UNIT-V

WEB SERVICES

- Web Services can convert your application into a Web-application, which can publish its function or message to the rest of the world.
- The basic Web Services platform is XML + HTTP.

5.1 INTRODUCTION TO WEB SERVICES

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

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
- XML is the basis for Web services

How does it Work?

- 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:

1. SOAP (Simple Object Access Protocol)
2. UDDI (Universal Description, Discovery and Integration)
3. WSDL (Web Services Description Language)

Why Web Services?

Interoperability has Highest Priority

- When all major platforms could access the Web using Web browsers, different platforms couldn't interact. For these platforms to work together, Web-applications were developed.
- Web-applications are simply applications that run on the web. These are built around the Web browser standards and can be used by any browser on any platform.

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

1. **Reusable application-components**
   - Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.
2. **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 Platform Elements

Web Services have three basic platform elements:

1. SOAP
2. WSDL and
3. UDDI

5.2 UDDI

Universal Description, Discovery and Integration (UDDI) are a directory service where businesses can register and search for Web services.

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.

Who is Supporting UDDI?

- UDDI is a cross-industry effort driven by all major platform and software providers like Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, and Sun, as well as a large community of marketplace operators, and e-business leaders.
- Over 220 companies are members of the UDDI community.

UDDI has two parts:

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

UDDI Elements

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

(1) White pages:

This category contains:

- 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.

(2) Yellow pages:

This category contains:

- This has more details about the company, and includes descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it.
- It 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.

(3) Green pages:
This category contains technical information about a web service. This is what allows someone to bind to a Web service after it's been found. This includes:

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

UDDI Technical Architecture:
The UDDI technical architecture consists of three parts:

1. **UDDI data model:**
   - An XML Schema for describing businesses and web services. The data model is described in detail in the "UDDI Data Model" section.

2. **UDDI API Specification:**
   - A Specification of API for searching and publishing UDDI data.

3. **UDDI cloud services:**
   - This is operator sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.

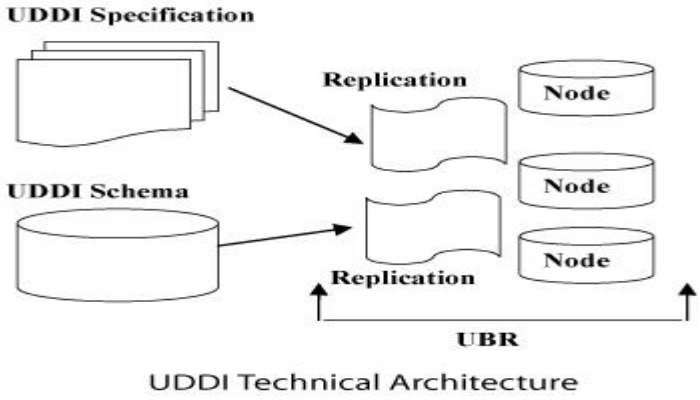

Fig 5.1: UDDI Architecture

- The UDDI Business Registry (UBR), also known as the Public Cloud, is a conceptually single system built from multiple nodes that has their data synchronized through replication.
- The current cloud services provide a logically centralized, but physically distributed, directory. This means that data submitted to one root node will automatically be replicated across all the other root nodes. Currently, data replication occurs every 24 hours.
- UDDI cloud services are currently provided by Microsoft and IBM.

It is also possible to set up private UDDI registries. For example, a large company may set up its own private UDDI registry for registering all internal web services. As these registries are not automatically synchronized with the root UDDI nodes, they are not considered part of the UDDI cloud.

UDDI Data Model:
UDDI includes an XML Schema that describes four five data structures:

1. businessEntity
2. businessService
3. bindingTemplate
4. tModel
5. publisherAssertion

Business Entity 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.

Business Service 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 and what type of web service it is.

Binding Template 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.

5.3 SOAP

SOAP INTRODUCTION:

- SOAP is a simple XML-based protocol to let applications exchange information over HTTP.
- SOAP is a protocol for accessing a Web Service.

