

# Web Services

# T.Y.B.Sc. Computer Science

# Semester V



# Unit I - Web services basics

- What Are Web Services? Types of Web Services (Ref 1: Chap1)
- Distributed computing infrastructure (Ref 1: Chap 2)
- Overview of XML (Ref 1: Chap 3)
- SOAP (Ref 1: Chap 4)
- Building Web Services with JAX-WS (Add Ref 2: Chap 19)
- Registering and Discovering Web Services (Ref 1: Chap 6)
- Service Oriented Architecture (Ref 1: Chap 8)
- Web Services Development Life Cycle (Ref 1: Chap 16)
- Developing and consuming simple Web Services across platform

# Chapter 1

# Web Services

# Web Service

- A Web service is a self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application.
- Services available over the internet.
- Web services constitute a distributed computer infrastructure made up of many different interacting application modules trying to communicate over private or public networks (including the Internet and Web) to virtually form a single logical system.

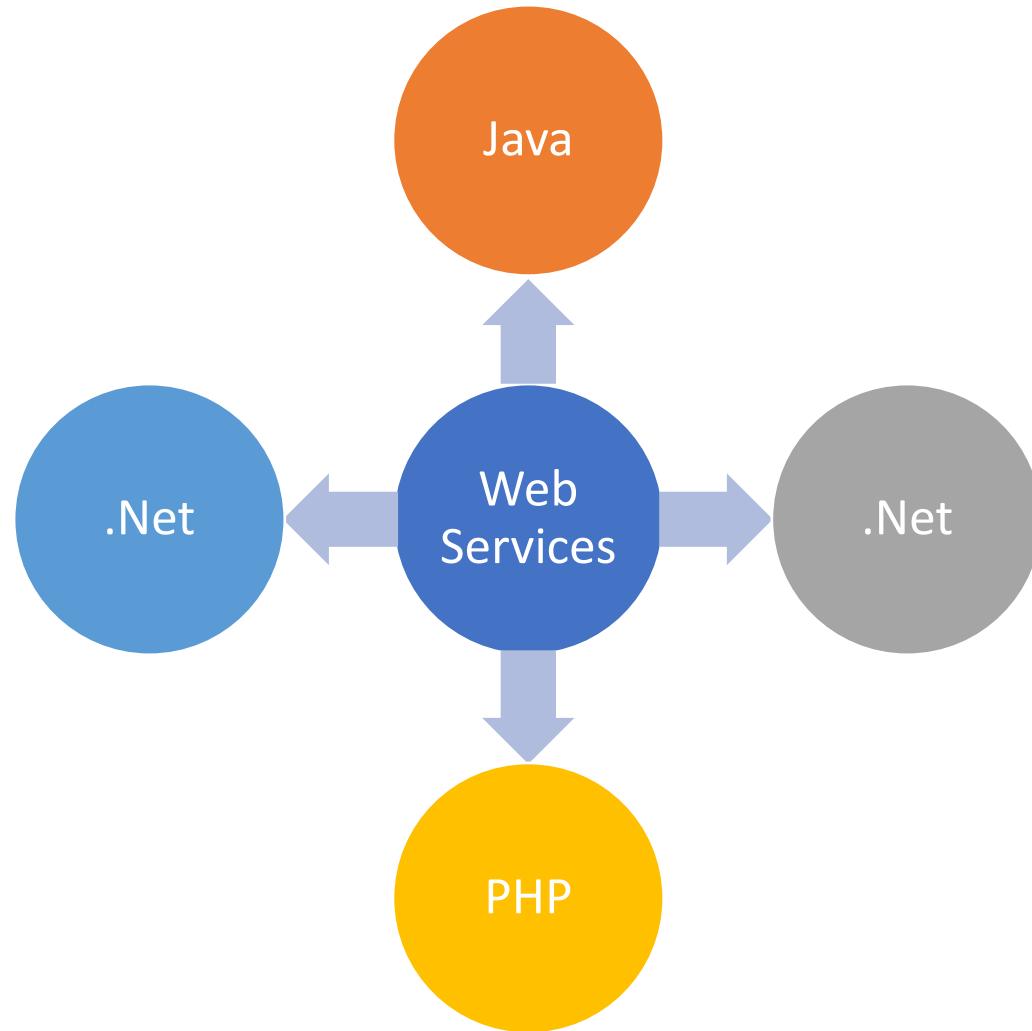
# Web Service

- A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system.
- XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response.
- As all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix applications.

# Web Service

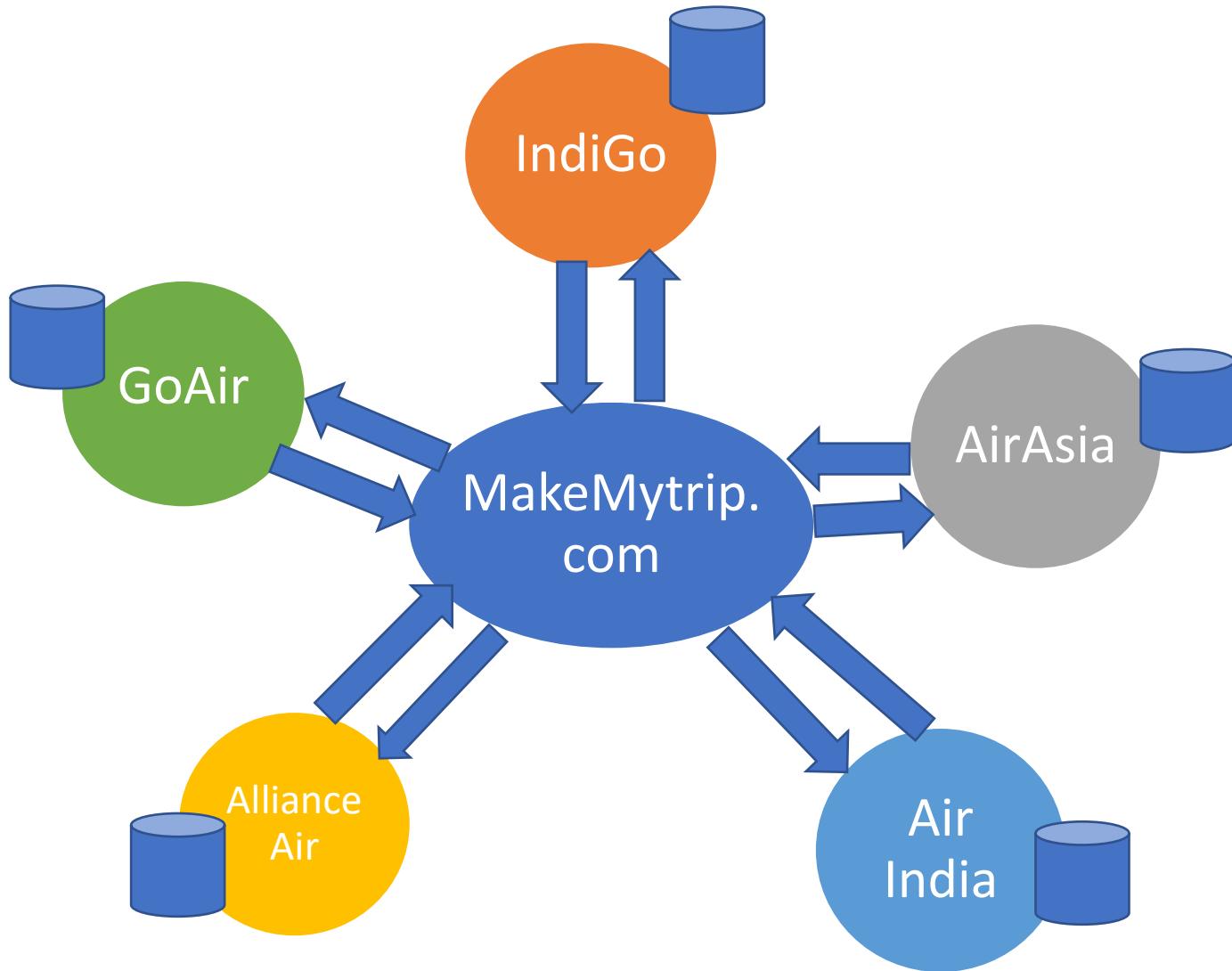
- service available over the web
- enables communication between applications over the web
- provides a standard protocol/format for communication
  
- platform independent communication
- using web services two different applications (implementation) can talk to each other and exchange data/information

# What is Web Service

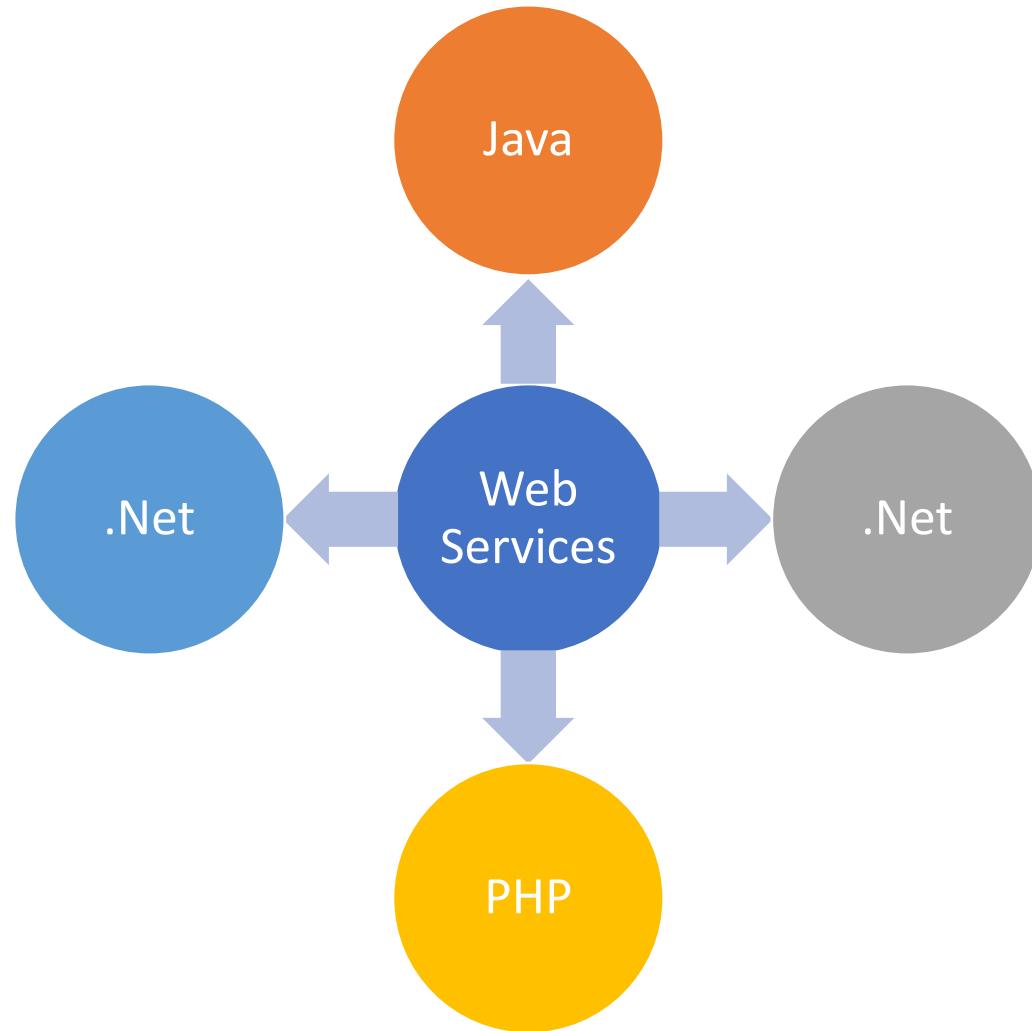


Java, .net, and PHP applications can communicate with other applications through web service over the network. So web service is a language independent way of communication.

e.g. Web Service e.g.



# What is Web Service



Java, .net, and PHP applications can communicate with other applications through web service over the network. So web service is a language independent way of communication.

# Benefits of Web Services

**Interoperability**

- accessible over network and uses HTTP/SOAP protocol
- XML/JSON

**Reusability**

- once written can be used by many client

**Loose Coupling**

- totally independent with server code

**Standardized Protocol**

- use well-defined protocols in the web services protocol

**Low Cost Communication**

- use SOAP over HTTP protocol, so you can use your existing low-cost internet

**Exposing the Existing Function on the network**

Easy to deploy and integrate, just like web applications

# Benefits of Web Services

- **Interoperability:** Web services are accessible over network and uses HTTP/SOAP protocol vis XML/JSON for data transportation. JSON and XML can be programmed in any programming language.
- **Reusability:** A web services once written can be used by many client applications at the same time.
- **Loose Coupling:** Web services client code is totally independent with server code, so we have achieved loose coupling in our application.

# Benefits of Web Services ...

- **Standardized Protocol:** All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack. This standardization of protocol stack gives the business many advantages such as a wide range of choices, reduction in the cost due to competition, and increase in the quality.
- **Low Cost Communication:** Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services.
- **Exposing the Existing Function on the network:**
  - Easy to deploy and integrate, just like web applications.
  - Multiple service versions can be running at same time.

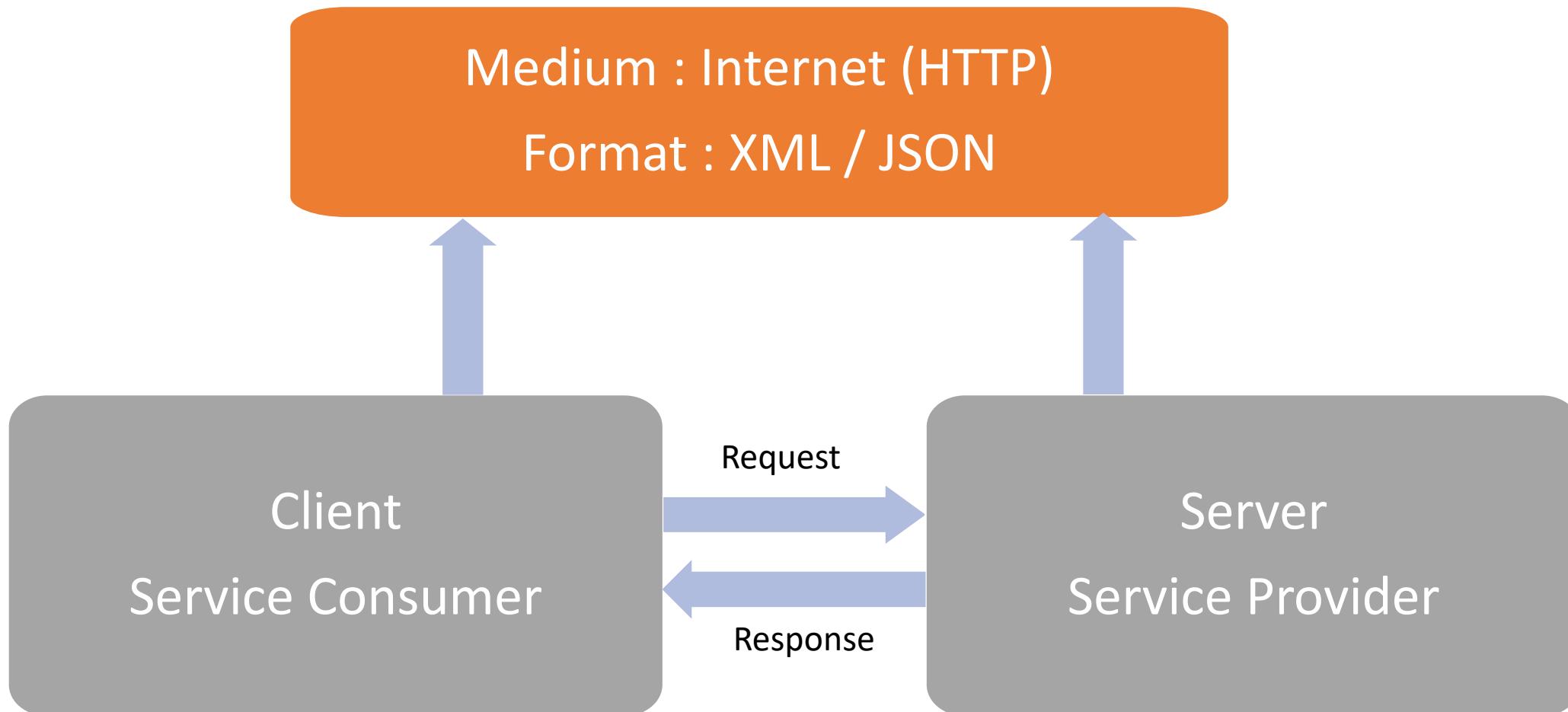
- Service Provider

Implements the application and makes it available over the internet.

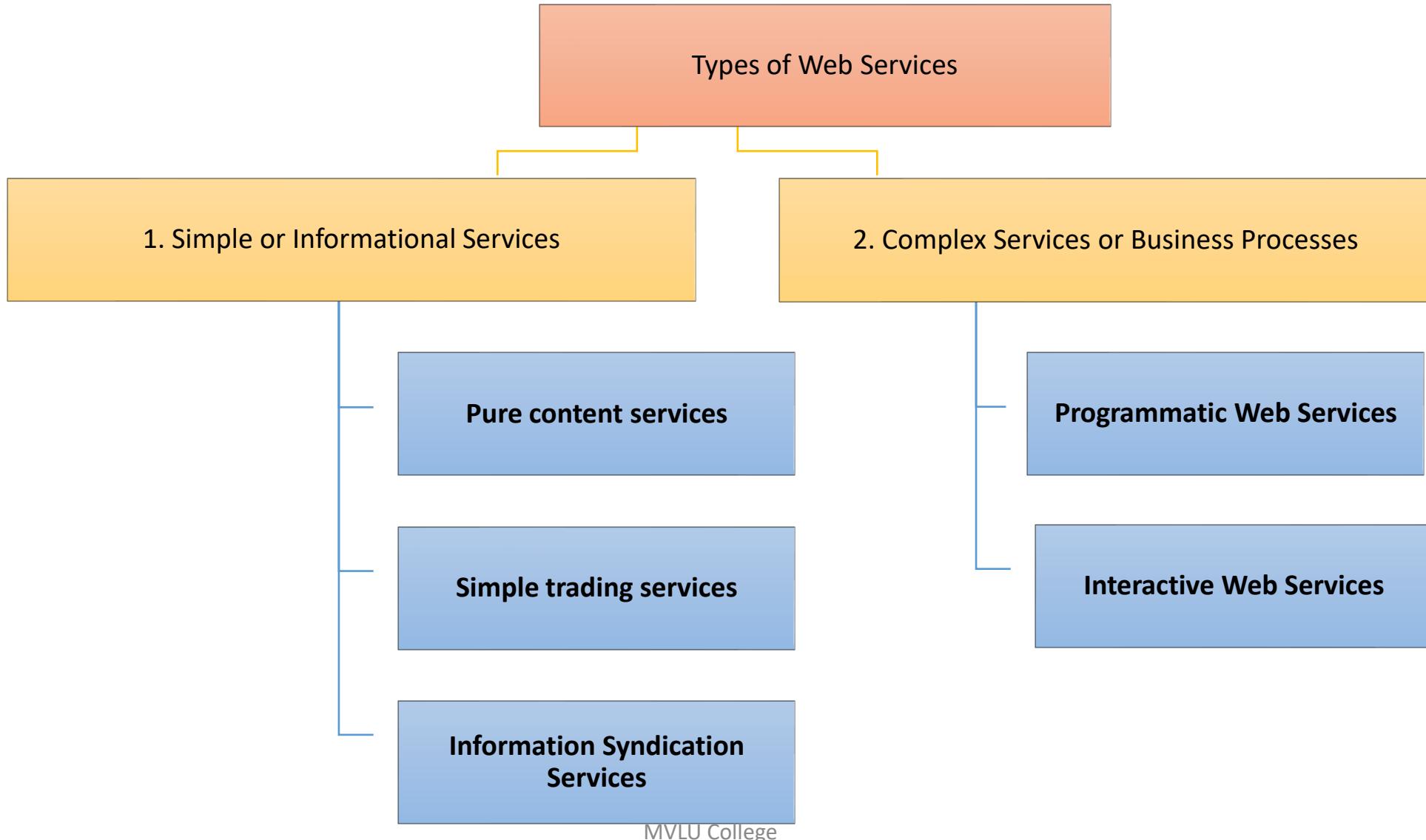
- Service Customer (Client)

To consume service

# Web Service

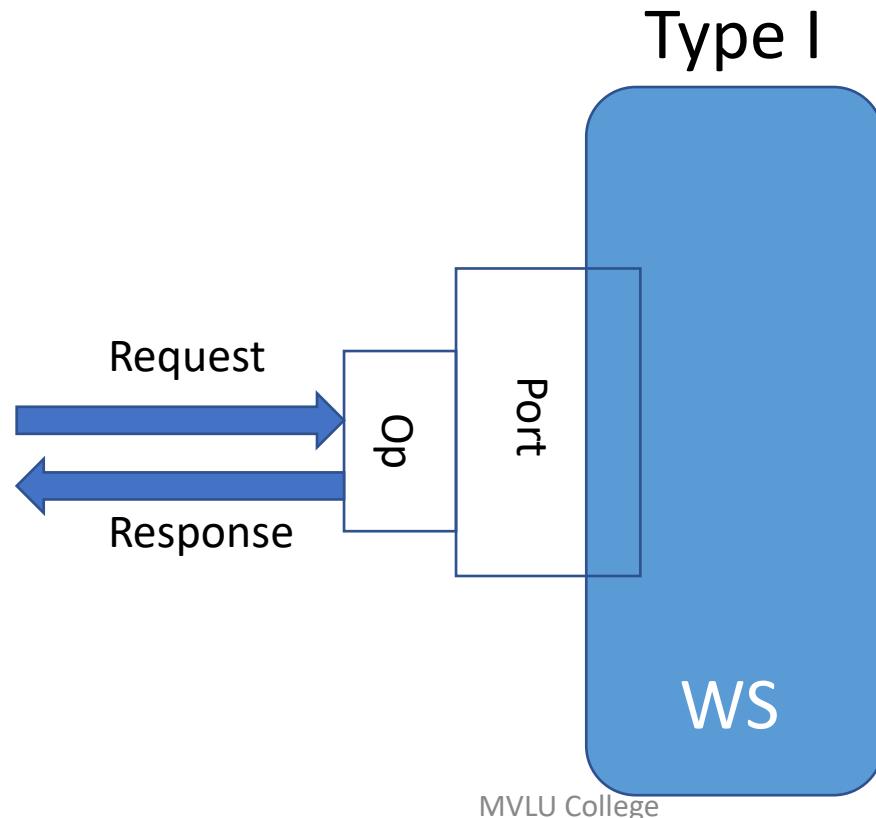


# Types of Web Services



# 1. Simple or Informational Services

- Its simplest form of web service which support only simple request/response operations and always wait for a request; they process it and respond.



# Informational Services are categories into three types



- 1. Pure content services :** These services provide programmatic access to content or information such as currency conversion, weather report information, simple financial information, stock quote information, design information, news items, and so on.

# Informational Services are categories into three types



**2. Simple trading services :** which are more complicated forms of informational services that can provide a seamless **aggregation** of information across disparate systems and information sources, including back-end systems, giving programmatic access to a business information system so that the requestor can make informed decisions.

E.g. Logistics Services

# Informational Services are categories into three types

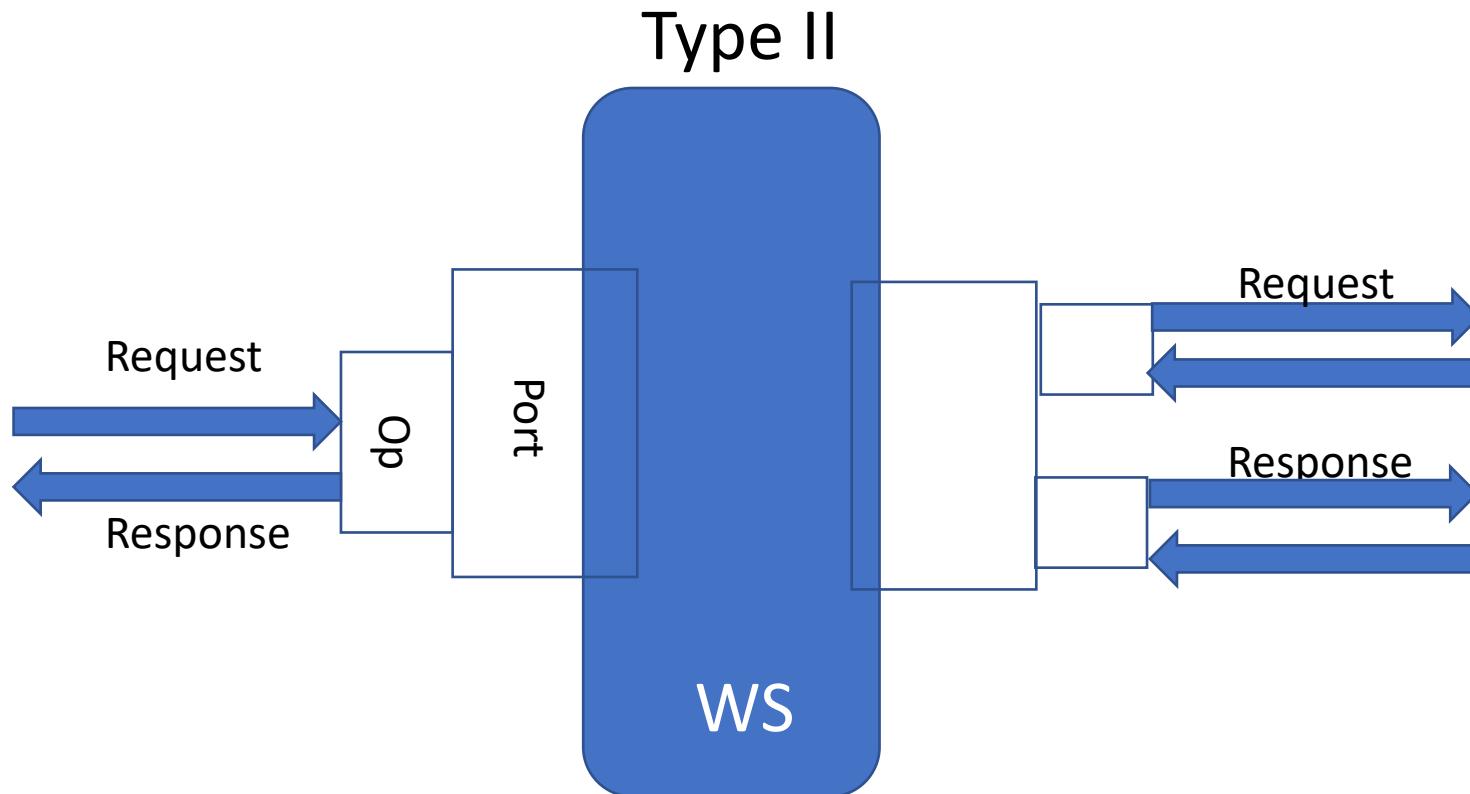


- **Information Syndication Services** : which are value-added information Web services that purport to “plug into” commerce sites of various types, such as e-marketplaces, or sell-sites.
- Typical examples of syndicated services might include reservation services on a travel site or rate quote services on an insurance site.

## 2. Complex Services or Business Process

- Complex Services generally are the combination of two or more services which are combining for business purpose . E.g. Supply-chain application involving order taking, sourcing, stocking orders, financials , inventory controls .
- When enterprises need to compose several services together to create a business process such as customized ordering, customer support, procurement, and logistical support, they need to use complex Web services.

## 2. Complex Services or Business Process



# Complex Services are categories into two types :

- **Programmatic Web Services** : Programmatic web services means services which are run by program, they does not required human interaction. For example. Stock availability, checking services
- **Interactive Web Services**: These Web services can need interactive business process into their Web applications. Presenting integrated application from external service providers means they require some sort of human interaction or interaction from main application.



# Web Service Questions

1. What is Web Services?
2. Explain the features of Web Services
3. What are the types of Web Services?
4. Write a short note on Simple or Informational Services.
5. Write a short note on Complex Services or Business Processes.

# Distributed computing infrastructure



- Distributed system is characterized as a collection of (probably heterogeneous) networked computers, which communicate and coordinate their actions by passing messages. Distribution is transparent to the user so that the system appears as a single integrated facility.
- A distributed system has numerous operational components (computational elements, such as servers and other processors, or applications) which are distributed over various interconnected computer systems.

# Distributed computing infrastructure



- Distributed systems usually use some kind of client–server organization. A computer system that hosts some component of a distributed system is referred to as a *host*. Distributed components are typically heterogeneous in that they are written in different programming languages and may operate under different operating systems and diverse hardware platforms

# Internet protocols



- To enable data to be transmitted across the Internet typical distributed platforms, such as J2EE, rely on the support of Internet protocols.
- The most prominent of the Internet protocols is the Transport Control Protocol over Internet Protocol (or TCP/IP). The Internet Protocol (IP), the basic protocol of the Internet, enables the unreliable delivery of individual packets from one host to another.
- The Transport Control Protocol (TCP) adds the notions of connection and reliability.
- These two protocols provide for the reliable delivery of streams of data from one host to another across communication networks, and the Internet in particular.

# Middleware

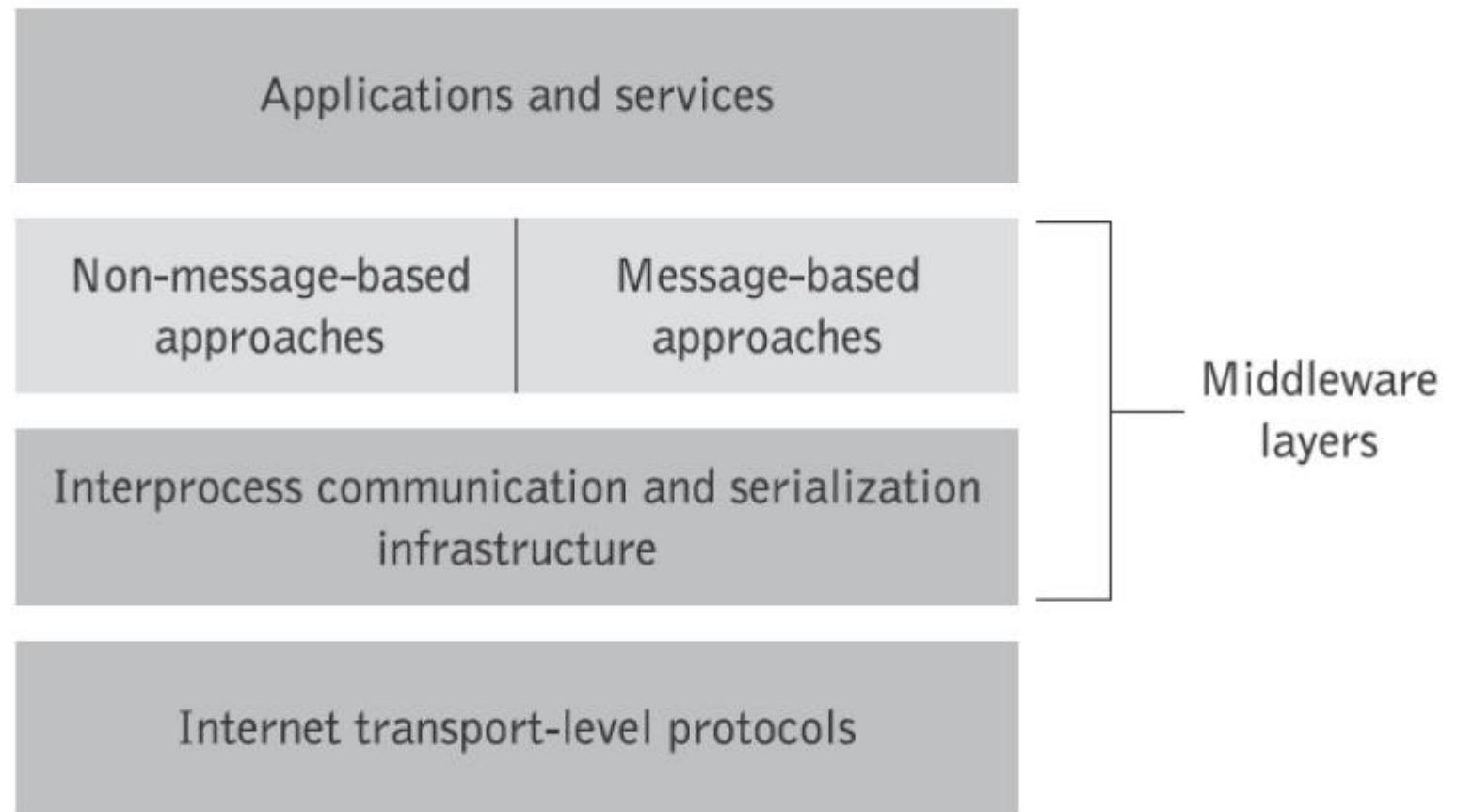
- Middleware is connectivity software that is designed to help manage the complexity and heterogeneity inherent in distributed systems by building a bridge between different systems thereby enabling communication and transfer of data.
- *Middleware* could be defined as a layer of enabling software services that allow application elements to interoperate across network links, despite differences in underlying communications protocols, system architectures, operating systems, databases, and other application services.

# Middleware

- The role of middleware is to ease the task of designing, programming, and managing distributed applications by providing a simple, consistent, and integrated distributed programming environment. Essentially, middleware is a distributed software layer, or “platform,” that lives above the operating system and abstracts over the complexity and heterogeneity of the underlying distributed environment with its multitude of network technologies, machine architectures, operating systems, and programming languages.

