Question :- Define the term web service. Explain two types of web service

Web services are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP). As described by the World Wide Web Consortium (W3C), web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions, thanks to the use of XML. Web services can be combined in a loosely coupled way to achieve complex operations. Programs providing simple services can interact with each other to deliver sophisticated added-value services.

Types of Web Services

On a technical level, web services can be implemented in various ways. The two types of web services discussed in this section can be distinguished as "big" web services and "RESTful" web services.

Big Web Services

In Java EE 6, JAX-WS provides the functionality for "big" web services. Big web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats. Such systems often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

The SOAP message format and the WSDL interface definition language have gained widespread adoption. Many development tools, such as NetBeans IDE, can reduce the complexity of developing web service applications.

A SOAP-based design must include the following elements.

> A formal contract must be established to describe the interface that the web service offers. WSDL can be used to describe the details of the contract, which may include messages, operations, bindings, and the location of the web service. You may also process SOAP messages in a JAX-WS service without publishing a WSDL.
>
> The architecture must address complex nonfunctional requirements. Many web service specifications address such requirements and establish a common vocabulary for them. Examples include transactions, security, addressing, trust, coordination, and so on.
>
> The architecture needs to handle asynchronous processing and invocation. In such cases, the infrastructure provided by standards, such as Web Services Reliable Messaging (WSRM), and APIs, such as JAX-WS, with their client-side asynchronous invocation support, can be leveraged out of the box.

RESTful Web Services

In Java EE 6, JAX-RS provides the functionality for Representational State Transfer (RESTful) web services. REST is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service–API definitions.

Because RESTful web services use existing well-known W3C and Internet Engineering Task Force (IETF) standards (HTTP, XML, URI, MIME) and have a lightweight infrastructure that allows services to be built with minimal tooling, developing RESTful web services is inexpensive and thus has a very low barrier for adoption. You can use a development tool such as NetBeans IDE to further reduce the complexity of developing RESTful web services.

A RESTful design may be appropriate when the following conditions are met.

The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.

A caching infrastructure can be leveraged for performance. If the data that the web service returns is not dynamically generated and can be cached, the caching infrastructure that web servers and other intermediaries inherently provide can be leveraged to improve performance. However, the developer must take care because such caches are limited to the HTTP GET method for most servers.

The service producer and service consumer have a mutual understanding of the context and content being passed along. Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it meaningfully. In the real world, most commercial applications that expose services as RESTful implementations also distribute so-called value-added toolkits that describe the interfaces to developers in popular programming languages.

Bandwidth is particularly important and needs to be limited. REST is particularly useful for limited-profile devices, such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.

Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style. Developers can use such technologies as JAX-RS and Asynchronous JavaScript with XML (AJAX) and such toolkits as Direct Web Remoting (DWR) to consume the services in their web applications. Rather than starting from scratch, services can be exposed with XML and consumed by HTML pages without significantly refactoring the existing web site architecture. Existing developers will be more productive because they are adding to something they are already familiar with rather than having to start from scratch with new technology.

Q. What are the two type of informational Services?Explain

Informational services are services of relatively simple nature. They either provide access

to content interacting with an end user by means of simple request/response sequences, or alternatively may expose back-end business applications to other applications. Web services that typically expose the business functionality of the applications and components that underlie them are known as *programmatic services*. For instance, they may expose function calls, typically written in programming languages such as Java/EJB, Visual Basic, or C++. The exposed programmatic simple services perform a request/response type of business task and can be viewed as "atomic" (or singular) operations. Applications access these function calls by executing a Web service through a standard programmatic interface specified in the Web Services Description Language or WSDL (see Chapter 5).
Informational services can be subdivided into three subcategories according to the business problems they solve:
1. Pure *content services*, which give programmatic access to content such as weather report information, simple financial information, stock quote information, design information, news items, and so on.
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. Such service requests may have complicated realizations. Consider, for example, "pure" business services, such as logistic services, where automated services are the actual front-ends to fairly complex physical organizational information systems.
3. *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. Generally speaking, these services are offered by a third party and run the whole range from commerce-enabling services, such as logistics, payment, fulfillment, and tracking services, to other value-added commerce services, such as rating services. Typical examples of syndicated services might include reservation services on a travel site or rate quote services on an insurance site.
Informational services are singular in nature in that they perform a complete unit of work that leaves its underlying datastores in a consistent state. However, they are not transactional in nature (although their back-end realizations may be). An informational service does not keep any memory of what happens to it between requests. In that respect this type of service is known as a *stateless Web service*.

1)Explain in detail about Synchronicity?

Answer:-We may distinguish between two programming styles for services:

a)Synchronous or remote procedure call(RPC) style.

b)Asynchronous or message style.

a) Synchronous Services:

Clients of synchronous services express their request as a method call with a set of arguments, which returns a response containing a return value. This implies that when a client sends a request message, it expects a response message before continuing with its computation.This makes the whole invocation an all or nothing proposition. If one operation is unable to complete for any reason, all other dependent operation will fail. Because of this type of bilateral communication between the client and service provider. RPC style Web service are normally used when an application exhibits the following characteristics:-

a)The client invoking the service requires an immediate response.

b)The client and service work in a back and forth conversational way.

Examples of typical simple synchronous services with an RPC-style include reurning the current price for a given stock;providing the current weather conditions ina particular location; or checking the credit rating of a potential trading partner prior to the completion of a business transaction.

b) <u>Asynchronous services</u>:-

Asynchronous service are document style or message driven services. When a client invokes a message style services , the client typically sends it an entire document, such as a purchase order, rather than a discrete set of parameters. The service accepts the entire document, it processes it and may or may not return a result message. A client that involves an asynchronous service does not need to wait for a response before it continues with the remainder of its application. The response from the service, if any, can appear hours or even days later.

Asynchronous interactions(messaging) are akey design pattern in loosely coupled environments. Messaging enables a loosely coupled environments in which an application does not need to know the intimate details of how to reach and interface with other applications. This allows a communication operation between any two process to be a self contained, standalone unit of work. Document style web services are normally used when an application exhibits the following characteristics:

a)The client does not require an immediate response.

b)The service is document oriented (the client typically sends an entire document, e.g., a purchase order, rather discrete parameters).

Examples of document style Web services include processing a purchase order;

Responding to a request for quote order from a customer ; or responding to an order

Placement by a particular customer. In all these cases, the client sends an entire document, such as purchase order, to the web service and assumes that the Web service is processing it in some way, but the client does not require an immediate answer.

4) Explain the Client Service Model with respect to Web Service.

A widely applied form of distributed processing is client–server computing. 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. In short, a client–server architecture is a computational architecture in which processing and storage tasks are divided between two classes of network members, clients and servers. 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. In general, client–server computing does not emphasize hardware distinctions; it rather focuses on the applications themselves. The same device may function as both client and server. For example, there is no reason why a Web server – which contains large amounts of memory and disk space – cannot function as both client and server when local browser sessions are run there.

In a client–server architecture the client machine runs software and applications that are
stored locally. Some of the applications may be stored and executed on the server, but most of them are on the client. The server also provides the data for the application. In a client–server architecture, the client actually has two tasks. The client makes requests to servers and is also responsible for the user interface. For example, a Web browser software (which is a client process) not only requests documents from Web servers, but must also display those documents for the user to make requests by typing a URL or by clicking on links. The Web server must store all of the documents and must be able to respond to requests from clients.

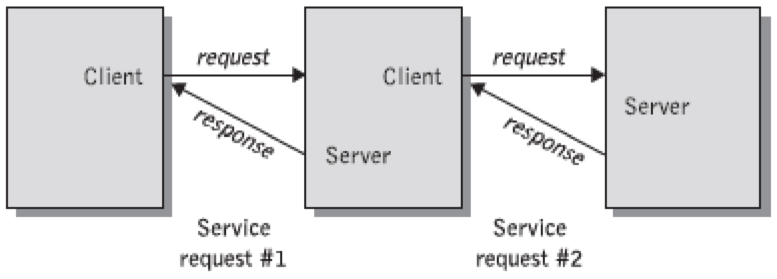


Figure 2.4 Client–server architecture

Question: What is SOAP ? Explain in detail.

Answer:

1. SOAP is known as the Simple Object Access Protocol

2. SOAP is an XML-based protocol for accessing web services over HTTP

3. Soap is a protocol which defines the rules of message passing between webservices or client applications

4.Soap work as a intermediate language between different webservices developing languages

5.SOAP gives standard specifications so as heteroginious programming languages can communicate with each other

6.SOAP was designed to work with XML over HTTP

7.Web services relay on SOAP for exchanging messages between computers regardless of their operating systems,programming environment,or object model framework

8.SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a means for transport

9.SOAP method is simply an HTTP request and response that complies with the SOAP encoding rules

10.SOAP can be defined as a lightweight wire protocol for exchanging structured and type information back and forth between disparate systems in a distributed environment such as internet or LAN

11.Lightweight protocol means that SOAP posses two fundamentals properties .It can send and receive HTTP transport protocol packets,and process XML messages.

ADVANTAGES:

a. Simplicity

b. Portability

c. Firewall

d. Open standards

e. Interoperability

f. Universal acceptance

g. Resilience to changes

DISADVANTAGES:

a.Statelessness

b.Serialization by value and not by reference

c. Slow

d.WSDL dependence

Que.10: Explain in detail the structure of SOAP Message ?

--> SOAP is a acronym for Simple Object Access Protocol.It is a xml based messaging protocol for exchanging information among computers. SOAP is an application of the XML specification.

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

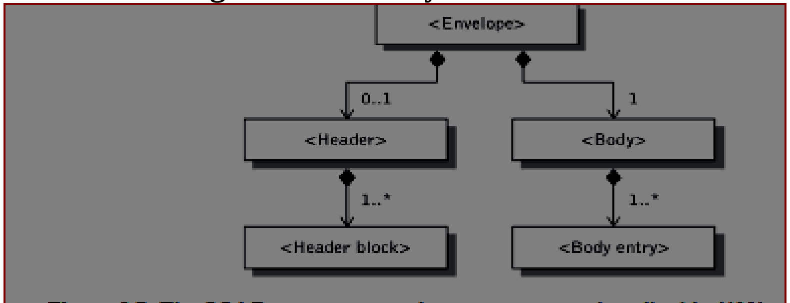
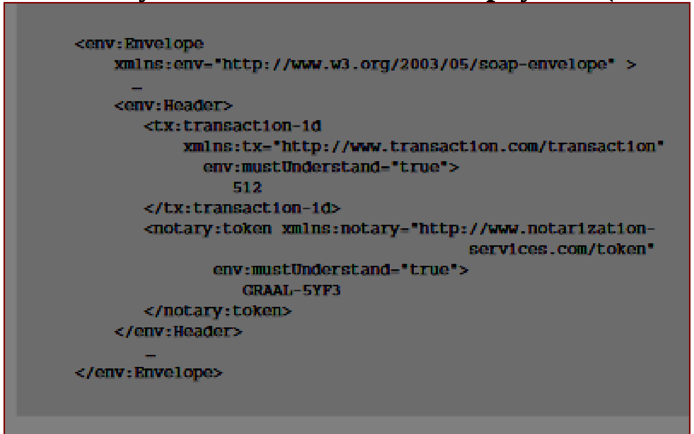


Figure 4.5 The SOAP message containment structure described in UML

i) <u>Envelope</u> : Defines the start and end of the message. It is a mandatory element.

ii) <u>Header</u> :
SOAP divides any message into two parts contained in a SOAP <Envelope:<Header>and <Body>. The <Envelope>may contain at most one <Header>child element, which contains all processing hints that are relevant for the endpoints or inter-mediate transport points. The <Header>may contain information about where the document shall be sent, where it originated, and may even carry digital signatures. This type of information must be separated from the SOAP <Body>which is mandatory and contains the SOAP payload (the XML document).



```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
      –
    <env:Header>
       <tx:transaction-id
           xmlns:tx="http://www.transaction.com/transaction"
             env:mustUnderstand="true">
                512
       </tx:transaction-id>
       <notary:token xmlns:notary="http://www.notarization-
                                         services.com/token"
              env:mustUnderstand="true">
                 GRAAL-5YF3
       </notary:token>
    </env:Header>
       –
</env:Envelope>
```

Listing 4.3 Example of a SOAP header

iii)<u>Body</u> : Contains the XML data comprising the message being sent. It is a mandatory element.The SOAP body is the area of the SOAP message where the application-specific XML data (payload) being exchanged in the message is placed. The <Body>element must be present and must be an immediate child of the envelope.

The <Body>element contains either the application-specific data or a fault messageApplication-specific data is the information that is exchanged with a Web service. It can be arbitrary XML data or parameters to a method call. The SOAP <Body>is where the method call information and its related arguments are encoded. It is where the response to a method call is placed, and where error information can be stored. The <Body>element has one distinguished root element, which is either the request or the response object. A fault message is used only when an error occurs. The

receiving node that discovers a problem, such as a processing error or a message that is improperly structured, sends it back to the sender just before it in the message path. A SOAP message may carry either application-specific data or a fault, but not both.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<m:order
xmlns:m="http://www.plastics_supply.com/purchase-order"
env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
env:mustUnderstand="true">
<m:order-no >uuid:0411a2daa</m:order-no>
<m:date>2004-11-8</m:date>
</m:order>
<n:customer xmlns:n="http://www.supply.com/customers"
env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
env:mustUnderstand="true">
<n:name> Marvin Sanders </n:name>
</n:customer >
</env:Header>
<env:Body>
<-- Payload element goes here -->
</env:Body>
        </env:Envelope>
```

iv) <u>Fault</u> : An optional Fault element that provides information about errors that occur while processing the message.

<u>SOAP Message Structure :</u>

-->The following block depicts the general structure of a SOAP message:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Header>
...
...
</SOAP-ENV:Header>
<SOAP-ENV:Body>
...
...
2Simple Object Access Protocol
<SOAP-ENV:Fault>
...
...
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

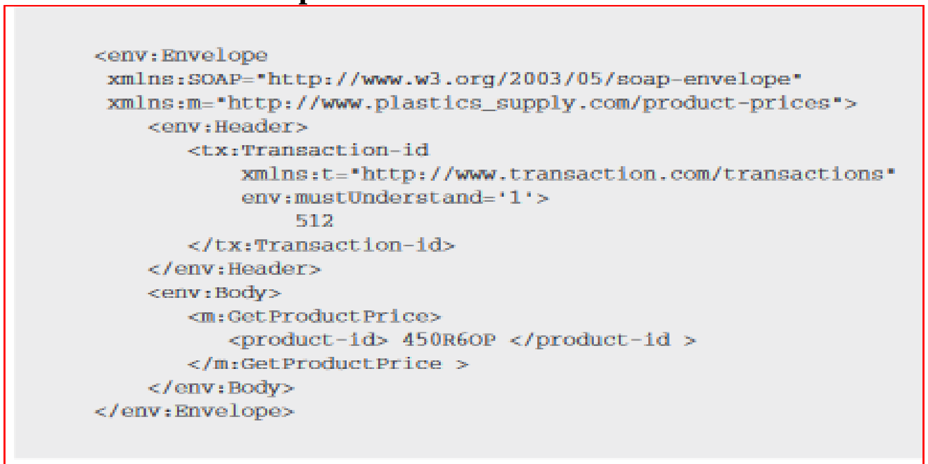
8). Explain in brief the following styles of SOAP services: A-RPC B-Document

→ The SOAP encoding style conveys information about how the contents of a particular element in the header blocks or the <Body> element of a SOAP message are encoded.

<u>A).RPC-style Web services:-</u>

→An RPC-style Web service appears as a remote object to a client application. The inter- action between a client and an RPC-style Web service centers around a service-specific interface.

→Clients express their request as a method call with a set of arguments, which returns a response containing a return value.RPC style supports automatic serialization/deserialization of messages, permitting developers to express a request as a method call with a set of parameters.(synchronous) model of communication between the client and service provider.



```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
           env:mustUnderstand='1'>
               512
       </tx:Transaction-id>
    </env:Header>
    <env:Body>
       <m:GetProductPrice>
          <product-id> 450R6OP </product-id >
       </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```

Listing 4.5 Example of an RPC-style SOAP body

→These components do not expli-citly exchange XML data; rather, they have methods with parameters and return values.The rules for packaging an RPC/Literal request in a SOAP envelope are quite simple:

◆A URI identifying the transport address for the call is required.

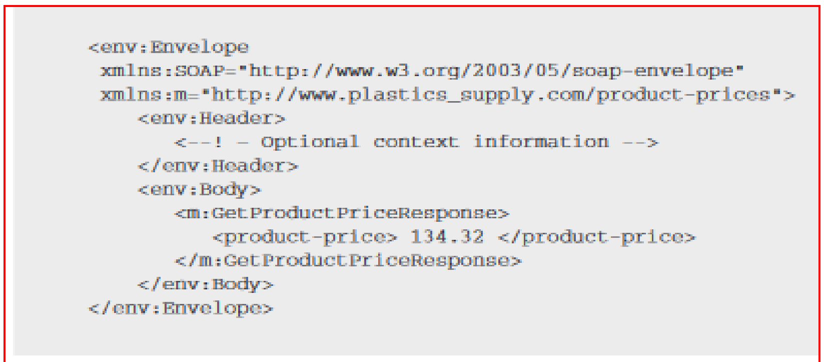
◆An RPC request message contains the method name and the input parameters of the call. The method call is always formatted as a single structure with each in or in–out parameter modeled as a field in that structure.

◆The names and the physical order of the parameters must correspond to the names and physical order of the parameters in the method being invoked.

◆An RPC response message contains the return value and any output parameters (or a fault). Method responses are similar to method calls in that the structure of theresponse is modeled as a single structure with a field for each parameter in the method signature.

→The method call requires the method name (GetProductPrice) and the parameters (product-id).

Once a SOAP message containing a call body has been sent, it is reasonable to expect that a response message will ensue.



```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <--! - Optional context information -->
    </env:Header>
    <env:Body>
       <m:GetProductPriceResponse>
          <product-price> 134.32 </product-price>
       </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

Listing 4.6 Example of a SOAP RPC response message

→A useful feature of the HTTP binding when RPC-style messages are transmitted is that it provides a way to automatically associate the request with the corresponding response.

<u>B). Document (message)-style Web services:-</u>

→SOAP can also be used for exchanging documents containing any kind of XML data. This enables complete reuse of code, from systems of any type, both within an enterprise and between business partners by fostering transparent integration to heterogeneous systems and infrastructures.

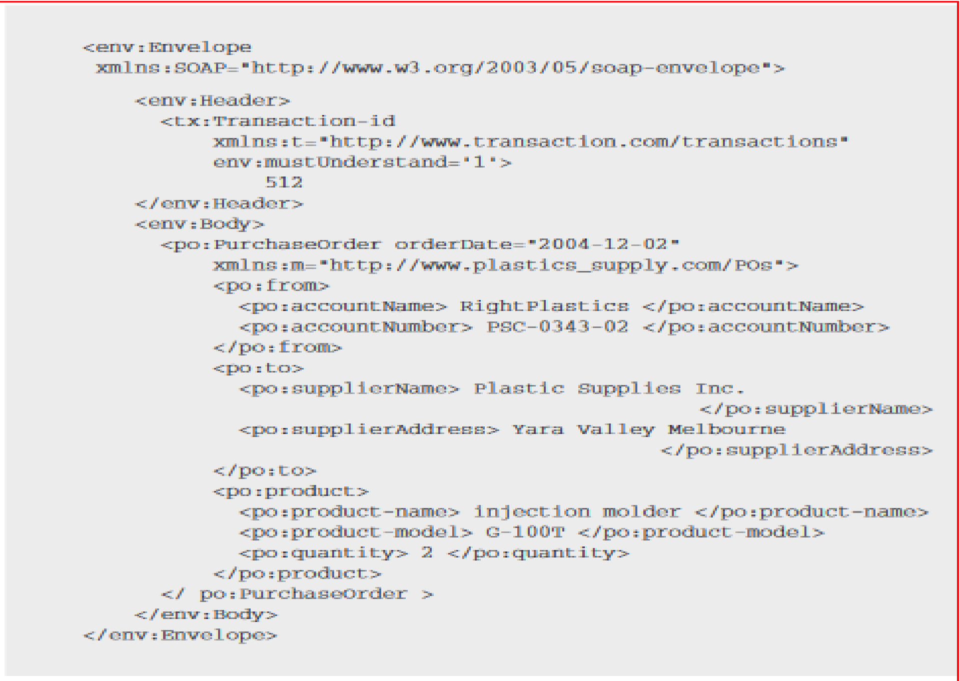
→Sending non-coded XML content in the body is known as document-style SOAP, since it focuses on the message as an XML document rather than an abstract data model that happens to be encoded into XML.

→Document-style Web services are message driven. When a client invokes a message-style Web service, the client typically sends it an entire document, such as a purchase order, rather than a discrete set of parameters.

→A set of XML elements that describes a purchase order, embedded within a SOAP message, is considered an XML document frag-ment.

→In the Document/Literal mode of messaging, a SOAP <Body>element contains an XML document fragment, a well-formed XML element that contains arbitrary application data that belongs to an XML schema and namespace separate from that of the SOAP message.

→The application aims to transfer the application data in one piece to a service provider for further processing. Under this scenario a document-style SOAP message carrying the entire application data as part of one concise, self-contained XML document (called purchase order) is the preffered option.



```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">

    <env:Header>
      <tx:Transaction-id
          xmlns:t="http://www.transaction.com/transactions"
          env:mustUnderstand='1'>
              512
    </env:Header>
    <env:Body>
      <po:PurchaseOrder orderDate="2004-12-02"
          xmlns:m="http://www.plastics_supply.com/POs">
          <po:from>
            <po:accountName> RightPlastics </po:accountName>
            <po:accountNumber> PSC-0343-02 </po:accountNumber>
          </po:from>
          <po:to>
            <po:supplierName> Plastic Supplies Inc.
                                       </po:supplierName>
            <po:supplierAddress> Yara Valley Melbourne
                                    </po:supplierAddress>
          </po:to>
          <po:product>
            <po:product-name> injection molder </po:product-name>
            <po:product-model> G-100T </po:product-model>
            <po:quantity> 2 </po:quantity>
          </po:product>
      </ po:PurchaseOrder >
    </env:Body>
</env:Envelope>
```

Listing 4.7 Example of a document-style SOAP body

Que.9: Write a short note on fault element in SOAP.

Ans.: (1) SOAP provides a model for handling situations when faults arise in the processing of a message. The SOAP <env:Fault> element carries error and status information in the SOAP message. If a Fault element is present, it must appear as a child element of the Body element.Because The SOAP <env:Body> element has another distinguishing role in that it is the place where fault information is placed. A Fault element can only appear once in a SOAP message.

(2) The SOAP fault model requires that all SOAP-specific and application-specific faults be reported using a special-purpose element called env:Fault. The env:Fault element is a reserved element predefined by the SOAP specification whose purpose is to provide an extensible mechanism for transporting structured and unstructured information about problems that have arisen during the processing of a SOAP message.

(3) The env:Fault element contains two mandatory sub-elements,< env:Code> and <env:Reason>, and (optionally) applicationspecific information in the <env:Detail> sub-element.

(4) **<env:Code>:**The <env:Code> element is a mandatory element in the <env:Fault> element. It provides information about the fault in a form that can be processed by software. It contains a <env:Value> element and an optional <env:Subcode> element.

(5) The <env:Value> element contains a number of standardized fault codes (values) in the form of XML qualified names (QNames) each of which identifies the kind of error that occurred.

(6) A set of code values is predefined by the SOAP specification, including <env:VersionMismatchFound> an invalid namespace for the SOAP Envelope

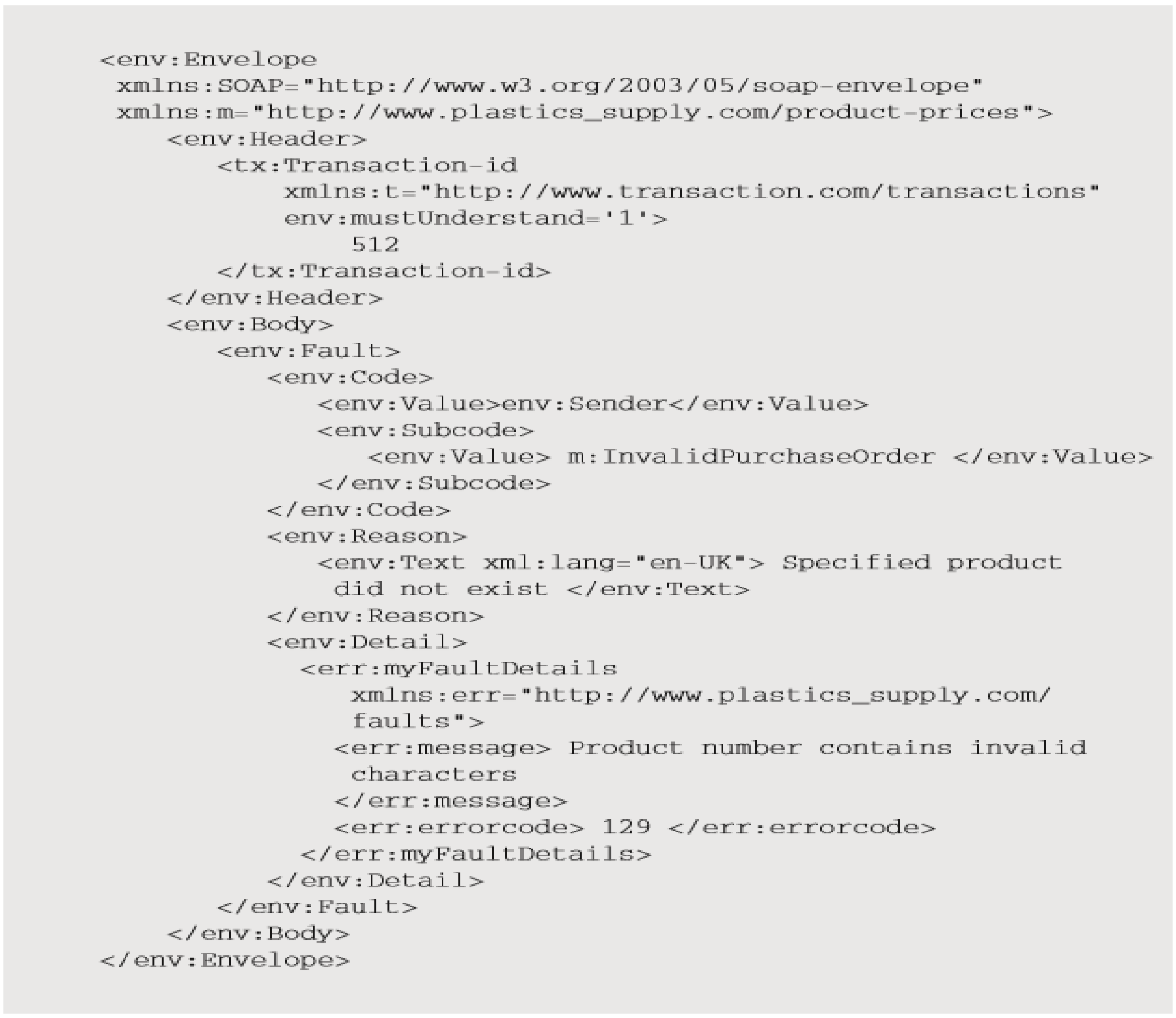
<env:elementMustUnderstand> SOAP header entry not understood by processing party<env:Sender>The message was incorrectly formed or contained incorrect information<env:Receiver>There was a problem with the server so the message could not proceed

(7) The<env:Subcode>that provides more information about the fault.This subelement can have recursive structure

(8) <env:Reason>: The <env:Reason> element is a mandatory element in the <env:Fault> element. The <env:Reason >sub-element contains a human-readable description that gives an account of the fault situation.

(9) <env:Detail>: The <env:detail> element carries application-specific error information related to the <env:Body> element. It must be present if the contents of the <env:Body> element were not successfully processed.

(10) Example of a fault SOAP message



