribute

Of use for specifying some type of XSD schema: `'xsd:string'`, `'xsd:integer'`, ... of XML

```
<variables>
        <variable name="response" type="xsd:string"/>
        <variable name="offer" type="xsd:float"/>
    </variables>
```

The functions `getVariableProperty` and `getVariableData`

These are internal to WS-BPEL

`getVariableProperty(variable name, property name)`

It allows to retrieve values of global properties of the variables

`getVariableData(variable name, part name, location path)`

It allows other parts of the process logic to access to information on the state of data stored in the variables.
To retrieve message data saved in variables

BPEL 2.0 dynamic initiation of variables

Although the following code is valid, however, some orchestration engines do not accept variable initialization on-line

```
<variables>
    <variable name="response" type="xsd:string">
        <from>"I'm not interested"</from>
    </variable>
    <variable name="offer" type="xsd:float">
        <from>100</from>
    </variable>
</variables>
```

Initialization of variables in BPEL 2.0 - II

```
<variables>
   <variable name="response" type="xsd:string"/>
   <variable name="offer" type="xsd:float"/>
</variables>
<sequence>
   <receive createInstance="yes" .../>
...
<assign name="initialization">
   <copy>
      <from>100</from>
      <to variable="offer"/>
   </copy>
   <copy>
      <from>"I'm␣not␣interested"</from>
      <to variable="response"/>
   </copy>
</assign>
```

Elements of data updating and movement

Use of these elements

This set of elements simply gives us the ability to copy values between process variables, which allows us to pass around data throughout a process as information is received and modified during the process execution

Characteristics

The copy construct can process a variety of data transfer functions (for example, only a part of a message can be extracted and copied into a variable). `from` and `to` elements also can contain optional part and query attributes that allow for specific parts or values of the variable to be referenced

The `assign` element

Fundamental idea

- Variable handling is made at the *endpoint*s of a WS
- or by assignments

```
<assign>
  <copy>
    <from><literal>HelloWorld</literal></from>
    <to>$helloWorld.value</to>
```

This example shows how a literal string value is assigned to the variable `helloWorld`; this variable is a WSDL message that has a part called 'value'

The `assign` element– II

- The syntax of the variables follows the one of `XPATH` expressions
- The symbol '.' is the separator of the WSDL-message part
- A separator is used to specify a sub-element within complex types: `helloWorld.value/sub_value`

`copy`, `from` and `to` elements

Within an `assign` command-element, the *payload* type is pointed out by the `copy` command (to 1 message-variable)

```
<bpel:assign validate="no" name="DetermineStock">
        <bpel:copy>
        <bpel:from>
          <bpel:literal>
                <tns:BookDataResponse
                xmlns:tns="http://packtpub.com/Bookstore/
                    BookstoreABPEL"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                    instance">
                <tns:BookISSN>tns:BookISSN</tns:BookISSN>
                <tns:StockQuantity>0</tns:StockQuantity>
                </tns:BookDataResponse>
          </bpel:literal>
        </bpel:from>
        <bpel:to variable="output" part="payload"></bpel:to>
</bpel:copy>  ....
```

The elements `copy`, `from` and `to`–II

Within an `assign` command, the contents of one expression and one variable are both copied to 2 different message variables

```
... <bpel:copy>
<bpel:from expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:
    sublang:xpath1.0">
    <![CDATA[number(5)]]>
    </bpel:from>
   <bpel:to part="payload" variable="output">
   <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
        sublang:xpath1.0">
    <![CDATA[tns:StockQuantity]]></bpel:query>
    </bpel:to></bpel:copy>
...
```