- The bottom layer is concerned with the characteristics of protocols for communicating between processes in a distributed system and how the data objects, e.g., a sales order, and data structures used in application programs can be translated into a suitable form for sending messages over a communications network, taking into account that different computers may rely on heterogeneous representations for simple data items. The layer above is concerned with interprocess communication mechanisms, while the layer above that is concerned with non-message- and message-based forms of middleware.

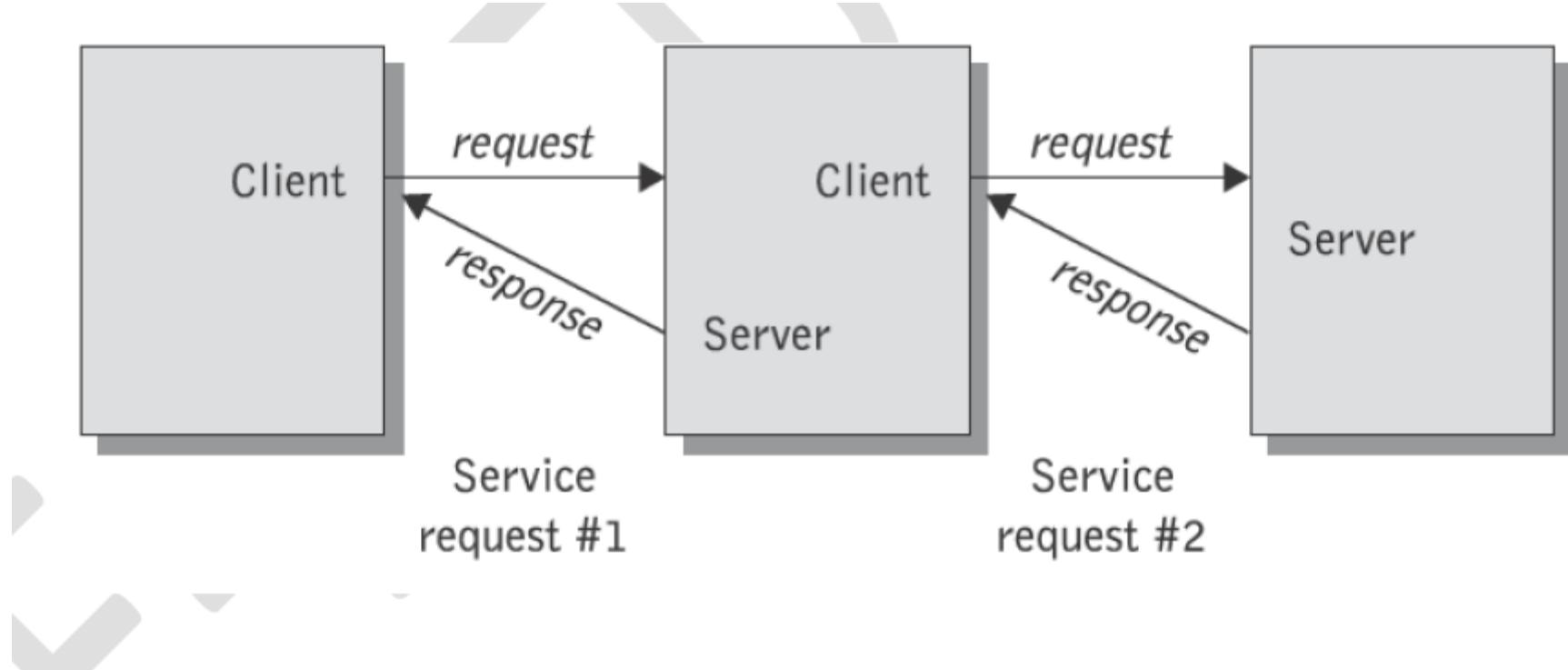
- *Non-message-based* forms of *middleware* provide synchronous communication mechanisms designed to support client–server communication.
- *Message-based* forms of *middleware* provide asynchronous messaging and event notification mechanisms to exchange messages or react to events over electronic networks.



# client–server model

- The client–server architecture is one of the common solutions to the conundrum of how to handle the need for both centralized data control and widespread data accessibility.
- Client–server involves client processes (service consumers) requesting service from server processes (service providers).
- Servers may in turn be clients of other servers. For instance, a Web server is often a client of a local file server (or database server) that manages the files (storage structures) in which Web pages are stored

# client–server model



## Messaging

- Distributed systems and applications communicate by exchanging messages. *Messaging* is a technology that enables high-speed, asynchronous, program-to-program communication with reliable delivery.
- Programs communicate by sending packets of data called *messages* to each other. The concept of a message is a well-defined, data-driven text format – containing the business message and a network routing header – that can be sent between two or more applications.
- A message typically comprises three basic elements: a *header*, its *properties*, and a *message payload* or *body*.

- The message **header** is used by both the messaging system and the application developer to provide information about characteristics such as the destination of a message, the message type, the message expiration time, and so forth.
- The **properties** of a message contain a set of application-defined name/value pairs.
- The **message** body carries the actual “payload” of the message. The format of the message payload can vary across messaging implementations. Most common formats are plain text, a raw stream of bytes for holding any type of binary data, or a special XML message type that allows the message payload to be accessed using any number of common XML parsing technologies.
- Message passing between a pair of processes is supported by two message communication operations: *send* and *receive*, defined in terms of destinations and messages.

- *Marshalling* is the process of taking an object or any other form of structured data items and breaking it up so that it can be transmitted as a stream of bytes over a communications network in such a way that the original object or data structure can be reconstructed easily on the receiving end.
- *Unmarshalling* is the process of converting the assembled stream of bytes on arrival to produce an equivalent object or form of structured data at the destination point. Therefore, marshalling comprises the transformation of structured data items and primitive values into an agreed standard form of representation for transmission across the network.
- Java and XML use the terms *serialization* and *deserialization* to denote the process of marshalling and unmarshalling

# Message destinations and sockets

- messages are sent to ports.
- Servers usually publicize their port numbers for use by clients and processes use multiple ports from which to receive messages.
- During interprocess communication messages are sent to (Internet address, local port) pairs.
- Many applications involve two processes in different hosts communicating with each other over a network. These two processes communicate with each other by exchanging (sending and receiving) messages. A process sends messages into, and receives messages from, the network through its *socket*.
- A process's socket could be thought of as the entry point to the process. Interprocess communication consists of transmitting a message between a client and a socket in another process. Once the message arrives at its destination, it passes through the receiving process's socket and the receiving process then acts on the message.

# Synchronous and asynchronous forms of message communication

- These modes are: *synchronous* or time dependent and *asynchronous* or time independent.
- The defining characteristic of a synchronous form of execution is that message communication is synchronized between two communicating application systems, which must both be up and running, and that execution flow at the client's side is interrupted to execute the call. Both the sending and the receiving application must be ready to communicate with each other at all times.
- When using asynchronous messaging, the caller employs a *send and forget* approach that allows it to continue to execute after it sends the message. With asynchronous communication, an application sends (requestor or sender) a request to another while it continues its own processing activities. The sending application does not have to wait for the receiving application to complete and for its reply to come back.

# Synchronous forms of middleware

- Programming models for synchronous forms of middleware are composed of cooperating programs running in several interacting distributed processes. Such programs need to be able to invoke operations synchronously in other processes, which frequently run in different computing systems.
- The most familiar approaches to non-message-based forms of middleware are typified by the remote procedure call (RPC) and the remote method invocation (RMI).

# Remote procedure calls

- RPC is a basic mechanism for interprogram communication. In effect, RPC is the middleware mechanism used to invoke a procedure that is located on a remote system, and the results are returned.
- By design, the RPC programming style mimics the serial thread of execution that a “normal” non-distributed application would use, where each statement is executed in sequence.

# Remote procedure calls



- The RPC mechanism is the simplest way to implement client–server applications because it keeps the details of network communications out of the application code.
- In RPC-style programming, an object and its methods are “remoted” such that the invocation of the method can happen across a network separation.
- In client application code, an RPC looks like a local procedure call, because it is actually a call to a local proxy known as a *client stub* (a surrogate code that supports RPCs).
- The client stub communicates with a *server stub* using the RPC runtime library, which is set of procedures that support all RPC applications. A server stub is like a *skeleton* method in that it unmarshals the arguments in the request message, calls the corresponding service procedure, and marshals the return results for the reply message.
- The server stub communicates its output to the client stub, again by using the RPC runtime library. Finally, the client stub returns to the client application code.





# Chapter 2

# Distributed Computing

# Distributing computing infrastructure



- Is a collection of separate and independent software/hardware components, called nodes, that are networked and work together coherently by coordinating and communicating through message passing or events, to fulfil one end goal.
- A distributed system has numerous operational components (computational elements, such as servers and other processors, or applications) which are distributed over various interconnected computer systems.

# Distributing computing infrastructure



- Nodes could be unstructured or highly structured, depending on the system requirements. And the complexities of the system are hidden to the end user, making the whole system appear as a single computer to its users.
- Distributed systems usually use some kind of client–server organization. A computer system that hosts some component of a distributed system is referred to as a host. Distributed components are typically heterogeneous in that they are written in different programming languages and may operate under different operating systems and diverse hardware platforms.

# Distributed systems are all around us!



- Google Search Engine
- Amazon Platforms
- Netflix
- Blockchain
- Online Gaming
- Money Transfer
- Online Banking

# What is a Distributed System?



# Characteristics Distributing computing

- Resource Sharing : is the ability to use any h/w , s/w or data anywhere in the systems.
- Openness : Is concerned with extensions and improvements of Distributed systems
- Concurrency : Multiple operations and tasks performed in parallel.
- Scalability : Increase the scale of systems.
- Fault tolerance : It cares the reliability of systems

# Client Server Model

- The web is service that allow computers to share and exchange a data.
- The web is referred to as Client-Server program



# Client



A client can be a machine or a program

A client machine : e.g. laptop,  
smartphone

A client program is a program that  
allows the user to make requests

E.g. Web Browser , Word processing  
tools.

# Servers

- A server is a computer program Not a Device
- High-Performance computers are called servers because they run server programs
- Servers provide functionality and serve other programs called clients.
- A single server can serve multiple clients at the same time
- We can run multiple servers on a single machine, they are called virtual servers
- A server is always listening for requests, and as soon as it receives one, responds with a message.

## Web Servers

- (Serve HTTP requests)

## Database Servers

- (Runs DBMS)

# The Interprocess Communication



- Inter process communication (IPC) is a system which permits processes to communicate or coordinate with one another and provide synchronization between their activates.
- Two different machines with different operating system can communicate with each other via interprocess communication.

# Message Passing

- In the distributed system two applications can communicate with each other by means of exchanging messages. Messaging provides Asynchronous, high speed, program-to program communication with reliable or trusted delivery.
- Two operations involved in message communication : Send and Receive, specified in form of source , destination and messages.
- Programs communicate by sending packets of data called *messages* to each other. The concept of a message is a well-defined, data-driven text format – containing the business message and a network routing header – that can be sent between two or more applications.

# Synchronous and Asynchronous Communication



- Messages can be passing using following two ways

Synchronous  
Communication

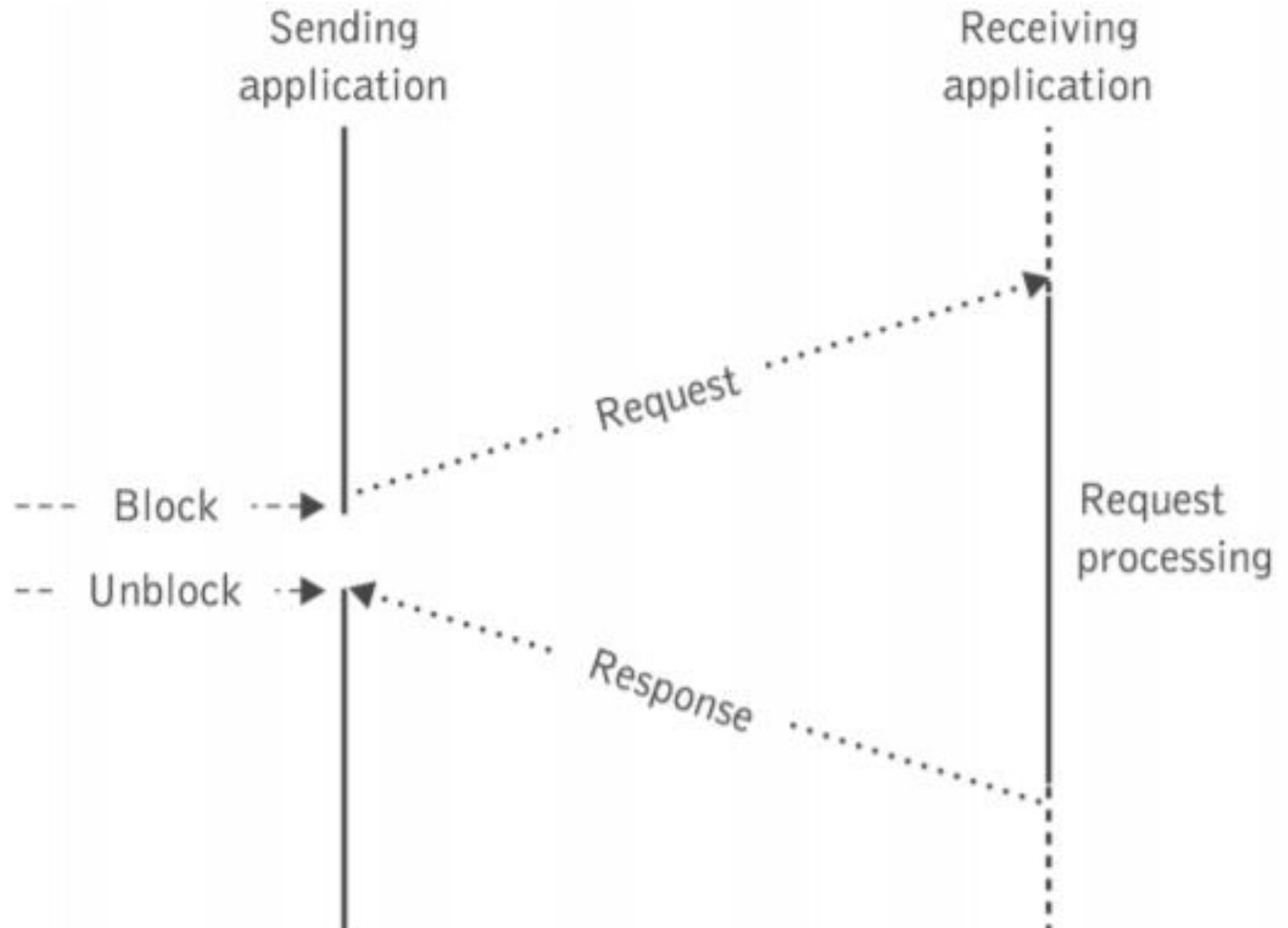
Asynchronous  
Communication

# Synchronous Communication

- Synchronous Communication : Synchronized means client side is interrupted or wait till execution of message passing. Client cannot send another message till previous message passing finish its working.
- E.g. Loading a web page or a conversation between two people

# Asynchronous Communication

- Asynchronous Communication : Asynchronous means simultaneous execution i.e. Client can send multiple messages, so no need to wait for processing of previous message.
- E.g. Email correspondence, SMS



# Question : synchronous and asynchronous ?

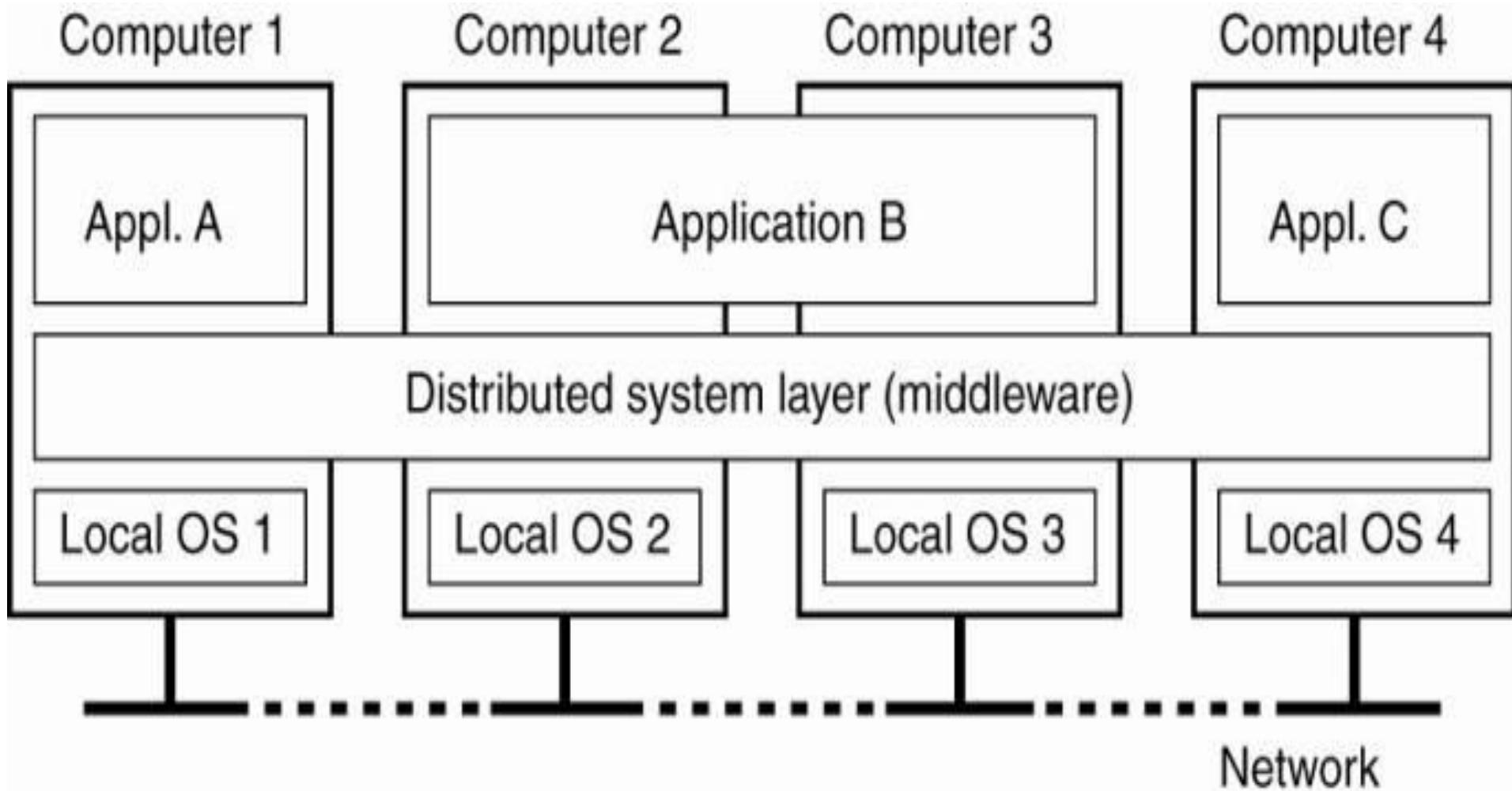


- Face to face Discussion
- Order food in Restaurant

# The Middleware

- In the distributed system, the middleware is software or component that provides interface or communication between two heterogenous applications.
- It supports the various components or device of a distributed system to communicate with each other and manages, transfer data properly.
- Middleware supports and simplifies communication between complex distributed applications.
- Middleware includes and integrates different tools and technologies like XML, Web Services, SOAP, SOA etc.

# The Middleware



- Layer between OS and distributed applications
- Hide complexity and heterogeneity of distributed system
- Bridges gap between low-level OS communication and programming language abstractions.
- Provides common programming abstraction and infrastructure for distributed applications

# Major Middleware Concepts

- Synchronous communication provides a tightly coupled environment in which an application strongly need to know the details of how to reach and communicate with other applications. Sender should wait till receiver receives and process the message through it wants to send another message.

## Major middleware concepts

i. Remote Procedure Call (RPC)

ii. Object-Oriented Middleware (OOM) e.g. RMI

iii. Message-Oriented Middleware (MOM) (Java Message Service)

iv. Event Based Middleware

## Middleware

1. Synchronous Forms of  
Middleware

2. Asynchronous Forms of  
Middleware

# Synchronous Forms of Middleware

- Synchronous communication provides a tightly coupled environment in which an application strongly need to know the details of how to reach and communicate with other applications. Send should wait till receiver receives and process the message through it wants to send another message.

## Synchronous Forms of Middleware

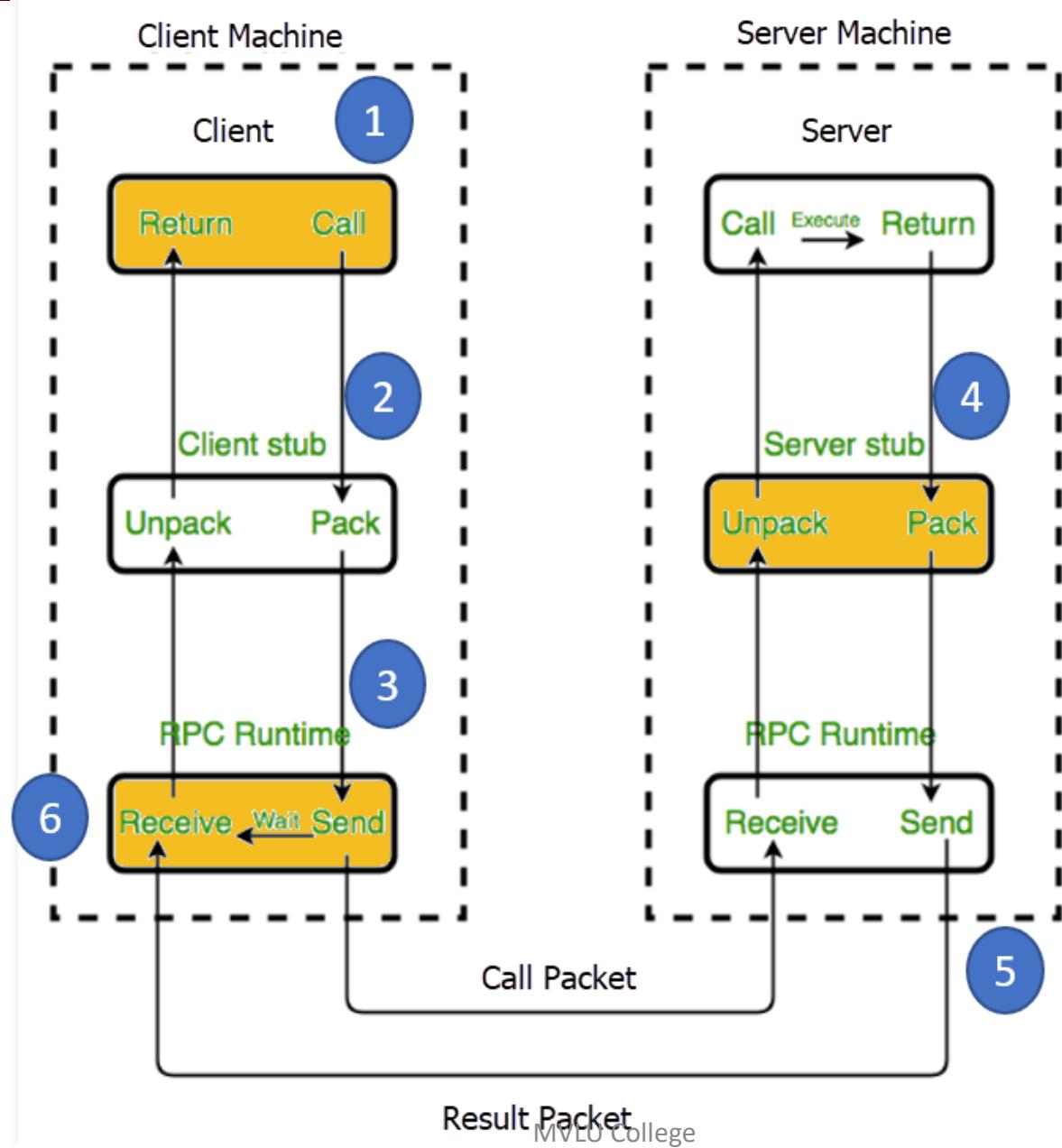
1. Remote Procedure Call  
(RPC)

2. Remote Method Invocation  
(RMI)

# 1. Remote Procedure Call (RPC)

- RPC is a basic mechanism for inter-program communication, where the application elements use a request/wait-for-reply (synchronous) model of communication.
- RPC translates a client request or call and sends to server.
- After that server receive the request and provide appropriate response to the client.

# 1. Remote Procedure Call (RPC)



## Following steps take place during RPC process:

1. The client, the client stub, and one instance of RPC run time execute on the client machine.
2. A client starts a client stub process by passing parameters.
3. In this stage, RPC accessed by the user by making regular Local Procedural Call. RPC Runtime manages the transmission of messages between the network across client and server. It also performs the job of retransmission, acknowledgment, routing, and encryption.

# Following steps take place during RPC process:



4. After completing the server procedure, it returns to the server stub, which packs (Marshalls) the return values into a message. The server stub then sends a message back to the transport layer.
5. In this step, the transport layer sends back the result message to the client transport layer, which returns back a message to the client stub.
6. In this stage, the client stub demarshalls (unpack) the return parameters, in the resulting packet, and the execution process returns to the caller.

# Features of RPC

- Simple call syntax
- Offers known semantics
- Provide a well-defined interface
- It can communicate between processes on the same or different machines

# Remote method invocation



- The Java RMI provides a simple and direct model for distributed computation with Java objects on the basis of the RPC mechanism.
- If the particular method happens to be on a remote machine, Java provides the capability to make the RMI appear to the programmer to be the same as if the method is on the local machine.

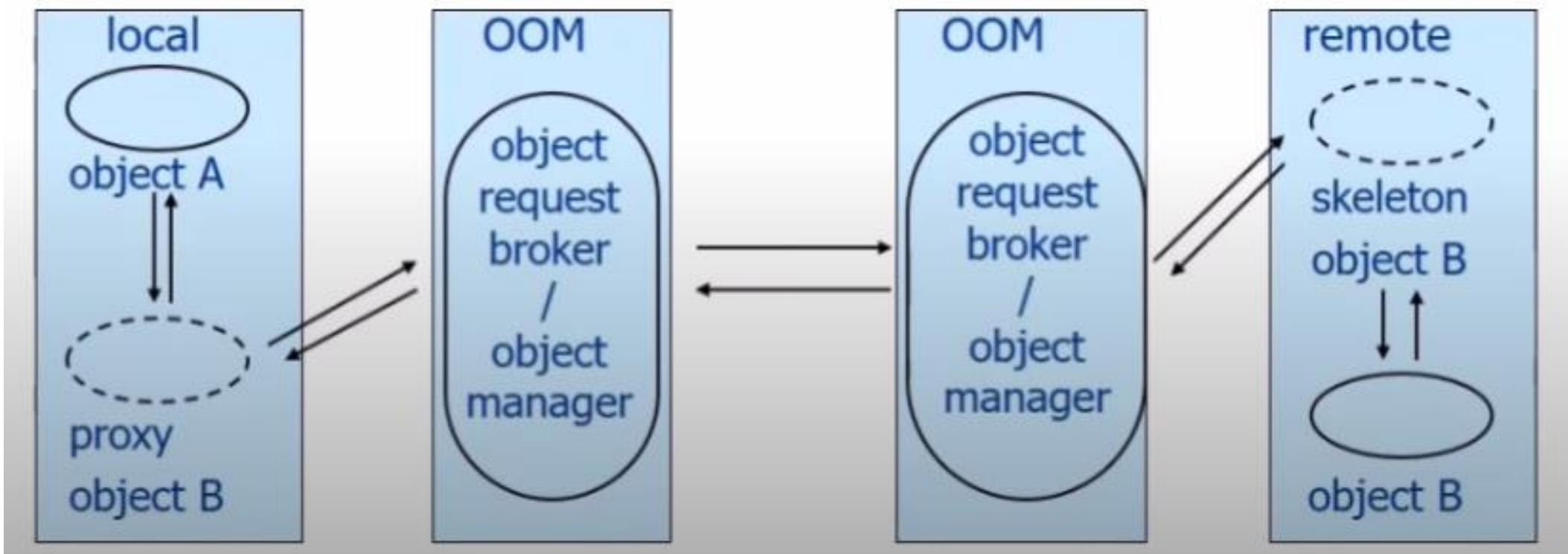
# Remote method invocation



- RMI applications comprise two separate programs: a server and a client. RMI provides the mechanism by which the server and the client communicate and pass information back and forth.
- There are two different kinds of classes that can be used in RMI: *remote* and *serializable* classes.
- A remote object is an instance of a remote class. When a remote object is used in the same address space, it can be treated just like an ordinary object. But if it is used externally to the address space, the object must be referenced by an object handle.

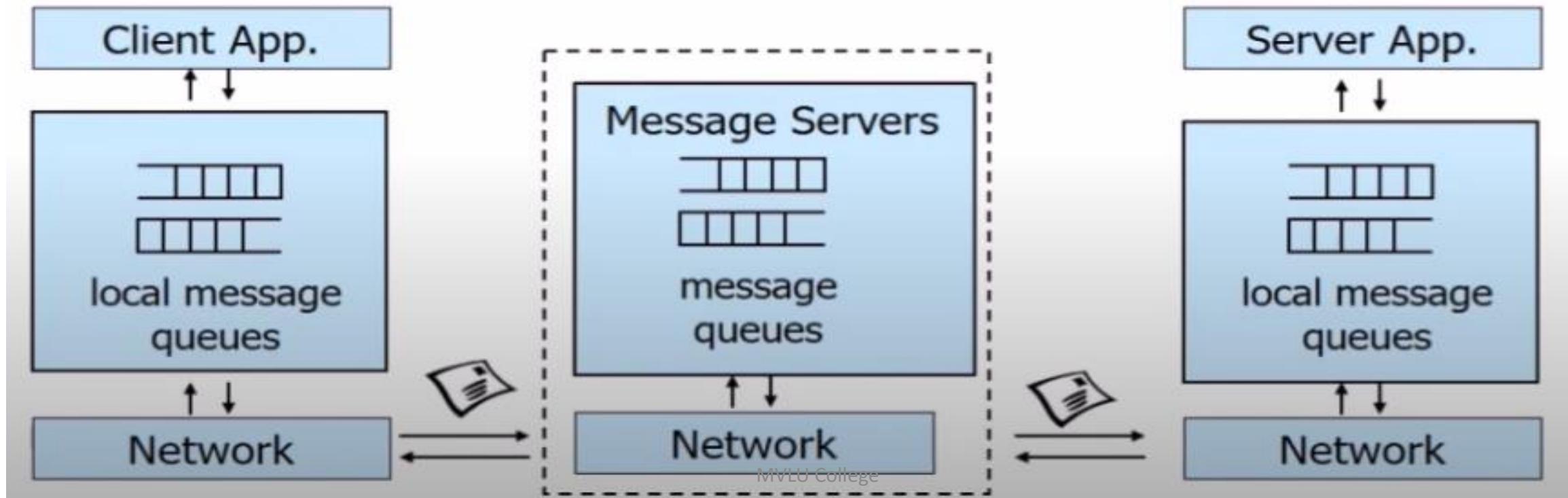
# Remote Method Invocation

- Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side). RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object.



# Message Oriented Middleware (MOM)

- Communication using messages.
- Messages stored in message queues.
- Message server decouple
- Asynchronous interaction



- JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication.
- **Type of JMS**
  - 1) Point-to-point
  - 2) Publish/subscribe
  -

# 1) Point-to-Point (One-to-One)

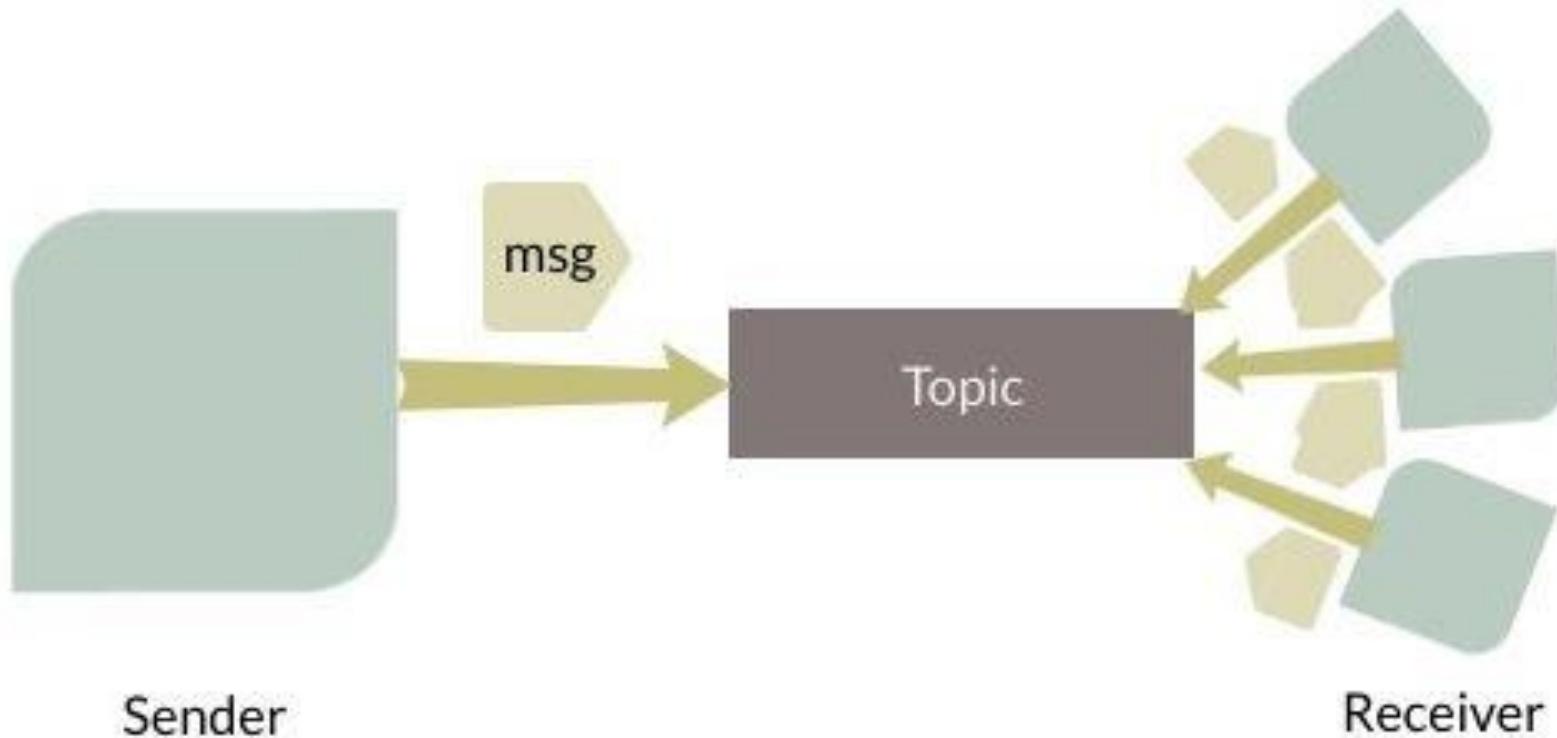
- In the point-to-point message bean, we have one sender and one receiver. It is like personal chat in mobile, but this messaging is between components. When the sender sends a message, if there is more than one message, then it has to wait in the queue. Once it reaches to a receiver, a receiver can consume it and acknowledge it.
- 



## 2) Publish/Subscriber (One-to-Many)

- Publish/subscriber is like Netflix. With Netflix, we have one provider (sender) and many consumers (receiver). Many users can subscribe to Netflix and watch a T.V. show or movie uploaded by Netflix. This is the same way as with the publish/ subscriber; the first component needs to subscribe. After subscribing, the component can consume and acknowledge messages. If there is more than one message, then it has to wait on a topic.

