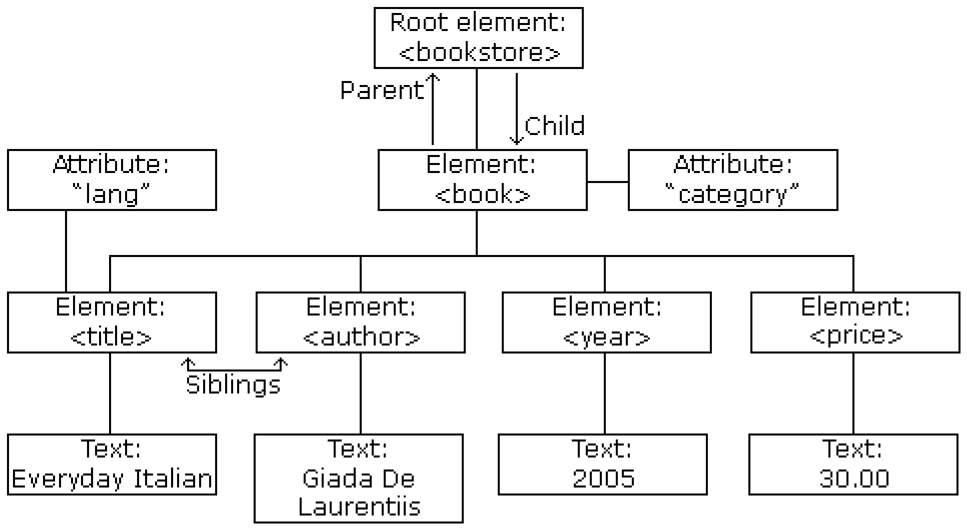
XML DOM

The DOM defines a standard for accessing and manipulating documents.

The XML DOM presents an XML document as a tree-structure.

The HTML DOM presents an HTML document as a tree-structure.

Understanding the DOM is a must for anyone working with HTML or XML.



The XML DOM defines a standard way for accessing and manipulating XML documents.

All XML elements can be accessed through the XML DOM.

The XML DOM defines the objects, properties and methods of all XML elements.

The XML DOM is:

A standard object model for XML

A standard programming interface for XML

Platform- and language-independent

A W3C standard

In other words: The XML DOM is a standard for how to get, change, add, or delete XML elements.

Loading an XML File

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

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

Example

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

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

<script>
var xhttp= new XMLHttpRequest();
xhttp.onreadystatechange= function(){
    if (xhttp.readyState== 4 &&xhttp.status== 200)
{myFunction(xhttp);
  }
};
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml){
    var xmlDoc=xml.responseXML;
  document.getElementById("demo").innerHTML=
  xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
</script>

</body>
</html>
```

XML DOM Properties
These are some typical DOM properties:
x.nodeName - the name of x
x.nodeValue - the value of x
x.parentNode - the parent node of x
x.childNodes - the child nodes of x
x.attributes - the attributes nodes of x
Note: In the list above, x is a node object.

XML DOM Methods
x.getElementsByTagName(name) - get all elements with a specified tag name
x.appendChild(node) - insert a child node to x
x.removeChild(node) - remove a child node from x
DOM Nodes
According to the DOM, everything in an XML document is a node.
The DOM says:
The entire document is a document node
Every XML element is an element node
The text in the XML elements are text nodes
Every attribute is an attribute node
Comments are comment nodes

DOM Example
Look at the following XML file (books.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">EverydayItalian</title>
    <author>GiadaDeLaurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">HarryPotter</title>
    <author>JKRowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">XQueryKickStart</title>
    <author>JamesMcGovern</author>
    <author>PerBothner</author>
    <author>KurtCagle</author>
    <author>JamesLinn</author>
    <author>VaidyanathanNagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="web" cover="paperback">
    <title lang="en">LearningXML</title>
    <author>ErikT.Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root node in the XML above is named <bookstore>. All other nodes in the document are contained within <bookstore>.

The root node <bookstore> holds four <book> nodes.

The first <book> node holds four nodes: <title>, <author>, <year>, and <price>, which contains one text node each, "Everyday Italian", "Giada De Laurentiis", "2005", and "30.00".

Text is Always Stored in Text Nodes

A common error in DOM processing is to expect an element node to contain text.

However, the text of an element node is stored in a text node.

In this example: <year>2005</year>, the element node <year>, holds a text node with the value "2005".

"2005" is not the value of the <year> element!

DOX Example

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

<p>The getAllResponseHeaders() function returns the header information of a resource, like length, server-type, content-type, last-modified, etc.</p>
```

```
<button onclick="loadXMLDoc('xmlhttp_info.txt')">Get header information</button>

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

<script>
function loadXMLDoc(url) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      document.getElementById("demo").innerHTML =
      xmlhttp.getAllResponseHeaders();
    }
  };
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
}
</script>

</body>
</html>
```

