AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING

DEPARTMENT OF INFORMATION TECHNOLOGY

IT6801- SERVICE ORIENTED ARCHITECUTRE

UNIT -I INTRODUCTION TO XML

PART A

1. **“XML Namespaces provide a method to avoid element name conflicts” Elucidate. [Nov/Dec 2019]**
   When using prefixes in XML, a so-called namespace for the prefix must be defined. The namespace is defined by the xmlns attribute in the start tag of an element. The namespace declaration has the following syntax.
   ```
   xmlns:prefix="URI".
   <root> <h:table xmlns:h="http://www.w3.org/TR/html4/"> <h:tr>
   <h:td>Apples</h:td>
   <h:td>Bananas</h:td>
   </h:tr> </h:table>
   <f:table xmlns:f="http://www.w3schools.com/furniture"> <f:name>African
   Coffee Table</f:name> <f:width>80</f:width>
   <f:length>120</f:length> </f:table> </root>
   ```

2. **What is an XML Schema? [Nov/Dec 2019]**
   An XML Schema describes the structure of an XML document.
   The XML Schema language is also referred to as XML Schema Definition (XSD).

   The purpose of an XML Schema is to define the legal building blocks of an XML document:
   - the elements and attributes that can appear in a document
   - the number of (and order of) child elements
   - data types for elements and attributes
   - default and fixed values for elements and attributes

3. **How tags in XML are defined? Give Example. [Apr/May 2019]**
   XML tags are the important features of XML document. It is similar to HTML but XML is more flexible then HTML. It allows to create new tags (user defined tags). The first element of XML document is called root element. The simple XML documents contain opening tag and closing tag. The XML tags are case sensitive i.e. <root> and <Root> both tags are different. The XML tags are used to define the scope of elements in XML document.

   ```
       <root>
       <name>GeeksforGeeks</name>
       <address>
   <sector>142</sector>
   <location>Noida</location>
        </address>
       </root>
   ```

4. **What is XML Document Type Definition? Give Example[Apr/May 2019]**
   A document type definition (DTD) is a set of markup declarations that define a document type for a SGML-family markup language (GML, SGML, XML, HTML). ... XML uses a subset of

SGML DTD. As of 2009, newer XML namespace-aware schema languages (such as W3C XML Schema and ISO RELAX NG) have largely superseded DTDs.

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

5. **Define Service Oriented Architecture [Apr/May 2018]**
   Service oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
6. **State Service Component. [Apr/May 2018]**
   Service-component architecture (SCA) is a group of specifications intended for the development of applications based on service-oriented architecture (SOA), which defines how computing entities interact to perform work for each other.
7. **List out the Advantages of XML over SGML. [Nov/Dec 2018]**

   XML is international. **XML** is based on Unicode.

   XML can be structured. Using DTDs, **XML** can be structured so that both the content and syntax can be easily validated.

   XML documents can be built using composition.

   XML can be a data container.

   XML offers flexibility

8. **Identify some Xpointer functions with their purposes. [Nov/Dec 2018]**
   X pointer is same as X Path. But differs from X Path in the aspect as it describes a location on an external document

9. **What is XML? [Nov/Dec 2018, Apr/May 2017]**
   Extensible Markup Language (XML) is a markup language that define a set of rules for encoding documents in a format which is both human readable and machine readable. It is defined by the W3c's XML 1.0 specification and by several other related specifications all of which are free open standards.

10. **What is XML Document Prolog? [Nov/Dec 2017]**
    XML Prolog is the component added in the beginning of an XML document. Otherwise, we can say that whatever it appears before the document's root element can be considered as Prolog. XML Prolog includes XML declaration, DOCTYPE and comments, processing instructions too.

11. **Define attributes in XML**
    When you create XML Schemas, you define the individual elements and attributes and assign valid types to them. Elements describe data, whereas attributes are like properties of an element, in that

they provide further definition about the element the way that properties describe characteristics of objects and classes.

10. Define well-formed documents.

An XML document is well formed if it follows all the preceding syntax rules of XML. On the other hand, if it includes inappropriate markup or characters that cannot be processed by XML parsers, the document cannot be considered well formed. It goes without saying that an XML document can't be partially well formed.

11. **What are the DTD Drawbacks and Alternatives?**

There are several drawbacks that limit the ability of DTDs to meet these growing and changing validation needs.
First and foremost, DTDs are composed of non-XML syntax. Given that one of the central tenets of XML is that it be totally extensible, it may not seem to make a lot of sense that this is the case for DTDs.

Additionally, there can only be a single DTD per document. It is true that there can be internal and external subsets of DTDs, but there can only be a single DTD referenced. In the modern programming world, we are used to being able to draw the programming constructs we use from different modules or classes.

12. How to create XML Schemas?

Authoring an XML schema consists of declaring elements and attributes as well as the "properties" of those elements and attributes. We will begin our look at authoring XML schemas by working our way from the least-complex example to the most-complex example. Because attributes may not contain other attributes or elements, we will start there.

PART B & C

1. Write about X-Files. How to identify the valid documents. [Apr/May 2019]

X-Files

- XLINK
- XPATH
- XQuery

XLink

In HTML, we know (and all the browsers know!) that the <a> element defines a hyperlink.
However, this is not how it works with XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what hyperlink elements will be called in XML documents. The solution for creating links in XML documents was to put a marker on elements that should act as hyperlinks.

Example:
```
<?xml version="1.0"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
<homepage xlink:type="simple" xlink:href="http://www.w3schools.com">Visit
W3Schools</homepage><homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit
W3C</homepage>
  </homepages>
```

XPATH
XPath is syntax for defining parts of an XML document. XPath uses path expressions to navigate in XML documents. XPath contains a library of standard functions. XPath is a major element in XSLT. XPath is a W3C Standard.

XPath Terminology
Nodes:
In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processinginstruction, comment, and document (root) nodes. XML documents are treated as trees of nodes.
The root of the tree is called the document node (or root node).
Relationship of Nodes

- Parent
- Children
- Siblings
- Ancestors
- Descendants

XQuery

XQuery is the language for querying XML data. XQuery for XML is like SQL for databases. XQuery is built on XPath expressions. XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.). XQuery is a W3C Recommendation.

```
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
<author>Per Bothner</author>
<author>Kurt Cagle</author>
<author>James Linn</author>
<author>Vaidyanathan Nagarajan</author>
<year>2003</year>
<price>49.99</price>
</book>
- <book category="WEB">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

Functions
XQuery uses functions to extract data from XML documents. The doc() function is used to open the "books.xml" file:
doc("books.xml"), Path Expressions
XQuery uses path expressions to navigate through elements in an XML document.The following path expression is used to select all the title elements in the "books.xml" file:
doc("books.xml")/bookstore/book/title(/bookstore selects the bookstore element, /book selects all the book elements under the bookstore element, and /title selects all the title elements under each book element), The XQuery above will extract the following:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Predicates
XQuery uses predicates to limit the extracted data from XML documents. The following

predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30: doc("books.xml")/bookstore/book[price<30]The XQuery above will extract the following:

```
<book category="CHILDREN">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
  </book>
```

2. Explain XML Namespaces with an example. [Apr/May 2019, Nov/Dec 2018, Nov/Dec 2017]

A Namespace is a set of unique names. Namespace is a mechanism by which element and attribute name can be assigned to group. The Namespace is identified by URI (Uniform Resource Identifiers).

Namespace Declaration

A Namspace is declared using reserved attributes. Such an attribute name must either be xmlns or begin with xmlns: shown as below:

```
<element xmlns:name="URL">
```

Syntax
The Namespace starts with the keyword xmlns.
The word name is the Namespace prefix.
The URL is the Namespace identifier.
Example
Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of

XML Namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="www.tutorialspoint.com/profile">
<cont:name>Tanmay Patil</cont:name>
<cont:company>TutorialsPoint</cont:company>
<cont:phone>(011) 123-4567</cont:phone>

</cont:contact>
```

Here, the Namespace prefix is cont, and the Namespace identifier (URI) as www.tutorialspoint.com/profile. This means, the element names and attribute names with the cont prefix (including the contact element), all belong to namespace.
XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts
In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
This XML carries HTML table information:

```
<table>
<tr>
<td>Apples</td>
<td>Bananas</td>
</tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
<name>African Coffee Table</name>
<width>80</width>
```

```
<length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
A user or an XML application will not know how to handle these differences.
Solving the Name Conflict Using a Prefix
Name conflicts in XML can easily be avoided using a name prefix.
This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table>
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.
XML Namespaces - The xmlns Attribute
When using prefixes in XML, a namespace for the prefix must be defined.
The namespace can be defined by an xmlns attribute in the start tag of an element.
The namespace declaration has the following syntax. xmlns:prefix="URI".

The purpose of using an URI is to give the namespace a unique name.
However, companies often use the namespace as a pointer to a web page containing namespace information.
1.3.3 Uniform Resource Identifier (URI)
A Uniform Resource Identifier (URI) is a string of characters which identifies an Internet Resource.
The most common URI is the Uniform Resource Locator (URL) which identifies an Internet domain address. Another, not so common type of URI is the Universal Resource Name (URN).
1.3.4 Default Namespaces
Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:
xmlns="namespaceURI"
This XML carries HTML table information:

```
<table xmlns="http://www.w3.org/TR/html4/">
<tr>
<td>Apples</td>
<td>Bananas</td>
</tr>
</table>
```

This XML carries information about a piece of furniture:

```
<table xmlns="http://www.w3schools.com/furniture">
<name>African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>
```

3. Outline XML Schema with an example. [Nov/Dec 2019,Apr/May 2019, Nov/Dec 2018, Nov/Dec 2017]

An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema defines elements that can appear in a document, defines attributes that can appear in a document, defines which elements are child elements, defines the order of child elements, defines the number of child elements, defines whether an element is empty or can include text, defines data types for elements and attributes, defines default and fixed values for elements and attributes.
Two Types of Schemas:
Simple Type

Complex Type

Simple Type
A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

Rules for XML structure
All XML elements must have a closing tag. XML tags are case sensitive, All XML elements must have a proper nesting, All XML Documents must contain a single root element, Attribute values must be quoted, Attributes may only appear once in the same start tag, Attribute values cannot contain references to external entities, All entities except amp, lt, gt, apos, and quote must be declared before they are used.
XML Schema has a lot of built-in data types. The most common types are:
xs:string
xs:decimal
xs:integer
xs:Boolean
xs:date
xs:time

Example:
Here are some XML elements:
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
And here are the corresponding simple element definitions:
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>

Complex type:
A complex element is an XML element that contains other elements and/or attributes.
Look at this simple XML document called "note.xml":
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget to submit the assignment this monday!</body>
</note>
Example:
The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):
<!ELEMENT note (to, from, heading, body)><!ELEMENT to (#PCDATA)>

<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com" xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
<xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Structuring with schemas

A Simple XML Document:

We‘ll talk about a shirt. There‘s actually a lot we can talk about with regard to a shirt: size, color, fabric, price, brand, and condition, among other properties. The Following example shows one possi- ble XML rendition of a document describing a shirt. Of course, there are many other possible ways to describe a shirt, but this example provides a foundation for our further discussions.

