2

Getting Started with Composite Applications

Objectives

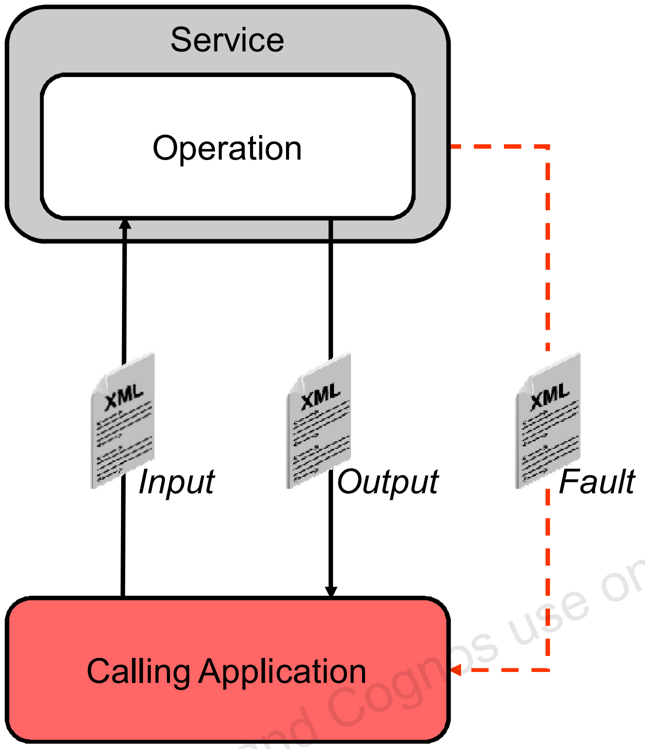
After completing this lesson, you should be able to:

- Describe how services use WSDL files to communicate
- Use Mediator components and File adapters to perform basic receiving, routing, and writing of messages
- Name and describe the contents of some of the files that are created as part of a composite application

Agenda

- How Services Use WSDL Files to Communicate
- Mediator Components
- File Adapters
- Composite Application Files

How Services Communicate



For our purposes, we define a service as software, which is available over the Internet or a network and which exposes functionality as one or more *operations,* or actions, to other applications.

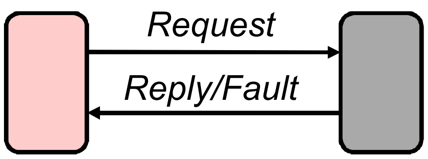
The service and calling application exchange messages. Although there are several different interaction patterns, in a common scenario, the calling application invokes the service and, as part of that invocation, provides input data. The service then either completes its unit of work and returns data or if it is unable to complete the requested operation, it may instead return information about the failure.

In order for the calling application to exchange messages with the service, a means must be defined for the application to provide the required information in an acceptable format, and for the application to interpret the output (or fault) message provided by the service. Although not a strict requirement, XML is commonly used for these *input, output*, and *fault* messages, and XML Schema Definition (XSD) is used to describe the structure of these XML messages.

Synchronous and Asynchronous Interactions

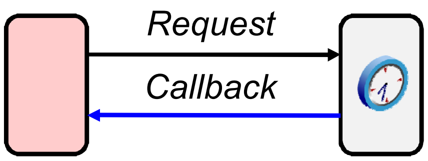
Synchronous request/response

- Real-time response or error feedback
- Client in waiting mode



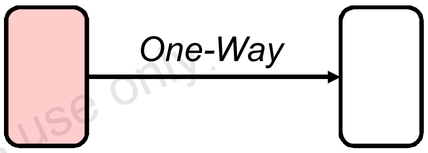
Asynchronous request/callback

- Client free after request submission
- Separate service invocation for response



Asynchronous request only

- Also known as "fire and forget"
- Client free after request submission
- No response message (ACK only)

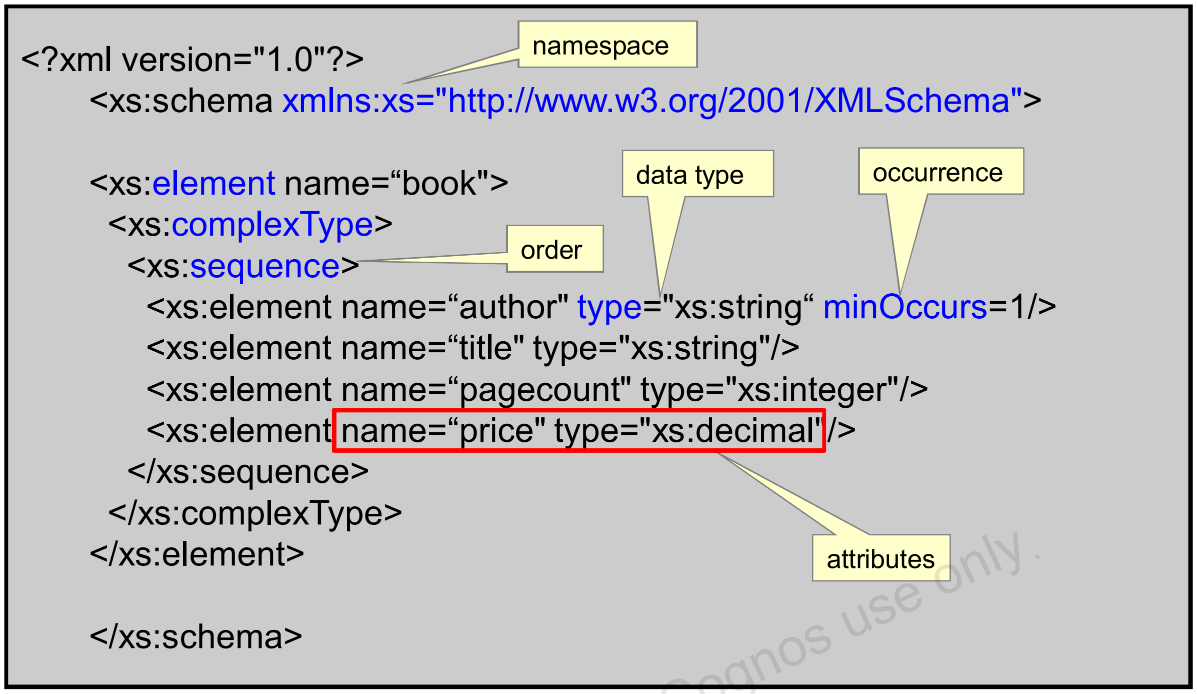


The interaction described in the previous slide included input, output, and (possibly) a fault message. Mediators support both synchronous (request-reply) and asynchronous (both one-way and callback response) interactions.

In a synchronous interaction, the client requests a service, and then waits for a response to the request. While the client waits, the communication channel between the parties is left open until the response occurs. This may be undesirable if large numbers of channels are left open for long periods of time. It may not be needed if the client does not need an immediate response. In these cases, an asynchronous response may be more appropriate.

In an asynchronous interaction, the client invokes the service but does not wait for a response before continuing. Asynchronous operations open a communication channel between the parties, make the request, and close the channel before the response occurs. The response may come at a later time via a callback operation, or not at all for one-way interactions.

Describing a Message with XSD



```
<?xml version="1.0"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="book">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="author" type="xs:string" minOccurs=1/>
       <xs:element name="title" type="xs:string"/>
       <xs:element name="pagecount" type="xs:integer"/>
       <xs:element name="price" type="xs:decimal"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>

    </xs:schema>
```

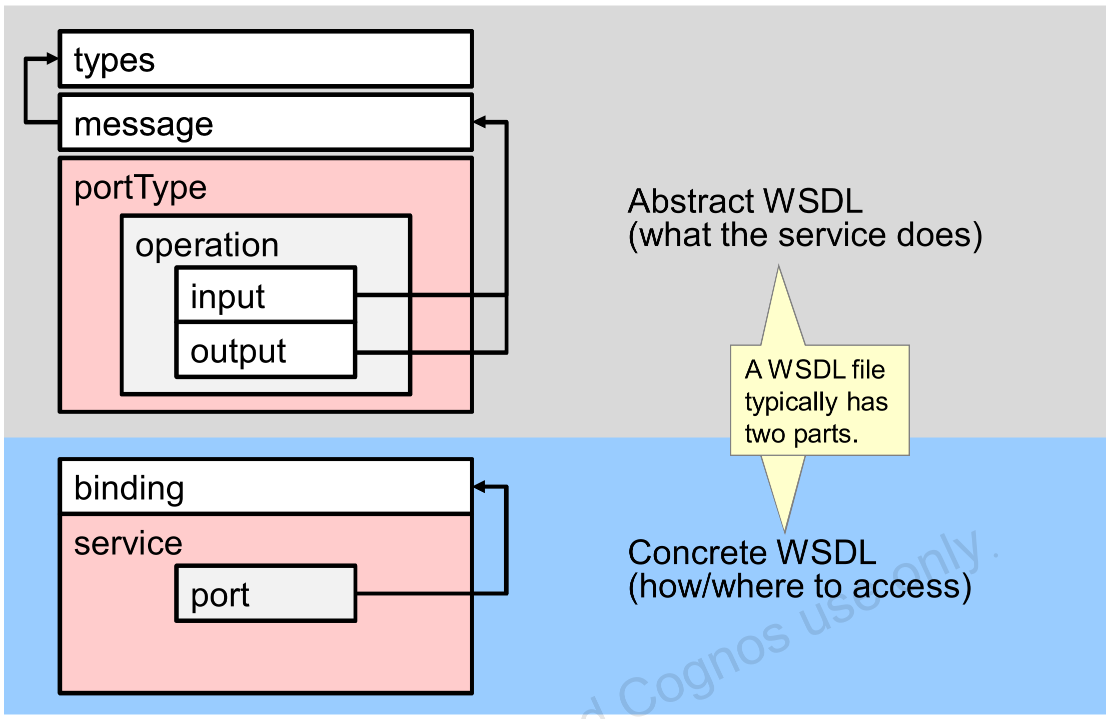
XSD stands for *XML schema definition* and, as its name suggests, an XSD is used to describe (or define) the structure (or schema) of an XML message. In fact, an XSD uses an XML format to provide that description.