The elements `copy`, `from` and `to`– III

Within an `assign` command, the contents of one variable and one expression are copied to 2 different message variables

```
... <bpel:copy>
        <bpel:from part="payload" variable="input">
      <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
          sublang:xpath1.0">
          <![CDATA[tns:BookISSN]]></bpel:query>
    </bpel:from>
    <bpel:to part="payload" variable="output">
          <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel
              :2.0:sublang:xpath1.0">
      <![CDATA[tns:BookISSN]]>
      </bpel:query>
    </bpel:to>
   </bpel:copy>
</bpel:assign>
```

The `sequence` element

Sequential construction

WS-BPEL provides numerous activities that can be used to express the work-flow logics within the definition of a process. The following descriptions of (WS-BPEL) elements explain the fundamental set of activities that are used as part of the case studies that will be used during the course

The sequence element - II

A building *skeleton* of one sequence command that contains only some of the elements of WS-BPEL

```
<sequence>
   <receive> ...
   </receive>
   <assign>
      ...
   </assign>
   <invoke>
      ...
   </invoke>
   <reply>
      ...
   </reply>
</sequence>
```

The `receive` element

Provided service specification

The `receive` element allows us to establish the information one process of a service expects upon receiving a request from an external client partner service. In this case, the process service is viewed as a *service provider* waiting to be called.

The `receive` element– II

`receive` element attributes

`partnerlink`	The client-service associated through its correspondent `partnerLink`
`portType`	`portType` element of the process of the service that expects to receive the request
`operation`	Service operation the process that receives the request
`variable`	The incoming request message is saved here
`createInstance`	If the value is “yes”, receiving a request will create a new process’ instance to serve that request

The `receive` element– III

The `receive` element used by the process "BookstoreABPEL" to describe the associated customer service that causes the process to start

```
<!-- Receive the caller's entry.
Nota: This maps to the operation defined in BookstoreABPEL.wsdl
    -->
<bpel:receive name="receiveInput" partnerLink="client"
          portType="tns:BookstoreABPEL"
         operation="getBookData" variable="input"
    createInstance="yes"/>
```

Counterpart of the `receive` element

The `reply` element

Where a receive element is programmed, there must be a response element when a synchronous exchange is being rendered. The reply element is responsible for setting the return details of the response message to the requesting client partner service. Because this element is associated with the same `partnerLink` element as the receive element, it repeats a number of the common attributes between them.

The `reply` element– II

Attributes of `reply` element

`partnerlink`	The same `partnerLink` element set in the `receive` element
`portType`	The same `portType` element set in the `receive` element that received the request
`operation`	The same `operation` element of the `receive` element
`variable`	The `variable` element of the service process that contains the message to return to the associated service
`messageExchange`	Allows the `reply` command to be associated to one activity capable of receiving a message

The `reply` element– III

One `reply` element that matches with the prior `receive` element

```
<bpel:reply name="replyOutput"
      partnerLink="client"
         portType="tns:BookstoreABPEL"
        operation="getBookData"
         variable="output"/>
```

`switch`, `case` and `otherwise` elements

They allow adding conditional logic to the definition of process services

The `switch` element establishes the scope of the conditional logic to be defined
When a condition attribute resolves to “true,” the activities defined within the corresponding case construct are executed
Several `case` constructions can be nested to verify if several conditions are met, each one depending on a different `condition` attribute.
The element `otherwise` is added as a default clause at the end of the `switch` element

The `switch`, `case` and `otherwise` elements– II

Skeleton of one `case` element where the `condition` attribute uses the function `getVariableData`

```
<switch>
   <case condition=
      "getVariableData('EmployeeResponseMessage',␣'
          ResponseParameter')=0">
      ...
   </case>
   <otherwise>
      ...
   </otherwise>
</switch>
```

The `invoke` element

Fundamental idea

The definition of a BPEL process does not specify exactly what a WS does and how it does it
The information regarding the definition and implementation of a WS is contained in its WSDL file

```
<invoke partnerLink="printerService"
   operation="print"  inputVariable="helloWorld"/>
```