```
<?xml version="1.0"?>
<shirt>
<model>Zippy Tee</model>
<brand>Tommy Hilbunger</brand>

<price currency="USD">14.99</price>
<on_sale/>
<fabric content="60%">cotton</fabric>
<fabric content="40%">polyester</fabric>
<options>
<colorOptions>
<color>red</color>
<color>white</color>
</colorOptions>
<sizeOptions>
<size>Medium</size>
<size>Large</size>
</sizeOptions>
</options>
<description> This is a <b>funky</b> Tee shirt similar to the Floppy Tee shirt </description>
</shirt>
```

4. Determine the rules of XML Document Structure. [Nov/Dec 2018, Apr/May 2017]

This page provides a description of XML structure including the document parts, the prologue, and provides a simple XML example document.
1.1.1 Document Parts
 Prolog
 Document Element (root element)

1.1.2 The Prologue

The prologue, equivalent to the header in HTML, may include the following:

 An XML declaration (optional) such as:

```
<?xml version="1.0"?>
```

 A DTD or reference to one (optional). An example reference to an external DTD file:

```
<!DOCTYPE LANGLIST SYSTEM "langlist.dtd">
```

Processing instructions - An example processing instruction that causes style to be determined by a style sheet:

```
<?xml-stylesheet type="text/css" href="xmlstyle.css"?>
```

An XML Document

Therefore a complete well formed XML document may look like:

```
<?xml version="1.0"?>
<LAND>
<FOREST>
<TREE>Oak</TREE>
<TREE>Pine</TREE>
<TREE>Maple</TREE>
</FOREST>
<MEADOW>
<GRASS>Bluegrass</GRASS>
<GRASS>Fescue</GRASS>
<GRASS>Rye</GRASS>
</LAND>
```

The LAND element, above, is the root element.

The below document is not an XML document since it does not qualify by the rules of a well-formed document. There is more than one top level element which disqualifies the document from being well formed.

```
<?xml version="1.0"?>
<FOREST>
<TREE>Oak</TREE>
<TREE>Pine</TREE>
<TREE>Maple</TREE>
</FOREST>
<MEADOW>
<GRASS>Bluegrass</GRASS>
<GRASS>Fescue</GRASS>
<GRASS>Rye</GRASS>
</MEADOW>
```

Defining Display

If the HTML document is not linked to a style sheet, the XML document will be displayed with tags included. The elements and tags may be color coded to aid in viewing the document. The document is displayed without tags according to the style sheet if a link to one is specified. The following document shows a document with a link to a cascading style sheet:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="xmlstyle.css"?>
<DATABASE>
<TITLE>List of Items Important to Markup Languages</TITLE>
<TITLE2>Languages</TITLE2>
<LANGUAGES>SGML<LANGUAGES>
<LANGUAGES>XML</LANGUAGES>
<LANGUAGES>HTML<LANGUAGES>
<TITLE2>Other</TITLE2>
<OTHER>DTD<OTHER>
<OTHER>DSSL<OTHER>
<OTHER>Style Sheets</OTHER>
```

```
</DATABASE>
```

The below line, which is a part of the XML document above, is a processing instruction and is a part of the prolog.

```
<?xml-stylesheet type="text/css" href="xmlstyle.css"?>
```

The style sheet, "xmlstyle.css", may look like:

```
DATABASE
{ display: block }
TITLE
{ display: block;
font-family: arial;

color: #008000;
font-weight: 600;
font-size: 22;
text-align: center }
TITLE2
{ display: block;
font-family: arial;
color: #000080;
font-weight: 400;
font-size: 20 }
LANGUAGES
{ display: block;
list-style-type: decimal;
font-family: arial;
color: #000000;
font-weight: 400;
font-size: 18 }
OTHER
{ display: block;
list-style-type: square;
font-family: arial;
color: #0000ff; font-weight: 200; font-size: 14 }
```

5. What is well-formed XML document structure? Create an XML document which contains student register number, name, date of birth, gender and programme admitted. Also define DTD and XSD for the created XML document. [Nov/Dec 2019]

The XML Recommendation states that an XML document has both logical and physical structure. Physically, it is comprised of storage units called entities, each of which may refer to other entities, similar to the way that include works in the C language. Logically, an XML document consists of declarations, elements, comments, character references, and processing instructions, collectively known as the markup.

An XML document consists of three parts, in the order given:

- An XML declaration (which is technically optional, but recommended in most normal cases)
- A document type declaration that refers to a DTD (which is optional, but required if you want validation)
- A body or document instance (which is required)

Collectively, the XML declaration and the document type declaration are called the XML prolog.

Student.xml

```
<?xml version="1.0"?>
<!DOCTYPE STUDENTS SYSTEM "E:\XML1\STUDENT.dtd">
```

```
<STUDENTS>
 <STUDENT>
 <STUDENTDATA>
    <NAME> SUTHA </NAME>
    <ID>  2007IT51 </ID>
    <AGE> 20 </AGE>
    <ADDRESS> 12,SATHY ROAD,GOBI </ADDRESS>
  </STUDENTDATA>
</STUDENT>
</STUDENTS>
```

Student.dtd:

```
<?xml version="1.0"?>
<!ELEMENT STUDENTS (STUDENT*)>
<!ELEMENT STUDENT (STUDENTDATA*)>
<!ELEMENT STUDENTDATA (NAME,ID,AGE,ADDRESS)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT AGE (#PCDATA)>
<!ELEMENT ADDRESS (#PCDATA)>
```

Student.xsd:

```
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   targetNamespace = "http://www.tutorialspoint.com"
   xmlns = "http://www.tutorialspoint.com"
   elementFormDefault = "qualified">

   <xs:element name = 'class'>
      <xs:complexType>
         <xs:sequence>
            <xs:element name = 'student' type = 'StudentType' minOccurs = '0'
               maxOccurs = 'unbounded' />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:complexType name = "StudentType">
      <xs:sequence>
         <xs:element name = "name" type = "xs:string"/>
         <xs:element name = "id" type = "xs:positiveInteger"/>
         <xs:element name = "address" type = "xs:string"/>
         <xs:element name = "age" type = "xs:positiveInteger"/>
      </xs:sequence>
      <xs:attribute name = 'rollno' type = 'xs:positiveInteger'/>
   </xs:complexType>
   </xs:schema>
```

6. Elaborate how namespaces and DTDs relate to one another with an example. [Nov/Dec 2019]

Namespaces are completely independent of DTDs and can be used in both valid and invalid documents. A document can have a DTD but not use namespaces or use namespaces but not have a DTD. It can use both namespaces and DTDs or neither namespaces nor DTDs. Namespaces do not in any way change DTD syntax nor do they change the definition of validity. For instance, the DTD of a valid document that uses an element named dc:title must include an ELEMENT declaration properly specifying the content of the dc:title element.

The name of the element in the document must exactly match the name of the element in the DTD including the prefix. The DTD cannot omit the prefix and simply declare a title element. The same is true of prefixed attributes. For instance, if an element used in the document has xlink:type and xlink:href attributes, then the DTD must declare the xlink:type and xlink:href attributes, not simply type and href.

Conversely, if an element uses an xmlns attribute to set the default namespace and does not attach prefixes to elements, then the names of the elements must be declared without prefixes in the DTD. The validator neither knows nor cares about the existence of namespaces. All it sees is that some element and attribute names happen to contain colons; as far as it's concerned, such names are perfectly valid as long as they're declared.

Parameter Entity References for Namespace Prefixes

Requiring DTDs to declare the prefixed names instead of the raw names or some combination of local part and namespace URI makes it difficult to change the prefix in valid documents. The problem is that changing the prefix requires changing all declarations that use that prefix in the DTD as well. However, with a little forethought, parameter entity references can alleviate the pain quite a bit.

The trick is to define as parameter entities both the namespace prefix and the colon that separates the prefix from the local name

The second step is to define the qualified names as more parameter entity references

UNIT II- BUILDING XML-BASED APPLICATIONS

PART A

1. **Present an Outline of SAX. [Nov/Dec 2019]**
SAX (Simple API for XML) is an event-driven online algorithm for parsing XML documents, with an API developed by the XML-DEV mailing list.[1] SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole—building the full abstract syntax tree of an XML document for convenience of the user—SAX parsers operate on each piece of the XML document sequentially, issuing parsing events while making a single pass through the input stream.

2. **Outline how to transform XML documents to other types with an example. [Nov/Dec 2019]**

XSL Transformations (XSLT 2.0) is a language for transforming XML documents into other XML documents, text documents or HTML documents. You might want to format a chapter of a book using XSL-FO, or you might want to take a database query and format it as HTML.

With XSLT 2.0, processors can operate not only on XML but on anything that can be made to look like XML: relational database tables, geographical information systems, file systems, anything from which your XSLT processor can build an XDM instance. In some cases an XSLT 2.0 processor might also be able to work directly from a database of XDM instances. This ability to operate on multiple input files in multiple formats, and to treat them all as if they were XML files, is very powerful. It is shared with XQuery, and with anything else using XPath 2.0

3. **Outline the functions performed by an XML Parser. [Apr/May 2019]**
An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML. XML parser validates the document and check that the document is well formatted.

4. **What is XSL? [Apr/May 2019]**
XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts:
XSLT, XPath, XSL-FO

5. **What is DOM? List some interfaces. [Nov/Dec 2017]**
The Document Object Model (DOM) provides a way of representing an XML document in memory so that it can be manipulated by your software. DOM is a standard application programming interface (API) that makes it easy for programmers to access elements and delete, add, or edit content and attributes. DOM was proposed by the World Wide Web Consortium (W3C) in August of 1997 in the User Interface Domain. The main reason for using DOM is to create or modify an XML document programmatically.
The DOM interfaces are Document, Node, NodeList, Element, Attr, CharacterData, Text, Comment, Processing Instruction and CDATA.

6. **List the Disadvantages of SAX [Nov/Dec 2017]**

- SAX is similar to a one-pass compiler. After it reads part of the document, it cannot navigate backward to reread the data it has processed, unless you start all over again.
- Because SAX does not store the data that it has processed, you cannot modify this data and store it back in the original document.
- Because SAX does not create an in-memory document structure, you cannot build an XML document by using a SAX parser.

7. **What is meant by XSL Formatting? [Apr/May 2017]**

   Extensible Style sheet Language (XSL) provides facilities to access and manipulate the data in XML documents.XSL is itself an XML dialect and provides two distinct and useful mechanisms for handling and manipulating XML documents. One is concerned with formatting data and other is concerned with data transformation.

   When XSL is used as a formatting language, the style sheets consist of formatting objects that prepare an XML document for presentation, usually in a browser.

8. **Define SOAP Message. [Apr/May 2018]**

   A SOAP message is an ordinary XML document containing the following elements − Envelope − Defines the start and the end of the message. It is a mandatory element. Header − Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point.

9. **List the disadvantages of SOAP [Nov/Dec 2018]**

   - Only XML can be used, JSON and other lightweight formats are not supported.
   - SOAP is based on the contract, so there is a tight coupling between client and server applications.
   - SOAP is slow because payload is large for a simple string message, since it uses XML format.
   - Anytime there is change in the server side contract, client stub classes need to be generated again.
   - Can't be tested easily in browser

10. **Mention the advanced features of XSLT [Nov/Dec 2018]**

    This supplementary XSLT tutorial concentrates on four topics: variables, keys, conditionals (<xsl:if> and <xsl:choose>), and the difference between push processing (<xsl:apply-templates> and <xsl:template>) and pull processing (<xsl:for-each> and <xsl:value-of>).

11. **List the Disadvantages of DOM. [Nov/Dec 2018]**

    It consumes more memory

    Its Operational Speed is lower

    Stores the entire document in memory

    Doesn't follow java programming conventions

12. What are complex types?

Complex types are important aspects of xml schema that allow application developers to define application-specific data types that can be checked by programs that check XML document for validity.

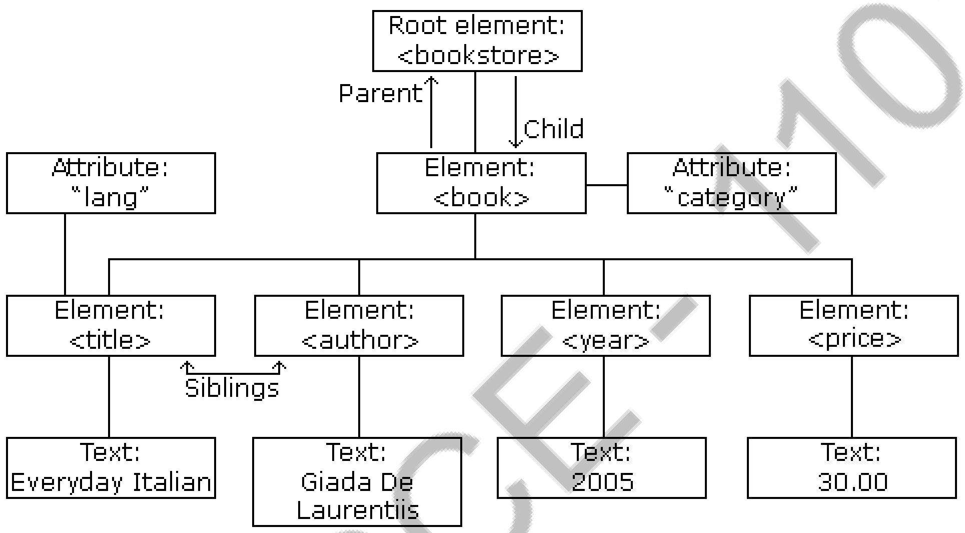
XML schema divides complex types into two categories: those with simple content & those with complex content.

13. What are all the Transformation techniques? [Nov/Dec 2016]
XSL - it is an XML- based languages used to transform XML documents into others format such as HTML for web display.
XLINK - highlighting that element or taking the user directly to that point in the document.

XPATH - Xpath gets its name from its use of a path notation to navigate through the hierarchical tree structure of an XML document XQUERY - it is w3c initiative to define a standard set of constructs for querying & searching XML document.

PART B & C

1. **Outline XML document object model with an example. [Nov/Dec 2019, Apr/May 2019]**
"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."



The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.

The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

All HTML elements can be accessed through the HTML DOM.

This example changes the value of an HTML element with id="demo":

```
<h1 id="demo">This is a Heading</h1>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

The XML DOM
All XML elements can be accessed through the XML DOM.

The XML DOM is:

A standard object model for XML
A standard programming interface for XML

Platform- and language-independent
A W3C standard
In other words: The XML DOM is a standard for how to get, change, add, or delete XML elements.

Get the Value of an XML Element
This code retrieves the text value of the first <title> element in an XML document:

Example
```
txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

he XML file used in the examples below is books.xml.

This example reads "books.xml" into xmlDoc and retrieves the text value of the first <title> element in books.xml:

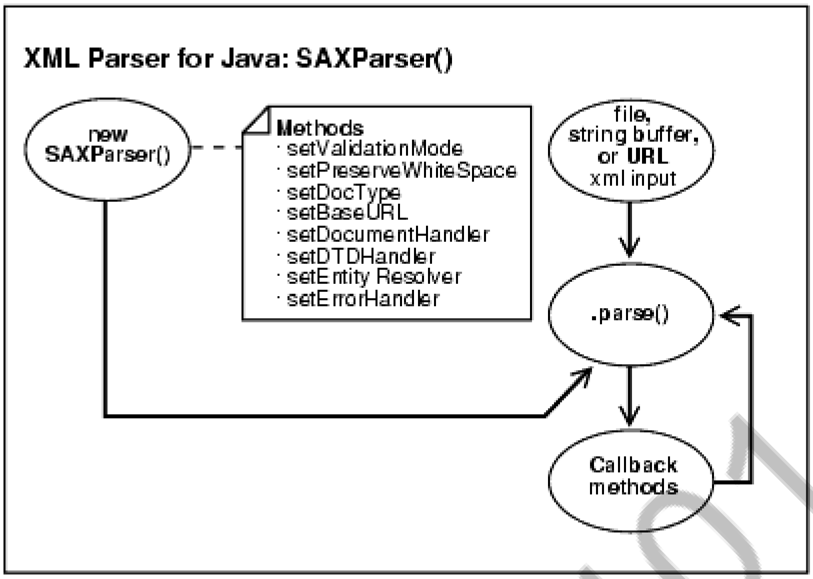
Example
```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
   myFunction(this);
   }
};
xhttp.open("GET", "books.xml", true);
xhttp.send();

function myFunction(xml) {
   var xmlDoc = xml.responseXML;
   document.getElementById("demo").innerHTML =
   xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
</script>

</body>
</html>
```

2. **Outline the working of SAX Parser. [Apr/May 2019, Nov/Dec 2018, Nov/Dec 2017]**
SAX (Simple API for XML)
A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser
It does not create any internal structure.
Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.
It is an event based parser, it works like an event handler in Java.

Advantages
1) It is simple and memory efficient.
2) It is very fast and works for huge documents.

Disadvantages
1) It is event-based so its API is less intuitive.
2) Clients never know the full information because the data is broken into pieces.



3. Outline the modeling databases in XML with an example. [Apr/May 2019, Apr/May 2017]
XML DATABASES
XML stands for EXtensible Markup Language
XML was designed to describe data.
XML tags are not predefined unlike HTML
XML DTD and XML Schema define rules to describe data
XML example of semi structured data

The database
We can model the database with a document node and its associated element node:

```
<?xml version="1.0" ?>
<myDatabase>
table1
table2
...
tablen
</myDatabase>
```

Order of tables is immaterial
The table
Each table of the database is represented by an element node with the records as its children:

```
<customer>
record1
record2
...
recordm
</customer>
```

Again, order of the records is immaterial, since the relational data model defines no ordering on them.
The record
A record is also represented by an element node, with its fields as children:

```
<custRec>
```

```
field1
field2
...
fieldm
</custRec>
```
The name is arbitrary, since the relational data model doesn't define a name for a record type

The field
A field is represented as an element node with a data node as its only child:
```
<custName type="t">
d
</custName>
```
If d is omitted, it means the value of the fields is the empty string.
The value of t indicates the type of the value.

4. Outline the use of XSLT for Document publishing. Illustrate the process for converting XML Document to HTML Document. [Nov/Dec 2018, Apr/May 2017]

XSLT stands for Extensible Stylesheet Language Transformation.

XSLT is used to transform XML document from one form to another form.
XSLT uses Xpath to perform matching of nodes to perform these transformation .
The result of applying XSLT to XML document could be an another XML document, HTML, text or any another document from technology prespective.
The XSL code is written within the XML document with the extension of (.xsl).
In other words, an XSLT document is a different kind of XML document.

XML Namespace: XML Namespaces are the unique names .

XML Namespace is a mechanism by which element or attribute is assigned to a group.
XML Namespace is used to avoid the name conflicts in the XML document.
XML Namespace is recommended by W3C.

XML Namespace Declaration:
It is declared using reserved attribute such as the attribute is xmlns or it can begin with xmlns:

Syntax:
```
 <element xmlns:name = "URL">
```

Where,

Namespace starts with the xmlns.
The word name is the namespace prefix.
the URL is the namespace identifier.

Example:
Consider the following xml document named Table.xml :-
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="rule.css"?>
<tables>
<table>
<tr>
          <td>Apple</td>
          <td>Banana</td>
</tr>
```

```
</table>
<table>
<height>100</height>
<width>150</width>
</table>
</tables>
```

In the above code, there would be a name conflict, both of them contains the same table element but the contents of the table element is different .To handle this situation, the concept of XML Namespace is used.

Example:
Consider the same XML document to resolve name conflict:
filter_none
brightness_4

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="rule.css"?>
 <tables>
  <m:table xmlns:m=""http://www.google.co.in">
   <m:tr>
    <m:td>Apple</m:td>
    <m:td>Banana</m:td>
   </m:tr>
  </m:table>
  <n:table xmlns:m=""http://www.yahoo.co.in">
   <n:height>100</n:height>
   <n:width>150</n:width>
  </n:table>
 </tables>
```

Xpath:

Xpath is an important component of XSLT standard.
Xpath is used to traverse the element and attributes of an XML document.
Xpath uses different types of expression to retrieve relevant information from the XML document.
Xpath contains a library of standard functions.

Example:
bookstore/book[1] => Fetches details of first child of bookstore element.
bookstore/book[last()] => Fetches details of last child of bookstore element.
Templates:

An XSL stylesheet contains one or more set of rules that are called templates.
A template contains rules that are applied when the specific element is matched.
An XSLT document has the following things:
The root element of the stylesheet.
A file of extension .xsl .
The syntax of XSLT i.e what is allowed and what is not allowed.
The standard namespace whose URL is http://www.w3.org/1999/XSL/Transform.
Example:
In this example, creating the XML file that contains the information about five students and displaying the XML file using XSLT.

XML file:
Creating Students.xml as:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?xml-stylesheet type="text/xsl "href="Rule.xsl" ?>
<student>
<s>
<name> Divyank Singh Sikarwar </name>
<branch> CSE</branch>
<age>18</age>
<city> Agra </city>
</s>
<s>
<name> Aniket Chauhan </name>
<branch> CSE</branch>
<age> 20</age>
<city> Shahjahanpur </city>
</s>
<s>
<name> Simran Agarwal</name>
<branch> CSE</branch>
<age> 23</age>
<city> Buland Shar</city>
</s>
<s>
<name> Abhay Chauhan</name>
<branch> CSE</branch>
<age> 17</age>
<city> Shahjahanpur</city>
</s>
<s>
<name> Himanshu Bhatia</name>
<branch> IT</branch>
<age> 25</age>
<city> Indore</city>
</s>
</student>
```

In the above example, Students.xml is created and linking it with Rule.xsl which contains the corresponding XSL style sheet rules.

XSLT Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h1 align="center">Students' Basic Details</h1>
<table border="3" align="center" >
<tr>
        <th>Name</th>
        <th>Branch</th>
        <th>Age</th>
        <th>City</th>
</tr>
        <xsl:for-each select="student/s">
<tr>
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="branch"/></td>
        <td><xsl:value-of select="age"/></td>
```

```
            <td><xsl:value-of select="city"/></td>
</tr>
            </xsl:for-each>
            </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

5. **Examine DOM Parsing for a "Purchase Order" xml file that contains bill to and send to warehouse details provided as XML Schema with DocBuilder Output. [Nov/Dec 2017]**

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
```

```
        <ShipDate>1999-05-21</ShipDate>
       </Item>
      </Items>
     </PurchaseOrder>
```

6. Explain the XML Parsing using DOM Parser with suitable illustrations. [Nov/Dec 2019]

The Document Object Model (DOM) provides a way of representing an XML document in main memory as tree structure, so that it can be manipulated by your software.

- DOM is a standard application programming interface (API) that makes it easy for programmers to access elements and delete, add, or edit content and attributes.

DOM was proposed by the World Wide Web Consortium (W3C) in August of 1997
- DOM by itself is just a specification for a set of interfaces defined by W3C
- The DOM interfaces are defined independent of any particular programming language. You can write DOM code in just about any programming language, such as Java, ECMAScript (a standardized version of JavaScript/JScript), or C++

- There are DOM APIs for each of these languages. W3C uses the Object Management Group's (OMG) Interface Definition Language (IDL) to define DOM in a language- neutral way. Languagespecific bindings, or DOM interfaces, exist for these languages. The DOM specification itself includes bindings for Java and ECMAScript

DOM Levels
- The DOM working group works on phases (or levels) of the specification. There are three levels are in the works.
- The DOM Level 1 and Level 2 specifications are W3C recommendations. The specification of level 1and 2 are final
Level 1:
- Level 1 allows traversal of an XML document as well as the manipulation of the content in that document
Level2:
- Level 2 extends Level 1 with additional features such as namespace support, events, ranges, and so on
Level 3:
- It is currently a working draft. This means that it is under active development and subject to change.

Common DOM methods
When you are working with the DOM, there are several methods you'll use often:

- Document.getDocumentElement() - Returns the root element of the document.
- Node.getFirstChild() - Returns the first child of a given Node.
- Node.getLastChild() - Returns the last child of a given Node.
- Node.getNextSibling() - These methods return the next sibling of a given Node.
- Node.getPreviousSibling() - These methods return the previous sibling of a given Node.
- Node.getAttribute(attrName) - For a given Node, returns the attribute with the requested name

Steps to be followed when Using DOM
Following are the steps used while parsing a document using DOM Parser.

- Import XML-related packages.
- Create a DocumentBuilder
- Create a Document from a file or stream
- Extract the root element
- Examine attributes
- Examine sub-elements

UNIT -III SERVICE ORIENTED ARCHITECTURE

PART A

1. **What are loosely coupled systems in SOA? [Nov/Dec 2019]**

   The concept of loose coupling within SOA is directly influenced by the object-oriented design paradigm, whereby the objective is to reduce coupling between classes in order to foster an environment where both the classes, although somehow related to each other, can be changed in a manner that such a change does not break the existing relationship, which is necessary for the working of a software program.

2. **Outline the steps in creating XML databases with an example. [Nov/Dec 2019]**

   Step 1: Create a console application.
   Step 2: Declare the connection string :
   string connectionString ="Data Source=your datasource;Initial Catalog=catalog name;uid=userid;pwd=password";
   SqlConnection mySqlConnection =new SqlConnection(connectionString);
   Step 3: Write the query :
   String selectString ="SELECT ACCOUNT_BUSINESS_UNIT,ACCOUNT_NAME,GENERAL_MANAGER,OFFSHORE_OPERATIONS_MANAGER FROM [Database].[Table] GROUP BY ACCOUNT_BUSINESS_UNIT,ACCOUNT_NAME,GENERAL_MANAGER,OFFSHORE_OPERATIONS_MANAGER";

   Step 4: Create an arbitrary first value for the XDocument class. This is an example of using XElement :

   ```
   XElement newContent = new XElement("AccountInformations",
                             new XElement("Account",
                                 new XElement("AccountBusinessUnit", "A"),
                                 new XElement("AccountName", "B"),
                                 new XElement("GeneralManager", "C"),
                                 new XElement("OffshoreOperationsManager", "D")));
   ```

3. **What is distributed system? [Apr/May 2019]**

   SOA is a distributed system: meaning it invokes services located across platforms, enterprises (from architecture and implemented Infrastructure point of view) or geographical locations. Example: A new application can be developed using existing, remote services that may be provided by other applications.

4. **List out the Characteristics of SOA [Apr/May 2017, Nov/Dec 2017]**

   Some of the characteristics of contemporary SOA are:-
   i. Contemporary SOA is at the core of the service oriented platform.
   ii. Contemporary SOA increases quality of service.
   iii. Contemporary SOA is fundamentally autonomous.
   iv. Contemporary SOA is based on open standards.
   v. Contemporary SOA supports vendor diversity.
   vi. Contemporary SOA fosters intrinsic interoperability.
   vii. Contemporary SOA promotes discovery.
   viii. Contemporary SOA promotes federation.
   ix. Contemporary SOA promotes architectural composability.
   x. Contemporary SOA fosters inherent reusability

5. **State the Characteristics of Orchestration Service Layer. [Apr/May 2018]**

The orchestration service layer introduces a parent level of abstraction that alleviates the need for other services to manage interaction details required to ensure that service operations are executed in a specific sequence (Figure 9.5). Within the orchestration service layer, process services compose other services that provide specific sets of functions, independent of the business rules and scenario-specific logic required to execute a process instance.

6. What is Fundamental Part of SOA Framework? [Nov/Dec 2017]

- Enterprise service bus (ESB)
- Service registry
- Business processes
- MDM hub
- Data management

7. What are the benefits of SOA? [Apr/May 2017]

Service-oriented architecture (SOA) enables increased business agility, improved business workflows, extensible architecture, enhanced reuse, and a longer life span of applications. Adopting Service Oriented Architecture realize many benefits

Reusability

Rich Testability

Parallel Development

Higher Availability and Better Scalability

8. Write the IT Centric entry points. [Apr/May 2018]

IT-centric: Lay the technical groundwork for integration by leveraging the entry points of connectivity and reuse.

9. What is responsibility of a service? [Apr/May 2018]

These components are **responsible** for realizing the functionality of **services**. The middle layer is the **Service** Layer, which is where exposed **services** (both individual and composite **services**) carrying out business functions reside.

10. Give some common principles of service orientation. [Nov/Dec 2018]

- Standardized service contract
- Loose coupling.
- Abstraction
- Reusability
- Autonomy
- Discoverability
- Composability.

11. Define Service Oriented Architecture. [Apr/May 2019, Nov/Dec 2018, Apr/May 2018]

SOA represents an open, live, extensible, federated (joined together), composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, potentially reusable services, and implemented as Web services.

10. State Service Component. [Apr/May 2018]

This is the true heart of the SOA. The Service Component is that logical unit of code which implements the functionality to support the Service. The Service Component exposes one or more Services. A Service Component is also usually associated with a data store of some kind. This can contain data about a fundamental data type, control data, process, data, etc depending on the nature of the particular service component.

PART B & C

1. What are the components of SOA? How Components in SOA inter-relate? Give Example [Apr/May 2019]

Orchestration or Process Layer.

Access Framework.

Business Activity Monitoring.

Operational Data Store.

Business Intelligence.

Security.

Management.

2. What is Service Orientation? Outline the common Principles of Service orientation. [Nov/Dec 2019, Apr/May 2019, Apr/May 2018]

Common principles of service-orientation

• A common set of principles most associated with service-orientation.

– Services are reusable: services are designed to support potential reuse.

– Services share a formal contract for services to interact, they need not share

anything but a formal contract that describes each service and defines the terms

of information exchange.

– Services are loosely coupled Services must be designed to interact without the

need for tight, cross-service dependencies.

– Services abstract underlying logic The only part of a service that is visible to

the outside world is what is exposed via the service contract. Underlying logic,

beyond what is expressed in the descriptions that comprise the contract, is

invisible.

– Services are composable Services may compose other services. This promotes reusability and the creation of abstraction layers

– Services are autonomous The logic governed by a service resides within an

explicit boundary. The service has control within this boundary and is not

dependent on other services.

– Services are stateless Services should not be required to manage state

information.

– Services are discoverable Services should allow their descriptions to be

discovered and understood by humans and service requestors that may be able

to make use of their logic.

Services are composable

– A service can represent any range of logic from any types of sources,

including other services

Services are autonomous

– Autonomy requires that the range of logic exposed by a service exist within an

explicit boundary. This allows the service to execute self-governance of all its

processing.

– It also eliminates dependencies on other services

3. List out the Primary Characteristics of SOA. [Nov/Dec 2018, Apr/May 2018, Apr/May 2017]

The following are the Characteristics of SOA:

Contemporary SOA is at the core of the service-oriented computing platform

SOA is used to qualify products, designs, and technologies

an application computing platform consisting of Web services technology and service orientation principles

Contemporary SOA represents an architecture that promotes service-orientation through the use

of Web services.

Contemporary SOA increases quality of service

The ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services.

Allowing tasks to be carried out reliably so that message delivery or notification of failed delivery can be guaranteed.

Performance requirements to ensure that the overhead imposed by SOAP message and XML content processing does not inhibit the execution of a task.

Transactional capabilities to protect the integrity of specific business tasks with a guarantee that should the task fail, exception logic is executed.

Contemporary SOA is fundamentally autonomous

The service-orientation principle of autonomy requires that individual services be as independent and self-contained as possible wiith respect to the control they maintain over their underlying logic.

Contemporary SOA is based on open standards

Perhaps the most significant characteristic of Web services is the fact that data exchange is governed by open standards. After a message is sent from one Web service to another it travels via a set of protocols that is globally standardized and accepted.

Contemporary SOA supports vendor diversity

Organizations can certainly continue building solutions with existing development tools and server products. In fact, it may make sense to do so, only to continue leveraging the skill sets of in-house resources. However, the choice to explore the offerings of new vendors is always there. This option is made possible by the open technology provided by the Web services framework and is made more attainable through the standardization and principles introduced by SOA.

Contemporary SOA promotes discovery

SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond. A serious SOA will likely rely on some form of service registry or directory to manage service descriptions

Contemporary SOA fosters intrinsic interoperability

Further leveraging and supporting the required usage of open standards, a vendor diverse environment, and the availability of a discovery mechanism, is the concept of intrinsic interoperability. Regardless of whether an application actually has immediate integration requirements, design principles can be applied to outfit services with characteristics that naturally promote interoperability

Contemporary SOA promotes architectural composability

Composability is a deep-rooted characteristic of SOA that can be realized on different levels. For example, by fostering the development and evolution of composable services, SOA supports the automation of flexible and highly adaptive business processes.

Contemporary SOA emphasizes extensibility

Extensibility is also a characteristic that is promoted throughout SOA as a whole. Extending entire solutions can be accomplished by adding services or by merging with other serviceoriented applications (which also, effectively, "adds services"). Because the loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.

Contemporary SOA implements layers of abstraction

One of the characteristics that tends to evolve naturally through the application of serviceoriented design principles is that of abstraction. Typical SOAs can introduce layers of abstraction by positioning services as the sole access points to a variety of resources and processing logic.

Contemporary SOA is a building block

Service-oriented application architecture will likely be one of several within an organization committed to SOA as the standard architectural platform. Organizations standardizing on SOA work toward an ideal known as the service-oriented enterprise (SOE), where all business processes are composed of and exist as services, both logically and physically

Contemporary SOA is an evolution

SOA defines an architecture that is related to but still distinct from its predecessors. It differs from traditional client-server and distributed environments in that it is heavily influenced by the concepts and principles associated with service-orientation and Web services. It is similar to previous platforms in that it preserves the successful characteristics of its predecessors and builds upon them with distinct design patterns and a new technology set.

Contemporary SOA is still maturing

While the characteristics described so far are fundamental to contemporary SOA, this point is obviously more of a subjective statement of where SOA is at the moment. Even though SOA is being positioned as the next standard application computing platform, this transition is not yet complete. Despite the fact that Web services are being used to implement a great deal of application functionality, the support for a number of features necessary for enterprise-level computing is not yet fully available.

Contemporary SOA is an achievable ideal

A standardized enterprise-wide adoption of SOA is a state to which many organizations would like to fast-forward. The reality is that the process of transitioning to this state demands an enormous amount of effort, discipline, and, depending on the size of the organization, a good amount of time. Every technical environment will undergo changes during such a migration, and various parts of SOA will be phased in at different stages and to varying extents. This will likely result in countless hybrid architectures, consisting mostly of distributed environments that are part legacy and part service-oriented.

4. Summarize the tangible benefits of SOA. [Nov/Dec 2019, Nov/Dec 2018]

Separating concrete characteristics- Benefits of SOA

• SOA is at the core of the service-oriented computing platform.

• SOA is a building block.

• SOA is an evolution.

• SOA is still maturing.

• SOA is an achievable ideal.

By trimming (decoration) these items, along with some extra wording, we end up with the following set of concrete characteristics:

• Based on open standards

• Architecturally composable

• Capable of improving QoS SOA supports:

• Vendor diversity

• Intrinsic interoperability

• Discoverability

• Federation

• Inherent reusability

• Extensibility

• Service-oriented business modeling

• Layers of abstraction

• Enterprise-wide loose coupling

• Organizational agility (liveliness).

5. Give a note on three primary service layers for SOA. [Nov/Dec 2018, Nov/Dec 2017]

Application Layer Guidelines

When providing application functionality through services, it is important to separate the service functionality into a separate service layer. This chapter will help you to understand how the service layer fits into the application architecture, and learn the steps for designing the service layer. This includes guidance on the common issues you face when designing the service layer, and the key patterns and technology considerations for the service layer.

Within the service layer, you define and implement the service interface and the data contracts (or message types). One of the more important concepts to keep in mind is that a service should never expose details of the internal processes or the business entities used within the application. In particular, you should ensure that your business layer entities do not unduly influence your data contracts. The service layer should provide translator components that translate data formats between the business layer entities and the data contracts.

Business Layer Guidelines Overview

This chapter describes the key guidelines for designing the business layer of an application. It will help you to understand how the business layer fits into the typical layered application architecture, the components it usually contains, and the key issues you face when designing the business layer. You will see guidelines for design, the recommended design steps, relevant design patterns, and technology options. Figure 1 show how the business layer fits into typical application architecture.

Orchestration service layer

- When orchestration is incorporated as part of a service-oriented architecture, it assumes the role of the process.
- Orchestration allows a direct link to process logic to service interaction within workflow logic.
- It combines business modeling with service-oriented modeling by means of business process and service layer position.
- A master composition controller orchestration language (WS-BPEL) realizes workflow management through a process service model.
- The orchestration layer introduces a parent level of abstraction that improves the need for other services.

- Within the orchestration layer, process services compose other services that provide specific sets of functions, independent of the business rules and scenario-specific logic required to execute a process instance.
- They are required to compose other services to execute business process logic
- Process services can also be referred to as orchestrated task services parent business process layer.

6. The web service architecture is based upon the interactions between three roles. Outline the roles and operations involved in the interactions. [Nov/Dec 2019]

Components

The Service: Whereas a web service is an interface described by a service description, its implementation is the service. A service is a software module deployed on network accessible platforms provided by the service provider. It exists to be invoked by or to interact with a service requestor. It may also function as a requestor, using other web services in its implementation.

The Service Description: The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information, and network location. It could also include categorization and other meta data to facilitate discovery and utilization by requestors. The complete description may be realized as a set of XML description documents. The service description may be published to a request or directly or to a discovery agency.

Roles

Service Provider: From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that hosts access to the service. It has also been referred to as a service execution environment or a service container. Its role in the client-server message exchange patterns is that of a server.

Service Requestor: From a business perspective, this is the business that requires certain function to be satisfied. From an architectural perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The requestor role can be played by a browser driven by a person or a program without a user interface, e.g. another web service. Its role in the client-server message exchange patters is that of a client.

Discovery Agency: This is a searchable set of service descriptions where service providers publish their service descriptions. The service discovery agency can be centralized or distributed. A discovery agency can support both the pattern where it has descriptions sent to it and where the agency actively inspects public providers for descriptions. Service requestors may find services and obtain binding information (in the service descriptions) during development for static binding, or during execution for dynamic binding. For statically bound service requestors, the service discovery agent is in fact an optional role in the architecture, as a service provider can send the description directly to service requestors. Likewise, service requestors can obtain a service description from other sources besides a service registry, such as a local file system, FTP site, URL, or WSIL document.

Operations

In order for an application to take advantage of Web services, three behaviors must take place: publication of service descriptions, finding and retrieval of service descriptions, and binding or invoking of services based on the service description. These behaviors can occur singly or iteratively, with any cardinality between the roles. In detail these operations are:

Publish: In order to be accessible, a service needs to publish its description such that the requestor can subsequently find it. Where it is published can vary depending upon the requirements of the application (see Service Publication Stck discussion for more details)

Find: In the find operation, the service requestor retrieves a service description directly or queries the registry for the type of service required (see Service Discovery for more details). The find operation may be involved in two different lifecycle phases for the service requestor: at design time in order to retrieve the service's interface description for program development, and at runtime in order to retrieve the service's binding and location description for invocation.

Interact: Eventually, a service needs to be invoked. In the interact operation the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service. Examples of the interaction include: single message one way, broadcast from requester to many services, a multi message conversation, or a business process. Any of these types of interactions can be synchronous or asynchronous.

UNIT-IV WEB SERVICES

PART-A

1. Define Web Service [Apr/May 2019]

Web Services can convert your applications into Web-applications. Web Services are published, found, and used through the Web.

2. What is UDDI? [Apr/May 2019, Nov/Dec 2018]

UDDI is an XML-based standard for describing, publishing, and finding web services. UDDI stands for Universal Description, Discovery, and Integration. UDDI is a specification for a distributed registry of web services.

3. Difference between abstract and concrete service descriptions [Nov/Dec 2018]

An abstract WSDL document defines: the operations provided by the web service. The input, output and fault messages used by each operation to communicate with the web service, and their format.

A concrete WSDL document adds the information about how the web service communicates and where you can reach it.

4. Identify some types of Message Exchange Patterns [Nov/Dec 2018]

Common set of primitive meps are listed below

I. Request-response
Ii. Fire-and-forget
Iii. Complex meps

5. List down basic platform blocks. [Apr/May 2018]

1.services
2.Business processes
3.Human actors
4. Events
5.Service Descriptions, Contracts, and Policies
6.Service Compositions
7.Programs
8.Information Items, Data Items, and Data Stores

6. Define ASP.NET Web Forms [Apr/May 2018]

ASP.NET Web Forms is a part of the ASP.NET web application framework and is included with Visual Studio. Web Forms are pages that your users request using their browser. These pages can be written using a combination of HTML, client-script, server controls, and server code.

7. Write about JAX-WS. [Apr/May 2017]

The Java API for XML Web Services is a Java programming language API for creating web services, particularly SOAP services. JAX-WS is one of the Java XML programming APIs. It is part of the Java EE platform.

8. What is J2EE? [Apr/May 2017]

J2EE is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications.

9. List out any four pitfalls of SOA. [Nov/Dec 2017]

Implementation pitfalls

• Not Invented Here Syndrome
• Versioning

• Security

Architectural/design pitfalls

• Incorrect Granularity of Services
• SOA does not solve complexity automatically
• Big Design Upfront
• Incorrectly applied Canonical Data Model
• - Missing skills

Organizational pitfalls:

Unclear ownership/Project based funding

• Ignoring culture when introducing SOA

10. Define SOAP

SOAP is a method of accessing remote objects by sending XML messages, which provides platform and language independence. It works over various low-level communications protocols, but the most common is HTTP.

11. Explain about the operations in entity-centric

- GetSomething
- UpdateSomething
- AddSomething
- DeleteSomething

12. List the types of Choreography.

- Abstract Choreography
- "portable" choreography
- Concrete Choreography

13. Define Service Layers

When building various types of services, it becomes evident that they can be categorized depending on: - the type of logic they encapsulate - the extent of reuse potential this logic has - how this logic relates to existing domains within the enterprise As a result, there are three common service classifications that represent the primary service models used in SOA projects: - Entity Services - Task Services - Utility Services

14. Define Orchestration. [Nov/Dec 2019]

Refers to an executable business process that may interact with both internal and external Web services. Orchestration describes how Web services can interact at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional process. With orchestration, the process is always controlled from the perspective of one of the business parties.

15. Present an outline of WSDL. [Nov/Dec 2019]

WSDL is an XML format that describes distributed services on the Internet. A WSDL file describes the location of the service and the data to be passed in messages for particular operations. With regard to Sites, these messages contain remote procedure calls.

PART-B & C

1. What is WSDL? Explain the WSDL document structure with an example. [Apr/May 2019, Nov/Dec 2017]

* WSDL stands for Web Services Description Language
* WSDL is written in XML
* WSDL is an XML document
* WSDL is used to describe Web services

* WSDL is also used to locate Web services
* WSDL is a W3C recommendation

WSDL Describes Web Services. WSDL stands for Web Services Description Language. WSDL is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes. A WSDL document is just a simple XML document. It contains set of definitions to describe a web service.

4.2.2 The WSDL Document Structure

A WSDL document describes a web service using these major elements:

Element	Description
<types>	A container for data type definitions used by the web service
<message>	A typed definition of the data being communicated
<portType>	A set of operations supported by one or more endpoints
<binding>	A protocol and data format specification for a particular port type

Operation Types

The request-response type is the most common operation type, but WSDL defines four types:

Type	Definition
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

WSDL Elements

WSDL breaks down Web services into three specific, identifiable elements that can be combined or reused once defined.

Three major elements of WSDL that can be defined separately and they are:

- Types
- Operations
- Binding

A WSDL document has various elements, but they are contained within these three main elements, which can be developed as separate documents and then they can be combined or reused to form complete WSDL files.

Following are the elements of WSDL document. Within these elements are further sub elements, or parts:

* Definition: element must be the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
* Data types: the data types - in the form of XML schemas or possibly some other mechanism - to be used in the messages
* Message: an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
* Operation: the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message
* Port type: an abstract set of operations mapped to one or more end points, defining the collection of operations for a binding; the collection of operations, because it is abstract, can be mapped to multiple transports through various bindings.
* Binding: the concrete protocol and data formats for the operations and messages defined for a particular port type.
* Port: a combination of a binding and a network address, providing the target address of the service communication.
* Service: a collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

In addition to these major elements, the WSDL specification also defines the following utility elements:

* Documentation: element is used to provide human-readable documentation and can be included inside any other WSDL element.
* Import: element is used to import other WSDL documents or XML Schemas.

WSDL Definition Element

The <definition> element must be the root element of all WSDL documents. It defines the name of the web service.

Here is the example piece of code from last session which uses definition element.

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   ................................................
</definitions>
```

2. What is SOAP? Explain SOAP Message Framework with Example. [Nov/Dec 2019, Apr/May 2019, Nov/Dec 2018, Apr/May 2017]

SOAP is an XML-based protocol to let applications exchange information over HTTP. Or simpler: SOAP is a protocol for accessing a Web Service.

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C standard.

3. Outline the concept of UDDI. [Nov/Dec 2019, Nov/Dec 2018]

UDDI is an XML-based standard for describing, publishing, and finding web services.

- UDDI stands for Universal Description, Discovery, and Integration.
- UDDI is a specification for a distributed registry of web services.
- UDDI is a platform-independent, open framework.
- UDDI can communicate via SOAP, CORBA, Java RMI Protocol.
- UDDI uses Web Service Definition Language(WSDL) to describe interfaces to web services.
- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative, enabling businesses to discover each other and define how they interact over the Internet.

UDDI has two sections −

- A registry of all web service's metadata, including a pointer to the WSDL description of a service.
- A set of WSDL port type definitions for manipulating and searching that registry.

History of UDDI

- UDDI 1.0 was originally announced by Microsoft, IBM, and Ariba in September 2000.
- Since the initial announcement, the UDDI initiative has grown to include more than 300 companies including Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, and Sun.
- In May 2001, Microsoft and IBM launched the first UDDI operator sites and turned the UDDI registry live.
- In June 2001, UDDI announced Version 2.0.

Currently UDDI is sponsored by OASIS.

Partner Interface Processes

Partner Interface Processes (PIPs) are XML based interfaces that enable two trading partners to exchange data. Dozens of PIPs already exist. Some of them are listed here −

PIP2A2 − Enables a partner to query another for product information.

PIP3A2 − Enables a partner to query the price and availability of specific products.

PIP3A4 − Enables a partner to submit an electronic purchase order and receive acknowledgment of the order.

PIP3A3 − Enables a partner to transfer the contents of an electronic shopping cart.

PIP3B4 − Enables a partner to query the status of a specific shipment.

Private UDDI Registries

As an alternative to using the public federated network of UDDI registries available on the Internet, companies or industry groups may choose to implement their own private UDDI registries.

These exclusive services are designed for the sole purpose of allowing members of the company or of the industry group to share and advertise services amongst themselves.

Regardless of whether the UDDI registry is a part of the global federated network or a privately owned and operated registry, the one thing that ties them all together is a common web services API for publishing and locating businesses and services advertised within the UDDI registry.

4. Give Detailed note on WS-Atomic Transactions. [Nov/Dec 2018]

WS-AtomicTransaction

Atomic transactions greatly simplify application programming by shielding the application from system-generated exceptions, especially when they do not recur on retry.

Two-Phase Commit (2PC)

The atomic transaction 2PC protocol coordinates registered services to reach a commit or abort decision, and informs all services of the final result. The decision is the same for all the services in the transaction. Making this group decision uses two phases:

- Prepare phase: All participants are asked to get ready to either commit or abort, and then vote on the overall outcome. The vote is propagated up to the root coordinator to make the overall group decision.

- Commit phase: If all participants vote to commit, the decision will be to commit. Otherwise, it will be to abort.

An atomic transaction provides a simple programming model abstraction by making the following assumptions:

- Updates are hidden (isolated) until they are committed. For even the weakest database isolation levels, write locks are held on updated data until the end of the transaction, so that updates are hidden from view by other transactions until commit. When a transaction spans multiple systems using 2PC, all of these systems hold write locks until commit, so the systems must trust each other to be responsive.
- There is a single all-or-nothing decision for all participants. Any one of the systems can abort the entire atomic transaction, so these systems must trust each other to have cooperative intentions.
- The 2PC protocol also requires that services in the Prepare phase waiting for the final outcome to be communicated by the coordinator do not drop unilaterally their participation in the activity. This is quite a stringent constraint, as the Prepare phase may be of long duration, so these systems must trust each other to be responsive.

The Coordination Protocols

The three coordination protocols for atomic transactions specified in WS-AtomicTransaction are the following:

- Completion: An application uses this protocol to tell the coordinator to either try to commit or abort an atomic transaction. Upon receiving a Commit or Rollback message, the coordinator begins with Volatile2PC prepare phase and then proceeds through Durable2PC. The final result is signaled to the Completion protocol participant. This protocol does not guarantee that the message with the final outcome be delivered to the participant.
- Volatile2PC: Participants managing volatile resources, such as a cache, should use this 2PC protocol. Upon receiving a Completion protocol Commit message, the root coordinator begins the prepare phase of all participants registered for the Volatile2PC protocol. All participants registered for this protocol must respond before a Prepare message is sent to a participant registered for the Durable2PC protocol. Additional participants may register with the coordinator until the coordinator issues a Prepare to any durable participant. This protocol does not guarantee that the message with the final outcome be delivered to the participant.
- Durable2PC: Participants managing durable resources such as a database, a transactional file system or a durable queue service should use this 2PC protocol. Upon completing the prepare phase for Volatile2PC participants, the root coordinator begins the prepare phase for Durable2PC participants. All participants registered for this protocol must respond Prepared or ReadOnly before a Commit notification is issued to a participant registered for either protocol. This protocol guarantees that the message with the final outcome will be delivered to each of the participants.

5. Discuss on how SOA is related to the layers of J2EE Platform. [Nov/Dec 2019, Apr/May 2018]

J2EE solutions inherently are distributed and therefore componentized. The following types of components can be used to build J2EE Web applications:

Java Server Pages (JSPs) Dynamically generated Web pages hosted by the Web server. JSPs exist as text files comprised of code interspersed with HTML.

Struts An extension to J2EE that allows for the development of Web applications with sophisticated user -interfaces and navigation.

Java Servlets These components also reside on the Web server and are used to process HTTP request and response exchanges. Unlike JSPs, Servlets are compiled programs.

Enterprise JavaBeans (EJBs) The business components that perform the bulk of the processing within enterprise solution environments. They are deployed on dedicated application servers and can therefore leverage middleware features, such as transaction support.

While the first two components are of more relevance to establishing the presentation layer of a service-oriented solution, the latter two commonly are used to realize Web services.

6. Mention the benefits of JAX-RPC Concepts. [Apr/May 2018]

The Java API for XML-based RPC (JAX-RPC) specification enables you to develop SOAP-based interoperable and portable web services and web service clients. JAX-RPC 1.1 provides core APIs for developing and deploying web services on a Java platform and is a part of the Web Services for Java Platform, Enterprise Edition (Java EE) platform. The Java EE platform enables you to develop portable web services.

WebSphere® Application Server implements JAX-RPC 1.1 standards.

The JAX-RPC standard covers the programming model and bindings for using Web Services Description Language (WSDL) for Web services in the Java language. JAX-RPC simplifies development of web services by shielding you from the underlying complexity of SOAP communication.

On the surface, JAX-RPC looks like another instantiation of remote method invocation (RMI). Essentially, JAX-RPC enables clients to access a web service as if the web service was a local object mapped into the client's address space even though the web service provider is located in another part of the world. The JAX-RPC is done by using the XML-based protocol SOAP, which typically rides on HTTP.

JAX-RPC defines the mappings between the WSDL port types and the Java interfaces, as well as Java language and Extensible Markup Language (XML) schema types.

A JAX-RPC web service can be created from a JavaBeans or a enterprise bean implementation. You can specify the remote procedures by defining remote methods in a Java interface. You only need to code one or more classes that implement the methods. The remaining classes and other artifacts are generated by the web service vendor's tools. The following is an example of a web service interface:

```
package com.ibm.mybank.ejb;
import java.rmi.RemoteException;
import com.ibm.mybank.exception.InsufficientFundsException;
/**
  * Remote interface for Enterprise Bean: Transfer
  */
public interface Transfer_SEI extends java.rmi.Remote {
                    public void transferFunds(int fromAcctId, int toAcctId, float amount)
                                            throws java.rmi.RemoteException;

}Copy
```

The interface definition in JAX-RPC must follow specific rules:

The interface must extend java.rmi.Remote just like RMI.

Methods must create java.rmi.RemoteException.

Method parameters cannot be remote references.

Method parameter must be one of the parameters supported by the JAX-RPC specification. The following list are examples of method parameters that are supported. For a complete list of method parameters see the JAX-RPC specification.

Primitive types: boolean, byte, double, float, short, int and long

Object wrappers of primitive types: java.lang.Boolean, java.lang.Byte, java.lang.Double, java.lang.Float, java.lang.Integer, java.lang.Long, java.lang.Short

java.lang.String

java.lang.BigDecimal

java.lang.BigInteger

java.lang.Calendar

java.lang.Date

Methods can take value objects which consist of a composite of the types previously listed, in addition to aggregate value objects.

A client creates a stub and invokes methods on it. The stub acts like a proxy for the web service. From the client code perspective, it seems like a local method invocation. However, each method invocation gets marshaled to the remote server. Marshaling includes encoding the method invocation in XML as prescribed by the SOAP protocol.

The following are key classes and interfaces needed to write web services and web service clients:

Service interface: A factory for stubs or dynamic invocation and proxy objects used to invoke methods

ServiceFactory class: A factory for Services.

loadService

The loadService method is provided in WebSphere Application Server Version 6.0 to generate the service locator which is required by a JAX-RPC implementation. If you recall, in previous versions there was no specific way to acquire a generated service locator. For managed clients you used a JNDI method to get the service locator and for non-managed clients, you were required to instantiate IBM's specific service locator ServiceLocator service=new ServiceLocator(...); which does not offer portability. The loadService parameters include:

wsdlDocumentLocation: A URL for the WSDL document location for the service or null.

serviceName: A qualified name for the service

properties: A set of implementation-specific properties to help locate the generated service implementation class.

isUserInRole

The isUserInRole method returns a boolean indicating whether the authenticated user for the current method invocation on the endpoint instance is included in the specified logical role.

role: The role parameter is a String specifying the name of the role.

Service

Call interface: Used for dynamic invocation

Stub interface: Base interface for stubs

If you are using a stub to access the web service provider, most of the JAX-RPC API details are hidden from you. The client creates a ServiceFactory (java.xml.rpc.ServiceFactory). The client instantiates a Service (java.xml.rpc.Service) from the ServiceFactory. The service is a factory object that creates the port. The port is the remote service endpoint interface to the web service. In the case of DII, the Service object is used to create Call objects, which you can configure to call methods on the Web service's port.

7. How the Challenge of Coordinating Message Exchange Patterns? [Nov/Dec 2017]

4.5.1 The Need for Messaging

Web Services must exchange critical business information to succeed. Main features for business critical systems include the following

* Security
* Performance

* Reliability
* Integration

Web services are revolutionizing the Internet by enabling applications to speak a common language: XML. While Web services technology enables the execution of remote functions, it does not provide a robust infrastructure for handling information.

As the solution of this problem this article describes the concepts of web services with messaging.

Web Services

Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes.

How it differs from web application

Where the current web enables users to connect to applications, the web services architecture enables applications to connect to other applications. A web service is therefore a key technology in enabling business models to move from B2C (Business to Consumer) to B2B (Business to Business).

Web Services provide companies with a standards-based technology to simply integrate applications, share information with partners, and provide access to enterprise systems through a variety of devices.

Where is the problem?

An enterprise-class application that communicates with Web services must ensure that the data can be handled appropriately.

When employing Web services, one must ask questions like: Can our application scale to increased messaging demands? If our application crashes, is the Web service's data lost? How do we connect our Web-services-facing applications to back-end systems? These problems are created but unfortunately not solved through the Web services architecture. Web services must be combined with additional technology called Messaging for robust enterprise messaging.

Messaging Concept

Messaging provides high-speed, asynchronous, program-to-program communication with guaranteed delivery.

A simple way to understand what messaging does is to consider voice mail (as well as answering machines) for phone calls. Before voice mail, when someone called, if the receiver could not answer, the caller hung up and had to call back later to see if the receiver would answer at that time. With voice mail, when the receiver does not answer, the caller can leave him a message; later the receiver (at his convenience) can listen to the messages queued in his mailbox. Voice mail enables the caller to leave a message now so that the receiver can listen to it later, which is often a lot easier than trying to get the caller and the receiver on the phone at the same time. Voice mail bundles (at least part of) a phone call into a message and queues it for later; this is essentially how messaging works.

In enterprise computing, messaging makes communication between processes reliable, even when the processes and the connection between them are not so reliable.

There are two main defacto messaging standards for enterprise computing:

1. The Java 2 Platform, Enterprise Edition (J2EE) includes the Java Message Service API (JMS).

2. The Microsoft .NET Framework SDK (.NET) includes the System.Messaging namespace for accessing Microsoft Message Queue (MSMQ).

JMS

At its simplest level, JMS is java based messaging that sends messages between servers and clients. The format of these messages is quite flexible and can include ordinary text messages (including raw text, SOAP, and XML), entire Java objects, and “empty” messages that are suitable for basic communication (like acknowledgments).

What’s different about JMS compared with, say, low-level TCP/IP packets and Java Remote Method Invocation (RMI) is that while the other methods normally require real-time connectivity and messages that are sent synchronously, JMS systems are more flexible. In asynchronous mode, which is the default mode for JMS, clients don’t have to be connected all the time.

MSMQ

MSMQ is Microsoft's implementation of Messaging. It supports both Point-to-Point and Publisher-Subscriber models for messaging. Messages are typically kept in queues that are managed by Queue managers and applications access MSMQ via a simple client API.

Messages can be prioritized and delivered depending on their position in the queue, the first on queue having the highest priority. Queues can be implemented both in memory as well as on secondary storage such as disk. While express messages are stored in memory, recoverable messages are stored on disk.

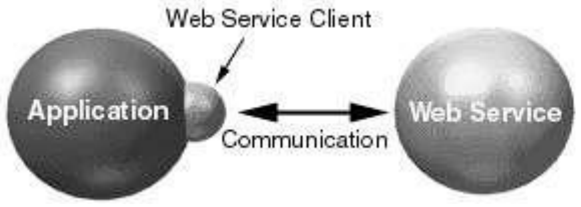
There are two types of queues in MSMQ

Public queues: These are queues published in the MQIS (Message Queue Information Store) and replicated throughout the Enterprise. Any computer on the Enterprise can hence locate them.

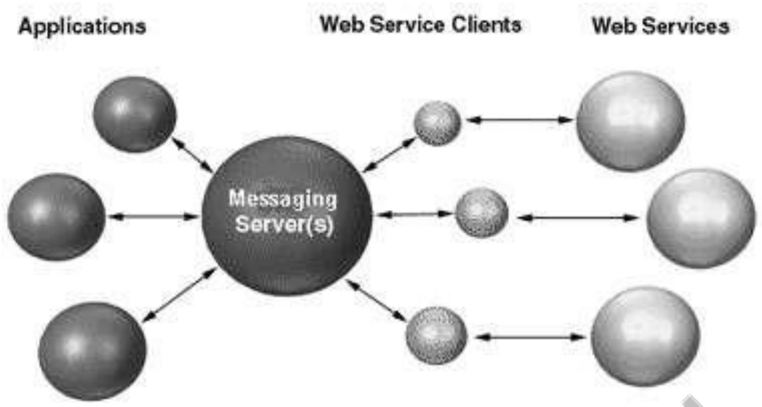
Private queues: These are queues that are not published in the MQIS and can only be accessed by systems that have access to the full path name or format name of the queue.

4.5.2 Combining Messaging with Web Services

A first-generation Web-services-enabled application contains or directly interfaces with a client that communicates with Web services as following image shows.



This architecture enables the application to find and communicate with remote systems, but does not implement data reliability, scalability. The addition of Messaging creates a second generation for architecting Web services systems, as shown in following Figure.



The inclusion of messaging servers decouples the application from the tasks of handling messages and web service clients. Applications communicate through an adapter to the messaging server.

In this new architecture, hybrid Messaging and Web services clients handle the bulk of the messaging duties. Information is passed through the Messaging server, which natively handles issues like fail over, load balancing, and guaranteed message delivery. By decoupling the Web services client from the application, several applications can effortlessly reuse a single Web services client. Decoupling makes it a simpler process to upgrade the Web service as inevitable software changes occur. Additionally, an application that becomes busy will have its Web services data automatically queued in the Messaging server until it is able to process the messages.

XML Messaging over Different Communication Protocols

In the early days of integrating applications together across the Internet, it was all about trying to blend together the most ubiquitous communication protocol, HTTP, with the most popular data format, XML. After all, every platform had an HTTP stack and an XML parser.

HTTP is a protocol that was designed to do a particular job, however, and has a very precise mode of operation—the client establishes a connection, and then initiates the conversation by sending a request to the server and synchronously getting back a response, with either party (or any HTTP intermediary between them) able to close the connection. Such characteristics make HTTP inappropriate for certain message exchange scenarios, such as peer-to-peer communication, where either party may establish the connection or initiate the conversation, or perhaps a single request/multiple response communication. It is lucky, then, that there are other successful Internet-friendly protocols, such as TCP, FTP, and SMTP, which have slightly different characteristics and that lend themselves to other types of application not well catered for by HTTP.

To carry XML messages unchanged over different communication protocols, those XML messages need to be self-contained and isolated from the underlying transport. This means that messages must hold all the information required to get the message where it is going and to process it. For instance, holding addressing information outside the XML message in the communication protocol layer alone makes it more difficult to use different transports to send the message. If addressing information is held inside the XML message, then it can be mapped to the needs of any transport used to send it. Similarly, imagine you wanted to exchange a series of XML messages that built up some conversational state. Holding the session identification information outside the XML messages in the communication protocol layer means it is difficult to change the transport used to send the messages. If such information is held inside the XML messages, it can be carried over any transport.

One of the fundamental tenets of SOAP is transport-neutral messaging. The SOAP message format, shown below, allows the protocol information needed to build higher-level message exchanges to be carried in the XML message, and independently of the underlying communication protocol.

UNIT -V BUILDING SOA-BASED APPLICATIONS

PART-A

1. Define loose Coupling. [Apr/May 2019]

Loose coupling is an approach to interconnecting the components in a system or network so that those components, also called elements, depend on each other to the least extent practicable. Coupling refers to the degree of direct knowledge that one element has of another.

2. What is web service composition? [Nov/Dec 2019, Apr/May 2019]

They have functional, behavioral, non-functional, and semantic characteristics. Web service composition is the mechanism for combining and reusing existing Web services to create new Web services. ... The proposed architecture extends the standard Web services business model to explicitly support Web services composition.

3. Mention the goals of performing a service-oriented analysis. [Nov/Dec 2018]

The overall goals of performing a service-oriented analysis are as follows:

- Define a preliminary set of service operation candidates.
- Group service operation candidates into logical contexts. These contexts represent service candidates.
- Define preliminary service boundaries so that they do not overlap with any existing or planned services.
- Identify encapsulated logic with reuse potential.
- Ensure that the context of encapsulated logic is appropriate for its intended use.
- Define any known preliminary composition models.

4. Show the structure of common WS-BPEL process definition. [Nov/Dec 2018]

The Web Services Business Process Execution Language (WS-BPEL), commonly known as BPEL (Business Process Execution Language), is an OASIS[1] standard executable language for specifying actions within business processes with web services. Processes in BPEL export and import information by using web service interfaces exclusively.

5. Mention the three types of Choreography. [Apr/May 2018]

Abstract, Portable and Concrete**.**

6. Define WS-Policy. [Apr/May 2018]

The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe the policies of a Web Service. WS-Policy defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements and capabilities.

7. What is Service Modeling Process? [Nov/Dec 2017]

SOMF is a service-oriented development life cycle methodology, a discipline-specific modeling process. It offers a number of modeling practices and disciplines that contribute to a successful service-oriented life cycle development andmodeling during a project.

8. Write any four attributes of "invoke" element of BPEL. [Nov/Dec 2017]

Domain

Elation

Depression

Enlightment

9. What are the Standards that Web Service depends on? [Apr/May 2017]

Web Services standards. **Web Services** are based on **standard** infrastructures, using XML to communicate across such **standard** protocols as HTTP, TCP/IP, SMTP or FTP. Because all communication is in XML, **Web Services** are not tied to any one operating system or programming language

10. What do you mean by WS-Security? [Apr/May 2017]

Web Services Security (WS Security) is a specification that defines how security measures are implemented in web services to protect them from external attacks. It is a set of protocols that ensure security for SOAP-based messages by implementing the principles of confidentiality, integrity and authentication.

11. Define - WS-Coordination framework. [Nov/Dec 2019]

The WS-Coordination framework exposes an Activation Service which supports the creation of coordinators for specific protocols and their associated contexts. The process of invoking an activation service is done asynchronously, and so the specification defines both the interface of the activation service itself, and that of the invoking service, so that the activation service can call back to deliver the results of the activation - namely a context that identifies the protocol type and coordinator location.

PART-B & C

1. Highlight the process of web service business process execution language and outline the structure of the same with an example. [Apr/May 2019]

Web Services Business Process Execution Language (WS-BPEL) is a programming language that, like Extensible Markup Language (XML), enables the defining and creation of business processes as Web services.

Structure of a BPEL4WS Process

```
<process ...>

<partners> ... </partners>

<!-- Web services the process interacts with -->

<containers> ... </containers>

<!– Data used by the process -->

<correlationSets> ... </correlationSets>

<!– Used to support asynchronous interactions -->

<faultHandlers> ... </faultHandlers>

<!–Alternate execution path to deal with faulty conditions -->

<compensationHandlers> ... </compensationHandlers>

<!–Code to execute when “undoing” an action --> (process body)

<!– What the process actually does -->

</process>
```

5.3.2 BPEL Basic Activities

```
<invoke partner=“...” portType=“...” operation=“...” inputContainer=“...” outputContainer=“...”/>

<!-- process invokes an operation on a partner:          -->

<receive partner=“...” portType=“...” operation=“...” container=“...” [createInstance=“...”]/>

<!-- process receives invocation from a partner:         -->

<reply partner=“...” portType=“...” operation=“...” container=“...”/>

<!-- process send reply message in partner invocation:          -->
```

```
<assign>
<copy>
<from container="..."/> <to container="..."/>
</copy>+
</assign>
<!– Data assignment between containers:      --> Partner Definitions and Links
<partner name="..." serviceLinkType="..." partnerRole="..." myRole="..."/>
<!– A partner is accessed over a WS "channel", defined by
a service link type -->
<serviceLinkType name="...">
<role name="...">
<portType name="..."/>*
</role>
 <role name="...">
<portType name="..."/>*
</role>
</serviceLinkType>
<!– A SLT defines two roles and the portTypes that each role needs to support -->
```

2. What is J2EE? Write a detailed note on SOA support with J2EE. [Apr/May 2019, Nov/Dec 2018, Apr/May 2017]

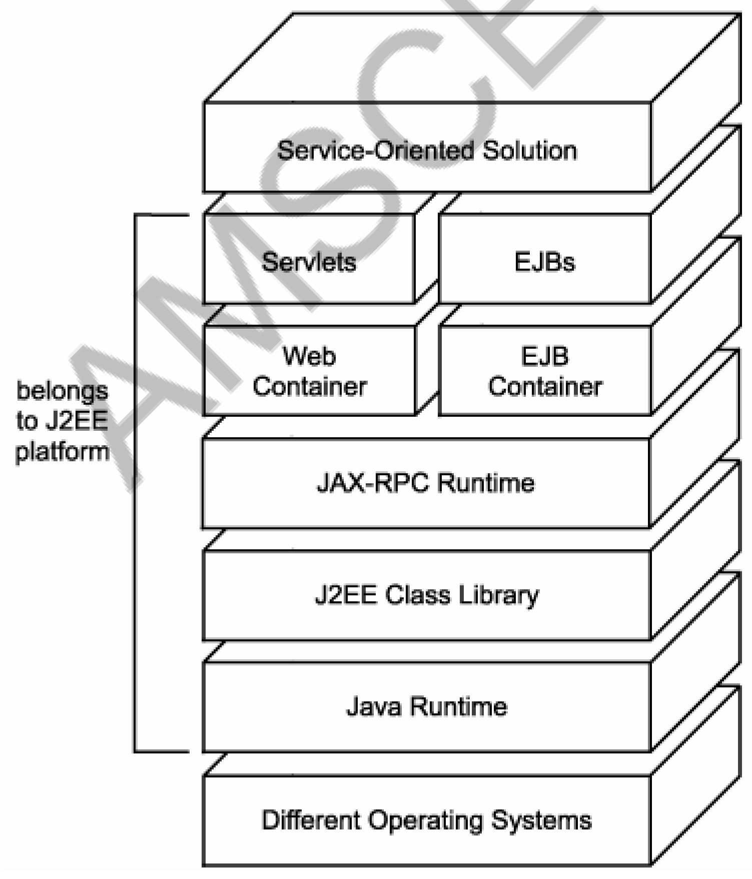
The Java 2 Platform Enterprise Edition (J2EE) is one of the two primary platforms currently being used to develop enterprise solutions using Web services. This section briefly introduces parts of the J2EE platform relevant to SOA. We then proceed to revisit the service-orientation principles and primary primitive and contemporary SOA characteristics established earlier in this book to discuss how these potentially can be realized using the previously explained parts of J2EE.

It is important to note that this section does not provide an in-depth explanation of the J2EE platform, nor do we get into how to program Web services using Java. There are already many comprehensive books that cover this vast subject area (see www.serviceoriented.ws for recommended reading). The purpose of this section is simply to continue our exploration of SOA realization. In doing so, we highlight some of the main areas of interest within the J2EE platform.

The Java 2 Platform is a development and runtime environment based on the Java programming language. It is a standardized platform that is supported by many vendors that provide development tools, server runtimes, and middleware products for the creation and deployment of Java solutions.

The Java 2 Platform is divided into three major development and runtime platforms, each addressing a different type of solution. The Java 2 Platform Standard Edition (J2SE) is designed to support the creation of desktop applications, while the Micro Edition (J2ME) is geared toward applications that run on mobile devices. The Java 2 Platform Enterprise Edition (J2EE) is built to support large-scale, distributed solutions. J2EE has been in existence for over five years and has been used extensively to build traditional n-tier applications with and without Web technologies.

The J2EE development platform consists of numerous composable pieces that can be assembled into full-fledged Web solutions. Let's take a look at some of the technologies more relevant to Web services.



The Servlets + EJBs and Web + EJB Container layers (as well as the JAX-RPC Runtime) relate to the Web and Component Technology layers established earlier in the SOA platform basics section. They do not map cleanly to these layers because to what extent component and Web technology is incorporated is largely dependent on how a vendor chooses to implement this part of a J2EE architecture.

3. Explain the steps involved in service modelling process. [Nov/Dec 2018, Nov/Dec 2017]

Step 1: Decompose the business process

Step 2: Identify business service operation candidates

Step 3: Abstract orchestration logic

Step 4: Create business service candidates

Step 5: Refine and apply principles of service-orientation

Step 6: Identify candidate service compositions

Step 7: Revise business service operation grouping

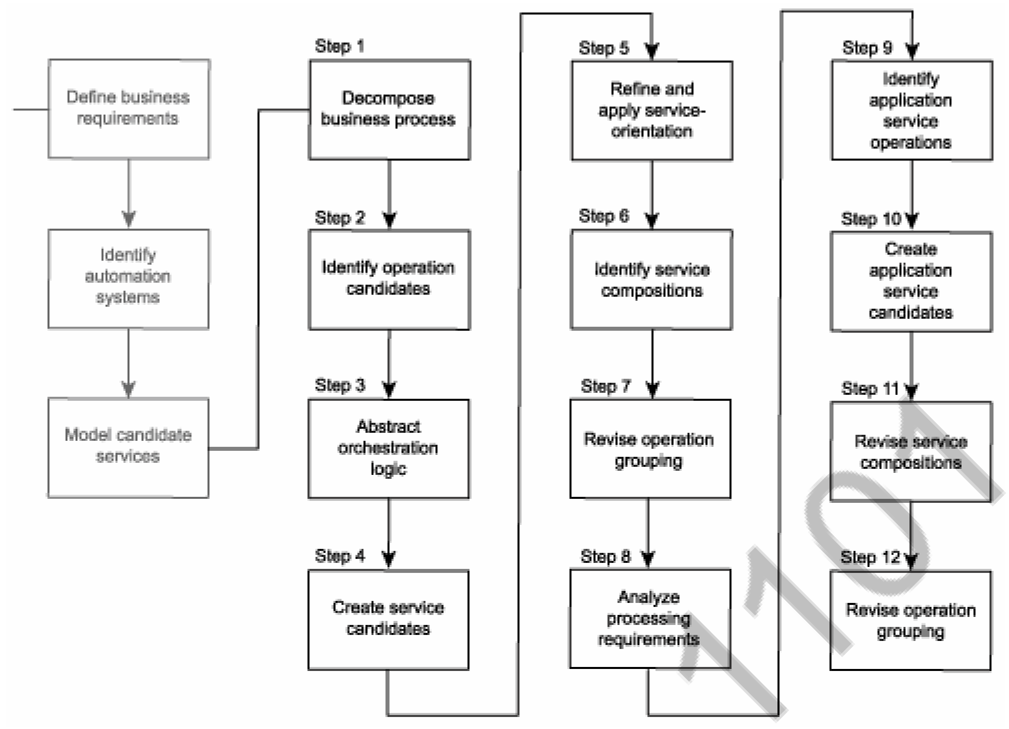
Step 8: Analyze application processing requirements

Step 9: Identify application service operation candidates

Step 10: Create application service candidates

Step 11: Revise candidate service compositions

Step 12: Revise application service operation grouping


Define business requirements
Identify automation systems
Model candidate services
Step 1
Decompose business process
Step 2
Identify operation candidates
Step 3
Abstract orchestration logic
Step 4
Create service candidates
Step 5
Refine and apply service-orientation
Step 6
Identify service compositions
Step 7
Revise operation grouping
Step 8
Analyze processing requirements
Step 9
Identify application service operations
Step 10
Create application service candidates
Step 11
Revise service compositions
Step 12
Revise operation grouping

4. Give the skeleton of Co-ordination context construct in WS-Coordination. [Nov/Dec 2018]

Refer Notes

5. List out the Security Threads in detail. [Apr/May 2018]

Refer Notes

6. Describe the Web Service Security Requirements in detail. [Nov/Dec 2019, Apr/May 2018, Nov/Dec 2017]

Refer Notes

7. Explain the various standards in development of web services. [Apr/May 2017]

Refer Notes

8. What is BPEL? Outline the building blocks of BPEL with example. [Nov/Dec 2019]

Refer Notes