XML documents are described in terms of their component pieces, or *elements*. An XML schema definition is used to define those elements. It can be used to define their *sequence*, their attributes, their relationship (parent/child), a permissible number of occurrences, data types, and default or fixed values, if any.

Built-in data *types* (a list of commonly used examples would include integer, double, Boolean, date, and string, among others), simplify the description of permissible document content and data formats in XML schemas. XSD also supports *complex types*, which describe elements that are made up of other elements.

Like other XML documents, the example in the slide includes a *namespace* definition. Namespace definitions are used in XML to allow programs to disambiguate among document elements that may have the same name but are drawn from different sources and thus have different meanings. Subsequent elements in the document begin with their namespace prefix.

Web Services Description Language



Describing the structure of messages to be exchanged between a calling application and a service is important, but is not enough to enable the applications to converse. For this, a *Web Service Description Language* or WSDL document is used. This (XML) description includes the format of input, output, and fault messages, and the data types they use. It also includes the names of operations that the service provides, and how and where to find the service.

The information in a single WSDL document is organized into two parts: the *abstract* definition (what a web service does) and the *concrete* definition (how and where to access the service).

Abstract definitions include *types, message, operation,* and *portType.*

- **Types** define the data types used in messages. These types are often drawn from the XML Schema Language.
- **Messages** describe the parts of the input, output, and fault messages that are exchanged with the calling program.
- **Operations** provide a name for the action performed on messages.
- **PortTypes** group messages with operations.