The BPEL process passes the data from `helloWorld`, stored in the variable, to the Web service (WS) `print`.
The specified `partnerLink` tells the BPEL engine the WS address to be called here. The `print` operation specifies what the WS actaully will do and the input variable that the incoming WSDL message must come from the `helloWorld` variable

The `invoke` element– II

Operation of an associated service

The `invoke` construct identifies the operation of an associated service that the process intends to invoke during its execution
The `invoke` element is equipped with five common attributes

```
<invoke name="ValidateWeeklyHours"
    partnerLink="Employee"
    portType="emp:EmployeeInterface"
    operation="GetLimitWeeklyHours"
    inputVariable="PetitionEmployeeHours"
    outputVariable="ResponseEmployeeHours"/>
```

With the `invoke` construct we can have a single activity to handle multiple operations and thus by updating the partners in the process definition we can upgrade the code easily

The `invoke` element– III

Attributes of `invoke` element

`partnerlink`	Name the associated service through its partnerLink-partner. Communicates to the BPEL engine the address of the SW that is invoked
`portType`	Identifies the `portType` element of an associated service
`operation`	Operation to which the process sends the request (It is an operation of the partner service)
`inputVariable`	input message, which is used to communicate with the associated operation (we refer to this attribute as a variable because it is referencing a variable element of WS-BPEL with an attribute `messageType`)
`outputVariable`	The returned value is saved in a separate `variable` This element is used when the communication is based in the MEP request-response

The `faultHandlers`, `catch` and `catchAll` elements

Characteristics

This construct can contain multiple `catch` elements, each of which provides activities that perform exception handling for a specific type of error condition. Faults can be generated by the receipt of a WSDL-defined fault message, or they can be explicitly triggered through the use of the `throw` element. The `faultHandlers` construct can consist of (or end with) a `catchAll` element to house default error handling activities

The `faultHandlers`, `catch` and `catchAll` elements– II

Characteristics

- A WSDL failure message or the use of the clause `throw` may cause service failures
- A construction `faultHandlers` can contain multiple elements `catch` in order to program error handling activities when the latter ones occur
- It can end up with a `catchAll` element that contains default error handling activities

The faultHandlers, catch and catchAll elements– IV

Example of construct faultHandlers containing elements catch and catchAll

```
<faultHandlers>
   <catch faultName="SomethingHappened"
          faultVariable="WorkSheetFails">
      ...
   </catch>
   <catchAll>
      ...
   </catchAll>
</faultHandlers>
```

Different types of handlers

Element	Description
`compensationHandler`	A WS-BPEL process definition can define a compensation process that initiates a series of activities when certain conditions occur to justify a compensation. These activities are kept in the compensationHandler.
`correlationSets`	WS-BPEL uses this element to implement correlation, primarily to associate messages with process instances. A message can belong to multiple correlationSets. Further, message properties can be defined within WSDL documents.
`empty`	This simple element allows you to state that no activity should occur for a particular condition.

Different types of handlers– II

Element	Description
`eventHandlers`	The eventHandlers element enables a process to respond to events during the execution of process logic. This construct can contain onMessage and onAlarm child elements that trigger process activity upon the arrival of specific types of messages (after a predefined period of time, or at a specific date and time, respectively).
`exit`	Execute the terminate element description that follows
`flow`	A flow construct allows you to define a series of activities that can occur concurrently and are required to complete after all have finished executing. Dependencies between activities within a flow construct are defined using the child link element.

Different types of handlers– III

Element	Description
`pick`	Similar to the eventHandlers element, this construct also can contain child onMessage and onAlarm elements but is used more to respond to external events for which process execution is suspended.
`scope`	Portions of logic within a process definition can be subdivided into scopes using this construct. This allows you to define variables, faultHandlers, correlationSets, compensationHandler, and eventHandlers elements local to the scope.
`terminate`	This element effectively destroys the process instance. The WS-BPEL 2.0 specification proposes that this element be renamed exit.

