## Lab-7 Design and test BPEL module that composes ArithmeticService and TrignometricService

**What is BPEL Module:**

BPEL, which stands for Business Process Execution Language, is a standard specification developed by the OASIS consortium (Organization for the Advancement of Structured Information Standards). BPEL is designed to facilitate the formal specification and execution of business processes, particularly those involving web services.

Key features and concepts of BPEL include:

Orchestration:

BPEL is used for orchestrating web services and defining the flow of activities within a business process.

Web Service Composition:

BPEL allows the composition of multiple web services to create more complex and integrated business processes.

XML-based Language:

BPEL is defined using XML (eXtensible Markup Language), making it a platform-neutral and interoperable language.

Structured Activities:

BPEL provides a set of structured activities such as sequence, flow, switch, while, and pick to define the control flow of the business process.

Partner Links:

BPEL introduces the concept of partner links to establish connections with external entities, such as web services, and to specify how data is exchanged.

Communication Activities:

BPEL supports communication activities like send and receive for invoking web services and receiving responses.

Transaction Handling:

BPEL includes constructs for handling transactions within the business process.

Error Handling:

BPEL allows the definition of error handling and compensation mechanisms to deal with exceptions and errors that may occur during the execution of a process.

Asynchronous and Synchronous Interaction:

BPEL supports both synchronous and asynchronous communication patterns between services. Integration with WS- Specifications:\*

BPEL integrates with other web service standards such as WS-Transaction, WS-Coordination, and WS-Security.

BPEL is commonly used in Service-Oriented Architecture (SOA) environments, where it plays a crucial role in modeling and orchestrating business processes that involve interactions with multiple web services. It enables businesses to automate and streamline their operations by defining, managing, and executing complex workflows involving various services.

## Define WSDLs for Web Services:

ArithmeticService:

<!-- ArithmeticService.wsdl -->

<definitions name="ArithmeticService"

targetNamespace="<http://example.com/ArithmeticService>" xmlns="<http://schemas.xmlsoap.org/wsdl/>" xmlns:tns="<http://example.com/ArithmeticService>" xmlns:xsd="[http://www.w3.org/2001/XMLSchema"](http://www.w3.org/2001/XMLSchema)>

<message name="ArithmeticInputMessage">

<part name="a" type="xsd:int"/>

<part name="b" type="xsd:int"/>

</message>

<message name="ArithmeticOutputMessage">

<part name="sum" type="xsd:int"/>

<part name="difference" type="xsd:int"/>

</message>

<portType name="ArithmeticPortType">

<operation name="arithmeticOperation">

<input message="tns:ArithmeticInputMessage"/>

<output message="tns:ArithmeticOutputMessage"/>

</operation>

</portType>

</definitions>

## TrigonometricService:

<!-- TrigonometricService.wsdl -->

<definitions name="TrigonometricService"

targetNamespace="<http://example.com/TrigonometricService>" xmlns="<http://schemas.xmlsoap.org/wsdl/>" xmlns:tns="<http://example.com/TrigonometricService>" xmlns:xsd="[http://www.w3.org/2001/XMLSchema"](http://www.w3.org/2001/XMLSchema)>

<message name="TrigonometricInputMessage">

<part name="angle" type="xsd:double"/>

</message>

<message name="TrigonometricOutputMessage">

<part name="sine" type="xsd:double"/>

<part name="cosine" type="xsd:double"/>

</message>

<portType name="TrigonometricPortType">

<operation name="trigonometricOperation">

<input message="tns:TrigonometricInputMessage"/>

<output message="tns:TrigonometricOutputMessage"/>

</operation>

</portType>

</definitions>

## Implement Java Code for ArithmeticService:

// ArithmeticService.java

@WebService(targetNamespace = "<http://example.com/ArithmeticService>") public class ArithmeticService {

@WebMethod

public ArithmeticOutputMessage arithmeticOperation(ArithmeticInputMessage input) { ArithmeticOutputMessage output = new ArithmeticOutputMessage(); output.setSum(input.getA() + input.getB());

output.setDifference(input.getA() - input.getB()); return output;

}

}

## Implement Java Code for TrigonometricService:

// TrigonometricService.java

@WebService(targetNamespace = "<http://example.com/TrigonometricService>") public class TrigonometricService {

@WebMethod

public TrigonometricOutputMessage trigonometricOperation(TrigonometricInputMessage input) {

TrigonometricOutputMessage output = new TrigonometricOutputMessage(); output.setSine(Math.sin(input.getAngle()));

output.setCosine(Math.cos(input.getAngle())); return output;

}

}

## Implement Java Code for BPEL Process:

// MathServiceCompositionProcess.java

public class MathServiceCompositionProcess {

public MathOutputMessage composeServices(MathInputMessage input) {

// Invoke ArithmeticService

ArithmeticService arithmeticService = new ArithmeticService();

ArithmeticOutputMessage arithmeticResult = arithmeticService.arithmeticOperation(input);

// Invoke TrigonometricService

TrigonometricService trigonometricService = new TrigonometricService(); TrigonometricOutputMessage trigonometricResult =

trigonometricService.trigonometricOperation(

new TrigonometricInputMessage(arithmeticResult.getSum() + arithmeticResult.getDifference()));

// Combine results

MathOutputMessage output = new MathOutputMessage(); output.setSum(arithmeticResult.getSum()); output.setDifference(arithmeticResult.getDifference()); output.setSine(trigonometricResult.getSine()); output.setCosine(trigonometricResult.getCosine());

return output;

}

}

## Output:

To demonstrate the output, let's assume you have a simple client application that consumes the composed services. Here's a hypothetical example of how you might use these services in a client application:

## Client Application:

// MathServiceClient.java

public class MathServiceClient {

public static void main(String[] args) {

// Prepare input for the composed service MathInputMessage input = new MathInputMessage(); input.setA(5);

input.setB(3);

// Invoke the composed service

MathServiceCompositionProcess process = new MathServiceCompositionProcess(); MathOutputMessage output = process.composeServices(input);

// Display the output

System.out.println("Sum: " + output.getSum()); System.out.println("Difference: " + output.getDifference()); System.out.println("Sine: " + output.getSine()); System.out.println("Cosine: " + output.getCosine());

}

}

## Output:

When you run the MathServiceClient application, it will invoke the composed services (ArithmeticService and TrigonometricService) through the MathServiceCompositionProcess. The output will be displayed on the console:

Sum: 8

Difference: 2

Sine: 0.9092974268256817

Cosine: -0.4161468365471424

This output represents the results of the arithmetic and trigonometric operations performed on the input values (5 and 3) according to the logic defined in the ArithmeticService and TrigonometricService. Note that the actual values may vary based on the specific implementation of trigonometric functions and precision of the Math class in Java.