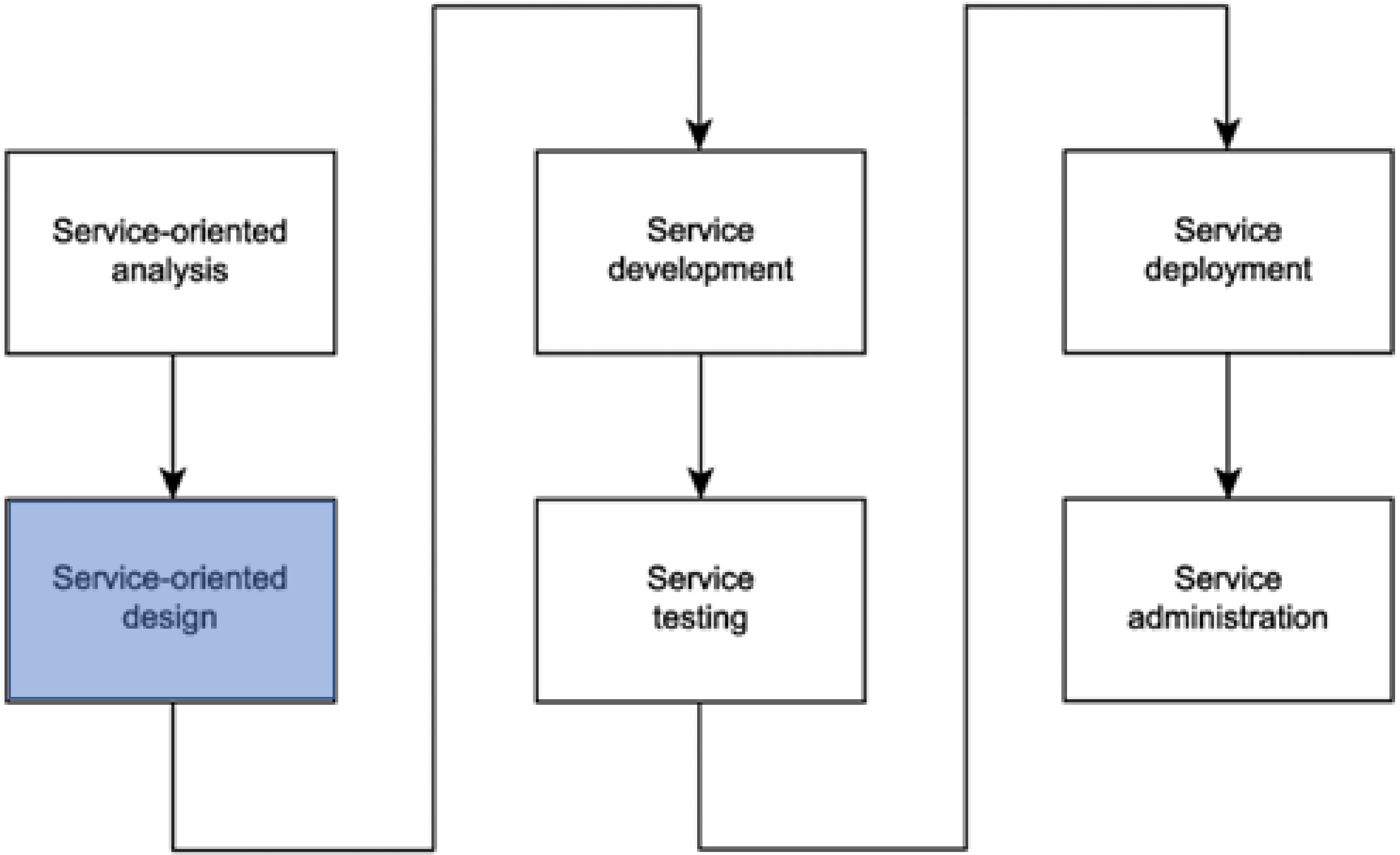
Reminder
Hot topic study – task 3 – individual/group

- Create a home page in your groups wiki area devoted to your selected topic – give an overview of your topic
- Deadline: 28th April (Thursday)

Module Six: Service Engineering

SOA delivery lifecycle



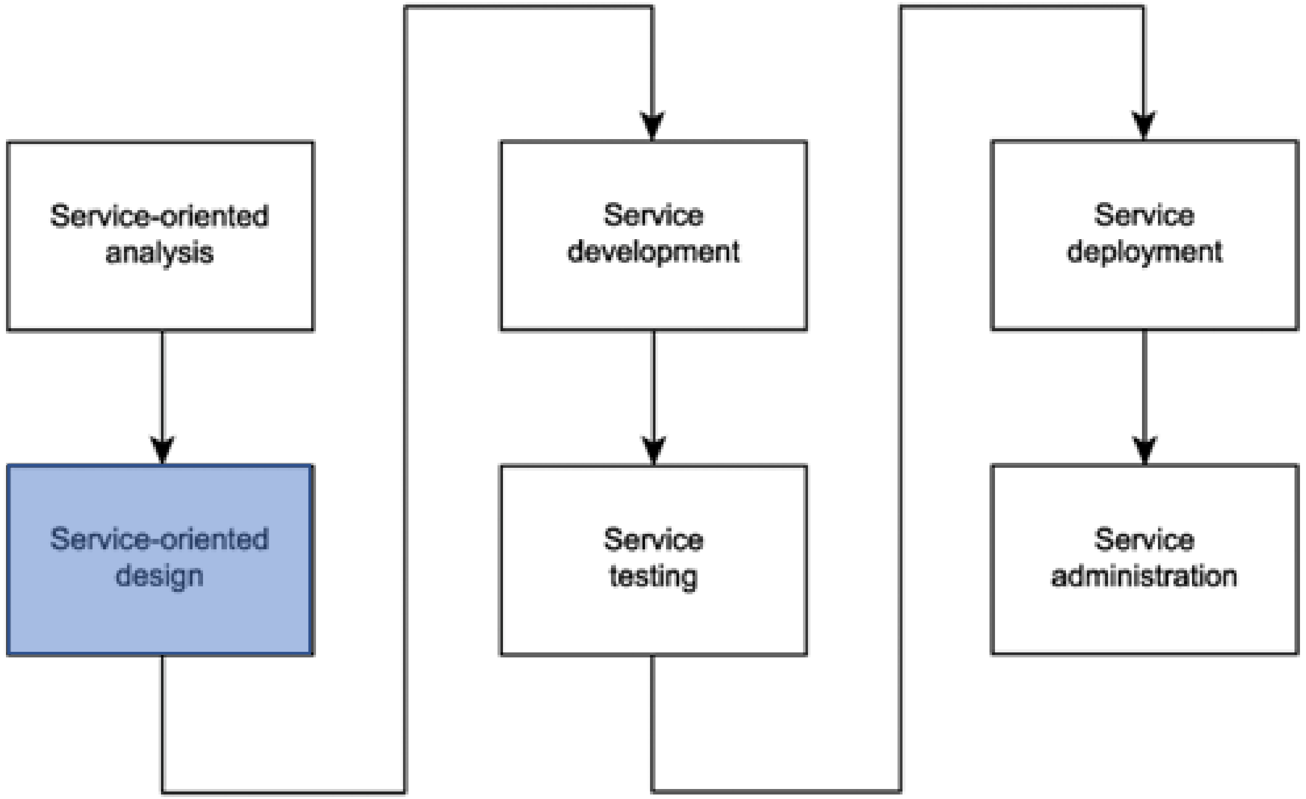
Service-oriented design

- Service-oriented design is the process by which concrete physical service designs are derived from logical service candidates and then assembled into abstract compositions that implement a business process.

Objectives of service-oriented design

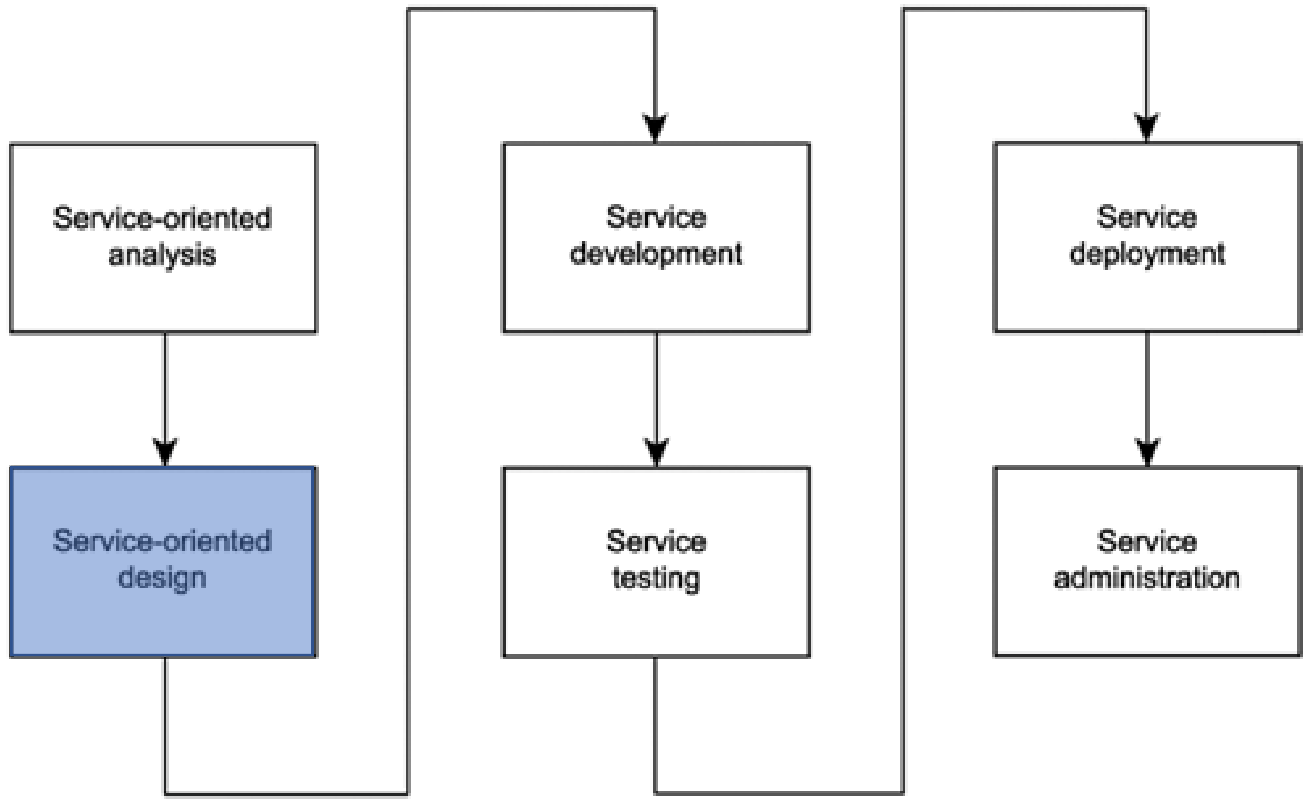
- Determine the core set of architectural extensions.
- Set the boundaries of the architecture.
- Identify required design standards.
- Define abstract service interface designs.
- Identify potential service compositions.
- Assess support for service-orientation principles.
- Explore support for characteristics of contemporary SOA.

Who should participate in this step?



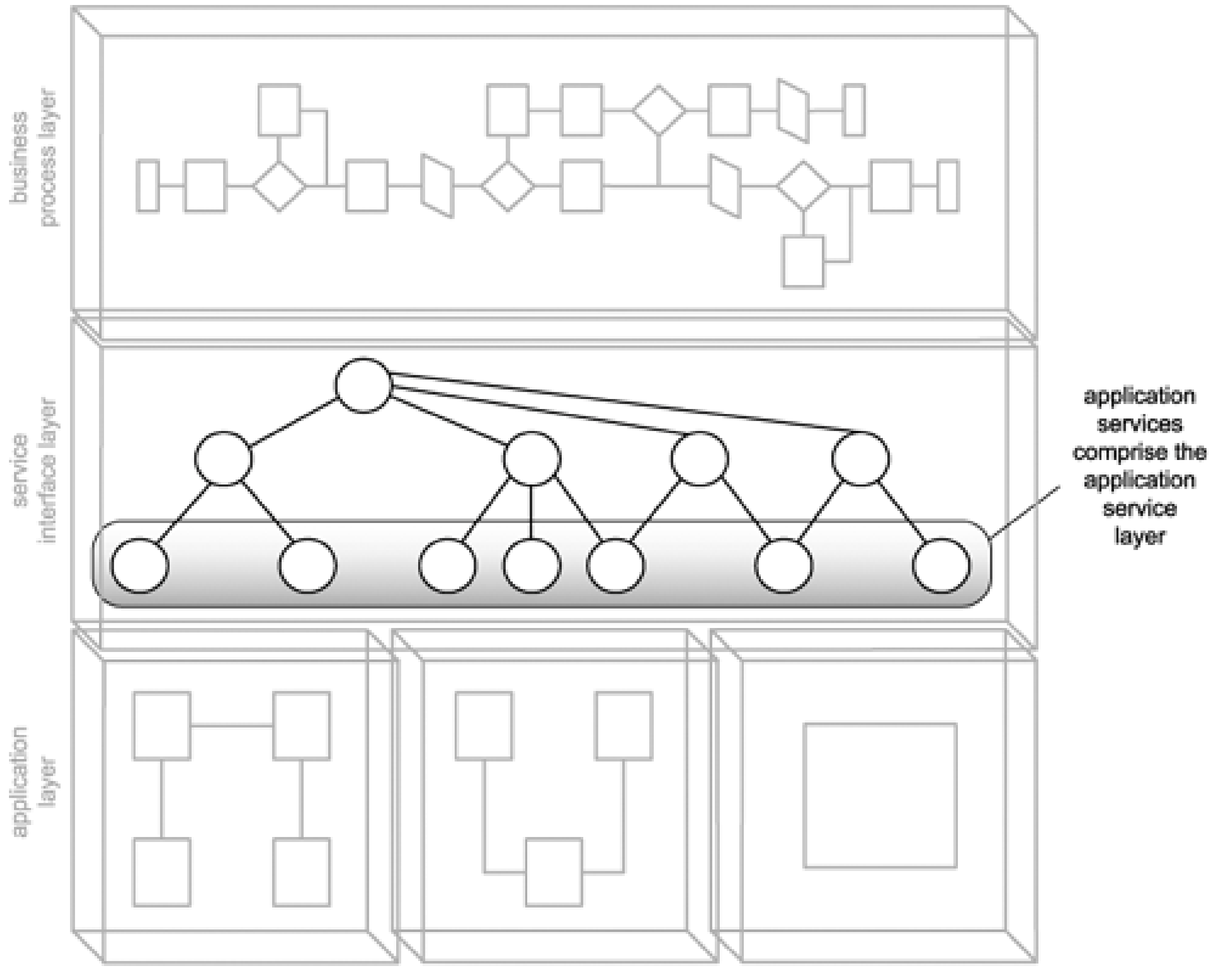
Who should participate in this step?

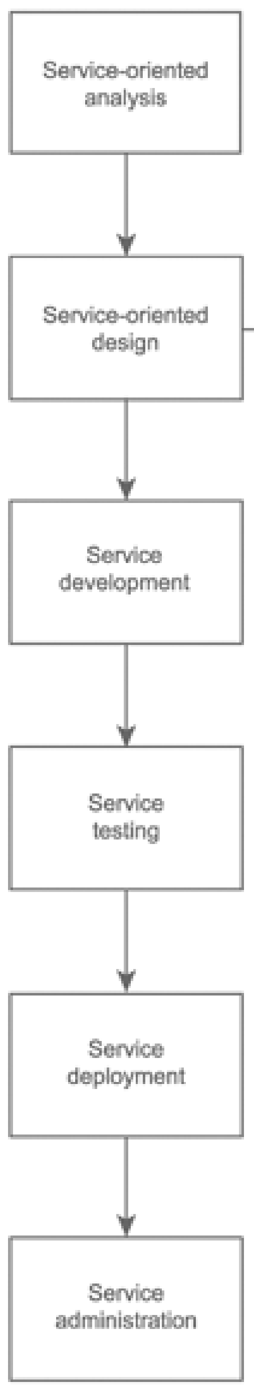
- Best defined by those who understand the enterprise technical environments the most
- Business analysis expertise not required