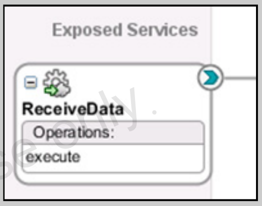
Abstract WSDL

```
<?xml version= '1.0' encoding= 'UTF-8' ?>
<wsdl:definitions
    name="ReceiveData"
    targetNamespace="http://oracle.com/sca/soapservice/Basics/HelloWorld/ReceiveData"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:inp1="http://www.example.org/ns/porder"
    xmlns:tns="http://oracle.com/sca/soapservice/Basics/HelloWorld/ReceiveData"
  >
  <wsdl:types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" >
      <import namespace="http://www.example.org/ns/porder" schemaLocation="xsd/po.xsd" />
    </schema>
  </wsdl:types>
  <wsdl:message name="requestMessage">
    <wsdl:part name="part1" element="inp1:PurchaseOrder"/>
  </wsdl:message>
  <wsdl:portType name="execute_ptt">
    <wsdl:operation name="execute">
      <wsdl:input message="tns:requestMessage"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

1

2

3

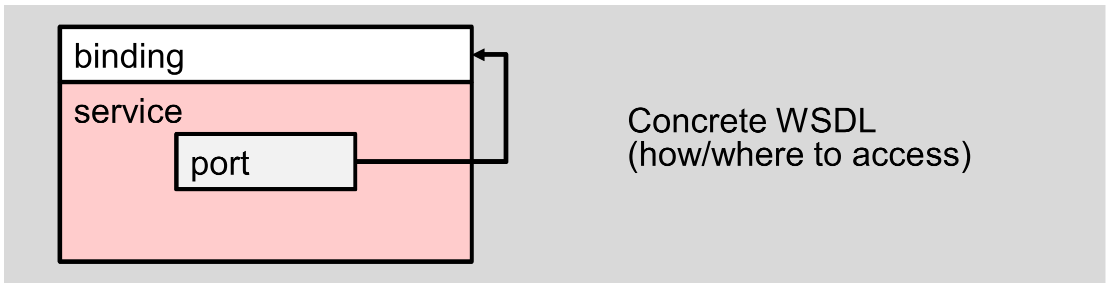


In the sample abstract WSDL file shown in the slide:

1. Section 1 includes an `import` statement and a reference to the `po.xsd` file. Although the data types could have been included directly in this WSDL file, it is a better practice (because it promotes reuse) to separate the data type definitions into an XSD file, and then to import the file.
2. Section 2 describes the `requestMessage` message. The description includes the name of the (single, in this example) part, and the name of the element that describes the message structure of that part. The message element `PurchaseOrder` is fully described in the imported `po.xsd` document.
3. Finally, section 3 describes the port type `execute_ptt`, which groups a specific operation (`execute`) with the message `requestMessage`.

Note: The number and order of messages in the portType definition of a WSDL is important. Because you see only a single message listed in this example, you know that operation `execute` is a one-way operation. If you saw a second message, you could assume that the operation returned a response upon completion.

Concrete WSDL



The WSDL documents generated by JDeveloper for local services (those within the WebLogic Server) may not include the concrete portion of the WSDL. However, the WSDL documents for services deployed outside the WebLogic Server will include the concrete elements of the WSDL, which provide additional information about how and where to access the service:

- The ***binding*** describes how a given portType operation is transmitted, such as HTTP or SOAP (that is, the protocol), and provides information about where the service is located.
- The ***port*** specifies a combination of a network address and a binding, which constitute an endpoint.
- A ***service*** groups ports together. A service reveals to a calling program where to access the web service, and through which port. It also describes how the communication messages are defined.

Quiz

Abstract and concrete WSDLs are two separate but related files that describe the conversation between a service and the calling application.

a. True

b. False

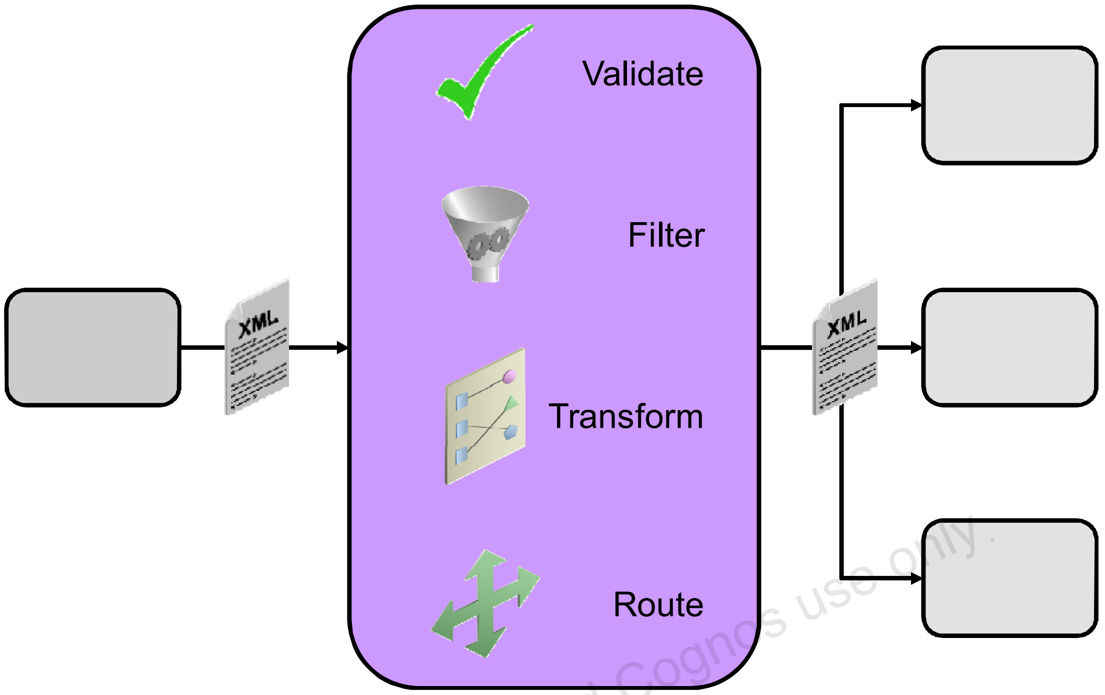
Answer: b

False. The information in a single WSDL document is organized into two parts: the abstract definition (what a web service does) and the concrete definition (how and where to access the service).

Agenda

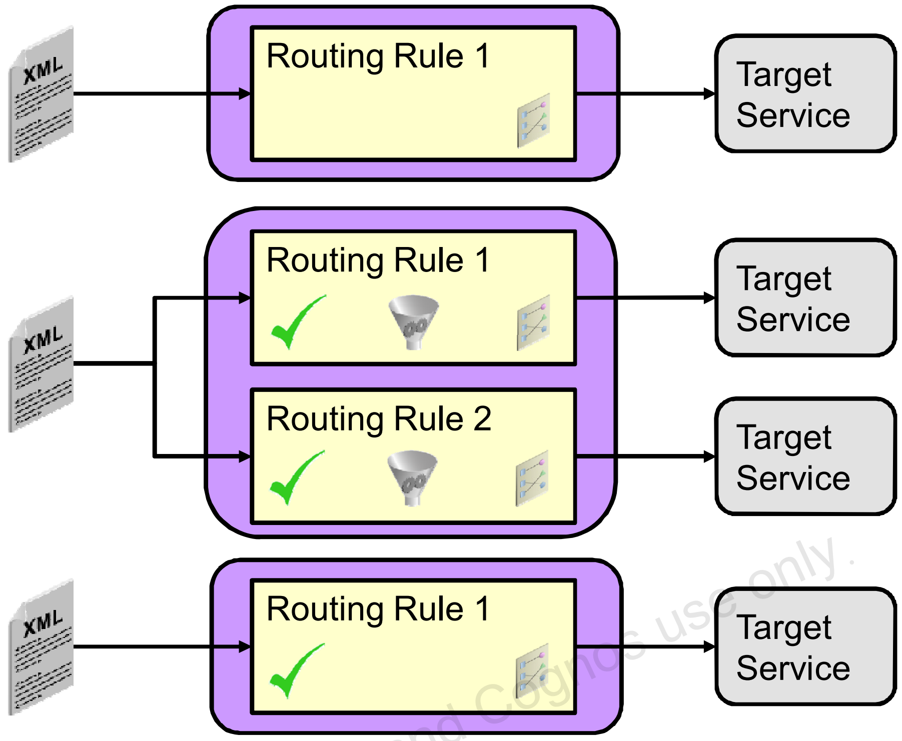
- How Services Use WSDL Files to Communicate
- Mediator Components
- File Adapters
- Composite Application Files

Mediator Components: Introduction



Data flowing through a composite application may need to be filtered, transformed, validated, and routed from a source to one or more target services. *Mediator* components (or simply *Mediators*) offer these capabilities. A Mediator provides a lightweight mediation framework to manage data between clients and services.

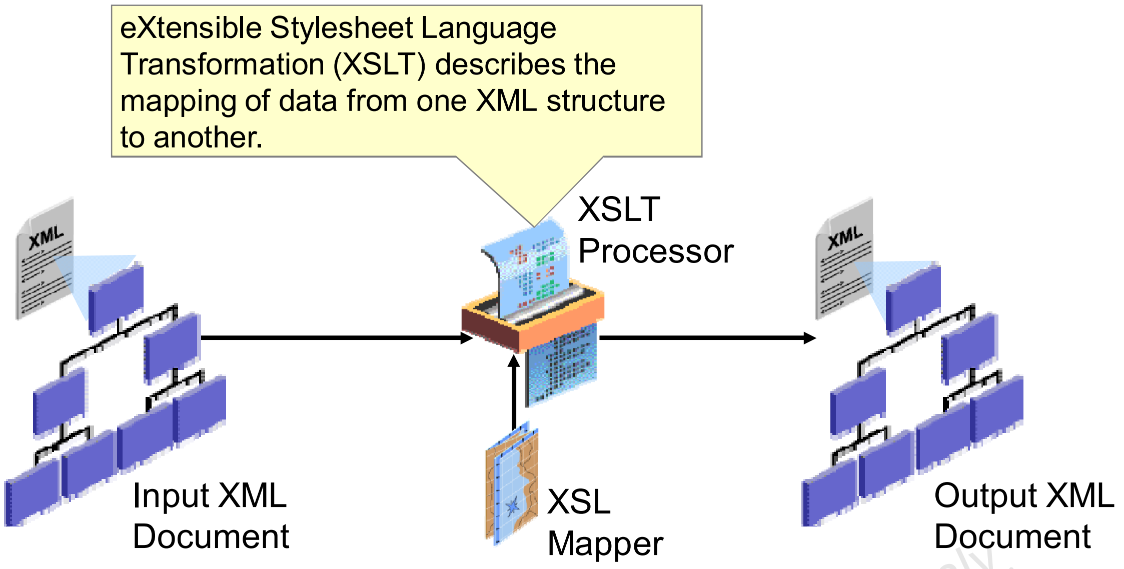
Routing Data



Routing data, which is the primary purpose of a Mediator component, is accomplished by adding one or more *routing rules*. Each routing rule specifies a target service for the data. Routing rules allow you to specify validation, filters, and transformations. Routing rules can handle returned responses, callbacks, faults, and timeouts.

- In the first example in the slide, a routing rule is applied to an incoming XML message. A transformation is performed and the message is passed to a specified destination. Transformations are a required part of a routing rule configuration.
- In the second example, two routing rules are defined. Each is applied to the incoming message. In the diagram, each rule has filtering and validation defined. These are defined in addition to the message transformation. For each routing rule, the message must pass validation, meet filter criteria, and be transformed before it can be routed to the specified destination.
- The third example shows a routing rule with validation and a transformation defined, but no filter. Filters and validations are always optional in the configuration of a routing rule.

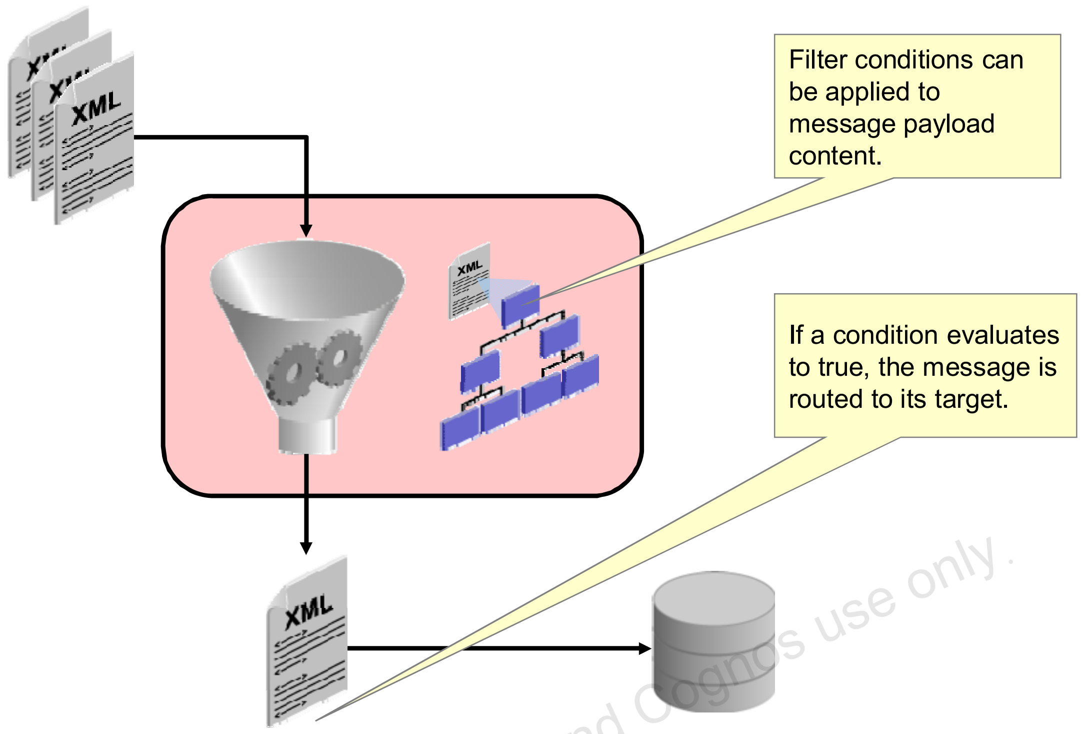
Transforming Data