## 2) Publish/Subscriber (One-to-Many)

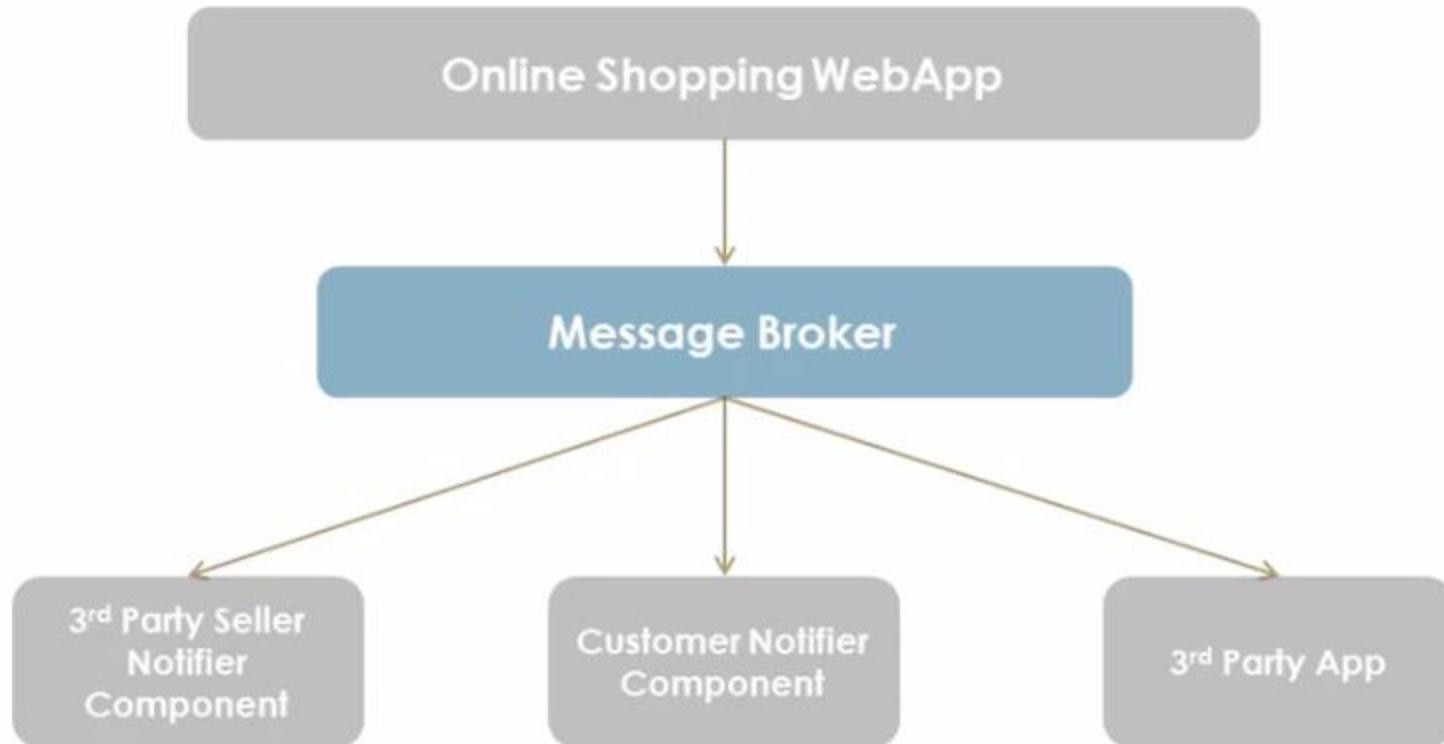


Sender

Receiver

# Message Oriented Middleware (MOM)

## Real-world Use Cases



# Popular Message Brokers

- RabbitMQ
- Apache ActiveMQ
- ZeroMQ
- Kafka

# Distributing Computing Questions

1. Write short note on distributing computing.
2. Write a short note on Client-Server Architecture.
3. What is interprocess Communication?
4. Explain the working of Middleware
5. What are the types of Middleware
6. Explain Synchronous forms of middleware.
7. What is RPC? Explain the its advantages.
8. What is Java Messages Services ? Types of JMS ?

# Chapter 3

# Overview of XML

# Extensible Markup Language

- **XML** stands for **eXtensible Markup Language** and it is used for storing and transferring data. XML doesn't depend on the platform and the software(programming language). You can write a program in any language on any platform (Operating System) to send, receive or store data using XML.

## XML e.g.

```
<?xml version="1.0"?>
<BooksDetails>
    <book>
        <name>A Song of Ice and Fire</name>
        <author>George R. R. Martin</author>
        <language>English</language>
    </book>
</ BooksDetails>
```

# XML Properties



1. XML is a markup language that focuses on data rather than how the data looks.
2. XML is designed to send, store, receive and display data.
3. XML became a W3C (W3C stands for World Wide Web Consortium)
4. XML is different from HTML. XML focuses on data while HTML focuses on how the data looks.
5. XML does not depend on software and hardware, it is platform and programming language independent.
6. XML doesn't have predefined tags, rather you have to create your own tags.

# XML Properties



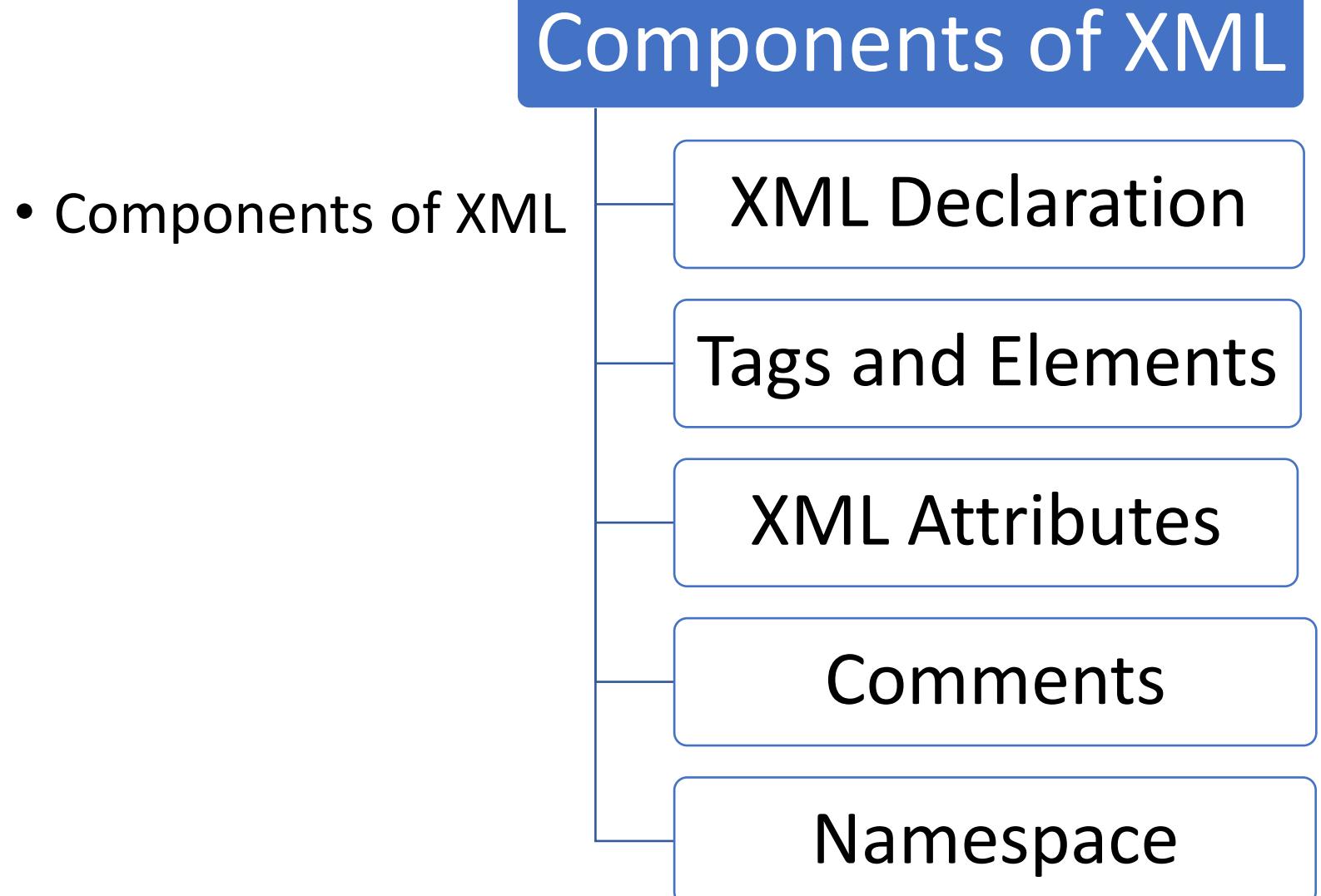
Key Point	XML	HTML
stands for	eXtensible Markup Language	Hyper Text Markup Language
Derived from	XML derived from SGML(Standard Generalized Markup Language).	Where as the HTML derived same from SGML.
Purpose	XML was designed for <b>holds data</b> . Use for <b>transport data between application and database</b> .	HTML was designed for <b>specify how to data should be display on web page</b> .
Rules	XML was follow <b>strict rules</b> . Any time terminate the process if rules break.	HTML was <b>not following any strict rules</b> . All browser try to display data to the best as per its ability.

Case Sensitive	XML is <b>case sensitive</b> .	HTML is <b>not a case sensitive</b> .
Uses	XML uses for <b>temporary/permanent storing data</b> . But now a day use for write markup language that are render/connect to a application and database.	HTML uses for <b>web presentation layer</b> along with CSS you can make very interactive design.
Tags	XML tags you can <b>define custom tag</b> by ourself.	HTML tags are <b>predefined</b> .
How to write	XML tags must have <b>closing tag</b> . Example. <note>Travel experience</note>	HTML tags are two type <b>closing tag</b> or <b>self-closing tag</b> . Example. <b>self-closing tag</b>  , <b>Closing tag</b> <p>Travel experience</p>
Whitespace	XML was preserve only <b>one whitespace</b> .	Where as the HTML was preserve only <b>one whitespace</b> .
Behaviour	XML was <b>dynamic</b> for holding data.	While HTML was <b>static</b> for displaying data.

# XML e.g.

```
<students>
<student>
  <name>Rick Grimes</name>
  <age>35</age>
  <subject>Maths</subject>
  <gender>Male</gender>
</student>
<student>
  <name>Daryl Dixon </name>
  <age>33</age>
  <subject>Science</subject>
  <gender>Male</gender>
</student>
<student>
  <name>Maggie</name>
  <age>36</age>
  <subject>Arts</subject>
  <gender>Female</gender>
</student>
</students>
```

# Components of XML



# XML Declaration

- In XML document , XML declarion is optional .
- `<?xml version="1.0" encoding="UTF-8"?>`
- XML declaration is option, but if you specify it should be the first line of XML document.

# Tags and elements

- Tags and elements
- An XML document consist of different user defined XML elements , also called XML nodes or XML tags.
- <name>Daryl Dixon </name>

# Rules for tags and elements

- Container XML element
- Nesting of elements
- Root Element
- Case sensitive

# XML Attributes

- Attributes provides extra information about elements which is specified using a name/ value pair. An XML element can have any number of attributes. E.g.

# Attributes Rules



- 1. Attributes are name & value pairs. The attribute name should not be in quotes, however the attribute value must always be in quotes (single or double)

<car brand="Ford Figo"> --> Correct

- 2. XML element can have more than one attributes. This we have already seen in the above example, where brand and category attributes are linked to the element <car>.
- 3. Attributes cannot contain duplicate multiple values.

## e.g. of Attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
    <book id="10001">
        <name>A Song of Ice and Fire</name>
        <author>George R. R. Martin</author>
        <language>English</language>
        <genre>Epic fantasy</genre>
    </book>
</Books>
```

# XML comments



- Xml Comments
- XML comments are just like HTML comments. We know that the comments are used to make codes more understandable other developers.
- <!-- This is just a comment -->

# Namespace

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

# XML Validator



- The cool thing about XML is that it can be validated for syntax error using DTD or Schema.

# Well Formed XML document rules

An XML document with correct syntax is known as valid XML document.

1. All XML documents must have a root element.
2. XML is a case sensitive language so you should be careful with the case while opening and closing tags.
3. All XML tags must have a closing tag.
4. XML attribute name should not be quoted while its value must be quoted.

# Ways to check a valid XML document

- There are two document type definitions that can be used with XML documents to check whether the XML document is valid.

**XML DTD**

- Document Type Definition

**XML Schema**

- An XML-based alternative to the Document Type Definition

# 1. XML DTD

- DTD defines the structure of XML document that can be validated against the XML document to check for the syntax errors. XML DTD defines the structure by mentioning the XML elements in such a way so that the complete structure of XML document can be understood. The DTD file has .dtd extension.

# XML DTD (Document Type Definition)



- A DTD defines the legal elements of an XML document
- In simple words we can say that a DTD defines the document structure with a list of legal elements and attributes.
- XML schema is a XML based alternative to DTD.
- Actually DTD and XML schema both are used to form a well formed XML document.
- We should avoid errors in XML documents because they will stop the XML programs.

- A DTD is set of rules that allows us to specify our own set of elements, attributes, and entities . Its grammar that indicates what tags are allowed , in what order they can appear, and how they can be nested.

- The elements which can appear in an XML document.
- The order of elements
- Optional and mandatory elements
- Attributes of elements and also whether they are optional or mandatory.
- Whether or not attributes have default values.

# Types of DTD

- DTD declarations either internal XML document or make external DTD file, after linked to a XML document.

**Internal DTD**

**External DTD**

## Internal DTD

- You can write rules inside XML document using declaration. Scope of this DTD within this document. Advantages is document validated by itself without external reference.

```
<?xml version="1.0"?>
<!DOCTYPE employees [
    <!ELEMENT employees (employee*)>
    <!ELEMENT employee (name,salary)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT salary (#PCDATA)>
    <!ATTLIST employee id CDATA #REQUIRED>
]>
<employees>
    <employee id="1">
        <name>Amit</name>
        <salary>50000</salary>
    </employee>
    <employee id="2">
        <name>Rajesh</name>
        <salary>450000</salary>
    </employee>
</employees>
```

- You can write rules in a separate file (with .dtd extension). later this file linked to a XML document. This way you can linked several XML documents refer same DTD rules.
- External DTD are better because , we can sharing definitions between XML documents. The documents that share the same DTD are more uniform and easier to retrieve. Only the valid XML documents which follows are valuable for exchanging and retrieving information.

```
<?xml version="1.0"?>
<!DOCTYPE employees SYSTEM "employees.dtd">
<employees>
    <employee>
        <firstname>vimal</firstname>
        <lastname>jaiswal</lastname>
        <email>vimal@javatpoint.com</email>
    </employee>
</employees>
```

# Employee.dtd

```
<!ELEMENT employees (employee*)>
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

- **<!DOCTYPE employees :** It defines that the root element of the document is employee.
- **<!ELEMENT employee:** It defines that the employee element contains 3 elements "firstname, lastname and email".
- **<!ELEMENT firstname:** It defines that the firstname element is #PCDATA typed. (parse-able data type).
- **<!ELEMENT lastname:** It defines that the lastname element is #PCDATA typed. (parse-able data type).
- **<!ELEMENT email:** It defines that the email element is #PCDATA typed. (parse-able data type).

- PCDATA – It is a text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.
- CDATA – It specify the character string data. It is text that will not be parsed by a parser. Tags inside the text will not be treated as markup and entities will not be expanded.

- This rule tells us that the element BOOKLIST consists of zero or more BOOK elements. The \* after BOOK indicates how many BOOK elements can appear inside the BOOKLIST element.  
\* denotes zero or more occurrences.
  - \*+ denotes one or more occurrences,
  - ? Denotes zero and one occurrence.
  - + to BOOKLIST has at least one book
- 
- E.g <! ELEMENT BOOKLIST (BOOK)+>

# Namespace

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

# Namespaces

- XML Namespaces is primary purpose to a distinguish between duplicate element and attribute names.
- XML data has to be exchanged between several application.
- Same tag name may have different meaning in different application. So It's create confusion on exchanging documents.
- Specifying prefix name to an element or attribute names to avoid this confusion.

# XML Namespaces

- Elements and attributes name along with namespace have **exactly one colon**.
- **Colon before text prefix name and colon after text element or attribute name.**
- Each and every prefix name is **associated with one URI**.
- Prefix name associated with the **same URI** are in the **same namespace**.
- Full qualified name including **prefix, colon** is called the **XML qualified name**.

# Namespace declaration

- A Namespace is declared using reserved attributes. Such an attribute name must either be `xmlns` or begin with `xmlns:` shown as below –
- `<element xmlns: name = "URL">`

```
<r:student xmlns:r="http://www.way2tutorial.com/xml/">  
  <!--  
    Scope within this element.  
    prefix r is associated with the xml namespace  
  -->  
</r:student>
```

# Name Conflicts Example

- Following example XML data for storing student marks

```
<student>
  <result>
    <name>Opal Kole</name>
    <sgpa>8.1</sgpa>
    <cgpa>8.4</cgpa>
  </result>
  <cv>
    <name>Opal Kole</name>
    <cgpa>8.4</cgpa>
  </cv>
</student>
```

# xmlns attribute with XML namespaces

- Specify all XML namespaces within the root element. Specification given for the that element is valid for all element occurring within scope.
- The namespace declaration Syntax: `xmlns:prefix_name="URI"`.

```
<s:student>
  <r:result>
    <r:name>Opal Kole</r:name>
    <r:sgpa>8.1</r:sgpa>
    <r:cgpa>8.4</r:cgpa>
  </r:result>
  <res:cv>
    <res:name>Opal Kole</res:name>
    <res:cgpa>8.4</res:cgpa>
  </res:cv>
</s:student>
```

# xmlns attribute with XML namespaces

- Specify all XML namespaces within the root element. Specification given for the that element is valid for all element occurring within scope.

```
<student xmlns:r="http://www.way2tutorial.com/some_url1"
         xmlns:res="http://www.way2tutorial.com/some_url2">

    <r:result>
        <r:name>Opal Kole</r:name>
        <r:sgpa>8.1</r:sgpa>
        <r:cgpa>8.4</r:cgpa>
    </r:result>

    <res:cv>
        <res:name>Opal Kole</res:name>
        <res:cgpa>8.4</res:cgpa>
    </res:cv>

</student>
```

# Namespace declaration

```
<?xml version="1.0" encoding="UTF-8"?>
<persondata
xmlns:a = "http://www.abc.com/anything"
xmlns:b = "http://www.xyz.com/anything">
    <user id="1">
        <a:fname> Raj </a:fname>
        <a:lname> Sharma </a:lname>
        <a:email> rajsharma@gmail.com </a:email>
    </user>
    <user id="2" >
        <b:fname> Raj </b:fname>
        <b:lname> Sharma </b:lname>
        <b:email> rajsharma@gmail.com </b:email>
    </user>
</persondata>
```

## e.g. of namespace

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

# XML Schema

## 2. XML Schema

- XML schema is a language used to describe XML documents constraints.
- The structure of an XML document is specified with an XML schema. It is like DTD but offers more XML structure control.
- The Schema does the same thing that a DTD can do. It also defines the structure of the XML document but unlike DTD it is an XML file, in addition to that Schema supports data types and namespaces.

# XML Schema – XSD (XML Schema Definition)



- XML schema is an alternative to DTD. An XML document is considered “well formed” and “valid” if it is successfully validated against XML Schema. The extension of Schema file is .xsd.
- XML Schema is also used to check whether the given XML document is “well formed” and “valid”.

# Checking Validation

- If an XML document includes the correct syntax, it is considered "well-formed." A well-formed, valid XML document is one validated against schema.
- Visit <http://www.xmlvalidation.com> to validate the XML file against schema or DTD.

## E.g. of XML Schema

```
<?xml version="1.0"?>
<employees xmlns="http://www.empinfo.com
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://www.empinfo.com employee.xsd">
  <employee>
    <empNo>101</empNo>
    <name>Rajesh</name>
    <salary>12000</salary>
  </employee>
  <employee>
    <empNo>102</empNo>
    <name>Suresh</name>
    <salary>15000</salary>
  </employee>
  <employee>
    <empNo>103</empNo>
    <name>Naresh</name>
    <salary>7890</salary>
  </employee>
</employees>
```

# XSD example

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.empinfo.com
xmlns="http://www.w3.org/2001/XMLSchema
elementFormDefault="qualified">
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

# Description of XML Schema

- <xs:element name="employee"> :It determines employee name of element.
- <xs:complexType> : This determines that the 'employee' aspect is type complex.
- <xs:sequence> :This determines that the form of complex is a sequence of elements.
- <xs:element name="firstname" type="xs:string"/> :This determines the string / text type is the element 'firstname.'
- <xs:element name="lastname" type="xs:string"/> :This determines the string / text sort is the element 'lastname.'
- <xs:element name="email" type="xs:string"/> :This determines the string / text sort is the element 'email.'

# XML Schema Data Types

1. **simpleType** – A simpleType element can contain text, they do not contain other elements. In the above example, the elements to, from, subject and message are simpleType element.
  
2. **complexType** – A complexType element can contain attributes, other elements, and text. In the above example, the element beginnersbook is of type complexType because it contains other elements.

- **Advantages of using XML Schema over DTD**

1. Schema uses XML as language so you don't have to learn new syntax.
2. XML schema supports data types and namespaces.
3. You can use XML parser to parse the XML schema as well.
4. Just like XML, the XML schema is extensible which means you can reuse the schema in other schema, as well as you can reference more than one schemas in a single XML document.

# DTD vs XSD

- DTD (Document Type Definition) and XSD (XML Schema Definition) vary from each other in several respects. In brief, DTD has less control over the structure of XML whereas XSD (XML schema) offers greater control.

## **DTD**

DTD stands for Document Type Definition.

DTDs are derived from SGML syntax.

DTD doesn't support datatypes.

DTD doesn't support namespace.

DTD doesn't define order for child elements.

DTD is not extensible.

DTD is not simple to learn.

DTD provides less control on XML structure.

## **XSD**

XSD stands for XML Schema Definition.

XSDs are written in XML.

XSD supports datatypes for elements and attributes.

XSD supports namespace.

XSD defines order for child elements.

XSD is extensible.

XSD is simple to learn because you don't need to learn new language.

XSD provides more control on XML structure.



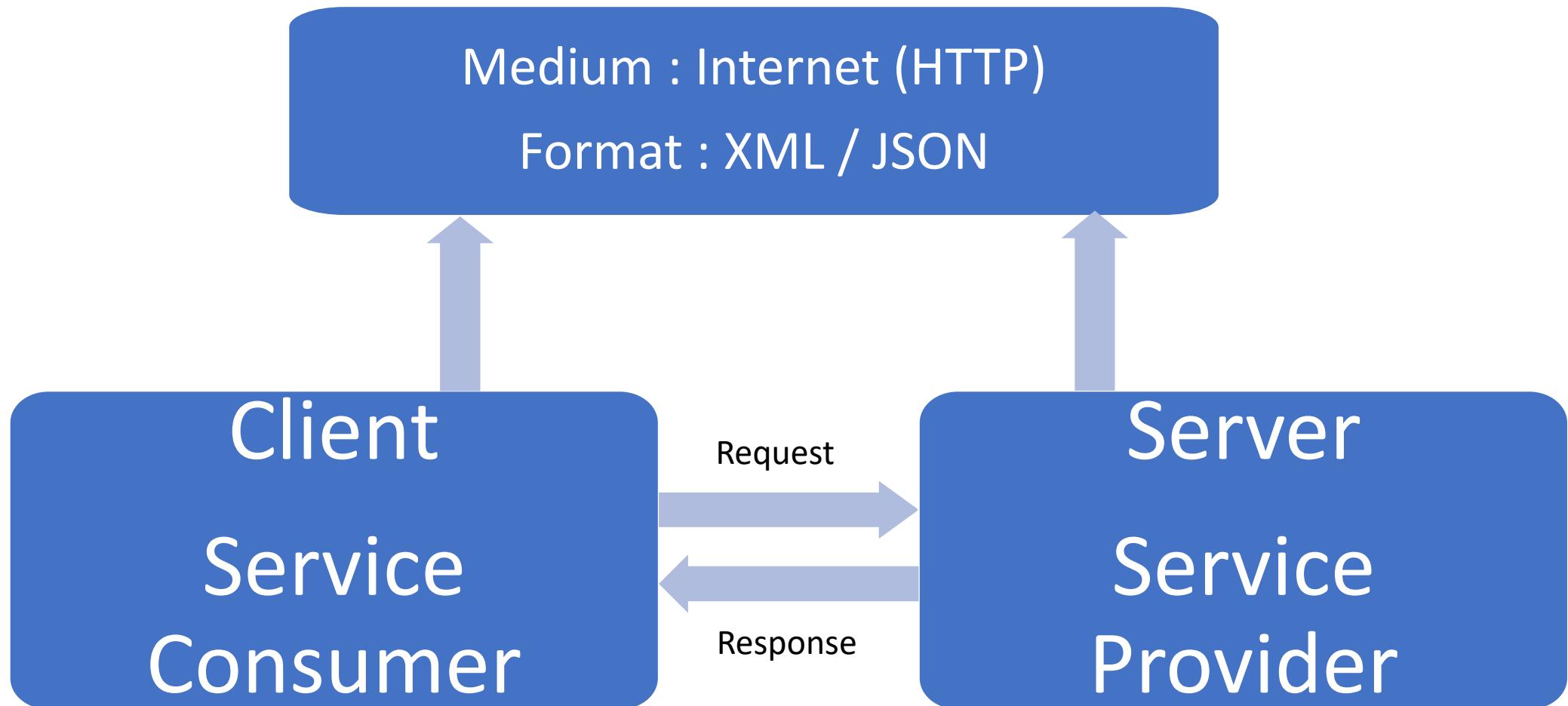
# Overview of XML Questions

1. Explain the structure of XML document.?
2. Write a short note on components of XML?
3. Explain XSD with suitable example.?

# Chapter 4

# SOAP, WSDL and JAX-WS

# Web Service



# Two types of Web Service

Web Services are implemented into two types

## SOAP (Simple Object Access Protocol)

- is an XML based protocol which allows communications between different applications through only XML messages. It doesn't allow other data format except XML. Java provides JAX-WS API for SOAP web services.

## RESTful Web Service

- REST (REpresentational State Transfer) is an architecture style, not a protocol. It allows communication between client and server via standardized protocols. Java provides JAX-RS API for RESTful web services. It allows other data formats like HTML, XML, JSON.

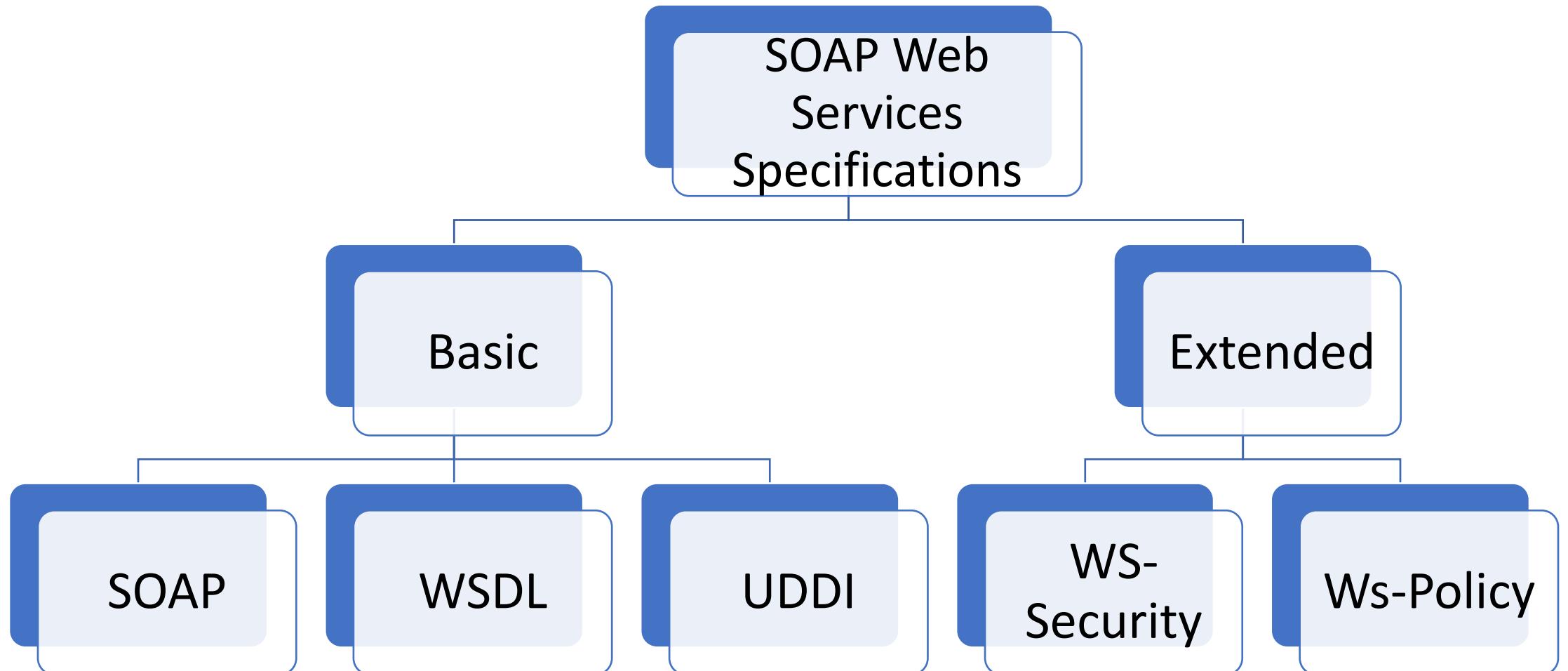
- How web services work and how the communication happens ?
- Web Services are services are available over web
- They has to be Service Provider and
- They has to Service Consumer.
- They interact with each other and to enable this interactions there are two major things are require.
- Medium
- Format



# SOAP Web Services

- A Web Services that complies to the **SOAP Web Services specifications** is a SOAP Web Service.