Different types of handlers– IV

Element	Description
`throw`	WS-BPEL supports numerous fault conditions. Using the throw element allows you to explicitly trigger a fault state in response to a specific condition.
`wait`	The wait element can be set to introduce an intentional delay within the process. Its value can be a set time or a predefined date.
`while`	This useful element allows you to define a loop. As with the case element, it contains a condition attribute that, as long as it continues resolving to "true", will continue to execute the activities within the while construct

Web Service “ Printing ” built using the Java binding

Starting from the BPEL process “ HelloWorld ” that passes on a literal into the “ print ” service’s input variable, define one process that does the following:

- Printing operation that prints the literal
- A WSDL file that defines:
    1. How to use the aforementioned WS? (define its API)
    2. How WS is linked to Java code?

Web Service “ Printing ” built using the Java binding - II

Things to define in BPEL 2.0

1. The *target* name space
2. The WSDL messages
3. WSDL PortTypes
4. PortType binging
5. The WSDL service
6. The `PartnerLink` types

(1) Name space

Fundamental idea

Similar to Java packages
BPEL XML and `targetNamespace` *namespaces* are used to discriminate homonyms messages but directed to different WS

```
<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/">
```

WSDL messages and 'port' types created in a WSDL file inherit the *namespace* specified by the attribute '`targetNamespace`'

(2) WSDL messages

Are of use for

The WSDL message definition specifies how containers should be to correctly maintain the data of an invoked WSDL-operation
These are lists of parts, each one of which is of a simple or complex XSD type

(3) WSDL PortTypes

Are of use to

- Describe the API or the interface of the SW itself
- They represent a list of operations, including their input and output parameters, each of which is a predefined WSDL message
- Each of the operations may have associated fault treatment elements

They are like interfaces or Java abstract classes. They do not specify any particular implementation of some kind, but they exactly define what can be done and what comes in and goes out

(4) WSDL PortTypes Binding

Fundamental Idea

- They are of use to specify how a WS is actually implemented
- They describe what is in the ’other side’, with which we deal with when we require a service

Possible bindings for a WS

- SOAP/HTTP A WS implementation would be listening on a specific port and would accept SOAP messages through the HTTP transport layer
- Java Some Java class would be mapped to the port type and used directly as a service implementation

(4) WSDL PortTypes Binding– II

The operation of a WSDL PortType is mapped to a Java method: `print()`.

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

This class is instantiated and when calls are made to the operation, these are resolved resorting to the `print()` method defined in the class

(4) WSDL PortTypes Binding– III

The String type of XSD has been mapped to the Java type `String`

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

Any string of XSD will also be converted to a Java String, as indicated in the binding

(5) Service description with WSDL

It is useful for

- WS Instance specification, which is implemented using a specific link and which is available in a certain address
- The WSDL service is of use for specifying a WSDL port
- The address is particular to each binding.
- The Java binding knows how to interpret the attribute 'className' as a qualified class name and understands how to instantiate the class and solve the WSDL operations with the methods of that class

(6) `PartnerLink` types

Fundamental idea

- They are a BPEL construct, not a WSDL one
- Comply with the BPEL requirement that every instance of a `partnerlink` must be associated with 1 particular WSDL port type

`partnerlink` roles

- `partner role`: to make the BPEL process to communicate with the service
- `my role`: the other clients communicate with this one

Realization of the solution to the *printing* service

The “Target” Namespace

```
<?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/">
```

The WSDL messages

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

The PortTypes of WSDL

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

Bindings of PortType

```
<binding name="PrintPortWsBinding" 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>
```

The WSDL service

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

The `PartnerLink` types

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

Bibliografía

For more information, additional bibliography, or “simply inspiration” on the subject, you might consult:

http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

Capel, M.I. *Desarrollo de Software y Sistemas Basados en Componentes y Servicios*. Garceta Grupo Editorial, Madrid (1st edition, 2016)