eXtensible Stylesheet Language Transformation (XSLT) describes the mapping of data from one XML structure to another.
XSLT Processor
XML
Input XML Document
XSL Mapper
XML
Output XML Document

Mediator routing rules support data transformation from one XML schema to another. This enables data interchange and enrichment between applications by using different schemas. For example, the source XML schema may have last name and first name as separate values. The target XML schema may have the full name (both first and last) stored as one value. Through transformations, you can concatenate the first and last names from the source data before it reaches the target data.

Transformations are defined in XSL files that can be visually designed by using the XSLT Mapper (a visual editing tool), which is provided by Oracle JDeveloper 12*c*. The XSL file is deployed with the composite application so that the Mediator engine can process the mapping rules in an XSL processor that is available in the runtime environment.

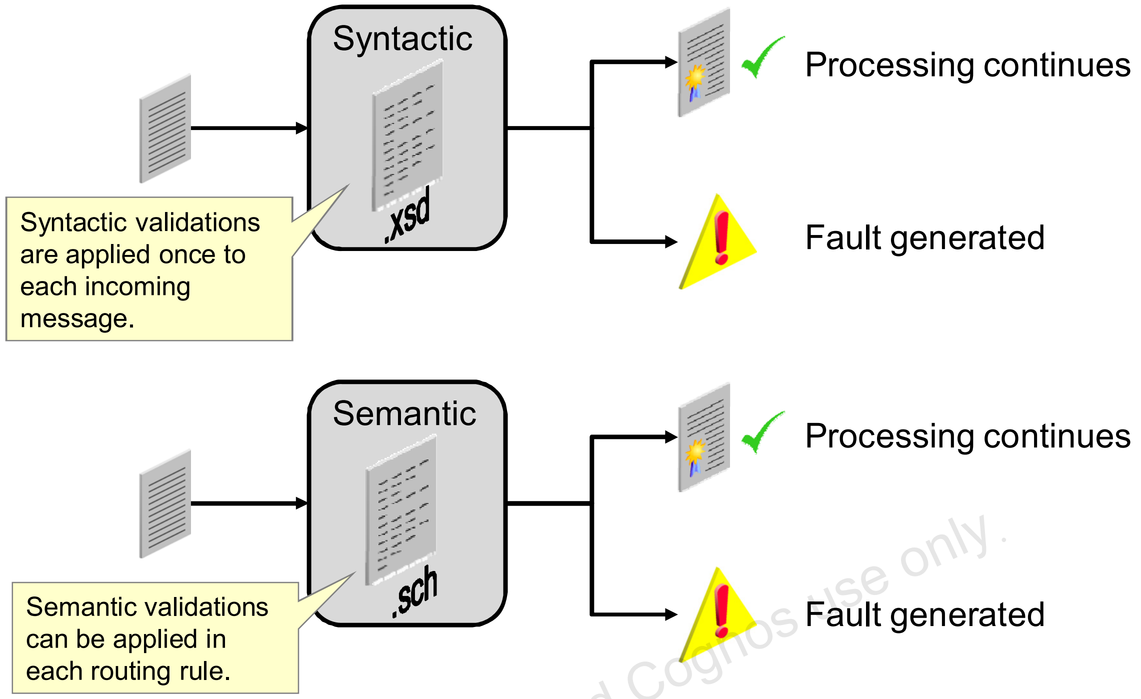
Filtering Data


Filter conditions can be applied to message payload content.
If a condition evaluates to true, the message is routed to its target.
XML

Mediator routing rules provide content (payload)–based filtering criteria to ensure that data is routed to a suitable target. If the condition evaluates to true for a given message instance, the data is routed to the target defined in the routing rule.

For example, a Mediator receives a file from an application or service that contains data about new customers. Based on the country mentioned in the customer's address, data can be routed to the database that stores data for that particular country.

Validating Data

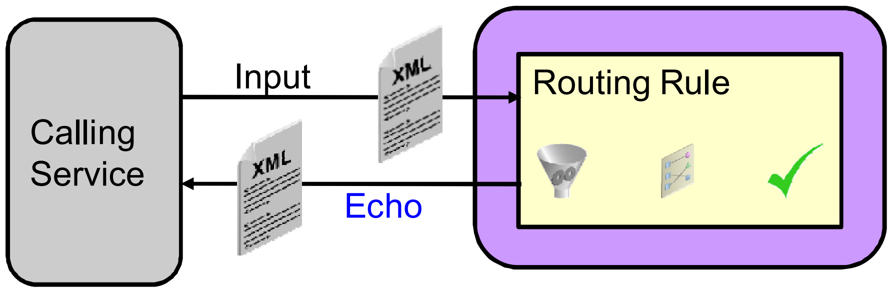


Syntactic
.xsd
Syntactic validations are applied once to each incoming message.
Processing continues
Fault generated
Semantic
.sch
Semantic validations can be applied in each routing rule.
Processing continues
Fault generated

Mediator routing rules provide support for two levels of validation of incoming message payloads. The first level uses an XSD file to validate the payload structure, ensuring that it has the correct element hierarchy and relationships based on the XSD definition. The second level of validation is semantic. It uses a *schematron* (`.sch`) file to validate the message content. You can specify schematron files for each inbound message part and Mediator can execute schematron validations for those parts. For example, you can check whether a price element value contains suitably formatted numeric data and is within a range of values. If either syntactic or semantic validation fails, the routing rule is not processed and a fault is generated, which is returned to the caller of a synchronous operation.

Note: Enabling validation incurs some overhead because of the execution time required to perform the validation rules. For more information about Schematron specifications, refer to http://www.schematron.com.

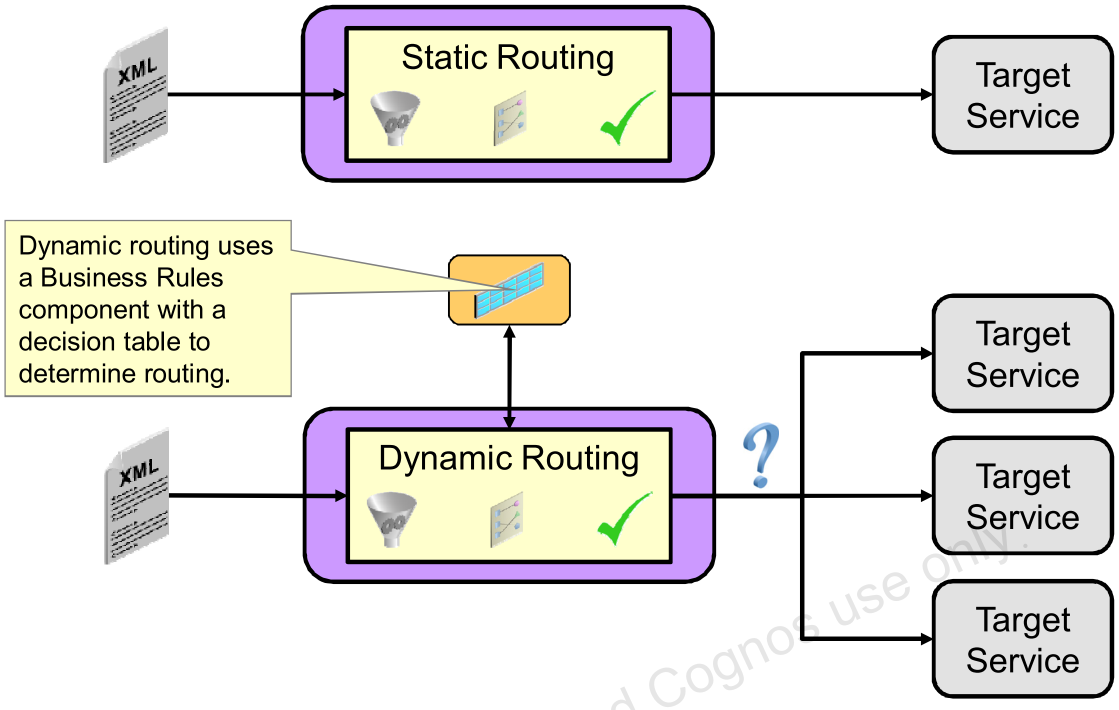
Mediator as a Callable Service



Oracle Mediator functionality can be exposed as a callable service without having to route it to any other service. For example, you can call an Oracle Mediator to perform a transformation, a validation, or an assignment, and then echo the Oracle Mediator back to your application without routing it anywhere else.

Note: An echo can be either a synchronous or an asynchronous interface. The echo option is available only for inbound service operations and is not available for event subscriptions.

Dynamic and Static Routing Rules



The Mediator component enables you to create both *static* and *dynamic* routing rules.

Static routing rules can be created for event subscriptions and incoming operations for both synchronous and asynchronous service interactions. Static routing rules support the following types of targets:

- **Service type:** To route a message to a target service, either another component within the same composite application or an external service reference
- **Event type:** To publish an event to the Event Delivery Network
- **Echo service:** To use an internal echo mechanism that enables the request message to be echoed as a response, which can be transformed before being returned to the caller

A ***dynamic routing rule*** configures a Business Rules component that uses a decision table to determine the target endpoint to which a message is routed through a dynamic external service interface. Dynamic routing rules can be created only for asynchronous interactions.

Additional Features of Mediators

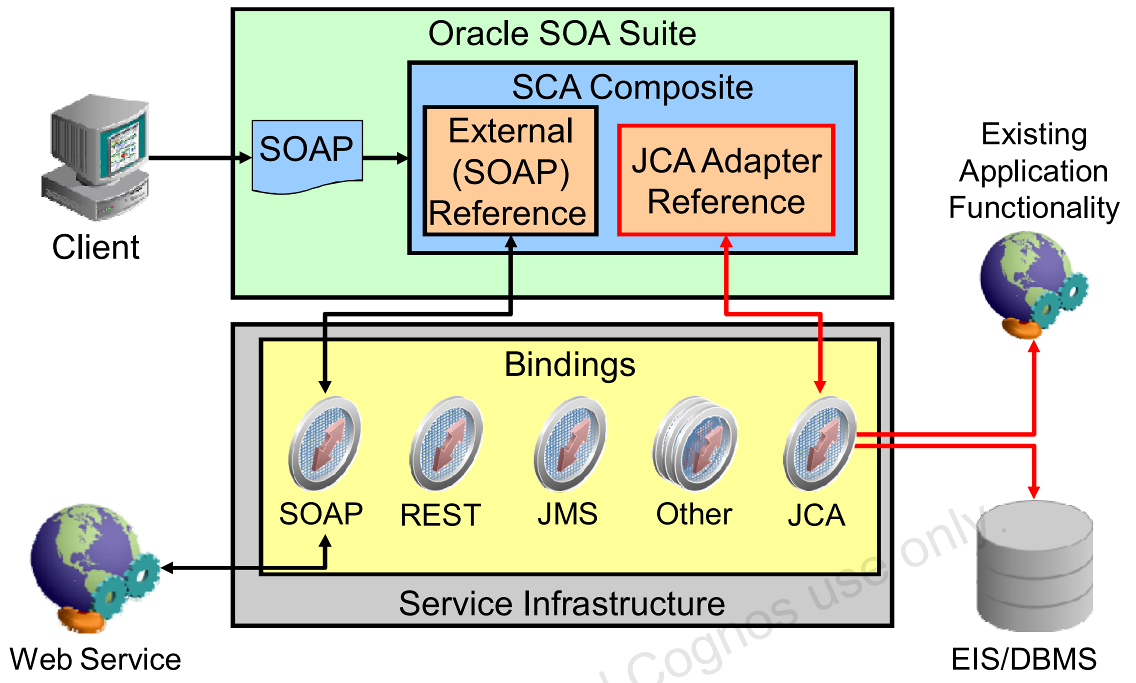
Mediators provide the following additional features, which are covered in later lessons:

- Error routing and management
  - Mediator components support fault policy–based error handling. A fault policy consists of conditions and actions. Conditions specify the action to be carried out for a particular error condition.
- Event Handling
  - Mediator components provide support for subscribing to or raising business events that are delivered through the Oracle SOA Suite 12c Event Delivery Network (EDN).

Agenda