# SOAP Web Services Specifications

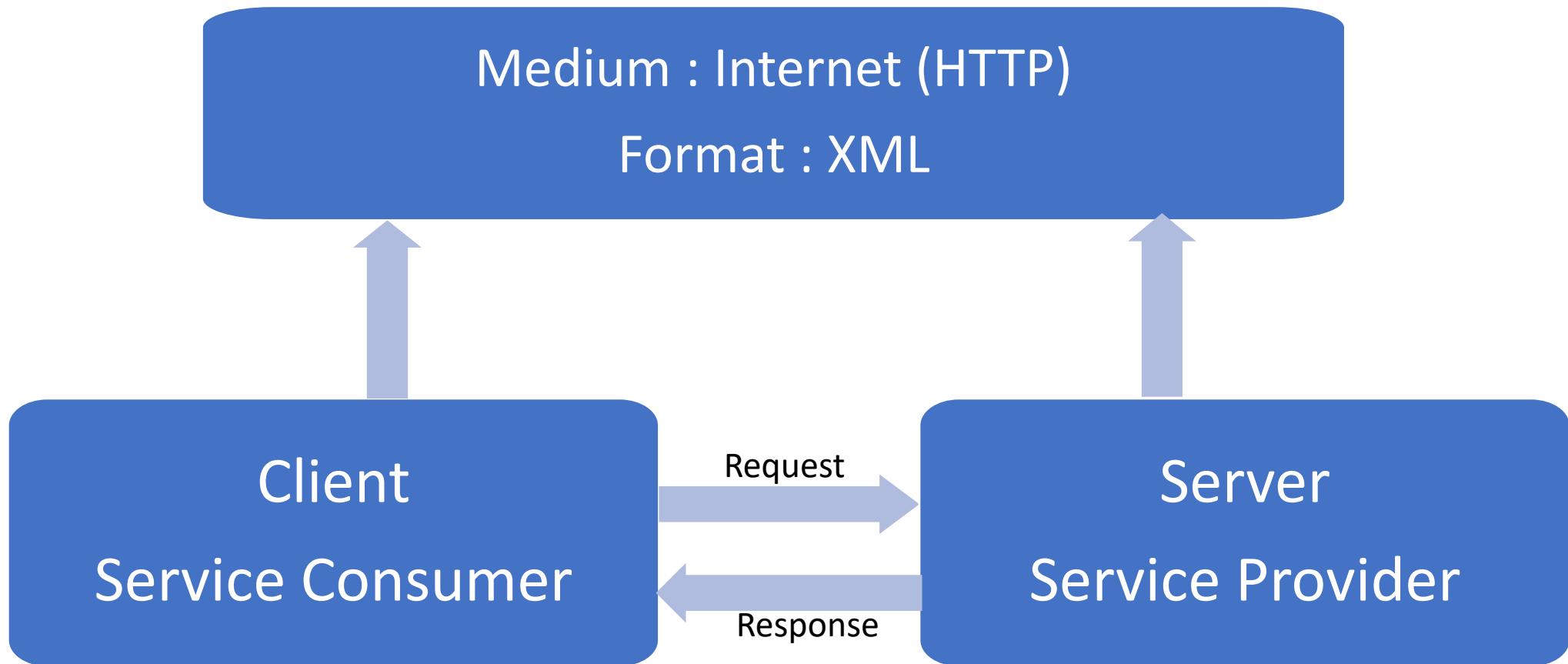


# SOAP Web Services Specifications (SOAP)



- SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.
- SOAP is a W3C recommendation for communication between two applications.
- It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.
- A SOAP endpoint is nothing but HTTP-based URL that used to identify the target for method invocation.

# Web Service



# Features of SOAP

- It is basically a communication protocol introduce to provide communication over Internet.
- It can extend HTTP for XML messaging.
- It facilities data transfer for the web Services.
- It can used to exchange complete document
- It can be utilize to a call a remote procedure.
- It is language and platform independent.

# Advantages of SOAP

- **Simplicity:** XML is properly structured and easy to translate or parse. SOAP is based on XML that's way it is simple to implement.
- **Portability :** Xml parse is present on every machine and because of which soap is portable.
- **Firewall-friendly:** SOAP can easily pass through firewall. Also HTTP is widely used delivery mechanism.

# Advantages of SOAP

- **Use of open standards:** XML is open –source . W3c recommendation, used to carry and transfer data over network. SOAP uses XML to format the data, which makes it easily exchange between devices over internet.
- **Interoperability :** SOAP is implemented on open, rather than vendor-specific technologies, it facilities true distributed interoperability and loosely couples application by mean of XML and HTTP.
- **Universal acceptance :** From all message delivery mechanism, SOAP is the most widely accepted standard.

# Disadvantages of SOAP

- **Statelessness** : SOAP clients cannot maintain any stateful references to remove objects.
- **Slow** : XML format need to be parsed. SOAP uses XML which makes it slower.
- **Serialization** : In SOAP, serialization happened with value not by reference
- **Security**: Security policies are not properly mentioned in the specification of SOAP.
- **WSDL Dependency** : It strongly depends on WSDL and it does not have any scandalized mechanism for dynamic discovery of the services.

# SOAP Message

- All information / message exchange happens over a common format



- XML message has a defined structure :

• **SOAP MESSAGE**

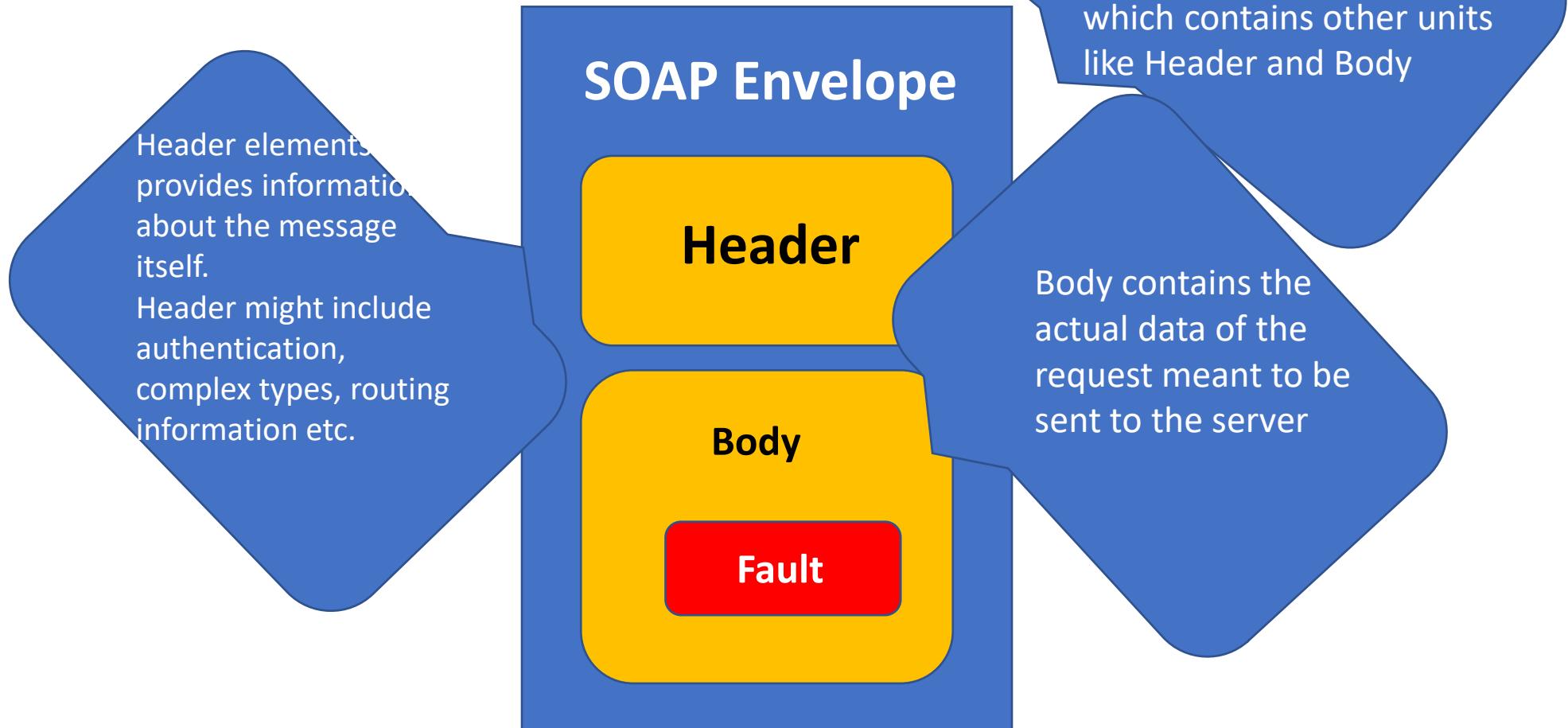
# Structure of a SOAP Message

- A SOAP message is an ordinary XML document containing the following elements:
- **Envelope** : It specifies the start and the end of the message. It's an overall wrapper of SOAP message. This is a mandatory element.
- **Header** : It consists of any optional attributes of the message used for the processing purpose of the message. It is used to specify application specific information.

# Structure of a SOAP Message

- **Body:** It contain the original message to be transfer to the web service. It is a mandatory element.
- **Fault:** It's an optional. This element gives information about errors that occurs while processing the message.

# SOAP Message Structure



# 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>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:c="http://www.acmeOrders.com/OrderService"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <c:OrderMessage>
            <localElement>
                <FirstName>John</FirstName>
                <LastName>Smith</LastName>
                <Street>High Street</Street>
                <City>London</City>
                <ZipCode>W1A1AA</ZipCode>
                <PartNumber>ABC1234</PartNumber>
                <Quantity>1</Quantity>
            </localElement>
        </c:OrderMessage>
    </soap:Body>
</soap:Envelope>
```

# The SOAP Communication Model



## SOAP Communication Model

1. RPC-Style SOAP Service

2. Document(Message)-Style SOAP Services

# 1. RPC-Style SOAP Service

- In RPC style SOAP services to generate XML structure, it uses method names and parameters. These SOAP parameters consists of request message and return value in response message.
- In RPC style SOAP message is sent as many elements. This style of SOAP message is tightly couple. In RPC style, SOAP message carries the names of the operations.

# 1. RPC-Style SOAP Service

- As per developer points of view, RPC is simpler style of SOPA. As there is no type section, the automatically generated WSDL is comparatively simple and short. But the automatically generated WSDL is difficult to be validated against schema.

# 1. RPC-Style SOAP Service

```
<message name="getHelloWorldAsString">
<part name="arg0" type="xsd:string"/>
</message>
<message name="getHelloWorldAsStringResponse">
<part name="return" type="xsd:string"/>
</message>
```

# Document Style

- Document style web services **can be validated against predefined schema.**
- In document style, SOAP message is **sent as a single document.**
- Document style message is **loosely coupled.**
- In Document style, SOAP message **loses the operation name.**
- In Document style, parameters are sent in **XML format.**

# Document Style

```
<types>
<xsd:schema>
<xsd:import namespace="http://abc.com/" schemaLocation="http://lo
calhost:7779/ws/hello?xsd=1"/>
</xsd:schema>
</types>
```

# Document Style

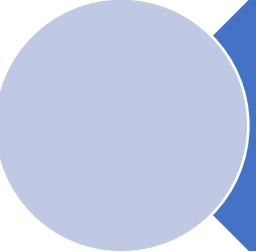
RPC Style	Document Style
RPC style web services use method name and parameters to generate XML structure.	
In RPC style, the generated WSDL is <b>difficult to be validated against schema</b> .	Document style web services can be <b>validated against predefined schema</b> .
In RPC style, SOAP message is sent as <b>many elements</b> .	In document style, SOAP message is sent as a <b>single document</b> .
RPC style message is <b>tightly coupled</b> .	Document style message is <b>loosely coupled</b> .

# Document Style

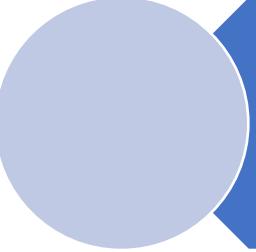
In RPC style, SOAP message keeps the operation name.	In Document style, SOAP message loses the operation name.
The parameters are sent as <b>discrete values</b> .	The parameters are sent in <b>XML format</b> .
It defines name and type attribute for message part.	It defines name and element attribute for message part.
For soap:body, it defines use and namespace attributes.	For soap:body, it defines use attribute only.

# Web Service Components

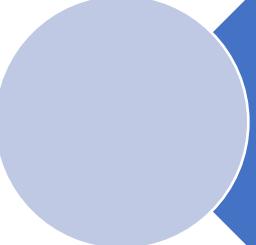
There are three major web service components.



**SOAP**

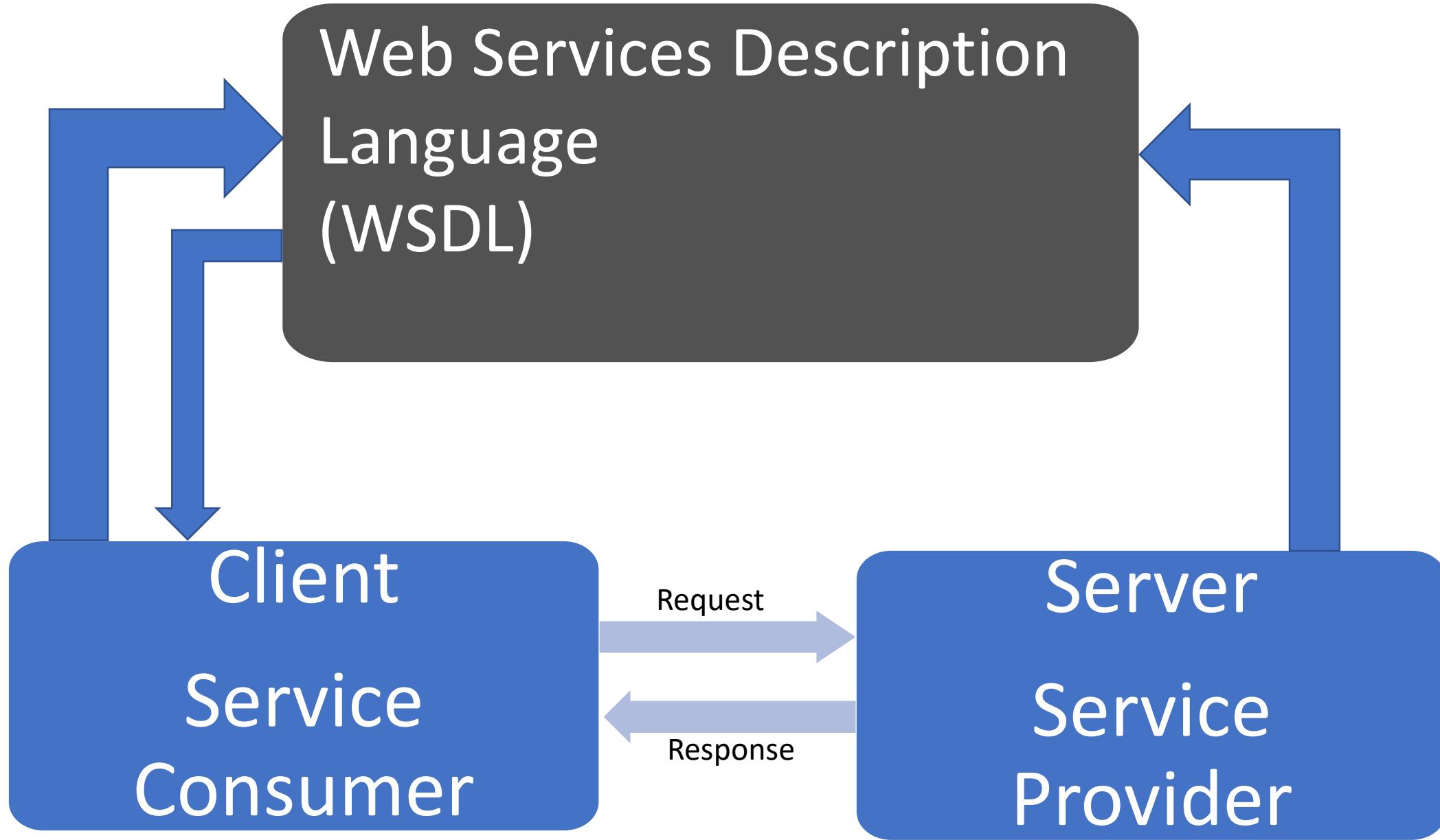


**WSDL**



**UDDI**

# Web Services Description Language (WSDL)





# (WSDL)

- Wsdl is a interface for web Services.
- Service provider create this interface for his web service and the service consumer can get his WSDL and the use web services and all requests of the web services.

- There are two ways a service consumer can get hold of these WSDL document.
- 1. One way is if the Service Consumer and Service Provider already know each other, then the Service Provider can hand over directly these WSDL document or WSDL URL to the client and the client can use that web service.

- 2. When the Service Consumer and Service Provider does not know each other, how can Service Consumer get hold of WSDL document.

- For that All we have to do is to publish our service (WSDL) on an UDDI Registry and then search the web for the required service. This Registry is called Universal Description, Discovery, and Integration (UDDI).
- Is an XML based standard for publishing and finding web Services.

## Universal Description, Discovery, and Integration (UDDI)

# WSDL Elements/WSDL Components



- WSDL breaks down web offerings into three particular, identifiable factors that may be mixed or reused once described.
- Types
- Operations
- Binding

# WSDL Elements

**<definitions>**

**<types>**

definition of types.....

**</types>**

**<message>**

definition of a message....

**</message>**

**<portType>**

**<operation>**

definition of a operation.....

**</operation>**

**</portType>**

**<binding>**

definition of a binding....

**</binding>**

**<service>**

definition of a service....

**</service>**

**</definitions>**

# WSDL Elements

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/definitions"
    xmlns:tns="http://example.com/stockquote/definitions"
    xmlns:xsd1="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/schemas"
          location="http://example.com/stockquote/stockquote.xsd"/>

<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
</definitions>
```

# WSDL Components

- **Definition:** it is the basis detail of all WSDL files. It defines the call of the web service, declares more than one namespaces used during the remainder of the record, and includes all the carrier elements described here.
- **Data types:** The information types to be used inside the messages are within the form of XML schemas.
- **Message:** it is an summary definition of the information, in the form of a message provided both as an entire file or as arguments to be mapped to a technique invocation.

# WSDL Components



- **Operation:** it is the summary definition of the operation for a message, including naming a way, message queue, or business manner, to be able to receive and technique the message.
- **Port type :** it is an summary set of operations mapped to one or more end-points, defining the collection of operations for a binding; the collection of operations, as it is summary, can be mapped to a couple of transports through numerous bindings.
- **Binding:** it is the concrete protocol and information formats for the operations and messages described for a selected port type.

# WSDL Components

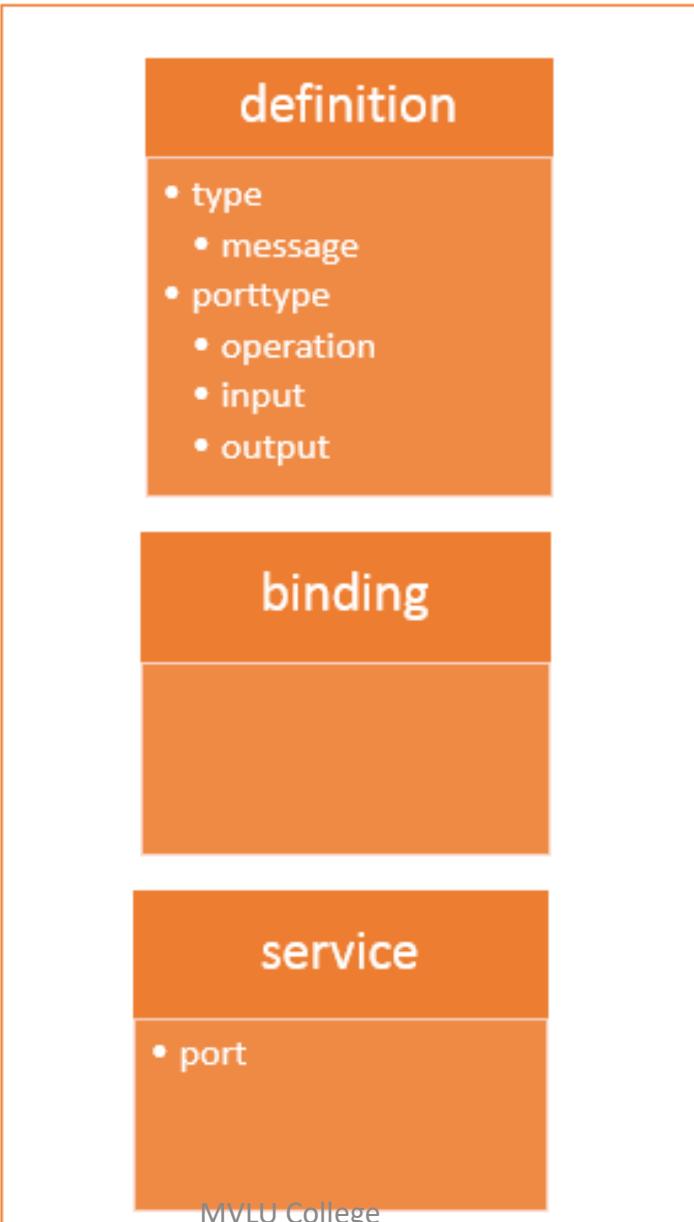
- **Port:** it is a combination of a binding and a network address, offering the target address of the service communication.
- **Service:** it is a set of associated end-points encompassing the provider definitions inside the file; the services map the binding to the port and include any extensibility definitions.

# WSDL Components



- Similarly to those primary elements, the WSDL specification also defines the following application elements:
- **Documentation:** This detail is used to offer human-readable documentation and may be included inside another WSDL element.
- **Import:** This element is used to import other WSDL documents or XML Schemas.

# WSDL Components structure





# WSDL Components

- E.g.

# WSDL <definitions> Element



- The **<definitions>** It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the document, and contains all the other service elements ( type , message, parttype, operation , input , output)

# WSDL <types> Element

- The type element defines all the data types used between the server and the client.
- To define data types WSDL uses the W3C XML Schema Specification as its default choice.
- Type element is not required if the service uses only simple XML schema types like integers and strings.
- To reuse the type with multiple web services, WSDL allows to define types in a separate elements.

```
<types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TradePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
        </element>
        <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
        </element>
    </schema>
</types>
```

# WSDL <message> Element

- The **<message>** detail describes the information being exchanged among the web service providers and the service consumer.
- Each web provider has messages: input and output.
- The input describes the parameters for the web service and the output describes the return information from the web provider.
- Each message includes zero or greater **<part>** parameters, one for every parameter of the web provider function.
- Every **<part>** parameter associates with a concrete type defined in the **<types>** container element.

# WSDL <message> Element e.g.

```
<message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
</message>
```

# WSDL <message> Element

```
<wsdl:message name="GetBookRequest">
    <wsdl:part element="tns:GetBook" name="parameters" />
</wsdl:message>
<wsdl:message name="GetBookResponse">
    <wsdl:part element="tns:GetBookResponse" name="parameters" />
</wsdl:message>
<wsdl:message name="AddBookRequest">
    <wsdl:part name="parameters" element="tns:AddBook"></wsdl:part>
</wsdl:message>
<wsdl:message name="AddBookResponse">
    <wsdl:part name="parameters" element="tns:AddBookResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="GetAllBooksRequest">
    <wsdl:part name="parameters" element="tns:GetAllBooks"></wsdl:part>
</wsdl:message>
<wsdl:message name="GetAllBooksResponse">
    <wsdl:part name="parameters" element="tns:GetAllBooksResponse"></wsdl:part>
</wsdl:message>
```

# WSDL <portType> Element

- <**portType**> element provides the listing of all the operations performed by the web services which are supported on a specific end point
- A portType can outline multiple operations.
- As an <portType> example, a can integrate one request and one reaction message right into a single request/response operation. this is most generally utilized in soap offerings.

# WSDL <portType> Element

```
<portType name="Hello_PortType">
    <operation name="sayHello">
        <input message="tns:SayHelloRequest"/>
        <output message="tns:SayHelloResponse"/>
    </operation>
</portType>
```

# WSDL <portType> Element

```
<wsdl:portType name="BookService">
    <wsdl:operation name="GetBook">
        <wsdl:input message="tns:GetBookRequest" />
        <wsdl:output message="tns:GetBookResponse" />
    </wsdl:operation>
    <wsdl:operation name="AddBook">
        <wsdl:input message="tns:AddBookRequest"></wsdl:input>
        <wsdl:output message="tns:AddBookResponse"></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetAllBooks">
        <wsdl:input message="tns:GetAllBooksRequest"></wsdl:input>
        <wsdl:output message="tns:GetAllBooksResponse"></wsdl:output>
    </wsdl:operation>
</wsdl:portType>
```

# WSDL <operation> Element

- This element describes the information about method used in web service.
- Operation element provides the name of method and the process used in the method.
- This element basically describes the actions supported by the web service.

# WSDL <binding> Element

Binding describes the invocation of a Web Service. Binding represents the concrete protocols and data format specifications used in operations and messages discussed earlier.

- The binding element defines exactly how each operation will take place over the network.
- The **<binding>** tag is used to bind the operation to the particular port type.
- The bindings may be made available through multiple transports such as HTTP GET, HTTP post, or soap.
- You may specify more than one bindings for a single portType.
- The binding detail has two attributes : name and type attribute.

# WSDL <binding> Element

- The binding detail has two attributes : name and type attribute.

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
```

# WSDL <port> Element



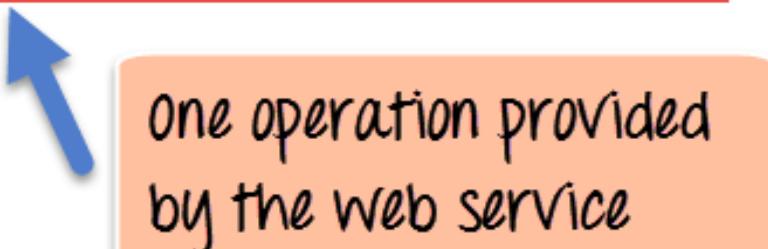
- A <**port**> combines the protocol used and the network address. Port is basically a combination of bindings and the URL used for the web service.
- It is a single endpoint defines as a combination of a binding and network address.
- The port detail has two attributes: name and binding .

# WSDL <ports> Element

```
<service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address
            location="http://www.examples.com/SayHello/">
    </port>
</service>
```

```
<portType name="Tutorial_PortType">  
  
    <operation name="Tutorial">  
        <input message="tns:TutorialRequest"/>  
        <output message="tns:TutorialResponse"/>  
    </operation>
```

```
</portType>
```



# WSDL <service> Element



- The **<service>** Service elements represents the collection of all end points
- Service element also provide documentation about the web service.
- Web provider clients can learn the following from the carrier element:
  - Where to get entry to the service,
  - Through which port to access the web provider, and
  - How the communication messages are defined.

```
<wsdl:service name="BookService">
    <wsdl:port binding="tns:BookServiceSOAP" name="BookServiceSOAP">
        <soap:address location="http://www.example.org/BookService" />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

# WSDL <service> Element



```
<service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address
            location="http://www.examples.com/SayHello/">
    </port>
</service>
```

# For example calculator web Services.

1. What are the things we can do with application ?
  - [addition] → **Operation**
2. What are inputs we should pass ?
  - [two inputs] as integer → **Input Message**  
Part 1  
Part 2
3. What is the output we will get ?
  - [integer sum of two numbers] → **Output Message**  
Part 1
4. Where should we send the request?
  - [endpoint where request to be send ] where your service running. → **binding**

```
<message name="getSumRequest">
  <part name="number1" type="xs:int"/>
  <part name="number2" type="xs:int"/>
</message>
```

```
<message name="getSumResponse">
  <part name="sum" type="xs:int"/>
</message>
```

```
<portType name="calculator">
  <operation name="getSum">
    <input message="getSumRequest"/>
    <output message="getSumResponse"/>
  </operation>
</portType>
```

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

**Input Message**

**Output Message**

**Operation provided  
by WSDL**

**Name and type of  
binding**

```
<service name = "Calculator_Service">
    <documentation>WSDL File for Calculator webservice</documentation>
    <port binding = "calculator" name = "calculator_Port">
        <soap:address location = "http://www.examples.com/calculator/" />
    </port>
</service>
```

**service**



# WSDL message exchange patterns

- WSDL interfaces provide support to four types of operations by combining the input and output messages.

One-way operation

Request/Response operation

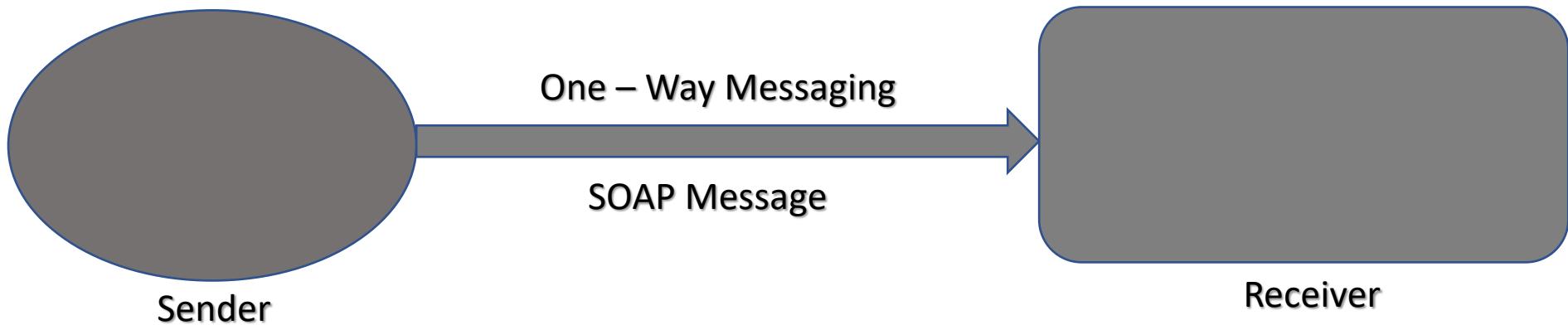
Notification operation

Solicit/Response operation

# One-way operation

- In this type of operation the service endpoint can receives a message, but cannot send a response. E.g. submission of an order to a purchasing system. Once the order is sent , there is no need to receive immediate response.
- This message exchange pattern is basically represents as asynchronous messaging. A one-way message specifies only input message.
- It does not require any output message and fault.

# One-way operation



# One-way operation



- The service receives a message. The operation therefore has a single input element.

```
<wsdl:definitions ... >
  <wsdl:portType ... > *
    <wsdl:operation name="nmtoken">
      <wsdl:input name="nmtoken"? message="qname"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

# One-way operation example



- **Weather Summary Service Description**

The Weather Summary service represents a centralized provider of weather condition data for some number of zip code locations in some area.

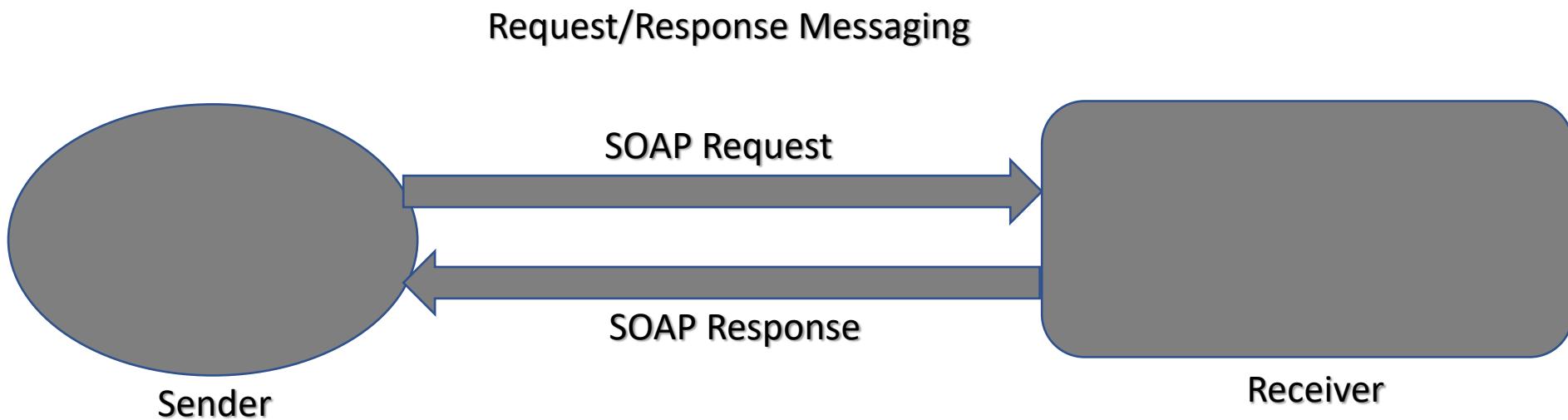
Participating zip code locations send updated weather data to the service when local conditions change, using the one-way operation.

# Request/Response operation



- In this type of operation the service end point receives a message and also return a message in response. If an `<operation>` element is declared with a single `<input>` element and a single `<output>` element. It shows a request/response operation.
- By listing the `<input>` tag first, the `<operation>` indicates that the Web Service receives a message which is sent by the client and then the `<output>` tag second shows that the Web Service must have to respond to the message.

# Request/Response Messaging



# Request/Response Messaging



- The service receives a message and sends a response. The operation therefore has one input element, followed by one output element. To encapsulate errors, an optional fault element can also be specified.

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

# Request/Response Messaging example



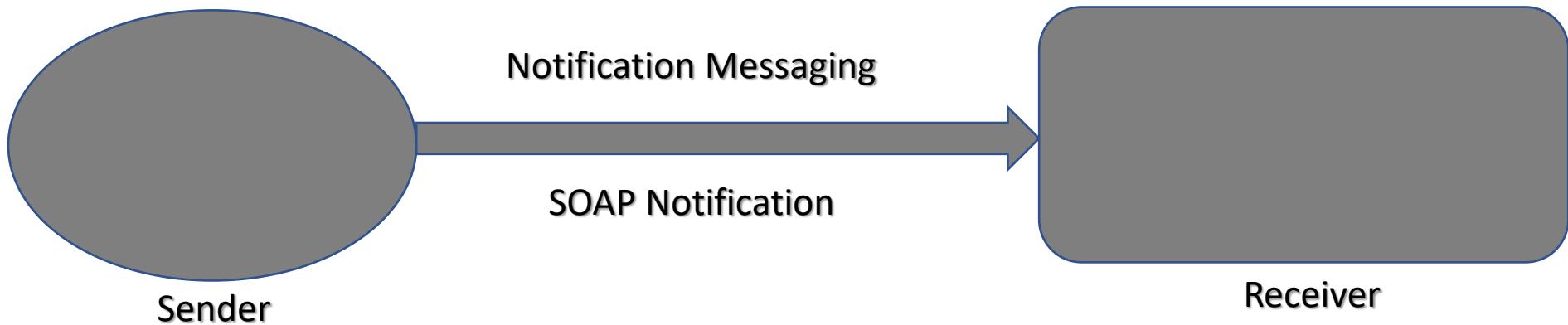
- **Weather Summary Service Description**
- The Weather Summary service represents a centralized provider of weather condition data for some number of zip code locations in some area.
- A client, or individual, requests weather data based on a zip code, and the service provides it, through the request-response operation.

# Notification Messaging



- In this type of operation the service endpoint sends a message to a client, but it does not expect to receive any response.
- This type of messaging is utilized by services that wants to notify clients of events. Notification is provided when a `<portType>` elements contains an `<output>` tag, but no `<input>` tag or message definitions.
- e.g. Client has registered with the Web service to receive messages or notifications about an event. No response is needed in this situation.

# Notification Messaging



# Notification Messaging



- The service sends a message. The operation therefore has a single output element.