```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
           env:mustUnderstand='1'>
               512
       </tx:Transaction-id>
    </env:Header>
    <env:Body>
       <env:Fault>
          <env:Code>
             <env:Value>env:Sender</env:Value>
             <env:Subcode>
                <env:Value> m:InvalidPurchaseOrder </env:Value>
             </env:Subcode>
          </env:Code>
          <env:Reason>
             <env:Text xml:lang="en-UK"> Specified product
              did not exist </env:Text>
          </env:Reason>
          <env:Detail>
            <err:myFaultDetails
               xmlns:err="http://www.plastics_supply.com/
               faults">
              <err:message> Product number contains invalid
               characters
              </err:message>
              <err:errorcode> 129 </err:errorcode>
            </err:myFaultDetails>
          </env:Detail>
       </env:Fault>
    </env:Body>
</env:Envelope>
```

(11) How SOAP Fault are Processed

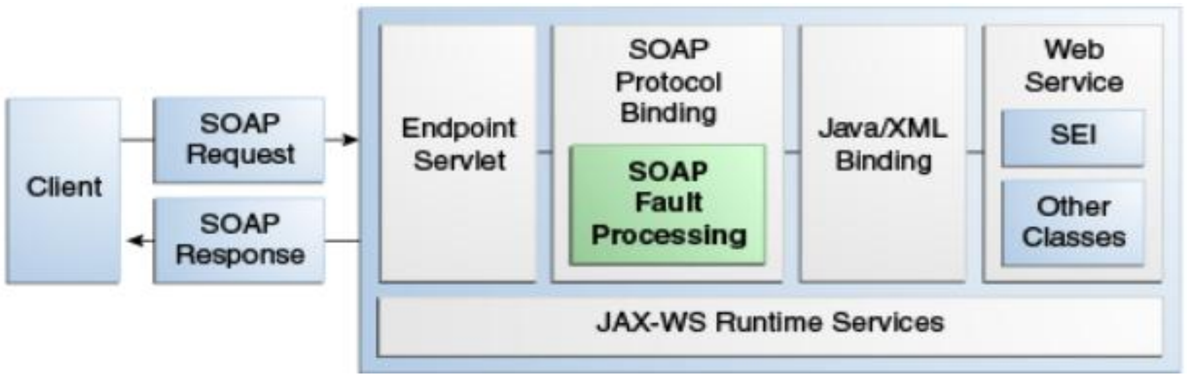


Figure: How SOAP Faults are Processed

Que.10:Describe with a diagram a RPC call using SOAP over http

1. 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. With this type of middleware the application elements communicate with each other synchronously, meaning that they use a request/wait-for-reply model of communication.

2.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.

3. 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 mimics the interface of the remote object and its methods.

4. RPCs work well for smaller, simple applications where communication is primarily point to point (rather than one system to many). RPCs do not scale well to large, missioncritical applications, as they leave many crucial details to the programmer, including handling network and system failures, handling multiple connections, and synchronization between processes.

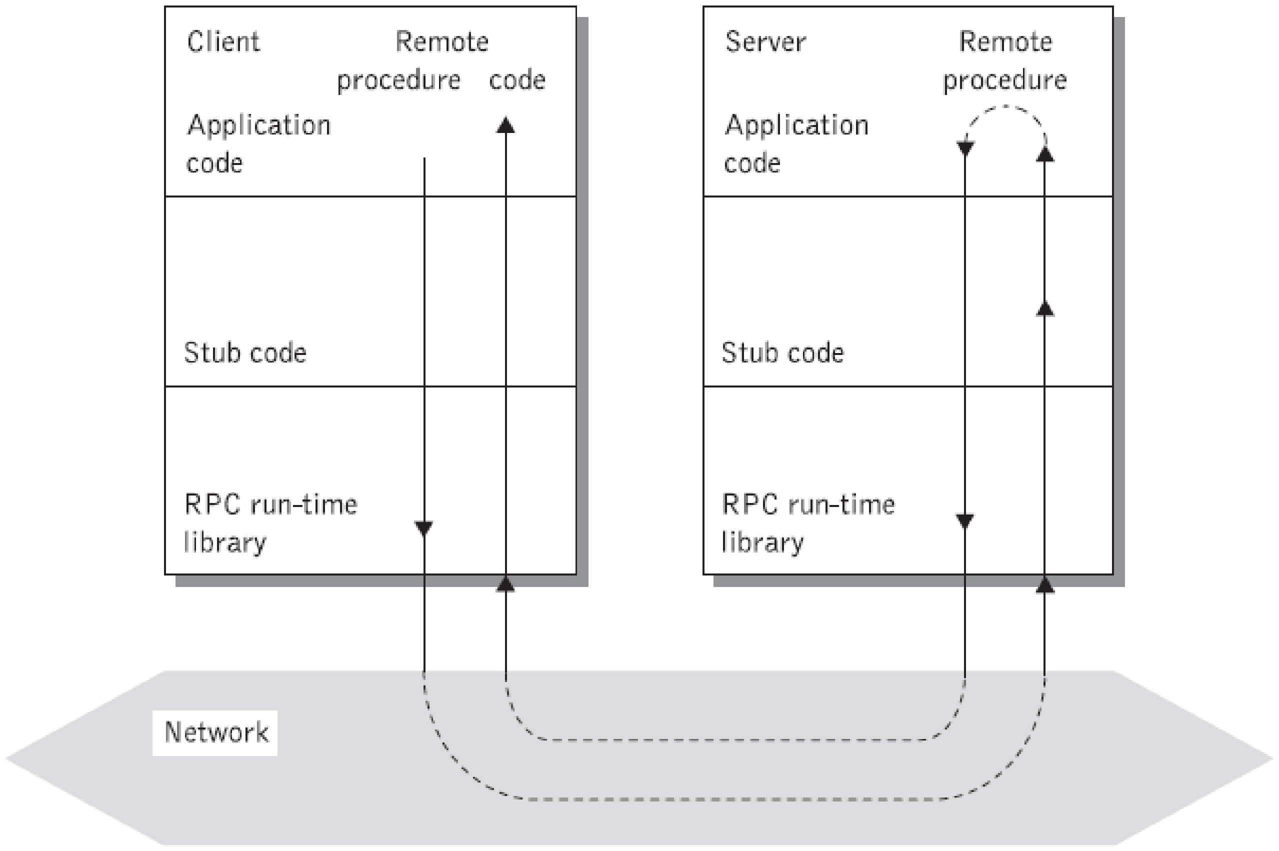


Figure : RPC communication

5. This can be understood by the fact that when performing a synchronous operation across multiple processes, the success of one RPC call depends
on the success of all downstream RPC-style calls that are part of the same synchronous request/response cycle. This makes the invocation a whole-or-nothing proposition [Chappell 2004]. If one operation is unable to complete for any reason, all other dependent operations will fail with it. This is shown in Figure .

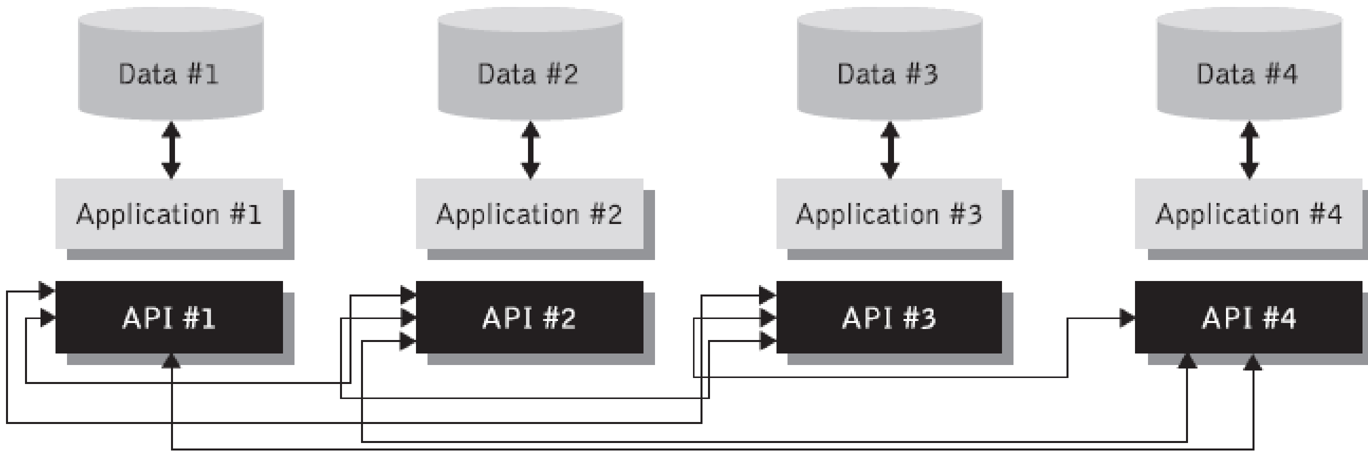


Figure : Tightly coupled RPC point-to-point integrations

6. RPC-style programming leads to *tight coupling* of interfaces and applications. In an RPC environment each application needs to know the intimate details of the interface of every other application – the number of methods it exposes and the details of each method signature it exposes. **7**.This figure clearly shows that the synchronized nature of RPC tightly couples the client to the server. The client cannot proceed – it is blocked – until the server responds and the client fails if the server fails or is unable to complete.
8. Technologies that predominantly use RPC-style communication include the Common Object Request Broker Architecture (CORBA), the Java Remote Method Invocation (RMI), DCOM, Active X, Sun-RPC, Java API for XML-RPC (JAX-RPC), and the Simple Object Access Protocol (SOAP) v1.0 and v1.1.
9. Component-based architectures such as Enterprise Java Beans are also built on top of this model. However, due to their synchronous nature, RPCs are not a good choice to use as the building blocks for enterprise-wide applications where high performance and high reliability are needed.
10 . The synchronous tightly coupled nature of RPCs is a severe hindrance in system-tosystem processing where applications need to be integrated together. Under synchronous solutions, applications are integrated by connecting APIs together on a point-to-point basis.

Q11. List the advantages and disadvantages of SOAP.

Ans. As with any other protocol, several aspects of using SOAP can be seen as advantages, whereas other aspects can be seen as limitations.

Following are advantages of Soap –

1) Simplicity: SOAP is simple as it based on XML, which is highly structured and easy to parse

2) Portability: SOAP is portable without any dependencies on the underlying Platform like byte-ordering issues or machine-word widths. Today, XML parsers exist for virtually any platform from mainframes to write-watch-size devices.

3) Firewall-friendly: Posting data over HTTP means not only that the delivery mechanism is widely available but also that SOAP is able to get past firewalls that pose problems for other methods.

4) Use of open standards: SOAP is built on open, rather than vendor-specific, technologies and facilitates true distributed interoperability and loosely coupled applications. Because SOAP is a wire protocol based on XML and HTTP, it is possibly the most widely interoperable protocol to date and can be used to describe message exchanges between autonomous technical enivironments and highly diverse enterprise applications.

5) Universal acceptance: SOAP is the most widely accepted standard in the message communication domain.

6)Resilience to changes: Changes to the SOAP infrastructure will likely not affect applications using the protocol, unless significant serialization changes are made to the SOAP specification.

Several aspects of SOAP that can be considered as disadvantageous and the disadvantages are as follows –

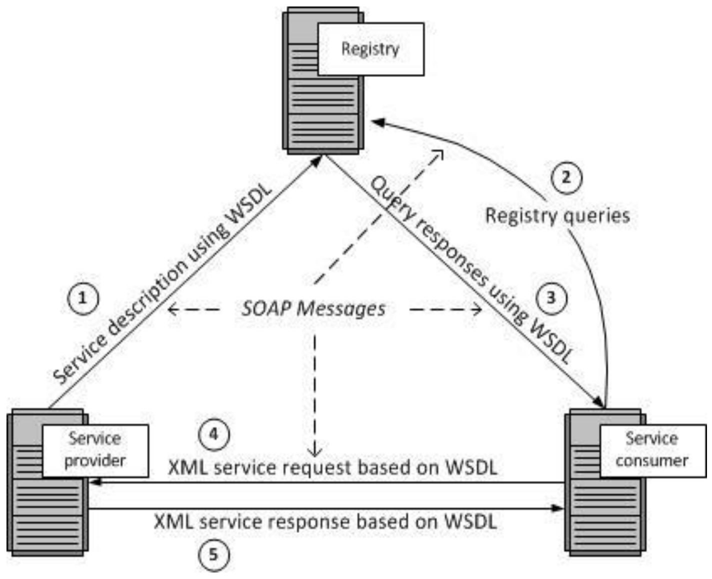
- SOAP was initially tied to HTTP and this mandated a request/response architecture that was not appropriate for all situations. HTTP is a relatively slow protocol and of course the performance of SOAP suffered. The latest version of SOAP specification loosens this requirement.
- SOAP is stateless. The stateless nature of SOAP requires that the requesting application must reintroduce itself to other applications when more connections are required as if it had never been connected before. Maintaining the state of a connection is desirable when multiple Web services interact in the context of business processes and transactions.
- SOAP serializes by value and does not support serialization by reference. Serialization by value requires that multiple copies of an object will, over time, contain state information that is not synchronized with other dislocated copies of the same object. This means that currently it is not possible for SOAP to refer or point to some external data source (in the form of an object reference).

Q.12. Why is a service description needed? Explain in detail WSDL.

Ans. Service description is needed because of following functionality

- A service provider describes its service using WSDL. This definition is published to a repository of services. The repository could use Universal Description, Discovery, and Integration (UDDI). Other forms of directories could also be used.
- A service consumer issues one or more queries to the repository to locate a service and determine how to communicate with that service.
- Part of the WSDL provided by the service provider is passed to the service consumer. This tells the service consumer what the requests and responses are for the service provider.
- The service consumer uses the WSDL to send a request to the service provider.
- The service provider provides the expected response to the service consumer.

The Web Services Description Language (WSDL) forms the basis for the original Web Services specification. The following figure illustrates the use of WSDL. At the left is a service provider. At the right is a service consumer.


Registry
2
Registry queries
1
Service description using WSDL
Query responses using WSDL
3
SOAP Messages
Service provider
Service consumer
4
XML service request based on WSDL
XML service response based on WSDL
5

Q3.Describe The Structure OF WSDL document?

Ans.WSDL is an acronym for Web Services Description Language. It is an XML based interface definition language that is used for the defining the functionality which is being provided by the web service. The acronym is also used for a specific WSDL description of a web service (WSDL file), which provides a machine readable description of how the service can be called, what parameters it expects and what data structure it returns. The WSDL describes services as collections of network endpoints, or ports.

A WSDL file has a description element which defines or contains the other elements of the WSDL file, which are :-

1). Service
2). Endpoint
3). Binding
4). Interface
5). Operation
6). Types

The WSDL version 2.0 does not contain any alternative for the message element which was present in WSDL version 1.1

Service:

- Contains a set of system functions which have been exposed to the Web Based protocols.

Endpoint:

- Defines the address point to a web service.
- It is typically represented by using simple HTTP URL string.

Binding:

- Specifies the interface and defines the SOAP binding style and transport.
- The binding section also defines the operations.

Interface:

- Defines a web service, the operations that can be performed, and the messages that are used to perform the operation.

Operation:

- Defines the SOAP actions and the way the message is encoded, for example, ”literal”.
- An operation is like a method or function call in a traditional programming language.

Types:

- Describes the data.
- The XML Schema language (also known as XSD) is used (inline or reference) for this purpose.

Message:(Message is no longer supported in WSDL 2.0 version)

- Typically, a message corresponds to an operation.
- The message contains the information needed to perform the operation.
- Each message is made up of one or more logical parts.
- Each part is associated with a message-typing attribute.
- The message name attribute provides a unique name among all messages. The part name attribute provides a unique name among all the parts of the enclosing message. Parts are a description of the logical content of a message.
- In RPC binding, a binding may reference the name of a part in order to specify binding-specific information about the part. A part may represent a parameter in the message; the bindings define the actual meaning of the part.

Messages have been removed in WSDL 2.0, in which XML schema types for defining bodies of inputs, outputs and faults are referred to simply and directly.

Basic Structure OF An WSDL Document :-

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
        xmlns:tns="http://www.tmsws.com/wsdl20sample"
        xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
        targetNamespace="http://www.tmsws.com/wsdl20sample">

<documentation>
   This is a sample WSDL 2.0 document.
</documentation>

<!-- Abstract type -->
  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns="http://www.tmsws.com/wsdl20sample"
```

```
             targetNamespace="http://www.example.com/wsdl20sample">

        <xs:element name="request"> ... </xs:element>
        <xs:element name="response"> ... </xs:element>
      </xs:schema>
    </types>

  <!-- Abstract interfaces -->
    <interface name="Interface1">
      <fault name="Error1" element="tns:response"/>
      <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
        <input messageLabel="In" element="tns:request"/>
        <output messageLabel="Out" element="tns:response"/>
      </operation>
    </interface>

  <!-- Concrete Binding Over HTTP -->
    <binding name="HttpBinding" interface="tns:Interface1"
           type="http://www.w3.org/ns/wsdl/http">
      <operation ref="tns:Get" whttp:method="GET"/>
    </binding>

  <!-- Concrete Binding with SOAP-->
    <binding name="SoapBinding" interface="tns:Interface1"
           type="http://www.w3.org/ns/wsdl/soap"
           wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
           wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
      <operation ref="tns:Get" />
    </binding>

  <!-- Web Service offering endpoints for both bindings-->
    <service name="Service1" interface="tns:Interface1">
      <endpoint name="HttpEndpoint"
             binding="tns:HttpBinding"
             address="http://www.example.com/rest/"/>
      <endpoint name="SoapEndpoint"
             binding="tns:SoapBinding"
             address="http://www.example.com/soap/"/>
    </service>
  </description>
```

Explain in brief the various tags in WSDL document.

Ans. In WSDL(Web Service Description Language) document the <types>, <messages>, <part>, <port Type>, and <operations> elements describe the abstract interface of a Web Service. The <port Type> element is essentially an abstract interface that combines <operations> and <message> definitions. A <port Type> element defines an abstract type and its operations but not the implementation. A <port Type> element is simply a logical group of operations, which describes the interface of a Web service and defines its methods. In WSDL the <port Type> element is implemented by the <binding> and <service> elements, which dictate the Internet protocols, encoding schemas, and an Internet address used by the Web service implementation. Each <operation> element is composed of at most one <input> or one <output> element and any number of <fault> elements.

Each <message> definition describes the payloads of incoming and outgoing messages, i.e. messages that are sent or received by a Web Service. The <message> element corresponds to a single piece of information moving between the invoker and a Web service. Messages consists of <part> elements, each of which represent an instance of particular type(typed parameter). A message can consist of one or more <part> elements with each part representing an instance of a particular type. WSDL allows a message <part> to declare either an <element> or a <type> attribute, but not both. Each <operation> element is declared by <port Type> element and contains a number of <message> definitions describing its input and output parameters as well as any faults.

The root element or the <definition> element encapsulates the entire WSDL document and provides it with its name. It also declares an attribute called target Namespace, which gives the logical namespace for elements defined within the WSDL document and characterizes the service.

<types> element serves as container that contains all abstract data types that define a Web service interface. This element is used to contain XML schemas or external references to XML schemas that describe the data type definitions used within the WSDL document. It also helps in defining all the data types that are described by the built-in primitive data types that XML Schema Definition defines, such as int, float, long, short, string, Boolean, and so on, and allows developers to either use them directly or build complex data types based on those primitive ones before using them in messages.
The elements <sequence> and <all> are standard XSD (XML Schema Definition) elements. The construct <sequence> requires that the content model follows the element sequence defined, while the construct <all> denotes that all the elements that are declared in the <complexType> statement must appear in an instance document.

Q5: What is Web Service Interface Definition?Explain in brief?

Ans:The WSDL interface document defines the message format for operations and messages defined by a particular port type.Web Services Description Language (WSDL) is a standard specification for describing networked, XML-based services. It provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying runtime implementation.Web service interface definition describes the generic web service, including the port type, messages, parts of the messages, and bindings.The WSDL document specifies a web service interface definition, which includes the elements shown in the following table:

Port type component contains Operation signature,Messages component contains Parameter definitions,Types component contains Complex type definitions and Bindings component contains Transport protocol and preload format.

portType:-The description of the operations and their associated messages. PortTypes define abstract operations.

message:-The description of parameters (input and output) and return values.

type:-The schema for describing XML complex types used in the messages.

binding:-Bindings describe the protocol used to access a service, as well as the data formats for the messages defined by a particular portType.

Write a short note on WSDL message exchange patterns.

Ans. A WSDL 2.0 document contains a message exchange pattern (MEP) that defines the way that SOAP 1.2 messages are exchanged between the web service requester and web service provider.

CICS® supports four out of the eight message exchange patterns that are defined in the WSDL 2.0 Part 2: Adjuncts specification and the WSDL 2.0 Part 2: Additional MEPs specification for both service provider and service requester applications. The following MEPs are supported:

In-Only

A request message is sent to the web service provider, but the provider is not allowed to send any type of response to the web service requester.

In requester mode, CICS sends the request message to the web service provider and does not wait for a response.

In-Out

A request message is sent to the web service provider, and a response message is returned to the web service requester. The response message could be a normal SOAP message or a SOAP fault.

In provider mode, when CICS receives a request message from a web service that uses the In-Out MEP, it returns a response message to the requester.

In-Optional-Out

A request message is sent to the web service provider, and a response message is optionally returned to the web service requester. If there is a response, it could be either a normal SOAP message or a SOAP fault.

In provider mode, the decision about whether to return a SOAP response message, a SOAP fault, or no response, happens at run time and is dependant on the service provider application logic.

Robust In-Only

A request message is sent to the web service provider, and a response message is only returned to the web service requester if an error occurs. If there is an error, a SOAP fault message is sent to the requester.

In provider mode, if the pipeline successfully passes the request message to the application, a DFHNORESPONSE container is put in the SOAP handler channel to indicate that the pipeline must not send a response message. If an error occurs in the pipeline, a SOAP fault message is returned to the requester.

Q List and Explain different types of service discoveries?

Ans. Explanation:

1.) Service discovery is the process of locating Web service providers, and extracting Web services descriptions that are already published.

2.) Interrogating services send query to the service registry to search Web services required.

3.) A query consists of search criteria's.

4.) Discovering Web services dependents on the architecture of the service registry.

5.) After the discovery process is complete, the service developer or client application should know the exact location of a Web service(URI), its capabilities, and how to interface with it.

Types of Service Discovery:

* There are two types of Service Discovery such as:

a.) Static

b.) Dynamic

a.) Static

1.) Binding of the service implementation details are done at design time and a service retrieval is performed on a service registry.

2.) The results of the retrieval operation are examined usually by a human designer and the service description returned by retrieval operation is incorporated into the application logic.

b.) Dynamic

1.) Binding of service implementation details are determined at run-time.

2.) The Web service requestor has to specify preferences to enable the application to infer/reason which Web service(s) the requestor is most likely to want to invoke.

3.) Based on application logic and quality of service the application chooses the most appropriate service, binds to it, and invokes it.

Q. What is UDDI? Explain UDDI usage model.

UDDI: Universal Description, Discovery, and Integration
To address the challenges of service registration and discovery, the Universal Description, Discovery, and Integration specification was created. UDDI is a cross-industry initiative to

create a registry standard for Web service description and discovery together with a registry facility that supports the publishing and discovery processes.
UDDI is a registry that contains relatively lightweight data. As a registry its prime purpose is to provide network addresses to the resources it describes, e.g., schemas, interface definitions, or endpoints, in locations across the network. The core concept of the UDDI initiative is the UDDI business registration, an XML document used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI business registration consists of three interrelated components: “white pages”, including address, contact, and other key points of contact; “yellow pages” , the classification of information according to industrial classifications based on standard industry taxonomies; and “green pages”, the technical capabilities and information about services that are exposed by the business including references to specifications for Web services and pointers to various file- and URL-based discovery mechanisms. The companies that host the UDDI global registry are called the UDDI *operator nodes*. UDDI has been designed in a highly normalized fashion, not bound to any technology. In other words, an entry in the UDDI registry can contain any type of resource, independently of whether the resource is XML based or not.

UDDI usage model

The UDDI usage model involves standard bodies and industry consortia publishing the descriptions of available services. The basic UDDI usage model is illustrated in **Figure 6.1.**Once the descriptions of available services have been published, service providers implement and deploy Web services conforming to these type definitions. Prospective clients can then query the UDDI registry based on various criteria such as the name of the business, product classification categories, or even services that implement a given service type definition. These clients can then get the details of the service type definition from the location specified. Finally, the clients can invoke the Web service because they have the service endpoint, and also the details on how to exchange messages with it.