- How Services Use WSDL Files to Communicate
- Mediator Components
- File Adapters
- Composite Application Files

Adapters: Introduction



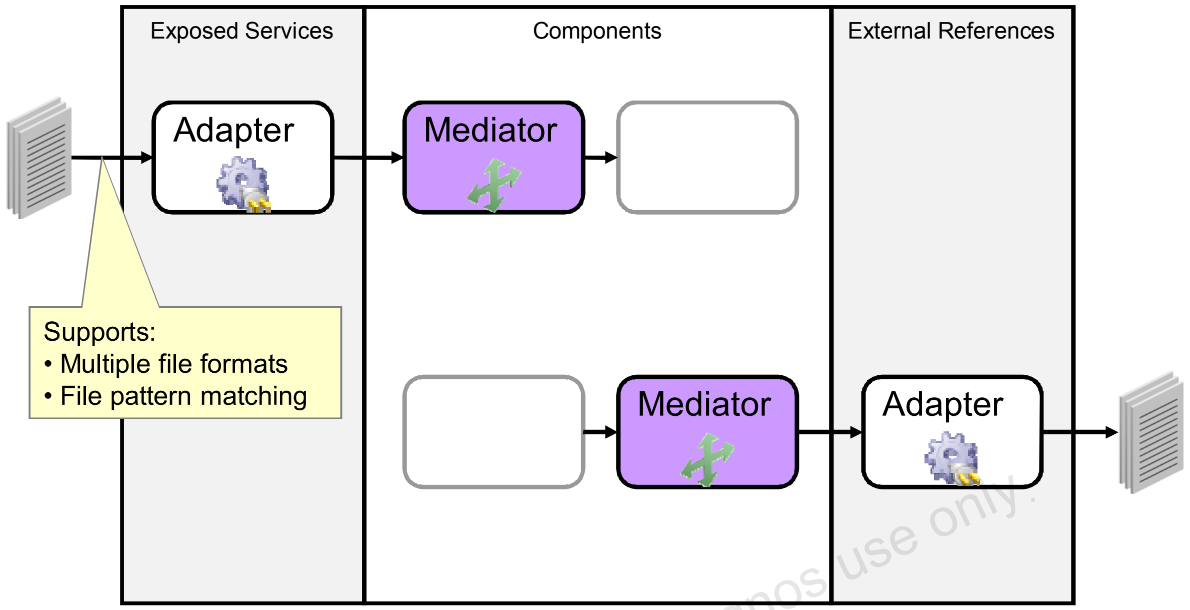
JCA stands for *JEE Connector Architecture* and is part of the JEE specification. JCA describes the concept of *resource adapters*, or simply *adapters*. Adapters are deployed in a Java EE container. An adapter exposes an API through which a service component can access an external application as if it were a web service. Adapters are described by a WSDL document, which is the only view the service component has of the external system. The adapter itself is transparent to the service component. This enables interoperability with heterogeneous applications, which are provided by different vendors, based on different technologies and platforms. By creating a service interface where none existed, an adapter provides another type of service in your “service portfolio.”

A number of technology adapters are available for Oracle SOA Suite 12*c* Mediator and BPEL components, including:

- **REST binding:** The Representational State Transfer (REST) binding is available for SOA composites and Service Bus services. It allows the configuration of REST interactions as exposed interfaces and allows the invocation of externally available REST services.

- **JMS/AQ adapter:** Either inbound or outbound, a JMS/AQ adapter transmits an XML message to or from an Oracle SOA Suite component through a message-oriented service such as an Oracle Advanced Queuing queue and other JMS providers.
- **Database adapter:** An inbound Database adapter service sends an XML message to an Oracle SOA Suite component when a SQL insert, update, or delete operation is performed against a database. An outbound Database adapter transforms the contents of an XML message into a SQL insert, update, or delete operation on the target database.
- **File/FTP adapters:** An inbound File or FTP adapter service reads data from a local or remote file system, transforms the file data into an XML message, and sends it to an Oracle SOA Suite component when a new text file appears in a local file system. An outbound File adapter service transforms the contents of an XML message to a text file, and writes it to a local or remote file system.
- **Oracle Applications (OA) adapter:** An inbound OA adapter sends XML messages to Oracle SOA Suite on receiving messages from an Oracle E-Business Suite interface. An outbound OA adapter sends data from Oracle SOA Suite to Oracle Applications by using interface tables, APIs, and concurrent programs.
- **Socket adapters:** These communicate either as a client or server by using TCP/IP-based protocols.

File Adapter



As the name implies, a File adapter reads and writes files to the operating system. Pattern matching capabilities enable filter processing for specific sets of files. XML schema documents are used to support the parsing and formatting of different file formats such as delimited, positional, XML, binary, and COBOL copy books. Error recovery and checkpoint capabilities facilitate management of exception conditions. Scalability is obtained through the multi-threaded execution environment, which supplies connection management, and batching and debatching capabilities.

When configured to *read* files, a File adapter is configured as an *exposed service*, forming the composite application's entry point. Within the composite application, it is wired to either a Mediator or a BPEL component. When configured to *write* files, the File adapter is configured as an external reference, and serves as the target of a BPEL service invocation (more on this later) or as the target of a Mediator routing rule.

Reading or writing a file thus is a one-way operation—that is, in the case of a write operation, the File adapter service writes the payload to a specified file name in a specified directory. It does not send a response to the composite application. In the case of a read operation, the File adapter polls a specific directory for new files. When found, the File adapter passes the file payload to the composite application, but does not wait for a response from the composite.

Quiz

Which Mediator component feature enables target services to receive data in their desired format?

a. Filter expression
b. XSL transformation
c. Validation
d. Routing rule

ORACLE

Answer: b

Although a filter can be used to prevent undesirable data from reaching a target service and validations can prevent ill-formed XML data from being passed to a target service, it is XSL transformation that converts data into a format that is desired and required by a target service.

Quiz

The File adapter is capable of representing many file formats as XML data.

a. True

b. False

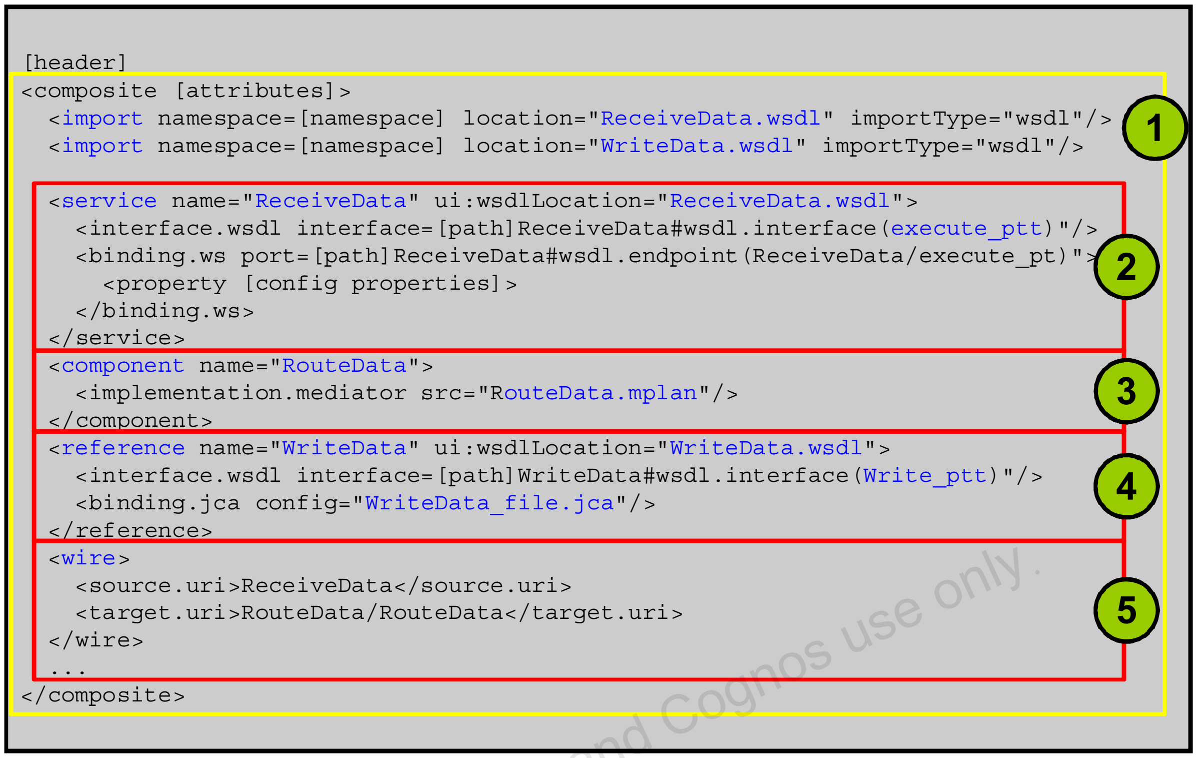
Answer: a

True. The File adapter supports multiple file formats. It can convert native format data to XML, and XML to native formats.

Agenda

- How Services Use WSDL Files to Communicate
- Mediator Components
- File Adapters
- Composite Application Files

Examining the `composite.xml` Source

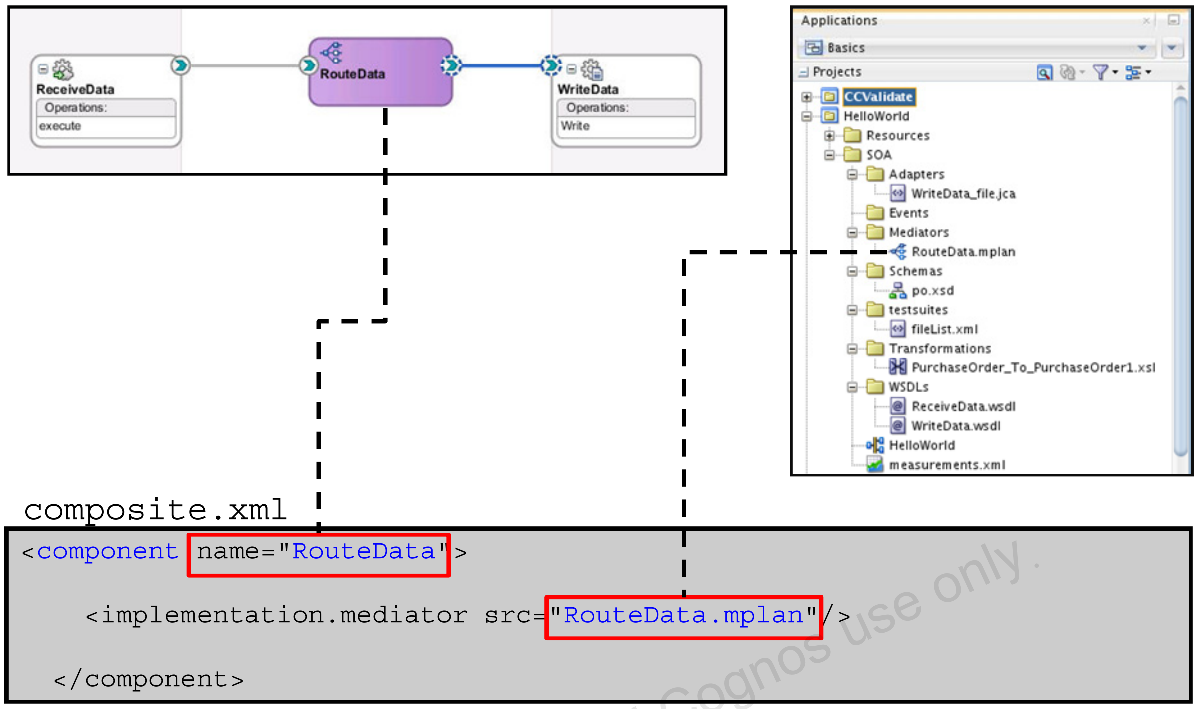