```
<wsdl:definitions ... >
    <wsdl:portType ... > *
        <wsdl:operation name="nmtoken">
            <wsdl:output name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

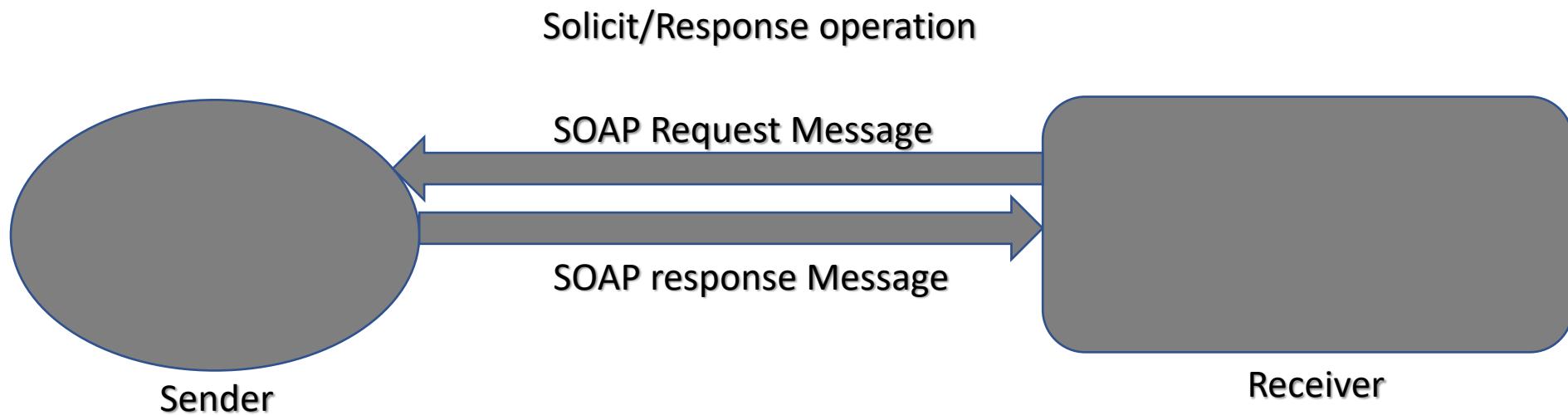
- **Weather Summary Service Description**
- The Weather Summary service represents a centralized provider of weather condition data for some number of zip code locations in some area.
- Clients can subscribe to receive notification of changed conditions through the notification operation.
-

# Solicit/Response operation



- In this type of operation the service endpoint sends a message to a client, but it does not expect to receive any response.
- This is the reverse operation of the request/response operation because the service endpoint is imitating the operation means soliciting the client, rather than responding to a request.
- Client is expected to respond to the Web Service. In this type of messaging the `<portType>` element first declares an `<output>` tag and then a `<input>` tag or message definition.
- E.g. a service that sends out order status to a client and also receives back a receipt.

# Solicit/Response operation



# Solicit/Response operation



- The service sends a message and receives a response. The operation therefore has one output element, followed by one input element. To encapsulate errors, an optional fault element can also be specified.

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

- **Weather Summary Service Description**
- The Weather Summary service represents a centralized provider of weather condition data for some number of zip code locations in some area.
- Finally, the service periodically polls the client to confirm that the client wishes to continue receiving notifications, through the solicit-response operation.

# JAX-WS

- JAX-WS stands for Java API for XML web base services. JAX-WS is one in all a collection of Java technologies utilize to develop web services . Different technologies may additionally return from the core web services group (like JAXB, WSIT, WS-Security, JAX-RPC , WS-Management and etc.

- JAX-WS represents RPC or message using XML-based protocols such as SOAP, but complexity of SOAP is hidden behind a Java-based API.
- API can be used by developers to specify methods and then create classes to implement those methods. And all other communication details can be handled by JAX-WS API. Clients create local copy or proxy to represents a service, and then invoke methods on this proxy.

# Advantages of JAX-WS

- Simple interoperable services can be created easily.
- Developer is unaware of Protocol (it is hidden)
- Asynchronous Service are available through java threading
- Error faults can be send back to client

# SOAP, WSDL and JAX-WS Questions

1. What is SOAP? Explain its features ?
2. Explain the structure of SOAP.
3. What are different types of SOAP Communication Model?
4. Explain advantages and disadvantages of SOAP.
5. Write a short note on WSDL.
6. Explain the structure of WSDL.
7. Explain with neat diagram the WSDL message exchange pattern.
8. Explain advantages and disadvantages of JAX-WS.

# Chapter 5

# UDDI

# Service Registries



Service registry architecture that provides and supports enterprises to introduce a global, open, platform-independent framework for busses to:

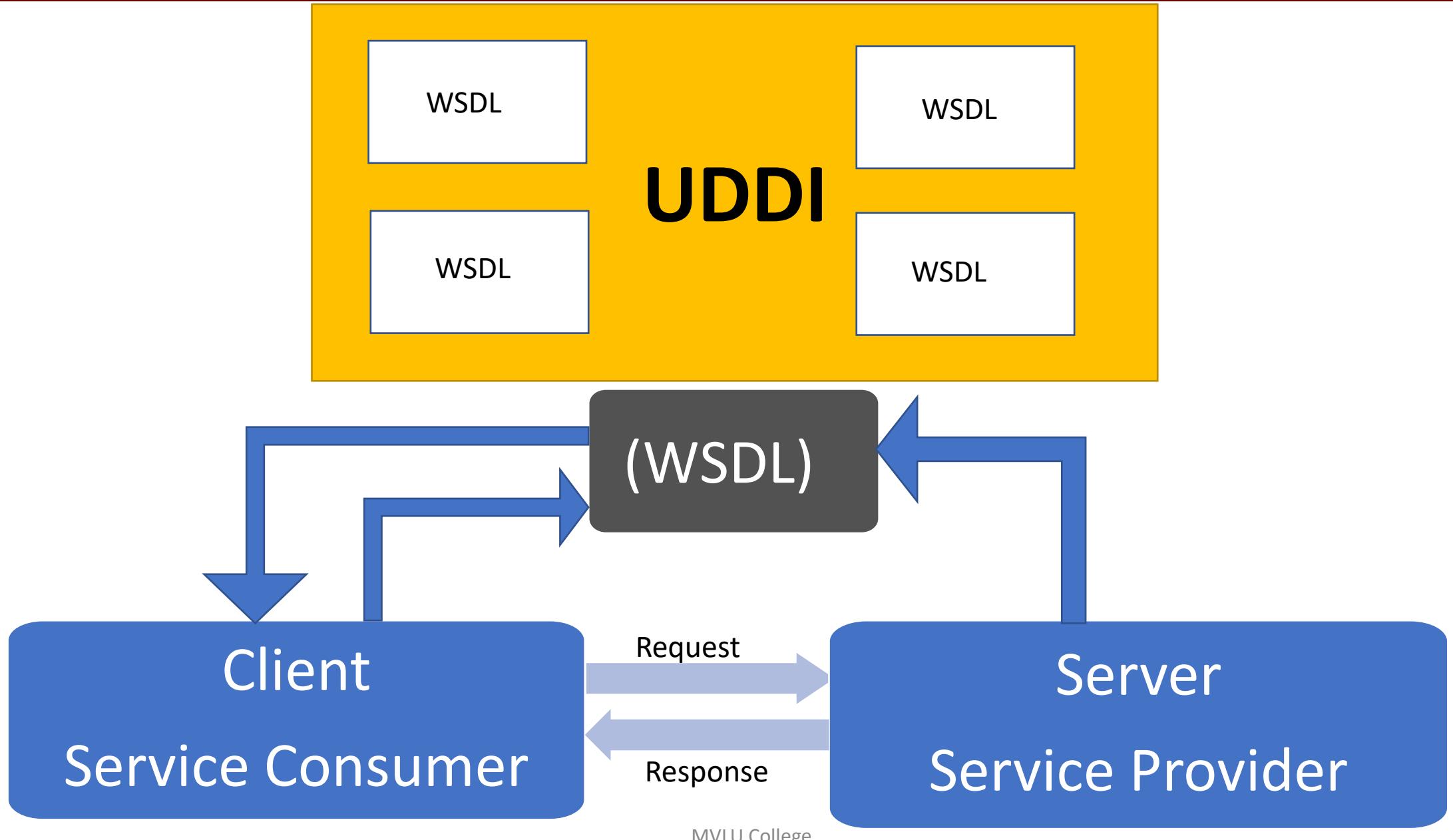
- Different organization discover each other,
- Define rules and guidelines, how they communicate via network and
- Share information in a global registry will help to improve the e-business.

# Service Discovery



Service discovery is an essential element of a Service-Oriented Architecture (SOA). Service discovery involves the steps of locating Web service providers and retrieving web service description that have been published previously.

# UDDI



# Universal Description, Discovery, and Integration



- 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.

# Three elements of UDDI

## Three elements of UDDI

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

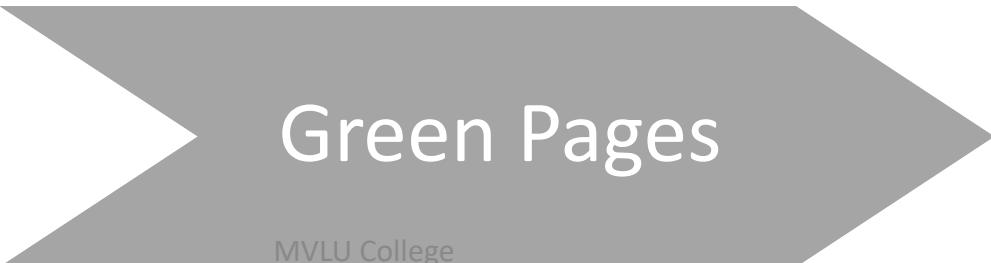
These three elements are –



White Pages



Yellow Pages



Green Pages

# White Pages

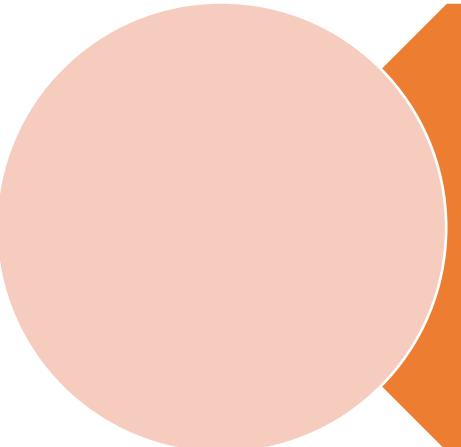


Basic information about the company and its business.

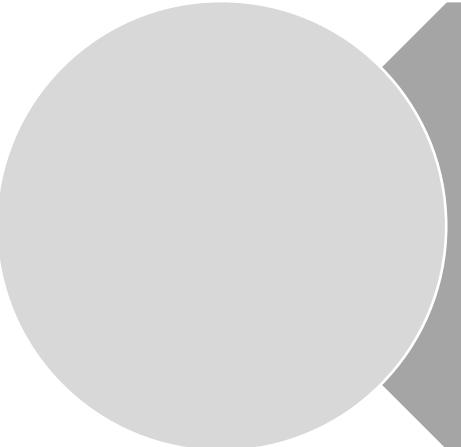
Basic contact information including business name, address, contact phone number, etc.

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.
-  Operating platform,
-  Purchasing methods,
-  Shipping and billing requirements

- The information that is stored in the UDDI registry allows applications and developers to determine *who* the business entity represents, *what* they do, *where* the services they provide can be found, and *how* they can be accessed.

# UDDI data structures

# UDDI data structures



UDDI includes an XML Schema that describes the following data structures

businessEntity

businessService

bindingTemplate

tModel

publisherAssertion

# UDDI data structures

- For Web services to be meaningful there is a need to provide information about them beyond the technical specifications of the service itself.

# UDDI – main characteristics

UDDI provides a mechanism to categorize businesses and services using taxonomies.

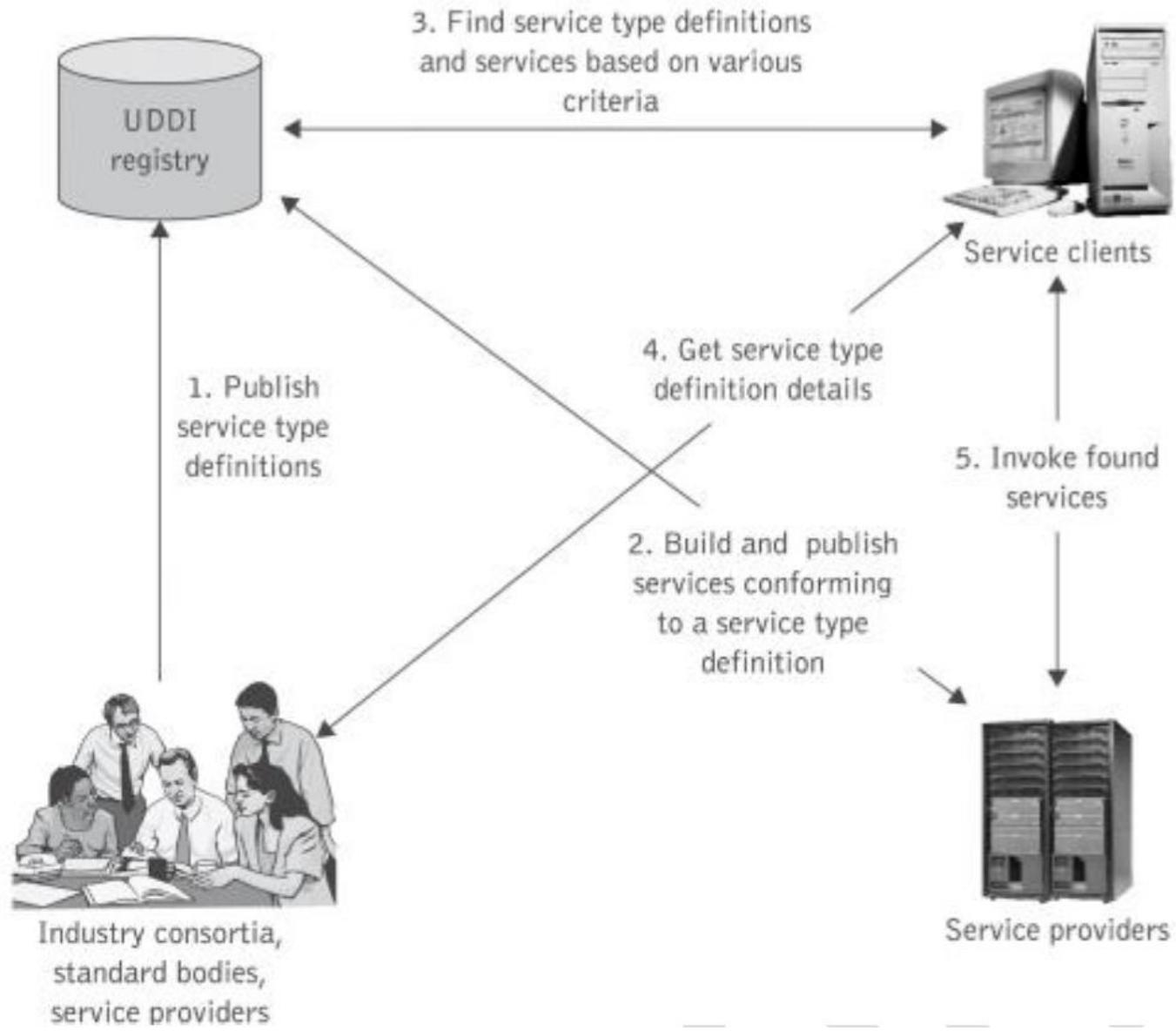
- Service providers can use a taxonomy to indicate that a service implements a specific domain standard, or that it provides services to a specific geographic area
- UDDI uses standard taxonomies so that information can be discovered on the basis of categorization.

## UDDI business registration

- an XML document used to describe a business entity and its Web services

# UDDI usage model

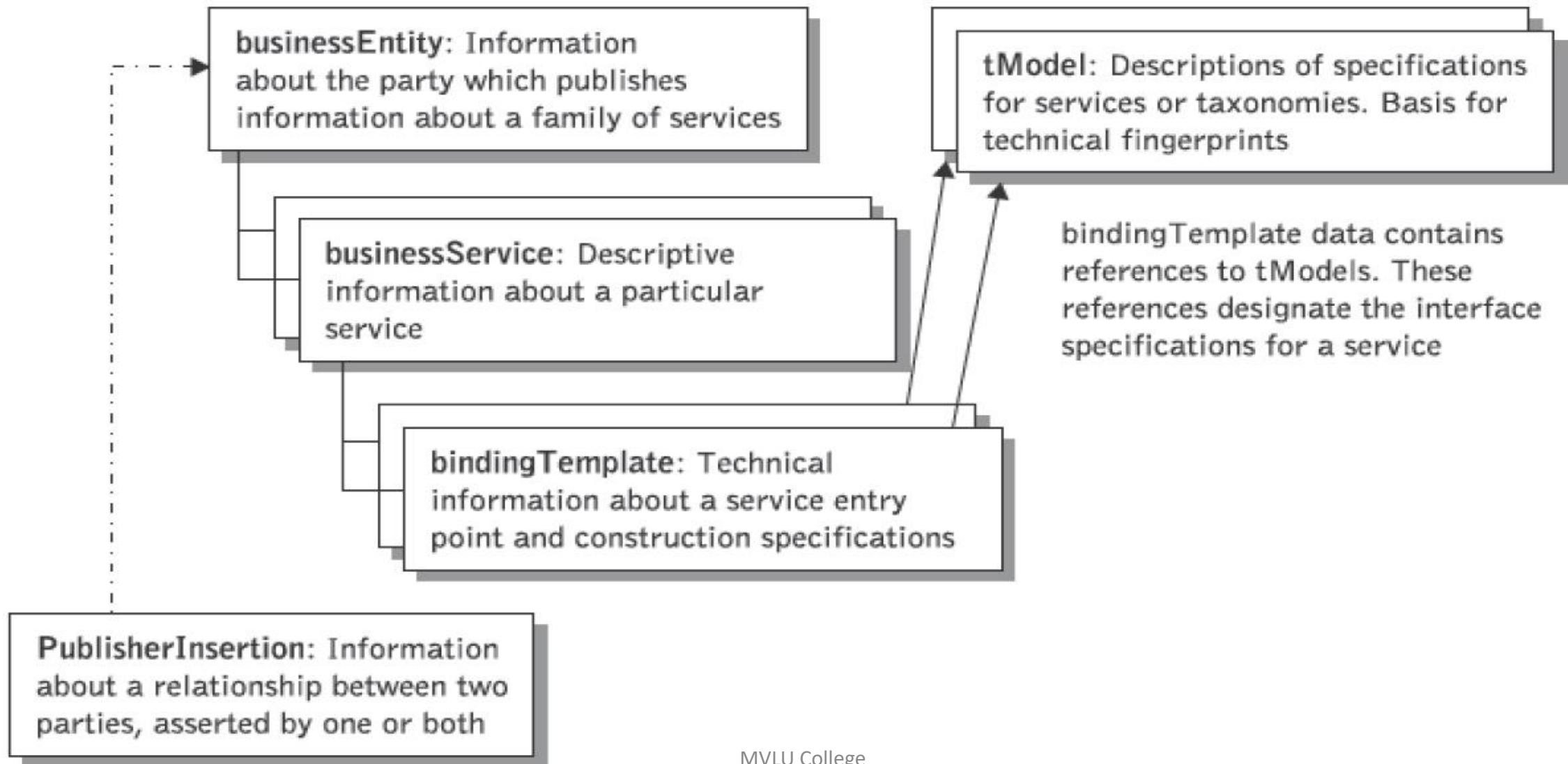
- An enterprise may set up multiple private UDDI registries in-house to support intranet and eBusiness operations.
- Public UDDI registries can be set up by customers and business partners of an enterprise.
- Services must be published in a public UDDI registry so that potential clients and service developers can discover them.



# Queries address the following

- Find Web services implementations that are based on a common abstract interface definition.
- Find Web service providers that are classified according to a known classification scheme or identifier system.
- Issue a search for services based on a general keyword.
- Determine the security and transport protocols supported by a given Web service.
- Cache the technical information about a Web service and then update that information at run-time.

# Overview of UDDI structure



- The UDDI XML schema defines four core types of information that provide the white/yellow/green page functions.
- These are: business entities, business services, binding templates; and information about specifications for services (technical or tModels)

- The UDDI XML schema specifies information about the business entity, e.g., a company, that offers the service (`<businessEntity>`), describes the services exposed by the business (`<businessService>`), and captures the binding information (`<bindingTemplate>`) required to use the service.

A UDDI **businessEntity** describes a business and its services:

```
<businessEntity businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64"
    operator="www.ibm.com/services/uddi"
    authorizedName="0100001QS1">
<discoveryURLs>
    <discoveryURL useType="businessEntity">http://www.ibm.com/services/uddi/uddiget?businessKey=BA744ED0-3AAF-11D5-80DC-002035229C64</discoveryURL>
</discoveryURLs>
<name>XMethods</name>
<description xml:lang="en">Web services resource site</description>
<contacts>
    <contact useType="Founder">
        <personName>Tony Hong</personName>
        <phone useType="Founder" />
        <email useType="Founder">thong@xmethods.net</email>
    </contact>
</contacts>
<businessServices>
    <businessService serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
        businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
        <name>XMethods Delayed Stock Quotes</name>
        <description xml:lang="en">20-minute delayed stock quotes</description>
        <bindingTemplates>
            <bindingTemplate bindingKey="d594a970-3e16-11d5-98bf-002035229c64"
                serviceKey="d5921160-3e16-11d5-98bf-002035229c64">
                <description xml:lang="en">SOAP binding for delayed stock quotes service</description>
                <accessPoint URLType="http">http://services.xmethods.net:80/soap</accessPoint>
                <tModelInstanceDetails>
                    <tModelInstanceInfo tModelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64" />
                </tModelInstanceDetails>
            </bindingTemplate>
        </bindingTemplates>
    </businessService>
</businessServices>
</businessEntity>
```

# *Service provider information*

- Partners and potential clients of an enterprise's services that need to be able to locate information about the services provided would normally have as a starting point a small set of facts about the service provider.

# 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.

- ***The business entity element and business key attribute.*** The core XML elements for supporting publishing and discovering information about a business – the UDDI Business Registration – are contained in an element named `<businessEntity>`.
- ***The discovery URLs element.*** This is an optional element and contains the URLs that point to alternate Web addressable (via HTTP GET) discovery documents.
- ***The name element.*** The name element contains the common name of the organization that a business entity represents. A `<businessEntity>` may contain more than one name.
- ***The description element.*** This element is a short narrative description for the business. A `<businessEntity>` can contain several descriptions, e.g., in different languages.

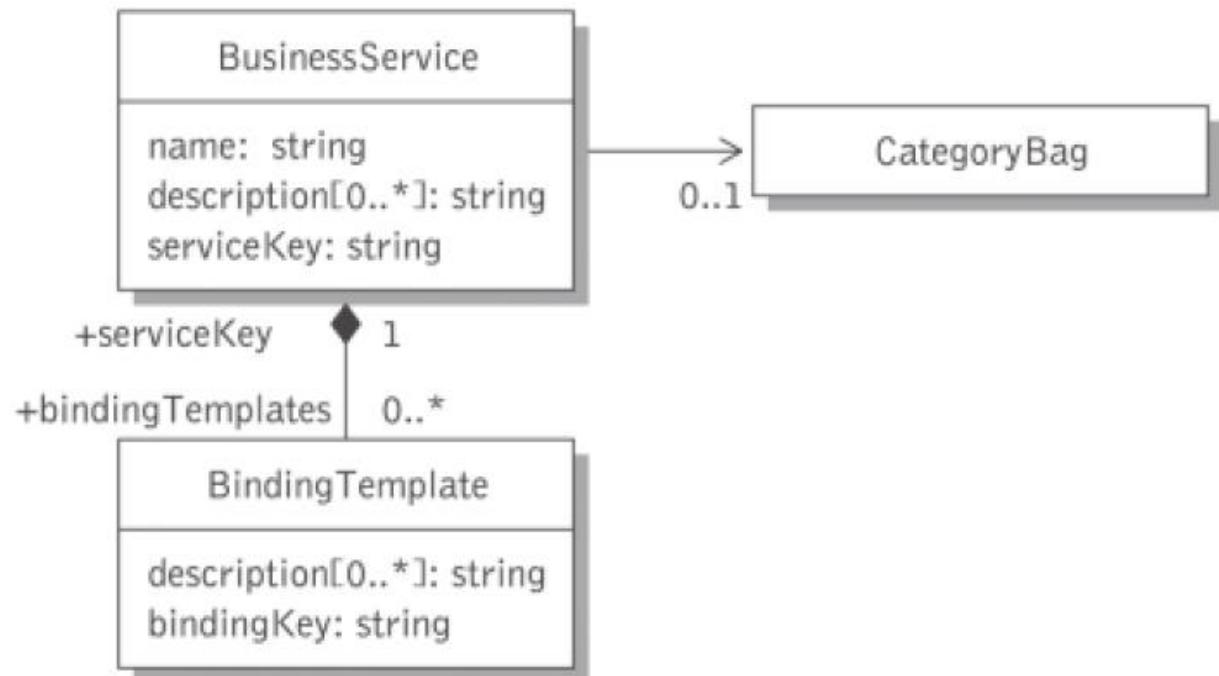
- ***The business services element.*** This is an optional list containing information describing logical families of the business services that this business entity provides.
- ***The identifier bag element.*** In addition to the descriptive information the UDDI registry provides information about enterprises and their services
- ***The contacts element.*** This element is an optional list of contact information for the organization
- ***The category bag element.*** A <categoryBag> element is similar to the <identifierBag>. This element is a list of one or more <keyedReference> structures that tag the business entity with specific classification information, e.g., industry, product, or geographic codes.

Example - Business entity

```
<businessEntity businessKey="d2300-3aff-.." xmlns = "urn:uddi-org:api_v2">
<name xml: lang="en"> Automotive Equipment Manufacturing Inc. </name>
<description xml: lang="en">
Automotive Equipment, Accessories and Supplies for European firms
</description>
<contacts>
<contact useType="Sales Contact">
<description xml: lang="en"> Sales Representative </description>
<personName> Reginald Murphy </personName>
<email useType="primary"> joe.murphy@automeq.com </email>
<address useType="http">
<addressLine> http://www.medeq.com/sales/ </addressLine>
</address>
</contact>
</contacts>
<businessServices>
<!-- Business service information goes here --&gt;
&lt;/businessServices&gt;
&lt;identifierBag&gt;
<!-- DUNS Number identifier System --&gt;
&lt;keyedReference keyName="DUNS Number" keyValue="..." tModelKey="..."/&gt;
&lt;/identifierBag&gt;
&lt;categoryBag&gt;
<!--North American Industry Classification System (NAICS) --&gt;
&lt;keyedReference keyName="Automotive parts distribution" keyValue="..." tModelKey="..."/&gt;
.....
&lt;/categoryBag&gt;
&lt;/businessEntity&gt;</pre>
```

# 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.



---

## example : business service

```
<businessServices>
<businessService serviceKey=" " >
<name> Search the Automotive Equipment Manufacturing parts Registry </name>
<description lang="en">
Get to the Automotive Equipment Manufacturing parts Registry
</description>
<bindingTemplates>
<bindingTemplate bindingKey=".." >
<description lang="en">
Use your Web Browser to search the parts registry
</description>
<accessPoint URLType="http">
http://www.automeq.com/b2b/actions/search.jsp
</accessPoint>
<tModelInstanceDetails>
<tModelInstanceStateInfo
tModelKey="uddi:..."/>
<tModelInstanceDetails>
</bindingTemplate>
</bindingTemplates>
</businessService>
</businessServices>
```

# 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.

A binding template contains the technical information associated to a particular service:

- bindingKey
- serviceKey
- description
- accessPoint: the network address of the service being provided
- tModels: a list of entries corresponding to tModels associated with this particular binding
- categoryBag: additional information about the service and its binding (e.g., whether it is a test binding, it is on production, etc).

```
<bindingTemplate bindingKey="">
  <description lang="en">
    Use your Web Browser to search the parts registry
  </description>
  <accessPoint URLType="http">
    http://www.automeq.com/b2b/actions/search.jsp
  </accessPoint>
  <tModellInstanceDetails>
    <tModellInstanceInfo
      tModelKey="uddi:.."/>
    <tModellInstanceDetails>
  </bindingTemplate>
```

A tModel is a generic container of information which contains technical information associated with the use of a Web service:

- the actual interface and protocol used, including a pointer to the WSDL description;
- description of the business protocol and conversations supported by the service.

```
<tModel tModelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64"
         operator="www.ibm.com/services/uddi"
         authorizedName="0100001QS1">
    <name>XMethods Simple Stock Quote</name>
    <description xml:lang="en">Simple stock quote interface</description>
    <overviewDoc>
        <description xml:lang="en">wsdl link</description>
        <overviewURL>http://www.xmethods.net/tmodels/SimpleStockQuote.wsdl</overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
                       keyName="uddi-org:types"
                       keyValue="wsdlSpec" />
    </categoryBag>
</tModel>
```

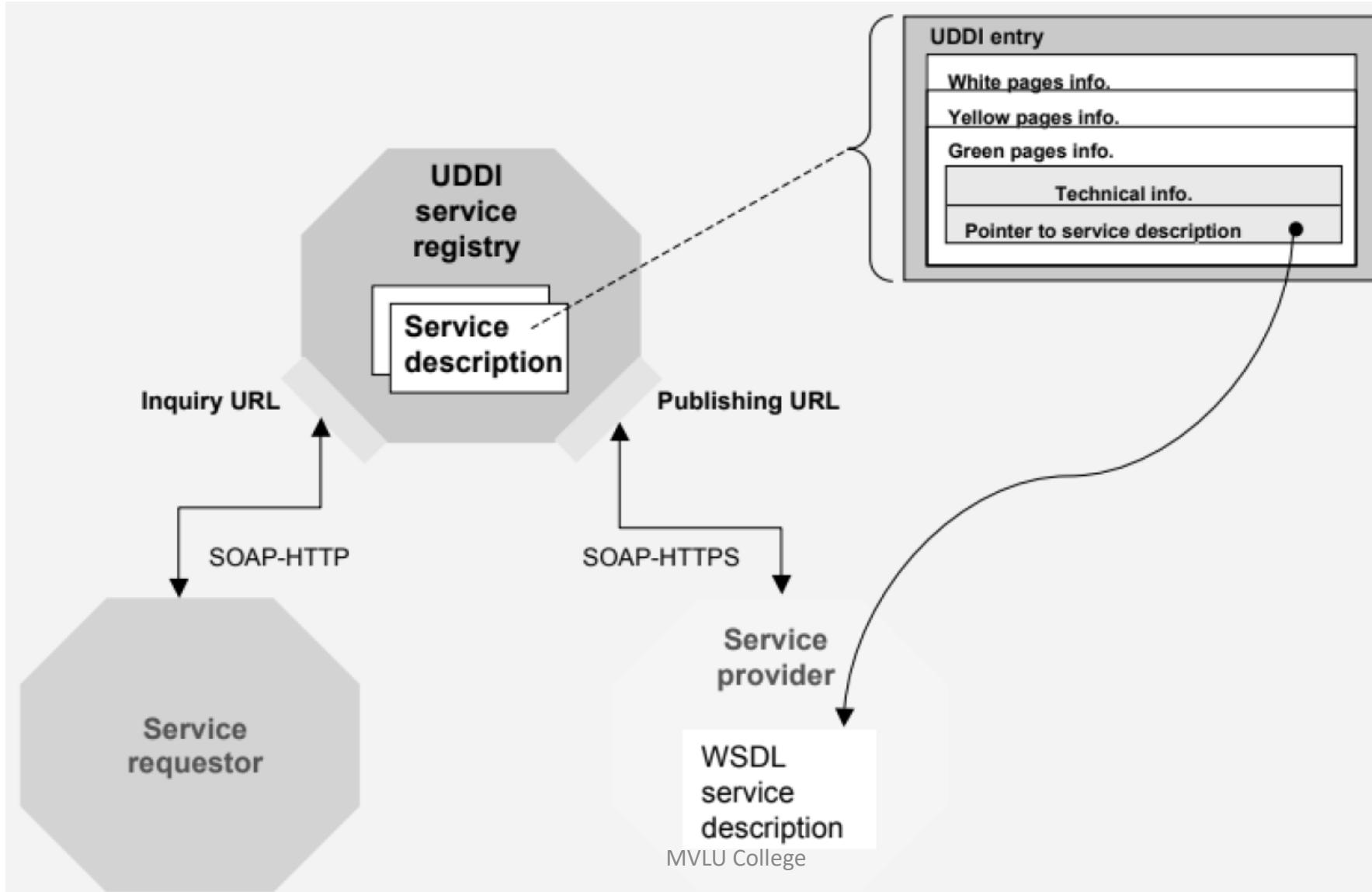
# publisherAssertion

- 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.