Output

The getAllResponseHeaders() function returns the header information of a resource, like length, server-type, content-type, last-modified, etc.

Get header information

```
Date: Wed, 30 Mar 2016 05:51:04 GMT Last-Modified: Thu, 17 Sep 2015 12:15:18 GMT Server: ECS (sjc/4E67) X-Powered-By: ASP.NET Etag: "633a658442f1d01:0+ident+gzip" Vary: Accept-Encoding X-Cache: HIT Content-Type: text/plain Cache-Control: public,max-age=3600,public Transfer-Encoding: chunked
```

XSLT

XSL stands for EXtensible Stylesheet Language, and is a style sheet language for XML documents.

XSLT stands for XSL Transformations. In this we will learn how to use XSLT to transform XML documents into other formats, like XHTML

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 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 - Transformation

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Start with a Raw XML Document

We want to transform the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
 <cd>
  <title>EmpireBurlesque</title>
  <artist>BobDylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
 </cd>
.
.
</catalog>
```

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
 <html>
 <body>
 <h2>MyCDCollection</h2>
 <table border="1">
  <tr bgcolor="#9acd32">
   <th>Title</th>
   <th>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>
```

XSLT <xsl:template> Element
An XSL style sheet consists of one or more set of rules that are called templates.
A template contains rules to apply when a specified node is matched.

The <xsl:template> Element
The <xsl:template> element is used to build templates.
The match attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).
Ok, let's look at a simplified version of the XSL file from the previous chapter:
Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>MyCDCollection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

XSLT <xsl:value-of> Element
The <xsl:value-of> Element
The <xsl:value-of> element can be used to extract the value of an XML element and

add it to the output stream of the transformation:

The <xsl:for-each> Element

The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set

XSLT <xsl:sort> Element

The <xsl:sort> element is used to sort the output.

The <xsl:if> Element

To put a conditional if test against the content of the XML file, add an <xsl:if> element to the XSL document.

Syntax

```
<xsl:if test="expression">
 ...some output if the expression is true...
</xsl:if>
```

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

The <xsl:choose> Element

Syntax

```
<xsl:choose>
 <xsl:when test="expression">
   ... some output ...
 </xsl:when>
 <xsl:otherwise>
   ... some output ....
 </xsl:otherwise>
</xsl:choose>
```

The <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

DOM & SAX

DOM Stands for Document Object Model and it represent an XML Document into tree format which each element representing tree branches. DOM Parser creates an In Memory tree representation of XML file and then parses it, so it requires more memory and its advisable to have increased heap size for DOM parser in order to avoid Java.lang.OutOfMemoryError:java heap space . Parsing XML file using DOM parser is quite fast if XML file is small but if you try to read a large XML file using DOM parser there is more chances that it will take a long time or even may not be able to load it completely simply because it requires lot of memory to create XML Dom Tree. Java provides support DOM Parsing and you can parse XML files in Java using DOM parser. DOM classes are in w3c.dom package while DOM Parser for Java is in JAXP (Java API for XML Parsing) package.

SAX XML Parser in Java

SAX Stands for Simple API for XML Parsing. This is an event based XML Parsing and it parse XML file step by step so much suitable for large XML Files. SAX XML Parser fires event when it encountered opening tag, element or attribute and the parsing works accordingly. It's recommended to use SAX XML parser for parsing large xml files in Java because it doesn't require to load whole XML file in Java and it can read a big XML file in small parts. Java provides support for SAX parser and you can parse any xml file in Java using SAX Parser, I have covered example of reading xml file using SAX Parser here. One disadvantage of using SAX Parser in java is that reading XML file in Java using SAX Parser requires more code in comparison of DOM Parser.

Here are few high level differences between DOM parser and SAX Parser in Java:

1) DOM parser loads whole xml document in memory while SAX only loads small part of XML file in memory.

2) DOM parser is faster than SAX because it access whole XML document in memory.

3) SAX parser in Java is better suitable for large XML file than DOM Parser because it doesn't require much memory.

4) DOM parser works on Document Object Model while SAX is an event based xml parser.

SAX

SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content

SAX is an event-driven interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XML parser

AJAX

AJAX is an acronym for **Asynchronous JavaScript and XML**. AJAX is a new technique for creating better, faster and interactive web applications with the help of JavaScript, DOM, XML, HTML, CSS etc. AJAX allows you to send and receive data asynchronously without reloading the entire web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

Ajax is the most viable Rich Internet Application(RIA) technique so far.

Where it is used?

There are too many web applications running on the web that are using AJAX Technology. Some are

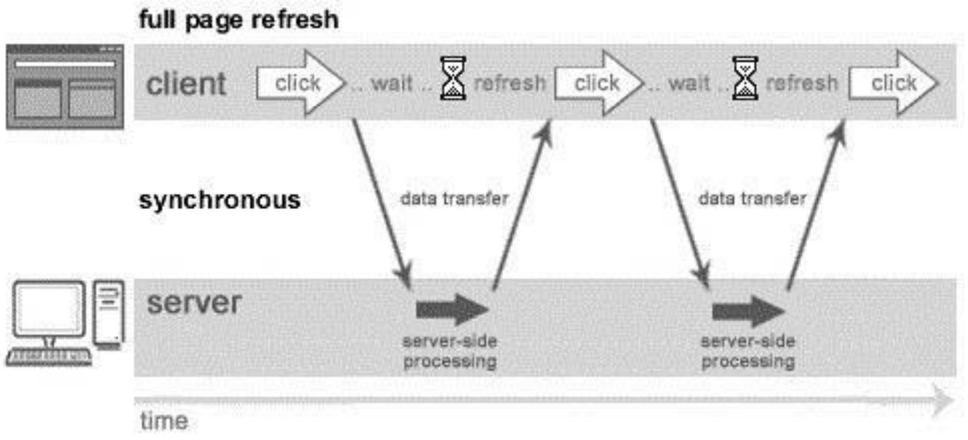
1. Gmail
2. Face book
3. Twitter
4. Google maps
5. YouTube etc.,

Synchronous Vs. Asynchronous Application

Before understanding AJAX, let's understand classic web application model and AJAX Web application model.

❖ **Synchronous (Classic Web-Application Model)**

A synchronous request blocks the client until operation completes i.e. browser is not unresponsive. In such case, JavaScript Engine of the browser is blocked.