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 SYNTAX:

SOAP Building Blocks

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

1. An **Envelope element** that identifies the XML document as a SOAP message.
2. A **Header element** that contains header information.
3. A **Body element** that contains call and response information.
4. **A Fault element** containing errors and status information.

All the elements above are declared in the default namespace for the SOAP envelope:

http://www.w3.org/2001/12/soap-envelope

and the default namespace for SOAP encoding and data types is:

http://www.w3.org/2001/12/soap-encoding

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 Instruction

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>
```

1. 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 encoding Style Attribute

- The encoding Style 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.
- 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>
```

2. SOAP Header:

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 "must understand" 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: must Understand, actor, and encoding Style.
- The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

The must Understand 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 must Understand="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"

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"

The encoding Style Attribute

- The encoding Style 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"

3. SOAP Body Element

- The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.
- Immediate child elements of the SOAP Body element may be namespace-qualified.

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:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP namespace.
A SOAP response could look something like this:

```
<?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:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

4. SOAP Fault Element

- The SOAP Fault element holds errors and status information for a SOAP message.
- The optional SOAP Fault element is used to indicate error messages.
- If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

SOAP Example

In the example below, a GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response. The namespace for the function is defined in "http://www.example.org/stock".

SOAP request:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?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:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
```

```
            </soap:Body>
        </soap:Envelope>
```

SOAP response:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
    <?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:Body xmlns:m="http://www.example.org/stock">
                <m:GetStockPriceResponse>
                <m:Price>34.5</m:Price>
                </m:GetStockPriceResponse>
            </soap:Body>
        </soap:Envelope>
```

5.4 WSDL

WSDL (Web Services Description Language) is an XML-based language for describing Web services and how to access them.

Introduction to WSDL

- WSDL is an XML-based language for describing Web services and how to access them.

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.

WSDL Documents

- A WSDL document is just a simple XML document.
- It contains set of definitions to describe a web service.

The WSDL Document Structure:

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

Element	Description

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

The main structure of a WSDL document looks like this:

```
<definitions>
    <types>
        data type definitions.....
    </types>
    <message>
        definition of the data being communicated....
    </message>
    <portType>
        set of operations.....
    </portType>
    <binding>
        protocol and data format specification....
    </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 data format and protocol for each port type.

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 PortType

The <portType> element is the most important WSDL element.

WSDL - The <portType> Element

- The <portType> element defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.
- <portType> 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 portType "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 portType "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".

5.5 Web Service Architecture (WSA)

Web services are

- Applications that enable remote procedure calls over a network or the Internet often using XML and HTTP.
- Web services architecture is an interoperability architecture-it identifies those global elements of the global web services network that are required in order to ensure interoperability between web services.

Web Service Model:

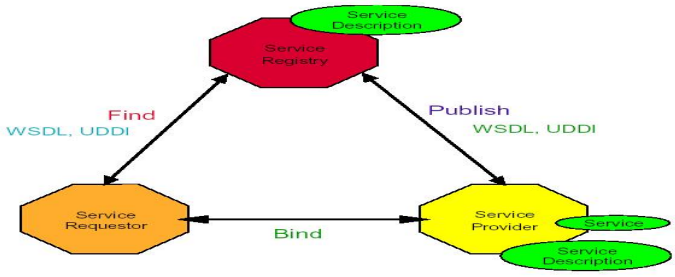

Fig 5.2 Web Service

Roles in Web Service architecture

1. Service provider
   - Owner of the service.
   - Platform that hosts access to the service.
2. Service requestor
   - Business that requires certain functions to be satisfied.
   - Application looking for and invoking an interaction with a service.
3. Service registry
   - Searchable registry of service descriptions where service providers publish their service descriptions.

Operations in Web Service Architecture

- Publish
  Service descriptions need to be published in order for service requestor to find them.
- Find
  Service requestor retrieves a service description directly or queries the service registry for the service required.
- Bind
  Service requestor invokes or initiates an interaction with the service at runtime.

Web services specification and technologies

- Security - for Web services confidentiality, integrity, authentication, and authorization.
- Process flow - for arranging the flow of execution across Web services.
- Transactions - for coordinating the results of multiple Web services.
- Messaging - for configuring message paths and routing messages reliably across multiple network hops.