## **example of publisherAssertion**

```
<publisherAssertion>
<fromKey> FE565 ... <\fromKey>
<toKey> A237B ... <\toKey>
<keyedReference tModelKey="uuid:807A .. " />
keyName="subsidiary"
keyValue="parent-child">
</keyedReference>
</publisherAssertion>
```

# WSDL to UDDI mapping model



# Service-Oriented Architecture

- SOA is an architectural style for building software applications that use services available in a network such as the web.
- It promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services.
- A service is an implementation of a well-defined business functionality, and such services can then be consumed by clients in different applications or business processes.

# SOA

- SOA allows for the reuse of existing assets where new services can be created from an existing IT infrastructure of systems.
- In other words, it enables businesses to reuse existing applications, and promises interoperability between heterogeneous applications and technologies.
- SOA provides a level of flexibility that wasn't possible before in the sense that:

- Services are software components with well-defined interfaces that are implementation-independent. An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how). Such services are consumed by clients that are not concerned with how these services will execute their requests.
- Services are self-contained (perform predetermined tasks) and loosely coupled (for independence)
- Services can be dynamically discovered
- Composite services can be built from aggregates of other services

# Advantages of SOA



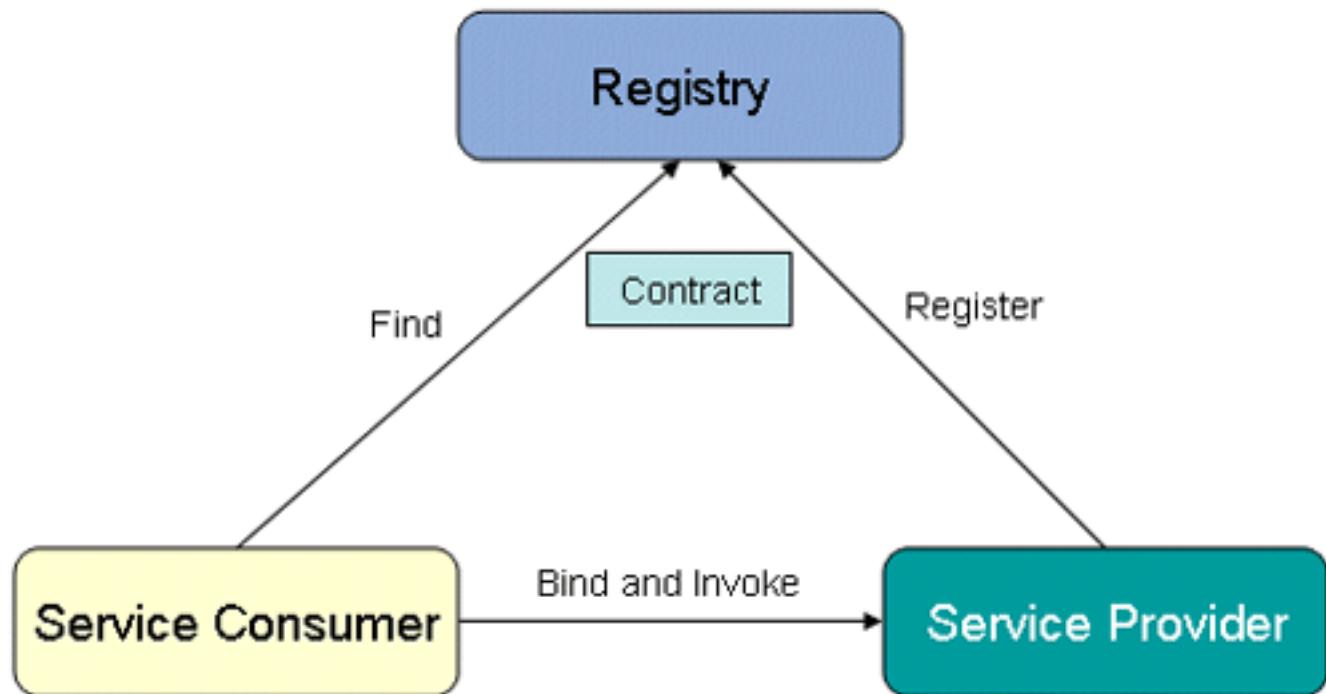
- **Easy to integrate** - In a service-oriented architecture, the integration is a service specification that provides implementation transparency.
- **Manage Complexity** - Due to service specification, the complexities get isolated, and integration becomes more manageable.
- **Platform Independence** - The services are platform-independent as they can communicate with other applications through a common language.
- **Loose coupling** - It facilitates to implement services without impacting other applications or services.

# Advantages of SOA



- **Parallel Development** - As SOA follows layer-based architecture, it provides parallel development.
- **Available** - The SOA services are easily available to any requester.
- **Reliable** - As services are small in size, it is easier to test and debug them.

# SOA



# Components of SOA



## Components of SOA

Service

Service Description



# The Service

- A service a software module deployed or installed on network provided by the service provider. The client application may request a service for the execution of their applications.

# The Service Description



- The service descriptions consists of the detail information about the interface and implementation of the service. This involves its data types, binding information, operation and network location.

## Roles of SOA

Service Provider

Service Requestor

Discovery Agency  
or Service Registry

# Service Provider



- From a business point of view, Service Provider is the owner of the service . From an architectural point of view, this is the device or platform that hosts and provides the access to the service.

# Service Requestor



- From a business point of view, Service Requestor is the business or client that required certain function to be fulfilled. From an architectural point of view, this is the application that is want to invoke or initialize the interaction with a service.

# Discovery Agency or Service Registry



- This is the registry or agency, means a searchable set of service descriptions where services providers publish or stores their service description. The service discovery agency can be in the both form i.e. centralized or distributed.

# Operation

## Operation of SOA

1. Publish

2. Find

3. Bind

# Operation of SOA

## Publish

- In order to get accessed by requestor, services suppose to publish its description .
- The location with the services is published by provider or publisher is the Service Registry or Discovery Agencies.

## Find

- In the find operation, the service requestor retrieves or find a service description directly or queries the service registry if the specific sort services required.

## Bind

- In the bind operation the service requestor invokes an service at runtime by using the binding information in the service description to locate , contact and invoke the service.

# UDDI and SOA Questions

1. What is mean by Registering and discovering Web services?
2. Explain UDDI and its basic components
3. What are the different types of UDDI – data structures?
4. Explain the Service oriented Architecture with diagram
5. What are the Roles and Operations of SOA?

## Unit II Chapter I

# Web Services Development Lifecycle

# Web Services Development Lifecycle



- A Service Oriented Design and development methodology provide model, references architecture and run time environment.
- A Service Oriented Design and Development methodology provide guidance through all SOA development, deployment and production phase.

# Web Services Development Lifecycle



# 1. The Service Planning Phase

- The planning phase decide the development process , models to be used and development.
- This phase the project flexibility study, constraints, goals and requirement gathering performed.
- The objective of this phase is to understand the business requirements and translating them into WS requirements in terms of the features, the functional and non-functional requirements, and the WS Constraint.

## 2. The Service Analysis Phase

- Service analysis aims at identifying and describing the processes in a business problem domain And discover potential overlaps.
- Develops in-depth understanding of functionality, scope, reuse and granularity of candidate business process and service.

### 3. Service Design Phase

- Service design required developers to define related, well documented interfaces for all conceptual services.

## 4. The Service Construction Phase



- This phase implements web services and business processes using specification define in the design phase.
- This phase create hosting environment and deploy the SOA based application.
- It include development of web service implementation, definition of service interface etc.

## 5. The Service Test Phase

- The goal of testing is to analyse implementation in attest environment in a test environment
- It exposes any failure that is in the code as it is integrated, configure and run on diverse platform.
- The testing phase assure the application satisfies the customer needs.

## 6. Service Provisioning Phase



- It is a complex mixture of technical and business aspects for supporting service client and provide activities.
- Service that flow between enterprises have define owner with established ownership and governance responsibilities.
- SOA governance refer to the organization, process, policies and metrics that are required to manage an SOA successfully.
- Two different governance model are possible :
  - Central governance
  - Versus federated governance.



## 7. The Service Deployment Phase

- Deployment is basically actual installation, implementation and publication of web service.

## 8. Service Execution Phase

- The service execution phase in the SDLC approach concern with the execution of Web services. This phase includes the actual binding and run-time invocation of the deployed services, managing and monitoring their lifecycle.

## 9. Service Monitoring Phase

- The service monitoring phase related to service-level measurements.
- The monitoring is the continuous and closed loop procedure of measuring , reporting.

# Unit II

# Chapter II

# The REST Architectural Style

# What are RESTful Web Services

- A Web Service that communicates or exchanges information between two applications using REST architecture / principles is called a RESTful Web Services.
- Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web Services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

# Introduction to Restful web Services



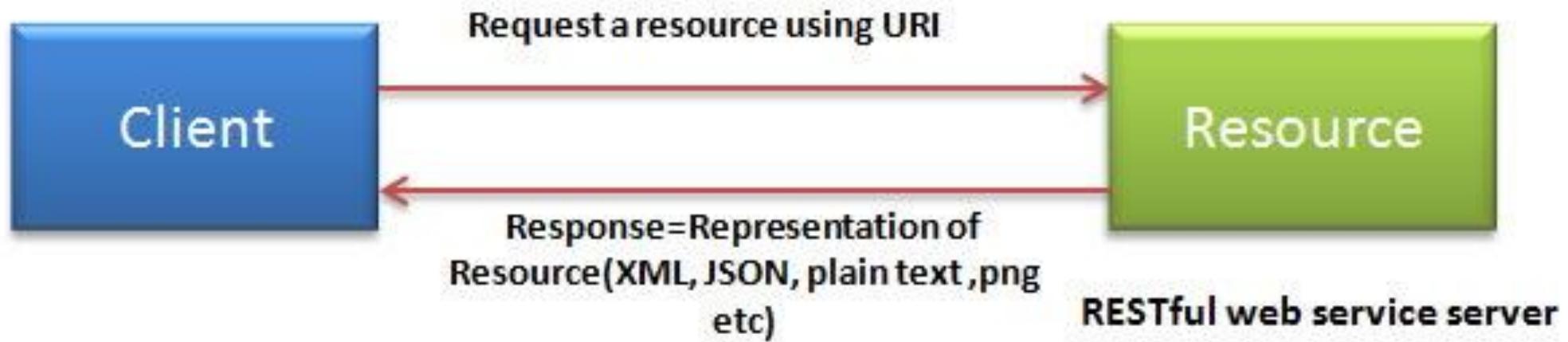
- REST is an architectural style which was brought in by Roy Fielding in 2000 in his doctoral thesis. In the web services terms, REpresentational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URLs. Web service clients that want to use these resources access via globally defined set of remote methods that describe the action to be performed on the resource.
- Unlike SOAP it is not protocol. There are no enforced specifications. There is no central body which is controlling the specifications

# Introduction to Restful web Services



- It consists of two components REST
- **Server** which provides access to the resources and a **REST client** which accesses and modify the REST resources.
- In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol. REST isn't protocol specific.
- The response from server is considered as the representation of the resources. This representation can be generated from one resource or more number of resources.

# Introduction to Restful web Services



# Introduction to Restful web Services



**1. Resource :** everything is a resource

**Resources**

**Employees**

Id

Name

**Departments**

Id

Name

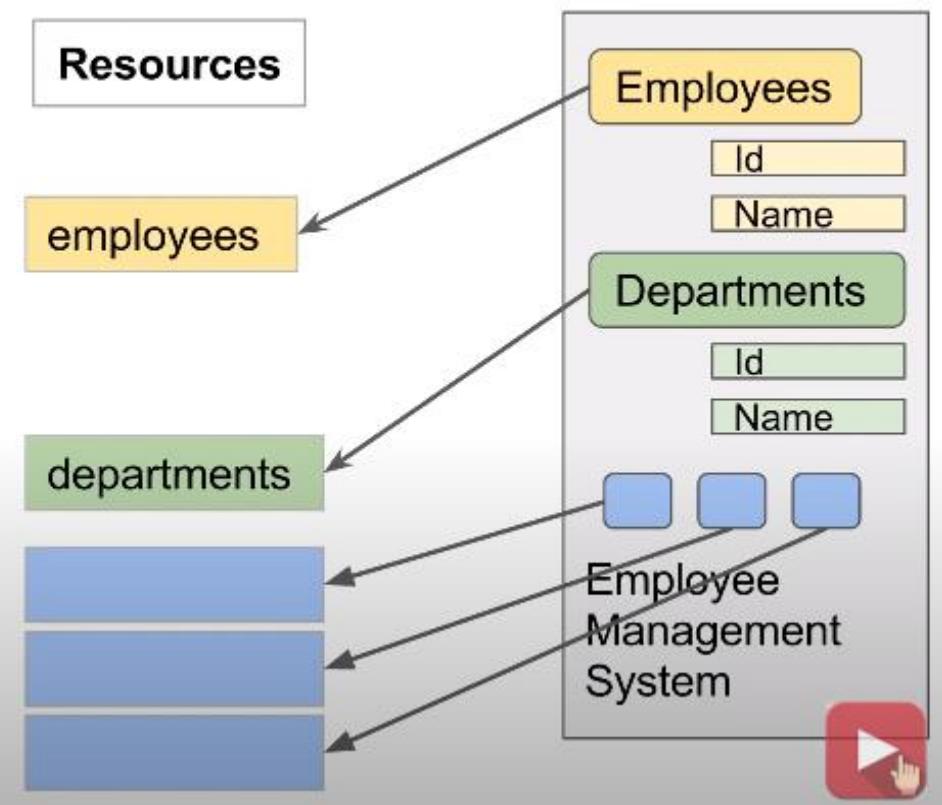


**Employee  
Management  
System**



# Introduction to Restful web Services

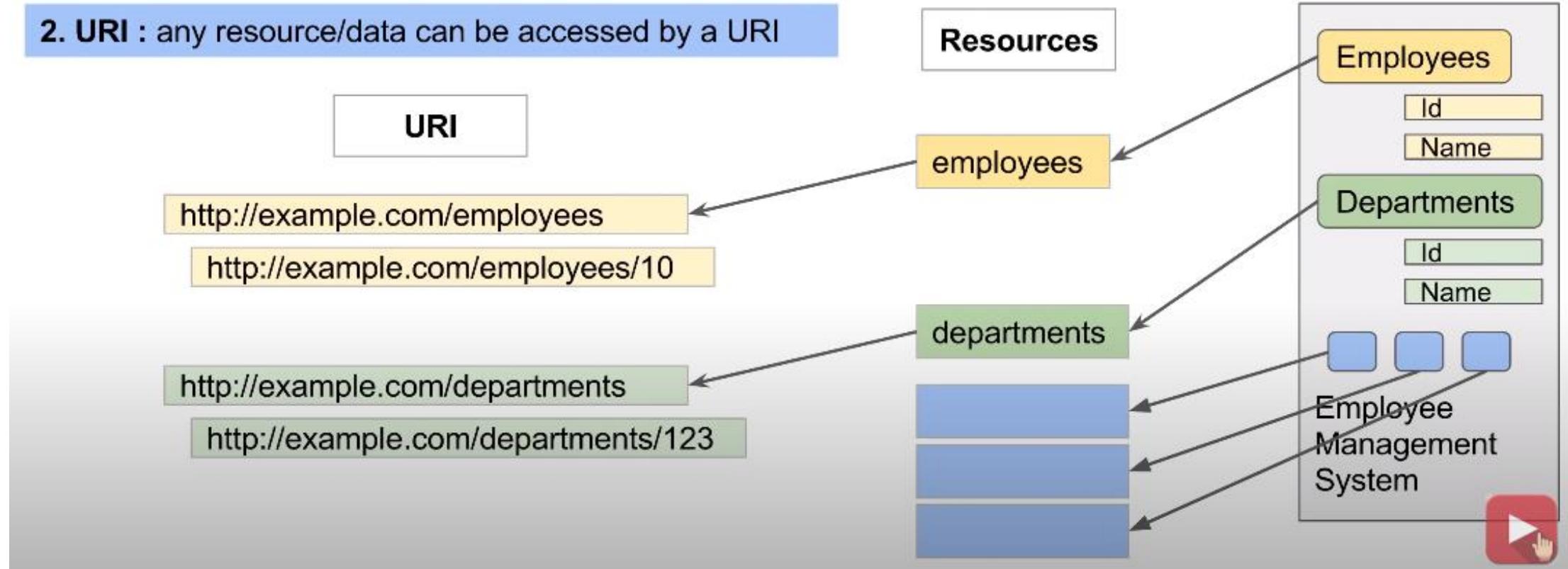
## 1. Resource : everything is a resource



# Introduction to Restful web Services

**1. Resource :** everything is a resource

**2. URI :** any resource/data can be accessed by a URI



# Introduction to Restful web Services

**1. Resource :** everything is a resource

**2. URI :** any resource/data can be accessed by a URI

**3. HTTP :** make explicit use of HTTP methods

HTTP

URI

GET

POST

PUT

DELETE

...

**CRUD**

C = CREATE = POST  
 R = READ = GET  
 U = UPDATE = PUT  
 D = DELETE = DELETE

com/employees

com/employees/10

com/departments

com/departments/123

<http://example.com/departments/123/employees>

Resources

employees

departments

Employees

Id

Name

Departments

Id

Name

Employee Management System



**1. Resource** : everything is a resource

**2. URI** : any resource/data can be accessed by a URI

**3. HTTP** : make explicit use of HTTP methods

HTTP

URI

GET

http://example.com/employees

POST

GET - http://example.com/employees/10

**Response** : details of employee with id=10

PUT

http://example.com/departments

DELETE

http://example.com/departments/123

...

http://example.com/departments/123/employees

Resources

employees

Employees

Id

Name

Departments

Id

Name

Employee  
Management  
System



# Introduction to Restful web Services



Using HTTP Methods along with URI, we can access/modify any resource or resource information.

## REQUEST

GET - <http://example.com/employees>

GET - <http://example.com/employees/10>

DELETE - <http://example.com/employees/10>

POST - <http://example.com/employees>

+

Data of new employee

PUT - <http://example.com/employees/10>

+

Data to be changed

## RESPONSE

list of employees

details of employee with id=10

deletes employee with id=10

id of new employee

modifies data for employe 10

# Constraints of RESTful web services

# Constraints of RESTful Web Services



- REST defines a set of principles to be followed while designing a service for communication or data exchange between two applications.

# What are the principles / Constraints of REST architecture



1. Client-Server

2. Stateless

3. Cacheable

4. Uniform interface

5. Layered system

6. Code on demand (optional)

# 1. Client-Server



- The first constraint of a REST full implementation is the separation of client and Servers. A client is defined as a machine requesting a resources, where the Server is the machine that response with the requested resource.
- This constraint keeps the client and the server loosely coupled. In this case, the client does not need to know the implementation details in the server and the server is not worried about how the data is used by the client. However, a common interface is maintained between the client and the server to ease the communication.

## 2. Stateless

- A truly RESTful architecture is not allowed to retain information about the state of another machine during the communication process.
- It means that the Server does not maintain any state of system. Hence the Client has to send the request which is complete in itself.
- Each request from a client to Server must be treated as though it was first request the Server has every seen from that client.
- There should be no need for the service to keep users' sessions. In other words, each request should be independent of the others.
- Improves Web Service performance

### 3. Cacheable

- This way , responses can be cached by the Client if the information on the Server hasn't changed since the last request.
- Server generates responses that indicate whether they are cacheable or not to improve the performance by reducing number of request for duplicate resources.
- Server does this by including a **Cache-Control** and **last-Modified** (data value) in HTTP Response headers

## 4. Uniform interface



- RESTful architectures must have a uniform interface between all clients and Servers
- This constraint indicates a generic interface to manage all interactions between the client and the server in a unified way, which simplifies and decouples the architecture. This constraint indicates that each resource exposed for use by the client must have a unique address and should be accessible through a generic interface. The client can act on the resources by using a generic set of methods.
- Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.

## 4. Uniform interface ...

- For us this means
  - HTTP Verbs (GET,POST,PUT,DELETE)
  - URIs (resource name)
  - HTTP response (status and body)

## 4. Uniform interface ...

The four guiding principles of the uniform interface are:

- 1.1) **Identification of resources** – Each resources has a URI and is accessed through a define set of HTTP methods (GET, PUT , POST, DELETE)
- Resources + URI + HTTP

HTTP/1.1 GET http://abc.com/users/dinesh

## 4. Uniform interface ...

### 1.2) Resource representation –

Each resource can have one or more representations. Is how the resource will return to the client. This representation can be in HTML, XML, JSON, TXT, and more.

```
{  
  "name": "Dinesh Rajput",  
  "job": "REST API Developer",  
  "hobbies": ["blogging", "coding", "music"]  
}
```

## 4. Uniform interface ...

### 1.3) Self-descriptive Messages –

Request and Responses not only contain data but additional headers , describing how content should be handled. Such as if it should be cached, authentication requirements, , HTTP response code, Host, Content-Type etc etc.

GET /#/user/dinesh HTTP/1.1

User-Agent: Chrome/37.0.2062.94

Accept: application/json

Host: dineshonjava.com

## 4. Uniform interface ...

- **1.4) Hypermedia as the Engine of Application State (HATEOAS)-**  
Clients deliver state via body contents, query-string parameters, request headers and the requested URI (the resource name). Services deliver the state to clients via body content, response codes, and response headers. This part is often overlooked when talking about REST. It returns all the necessary information in response to the client knows how to navigate and have access to all application resources.

## 5. Layered system



- A layered system means that Client can have access to endpoint that relies on other endpoints without having to understand all of the underlying implementations
- The server can have multiple layers for implementation. This layered architecture helps to improve scalability by enabling load balancing. It also improves performance by providing shared caches at different levels. The top layer, being the door to the system, can enforce security policies as well.
- intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

## 5. Layered system ...

- Layering allows very complicated tasks to be completed without having to understand all of the underlying complexities that are required to generate the response.

## 6. Code<sup>z</sup>on demand (optional)



- This constraint is optional. This constraint indicates that the functionality of the client applications can be extended at runtime by allowing a code download from the server and executing the code. Some examples are the applets and the JavaScript code that get transferred and executed on the client side at runtime.

- <http://www.thomas-bayer.com/sqlrest>

## Resources:

The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is **http://abc.com**. Now in order to access an employee record resource via REST, one can issue the command **http://demo.abc/employee/1** - This command tells the web server to please provide the details of the employee whose employee number is 1.

# Request Verbs



- These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example **http://abc.com/employee/1**, the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.

# Request Headers



**Request Headers** – These are additional instructions sent with the request. These might define the type of response required or the authorization details.



# Request Body

Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.



# Response Body

**Response Body** – This is the main body of the response. So in our example, if we were to query the web server via the request **<http://abc.com/employee/1>** , the web server might return an XML document with all the details of the employee in the Response Body.

# Response Status codes

**Response Status codes** – These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

# SOAP vs REST Web Services

SOAP	REST
SOAP is a <b>protocol</b> .	REST is an <b>architectural style</b> .
SOAP stands for <b>Simple Object Access Protocol</b> .	REST stands for <b>REpresentational State Transfer</b> .
SOAP <b>can't use REST</b> because it is a protocol.	REST <b>can use SOAP</b> web services because it is a concept and can use any protocol like HTTP, SOAP.
<b>SOAP uses services interfaces to expose the business logic.</b>	<b>REST uses URI to expose business logic.</b>
<b>JAX-WS</b> is the java API for SOAP web services.	<b>JAX-RS</b> is the java API for RESTful web services.

# SOAP vs REST Web Services ...

SOAP	REST
SOAP <b>defines standards</b> to be strictly followed.	REST does not define too much standards like SOAP.
SOAP <b>requires more bandwidth</b> and resource than REST.	REST <b>requires less bandwidth</b> and resource than SOAP.
SOAP <b>defines its own security.</b>	RESTful web services <b>inherits security measures</b> from the underlying transport.
SOAP <b>permits XML</b> data format only.	REST <b>permits different</b> data format such as Plain text, HTML, XML, JSON etc.
SOAP is <b>less preferred</b> than REST.	REST <b>more preferred</b> than SOAP.

# SOAP vs REST Web Services ...

<p>SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.</p>	<p>REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.</p>
<pre>&lt;?xml version="1.0"?&gt; &lt;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"&gt;  &lt;soap:Body&gt; &lt;Demo.guru99WebService  xmlns="http://tempuri.org/"&gt;  &lt;EmployeeID&gt;int&lt;/EmployeeID&gt;  &lt;/Demo.guru99WebService&gt; &lt;/soap:Body&gt; &lt;/SOAP-  ENV:Envelope&gt;</pre>	<pre>{"city":"Mumbai","state":"Maharashtra"}</pre>



# Unit II

# Chapter III

# Hypertext Transfer Protocol (HTTP)

# Introduction HTTP

- Many application are running concurrently on the web, such as web browsing/surfing, email, file transfer, audio and video streaming and so on. For proper communication between the client and the sever , these applications must agree application-level protocol such as HTTP, FTP, SMTP, POP and more.

# Introduction HTTP

- Hypertext Transfer Protocol (HTTP) is the foundation of data communication for WWW. This protocol defines how messages are formatted, transmitted, and processed over the Internet.
- HTTP is most popular request-response client-server protocol. An HTTP client send a request message to an HTTP server for communication.

# Introduction HTTP ...

- HTTP is a stateless protocol. (does not know what has been done in previous request)
- Hypertext transfer protocol is an application –level protocol for the distributed, collaborative, hypermedia information systems.

# HTTP Versions



- HTTP has been consistently evolving over time. So far, there are three versions. HTTP/0.9 was the first documented version, which was released in the year 1991. This was very primitive and supported only the GET method.
- Later, HTTP/1.0 was released in the year 1996 with more features and corrections for the shortcomings in the previous release. HTTP/1.0 supported more request methods such as GET, HEAD, and POST. The next release was HTTP/1.1 in the year 1999. This was the revision of HTTP/1.0. This version is in common use today.
- HTTP/2 (originally named HTTP 2.0) It is mainly focused on how the data is framed and transported between the client and the server.

# Introduction HTTP ...

- HTTP makes use of the Uniform Resource Identifier (URI) to identify any given resource and establish a connection.
- The generic message of any HTTP request and response, consists of the following four items:
  - A start line
  - Zero or more Headers
  - Empty line indicating the end of the header fields
  - and, Optional message Body.

# Introduction HTTP ...

- HTTP header holds the metadata and information about the HTTP method, while the body contains the data that we want to send to the server.

# Understanding the HTTP request-response model





# Uniform resource Identifier

- URI is a text that identifies any resource or name on the Internet. One can further classify a URI as a Uniform Resource Locator (URL) if the text used for identifying the resource also holds the means for accessing the resource such as HTTP or FTP

# HTTP Methods



- The most important part of a request is the HTTP Method (verbs). These methods tell the server what to do with the data received in the request. You must be familiar with GET and POST as they are frequently used in FORM submission in HTML.
- Although there are 8 different types of methods but the 4 most important HTTP methods will be discussed here which are equivalent to the CRUD operations.

# Understanding the HTTP request methods

Method	Description
<b>GET</b>	This method is <b>used for retrieving resources</b> from the server by using the given URI.
<b>HEAD</b>	This method is the same as the GET request, but it <b>only transfers the status line and the header section without the response body</b> .
<b>POST</b>	This method is used for posting data to the server. <b>The server stores the data (entity) as a new subordinate of the resource identified by the URI</b> . If you execute POST multiple times on a resource, it may yield different results.

# Understanding the HTTP request methods ...



Method	Description
PUT	This method is <b>used for updating the resource pointed at</b> by the URI. If the URI does not point to an existing resource, the server can create the resource with that URI.
DELETE	This method <b>deletes the resource pointed at</b> by the URI.
TRACE	This method is used for echoing the contents of the received request. This is <b>useful for the debugging purpose</b> with which the client can see what changes (if any) have been made by the intermediate servers
OPTIONS	This method <b>returns the HTTP methods</b> that the server supports for the specified URI
CONNECT	This method is <b>used for establishing a connection</b> to the target server over HTTP.

# HTTP methods : GET

- The GET method is used to get the information from the server. This is equivalent to the SELECT statement in SQL. There is no message body in the GET request as it is meant to only fetch the data from the server.
- This is the simplest type, and is used by the browser every time you visit any link, to fetch the data. A GET request should never change the data on server, and should only be used to read data.

## **GET /students**

- The above should return the students data.

# HTTP Methods: POST

- This is used to send the data to the server. This can be students information, some XML file or some JPG file. The message body contains the data. POST requests are generally used to create but it can be used to update the existing data as well.

## **POST /students**

- The above should be able to create a new student entry, provided student data is sent in appropriate format in the message body. This can be used to update student data as well.

# HTTP Methods: PUT

- This should be used when we have to replace/update/create the data, which is already present/not present on the server with the data that we pass in the message body.

**POST /students/robin**

# HTTP Methods: DELETE

- This should be used to delete the data on the server. For example, the following URL when sent a DELETE request, should delete the student entry.

**DELETE /students/viraj**

# HTTP status codes

- For every HTTP request, the server returns a status code indicating the processing status of the request.

**1xx**

- Information

**2xx**

- Success

**3xx**

- Redirection

**4xx**

- Client error

**5xx**

- Server error

# 1xx Informational

This series of status codes indicates informational content. This means that the request is received and processing is going on. Here are the frequently used informational status codes:

**100 Continue:** This code indicates that the server has received the request header and the client can now send the body content. In this case, the client first makes a request

**101 Switching Protocols:** This code indicates that the server is OK for a protocol switch request from the client.

**102 Processing:** This code is an informational status code used for long running processing to prevent the client from timing out. This tells the client to wait for the future response, which will have the actual response body.

# 2xx Success

This series of status codes indicates the successful processing of requests. Some of the frequently used status codes in this class are as follows:

**200 OK:** This code indicates that the request is successful and the response content is returned to the client as appropriate.

**201 Created:** This code indicates that the request is successful and a new resource is created.

**202 Accepted**

**204 No Content:** This code indicates that the request is processed successfully, but there's no return value for this request. For instance, you may find such status codes in response to the deletion of a resource.

# 3xx Redirection

This series of status codes indicates that the client needs to perform further actions to logically end the request. A frequently used status code in this class is as follows:

**300 Multiple choice**

**301 Moved permanently**

**302 Found**

**304 Not Modified:** This status indicates that the resource has not been modified since it was last accessed.

# 4xx Client Error

This series of status codes indicates an error in processing the request. Some of the frequently used status codes in this class are as follows:

**400 Bad Request:** This code indicates that the server failed to process the request because of the malformed syntax in the request. The client can try again after correcting the request.

**401 Unauthorized:** This code indicates that authentication is required for the resource. The client can try again with the appropriate authentication.

**403 Forbidden:** This code indicates that the server is refusing to respond to the request even if the request is valid. The reason will be listed in the body content if the request is not a HEAD method.

**404 Not Found:** This code indicates that the requested resource is not found at the location specified in the request.

**408 Request timeout**

# 5xx Server Error



This series of status codes indicates server failures while processing a valid request. Here is one of the frequently used status codes in this class:

**500 Internal Server Error:** This code indicates a generic error message, and it tells that an unexpected error occurred on the server and the request cannot be fulfilled.