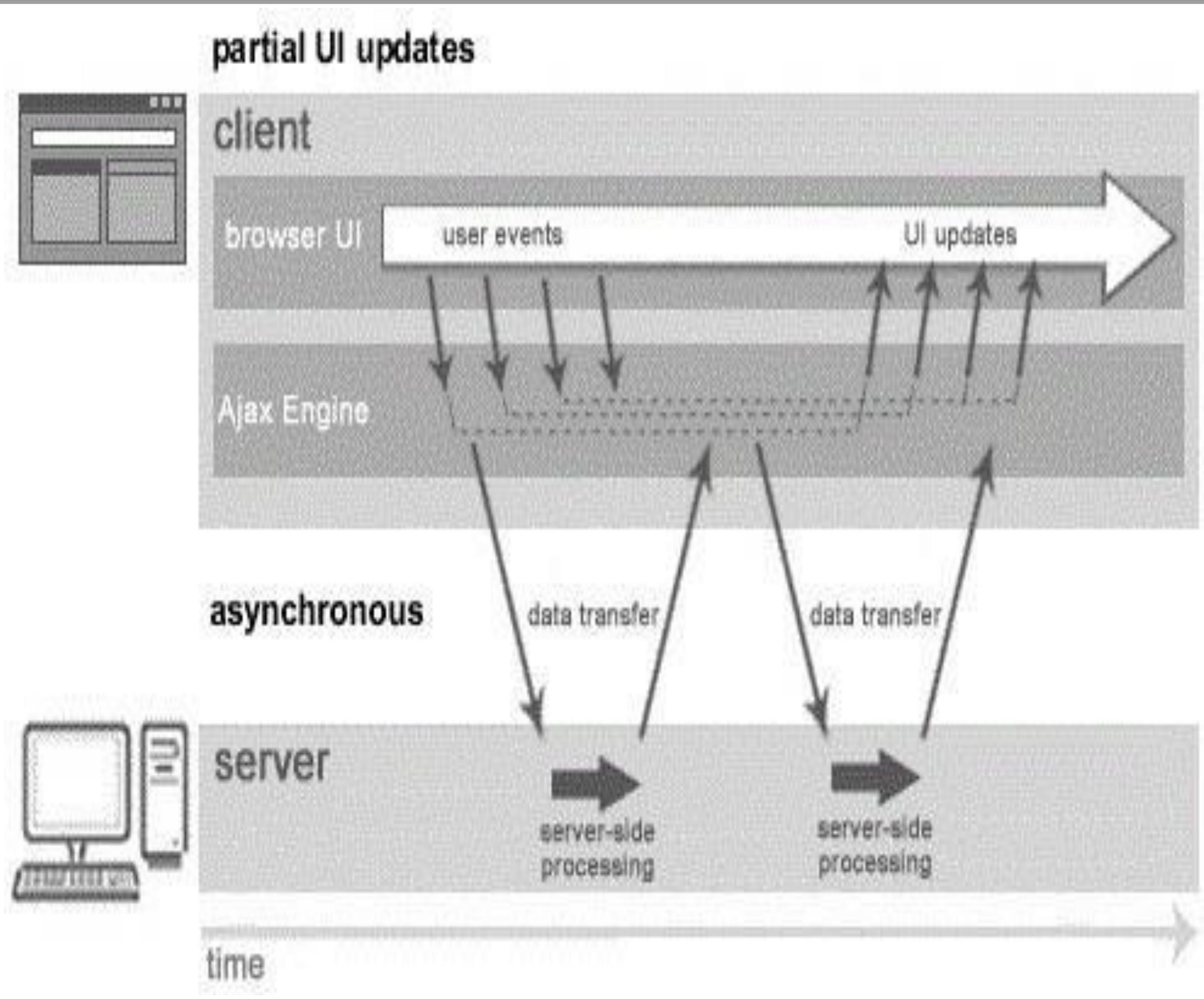


As you can see in the above image, full page is refreshed at request time and user is blocked until request completes.

❖ **Asynchronous (AJAX Web-Application Model)**

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform other operations also. In such

case, JavaScript Engine of the browser is not blocked.


partial UI updates
client
browser UI
user events
UI updates
Ajax Engine
asynchronous
data transfer
data transfer
server
server-side processing
server-side processing
time

10

As you can see in the above image, full page is not refreshed at request time and user gets response from the AJAX Engine. Let's try to understand asynchronous communication by the image given below.

AJAX Components

AJAX is not a technology but group of inter-related technologies. AJAX Technologies includes:

- ❖ HTML/XHTML and CSS
- ❖ DOM
- ❖ XML or JSON(JavaScript Object Notation)
- ❖ XMLHttpRequest Object
- ❖ JavaScript

- **HTML/XHTML and CSS**
  These technologies are used for displaying content and style. It is mainly used for presentation.

- **DOM**
  It is used for dynamic display and interaction with data.

- **XML or JSON(Javascript Object Notation)**
  For carrying data to and from server. JSON is like XML but short and faster than XML.

- **XMLHttpRequest Object**
  For asynchronous communication between client and server.

- **JavaScript**
  It is used to bring above technologies together. Independently, it is used mainly for client-side validation.

Understanding XMLHttpRequest

It is the heart of AJAX technique. An object of XMLHttpRequest is used for asynchronous communication between client and server.it provides a set of useful methods and properties that are used to send HTTP Request to and retrieve data from the web server. It performs following operations:

1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.

- **Methods of XMLHttpRequest object**

Method	Description
void open(method, URL)	Opens the request specifying get or post method and url.
void open(method, URL, async)	Same as above but specifies asynchronous or not.

void open(method, URL, async, username, password)	Same as above but specifies username and password.
void send()	Sends GET request.
void send(string)	Sends POST request.
setRequestHeader(header,value)	It adds request headers.

Syntax of open() method:

xmlHttp.open("GET","conn.php",true); which takes three attributes

1. An HTTP method such as GET ,POST , or HEAD
2. The URL of the Server resource
3. A boolean Flag that indicates whether the request should be asynchronously(true) or synchronously(false)

Properties of XMLHttpRequest Object:

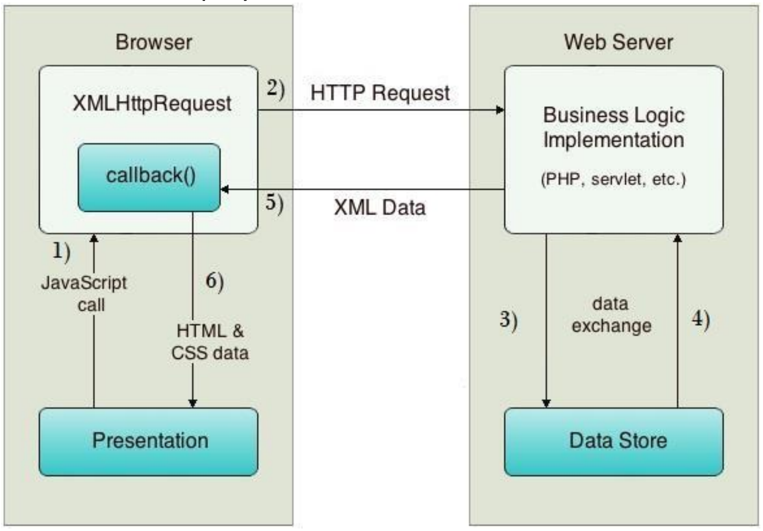
Property	Description
readyState	Represents the state of the request. It ranges from 0 to 4. **0 UN INITIALIZED** – After creating XMLHttpRequest Object before calling *open()* method. **1 CONNECTION ESTABLISHED** – open() is called but send() is not called. **2 REQUEST SENT-** send() is called. **3 PROCESSING** - Downloading data; responseText holds the data. **4 DONE -** The operation is completed successfully.
onReadyStateChange	It is called whenever *readystate* attribute changes. It must not be used with synchronous requests.
reponseText	Returns response as TEXT.
responseXML	Returns response as XML

How AJAX Works?

AJAX communicates with the server using XMLHttpRequest object. Let's understand the flow of AJAX with the following figure:

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.

4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.



Introduction to Web Services

Technology keep on changing, users were forces to learn new application on continuous basis. With internet, focus is shifting to-wards services based software. Users may access these services using wide range of devices such as PDAs, mobile phones, desktop computers etc. Service oriented software development is possible using man known techniques such as COM, CORBA, RMI, JINI, RPC etc. some of them are capable of delivering services over web and some or not. Most of these technologies uses particular protocols for communication and with no standardization. **Web service** is the concept of creating services that can be accessed over web. Most of these

What are Web Services?

