SEMESTER-IV

17CAP405W **WEB SERVICES** **4H - 4C**

Instruction Hours / week: L: 4 T: 0 P: 0 C : 4 **Marks:** Internal: **40** External: **60** Total: **100**

End Semester Exam: 3Hours

Scope: This course will give students an overview of the web services architectures, and then learn the standard APIs for SOAP messaging and WSDL-driven, component-based service development.

Objective:

- Be able to describe the interoperable web services architecture, including the roles of SOAP and WSDL.
- Use lower-level SOAP and XML APIs for services and/or clients.
- Build and Host Web Services.

UNIT I

Introduction: What are Web Services – Importance of web services – Web services and enterprises; XML Fundamentals:: XML Documents - Namespaces – Schema – Processing XML.

UNIT II

SOAP: SOAP Model – messages – Encoding – RPC – Alternative SOAP encodings – Document, RPC, Literal, Encoded – SOAP, Web Services and the REST Architecture
WSDL: Structure – Using SOAP and WSDL. UDDI- UDDI Business Registry – Specification – Data Structures – Life cycle Management – Dynamic Access Point Management.

UNIT III

Advanced Web Services Technologies and Standards: Conversation – Overview – Web Services Conversation Language – WSCL Interface Components- Workflow-Business Process Management – Workflow and Workflow Management systems – BPEL. Transaction –ACID transaction – Distributed Transaction – OASIS Business Transaction Protocol.

UNIT IV

Security – Security Basics – Security Issues – Types of Security Attacks – WS –Security.
Mobile and Wireless – Mobile Web Services – Challenges with mobile – Proxy Based Mobile Systems -Direct Mobile Web service access - J2ME Web Services.

17CAP405W **WEB SERVICES** **4H - 4C**

UNIT V

Building Real World Enterprise Web Service and Applications: Real World Web Service Application Development – Development of Web services and Applications onto Tomcat application Server and Axis Soap Server.

SUGGESTED READINGS

1. Sandeep Chatterjee, James Webber (2009), Developing Enterprise Web Services: An Architect's Guide, 4th Edition, Pearson Education, New Delhi.
2. Martin Kalin (2013), Java Web Services: Up and Running, 2nd Edition, O'Reilly Media,USA.
3. Vikram Ramchand, Sonal Mukhi (2008), XML Web Services and SOAP , 1st Edition, BPB Publications, New Delhi.
4. Eric A Marks and Mark J Werrell. (2003), Executive Guide to Web Services, 1st Edition, John Wiley and Sons, New Delhi.

WEB SITES

1. www.w3schools.com/webservices/default.asp
2. en.wikipedia.org/wiki/Web_service
3. www.webservices.org/
4. https://www.cl.cam.ac.uk/~ib249/teaching/Lecture1.handout.pdf
5. http://www.codejava.net/java-ee/web-services/create-client-server-application-for-web-service-in-java

KarpagamAcademy of Higher Education
(Established Under Section 3 of UGC Act 1956)
Eachanari Post, Coimbatore – 641 021. INDIA
Phone : 0422-2611146, 2611082 Fax No : 0422 -2611043

17CAP405W WEB SERVICES

LECTURE PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT I	
1	1	Introduction: What are Web Services	T1: 2-8
2	1	Importance of web services	T1: 8-11
3	1	Web services and enterprises	T1:11-13
4	1	XML Fundamentals	T1: 17-19
5		XML Documents, Namespaces	T1: 19-20,W1
6	1	Schema	T1: 21-25
7	1	Processing XML	R1: 15-21
8	1	Recapitulation and discussion of Important Questions	
Total No. of Hours planned for Unit-I : 8 Hours			

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT II	
1	1	SOAP: SOAP Model, messages	T1:71-80,J2
2	1	Encoding, RPC, Alternative SOAP encodings	T1:80-85
3	1	Document, RPC, Literal, Encoded	T1:85-90
4	1	SOAP, Web Services and the REST Architecture	W2, T1:90-93
5	1	WSDL: Structure, Using SOAP and WSDL, UDDI	T1:94-97,R2:31-36
6	1	UDDI Business Registry, Specification, Data Structures	T1:100-112
7	1	Life cycle Management	T1:113-119
8	1	Dynamic Access Point Management	T1:121-128
9	1	Recapitulation and discussion of Important Questions	
Total No. of Hours planned for Unit II : 9 Hours			

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT III	
1	1	Advanced Web Services	T1:145-150
2	1	Technologies and Standards: Conversation	T1:150-155
3	1	Overview, Web Services Conversation Language	W2,T1:160-165
4	1	WSCL Interface Components	T1:177-180
5	1	Workflow-Business Process Management	T1:180-186, R2:48-54
6	1	Workflow and Workflow Management systems	T1:187-197
7	1	BPEL,Transaction, ACID transaction	J2,T1:198-205
8	1	Distributed Transaction, OASIS Business Transaction Protocol	T1:205-218
9	1	Recapitulation and Discussion of important Questions	
Total No. of Hours planned for Unit III : 9 Hours			

LECTURE PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT IV	
1	1	Security, Security Basics	T1:307-312
2	1	Types of Security Attacks	T1:312-325
3	1	Security Issues , WS –Security.	J3,T1:325-328
4	1	Mobile and Wireless	T1:330-336
5	1	Mobile Web Services	T1:337-345
6	1	Challenges with mobile, Proxy Based Mobile Systems	T1:350-357
7	1	Direct Mobile Web service access	T1:358-369
8	1	J2ME Web Services	R3:79-85
9	1	Recapitulation and Discussion of important Questions	
Total No. of Hours planned for Unit IV : 9 Hours			

LECTURE PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
		UNIT V	
1	1	Building Real World Enterprise Web Service and Applications	T1:439-445
2	1	Real World Web Service Application Development	T1:446-454
3	1	Development of Web services	T1:455-464
4	1	Applications onto Tomcat application Server	T1:465-472
5	1	Axis Soap Server	W4,T1:473-479
6	1	Recapitulation and Discussion of important Questions	
7	1	Discussion of Previous ESE papers	
8	1	Discussion of Previous ESE papers	
9	1	Discussion of Previous ESE papers	
		Total No. of Hours planned for Unit V : 9 Hours	
	TOTAL PLANNED HOURS : 44		

TEXT BOOK

T1: Sandeep Chatterjee, James Webber, 2009, "Developing Enterprise Web Services: An Architect's Guide", 4 th Edition, Pearson Education, New Delhi.

REFERENCES

R1: Martin Kalin, 2013, 2nd Edition "Java Web Services: UP and Running", O'Relly Media, USA
R2: Vikram RamChand, Sonal Mukhi, 1st Edition 2008, "XML Web Services and SOAP", BPB Publications, New Delhi.
R3: Eric Marks and Mark J Werrell, 1st Edition, 2003, Executive Guide to Web Services, John Wiley and sons, New Delhi.

WEBSITES

W1: www.w3schools.com/web services/default.asp
W2: en.wikipedia.org/wiki/web-service
W3: www.w3schools.com/web services/default.asp
W4:http://en.wikipedia.org/wiki/Apache-Axis

JOURNALS

J1: "Investigating SOAP and XML Technologies in Web services", IJSC ,Volume 3, No.4 , Nov 2012
J2: "Web Services Based on SOAP and REST Principles", IJSR ,Volume 3, Issue 5 , May 2013
J3: "Model driven Development of Web Service Transactions", Enterprise Modelling and Information System Architecture , Volume 1, No1, Oct 2005
J4:"Security Issues in Web services: A Review and Development Approach of Research Agenda", Assam University Journal of Science and Technology, Volume 5, No2 , 2010.

UNIT-I

SYLLABUS

Introduction: What are Web Services – Importance of web services – Web services and enterprises; **XML Fundamentals:** XML Documents - Namespaces – Schema – Processing XML.

INTRODUCTION

- **Web service** is a method of communications between two electronic devices over World Wide Web. A Web service is a software function provided at a network address over the web or the cloud; it is a service that is "always on" as in the concept of utility computing.
- The Web service can be defined as software system designed to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.
- **We can identify two major classes of Web services:**
    - REST-*compliant Web services*, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and *arbitrary Web services*, in which the service may expose an arbitrary set of operations

What are Web Services?

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications

- A web API is a development in web services where emphasis has been moving to simpler representational state transfer (REST) based communications. RESTful APIs do not require XML-based web service protocols (SOAP and WSDL) to support their light-weight interfaces.

The basic Web services platform is XML + HTTP.

- XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions.
- The HTTP protocol is the most used Internet protocol.

Web services platform elements:

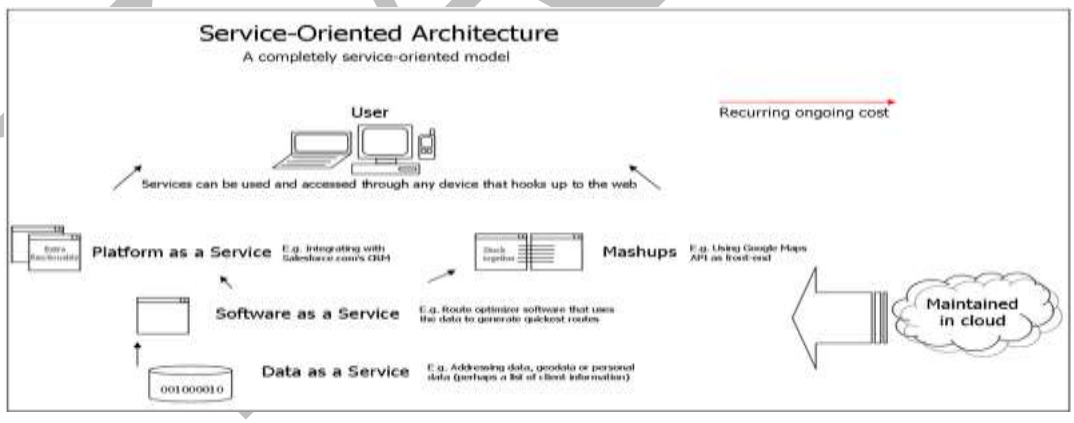
- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language

Interoperability has Highest Priority

- When all major platforms could access the Web using Web browsers, different platforms could interact. For **these platforms to work together, Web-applications were developed.**
- **Web-applications are** simple applications that run on the web. These are built around the Web browser standards and can be used by any browser on any platform.

XML web services:

- XML web services use Extensible Markup Language (XML) messages that follow the SOAP standard and have been popular with the traditional enterprises. In such systems, there is often a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL).
- The latter is not a requirement of a SOAP endpoint, but it is a prerequisite for automated client-side code generation in many Java and .NET SOAP frameworks (frameworks such as Apache Axis2, Apache CXF, Spring, gSOAP being notable exceptions). Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a web service.



AUTOMATED DESIGN METHODS

- Automated tools can aid in the creation of a web service. For services using WSDL it is possible to either automatically generate WSDL for existing classes (a bottom-up strategy) or to generate a class skeleton given existing WSDL (a top-down strategy).

- A developer using a bottom up method writes implementing classes first (in some programming language), and then uses a WSDL generating tool to expose methods from these classes as a web service. This is simpler to develop but may be harder to maintain if the original classes are subject to frequent change
- A developer using a top down method writes the WSDL document first and then uses a code generating tool to produce the class skeleton, to be completed as necessary. This way is generally considered more difficult but can produce cleaner designs and is generally more resistant to change.
- As long as the message formats between sender and receiver do not change, changes in the sender and receiver themselves do not affect the web service. The technique is also referred to as "contract first" since the WSDL (or contract between sender and receiver) is the starting point

Web Services take Web-applications to the Next Level

- By using Web services, your application can publish its function or message to the rest of the world. Web services use XML to code and to decode data, and SOAP to transport it (using open protocols). With Web services, your accounting department's Win 2k server's billing system can connect with your IT supplier's UNIX server.

Web Services have Two Types of Uses

Reusable application-components.

- Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.

Connect existing software.

- Web services can help to solve the interoperability problem by giving different applications a way to link their data.
- With Web services you can exchange data between different applications and different platforms

WEB SERVICES AND ENTERPRISES

- Enterprise web services can be considered to be a set of consistent, persistent, secure and RESTful web services that allow developers to integrate and link applications and web pages that contain essential business systems. These services include application and software development that are serving businesses.
- Most web services allow data to be accessed in a programmatic fashion, which allows for in-house or 3rd-party API (application programming interfaces) integration.

In-house vs 3rd-party APIs

- In-house APIs are those APIs developed internally, because they are developed using internal talent or resources. There are much significant strength inherent in the in-house API that make this format extremely attractive to a business. These strengths are:

    - High level of control over the process of development. Because the API is developed in-house, they can be dictated by the developer requesting it.
    - In-house APIs help encapsulate custom logic that's needed internally and they can be designed for very specific needs.
    - In-house APIs can be controlled from project's start to finish.

- 3rd-party APIs are those written by others and that provide additional functionality and data. Their main benefits include:

    - Utilization of experiences from different resource – It is, indeed very beneficial to use outside perspectives, because they can create different approaches which often can lead to better project results.
    - Time To Market – It is usually requires less time and money to use a 3d-party API.
    - Connect with Data – Other companies may have helpful value-add data streams you can access and consume into your own application. You'll need to make 3rd party API calls and handle the data in a way that promotes security, availability, fault tolerance and scalability.
    - Augment System functionality – Since you're connect with another company's API, then you're able to get data and functionality managed by them and augment your own systems.

- Both in-house and 3rd-part APIs are great and work well depending on which requirements you have. The goal is to blend each to deliver a value added solution.

Enterprise Web Services

- Enterprise Web Applications are dynamic web sites combined with server side programming that assist businesses to automatically manage their businesses online and enables them to utilize additional time with other beneficial activities.
- Examples of Web Applications are Online Banking, Social Networking, Online Reservations, eCommerce / Shopping Cart Applications, Interactive Games etc.
- How enterprises benefit from web app development:

    - **Cross-Platform** – Users can access web applications regardless of what operating systems they have (Windows, Mac, etc.) Moreover, with the variety of Internet browsers, such as Internet Explorer, Firefox or Chrome, users run into issues with software compatibility.

- **Interactivity –** It is important to retain existing clients and attract new ones. This requires continued interaction and a web-based application can do that. For example, a web app can be customized for sending and receiving emails. Users can strengthen communication effectively with web apps and eventually increase referrals.
- **Easy maintenance –** Using web-based applications does not require installation on users' hard drives which results in the reduction of memory on users' computers. It can be done directly onto a server and all the updates can be deployed effectively to users' computers.
- **Cost reduction –** Companies can save money by using web developed software, because they won't need to purchase a difficult hardware to support it and perform time-consuming updates.

THE IMPORTANCE OF WEB SERVICES

- You've likely noticed that vendors are increasingly touting the wonders of Web Services or other associated technologies such as SOAP, XML, WSDL, etc… And it's with good reason.

What is SOAP?

- SOAP is an XML-based protocol to let applications exchange information over HTTP.
- 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

What is WSDL?

- WSDL is an XML-based language for locating and describing Web services.

  - WSDL stands for Web Services Description Language
  - WSDL is based on XML
  - WSDL is used to describe Web services
  - WSDL is used to locate Web services
  - WSDL is a W3C standard

What is UDDI?

- UDDI is a directory service where companies can register and search for Web services.

  - UDDI stands for Universal Description, Discovery and Integration
  - UDDI is a directory for storing information about web services
  - UDDI is a directory of web service interfaces described by WSDL
  - UDDI communicates via SOAP
  - UDDI is built into the Microsoft .NET platform

A Web Service Example

- In the following example we will use ASP.NET to create a simple Web Service that converts the temperature from Fahrenheit to Celsius, and vice versa:

```
<%@ WebService Language="VBScript" Class="TempConvert" %>

Imports System
Imports System.Web.Services

Public Class TempConvert :Inherits WebService

<WebMethod()> Public Function FahrenheitToCelsius
(ByVal Fahrenheit As String) As String
  dim fahr
  fahr=trim(replace(Fahrenheit,",","."))
  if fahr="" or IsNumeric(fahr)=false then return "Error"
  return ((((fahr) - 32) / 9) * 5)
end function

<WebMethod()> Public Function CelsiusToFahrenheit
(ByVal Celsius As String) As String
  dim cel
  cel=trim(replace(Celsius,",","."))
  if cel="" or IsNumeric(cel)=false then return "Error"
  return ((((cel) * 9) / 5) + 32)
end function
end class
```

- This document is saved as an .asmx file. This is the ASP.NET file extension for XML Web Services.
- The first line in the example states that this is a Web Service, written in VBScript, and has the class name "TempConvert":

```
<%@ WebService Language="VBScript" Class="TempConvert" %>
```

- The next lines import the namespace "System.Web.Services" from the .NET framework:

```
Imports System
Imports System.Web.Services
```

- The next line defines that the "TempConvert" class is a WebService class type:

```
Public Class TempConvert :Inherits WebService
```

- The next steps are basic VB programming. This application has two functions. One to convert from Fahrenheit to Celsius, and one to convert from Celsius to Fahrenheit. The only difference from a normal application is that this function is defined as a "WebMethod()".
- Use "WebMethod()" to convert the functions in your application into web services:

```
<WebMethod()> Public Function FahrenheitToCelsius
(ByVal Fahrenheit As String) As String
  dim fahr
  fahr=trim(replace(Fahrenheit,",","."))
  if fahr="" or IsNumeric(fahr)=false then return "Error"
  return ((((fahr) - 32) / 9) * 5)
end function

<WebMethod()> Public Function CelsiusToFahrenheit
(ByVal Celsius As String) As String
  dim cel
  cel=trim(replace(Celsius,",","."))
  if cel="" or IsNumeric(cel)=false then return "Error"
  return ((((cel) * 9) / 5) + 32)
end function
```

Then, end the class:

```
end class
```

Publish the .asmx file on a server with .NET support, and you will have your first working Web Service.

XML FUNDAMENTALS

- XML stands for extensible Markup Language.
- XML is designed to transport and store data.
- XML is important to know, and very easy to learn.

XML DOCUMENT

- XML Document Example

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

<note>

  <to>Tove</to>
```

<from>Jani</from>

</note>

What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

The Difference Between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:
  - XML was designed to transport and store data, with focus on what data is
  - HTML was designed to display data, with focus on how data looks
- HTML is about displaying information, while XML is about carrying information.

XML Does Not DO Anything

- Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information.
- The following example is a note to Tove, from Jani, stored as XML:

<note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

- The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.

- But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.
- With XML You Invent Your Own Tags
- The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.That is because the XML language has no predefined tags.
- The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).
- XML allows the author to define his/her own tags and his/her own document structure.

XML is Not a Replacement for HTML

- Many application programming interfaces (APIs) have been developed to aid software developers with processing XML data, and several schema systems exist to aid in the definition of XML-based languages. Hundreds of document formats using XML syntax have been developed,including RSS, Atom, SOAP, and XHTML.
- XML-based formats have become the default for many office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice.org and LibreOffice (OpenDocument), and Apple's iWork. XML has also been employed as the base language for communication protocols, such as XMPP.

XML is a complement to HTML.

- It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.
- My best description of XML is this

- XML is a software- and hardware-independent tool for carrying information.
- XML is a W3C Recommendation
- XML became a W3C Recommendation February 10, 1998.
- XML is Everywhere
- XML is now as important for the Web as HTML was to the foundation of the Web.
- XML is the most common tool for data transmissions between all sorts of applications.
- XML Separates Data from HTML
- If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.
- With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML.
- With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

XML Simplifies Data Sharing

- In the real world, computer systems and databases contain data in incompatible formats.
- XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport

- One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.
- Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes

- Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.
- XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Your Data More Available

- Different applications can access your data, not only in HTML pages, but also from XML data sources.
- With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.
- XML is Used to Create New Internet Languages
- A lot of new Internet languages are created with XML.
- Here are some examples:

    - XHTML
    - WSDL for describing available web services
    - WAP and WML as markup languages for handheld devices
    - RSS languages for news feeds
    - RDF and OWL for describing resources and ontology
    - SMIL for describing multimedia for the web

XML Naming Rules

- XML elements must follow these naming rules:

    - Names can contain letters, numbers, and other characters

- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

What is an XML Element?

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can contain:

other elements

text

attributes

or a mix of all of the above...

```
<bookstore>
 <book category="CHILDREN">
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
 <book category="WEB">
  <title>Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>
```

</bookstore>

- In the example above, <bookstore> and <book> have element contents, because they contain other elements. <book> also has an attribute (category="CHILDREN"). <title>, <author>, <year>, and <price> have text content because they contain text.

XML Attributes

- In HTML, attributes provide additional information about elements:

  <img src="computer.gif">

  <a href="demo.asp">

- Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

  <file type="gif">computer.gif</file>

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used. For a person's sex, the person element can be written like this:

<person sex="female">

or like this:

<person sex='female'>

- If the attribute value itself contains double quotes you can use single quotes, like in this example:

  <gangster name='George "Shotgun" Ziegler'>

  or you can use character entities:

  <gangster name="George "Shotgun" Ziegler">

XML Elements vs. Attributes

- Take a look at these examples:

  <person sex="female">

```
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>

<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

- In the first example sex is an attribute. In the last, sex is an element. Both examples provide the same information.
- There are no rules about when to use attributes or when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead.

XML NAMESPACES

- 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.
- An XML parser 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 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>
```

- In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.
- When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.
- Namespaces can be declared in the elements where they are used or in the XML root element:
- The purpose is to give the namespace a unique name. However, often companies use the namespace as a pointer to a web page containing namespace information.

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).

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>
```

- Namespaces in Real Use
- XSLT is an XML language that can be used to transform XML documents into other formats, like HTML.
- In the XSLT document below, you can see that most of the tags are HTML tags.
- The tags that are not HTML tags have the prefix xsl, identified by the namespace

xmlns:xsl="http://www.w3.org/1999/XSL/Transform":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
```

```
<html>
<body>
 <h2>My CD Collection</h2>
 <table border="1">
  <tr>
   <th align="left">Title</th>
   <th align="left">Artist</th>
  </tr>
  <xsl:for-each select="catalog/cd">
  <tr>
   <td><xsl:value-of select="title"/></td>
   <td><xsl:value-of select="artist"/></td>
  </tr>
  </xsl:for-each>
 </table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Example XML: Show XML data inside an HTML div element

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

<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.open("GET","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++)
  {
```

```
 document.write("<tr><td>");
 document.write(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
 document.write("</td><td>");
 document.write(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
 document.write("</td></tr>");
 }
document.write("</table>");
</script>

</body>
</html>
```

Example XML: Get the value of an XML attribute

```
<!DOCTYPE html>
<html>
<body>
<script>
if (window.XMLHttpRequest)
 {// code for IE7+, Firefox, Chrome, Opera, Safari
 xmlhttp=new XMLHttpRequest();
 }
else
 {// code for IE6, IE5
 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
xmlhttp.open("GET","books.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

txt=xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
document.write(txt);
</script>
</body>
</html>
```

Namespaces in APIs and XML object models

- Different specifications have taken different approaches on how namespace information is presented to applications. Nearly all programming models allow the name of an element or attribute node to be retrieved as a three-part name: the local name, the namespace prefix, and the namespace URI.
- Applications should avoid attaching any significance to the choice of prefix, but the information is provided because it can be helpful to human readers. Names are considered

equal, if the namespace URI and local name match. In addition, most models provide some way of determining which namespaces have been declared for a given element. This information is needed because some XML vocabularies allow qualified names (containing namespace prefixes) to appear in the content of elements or attributes, as well as in their names. There are three main ways this information can be provided:

- As attribute nodes named "xmlns" or "xmlns:xxx", exactly as the namespaces are written in the source XML document. This is the model presented by DOM.
    - As namespace declarations: distinguished from attributes, but corresponding one-to-one with the relevant attributes in the source XML document. This is the model presented by JDOM.
    - As in-scope namespace bindings: in this model, the application is able to determine which namespaces are in scope for any given element, but is not able to determine which elements contain the actual declarations. This is the model used in XPath, XSLT, and XQuery.

- Transaction Processing over XML (TPoX) is a computing benchmark for XML database systems. As a benchmark, TPoX is used for the performance testing of database management systems that are capable of storing, searching, modifying and retrieving XML data. The goal of TPoX is to allow database designers, developers and users to evaluate the performance of XML database features, such as the XML query languages XQuery and SQL/XML, XML storage, XML indexing, XML Schema support, XML updates, transaction processing and logging, andconcurrency control. TPoX includes XML update tests based on the XQuery Update Facility.
- The TPoX benchmark exercises the processing of data-centric XML, in contrast to content- or document-centric XML.
- TPoX was originally developed and tested by IBM and Intel, but became an open source project on SourceForge in January 2007. TPoX 1.1 was released in June 2007. TPoX 2.0 was released in July 2009.
- The TPoX benchmark package contains the following:
- XML Schemas that define the XML data used in the benchmark.
- An XML data generation tool to generate an arbitrary number of XML documents with well-defined value distributions and referential integrity across documents. The XML data is generated conforming to industry schema such as FIXML to model real-world applications.

- Workloads which are executed on the generated data. A workload is a set of transactions. A transaction can be a query in XQuery or SQL/XML notation or an insert, update or delete operation.
- A Java application which acts as a workload driver. It is configurable and can spawn 1 to n parallel threads to simulate concurrent database users. Each user connects to the database and executes a random sequence of transactions defined in the workload. Parameter markers in the transactions are replaced by real values that are drawn from random value distributions. The workload driver collects and reports performance metrics, such as the transaction throughput as well as minimum, maximum and average response times.

XML SCHEMA (W3C)

- A newer schema language, described by the W3C as the successor of DTDs, is XML Schema, often referred to by the initialism for XML Schema instances, XSD (XML Schema Definition). XSDs are far more powerful than DTDs in describing XML languages. They use a rich datatyping system and allow for more detailed constraints on an XML document's logical structure. XSDs also use an XML-based format, which makes it possible to use ordinary XML tools to help process.

RELAX NG

- RELAX NG was initially specified by OASIS and is now also an ISO/IEC International Standard (as part of DSDL). XML Schema[edit]
- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against an XML Schema is both "Well Formed" and "Valid".

XML SCHEMA

- XML Schema is an XML-based alternative to DTD:

```
<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>
```

- The Schema above is interpreted like this:

  <xs:element name="note"> defines the element called "note"

  <xs:complexType> the "note" element is a complex type

  <xs:sequence> the complex type is a sequence of elements

  <xs:element name="to" type="xs:string"> the element "to" is of type string (text)

  <xs:element name="from" type="xs:string"> the element "from" is of type string

  <xs:element name="heading" type="xs:string"> the element "heading" is of type string

  <xs:element name="body" type="xs:string"> the element "body" is of type string

- XML Schemas are More Powerful than DTD
- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

Why Use an XML Schema?

- With XML Schema, your XML files can carry a description of its own format.
- With XML Schema, independent groups of people can agree on a standard for interchanging data.
- With XML Schema, you can verify data.

XML Schemas Support Data Types

- One of the greatest strengths of XML Schemas is the support for data types:
- It is easier to describe document content
- It is easier to define restrictions on data
- It is easier to validate the correctness of data
- It is easier to convert data between different data types

XML Schemas use XML Syntax

- Another great strength about XML Schemas is that they are written in XML:
- You don't have to learn a new language
- You can use your XML editor to edit your Schema files

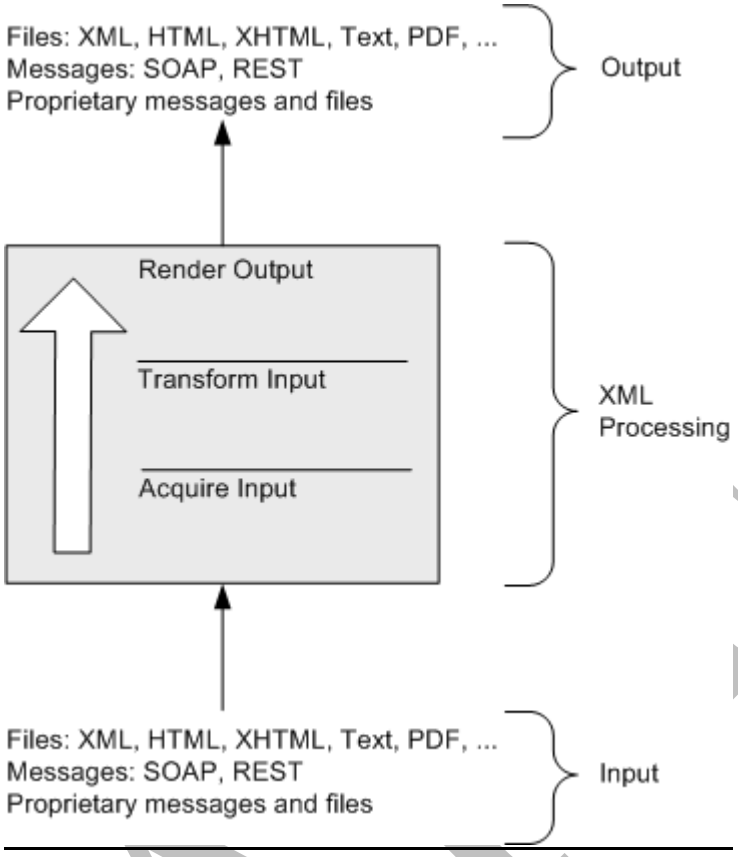
- You can use your XML parser to parse your Schema files
- You can manipulate your Schemas with the XML DOM
- You can transform your Schemas with XSLT

Processing XML

- XML can be used for the presentation, communication, and storage of data. This webpage provides an overview of the XML processing to accomplish this.
- The figure below illustrates transforming the contents of files, messages, or data from one format to another. Note that the input and output options are the same. This is meant to illustrate that the transformation can be from some non-XML format to XML, from XML to some non-XML format, from some non-XML format to another non-XML format, or from one XML vocabulary to another XML vocabulary. Nevertheless, since this website emphasizes the use of XML in service-oriented architectures, some XML format or message that uses XML will most likely be on one end fo the transformation or the other. This is why it is called "XML processing."

XML Processing

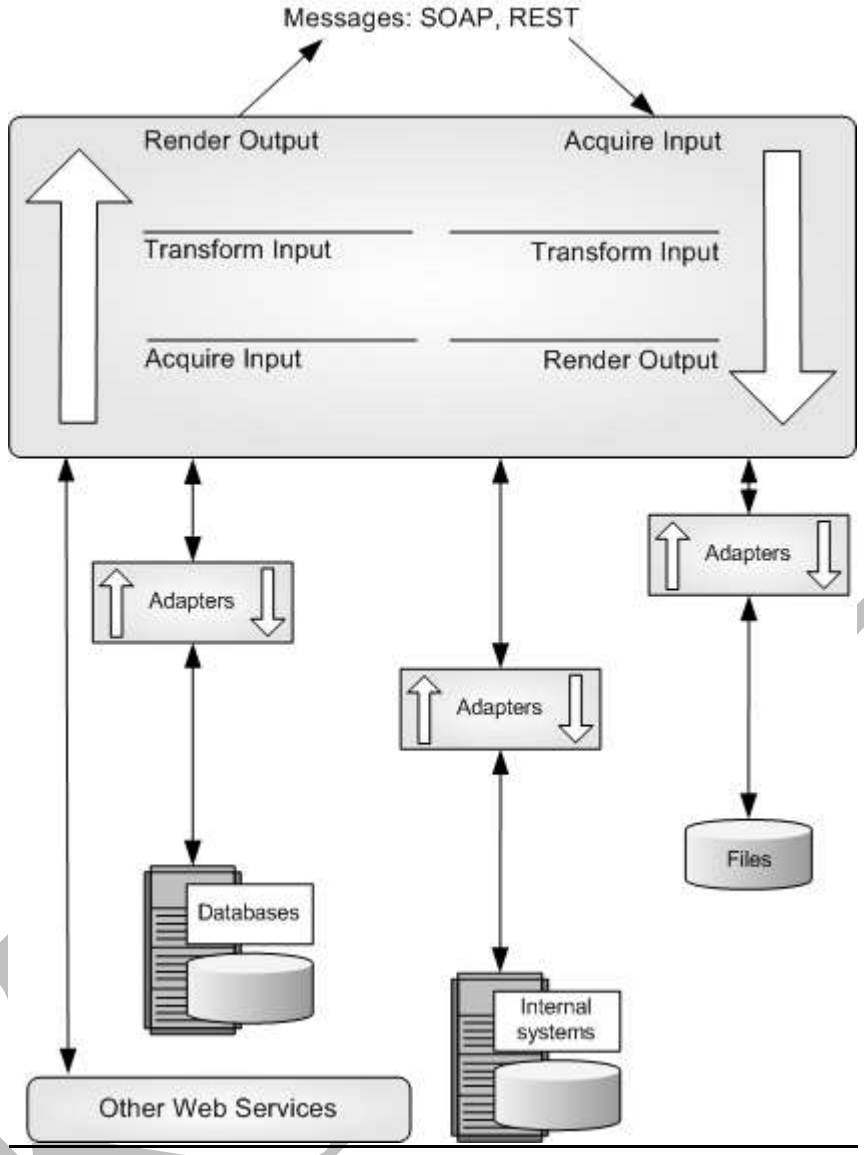
- There are many ways to accomplish the transformation. You could use custom programming, a tool that generates code, or a specialized scripting language. More information on some options can be found in the XML resources under the related content heading below.



- The basic process is to acquire the input, transform the input, and then render the output in the desired format.
- The next figure places XML processing in the context of Web Services and a service-oriented architecture (SOA). The large shared box near the top represents a service that provides or consumes SOAP or REST messages.
- Note that to provide a message, the service will need to take in possibly multiple inputs, transform those inputs, and render a message. Conversely, the service will need to take in a message, transform the XML in the message into possibly multiple outputs, and then render those outputs. The latter is shown at the right of the large shaded box.

XML Processing for Web Services

- Adapters are also shown in this figure using the smaller shaded boxes. Each adapter will need to deal with some non-XML format. It will need to use that non-XML format as input and then transform and render it as XML to be used by the service at the top of the figure. Conversely, the adapter will need to take XML as input from the service and then tranform and render it to some non-XML format. More on adapters.



- This figure shows the service and adapters as separate entities in the architecture. Of course, the service and the adapters and all the XML processing could be written as one program. It depends on how you structure your architecture.

Programming Interface

- The design goals of XML include, "It shall be easy to write programs which process XML documents." Despite this, the XML specification contains almost no information about how programmers might go about doing such processing. The XML Infoset specification provides a vocabulary to refer to the constructs within an XML document, but also does not provide any guidance on how to access this information. A variety of APIs for accessing XML have been developed and used, and some have been standardized.

Existing APIs for XML processing tend to fall into these categories:

- Stream-oriented APIs accessible from a programming language, for example SAX and StAX.
- Tree-traversal APIs accessible from a programming language, for example DOM.
- XML data binding, which provides an automated translation between an XML document and programming-language objects.
- Declarative transformation languages such as XSLT and XQuery.
- Stream-oriented facilities require less memory and, for certain tasks which are based on a linear traversal of an XML document, are faster and simpler than other alternatives. Tree-traversal and data-binding APIs typically require the use of much more memory, but are often found more convenient for use by programmers; some include declarative retrieval of document components via the use of XPath expressions.
- XSLT is designed for declarative description of XML document transformations, and has been widely implemented both in server-side packages and Web browsers. XQuery overlaps XSLT in its functionality, but is designed more for searching of large XML databases.

SIMPLE API FOR XML

- Simple API for XML (SAX) is a lexical, event-driven interface in which a document is read serially and its contents are reported as callbacks to various methods on a handler object of the user's design. SAX is fast and efficient to implement, but difficult to use for extracting information at random from the XML, since it tends to burden the application author with keeping track of what part of the document is being processed.
- It is better suited to situations in which certain types of information are always handled the same way, no matter where they occur in the document.

Pull parsing

- Pull parsing treats the document as a series of items which are read in sequence using the Iterator design pattern. This allows for writing of recursive-descent parsers in which the structure of the code performing the parsing mirrors the structure of the XML being parsed, and intermediate parsed results can be used and accessed as local variables within

the methods performing the parsing, or passed down (as method parameters) into lower-level methods, or returned (as method return values) to higher-level methods. Examples of pull parsers include StAX in the Javaprogramming language, XMLReader in PHP, ElementTree.iterparse in Python, System.Xml.XmlReader in the .NET Framework, and the DOM traversal API (NodeIterator and TreeWalker).

- A pull parser creates an iterator that sequentially visits the various elements, attributes, and data in an XML document. Code which uses this iterator can test the current item (to tell, for example, whether it is a start or end element, or text), and inspect its attributes (local name, namespace, values of XML attributes, value of text, etc.), and can also move the iterator to the next item. The code can thus extract information from the document as it traverses it.
- The recursive-descent approach tends to lend itself to keeping data as typed local variables in the code doing the parsing, while SAX, for instance, typically requires a parser to manually maintain intermediate data within a stack of elements which are parent elements of the element being parsed. Pull-parsing code can be more straightforward to understand and maintain than SAX parsing code.

Document Object Model

- The Document Object Model (DOM) is an interface-oriented application programming interface that allows for navigation of the entire document as if it were a tree of node objects representing the document's contents.
- A DOM document can be created by a parser, or can be generated manually by users (with limitations). Data types in DOM nodes are abstract; implementations provide their own programming language-specific bindings. DOM implementations tend to be memory intensive, as they generally require the entire document to be loaded into memory and constructed as a tree of objects before access is allowed.

Data binding

- Another form of XML processing API is XML data binding, where XML data are made available as a hierarchy of custom, strongly typed classes, in contrast to the generic objects created by aDocument Object Model parser. This approach simplifies code development, and in many cases allows problems to be identified at compile time rather than run-time. Example data binding systems include the Java Architecture for XML Binding (JAXB) and XML Serialization in .NET.

XML as data type

- XML has appeared as a first-class data type in other languages. The ECMAScript for XML (E4X) extension to the ECMAScript/JavaScript language explicitly defines two specific objects (XML and XMLList) for JavaScript, which support XML document nodes and XML node lists as distinct objects and use a dot-notation specifying parent-child relationships.[19] E4X is supported by theMozilla 2.5+ browsers (though now deprecated) and Adobe Actionscript, but has not been adopted more universally. Similar notations are

used in Microsoft's LINQ implementation for Microsoft .NET 3.5 and above, and in Scala (which uses the Java VM).

XSLT

- **XSLT** is a language for transforming XML documents.
- **XPath** is a language for navigating in XML documents.
- **XQuery** is a language for querying XML documents.

- It Started with XSL

XSL stands for EXtensible Stylesheet Language.

- The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.

CSS = Style Sheets for HTML

- HTML uses predefined tags. The meaning of, and how to display each tag is well understood.
- CSS is used to add styles to HTML elements.

XSL = Style Sheets for XML

- XML does not use predefined tags, and therefore the meaning of each tag is not well understood.
- A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!
- So, XSL describes how the XML elements should be displayed.
- XSL - More Than a Style Sheet Language
- XSL consists of four parts:
    - XSLT - a language for transforming XML documents
    - XPath - a language for navigating in XML documents
    - XSL-FO - a language for formatting XML documents (discontinued in 2013)
    - XQuery - a language for querying XML documents
- With the CSS3 Paged Media Module, W3C has delivered a new standard for document formatting. So, since 2013, CSS3 is proposed as an XSL-FO replacement

What is XSLT?

- XSLT stands for XSL Transformations

- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation
- XSLT = XSL Transformations
- XSLT is the most important part of XSL.
- XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

- With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more
- A common way to describe the transformation process is to say that XSLT transforms an XML source-tree into an XML result-tree.

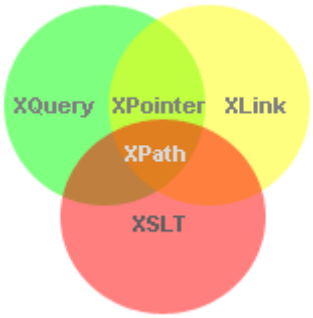
XSLT Uses XPath

- XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

XSLT Browser Support

- All major browsers support XSLT and XPath.
- XPath can be used to navigate through elements and attributes in an XML document.

XPath



- XPath stands for XML Path Language
- XPath uses "path like" syntax to identify and navigate nodes in an XML document
- XPath contains over 200 built-in functions
- XPath is a major element in the XSLT standard
- XPath is a W3C recommendation

- XPath Path Expressions
- XPath uses path expressions to select nodes or node-sets in an XML document.
- XPath Standard Functions
- XPath includes over 200 built-in functions.
- There are functions for string values, numeric values, booleans, date and time comparison, node manipulation, sequence manipulation, and much more
- Today XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

- XPath is Used in XSLT
- XPath is a major element in the XSLT standard.
- With XPath knowledge you will be able to take great advantage of your XSLT knowledge.
- XLink is used to create hyperlinks in XML documents.

XPath

- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files
- XLink is a W3C Recommendation
- XLink Browser Support
- There is no browser support for XLink in XML documents
- However, all major browsers support XLinks in SVG.

XLink

XLink Syntax

- In HTML, the <a> element defines a hyperlink. However, this is not how it works in XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what link elements will be called in XML documents.

- Below is a simple example of how to use XLink to create links in an XML document:

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

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

- To get access to the XLink features we must declare the XLink namespace. The XLink namespace is: "http://www.w3.org/1999/xlink".

- The xlink:type and the xlink:href attributes in the <homepage> elements come from the XLink namespace.

- The xlink:type="simple" creates a simple "HTML-like" link (means "click here to go there").

- The xlink:href attribute specifies the URL to link to.

XLink Example

- The following XML document contains XLink features:

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

<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">

<book title="Harry Potter">
  <description
  xlink:type="simple"
  xlink:href="/images/HPotter.gif"
  xlink:show="new">
  As his fifth year at Hogwarts School of Witchcraft and
  Wizardry approaches, 15-year-old Harry Potter is.....
  </description>
</book>

<book title="XQuery Kick Start">
  <description
  xlink:type="simple"
  xlink:href="/images/XQuery.gif"
  xlink:show="new">
  XQuery Kick Start delivers a concise introduction
  to the XQuery standard.....
  </description>
</book>

</bookstore>
```

Example explained:

The XLink namespace is declared at the top of the document (xmlns:xlink="http://www.w3.org/1999/xlink")
The xlink:type="simple" creates a simple "HTML-like" link
The xlink:href attribute specifies the URL to link to (in this case - an image)

The xlink:show="new" specifies that the link should open in a new window

XLink - Going Further

- In the example above we have demonstrated simple XLinks. XLink is getting more interesting when accessing remote locations as resources, instead of standalone pages.

- If we set the value of the xlink:show attribute to "embed", the linked resource should be processed inline within the page. When you consider that this could be another XML document you could, for example, build a hierarchy of XML documents.

- You can also specify WHEN the resource should appear, with the xlink:actuate attribute.

XLink Attribute Reference

- Attribute Value Description
- xlink:actuate onLoad
- onRequest
- other
- none Defines when the linked resource is read and shown:
- onLoad - the resource should be loaded and shown when the document loads
- onRequest - the resource is not read or shown before the link is clicked
- xlink:href URL Specifies the URL to link to
- xlink:show embed
- new
- replace
- other
- none Specifies where to open the link. Default is "replace"
- xlink:type simple
- extended
- locator
- arc
- resource
- title
- none Specifies the type of link

XPointer

- XPointer allows links to point to specific parts of an XML document
- XPointer uses XPath expressions to navigate in the XML document
- XPointer is a W3C Recommendation
- XPointer Browser Support
- There is no browser support for XPointer. But XPointer is used in other XML languages.

XPointer Example

- In this example, we will use XPointer in conjunction with XLink to point to a specific part of another document.

- We will start by looking at the target XML document (the document we are linking to):

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

<dogbreeds>

<dog breed="Rottweiler" id="Rottweiler">
  <picture url="https://dog.com/rottweiler.gif" />
  <history>The Rottweiler's ancestors were probably Roman
  drover dogs.....</history>
  <temperament>Confident, bold, alert and imposing, the Rottweiler
  is a popular choice for its ability to protect....</temperament>
</dog>

<dog breed="FCRetriever" id="FCRetriever">
  <picture url="https://dog.com/fcretriever.gif" />
  <history>One of the earliest uses of retrieving dogs was to
  help fishermen retrieve fish from the water....</history>
  <temperament>The flat-coated retriever is a sweet, exuberant,
  lively dog that loves to play and retrieve....</temperament>
</dog>

</dogbreeds>
```

- Note that the XML document above uses id attributes on each element!

- So, instead of linking to the entire document (as with XLink), XPointer allows you to link to specific parts of the document. To link to a specific part of a page, add a number sign (#) and an XPointer expression after the URL in the xlink:href attribute, like this: xlink:href="https://dog.com/dogbreeds.xml#xpointer(id('Rottweiler'))". The expression refers to the element in the target document, with the id value of "Rottweiler".

- XPointer also allows a shorthand method for linking to an element with an id. You can use the value of the id directly, like this: xlink:href="https://dog.com/dogbreeds.xml#Rottweiler".

- The following XML document contains links to more information of the dog breed for each of my dogs:

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

<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

<mydog>
 <description>
 Anton is my favorite dog. He has won a lot of.....
 </description>
 <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#Rottweiler">
 Fact about Rottweiler
 </fact>
</mydog>

<mydog>
 <description>
 Pluto is the sweetest dog on earth.....
 </description>
 <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#FCRetriever">
 Fact about flat-coated Retriever
 </fact>
</mydog>

</mydogs>
```

XQuery

What is XQuery?

- XQuery is to XML what SQL is to databases.
- XQuery was designed to query XML data.

XQuery Example

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

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 major databases

- XQuery is a W3C Recommendation
- XQuery is About Querying XML
- XQuery is a language for finding and extracting elements and attributes from XML documents.

Here is an example of what XQuery could solve:

"Select all CD records with a price less than $10 from the CD collection stored in cd_catalog.xml"

XQuery and XPath

- XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators. If you have already studied XPath you will have no problems with understanding XQuery.

- XQuery - Examples of Use
- XQuery can be used to:

- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information

POSSIBLE QUESTIONS

PART – A

(Online Exam 20*1=20 marks)

- **Multiple Choice Questions available in Moodle**

PART – B

(Each Question carries 6 marks)

1. Explain the importance of Web services
2. Discuss in detail about XML Schema
3. Discuss in detail about XML namespaces.
4. Explain about XML document
5. Explain the advantages of XML and its process
6. Differentiate between XML and HTML
7. Explain about XML Processing
8. Explain the different Components of XML tree Structure
9. Explain about Default Namespace
10. Discuss in detail (i) XSL (ii) X-Link

PART – C

(Compulsory Question carries 10 marks)

1. Create a menu in XML
2. Write a program to view an Employee details and display the student details with an XSLT Stylesheet
3. Write a program to show XML data inside an HTML div element
4. Difference between XML and XML Schema
5. Explain about Web services and Enterprises

Questions	Opt1	Opt2	Opt3	Opt4	Opt5	Opt6
XML refers to _____	eXtended Markup Language	Extended Markup language	eXtensible Markup Language	Extra Markup Language		
ROI refers to_____	Return of Investment	Return on Investment	return of interest	Return of Image		
web services is created to support_____	ROI	WSDL	XML	QoS		
WSDL means _____	Web services Description Language	World services description language	Web server Description Language	Web service development life cycle		
_____be created to support QoS	WSDL	XML	Web services	HTML		
QoS means_____	Quality-of-Service	Quality-of-Server	Quantity-of-service	Quantity-of-server		
_____platform consisting of SOAP,WSDL and UDDI	UNIX	WINDOWS	Web service	LINUX		
_____represent a new architectural paradigm for applications	top-down	bottom up	Web service	Middle level		
_____represent reusable software building blocks that are URL addressable	web service	Web development	Web technology	w3c		

UDDI means _____	Universal discovery description and integration	Universal description discovery and integration	Universal description discovery and invention	Universal discriptin data interface		
SOAP means_____	Single object access protocol	Simple object access protocol	single object accept protocol	Sevice Oriented application Protocol		
WSDL means_____	Web services Description Language	Web server descrption Language	web services descriptive language	Web service development life cycle		
_____such as a routine for calculating the integral square root of a number.	function	data	Business processes	Application		
_____such as fetching the quantity of a particular widget a vendor has on hand	function	data	business processes	Control		
_____ such as accepting an order for a widget,and sending an invoice	function	data	business processes	Application		
function such as _____ for calculating the integral square root of a number.	fecthing	routine	accepting	Exit		
Data such as _____ the quantity of a particular widget a vendor has on hand.	fecthing	routine	accepting	Put		
_____ is an XML based mechanism	SOAP	WSDL	UDDI	XML		
SOAP is an _____based mechanism	WML	XML	HTML	HTTP		
RPC means_____	Remote Procedure call	Remote procedural call	Reverse procedure call	Remote Protocol call		
_____is used for exchanging information between applications within a distributed environment.	WSDL	WML	SOAP	WSCL		

SOAP is used for exchanging information between ____within a distributed environment.	Functions	applications	procedures	DATA		
_____ is an information exchange mechanism	SOAP	WSDL	XML	HTML		
_____can be used to send messages between applications.	SOAP	WSDL	XML	SMTP		
WSDL is a _____application	client	server	service	WEB		
_____ files are typically stored in registries called UDDI.	SOAP	WSDL	XML	WSCL		
WSDL files are typically stored in registries called_____	pointer	registor	UDDI	WSDL		
The key enabler for web services is _____	WML	XML	HTML	SMTP		
CORBA means_____	Common Object Request Broker Architecture	Common Object response Broker Architecture	control object request Broker Architecture	Common object Request		
DCOM means_____	Document Component Object model	Data component Object Model	Directory component object model	documet object method		
_____defines a means to publish,discover information, about web services.	UDDL	UDDI	WSCL	WSDL		
UBR means_____	UDDL Business registry	UDDI Business registry	UDDL broker registry	UDDI Broker Register		
._____is a global implementation of the UDDI specification.	UDDL	UBR	UDR	UDDI		
_____are applications that are available via the World wide web its capabilities.	web applications	Wireless applications	system applications	Mobile Application		
Web applications are available via the _____and allow any user to access.	web server	Web browser	Web	Web Service		

HTML means _____	Hyper Text Markup language	Hyper Tool Markup language	Hyper Text Management Language	Hyper text markup level		
HTML and XML are_____readable language	Machine	human	robot	Compiler		
_____is for presentation markup	HTML	XML	WML	SMTP		
HTML is for _____ markup	Semanic	presentation	constructing	Designing		
_____ is for semanic markup	WML	HTML	XML	TCP		
XML is for _____markup	Semanic	Presentation	constructing	Designing		
_____enables any-to-any information exchange	HTML	XML	WML	UML		
_____is used for structuring data	HTML	XML	WML	HTTP		
XML is used for _____data	Structuring	Functioning	constructing	Coding		
_____data includes things like Spreadsheets, address books, configuration detail.	WML	HTML	XML	UDDI		
____a set of rules to design text formats to support developer in creating structured data.	WML	HTML	XML	IP		
The purpose of an XML document is to capture _____data.	Structured	Functioning	semi structured	Unstructured		
The purpose of an _____document is to capture structured data	WML	HTML	XML	API		
_____ are structured into number of elements	Documents	functions	semi structured	File		
Documents are structured into number of _____	Data	Functions	Structures	elements		
_____document can have a single route element	WML	HTML	XML	HEAD		
XML document can have _____root element.	Single	2	0	3		

_____ is extremely strict in its syntax.	HTML	XML	WML	URL		
____in OOP ‘slanguages allow developers to name classes unambiguously.	Functions	Namespc es	Structure	Header		
URI means _____	Uniform Resource Identifier	Unified resource identifier	Universal resource Identifier	Universal Resorce Intraction		
_____ is the union of _____ and URL	URN,UR I	URI,UR N	URI,URL	UML,UR L		
XML is to use a _____ as the namespace identifier	URI	URL	URN	HTTP		
URN means _____	Universal Resource name	Uniform resource name	unified resource name	United Resource identifier		
URL means_____	Universal Resource Locator	Uniform resource locator	unified resource locator	United Resource Locator		
XML schema provides a total of _____simple types	55	47	44	43		

Answer
eXtensible Markup Language
Return on Investment
XML
Web services Description Language
Web services
Quality-of-Service
Web service
Web service
web service

Universal description discovery and integration
Simple object access protocol
Web services Description Language
function
data
business processes
routine
fecthing
SOAP
XML
Remote Procedure call
SOAP

applications
SOAP
SOAP
client
WSDL
UDDI
XML
Common Object response Broker Architecture
Document Component Object model
UDDI
UDDI Business registry
UBR
web applications
Web

Hyper Text Markup language
human
XML
Semanic
HTML
Presentation
XML
XML
Structuring
XML
XML
Structured
XML
Documents
elements
XML
Single

XML
Namespces
Universal resource Identifier
URI,URN
URN
Uniform resource name
Uniform resource locator
44

UNIT-II

SYLLABUS

SOAP: SOAP Model – messages – Encoding – RPC – Alternative SOAP encodings – Document, RPC, Literal, Encoded – SOAP, Web Services and the REST Architecture. **WSDL:** Structure – Using SOAP and WSDL. UDDI- UDDI Business Registry – Specification – Data Structures – Life cycle Management – Dynamic Access Point Management.

SOAP

SOAP Model

- The Simple Object Access Protocol provides a model for distributed processing assuming that a SOAP message is originated at a particular SOAP sender and is ultimately received by an ultimate SOAP receiver, and in the path of the message there could be zero or many SOAP intermediaries.
- This processing model defines how any SOAP receiver should process the SOAP message and applies to a single message only isolating itself from any other.

- **SOAP Nodes**

  - A SOAP node is any node that participates in the SOAP message path. It can be the initial SOAP node, in which case it is a SOAP Sender, the ultimate SOAP receiver or a SOAP intermediary. Any SOAP node that receives a SOAP message must perform several processes according to the SOAP specification.

- **SOAP Roles**

  - While processing a SOAP message a SOAP node performs one or several SOAP roles, each of them are identified by a URI, also known as the SOAP role name.
  - SOAP is a way for a program running in one operating system to communicate with a program running in either the same or a different operating system, using HTTP (or any other transport protocol) and XML

What is SOAP?

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is for communication between applications
- SOAP is a format for sending messages
- SOAP communicates 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 recommendation

Why SOAP?

- It is important for application development to allow Internet communication between programs.
- Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.
- A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.
- SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

SOAP Message

SOAP Building Blocks

- A SOAP message is an ordinary XML document containing the following elements:

  - An Envelope element that identifies the XML document as a SOAP message
  - A Header element that contains header information
  - A Body element that contains call and response information
  - A Fault element containing errors and status information

- All the elements above are declared in the default namespace for the SOAP envelope and the default namespace for SOAP encoding and data types is:

Syntax Rules

- Here are some important syntax rules:

  - A SOAP message MUST be encoded using XML
  - A SOAP message MUST use the SOAP Envelope namespace
  - A SOAP message MUST use the SOAP Encoding namespace
  - A SOAP message must NOT contain a DTD reference
  - A SOAP message must NOT contain XML Processing Instructions

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

The SOAP Envelope Element

- The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

Example

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

The xmlns:soap Namespace

- Notice the xmlns:soap namespace in the example above. It should always have the value of: "http://www.w3.org/2001/12/soap-envelope".
- The namespace defines the Envelope as a SOAP Envelope.

- If a different namespace is used, the application generates an error and discards the message.

The encodingStyle Attribute

- The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements.
- A SOAP message has no default encoding.

Syntax

soap:encodingStyle="URI"

Example

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

The SOAP Header Element

- The optional SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message.
- If the Header element is present, it must be the first child element of the Envelope element.
- **Note:** All immediate child elements of the Header element must be namespace-qualified.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
```

```
...
...
</soap:Envelope>
```

- The example above contains a header with a "Trans" element, a "mustUnderstand" attribute with a value of 1, and a value of 234.
- SOAP defines three attributes in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). These attributes are: mustUnderstand, actor, and encoding Style.
- The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

The mustUnderstand Attribute

- The SOAP must understand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.
- If you add mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it will fail when processing the Header.

Syntax

```
soap:mustUnderstand="0|1"
```

Example

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The actor Attribute

- A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint, instead, it may be intended for one or more of the endpoints on the message path.
- The SOAP actor attribute is used to address the Header element to a specific endpoint.

- **Syntax**

  soap:actor="*URI*"

- **Example**

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
<m:Trans xmlns:m="http://www.w3schools.com/transaction/"
soap:actor="http://www.w3schools.com/appml/">234
</m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The encodingStyle Attribute

- The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements.
- A SOAP message has no default encoding.

- **Syntax**

soap:encodingStyle="*URI*"

REMOTE PROCEDURE CALLS (RPC)

- Remote procedure calls in SOAP are essentially client-server interactions over HTTP where the request and response comply with SOAP encoding rules. The Request-URI (Universal Resource Identifier) in HTTP is typically used at the server end to map to a class or an object, but this is not mandated by SOAP. Additionally, the HTTP header SOAPAction specifies the interface name (a URI) and the name of the method to

be called on the server.

- The SOAP message is an XML document whose root element, the Envelope, specifies the overall structure of the message, its intended recipient, and other attributes of the message. SOAP specifies a remote procedure call convention, which includes the representation and format to be used for calls and responses. A method call is modeled as a compound data element consisting of a sequence of fields (accessors), one for each parameter. A return structure consists of the return value as well as the out and in/out parameters. SOAP encoding rules specify the serialization for primitive and application-defined datatypes.
- Figures 1 and 2 show the request and response structure of a remote procedure call transported as an HTTP request carrying a SOAP payload.

Figure 1: Example of a SOAP request sent via HTTP.

```
POST /Temperature HTTP/1.1
Host: www.temperature-service.com
Content-Type: text/xml
Content-Length: 357
SOAPAction: "http://weather.org/query#GetTemperature"

< SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 < SOAP-ENV:Body>
  < m:GetTemperature xmlns:m="http://weather.org/query">
   < longitude>39W< /longitude>
   < latitude>62S< /latitude>
  < /m:GetTemperature>
 < /SOAP-ENV:Body>
< /SOAP-ENV:Envelope>
```

Figure 2: Example of a SOAP response received via HTTP.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 343

< SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 < SOAP-ENV:Body>
  <                                    m:GetTemperatureResponse
xmlns:m="http://weather.org/query">
```

```
< centigrade>28.4< /centigrade>
< /m:GetTemperatureResponse>
< /SOAP-ENV:Body>
< /SOAP-ENV:Envelope>
```

- SOAP allows hierarchically structured queries and responses, and specifies serialization of primitive string, numeric and date datatypes, and aggregates like arrays and vectors. Sparse arrays, and protocols for sending parts of them are also supported. New types may be defined using the <complexType> construct inside a schema definition.

- Overall, SOAP provides many advantages. Unfortunately, its universality comes with a performance penalty: XML messages are textual and so the sizes of its messages are significantly larger than protocols which send binary data. Since a distinguishing characteristic of scientific computation is the need to handle large data sets, the performance of SOAP relative to specialized protocols that can use binary representations is an important issue. The next section tests SOAP performance relative to other communication protocols.

REST

- Despite the name, Web service technology offers several advantages in non-Web environments. For example, Web service technology facilitates the integration of J2EE components with .NET components within an enterprise or department in a straightforward manner.
- But as shown, Web services can be implemented in Web environments, too, on top of basic Web technologies such as HTTP, Simple Mail Transfer Protocol (SMTP), and so on. Representational State Transfer (REST) is a specific architectural style introduced in . Simply put, it is the architecture of the Web.
- Consequently, the question arises about how Web services compare to the Web, or how the corresponding underlying architectural styles SOA and REST compare.

"Representational" in REST

- The basic concept of the REST architecture is that of a resource. A resource is any piece of information that you can identify uniquely.
- In REST, requesters and services exchange messages that typically contain both data and metadata. The data part of a message corresponds to the resource in a particular representation as described by the accompanying metadata (format), which might also contain additional processing directives. You can exchange a resource in multiple representations. Both communication partners might agree to a particular representation in advance.
- For example, the data of a message might be information about the current weather in New York City (the resource). The data might be rendered as an HTML document, and the language in which this document is encoded might be German. This makes up the

representation of the resource "current weather in New York City." The processing directives might indicate that the data should not be cached because it changes frequently.

"State Transfer" in REST

- Services in REST do not maintain the state of an interaction with a requester; that is, if an interaction requires a state, all states must be part of the messages exchanged. By being stateless, services increase their reliability by simplifying recovery processing. (A failed service can recover quickly.)
- Furthermore, scalability of such services is improved because the services do not need to persist state and do not consume memory, representing active interactions. Both reliability and scalability are required properties of the Web. By following the REST architectural style, you can meet these requirements.

REST Interface Structure

- REST assumes a simple interface to manipulate resources in a generic manner. This interface basically supports to create, retrieve, update, and delete (CRUD) method.
- The metadata of the corresponding messages contains the method name and the identifier of the resource that the method targets. Except for the retrieval method, the message includes a representation of the resource. Therefore, messages are self-describing.
- Identifier being included in the messages is fundamental in REST. It implies further benefits of this architectural style.
- For example, by making the identifier of the resource explicit, REST furnishes caching strategies at various levels and at proper intermediaries along the message path. An intermediary might determine that it has a valid copy of the target resource available at its side and can satisfy a retrieval request without passing the request further on. This contributes to the scalability of the overall environment.
- If you are familiar with HTTP and URIs, you will certainly recognize how REST maps onto these technologies.

REST and Web Services

- At its heart, the discussion of REST versus Web services revolves around the advantage and disadvantages of generic CRUD interfaces and custom-defined interfaces.
- Proponents of REST argue against Web service technology because custom-defined Web service interfaces do not automatically result in reliability and scalability of the implementing Web services or cache ability of results, as discussed earlier.
- For example, caching is prohibited mainly because neither identifiers of resources nor the semantics of operations are made explicit in messages that represent Web service operations. Consequently, an intermediary cannot determine the target resource of a request message and whether a request represents retrieval or an update of a resource. Thus, an intermediary cannot maintain its cache accordingly.
- Proponents of Web service technology argue against REST because quality of service is only rudimental addressed in REST. Scenarios in which SOA is applied require qualities

of services such as reliable transport of messages, transactional interactions, and selective encryption of parts of the data exchanged.

- Furthermore, a particular message exchange between a requester and a service might be carried out in SOA over many different transport protocols along its message path—with transport protocols not even supported by the Web. Thus, the tight coupling of the Web architecture to HTTP (and a few other transport protocols) prohibits meeting this kind of end-to-end qualities of service requirement.
- Metadata that corresponds to qualities of services cannot—in contrast to what REST assumes—be expected as metadata of the transport protocols along the whole message path. Therefore, this metadata must be part of the payload of the messages. This is exactly what Web service technology addresses from the outset, especially via the header architecture of SOAP.
- From an architectural perspective, it is not "either REST or Web services." Both technologies have areas of applicability. As a rule of thumb, REST is preferable in problem domains that are query intense or that require exchange of large grain chunks of data. SOA in general and Web service technology as described in this book in particular is preferable in areas that require asynchrony and various qualities of services. Finally, SOA-based custom interfaces enable straightforward creation of new services based on choreography.
- You can even mix both architectural styles in a pure Web environment. For example, you can use a regular HTTP GET request to solicit a SOAP representation of a resource that the URL identifies and specifies in the HTTP message. In that manner, benefits from both approaches are combined. The combination allows use of the SOAP header architecture in the response message to built-in quality of service that HTTP does not support (such as partial encryption of the response). It also supports the benefits of REST, such as caching the SOAP response.
- Representational State Transfer (REST) is an architectural style that abstracts the architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. REST has emerged as a predominant web API design model.

Scope of Applicability of SOA and Web Service

As indicated throughout the first three chapters of this book, Web service technology provides a uniform usage model for components/services, especially within the context of heterogeneous distributed environments. Web service technology also virtualizes resources (that is, components that are software artifacts or hardware artifacts). Both are achieved by shielding idiosyncrasies of the different environments that host those components. This shielding can occur by dynamically selecting and binding those components and by hiding the communication details to properly access those components.

Furthermore, interactions between a requester and a service might show configurable qualities of service, such as reliable message transport, transaction protection, message-level security, and so

on. These qualities of services are not just ensured between two participants but between any number of participants in heterogeneous environments.

Given this focus, the question about the scope of applicability of SOA in general and Web service technology in particular is justified. As with most architectural questions, there is no crisp answer, no hard or fast rule to apply. Given this, common sense should prevail. For example:

SOA is not cost effective for organizations that have small application portfolios or those whose new interface requirements are not enough to benefit from SOA.

SOA does not benefit organizations that have relatively static application portfolios that are already fully interfaced.

If integration of components within heterogeneous environments or dynamically changing component configurations is at the core of the problem being addressed, consider SOA and Web service technology. SOA offers potentially significant benefits to organizations with large application portfolios that undergo frequent change (lots of mergers and acquisitions or frequent switching of service providers).

If reusability of a function (in the sense of making it available to all kinds of requesters) is important, providing the function as a Web service is a good approach.

Currently, the XML footprint and parsing cost at both ends of a message exchange does take up time and resources. If high performance is the most important criterion for primary implementation, consider the use of Web service technology with care. Use of binary XML for interchange might help this, but currently there are no agreed-upon standards for this.

Similarly, if the problem in hand is within a homogeneous environment, and interoperation with other external environments is not an issue, Web service technology might not have significant benefit.

WSDL

What is WSDL?

- 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.

The WSDL Document Structure

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

Element	Defines
<types>	The data types used by the web service
<message>	The messages used by the web service
<portType>	The operations performed by the web service
<binding>	The communication protocols used by the web service

- The main structure of a WSDL document looks like this:

```
<definitions>

<types>
  definition of types.....
</types>

<message>
  definition of a message....
</message>

<portType>
  definition of a port.....
</portType>

<binding>
  definition of a binding....
</binding>

</definitions>
```

- A WSDL document can also contain other elements, like extension elements, and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

WSDL Ports

- The **<portType>** element is the most important WSDL element.
- It describes a web service, the operations that can be performed, and the messages that are involved.
- The <portType> element can be compared to a function library (or a module, or a class) in a traditional programming language.

WSDL Messages

- The **<message>** element defines the data elements of an operation.
- Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

WSDL Types

- The **<types>** element defines the data types that are used by the web service.
- For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

WSDL Bindings

- The **<binding>** element defines the message format and protocol details for each port.

WSDL Example

- This is a simplified fraction of a WSDL document:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
```

```
</portType>
```

- In this example the **<portType>** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.
- The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".
- The **<message>** elements define the **parts** of each message and the associated data types.
- Compared to traditional programming, glossaryTerms is a function library, "getTerm" is a function with "getTermRequest" as the input parameter, and getTermResponse as the return parameter.

WSDL Ports

- The **<portType>** element is the most important WSDL element.
- It defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.
- The port defines the connection point to a web service. It can be compared to a function library (or a module, or a class) in a traditional programming language. Each operation can be compared to a function in a traditional programming language.

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

One-Way Operation

- A one-way operation example:

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
```

```
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

- In the example above, the port "glossaryTerms" defines a one-way operation called "setTerm".
- The "setTerm" operation allows input of new glossary terms messages using a "newTermValues" message with the input parameters "term" and "value". However, no output is defined for the operation.

Request-Response Operation

A request-response operation example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

In the example above, the port "glossaryTerms" defines a request-response operation called "getTerm".

The "getTerm" operation requires an input message called "getTermRequest" with a parameter called "term", and will return an output message called "getTermResponse" with a parameter called "value".

Binding to SOAP

- A request-response operation example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

- The **binding** element has two attributes - name and type.
- The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "glossaryTerms" port.
- The **soap:binding** element has two attributes - style and transport.
- The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use. In this case we use HTTP.
- The **operation** element defines each operation that the port exposes.
- For each operation the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded. In this case we use "literal".

UDDI

What is UDDI

- UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet.

  - UDDI stands for Universal Description, Discovery and Integration
  - UDDI is a directory for storing information about web services
  - UDDI is a directory of web service interfaces described by WSDL
  - UDDI communicates via SOAP
  - UDDI is built into the Microsoft .NET platform

What is UDDI Based On?

- UDDI uses World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) Internet standards such as XML, HTTP, and DNS protocols.
- UDDI uses WSDL to describe interfaces to web services
- Additionally, cross platform programming features are addressed by adopting SOAP, known as XML Protocol messaging specifications found at the W3C Web site.

UDDI Benefits

- Any industry or businesses of all sizes can benefit from UDDI.
- Before UDDI, there was no Internet standard for businesses to reach their customers and partners with information about their products and services. Nor was there a method of how to integrate into each other's systems and processes.
- Problems the UDDI specification can help to solve:

  - Making it possible to discover the right business from the millions currently online
  - Defining how to enable commerce once the preferred business is discovered
  - Reaching new customers and increasing access to current customers
  - Expanding offerings and extending market reach
  - Solving customer-driven need to remove barriers to allow for rapid participation in the global Internet economy
  - Describing services and business processes programmatically in a single, open, and secure environment

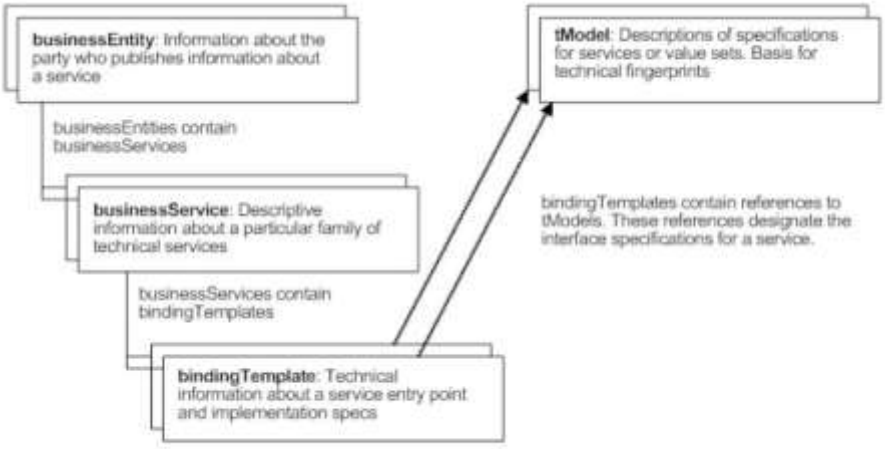
How can UDDI be Used

- If the industry published an UDDI standard for flight rate checking and reservation, airlines could register their services into an UDDI directory. Travel agencies could then search the UDDI directory to find the airline's reservation interface. When the interface is

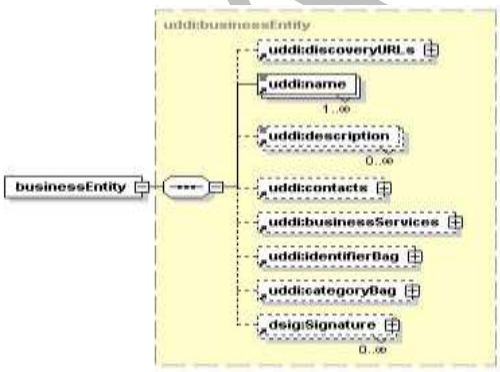
found, the travel agency can communicate with the service immediately because it uses a well-defined reservation interface.

UDDI BUSINESS REGISTRY

Relationship of UDDI Core Data Types



businessEntity fields



<identifierBag>

<keyedReference

tModelKey="uddi:uddi.org:ubr:identifier:dnb.com:d-u-n-s"

keyName="SAP AG"

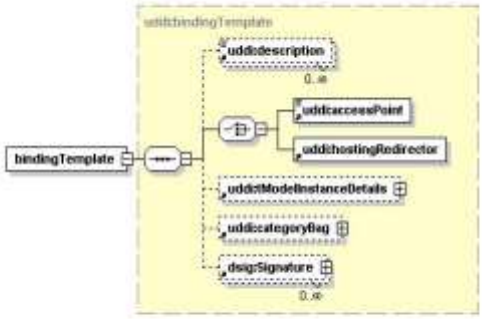
keyValue="31-626-8655" />

</identifierBag

Business Service

- Each business Service structure represents a logical grouping of Web services. At the service level, there is still no technical information provided about those services; rather, this structure allows the ability to assemble a set of services under a common rubric. Each businessService is the logical child of a single businessEntity. Each businessService contains descriptiveinformation – again, names, descriptions and classification
- information -- outlining the purpose of the individual Web services found within it. For example, a businessService structure could contain a set of Purchase Order Web services (submission, confirmation and notification) that are provided by a business.

BindingTemplate fields



SPECIFICATION

- The UDDI project also defines a set of XML Schema definitions that describe the data formats used by the various specification APIs. These documents are all available for download at www.uddi.org. The current version of all specification groups is Version 2.0.
- The specifications include the following −
    - UDDI Replication,

- UDDI Operators,
- UDDI Programmer's API, and
- UDDI Data Structures

UDDI Replication

- This document describes the data replication processes and interfaces to which a registry operator must conform to achieve data replication between sites. This specification is not a programmer's API; it defines the replication mechanism used among UBR nodes.

UDDI Operators

- This document outlines the behavior and operational parameters required by the UDDI node operators. This specification defines data management requirements to which operators must adhere.

UDDI Programmer's API

- This specification defines a set of functions that all UDDI registries support for inquiring about services hosted in a registry and for publishing information about a business or a service to a registry. This specification defines a series of SOAP messages containing XML documents that a UDDI registry accepts, parses, and responds to. This specification, along with the UDDI XML API schema and the UDDI Data Structure specification, makes up a complete programming interface to a UDDI registry.

UDDI Data Structures

- This specification covers the specifics of the XML structures contained within the SOAP messages defined by the UDDI Programmer's API. This specification defines five core data structures and their relationships with one another.
- The UDDI XML API schema is not contained in a specification; rather, it is stored as an XML Schema document that defines the structure and datatypes of the UDDI data structures.
- UDDI includes an XML Schema that describes the following data structures −
  - businessEntity
  - businessService
  - bindingTemplate
  - tModel
  - publisherAssertion

businessEntity Data Structure

- The business entity structure represents the provider of web services. Within the UDDI registry, this structure contains information about the company itself, including contact information, industry categories, business identifiers, and a list of services provided.

- Here is an example of a fictitious business's UDDI registry entry −

```
<businessEntity businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-
70E212685A40"
  operator = "http://www.ibm.com" authorizedName = "John Doe">
  <name>Acme Company</name>
  <description>
    We create cool Web services
  </description>

  <contacts>
    <contact useType = "general info">
      <description>General Information</description>
      <personName>John Doe</personName>
      <phone>(123) 123-1234</phone>
      <email>jdoe@acme.com</email>
    </contact>
  </contacts>

  <businessServices>
    ...
  </businessServices>
  <identifierBag>
    <keyedReference tModelKey = "UUID:8609C81E-EE1F-4D5A-B202-
3EB13AD01823"
      name = "D-U-N-S" value = "123456789" />
  </identifierBag>

  <categoryBag>
```

```
        <keyedReference tModelKey = "UUID:C0B9FE13-179F-413D-8A5B-
      5004DB8E5BB2"
          name = "NAICS" value = "111336" />
      </categoryBag>
    </businessEntity>
```

businessService Data Structure

- The business service structure represents an individual web service provided by the business entity. Its description includes information on how to bind to the web service, what type of web service it is, and what taxonomical categories it belongs to.

- Here is an example of a business service structure for the Hello World web service.

```
<businessService serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
  businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
  <name>Hello World Web Service</name>
  <description>A friendly Web service</description>
  <bindingTemplates>
    ...
  </bindingTemplates>
  <categoryBag />
</businessService>
```

Notice the use of the Universally Unique Identifiers (UUIDs) in the businessKey and serviceKey attributes. Every business entity and business service is uniquely identified in all the UDDI registries through the UUID assigned by the registry when the information is first entered.

bindingTemplate Data Structure

- Binding templates are the technical descriptions of the web services represented by the business service structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the web service.
- Here is an example of a binding template for Hello World.

```
<bindingTemplate serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
  bindingKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
```

```
<description>Hello World SOAP Binding</description>
<accessPoint URLType = "http">http://localhost:8080</accessPoint>

<tModelInstanceDetails>
  <tModelInstanceInfo tModelKey = "uuid:EB1B645F-CF2F-491f-811A-
4868705F5904">
    <instanceDetails>
      <overviewDoc>
        <description>
          references the description of the WSDL service definition
        </description>

        <overviewURL>
          http://localhost/helloworld.wsdl
        </overviewURL>
      </overviewDoc>
    </instanceDetails>
  </tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>
```

As a business service may have multiple binding templates, the service may specify different implementations of the same service, each bound to a different set of protocols or a different network address.

tModel Data Structure

- tModel is the last core data type, but potentially the most difficult to grasp. tModel stands for technical model.

- tModel is a way of describing the various business, service, and template structures stored within the UDDI registry. Any abstract concept can be registered within the UDDI as a tModel. For instance, if you define a new WSDL port type, you can define a tModel that represents that port type within the UDDI. Then, you can specify that a given business service implements that port type by associating the tModel with one of that business service's binding templates. Here is an example of a tModel representing the Hello World Interface port type.

```
<tModel tModelKey = "uuid:xyz987..." operator = "http://www.ibm.com"
authorizedName = "John Doe">
<name>HelloWorldInterface Port Type</name>
<description>
An interface for a friendly Web service
</description>

<overviewDoc>
<overviewURL>
http://localhost/helloworld.wsdl
</overviewURL>
</overviewDoc>
</tModel>
```

publisherAssertion Data Structure

- This is a relationship structure putting into association two or more businessEntity structures according to a specific type of relationship, such as subsidiary or department.
- The publisherAssertion structure consists of the three elements: fromKey (the first businessKey), toKey (the second businessKey), and keyedReference.
- The keyedReference designates the asserted relationship type in terms of a keyName keyValue pair within a tModel, uniquely referenced by a tModelKey.

```
<element name = "publisherAssertion" type = "uddi:publisherAssertion" />
<complexType name = "publisherAssertion">
 <sequence>
   <element ref = "uddi:fromKey" />
   <element ref = "uddi:toKey" />
   <element ref = "uddi:keyedReference" />
 </sequence>
                                    </complexType>
```

POSSIBLE QUESTIONS

PART – A

(Online Exam 20*1=20 marks)

- **Multiple Choice Questions available in Moodle**

POSSIBLE QUESTIONS

PART – B

(Each Question carries 6 marks)

1. Explain about SOAP Model
2. Explain the process of SOAP Messages
3. Explain the process of REST Architecture
4. Write in detail: (i) Encoding (ii) RPC
5. Explain the Structure of WSDL
6. Explain about UDDI and its process
7. Discuss in detail about UDDI Business Registry
8. Explain the Operation types of WSDL
9. Discuss in detail about UDDI Data Structures
10. Differentiate between SOAP and WSDL

PART – C

(Compulsory Question carries 10 marks)

1. Explain how WSDL describes Web Services.
2. Explain about SOAP Envelope Element
3. Explain about publisher Assertion Data Structure
4. Write a program to demonstrate SOAP Binding
5. Write a program to simplified fraction of a WSDL document

Questions	Opt1	Opt2	Opt3	Opt4
SOAP is an_____ based protocol	JAVA	XML	HTTP	FTP
SOAP is a format for sending messages and is called as _____	communication protocol	linkinkg p	carrier pro	concatination protocol
_____ is the directory for strong information about web services	UDDI	FTTP	SMTP	FTP
SOAP is a _____	carrier	language	protocol	markup
WSdL is a_____	web service description language	web servic	web servic	web service dynamic leader
WSdL is pronounced as_____	wis-dell	wis-doll	wizz-dull	wiz-dell
_____ is an extenstion of WSDL	x link	x lang	x query	x path
WSDL is not a_____	ibm standard	microsoft	www stan	w3c standa
The first working draft of WSDL 1.2 was released by WBC in	2002 june	2002 july	2003 july	2012 july
The binding element has _____ attributes	one	two	three	four
which of these are WSDL operator types?	one way	error mess	request res	solicst resp
The SOAP binding element has two attributes they are	style	font	style and t	font and bu
UDDI stands for _____	integration destination	universal description discover and integration	universal description discover and inspection	universal description discover and intent
BPEL standards for_____	business process excecution language	business process equation language	business process enabling language	business process eqalling language
_____is used to model behaviourof the both executive and abstract classes	BHEL	BPPE	BPL	BPEL

_____document specifies the location of service	WSDL	WWW	WWWW	WSCL
_____ defines the message format & protocol design for webservice	WSDL DESINGING	WSDL bir	WSDL tat	WSDL tag
Intial SOAP is also called as _____	maker	finder	orginator	intimater
SOAP _____ is a node that receives , accepts the messages passed by user	entertainer	receiver	sender	linker
SOAP _____is between SOAP sender and SOAP receiver	diary	intermedit	involency	inspecting
SOAP messages can be attached with _____ extensions	mime	fine	hype	rime
_____is a node that transmits messages received by receiver	SOAP caster	SOAP sen	SOAP rea	SOAP play
SOAP objects are _____ and hard to maintain	statefull	stable less	state less	stunning
SOAP identify the objects other than _____end point	FTP	HTP	CMTP	URL
the data that is exchanged between client and server is in XML format and_____	canal passes	river passe	SOAP pas	calibre pas
SOM stands for _____	syndicate deployment model	service de	service de	service deployment model
SOAP is a_____	connecting protocol	concept pr	converse p	communication protocol
SOAP uses _____channel to transport	FTP	HTTP	HPR	HDR
_____ and transport layers of a network are used by SOAP	application	integration	immigrati	inteligence
_____ is a light weight protocol	TOMB	SOAP	SOAR	SOME

SOAP protocol posses only _____ fundamental properties	3	2	1	5
SOAP is based on_____	ruling	sealing	message e	message converge nce
Application and _____ layers of network is used by SOAP	transport	intelligent	integratior	vibration
A SOAP _____ contains block of information or how messages is processed	<header>	<footer>	<middle>	<body>
The <body> elements contains either application specified data or _____	documents	links	images	fault messa
SOAP supports the documents and _____	APC	LLC	RPC	IC
RPC is abbreviated as _____	random proven connection	remote pro	remote po	rival point connectio n
The _____ elements reflects no explicit XML structure	<body>	<tag>	<title>	<header>
SOAP provides model for handling _____	famaliness	conveigen	scalablity	fault arise
fault information is placed in SOAP's _____	<body>	<header>	<title>	<footer>
SOAP uses same error and _____	stubs	structure	unions	status code
In SOAP serialization is done by_____and not by the reference	data	language	images	value
_____ is an XML based standard for describing web services	UDDI	FTM	FTP	HTP
UDDI Is a platform independent _____	opening	frameset	websites	open frame
UDDI is an _____ university	open	closed	commited	controlled

_____ and IBM launched the first UDDI open sites	microsoft	nokia	samsung	intel
UDDI version 2.0 announced in the year _____	2001	2012	1993	1998
currently UDDI is sponsered by _____	sun	opera	intel	oasis
Three elements of UDDI are_____, yellow, green pages	pink	blue	black	white
white pages contain _____ information	basic	highlevel	average	waste
green pages contain _____ information about webservices	basic	random	technical	risky
UBR is abbreviated as_____	under black reaction	ultimate bouncer racing	UDDI business registry	UDDI business reaction
_____ is an XML format for describing network service	www	wsll	wsdl	wwe
WSDL is written in _____	XML	C	C++	phython
_____ defines protocol and data format for each port type	<binding>	<tagline>	<carrying>	<message>
_____ describes the operation that can be performed and message involved	<port link>	<port tag>	<port type	<port struc
_____ defines the data types used by web service	<types>	<messages	<tag>	<binding>
_____defines the data element for each operation	<types>	<messages	<tag>	<binding>
The _____ operation has an inputed messages called "get term request"	"get it"	"get term"	"get"	"gets'
_____ response can be sent as request and will wait for response	solicit	stable	suthing	simple

Opt5	Opt6	Answer
		JAVA
		communication protocol
		UDDI
		protocol
		web service description language
		wizz-dull
		x lang
ard		w3c standard
		2002 july
		two
ponse		error messages
ulet		style and transport
		universal descriptio n discover and integratio n
		business process equation language
		BPEL

		WWW
;		WSDL binding
		orginator
		sender
		intermeditary
		mime
yer		SOAP sender
		stable less
		URL
sses		SOAP passes
		service development model
		communication protocol
		HTTP
e		application
		SOAP

		2
		message exchnges
		transport
		<header>
ages		fault messages
		RPC
		remote procedure call
		<body>
		fault arise
		<body>
es		status codes
		value
		UDDI
e work		open frame work
		open

		microsoft
		2012
		oasis
		white
		basic
		technical
		UDDI business registry
		wsdl
		XML
›		<binding>
:t>		<port type>
		<types>
		<messages>
		"get term"
		solicit

UNIT-III

SYLLABUS

Advanced Web Services Technologies and Standards: Conversation – Overview – Web Services Conversation Language – WSCL Interface Components- Workflow-Business Process Management – Workflow and Workflow Management systems – BPEL. Transaction –ACID transaction – Distributed Transaction – OASIS Business Transaction Protocol.

ADVANCED WEB SERVICES TECHNOLOGIES AND STANDARDS

CONVERSATION

Each conversation definition has the root element Conversation.

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation" version="1.01"
        xmlns="http://www.w3.org/2002/02/wscl10"
        initialInteraction="Start"  finalInteraction="End"
        targetNamespace="http://example.com/conversations/StoreFront101"
        hrefSchema="http://example.com/schema_files/StoreFront101.wscl"
        description="Conversation for a Store Front Service" >
 <ConversationInteractions>
        list of all the interactions
 </ConversationInteractions>
 <ConversationTransitions>
        list of all the transitions
 </ConversationTransitions>
</Conversation>
```

The Conversation element contains the following two sub-elements:

ConversationInteractions lists all Interaction elements.

ConversationTransitions lists all Transition elements.

The Conversation element contains the following attributes:

initialInteraction and finalInteraction reference the initial and final interactions of the conversation.

name is the name of a conversation. The name is the shared piece of information needed by both parties so they realize the same conversation type in their service implementation. This name would also appear in the header elements of the actual messages exchanged. WSCL does not specify how to name conversations. Conversation names do not have to be URIs. If a conversation is published in a UDDI directory, the conversation name could be a tModel key.

version (optional) is the version of the conversation. If no version number is given, the name of the conversation must be unique.

targetNamespace (optional) is the namespace of this conversation as it should be used when elements of this conversation are referenced in other XML documents.

hrefSchema (optional) is the URL of the file containing this conversation definition.

description (optional) is the textual description of the conversation.

OVERVIEW

WEB SERVICES CONVERSATION LANGUAGE

- Electronic commerce is moving toward a vision in which corporate enterprises use Web services to interact with each other based on well-defined standards in a dynamic and loosely coupled way. To use Web services effectively, the interacting parties need to know and agree on the following:

  - Business payload – Both parties need to know which information to exchange.
  - Protocol – Both parties need to know how to exchange business payload. They must agree on the message structure, message header information, mapping to underlying transfer protocols, and overall framework for communication and error handling. Protocols for Web services include SOAP-RPC , asynchronous SOAP with specific profiles, RNIF, and ebXML TR&P.

- Service location – To interact with a specific service, both parties need to know which protocols the service supports, which payload it exchanges, and its location; for example, an HTTP URL.

Elements of a WSCL Specification

- The complete schema for WSCL is listed in Appendix A. There are four main elements to a WSCL specification, as depicted in the UML model shown in Figure 3.
- Document type descriptions specify the types (schemas) of XML documents the service can accept and transmit in the course of a conversation. The schemas of the documents exchanged are not specified as part of the WSCL specification document; the actual document schemas are separate XML documents referenced by their URL in the XMLDocumentType elements of the conversation specification.
- Interactions model the actions of the conversation as document exchanges between two participants. WSCL supports five types of interactions: Send (the service sends out an outbound document); Receive (the service receives an inbound document); SendReceive (the service sends out an outbound document and then expects to receive an inbound document in reply); ReceiveSend (the service receives an inbound document and then sends out an outbound document); Empty does not contain any documents exchanged, but is used only for modeling the start and end of a conversation.

Transitions specify the ordering relationships between interactions. A transition specifies a source interaction, a destination interaction, and, optionally, a document type of the source interaction as an additional condition for the transition.

Conversations list all the interactions and transitions that make up the conversation. A conversation contains additional information about the conversation, including the conversation's name and the interaction the conversation can start with and end with. Conversations can be thought of as interfaces or public processes supported by a service. They differ from interfaces as defined by CORBA IDE or Java interfaces because they also specify the possible ordering of operations, i.e. the possible sequences in which documents may be exchanged.

Document Types

The interaction between service-consumer and service-provider is achieved through XML document exchange. A conversation definition language must be able to define all the input and output document types. The attribute hrefSchema of the elements InboundXMLDocument and OutboundXMLDocument refers to the schema to which the document corresponds. WSCL supports only XML schema specifications of payload because schemas seem to be the prevailing means of describing data exchanged on the Internet. Existing DTD specifications can be translated easily into XML schemas. WSCL, a very simple and basic conversation definition language, does not directly support the specification of non-XML payload-like binary attachments. Yet the attribute hrefSchema is optional, which allows the schema definition to be omitted in case of binary payload.

InboundXMLDocument and OutboundXMLDocument also serve to declare an ID for the document that can be used within the rest of the conversation definition. In the following example, the definition defines an input document that conforms to a purchase order schema defined in PurchaseOrderRQ.xsd.

<InboundXMLDocument

hrefSchema="http://foo.org/PurchaseOrderRQ.xsd"

id="PurchaseOrderRQ"

</InboundXMLDocument>

- In a WSCL conversation definition, the document types are declared within the interaction definitions as either an InboundXMLDocument or OutboundXMLDocument, depending on whether the document is expected as input or is produced as output in the interaction.
- WSCL only references the schemas for the business payload of any messages. Of course, the business documents will be complemented by XML message header information for the exchange over the Internet. These message header schemas, however, are defined by the protocol binding and should not appear in the business documents referenced by the document types.

INTERACTIONS

- An interaction is an exchange of one or two documents between a service and its client. WSCL only models business level interactions. It only specifies which business level documents are exchanged and does not model how this exchange is carried out by lower-level messaging protocols. The actual messaging may involve more than one message per business document exchange; for example, an HTTP-POST as well as an HTTP-RESPONSE per business document. The messaging logic may even introduce additional transport level acknowledge messages; for example, to achieve reliable messaging.
- The following interaction types are currently defined in WSCL: Receive and Send (one-way interactions), ReceiveSend and SendReceive (two-way interactions), and Empty.

One-Way Interactions

- One-way interactions represent a single one-way message being sent or received by a participant. There are two sub-types of one-way interactions: Receive and Send. Send represents a document sent out by a participant. Receive represents a document received by a participant.
- The following example represents a Receive interaction that receives a login document:
- <Interaction interactionType="Receive" id="LoginRegInput">
- <InboundXMLDocument id="LoginRequestData"
- hrefSchema=http://foo.org/LoginRequestData.xsd />
- </Interaction>

- A Receive interaction must contain one InboundXMLDocument element. A Send interaction must contain one OutboundXMLDocument element.

Two-Way Interactions

- Two-way interactions can be either SendReceive or ReceiveSend, depending on whether the participant sends out a message for which it gets a response (SendReceive) or responds to a request that it receives (ReceiveSend).
- SendReceive: Each SendReceive interaction is the logical unit of sending a request and then receiving a response. The interaction is not complete until the response has been exchanged.

```
<Interaction interactionType="SendReceive" id="Payment">
  <OutboundXMLDocument id="Invoice"
        hrefSchema="http://foo.org/InvoiceRS.xsd">
  </OutboundXMLDocument>
  <InboundXMLDocument id="Payment"
        hrefSchema="http://foo.org/Payment.xsd">
  </InboundXMLDocument>
</Interaction>
```

ReceiveSend: Each ReceiveSend interaction is the logical unit of receiving a request and then returning a response. The interaction is not complete until the response has been sent.

```
<Interaction interactionType="ReceiveSend" id="Quotation">
  <InboundXMLDocument id="PurchaseOrderRQ"
        hrefSchema="http://foo.org/PurchaseOrderRQ.xsd">
  </InboundXMLDocument>
  <OutboundXMLDocument id="InvoiceRS"
        hrefSchema="http://foo.org/InvoiceRS.xsd">
  </OutboundXMLDocument>
```

</Interaction>

- The following example shows a Purchase interaction that specifies two additional outbound document types for the cases of invalid payment and out of stock.

```
<Interaction interactionType="ReceiveSend" id="Purchase" >
  <InboundXMLDocument hrefSchema="http://conv123.org/PurchaseOrderRQ.xsd"
        id="PurchaseOrderRQ" />
  <OutboundXMLDocument  id="PurchaseOrderAcceptedRS"
        hrefSchema="http://conv123.org/PurchaseOrderAcceptedRS.xsd" />
  <OutboundXMLDocument  id="InvalidPaymentRS"
        hrefSchema="http://conv123.org/InvalidPaymentRS.xsd" />
  <OutboundXMLDocument  id="OutOfStockRS"
        hrefSchema="http://conv123.org/OutOfStockRS.xsd" />
</Interaction>
```

Similarly, a SendReceive interaction can specify more than one InboundXMLDocument.

Transitions

- A conversation can proceed from one interaction to another as allowed by the permissible sequencing defined in the transition elements.

```
<Transition>
    <SourceInteraction href="Invoice"/>
    <DestinationInteraction href="Receipt"/>
    <SourceInteractionCondition href="InvoiceRS"/>
</Transition>
```

- SourceInteraction references an interaction that can precede the DestinationInteraction when the conversation is executed. Similarly, DestinationInteraction references one of the

interactions that can follow the SourceInteraction when the conversation is executed. Together, all transitions specify all possible sequences of the interactions.

- SourceInteractionCondition is an additional constraint on the transition. It is needed when the Source Interaction specifies more than one possible document to be exchanged and the type of document exchanged has an influence on the possible next interactions. Source Interaction Condition references an OutboundXMLDocument of the SourceInteraction when it is a Receive Send interaction, or an InboundXMLDocument from the SourceInteraction when it is a Send Receive interaction. If no SourceInteractionCondition is listed, Destination Interaction can be triggered independent of the types of documents exchanged in Source Interaction. There is one important limitation to specifying transitions without Source Interaction Condition: If there exists a transition from Source Interaction A to Destination Interaction B that specifies a SourceInteraction Condition, then it is not possible to also specify a transition from SourceInteraction A to Destination Interaction B without a Source Interaction Condition.
- Transitions define all the permissible orders of interactions but do not specify under which condition which permissible sequence is chosen. This is determined by the internal application logic of both participants, often based on back-end information; for example, in the example in Appendix B, it is the buyer who decides whether the CatalogInquiry interaction is followed by a Quote interaction or a Purchase interaction, by sending either a PurchaseRQ or QuoteRQ document. A PurchaseRQ document triggers the Purchase interaction; a QuoteRQ document triggers the Quote interaction.

Initial and Final Interactions

- Part of defining the possible ordering of interactions is the specification of the first and last interactions of a conversation. In WSCL, this is done by the attributes initialInteraction and finalInteraction of the Conversation element. The attribute initialInteraction references the ID of the first interaction to be executed in the conversation. The attribute finalInteraction references the ID of the last interaction to be executed.

<Conversation name="ExampleConversation"

initialInteraction = "Login"

finalInteraction = "Purchase" >

- Of course, there might be more than one interaction with which the conversation can start or end. In the example shown in Figure 2 and Appendix B, the conversation can end after a CatalogInquiry interaction, a Quote interaction, a Purchase interaction (in case the purchase cannot go through), or the Shipping interaction. In addition, the conversation can start with either a Registration or a Login interaction, depending on whether or not the client is already a registered user. To specify several possible start and end interactions, interactions of type Empty are used. In Figure 2, an empty interaction Start is added, plus transitions from Start to Registration and Login. The attribute initialInteraction references

this empty interaction Start. An empty interaction, End, is introduced with various transitions to it, and is referenced in the attribute finalInteraction.

- An interactionType Empty is an interaction in which no documents are exchanged. Its specification does not contain the sub-elements InboundXMLDocument and OutboundXMLDocument. Currently, the only situation in which empty interactions can be used in a conversation definition is when several possible final or initial interactions need to be modeled. The conversation example in Appendix B shows empty interactions.
- Where WSDL and WSCL overlap, we can map the different terminology used as follows:

WSDL	WSCL
Port Type	Conversation
Operation: - One-way - Request-response - Solicit-response - Notification	Interaction*: - Receive - ReceiveSend - SendReceive - Send
Input	InboundXMLDocument
Output, Fault	OutboundXMLDocument
Names of Operation, Input, Output, Fault	ID of Interaction, InboundXMLDocument, OutboundXMLDocument
Message	URL of XML schema (WSCL delegates the specification of the payload entirely to an external XML schema, whereas WSDL directly uses XML data types)

*Empty does not appear in this list because it is used only for modeling the start and end state of conversations, and does not contain any documents exchanged.

- There are three approaches for combining WSDL and WSCL descriptions of a Web service:
- Adding protocol bindings in WSDL to a conversation described in WSCL: If the abstract interface is already completely described by a WSCL document, we can use the WSDL Binding elements to describe the protocol binding; for example:

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

<definitions name="MyStoreFrontService"

    xmlns="http://schemas.xmlsoap.org/wsdl/"

    targetNamespace=...

    xmlns:conv="http://example.com/Conversations/StoreFront.cdl" >
```

```
<binding name="StoreFrontServiceConversationBinding"
        type="conv:StoreFrontServiceConversation">
  <soap:binding style="document"/>
  <operation name="conv:Login">
    <soap:operation soapAction="Login">
  </operation>
  <operation name="conv:Registration">
    <soap:operation soapAction="Registration">
  </operation>
  ... for all other interactions
</binding>

<service name="MyStoreFrontService">
  <port name="MyStoreFrontServiceAccessPoint"
        binding="StoreFrontServiceConversationBinding">
       <soap:Address location="http://mystore.com/storefront" />
  </port>
</service>
</definitions>
```

In the example, the attribute type in the element binding refers to the name of the conversation in the WSCL document describing this conversation. The names of the operations refer to the IDs of the interactions.

- Adding choreography to a WSDL port type description: If a port type is already described by a WSDL document, we can describe the choreography in an additional WSCL document that only contains transition elements. The sub-elements SourceInteraction and

DestinationInteraction refer to the names of operations from the WSDL document. The sub-element SourceInteractionCondition refers to an output or fault message of the operation. WSCL uses attributes of type HREF to refer to other elements. Therefore, to refer to operations from a WSDL document, the WSCL schema needs to be slightly adapted to also accept values of type QNAME.

- Providing a full WSDL and WSCL description for the same port type/conversation: We can also express which documents get exchanged by both a WSDL port type and a WSCL conversation with the conversation also describing the choreography used. The following example describes the interaction "Login" from the WSCL example in Appendix B as a WSDL operation:

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

<definitions name="MyStoreFrontService"

        xmlns="http://schemas.xmlsoap.org/wsdl/"

        targetNamespace=...

        xmlns:xsd1="http://example.com/types/LoginRQ.xsd"

        xmlns:xsd2="http://example.com/types/ValidLoginRS.xsd"

        xmlns:xsd3="http://example.com/types/InvalidLoginRS.xsd" >

 <message name="LoginRequestDocument">

    <part name="body" element="xsd1:LoginRQElement" />

 </message>

 <message name="ValidLoginDocument">

    <part name="body" element="xsd2:ValidLoginRSElement" />

 </message>

 <message name="InvalidLoginDocument">

    <part name="body" element="xsd3:InvalidLoginRSElement" />

 </message>
```

```
<portType name="StoreFrontServiceConversation" >
  <operation name="Login">
    <input name="LoginRQ" message="LoginRequestDocument"/>
    <output name="ValidLoginRQ" message="ValidLoginDocument"/>
    <fault name="InvalidLoginRQ" message="InvalidLoginDocument"/>
  </operation>
</portType>
</definitions>
```

Extending WSCL

- The WSCL specification contains the smallest possible set of elements and attributes to describe conversations. This set is sufficient to model many of the conversations needed for Web services. However, there are more complex B2B interactions that need additional capabilities from a conversation definition language. Such additional requirements include the following examples:
- Defining document types that have non-XML content; for example, binary attachments
- Explicit description of roles of participants
- Multi-party conversations with three or more participants or roles
- Expressing timeouts and other quality of service characteristics of individual interactions
- Expressing more complex SourceInteractionConditions; for example, listing several documents, excluding documents, or even referencing the content of documents
- Events, i.e. interactions that can occur at any time within a conversation instance
- Recursive conversations, aggregating conversations into larger conversations
- Sub-typing and extending existing conversation definitions

WORKFLOW AND WORKFLOW MANAGEMENT SYSTEMS

Introduction

- − Workflow & Web Service Composition
- – Workflow Management Systems

• Automatic Web Service Composition

- – Process definition language: BPEL

– Semantics & Semantic Web Services

– Process composer (Planner)

– Workflow Engine

Workflow – definition

- Workflow: Process that can be automated by invoking applications or external services and/or assigning manual tasks

• Workflow Composition: arranging activities to form a business process

– if invocation is limited to Web Services calls:

Workflow Composition

- Web Service Composition
- Web Service Composition in a broad sense:
- „the automatic selection, composition, and
- interoperation of Web services to perform some
- task, given a high-level description of an
- objective“ [OWL-S: Semantic Markup for Languages]

– Web Service discovery

– Composition (in a narrow sense) of new Services

– Execution of new, composite Services

two different approaches:

1. low-level process modeling and execution languages (like WS-BPEL)

- directly executable in existing engines

- manual definition of new (composed) processes that interact with existing ones

- does not allow for automation

2. high-level unambigious description language for Web Services (like OWL-S)

- allows reasoning about web services

- automation of discovery and composition possible

Parts allowing WS Composition

Language for process definitions: BPEL

– must be supported by available workflow engines

2. Semantics to describe capabilities of WS and requested functionality

3. Process composer (Planner) which creates BPEL process definitions to fulfill a request

– based on semantic descriptions of

- request
- available atomic Web Services

4. Workflow Engine that can work with BPEL process definitions

Process definition language: Why not WSDL?

typical business interactions:

– sequences of peer-to-peer messages (synchronous and asynchronous)

– long-running, stateful

 protocol for message exchange needed

• WSDL

– based on stateless interaction model

– only synchronous (request/response) and uncorrelated asynchronous interactions

 specifies only method invocations (no order)

Process modeling language based on Web services

• BPEL4WS was originally developed by BEA, IBM,

Microsoft and (later joined) SAP, Siebel

– version 1.0 proposed in 2002

– current version: 1.1, 2.0 as draft

– based on the specifications: WSDL, XML Schema, XPath,

WS-Addressing (allows standardized addressing)

• Today OASIS is in charge of the standardization of

BPEL called WS-BPEL

Process definition language: BPEL

conceptual separation of

– abstract vs. executable process

 internal, executable process can be altered without changing the abstract process

• supporting two-level programming model

– „programming in the large“ abstract process vs.

– „programming in the small“ executable process

• common core of process descriptions concepts

– BPEL specification focused on common core

– extensions required for private, abstract processes

– BPEL

BPEL

- Web service interactions can be described in two ways: executable business processes and abstract business processes. Executable business processes model actual behavior of a participant in a business interaction. Abstract business processes are partially specified processes that are not intended to be executed. An Abstract Process may hide some of the required concrete operational details. Abstract Processes serve a descriptive role, with more than one possible use case, including observable behavior and/or process template. WS-BPEL is meant to be used to model the behavior of both Executable and Abstract Processes.
- WS-BPEL provides a language for the specification of Executable and Abstract business processes. By doing so, it extends the Web Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that

should facilitate the expansion of automated process integration both within and between businesses.

- The origins of BPEL can be traced to WSFL and XLANG. It is serialized in XML and aims to enable programming in the large. The concepts of programming in the large and programming in the small distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes.
- Programming in the large generally refers to the high-level state transition interactions of a process—BPEL refers to this concept as an Abstract Process. A BPEL Abstract Process represents a set of publicly observable behaviors in a standardized fashion. An Abstract Process includes information such as when to wait for messages, when to send messages, when to compensate for failed transactions, etc. Programming in the small, in contrast, deals with short-lived programmatic behavior, often executed as a single transaction and involving access to local logic and resources such as files, databases, etc. BPEL's development came out of the notion that programming in the large and programming in the small required different types of languages.

- IBM and Microsoft had each defined their own, fairly similar, "programming in the large" languages: WSFL and XLANG, respectively. With the advent and popularity of BPML, and the growing success of BPMI.org and the open BPMS movement led by JBoss and Intalio Inc., IBM and Microsoft decided to combine these languages into a new language, BPEL4WS. In April 2003, BEA Systems, IBM, Microsoft, SAP and Siebel Systems submitted BPEL4WS 1.1 to OASIS for standardization via the Web Services BPEL Technical Committee. Although BPEL4WS appeared as both a 1.0 and 1.1 version, the OASIS WS-BPEL technical committee voted on 14 September 2004 to name their spec "WS-BPEL 2.0". (This change in name aligned BPEL with other Web Service standard naming conventions which start with "WS-" and took account of the significant enhancements made between BPEL4WS 1.1 and WS-BPEL 2.0.) If not discussing a specific version, the moniker BPEL is commonly used[citation needed].
- In June 2007, Active Endpoints, Adobe Systems, BEA, IBM, Oracle and SAP published the BPEL4People and WS-HumanTask specifications, which describe how human interaction in BPEL processes can be implemented.
- There were ten original design goals associated with BPEL:
- Define business processes that interact with external entities through web service operations defined using WSDL 1.1, and that manifest themselves as Web services defined using WSDL 1.1. The interactions are “abstract” in the sense that the dependence is on portType definitions, not on port definitions.
- Define business processes using an XML-based language. Do not define a graphical representation of processes or provide any particular design methodology for processes.
- Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process. Such a business process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e. abstract view and executable view) will require a few specialized extensions, but these

extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.

- Provide both hierarchical and graph-like control regimes, and allow their use to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.
- Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.
- Support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be defined by partners and may change.
- Support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations such as "suspend" and "resume" may be added in future releases for enhanced lifecycle management.
- Define a long-running transaction model that is based on proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.
- Use Web Services as the model for process decomposition and assembly.
- Build on Web services standards (approved and proposed) as much as possible in The BPEL language
- BPEL is an orchestration language, not a choreography language. The primary difference between orchestration and choreography is executability and control. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. A choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing interoperability. Such a protocol is not directly executable, as it allows many different realizations (processes that comply with it). A choreography can be realized by writing an orchestration (e.g. in the form of a BPEL process) for each peer involved in it. The orchestration and the choreography distinctions are based on analogies: orchestration refers to the central control (by the conductor) of the behavior of a distributed system (the orchestra consisting of many players), while choreography refers to a distributed system (the dancing team) which operates according to rules (the choreography) but without centralized control.
- BPEL's focus on modern business processes, plus the histories of WSFL and XLANG, led BPEL to adopt web services as its external communication mechanism. Thus BPEL's messaging facilities depend on the use of the Web Services Description Language (WSDL) 1.1 to describe outgoing and incoming messages.
- In addition to providing facilities to enable sending and receiving messages, the BPEL programming language also supports:
- A property-based message correlation mechanism

BPEL syntax in XML

<process>

<partnerLinks> … </partnerLinks>

<partners> … </partners>

<variables> … </variables>

<correlationSets> … </correlationSets>

<faultHandler> … <faultHandler>

<compensationHandler> … </compensationHandler>

<eventHandler> … </eventHandler>

(activities)*

</process>

XML and WSDL typed variables

- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports XPath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

Relationship of BPEL to BPMN

- There is no standard graphical notation for WS-BPEL, as the OASIS technical committee decided this was out of scope. Some vendors have invented their own notations. These notations take advantage of the fact that most constructs in BPEL are block-structured (e.g. sequence, while, pick, scope, etc.) This feature enables a direct visual representation of BPEL process descriptions in the form of structograms, in a style reminiscent of a Nassi–Shneiderman diagram.
- Others have proposed to use a substantially different business process modeling language, namely Business Process Modeling Notation (BPMN), as a graphical front-end to capture BPEL process descriptions. As an illustration of the feasibility of this approach, the BPMN specification includes an informal and partial mapping from BPMN to BPEL 1.1. A more detailed mapping of BPMN to BPEL has been implemented in a number of tools, including an open-source tool known as BPMN2BPEL. However, the development of these tools has exposed fundamental differences between BPMN and BPEL, which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models. Even more difficult is the problem of BPMN-to-BPEL round-trip engineering: generating BPEL code from BPMN diagrams and maintaining the original

BPMN model and the generated BPEL code synchronized, in the sense that any modification to one is propagated to the other.

Adding 'programming in the small' support to BPEL

- BPEL's control structures such as 'if-then-elseif-else' and 'while' as well as its variable manipulation facilities depend on the use of 'programming in the small' languages to provide logic. All BPEL implementations must support XPath 1.0 as a default language. But the design of BPEL envisages extensibility so that systems builders can use other languages as well. BPELJ is an effort related to JSR 207 that may enable Java to function as a 'programming in the small' language within BPEL.

WS-BPEL 2.0

What's new in WS-BPEL 2.0?

- New activity types: repeatUntil, validate, forEach (parallel and sequential), rethrow, extensionActivity, compensateScope
- Renamed activities: switch/case renamed to if/else, terminate renamed to exit
- Termination Handler added to scope activities to provide explicit behavior for termination
- Variable initialization
- XSLT for variable transformations (New XPath extension function bpws:doXslTransform)
- XPath access to variable data (XPath variable syntax $variable[.part]/location)
- XML schema variables in Web service activities (for WS-I doc/lit style service interactions)
- Locally declared messageExchange (internal correlation of receive and reply activities)
- Clarification of Abstract Processes (syntax and semantics)
- Enable expression language overrides at each activity

ActiveBPEL – deploying processes

- create deployment archive file
- – normal .jar archive
- – named .bpr
- – copy to subdirectory „bpr“ in Tomcat folder
- – following directory/file structure

BPEL covers all necessary workflow patterns

- – output of composition: executable process
- - provides no adequate semantic to describe WS
- • Planner provides meta-model for modelling capabilities need to describe all Services in this language need to transfer into BPEL process definition better way using standardized language ?

- • ActiveBPEL is currently the best available free workflow engine that supports BPEL
- Semantic Web Services (SWS)
- "Semantic Web concepts are used to define intelligent web service, i.e., services supporting automatic discovery, composition, invocation and interoperation."
- „These efforts try to improve current web service technology around SOAP, WSDL and UDDI, which provides very limited support for real automation of services.

Challenges to tackle with SWS

- Automatic discovery of services

– Semantic match between declaritive description of services sought and services offered

• Automatic composition of services

– Allow the composition of services to provide functionality that available services cannot provide

• Both tasks need a declarative language to describe semantics of available and sought services (goal) The OASIS Business Transaction Protocol (or simply BTP) is a protocol for coordinating loosely coupled systems like Web services. BTP was the product of more than a year's work from several major vendors including BEA, Hewlett-Packard, Sun Microsystems, and Oracle, which set out to reconcile transactionality with systems consisting of loosely coupled autonomous parties. In short, the result of this collaboration has been to produce a specification which, although based on the traditional two-phase approach, is still suitable for supporting transactions on the Web.

OASIS BUSINESS TRANSACTION PROTOCOL

Interoperation

- Using XML, over multiple communications protocols

Coordination of autonomous parties

- Relationships are governed by contracts, rather than the dictates of a central design authority

Drop less ACID

- Multiple possible successful outcomes to a transaction
- Relaxed isolation, volatile results

Discontinuous service

- Work unit lifespans exceed sub-system MTBFs

Business Transactions and Business Process

BTP complements BP/Collaboration frameworks

- A coordinated, mutually understood outcome requires
- Special messages and acknowledgements
- Consistent, durable record of decisions
- Asynchronous failure recovery operations
- These features are tricky, error-prone and intrusive
- "Build or buy"
- BTP lets business people concentrate on business process
- Puts housekeeping work in the background
- Minimizes application exchanges
- Reduces complexity of collaborative process schemas or scripts
- Reduces conformance testing
- Deploy new trading protocols or conventions more quickly

- BTP uses a two-phase outcome coordination protocol to create atomic effects (results of computations).
- BTP permits the composition of atomic units of work into cohesive business transactions (cohesions) which allow application selection of which work units will be confirmed (or cancelled)
- Atoms are cohesions where the underlying work units are either all confirmed, or are all cancelled

Messaging

- All BTP messages are XML documents
- Can be compounded for optimization
- "One-shot requests": only 2 WAN messages instead of 6
- Application response + ENROL/PREPARE
- "One wire" application topologies
- All traffic between two business entities over a single, authenticated link

BPEL 1.1 and OASIS WSBPEL

- The original BPEL specification that we have considered so far in this chapter has been superceded as part of the original vendor's efforts to standardize the technology. As such, as their submission to OASIS under the WSBPEL (Web Services Business Process Execution Language) Technical Committee, IBM, Microsoft, BEA and partners have updated the specification to version 1.1 with a number of changes.

- The most obvious changes in BPEL 1.1 is that the term "container" has been replaced with the more traditional term "variable," though its type is still considered in terms of messages. These variables are now supported at arbitrary scope, unlike BPEL 1.0 which only supported containers at the global process scope.

. Transaction

- Transactions are a fundamental abstraction in dependable computing systems. Put simply, a transaction is a unit of work which either succeeds completely or fails without leaving any side effects. To illustrate, a commonly cited example of a transaction use-case is where an amount of money is to be moved between bank accounts. In this case the goal is to ensure that the money both leaves the sender's account and is received by the recipient's account, or if something fails, for it to appear as if the transaction itself logically never occurred.

- This is the inherent value of transactional systems; if something goes wrong, they allow us as programmers to make it appear as if it never happened in the first place. In an inherently unreliable world, transactions can truly be a godsend, especially when we consider the amount of computing infrastructure involved in something apparently as simple as moving money between accounts. Any aspect of that computing system from the network cables through to the software and everything in between has a small yet significant chance of failing—something we'd rather it didn't do while it's processing our money! The point is that using transactions to safeguard against failures allows us to reverse any partial work performed during a failed transaction and, thus, prevent our money from disappearing into a banking black hole.

ACID transaction

- The field of transaction processing is by no means a new discipline. Transaction processing infrastructures such as CICS and Tuxedo, OTS/JTS and MS DTC have been around for decades, and much innovative and interesting work continues in the field today. However, underlying the development of transaction technology over the decades has been one pervasive notion: ACID.

Distributed Transactions and Two-Phase Commit

- The evolution of transactions from centralized to distributed systems has followed the evolution of computing from a centralized resource model to its modern day distributed and federated architectures. The underlying goals in distributed transaction processing are the same as in the traditional centralized model: to ensure a unit of work either successfully completes or logically appears to have never been run at all.

Dealing with Heuristic Outcomes

- An heuristic outcome to a transaction is simply the worst thing that can arise during a 2PC-based transaction system. They may occur where a participant in the transaction

reneges on its promise to either commit or abort and instead does the exact opposite. This means that the ACIDity of the transaction is compromised, since changes to data in some participants will have been made durable, while changes to data managed by other participants will have been discarded. As was alluded to earlier, the chances of such an outcome occurring are significantly increased if the period of uncertainty (the gap between prepare and commit phases) is too long, and we may arrive at a situation where participants in a transaction start to guess at outcomes for themselves (that is, they may make heuristic decisions).

Scaling Transactions to Web Services

- ACID transactions in their centralized or distributed variations are especially suited to the bank account credit-debit type of problem, and offer the kinds of guarantees on which truly dependable back-end computing systems can be built. There is, however, a practical limitation to how far ACID transactions can be applied to the Web services world, even with advanced mechanisms like nesting and interposition.
  - Distributed Transaction – OASIS Business Transaction Protocol.

POSSIBLE QUESTIONS

PART – A

(Online Exam 20*1=20 marks)

- **Multiple Choice Questions available in Moodle**

PART – B

(Each Question carries 2 marks)

1. Explain about Conversation and its process
2. Discuss in detail about Web services Conversation Language
3. Explain about WSCL Interface Components
4. Discuss in detail about Workflow
5. Explain about Business Process Management
6. Explain about Workflow Management System
7. Discuss in detail about BPEL
8. Explain about ACID Transaction
9. Explain about Distributed Transaction
10. Discuss in detail about OASIS Business Transaction Protocol.

PART – C

(Compulsory Question carries 10 marks)

11. Difference between WSCL and WSDL
12. Explain one and two way interaction in WSCL
13. Explain the Elements of WSCL Specification
14. Discuss in detail about Transaction
15. Explain about WS BPEL2.0

Questions	Opt1	Opt2	Opt3	Opt4	Opt5	Opt6	Answer
The attribute_____is optional that allows schema definiton to be owned in case of binary upload.	hrefshcema	hrefload	hbinary	thread			hrefshcema
WSCL paves way or creation of _____ framework.	sigma	standard	service	reading			service
Web service conversation language loses_____	infinite automation	file state automation	simple state automation	file state transfer			simple state automation
_____ element acts as a container for interaction element	transition	transversal	travellogue	timer			transition
_____ condition can be used to guard against taking particular transition in a conversation	destination	source interaction	source	destination interaction			destination interaction
The WSCL specification does not mandate _____ interaction.	start and end	forward and backward	front and back	up and down			start and end
The _____ part of WSCL description draws together messages,interactions and transitions	conversation	translation	trading	specification			conversation
the _____ elements attribute declares the entry and exit point .	root	basic	link	group			root
_____ system is the basic function of workflow management systems.	routing	distribution	agent	assistant			routing
_____ system detects exceptonal circumstances.	routing	distribution	agent	assistant			distribution
_____ is an example of workflow management software .	fox hub	fox sub	fire hub	fire sub			fire sub
BPEL supports _____ based message conolation.	strategy	standard	property	document			property

A _____ system allows the encapsulation of logic with local variables.	robing	funding	scoping	rhyming			scoping
_____ extended BPEL to role based human activities a well.	people hub	BPEL people	BPEL4 people	BPEL public			BPEL4 people
_____ is a type of webservice language	oasis	blooper	tropics	robotics			oasis
_____ defines notation for specifying business process behaviour based on web services.	BPRL	BSNL	BPFL	BPEL			BPEL
WS BPEL defines an interoperable _____ model.	seggregation	integration	repitation	sequencing			integration
MS-transaction decribes Co-ordinatio types that are used with _____	right	extensible	extended	experienced			extensible
WS_BPEL provides a faster way to compose and orchestrate services by _____	reuse	postuse	preuse	usage			reuse
_____ logic is centralised within one location rather than distributed across multiple services.	Business process	backup	read only	centralised			Business process
FSA stands for _____	file state automobile	file state automation	for super area	file state transfer			file state transfer
WSCL are themselves _____ documents	XML	FTP	HTTP	SMTP			XML
_____ refers to the schema to which the documents correspond.	hrefschema	href WS	hregschema	hret schema			hrefschema
An _____ is an exchange of documents between a service and client.	deviation	automation	interaction	involvement.			interaction
In _____ an outbound document is sent in an inbound document is taken as reply	recieve	send	sending	send-recieve			send-recieve

A _____ interaction does not contain any document exhanged.	full	half	empty	big			empty
_____ interaction does not proceede the destination interaction when conversation is executed.	force	master	important	source			source
_____ is an additional condition for transaction.	final interaction	beginning interaction	source interaction	conversation			conversation
_____ is used to orchestrate the various message exchange that occurs in each stage of conversation	WWW	WSCL	WLTP	OLTP			WSCL
____ types of interactions are supported by WSCL	6	2	3	5			5
There might be _____ interaction with which the conversation ca start or end	less	one	zero	more than one			more than one
_____ conversations comprises 3 or more participation roles	multiparty	multifunction	multi role	multirot			multiparty
_____ refers to ID of the first interaction to be executed in conversation	initial interaction	final interaction	all interactions	full interaction			initial interaction
WS-secure conversation provides secure commuication using _____	lock-keys	fine keys	syntax keys	session keys			session keys
A_____ manages the ranscational state and enables web services and clients to register as participants	coordinator	conductor	manager	mover			coordinator
_____ service enables the application to activate a transcation.	termination	determination	activation	allocation			activation

_____ transaction are be used to bridge between proprietory transaction service implementary.	ACID	BTL	ERP	UML			ACID
_____ transaction may be structured as a collection of atomic transactions	business process	ACIS	OASIS	business buildings			business process
BTP is _____	interoperable	not interoperable	secure	scalable			not interoperable
_____ can be eithe send receive or receive sent	One way interaction	2 way interaction	3 way interaction	4 way interaction			One way interaction
WSCL only models _____ level interaction	Business	Hier	Physical	Application			Business
_____ interaction follow the source interaction when conversation is executed	Source	Voice	Mail	Destination			Destination
____ interaction preced the destination interaction	Voice	Source	Destination	Mail			Source
_____ interaction condition is an additional condition for transaction	Voice	Destination	Source	Mail			Source
_____ specify ordering relationships between interactions	Transactions	Transistors	Trans-associates	Trans-roots			Transactions
WSCL conversation definitions are themselves_____ documents	HTML	XML	FTP	SMTP			XML
WSCL provides a formal language for specifying _____	Conversations	Document	Image	Audio			Conversations
Two subtypes of one way interactions are _____	Send,receive	Send,resend	Send,send back	Send, read			Send,receive
A receive send interactions can specify ore than one _____ XML document	Inbound	Outbound	Middle bound	Startup			Outbound

The registrations and login interactions areonly allowed in case of_____	Valid login RS	Invalid login RS	Valid logbook	Invalid password			Invalid login RS
_____ consists of an orchestard and repeatable pattern of business activity	System	Software	Work-minder	Workflow			Workflow
___ process flow will create a visualization of business process flow	Work	Business	Random	Logic			Business
Workflows can be started by users or can be _____	Written	Structured	Collapsed	Automated			Automated
_____ are asynchronous or real-time processes	Workships	Workflows	Workman	Worktide			Workflows
_____ process run in background	Asynchronous	Synchronous	Runtime	Data			Asynchronous
___ process run immediately	Asynchronous	Synchronous	Runtime	Data			Synchronous
Transaction specify ordering relationships between	interactions	Transistors	Trans-associates	Trans-roots			interactions
source interaction condition is an additional condition for _____	Voice	Destination	transaction	Mail			transaction
Send-receive is a type of _____interaction.	one way	two way	threeway	four way			one way
In case of Invalid login RS_____interaction is allowed	state and form	registrations and login	password change	stub check			registrations and login

UNIT-IV

SYLLABUS

Security – Security Basics – Security Issues – Types of Security Attacks – WS –Security.Mobile and Wireless – Mobile Web Services – Challenges with mobile – Proxy Based Mobile Systems - Direct Mobile Web service access - J2ME Web Services.

SECURITY

SECURITY

- The most critical issue limiting the widespread deployment of Web services by organizations is the lack of understanding of the security risks involved as well as the best practices for addressing those risks. Development and IT managers want to know whether the security risks and the types of attack common for Web sites will be the same for Web services. Will existing enterprise security infrastructure already in place, such as firewalls and well-understood technologies like Secure Sockets Layer (SSL), be sufficient to protect their companies from Web service security risks?

- Much of the experience companies have with security is with Web sites and Web applications. Both Web sites and Web applications involve sending HTML between the server and the client (e.g., Internet browser). Web services involve applications exchanging data and information through defined application programming interfaces (APIs). These APIs can contain literally dozens of methods and operations, each of which presents hackers with potential entry points to compromise the security and integrity of a system.

SECURITY

Everyday Security Basics

- The issues underlying security on the Internet are not that different from security issues we face when making everyday transactions. These issues are not just about monetary transactions that involve the exchange of money during purchases, but any transaction where there is dissemination of critical information—such as social security or other uniquely identifying numbers, or of limited resources, such as money.

- Transactions rely on a level of trust between the parties involved in the transaction. The person disseminating information must prove to the person receiving information that she is who she claims to be. The person receiving the information must also prove to the person disseminating the information that he will hold the information in confidence and use it responsibly and appropriately. In many situations, the second person may also have information to disseminate based on the received information. In this case, both parties must prove their identities to the other.

Security Is An End-to-End Process

- Contrary to popular belief, security is important not only during data transport between one computer and another computer, but also after transport. In the process of doing a transaction the path that data follows is oftentimes complex and long, involving multiple hops. If a small section of this path is insecure, the security of the entire transaction and the system is compromised.

- Today, many systems that are seemingly secure are in fact insecure. Designers and architects usually focus on the security issues relating to the transmission of data between the client and the server, while other segments of the data transmission value chain are assumed to be secure and do not get much attention. Many of today's Web site and Web application vulnerabilities are directly applicable to Web services as these and other legacy systems are oftentimes just being wrapped and made available as Web services.

SECURITY ISSUES

- Before we can address Web services security, we must first understand the areas where potential threats may occur as well as how they may occur. In this section, we look at the issues surrounding Web services security flow of a Web service invocation starting from the client application through the actual business logic implementation of the service and back to the client again. Throughout this flow, we analyze the possible threats and best practices to address those threats.

Data Protection and Encryption

- Data protection refers to the management of transmitted messages so that the contents of each message arrives at its destination intact, unaltered, and not viewed by anyone along the way. The concept of data protection is made up of the sub-concepts of data integrity and data privacy:

TYPES OF SECURITY ATTACKS AND THREATS

Types of Security Attacks and Threats

- In this section we briefly look at the types of security attacks and threats that are possible within a Web services environment. Since Web services leverage much of the infrastructure developed for Web sites, it is understandable that the types of security breaches that are common for Web sites will also be common for Web services.

- However, since Web services provide an application programming interface (API) for external agents to interact with it and also provides a description of this API (in the form of WSDL files), Web service environments facilitate and in fact attract attacks. These environments also make it more difficult to detect attacks from legitimate interactions.

Malicious Attacks

- As we have discussed, Web service traffic shares a lot in common with Web site traffic. Both types of traffic usually flow through the same ports and while many firewalls can recognize SOAP traffic, they usually treat them as standard HTTP traffic.

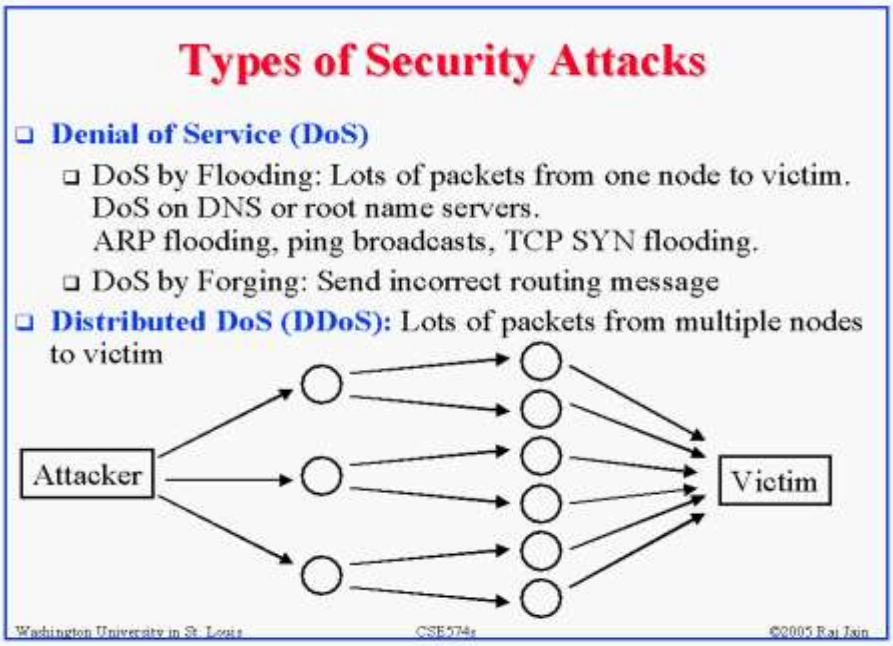
WS-SECURITY

- WS-Security is a specification that unifies multiple Web services security technologies and models in an effort to support interoperability between systems in a language- and platform-independent manner. More specifically, WS-Security specifies a set of SOAP extensions that can be used to implement message integrity and confidentiality. To this end, the WS-Security specification brings together a number of XML security technologies and positions them within the context of SOAP messages.

- The origins of WS-Security are with Microsoft, IBM, and VeriSign submitting a group of security specifications to the Organization for the Advancement of Structured Information Standards (OASIS). Later, Sun Microsystems started to cooperate to further develop the specifications. With such heavyweights behind it, WS-Security is emerging as the de facto standard for Web services security.

- he lack of a coherent security model and policy is the most often cited reason for the slow deployment of externally facing Web services by enterprises. Addressing security issues with vigor will not only make Web services (as well as the applications that consume Web services) more secure, but will also increase the community's confidence in Web service technologies. This improved confidence will likely result in increased numbers and types of available Web services.

- In this chapter we took a broad look at security, and then focused on the security issues specific to Web services environments. We described how security is really an end-to-end process, and a secure system cannot be implemented by simply using a few technologies within a service or application.


Types of Security Attacks
Denial of Service (DoS)
DoS by Flooding: Lots of packets from one node to victim.
DoS on DNS or root name servers.
ARP flooding, ping broadcasts, TCP SYN flooding.
DoS by Forging: Send incorrect routing message
Distributed DoS (DDoS): Lots of packets from multiple nodes to victim
Attacker
Victim
Washington University in St. Louis
CSE574s
©2005 Raj Jain
11-3

Security Attacks (Cont)

- **Trojan Horse**: Programs with hidden functionality. Could be triggered when a specific time or condition.
- **Trap Doors**: Backdoor. Code segment to circumvent access control.
- **Virus**: A program that reproduces by introducing a copy of itself in other programs. Jump to Viral code and return to beginning.
- **Worms**: Creates copies of itself on other machines. Unlike virus, worms do not require user action. Morris worm spread by finding IP addresses on the machine. Slammer worm sent UDP packets to cause buffer overflow.
- **Buffer Overflow**: Overwrite code segments and execute code in data space. Many programming languages do enforce bound checking.

Washington University in St. Louis CSE574s ©2005 Raj Jain

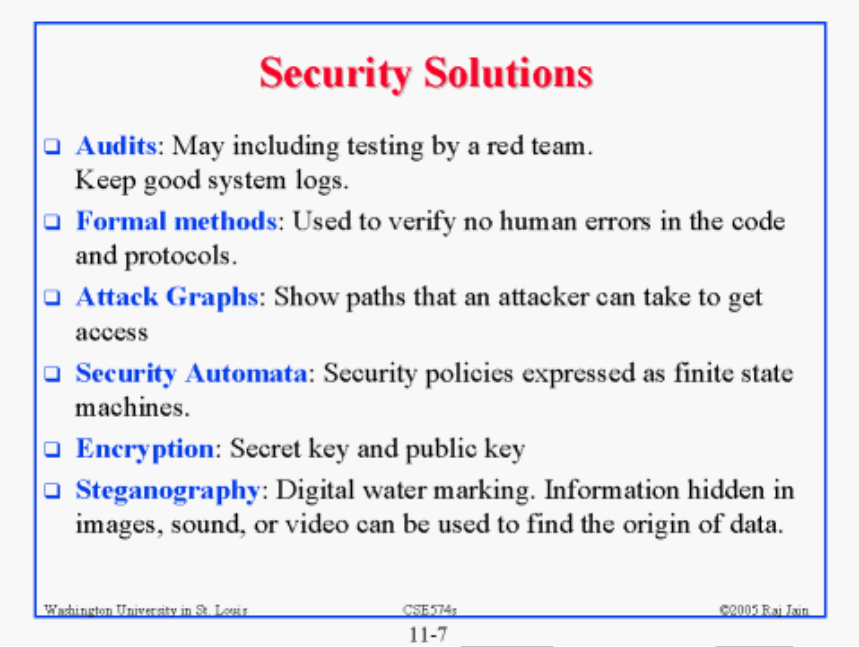
11-5

Security Attacks (Cont)

- **Covert Communications Channel**: Hidden channel.
  - Capture electromagnetic radiations from keyboards, screens, and processors.
  - Pizza deliveries to White House
- **Steganography** or Information Hiding: Lower bits of pictures or music files.
- **Reverse Engineering**: dismantling and inspecting to infer internal function and structure. Code dumping and decompiling
- **Scavenging**: Acquisition of data from residue. Searching through rubbish bins. Buffer space in memory, deleted files on disks, bad blocks on disks
- **Cryptanalysis**: Find encryption key, encryption method, or clear text. Get plain-text and cipher text pairs.

Washington University in St. Louis CSE574s ©2005 Raj Jain

11-6



MOBILE AND WIRELESS

What is a Mobile Agent?

- This is a simpler question to answer - at it's most basic level we can say that an agent is mobile if we don't need to know where in the system it resides. Thus, a mobile agent could be found on a desktop computer, a mobile device such as a PDA, or even in the depths of a server. All we need to know is how to find the agent, and how to communicate with it. Once we have this conceptual framework in place, we have the abstraction and flexibility required to design our system.
- How mobile is a mobile agent, though? Currently, in order to move from one system to another, or even to communicate amongst themselves, mobile agents need a common platform on which to operate. Thus, in order for our mobile agents to be useful to our business partners (and to ourselves by operating with our partners), we have to share a platform with our partners, something we cannot guarantee being able to do. Even with the best developers in the world, this will take time. In order for intelligent agents to communicate, they would need to share a common language - the best candidate for this would be XML, or SOAP.

Mobile Agents and Web Services

- With the advent of Web Services, we can develop mobile agents that can exist on an Intranet, or even in the deeper waters of the Internet. How? Because exposing our mobile agents as Web Services means that we can take advantage of the already existing network infrastructure and communication protocols provided by the Internet. As we expose our

existing mobile agents as Web Services, or write future agents to take full advantage of the situation, not only do we make it easy for ourselves, but also quicker and more convenient.

- Mobile intelligent agents can be developed to the Web Services standard using SOAP, WSDL, and UDDI. The mobile intelligent agents are Web Services ready so they can be plugged into this network and serve its knowledge. Using WSDL give the agent the ability to describe it's capacity, invaluable if it is to become part of a domain agent society.
- As if the ability to use the Internet to further the reach of our mobile agents wasn't enough, there are companies dedicated to providing networking facilities for Web Services. With Web Services essentially moving components onto the Internet, it makes sense for companies to provide technology that will ease this process. Using such facilities, we can launch our mobile agents into the wider world should we desire. Companies such as Grand Central and Flamenco Networks are providing these Web Services network solutions.
- **Advantages of using Web Services**
- As well as making it easy for our mobile agents to be truly mobile, Web Services can provide other important benefits. The following list represents the key additional benefits:

1. Conservation of Enterprise bandwidth
2. Reduction in latency
3. Conflict Knowledge Credibility Management
4. Support for Dynamic Deployment
5. Improved decision support workflow

- Enterprise bandwidth is conserved through the use of mobile agents due to the reduction in network traffic they enable. Rather than pass numerous requests across the network, we can send a mobile agent to the site requiring processing; once there, the agent can conduct the operation, and return to the server only the information that actually needs to be returned. Of course, there will be a point when it isn't efficient to use a mobile agent, like when the process involves only two or three network calls.
- Our mobile agents would ideally reside on a Web Services network. If a user requests knowledge from a mobile agent very frequently, the Web Services network can intelligently save this knowledge into a cache. So when a user asks the same questions, the Web Service cache will broadcast it for mobile agent if there is nothing new in the request.
- A Web Services network will host many mobile agents, each of which will have expert knowledge in a particular area. The Web Services network will also need intelligence to be able to tell which agent has the best knowledge or if the knowledge conflicts with that of another agent. The Web Services network will judge and pick the best by its own intelligence and broadcast to the requesting agent.

- As you would expect, mobile agents can aid in the dynamic deployment of new components in a Web Service. Additionally, a mobile agent can act as a remote controller in a process, allowing decision support workflow to be improved.
- With our mobile agents residing in a Web Services network, many of our infrastructure concerns are handled for us. When a user requests certain knowledge, the Web Service network will route this message to a mobile agent. The network determines which is the nearest agent that has the knowledge requested, and routes the query by the quickest path.

MOBILE WEB SERVICES

- The challenge for architects designing software is to be able to seamlessly support mobile devices. Where a software architect could before count on a keyboard, mouse, and monitor together with a hard disk drive, a reasonable amount of memory and processing power, she can no longer count on such a fixed and well-defined target platform. Instead, the architect must now concern herself with whether a display is available at all, and whether the available processing resources are sufficient to provide a reasonable latency to the application. Moreover, as the number and type of mobile devices increase, architects must think about how the nuances between different devices will affect their software.

- Web services are an interesting addition to the technology mix for developing mobile applications. The use of Web services allows some, if not most, of the application's business logic to run on remote servers, which are independent of the mobile device's computational resource limitations. This has the added benefit that a variety of mobile devices can effectively access the same functionality or business logic.

CHALLENGES WITH MOBILE

- Many challenges exist in the development of mobile applications and systems that are usually not an issue in the development of non-mobile systems. For instance, non-mobile developers rarely think about the ramifications that their application architecture will have on a system's energy consumption. Non-mobile developers also do not usually think about how their application's network utilization will affect the user's monthly wireless subscription bill.

- In this section, we briefly look at the issues that are inherent to the development of mobile systems. Then, we discuss approaches for addressing these issues and solution best practices.

PROXY-BASED MOBILE SYSTEMS

- When developing mobile Web services-based applications, a variety of different architectures are possible. A mobile systems architecture that is commonly used is a proxy-based one in which the mobile application communicates only with a proxy server. The proxy server in turn communicates with and manages the back-end resources, such as Web services.

- The mobile device is capable of running multiple applications. Each of these mobile applications presents a front-end user interface that presents information to users and captures user input. The application does only enough processing on the user input so that the proxy server can properly interpret the data. Different proxy servers may exist for different applications, or a single server may handle multiple applications. The role of the proxy server is to implement all required business logic to service the needs of the mobile application. The implementation of the business logic may include Web services or other distributed computing resources.

DIRECT MOBILE WEB SERVICE ACCESS

- Mobile applications can directly access Web services without an intermediary proxy server. MobileDirectTestServiceInvoke.java is a simple Java program that uses the InvokeService class we discussed earlier in the chapter and directly calls the ChessMaster Web service without using a proxy server for the invocation.

J2ME WEB SERVICES

- The Java 2 Micro Edition (J2ME) Web Service specification is an attempt at standardizing programmatic access to Web services from J2ME client applications. Currently, this specification exists as Java Specification Request (JSR) 172 as part the Java Community Process (JCP). More information about JSR-172 can be found at http://www.jcp.org.

- Web services can be accessed by J2ME clients today, but involve low-level network APIs that result in non-standard and proprietary implementations. Other specifications and initiatives, such as Java API for XML Processing (JAXP) and Java API for XML-based RPC (JAX-RPC), have addressed high-level and standardized means of accessing XML-based Web services from Java platforms. However, these initiatives do not sufficiently address the unique requirements and limitations of mobile environments, including footprint, computational resources, and wireless networks.

Prepare the J2ME Web service client and Java Web service

- Run KToolbar of J2ME Wireless Toolkit.
- Create the new project (Hello).
- Specify the Project Name (Hello).
- Specify the MIDlet Class Name (HelloMidlet).
- Select the Additional APIs - Web Service Access for J2ME (JSR 172).
- Create the Web service directory (C:\WTK22\apps\Hello\server).
- Create the source directory of Web service (C:\WTK22\apps\Hello\server\src\hello).
- Write the property file (build.properties) for Apache Ant (a Java-based build tool) (in C:\WTK22\apps\Hello\server) and these properties must be set according to your environment (project-root=C:/WTK22/apps/Hello/server).
- Write the compilation and deployment file (build.xml) (in C:\WTK22\apps\Hello\server) for Apache Ant and specify the project name according to your environment (project name = "helloservice").

Create and deploy a Java Web service.

- Write the Web service interface class (Hello.java) (in C:\WTK22\apps\Hello\server\src\hello).

```
package hello;

import java.rmi.*;

public interface Hello extends Remote {
  public String getHello(String name) throws RemoteException;
}
```

Write the Web service implementation class (HelloImpl.java) (in C:\WTK22\apps\Hello\server\src\hello).

```
package hello;

import java.rmi.RemoteException;

public class HelloImpl implements Hello {
  public String getHello(String name) throws RemoteException {
    return "Hello, " + name + "!";
  }
}
```

- Write the associated descriptor files (all XML files) (in C:\WTK22\apps\Hello\server\src).
  - config.xml
  - jaxrpc-ri.xml
  - web.xml
- Compile the service (Hello.java and HelloImpl.java) and generate Web services stubs/ties (C:\WTK22\apps\Hello\server\generated) and WSDL file (C:\WTK22\apps\Hello\server\WEB-INF\classes\helloservice.wsdl) using tools provided with the Java Web Services Developer Pack:
- ant compile
- Deploy the web service to the Tomcat server.
  - Build the web service to generate a web application archive (C:\WTK22\apps\Hello\server\helloservice.war):
  - ant build
  - Remove the previously installed Web service (C:\tomcat50-jwsdp\webapps\helloservice.war).
    If it is not removed before deployment, the incorrect WSDL (helloservice.wsdl) might be installed.
  - Deploy the web service to the Tomcat server (C:\tomcat50-jwsdp\webapps\helloservice.war):

- ant deploy
- Verify if the Web service is correctly deployed.
  - Access http://localhost:8080/helloservice/helloservice.
  - Click http://localhost:8080/helloservice/helloservice?WSDL

Create the J2ME Web service client

- Use the generated WSDL file and the tools built into the Wireless Toolkit to generate the stubs and supporting code used by the MIDlet to access the Web service.

  Specify the location of the Web service. (in C:\WTK22\apps\Hello\server\WEB-INF\classes\helloservice.wsdl)

  <soap:address location="http://localhost:8080/helloservice/helloservice"/>

  Click the Project menu and select Stub Generator.

  Specify WSDL Filename or URL (C:\WTK22\apps\Hello\server\WEB-INF\classes\helloservice.wsdl).

  Specify Output Package (helloservice) and generate client stubs (in C:\WTK22\apps\Hello\src\helloservice).

Code the MIDlet (HelloMidlet.java) (in C:\WTK22\apps\Hello\src) and associated classes using JAX-RPC to invoke the Web service and JAXP to process the SOAP message.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import java.util.*;
import java.rmi.RemoteException;

import helloservice.*;

public class HelloMidlet extends MIDlet implements Runnable, CommandListener {

  Hello_Stub service;
  Display display;
  private Form f;
  private StringItem si;
  private TextField tf;
  private Command sendCommand = new Command("Send", Command.ITEM, 1);
  private Command exitCommand = new Command("Exit", Command.EXIT, 1);

  String name = "";
```

```
  public void startApp() {
    display = Display.getDisplay(this);
    f = new Form("Hello Client");
    tf = new TextField("Send:", "", 30, TextField.ANY);
    si = new StringItem("Status:" , " ");
    f.append(tf);
    f.append(si);
    f.addCommand(sendCommand);
    f.addCommand(exitCommand);
    f.setCommandListener(this);
    display.setCurrent(f);
  }

  public void pauseApp() {}

  public void destroyApp(boolean unconditional) {}

  public void commandAction(Command c, Displayable d) {
    if (c == sendCommand) {
      name = tf.getString();
      si.setText("Message sent: " + name);
      /*
       * Start a new thread so that the remote invocation won't block
       * the process.
       */
      new Thread(this).start();
    }
    if (c == exitCommand) {
      notifyDestroyed();
      destroyApp(true);
    }
  }

  public void run() {
    try {
      service = new Hello_Stub();
      service._setProperty(Hello_Stub.SESSION_MAINTAIN_PROPERTY, new
Boolean(true));
      String msg = service.getHello(name);
      si.setText("Message Receive: " + msg);
    } catch (Exception exception) {}
  }
}
```

POSSIBLE QUESTIONS

PART – A

(Online Exam 20*1=20 marks)

- **Multiple Choice Questions available in Moodle**

PART – B

(Each Question carries 6 marks)

1. Explain the process of Web Security
2. Explain the feature of Security Basics
3. Explain about Security Issues
4. Explain about J2ME Web Services
5. Explain the types of Security attacks
6. Discuss in detail about the Challenges with mobile
7. Explain about Proxy Based Mobile Systems
8. Explain about Direct Mobile Web service access
9. Explain about Mobile Web Services
10. Explain the advantages of using Web services

PART – C

(Compulsory Question carries 10 marks)

11. Write a program to generate WSDL file and the tools built into the Wireless Toolkit to generate the stubs and supporting code used by the MIDlet to access the Web service.
12. Explain how to create and deploy java web service'
13. Discuss in detail about Security solution for threads
14. Explain how to create the J2ME Web service client
15. Difference between Mobile web service and J2ME Web services

Questions	Opt1	Opt2	Opt3	Opt4	Opt5	Opt6	Answer
Hardware threats are easy to detect in comparison	Software threats	Malware	functions	Hardware parts			**Software threats**
Physical electrical environmental and	S/W threats	H/W threats	software	Applications			**H/W threats**
H/W threats are classified as physical , electrical _____and	Environmental	electronic	engine	effective			**Environmental**
In _____attack an adversery deploys a sniffer tool and wait	active attack	passive atack	adaptive attack	aggressive attack			**passive atack**
There are _____ goals for security threat	3	4	5	1			**3**
A ____ is a group of hijacked computer	Botnet	cardcore	boldnet	bottlenet			**Botnet**
A _____ reproduces itself by ataching to other files	DOS	Trojans	Virus	Logic bomb			**Virus**
In security , DOS is _____	Denial of	Denial of	Demo of	Denial of sustainin			**Denial of service**
SOA's are implemented with_____	Web apps	Web services	Web technol	webminars			**Web services**
In tear drop second packet has_____	fragmentation	Functions	frameworks	modules			**fragmentation offset**
Through _____ operators can offer value added services	Mobile web services	cellphone towers	mobile media	web functions			**Mobile web services**
_____ attack usually takes place between running	Hijack	Binary	run	Half way			**Hijack**
In password attack someone tries to login	Key	column name	username	guessed password			**guessed password**
_____ pings all possible IP addresses	Run sleep	ping-pong	Ping sweep	ping-comb			**Ping sweep attack**
Cisco secure agent works like_____	Antivirus	malware	threat	guard			**Antivirus**
_____ is used to take finger print of data	Data making	Data hashing	Data filling	Limiting data			**Data hashing**
_____ filters all network traffic for possible	Cisca IPS	Cisca RAM	Cisca ROM	Cisca CPC			**Cisca IPS**
IPS can be integrated as_____	Stand alone device	Rimming device	Rivial device	Stand out device			**Stand alone device**

_____ attack occurs when someone is between you and	Man in middle	Man in edge	Man in corner	Man in front			**Man in middle**
Man in middle attack occurs when someone is between you	system	software	hardware	Communicating person			**Communicating person**
Computers are mostly communicating between _____ of network layers	High levels	low levels	peak level	custom level			**low levels**
A ___ is an application or device that can read,monitor and	Sniffer	bufer	hop	hustle			**Sniffer**
Even encapsulated packets can be broken open and read	altered	encrypted	hidden	played			**encrypted**
_____attack is used after Re connaissance attack	Exploit attack	Car attack	SOS attack	Buffer attack			**Exploit attack**
Exploit attack is used after _____	Re connaissance	Car attack	SOS attack	Buffer attack			**Re connaissance attack**
_____has much more feature than antivirus	CSA	VLSI	RPC	CSS			**CSA**
_____ includes audit logs, malicious mobile	ADT	VLSI	RPC	CSA			**CSA**
Blocking traffic will result in _____ to	loss of access	reading	writing	loss of terminal			**loss of access**
A _____ is the secret code or number necessary to interpret secured	key	log	hype	chain			**key**
After an _____ obtains key, the key is	attracker	denoter	writer	blogger			**attracker**
After an attracker obtains key, the key is called as	Compromised	column key	custom key	casper key			**Compromised key**
Enabling the mobile web services will automatically enable _____ protocol	REST	BBC	SLEEP	KINDLE			**REST**
Mobile web services will allow Xmlrpc:use for _____	Abacus	Acuratec	Authenticated	Appropriate			**Authenticated**
_____ is dormant until an event triggers it	Gun	Logic bomb	Knife hub	Hydrogen bomb			**Logic bomb**
_____ is a self	Horn	Tbalet	Tuffled	Worm			**Worm**

_____are used in denial of service attacks, typically against targeted web sites.	Worm	Zombie	Virus	Trojan horse			**Zombie**
Select the correct order for the different phases of virus execution.i) Propagation phase ii) Dormant phase iii)	i, ii, iii, and iv	i, iii, ii and iv	ii, i, iv an iii	ii, iii, iv and i			**ii, i, iv an iii**
A _____attaches itself to executable files and replicates, when the infected program is executed, by finding	Stealth virus	Polymorphic Virus	Parasitic Virus	Macro Virus			**Stealth virus**
_____ is a form of virus explicitly designed to hide itself from detection	Stealth virus	Polymorphic Virus	Parasitic Virus	Macro Virus			**Parasitic Virus**
A _____ creates copies during replication that are functionally equivalent but have	Boot Sector Virus	Polymorphic Virus	Parasitic Virus	Macro Virus			**Polymorphic Virus**
A portion of the Polymorphic virus, generally called a _____ , creates, a random encryption, key to	mutual engine	mutation engine	multiple engine	polymorphic engine			**mutation engine**
A macro virus is_____ independent.	platform	system	directory	hardware			**platform**
ACID Transaction processing system might use _____and	Commit	Start	Open	close			**Commit**
In _____, the virus places an identical copy of itself into other programs or into certain system areas	Dormant phase	Propagation phase	Triggering phase	Execution phase			**Propagation phase**
A _____ is a program that secretly takes over another Internet-attached computer and then uses	Worm	Zombie	Virus	Trap doors			**Zombie**
What is “Trend Micro”?	anti-virus software	program	virus program	virtual program			**anti-virus software**

What is the name of the viruses that fool a user into downloading and/or executing them by pretending to be useful	Cracker	Worm	Trojan horses	Keylogger			**Trojan horses**
The virus that spread in application software is called as	Boot virus	Macro virus	File virus	Anti virus			**Macro virus**
How does a Le-Hard virus come into existence?	Hardware	Software	FRIDAY 13	Command.Com			**Command.Com**
What is the virus that spread in computer?	hardware	system software	computer program	windows tool			**computer program**
What kind of attempts is made by individuals to obtain confidential information	Computer viruses	Spyware scams	Phishing scams	Logic scams			**Phishing scams**
Delayed payload of some viruses is also called as	Time	Bomb	Anti-virus				**Bomb**
What is the first boot	Brain	Mind	ELK	Trojan			**Brain**
What is the name of first computer virus?	The Famous	HARLIE	PARAM	Creeper			**Creeper**
What is anti-virus?	Acomputer	program code	compan	Hardware			**program code**
_____ filters all network traffic for possible	Cisca IPS	Cisca RAM	Cisca ROM	Cisca CPC			**Cisca IPS**
which is an external security threat ?	front door	back door	underground	DOS			**underground**
A _____ attaches itself to executable files and replicates, when the infected program is executed, by finding other	Stealth virus	Polymorphic Virus	Parasitic Virus	Macro Virus			**Stealth virus**
A key security issue in design of OS is to	control	prevent	correct	handle			**prevent**
_____ infect documents, not executable	Macro viruses	Mini viruses	Maxi viruses	Micro viruses			**Macro viruses**

UNIT-V

SYLLABUS

Building Real World Enterprise Web Service and Applications: Real World Web Service Application Development – Development of Web services and Applications onto Tomcat application Server and Axis Soap Server.

REAL WORLD WEB SERVICES DEVELOPMENT AND DEPLOYMENT

Introduction

- This document describes how to install Apache Axis. It assumes you already know how to write and run Java code and are not afraid of XML. You should also have an application server or servlet engine and be familiar with operating and deploying to it. If you need an application server, we recommend Jakarta Tomcat. [If you are installing Tomcat, get the latest 4.1.x version, and the full distribution, not the LE version for Java 1.4, as that omits the Xerces XML parser]. Other servlet engines are supported, provided they implement version 2.2 or greater of the servlet API. Note also that Axis client and server requires Java 1.3 or later.

Enterprise Procurement

- Most companies have to buy goods or services from other companies in the course of doing their business. The procurement can be for physical components such as silicon chips, plastic components, or power modules, as well as for services such as contract manufacturing or overnight courier services.

System Functionality and Architecture

- The basic functionality of our Enterprise Procurement System (EPS) application is to:
  - Provide a Web-based interface that allows users to browse a catalog of goods and services from which they can select a particular part number.
  - Provide a Web-based interface that allows users to enter a part number to be located, the desired quantity, and whether the user is interested in optimizing for cost (if a high-volume product is being built) or for lead time (if a prototype needs to be built rapidly).
  - Connect to the Web services of a set of vendors and use the entered part number, quantity, and optimization criteria (cost or lead time) to get a price or lead time quote.
  - Analyze the returned values from the vendor Web services, and select the best choice.

- Present the best price or lead-time from the available vendors that meet the user's procurement needs.
- If the part cannot be located, consult another Web service to find alternate part numbers that may have similar functionality to that of the first part.
- Present the alternate part number to the user; otherwise, inform the user that there is no such alternate part.

Running the EPS Application

- In this section, we briefly look at the flow through the EPS application from the user's perspective. In the next section, we delve into the actual development of the EPS.
- The static HTML page, generated by the EPS.html file, is the main form in which the user enters information about the component to be procured. In the screenshot, information for part number 15151, desired quantity of 4, and optimization criteria of lead-time are specified.
- For more details on using Axis, please see the user guide.
- Things you need to know before writing a Web Service:

1. Core Java datatypes, classes and programming concepts.
2. What threads are, race conditions, thread safety and sychronization.
3. What a classloader is, what hierarchical classloaders are, and the common causes of a "ClassNotFoundException".
4. How to diagnose trouble from exception traces, what a NullPointerException (NPE) and other common exceptions are, and how to fix them.
5. What a web application is; what a servlet is, where classes, libraries and data go in a web application.
6. How to start your application server and deploy a web application on it.
7. What a network is, the core concepts of the IP protocol suite and the sockets API. Specifically, what is TCP/IP.
8. What HTTP is. The core protocol and error codes, HTTP headers and perhaps the details of basic authentication.
9. What XML is. Not necessarily how to parse it or anything, just what constitutes well-formed and valid XML.

- Axis and SOAP depends on all these details. If you don't know them, Axis (or anyone else's Web Service middleware) is a dangerous place to learn. Sooner or later you will be forced to discover these details, and there are easier places to learn than Axis.
- If you are completely new to Java, we recommend you start off with things like the Java Tutorials on Sun's web site, and perhaps a classic book like Thinking in Java, until you

have enough of a foundation to be able to work with Axis. It is also useful to have written a simple web application, as this will give you some knowledge of how HTTP works, and how Java application servers integrate with HTTP. Be aware that there is a lot more needed to be learned in order to use Axis and SOAP effectively than the listing above. The other big area is "how to write internet scale distributed applications". Nobody knows how to do that properly yet, so that you have to learn this by doing.

Step 0: Concepts

- Apache Axis is an Open Source SOAP server and client. SOAP is a mechanism for inter-application communication between systems written in arbitrary languages, across the Internet. SOAP usually exchanges messages over HTTP: the client POSTs a SOAP request, and receives either an HTTP success code and a SOAP response or an HTTP error code. Open Source means that you get the source, but that there is no formal support organisation to help you when things go wrong.
- SOAP messages are XML messages. These messages exchange structured information between SOAP systems. Messages consist of one or more SOAP elements inside an envelope, Headers and the SOAP Body. SOAP has two syntaxes for describing the data in these elements, Section 5, which is a clear descendant of the XML RPC system, and XML Schema, which is the newer (and usually better) system. Axis handles the magic of converting Java objects to SOAP data when it sends it over the wire or receives results. SOAP Faults are sent by the server when something goes wrong; Axis converts these to Java exceptions.
- SOAP is intended to link disparate systems. It is not a mechanism to tightly bind Java programs written by the same team together. It can bind Java programs together, but not as tightly as RMI or Corba. If you try sending many Java objects that RMI would happily serialize, you will be disappointed at how badly Axis fails. This is by design: if Axis copied RMI and serialized Java objects to byte streams, you would be stuck to a particular version of Java everywhere.
- Axis implements the JAX-RPC API, one of the standard ways to program Java services. If you look at the specification and tutorials on Sun's web site, you will understand the API. If you code to the API, your programs will work with other implementations of the API, such as those by Sun and BEA. Axis also provides an extension feature that in many ways extends the JAX-RPC API. You can use these to write better programs, but these will only work with the Axis implementation. But since Axis is free and you get the source, that should not matter.
- Axis is compiled in the JAR file axis.jar; it implements the JAX-RPC API declared in the JAR files jaxrpc.jar and saaj.jar. It needs various helper libraries, for logging, WSDL

processing and introspection. All these files can be packaged into a web application, axis.war, that can be dropped into a servlet container. Axis ships with some sample SOAP services. You can add your own by adding new compiled classes to the Axis webapp and registering them.

- Before you can do that, you have to install it and get it working.

Step 1: Preparing the webapp

- Here we assume that you have a web server up and running on the localhost at port 8080. If your server is on a different port, replace references to 8080 to your own port number.
- In your Application Server installation, you should find a directory into which web applications ("webapps") are to be placed. Into this directory copy the webapps/axis directory from the xml-axis distribution. You can actually name this directory anything you want, just be aware that the name you choose will form the basis for the URL by which clients will access your service. The rest of this document assumes that the default webapp name, "axis" has been used; rename these references if appropriate.

Step 2: Setting up the libraries

- In the Axis directory, you will find a WEB-INF sub-directory. This directory contains some basic configuration information, but can also be used to contain the dependencies and web services you wish to deploy.
- Axis needs to be able to find an XML parser. If your application server or Java runtime does not make one visible to web applications, you need to download and add it. Java 1.4 includes the Crimson parser, so you can omit this stage, though the Axis team prefer Xerces.
- To add an XML parser, acquire the JAXP 1.1 XML compliant parser of your choice. We recommend Xerces jars from the xml-xerces distribution, though others mostly work. Unless your JRE or app server has its own specific requirements, you can add the parser's libraries to axis/WEB-INF/lib. The examples in this guide use Xerces. This guide adds xml-apis.jar and xercesImpl.jar to the AXISCLASSPATH so that Axis can find the parser (see below).
- If you get ClassNotFound errors relating to Xerces or DOM then you do not have an XML parser installed, or your CLASSPATH (or AXISCLASSPATH) variables are not correctly configured.

Tomcat 4.x and Java 1.4

- Java 1.4 changed the rules as to how packages beginning in java.* and javax.* get loaded. Specifically, they only get loaded from *endorsed* directories. jaxrpc.jar and saaj.jar contain javax packages, so they may not get picked up. If happyaxis.jsp (see below) cannot find the relevant packages, copy them from axis/WEB-INF/lib to CATALINA_HOME/common/lib and restart Tomcat.

WebLogic 8.1

- WebLogic 8.1 ships with webservices.jar that conflicts with Axis' saaj.jar and prevents Axis 1.2 from working right out of the box. This conflict exists because WebLogic uses an older definition of javax.xml.soap.* package from Java Web Services Developer Pack Version 1.0, whereas Axis uses a newer revision from J2EE 1.4.
- However, there are two alternative configuration changes that enable Axis based web services to run on Weblogic 8.1.
  - In a webapp containing Axis, set <prefer-web-inf-classes> element in WEB-INF/weblogic.xml to true. An example of weblogic.xml is shown below:

```
<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
    </weblogic-web-app>
```

    - If set to true, the <prefer-web-inf-classes> element will force WebLogic's classloader to load classes located in the WEB-INF directory of a web application in preference to application or system classes. This is a recommended approach since it only impacts a single web module.
- In a script used to start WebLogic server, modify CLASSPATH property by placing Axis's saaj.jar library in front of WebLogic's webservices.jar.
    - **NOTE:** This approach impacts all applications deployed on a particular WebLogic instance and may prevent them from using WebLogic's web services.
- For more information on how Web Logic's class loader works, see Web Logic Server Application Classloading.

Step 3: starting the web server

- This varies on a product-by-product basis. In many cases it is as simple as double clicking on a startup icon or running a command from the command line.

Step 4: Validate the Installation

- After installing the web application and dependencies, you should make sure that the server is running the web application.
- Look for the start page
- Navigate to the start page of the webapp, usually http://127.0.0.1:8080/axis/, though of course the port may differ.
- You should now see an Apache-Axis start page. If you do not, then the webapp is not actually installed, or the appserver is not running.
- Validate Axis with happyaxis
- Follow the link Validate the local installation's configuration
  This will bring you to happyaxis.jsp a test page that verifies that needed and optional libraries are present. The URL for this will be something like
  http://localhost:8080/axis/happyaxis.jsp
- If any of the needed libraries are missing, Axis will not work.
  You must not proceed until all needed libraries can be found, and this validation page is happy.
  Optional components are optional; install them as your need arises. If you see nothing but an internal server error and an exception trace, then you probably have multiple XML parsers on the CLASSPATH (or AXISCLASSPATH), and this is causing version confusion. Eliminate the extra parsers, restart the app server and try again.

Look for some services

- From the start page, select View the list of deployed Web services. This will list all registered Web Services, unless the servlet is configured not to do so. On this page, you should be able to click on (wsdl) for each deployed Web service to make sure that your web service is up and running.
- Note that the 'instant' JWS Web Services that Axis supports are not listed in this listing here. The install guide covers this topic in detail.

Test a SOAP Endpoint

- Now it's time to test a service. Although SOAP 1.1 uses HTTP POST to submit an XML request to the endpoint, Axis also supports a crude HTTP GET access mechanism, which is useful for testing. First let's retrieve the version of Axis from the version endpoint, calling the getVersion method:

http://localhost:8080/axis/services/Version?method=getVersion

This should return something like:

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <getVersionResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <getVersionReturn
      xsi:type="xsd:string">
       Apache Axis version: 1.1 Built on Apr 04, 2003 (01:30:37 PST)
     </getVersionReturn>
   </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The Axis version and build date may of course be different.

Test a JWS Endpoint

- Now let's test a JWS web service. Axis' JWS Web Services are java files you save into the Axis webapp *anywhere but the WEB-INF tree*, giving them the .jws extension. When someone requests the .jws file by giving its URL, it is compiled and executed. The user guide covers JWS pages in detail.
- To test the JWS service, we make a request against a built-in example, EchoHeaders.jws (look for this in the axis/ directory).
- Point your browser at http://localhost:8080/axis/EchoHeaders.jws?method=list.

This should return an XML listing of your application headers, such as

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <listResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <listReturn xsi:type="soapenc:Array"
      soapenc:arrayType="xsd:string[6]"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
       <item>accept:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*</item>
       <item>accept-language:en-us</item>
       <item>accept-encoding:gzip, deflate</item>
       <item>user-agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)</item>
       <item>host:localhost:8080</item>
       <item>connection:Keep-Alive</item>
     </listReturn>
   </listResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

- Again, the exact return values will be different, and you may need to change URLs to correct any host, port and webapp specifics.

Step 5: Installing new Web Services

- So far you have got Axis installed and working--now it is time to add your own Web Service.
- The process here boils down to (1) get the classes and libraries of your new service into the Axis WAR directory tree, and (2) tell the AxisEngine about the new file. The latter is done by submitting an XML deployment descriptor to the service via the Admin web service, which is usually done with the AdminClient program or the <axis-admin> Ant task. Both of these do the same thing: they run the Axis SOAP client to talk to the Axis administration service, which is a SOAP service in its own right. It's also a special SOAP service in one regard--it is restricted to local callers only (not remote access) and is password protected to stop random people from administrating your service. There is a default password that the client knows; if you change it then you need to pass the new password to the client.
- The first step is to add your code to the server.
- In the WEB-INF directory, look for (or create) a "classes" directory (i.e. axis/WEB-INF/classes). In this directory, copy the compiled Java classes you wish to install, being careful to preserve the directory structure of the Java packages.
- If your classes services are already packaged into JAR files, feel free to drop them into the WEB-INF/lib directory instead. Also add any third party libraries you depend on into the same directory.
- After adding new classes or libraries to the Axis webapp, you must restart the webapp. This can be done by restarting your application server, or by using a server-specific mechanism to restart a specific webapp.
- Note: If your web service uses the simple authorization handlers provided with xml-axis (this is actually not recommended as these are merely illustrations of how to write a handler than intended for production use), then you will need to copy the corresponding users.lst file into the WEB-INF directory.

Step 6: Deploying your Web Service

- The various classes and JARs you have just set up implement your new Web Service. What remains to be done is to tell Axis how to expose this web service. Axis takes a Web Service Deployment Descriptor (WSDD) file that describes in XML what the service is, what methods it exports and other aspects of the SOAP endpoint.
- The users guide and reference guide cover these WSDD files; here we are going to use one from the Axis samples: the stock quote service.

- Classpath setup
- In order for these examples to work, java must be able to find axis.jar, commons-discovery.jar, commons-logging.jar, jaxrpc.jar, saaj.jar, log4j-1.2.8.jar (or whatever is appropriate for your chosen logging implementation), and the XML parser jar file or files (e.g., xerces.jar). These examples do this by adding these files to AXISCLASSPATH and then specifying the AXISCLASSPATH when you run them. Also for these examples, we have copied the xml-apis.jar and xercesImpl.jar files into the AXIS_LIB directory. An alternative would be to add your XML parser's jar file directly to the AXISCLASSPATH variable or to add all these files to your CLASSPATH variable.
- On Windows, this can be done via the following. For this document we assume that you have installed Axis in C:\axis. To store this information permanently in WinNT/2000/XP you will need to right click on "My Computer" and select "Properties". Click the "Advanced" tab and create the new environmental variables. It is often better to use WordPad to create the variable string and then paste it into the appropriate text field.

```
set AXIS_HOME=c:\axis

set AXIS_LIB=%AXIS_HOME%\lib

set AXISCLASSPATH=%AXIS_LIB%\axis.jar;%AXIS_LIB%\commons-discovery.jar;

 %AXIS_LIB%\commons-logging.jar;%AXIS_LIB%\jaxrpc.jar;%AXIS_LIB%\saaj.jar;

 %AXIS_LIB%\log4j-1.2.8.jar;%AXIS_LIB%\xml-
apis.jar;%AXIS_LIB%\xercesImpl.jar
```

- Unix users have to do something similar. Below we have installed AXIS into /usr/axis and are using the bash shell. See your shell's documentation for differences. To make variables permeate you will need to add them to your shell's startup (dot) files. Again, see your shell's documentation.

```
 set AXIS_HOME=/usr/axis

 set AXIS_LIB=$AXIS_HOME/lib

 set AXISCLASSPATH=$AXIS_LIB/axis.jar:$AXIS_LIB/commons-discovery.jar:

  $AXIS_LIB/commons-logging.jar:$AXIS_LIB/jaxrpc.jar:$AXIS_LIB/saaj.jar:

  $AXIS_LIB/log4j-1.2.8.jar:$AXIS_LIB/xml-apis.jar:$AXIS_LIB/xercesImpl.jar

export AXIS_HOME; export AXIS_LIB; export AXISCLASSPATH
```

- To use Axis client code, you can select AXISCLASSPATH when invoking Java by entering

  java -cp %AXISCLASSPATH% ...

  or

  java -cp "$AXISCLASSPATH" ...

- depending on ou may omit the quotes if your CLASSPATH doesn't have spaces in it.
- Also, it is probably a good time to add the AXISCLASSPATH variable to your CLASSPATH variable. This will enable you to not include the AXISCLASSPATH variable when launching the examples in this guide. This document assumes that you have NOT done this.

Find the deployment descriptor

- Look in axis/samples/stock for the file deploy.wsdd. This is the deployment descriptor we want to tell Axis about. Deployment descriptors are an Axis-specific XML file that tells Axis how to deploy (or undeploy) a Web Service, and how to configure Axis itself. The Axis Administration Web Service lets the AdminClient program and its Ant task counterpart submit a new WSDD file for interpretation. The Axis 'engine' will update its configuration, then save its state.
- By default Axis saves it state into the global configuration file axis/WEB-INF/server-config.wsdd. Sometimes you see a warning message about such a file not being found--don't worry about this, because Axis auto-creates the file after you deploy something to it. You can check in the webapp to see what this file looks like--and even copy it to other systems if you want to give them identical configurations. Note that Axis needs an expanded web application and write access to the WEB-INF dir to save its state in this location.

Run the admin client

- Execute the following command from the samples/stock directory. If you are not in this directory you will get a "java.io.FileNotFoundException: deploy.wsdd (The system cannot find the file specified)" exception.

On Windows

```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient
    -lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

On UNIX

```
java -cp $AXISCLASSPATH org.apache.axis.client.AdminClient
    -lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

- If you get some java client error (like ClassNotFoundException), then you haven't set up your AXISCLASSPATH (or CLASSPATH) variable right, mistyped the classname, or did some other standard error. Tracking down such problems are foundational Java development skills--if you don't know how to do these things, learn them now!
- Note: You may need to replace localhost with your host name, and 8080 with the port number used by your web server. If you have renamed the web application to something other than "axis" change the URL appropriately.
- If you get some AxisFault listing, then the client is working, but the deployment was unsuccessful. This is where the knowledge of the sockets API to TCP and the basics of the HTTP that Web Service development requires begins to be needed. If you got some socket error like connection refused, the computer at the far end isn't talking to you, so find the cause of that and fix it. If you get an HTTP error code back find out what the error means and correct the problem. These skills are fundamental to using web services.
- The user's guide covers the AdminClient in more detail, and there is also an Ant task to automate the use of Axis in your Ant build scripts.

Step 7: Testing

- This step is optional, but highly recommended. For illustrative purposes, it is presumed that you have installed and deployed the stock quote demo.

- Change directory to the distribution directory for xml-axis and execute the following command (or its Unix equivalent):

On Windows

```
java -cp .;%AXISCLASSPATH% samples.stock.GetQuote
    -lhttp://localhost:8080/axis/servlet/AxisServlet
    -uuser1 -wpass1 XXX
```

On UNIX

```
java -cp $AXISCLASSPATH samples.stock.GetQuote
    -lhttp://localhost:8080/axis/servlet/AxisServlet
    -uuser1 -wpass1 XXX
```

- You should get back "55.25" as a result.

Note: Again, you may need to replace localhost with your host name, and 8080 with the port number used by your web server. If you have renamed the web application to something other than "axis" change the URL appropriately.

Advanced Installation: adding Axis to your own Webapp

- If you are experienced in web application development, and especially if you wish to add web services to an existing or complex webapp, you can take an alternate approach to running Axis. Instead of adding your classes to the Axis webapp, you can add Axis to your application.

The core concepts are

1. Add axis.jar, wsdl.jar, saaj.jar, jaxrpc.jar and the other dependent libraries to your WAR file.
2. Copy all the Axis Servlet declarations and mappings from axis/WEB-INF/web.xml and add them to your own web.xml
3. Build and deploy your webapp.
4. Run the Axis AdminClient against your own webapp, instead of Axis, by changing the URL you invoke it with.

What if it doesn't work?

- Axis is a complicated system to install. This is because it depends on the underlying functionality of your app server, has a fairly complex configuration, and, like all distributed applications, depends upon the network too.
- We see a lot of people posting their problems on the axis-user mailing list, and other Axis users as well as the Axis developers do their best to help when they can. But before you rush to post your own problems to the mailing list, a word of caution:
- Axis is free. This means nobody gets paid to man the support lines. All the help you get from the community is voluntary and comes from the kindness of their hearts. They may

be other users, willing to help you get past the same hurdles they had to be helped over, or they may be the developers themselves. But it is all voluntary, so you may need to keep your expectations low!

1. Post to the user mail list, not the developer list. You may think the developer mail list is a short cut to higher quality answers. But the developers are also on the user list along with many other skilled users--so more people will be able to answer your questions. Also, it is helpful for all user issues to be on one list to help build the searchable mailing list archive.
2. Don't ask non-Axis-related questions. The list is not the place to ask about non-Axis, non-SOAP, problems. Even questions about the MS Soap toolkit or .NET client side, don't get many positive answers--we avoid them. That also goes for the Sun Java Web Services Developer Pack, or the Jboss.net stuff that they've done with Axis.
3. Never bother posting to the soapbuilders mailing list either, that is only for people developing SOAP toolkits, not using them--all off-topic messages are pointedly ignored.
4. There is no guarantee that anyone will be able to solve your problem. The usual response in such a situation is silence, for a good reason: if everybody who didn't know the answer to a question said "I don't know", the list would be overflowed with noise. Don't take silence personally.
5. Never expect an immediate answer. Even if someone knows the answer, it can take a day or two before they read their mail. So if you don't get an answer in an hour or two, don't panic and resend. Be patient. And put the time to use by trying to solve your problems yourself.
6. Do your homework first. This document lists the foundational stuff you need to understand. It has also warned you that it can take a day to get a reply. Now imagine you get a HTTP Error '404' on a SOAP call. Should you rush to post a 'help' request, or should you try and find out what an HTTP error code is, what #404 usually means and how to use a Java debugger. We provide the source to make that debugging easier :)
7. Post meaningful subject lines. You want your message read, not deleted unread. A subject line of 'Axis problem', 'Help with Axis', etc. is not meaningful, and is not likely to get many readers.
8. Search the mailing list archives FIRST to see if someone had the same problem. This list is searchable--and may save you much time in getting an answer to your problem.
9. Use the jira database to search for Axis bugs, both open and closed.
10. Consult the Axis Wiki for its Frequently Asked Questions (FAQ), installation notes, interoperability issues lists, and other useful information.
11. Don't email people for help directly, unless you know them. It's rude and presumptuous. Messages sent over the mail list benefit the whole community--both the original posters and people who search the list. Personal messages just take up the recipients time, and are unwelcome. Usually, if not ignored outright, recipients of personal requests will just respond 'ask the mail list' anyway!
12. Know that configuration problems are hard to replicate, and so can be difficult to get help on. We have tried with the happyaxis.jsp demo to automate the diagnostics gathering for you, but it can be hard for people to be of help here, especially for obscure platforms.

13. Keep up to date with Axis releases, even the beta copies of forthcoming releases. You wouldn't want your problem to be a bug that was already known and fixed in a more recent release. Often the common response to any question is 'have you tried the latest release'.
14. Study and use the source, and fix it when you find defects. Even fix the documentation when you find defects. It is only through the participation of Axis' users that it will ever get better.

- Has this put you off joining and participating in the Axis user mail list? We hope not--this list belongs to the people who use Axis and so will be your peers as your project proceeds. We just need for you to be aware that it is not a 24x7 support line for people new to server side Java development, and that you will need to be somewhat self sufficient in this regard. It is not a silver bullet. However, knowing how to make effective use of the list will help you develop better with Axis.

Appendix: Enabling the SOAP Monitor

- SOAP Monitor allows for the monitoring of SOAP requests and responses via a web browser with Java plug-in 1.3 or higher. For a more comprehensive explanation of its usage, read Using the SOAP Monitor in the User's Guide.
- By default, the SOAP Monitor is not enabled. The basic steps for enabling it are compiling the SOAP Monitor java applet, deploying the SOAP Monitor web service and adding request and response flow definitions for each monitored web service. In more detail:

1. Go to $AXIS_HOME/webapps/axis (or %AXIS_HOME%\webapps\axis) and compile SOAPMonitorApplet.java.

On Windows

```
javac -classpath %AXIS_HOME%\lib\axis.jar SOAPMonitorApplet.java
```

On Unix

```
javac -classpath $AXIS_HOME/lib/axis.jar SOAPMonitorApplet.java
```

- When using the Java version of Axis there are two ways to expose Java code as Web service. The easiest one is to use Axis native JWS (Java Web Service) files. Another way is to use custom deployment. Custom deployment enables you to customize resources that should be exposed as Web service.

2. Copy all resulting class files (i.e. SOAPMonitorApplet*.class) to the root directory of the web application using the SOAP Monitor (e.g. .../tomcat/webapps/axis)

3. Deploy the SOAPMonitorService web service with the admin client and the deploy-monitor.wsdd file (shown below).

Go to the directory deploy-monitor.wsdd is located and execute the command below. The command assume that /axis is the intended web application and it is available on port 8080.

On Windows

```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient
    -lhttp://localhost:8080/axis/services/AdminService deploy-monitor.wsdd
```

On UNIX

```
java -cp $AXISCLASSPATH org.apache.axis.client.AdminClient
    -lhttp://localhost:8080/axis/services/AdminService deploy-monitor.wsdd
```

SOAPMonitorService Deployment Descriptor (deploy-monitor.wsdd)

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
        xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <handler name="soapmonitor"
       type="java:org.apache.axis.handlers.SOAPMonitorHandler">
   <parameter name="wsdlURL"
           value="/axis/SOAPMonitorService-impl.wsdl"/>
   <parameter name="namespace"
           value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/>
   <parameter name="serviceName" value="SOAPMonitorService"/>
   <parameter name="portName" value="Demo"/>
 </handler>
```

```
<service name="SOAPMonitorService" provider="java:RPC">
  <parameter name="allowedMethods" value="publishMessage"/>
  <parameter name="className"
        value="org.apache.axis.monitor.SOAPMonitorService"/>
  <parameter name="scope" value="Application"/>
 </service>
</deployment>
```

4. For each service that is to be monitored, add request and response flow definitions to the service's deployment descriptor and deploy (or redeploy) the service. The **requestFlow** and **responseFlow** definitions follow the start tag of the **<service>** element. If a service is already deployed, undeploy it and deploy it with the modified deployment descriptor. An example is shown below:

```
...
  <service name="xmltoday-delayed-quotes" provider="java:RPC">
   <requestFlow>
    <handler type="soapmonitor"/>
   </requestFlow>
   <responseFlow>
    <handler type="soapmonitor"/>
   </responseFlow>
  ...
```

13. With a web browser, go to http[s]://host[:port][/webapp]/SOAPMonitor (e.g. http://localhost:8080/axis/SOAPMonitor) substituting the correct values for your web application. This will show the SOAP Monitor applet for viewing service requests and responses. Any requests to services that have been configured and deployed correctly should show up in the applet.

Apache Axis (**A**pache e**X**tensible **I**nteraction **S**ystem) is an open source, XML based Web service framework. It consists of a Java and a C++implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create interoperable, distributed computing applications. Axis is developed under the auspices of the Apache Software Foundation.

Axis for Java

JWS Web service creation

- JWS files contain Java class source code that should be exposed as Web service. The main difference between an ordinary java file and jws file is the file extension. Another difference is that jws files are deployed as source code and not compiled class files.
- The following example is taken from http://axis.apache.org/axis/java/user-guide.html#Publishing_Web_Services_with_Axis . It will expose methods add and subtract of class Calculator.

```
public class Calculator
{
  public int add(int i1, int i2)
  {
    return i1 + i2;
  }

  public int subtract(int i1, int i2)
  {
    return i1 - i2;
  }
}
```

JWS Web service deployment

Once the Axis servlet is deployed, you need only to copy the jws file to the Axis directory on the server. This will work if you are using an Apache Tomcat container. In the case that you are using another web container, custom WAR archive creation will be required .

JWS Web service access

JWS Web service is accessible using the URL http://localhost:8080/axis/Calculator.jws . If you are running a custom configuration of Apache Tomcat or a different container, the URL might be different.

Custom deployed Web service

- Custom Web service deployment requires a specific deployment descriptor called WSDD (Web Service Deployment Descriptor) syntax. It can be used to specify resources that should be exposed as Web services. Current version (1.3) supports
- RPC services
- EJB - stateless (Enterprise Java Bean)
- Automated generation of WSDL[edit]
- When a Web service is exposed using Axis it will generate a WSDL file automatically hen accessing the Web service URL with ?WSDL appended to it.

Axis for C++

- An example for implementing and deploying a simple web-service with the C++ version of Axis can be found in the Axis-CPP Tutorial (link in the Reference section below).

The steps necessary are:

- Create the wsdl file
- Generate client and server stubs using wsdl2ws
- Provide the server side web service implementation (e.g. the add method of the calculator service)
- Build the server side code and update the generated deploy.wsdd with the .dll path
- Deploy the binaries to the directory specified in the wsdd
- Build client
- Run and enjoy...

DEVELOPING WEB SERVICES USING AXIS AND TOMCAT

- The Apache Axis engine is one of the most commonly used Web Services engines in the Java Web Services realm, and it is the third-generation implementation of Apache SOAP. Essentially, Axis is a SOAP engine, self-proclaimed to be "a framework for constructing SOAP processors such as clients, servers, gateways, etc." In addition to being a SOAP engine, it also includes the following (excerpt taken from the Apache Axis Web site):

- A simple stand-alone server
- A server that plugs into Servlet engines such as Tomcat
- Extensive support for the Web Services Description Language (WSDL)
- Emitter tooling that generates Java classes from WSDL
- Some sample programs
- A tool for monitoring TCP/IP packets

➢ Furthermore, here is the feature set that Axis delivers, making it a desireable technology:

• Speed: Axis uses SAX (event-based) parsing to acheive significantly greater speed than earlier versions of Apache SOAP.

• Flexibility: The Axis architecture gives the developer complete freedom to insert extensions into the engine for custom header processing, system management, or anything else you can imagine.

• Stability: Axis defines a set of published interfaces which change relatively slowly compared to the rest of Axis.

• Component-oriented deployment: You can easily define reusable networks of Handlers to implement common patterns of processing for your applications, or to distribute to partners.

• Transport framework: A clean and simple abstraction for designing transports (i.e., senders and listeners for SOAP over various protocols such as SMTP, FTP, message-oriented middleware, etc), and the core of the engine is completely transport-independent.

- WSDL support: Axis supports the Web Service Description Language, version 1.1, which allows you to easily build stubs to access remote services, and also to automatically export machine-readable descriptions of your deployed services from Axis.

➢ A final note on the inherent success of Axis is that it has been the SOAP implementation inside of JBoss and WebSphere (at the time of this writing.)

Tomcat and Axis Setup

- Before you can begin using Axis, you need to download a copy of Jakarta Tomcat and Apache Axis:
    - Jakarta Tomcat
    - Apache Axis
- After installing Tomcat and decompressing Axis, copy the "axis" folder from <axis-home>/webappsto <tomcat-home>/webapps. This gives you the following folder:
- <tomcat-home>/webapps/axis
- Before starting Tomcat, you need to download a few support libraries that Axis needs:
    - Java Activation Framework
    - JavaMail
    - XML Security
- Then copy the following files to the <tomcat-home>/webapps/axis/WEB-INF/lib folder:

- activation.jar
- mail.jar
- xmlsec-1.2.1.jar
- xalan.jar

- Start Tomcat by executing the startup script from the <tomcat-home>/bin folder. For example, on Windows:
- C:\jakarta-tomcat-5.5.9\bin> startup
- Now you can test your installation by directing your Web browser to the following URL:
- http://localhost:8080/axis
- Click on "Validation" or go directly to the URL:
- http://localhost:8080/axis/happyaxis.jsp

- This page tells you if Axis located the libraries that it needs to run properly. If you have any errors, you're provided links to the required libraries. Follow the links, download the libraries, and copy the JAR files to the <tomcat-home>/webapps/axis/WEB-INF/lib folder.

NOTE

- You may need to restart Tomcat for the changes to take effect; execute the shutdown script followed by the startup script from the <tomcat-home>/bin folder.
- Finally, you might want to test your installation by listing the currently deployed Web services. Click "List" from the Axis homepage:
- http://localhost:8080/axis
- You should initially see two services:
  - AdminService
  - Version
- Click on the wsdl link for the Version service to validate that Axis is properly serving its content.

Building the Web Service

- We'll continue with the "Age" Web service that we defined in the JBoss example, but in the context of Axis. An Axis Web Service only requires two things:

1. A normal Java class that provides the Web Service implementation methods
2. A Web Services Deployment Descriptor (WSDD)

Listing 15 shows the source code for the AgeService class.

Listing 15. AgeService.java

```
package com.javasrc.webservices.age;

public class AgeService {
 public String age( String name, Integer age ) {
  return name + " is " + age + " years old!";
```

```
  }
}
```

This is a standard Java class with a single method: age() accepts a name and an age and returns a String.

Listing 16 shows the Web Services Deployment Descriptor for the AgeService.

Listing 16. AgeService.wsdd

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="AgeService" provider="java:RPC">
  <parameter name="className" value="com.javasrc.webservices.age.AgeService"/>
  <parameter name="allowedMethods" value="*"/>
 </service>
</deployment>
```

The deployment descriptor provides some basic wrapping around three key pieces of information:

1. The name of the Web Service
2. The class that implements the Web Services
3. The methods in the class that we want Axis to expose

In this case, we map the name "AgeService" to the com.javasrc.webservices.age.AgeServiceclass and expose all of its methods.

And that is all you need to build a Web Service using Axis! I think it is quite elegant!

Deploying the Web service

- Deploying the Web Service to Axis is a little different from traditional application servers. The Web Service implementation class is packaged together with the Web Services engine, rather than existing separately and allowing the Servlet or EJB container to map Web Service requests to the implementation class. In this case, you need to copy your compiled Web Services classes (both the Web Service implementation and all classes on

which it depends) to the <tomcat-home>/webapps/axis/WEB-INF/classes folder, and remember to preserve the package directory structure.

- The AgeService.class file should be copied to the following folder:

<tomcat-home>/webapps/axis/WEB-INF/classes/com/javasrc/webservices/age

So far, so good. Now you need to tell Axis about the Web Service by integrating its WSDD file with Axis and allowing Axis to generate the WSDL file. This can all be accomplished by using the Axisorg.apache.axis.client.AdminClient class from a command prompt. As with our previous examples, it can be somewhat of a chore determining the JAR files to add to the CLASSPATH to utilize Web Services functionality. To launch the AdminClient you need to include the following JAR files in your CLASSPATH:

<axis-home>\lib\axis.jar

<axis-home>\lib\jaxrpc.jar

<axis-home>\lib\saaj.jar

<axis-home>\lib\wsdl4j.jar

<axis-home>\lib\commons-logging.jar

<axis-home>\lib\commons-discovery.jar

<java-activation-framework-home>\activation.jar

<javamail-home>\mail.jar

- Then launch the AdminClient, passing it the WSDD file as its command line argument:

java org.apache.axis.client.AdminClient AgeService.wsdd

- You should see something similar to the following:

Processing file AgeService.wsdd

<Admin>Done processing</Admin>

- Navigate to the Axis homepage and click on "List" to see if your Web Service was successfully deployed. If so, click on its wsdl link to see the WSDL file that Axis generated, or access it directly through the following URL:
- http://localhost:8080/axis/services/AgeService?wsdl
- Listing 17 shows the contents of that file in my environment.

Listing 17. AgeService WSDL File (generated)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/AgeService"
        xmlns:apachesoap="http://xml.apache.org/xml-soap"
        xmlns:impl="http://localhost:8080/axis/services/AgeService"
        xmlns:intf="http://localhost:8080/axis/services/AgeService"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <!-- WSDL created by Apache Axis version: 1.2RC3 Built on Feb 28, 2005 (10:15:14 EST)--
>
  <wsdl:message name="ageRequest">
   <wsdl:part name="in0" type="soapenc:string"/>
   <wsdl:part name="in1" type="soapenc:int"/>
  </wsdl:message>
  <wsdl:message name="ageResponse">
   <wsdl:part name="ageReturn" type="soapenc:string"/>
  </wsdl:message>
  <wsdl:portType name="AgeService">
   <wsdl:operation name="age" parameterOrder="in0 in1">
     <wsdl:input message="impl:ageRequest" name="ageRequest"/>
     <wsdl:output message="impl:ageResponse" name="ageResponse"/>
   </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="AgeServiceSoapBinding" type="impl:AgeService">
   <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="age">
     <wsdlsoap:operation soapAction=""/>
     <wsdl:input name="ageRequest">
```

```
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://age.webservices.javasrc.com" use="encoded"/>
</wsdl:input>
<wsdl:output name="ageResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/services/AgeService" use="encoded"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="AgeServiceService">
<wsdl:port binding="impl:AgeServiceSoapBinding" name="AgeService">
<wsdlsoap:address location="http://localhost:8080/axis/services/AgeService"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

The important things you need to know from this file when building our client are:

1. The address (URL) of the Web Service: http://localhost:8080/axis/services/AgeService
2. The namespace of the AgeService (used later when creating a QName): http://age.webservices.javasrc.com

With this information in hand, we are ready to write a test client to test our Web Service.

Testing the Web Service

- In order to test our Web Service, we'll build a test client that uses the Apache Axis client libraries. For this exercise, keep the same CLASSPATH that you used when launching the AdminClientclass:

  <axis-home>\lib\axis.jar
  <axis-home>\lib\jaxrpc.jar
  <axis-home>\lib\saaj.jar
  <axis-home>\lib\wsdl4j.jar

<axis-home>\lib\commons-logging.jar

<axis-home>\lib\commons-discovery.jar

<java-activation-framework-home>\activation.jar

<javamail-home>\mail.jar

Listing 18 shows the source code for the AgeServiceClient class.

Listing 18. AgeServiceClient.java

```
package com.javasrc.webservices.age;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;
public class AgeServiceClient
{
 public static void main(String [] args)
 {
  try {
   Options options = new Options(args);
        String endpointURL = options.getURL();
   String name;
   Integer age;
        args = options.getRemainingArgs();
   if ((args == null) || (args.length < 2)) {
    name = "NoName";
    age = new Integer( 0 );
   } else {
    name = args[ 0 ];
    age = new Integer( args[ 1 ] );
```

```
        }
        Service service = new Service();
        Call   call  = (Call) service.createCall();
        call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
        call.setOperationName( new QName("http://age.webservices.javasrc.com", "age") );
        call.addParameter( "arg1", XMLType.XSD_STRING, ParameterMode.IN);
        call.addParameter( "arg2", XMLType.XSD_INT, ParameterMode.IN);
        call.setReturnType( org.apache.axis.encoding.XMLType.XSD_STRING );

        String ret = (String) call.invoke( new Object[] { name, age } );

        System.out.println("Age result : " + ret);
      } catch (Exception e) {
        System.err.println(e.toString());
      }
    }
}
```

- Some of this method should look familiar, in concept anyway. We create a new Service instance (in this case, we create it directly rather than using a ServiceFactory because Axis only support JAX-RPC calls) but then we ask the Service to create a call for us. By doing this (and not involving the Remote interface as we did in the JBoss implementation), we keep everything generic; theAgeServiceClient does not need to know anything about the AgeService, and the Web Services framework handles all of the details.
- To help us manage our command line options, Apache provides the following class:org.apache.axis.utils.Options. From this class, we can extract the URL of the destination Web Service through its getURL() method. We tell the Call that this URL is the target endpoint to connect to. Next, we define the operation name to execute; this involves the creation on a QName. The QName uses the AgeService namespace that we found in the WSDL file and the name of the method on the AgeService that we want to execute: age.
- Next, we tell the call that we have two input parameters -- a String followed by an Integer -- and that we expect a String to be returned to us.

- Finally, we invoke the Web Service call, passing it our parameters, wrapping in an Object array, and capture the response.
- You can execute this test client as follows:
- java com.javasrc.webservices.age.AgeServiceClient -lhttp://localhost:8080/axis/services/AgeService "Steve" 33
- And you should get a response similar to the following:
- Age result : Steve is 33 years old!

What is Tomcat?

- Tomcat is an open source server from the Apache Software Foundation. It is a Web application server, which means that it comes ready to support programming using JavaServer Pages (JSPs) and servlets.
- Since early 2000, Tomcat has served as the reference implementation for the latest Java Servlet and JSP specifications. Tomcat 5.5, the latest Tomcat version as of this writing, supports the latest Java Servlet 2.4 and JavaServer Pages 2.0 standards (seeResources). Tomcat also includes a limited Web server that can serve static Web pages when executed in stand-alone mode (by default).
- Because of a variety of open source libraries and extensions, Tomcat supports:

- Web services using the Apache Axis servlet
- Development frameworks, such as Apache Struts
- Templating engines, such as Apache Jakarta Velocity
- Object-relational mapping technology, such as Hibernate

- This tutorial shows you how to use Tomcat to learn JSP, servlet, and Web services programming. Use of Struts, Velocity, and Hibernate with Tomcat is beyond this tutorial's scope.
- In the past, because a high level of expertise was required to configure and administer Tomcat, the primary Tomcat users were advanced server-side application developers. Now -- thanks to the maturing of Tomcat's GUI installer, the ability to install the server as a system service, and stabilization of the server's features -- even beginning Web developers can take advantage of this versatile server.

Tomcat installation and setup

Downloading Tomcat

To download the latest version of Tomcat, go to the Apache Tomcat home page (see Prerequisites), shown in Figure 1, and click the **Tomcat 5.x** link under the **Download** heading (the area outlined in red in Figure 1):

Figure 1. Apache Tomcat project home page

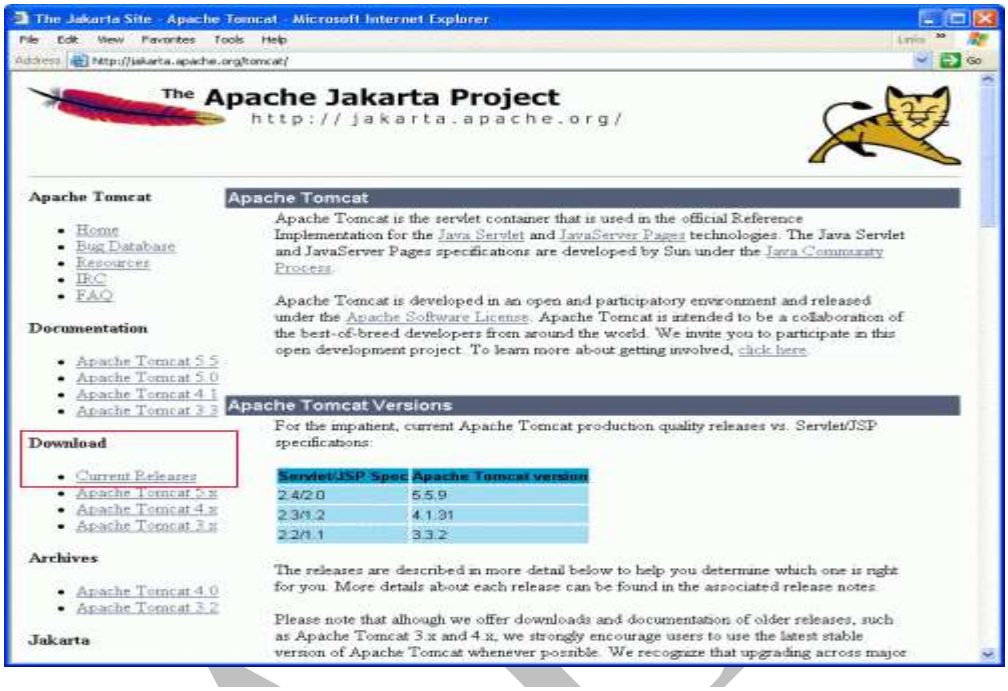


Figure 1. Apache Tomcat project home page You have a choice among the latest 5.5.x releases. Choose the binary distribution of the latest stable (nonbeta and nonalpha) release. For Windows systems, download the EXE binary for simple installation.

Back to top

Installing Tomcat

The EXE binary installer does the following:

- Unpacks and installs the Tomcat server components.

Lets you specify the TCP port that the server will use when listening for incoming requests. (A TCP port is a networking endpoint, represented by a number, that a client application can specify when connecting to the server.)

- Configures the server to run as a system service.

Start the installation EXE. You'll see the initial splash screen, shown in Figure 2:

Figure 2. Tomcat setup wizard splash screen

Figure 2. Tomcat setup wizard splash screen

The installation EXE runs a wizard-based installer with step-by-step instructions. You must have administrator privileges on the machine, because Tomcat is installed as a system service. If you are on your own PC as the default user and have installed other software successfully, you probably already have administrator privileges.

Table 1 describes the items that each screen of the setup wizard prompts for, along with the responses you should make.

Table 1. Tomcat setup wizard prompts

Setup wizard screen	Description
License agreement	This is the Apache License 2.0, one of the more liberal open source software licenses in existence. Read the license terms carefully. If you agree to the terms, click the I Agree button to proceed.
Choose components	Select the components of Tomcat to install. By default, the mandatory components are checked. If you have enough disk space, consider installing the examples. They are great for learning Web application programming.

Choose install location	Select the directory on your computer where the Tomcat server will install. If this is your first installation, the default that the wizard selects should be fine. This screen also shows how much disk space the Tomcat installation will take up and the amount of free space you have on the disk.
Configuration	This screen lets you performs basic Tomcat server configuration. You can select the TCP port that the server listens on, as well as an administrator username and password. It is recommended that you leave the TCP port at 8080. Leave the administrator username as admin and enter your own administrator password. Do not forget the password; you'll need it later to deploy the examples in this tutorial.
Java Virtual Machine	This screen lets you select the JVM that Tomcat runs under. Unless you have multiple JDKs installed on your machine, you can use the default. For the latest Tomcat 5.5 release, you should select JVM version 1.5.0 or later.
Completing the Apache Tomcat Setup Wizard	This is the final step of the installation. Select the Run Apache Tomcat checkbox. This starts the system service immediately after installation.

Note that on some versions of Windows with a firewall, you might need to give Tomcat explicit permission to listen to the TCP port for requests.

Verifying server operations

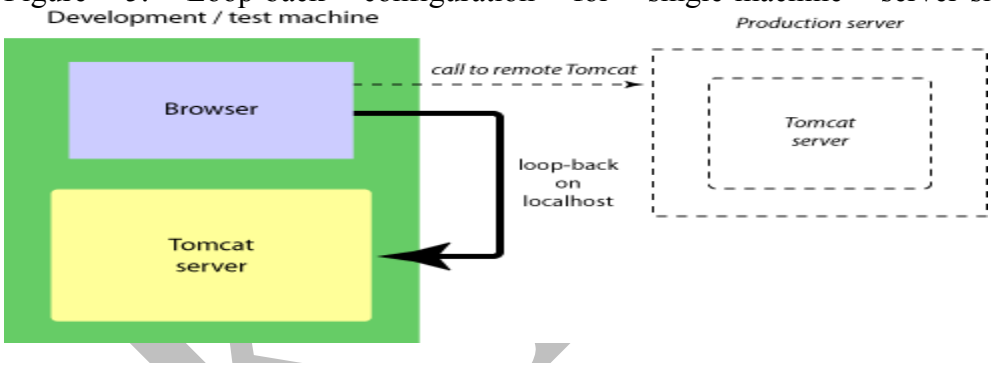
It's simple to access the running Tomcat server and verify that the installation was successful. Start a browser and point it to the address http://localhost:8080/.

The Tomcat server is listening at port 8080. (You configured this during the installation.) Figure 4 shows the welcome screen that Tomcat displays:

Figure 4. Tomcat's welcome screen

By running the Tomcat server on the same machine as your browser, you are simulating a networked environment. Figure 5 shows this loop-back network configuration:

Figure 5. Loop-back configuration for single-machine server-side development



- In Figure 5, both the client (browser) and the server (Tomcat) are running on the same machine. The TCP connection between the client and the server is running in a loop-back mode. This is a common practice in Web development, enabling you to perform server-side development using a single machine. In actual production, you can change the host name of the URL from localhost to the IP address of your networked production Tomcat server (shown within dash).

Discovering Web services development with Tomcat

Brief introduction to Web services

- Web services are server-side code components that can expose their functionality for access over a TCP/IP network using the standard HTTP protocol. This exposure lets Web service users, called consumers, consume Web services over most network connections -- and even through firewalls.

- A Web service processes incoming requests and generates responses. This is exactly what a servlet does, so it is quite natural to implement Web services using servlets.

- Web services are becoming increasingly popular because they can be effectively used for business-to-business or business-to-consumer interfaces. They let requests be sent, and responses received, through the Internet. Any user who can access your Web site can also access your Web service(s). For example, both eBay and Amazon.com offer Web services that their partners and users can consume.

- Web services depend on passing XML-based messages between the consumer and the service. The messages are packaged and sent according to the Simple Object Access Protocol (SOAP).

- Apache Axis is a Web services development kit that can be used as an add-on to Tomcat. The next section shows you how to create a simple Web service, using Apache Axis, and deploy it on your Tomcat server. See Resources for articles and tutorials that can help you learn more about Web services programming.

Adding Axis to Tomcat

- Axis can run as a servlet on Tomcat. If you haven't already done so, download the latest version of Axis (see Prerequisites). Unarchive the Axis distribution.
- Copy all the files under the webapps/axis directory of the Axis distribution to the step3/axis directory of this article's code distribution (see Download).

- You can use the makewar.bat batch file, found in the step3/axis directory, to create an axis.war file that can be deployed to Tomcat as a Web application.
- Before Axis will run properly on Tomcat, you might need to download some additional JAR files over the Internet and place them into the WEB-INF/lib directory of the step3 application. If you're using Axis 1.2.1, you need to download the following:

- **activation.jar** from http://java.sun.com/products/javabeans/glasgow/jaf.html
- **xmlsec-1.2.1.jar** from http://xml.apache.org/security/
- **mail.jar** from http://java.sun.com/products/javamail/

- If you are not using version 1.2.1 of Axis, the above list might differ slightly. See the documentation accompanying the Axis distribution for more information.

POSSIBLE QUESTIONS

PART – A

(Online Exam 20*1=20 marks)

- **Multiple Choice Questions available in Moodle**

PART – B

(Each Question carries 2 marks)

1. Explain the Real world Web service Application Development
2. Explain about Development of Web services and Application
3. Explain the Functionality of Enterprise Procurement System (EPS)
4. Explain about Axis SOAP server
5. Differentiate Tomcat application server and Axis SOAP server
6. Explain the steps involved for running the EPS Application
7. Explain how to test a JWS End point
8. Explain the steps involved for installing the new Web services
9. Explain how to discover the Web services development with Tomcat
10. Explain how to test a web service

PART – C

(Each Question carries 6 marks)

1. Explain the steps involved for installing the Tomcat Web server
2. Explain how to build a web service
3. Explain how to deploy a web service
4. Explain about JWS web service creation
5. Explain about Security holes in web services

Questions	Opt 1	Opt 2	Opt 3	Opt 4	Opt 5	Opt 6	Answer
_____ implements servlet 3.0 and JSP 2.2	Tomcat 7.x	Tomcat 8.x	Tomcat 1.2	Tomcat 5.x			Tomcat 7.x
_____ new components were added to release Tomcat 7	13	7	3	4			3
3 new components were added to release _____	Tomcat 8.2	tomcat 7	tomcat 5	tomcat 9			tomcat 7
_____ has been added to manage large documents	cluster	buster	foster	looser			cluster
_____ implements serrel 3.1 and JSP 2.4	Tomcat 7.x	tomcat 8.x	tomcat 1.2	tomcat 5.x			tomcat 8.x
Tomcat book was published with _____ on cover	polar bear	wild bear	snow leaopard	fox			snow leaopard
A _____ feature has been added to faciliate the scheduling of system upgrades	low density	high density	low availability	high availability			high availability
___ ant software build automation software was developed as a side effect of creatop of tomcat	Apache ant	apache bear	apache app	apache arms			Apache ant
_____ is a JDT java compiler	Asper 2	Fazer 2	Jaser 2	Jasper 3			Jaser 2
Jasper 2 uses _____	Eclipse JDT	Java bean	CSS	HTML			Eclipse JDT
During jsp runtime _____ can be inserted and included	Pages	Ruler	Scanner	Page maker			Pages
Each tag markup in JSP file is handled by____ class	File handler	Tag handler	Merge handler	Source handler			Tag handler
The ___ servlet is deleted once new servlet is finished being compiled	New JSP	Older JSP	Current JSP	Last JSP			Older JSP
The older JSP servlet is deleted once _____ servlet is finished	New JSP	Older JSP	Current JSP	Last JSP			New JSP
_____ support currently requires JDK Version 1.5 or later	clustering	foaming	grouping	kayaking			clustering

Clustering suports currently requires JDK_____ or later	Version 6.1	Version 7.1	Versio n 1.5	Version 1.7			Version 1.5
_____ is a connector component for tomcat that supports the HTTP	Ryote	Ryte	Coyote	Chandl er			Coyote
Tomcat 4.x was released with _____	Catalina	Casolin a	Casper	Caterer			Catalina
_____ is tomcats servlet container	Catalina	casolina	Casper	caterer			Catalina
In tomcat a realm represents a _____of usernames, passwords and roles	Databas e	Recipie nt	Sender	Worker			Database
_____ is built as a community process that involves both user and developer mailing lists	Apache software	Torand o softwar e	Avira softwar e	Avlon softwar e			Apache software
Apache software is built as a community process that involves both user and developer____	Holding list	Trackin g list	Hoppin g list	Mailing list			Mailing list
_____ implementation allow catalina to be integrated into environments where authentication information is already being created	Realm	Roman	Rando m	Rival			Realm
_____ listens for incoming connections to server on specific TCP port	coyote	boycott	canal	casper			coyote
Coyote listens for incoming connections to server on specific ____ port	FTTP	TCP	FTP	SMTP			TCP
_____ forward request to another web server such as apache using JK protocol	Coyote RT	Coyote JK	Coyote PG	Coyote AL			Coyote JK
As of _____ tomcat user jasper2	version 4	version 5	version 3	version 9			version 5

_____ feature will not affect the line environment	High availability	Low availability	Random availab	Function availabi			High availability
High availability feature will not affect the _____	Mortal environment	Immortal environ	Line environment	Random species			Line environment
Implementations of Axis 2 are available in_____ and C	JAVA	C	C++	Python			JAVA
___ is a sponsered community of developers and operator who are running apache tomcat in large scale production environment	Random source	Data source	Spring source	Dialup source			Spring source
Spring source is a sponsered community of _____ who are running apache tomcat in large scale production environment	Prons and corns	Developer and operator	Developers only	Operators only			Developer and operator
_____availability is done by dispatching live traffic requests	High	Low	Medium	Random			High
In tomcat a _____ represents a database of usernames, passwords and roles	Realm	Recipient	Sender	Worker			Realm
_____ offers better performance than HTTP protocol	JK protocol	JSP page	XML	Servlet			JK protocol
_____ deloyment requires a specific deployment descriptor called WSDP	Custom web service	Honour webb service	General web service	Existing web service			Custom web service
Custom web service deloyment requires a specific deployment descriptor called _____	WSDP	HTML	DHTML	WSCL			WSDP
For b2b one needs _____rather than reliability	Versatality	Outspokness	Accountability	Interiority			Accountability

Webservices can be combined in _____ way to achieve complex operations	Loosely coupled	Closely coupled	Lightly coupled	Strongly coupled			Looselycoupled
It is important to build infrastructure allowing components to conform _____	High needs	All needs	Strict needs	Basic needs			Basic needs
_____is often referred to as Tomcat	Apache tomcat	Arise tomcat	Apollo tomcat	Android			Apache tomcat
_____ implements several JAVA EE	Tomcat	CSS	HTML	XML			Tomcat
_____ is tomcats JSP engine	Caster	Micer	Jasper	Miranndo			Jasper
____ is the connector component of Tomcat	Coyote	Miot	Tungsten	Silicon			Coyote
While recompling modified JSP java code , the older version is still available for _____	Server request	client request	kernal request	hiper request			Server request
JWS is abbreviated as_____	Java web security	Java web service	Java world service	Java world security			Java web service
_____ is the core engine for web service	Apache axis 2	Android	Honey combs	Ginger bread OS			Apache axis 2
cluster has been added to manage_____	word document	excel document	small document	large document			large document
Coyote forward request using ____ protocol	JK	FTP	HTTP	SMTP			JK
Tomcat_____ was released with catalina	6.x	3.x	4.x	8.x			4.x
Accountability is more important than _____in b2b	Versatality	Outspokness	Reliability	Interiority			Reliability
Apache tomcat is often referred to as_____	Tomcat	Type pad	Server	Service provide			Tomcat
High availability is done by dispatching _____	Low traffic	Live traffic	Visible traffic	Random			Live traffic
Coyote is a connector component for tomcat that supports the _____	HTTP	CSS	XML	HTML			HTTP

The tag markup in ____ file is handled by tag handler class	JSP	ASP	FTTP	FTP			JSP
A high availability feature helps in ___ of system upgrades	scheduling	proxy	copying	pasting			scheduling
Pages can be inserted and included into ___ run time	JSP	HTML	XML	XSS			JSP
Tomcat version 5 uses_____	Jasper 6	Jaser 2	Jasper 4	Jaser 1			Jaser 2
_____offers better performance than JK protocol	JP protocol	JK protocol	FTTP protocol	HTTP protocol			HTTP protocol
Several _____ implementations are done in Tomcat	JAVA AB	JAVA AD	JAVA EE	JAVA ED			JAVA EE

Reg. No.....

[15CAP405W]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2015 onwards)

MCA DEGREE EXAMINATION, APRIL 2017

Fourth Semester

COMPUTER APPLICATIONS

WEB SERVICES

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

PART B (5 x 6 = 30 Marks)
Answer ALL the Questions

21. a. Write a short note on the importance of web services.
Or
b. Explain the term namespaces in xml.

22. a. Describe the concept on REST architecture.
Or
b. Give a note on WSDL.

23. a. What are the WSDL interface components? Explain.
Or
b. Explain the workflow management systems.

24. a. Explain the types of security attacks.
Or
b. Write a note on proxy based mobile systems.

25. a. Write a note on web service and applications.
Or
b. Explain about the tomcat application server.

PART C (1 x 10 = 10 Marks)
(Compulsory)

26. Give a note on Business transaction protocol.

Reg. No.....

[13CAP504W]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2013 onwards)

MCA DEGREE EXAMINATION, NOVEMBER 2015

Fifth Semester

COMPUTER APPLICATIONS

WEB SERVICES

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

PART B (5 x 8 = 40 Marks) (2 ½ Hours)
Answer ALL the Questions

21. a. Describe XML Schema and attribute in detail.

Or

b. Explain processing of XML document with XSLT and XPATH Language.

22. a. Explain the UDDI Specification and its core data structures.

Or

b. Explain the SOAP Model and its Message Format.

23. a. Explain the ACID Transaction.

Or

b. Discuss about OASIS Business Transaction Protocol.

24. a. Explain any four elements supported in web service security application.

Or

b. Discuss about Direct Mobile Web Service Access.

25. a. Elaborate the functionalities and architecture of EPS application

Or

b. Explain how to secure the application.

Reg. No.....

[14CAP504W]

KARPAGAM UNIVERSITY
Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2014 onwards)

MCA DEGREE EXAMINATION, NOVEMBER 2016
Fifth Semester

COMPUTER APPLICATIONS

WEB SERVICES

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

PART B (5 x 8 = 40 Marks) (2 ½ Hours)
Answer ALL the Questions

21. a. Discuss in detail about XML namespaces.
Or
b. What is XML? Write down the steps in writing an XML document

22. a. Explain REST architecture with diagram.
Or
b. Explain in detail, the possible communication styles in SOAP

23. a. Explain one and two way interaction in WSCL
Or
b. Define workflow management system. Discuss the features of a good workflow management system

24. a. Compare SOAP and REST in respect to mobile web services.
Or
b. What is a proxy based mobile system? What are the challenges in using proxies.

25. a. What are the tools used in web services briefly discuss about tomcat application server
Or
b. What are the tools used in web services briefly discuss about server axis soap server

---

Reg. No.....

[12CAP504W]

KARPAGAM UNIVERSITY

(Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2012 onwards)

MCA DEGREE EXAMINATION, NOVEMBER 2014

Fifth Semester

COMPUTER APPLICATIONS

WEB SERVICES

Time: 3 hours Maximum : 100 marks

PART – A (15 x 2 = 30 Marks)
Answer ALL the Questions

1. Define the term UDDI.
2. Write a note on XML namespace.
3. List out XML schema complex type compositors.
4. What do you mean by SOAP fault?
5. What do you mean by portType element in WSDL?
6. Define the term white pages.
7. Define web service conversation language.
8. What do you mean by abstract interface?
9. Write a short note on BPEL container.
10. Define Data privacy.
11. What do you mean by dictionary attack?
12. Define the term proxy server.
13. Define client-side binding stubs.
14. What do you mean by transaction?
15. Define client application.

PART B (5 X 14= 70 Marks)
Answer ALL the Questions

16. a) Discuss the architecture of Web application and Web services.

Or

b) Briefly explain about XML namespace.

17. a) Explain the following terms
i) SOAP RPC ii) SOAP faults

Or

b) With neat diagram, explain about WSDL architecture.

18. a) What are the activity types in BPEL? Discuss.

Or

b) Explain the following terms
i) ACID transaction ii) Two phase commit

19. a) Discuss different types of security attacks.

Or

b) Briefly explain about proxy based mobile system.

20. **Compulsory : -**

Design and develop an enterprise procurement system for any application. State the functional requirements you are taking into consideration.

Reg. No...18LCAP028.....

[17CAP405W]

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Pollachi Main Road, Eachanari Post, Coimbatore – 641 021
(For the candidates admitted from 2017 onwards)

MCA DEGREE EXAMINATION, APRIL 2019

Fourth Semester

COMPUTER APPLICATIONS

WEB SERVICES

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

(Part – B & C 2 ½ Hours)

PART B (5 x 6 = 30 Marks)
Answer ALL the Questions

21. a. What are the role of Web Services.
Or
b. Explain about the XML basis.

22. a. Discuss about the Middleware RPC.
Or
b. Difference between UDDI and WSDL.

23. a. Give a note on WSCL.
Or
b. Explain about the ACID Transaction.

24. a. What are the various security issues.
Or
b. Explain about the Multiple Web services.

25. a. What are the applications of Web Services.
Or
b. Give a account on Real World Web Services.

PART C (1 x 10 = 10 Marks)
COMPULSORY

26. Briefly and Explain the BPEL component model.