# Description and Discovery of RESTful web services



- As you know, WSDL is used for describing the functionality offered by a SOAP web service. For a SOAP web service, this is a widely accepted standard and is supported by many enterprises today. In contrast, for RESTful web services, there is no such standard and you may find different metadata formats used by various enterprises.
- However, in general, you may see the following goals in common among all these metadata formats for RESTful APIs, although they differ in their syntax and semantics:

# Description and Discovery of RESTful web services



- Entry points for the service
- Resource paths for accessing each resource
- HTTP methods allowed to access these resources, such as GET, POST, PUT, and DELETE
- Additional parameters that need to be supplied with these methods, such as pagination parameters, while reading large collections
- Format types used for representing the request and response body contents such as JSON, XML, and TEXT
- Status codes and error messages returned by the APIs
- Human readable documentation for REST APIs, which includes the documentation of the request methods, input and output parameters, response codes (success or error), API security, and business logic

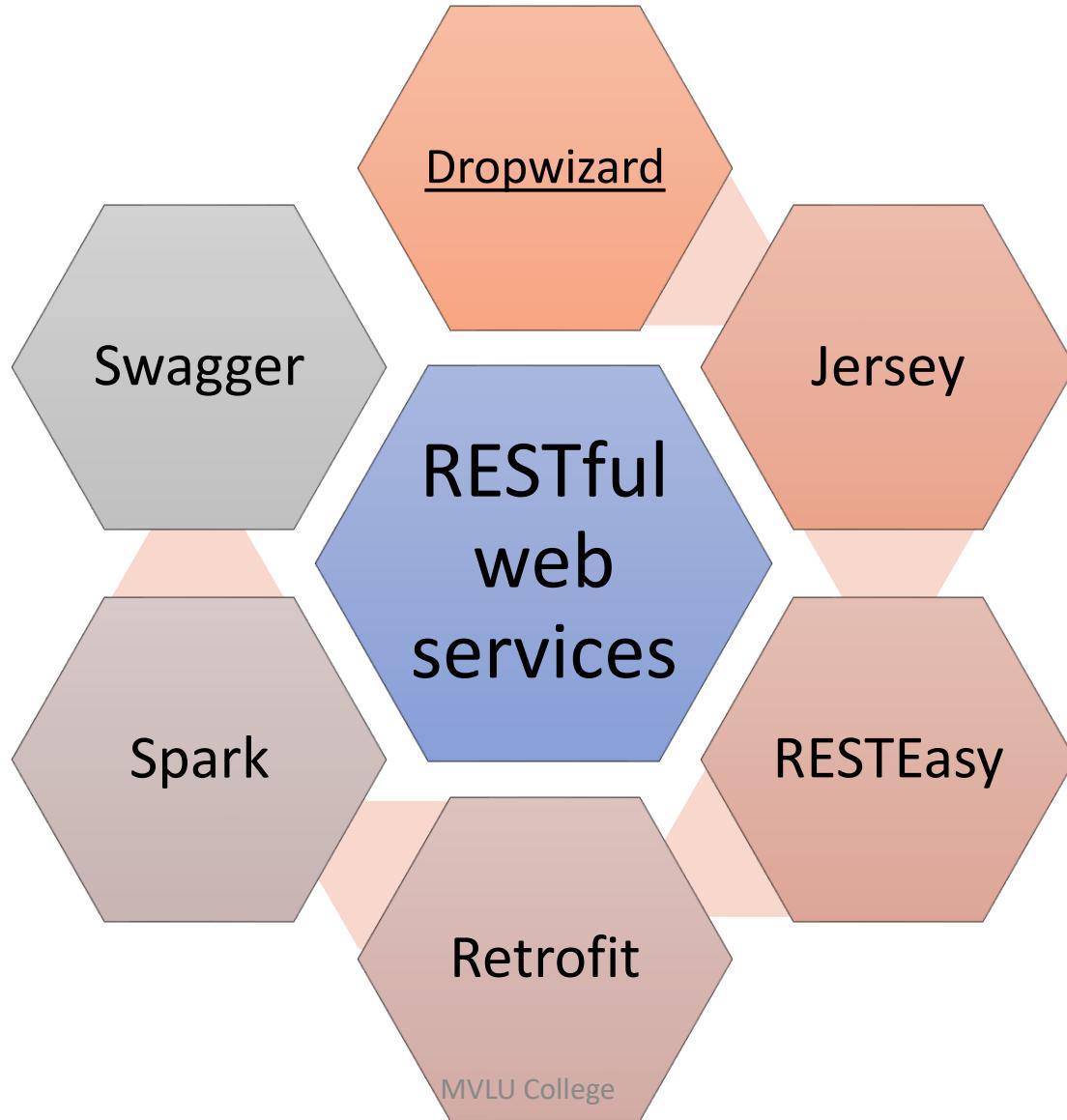
# Description and Discovery of RESTful web services



- Some of the popular metadata formats used for describing REST APIs are Web Application Description Language (WADL), Swagger, RESTful API Modelling Language (RAML), API Blueprint, and WSDL 2.0.

- the REST architectural style has become very popular in the industry and many enterprises have accepted it as the current standard for building public web APIs, particularly when scalability and simplicity are major concerns for them.
- The Java API for RESTful web services (JAX-RS) is the Java API for creating RESTful web services following the REST architectural pattern
- JAX-RS is a part of the Java Platform Enterprise Edition (Java EE) platform and is designed to be a standard and portable solution.

# Java tools and frameworks for building RESTful web services



## 1) Dropwizard

- Dropwizard is an open source Java framework used for the fast development of RESTful web services. it's a light-weight best-in-class set of tools and frameworks for building RESTful web services.
- high-performance
- framework for setting up modern web applications, includes Jetty, Jackson, Jersey and Metrics.
- Dropwizard has out-of-the-box support for sophisticated configuration, application metrics, logging, operational tools, and much more, allowing you and your team to ship a production-quality HTTP+JSON web service in the shortest time possible.

- Jersey is a JAX-RS reference implementation.
- [Jersey RESTful Web Services](#) framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS ([JSR 311](#) & [JSR 339](#)) Reference Implementation. Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides it's own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development.

- RESTEasy is a fully certified and portable implementation of the JAX-RS specification which provides a Java API for RESTful Web Services over the HTTP protocol. RESTEasy is a JBoss project that provides various frameworks to help you build RESTful Web Services and RESTful Java applications. JAX-RS is a new JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol.

# Retrofit

Retrofit is a type-safe REST client for Java and Android . Retrofit is a library that will let you define your API in a simple Java interface and will automatically convert it into a full-blown REST client. Retrofit makes it easy to consume JSON or XML data which is parsed into Plain Old Java Objects (POJOs).

# Spark



- Apache Spark is a lightning-fast cluster computing designed for fast computation. It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations which includes Interactive Queries and Stream Processing. This is a brief tutorial that explains the basics of Spark Core programming.

# Swagger



Swagger is a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTful web services. Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability.

# Unit II

# Chapter IV

# JavaScript Object Notation (JSON)

# JavaScript Object Notation JSON



- JSON stands for JavaScript Object Notation.
- JSON objects are used for storing and exchanging data.
- XML serves the same purpose.
- Basically it was designed for human-readable data interchange.
- JSON is text, written with Java Script Object Notation
- It has been extended from the JavaScript script language
- The extension is .json
- JSON internet Media type is application/json

# Uses of JSON

- Helps you to transfer data from a server
- JSON format helps transmit and serialize all types of structured data.
- All you to perform asynchronous data call without the need to do a page refresh (AJAX)
- It is widely used for JavaScript-based application, which includes browser extension and websites.
- You can transmit data between the server and web application using JSON.
- We can use JSON with moden programming language.
- It us used for writing JavaScript-based application that include browsers add-ons.
- Web services and APIs use the JSON FORMAT to get public data

# Characteristics of JSON

- JSON is easy to read and write
- It is lightweight text-based interchange format
- JSON is language independent

# JSON example

```
var employee = {  
    "firstName" : "Ramesh",  
    "lastName" : "Singh",  
    "age" : "28"  
};
```

# Features of JSON



- It is light-weight
- It is language independent
- Easy to read and write
- Fast
- Text based, human readable data exchange format

# Syntax Rules of JSON

Data is in name/value pairs.

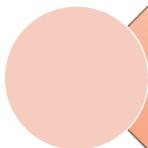
Data is separated by Commas

Curly braces hold objects.

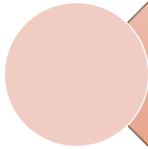
Square brackets hold arrays.

```
{  
  "student": [  
    {  
      "id": "01",  
      "name": "Tom",  
      "lastname": "Price"  
    },  
    {  
      "id": "02",  
      "name": "Nick",  
      "lastname": "Thameson"  
    }  
  ]  
}
```

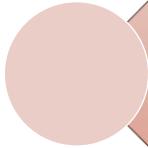
# JSON Data types



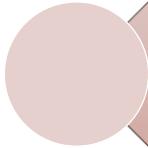
Number



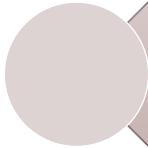
String



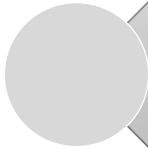
Boolean



Null



Object



Array

# string data types

- String in JSON must be written in double quotes
- E.g. { “name” : “John”}

# Numbers data types

- Numbers in JSON must be an integer or a floating point.
- E.g
- { “age” : 30}

# Object data types

- Values in JSON can be objects

- .e.g.

```
{  
  "employee" : { "name" : "Arun",  
                 "age" : 25,  
                 "city": "Navi Mumbai"  
               }  
}
```

# Array data types

- Values in JSON can be array

```
{  
    "employees" : [ "john", "Smith", "Peter" ]  
}
```

# Boolean data types

- Values in JSON can be true/false
- { “sales” : true }

# Null data types

- Values in JSON can be null
- { “middlename” : null }

# JSON vs XML

JSON	XML
JSON is simple to read and write	XML is less simple as compared to JSON.
It also supports array.	It doesn't support array.
JSON files are more human-readable than XML	XML files are less human-readable.
It supports only text and number data type	It supports many data types such as text, number, images, charts, graphs etc.

# JSON vs XML

```
{"students": [
    {"name": "John", "age": "23", "city": "Agra"},
    {"name": "Steve", "age": "28", "city": "Delhi"},
    {"name": "Peter", "age": "32", "city": "Chennai"},
    {"name": "Chaitanya", "age": "28", "city": "Bangalore"}
]}
```

```
<students>
  <student>
    <name>John</name> <age>23</age> <city>Agra</city>
  </student>
  <student>
    <name>Steve</name> <age>28</age> <city>Delhi</city>
  </student>
  <student>
    <name>Peter</name> <age>32</age> <city>Chennai</city>
  </student>
  <student>
    <name>Chaitanya</name> <age>28</age> <city>Bangalore</city>
  </student>
</students>
```

# JSON data structure types

- 1. An unordered collection of name-value pairs (representing an object):
- 2. An ordered collection of values (representing an array):

# 1. An unordered collection of name-value pairs

The attributes of an object and their values are represented in the name-value pair format

- the name and the value in a pair is separated by a colon (:)
- Names in an object are strings, and values may be of any of the valid JSON data types such as number, string, Boolean, array, object, or null.
- Each name: value pair in a JSON object is separated by a comma (,).
- The entire object is enclosed in curly braces ({}).

e.g.:

```
{"departmentId":10, "departmentName":"IT", "manager":"John Chen"}
```

## 2. An ordered collection of values

- Arrays are enclosed in square brackets ([ ]),
- and their values are separated by a comma (,).

# Similarities between JSON and XML

- Both are simple and open
- Both supports Unicode. So Internationalization is supported by JSON and XML both
- Both represents self-describing data
- Both are language-independent.

```
{  
    "firstName": "John",  
    "lastName" : "doe",  
    "age"      : 26,  
    "address" : {  
        "streetAddress": "naist street",  
        "city"        : "Nara",  
        "postalCode"  : "630-0192"  
    },  
    "phoneNumbers": [  
        {  
            "type" : "iPhone",  
            "number": "0123-4567-8888"  
        },  
        {  
            "type" : "home",  
            "number": "0123-4567-8910"  
        }  
    ]  
}
```

```
{  
  "books":  
  [  
    {  
      "title": "Professional JavaScript",  
      "authors": [  
        "Nicholas C. Zakas"  
      ],  
      "edition": 3,  
      "year": 2011  
    },  
    {  
      "title": "Professional JavaScript",  
      "authors": [  
        "Nicholas C.Zakas"  
      ],  
      "edition": 2,  
      "year": 2009  
    },  
    {  
      "title": "Professional Ajax",  
      "authors": [  
        "Nicholas C. Zakas",  
        "Jeremy McPeak",  
        "Joe Fawcett"  
      ],  
      "edition": 2,  
      "year": 2008  
    }  
  ]
```

## 1. Object model:

- In this model, the entire JSON data is read into memory in a tree format.
- This tree can be traversed, analysed, or modified with the appropriate APIs. As this approach loads the entire content into the memory first and then starts parsing, it ends up consuming more memory and CPU cycles.
- This model gives more flexibility while manipulating the content.

## 2. Streaming model:

- The term streaming is very generic in meaning and can be used in many aspects.
- This model does not read the entire JSON content into the memory to get started with parsing; rather, it reads one element at a time.

# Tools and frameworks

- APIs available on the Java EE platform for processing JSON. Java EE 7 has standardized the JSON processing APIs with Java Specification Request (JSR), that is, JSR 353 - Java API for JSON Processing.
- This JSR offers portable APIs to parse, generate, transform, and query JSON data.

# Processing JSON with JSR 353 object model APIs



- This category of APIs generates an in-memory tree model for JSON data and then, starts processing it as instructed by the client. This is conceptually similar to the Document Object Model (DOM) API for XML.

# Processing JSON with JSR 353 object model APIs



`javax.json.Json`

- This class is the main factory class for creating JSON processing objects such as `JsonReader` and `JsonWriter`.

`javax.json.JsonReader`

- This interface reads the JSON content and generates a JSON object or array as appropriate

`javax.json.JsonWriter`

- This interface writes a JSON object or array to an output source.

`javax.json.JsonArrayBuilder`

- This interface offers APIs for generating `JsonArray` models from scratch.

`javax.json.JsonValue`

- `javax.json.JsonObject`, `javax.json.JsonArray`,  
`javax.json.JsonNumber`, `javax.json.JsonString`,  
`javax.json.JsonValue.TRUE`

# Processing JSON with JSR 353 streaming APIs



- This interface offers APIs for generating **JsonObject** models from scratch.
- This category of APIs supports the streaming model for both reading and writing the JSON content. This model is designed to process a large amount of data in a more efficient way. Conceptually, this model is similar to the **Streaming API for XML (StAX) parser**
- Streaming APIs are grouped in the `javax.json.stream` package in the JSR specification.

`javax.json.stream.Json  
Parser`

- This class provides forward read-only access to JSON data by using the pull parsing programming model.

`javax.json.stream.Json  
Generator`

- This class writes JSON to an output source as specified by the client application. It generates the name-value pairs for the JSON objects and values for the JSON arrays.

# Create simple restful web service using java



- Create simple restful web service using java

# RESTful web services – JAX-RS Annotations

@GET

@POST

@PUT

@DELETE

@Path

@PathParam

@Consumes

@Produces

# @GET and @POST

## **@GET**

Annotate your Get request methods with @GET.

## **@POST**

Annotate POST request methods with @POST.

# @PUT and @DELETE



## **@PUT**

Annotate PUT request methods with @PUT.

## **@DELETE**

Annotate DELETE request methods with @DELETE.

**@Path** annotation specify the URL path on which this method will be invoked.



# @PathParam

We can bind REST-style URL parameters to method arguments using @PathParam annotation.

# @PathParam examples

```
@Path("convertCtoFInput/{c}")
@GET
@Produces("application/xml")
public String convertCtoFInput(@PathParam("c") Double c) {
    Double fahrenheit;
    fahrenheit = ((c * 9) / 5) + 32;
    return "<ctofservice><celsius>" + c + "</celsius><fahrenheit>" + fahrenheit + "</fahrenheit></ctofservice>";
}
```



# @Consumes

The `@Consumes` annotation is used to specify the MIME media types a REST resource can consume.



# @Produces

**@Produces** annotation specifies the type of output this method (or web service) will produce.

1. Naming RESTful web resources.
2. Not a good practice to use URLs with CRUD operations
3. Resources are handles by HTTP verbs (GET, POST, PUT, and DELETE)
4. HTTP response status codes
5. Field name casing convention
6. Searching, sorting, filtering and pagination
7. Versioning

# 1. Naming RESTful web resources

- It is recommended to use nouns to name both resources and path segments , that will appear in the resource URI. You should avoid using verbs for naming resources and resources path segments.
- Use Plural nouns when required.
- E.g
- <http://localhost:5000/organization/Departments/1>
- To indicate that there are many departments with an organizations



# Design to improve readability

Design to improve readability

E.g

<http://localhost:5000/organization/Product/user-ratings>

Rather than

<http://localhost:5000/organization/Product/userRatings>

# Do not use file extensions like html, aspx etc.



- Do not use file extensions like html, aspx etc.

## 2. Not good practice to use URLs with CRUD operations



- E.g. instead of this URL

<http://localhost:5000/organization/GetDepartments/1>

Use

<http://localhost:5000/organization/Departments/1>

With a GET request

### 3. Resources are handles by HTTP verbs (GET, POST, PUT, and DELETE)

- Uses http verbs to operate on the collections and elements.

Resources	POST (Create)	GET (Read)	PUT (Update)	Delete
/employees	Create new employee	List Employees	Bulk update employees	Delete all employees
/employees/105	Error	Show employees details of 105	If exists update employee details  Else Error	Delete employee 105

## 4. HTTP response status codes

- When the client raises a request to the server, the client should know the feedback, whether it failed, passed or the request was wrong. HTTP status codes are bunch of standardized codes which has various explanations in various scenarios. The server should always return the right status code.

## 5. Field name casing convention

- You can follow any casing convention, but make sure it is consistent across the application. If the request body or response type is JSON then please follow camelCase to maintain the consistency.

## 6. Searching, sorting, filtering and pagination



- Sorting In case, the client wants to get the sorted list of companies, the GET /companies endpoint should accept multiple sort params in the query.

E.g

GET /companies?sort=rank\_asc

would sort the companies by its rank in ascending order.

# Create simple restful web service using java



- Filtering For filtering the dataset, we can pass various options through query params.

E.g

GET /companies?category=banking&location=India

would filter the companies list data with the company category of Banking and where the location is India.

# Create simple restful web service using java



Searching When searching the company name in companies list the API endpoint should be

GET /companies?search=Digital

Pagination When the dataset is too large, we divide the data set into smaller chunks, which helps in improving the performance and is easier to handle the response.

E.g.

GET /companies?page=23

means get the list of companies on 23rd page.

## 7. Versioning

- When your services are being consumed by the world, upgrading the APIs with some breaking change would also lead to breaking the existing products or services using your services.
- <http://api.yourservice.com/v1/companies/34/employees> is a good example, which has the version number of the API in the path.
- If there is any major breaking update, we can name the new set of APIs as v2 or v1.x.x



# Unit III

## Chapter I

### Secure RESTful web services

# OAuth2

- OAuth is an open authentication and authorization protocol, which allows accessing the resources of the resource owner by enabling the client applications on HTTP services.
- Such as Facebook, GitHub, Google etc. It allows sharing of resources stored on one site to another site without using their credentials. It uses username and password tokens instead.
- It is token based authentications version 2.0



GitHub





# Why Use Oauth 2.0?

- You can use OAuth 2.0 to read data of a user from another application.
- It supplies the authorization workflow for web, desktop applications, and mobile devices.
- It is a server side web app that uses authorization code and does not interact with user credentials.

# Features of OAuth

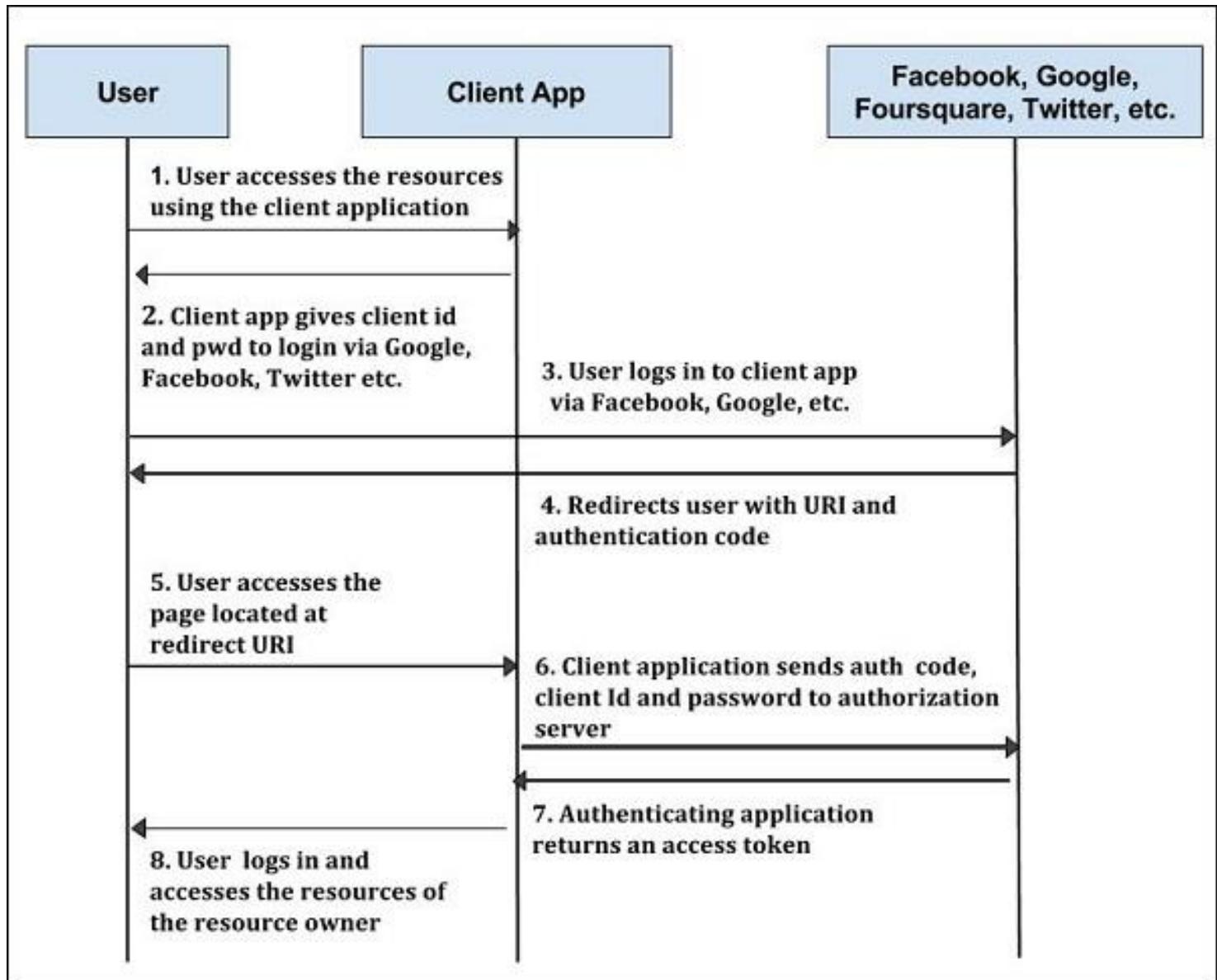


- OAuth 2.0 is a simple protocol that allows to access resources of the user without sharing passwords.
- It provides user agent flows for running clients application using a scripting language, such as JavaScript. Typically, a browser is a user agent.
- It accesses the data using tokens instead of using their credentials and stores data in online file system of the user such as Google Docs or Dropbox account.

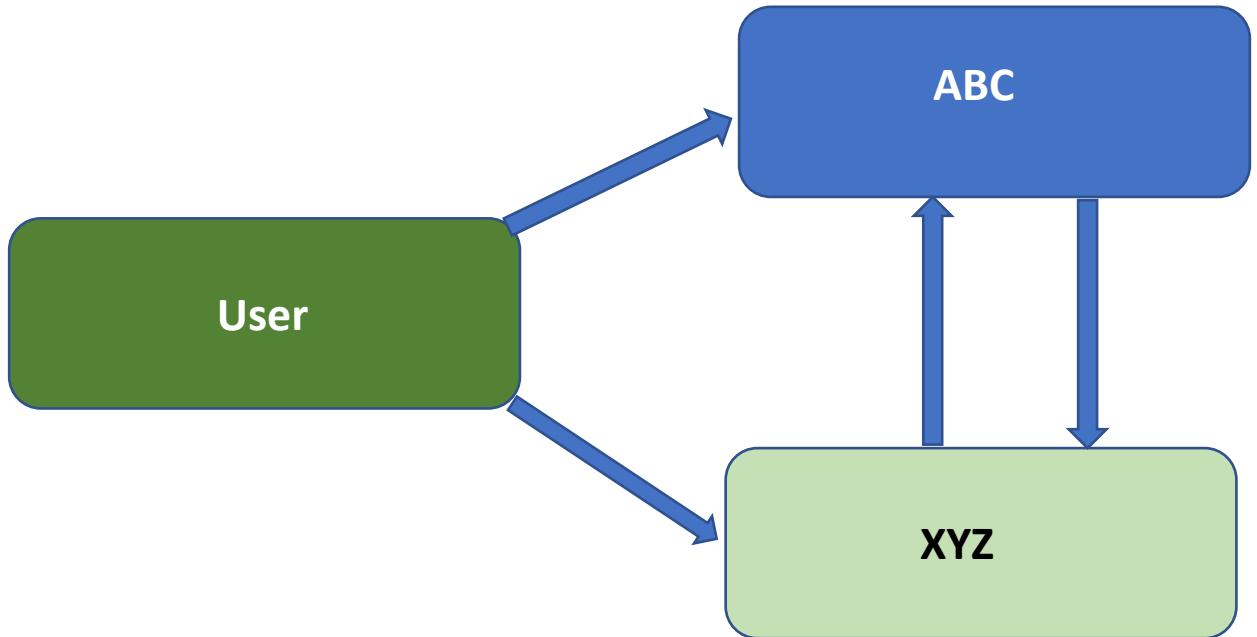
# Advantages of OAuth 2.0

- OAuth 2.0 is a very flexible protocol that relies on SSL (Secure Sockets Layer that ensures data between the web server and browsers remain private) to save user access token.
- OAuth 2.0 relies on SSL which is used to ensure cryptography industry protocols and are being used to keep the data safe.
- It allows limited access to the user's data
- It has ability to share data for users without having to release personal information.
- It is easier to implement and provides stronger authentication.

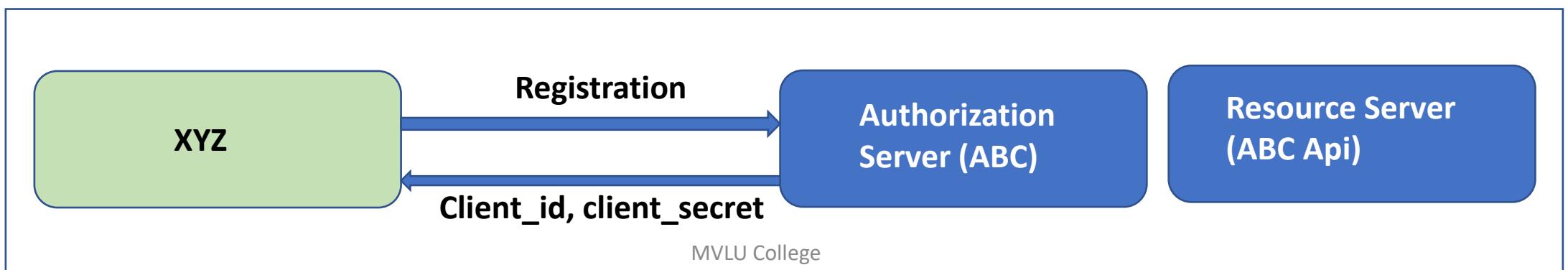
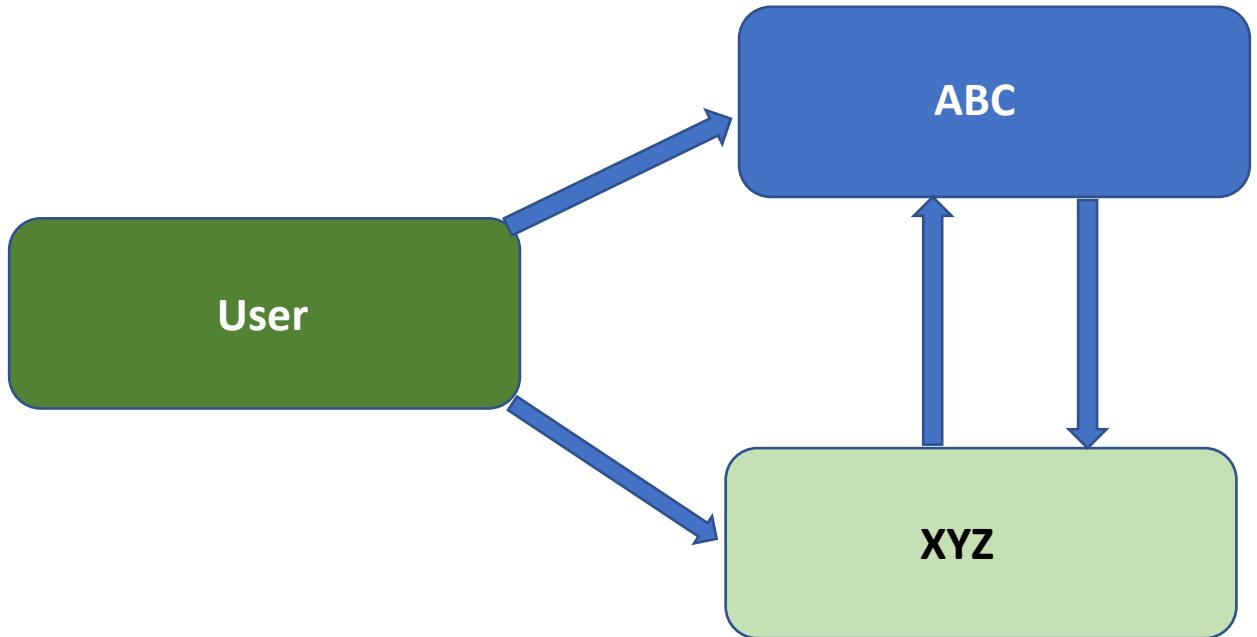
# Flow of OAuth



# How OAuth Works?



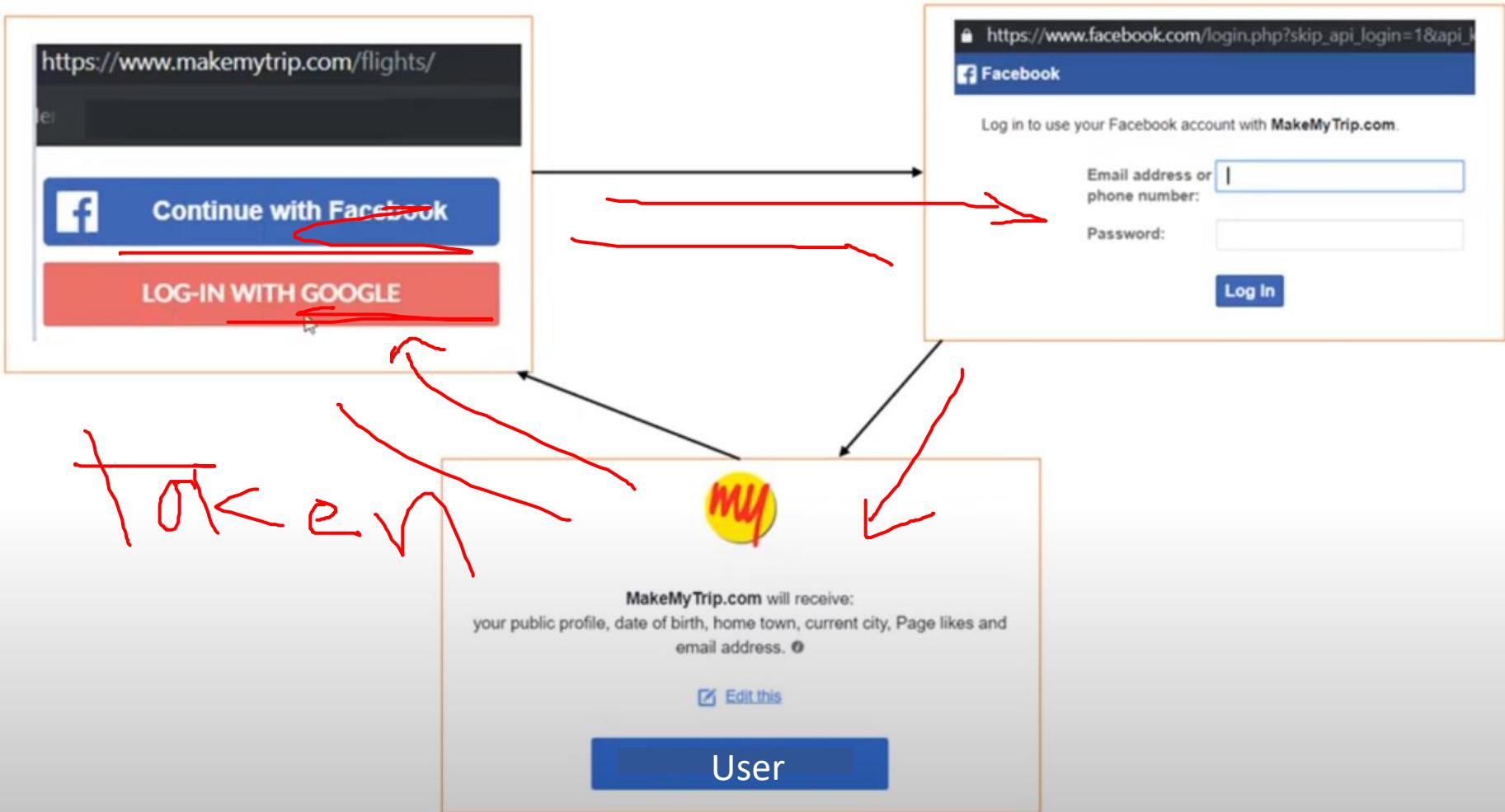
# How OAuth Works?



# Oauth example

OAuth2 Basics

## Login Using Facebook



# OAuth Roles

## Resource Server

- Protected API/ Service
- That need to be protected by OAuth.

## Authorization Server

- Authorizes app
- This is a server that is responsible for authentication and authorization the user and client app

## Resource Owner

- User (Account User)
- This is user of that resource

## Client

- Third Party Application
- That want to access protected resource by using Oauth.

# Application Registration



- Before using OAuth with your application, you must register your application with the service. This is done through a registration form in the “developer” or “API” portion of the service’s website, where you will provide the following information (and probably details about your application):
  - Application Name
  - Application Website
  - Redirect URI or Callback URL
  - The redirect URI is where the service will redirect the user after they authorize (or deny) your application, and therefore the part of your application that will handle authorization codes or access tokens.

# Client ID and Client Secret



- Once your application is registered, the service will issue “client credentials” in the form of a *client identifier* and a *client secret*. The Client ID is a publicly exposed string that is used by the service API to identify the application, and is also used to build authorization URLs that are presented to users. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user’s account, and must be kept private between the application and the API.

# Authorization Grant



- the first four steps cover obtaining an authorization grant and access token. The authorization grant type depends on the method used by the application to request authorization, and the grant types supported by the API. OAuth 2 defines four grant types, each of which is useful in different cases:
- **Authorization Code**: used with server-side Applications
- **Implicit**: used with Mobile Apps or Web Applications (applications that run on the user's device)
- **Resource Owner Password Credentials**: used with trusted Applications, such as those owned by the service itself
- **Client Credentials**: used with Applications API access



# Windows Communication Foundation

# Introduction to Windows Communication Foundation



- WCF stands for Windows Communication Foundation.
- WCF is used for building the distributed and interoperable applications.
- WCF is Microsoft platform a part of .NET 3.0 Framework.
- Unified programming model for building service-oriented applications.
- It enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing.