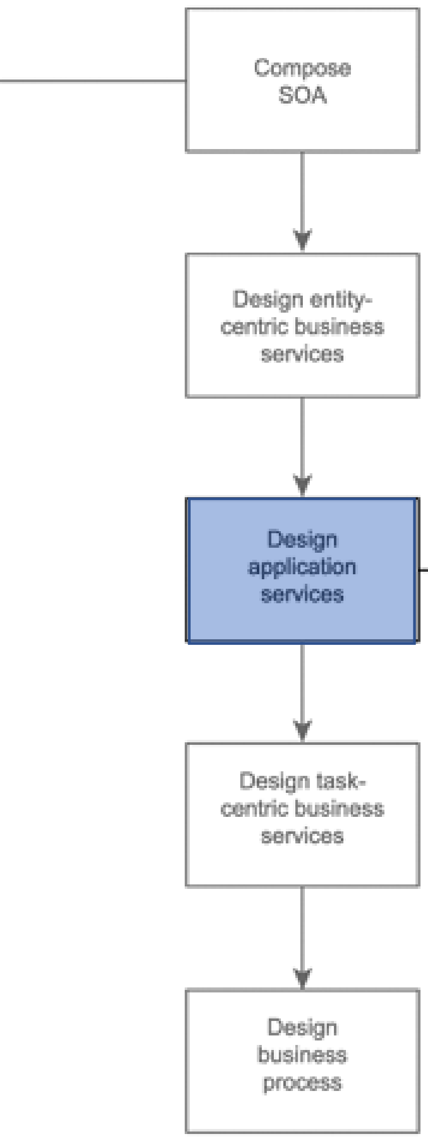


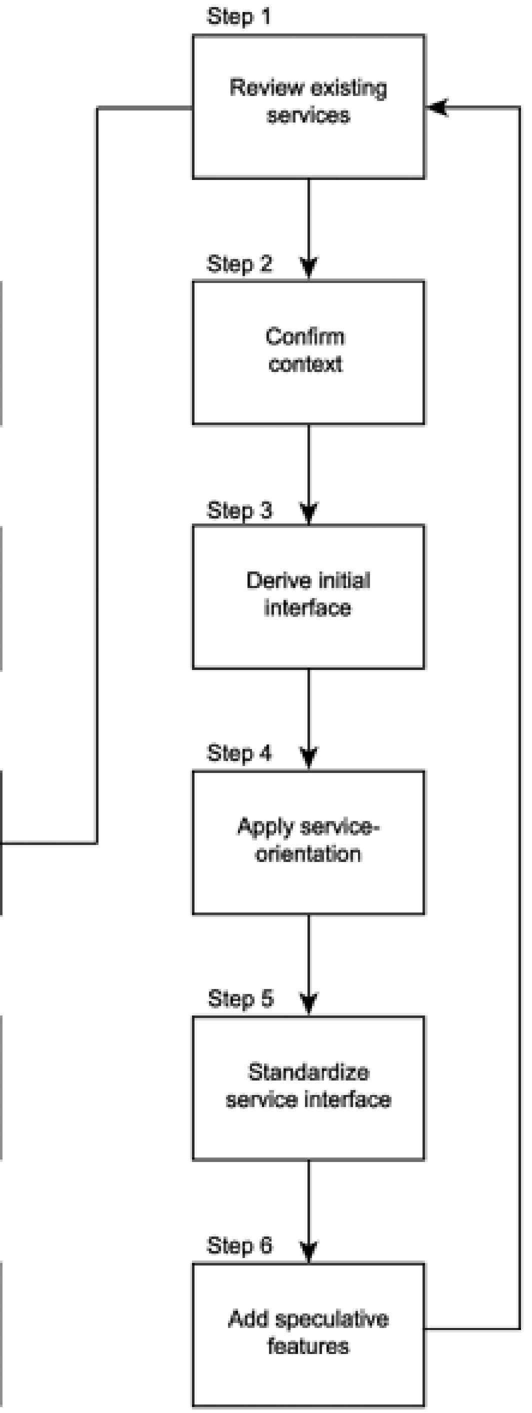
Service-oriented design

Design application services



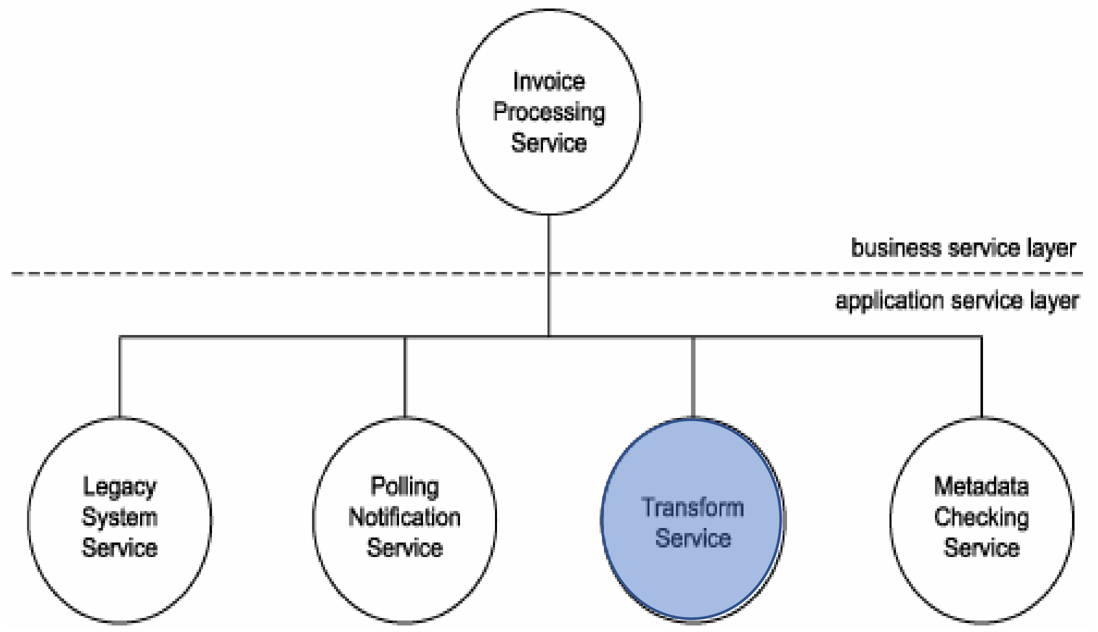


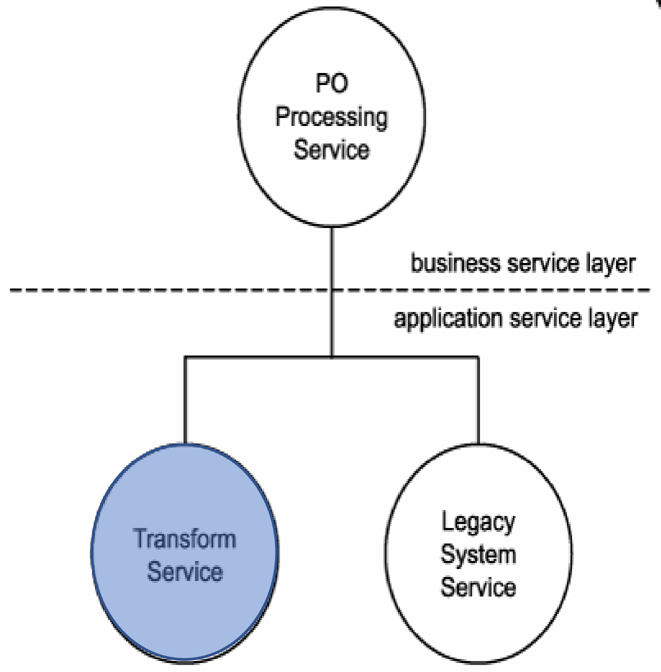

Compose SOA
Design entity-centric business services
Design application services
Design task-centric business services
Design business process

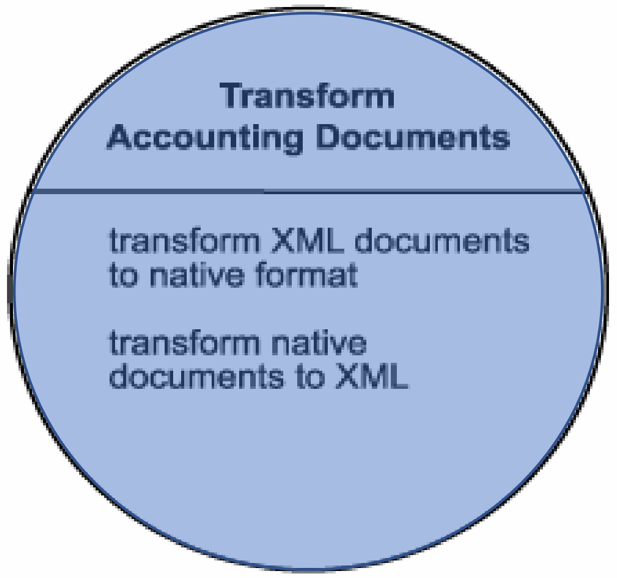


The design of the Transform application service candidate

Transform Service

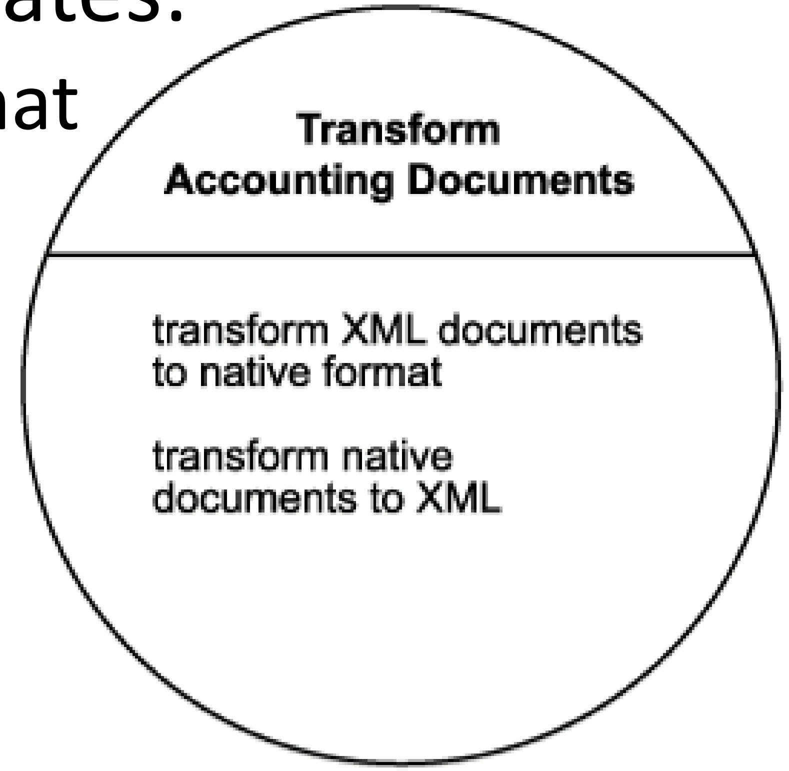

Invoice Processing Service
business service layer
application service layer
Legacy System Service
Polling Notification Service
Transform Service
Metadata Checking Service




Transform Accounting Documents
transform XML documents to native format
transform native documents to XML

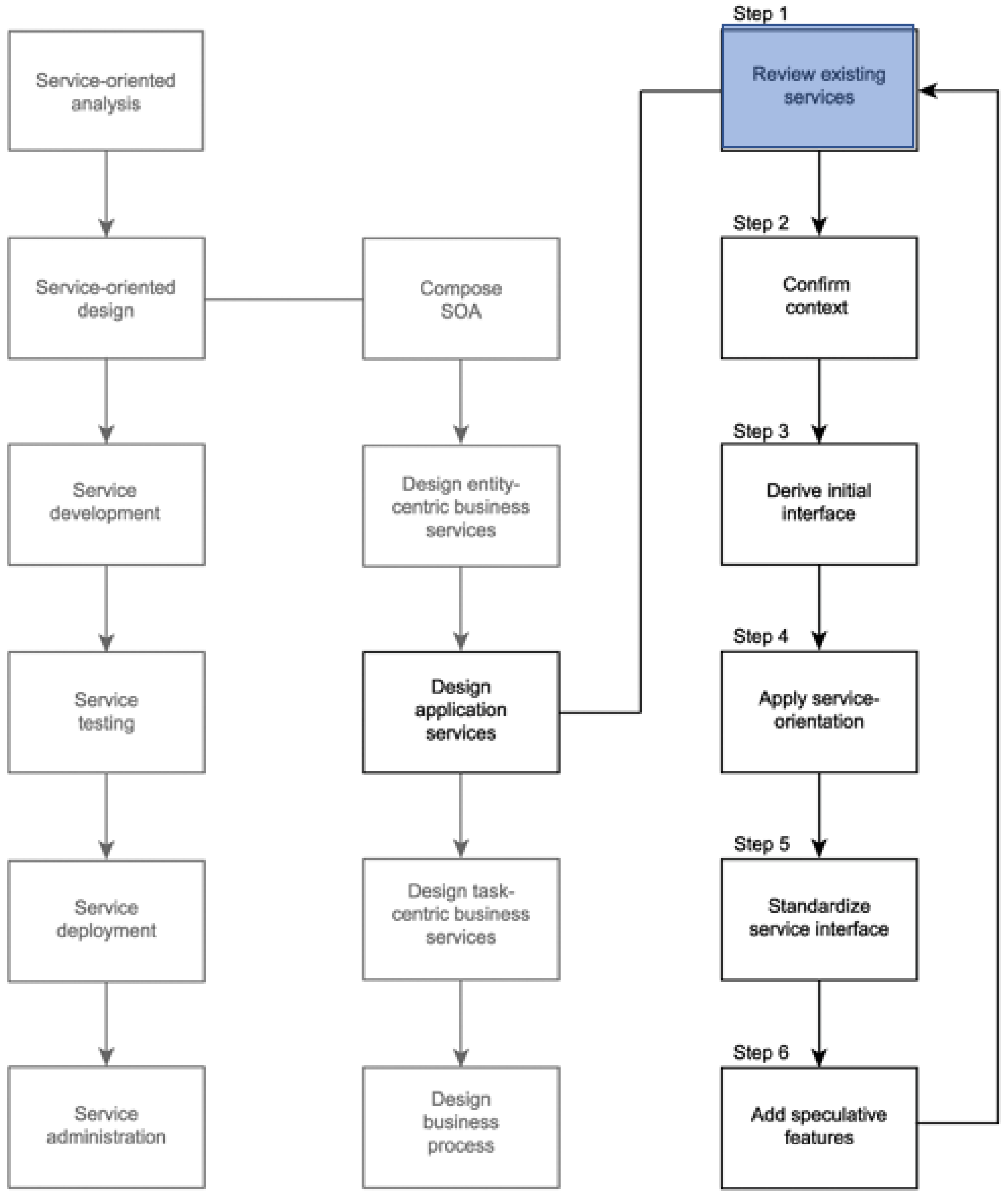
Service Design - Transform Service

- This candidate establishes a "document transformation context," which justifies the grouping of its two very similar operation candidates:
  - transform XML documents to native format
  - transform native documents to XML


Transform
Accounting Documents
transform XML documents to native format
transform native documents to XML

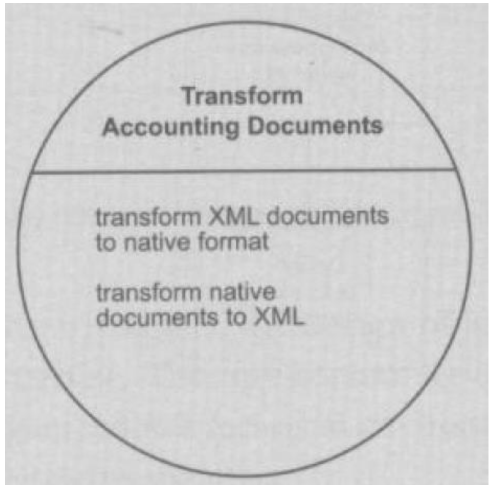
- We studied some design decisions that RailCo has made when developing this service

Service-oriented design



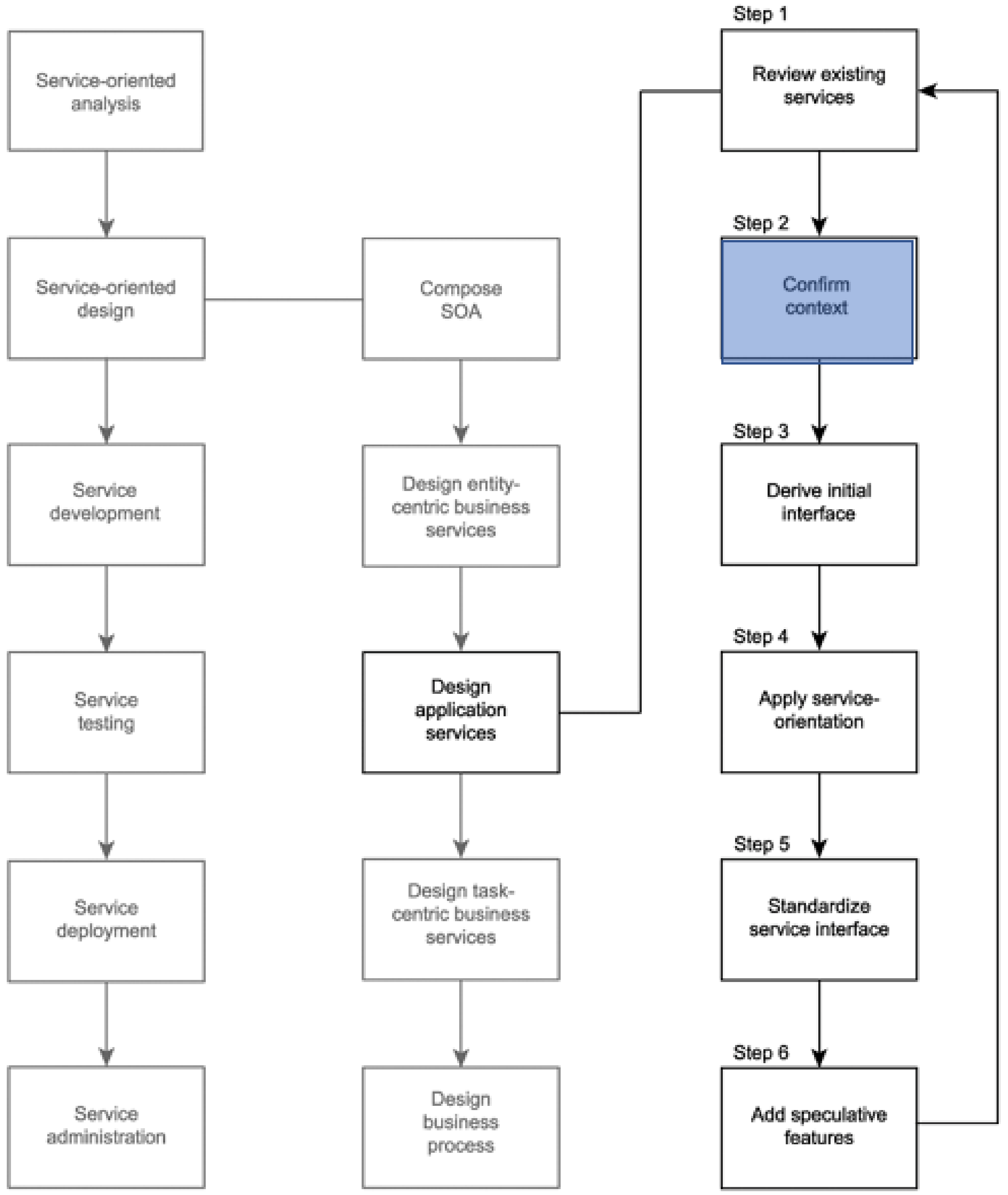
Step 1: Review existing services

- Check existing services for redundant features.
- Check if the required feature can be purchased or leased from vendors.


Transform
Accounting Documents
transform XML documents to native format
transform native documents to XML

- Checking names, service descriptions and metadata of existing services (i.e. TLS Subscription Service), RailCo concludes that no overlap exists

Service-oriented design



Step 2: Confirm context

- Reassess service context if operation candidate grouping from the analysis phase is appropriate, e.g. operations may better belong in other services.

Discussion

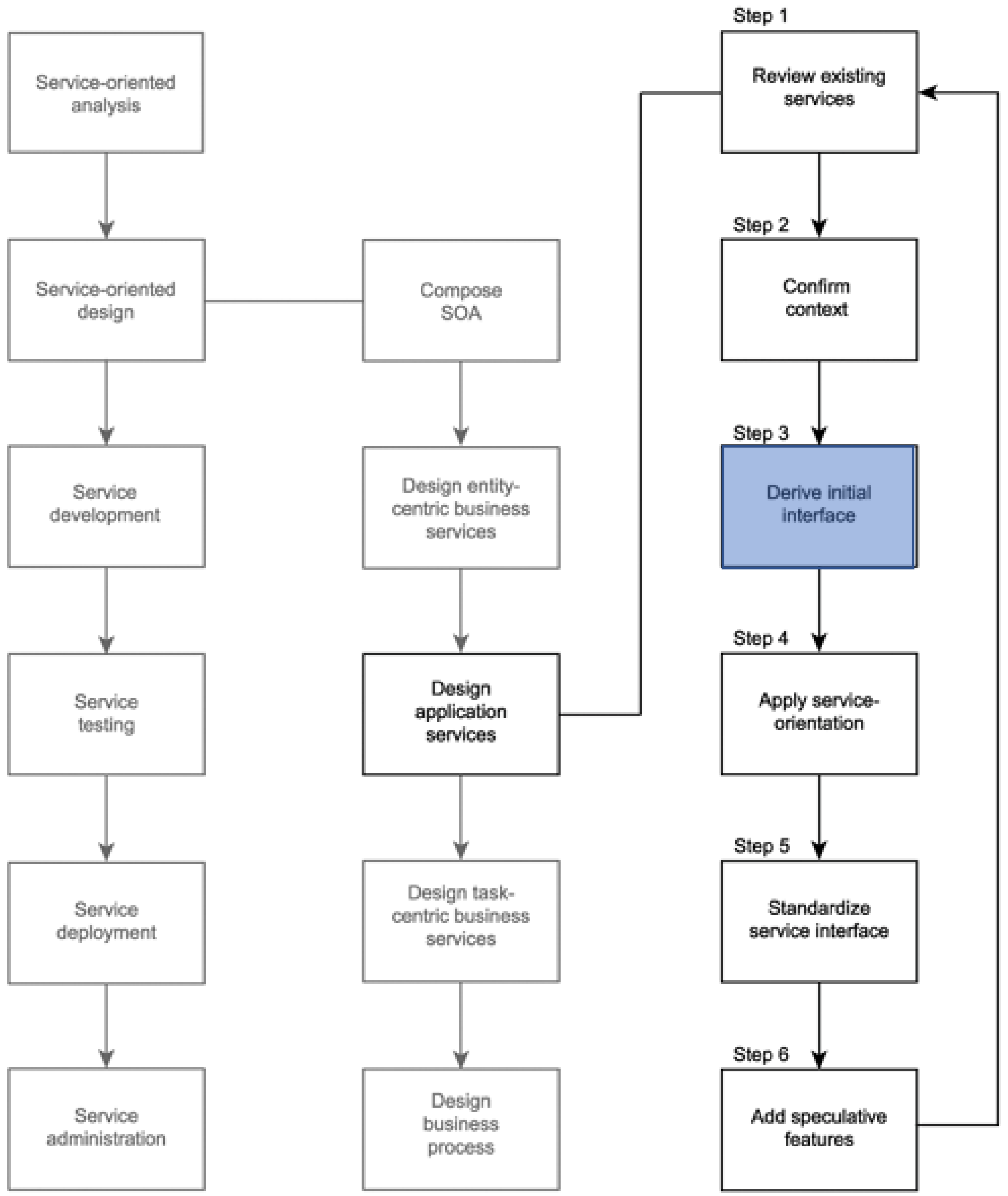
- Why there is a need for the design phase to reassess what was already decided during the analysis phase?

- Last time we have finished our discussion here

Step 2: Confirm context

- Reassess service context if operation candidate grouping from the analysis phase is appropriate, e.g. operations may better belong in other services.
- Review of Transform Accounting Documents service against TLS Subscription service confirms that grouping context is valid.

Service-oriented design



Step 3: Derive an initial service interface

- Confirm that each operation candidate is suitably reusable and its granularity is appropriate.

Step 3: Derive an initial service interface

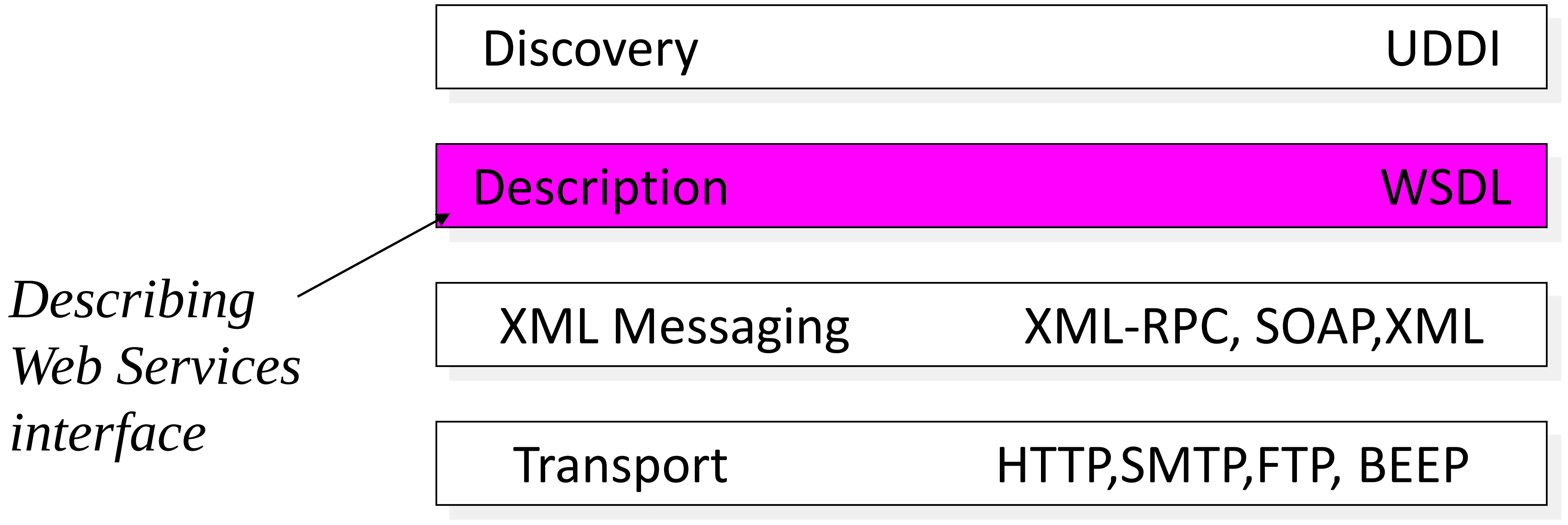
- Confirm that each operation candidate is suitably reusable and its granularity is appropriate.
- Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema.

Step 3: Derive an initial service interface

- Confirm that each operation candidate is suitably reusable and its granularity is appropriate.
    1. Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema.
    2. Establish a set of operation names and define WSDL <portType> with <operation>.
    3. Define WSDL <message> with <part>.

- A quick review of most relevant information about WSDL

Web Service Protocol Stack



- A client program reads a WSDL document to understand what a Web service can do; then it uses SOAP to actually invoke the functions listed in the WSDL document.

Service Description

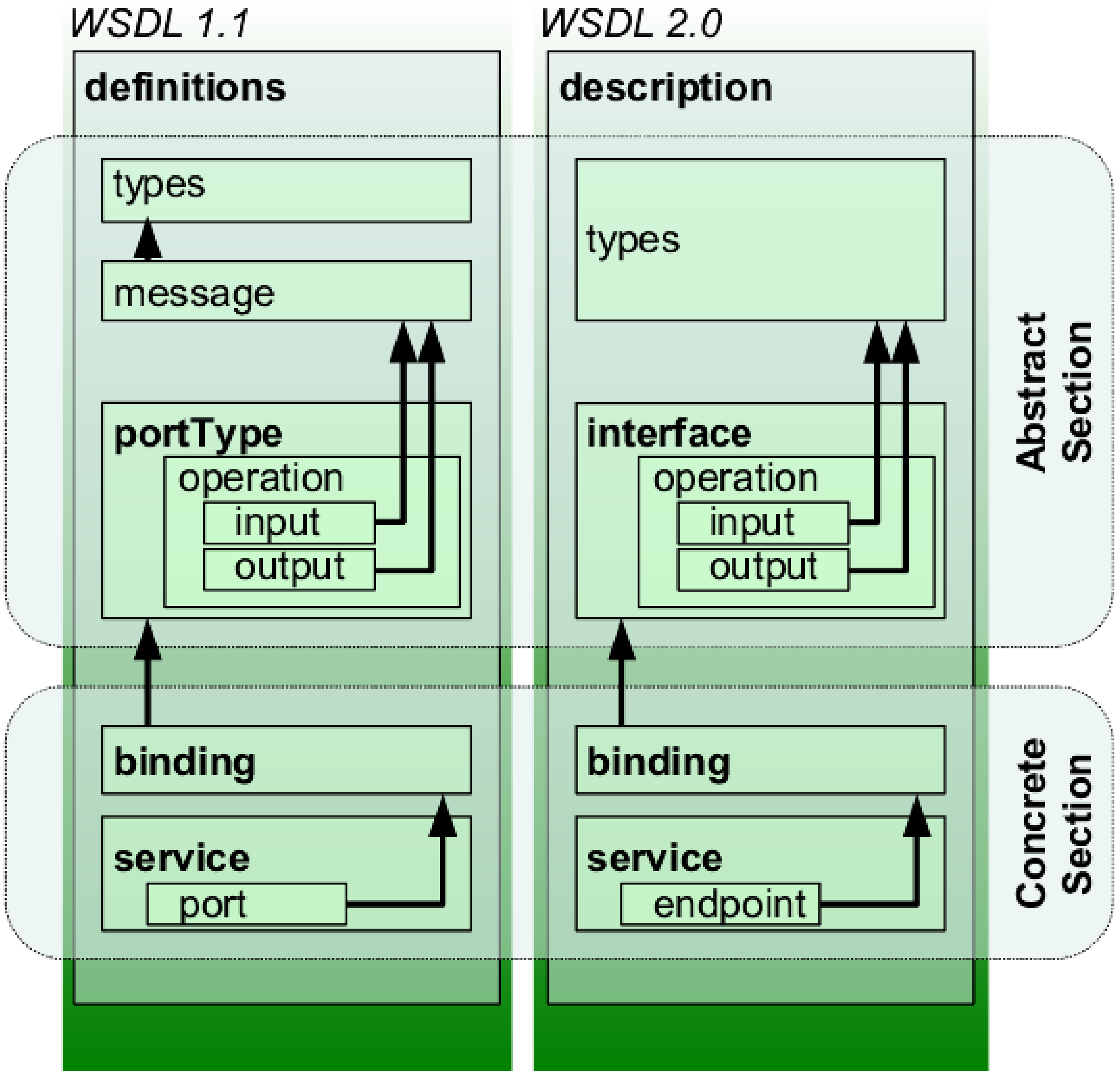
- WSDL provides a notation to answer the following three questions:
  - What is the service about?
  - Where does it reside?
  - How can it be invoked? (i.e., what, where, and how)

What is WSDL

- WSDL is a specification defining how to describe web services in a common XML grammar.
- WSDL describes four critical pieces of data:
  - Interface information describing all publicly available functions
  - Data type information for all message requests and message responses
  - Binding information about the transport protocol to be used
  - Address information for locating the specified service

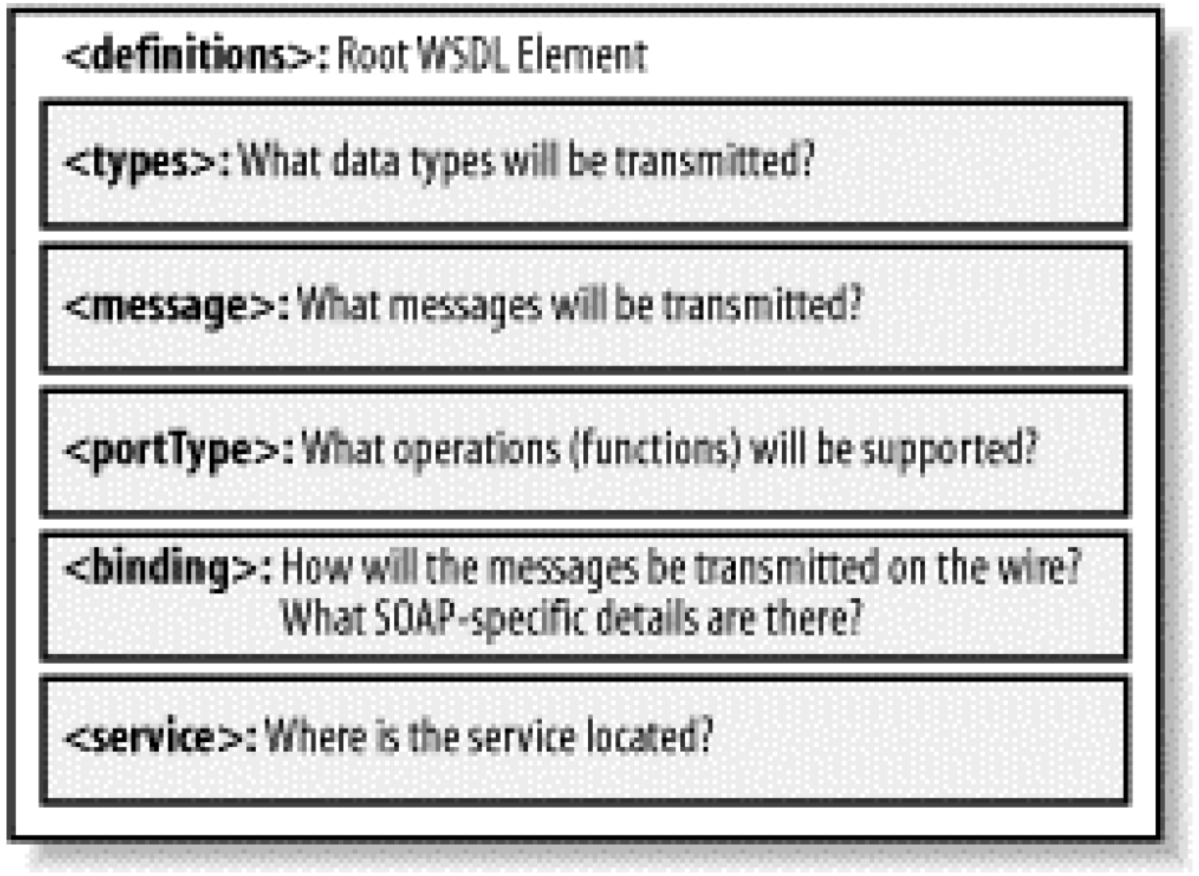
WSDL

- An XML-based interface description language
- Used for describing the functionality offered by a web service
- WSDL describes services as collections of network endpoints or ports



WSDL Specification major Elements

- definitions
- types
- message
- portType
- binding
- service



<definitions>: Root WSDL Element
<types>: What data types will be transmitted?
<message>: What messages will be transmitted?
<portType>: What operations (functions) will be supported?
<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?
<service>: Where is the service located?

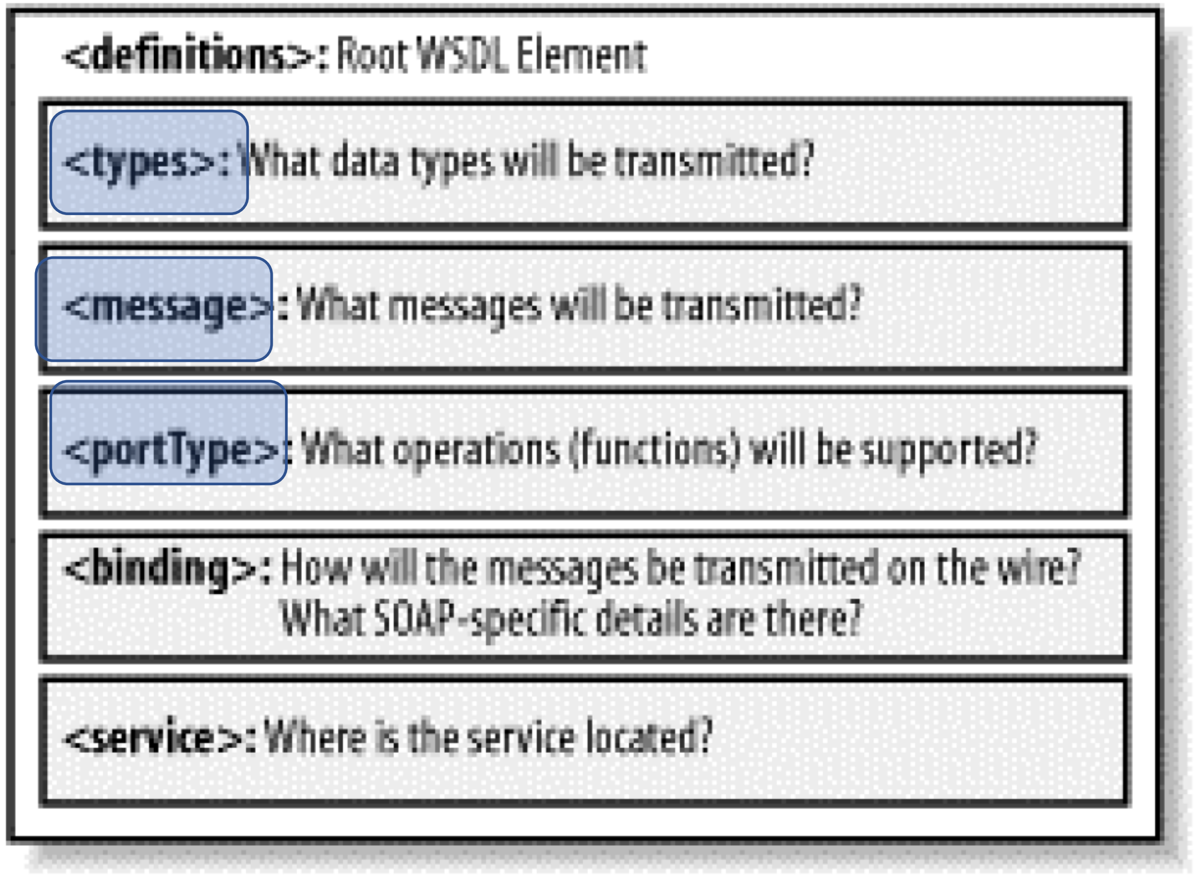
- Back to our example

Step 3: Derive an initial service interface

- Confirm that each operation candidate is suitably reusable and its granularity is appropriate.
    1. Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema.
    2. Establish a set of operation names and define WSDL <portType> with <operation>.
    3. Define WSDL <message> with <part>.

Step 3: Derive an initial service interface

- Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema.
- Establish a set of operation names and define WSDL <portType> with <operation>.
- Define WSDL <message> with <part>.


<definitions>: Root WSDL Element

<types>: What data types will be transmitted?

<message>: What messages will be transmitted?

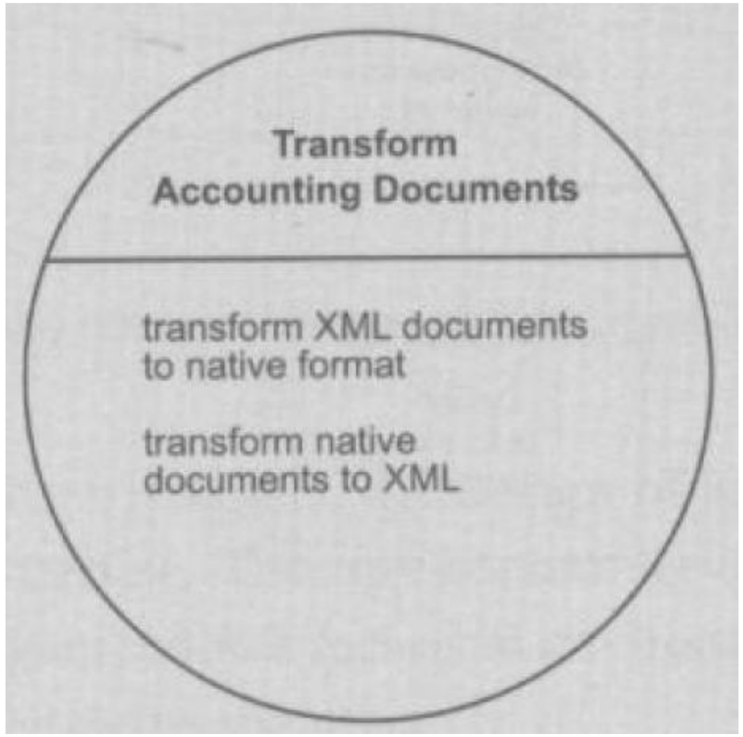
<portType>: What operations (functions) will be supported?

<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?

<service>: Where is the service located?

Activity

- Let's first brainstorm potential names for the operations of our service


Transform
Accounting Documents
transform XML documents to native format
transform native documents to XML

Open Question Points: 10

Setting

Currently the candidate service operations in the Transform Service are called

- transform XML documents to native format
- transform native documents to XML

Propose two names for the operations of our service that can be used in WSDL file.

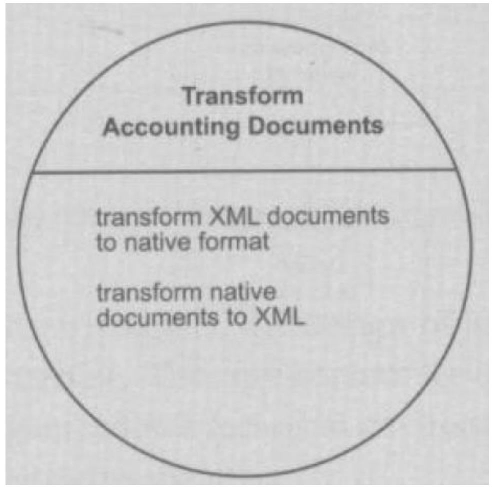
The name should be as short as possible, yet descriptive enough to understand the function of the operation.

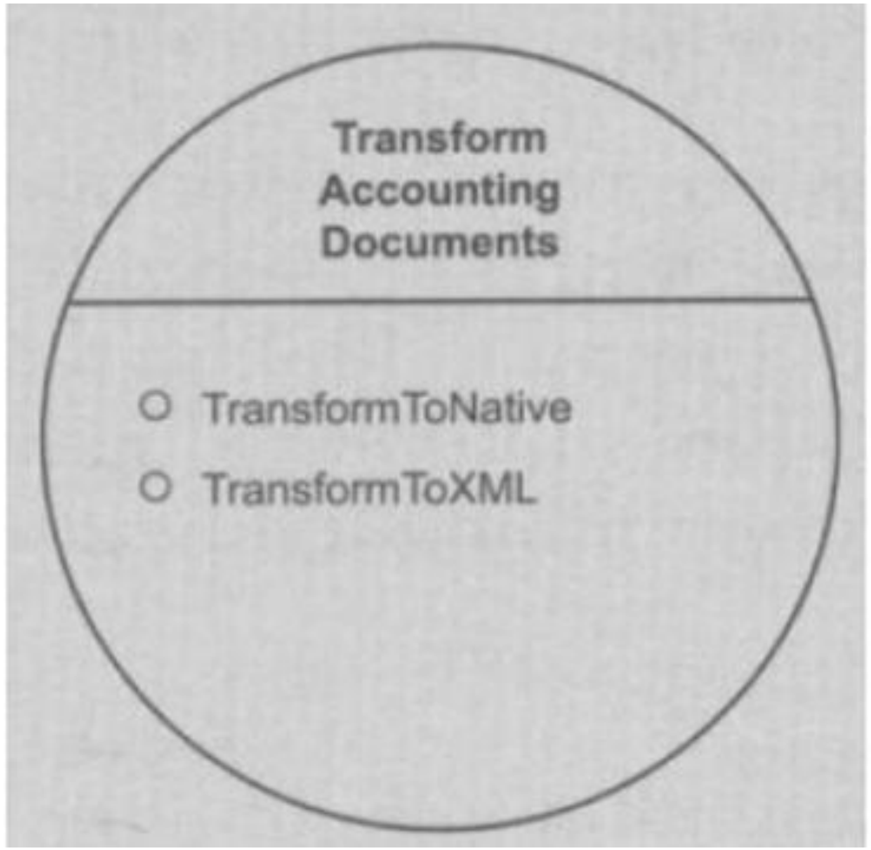
Open Question is only supported on Version 2.0 or newer.

Answer

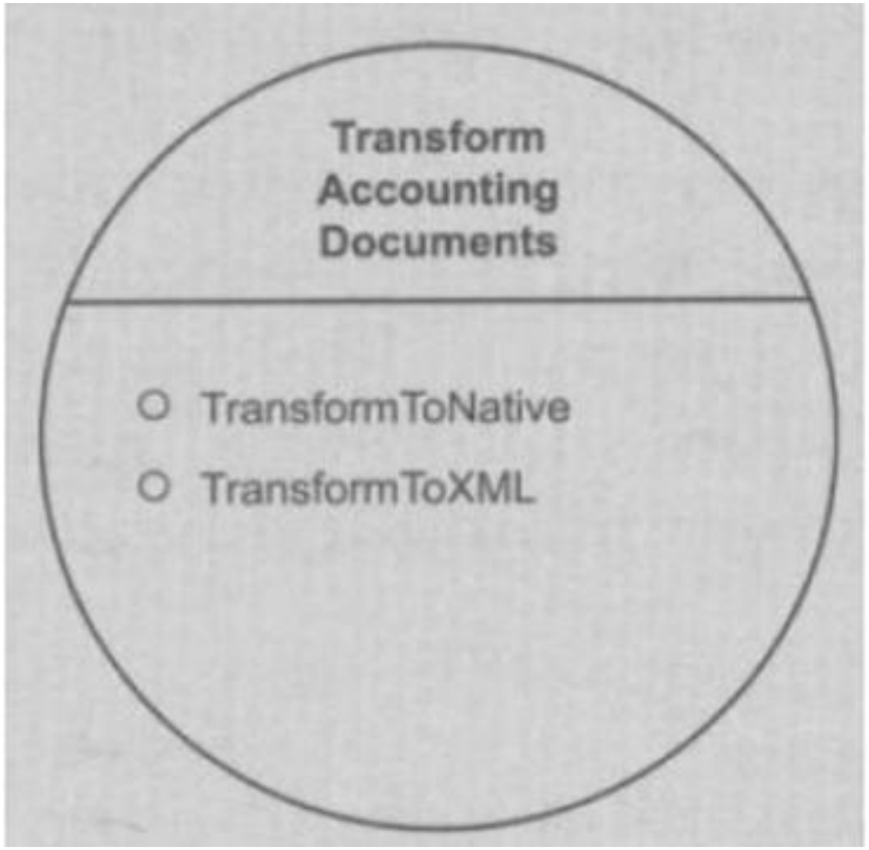
Confirm that each operation candidate is suitably reusable and its granularity is appropriate

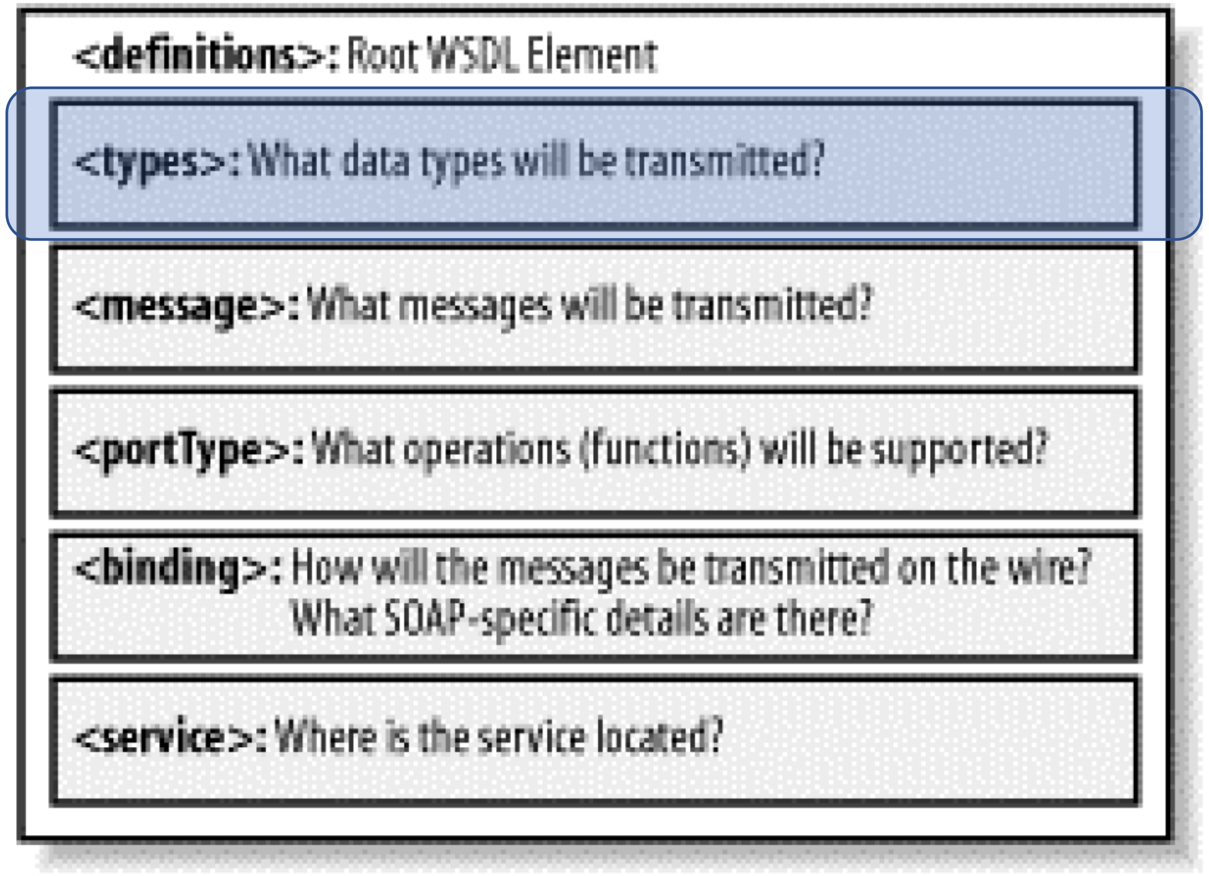
- RailCo begins by the two operation names shown here
- It then moves on to define the *types* construct of its service definition to formalize the message structures

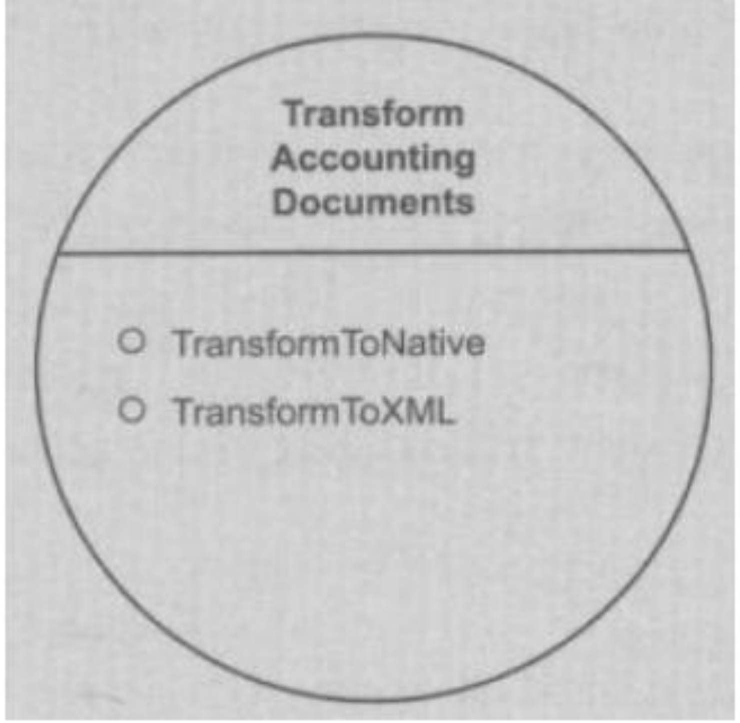

Transform Accounting Documents
transform XML documents to native format
transform native documents to XML



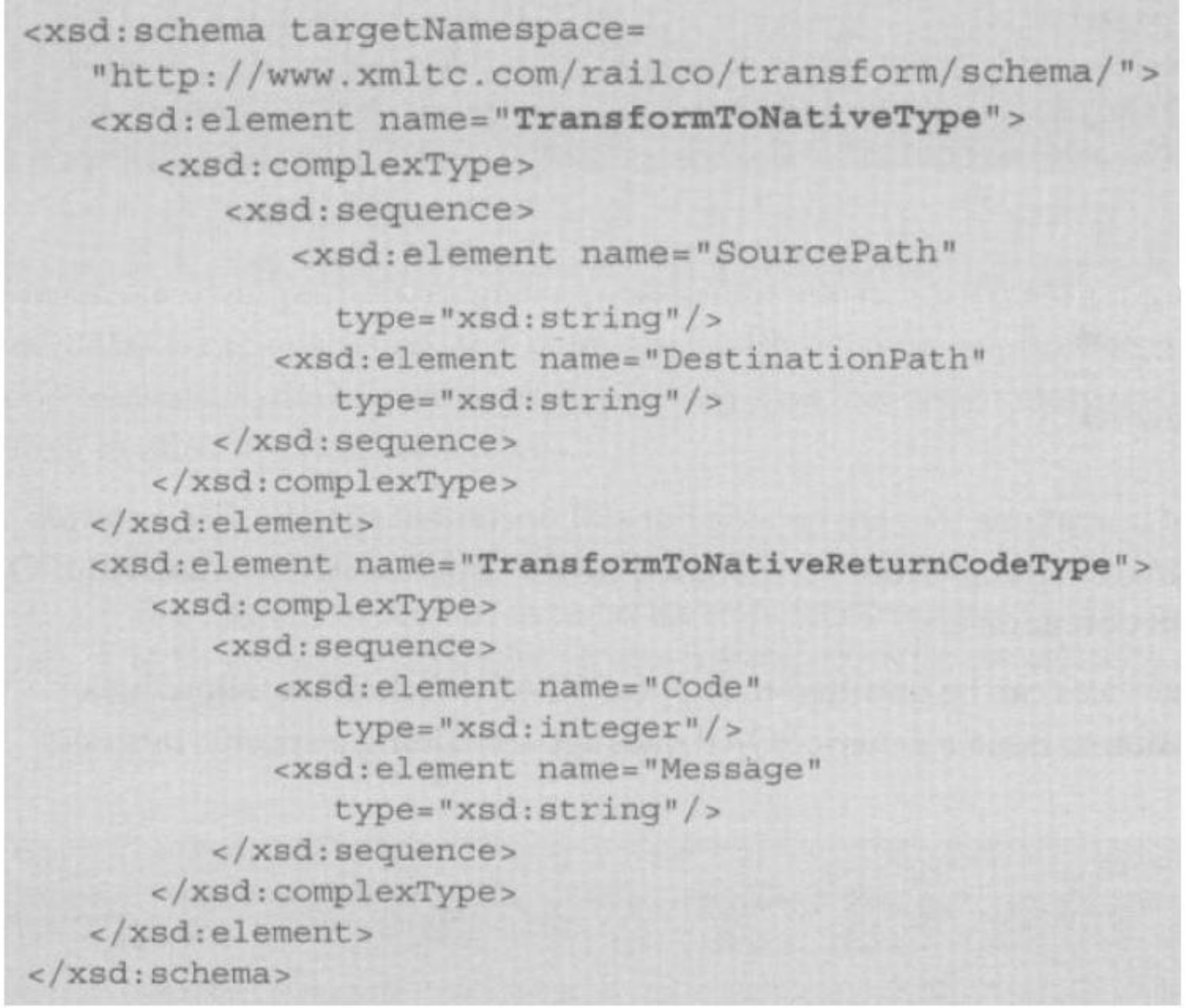
Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema




<definitions>: Root WSDL Element
<types>: What data types will be transmitted?
<message>: What messages will be transmitted?
<portType>: What operations (functions) will be supported?
<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?
<service>: Where is the service located?



The request and response messages for the TransformtoNative operation



```
<xsd:schema targetNamespace=
    "http://www.xmltc.com/railco/transform/schema/">
    <xsd:element name="TransformToNativeType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="SourcePath"
                    type="xsd:string"/>
                <xsd:element name="DestinationPath"
                    type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="TransformToNativeReturnCodeType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Code"
                    type="xsd:integer"/>
                <xsd:element name="Message"
                    type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Activity

- We have designed the request and response messages for the TransformtoNative operation
- The next step is to design the request and response messages for the TransformtoXML operation

Open Question Points: 10

Setting

Design the request and response messages for the TransformtoXML operation. (*hint: the types for the TransformtoXML operation are the same as the TransformtoNative operation*)

Open Question is only supported on Version 2.0 or newer.

Answer

Discussion

- We observe, that the types for the second operation are the same as the first operation – in such case, the same schema for these two operations be shared?

Multiple Choice(single)

Points: 1

设置

We have designed the request and response messages for the TransformtoNative operation
The next step is to design the request and response messages for the TransformtoXML operation
We observe, that the types for the second operation are the same as the first operation – in such case, the same schema for these two operations be shared?

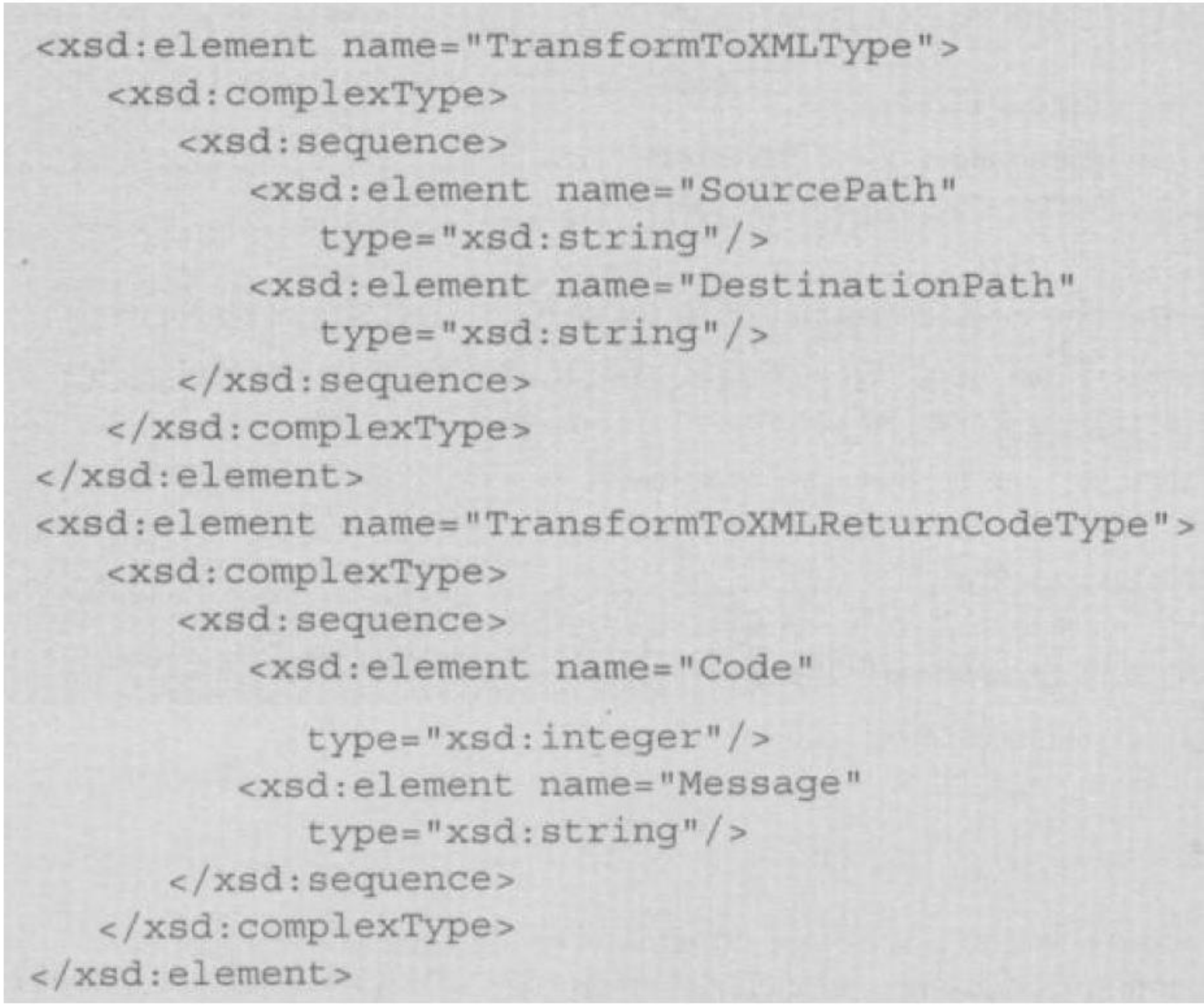
[✓ A] YES

B NO

提交

Types for TransformToXML are identical to those of TransformToNative and can be shared. But RailCo decides to keep a separate schema.

What is the benefit of this decision?



```
<xsd:element name="TransformToXMLType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="SourcePath"
            type="xsd:string"/>
         <xsd:element name="DestinationPath"
            type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="TransformToXMLReturnCodeType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Code"

           type="xsd:integer"/>
        <xsd:element name="Message"
           type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Open Question Points: 10

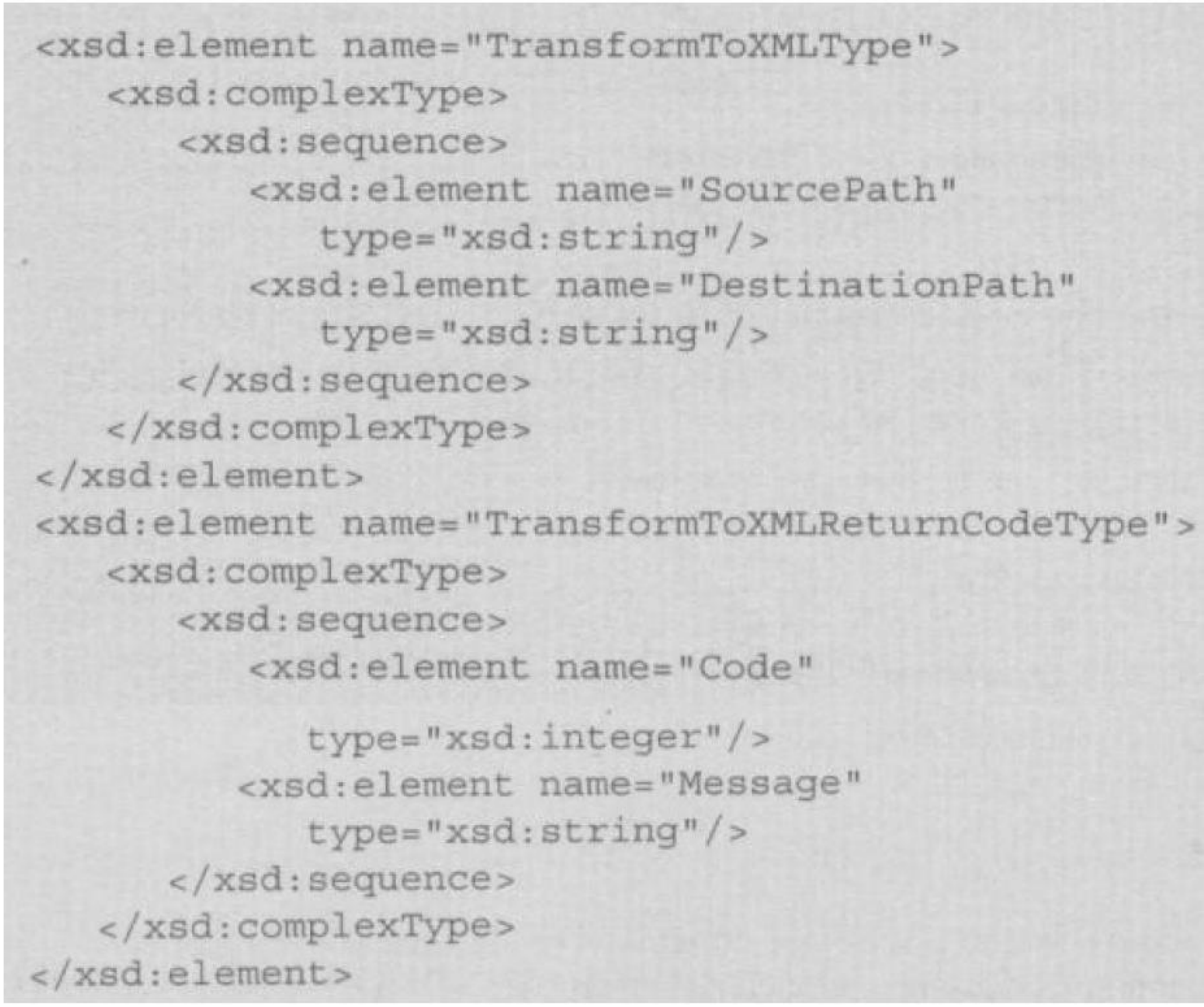
Setting

Even though types for TransformToXML are identical to those of TransformToNative and can be shared, however RailCo decides to keep a separate schema.

What is the benefit of keeping these two schema in separate files?

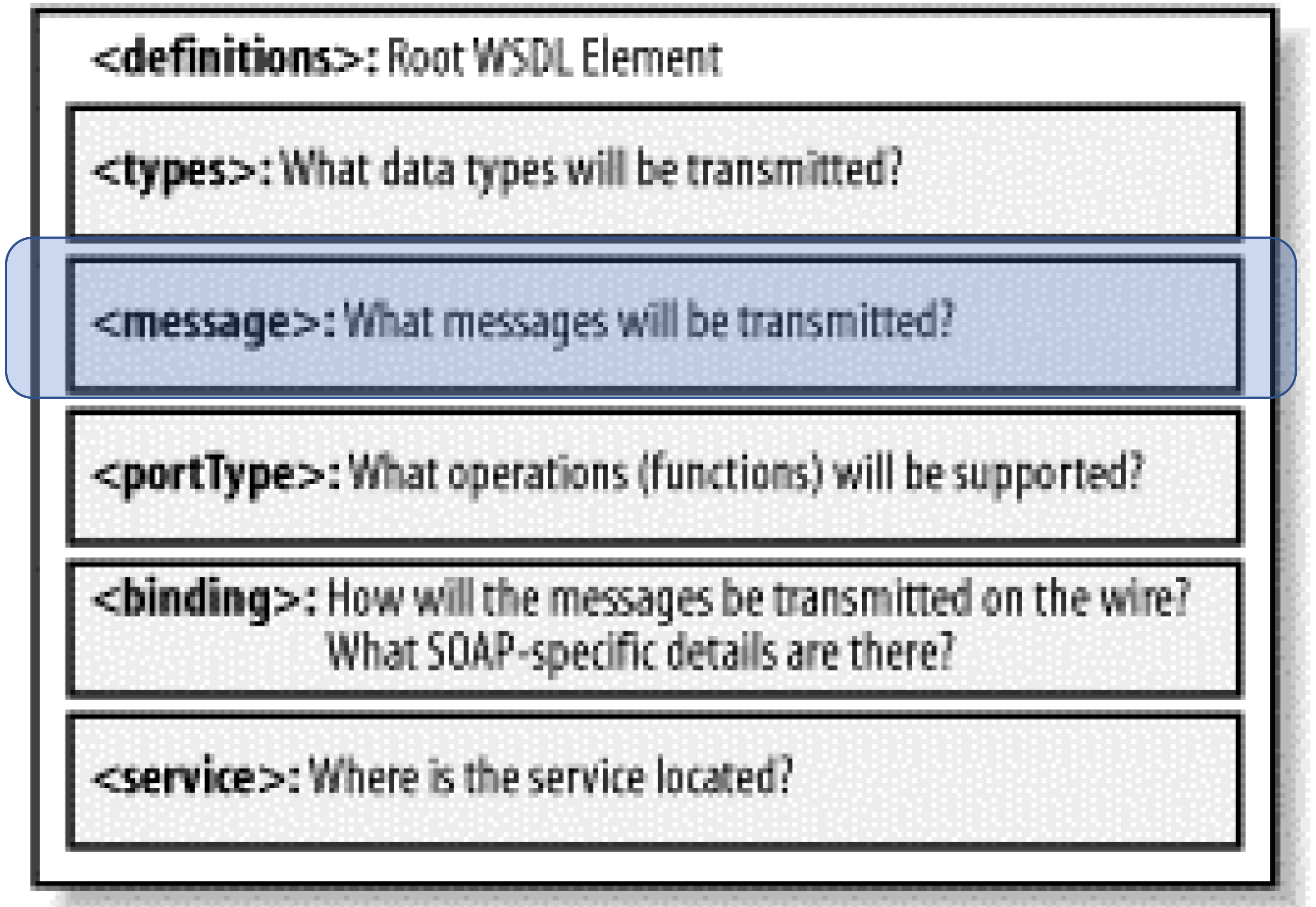
Answer

Types for TransformToXML are identical to those of TransformToNative and can be shared. But RailCo decides to keep a separate schema to have freedom to change them independently.



```
<xsd:element name="TransformToXMLType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="SourcePath"
            type="xsd:string"/>
         <xsd:element name="DestinationPath"
            type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="TransformToXMLReturnCodeType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Code"
            type="xsd:integer"/>
         <xsd:element name="Message"
            type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Define WSDL <message> with <part>



<definitions>: Root WSDL Element

<types>: What data types will be transmitted?

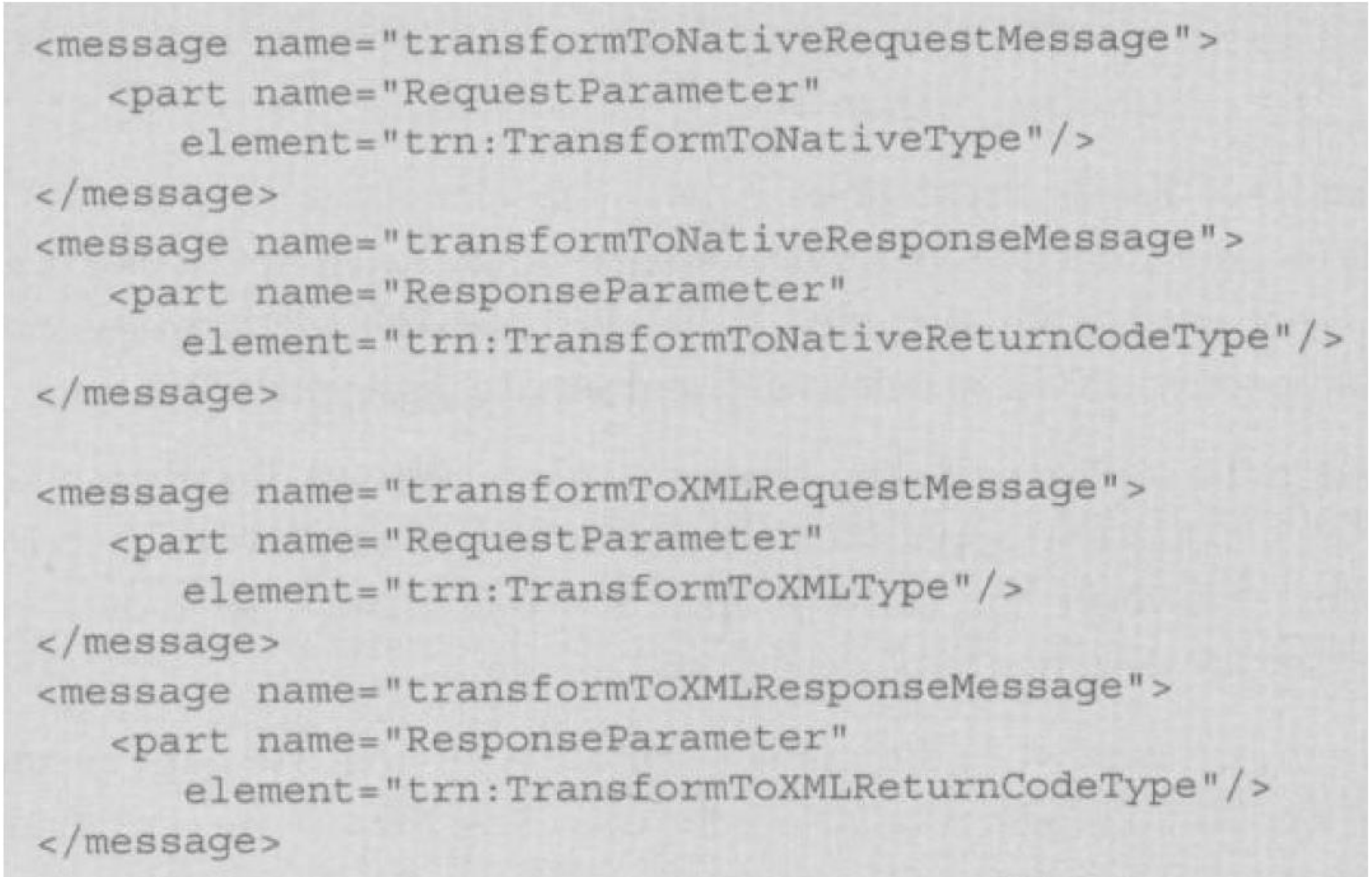
<message>: What messages will be transmitted?

<portType>: What operations (functions) will be supported?

<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?

<service>: Where is the service located?

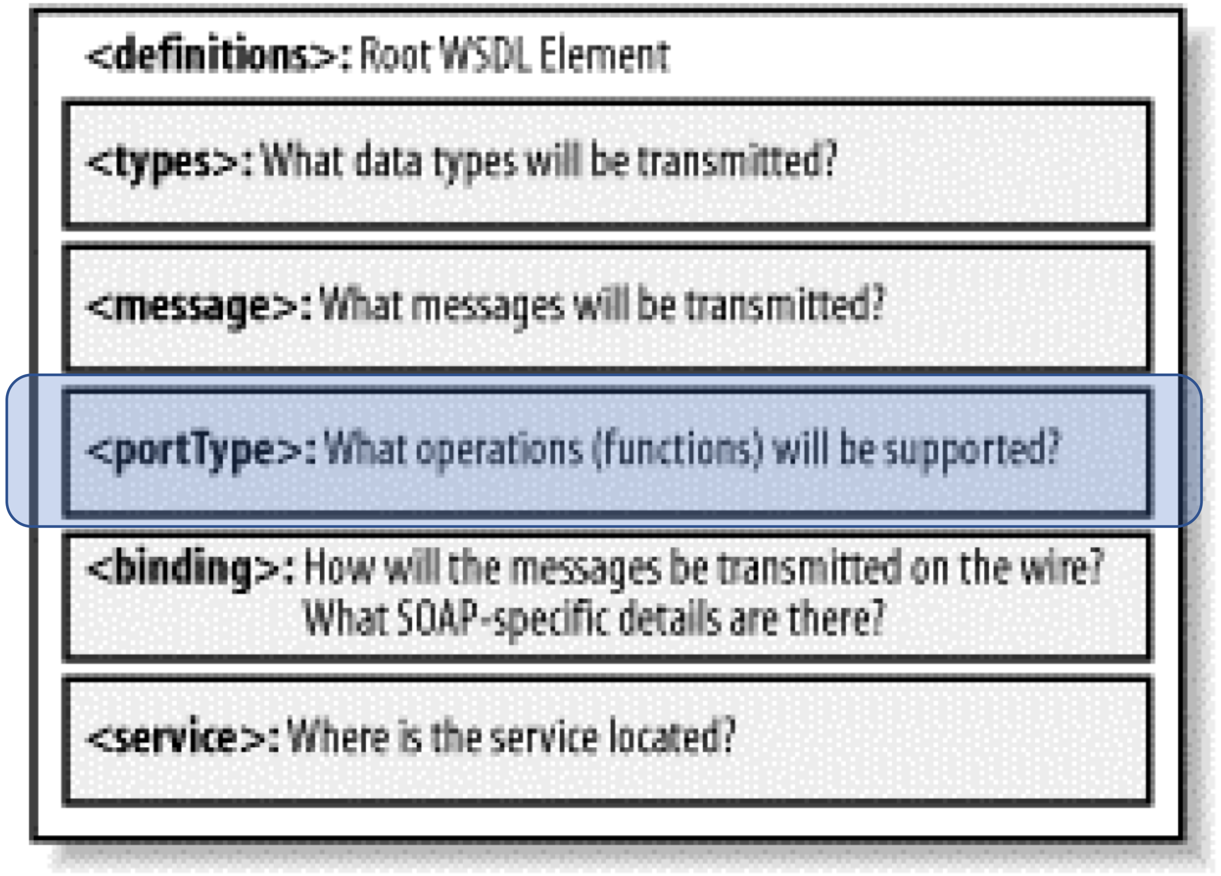
Define WSDL <message> with <part>



```
<message name="transformToNativeRequestMessage">
   <part name="RequestParameter"
      element="trn:TransformToNativeType"/>
</message>
<message name="transformToNativeResponseMessage">
   <part name="ResponseParameter"
      element="trn:TransformToNativeReturnCodeType"/>
</message>

<message name="transformToXMLRequestMessage">
   <part name="RequestParameter"
      element="trn:TransformToXMLType"/>
</message>
<message name="transformToXMLResponseMessage">
   <part name="ResponseParameter"
      element="trn:TransformToXMLReturnCodeType"/>
</message>
```

Establish a set of operation names and define WSDL <portType> with <operation>



<definitions>: Root WSDL Element

<types>: What data types will be transmitted?

<message>: What messages will be transmitted?

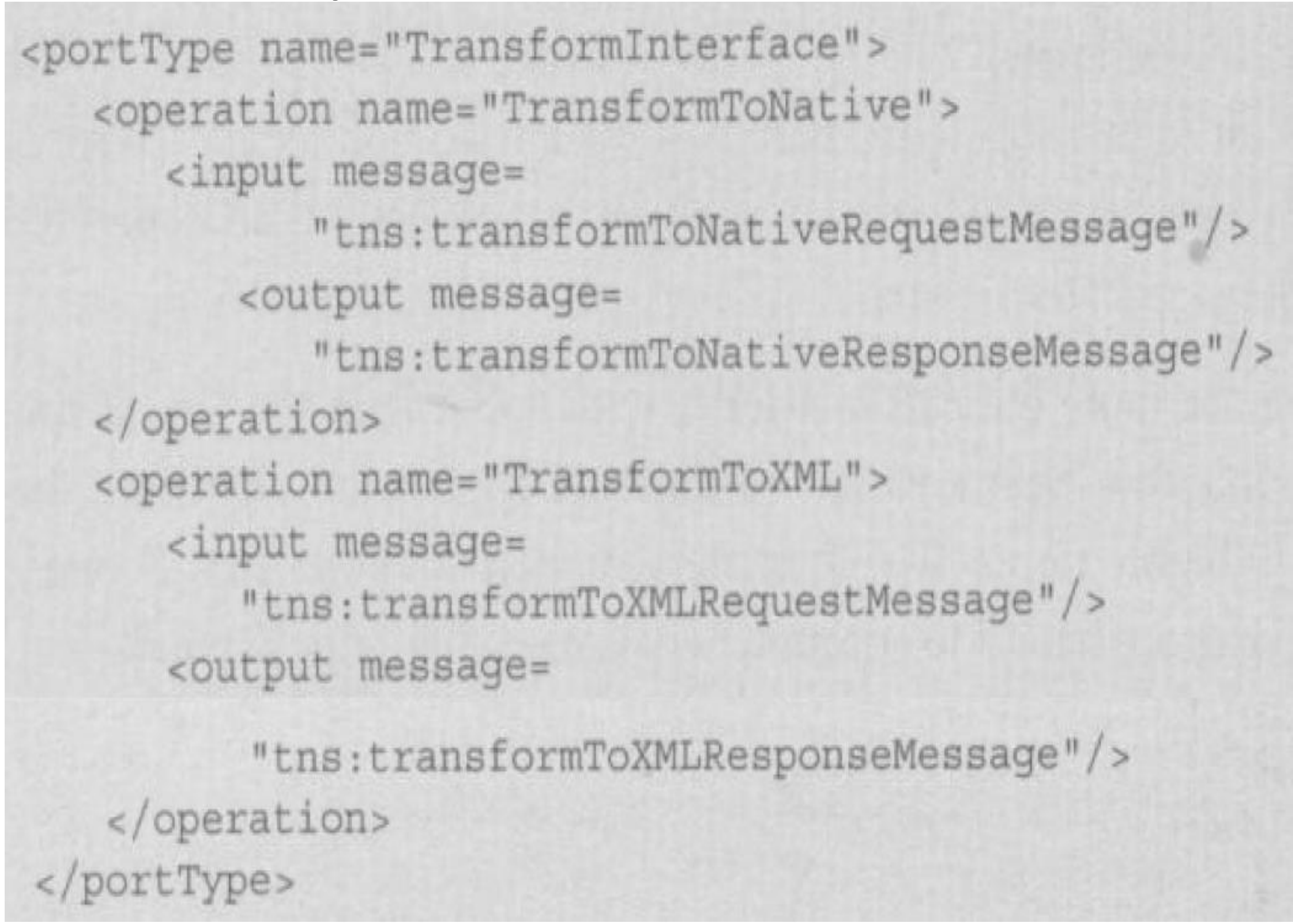
<portType>: What operations (functions) will be supported?

<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?

<service>: Where is the service located?

Establish a set of operation names and define WSDL <portType> with <operation>

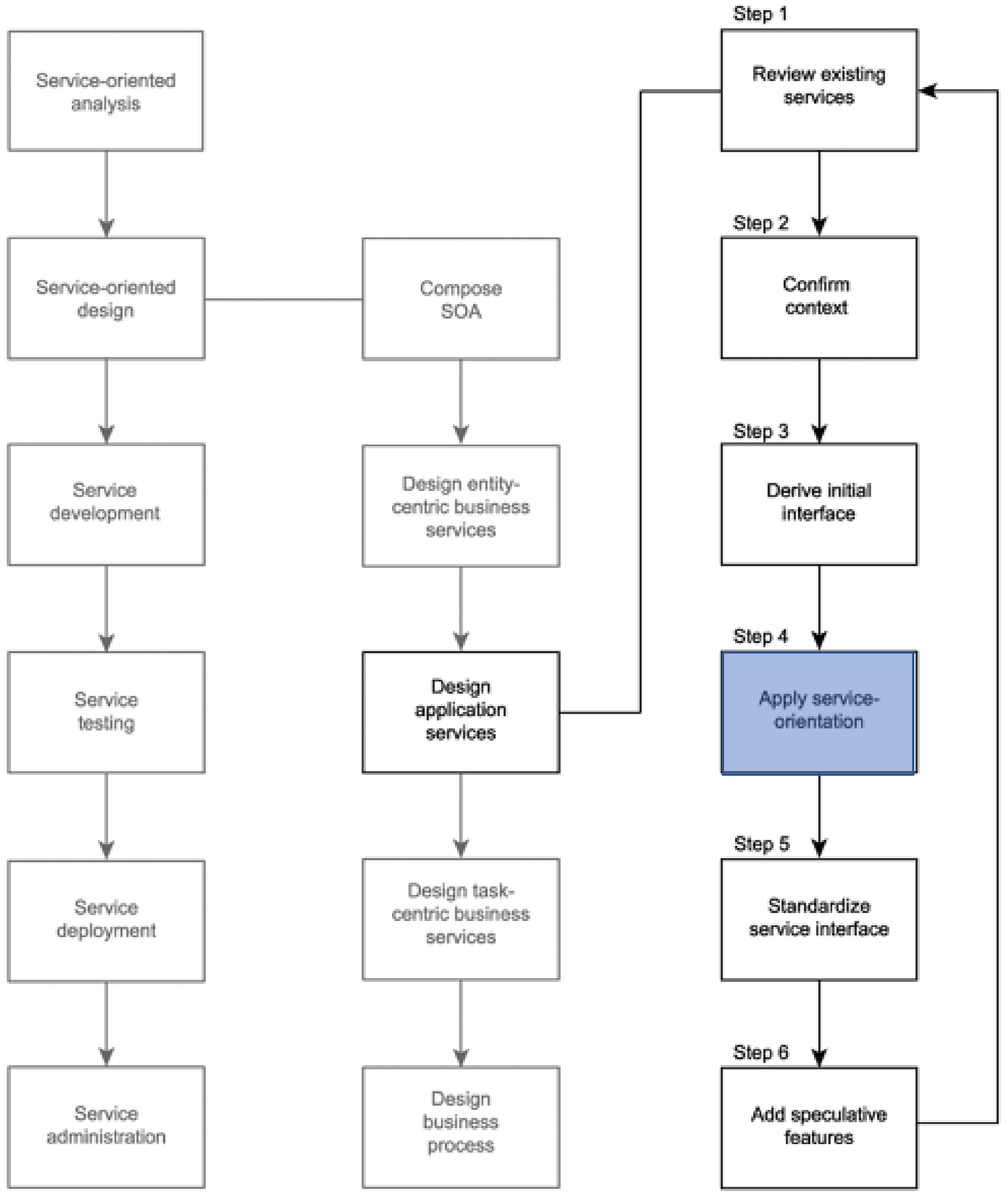
Now the initial version of the abstract service definition for the RailCo Transform Account Document Service is completed



```
<portType name="TransformInterface">
   <operation name="TransformToNative">
      <input message=
            "tns:transformToNativeRequestMessage"/>
         <output message=
            "tns:transformToNativeResponseMessage"/>
   </operation>
   <operation name="TransformToXML">
      <input message=
         "tns:transformToXMLRequestMessage"/>
      <output message=

         "tns:transformToXMLResponseMessage"/>
   </operation>
</portType>
```

Service-oriented design



Step 4: Apply service-orientation

- Revisit reusability, autonomy, statelessness, discoverability

Discussion

- Service orientation was already applied during the analysis phase. Why do we need to do it again?

Open Question Points: 10

Setting

Service orientation was already applied during the analysis phase. Why do we need to do it again?

Answer

Activity

- To provide higher reuse potential - could we combine the two operations TransformToNative and TransformToXML?

Multiple Choice(single) Points: 1

Setting

To provide higher reuse potential - could we combine the two operations TransformToNative and TransformToXML?

[✓ A] Yes, combining these two operations will make them more generic and possible to reuse in more situations

B No, there is no benefit from combining these two operations

Submit

Step 4: Apply service-orientation

- Revisit reusability, autonomy, statelessness, discoverability
- Reuse potential - the two operations TransformToNative and TransformToXML are discussed to be combined into one generic operation. But RailCo decides to keep them both for the descriptive nature of the operations and freedom to evolve each of them separately

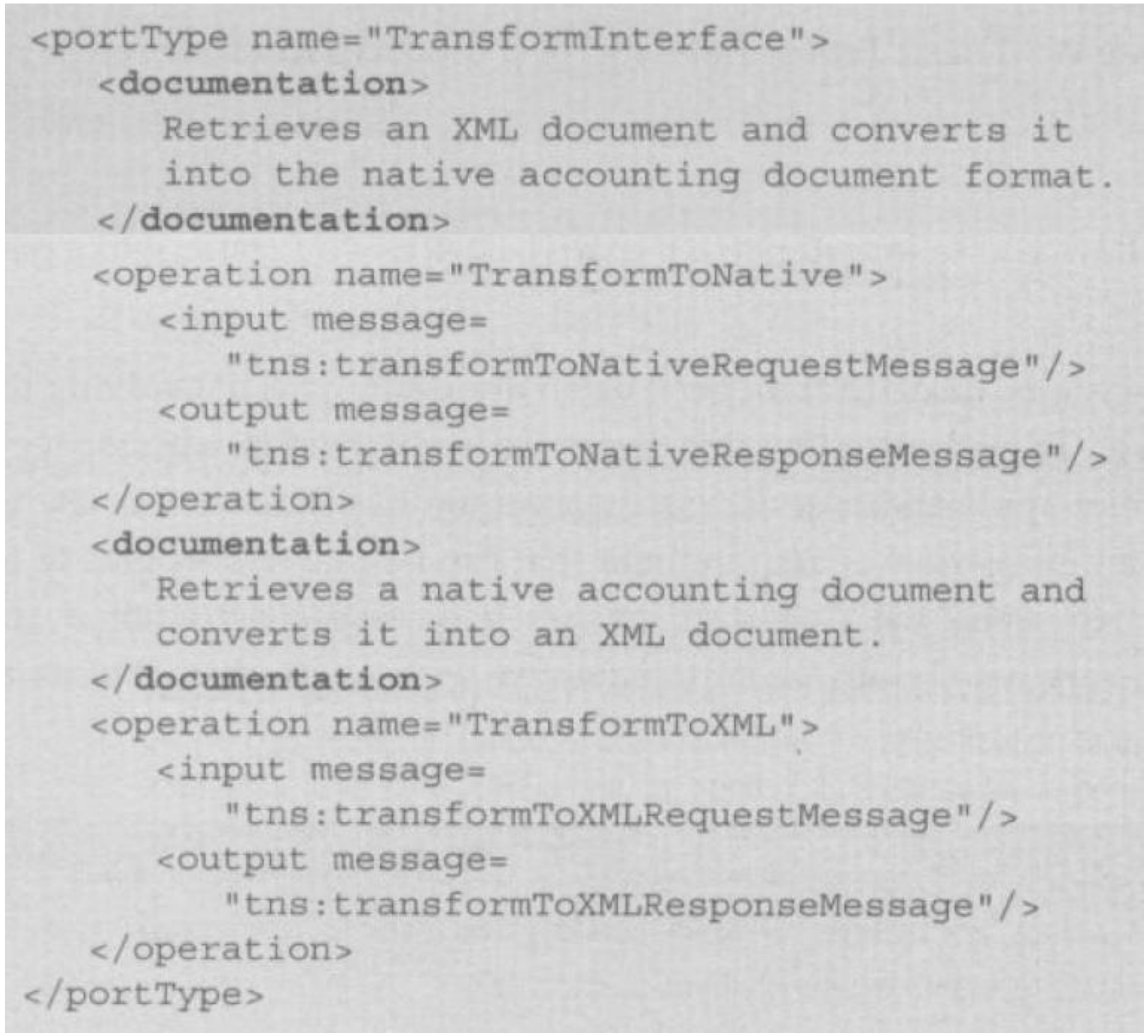
Step 4: Apply service-orientation

- Revisit reusability, autonomy, statelessness, discoverability
- Reuse potential - the two operations TransformToNative and TransformToXML are discussed to be combined into one generic operation. But RailCo decides to keep them both for the descriptive nature of the operations and freedom to evolve each of them separately
- Autonomy and statelessness - no problem as the logic required to carry out the transformation is contained within the particular underlying application logic

Step 4: Apply service-orientation

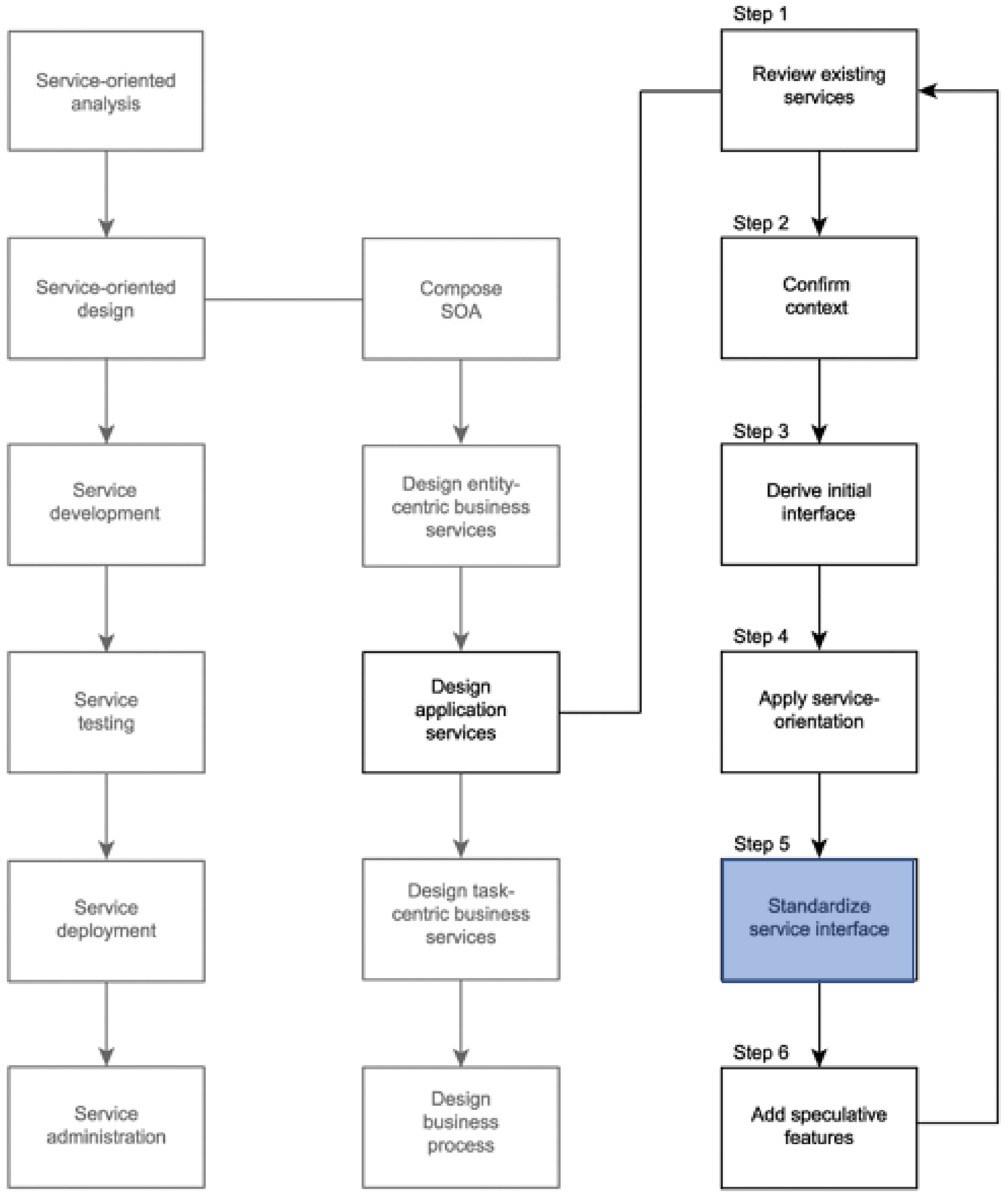
- Revisit reusability, autonomy, statelessness, discoverability
- Reuse potential - the two operations TransformToNative and TransformToXML are discussed to be combined into one generic operation. But RailCo decides to keep them both for the descriptive nature of the operations and freedom to evolve each of them separately
- Autonomy and statelessness - no problem as the logic required to carry out the transformation is contained within the particular underlying application logic
- Discoverability - metadata are added

Service definition is outfitted with additional metadata documentation (check chapter 15.5 for service design guidelines)



```
<portType name="TransformInterface">
   <documentation>
      Retrieves an XML document and converts it
      into the native accounting document format.
   </documentation>
   <operation name="TransformToNative">
      <input message=
         "tns:transformToNativeRequestMessage"/>
      <output message=
         "tns:transformToNativeResponseMessage"/>
   </operation>
   <documentation>
      Retrieves a native accounting document and
      converts it into an XML document.
   </documentation>
   <operation name="TransformToXML">
      <input message=
         "tns:transformToXMLRequestMessage"/>
      <output message=
         "tns:transformToXMLResponseMessage"/>
   </operation>
</portType>
```

Service-oriented design

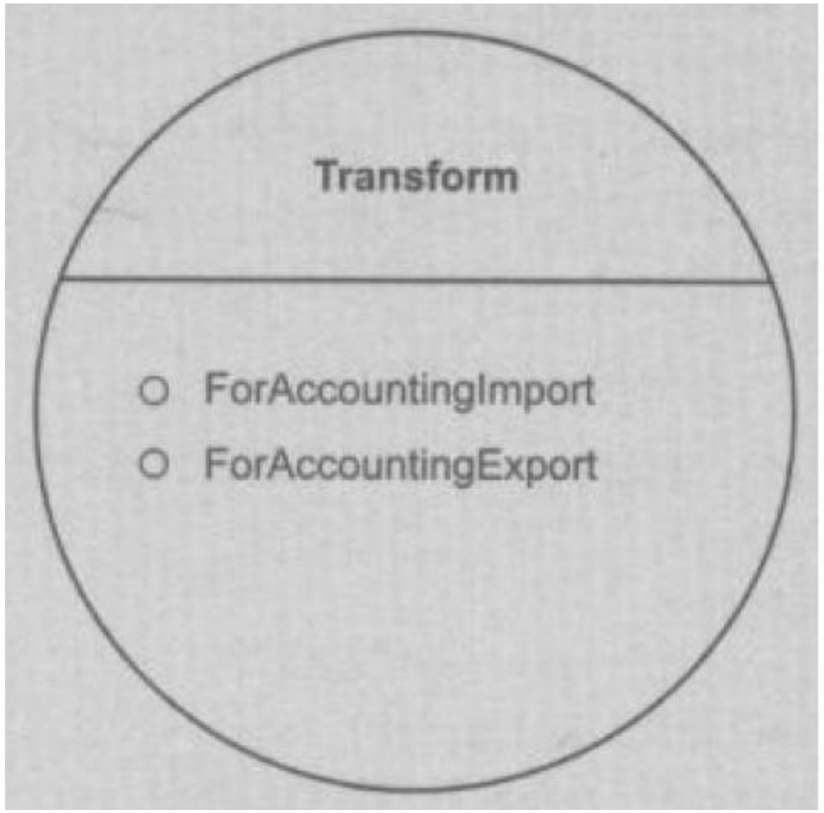


Step 5: Standardize service interface

- Apply appropriately any existing design standards and guidelines.
  - E.g. naming convention: Names should be generic for high reuse potential and clearly communicate the processing context (verb+noun or noun).
- Consider quality service design, e.g. extensibility.

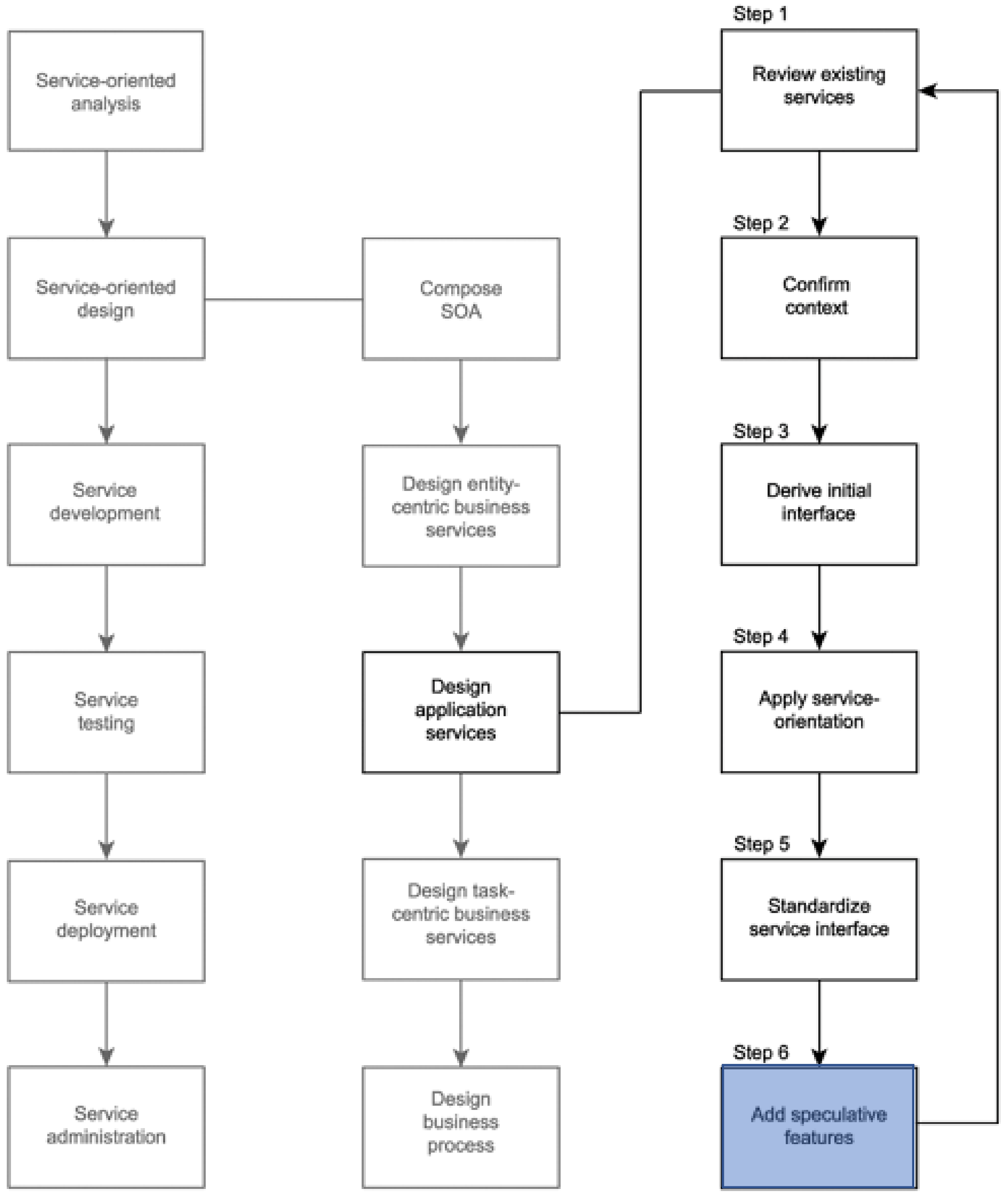
Step 5: Standardize service interface

- RailCo thinks service and operation names are already appropriate. However, the names are changed for future extensibility. The service becomes a transformation utility that can be extended in the future to offer other transformation related features (not only for use by an accounting system).



```
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/railco/transform/schema/">
      <xsd:element name="ForImportType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="SourcePath"
                  type="xsd:string"/>
               <xsd:element name="DestinationPath"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForImportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code"
                  type="xsd:integer"/>
               <xsd:element name="Message"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForExportType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="SourcePath"
                  type="xsd:string"/>
               <xsd:element name="DestinationPath"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForExportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code"
                  type="xsd:integer"/>
               <xsd:element name="Message"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>
```

Service-oriented design



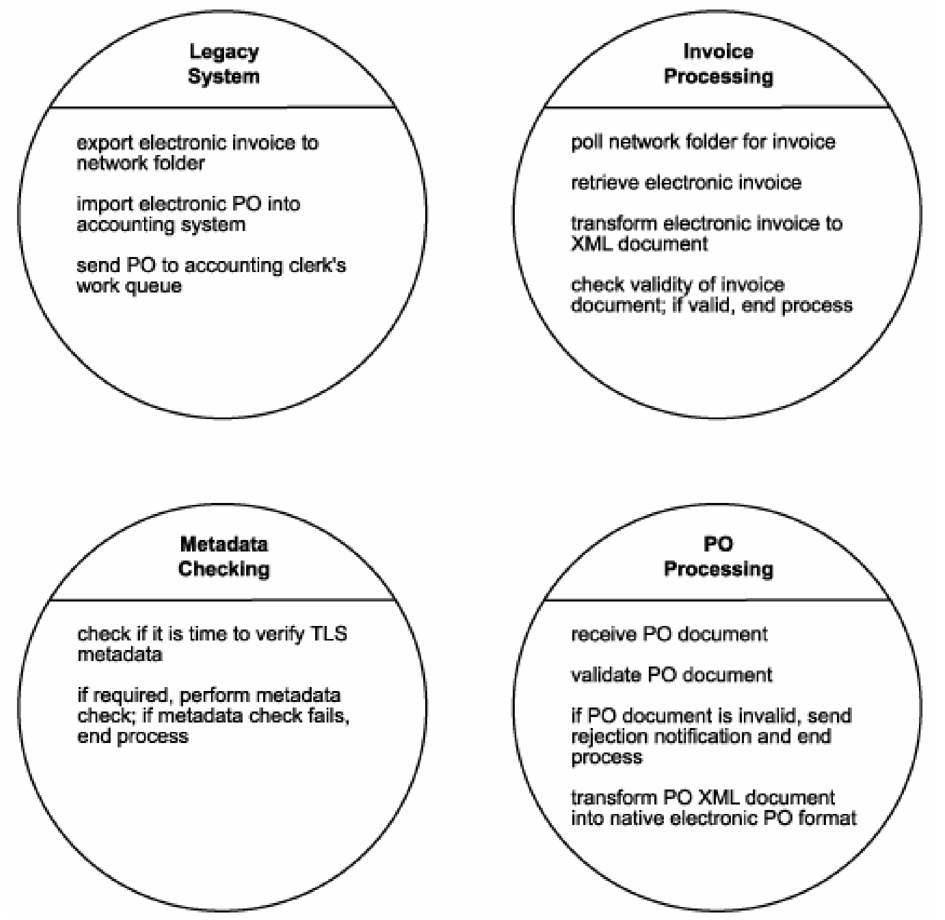
Step 6: Add speculative features

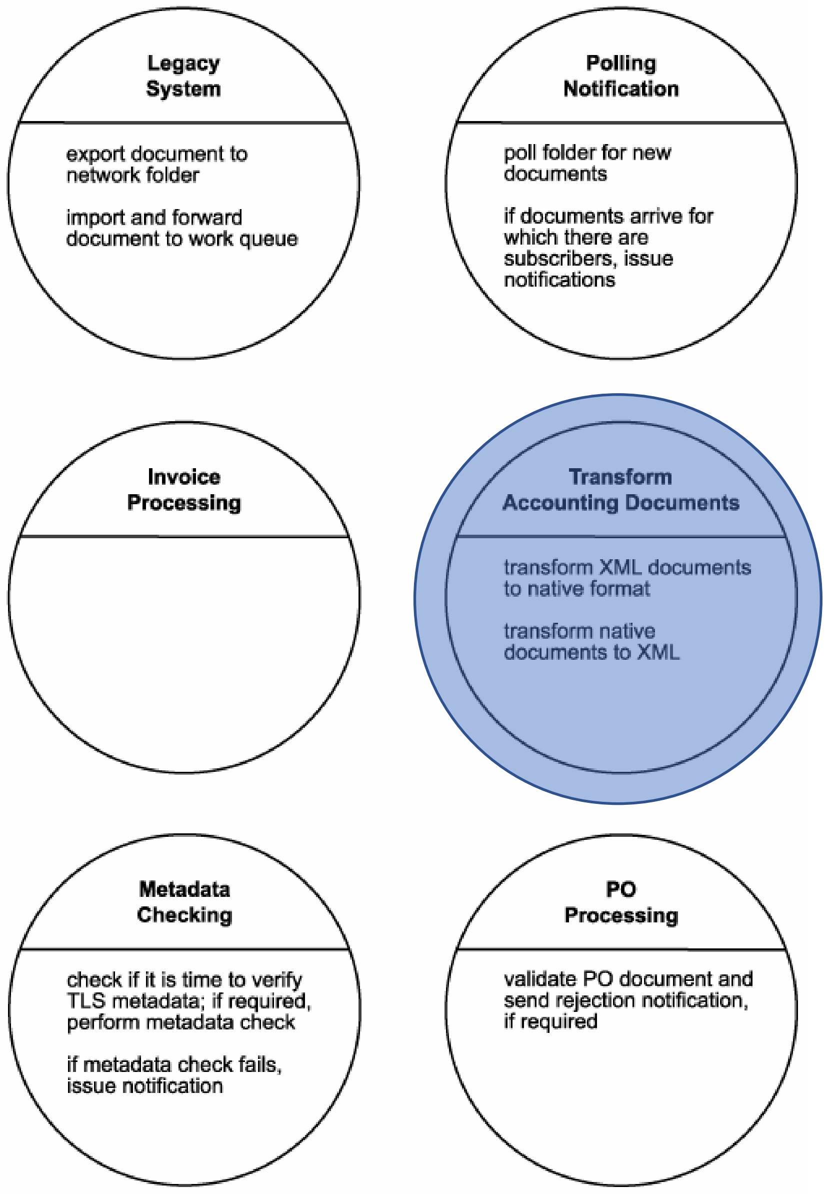
- Perform speculative analysis as to what other processing falls within the service context
- Repeat steps 1-5

Step 6: Add speculative features

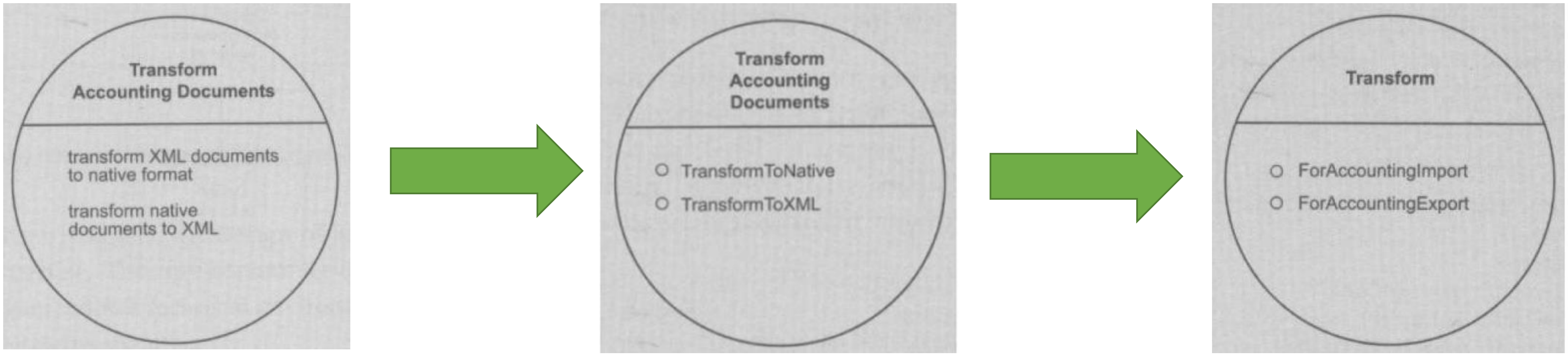
- Perform speculative analysis as to what other processing falls within the service context.
- Repeat steps 1-5.
- RailCo cannot afford any extension at this time. But the design of Transform service already gives future reusability opportunity.

From service candidates to refined service candidates during service analysis


Legacy System
export electronic invoice to network folder
import electronic PO into accounting system
send PO to accounting clerk's work queue
Invoice Processing
poll network folder for invoice
retrieve electronic invoice
transform electronic invoice to XML document
check validity of invoice document; if valid, end process
Metadata Checking
check if it is time to verify TLS metadata
if required, perform metadata check; if metadata check fails, end process
PO Processing
receive PO document
validate PO document
if PO document is invalid, send rejection notification and end process
transform PO XML document into native electronic PO format

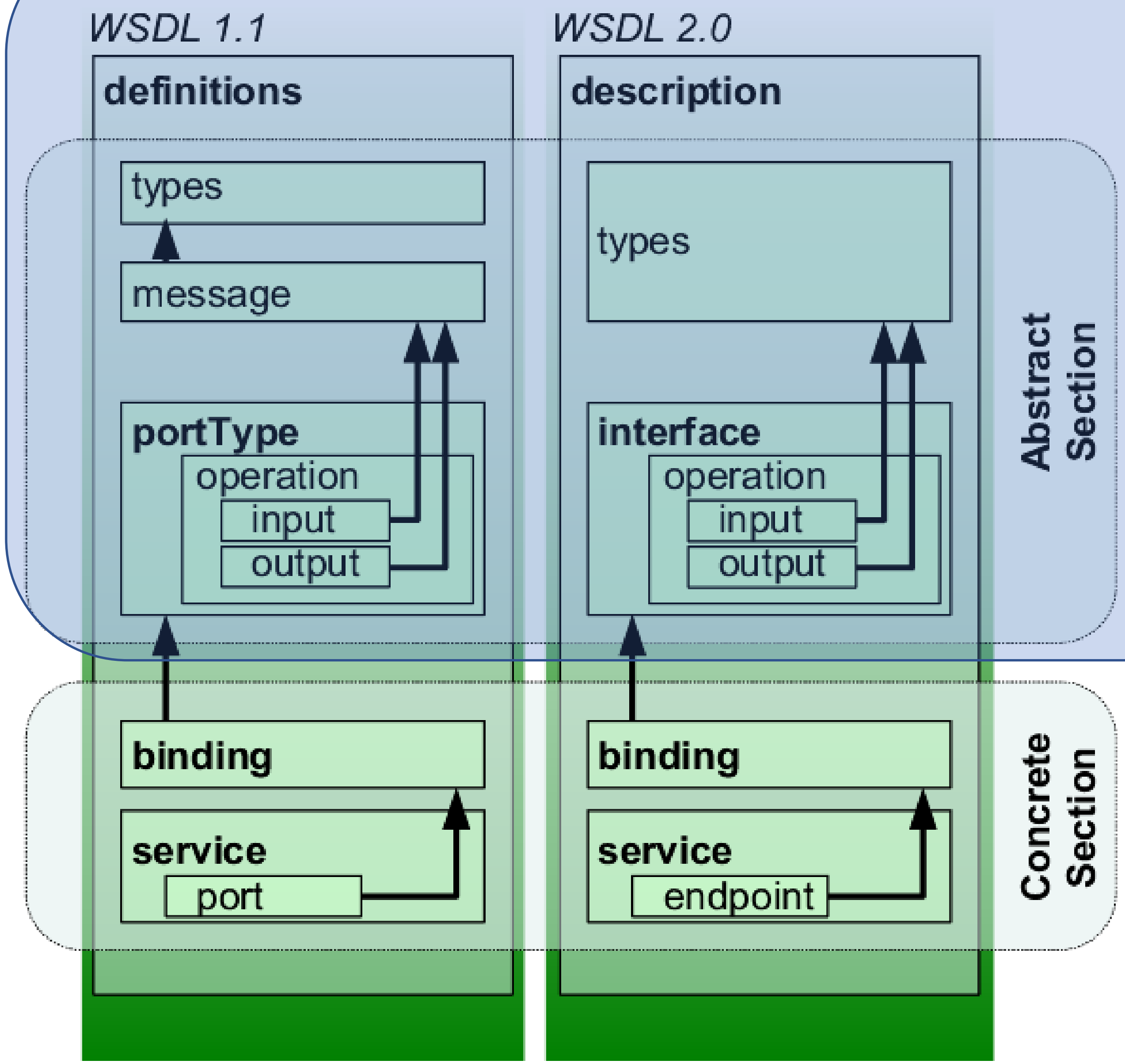

Legacy System
export document to network folder
import and forward document to work queue
Polling Notification
poll folder for new documents
if documents arrive for which there are subscribers, issue notifications
Invoice Processing
Transform Accounting Documents
transform XML documents to native format
transform native documents to XML
Metadata Checking
check if it is time to verify TLS metadata; if required, perform metadata check
if metadata check fails, issue notification
PO Processing
validate PO document and send rejection notification, if required

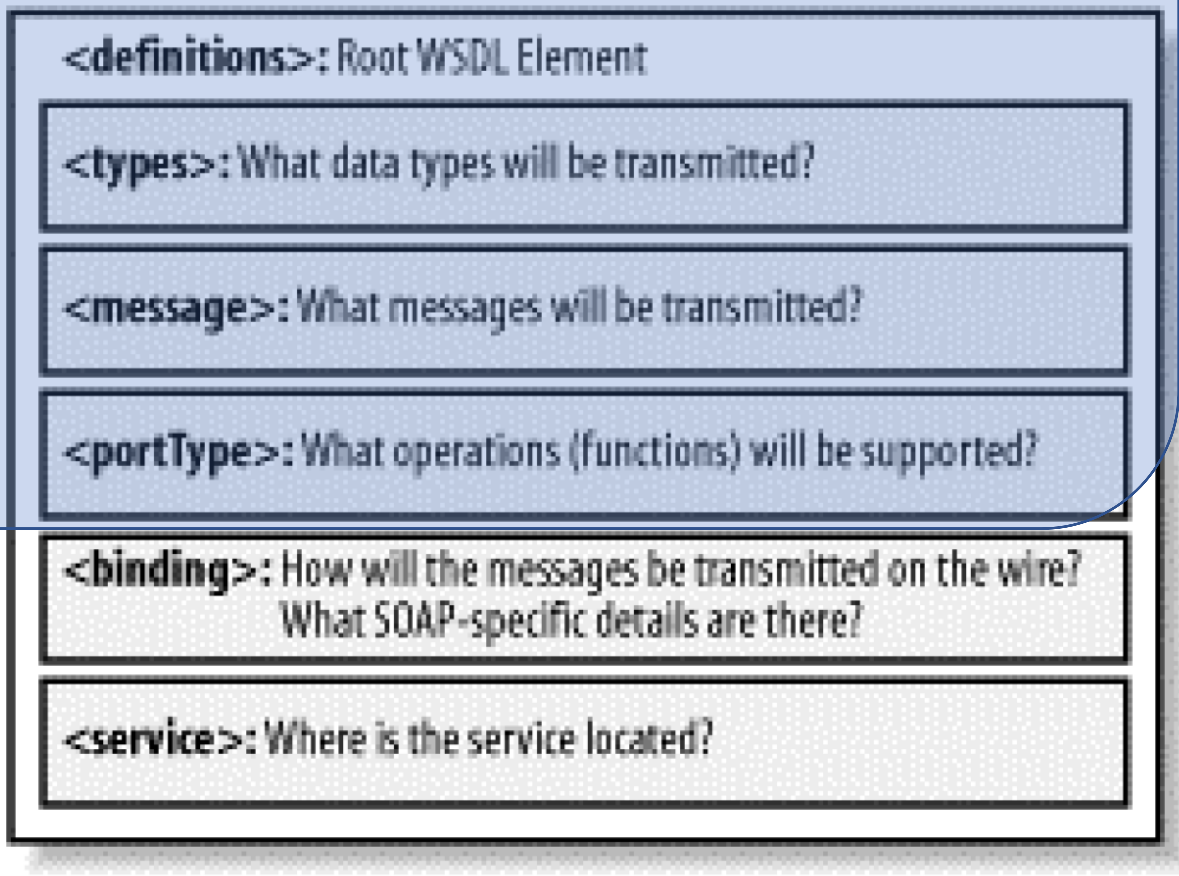
Further refinements during service design



Service design

- In this phase, we got our abstract part of WSDL, we still need to design a full WSDL




<definitions>: Root WSDL Element

<types>: What data types will be transmitted?

<message>: What messages will be transmitted?

<portType>: What operations (functions) will be supported?

<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?

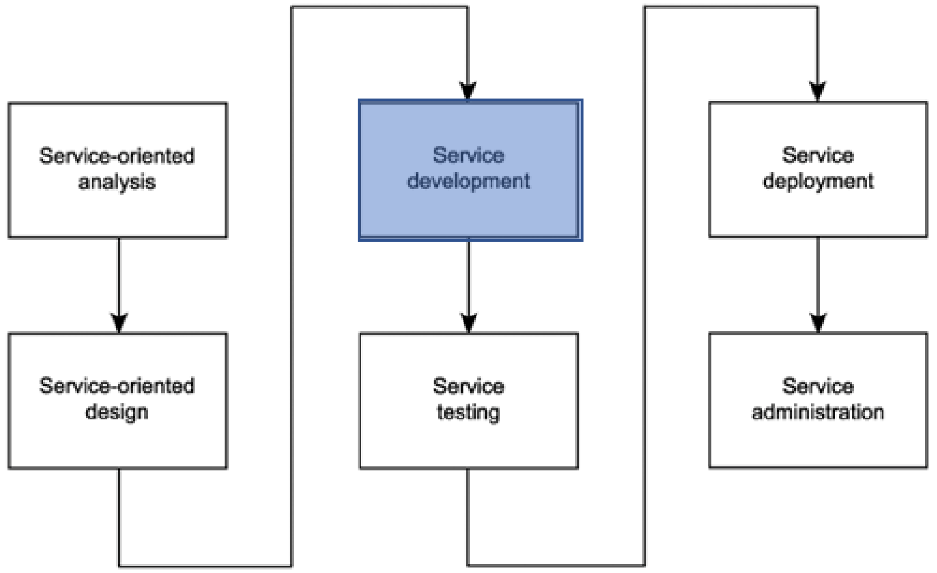
<service>: Where is the service located?

Service design

- What happens after service design, what is the next step in SOA delivery lifecycle?

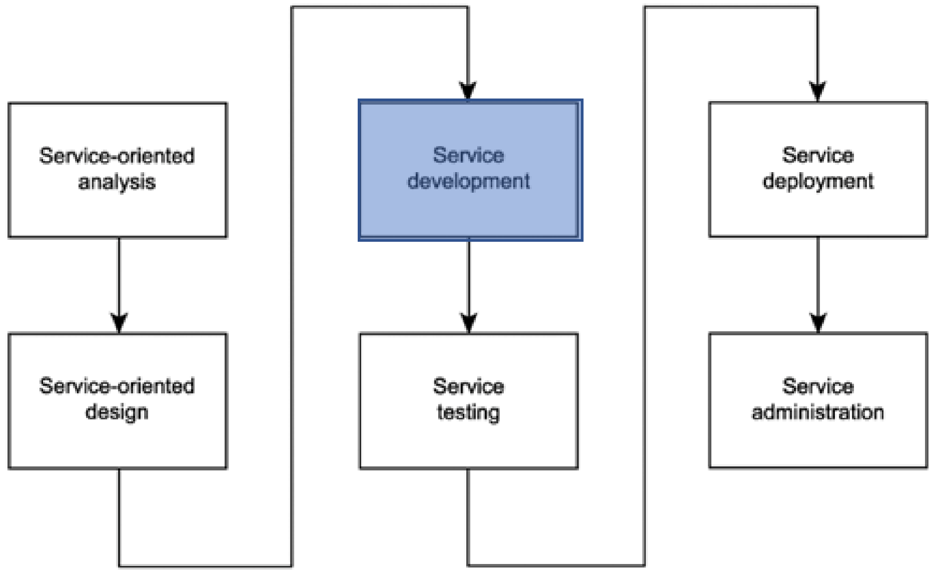
Service development

- Actual construction phase



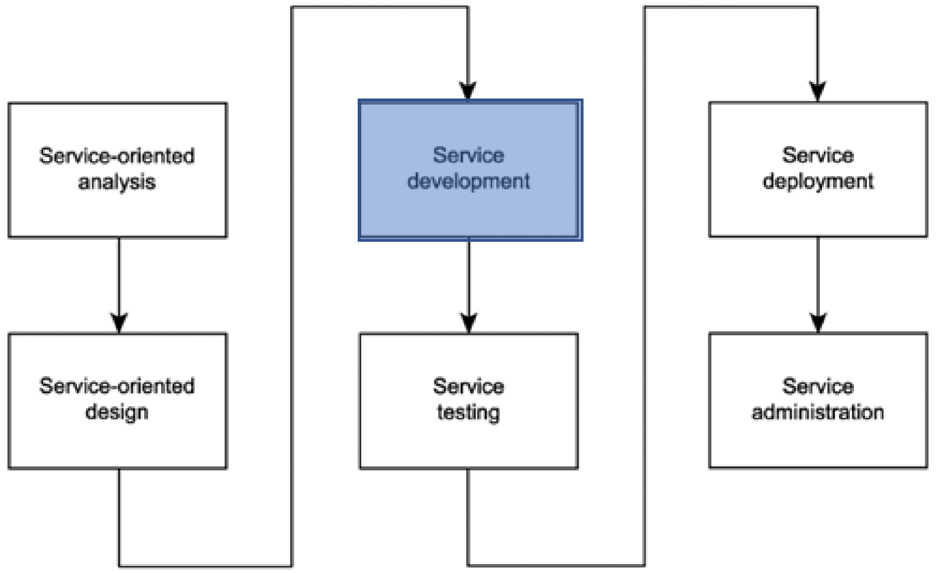
Service development

- Actual construction phase
- The choice of programming language



Service development

- Actual construction phase
- The choice of programming language
- The choice of development environment



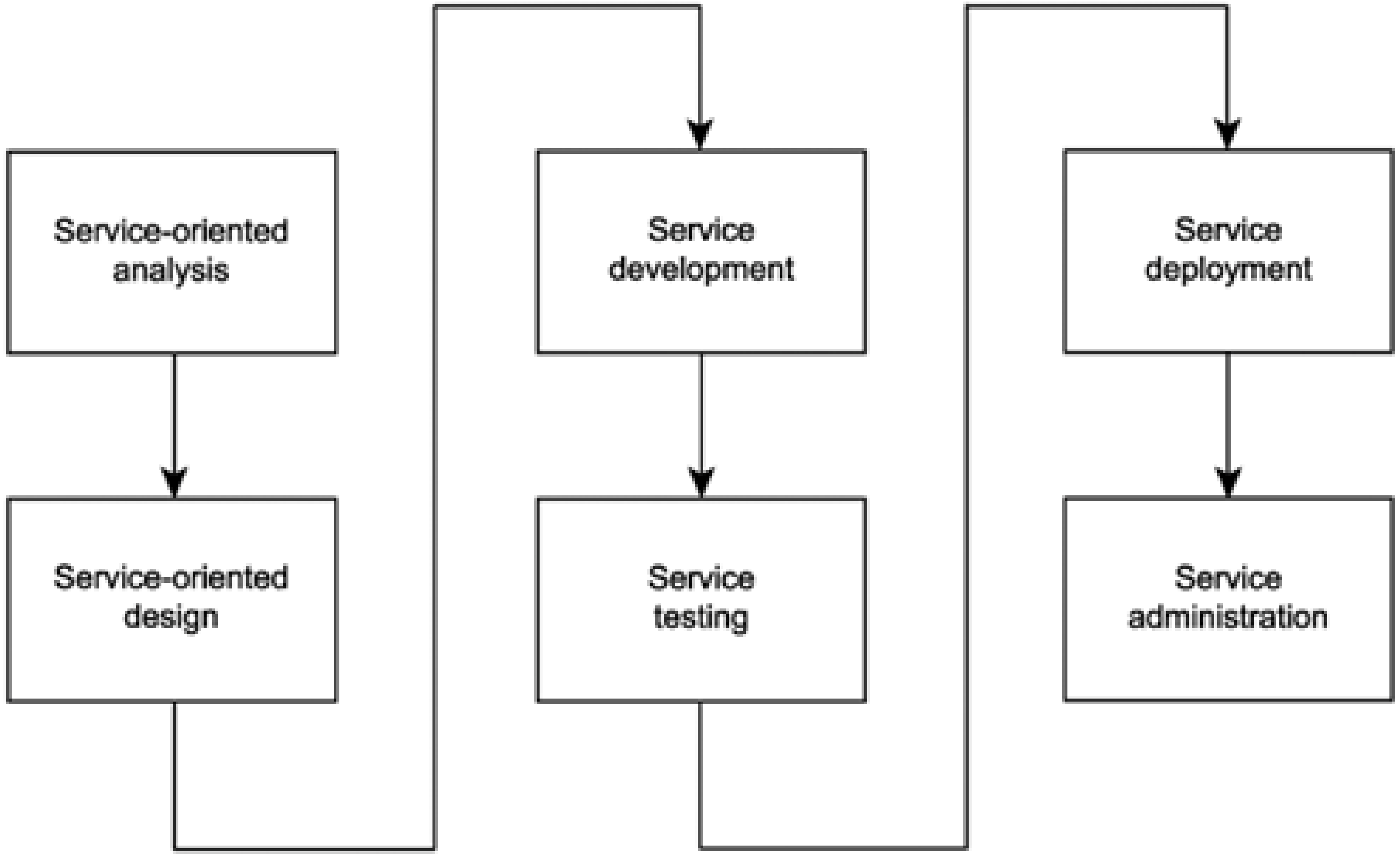
Open Question Points: 10 Setting

What are some examples of service development technologies?

Open Question is only supported on Version 2.0 or newer.

Answer

Summary


Service-oriented analysis
Service-oriented design
Service development
Service testing
Service deployment
Service administration

Module 6 Summary

- SOA lifecycle and SOA delivery strategies
- How to conduct service-oriented analysis
- How to design services