Web Services involve three major roles

1. Service Provider
2. Service Registry
3. Service Consumer

Three major operations surround web services

- Publishing – making a service available
- Finding – locating web services
- Binding – using web services

Security

- Security is one of the most important and most complex issues in the Internet and Web services.
- Basic security issues include
  - data confidentiality and integrity—to ensure your credit card number,
  - Authentication/authorization, which deals with the rights of individuals or groups to access a certain resource, such as a given Web service interface.

SAML

- Security Assertions Markup Language (SAML) provides a standard way to profile information in XML documents and to define user identification and authorization information.

XKMS

- The XML Key Management Specification (XKMS) defines a protocol for distributing and registering public keys used in encrypting and decrypting messages transmitted using SOAP.
- XML-based security standards
  - authentication and authorization (SAML)
  - public key management (XKMS).
  - WS-License and WS-Security
  - fundamental to all Internet security,
  - the firewalls that protect private networks

5.6 AJAX –improving web page performance using AJAX

AJAX = Asynchronous JavaScript and XML.

- AJAX is not a new programming language, but a new way to use existing standards.
- AJAX is the art of exchanging data with a server, and updating parts of a web page – without reloading the whole page.

AJAX Introduction

- AJAX is about updating parts of a web page, without reloading the whole page.

What is AJAX?

- **AJAX = Asynchronous JavaScript and XML.**
- AJAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
- Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

How AJAX Works

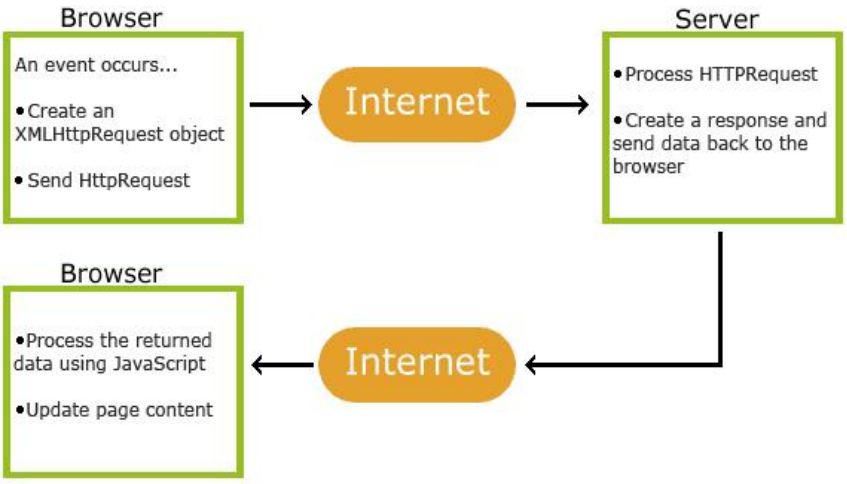

Fig 5.3 Working of AJAX

AJAX is based on Internet Standards

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)
- AJAX applications are browser and platform-independent!

Google Suggest

- AJAX was made popular in 2005 by Google, with Google Suggest.
- Google Suggest is using AJAX to create a very dynamic web interface: When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

AJAX Example

To understand how AJAX works, we will create a small AJAX application:

```
<html>
    <head>
        <script>
        function loadXMLDoc()
        {
            var xmlhttp;
            if (window.XMLHttpRequest)
            {    // code for IE7+, Firefox, Chrome, Opera, Safari
                xmlhttp=new XMLHttpRequest();
            }
            else
            {    // code for IE6, IE5
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }
            xmlhttp.onreadystatechange=function()
            {
                if (xmlhttp.readyState==4 && xmlhttp.status==200)
                {
```

```
        document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
                        }
                }
                xmlhttp.open("GET","ajax_info.txt",true);
                xmlhttp.send();
        }
        </script>
</head>
<body>
        <div id="myDiv"><h2>Let AJAX change this text</h2></div>
        <button type="button" onclick="loadXMLDoc()">Change Content</button>
</body>
</html>
```

OUTPUT:

Let AJAX change this text

AJAX XMLHttpRequest:

AJAX - Create an XMLHttpRequest Object

- The keystone of AJAX is the XMLHttpRequest object.