# WCF features

- WCF combines the features of all the distributed technologies, such as:
  1. COM+ Services
  2. .NET Remoting
  3. Web Services



# Distributed:

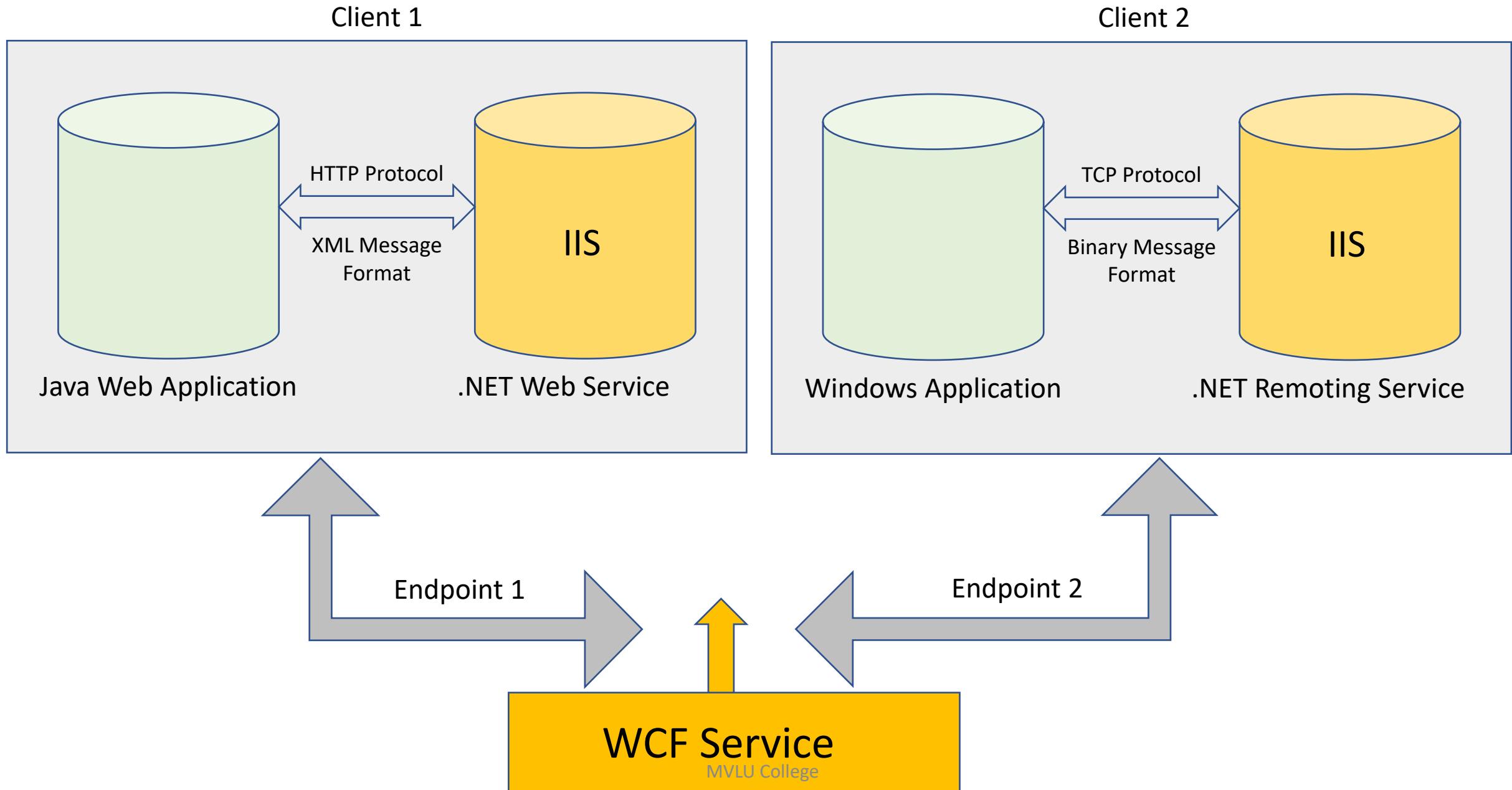
- Each computer in the network runs one (or more) of these components which perform different tasks.
  - That means WCF supports different protocol
  - HTTP protocol
  - TCP protocol
- 
- So based on type of request, it will perform one or more tasks.



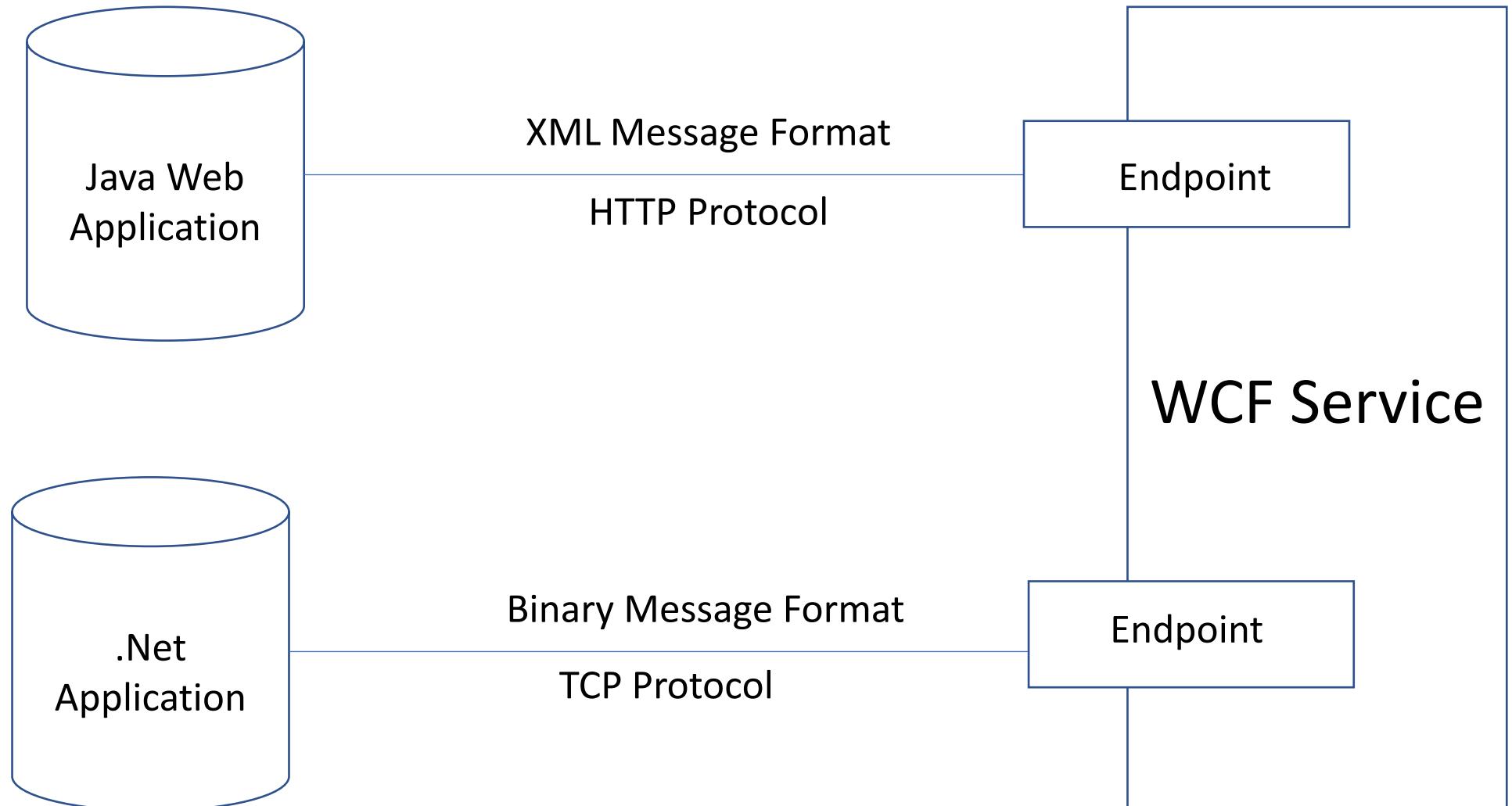
# Interoperable

- An application that can communicate with any other application built on any platform.

# Why WCF Services?



# With WCF



# WCF supported Protocols are:

- HTTP (Hypertext Transfer Protocol)
- TCP (Transmission Control Protocol)
- MSMQ (Message Queuing)
- Named Pipes

# Introduction to Windows Communication Foundation



- 1.WCF is interoperable with other services when compared to .Net Remoting where the client and service have to be .Net.
- 2.WCF services provide better reliability and security in compared to ASMX web services.
- 3.In WCF, there is no need to make much change in code for implementing the security model and changing the binding. Small changes in the configuration will make your requirements.
- 4.WCF has integrated logging mechanism, changing the configuration file settings will provide this functionality. In other technology developer has to write the code.

# Introduction to Windows Communication Foundation



- .NET provides various options for creating services under .NET Framework. Before WCF, ASP.NET Web Services are used to create service in .NET Framework. Now WCF is the latest programming model for building and developing the service-oriented application.
- Webservice is language independent and Web Services communicate by using standard web protocols and data formats, such as - Http, XML, Soap.
- WCF allows applications to communicate with each other in the distributed environment. WCF is a set of technologies that covers ASMX web services, Web Services Enhancements (WSE), .NET Remoting and MSMQ.

# Differences between WCF and ASP.NET Web Service



WCF	ASP.NET Web Service
ServiceContract and OperationContract attributes are used for defining WCF service.	WebService and WebMethod attributes are used for defining web service.
Supports various protocols like HTTP, HTTPS, TCP, Named Pipes and MSMQ.	Supports only HTTP, HTTPS protocols.
Hosted in IIS, WAS (Windows Activation Service), Self-hosting, Windows Service.	Hosted only in IIS.
Supports security, reliable messaging, transaction and AJAX and REST supports.	Support security but is less secure as compared to WCF.
Supports DataContract serializer by using System.Runtime.Serialization.	Supports XML serializer by using System.Xml.Serialization.

# Differences between WCF and ASP.NET Web Service



WCF	ASP.NET Web Service
Supports One-Way, Request-Response and Duplex service operations.	Supports One-Way and Request-Response service operations.
WCF are faster than Web Services.	Web Services are slower than WCF.
Unhandled Exceptions does not return to the client as SOAP faults. WCF supports better exception handling by using FaultContract.	Unhandled Exceptions returns to the client as SOAP faults.
Supports XML, MTOM, Binary message encoding.	Supports XML and MTOM (Message Transmission Optimization Mechanism) message encoding.
Supports multi-threading by using ServiceBehaviour class.	Doesn't support multi-threading.

# Differences between WCF and ASP.NET Web Service



WCF	ASP.NET Web Service
Protocols-Security, Reliable Messaging , Transactions	Protocols-Security.
WCF is more secure	Web Service required addition things to make it secure (Authentication, SSL etc.)

# Introduction to Windows Communication Foundation



- Windows communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous message from one service endpoint to another.
- A Service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application.

# Introduction to Windows Communication Foundation



- WCF supported protocol bindings :

# Message



## Message

A message is the communication unit in the form of an envelope for the transmission of the data from the client to Service and Service to the client and has two sections: Header, Body.

- Endpoint describes the address of Web Service from where a user can receive and send the message by specifying the communication mechanism of how the message will be sent or received.
- It consists of three things (A, B, and C) with question marks:
  - Address (Where?)
  - Binding (how?)
  - Contract (What?)
- **Endpoint = A + B+ C**

# Address

- locates the WCF Service, where the Service is hosted? With the exact URL of Web Service as given-
- Scheme://domain[:port]/path  
net.tcp://localhost:1234/MyService  
http://localhost:1234/MyService

# Binding



- Binding is the mechanism by which the user communicates with Web Service. It is the composition of some elements that bind together and creates the structure of communication such as some transport protocols like HTTP, TCP, etc. including message format or security techniques also.
- It specifies how the client will communicate to the service. We have different protocols (like http,tcp,named pipe,msmq) for the WCF to communicate to the client.

# contract

- A **contract** is the interface name which has all the operation need to be exposed as it specifies what functionality and operation are being provided by the service. It also specifies what functionality and operation need to be exposed to the client.
- It specifies what the service will do. For this purpose, we have a different contract like as Data Contract, Operation Contract, Message Contract, Fault Contract.

# Hosting

- A WCF service is a component that can be called by other applications. It must be hosted in an environment in order to be discovered and used by others. The WCF host is an application that controls the lifetime of the service. With .NET 3.0 and beyond, there are several ways to host the service.
- IIS Hosting
- Self-Hosting
- WAS Hosting

# Self Hosting



- A WCF service can be self-hosted, which means that the service runs as a standalone application and controls its own lifetime. This is the most flexible and easiest way of hosting a WCF service, but its availability and features are limited.

- A WCF service can also be hosted as a Windows service. A Windows service is a process managed by the operating system and it is automatically started when Windows is started (if it is configured to do so). However, it lacks some critical features (such as versioning) for WCF services.

# IIS hosting



- A better way of hosting a WCF service is to use IIS. This is the traditional way of hosting a web service. IIS, by nature, has many useful features, such as process recycling, idle shutdown, process health monitoring, message-based activation, high availability, easy manageability, versioning, and deployment scenarios. All of these features are required for enterprise-level WCF services.

- The IIS hosting method, however, comes with several limitations in the service-orientation world; the dependency on HTTP is the main culprit. With IIS hosting, many of WCF's flexible options can't be utilized. This is the reason why Microsoft specifically developed a new method, called Windows Activation Services, to host WCF services.
- Windows Process Activation Service (WAS) is the new process activation mechanism for Windows Server 2008 that is also available on Windows Vista. WAS hosting is possible only with IIS 7.0. Additional WCF components also plug into WAS to provide message-based activation over the other protocols that WCF supports, such as TCP, MSMQ, and named pipes. This allows applications that use the non-HTTP communication protocols to use the IIS features such as process recycling, rapid fail protection, and the common configuration systems that were only available to HTTP-based applications.

# Understanding different types of WCF Contracts



- WCF contract specify the service and its operations. WCF has five types of contracts: service contract, operation contract, data contract, message contract and fault contract.

# Service Contract



A service contract defines the operations which are exposed by the service to the outside world. A service contract is the interface of the WCF service and it tells the outside world what the service can do. It may have service-level settings, such as the name of the service and namespace for the service.

Eg.

```
[ServiceContract]
interface IMyContract
{
    [OperationContract]
    string MyMethod();
}

class MyService : IMyContract
{
    public string MyMethod()
    {
        return "Hello World";
    }
}
```

# Operation Contract



- An operation contract is defined within a service contract. It defines the parameters and return type of an operation. An operation contract can also define operation-level settings, like as the transaction flow of the operation, the directions of the operation (one-way, two-way, or both ways), and fault contract of the operation.



e.g.

```
[ServiceContract]
interface IMyContract
{
    [FaultContract(typeof(MyFaultContract))]
    [OperationContract]
    string MyMethod();
}
```

# Data contract



- A data contract defines the data type of the information that will be exchanged between the client and the service. A data contract can be used by an operation contract as a parameter or return type, or it can be used by a message contract to define elements.

e.g.

```
[DataContract]
```

```
class Person
```

```
{
```

```
[DataMember]
```

```
public string ID;
```

```
[DataMember]
```

```
public string Name;
```

```
}
```

```
[ServiceContract]
```

```
interface IMyContract
```

```
{
```

```
[OperationContract]
```

```
Person GetPerson(int ID);
```

```
}
```

# Message Contract



- When an operation contract required to pass a message as a parameter or return value as a message, the type of this message will be defined as message contract. A message contract defines the elements of the message (like as Message Header, Message Body), as well as the message-related settings, such as the level of message security.
- Message contracts give you complete control over the content of the SOAP header, as well as the structure of the SOAP body.

# e.g.

- [OperationContract]
- double CalPrice(PriceCalculate request);
- }
  
- [MessageContract]
- public class PriceCalculate
- {
- [MessageHeader]
- public MyHeader SoapHeader { get; set; }
- [MessageBodyMember]
- public PriceCal PriceCalculation { get; set; }
- }
  
- [DataContract]
- public class MyHeader
- {
- [DataMember]
- public string UserID { get; set; }
- }
  
- [DataContract]
- public class PriceCal
- {
- [DataMember]
- public DateTime PickupDateTime { get; set; }
- [DataMember]
- public DateTime ReturnDateTime { get; set; }
- [DataMember]
- public string PickupLocation { get; set; }
- [DataMember]
- public string ReturnLocation { get; set; }

# Fault contract

- A fault contract defines errors raised by the service, and how the service handles and propagates errors to its clients. An operation contract can have zero or more fault contracts associated with it.

e.g.

```
[ServiceContract]
interface IMyContract
{
    [FaultContract(typeof(MyFaultContract1))]
    [FaultContract(typeof(MyFaultContract2))]
    [OperationContract]
    string MyMethod();

    [OperationContract]
    string MyShow();
}
```

# Unit III

# Chapter III

## Fundamental Windows Communication Foundation Concepts

# Fundamental Windows Communication Foundation Concepts



- WCF is a runtime and a set of APIs for creating systems that send messages between services and clients. The same infrastructure and APIs are used to create applications that communicate with other applications on the same computer system or on a system that resides in another company and is accessed over the Internet.

# Messaging and Endpoints

- WCF is based on the notion of message-based communication, and anything that can be modelled as a message (for example, an HTTP request or a Message Queuing (also known as MSMQ) message) can be represented in a uniform way in the programming model. This enables a unified API across different transport mechanisms.

# Messaging and Endpoints...

- Messages are sent between endpoints. *Endpoints* are places where messages are sent or received (or both), and they define all the information required for the message exchange. A service exposes one or more application endpoints (as well as zero or more infrastructure endpoints), and the client generates an endpoint that is compatible with one of the service's endpoints.
- An *endpoint* describes in a standard-based way where messages should be sent, how they should be sent, and what the messages should look like. A service can expose this information as metadata that clients can process to generate appropriate WCF clients and communication *stacks*.

# Communication Protocols

- One required element of the communication stack is the *transport protocol*. Messages can be sent over intranets and the Internet using common transports, such as HTTP and TCP. Other transports are included that support communication with Message Queuing applications and nodes on a Peer Networking mesh. More transport mechanisms can be added using the built-in extension points of WCF.
- Text encoding, an interoperable encoding.
- Message Transmission Optimization Mechanism (MTOM) encoding, which is an interoperable way for efficiently sending unstructured binary data to and from a service.
- Binary encoding for efficient transfer.

# Message Patterns

- WCF supports several messaging patterns, including request-reply, one-way, and duplex communication. Different transports support different messaging patterns, and thus affect the types of interactions that they support. The WCF APIs and runtime also help you to send messages securely and reliably.

# Binding

- Defines how an endpoint communicates to the world. It is constructed of a set of components called binding elements that "stack" one on top of the other to create the communication infrastructure. At the very least, a binding defines the transport (such as HTTP or TCP) and the encoding being used (such as text or binary). A binding can contain binding elements that specify details like the security mechanisms used to secure messages, or the message pattern used by an endpoint.

# Hosting

- A service must be hosted in some process. A *host* is an application that controls the lifetime of the service. Services can be self-hosted or managed by an existing hosting process.

# Metadata

- In a service, describes the characteristics of the service that an external entity needs to understand to communicate with the service. Metadata can be consumed by the [Service Model Metadata Utility Tool \(Svcutil.exe\)](#) to generate a WCF client and accompanying configuration that a client application can use to interact with the service.
- The metadata exposed by the service includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service.
- When enabled, metadata for the service is automatically generated by WCF by inspecting the service and its endpoints. To publish metadata from a service, you must explicitly enable the metadata behaviour.

- A client-application construct that exposes the service operations as methods (in the .NET Framework programming language of your choice, such as Visual Basic or Visual C#). Any application can host a WCF client, including an application that hosts a service. Therefore, it is possible to create a service that includes WCF clients of other services.
- A WCF client can be automatically generated by using the [Service Model Metadata Utility Tool \(Svcutil.exe\)](#) and pointing it at a running service that publishes metadata.

# Channel

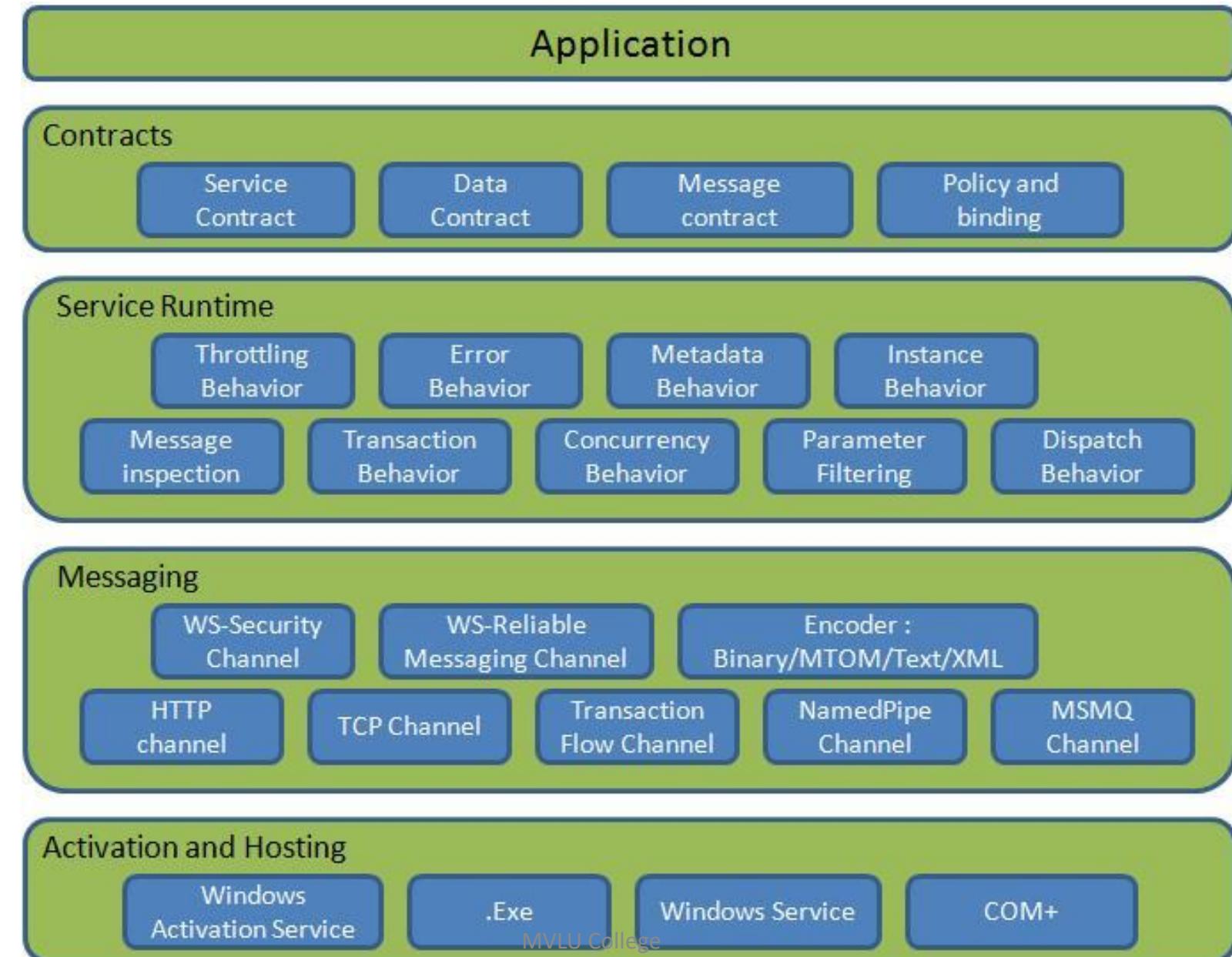
- A concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration. Therefore, there is a channel associated with each binding element. Channels stack on top of each other to create the concrete implementation of the binding: the channel stack.

# Unit III

## Chapter III

# Windows Communication Foundation Architecture

# WCF Architecture





# Contracts

- Contracts layer are next to that of Application layer. Developer will directly use this contract to develop the service.

# Service Contracts



- Describe about the operation that service can provide. Example, Service provided to know the temperature of the city based on the zip code, this service we call as Service contract. It will be created using Service and Operational Contract attribute.

# Data contract



- It describes the custom data type which is exposed to the client. This defines the data types, are passed to and from service. Data types like int, string are identified by the client because it is already mentioned in XML schema definition language document, but custom created class or datatype cannot be identified by the client e.g. Employee data type. By using DataContract we can make client aware that we are using Employee data type for returning or passing parameter to the method.

# Message Contract



- Default SOAP message format is provided by the WCF runtime for communication between Client and service. If it is not meeting your requirements then we can create our own message format. This can be achieved by using Message Contract attribute.

# Policies and Binding



- Specify conditions required to communicate with a service e.g security requirement to communicate with service, protocol and encoding used for binding.

# Service Runtime



- It contains the behaviors that occur during runtime of service.
- Throttling Behavior- Controls how many messages are processed.
- Error Behavior - Specifies what occurs, when internal error occurs on the service.
- Metadata Behavior - Tells how and whether metadata is available to outside world.
- Instance Behavior - Specifies how many instance of the service has to be created while running.
- Transaction Behavior - Enables the rollback of transacted operations if a failure occurs.
- Dispatch Behavior - Controls how a message is processed by the WCF Infrastructure.

# Messaging



- Messaging layer is composed of channels. A channel is a component that processes a message in some way, for example, by authenticating a message. A set of channels is also known as a channel stack. Channels are the core abstraction for sending message to and receiving message from an Endpoint. Broadly we can categories channels as
  - Transport ChannelsHandles sending and receiving message from network. Protocols like HTTP, TCP, name pipes and MSMQ.
  - Protocol ChannelsImplements SOAP based protocol by processing and possibly modifying message. E.g. WS-Security and WS-Reliability

# Activation and Hosting



Services can be hosted or executed, so that it will be available to everyone accessing from the client. WCF service can be hosted by following mechanism

- IIS Internet information Service provides number of advantages if a Service uses Http as protocol. It does not require Host code to activate the service, it automatically activates service code.

# Windows Activation Service



- Windows Activation Service(WAS) is the new process activation mechanism that ships with IIS 7.0. In addition to HTTP based communication, WCF can also use WAS to provide message-based activation over other protocols, such as TCP and named pipes.



# Self-Hosing

- WCF service can be self hosted as console application, Win Forms or WPF application with graphical UI.



# Window Service

- WCF can also be hosted as a Windows Service, so that it is under control of the Service Control Manager (SCM).

# Unit III Questions

- Explain WCF in details
- What are the features of WCF
- What is a service contract in WCF?
- Explain advantages and disadvantages of WCF
- Name some different types of contracts in WCF. Explain any two contract in details
- What are the differences of Web Services and WCF?
- What are the fundamental Concept of WCF?
- Explain basic terms related to the WCF?
- With the neat labelled diagram explain the Architecture of WCF.
- What is Windows Communication Foundation (WCF)

# Unit III Questions

- What is Message Contract in WCF?
- What is an Operation Contract in WCF?
- What are the differences between ASP.NET Web Service and WCF?
- What is Self Hosting in WCF?
- Explain the three main components of WCF.

- .NET Framework Client Profile is a lightweight version of the full .NET Framework designed for clients that don't need the entire framework. Not all of Windows Communication Foundation is supported by the client framework

# Why .NET Client Profile

- There are many types of applications and hence what is required by one application may not be necessarily required by other applications. For example System.Web is only used by ASP.NET whereas System.Web is of no use for Windows Forms application.

Now if you think from a deployment point of view, you would want your installer package to be as optimized and small as possible, so it requires a minimum of what is required on the client machine. This can be done by the Client Profile flavour of .NET 3.5 and 4.0.

# Why .NET Client Profile

- Project Templates that Target the .NET Framework Client Profile
- Several project templates in Visual Studio 2010 target the .NET Framework 4 Client Profile. The following is a list of the project templates in Visual Studio 2010 that target the .NET Framework 4 Client Profile by default. All other projects target the .NET Framework 4 by default.
- Windows
  - WPF Application
  - WPF Browser Application
  - WPF Custom Control Library
  - WPF User Control Library
  - Windows Forms Application
  - Windows Forms Control Library
  - Console Application
  - Empty Project
  - Window Service



# Why .NET Client Profile

- Office
  - All Office 2007 and Office 2010
  - project templates



# Why .NET Client Profile

- WCF
  - WCF Service Library

# Web Service QoS

The dynamic e-business vision calls for a seamless integration of business processes, applications, and Web services over the Internet. Delivering QoS on the Internet is a critical and significant challenge because of its dynamic and unpredictable nature. Applications with very different characteristics and requirements compete for scarce network resources. Changes in traffic patterns, denial-of-service attacks and the effects of infrastructure failures, low performance of Web protocols, and security issues over the Web create a need for Internet QoS standards.

- Any service has both functional and non-functional requirements.
- Functional requirements is easy to understand because it's a direct expectation of client.
- But there some indirect expectation or non-functional requirements, known as a Quality of Service(QoS), that should also be fulfilled.

# Web Service QoS

QoS covers a whole range of techniques that match the needs of service requestors with those of the service provider's based on the network resources available. By QoS, we refer to non-functional properties of Web services such as performance, reliability, availability, and security.

# Web service QoS requirements



**Availability:** Availability is the quality aspect of whether the Web service is present or ready for immediate use. Larger values represent that the service is always ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time. Also associated with availability is time-to-repair (TTR). TTR represents the time it takes to repair a service that has failed. Ideally smaller values of TTR are desirable.



# Accessibility

The accessibility means the degree with which the web service request is served. It can be expressed as a probability measure showing the success rate or the chance of a successful service instantiation at a point in time. A high degree of accessibility means that a service is available for a large number of clients and those clients can utilize the service easily.



# Integrity

Integrity specifies the degree with which a Web service perform its functions according to its WSDL description as well as compliance with Service-Level Agreement (SLA). A higher degree of integrity it nothing but the functionality of a service closer to its WSDL description or SLA.

# Performance

The performance is measured on the basis of two factors. Throughput means the number of Web services requests served at a given period of time. Latency means the total time between sending a request and receiving the response. To achieve good performance of a Web Services , the higher throughput and lower latency value is required.



# Reliability

Reliability is the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service. In another sense, reliability refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers.

# Security

Security consists of aspects like authentication, authorization, confidentiality and message integrity. Security is an important thing because Web service invocation occurs via the Internet. The requirements of amount of security for a particular Web Service is specified in its SLA and service providers should maintain this level of security.

# Synchronous Web Services:

- In Synchronous call, if you are making any request, then you will have to wait till the response, you can't do any other thing until you will not get the response.
- Supports both SOAP and REST Web Services.
- Imagine yourself waiting at a ticket counter. You ask register guy for the ticket and wait on the window until he gives you the ticket.



# Asynchronous Web Services:

- In Asynchronous call, if you are making any request, then you don't need to wait for the response and you can perform any other task.
- Supports only SOAP Web Services.

# WCF Transaction

- A transaction in WCF is a set of operations that follow some properties, collectively known as ACID. Here, if a single operation fails, the entire system fails automatically. When an order is placed online, a transaction takes place.

# Example

- Suppose you have ordered an LCD television from an online store and you are going to pay the amount by your credit card. When you enter the requisite information to place the order, two operations occur simultaneously.
- One, the specified amount gets debited from your bank account and second, the vendor account is credited with the same amount. Both the operations must execute successfully in order to have a successful transaction.

- **Atomic** – All the operations must act as a single indivisible operation at the completion of a transaction.
- **Consistency** – Whatever may be the operation set, the system is always in a state of consistency, i.e., the outcome of the transaction is always as per the expectation.
- **Isolation** – The intermediary state of system is not visible to any entities of the outer world till the transaction is completed.
- **Durability** – Committed state is maintained regardless of any kind of failure (hardware, power outage, etc.)

- While configuring a WCF transaction, there are some factors that demand consideration. These are binding and operation behaviour.

**Binding** – The bindings that support transaction in WCF are only a few and it is vital to make a choice from only these bindings, which remain disabled by default and should be enabled to get the requisite support for transaction. These bindings are as follows –

- NetTcpBinding
- NetNamedPipeBinding
- WSHttpBinding
- WSDualHttpBinding
- WSFederationHttpBinding

# Operation Behaviour

- Operation Behaviour- While a binding facilitates the path for transaction propagation, an operation takes care of transaction processing as well as operation configuration. Operation behaviour primarily uses two attributes: TransactionFlow and TransactionScopeRequired. Here, it should be noted that TransactionFlow has mainly three values and these are: Allowed, Mandatory, and NotAllowed.
- In Allowed option there is a chance that whether transaction will work or not. In NotAllowed option it means transaction will not work and in Mandatory option transaction will work.

- The following code shows whether or not changing the configuration of binding and operation contract facilitates the propagation of client.

```
<bindings>
    <wsHttpBinding>
        <binding name = "MandatoryTransBinding" transactionFlow = "true">
            <reliableSession enabled ="true"/>
        </binding>
    </wsHttpBinding>
</bindings>
```

# Unit IV Questions

- Explain WCF and .Net Framework Client Profile in detailed.
- Explain different task that are required to build a simple WCF application.
- Define quality of service for Web services.
- Write a short note on WCF Transaction?
- Explain Asynchronous Web Services: