

# Lesson 10 – BPEL Introduction

Service Oriented Architectures Security

Module 1 - Basic technologies

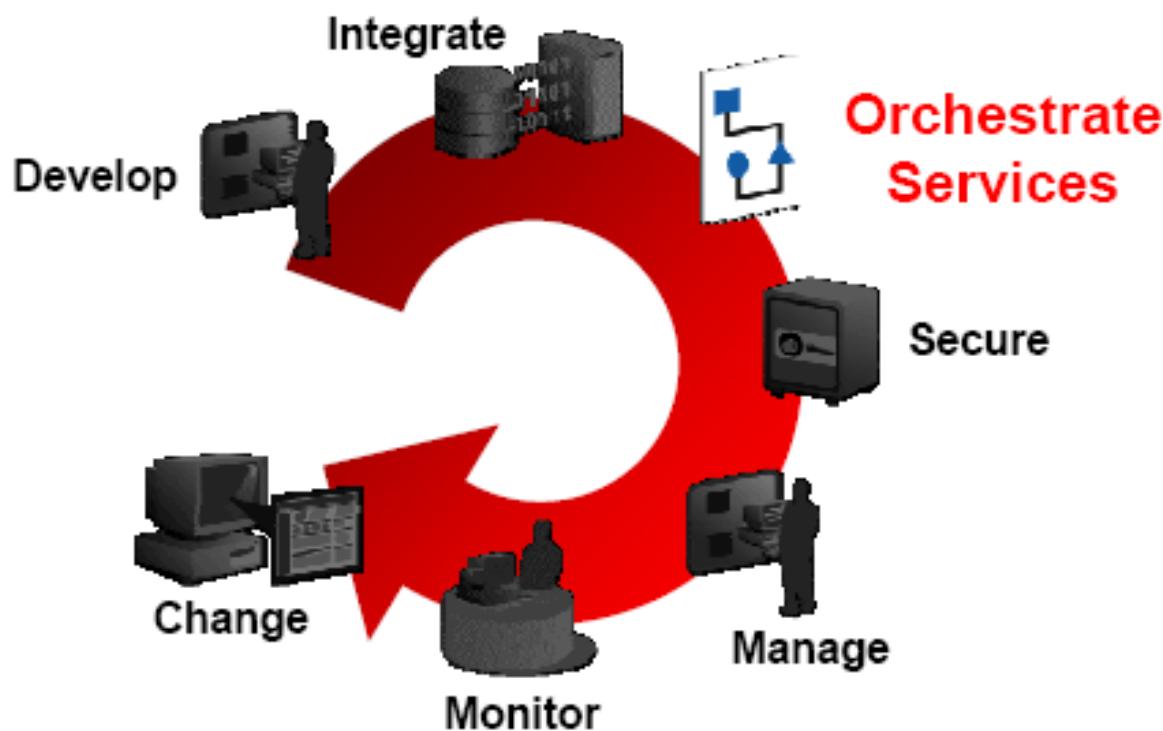
Unit 5 – BPEL

**Ernesto Damiani**

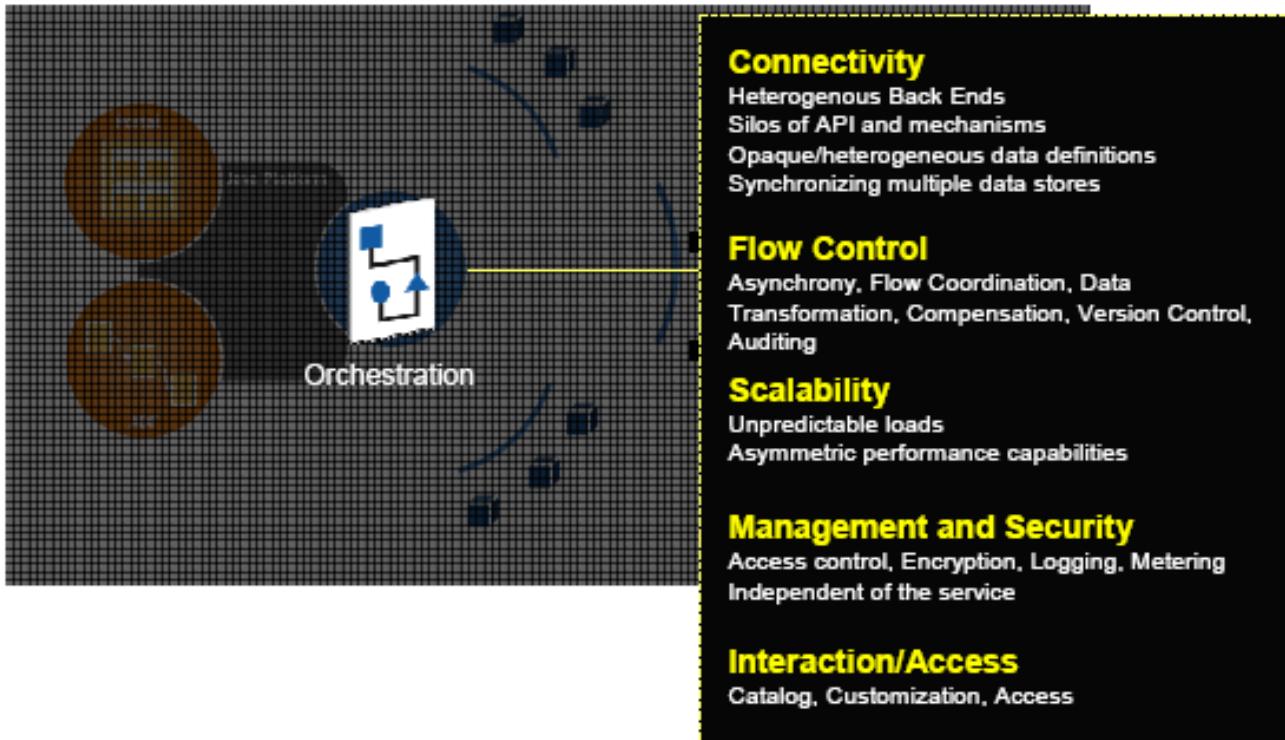
---

Università di Milano

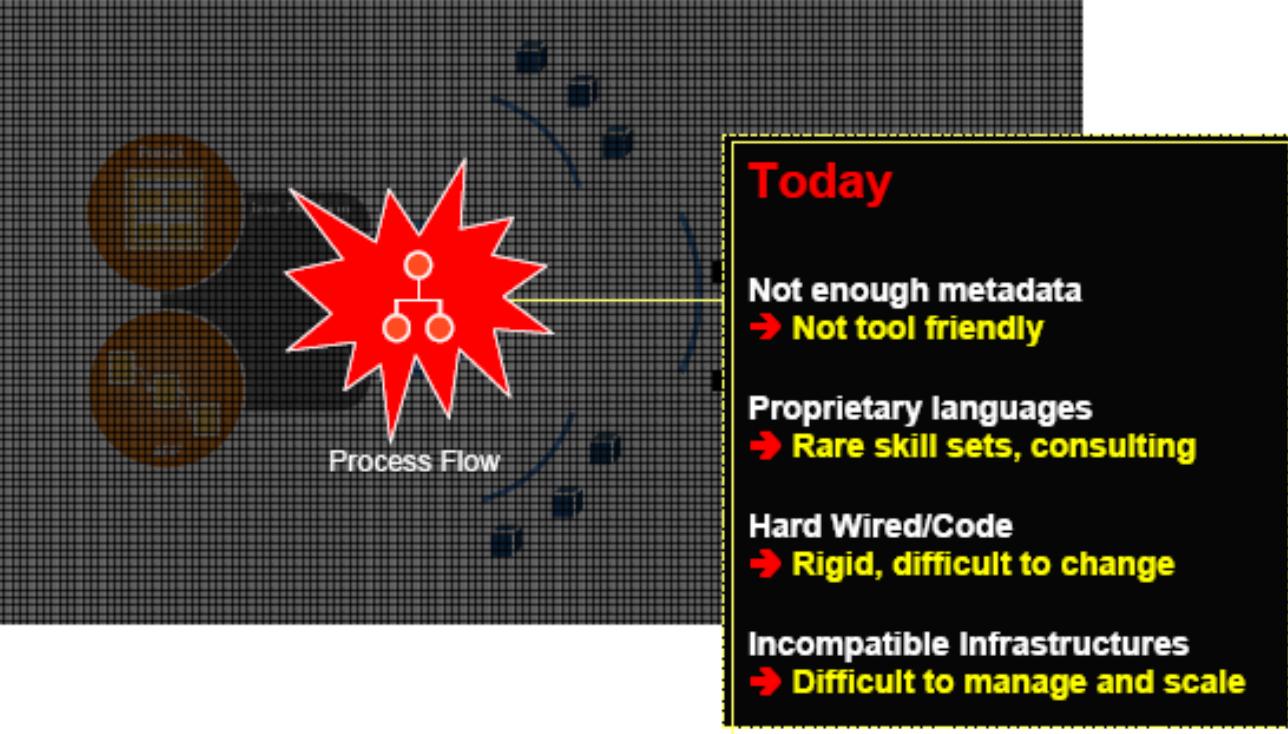
# Service-Oriented Architecture



# Orchestration Requirements



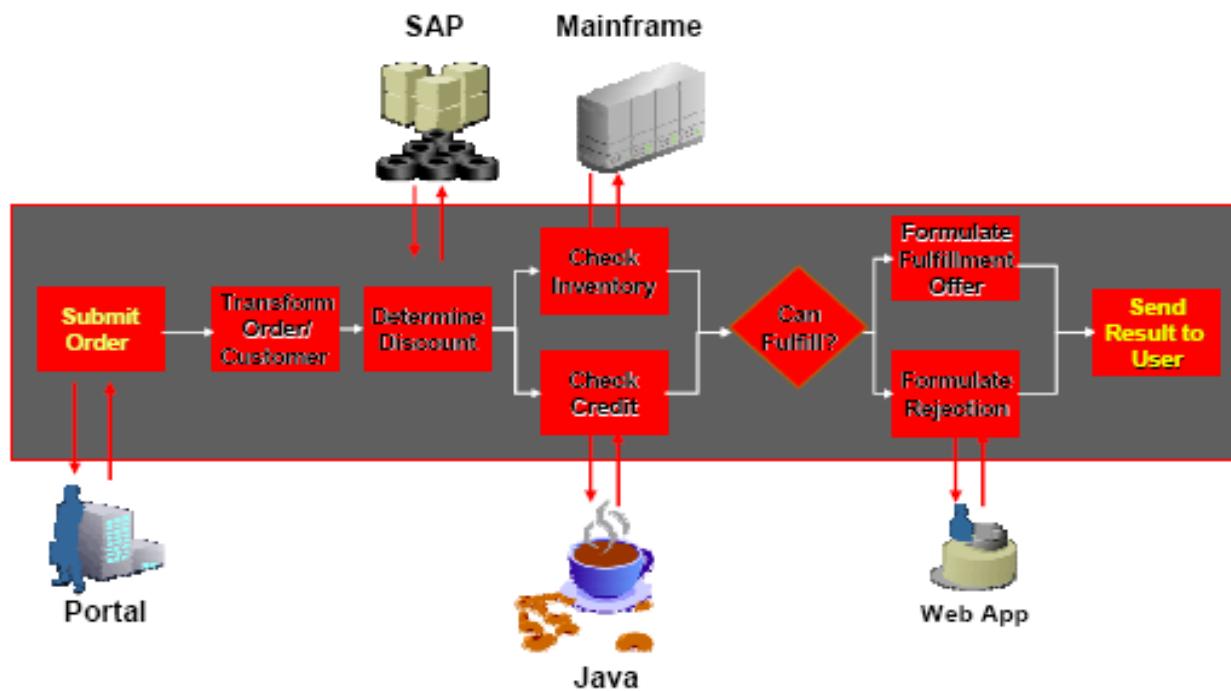
# Orchestration Today



**Today**

- Not enough metadata  
→ Not tool friendly
- Proprietary languages  
→ Rare skill sets, consulting
- Hard Wired/Code  
→ Rigid, difficult to change
- Incompatible Infrastructures  
→ Difficult to manage and scale

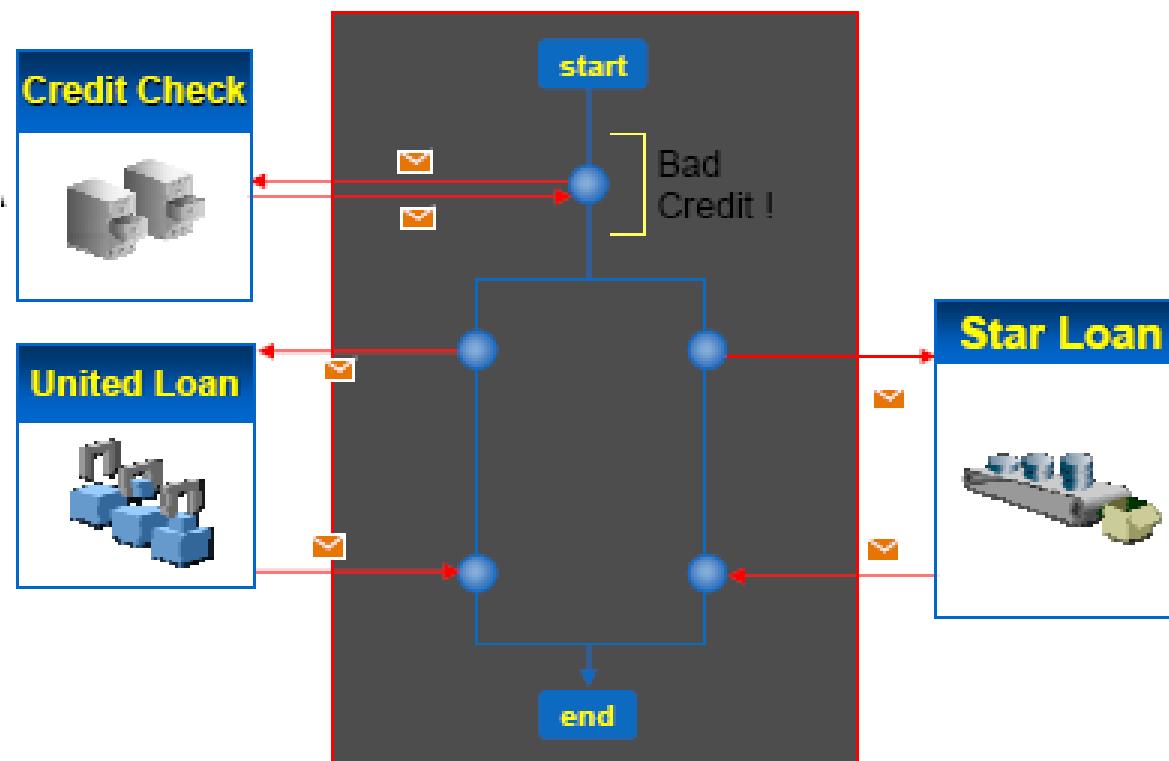
# Process Flow Use Case



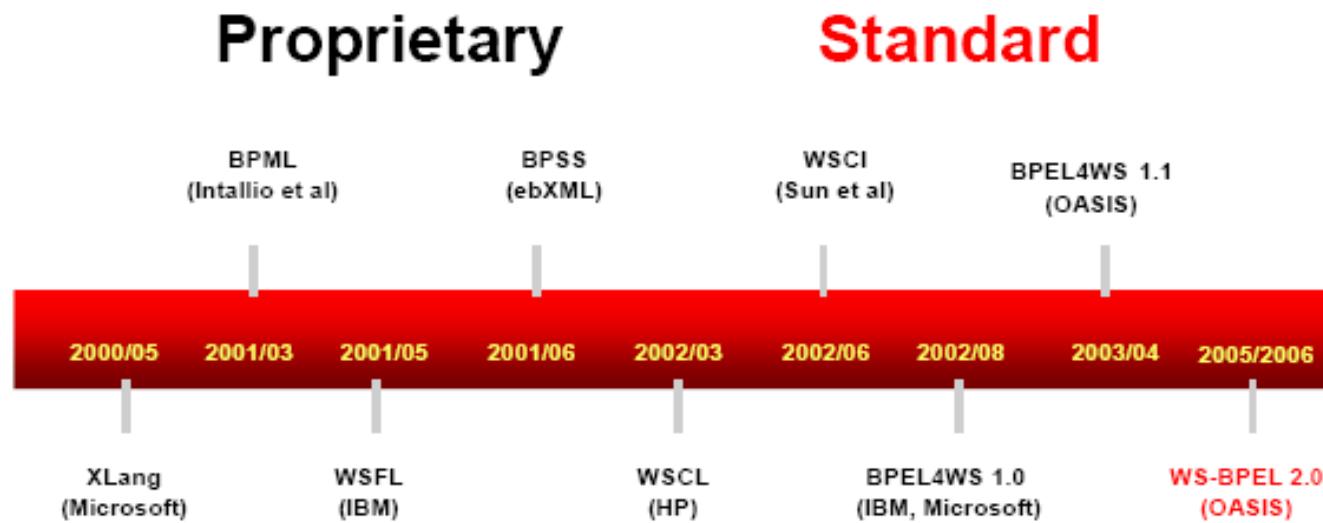
# What is BPEL? (1)

- Markup language for composing a set of discrete services into an end-to-end process flow
  - 10+ years of R&D from
  - MSFT and IBM
  - SOAP but also Java, JCA
  - Rich Flow Semantics
  - Optimized Bindings
  - XPATH+XSLT+XQuery
  - WS-Security
  - A Process is a Service

# What is BPEL? (2)



# BPEL History



# **BPEL (1)**

- BPEL is an XML programming language. As a programming language it has three basic components:
  - Programming logic
  - 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)

## BPEL (2)

- 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>
```

# import wsdl files (1)

```
<import importType="http://schemas.xmlsoap.org/wsdl/  
location="../../test_bucket/service_libraries/tptp_EnginePrinterPor  
t.wsdl" namespace="http://www.eclipse.org/tptp/choreography/20  
04/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
- WSDL files have a similar import mechanism allowing them to import common XSD files and even other WSDL files

## import wsdl files (2)

- In this case, a Hello World web service has been defined in the WSDL file '...tptp\_EnginePrinterPort.wsdl'

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

# Partner Links (1)

- 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

## Partner Links (2)

- 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 process 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

```
<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 (1)

```
<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'

# Variable Assignment (2)

- 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

- <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

# Tutorial: printout service in Java

- So now we have our Hello World BPEL process that passes a string to a printout web service, we need to have a printout web service that will print the string out for us
- In order to use a web service, it must be defined somewhere. That somewhere is in a WSDL file. The WSDL file describes how to use the web service (the API to the web service) and also how that web service is bound. It is possible to split up a WSDL file so that a web service can be described once but implemented multiple times in lots of different ways but in this simple example we'll just stick to one description - one implementation.

# 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="PrintPortWsdlBinding" type="tns:Print">
    <java:binding>
      <formatTypeMapping encoding="Java" style="Java">
        <formatTypeMap typeName="xsd:string" formatType="java.lang.String"/>
      </formatTypeMapping>
      <operation name="print">
        <java:operation methodName="print" parameterOrder="value"/>
      </operation>
    </java:binding>
  </binding>
  <service>
    <port name="JavaPrintPort" binding="tns:PrintPortWsdlBinding">
      <java:address className="org.eclipse.tptp.choreography.engine.internal.extensions.wsdlbinding.wsdl.ports.EnginePrinterPort"/>
    </port>
  </service>
  <partnerLinkType name="printLink">
    <role name="print@service" portType="tns:Print"/>
  </partnerLinkType>
</definitions>
```

# target namespace (1)

- 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

# target namespace (2)

- 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 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

# target namespace (3)

- 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 namespaces 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 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 (1)

- 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 and '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.

## WSDL Port Types (2)

- 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 'printMessage' 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 (1)

- 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

# WSDL Port Type Bindings (2)

- 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 the '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

- 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'.
- 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 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 two 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 is 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)

# BPEL by Example