```
[header]
<composite [attributes]>
  <import namespace=[namespace] location="ReceiveData.wsdl" importType="wsdl"/>
  <import namespace=[namespace] location="WriteData.wsdl" importType="wsdl"/>

  <service name="ReceiveData" ui:wsdlLocation="ReceiveData.wsdl">
    <interface.wsdl interface=[path]ReceiveData#wsdl.interface(execute_ptt)"/>
    <binding.ws port=[path]ReceiveData#wsdl.endpoint(ReceiveData/execute_pt)">
      <property [config properties]>
    </binding.ws>
  </service>
  <component name="RouteData">
    <implementation.mediator src="RouteData.mplan"/>
  </component>
  <reference name="WriteData" ui:wsdlLocation="WriteData.wsdl">
    <interface.wsdl interface=[path]WriteData#wsdl.interface(Write_ptt)"/>
    <binding.jca config="WriteData_file.jca"/>
  </reference>
  <wire>
    <source.uri>ReceiveData</source.uri>
    <target.uri>RouteData/RouteData</target.uri>
  </wire>
  ...
</composite>
```

A `composite.xml` file is automatically created when you create a SOA project. This file describes the entire composite assembly of services, service components, references, and wires. The example in the slide is formatted to make it easier to read, and shows the following key elements:

1. The <`composite`> element encapsulates the entire composite assembly information.
2. The <`service`> element represents the service that is exposed by the composite.
   - The <`interface.wsdl`> element names the portType of the service. *(Note that "`interface.wsdl`" does not refer to a file name in this case.)*
   - The <`binding.ws`> element includes port information to expose the service to external clients.
3. The <`component`> element represents a Mediator, and includes a link to the source implementation file. (Additional <`component`> elements would be included for each additional component in the application.)
4. The <`reference`> elements describe connections to the external services that are referenced by the components in the composite application.
5. The <`wire`> element defines how data will flow among components. (Additional wires would be present to connect additional components and external references.)

Creating a Mediator

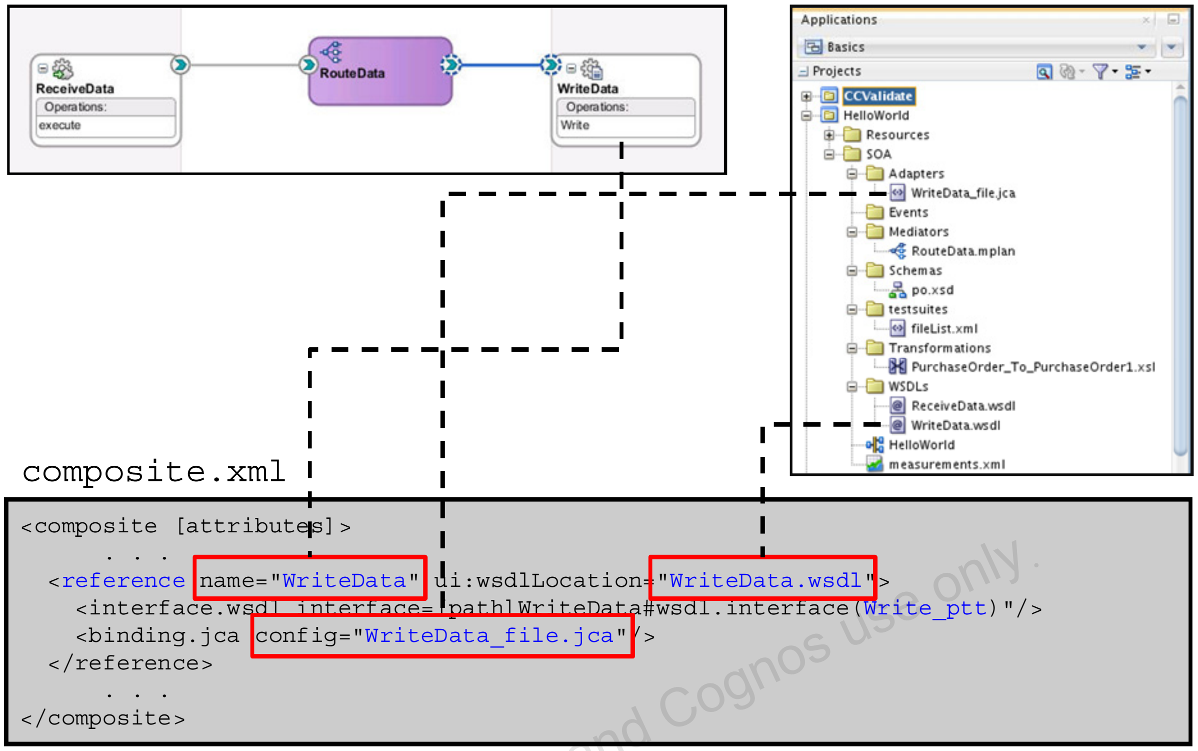


```
<component name="RouteData">

    <implementation.mediator src="RouteData.mplan"/>

  </component>
```

When a Mediator is added to a composite application, several project files are created or modified:

- A `.componentType` file describes the services and references for each component. This file tracks the wiring that is created between components, and ensures that the wires work as designed.
- An `.mplan` file contains the Mediator metadata. This includes implementation details such as routing rules, links to transformations, filters, validations, and references to target services.
- A `.wsdl` (Web Services Description Language) file specifies how other services call an Oracle Mediator. A `.wsdl` file defines the input and output messages and operations of an Oracle Mediator. The `.wsdl` file is not created in every circumstance, but is created if you specify an interface definition for the component by selecting the Asynchronous Interface, Synchronous Interface, the One-Way Interface, or the “Interface Definition from WSDL” templates.
- A `<component>` element is added to the `composite.xml` file.
- An `<import>` element is added to the `composite.xml` file for the component WSDL interface.

Creating an Adapter



```
<composite [attributes]>
    . . .
  <reference name="WriteData" ui:wsdlLocation="WriteData.wsdl">
    <interface.wsdl interface=[path]WriteData#wsdl.interface(Write_ptt)"/>
    <binding.jca config="WriteData_file.jca"/>
  </reference>
    . . .
</composite>
```

ORACLE

When an adapter is added to a composite application, several project files are created or modified:

- A `.componentType` file describes the services and references for each component. This file tracks the wiring that is created between components, and ensures that the wires work as designed.
- A `.jca` file contains the adapter implementation details. In the case of a File adapter that is configured to write data, the details would contain a directory name and a file naming pattern, as well as other information.
- A `.wsdl` file is created that includes the abstract WSDL information about the service that the adapter provides.
- An <`import`> statement is added to the `composite.xml` file, which makes the `.wsdl` file contents available.
- A <`reference`> element is added to the `composite.xml` file that aggregates the references to the adapter `.jca` and `.wsdl` files. This also includes the binding information that substitutes for the concrete information in a WSDL.

Summary

In this lesson, you should have learned how to:

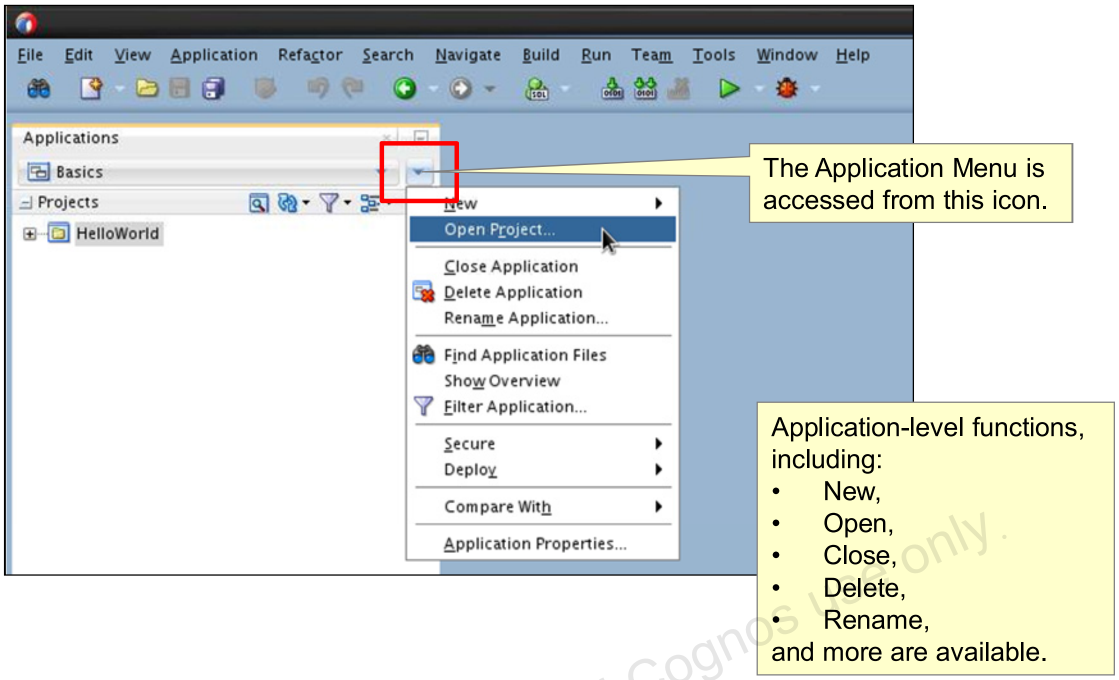
- Describe how services use WSDL files to communicate
- Use Mediator components and File adapters to perform basic receiving, routing, and writing of messages
- Name and describe the contents of some of the files that are created as part of a composite application

Practice 2 Overview

This practice covers the following topics:

- Creating a Composite Application
- Adding a Mediator Component to a Composite Application
- Adding a File Adapter to a Composite Application
- Examining Composite Application Files

Application Navigator



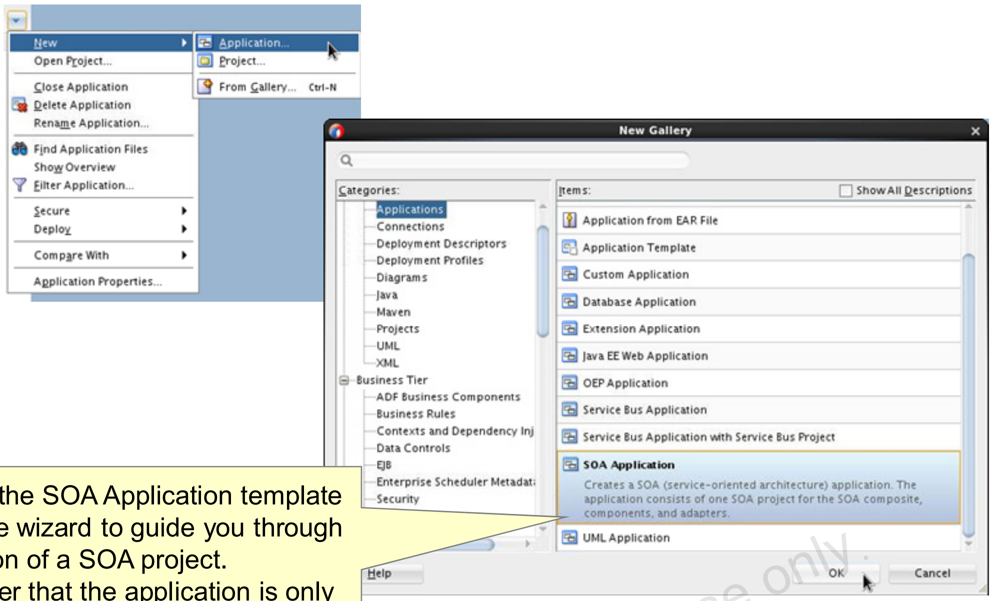
The Application Menu is accessed from this icon.

Application-level functions, including:
- New,
- Open,
- Close,
- Delete,
- Rename,

and more are available.

Access to application-level tasks is provided by the Application Navigator menu.

Creating a SOA Application



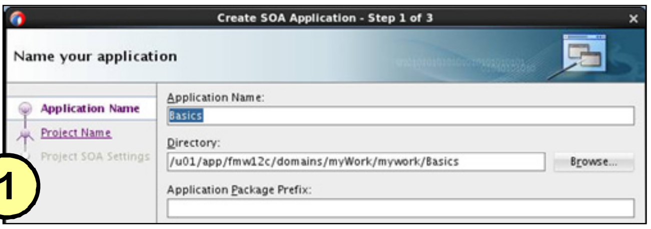
Creates a SOA (service-oriented architecture) application. The application consists of one SOA project for the SOA composite, components, and adapters.

Selecting the SOA Application template causes the wizard to guide you through the creation of a SOA project. (Remember that the application is only a container. The "real work" is defined in the projects that you add to that container.)

When you select New Project from the Application Navigator menu, the Create Application wizard is invoked. The wizard displays a series of windows in which you make choices and input values to create the new application. In the example in the slide, the SOA Application template is selected. This selection drives the content and choices of some of the subsequent windows in the wizard.

Creating a SOA Application

1



Create SOA Application - Step 1 of 3

Name your application

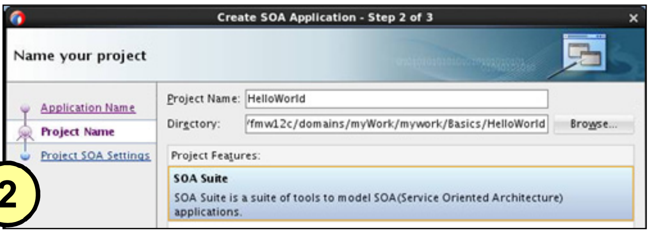
Application Name: Basics

Directory: /u01/app/fmw12c/domains/myWork/mywork/Basics

Application Package Prefix:

The first steps in building a new application are to assign it a name and to specify the directory in which to save the source files.

2



Create SOA Application - Step 2 of 3

Name your project

Project Name: HelloWorld

Directory: /fmw12c/domains/myWork/mywork/Basics/HelloWorld

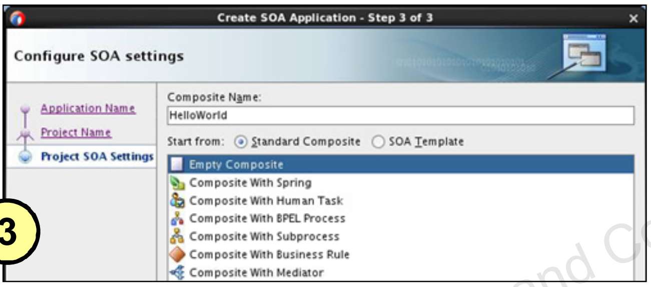
Project Features:

SOA Suite

SOA Suite is a suite of tools to model SOA(Service Oriented Architecture) applications.

Add a project to the application, assigning it a name and directory as well.

3



Create SOA Application - Step 3 of 3