The XMLHttpRequest Object

- All modern browsers support the XMLHttpRequest object (IE5 and IE6 use an ActiveXObject).
- The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

variable=new XMLHttpRequest();

Old versions of Internet Explorer (IE5 and IE6) uses an ActiveX Object:

variable=new ActiveXObject("Microsoft.XMLHTTP");

To handle all modern browsers, including IE5 and IE6, check if the browser supports the XMLHttpRequest object. If it does, create an XMLHttpRequest object, if not, create an ActiveXObject:

Example

```
var xmlhttp;
   if (window.XMLHttpRequest)
     {    // code for IE7+, Firefox, Chrome, Opera, Safari
       xmlhttp=new XMLHttpRequest();
      }
   else
```

```
{   // code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
```

AJAX - Send a Request To a Server

- The XMLHttpRequest object is used to exchange data with a server.

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Method	Description
open(method,url,async)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. method: the type of request: GET or POST url: the location of the file on the server async: true (asynchronous) or false (synchronous)
send(string)	Sends the request off to the server. string: Only used for POST requests

GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server)
- Sending a large amount of data to the server (POST has no size limitations)
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET

AJAX - Server Response

Server Response

To get the response from a server, use the responseText or responseXML property of the XMLHttpRequest object.

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

The responseText Property

- If the response from the server is not XML, use the responseText property.
- The responseText property returns the response as a string, and you can use it accordingly:

Example

```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

The responseXML Property

- If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property:

Example

Request the file cd_catalog.xml and parse the response:

```
xmlDoc=xmlhttp.responseXML;
txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)
 {
 txt=txt + x[i].childNodes[0].nodeValue + "<br>";
 }
document.getElementById("myDiv").innerHTML=txt;
```

5.7 PROGRAMMING IN AJAX

```
<html>
	<head>
		<script type="text/javascript">
			function loadXMLDoc()
			{
				var xmlhttp;

				if (window.XMLHttpRequest)
				{	// code for IE7+, Firefox, Chrome, Opera, Safari
					xmlhttp=new XMLHttpRequest();
				}
				else
				{	// code for IE6, IE5
					xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
				}
				xmlhttp.onreadystatechange=function()
				{
					if (xmlhttp.readyState==4 && xmlhttp.status==200)
					{

	document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
					}
				}
					xmlhttp.open("GET","ajax_info.txt",true);
					xmlhttp.send();
			}
		</script>
	</head>
	<body>
		<div id="myDiv"><h2>Let AJAX change this text</h2></div>
		<button type="button" onclick="loadXMLDoc()">Change
Content</button> </body></html>
```

Output:

AJAX is not a new programming language.

AJAX is a technique for creating fast and dynamic web pages.

Load an XML file with AJAX

```
<html>
	<head>
		<script type="text/javascript">
			function loadXMLDoc(url)
			{
				var xmlhttp;
			if (window.XMLHttpRequest)
			  {	// code for IE7+, Firefox, Chrome, Opera, Safari
				 xmlhttp=new XMLHttpRequest();
			  }
			else
			  {	// code for IE6, IE5
				  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
			  }
			xmlhttp.onreadystatechange=function()
			{
			  if (xmlhttp.readyState==4 && xmlhttp.status==200)
			{

document.getElementById('A1').innerHTML=xmlhttp.status;

document.getElementById('A2').innerHTML=xmlhttp.statusText;

	document.getElementById('A3').innerHTML=xmlhttp.responseText;
			 }
			}
			xmlhttp.open("GET",url,true);
			xmlhttp.send();
			}
		</script>
	</head>
<body>
	<h2>Retrieve data from XML file</h2>
	<p><b>Status:</b><span id="A1"></span></p>
	<p><b>Status text:</b><span id="A2"></span></p>
	<p><b>Response:</b><span id="A3"></span></p>
	<button onclick="loadXMLDoc('note.xml')">Get XML data</button>
</body>
</html>
```

Output:

Retrieve data from XML file
Status: 200
Status text: OK
Response: Tove Jani Reminder Don't forget me this weekend! s