A web services may be defines as: An application component accessible via standard web protocols. It is like unit of application logic. It provides services and data to remote clients and other applications. Remote clients and application access web services with internet protocols. They use XML for data transport and SOAP for using services. Accessing service is independent of implementation.

With component development model, web service must have following characteristics:

- ❖ Registration with lookup service
- ❖ Public interface for client to invoke service

Web services should also process following characteristics:

- ❖ It should use standard web protocols for communication
- ❖ It should be accessible over web
- ❖ It should support loose coupling between uncoupled distributed systems

Web services receive information from clients as messages, containing instructions about what client wants, similar to method calls with parameters. These message delivered by web services are encoded using XML.XML enabled web services are interoperable with other web services.

Web Service Technologies:

Wide variety of technologies supports web services. Following technologies are available for creation of web services. These are vendor neutral technologies. They are:

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

Simple Object Access Protocol (SOAP):

SOAP is a light weight and simple XML based protocol. It enables exchange of structured and typed information on web by describing messaging format for machine to machine communication. It also enables creation of web services based on open infrastructure.

- SOAP is application communication protocol designed to communicate via Internet.
- SOAP is a format for sending and receiving messages.
- SOAP provides data transport for Web services.
- SOAP is platform and language-independent.
- SOAP enables client applications to easily connect to remote services and invoke remote methods.
- SOAP can be used in combination with variety of existing internet protocols and formats including HTTP, SMTP etc.

A SOAP message is an ordinary XML document which consists of three parts:

- ❖ **SOAP Envelope**: defines what is in message, who is the recipient, whether message is optional or mandatory
- ❖ **SOAP Encoding Rules**: defines set of rules for exchanging instances of application defined data types
- ❖ **SOAP RPC Representation**: defines convention for representing remote procedure calls and
- ❖
- ❖ response

SOAP Message Structure

The following block depicts the general structure of a SOAP message −

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

```
    <SOAP-ENV:Fault>
        ...
        </SOAP-ENV:Fault>
  ...
  </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

If we are creating web service that offered latest stock quotes, we need to create WSDL file on server that describes service. Client obtains copy of this file, understand contract, create SOAP request based on contract and dispatch request to server using HTTP post. Server validates the request, if found valid executes request. The result which is latest stock price for requested symbol is then returned to client as SOAP response.
Typical SOAP message is shown below:

```
<IVORY:Envelope
            xmlns:IVORY="http://schemas.xmlsoap.org/soap/envelope"
            IVORY:encodingStyle="http://schemas.xmlsoap.org/soap/enc
            oding">
<IVORY:Body>
      <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
      </m:GetLastTradePrice>
</IVORY:Body>
</IVORY:Envelope>
```

The consumer of web service creates SOAP message as above, embeds it in HTTP POST request and sends it to web service for processing:

```
POST /StockQuote HTTP/1.1
Host:
www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-
URI"
      ….
SOAP Message
….
```

The message now contains requested stock price. A typical returned SOAP message may look like following:

```
<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:GetLastTradePrice xmlns:m="Some-URI">
                  <Price>34.5</Price>
            </m:GetLastTradePrice>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interoperability:

The major goal in design of SOAP was to allow for easy creation of interoperable distributed web services. Few details of SOAP specifications are open for interpretation; implementation may differ across different vendors. SOAP message though it is conformant XML message, may not strictly follow SOAP specification.