Configure SOA settings

Composite Name: HelloWorld

Start from: [✓ Standard Composite] SOA Template

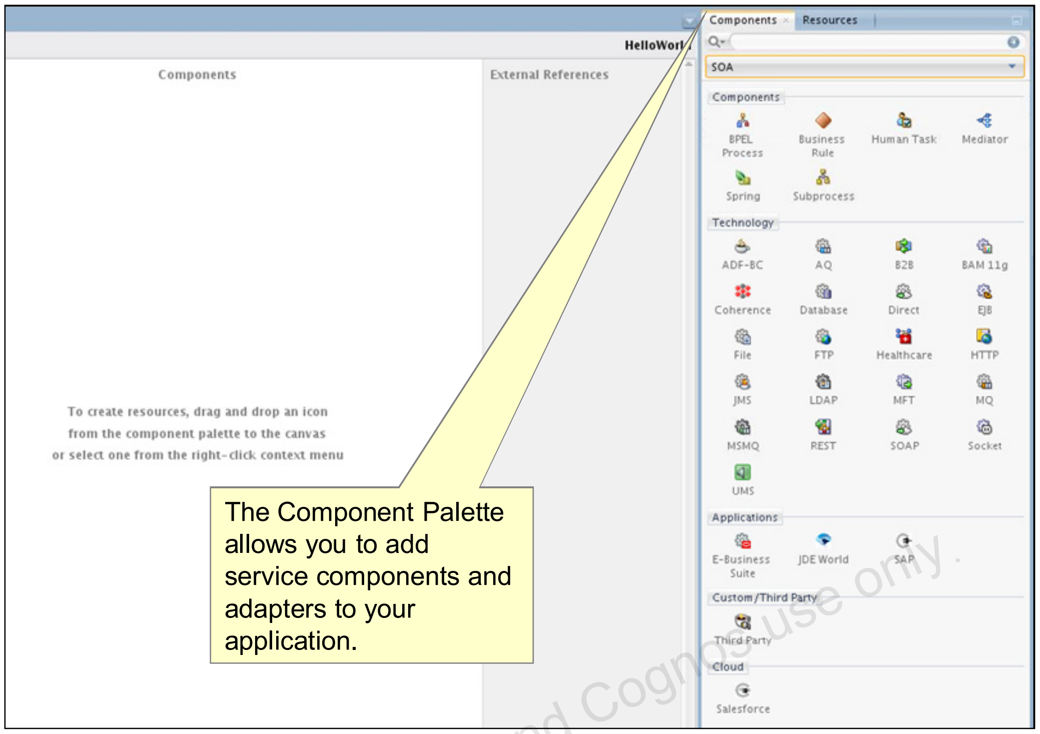
- Empty Composite
- Composite With Spring
- Composite With Human Task
- Composite With BPEL Process
- Composite With Subprocess
- Composite With Business Rule
- Composite With Mediator

Specify the starting point for the project.

In the Applications window on the upper left, select New Application from the Applications drop-down list.

1. On the “Name your application” page, you can optionally change the name and location for your application.
2. On the “Name your project” page, you can optionally change the name and location for your SOA project. By default, Oracle JDeveloper adds the SOA project technology, the `composite.xml` file that describes the SOA composite application, and the necessary libraries to your model project.
3. On the Configure SOA Settings page, click Empty Composite for this example, and click Finish.

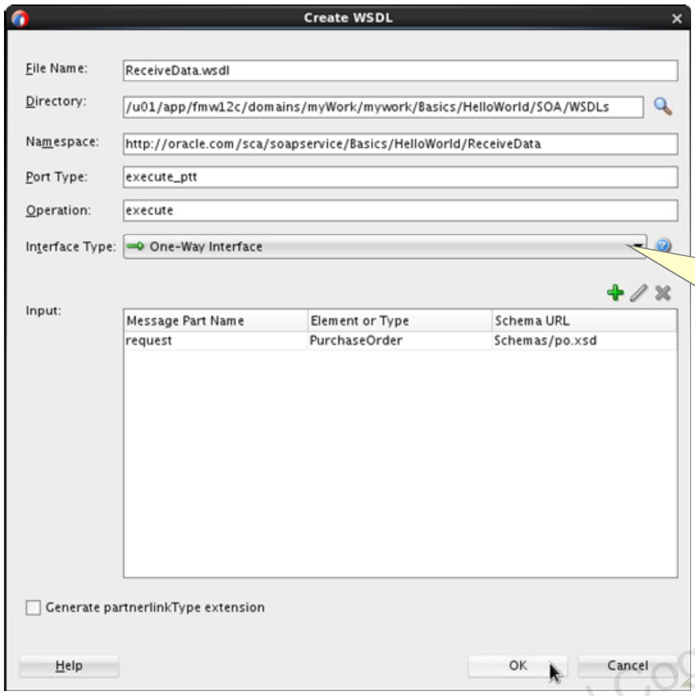
Component Palette



The Component Palette allows you to add service components and adapters to your application.

When the SOA Application is created, the Composite Editor is opened. It is on that blank canvas that you build the model of your composite application. The service components and adapters for this model are accessed from the Component Palette, which is displayed on the right side of JDeveloper.

Creating a WSDL



Create WSDL

File Name:	ReceiveData.wsdl
Directory:	/u01/app/fmw12c/domains/myWork/mywork/Basics/HelloWorld/SOA/WSDLs
Namespace:	http://oracle.com/sca/soapservice/Basics/HelloWorld/ReceiveData
Port Type:	execute_ptt
Operation:	execute
Interface Type:	One-Way Interface

Input:

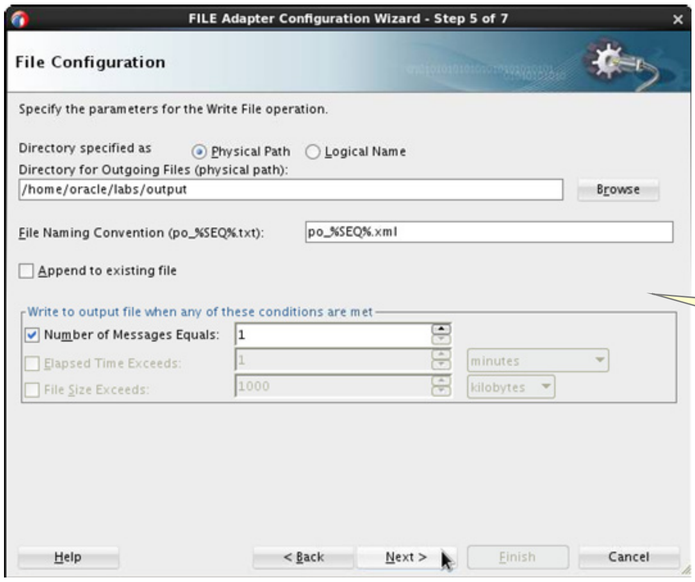
Message Part Name	Element or Type	Schema URL
request	PurchaseOrder	Schemas/po.xsd

Generate partnerlinkType extension

The Create WSDL wizard builds the necessary WSDL files automatically. You specify XSD files to describe the messages and modify any other values that you desire.

Recall from earlier in the lesson that WSDL documents are used to describe the means for accessing a service. JDeveloper offers a Create WSDL wizard that prompts you for the required information to automatically create WSDL documents.

Creating a File Adapter


FILE Adapter Configuration Wizard - Step 5 of 7

File Configuration

Specify the parameters for the Write File operation.

Directory specified as [✓ Physical Path] Logical Name

Directory for Outgoing Files (physical path): /home/oracle/labs/output

File Naming Convention (po_%SEQ%.txt): po_%SEQ%.xml

☐ Append to existing file

Write to output file when any of these conditions are met

☑ Number of Messages Equals: 1

☐ Elapsed Time Exceeds: 1 minutes

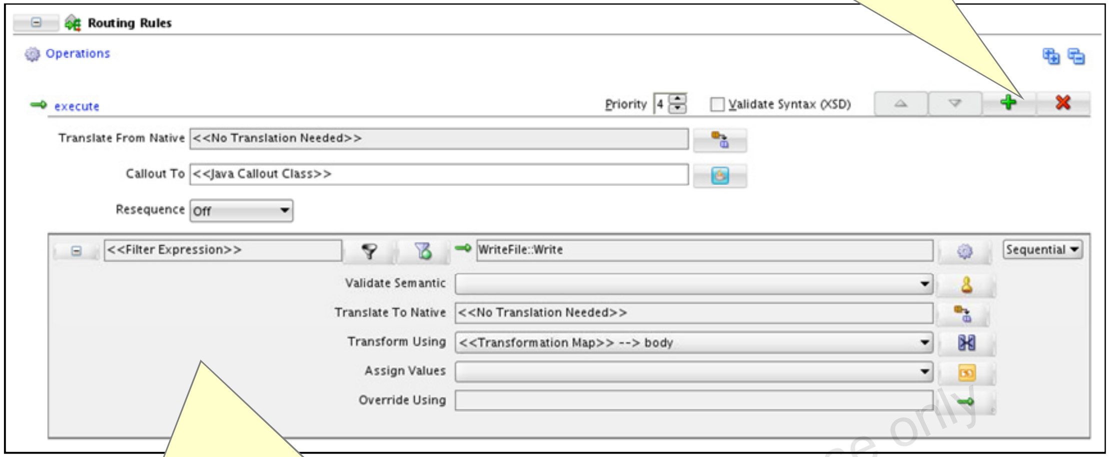
☐ File Size Exceeds: 1000 kilobytes

The File Adapter wizard guides you through the configuration of an adapter.

Adapter configuration is also wizard-driven. In the example in the slide, the File Adapter Configuration Wizard is displayed as a result of adding a File adapter to the SOA Application model. The wizard prompts for the inputs that are necessary to configure the File adapter.

Configuring Mediator Routing Rules

Add or delete routing rules as needed.



Priority 4 Validate Syntax (XSD)

Translate From Native: <<No Translation Needed>>

Callout To: <<Java Callout Class>>

Resequence: Off

<<Filter Expression>> WriteFile::Write Sequential

Validate Semantic:

Translate To Native: <<No Translation Needed>>

Transform Using: <<Transformation Map>> --> body

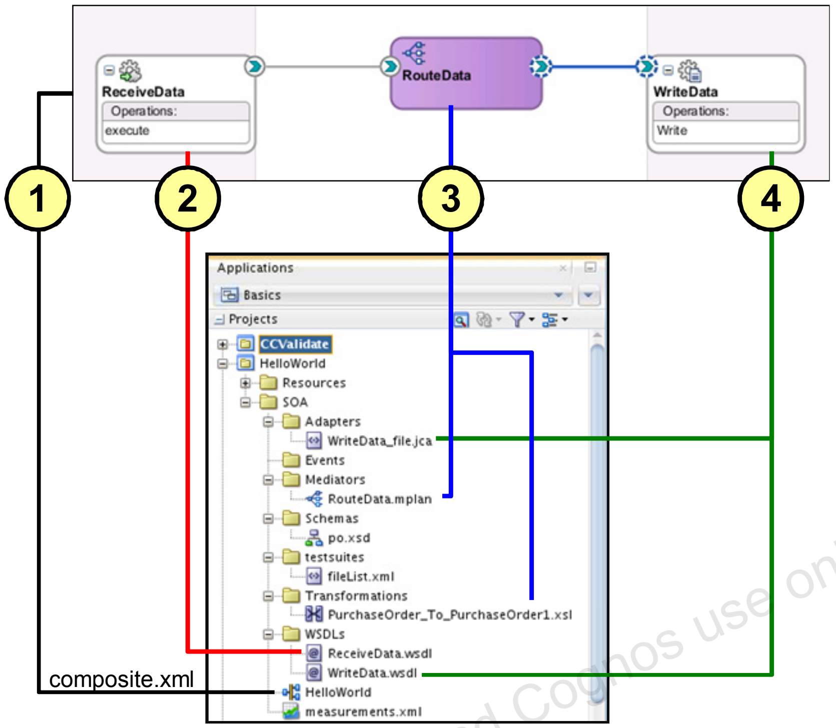
Assign Values:

Override Using:

Specify the validation, filtering, and transformation details for each Mediator routing rule.

Mediator routing rules are configured from an input window. From that window, filters, validations, and transformations can be defined and modified. In the practice, you create a minimum transformation. In later practices, you create more complex transformations, and define filters and validations.

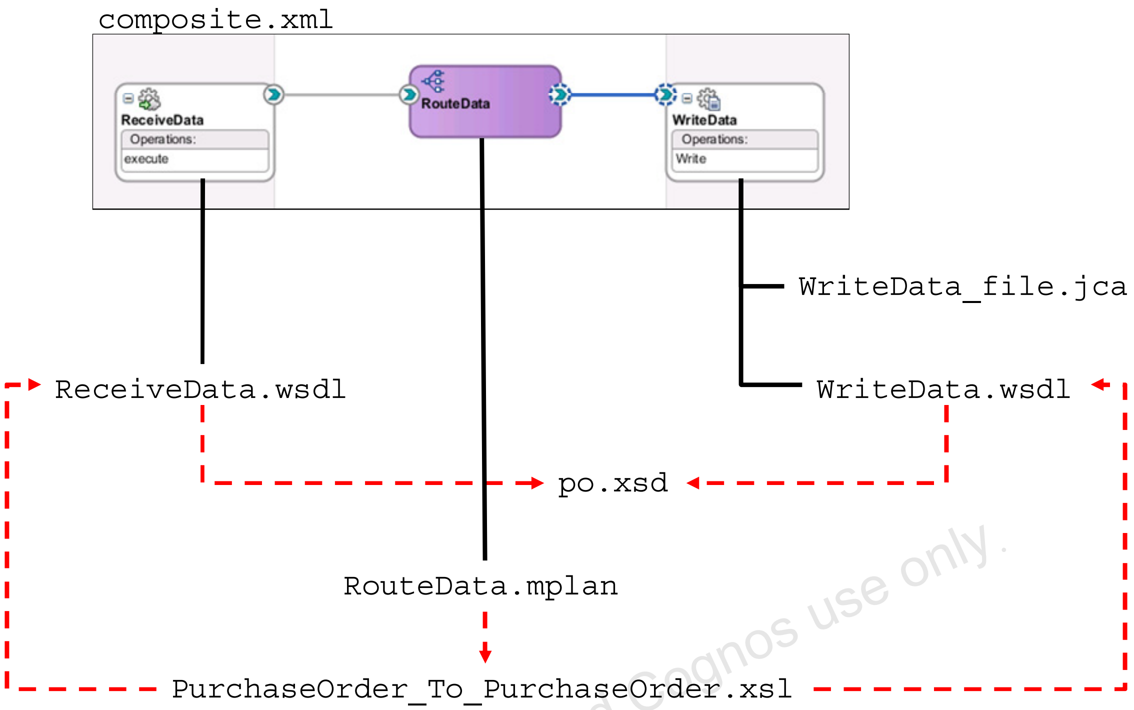
Project Files



Project files are organized in the Application Navigator under the *<ProjectName>/*SOA folder structure.

1. The `composite.xml` file is displayed as *<ProjectName>*. This makes it easier to navigate among the JDeveloper editing tabs when multiple `composite.xml` files are open.
2. The WSDL file for the service binding is stored in the SOA/WSDL folder.
3. The Mediator configuration files (`.mplan`) are stored in the `/Mediators` subfolder. The XSLT transformations, including those found in mediators, are stored in the `/Transformations` subfolder.
4. The Adapter files are stored in the `/Adapters` and `/WSDLs` subdirectories.

Project Files



When you have completed building your first composite application, it looks like the model at the top of the slide. The bottom portion of the slide lists the files that are created with each component. The red dashed lines indicate the references between files and, from that, the underlying relationships between the components.