Implementations:

SOAP technology was developed by DevelopMentor, IBM, Lotus, Microsoft etc. More than 50 vendors have currently implemented SOAP. Most popular implementations are by Apache which is open source java based implementation and by Microsoft in .NET platform. SOAP specification has been submitted to W3C, which is now working on new specifications called XMLP (XML Protocol)

SOAP Messages with Attachments (SwA)

SOAP can send message with an attachment containing of another document or image etc. On Internet, GIF, JPEG data formats are treated as standards for image transmission. Second iteration of SOAP specification allowed for attachments to be combined with SOAP message by using multipart MIME structure. This multi part structure is called as **SOAP Message Package**. This new specification was developed by HP and Microsoft. Sample SOAP message attachment is shown here:

```
MIME-Version: 1.0
Content-Type: Multipart/Related;
boundary=MIME_boundary; type=text/xml;
start="<myimagedoc.xml@mystie.com>" Content-
Description: This is the optional message description.
--MIME_boundary
Content-Type: text/xml;
charset=UTF-8 Content-Transfer-
Encoding: 8bit
Content-ID: <myimagedoc.xml@mysite.com>
<?xmll version="1.0" ?>
<SOAP-ENV: Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
<SOAP-ENV:Body>
        …
        <theSignedForm href="cid:myimage.tiff@mysite.com" />
        …
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
        --MIME_boundary
        Content-Type:
        image/tiff
        Content-Transfer-Encoding: binary
        Content-ID: <myimagedoc.xml@mysite.com>
        …binary TIFF image…
        --MIME_boundary--
```

Web Services Description Language (WSDL)

WSDL is an XML format for describing web service interface. WSDL file defines set of operations permitted on the server and format that client must follow while requesting service. WSDL file acts like contract between client and service for effective communication between two parties. Client has to request service by sending well formed and conformant SOAP request.

Features of WSDL

- WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
- WSDL definitions describe how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML- based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'.

WSDL Document:

WSDL document is an XML document that contains of set of definitions. First we declare name spaces required by schema definition:

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNameSpace=http://schemas.xmlsoap.org/wsdl/
elementFormDefault="qualified">
```

The root element is definitions as shown below:

```
<wsdl:defiinitions name="nmtoken"? targetNameSpace="uri"?>
        <import namespace="uri" location="uri"/>
<wsdl:documentation ….. />?
        …
</wsdl:definitions>
```

The *name* attribute is optional and can serve as light weight form of documentation. The *nmtoken* represents name token that are qualified strings similar to CDATA, but character usage is limited to letters, digits, underscores, colons, periods and dashes. A *targetNamespace* may be specified by providing uri. The *import* tag may be used to associate namespace with document locations. Following code segment shows how declared namespace is associated with document location specified in *import* statement:

```
<definitions name="StockQuote"
        targetNameSpace="http://example.com/stockquote/de
        fiinitions"
        xmlns:tns="http://example.com/stockquote/definitions
        "
        xmlns:xsdl="http://example.com/stockquote/schemas"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
<import
        namespace="http://example.com/stockquote/schem
        as"
        Location="http://example.com/stockquote/stockquo
        te.xsd"/>
```

Finally, optional *wsdl:documentation* element is used for declaring human readable documentation. The element may contain any arbitrary text. There are six major elements in document structure that describes service. These are as follows:

- **Types Element:** it provides definitions for data types used to describe how messages will exchange data. Syntax for types element is as follows:

```
        <wsdl:types> ?
                <wsdl:documentation …/>
                <xsd:schema …/>
                <-- extensibility element -->
        </wsdl:types>
```

  The *wsdl:documentation* tag is optional as in case of *definitions*. The *xsd* type system may be used to define types in message. WSDL allows type systems to be added via extensibility element.

- **Message Element:** It represents abstract definition of data begin transmitted. Syntax for message element:

```
        <wsdl:message name="nktoken"> *
                <wsdl;documentation …/>
                    <part name="nmtoken" element="qname"? type="qname"? /> *
        </wsdl:message>
```

  The *message name* attribute is used for defining unique name for message with in document scope. The *wsdl:documentation* is optional and may be used for declaring human readable documentation. The message consists of one or more logical parts. The *part* describes logical abstract content of message. Each part consists of name and optional element and type attributes.\

- **Port Type Element:** It defines set of abstract operations. An operation consists of both input and output messages. The *operation* tag defines name of operation, *input* defines input for operation and *output* defines output format for result. The *fault* element is used for describing contents of SOAP fault details element. It specifies abstract message format for error messages that may be output as result of operation:

```
<wsdl:portType name="nmtoken">*
                <wsdl:documentation …./>?
        <wsdl:operation name="nmtoken">*
                <wsdl:documentation …./>?
                        <wsdl:input name="nmtoken"? message="qname">?
                <wsdl:documentation …./>?
                </wsdl:input>
                        <wsdl:output name="nmtoken"? message="qname">?
                <wsdl:documentation ..../>?
                </wsdl:output>
                        <wsdl:fault name="nmtoken"? message="qname">?
                        <wsdl:documentation ..../>?
                </wsdl:fault>
        </wsdl:operation>
</wsdl:portType>
```

- **Binding Element:** It defines protocol to be used and specifies data format for operations and messages defined by particular *portType*. The full syntax for binding is given below:

```
<wsdl:binding name="nmtoken" type="qname"> *
                <wsdl:documentation ..../>?
                <--Extensibility element -->*
        <wsdl:operation name="nmtoken">*
                <wsdl:documentation ..../>?
                <--Extensibility element -->*
        <wsdl:input> ?
                <wsdl:documentation ..../>?
                <--Extensibility element -->*
        </wsdl:input>
        <wsdl:output> ?
                <wsdl:documentation ..../>?
                <--Extensibility element -->*
        </wsdl:output>
        <wsdl:fault name="nmtoken"> *
                <wsdl:documentation ..../>?
                <--Extensibility element -->*
        </wsdl:fault>
        </wsdl:operation>
</wsdl:binding>
```

The operation in WSDL file can be document oriented or remote procedure call (RPC) oriented. The style attribute of *<soap:binding>* element defines type of operation. If operation is document oriented, input and output messages will consist of XML documents. If operation is RPC oriented, input message contains operations input parameters and output message contains result of operation.

- **Port Element:** It defines individual end point by specifying single address for binding:

```
<wsdl:port name="nmtoken" binding="qname"> *
        <--Extensibility element (1) -->
</wsdl:port>
```

The *name* attribute defines unique name for port with current WSDL document. The *binding* attribute refers to binding and extensibility element is used to specify address information for port.

❖ **Service Element:** it aggregates set of related ports. Each port specifies address for binding:

```
<wsdl:service name="nmtoken"> *
            <wsdl:documentation …./>?
     <wsdl:port name="nktoken" binding="qname"> *
            <wsdl:documentation …/> ?
            <--Extensibility element -->
     </wsdl:port>
            <--Extensibility element -->
    </wsdl:service>
```

Universal Description, Discovery and Integration (UDDI)

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

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

UDDI has two sections:

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

A business or a company can register three types of information into a UDDI registry. This information is contained in three elements of UDDI. These three elements are:

- White Pages,
- Yellow Pages, and
- Green Pages.

White Pages

White pages contain:

- Basic information about the company and its business.
- Basic contact information including business name, address, contact phone number, etc.
- A Unique identifiers for the company tax IDs. This information allows others to discover your web service based upon your business identification.

Yellow Pages

- Yellow pages contain more details about the company. They include descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it.
- Yellow pages uses commonly accepted industrial categorization schemes, industry

codes, product codes, business identification codes and the like to make it easier for companies to search through the listings and find exactly what they want.

Green Pages

Green pages contains technical information about a web service. A green page allows someone to bind to a Web service after it's been found. It includes:

- The various interfaces
- The URL locations
- Discovery information and similar data required to find and run the Web service.

Implementation:

This is global, public registry called UDDI business registry. It is possible for individuals to set up private UDDI registries. The implementations for creating private registries are available from IBM, Idoox etc. Microsoft has developed UDDI SDK that allows visual basic programmer to write program code to interact with UDDI registry. The use of SDK greatly simplifies interaction with registry and shields programmer from local level details of XML and SOAP.