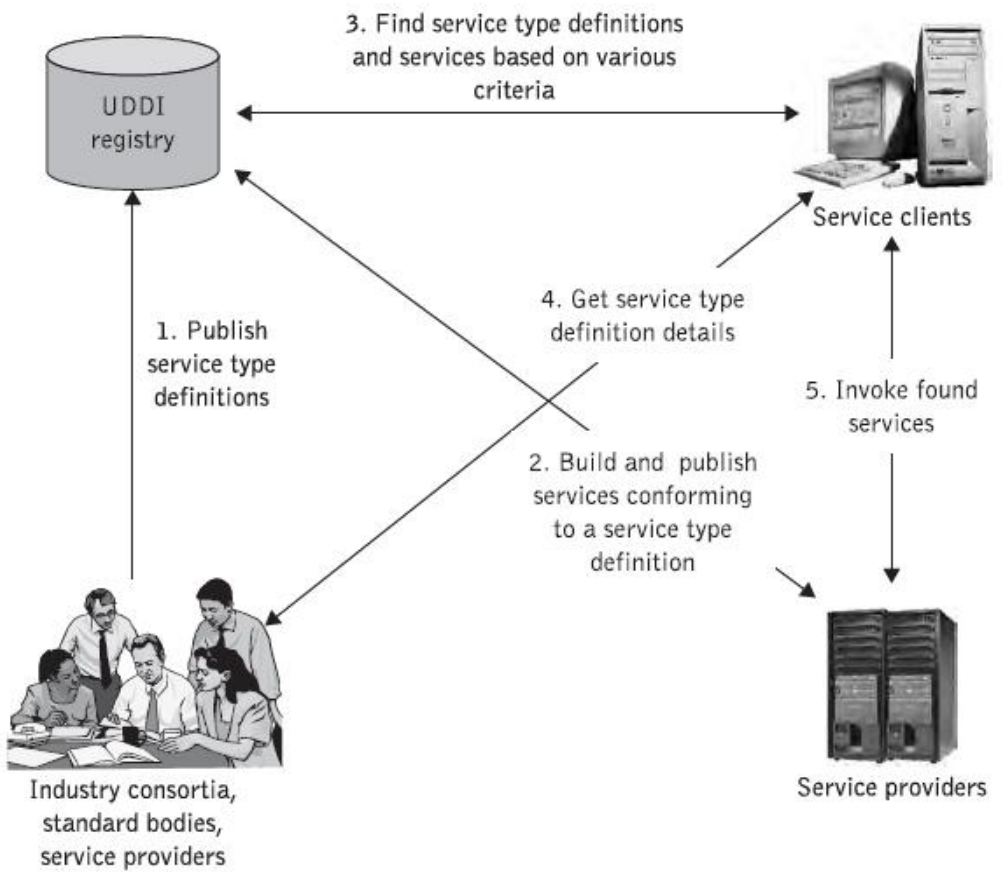

UDDI registry
3. Find service type definitions and services based on various criteria
Service clients
1. Publish service type definitions
4. Get service type definition details
5. Invoke found services
2. Build and publish services conforming to a service type definition
Industry consortia, standard bodies, service providers
Service providers

Figure 6.1 The UDDI usage model

EXPLAIN IN DETAIL THE DATA STRUCTURES USED IN UDDI

Ans:-UDDI includes an XML Schema that describes the following data structures −

businessEntity

businessService

bindingTemplate

tModel

publisherAssertion

businessEntity Data Structure

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

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

```
<businessEntity businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40"
   operator = "http://www.ibm.com" authorizedName = "John Doe">
```

```
  <name>Acme Company</name>
  <description>
    We create cool Web services
  </description>
  <contacts>
    <contact useType = "general info">
      <description>General Information</description>
      <personName>John Doe</personName>
      <phone>(123) 123-1234</phone>
      <email>jdoe@acme.com</email>
    </contact>
  </contacts>

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

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

businessService Data Structure

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

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

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

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

bindingTemplate Data Structure

Binding templates are the technical descriptions of the web services represented by the business service structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the web service.

Here is an example of a binding template for Hello World.

```
<bindingTemplate serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
  bindingKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
  <description>Hello World SOAP Binding</description>
  <accessPoint URLType = "http">http://localhost:8080</accessPoint>

  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey = "uuid:EB1B645F-CF2F-491f-811A-4868705F5904">
      <instanceDetails>
        <overviewDoc>
          <description>
            references the description of the WSDL service definition
          </description>
          <overviewURL>
            http://localhost/helloworld.wsdl
          </overviewURL>
        </overviewDoc>
      </instanceDetails>
    </tModelInstanceInfo>
  </tModelInstanceDetails>
</bindingTemplate>
```

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

tModel Data Structure

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

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

Here is an example of a tModel representing the Hello World Interface port type.

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

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

publisherAssertion Data Structure

This is a relationship structure putting into association two or more businessEntity structures according to a specific type of relationship, such as subsidiary or department.

The publisherAssertion structure consists of the three elements: fromKey (the first businessKey), toKey (the second businessKey), and keyedReference.

The keyedReference designates the asserted relationship type in terms of a keyName keyValue pair within a tModel, uniquely referenced by a tModelKey.

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

Q How WSDL to UDDI mapping model is done.

Ans: 1) The UDDI data model defines a generic structure for storing information about a business and the web services it publishes. The UDDI data model is completely extensible, including several repeating sequence structures of information.

2) However, WSDL is used to describe the interface of a web service. WSDL is fairly straightforward to use with UDDI.

3) WSDL is represented in UDDI using a combination of businessService, bindingTemplate, and tmodel information.

4) AS with any service registered in UDDI, generic information about the service is stored in the businessService data structure, and information specific to how and where the service is accessed is stored in one or more associated bindingTemplate structures. Each bindingTemplate structure includes an element that contains the network address of the service and has associated with it one or more tmodel structures that describe and uniquely identify the service.

5) When UDDI is used to store WSDL information, or pointers to WSDL files, the tmodel should be referred to by convention as type wsdlSpec, meaning that the overviewDoc element is clearly identified as pointing to WSDL service interface definition.

6)For UDDI, WSDL contents are split into two major elements the interface file and the implementation file.

Q 21. Describe the various business information provider roles wrt UDDI.

Ans. UDDI is a registry that contains relatively lightweight data. The concept of the UDDI initiative is the UDDI business registration, an XML document used to describe a business entity and its web services. The information provided in a UDDI business registration consists of three interrelated components: "white pages" ,including address, contact, and other key points of contact; "yellow pages" ,the classification of information according to industrial classifications based on standard industry taxonomies; and "green pages" ,the technical capabilities and information about services that are exposed by the business including references to specifications for Web services and pointers to various file- and URL-based discovery mechanisms.

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. For example, either its business name or perhaps some key identifiers, as well as optional categorization and contact information (white pages). 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>. This XML element serves as the top-level structure and contains white page information about a particular business unit (service provider).The <businessEntity> structure can be used to model any businesses and providers within UDDI. All other non- business or provider elements related to the organization that a <businessEntity> entity represents ,such as service description and technical information, are either contained in this entity or referred to by elements nested within it. For instance, each <businessService> contained within a <businessEntity> structure describes a logical service offered by the business. Similarly, each <bindingTemplate> contained within a given <businessEntity> provides the technical description of a Web service that belongs to the logical service that is described by the <businessService>.

Q.22. Write a short note on Web Service Development Lifecycle

Web services development lifecycle (SDLC), or service-oriented design and development,is a highly iterative and continuous approach to developing, implementing, deploying,and maintaining Web services in which feedback is continuously cycled to and from phases in iterative steps of refinement.

◆Managing the entire services lifecycle – including analyzing, identifying, designing, developing, deploying, finding, applying, evolving, and maintaining services.
◆Establishing a platform and programming model, which includes connecting, deploying, and managing services within a specific run-time platform.
◆Adopting best practices and tools for architecting services-oriented solutions in repeatable, predictable ways that deal with changing business needs. This includes mining existing applications to discover potential services, repurposing existing

assets and functionality to extend their utility and make those capabilities accessible as services, creating new services, and “wiring” together services by connectingbehavior exposed through their interfaces.
◆Delivering high-quality workable service-oriented solutions that respect QoS
requirements. These solutions may be implemented as best practices, such as tried
and tested methods for implementing security, ensuring performance, compliance
with standards for interoperability, and designing for change.Fundamental to the above capabilities is that business goals and requirements should always drive downstream design, development, and testing to transform business processes
into composite applications that automate and integrate enterprises. In this way
business requirements can be traced across the entire lifecycle from business goals,
through software designs and code assets, to composite applications.
The phases in the SDLC methodology encompass planning, analysis, design, construction,It is a premise of the lifecycle that these phases are traversed iteratively and that feedback is cycled to and from phases in iterative steps of refinement and that the methodology may actually be built using a blend of forward- and reverse-engineering techniques or other means to facilitate the needs of the business.

Q.23.What is web service? how does it work?

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. 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. A Web service can be: (i) a self-contained business task, such as a funds withdrawal or funds deposit service; (ii) a full-fledged business process, such as the automated purchasing of office supplies; (iii) an application, such as a life insurance application or demand forecasts and stock replenishment; or (iv) a service-enabled resource, such as access to a particular back-end database containing patient medical records. Web services can vary in function from simple requests to complete business applications that access and combine information from multiple sources, such as an insurance brokering system, an insurance liability computation, an automated travel planner, or a package tracking system.

How web service works :

WS works as request-response paradigm , there is an entity which will request for some service to it's specific counterpart namely service provider entity. Upon request, service provider will respond with a response message. So there are two message involved hear one Request message (XML)and one Response message (XML). There are bunch of ways to achieve these. Detail can be found at web service architecture

Beginner can start with JERSEY jsr311 standard reference implementation to build RESTful web services.

Example (jersey specific):

Step One : Creating root resources

```
// The Java class will be hosted at the URI path "/helloworld"
   @Path("/helloworld")
   public class HelloWorldResource {
      @GET
       @Produces("text/plain")
      public String getClichedMessage() {
          return "Hello World";
      }
  }
```

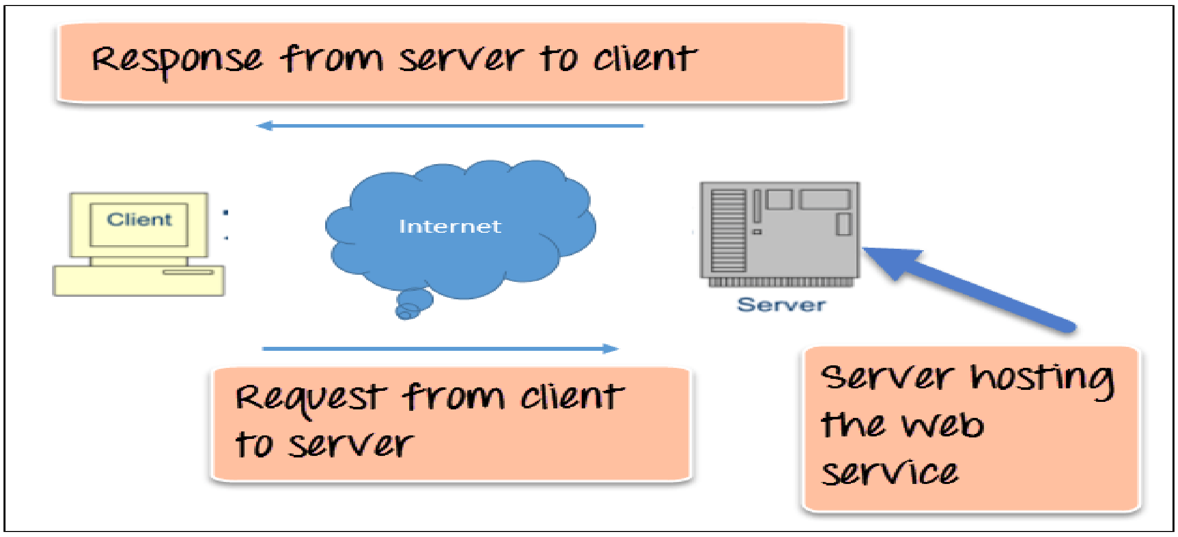
Step Two : Deploying

```
public class Main {
  private static URI getBaseURI() {
      return UriBuilder.fromUri("http://localhost/").port(8080).build();
  }
  public static final URI BASE_URI = getBaseURI();
  protected static HttpServer startServer() throws IOException {
      System.out.println("Starting ...");
      ResourceConfig resourceConfig = new
PackagesResourceConfig("com.sun.jersey.samples.helloworld.resources");
      return GrizzlyServerFactory.createHttpServer(BASE_URI, resourceConfig);
  }
  public static void main(String[] args) throws IOException {
      HttpServer httpServer = startServer();
      System.out.println(String.format("Jersey app started with WADL available at "
              + "%sapplication.wadl\nTry out %shelloworld\nHit enter to stop it...",
              BASE_URI, BASE_URI));
```

```
    System.in.read();

    httpServer.stop();

  }

}
```

Q.24.EXPLAIN THE TYPES OF WEB SERVICE IN DETAIL?

ANS : Web service is a standardized medium to promote communication between the client and server applications on the WWW(World Wide Web).A web service is a software module which is designed to perform a certain set of tasks.


Response from server to client
Client
Internet
Server
Request from client to server
Server hosting the web service

Web Service Architecture Diagram

Type of Web Service

There are mainly two types of web services.

1.SOAP web services.

SOAP (Simple Object Access Protocol).SOAP is known as a transport-independent messaging protocol. SOAP is based on transferring XML data as SOAP Messages. Each message has something which is known as an XML document. The best part of Web services and SOAP is that its all sent via HTTP, which is the standard web protocol. ***what a SOAP message consists of***

Each SOAP document needs to have a root element known as the <Envelope> element. The root element is the first element in an XML document.

The "envelope" is in turn divided into 2 parts. The first is the header, and the next is the body.

The header contains the routing data which is basically the information which tells the

XML document to which client it needs to be sent to.

The body will contain the actual message.

2.RESTful web services.

REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. **REST stands for REpresentational State Transfer.**

Restful Methods:

POST – This would be used to create a new employee using the RESTful web service

GET - This would be used to get a list of all employee using the RESTful web service

PUT - This would be used to update all employee using the RESTful web service

DELETE - This would be used to delete all employee using the RESTful web service

Q 25 .EXPLAIN XML SCHEMA DEFENITION LANGUAGE.

Ans. An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD). XSD (XML Schema Definition) is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an Extensible Markup Language (XML) document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed. XSD can also be used for generating XML documents that can be treated as programming objects. In addition, a variety of XML processing tools can also generate human readable documentation, which makes it easier to understand complex XML documents. The type xsd:language represents a natural language identifier, generally used to indicate the language of a document or a part of a document. Before creating a new attribute of type xsd:language, consider using the xml:lang attribute that is intended to indicate the natural language of the element and its content.

XSD has several advantages over earlier XML schema languages, such as Document Type Definition (DTD) or Simple Object XML (SOX). XSD is written in XML, which means that it doesn't require intermediary processing by a parser. Other benefits include self-documentation, automatic schema creation and the ability to be queried through XML Transformations (XSLT).There are many challenges and limitations with XSD as well. Some detractors have argued it is unnecessarily complex, lacks a formal mathematical description and has limited support for unordered content. the main purpose of XSD is that these files are used to validate XML - that is conforms to a certain format. In that respect they are similar to DTDs that existed before them. The main difference between XSD and DTD is that XSD is written in XML and is considered easier to read and understand. Without XML Schema (XSD file) an XML file is a relatively free set of elements and attributes. The XSD file defines which elements and attributes are permitted and in which order. In general XML is a metalanguage. XSD files define specific languages within that metalanguage. For example, if your XSD file contains the definition of XHTML 1.0, then your XML file is required to fit XHTML 1.0 rather than some other format.

Q.Explain XML Document structure with example?

→The XML Recommendation states that an XML document has both logical and physical structure. Physically, it is comprised of storage units called entities, each of which may refer to other entities, similar to the way that include works in the C language.

→ Logically, an XML document consists of declarations, elements, comments, character references, and processing instructions, collectively known as the markup.

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

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

XML Declaration

→The XML declaration is a piece of markup (which may span multiple lines of a file) that identifies this as an XML document.

→The declaration also indicates whether the document can be validated by referring to an external Document Type Definition (DTD).

→DTDs are the subject of chapter 4; for now, just think of a DTD as a set of rules that describes the structure of an XML document.

→The minimal XML declaration is:

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

→*XML is case-sensitive* (more about this in the next subsection), so it's important that you use lowercase for xml and version.In most cases, this XML declaration is present. If so, it must be the *very first line* of the document and must not have leading white space.

Document Type Declaration

→The document type declaration follows the XML declaration. The purpose of this declaration is to announce the root element (sometimes called the *document element*) and to provide the location of the DTD.[4] The general syntax is:

```
<!DOCTYPE RootElement (SYSTEM | PUBLIC)


   ExternalDeclarations? [InternalDeclarations]? >
```

where <!DOCTYPE is a literal string, RootElement is whatever you name the outermost element of your hierarchy, followed by either the literal keyword SYSTEM or PUBLIC.

→ The optional ExternalDeclarations portion is typically the relative path or URL to the DTD that describes your document type.

Document Body

→The document body, or instance, is the bulk of the information content of the document. Whereas across multiple instances of a document of a given type (as identified by the DOCTYPE) the XML prolog will remain constant, the document body changes with each document instance (in general).

→This is because the prolog defines (either directly or indirectly) the overall structure while the body contains the real instance-specific data. Comparing this to data structures in computer languages, the DTD referenced in the prolog is analogous to a struct in the C language or a class definition in Java, and the document body is analogous to a runtime instance of the struct or class.

Example:

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

<!DOCTYPE html

  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

<title>XHTML 1.0</title>

</head>

<body>

<h1>Simple XHTML 1.0 Example</h1>

<p>See the <a href=

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">DTD</a>.</p>

</body>
```

```
</html>
```

Q.Write short note on WSDL

1)Web Services Description Language (WSDL) is an XML-based language that describes Web services and their uses.
2)A WSDL document is a concrete description of a Web service that includes both abstract and concrete elements.
3)WSDL is the language that UDDI uses.
4)WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
5)WSDL definitions describe how to access a web service and what operations it will perform.
6)WSDL is a language for describing how to interface with XML-based services.
7)WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.
8)The three major elements of WSDL that can be defined separately are

- **Types**
- **Operations**
- **binding**

9)A WSDL document contains the following elements –
Definition ,Data types ,Message ,Operation,Port,type ,Binding Port
And Service .
10)structure of wsdl

```
<definitions>

<type>

Definition of type..

</type>

.....

....

<service>

Definition of service

</service>

</definitions>
```

11)WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet.
12)A client program connecting to a web service can read the WSDL to determine what functions are available on the server.

13)Any special datatypes used are embedded in the WSDL file in the form of XML Schema.
14)The client can then use SOAP to actually call one of the functions listed in the WSDL.
15)some additional major elements are documentation and import.
16)NOTE − WSDL parts are usually generated automatically using web services-aware tools.
17)A WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.
18) The <portType> element combines multiple message elements to form a complete one-way or round-trip operation.
19)one-way,request-response,solicit-response and notification all are the operation of port type.
20) Web services are open standard (XML, SOAP, HTTP, etc.) Web applications that interact with other Web applications for the purpose of exchanging data.
21)UDDI is an XML-based standard for describing, publishing, and finding Web services.
22)SOAP is a simple XML-based protocol that allows applications to exchange information over HTTP.
23) Features of WSDL

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

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

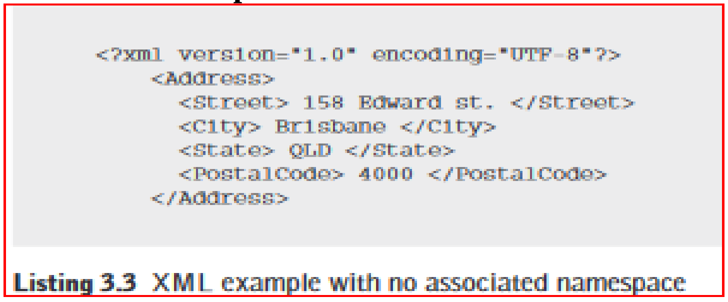
- **Type Definition**
  1)One-way= the operation can receive a message but will not return a response
  2)Request-Response = the operation can receive a request and will return a Response
  3)Solicit-Response = the operation can send a request and will wait for a Response
  4)Notification = the operation can send a message but will not wait for a response

Q. EXPLAIN CHARACTERISTICS OF WEB SERVICES.
→The most important characteristics of web services are such as simple and complex web services, stateful and stateless services , services granularity, loose coupling, synchronous and asynchronous services.
<u>a.) Simple or informational services:-</u>

→Informational services are services of relatively simple nature. they either provide access to content interacting with anend user by means of simple request/response sequences, or alternatively may expose back-end business applications to other applications.
→Informational services are singular in nature in that they perform a complete unit of work that leaves its underlying datastores in a content state.
b.) Complex services or business processes:-
→ The clients of these web services can assemble them to build complex services. An example typical of a simple service exhibiting programmatic behavior could be an inventory checking service that comprises part of an inventory management process.
→Complex services that expose interactive web services expose the functionality of a web application's presentation (browser) layer.
→They frequently expose a multi-step web application behavior that combines a web server, an application server and underlying database systems and typically deliver the application directly to a browser and eventually to a human user for interaction.
c.) State properties: -
→Services could be stateless or stateful. If services can be invoked repeatedly without having to maintain context or state they are called stateless, while services that may require their context to be preserved from one invocation to the next are called stateful.
d.) Loose coupling:-
→ web services interact with one another dynamically and use internet standard technologies, making it possible to build bridges between systems that otherwise would require extensive development efforts.
→ The term coupling indicates the degree of dependency any two systems have on each other.
e.) Synchronicity:-
→Clients of synchronous services express their request as a method call with a set of arguments, which returns a response containing a return value.
→Asynchronous services are document –style or message-driven services. When a client invokes a message-style service, the client typically sends it an entire document.
→Such as purchase order, rather than a discrete set of parameters. Asynchronous interactions (messaging) are a key design pattern in loosely coupled environments.

Q Describe URIs and XML namespace using example?
→A Web architecture starts with a uniform syntax for resource identifiers, so that one can refer to resources, access them, describe them, and share them. The Uniform Resource Identifier (URI) is the basis for identifying resources in WWW.
→A URI consists of a string of characters that uniquely identifies a resource. The URI provides the capability for an element name to be unique, such that it does not conflict with any other element names.
→The W3C uses the newer and broader term URI to describe network resources rather than the familiar but narrower term Uniform Resource Locator (URL). URI is all-inclusive, referring to Internet resource addressing strings that use any of the present or future addressing schemes [Berners-Lee 1998].
→URIs include URLs, which use traditional addressing schemes such as HTTP and FTP, and Uniform Resource Names (URNs).URNs are another form of URI that provide persistence as well as location independence URNs address Internet resources in a location-independent manner and unlike URLs they are stable over time.
```
<?xml version="1.0" encoding="UTF-8"?>
<BillingInformation customer-type="manufacturer">
```

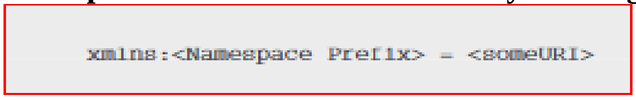
<Name> Right Plastic Products </Name>
<BillingDate> 2002-09-15 </BillingDate>
<Address>
<Street> 158 Edward st. </Street>
<City> Brisbane </City>
<State> QLD </State>
<PostalCode> 4000 </PostalCode>
</Address>
</BillingInformation>
Example of attribute use for Listing 3.1
→an example of an XML document containing address information without an associated namespace.



```
<?xml version="1.0" encoding="UTF-8"?>
    <Address>
      <Street> 158 Edward st. </Street>
      <City> Brisbane </City>
      <State> QLD </State>
      <PostalCode> 4000 </PostalCode>
    </Address>
```

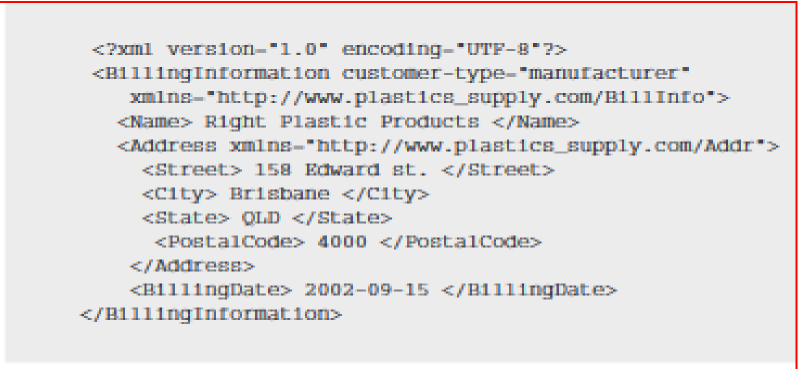
Listing 3.3 XML example with no associated namespace

→The BillingInformation markup in this, we observe that both markups contain references to Address elements. In fact, the Address markup has its own schema in XML Schema Definition Language.
→It is desirable that every time that address information is used in an XML document that the Address declaration is reused and is thus validated against the Address markup schema. This means that the Address element in Listing 3.2 should conform to the Address markup while the rest of the elements in this listing conform to the BillingInformation markup.
→A namespace declaration is indicated by a URI denoting the namespace name. The URI may be mapped to a prefix that may then be used in front of tag and attribute names, separ-ated by a colon. In order to reference a namespace, an application developer needs to first declare one by creating a namespace declaration using the form.



```
xmlns:<Namespace Prefix> = <someURI>
```

→When the prefix is attached to local names of elements and attributes, the elements and attributes then become associated with the correct namespace.
→As the most common URI is a URL, we use URLs as namespace names in our example (always assuming that they are unique identifiers). The two URLs used in this example serve as namespaces for theBillingInformation and Address elements, respectively.
→These URLs are simply used for identification and scoping purposes and it is, of course, not necessary that they point to any actual resources or documents.



```
<?xml version="1.0" encoding="UTF-8"?>
<BillingInformation customer-type="manufacturer"
   xmlns="http://www.plastics_supply.com/BillInfo">
  <Name> Right Plastic Products </Name>
  <Address xmlns="http://www.plastics_supply.com/Addr">
    <Street> 158 Edward st. </Street>
    <City> Brisbane </City>
    <State> QLD </State>
     <PostalCode> 4000 </PostalCode>
   </Address>
   <BillingDate> 2002-09-15 </BillingDate>
</BillingInformation>
```

Listing 3.4 An XML example using namespaces

→The xmlns declarations are the default namespaces for their associated element and all of its declarations. The scope of a default element applies only to the element itself and all of its descendants.

→This means that the declaration xmlns="http://www.plastics_supply.com/Addr" applies only to elements nested within Address.The declaration xmlns="http://www.plastics_supply.com/BillInfo"applies to all elements declared within BillingInformation but not to Address elements as they define their own default namespace.

Q Explain characteristics of Interprocess communication?

→ Processes on two different end systems (with potentially different operating systems) com-municate with each other by exchanging messages across a computer network.

→In the case of Java APIs, the sender specifies the destination using a socket – an indirect refer-ence to a particular port used by the destination process at the destination computer.

<u>1) Messaging:-</u>

→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. These properties are essentially parts of the message body that get promoted to a special section of the message so that filtering can be applied to the message by clients or specialized routers . The message body carries the actual "payload" of the message.

<u>2) Message destinations and sockets:-</u>

→ For interprocess communication processes may use multiple ports from which toreceive messages. Servers usually publicize their port numbers for use by clients and processes use multiple ports from which to receive messages.

→Any process that knows the identifier of a port can send a message to it.During interprocess communication messages are sent to (Internet address, local port)pairs. A local port is a message destination within a computer, specified as an identifier.

→There is a serious drawback with this approach in a case where the client uses a fixed address to a service: then that service must always run on the same computer for its address to remain valid. This can be avoided either if client applications refer to services by name and use a name server to translate names into server locations at run-time, or by having the operating system provide location-independent identifiers for messages allow-ing service relocation.

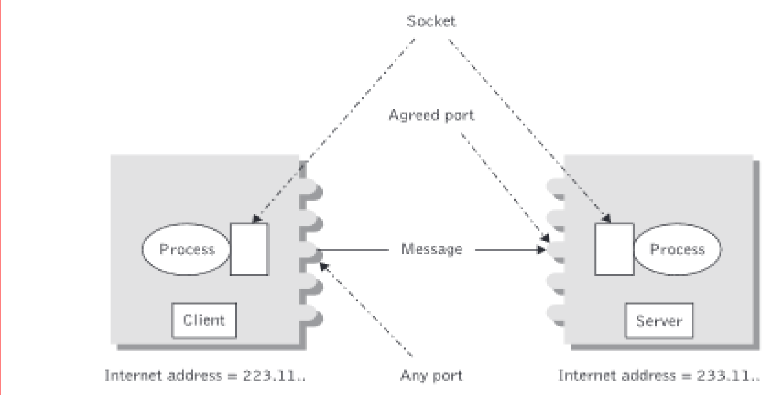


Figure 2.5 Ports and sockets

<u>3) Synchronous and asynchronous forms of message communication:-</u>

→Whilst there are different types of messaging middleware, they all can support one, or sometimes two, basic modes 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 com-munication 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.

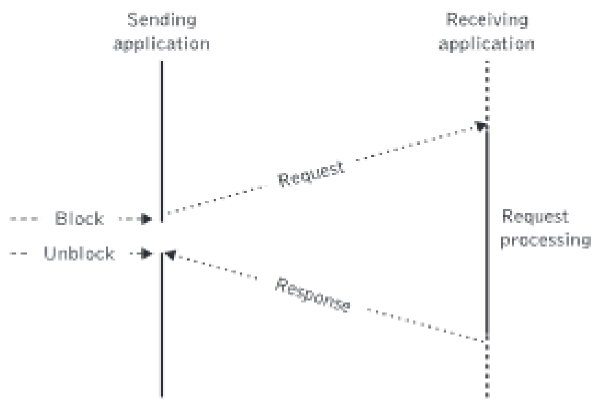


Figure 2.6 Synchronous messaging

→Asynchronous messaging is usually implemented by some queuing mechanism. Two types of message queues exist: these are store and forward and publish/subscribe.

Q31) Write a note on SOAP as messaging protocol?(by roll no 331)

The goal of SOAP is to diffuse the barriers of heterogeneity that separate distributed computing platforms. SOAP achieves this by following the same recipe as other successful Web protocols: simplicity, flexibility, firewall friendliness, platform neutrality, and XML messaging-based (text-based). SOAP is simply an attempt to codify the usage of existing

Internet technologies to standardize distributed communications over the Web, rather than being a new technological advancement.

A SOAP XML document instance is called a SOAP message (or SOAP envelope) and is usually carried as the payload of some other network protocol. As already explained, the most common way to exchange SOAP messages is via HTTP, used by Web browsers to access HTML Web pages. HTTP is simply a convenient way of sending and receiving SOAP messages.

Image not found or unsupported: Pictures/10000000000003320000155CE0304C0D2F77976.png

SOAP has a clear purpose: exchanging data over networks. Specifically, it concerns itself with encapsulating and encoding XML data and defining the rules for transmitting and receiving that data .In short, SOAP is a network application protocol that is used to transfer messages between service instances described by WSDL interfaces. Above figure illustrates this situation and also shows that SOAP messages use different protocols such as HTTP to transport messages and locate the remote systems associated with interacting Web services. SOAP describes how a message is formatted but it does not specify how it is delivered. The message must be embedded in a transport-level protocol to achieve this purpose. HTTP is the most commonly used transport protocol but also other protocols, such as SMTP, FTP, or RMI, may be used .A SOAP XML document instance is called a SOAP message (or SOAP envelope) and is usually carried as the payload of some other network protocol. As already explained, the most common way to exchange SOAP messages is via HTTP, used by Web browsers to access HTML Web pages. HTTP is simply a convenient way of sending and receiving SOAP messages.

Q32)Describe how SOAP act as a wire representation?(by roll no 332)

Answer: SOAP makes use of openly available technologies that, when combined, specify a wire Protocol. SOAP commonly uses HTTP to transport XML-encoded serialized method argument data from system to system. This serialized argument data is used on the remote end to execute a client’s method call on that system, rather than on a local system. If HTTP is used as a SOAP transport protocol, then SOAP processing is very much aligned with the Internet, which specifies a stateless programming model. The combination of the open XML encoding style and the pervasive HTTP makes SOAP possibly the most interoperable wire protocol invented [Scribner 2000]. Wire protocols, such as SOAP, are designed to meet specific design criteria, including [Scribner 2002] compactness, protocol efficiency, coupling, scalability, and interoperability:

Compactness refers to how terse a network package becomes while conveying the same information. Small degree of compactness is usually best.

Protocol efficiency is directly related to compactness. Efficiency is rated by examining the overhead required to send the payload. The more overhead required, the less efficient the protocol is.

Coupling is the rate at which the client application needs to adapt to changes.

Loosely coupled protocols are quite flexible and can easily adapt to changes, while tightly coupled protocols require significant modifications to both the server and existing clients.

Scalability addresses the ability of a protocol to work with a large number of potential recipients. Some protocols are limited to a few hundreds of clients, while others can easily handle millions.

Interoperability refers to the ability of the protocol to work with a variety of computing platforms. For instance, general-purpose protocols enable clients to send information to a variety of systems.

Protocols, including XML and SOAP, generally lie on a continuum of these characteristics.

No single protocol achieves all properties. For instance, XML and SOAP are both loosely coupled and interoperable. This adversely affects compactness and efficiency.

Both XML and SOAP as document-based protocols are rather verbose and this makes them rather inefficient. The SOAP commonly uses HTTP and is therefore very scalable in its native form. It is far more scalable than distributed object architecture protocols.

Q Soap Communication Model .

The Web services communication model describes how to invoke Web services and relies on SOAP. The SOAP communication model is defined by its communication style and its encoding style. SOAP supports two possible communication styles: RPC and document (or message). The SOAP encoding style conveys information about how the contents of a particular element in the header blocks or the element of a SOAP message are encoded. We shall first describe the SOAP encoding style and then the two SOAP communication styles in some detail.

SOAP defines encoding rules (commonly known as encoding styles) for serialization of a graph of typed objects. Encoding styles are about how applications on different platforms share and exchange data, even though they may not have common data types or representations. The encoding rules help in two areas. First, given a schema in any notation consistent with the type system described, a schema for an XML grammar may be constructed. Second, given a type-system schema and a particular graph of values conforming to that schema, an XML instance may be constructed, using this schema and elements in the SOAP.

Q35) Describe how SOAP works with WSDL.

SOAP and WSDL are related and complementary standards for web services, and they are typically used together.

SOAP is an XML messaging protocol. The SOAP specification describes the format and structure of an "envelope" that conveys an XML message, and it specifies how to "bind" the SOAP envelope to various communication protocols, such as HTTP. SOAP also defines some basic rules about processing the messages in a SOAP envelope.

WSDL is a service description language. It describes the interface of the service, i.e., the structure of the XML messages that the service can accept/return. WSDL also describes how those messages are encoded and indicates what protocols the service supports (e.g., SOAP over HTTP).

If you're familiar with CORBA, SOAP is like IIOP, and WSDL is like IDL. WSDL provides a programmatic description of the service. A tool can parse the WSDL and generate SOAP middleware code.

When building web services in Java, a developer typically uses a web services framework that generates the SOAP code. Popular open source web services frameworks for Java include Apache Axis2, Apache CXF, and Sun's JAX-WS reference implementation. All Java EE application servers also include a web services framework. The specific tooling used to generate the code is framework-specific:

- Apache Axis2 provides Java2WSDL and WSDL2Java tools.

- Apache CXF provides a WSDL2Java tool and supports WSDL generation from Java using annotations and a Java2WSDL tool. (Note that although the Axis2 and CXF tools have the same name, the tools are different and generate different code.)

- Sun's JAX-WS supports WSDL generation from Java using annotations and the apt and wsgen tools, and Java generation from WSDL using the wsimport tool.

All three frameworks also use configuration files for specifying runtime settings and mapping SOAP/XML constructs to Java constructs.

Q Briefly describe the phases of the Web development life cycle

Web site Planning:

Involves the identification of the Web site goals or purpose. The question to ask is: What is the purpose of this Web site?

In addition to understanding the Web site purpose, you should also ask: Who will use the Website? or knowing the target audience in terms of: age, gender, computer literacy, etc.
Understanding the computing environment will allow the designer to know what type of Technologies to use.
The last question is to ask who will provide the information included in the Web site.

Web Site Analysis:
During this phase, the Web designer needs to make decisions about the Web site content and functionality.
It includes a detailed analysis of the content of the Website in terms information covered, processing required, etc.

Web Site design and Development

After, the purpose of the Website has been found and the content has been defined, we need to organize the content of the Website. Many ways to organize the Website exists. Here are some general pointers:

Elements	Purpose
Titles	Use simple titles that clearly explain the purpose of the page
Headings	Use Headings to separate main topics

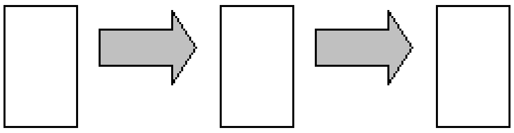
Horizontal rules	Use horizontal rules to separate main topics
Paragraphs	Use paragraphs to help divide large amount of data
Lists	Utilize list. Numbered or bullet when appropriate
Page length	Maintain suitable Web page lengths; about one or two pages are adequate
Information	Emphasize the most important information by placing it at the top of a Web page
Other	· Incorporate a contact e-mail address · Include the date of the last modification

Web site layouts:

Websites are designed using any of several different types of layouts, including linear, hierarchical, and Webbed. Each layout links, or connects, the Web pages in a different structure to define how users navigate through the site and view the Web pages. You should select a layout for your Web site based on how users will most easily navigate through the site to complete tasks and view the Web site contents.

A linear Web site layout connects Web pages in a straight line. A linear Web site layout connects Web pages in a straight line. A linear Web is appropriate if the information on the Web pages should be read in a specific order.

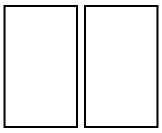
Linear Web Site Layout



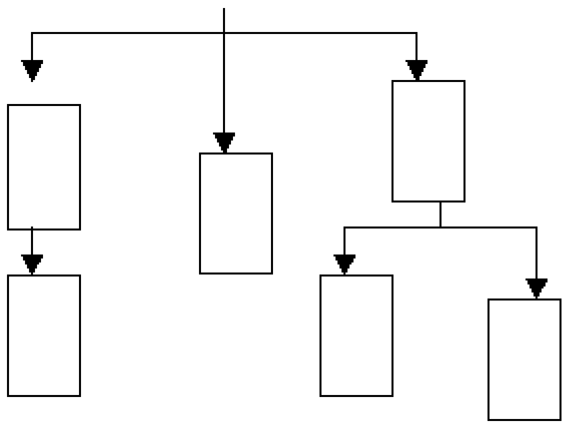
A hierarchical Web site layout connects Web pages in a tree-like structure. A hierarchical Web site layout works well on a site with a main index or table of contents page that links to all other Web pages. With this layout, the main index page would display general information and secondary pages include information that is more detailed.

A Webbed Web site layout has no set structure. A Webbed Web site layout works best on Web sites with information that does not need to be read in a specific order and with many navigation options that users can select.

Hierarchical Web Site layout



Webbed Web Site Layout



Most Web sites are a combination of the linear, hierarchical and Webbed layouts. Some of information on the Web site might be organized hierarchically from an index page; other information might be accessible from all areas of the site while other information might be organized linearly to be read in a specific order.

Using a combination of the three layouts is appropriate, if it helps users navigate through the site easily.

During the design and development phase, you should also consider what types of multimedia could contribute positively to the Web site experience.

Types of multimedia are graphics, photos, video and audio.

Web site testing:

A Web site should be tested at various stages of the Web design and development. This testing should include a review of page content, functionality and usability. Some basic steps to test content and functionality are:
Reviewing for accurate spelling and proofreading content including page titles.
Checking links to ensure that they are not broken and are linked correctly
Checking graphics to confirm they display properly and are linked correctly
Testing forms and other interactive page elements
Testing pages to check for speed of loading on lower speed connection
Printing each page to check how page s print
Testing each Web in several different browser types and versions to verify they display correctly

Usability is the measure of how well product, allows users to accomplish their goals. Usability testing is a method by which users of a Web site are asked to perform certain tasks in an effort to measure the ease of use of the product.

Site Implementation and Maintenance:

Once the Web site testing is complete and any required changes have been made, the Web site can be implemented. Implementation of a Web site means publishing the Web site or uploading it into a Web server.

Once, the Web site has been implemented, its maintenance will include updating the information content by removing the outdated one and putting in the new one.
Periodical checking of the links is also necessary to ensure that they are still active.

Finally, Website monitoring is another key aspect of maintenance. Usually, the Web servers that host the Web sites keep logs about Web site usage.

A log is the file that lists all the Web pages that have been requested from the Web site.
Analyzing the logs allows you to determine the number of visitors to your site and the browser types and versions they are using, as well as their connection speeds, most commonly requested pages.

Cookies:

Internet cookies are very small files that are downloaded from a Web server to a Web browser. Cookies are embedded in the HTML code related to downloading requested pages from a Web site.

When a Web browser first asks for a file from a Web server, the server creates a cookie containing information about the request and sends the cookie to the browser along with the requested file.

The next time a request is made from the browser to the server, the cookie is sent to the server along with the request. When the server returns the requested file, an updated cookie is also returned.

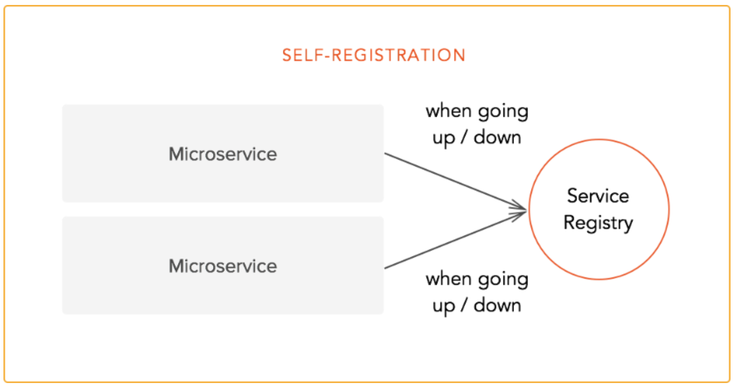
[Q.37] What are Service Registries and what is Service Discovery?
Answer:

The service registry is a database populated with information on how to dispatch requests to microservice instances. Interactions between the registry and other components can be divided into two groups, each with two subgroups:

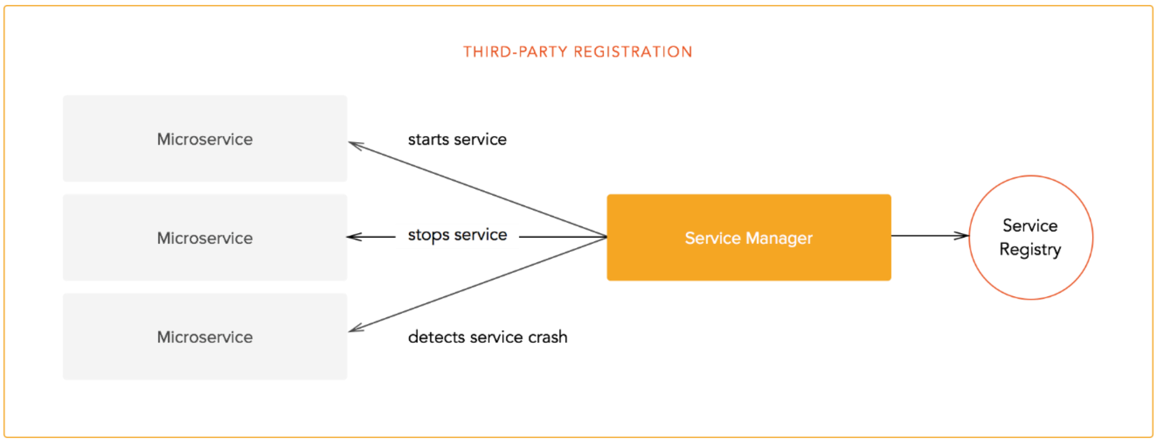
- Interactions between microservices and the registry (registration)
    - Self-registration
    - Third-party registration
- Interactions between clients and the registry (discovery)
    - Client-side discovery
    - Server-side discovery

Registration
Most microservice-based architectures are in constant evolution. Services go up and down as development teams split, improve, deprecate and do their work. Whenever a service endpoint changes, the **registry needs to know about the change**. This is what *registration* is all about: who publishes or updates the information on how to reach each service.

Self-registration forces microservices to interact with the registry by themselves. **When a service goes up, it notifies the registry**. The same thing happens when the service goes down. Whatever additional data is required by the registry must be provided by the **service itself**. If you have been following this series, you know that microservices are all about dealing with a *single concern*, so self-registration might seem like an anti-pattern. However, for simple architectures, self-registration might be the right choice.



Third-party registration is normally used in the industry. In this case, there is a **process or service that manages all other services**. This process polls or checks in some way which microservice instances are running and it automatically **updates the service registry**. Additional data might be provided in the form of per-service config files (or policy), which the registration process uses to update the database. Third-party registration is commonplace in architectures that use tools such as [Apache ZooKeeper] or [Netflix Eureka] and other service managers.

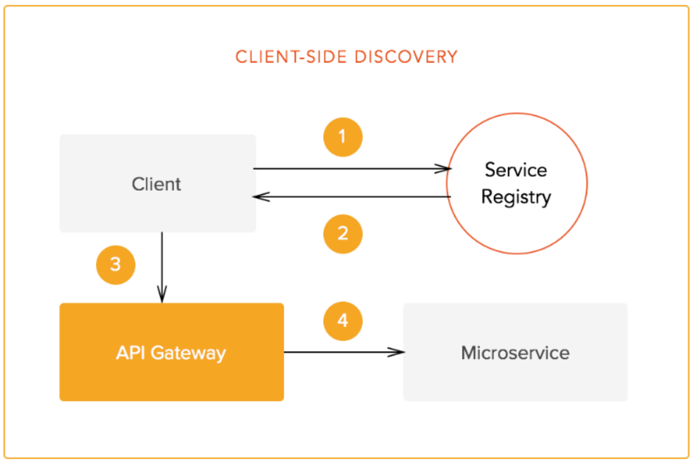


Discovery

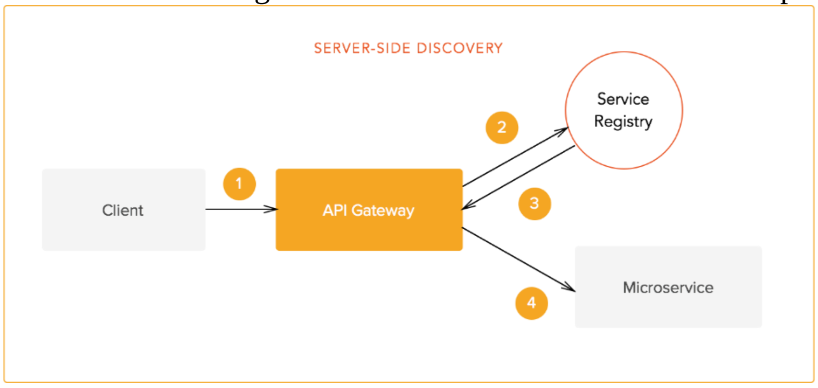
As you can imagine, discovery is the counterpart to registration from the point of view of clients. When a client wants to access a service, it must find out **where the service is located** (and other relevant information to perform the request).

Client-side discovery forces clients to **query a discovery service** before performing the actual requests. As happens with *self-registration*, this requires clients to deal with additional concerns other than their main objective. The discovery service may or may not be located behind the API gateway. If it is not located behind the gateway, balancing, authentication and other cross-cutting concerns may need to be re-implemented for the discovery service.

Additionally, each client needs to know the fixed endpoint (or endpoints) to contact the discovery service. These are all disadvantages. The one big advantage is not having to code the necessary logic in the gateway system. Study this carefully when picking your discovery method.



Server-side discovery makes the **API gateway handle the discovery** of the right endpoint (or endpoints) for a request. This is normally used in bigger architectures. As all requests are directly sent to the gateway, all the benefits discussed in relation to it apply (see part 2). The gateway may also implement discovery caching, so that many requests may have lower latencies. The logic behind cache invalidation is specific to an implementation



What is the UDDI and what are its major characteristics?
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 standardsof web services.
UDDI is an open industry initiative, enabling businesses to discover each other and define how they interact over the Internet

The technology builds on established standards,

including Extensible Markup Language (XML) schemas and files, Simple Object Access Protocol (SOAP)
and HTTP messaging, Domain Name System (DNS) lookup, and Web Services Description Language
(WSDL) Web service interface descriptions.
UDDI consists of four parts —
• The UDDI Registry
• The data, metadata, bindings and documents included in the registry
• The UDDI specifications
• Application Programming Interfaces (APIs) and services for publishing to, querying, and managing
information in the Registry.
The UDDI Registry lets organizations describe themselves and their Web services, provide instructions
for invoking services, or search for Web services from other organizations. The registry's data, metadata,
and bindings describe Web services and how to locate and invoke them. The specification provides
detailed instructions for operating both private and public UDDI Registries. The APIs provide client-side
tools for publishing, deleting, managing and querying Registry entries using SOAP and XML.
The public UDDI Registry — known as the Universal Business Registry (UBR) has been available since
November 2000. The UBR is open to anyone who wants to publish or search for Web services
information. IBM, Microsoft, SAP AG and NTT Communications each provide nodes for accessing or
publishing to the registry. Although they are physically separate, these five nodes replicate their
information so that they operate as a single logical registry.
The latest version of UDDI is version 3.0, a Published Specification submitted by the Organization for the
Advancement of Structured Information Standards (OASIS) on 30 July 2002. At that time, the UDDI
Community, a consortium of over 300 organizations that provided the technology's first three versions,
also became part of OASIS.
UDDI, together with SOAP and WSDL, form the Web services technology "canon." Figure "The Web
Services Stack" shows the relationship between these technologies and the overall Web services stack.

Q.39.What is the purpose of a WSDL to UDDI mapping model?
the fact that both UDDI and WSDL schema have been architected to delineate
clearly between interface and implementation, these two constructs will quite complement-
arily work together naturally. By decoupling a WSDL specification and registering it in
UDDI, we can populate UDDI with standard interfaces that have multiple implementa-
tions, providing a landscape of business applications that share interfaces.
The WSDL to UDDI mapping model is designed to help users find services that imple-

ment standard definitions. The mapping model describes how WSDL <portType> and
<binding> element specifications can become <tModel>s; how the <port>s of
WSDL become UDDI <bindingTemplate>s; and how each WSDL service is regis-
tered as a <businessService>.
As already explained, UDDI provides a method for publishing and finding service
descriptions. The service information defined in WSDL documents is complementary to
the information found in UDDI business and service entries. Since UDDI strives to
accommodate many types of service descriptions it has no direct support for WSDL.
However, since UDDI and WSDL distinguish clearly between interface and implementa-
tion, these two constructs work together quite naturally. The primary focus in this section
is on how to map WSDL service description into a UDDI registry, which is required by
existing Web service tools and run-time environments.
In this section we use the term WSDL interface file to denote a WSDL document that
contains the <types>, <message>, <portType> elements, and the term WSDL binding
file to denote a WSDL document that contains the <binding> element. The term WSDL
implementation file denotes a WSDL document that contains the <service> and
<port> elements. The WSDL implementation file imports the interface and binding file,
while the binding file imports the interface file.
A complete WSDL service description is a combination of a service interface, service
binding, and a service implementation document. Since the service interface and service
binding represent a reusable definition of a service, they are published in a UDDI registry
as a <tModel>. The service implementation describes instances of a service. Each instance
is defined using a WSDL <service> element. Each <service> element in a service
implementation document is used to publish a UDDI <businessService> and the
service <port>s of WSDL become UDDI binding templates. When publishing a WSDL
service description, a service interface must be published as a <tModel> before a service
implementation is published as a <businessService>. By decoupling a WSDL
specification and registering it in UDDI, we can populate UDDI with standard interfaces that
have multiple implementations. An overview of this mapping is given in Figure 6.7. We
summarize this process in two major steps: publication of service interfaces and service
implementations.

Q.40-:Reliable Messaging Model

WS-ReliableMessaging provides an interoperable protocol that a Reliable Messaging (RM) Source
and Reliable Messaging (RM) Destination use to provide Application Source and Destination a
guarantee that a message that is sent will be delivered. The guarantee is specified as a delivery
assurance. The protocol supports the endpoints in providing these delivery assurances. It is the
responsibility of the RM Source and RM Destination to fulfill the delivery assurances, or raise an
error.
There are four basic delivery assurances (Fig. 3) that endpoints can provide:
AtMostOnce Messages will be delivered at most once without duplication or an error will be
raised on at least one endpoint. It is possible that some messages in a sequence may not be
delivered.
AtLeastOnce Every message sent will be delivered or an error will be raised on at least one
endpoint. Some messages may be delivered more than once

ExactlyOnce Every message sent will be delivered without duplication or an error will be raised
on at least one endpoint. This delivery assurance is the logical ”and” of the two prior delivery assurances.
InOrder Messages will be delivered in the order that they were sent. This delivery assurance may be combined with any of the above delivery assurances. It requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications

Q.41 – What is the Enterprise Bus and How does is relate to SOA?

The Enterprise Service Bus is an open standards-based message backbone designed to enable the implementation, deployment, and management of SOA-based solutions with a focus on assembling, deploying, and managing distributed service-oriented architectures. An ESB is a set of infrastructure capabilities implemented by middleware technology that enable an SOA and alleviate disparity problems between applications running on heterogeneous platforms and using diverse data formats. The ESB supports service invocations, message, and event-based interactions with appropriate service levels and manageability.

The ESB is designed to provide interoperability between larger-grained applications and other components via standards-based adapters and interfaces. The bus functions as both transport and transformation facilitator to allow distribution of these services over disparate systems and computing environments.

An ESB provides an implementation backbone for an SOA that treats applications as services. The ESB is about configuring applications rather than coding and hardwiring applications together. It is a lightweight infrastructure that provides plug and play enterprise functionality. It is ultimately responsible for the proper control, flow, and even translations of all messages between services, using any number of possible messaging protocols. An ESB pulls together applications and discrete integration components to create assemblies of services to form composite business processes, which in turn automate business functions in an enterprise. It establishes proper control of messaging as well as applying the needs of security, policy, reliability, and accounting, in an SOA architecture.

The end result is that with an ESB it is then easier to create new composite applications that use pieces of application logic and/or data that reside in existing systems. The ESB distributed processing infrastructure is aware of applications and services and uses content-based routing facilities to make informed decisions about how to communicate with them.

Q.42 - Briefly describe the Key Capabilities of Enterprise Service Bus.

Dynamic connectivity capabilities: Dynamic connectivity is the ability to connect to Web services dynamically without using a separate static API or proxy for each service. Dynamic service connectivity is a key capability for a successful ESB implementation
Reliable messaging capabilities: Reliable messaging can be primarily used to ensure guaranteed delivery of these messages to their destination and for handling events. This capability is crucial for responding to clients in an asynchronous manner and for a successful ESB implementation.
Topic- and content-based routing capabilities: Topic-based routing assumes that messages can be grouped into fixed, topical classes, so that subscribers can explicate interest in a topic and as a consequence receive messages associated to that topic. Content based routing, on the

other hand, allows subscriptions on constraints of actual properties (attributes) of business events Content-based routing forwards messages to their destination based on the context or content of the service.
Transformation capabilities: The ESB transformation services make it possible to ensure that messages and data received by any component are in the format it expects, thereby removing the need to make changes.
Service enablement capabilities: Service enablement includes the ability to access already existing resources such as legacy systems – technically obsolete mission critical elements of an organization's infrastructure – and include them in an SOA implementation.
Endpoint discovery with multiple QoS capabilities: many network endpoints can implement the same service contract, the ESB should make it possible for the client to select the best endpoint at run-time, rather than hard-coding endpoints at build time.
Long-running process and transaction capabilities: long-running services – services that tend to run for long duration, exchanging message (conversation) as they progress. Examples are an on-line reservation system. ESB needs to be able to ensure that complex transactions are handled in a highly reliable manner and if failure should occur, transactions should be capable of rolling back processing to the original, pre-request state.
Security capabilities: Handling and enforcing security is a key success factor for ESB implementations. The ESB needs both to provide a security model to service consumers and to integrate with the (potentially varied) security models of service providers
Integration capabilities: To support SOA in a heterogeneous environment, the ESB needs to integrate with a variety of systems that do not directly support service-style interactions.
Management and monitoring capabilities: Applications overlap and can change over time. Managing these applications is a serious challenge. Monitoring is the ability to track service activities that take place via the bus and provide visibility into various metrics and statistics.
Scalability capabilities: With a widely distributed SOA, there will be the need to scale some of the services or the entire infrastructure to meet integration demands. A decentralized architecture enables independent scalability of individual services as well as the communications infrastructure itself.

Q.43 - What is SOAP? How it is connected to Service oriented architecture?

SOAP

SOAP was originally an acronym for Simple Object Access Protocol (now it is just a name). SOAP is the *de facto* standard messaging protocol used by Web services. SOAP's primary application is inter-application communication. SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a means for transport. In particular, a SOAP method is simply an HTTP request and response that complies with the SOAP encoding rules. A SOAP endpoint is simply an HTTP-based URL that identifies a target for method invocation. The term *lightweight wire protocol* means that SOAP possesses only two fundamental properties. It can send and receive HTTP (or other) transport protocol packets, and process XML messages. SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns. SOAP plays the role of a *binding* mechanism between two conversing endpoints. A SOAP endpoint is simply an HTTP-based URL that identifies a target for a method invocation. It is the goal of SOAP to allow for flexible binding.

How it is connected to Service oriented architecture (SOAP)?

The essential goal of an SOA is to enable general-purpose interoperability among existing technologies and extensibility to future purposes and architectures. SOA is an architectural style, inspired by the service-oriented approach to computing, for enabling extensible interoperability. Web services should be seen as a primary example of a message delivery model that makes it much easier to deploy an SOA.

WSDL is used to describe the service; UDDI, to register and look up the services; and SOAP, as a transport layer to send messages between service consumer and service provider. A consumer can search for a service in the UDDI registry, get the WSDL for the service that has the description, and invoke the service using SOAP.

Q.44 - what is JAX-WS? How it is useful for describing SOAP web services?

Java API for XML Web Services (JAX-WS) is one of a set of Java technologies used to develop Web services. JAX-WS belongs to what Sun Microsystems calls the "core Web services" group.

Like most of the core group, JAX-WS is typically used in conjunction with other technologies. Those other technologies may also come from the core Web services group (JAXB, for example), as well as from enhanced Web services (WSIT), secure Web services (WSIT, WS-Security) etc. groups. JAX-WS is a fundamental technology for developing SOAP (Simple Object Access Protocol) and RESTful web services. JAX-WS is also used to build Web services and corresponding clients that communicate using XML to send messages or use remote procedure calls to exchange data between client and service provider. The main goal of the JAX-RS specifications to make to make the RESTful Web services to development easier that it has been in the past. JAX-WS represents remote procedure calls or messages using XML-based protocols such as SOAP, but hides SOAP's innate complexity behind a Java-based API. Developers use this API to define methods, then code one or more classes to implement those methods and leave the communication details to the underlying JAX-WS API. Clients create a local proxy to represent a service, then invoke methods on the proxy. The JAX-WS runtime system converts API calls and matching replies to and from SOAP messages. Hence this is about JAX-RS and how it is useful for describing the SOAP web services.

Q.45 - How web service are related to distributed computing?

Since services may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration. Clients of services can be other solutions or applications within an enterprise or clients outside the enterprise, whether these are external applications, processes, or customers/users. This distinction between service providers and consumers is independent of the relationship between consumer and provider, which can be either client/server or peer to peer. For the service-oriented computing paradigm to exist, we must find ways for the services to be technology neutral, loosely coupled, and support location transparency.

One of the major advantages of services is that they may be implemented on a single machine or on a large number and variety of devices and be distributed on a local area network or more widely across several wide area networks (including mobile and ad hoc networks).

A particularly interesting case is when the services use the Internet (as the communication medium) and open Internet-based standards. This results in the concept of Web services, which share the characteristics of more general services, but they require special consideration as a result of using a public, insecure, low-fidelity mechanism, such as the Internet, for distributed service interactions.

Q.WHAT IS REST? EXPLAIN RESTFUL SERVICES.

REST (REpresentational State Transfer) is an architectural style for developing web services. REST is popular due to its simplicity and the fact that it builds upon existing systems and features of the internet's HTTP in order to achieve its objectives, as opposed to creating new standards, frameworks and technologies

RESTful web services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using **Uniform Resource Identifiers (URIs)**, typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI**: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery. See The @Path Annotation and URI Path Templates for more information.
- **Uniform interface**: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource. See Responding to HTTP Methods and Requests for more information.
- **Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control. See Responding to HTTP Methods and Requests and Using Entity Providers to Map HTTP Response and Request Entity Bodies for more information.
- **Stateful interactions through hyperlinks**: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction. See Using Entity Providers to Map HTTP Response and Request Entity Bodies and “Building URIs” in the JAX-RS Overview document for more information.

EXPLAIN THE CHRECTERISTICS REQUIRED FOR GOOD RESOURCE REPRESENTATION ?

.As most of the Web services are going to need to establish and adhere to standards, QoS will become an important selling and differentiating point of these services.
THE CHARECTERISTICS REQUIRED FOR GOOD RESOURCE REPRESENTATION ARE AS FOLLOWS:-

Availability:Availability represents the probability that a service is available. 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: Accessibility is the quality aspect of a service that represents the degree it is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible.
Integrity: Integrity is the quality aspect of how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction.All the activities have to be completed to make the transaction successful. When a transaction does not complete, all the changes made are rolled back.
Performance: Performance is the quality aspect of Web service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service. *Throughput* represents the number of Web service requests served at a given time period. *Latency* is the round-trip time between sending a request and receiving the response.
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
Regulatory: Regulatory is the quality aspect of the Web service in conformance with the rules, the law, compliance with standards, and the established service level agreement.
Security: Security is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control.
WHAT IS RESOURCES AND HOW IS IT REPRESENTED?

RESOURCES:- REST architecture treats every content as a resource. These resources can be Text Files, Html Pages, Images, Videos or Dynamic Business Data. REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource where Text, JSON, XML. The most popular representations of resources are XML and JSON.

REPRESENTATION:-

A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database. Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format and client can understand the same format.

For example, in RESTful Web Services - First Application chapter, a user is a resource which is represented using the following XML format −

```
<user>
  <id>1</id>
  <name>Mahesh</name>
  <profession>Teacher</profession>
</user>
```

The same resource can be represented in JSON format as follows −

```
{
  "id":1,
  "name":"Mahesh",
  "profession":"Teacher"
}
```

.

Following are some important points to be considered while designing a representation format of a resource in RESTful Web Services.

Understandability − Both the Server and the Client should be able to understand and utilize the representation format of the resource.

Completeness − Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent simple as well as complex structures of resources.

Linkablity − A resource can have a linkage to another resource, a format should be able to handle such situations.

However, at present most of the web services are representing resources using either XML or JSON format. There are plenty of libraries and tools available to understand, parse, and modify XML and JSON data.

In a RESTful system, we can easily map our CRUD actions on the resources to the appropriate HTTP methods such as POST, GET, PUT, and DELETE.

HTTP GET

Use GET requests **to retrieve resource representation/information only** – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be **safe methods**. Additionally, GET APIs should be **idempotent**, which means that making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

Example request URIs

HTTP GET http://www.appdomain.com/users

HTTP POST

Use POST APIs **to create new subordinate resources**, e.g. a file is subordinate to a directory containing it or a row is subordinate to a database table. Talking strictly in terms of REST, POST methods are used to create a new resource into the collection of resources.

Ideally, if a resource has been created on the origin server, the response SHOULD be HTTP response code 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header.

Example request URIs

HTTP POST http://www.appdomain.com/users

HTTP POST http://www.appdomain.com/users/123/accounts

HTTP PUT

Use PUT APIs primarily **to update existing resource** (if the resource does not exist then API may decide to create a new resource or not). If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code 201 (Created) response and if an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

The difference between the POST and PUT APIs can be observed in request URIs. POST requests are made on resource collections whereas PUT requests are made on an individual resource.

Example request URIs

HTTP PUT http://www.appdomain.com/users/123

HTTP PUT http://www.appdomain.com/users/123/accounts/456

HTTP DELETE

As the name applies, DELETE APIs are used **to delete resources** (identified by the Request-URI).

A successful response of DELETE requests SHOULD be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

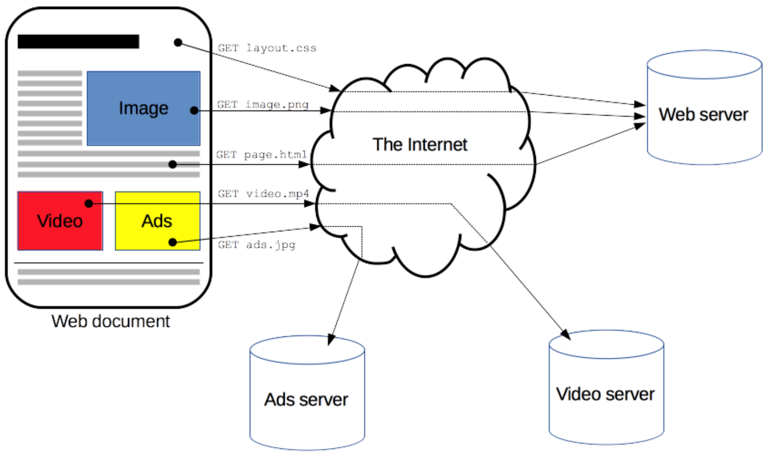
Example request URIs

HTTP DELETE http://www.appdomain.com/users/123

HTTP DELETE http://www.appdomain.com/users/123/accounts/456

Explain with the help of a diagram the various parts of a)http request b)http response

HTTP is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and a client-server protocol, which means requests are initiated by the recipient, usually the Web browser.



Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses.

A simple request message from a client computer consists of the following components:

A request line to get a required resource, for example a request GET /content/page1.html is requesting a resource called /content/page1.html from the server.

Headers (Example – Accept-Language: EN).
An empty line.
A message body which is optional.

A simple response from the server contains the following components:

HTTP Status Code (For example HTTP/1.1 301 Moved Permanently, means the requested resource was permanently moved and redirecting to some other resource).
Headers (Example – Content-Type: html)
An empty line.
A message body which is optional.

Q Write a note on SOAP as messaging protocol?(by roll no 331)

Answer:

The goal of SOAP is to diffuse the barriers of heterogeneity that separate distributed computing platforms. SOAP achieves this by following the same recipe as other successful Web protocols: simplicity, flexibility, firewall friendliness, platform neutrality, and XML messaging-based (text-based). SOAP is simply an attempt to codify the usage of existing Internet technologies to standardize distributed communications over the Web, rather than being a new technological advancement.

A SOAP XML document instance is called a SOAP message (or SOAP envelope) and is usually carried as the payload of some other network protocol. As already explained, the most common way to exchange SOAP messages is via HTTP, used by Web browsers to access HTML Web pages. HTTP is simply a convenient way of sending and receiving SOAP messages.

SOAP has a clear purpose: exchanging data over networks. Specifically, it concerns itself with encapsulating and encoding XML data and defining the rules for transmitting and receiving that data .In short, SOAP is a network application protocol that is used to transfer messages between service instances described by WSDL interfaces. Above figure illustrates this situation and also shows that SOAP messages use different protocols such as HTTP to transport messages and locate the remote systems associated with interacting Web services. SOAP describes how a message is formatted but it does not specify how it is delivered. The message must be embedded in a transport-level protocol to achieve this purpose. HTTP is the most commonly used transport protocol but also other protocols, such as SMTP, FTP, or RMI, may be used .A SOAP XML document instance is called a SOAP message (or SOAP envelope) and is usually carried as the payload of some other network protocol. As already explained, the most common way to exchange SOAP messages is via HTTP, used by Web browsers to access HTML Web pages. HTTP is simply a convenient way of sending and receiving SOAP messages.

Q32)Describe how SOAP act as a wire representation?(by roll no 332)

Answer: SOAP makes use of openly available technologies that, when combined, specify a wire Protocol. SOAP commonly uses HTTP to transport XML-encoded serialized method argument data from system to system. This serialized argument data is used on the remote end to execute a client’s method call on that system, rather than on a local system. If HTTP is used as a SOAP transport protocol, then SOAP processing is very much aligned with the Internet, which specifies a stateless programming model. The combination of the open XML encoding style and the pervasive HTTP makes SOAP possibly the most interoperable wire protocol invented [Scribner 2000]. Wire protocols, such as SOAP, are designed to meet specific design criteria, including [Scribner 2002] compactness, protocol efficiency, coupling, scalability, and interoperability:

Compactness refers to how terse a network package becomes while conveying the same information. Small degree of compactness is usually best.

Protocol efficiency is directly related to compactness. Efficiency is rated by examining the overhead required to send the payload. The more overhead required, the less efficient the protocol is.

Coupling is the rate at which the client application needs to adapt to changes.

Loosely coupled protocols are quite flexible and can easily adapt to changes, while tightly coupled protocols require significant modifications to both the server and existing clients.

Scalability addresses the ability of a protocol to work with a large number of potential recipients. Some protocols are limited to a few hundreds of clients, while others can easily handle millions.

Interoperability refers to the ability of the protocol to work with a variety of computing platforms. For instance, general-purpose protocols enable clients to send information to a variety of systems.

Protocols, including XML and SOAP, generally lie on a continuum of these characteristics.

No single protocol achieves all properties. For instance, XML and SOAP are both loosely coupled and interoperable. This adversely affects compactness and efficiency.

Both XML and SOAP as document-based protocols are rather verbose and this makes them rather inefficient. The SOAP commonly uses HTTP and is therefore very scalable in its native form. It is far more scalable than distributed object architecture protocols.

Q33) Soap Communication Model .(by roll no:333)

The Web services communication model describes how to invoke Web services and relies on SOAP. The SOAP communication model is defined by its communication style and its encoding style. SOAP supports two possible communication styles: RPC and document (or message). The SOAP encoding style conveys information about how the contents of a particular element in the header blocks or the element of a SOAP message are encoded. We shall first describe the SOAP encoding style and then the two SOAP communication styles in some detail.

SOAP defines encoding rules (commonly known as encoding styles) for serialization of a graph of typed objects. Encoding styles are about how applications on different platforms share and exchange data, even though they may not have common data types or representations. The encoding rules help in two areas. First, given a schema in any notation consistent with the type system described, a schema for an XML grammar may be constructed. Second, given a type-system schema and a particular graph of values conforming to that schema, an XML instance may be constructed, using this schema and elements in the SOAP.

Q35) Describe how SOAP works with WSDL.(by roll no : 335)

SOAP and WSDL are related and complementary standards for web services, and they are typically used together.

SOAP is an XML messaging protocol. The SOAP specification describes the format and structure of an "envelope" that conveys an XML message, and it specifies how to "bind" the SOAP envelope to various communication protocols, such as HTTP. SOAP also defines some basic rules about processing the messages in a SOAP envelope.

WSDL is a service description language. It describes the interface of the service, i.e., the structure of the XML messages that the service can accept/return. WSDL also describes how those messages are encoded and indicates what protocols the service supports (e.g., SOAP over HTTP).

If you're familiar with CORBA, SOAP is like IIOP, and WSDL is like IDL. WSDL provides a programmatic description of the service. A tool can parse the WSDL and generate SOAP middleware code.

When building web services in Java, a developer typically uses a web services framework that generates the SOAP code. Popular open source web services frameworks for Java include Apache Axis2, Apache CXF, and Sun's JAX-WS reference implementation. All Java EE application servers also include a web services framework. The specific tooling used to generate the code is framework-specific:

- Apache Axis2 provides Java2WSDL and WSDL2Java tools.

- Apache CXF provides a WSDL2Java tool and supports WSDL generation from Java using annotations and a Java2WSDL tool. (Note that although the Axis2 and CXF tools have the same name, the tools are different and generate different code.)

- Sun's JAX-WS supports WSDL generation from Java using annotations and the apt and wsgen tools, and Java generation from WSDL using the wsimport tool.

All three frameworks also use configuration files for specifying runtime settings and mapping SOAP/XML constructs to Java constructs.

Q.Write short note on JAX-RS

JAX-RS is an integral part of the Java EE platform, which ensures portability of your REST API code across all Java EE-compliant application servers. The first release of JAX-RS was based on JSR 311. The latest version is JAX-RS 2 (based on JSR 339), which was released as part of the Java EE 7 platform. There are multiple JAX-RS implementations available today by various vendors. Some of the popular JAX-RS implementations are as follows:

• Jersey RESTful web service framework: This framework is an open source framework for developing RESTful web services in Java. It serves as a JAX-RS reference implementation. You can learn more about this project at https://jersey.java.net.

• Apache CXF: This framework is an open source web services framework. CXF supports both JAX-WS and JAX-RS web services. To learn more about CXF, refer to http://cxf.apache.org.

• RESTEasy: This framework is an open source project from JBoss, which provides various modules to help you build a RESTful web service. To learn more about RESTEasy, refer to http://resteasy.jboss.org

• Restlet: This framework is a lightweight, open source RESTful web service framework. It has good support for building both scalable RESTful web service APIs and lightweight REST clients, which suits mobile platforms well. You can learn more about Restlet at http://restlet.com.

JAX-RS annotations The main goal of the JAX-RS specification is to make the RESTful web service development easier than it has been in the past. As JAX-RS is a part of the Java EE platform, your code becomes portable across all Java EE-compliant servers. In this section, we will learn how to use the JAX-RS annotations for building RESTful web services.

Q - Explain HTTP and its working with neat diagram
=> HTTP stands for HyperText Transfer Protocol. This is a basis for data communication in the internet. The operation of HTTP involves the communication between a HTTP client application (Usually web browser) and a HTTP server application (Web servers like IIS). HTTP uses Transmission Control Protocol (TCP) as the Transport Layer Protocol at Well Known port number 80. HTTP is a generic and stateless protocol.

Basic Features
There are three basic features that make HTTP a simple but powerful protocol:

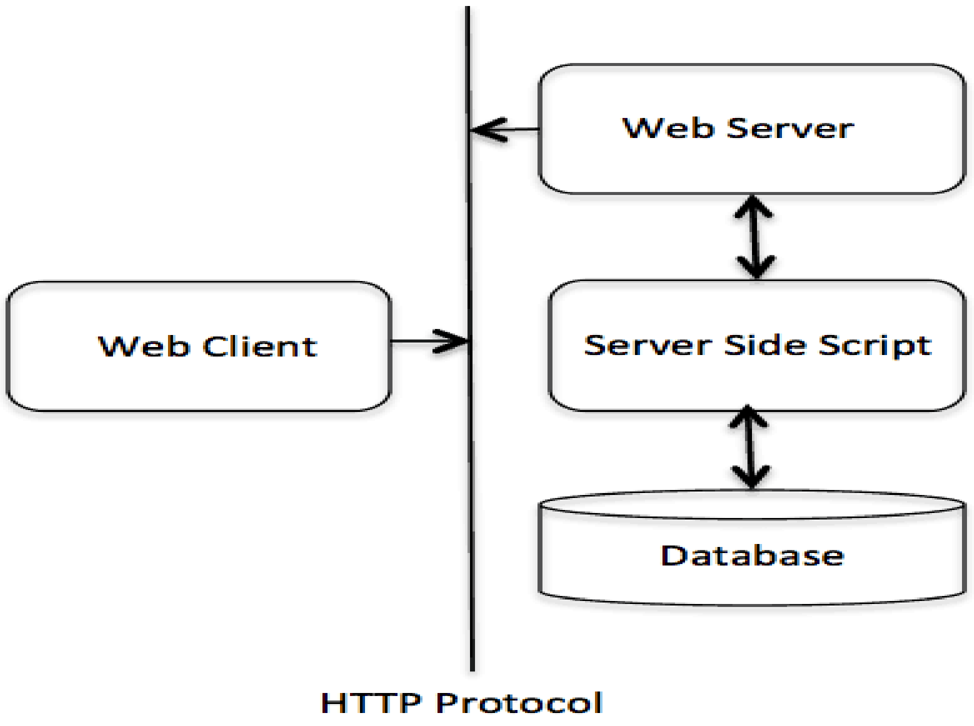
- **HTTP is connectionless:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.

- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.

- **HTTP is stateless:** As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

Basic Architecture

The following diagram shows a very basic architecture of a web application and depicts where HTTP sits:



The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

Client

The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.

Server

The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.

Roll no.:358

Q. What are various HTTP methods?

HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

Following are the methods of HTTP protocol:-

GET

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

HEAD

Same as GET, but transfers the status line and header section only.

POST

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

PUT

Replaces all current representations of the target resource with the uploaded content.

DELETE

Removes all current representations of the target resource given by a URI.

CONNECT

Establishes a tunnel to the server identified by a given URI.

OPTIONS

Describes the communication options for the target resource.

TRACE

Performs a message loop-back test along the path to the target resource.

HTTP basic authentication, form-based authentication, HTTPS mutual authentication.

HTTP basic authentication, form-based authentication is a relatively vulnerable authentication mechanism, since the content of the user dialogue is sent as plain text and the target server is not authenticated . With HTTPS mutual authentication both the client and the server use digital certificates to establish their identity, and authentication occurs over a channel protected by SSL.

In HTTP, URL begins with “http://” whereas URL starts with “https://”

HTTP uses port number 80 for communication and HTTPS uses 443

HTTP Works at Application Layer and HTTPS works at Transport Layer

In HTTP, Encryption is absent and Encryption is present in HTTPS

HTTP does not require any certificates and HTTPS needs SSL Certificates

In HTTP, the user sends data and the receiver receives it, irrespective of how the data flow is happening. Http is not concerned with how data is being transferred from start point to end point. In HTTPS, **SSL** differentiates the identity between a sender and a receiver. Thus starting and ending points are unique. Moreover the data flows in a very secured and encoded form, The SSL uses algorithms to encapsulate the data by hiding its original meaning.

Http is less expensive as it does not require any security certificates. Moreover it does not use SSL for data flow. Thus any website can go for http easily. HTTPS is more expensive as the encrypted certificates cost high. Thus many websites are not migrated to https yet.

Http has the ability to cache. Hence it is user friendly.
Https does not allow to cache thus many websites avoid using it.

Class: tybccs roll: 360

REST is not an architecture; rather, it is a set of constraints that creates a software architectural style, which can be used for building distributed applications.

the following constraints that define a RESTful system:

• Client-server: 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.

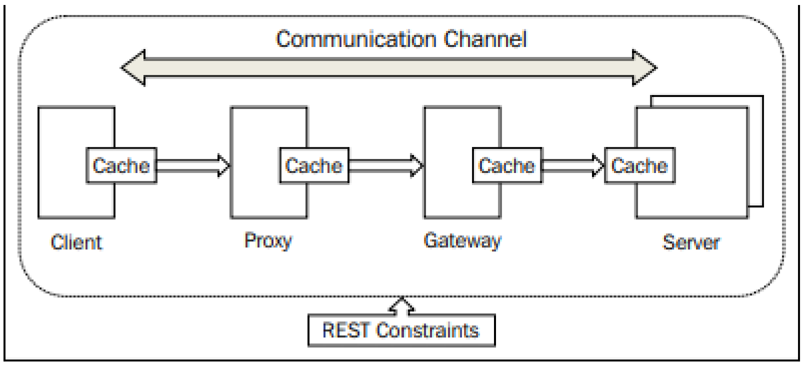
• Stateless: There should be no need for the service to keep users' sessions. In other words, each request should be independent of the others.

• Cacheable: This constraint has to support a caching system. Caching can avoid repeated round trips between the client and the server for retrieving the same resource.

• Uniform interface: 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.

• Layered system: The server can have multiple layers for implementation. This layered architecture helps to improve scalability by enabling load balancing.

• Code on demand: 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.



Web services that adhere to the REST architectural constraints are characterized as RESTful web services. Refer to the section, The REST architectural style, at the beginning of this chapter if you need a quick brush up on the architectural constraints for a RESTful system.

The RESTful web API or REST API is an API implemented using HTTP and the REST architectural constraints. Technically speaking, this is just another term for a RESTful web service.

The following are core elements that form a uniform interface for a RESTful system:

• Resources and their identifiers

• Representations of resources

• Generic interaction semantics for the REST resources

• Self-descriptive messages

• Hypermedia as the engine of an application state

Resources

A RESTful resource is anything that is addressable over the Web. By addressable, we mean resources that can be accessed and transferred between clients and servers. Subsequently, a resource is a logical, temporal mapping to a concept in the problem domain for which we are implementing a solution.

URI

A URI is a string of characters used to identify a resource over the Web. In simple words, the URI in a RESTful web service is a hyperlink to a resource, and it is the only means for clients and servers to exchange representations.

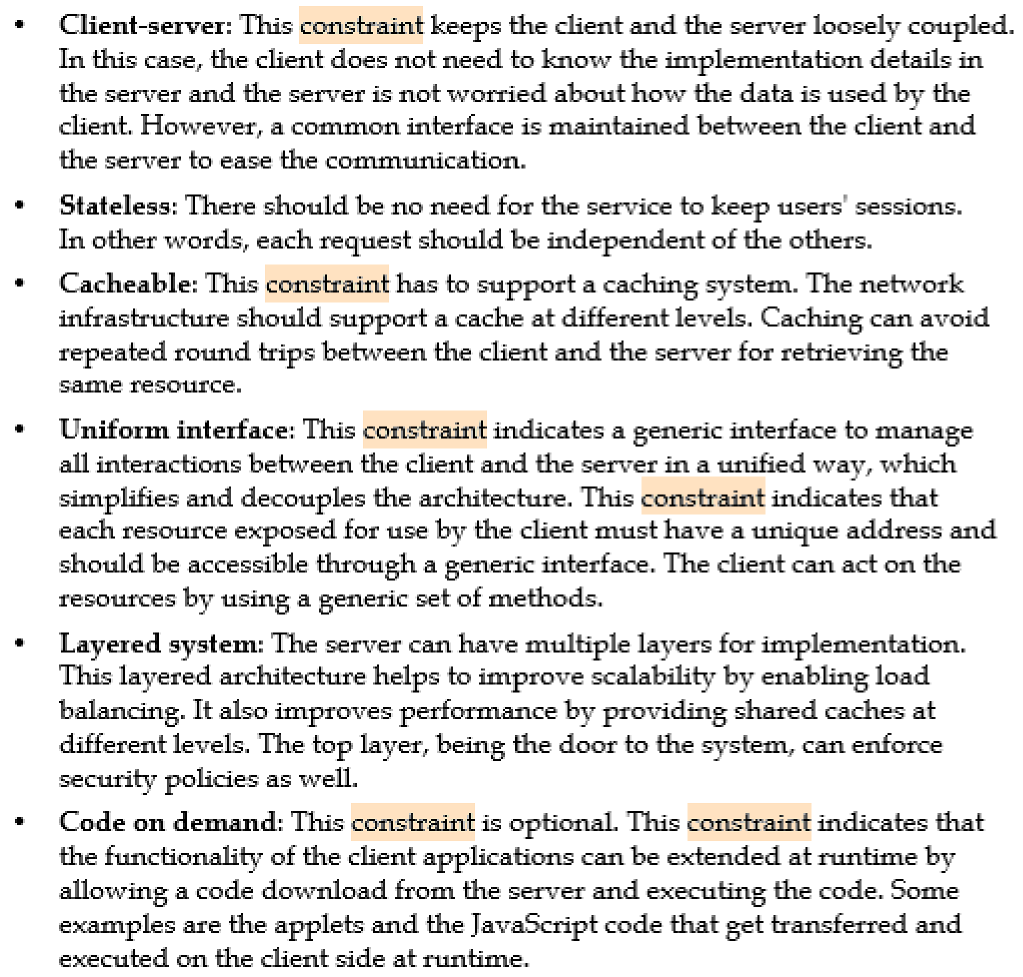
The representation of resources

A representation is a temporal state of the actual data located in some storage device at the time of a request. In general terms, it is a binary stream together with its metadata that describes how the stream is to be consumed by the client. The metadata can also contain extra information about the resource, for example, validation, encryption information, or extra code to be executed at runtime.

UNIT 2 – QUE NO.16 – 20

Q1. Describe the core constraint of RESTful system.

Ans.



- **Client-server**: 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.
- **Stateless**: There should be no need for the service to keep users' sessions. In other words, each request should be independent of the others.
- **Cacheable:** This constraint has to support a caching system. The network infrastructure should support a cache at different levels. Caching can avoid repeated round trips between the client and the server for retrieving the same resource.
- **Uniform interface**: 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.
- **Layered system**: 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.
- **Code on demand**: 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.

Q.2 Explain role of UDDI and SOAP in description and discovery of RESTful web services ?

Ans. SOAP: Conventional distributed communications protocols have a *symmetrical* requirement: both ends of the communication link would need to be implemented under the same distributed object model and would require the deployment of libraries developed in common. To address such limitations the Simple Object Access Protocol (SOAP) was developed. SOAP facilitates interoperability among a wide range of programs and platforms, making existing applications accessible to a broader range of users.

UDDI: To address the challenges of service registration and discovery, the Universal Description, Discovery, and Integration specification was created. UDDI is a cross-industry initiative to create a registry standard for Web service description and discovery together with a registry facility that supports the publishing and discovery processes. The UDDI specifications take advantage of World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) standards such as XML, HTTP, and Domain Name System (DNS) protocols. UDDI is designed for use by developer tools and applications that use Web services standards such as SOAP/XML and WSDL. UDDI provides a global, platform-independent, open framework making it easier to publish an enterprise's preferred means of conducting business, find trading partners, and interoperate with these trading partners over the Internet. By automating the registration and interrogation processes, UDDI enables service providers to describe their services and business processes in a global, open environment on the Internet, thus extending their reach. It also enables service clients to discover information about enterprises offering Web services; find descriptions of the Web services these enterprises provide; and, finally, find technical information about Web service interfaces and definitions of how the enterprises may interact over the Internet. It is a registry that contains relatively lightweight data.

Q.3 Explain any five java frameworks for building RESTful web services.

Ans. There are many tools and frameworks available in the market today for building RESTful web
services. You can use tools of your choice as long as the REST implementation meets the RESTful
architectural constraints.
There are some recent developments with resoect to the standardization of various framework API's by providing unified interfaces for a variety of implementations.

Some of the popular java frameworks are as follows:

1}**Jersey RESTful web service framework**: This framework is an open source framework for developing RESTful web service in Java. It serves as a JAVA-RS reference implementation. It provides its own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and clients development.

2}**Apache CXF**: This framework is an open source web services framework.CXF supports both JAX-WS and JAX-RS web services. It includes a "simple frontend" which allows creation of clients and endpoints without annotations.

3}**RESTEasy**: This framework is an open source project from JBoss, which provides various modules to help you build a RESTful web service. It is portable to Tomcat and many other app-server and has enhanced client framework, and is rich interceptor model.

4}**Restlet**: This framework is a lightweight, open source RESTful web services framework. It has good support for building both scalable RESTful web API's and lightweight REST clients, which suits mobile platform well. It's mapped to the REST and HTTP concepts, the framework can be used for both client and server-side development, using the same Java API which reduces both the learning curve and the software footprint. Restlet Framework is the most widely used open source solution for Java developers who want to create and use API's.

These are some of the Java Frameworks used for building RESTful web services.

Q.4 Write a short note on JSON object and JSON array

Ans. The JSON format is very simple by design. It is represented by the following two data structures:

• An unordered collection of name-value pairs (representing an object): 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 ({ }).

For instance, the JSON representation of a department object is as follows:

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

This example shows how you can represent various attributes of a department, such as departmentId, departmentName, and manager, in the JSON format.

• An ordered collection of values (representing an array): Arrays are enclosed in square brackets ([ ]), and their values are separated by a comma (,). Each value in an array may be of a different type, including another array or an object.

The following example illustrates the use of an array notation to represent employees working in a department. You may also see an array of locations in this example:

```
{"departmentName":"IT",
 "employees":[
 {"firstName":"John", "lastName":"Chen"},
 {"firstName":"Ameya", "lastName":"Job"},
 {"firstName":"Pat", "lastName":"Fay"}
 ],
 "location":["New York", "New Delhi"]
}
```

Q.5 How data exchange happens using JSON ? Explain

Ans. JSON (JavaScript Object Notation) is a text data format independent on platform. It is used for transferring data that can be organized into arrays or objects. JSON represents any data structure (number, string, boolean, null, object or array composed of these) written in text form (string). The hierarchical complexity is unlimited in theory. The JSON syntax is a valid syntax of the JavaScript language.

JSON data format is very simple, lightweight and easy to read. Compared to XML the JSON format is more efficient, readable and easier to process.

JSON data types

- **String**: Text string. The string must be entered into quotation marks. Example: "Temperature"

- **Number**: Number, integer or real number. The decimal separator is always a period.
Example: 1316, -1.23

- **Boolean**: Logical value.
Example: true, false

- **Null**: Value null (not set).
Example: null

- **Array**: Array (list of values). It is represented by square brackets **[ ]**. Array in JSON is a container with sequenced list of values. The values in array can be of any JSON data type including an object or an array.
Example:
[12, "Beethoven", 33.6, false]

- **Object**: Object (name-value pairs). It is represented by braces **{ }**. It is a container with data but no methods. Each data item (value) of the object has its key that is of the String type.
Example:
{"x": 100, "y": 100}

JSON itself is built on two universal structures:
Object
Array
Methods :
Pm.JsonParse method: Transforms JSON format text into object, array or elementary value
Pm.JsonStringify method: From object, array or elementary value creates text in JSON format

How restful web services can build with JAX-RS APIS ?
>Restful Web Services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a

client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.
(now lets write how it can be build using JAX-RS API'S)

Creating a RESTful Root Resource Class
Root resource classes are POJOs that are either annotated with @Path or have at least one method annotated with @Path or a request method designator, such as @GET, @PUT, @POST, or @DELETE. Resource methods are methods of a resource class annotated with a request method designator. This section explains how to use JAX-RS to annotate Java classes to create RESTful web services.

Developing RESTful Web Services with JAX-RS
JAX-RS is a Java programming language API designed to make it easy to develop applications that use the REST architecture.
The JAX-RS API uses Java programming language annotations to simplify the development of RESTful web services. Developers decorate Java programming language class files with JAX-RS annotations to define resources and the actions that can be performed on those resources. JAX-RS annotations are runtime annotations; therefore, runtime reflection will generate the helper classes and artifacts for the resource. A Java EE application archive containing JAX-RS resource classes will have the resources configured, the helper classes and artifacts generated, and the resource exposed to clients by deploying the archive to a Java EE server.

Annotation

Description

@Path
The @Path annotation's value is a relative URI path indicating where the Java class will be hosted: for example, /helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: /helloworld/{username}.
@GET
The @GET annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@POST
The @POST annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.

@PUT
The @PUT annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@DELETE
The @DELETE annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@HEAD
The @HEAD annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP HEAD requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@PathParam
The @PathParam annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation.
@QueryParam
The @QueryParam annotation is a type of parameter that you can extract for use in your resource class. Query parameters are extracted from the request URI query parameters.
@Consumes
The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client.
@Produces
The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, "text/plain".
@Provider
The @Provider annotation is used for anything that is of interest to the JAX-RS runtime, such as MessageBodyReader and MessageBodyWriter. For HTTP requests, the MessageBodyReader is used to map an HTTP request entity body to method parameters. On the response side, a return value is mapped to an HTTP response entity body by using a MessageBodyWriter. If the application needs to supply additional metadata, such as HTTP headers or a different status code, a method can return a Response that wraps the entity and that can be built using Response.ResponseBuilder.

Q.State design principle for building RESTFUL web services
The principles for the RESTFUL web services are as follows:
1. Uniform Interface Individual resources are identified using URLS. The resources (database) are themselves different from the representation (XML, JSON, HTML) sent to the client. The client can manipulate the resource through the representations provided they have the permissions. Each message sent between the client and the server is self-descriptive and includes enough information to describe how it is to be processed. The hypermedia that is

hyperlinks and hypertext act as the engine for state transfer.
2. Stateless Interactions none of the clients context is to be stored on the server side between the request. All of the information necessary to service the request is contained in the URL, query parameters, body or headers.
3. Cacheable Clients can cache the responses. The responses must define themselves as cacheable or not to prevent the client from sending the inappropriate data in response to further requests.
4. Client-Server The clients and the server are separated from each other thus the client is not concerned with the data storage thus the portability of the client code is improved while on the server side the server is not concerned with the client interference thus the server is simpler and easy to scale.
5. Layered System At any time client cannot tell if it is connected to the end server or to an intermediate. The intermediate layer helps to enforce the security policies and improve the system scalability by enabling load-balancing
6. Code on Demand an optional constraint where the server temporarily extends the functionality of a client by the transfer of executable code.

Q.State the importance and need of RESTFul web service
A service based on REST is called a RESTful service. REST is an architectural style not a protocol.
RESTful web services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.
Restful Web Services is a stateless client-server architecture where web services are resources and can be identified by their URIs.
REST Client applications can use HTTP GET/POST methods to invoke Restful web services. REST doesn't specify any specific protocol to use, but in almost all cases it's used over HTTP/HTTPS. When compared to SOAP web services, these are lightweight and doesn't follow any standard. We can use XML, JSON, text or any other type of data for request and response.
It is use to build Database applications which can be used with any client like a mobile app or javascript based Web Application. The Web/Mobile app is the client which makes RESTful API calls and GETS or POSTS the data. This is a convenient way to exchange data between several client applications and gives the developer to build applications without having to depend on the type of Client.
Fast: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
Language and Platform independent: RESTful web services can be written in any

programming language and executed in any platform.
Can use SOAP: RESTful web services can use SOAP web services as the implementation.
Permits different data format: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.
REST defines some 'verbs' in order to interact with the resources. Some of these are:
GET: to receive the resource representation
POST: to add some information to the resource
PUT: modify the resources
DELETE: delete the resources

Q 24.What are the different security practices and measures?
Ans:
Security practices:
The objective of Web services is to expose standardized interfaces to new and existing applications to allow the construction of applications and business processes that span organizational networks. One major concern is that Web services are designed to penetrate firewalls, evading their usefulness at the application layer. Web services are designed to go through network firewalls. Security is required to protect against XML and Web-services-related security threats. Security concern is that Web services are standardized and self-describing.
security also addresses confidentiality and privacy – using encryption to protect messages, guarantee message integrity, and provide audit functionality.
Web services security presents many similarities to application security with distributed technologies and is based on an understanding of:
◆ the resources being managed and protected by an application;
◆ the vulnerabilities relevant to the application’s base technology;
◆ the vulnerabilities relevant to the application’s specific logic; and
◆ the techniques that can be used to mitigate these risks.
Web services security concerns the WS-I Basic Security Profile. The WS-I Basic Security Profile into four broad categories:
Unauthorized access.
Unauthorized alteration of messages.
Man in the middle.
Denial-of-service attacks.
All the above issues point out that securing open, loosely coupled systems requires sophisticated security approach to support distributed clients (applications) and systems that may have different policies and possibly different security mechanisms.
Security measures:
Security requirements such as authentication , authorization, message integrity, confidentiality, and non-repudiation.
1.Authentication:
Authentication is the mechanism by which clients and service providers prove to one another that they are acting on behalf of specific users or systems.
2. Authorization:

Authorization mechanisms for distributed environments allow only authentic caller identities to access resources, such as hosts, files, Web pages, components, and database entries, to name a few.
3. Integrity and confidentiality: Integrity and confidentiality mechanisms can minimize such attacks.
4.Non-repudiation:

Non-repudiation is a property achieved through cryptographic methods to prevent an individual or entity from denying having performed a particular action related to data

5.Auditing:
Auditing is the practice of recording events, such as failed login attempts and denied requests to use a resource that may indicate attempts to violate enterprise security.
6.Security infrastructures:

Security infrastructures are distributed infrastructures used as a sound foundation on which applications can communicate and exchange data securely.

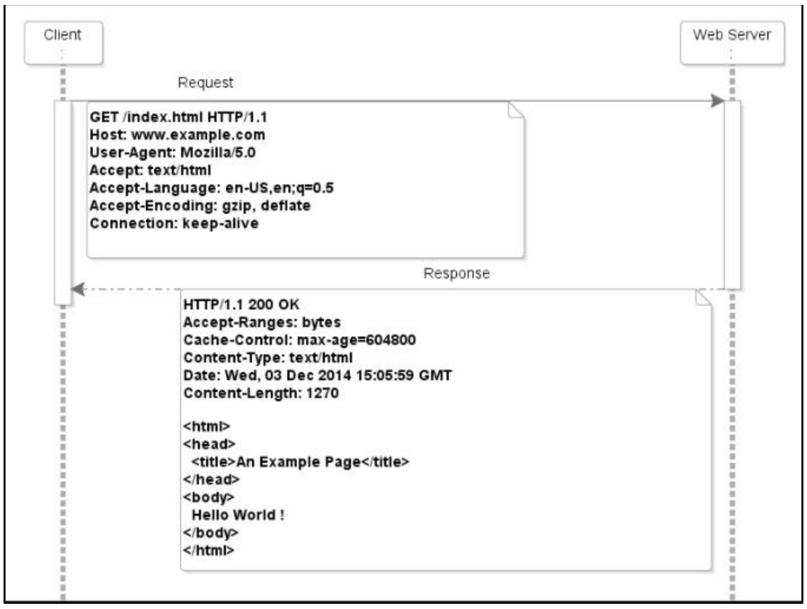
Q. What is HTTP status code? Explain representation of content type using HTTP header.

Ans: For every HTTP request, the server returns a status code indicating the processing status of the request. In this section, we will see some of the frequently used HTTP status codes.
• 1xx Informational: This series of status codes indicates informational content. This means that the request is received and processing is going on.
• 2xx Success: This series of status codes indicates the successful processing of requests.
• 3xx Redirection: This series of status codes indicates that the client needs to perform further actions to logically end the request.
• 4xx Client Error: This series of status codes indicates an error in processing the request.
-Representation of Content type using HTTP header :
The Content-Type header in an HTTP request or response describes the content type for the message body. The Accept header in the request tells the server the content types that the client is expecting in the response body. The content types are represented using the Internet media type. The Internet media type (also known as the MIME type) indicates the type of data that a file contains. Here is an example:
Content-Type: text/html
The Internet media types are broadly classified in to the following categories on the basis of the primary (or initial) Content-Type header:
• text: This type indicates that the content is plain text and no special software is required to read the contents. For instance, Content-Type: text/html indicates that the body content is html, and the client can use this hint to kick off an appropriate rendering engine while displaying the response.
• multipart: As the name indicates, this type consists of multiple parts of the independent data types. For instance, Content-Type: multipart/form- data is used for submitting forms that contain the files, non-ASCII data, and binary data.
• message: This type encapsulates more messages. It allows messages to contain other messages or pointers to other messages. For instance, the Content-Type: message/partial content type allows for large messages to be broken up into smaller messages. The full

message can then be read by the client (user agent) by putting all the broken messages together.
• image: This type represents the image data. For instance, Content-Type: image/png indicates that the body content is a .png image.
• audio: This type indicates the audio data. For instance, Content-Type: audio/mpeg indicates that the body content is MP3 or other MPEG audio.
• video: This type indicates the video data. For instance, Content-Type: video/mp4 indicates that the body content is MP4 video.
• application: This type represents the application data or binary data. For instance, Content-Type: application/json; charset=utf-8 designates the content to be in the JavaScript Object Notation (JSON) format, encoded with UTF-8 character encoding.

Q.26.Explain HTTP Request-Response model in detail. (Question no.26)

•HTTP works in a request-response manner. Let's take an example to understand this model better.
The following example illustrates the basic request-response model of communication between a web browser and a server over HTTP. The following sequence diagram illustrates the request and response messages sent between the client and the server:



```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Wed, 03 Dec 2014 15:05:59 GMT
Content-Length: 1270

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World !
</body>
</html>
```

Here is a detailed explanation of the sequence of actions shown in the preceding diagram.

•The user enters the following URL in the browser, http://www.example.com/ index.html, and then submits the request.

•The browser establishes a connection with the server and sends a request to the server in the form of a request method, URI, and protocol version, followed by a message containing request modifiers, client information, and possible body content. The sample request looks like the following:
GET /index.html HTTP/1.1

```
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/htmlAccept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

•Let's take a minute to understand the structure of the preceding message. The following code is what you see in the first lines of the request in our example:

```
GET /index.html HTTP/1.1
```

•The general format for the request line is an HTTP command, followed by the resource to retrieve, and the HTTP version supported by the client. The client can be any application that understands HTTP, although this example refers to a web browser as the client.

•The request line and other header fields must end with a carriage return character followed by a line feed character. In the preceding example, the browser instructstheservertogettheindex.htmlfilethroughtheHTTP 1.1protocol.
•The rest of the information that you may see in the request message is the HTTP header values for use by the server. The header fields are colon-separated key- value pairs in the plain-text format, terminated by a carriage return followed by a line feed character.

•The header fields in the request, such as the acceptable content types, languages, and connection type, are the operating parameters for an HTTP transaction. The server can use this information while preparing the response for the request. A blank line is used at the end of the header to indicate the end of the header portion in a request.
•The last part of an HTTP request is the HTTP body. Typically, the body is left blank unless the client has some data to submit to the server. In our example, the body part is empty as this is a GET request for retrieving a page from the server.
So far, we have been discussing the HTTP request sent by the client.

•Now, let's take a look at what happens on the server when the message is received. Once the server receives the HTTP request, it will process the message and return a response to the client. The response is made up of the reply status code from the server, followed by the HTTP header and a response content body:

```
 HTTP/1.1 200 OK
Accept-Ranges: bytes

Cache-Control: max-age=604800
Content-Type: text/html
Date: Wed, 03 Dec 2014 15:05:59 GMT
Content-Length: 1270
<html>
<head>
 <title>An Example Page</title>
</head>
<body>
 Hello World !
</body>
</html>.
```

•The first line in the response is a status line. It contains the HTTP version that the server is using, followed by a numeric status code and its associated textual phrase. The status code indicates one of the following parameters: informational codes, success of the request, client error, server error, or redirection of the request. In our example, the status line is as follows:

HTTP/1.1 200 OK

•The next item in the response is the HTTP response header. Similar to the request header, the response header follows the colon-separated name-value pair format terminated by the carriage return and line feed characters.

•The HTTP response header can contain useful information about the resource being fetched, the server hosting the resource, and some parameters controlling the client behavior while dealing with the resource, such as content type, cache expiry, and refresh rate.
•The last part of the response is the response body. Upon the successful processing of the request, the server will add the requested resource in the HTTP response body. It can be HTML, binary data, image, video, text, XML, JSON, and so on. Once the response body has been sent to the requestor, the HTTP server will disconnect if the connection created during the request is not of the keep-alive type (using the Connection: keep-alive header).

Q.27. The core architectural elements of a RESTful system

A uniform is fundamental to the architecture of any RESTful system. In plain words, this term refers to a generic interface to manage all interactions between a client and a server in a unified way. All resources (or business data) involved in the client-server interactions are dealt with by a fixed set of operations. The following are core elements that form a uniform interface for a RESTful system:
• Resources and their identifiers
• Representations of resources
• Generic interaction semantics for the REST resources
• Self-descriptive messages
• Hypermedia as the engine of an application state

RESTful RESOURCE
A RESTful resource is anything that is addressable over the Web. By addressable, we mean resources that can be accessed and transferred between clients and servers. Subsequently, a resource is a logical, temporal mapping to a concept in the problem domain for which we are implementing a solution.

URI
A URI is a string of characters used to identify a resource over the Web. In simple words, the URI in a RESTful web service is a hyperlink to a resource, and it is the only means for clients and servers to exchange representations.
The client uses a URI to locate the resources over Web and then, sends a request to the server and reads the response. In a RESTful system, the URI is not meant to change over time as it may break the contract between a client and a server.

Q 28-Describe the discovery and description of RESTful web services?

Answer-There are a variety of approaches to describe and document RESTful web API’s. An API documentation of RESTful web services provides a standard language-independent interface, which can be used by both humans and machines to discover the capabilities of the API’s without accessing the source code. Some solutions available today for describing, producing, consuming and visualizing RESTful webservices are

- WADL (web application description language): it was discovered by Sun in 2009, it is an XML description of HTTP-based web applications. It models the resources provided by a RESTful web service with relationship between the resources. It allows you to clearly represent the media types used for the request and response contents.
- RAML (RESTful API modelling language): it was first discovered by the RAML workgroup in 2013, it provides human-readable and machine processable documentation for your RESTful webservice. It is a text file with recommended extensions of .raml. It helps you to document resources, methods, parameters, responses, media types and other HTTP constructs. RAML is built on standards such as Yaml Aint’t Markup Language (YAML) which is a human friendly data serialization standard that works with any programming language.
- Swagger: it is developed by Wordnik and the first version was released in 2011, it offers a specification and complete framework implementation for describing RESTful webservices. It works with popular programming languages such as Java, Scala, Clojure, Groovy, JavaScript and .Net. Swagger framework has three major components which are server, client and user interface.

Q. 29.Write a note on media type annotation?

Ans**: Annotations for specifying request-respons**

media types.

The Content-Type header field in HTTP describes the body's content type present

in the request and response messages. The content types are represented using the

standard Internet media types. A RESTful web service makes use of this header

field to indicate the type of content in the request or response message body.

Media Type Annotations

There are two media type annotations:

•Consumes.

• Produces.

•**Consumes:**

The @javax.ws.rs.Consumes annotation defines the Internet media type(s) that the

resource class methods can accept. You can define the @Consumes annotation either

at the class level (which will get defaulted for all methods) or the method level. The

method-level annotations override the class-level annotations. The possible Internet media types that a REST API can consume are as follows:

• application/atom+xml

• application/json

•**Produces:**

The @javax.ws.rs.Produces annotation is used for defining the Internet media type(s) that a REST resource class method can return to the client. You can define this either at the class level (which will get defaulted for all methods) or the method level. The method-level annotations override the class-level annotations. The possible Internet media types that a REST API can produce are as follows:

• application/atom+xml

• application/json

• application/octet-stream

• application/svg+xml

• application/xhtml+xml

• application/xml

• text/html

• text/plain

• text/xml

Q.30.What are core Rest Architectural Elements?

Rest Architectural style defines parts of Restful service as key conceptual elements. For e.g. while we make a Rest request to a Server, the information returned by the Server is called a Resource. Let us take a quick look at the key conceptual elements in a RESTful Web Service.

Elements example are:-

1) Resource :-

Information stored on a Server, which can be requested by a client. It could be Weather info or may be employee details. A Resource can be

a)Temporal

b) Static

Temporal resource is one that keeps changing with time. For e.g. the Weather. A Static resource is one that stays constant over longer time durations. For e.g. a webpage containing some static text. Like the one you are reading now can just be a Static resource on the server.

2). Resource Identifier. :-

Now that we have a resource defined, we need to uniquely identify the resource. That is actually the complete URL.

3) Representation :-
A resource is the actual data. Now this data can be represented as an XML, HTML or may be simple text. That is what is called a Representation.

4) Representation Metadata:-
In order for the Clients to specify and process a resources given in a particular Representation (XML or HTML etc) some extra data (Metadata) needs to be passed in the request.

---@@@@@@@@@---

Write a short note on http basic authentication

Ans:

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and password) from a client. The client passes the authentication information to the server in an Authorization header. The authentication information is in base-64 encoding.

If a client makes a request for which the server expects authentication information, the server sends an HTTP response with a 401 status code, a reason phrase indicating an authentication error, and a WWW-Authenticate header. Most web clients handle this response by requesting a user ID and password from the end user.
The format of a WWW-Authenticate header for HTTP basic authentication is:

```
WWW-Authenticate: Basic realm="Our Site"
```

When the web client has obtained a user ID and password, it resends the original request with an Authorization header. Alternatively, the client can send the Authorization header when it makes its original request, and this header might be accepted by the server, avoiding the challenge and response process.
The format of the Authorization header is:

```
Authorization: Basic userid:password
```

HTTP provides a general framework for access control and authentication. The most common HTTP authentication is based on the "Basic" schema. This page shows an introduction to the HTTP framework for authentication and shows how to restrict access to your server using the HTTP "Basic" schema

Securing Web Services with open Authorization

Because of its nature (loosely coupled connections) and its use of open access (mainly HTTP), SOAP implemented by Web services adds a new set of requirements to the security landscape. Web services security includes several aspects:

Authentication—verifying that the user is who he claims to be. A user's identity is verified based on the credentials presented by that user, such as:

Authorization (or Access Control)—Granting access to specific resources based on an authenticated user's entitlements. Entitlements are defined by one or several attributes. An attribute is the property or characteristic of a user, for example, if "Marc" is the user, "conference speaker" is the attribute.

Confidentiality, privacy—keeping information secret. Accesses a message, for example a Web service request or an email, as well as the identity of the sending and receiving parties in a confidential manner. Confidentiality and privacy can be achieved by encrypting the content of a message and obfuscating the sending and receiving parties' identities.

Integrity, non-repudiation—making sure that a message remains unaltered during transit by having the sender digitally sign the message. A digital signature is used to validate the signature and provides non-repudiation. The timestamp in the signature prevents anyone from replaying this message after the expiration.

Web services security requirements also involve credential mediation (exchanging security tokens in a trusted environment), and service capabilities and constraints (defining what a Web service can do, under what circumstances).

Transport-level Security

Secure Socket Layer (SSL), otherwise known as Transport Layer Security (TLS), the Internet Engineering Task Force (IETF) officially standardized version of SSL, is the most widely used transport-level data-communication protocol providing:

Authentication (the communication is established between two trusted parties).
Confidentiality (the data exchanged is encrypted).
Message integrity (the data is checked for possible corruption).
Secure key exchange between client and server.
SSL provides a secure communication channel, however, when the data is not "in transit," the data is not protected. This makes the environment vulnerable to attacks in multi-step transactions. (SSL provides point-to-point security, as opposed to end-to-end security.)

Application-level Security

Application-level security complements transport-level security. Application-level security is based on XML frameworks defining confidentiality, integrity, authenticity; message structure; trust management and federation.

Data confidentiality is implemented by XML Encryption. XML Encryption defines how digital content is encrypted and decrypted, how the encryption key information is passed to a recipient, and how encrypted data is identified to facilitate decryption.

Data integrity and authenticity are implemented by XML Signature. XML Signature binds the sender's identity (or "signing entity") to an XML document. Signing and signature verification can be done using asymmetric or symmetric keys.

Write a Note on Swagger?

Swagger is a specification for documenting REST API. It specifies the format (URL, method, and representation) to describe REST web services. It provides also tools to generate/compute the documentation from application code.it mean that As an application developer, you write web services using your favorite framework, Swagger scans your code and exposes the documentation on some URL. Any client can consume this URL (which comes as XML or JSON documents) and learn how to use your REST web services: which HTTP methods to call on which URL, which input documents to send, which status code to expect, etc.

Advantages

With the Swagger framework, the server, client and documentation team can be in synchronization simultaneously.

The Swagger UI framework allows both implementers and users to interact with the API. It gives clear insight into how the API responds to parameters and options.

Swagger responses are in JSON and XML . Swagger implementations are available for various technologies like Scala, Java, and HTML5.

Client generators are currently available for Scala, Java, JavaScript, Ruby, PHP with more client support underway.

. Swagger Integration with Spring MVC.

Mentioned are some of the following Annotations provided from wordnik Spring MVC for the Swagger project to document the Rest

APIs.@Api

@ApiClass

@ApiError

@ApiErrors

@ApiOperation

@ApiParam

@ApiParamImplicit

@ApiParamsImplicit

@ApiProperty

@ApiResponse

@ApiResponses

@ApiModel

What is RAML? Explain its structure?

RAML stands for RESTful API Modelling Language. It's a way of describing practically-RESTful APIs in a way that's highly readable by both humans and computers. It focuses on cleanly describing resources, methods, parameters, responses, media types, and other HTTP constructs that form the basis for modern APIs that obey many, though perhaps not all, RESTful constraints. The aim is to help our current API ecosystem and solve immediate problems, and then gently encourage ever-better API patterns.

Structure:

Markup Language

This specification uses YAML 1.2 as its underlying format. YAML is a human-readable data format that aligns well with the design goals of this specification. As in YAML, all nodes such as keys, values, and tags, are case-sensitive.RAML API definitions are YAML 1.2-compliant documents that begin with a REQUIRED YAML-comment line that indicates the RAML version, as follows:The media type application/raml+yaml and its associated file extension .raml SHALL be used to designate files containing RAML API definitions, RAML fragments, and files that contain RAML markup.

The Root of the Document

The root section of the RAML document describes the basic information about an API, such as its title and version. The root section also defines assets used elsewhere in the RAML document, such as types and traits. Processors MUST preserve the order of nodes of the same kind within the same node of the definition tree. Examples of such nodes are resources that appear at the same level of the resource tree are :

baseUri?	A URI that serves as the base for URIs of all resources. Often used as the base of the URL of each resource containing the location of the API. Can be a template URI.

baseUriParameters?	Named parameters used in the baseUri (template).

User Documentation

The OPTIONAL documentation node includes a variety of documents that serve as user guides and reference documentation for the API. Such documents can clarify how the API works or provide technical and business context.

The value of the documentation node is a sequence of one or more documents. Each document is a map that MUST have exactly two key-value pairs described in following table:

This example shows an API definition having two user documents.

Welcome to the _Zencoder API_ Documentation. The _Zencoder API_ allows you to connect your application to our encoding service and encode videos without going through the web interface.

Base URI and Base URI Parameters

The OPTIONAL baseUri node specifies a URI as an identifier for the API as a whole, and MAY be used the specify the URL at which the API is served (its service endpoint), and which forms the base of the URLs of each of its resources. If the baseUri value is a Template URI, the following reserved base URI parameter is available. The baseUriParameters node has the same structure and semantics as the uriParameters node on a resource node, except that it specifies parameters in the base URI rather than the relative URI of a resource. example: baseUri: //api.test.com//common//

Protocols

The OPTIONAL protocols node specifies the protocols that an API supports.

Default Security

Specifying the OPTIONAL secured By node sets the default security schemes for, and protects, every method of every resource in the API. The value of the node is an array of security scheme names. See section Applying Security Schemes for more information, including how to resolve the application of multiple security schemes through inheritance. Example: oauth_1_0: !include securitySchemes/oauth_1_0.raml

Question: Explain authorizing restful web service access via security API

Ans: authorization refers to rules that determine "what user is allowed to do" and what he is not e.g. a normal user can post a message in any public group, but users only with editor role will be able to delete something. Authorization is often seen as both the introductory setting up of permissions by a system administrator and the checking of the permission values that have already been set up when a user is getting access to the system.

Apart from above concepts, you will usually need to secure your RESTful APIs in your company using below methods.

Let's note down some important points while designing security for your RESTful web services.

Use only HTTPS protocol so that your whole communication is always encrypted.

Never send auth credentials or API keys as query param. They appear in URL and can be logged or tracked easily.

Use hardest encryption level always. It will help in having more confidence.

For resources exposed by RESTful web services, it's important to make sure any PUT, POST, and DELETE request is protected from Cross Site Request Forgery.

Always validate the input data asap it is received in server method. Use only primitive data as input parameter as much as possible.

Rely on framework provided validation features as they are tested by large community already.

<u>Question: </u>Explain overview of WADL structure

<u>Answer</u>

Web Application Description Language (WADL) is an XML description of HTTP based web applications such as RESTful web services. WADL models the resources provided by a RESTful web service with relationships between the resources. It also allows you to clearly represent the media types used for the request and response contents.

The WADL schema that you use for building a WADL file is based on the WADL specification.

The top-level element in a WADL document is the application. It contains global information about RESTful web services, such as links to schema definition and documentation. Here is a quick summary of the elements that you see in WADL:

• <resources>: This element comes as wrapper for all resources exposed by a RESTful web application. The base attribute for resources identifies the base path of all resources in the API. • <resource>: This element appears as a child of the resources element. This element is used for describing a set of resources, each identified by a URI. This element can optionally have <param> to describe the path parameters present in the request, if any. • <method>: A resource may have one or more <method> definitions. This element defines the HTTP methods, such as GET, POST, PUT, and DELETE, available to the resource. Note that the presence of this element in WADL represents a REST resource method in a RESTful web service.

• <request>: A <method> definition can optionally include this element that describes an input to the resource method. A request can have one or more of the following elements as its child: ° <doc>: This element is used for documenting the parameters ° <representation>: This element specifies the Internet media type for the payload present in the body of the request ° <param>: This element specifies the query or header parameter present in the request • <response>: This element specifies the result of invoking an HTTP method on a resource. The optional status code present in the response describes the list of the HTTP status codes associated with the response. A response can have one or more of the following elements as its child: ° <doc>: This element is used for documenting the <response> header parameters ° <representation>: This element describes the Internet media type for the response content returned by the API ° <param>: This element specifies the HTTP header parameters present in the response.

Example:

This method returns department object for the department ID passed by client:

```
@GET
@PATH("{id}")
@Produces(:application/json")
public Department  findDepartment (@PathParam("id") short id){
//Method body is omitted for brevity
}
```

<u>Question-</u>How Open data Protocol is used with restful web APIs?

<u>Answer:</u>

1.REST is an architecture style for sending messages back and forth from the client to the server over HTTP.

2.The REST architectural style does not define any standard for querying and updating data.

3.Open Data Protocol (OData) defines a web protocol for querying and updating data via RESTful web APIs uniformly.
4. Many companies, including Microsoft, IBM, and SAP, support OData today, and it is governed by Organization for the Advancement of Structured Information Standards (OASIS).
5.The OData specification defines a set of recommendations for forming the URIs that identify OData-based REST APIs. Currently, the latest release of OData is Version 4.0.
6.The URI for an OData service may take up to three parts, as follows:

a.• Service root: This part identifies the root of an OData service.

b.• Resource path: This part identifies the resources exposed by an OData service.

c. • Query string options: This part identifies the query options (built-in or custom) for the resource.

Here is an example:
http://localhost:8080/hrapp/odata/Departments?$top=2&$orderby=Location
where,

http://localhost:8080/hrapp/odata - Service Root

/Departments?- Resource Root

$top=2&$orderby=Location - Query String Options

7. Resources from OData RESTful APIs are accessible via the HTTP GET request.
8. The result that you may get from the OData REST API server in response to the preceding call will be structured in accordance with the OData protocol specification. This keeps the client code simple and reusable.
9. OData supports various kinds of query options as query parameters. For instance, $orderby can be used for sorting the query results.
10. Some of the frequently used query options are listed below:

a.$filter - This option allows the client to filter a collection of resources

b. $expand - This option includes the specified (child) resource in line with the retrieved resources

c. $select - This option includes the supplied attributes alone in the resulting entity.

d.$orderby - This option sorts the query result by one or more attributes.

e. $top - This option returns only the specified number of items (from the top) in the result collection.

f. $skip - This option indicates how many items need to be skipped from the top while returning the result.

g.$count - This option indicates the total number of items in the result.

<u>Question:</u> Distinguish the features offered in WADL,RAML and Swagger

<u>Answer:</u>

- Web Application Description Language (WADL) is an XML description of HTTPbased web applications such as RESTful web services. RESTful API Modeling Language (RAML) provides human-readable and machine-processable documentation for your RESTful web services. Swagger offers a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTful web services.
- WADL was submitted to the World Wide Web Consortium (W3C) by Sun in 2009 but has not been standardized yet. RAML was first proposed in 2013 by RAML

Workgroup.Swagger was initially developed by Wordnik (a property of Reverb) for meeting their in-house requirements, and the first version was released in 2011.

- An WADL file is a text file with the recommended extension of .xml. A RAML file is a text file with the recommended extension of .raml. Swagger file is a text file with the recommended extension of json/yaml.
- WADL does not support Commercial Offering while RAML and Swagger support it.
- WADL does not cover authentication constructs for REST APIs. RAML covers authentication for Basic, Digest, OAuth 1, and OAuth 2 and Swagger authenticates for Basic, API Key, and OAuth 2.
- For WADL,some vendors think that XML is not the right way to go for describing RESTful web APIs because of the complexity involved in parsing XMLs. RAML allows you to develop and test the API client independent of the actual RESTful web service implementation. Swagger support for different languages and matured tooling support have really grabbed the attention of many API vendors.
- WADL supports only Java language whereas RAML supports JS, Java, Node, PHP, Python, and Ruby and Swagger framework works with many of the popular programming languages, such as Java, Scala, Clojure, Groovy, JavaScript, and .Net
- User cannot generate code for Java server for WADL whereas for RAML and Swagger user can generate code for Java server.

Question: Explain OAuth 2.0 with an example

Answer:

1)OAuth 2.0 is an open authorization protocol which enables applications to access each others data.
2)For instance, a game application can access a users data in the Facebook application

3)The game web application asks the user to login to the game via Facebook. The user logs into Facebook, and is sent back to the game. The game can now access the users data in Facebook, and call functions in Facebook on behalf of the user

4)OAuth 2.0 is a replacement for OAuth 1.0, which was more complicated. OAuth 1.0 involved certificates etc. OAuth 2.0 is more simple. It requires no certificates at all, just SSL / TLS.

5)OAuth 2.0 is a simple protocol that allows to access resources of the user without sharing passwords.

6)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.

7)If your favorite sites are connected to the central hub and the central account is hacked, then it will lead to serious effects across several sites instead of just one.

Question: How JAX-RS is useful in developing RESTFUL web services

Answer:

1)

JAX-RS stands for JAVA API for RESTful Web Services. JAX-RS is a JAVA based programming language API and specification to provide support for created RESTful Web Services.

2)JAX-RS uses annotations available from Java SE 5 to simplify the development of JAVA based web services creation and deployment.

3)

Following are the most commonly used annotations to map a resource as a web service resource.

Sr.No.	Annotation & Description
1	**@Path** Relative path of the resource class/method.
2	**@GET** HTTP Get request, used to fetch resource.
3	**@PUT** HTTP PUT request, used to create resource.
4	**@POST** HTTP POST request, used to create/update resource.
5	**@DELETE** HTTP DELETE request, used to delete resource.
6	**@HEAD** HTTP HEAD request, used to get status of method availability.
7	**@Produces** States the HTTP Response generated by web service. For example, APPLICATION/XML, TEXT/HTML, APPLICATION/JSON etc.
8	**@Consumes** States the HTTP Request type. For example, application/x-www-formurlencoded to accept

	form data in HTTP body during POST request.
9	**@PathParam** Binds the parameter passed to the method to a value in path.
10	**@QueryParam** Binds the parameter passed to method to a query parameter in the path.
11	**@MatrixParam** Binds the parameter passed to the method to a HTTP matrix parameter in path.
12	**@HeaderParam** Binds the parameter passed to the method to a HTTP header.
13	**@CookieParam** Binds the parameter passed to the method to a Cookie.
14	**@FormParam** Binds the parameter passed to the method to a form value.
15	**@DefaultValue** Assigns a default value to a parameter passed to the method.
16	**@Context** Context of the resource. For example, HTTPRequest as a context.

41)Annotations are like meta-tags that you can add to the code and apply to package declarations, type declarations, constructors, methods, fields, parameters, and variables. They provide helpful ways to indicate whether the methods are dependent on other methods

Some of the common annotations used to develop a Web service are described in the following sections.

@Path(‘Path‘)

@GET

@POST

@PUT

@DELETE

@Produces

@Consumes

@PUT

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP PUTrequest only.

@DELETE

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP DELETErequest only.

@Produces

Its a method or field level annotation, This tells which MIME type is delivered by the method annotated with @GET. I mean when ever we send a HTTP GET request to our RESTful service, it will invokes particular method and produces the output in different formats. There you can specifies in what are all formats (MIME) your method can produce the output, by using @produces annotation.
Remember: We will use @Produces annotation for GET requests only.

@Consumes

This is a class and method level annotation, this will define which MIME type is consumed by the particular method. I mean in which format the method can accept the input from the client.

@Path() Annotation

Its a Class & Method level of annotation

This will check the path next to the base URL

@GET

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP GET request only, i mean if we annotate our method with @GET, the execution flow will enter that following method if we send GET request from the client

@POST

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP POSTrequest only.

@PUT

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP PUTrequest only.

@DELETE

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP DELETErequest only.

@Produces

Its a method or field level annotation, This tells which MIME type is delivered by the method annotated with @GET. I mean when ever we send a HTTP GET request to our RESTful service, it will invokes particular method and produces the output in different formats. There you can specifies in what are all formats (MIME) your method can produce the output, by using @produces annotation.
Remember: We will use @Produces annotation for GET requests only.

@Consumes

This is a class and method level annotation, this will define which MIME type is consumed by the particular method. I mean in which format the method can accept the input from the client.

- **What is windows communication Foundation?**
  Ans: Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages 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. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data. A few sample scenarios include:

- A secure service to process business transactions.
- A service that supplies current data to others, such as a traffic report or other monitoring service.
- A chat service that allows two people to communicate or exchange data in real time.
- A dashboard application that polls one or more services for data and presents it in a logical presentation.
- Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.
- A Silverlight application to poll a service for the latest data feeds.

While creating such applications was possible prior to the existence of WCF, WCF makes the development of endpoints easier than ever. In summary, WCF is designed to offer a manageable approach to creating Web services and Web service clients. One consequence of using WS standards is that WCF enables you to create service oriented applications. Service-oriented architecture (SOA) is the reliance on Web services to send and receive data. The services have the general advantage of being loosely-coupled instead of hard-coded from one application to another. A loosely-coupled relationship implies that any client created on any platform can connect to any service as long as the essential contracts are met.

2)Explain the features of WCF in detail?

WCF stands for Windows Communication Foundation. The elementary feature of WCF is interoperability. It is one of the latest technologies of Microsoft that is used to build service-oriented applications. Based on the concept of message-based communication, in which an HTTP request is represented uniformly, WCF makes it possible to have a unified API irrespective of diverse transport mechanisms.
WCF was released for the first time in 2006 as a part of the .NET framework with Windows Vista, and then got updated several times. WCF 4.5 is the most recent version that is now widely used.
A WCF application consists of three components −
• WCF service,
• WCF service host, and
• WCF service client.
WCF platform is also known as the Service Model.
Fundamental Concepts of WCF
Message
This is a communication unit that comprises of several parts apart from the body. Message instances are sent as well as received for all types of communication between the client and the service.
Endpoint
It defines the address where a message is to be sent or received. It also specifies the communication mechanism to describe how the messages will be sent along with defining the set of messages. A structure of an endpoint comprises of the following parts −
Address (incomplete pl. Check from site)

3) Write a note on Fundamental Windows Communication Foundation Concepts?

WCF Fundamentals

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

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.

Another required element in the communication stack is the encoding that specifies how any given message is formatted. WCF provides the following encodings:

• 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.

More encoding mechanisms (for example, a compression encoding) can be added using the built-in extension points of WCF.

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

4) WCF Terms & Concepts.

Terms	**Concepts**
message	A self-contained unit of data that can consist of several parts, including a body and headers.
endpoint	It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that describes how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract) that describes what message can be sent.
application endpoint	An endpoint exposed by the application and that corresponds to a service contract implemented by the application.
infrastructure endpoint	An endpoint that is exposed by the infrastructure to facilitate functionality that is needed or provided by the service that does not relate to a service contract.
address	Specifies the location where messages are received. It is specified as a Uniform Resource Identifier(URI). The URI schema part names the transport mechanism to use to reach the address, such as HTTP and TCP.
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. 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.
binding element	Represents a particular piece of the binding, such as a transport, an encoding, an implementation of an infrastructure-level protocol or any other component of the

	communication stack.
behaviors	A component that controls various run-time aspects of a service, an endpoint, a particular operation, or a client. Behaviors are grouped according to scope: common behaviors affect all endpoints globally, service behaviors affect only service-related aspects, endpoint behaviors affect only endpoint-related properties, and operation-level behaviors affect particular operations.
system-provided bindings	WCF includes a number of system-provided bindings. These are collections of binding elements that are optimized for specific scenarios. These predefined bindings save time by presenting only those options that can be correctly applied to the specific scenario.
configuration versus coding	Control of an application can be done either through coding, through configuration, or through a combination of both.
service operation	A procedure defined in a service's code that implements the functionality for an operation. This operation is exposed to clients as methods on a WCF client.
service contract	Ties together multiple related operations into a single functional unit. The contract can define service-level settings, a corresponding callback contract, and other such settings.
operation contract	An operation contract defines the parameters and return type of an operation. The operations can be modeled as taking a single message and returning a single message, or as taking a set of types and returning a type. In the latter case, the system will determine the format for the messages that need to be exchanged for that operation.
message contract	Describes the format of a message. For example, it declares whether message elements should go in headers versus the body, what level of security should be applied to what elements of the message, and so on.
fault contract	Can be associated with a service operation to denote errors that can be returned to the caller. An operation can have zero or more faults associated with it.
data contract	The descriptions in metadata of the data types that a service uses. This enables others to interoperate with the service.
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.
self-hosted service	A service that runs within a process application that the developer created.
hosting	An application that is designed to host services. These include Internet

process	Information Services (IIS), Windows Activation Services (WAS), and Windows Services.
client application	A program that exchanges messages with one or more endpoints. The client application begins by creating an instance of a WCF client and calling methods of the WCF client.
channel	A concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration.
security	In WCF, includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources).
transport security mode	Specifies that confidentiality, integrity, and authentication are provided by the transport layer mechanisms (such as HTTPS).
message security mode	Specifies that security is provided by implementing one or more of the security specifications. Each message contains the necessary mechanisms to provide security during its transit, and to enable the receivers to detect tampering and to decrypt the messages. In this sense, the security is encapsulated within every message, providing end-to-end security across multiple hops.

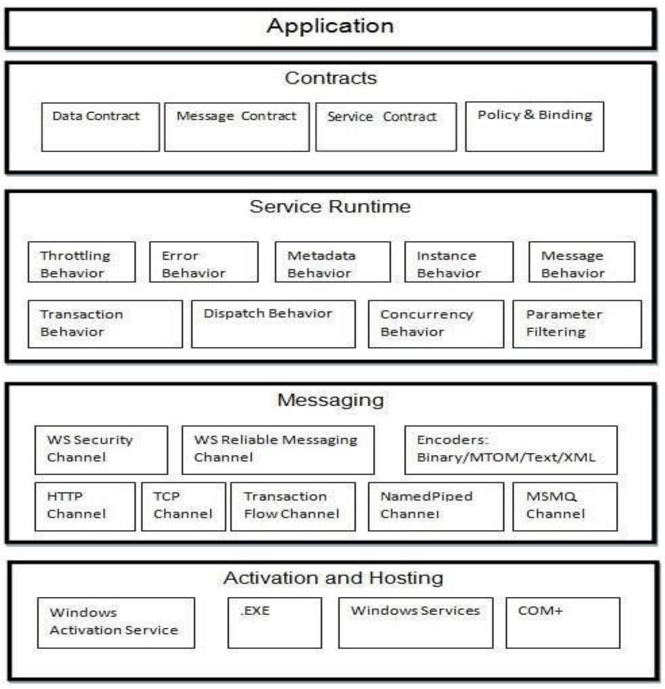
Contracts and Descriptions =Contracts define various aspects of the message system. The data contract describes every parameter that makes up every message that a service can create or consume. The message parameters are defined by XML Schema definition language (XSD) documents, enabling any system that understands XML to process the documents. The message contract defines specific message parts using SOAP protocols, and allows finer-grained control over parts of the message, when interoperability demands such precision. The service contract specifies the actual method signatures of the service, and is distributed as an interface in one of the supported programming languages, such as Visual Basic or Visual C#.Policies and bindings stipulate the conditions required to communicate with a service. For example, the binding must (at a minimum) specify the transport used (for example, HTTP or TCP), and an encoding. Policies include security requirements and other conditions that must be met to communicate with a service.

Service Runtime=The service runtime layer contains the behaviors that occur only during the actual operation of the service, that is, the runtime behaviors of the service. Throttling controls how many messages are processed, which can be varied if the demand for the service grows to a preset limit. An error behavior specifies what occurs when an internal error occurs on the service, for example, by controlling what information is communicated to the client. (Too much information can give a malicious user an advantage in mounting an attack.) Metadata behavior governs how and whether metadata is made available to the outside world. Instance behavior specifies how many instances of the service can be run (for example, a singleton specifies only one instance to process all messages). Transaction behavior enables the rollback of transacted operations if a failure occurs. Dispatch behavior is the control of how a message is processed by the WCF infrastructure.Extensibility enables customization of runtime processes. For example, message inspection is the facility to inspect parts of a message, and parameter filtering enables preset actions to occur based on filters acting on message headers.

Messaging=The 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 operate on messages and message headers. This is different from the service runtime layer, which is primarily concerned about processing the contents of message bodies.

There are two types of channels: transport channels and protocol channels.Transport channels read and write messages from the network (or some other communication point with the outside world). Some transports use an encoder to convert messages (which are represented as XML Infosets) to and from the byte stream representation used by the network. Examples of transports are HTTP, named pipes, TCP, and MSMQ. Examples of encodings are XML and optimized binary.

Protocol channels implement message processing protocols, often by reading or writing additional headers to the message. Examples of such protocols include WS-Security and WS-Reliability.



The messaging layer illustrates the possible formats and exchange patterns of the data. WS-Security is an implementation of the WS-Security specification enabling security at the message layer. The WS-Reliable Messaging channel enables the guarantee of message delivery. The encoders present a variety of encodings that can be used to suit the needs of the message. The HTTP channel specifies that the HyperText Transport Protocol is used for message delivery. The TCP channel similarly specifies the TCP protocol. The Transaction Flow channel governs transacted message patterns. The Named Pipe channel enables interprocess communication. The MSMQ channel enables interoperation with MSMQ applications.

Hosting and Activation=In its final form, a service is a program. Like other programs, a service must be run in an executable. This is known as a self-hostedservice.Services can also be hosted, or run in an executable managed by an external agent, such as IIS or Windows Activation Service (WAS). WAS enables WCF applications to be activated automatically when deployed on a computer running WAS. Services can also be manually run as executables (.exe files). A service can also be run automatically as a Windows service. COM+ components can also be hosted as WCF services

WCF stands for Windows Communication Foundation. The elementary feature of WCF is interoperability. It is one of the latest technologies of Microsoft that is used to build service-oriented applications. Based on the concept of message-based communication, in which an HTTP request is represented uniformly, WCF makes it possible to have a unified API irrespective of diverse transport mechanisms.

WCF was released for the first time in 2006 as a part of the .NET framework with Windows Vista, and then got updated several times. WCF 4.5 is the most recent version that is now widely used.

A WCF application consists of three components −

WCF service,

WCF service host, and

WCF service client.

WCF platform is also known as the Service Model.

Fundamental Concepts of WCF

Message

This is a communication unit that comprises of several parts apart from the body. Message instances are sent as well as received for all types of communication between the client and the service.

Endpoint

It defines the address where a message is to be sent or received. It also specifies the communication mechanism to describe how the messages will be sent along with defining the set of messages. A structure of an endpoint comprises of the following parts −

Address

Address specifies the exact location to receive the messages and is specified as a Uniform Resource Identifier (URI). It is expressed as scheme://domain[:port]/[path]. Take a look at the address mentioned below −

net.tcp://localhost:9000/ServiceA

Here, 'net.tcp' is the scheme for the TCP protocol. The domain is 'localhost' which can be the name of a machine or a web domain, and the path is 'ServiceA'.

Binding

It defines the way an endpoint communicates. It comprises of some binding elements that make the infrastructure for communication. For example, a binding states the protocols used for transport like TCP, HTTP, etc., the format of message encoding, and the protocols related to security as well as reliability.

Contracts

It is a collection of operations that specifies what functionality the endpoint exposes to the client. It generally consists of an interface name.

Hosting

Hosting from the viewpoint of WCF refers to the WCF service hosting which can be done through many available options like self-hosting, IIS hosting, and WAS hosting.

Metadata

This is a significant concept of WCF, as it facilitates easy interaction between a client application and a WCF service. Normally, metadata for a WCF service is generated automatically when enabled, and this is done by inspection of service and its endpoints.

WCF Client

A client application that gets created for exposing the service operations in the form of methods is known as a WCF client. This can be hosted by any application, even the one that does service hosting.

Channel

Channel is a medium through which a client communicates with a service. Different types of channels get stacked and are known as Channel Stacks.

SOAP

Although termed as ‘Simple Object Access Protocol’, SOAP is not a transport protocol; instead it is an XML document comprising of a header and body section.

Advantages of WCF

It is interoperable with respect to other services. This is in sharp contrast to .NET Remoting in which both the client and the service must have .Net.

WCF services offer enhanced reliability as well as security in comparison to ASMX (Active Server Methods) web services.

Implementing the security model and binding change in WCF do not require a major change in coding. Just a few configuration changes is required to meet the constraints.

WCF has built-in logging mechanism whereas in other technologies, it is essential to do the requisite coding.

WCF has integrated AJAX and support for JSON (JavaScript object notation).

It offers scalability and support for up-coming web service standards.

It has a default security mechanism which is extremely robust.

.net framework:-

The .NET Client Profile is a subset of the .NET Framework, which was provided with .NET Framework 4 and earlier versions and was optimized for client applications. The .NET Framework is a development platform for Windows, Windows Phone and Microsoft Azure and provides a managed app execution environment and the .NET Framework class library. The .NET Framework 4 and earlier versions provided two deployment options: the full .NET Framework and the Client Profile. The Client Profile enabled faster deployment and smaller app installation packages than the full .NET Framework.

Starting with the .NET Framework 4.5, the Client Profile has been discontinued and only the full redistributable package is available. Optimizations provided by the .NET Framework 4.5, such as smaller download size and faster deployment, have eliminated the need for a separate deployment package. The single redistributable streamlines the installation process and simplifies your app's deployment options

The .NET Framework Client Profile (Client Profile) was created to answer the feedback from many customers that a smaller framework was needed specifically for Client Applications. The Client Profile is a subset of assemblies already contained within .NET Framework 3.5 Service Pack 1. The Client Profile subset is focused on fulfilling the needs of Client applications.

The Client Profile contains the following features:

· Smaller framework deployment - ~28 MB client deployment package

· Smaller, faster client deployment boot strapper

· Client Application focused feature set:

o Common Language Runtime (CLR)

o ClickOnce

o Windows Forms

o Windows Presentation Foundation

o Windows Communication Foundation

· Visual Studio 2008 SP1 Integration – Applications can be targeted specifically for the Client Framework subset.

The Client Profile is made up of a combination of features. The first part is client redistributable packages that are installed by end-users. The second part is the client deployment boot strapper that manages the deployment of redistributable packages.

The Web Service Life Cycle

The purpose of this article is two fold:

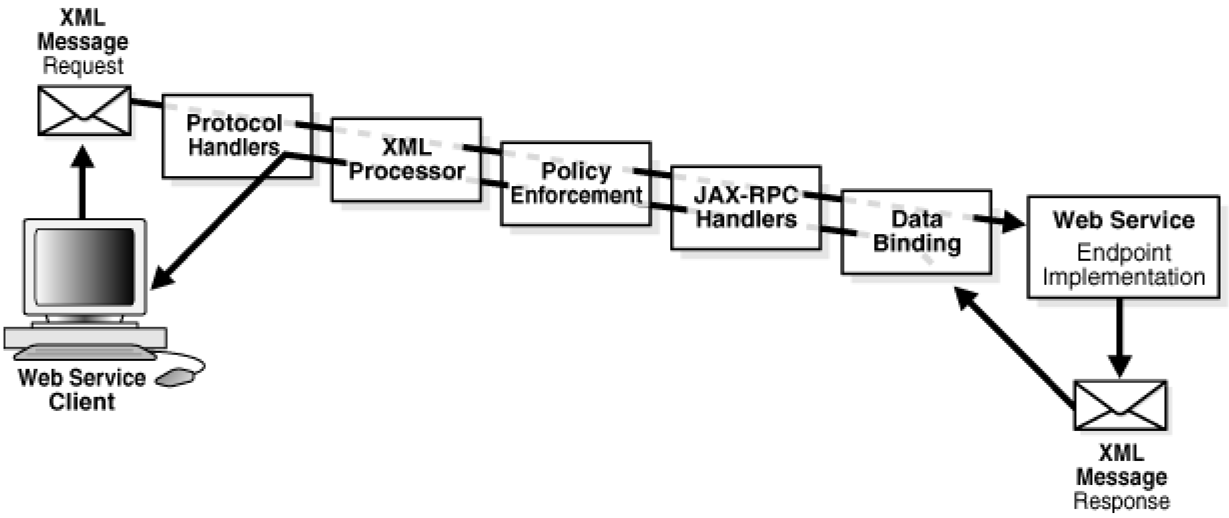
- To increase the understanding of the Web service life cycle and state management mechanisms
- To provide insights into how you can use this knowledge to your advantage when developing and deploying Web service applications.

Web Services Life Cycle

The life cycle of Web services may be simplified into four stages: design/build, test, deploy/ execute, and manage. Although one may think of Web services management as only happening at one stage of the life cycle — the management stage — Web services management is actually an important concept at each stage.

The initial stage is the design and build stage, where developers architect Web services using integrated development environment (IDE) tools — such as Sun™ ONE Studio software — to generate proxies and service end points. Web services management issues at this stage include design-time service aggregation and orchestration. If there are multiple versions for different markets or audience, each Web service deployed is given a unique service version in the SOAP message header or service registry (known as service versioning). In some cases, different remote business services are aggregated (service aggregation) and combined (service orchestration) at run time.

The test stage is next. Here, functional changes to the Web services go through integrated testing and regression testing. This is particularly important to ensure that all use cases are thoroughly tested before the Web services applications go into production. In the deploy and execute stage, management issues include reliable messaging, schema validation, and interoperability. It is in this stage that reliable transport mechanisms — such as SOAP over Java Message Service (JMS) technology, the ebXML Message Service (ebMS), and Web Service Reliability (WS-Reliability)— are used to provide reliable sending and receiving of SOAP messages. These messages may be encrypted, decrypted, compressed, or decompressed to ensure reliability and data integrity. To ensure data quality, a Web services proxy or agent can be used to validate the SOAP messages against the eXtensible Markup Language (XML) schema at run time in a process known as schema validation. If SOAP messages must be routed to other service end points, perhaps for security reasons, a Web services proxy or agent is used to process service end point forwarding. If the service requester is a .NET client using SOAP 1.1 and the service provider is a Java technology-based server using SOAP 1.2, the Web services proxy or agent can be customized to handle .NET to Java or SOAP 1.1/1.2 interoperability at run time.



The final stage is management,

which requires that predefined service-level agreements are met. Auditing, monitoring, and troubleshooting also occur at this stage, as well as service provisioning and service management. With the addition of a scalable and reliable resource provisioning infrastructure such as Sun’s N1™ technology, additional memory and CPU resources can be dynamically provisioned to the Web services infrastructure Universal Description, Discovery, and Integration (UDDI) service registry; ebXML registry/repository; or application servers running the service end points — without disruption. A Web services management server could also be used for more simple resource provisioning, such as administering different management policies that govern access rights for routing and accessing SOAP messages, as well as for collecting service information and analyzing exception.

What is service contract?

In WCF, all services expose contracts. The contract is a platform-neutral and standard way of describing what the service does. WCF defines four types of contracts.

1.Service contracts
Describe which operations the client can perform on the service.

2.Data contracts
Define which data types are passed to and from the service. WCF defines implicit contracts for built-in types such as int and string, but you can easily define explicit opt-in data contracts for custom types.

3.Fault contracts
Define which errors are raised by the service, and how the service handles and propagates errors to its clients.

4.Message contracts
Allow the service to interact directly with messages. Message contracts can be typed or untyped, and are useful in interoperability cases and when there is an existing message format you have to comply with.

The Service Contract
The ServiceContractAttribute is defined as:

```
[AttributeUsage(AttributeTargets.Interface|AttributeTargets.Class,
     Inherited = false)]
public sealed class ServiceContractAttribute : Attribute
{
  public string Name
  {get;set;}
  public string Namespace
  {get;set;}
  //More members
}
```
This attribute allows you to define a service contract.

The ServiceContract attribute maps a CLR interface (or inferred interface, as you will see later on) to a technology-neutral service contract. The ServiceContract attribute exposes a CLR interface (or a class) as a WCF contract, independently of that type's visibility. The type visibility has no bearing on WCF, because visibility is a CLR concept. Applying the ServiceContract attribute on an internal interface exposes that interface as a public service contract, ready to be consumed across the service boundary. Without the ServiceContract attribute, the interface is not visible to WCF clients, in line with the service-oriented tenet that service boundaries are explicit. To enforce that, all contracts must explicitly opt in: only interfaces (or classes) decorated with the ServiceContract attribute will be considered as WCF contracts. Other types will not.

Questio:- Explain contracts in web service.

Introduction

Contracts in WCF, provide interoperability they need to communicate with the client.

WCF defines four types of contracts

• Service Contract

• Data Contract

• Message Contract

• Fault Contract

Service contract describes the operations, or methods, that are available on the service endpoint, and exposed to the outside world. A Service contract describes the client-callable operations (functions) exposed by the service, apart from that it also describes.

• location of operations, interface and methods of your service to a platform-independent description
• message exchange patterns that the service can have with another party. might be one-way/requestreply/duplex.
A data contract is a formal agreement between a service and a client that abstractly
describes the data to be exchanged
It describes the external format of data ed to and from service operations It defines the
structure and types of data exchanged in service messages
It maps a CLR type to an XML Schema
It defines how data types are serialized and deserialized. Through serialization, you convert
an object into a sequence of bytes that can be transmitted over a network. Through
deserialization, you reassemble an object from a sequence of bytes that you receive from a
calling application.
It is a versioning system that allows you to manage changes to structured data .
A Message Contract is used to control the structure of a message body and serialization process. It is
also used to send / access information in SOAP headers. By default WCF takes care of creating SOAP
messages

according to service DataContracts and OperationContracts.
Fault Contract provides documented view for error accorded in the service to client. This help as to easy identity the what error has occurred, and where. By default when we throw any exception from
service, it will not reach the client side. The less the client knows about what happened on the server side, the more dissociated the interaction will be, this phenomenon (not allowing the

actual cause of
error to reach client). is known as error masking

Explain   Configure Web services

Basic configuration details of a Web service include the type of implementation (Java class packaged in a WAR file or a stateless EJB packaged in an EJB JAR file), the full name of the Web service, and so on.

A subset of Web service configuration properties are view only; you cannot edit them using the Administration Console.

Toconfigure a Web service:

In the left pane of the Administration Console, select Deployments.

In the right pane, navigate within the Deployments table until you find the Web service for which you want to view the configuration.

Web services are deployed as part of an Enterprise application, Web application, or EJB. To understand how Web services are displayed in the Administration Console, see View installed Web services.

In the Deployments table, click the name of the Web service.

Select Configuration > General to view and configure general information about the Web service.

This page lists the name and description of the Web service, its implementation type (JAX-WS or JAX-RPC) and source, and its WSDL configuration. You can modify the WSDL configuration only.

Select Configuration > Handlers to view and configure the SOAP handlers that are defined for the Web service.

This page lists the handlers configured for the Web service, the class that implements the handler, the Web service endpoint with which it is associated, and whether the handler processes any SOAP headers. Click the name of the handler to view additional information. For more information, see View the SOAP message handlers of a Web service.

Select Configuration > WS-Policy for information about the WS-Policy files attached to the Web service.

This page lists the policies currently attached to the Web service endpoint and operations. Expand the Web service endpoint to view its operations. Click the name of the Web service endpoint or operation to attach policy files to the inbound or outbound request. For more information, see Attach a WS-Policy file to a Web service.

Select Configuration > Ports for information about the Web service endpoints and operations.

Click the name of the Web service endpoint to customize its configuration. You can modify the following configuration:

For JAX-RPC Web services, you can configure reliable messaging only.

Reliable messaging. For more information, see Configure Web service reliable messaging.

Message buffering. For more information, see Configure message buffering for Web services.

Atomic transactions. For more information, see Configure Web service atomic transactions.

Persistence. For more information, see Configure Web service persistence.

Click the name of the Web service operation to customize its configuration. You can modify the configuration for atomic transactions only at the operation level

Binding and Endpoint.

1.)What a Binding Defines :- The information in a binding can be very basic, or very complex. The most basic binding specifies only the transport protocol (such as HTTP) that must be used to connect to the endpoint. More generally, the information a binding contains about how to connect to an endpoint falls into one of the following categories.

- Protocols:-Determines the security mechanism being used: either reliable messaging capability or transaction context flow settings.
- Encoding:-Determines the message encoding (for example, text or binary).
- Transport:-Determines the underlying transport protocol to use (for example, TCP or HTTP).

The Elements of a Binding :- A binding basically consists of an ordered stack of binding elements, each of which specifies part of the communication information required to connect to a service endpoint. The two lowest layers in the stack are both required. At the base of the stack is the transport binding element and just above this is the element that contains the message encoding specifications.

2.)Endpoint Creation :- All communication with a Windows Communication Foundation (WCF) service occurs through the *endpoints* of the service. Endpoints provide the clients access to the functionality that a WCF service offers. This section describes the structure of an endpoint and outlines how to define an endpoint in configuration and in code.

The Structure of an Endpoint :- Each endpoint contains an address that indicates where to find the endpoint, a binding that specifies how a client can communicate with the endpoint, and a contract that identifies the methods available.

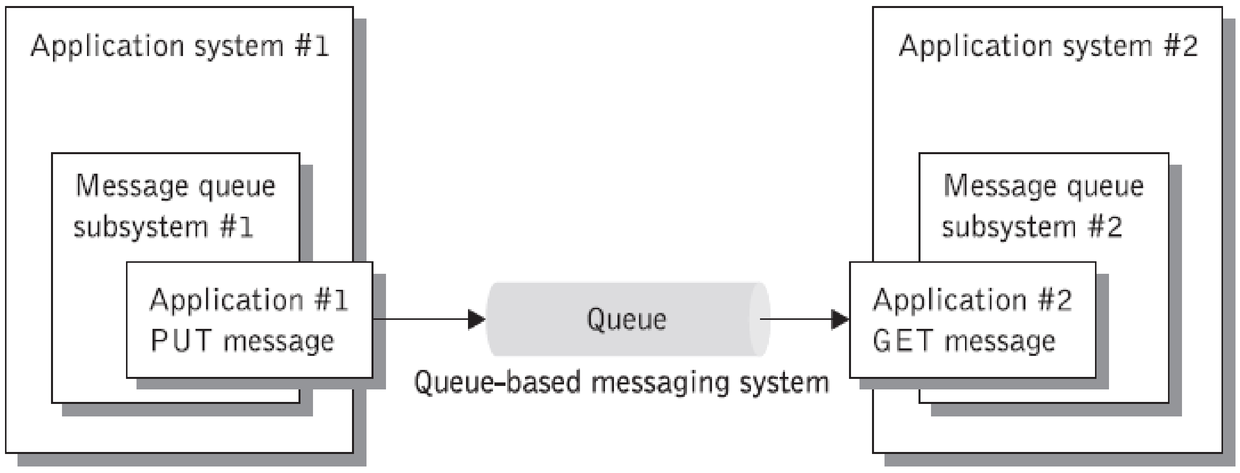
- **Address**. The address uniquely identifies the endpoint and tells potential consumers where the service is located. It is represented in the WCF object model by the EndpointAddress address, which contains a Uniform Resource Identifier (URI) and address properties that include an identity, some Web Services Description Language (WSDL) elements, and a collection of optional headers.

- **Binding**. The binding specifies how to communicate with the endpoint. The binding specifies how the endpoint communicates with the world, including which transport protocol to use (for example, TCP or HTTP), which encoding to use for the messages (for example, text or binary), and which security requirements are necessary (for example, Secure Sockets Layer [SSL] or SOAP message security).
- **Service contract**. The service contract outlines what functionality the endpoint exposes to the client. A contract specifies the operations that a client can call, the form of the message and the type of input parameters or data required to call the operation, and the kind of processing or response message the client can expect. Three basic types of contracts correspond to basic message exchange patterns (MEPs): datagram (one-way), request/reply, and duplex (bidirectional).

Q) Explain store and forward messaging.

Store and forward messaging

With the store and forward queuing mechanism, messages are placed on a virtual channel called a *message queue* by a sending application and are retrieved by the receiving application as needed. Messages are exchanged through a queue, which is the destination to which senders send messages and a source from which receivers receive messages. The queue is a container that can keep hold of a message until the recipient collects it. The message queue is independent of both the sender and receiver applications and acts as a buffer between the communicating applications. In this form of communication, two applications can be senders and receivers relative to the message queue


Application system #1
Message queue subsystem #1
Application #1 PUT message
Queue
Queue-based messaging system
Application system #2
Message queue subsystem #2
Application #2 GET message

2.10 Store and forward messaging

communication with the store and forward queuing mechanism allows work to be performed whenever applications are ready.

The message delivery semantics include several delivery options which range from *exactly-once delivery* to *at-least-once delivery* and to *at-most-once delivery*. It is critical for many applications to ensure guaranteed delivery of a message to its final destination and elimination of duplicates. This level of message service delivery is referred to as exactly-once message delivery. The at-least-once message delivery mode guarantees that messages will be delivered to their final destination at least once. The at-most-once delivery mode guarantees that messages will be delivered to their final destination at most once.

This latter mode of delivery is a less stringent QoS setting on a message as it implies that the messaging system is permitted to occasionally lose a message in the event of hardware, software, or network breakdown. The exactly-once guarantee of message delivery is a characteristic of message reliability, which we shall examine as part of the Web services standardsThe store and forward queuing mechanism is typical of a many-to-one messaging paradigm where multiple applications can send messages to a single application. The same application can be sender, receiver, or both sender and receiver. Message queuing provides a highly reliable, although not always timely, means of ensuring that application operations are completed.
In many applications there is an additional requirement that the concept of store and forward is capable of being repeated across multiple message servers that are chained together

Q) Explain Publish Subscribe Middleware

Ans) Another form of reliable messaging is publish/subscribe messaging. This mode of messaging is a slightly more scalable form of messaging when compared to the store and forward mechanism. With this type of asynchronous communication the application that produces information publishes it and all other applications that need this type of information subscribe to it. Messages containing the new information are placed in a queue for each subscriber by the publishing application. Each application in this scheme may have a dual role: it may act as a publisher or subscriber of different types of information.

The publish/subscribe messaging works as follows. Suppose that a publisher application publishes messages on a specific topic, such as sending out new product prices or new product descriptions to retailers. Multiple subscribing applications can subscribe to this topic and receive the messages published by the publishing application. Figure 2.12 shows message publishers publishing messages by sending them to topics and all the message subscribers who had registered to the topic for messages receiving them as soon as the publisher makes them available. This figure describes how the publish/subscribe semantics work:
1. Publishers publish messages to specific topics.
2. A message server keeps track of all the messages, and all its currently active and durable subscribers (subscribers who specifically expressed a durable interest in the topic). The message server provides a secure environment for the messaging system by handling authorization and authentication.
3. As soon as messages are published on a specific topic, they are distributed to all of its subscribers. Durable subscribers, who were not connected at the time of message delivery, can retrieve the messages if they come up within a specified time.
The message server takes the responsibility of delivering the published messages to the subscribing applications based on the subscribed topic. Every message has an expiration time that specifies the maximum amount of time that it can live from the time of its publication in a topic. The message server first delivers messages to its associated active subscribers, and then checks to make sure if there are any non-active durable subscribers
Asynchronous forms of middleware 69
Figure 2.11 Store and forward involving multiple chained message servers

subscribed to the published topic. If, after the initial delivery, any of the durable subscribers did not acknowledge receipt of the message, the message is retained in the message server for the period of the expiration time, in the hope that the durable subscribers, if any, will connect to the message server and accept delivery of the message. All subscribers have a message event listener that takes delivery of the message from the topic and delivers it to the messaging client application for further processing. Subscribers can also filter the messages that they receive by qualifying their subscriptions with a message selector. Message selectors evaluate a message's headers and properties (not their bodies) with the provided filter expression strings. The subscription list can be easily modified, on-the-fly, providing a highly flexible communications system that can run on different systems and networks. The publish/ subscribe messaging mode usually includes the ability to transform messages, acting as an interpreter, which enables applications that were not designed to work together to do so.

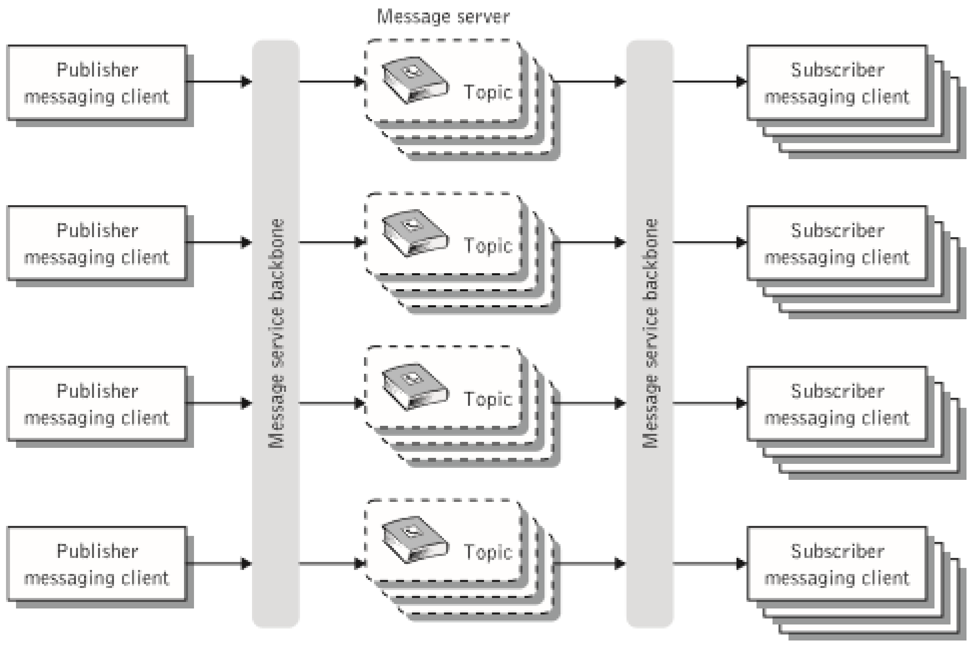

Message server
Publisher messaging client
Publisher messaging client
Publisher messaging client
Publisher messaging client
Message service backbone
Topic
Topic
Topic
Topic
Message service backbone
Subscriber messaging client
Subscriber messaging client
Subscriber messaging client
Subscriber messaging client

Figure 2.12 Publish/subscribe messaging

EXPLAIN POINT TO POINT QUEING MIDDLEWARE ?

ANSWER- Point-to-point integration (also known as *one-to-one integration*) is the simpler of the two integration models. Point-to-point integration is used when a sender has to send a message to a single receiver (that is, a 1:1 relationship). As an example, an organization may need to update a human resources database with information from an ERP system. In this model, the destination is a queue provided by the integration broker, in which the sender can place messages (a given message is placed in an individual queue).

The sequence in the point-to-point model is as follows:

- The sender places a message in the queue.

- The integration broker forwards the message to the appropriate receiver. (Before forwarding the message, the broker can also do additional processing such as transforming the message.)
- The receiver receives the message and processes it as appropriate.

While adequate for simple integration, this model is quickly unmanageable for larger integration requirements because of the *n(n-1) connections rule*

The point-to-point messaging model provides a straightforward asynchronous exchange of messages between software entities. In this model, shown in Figure 1.6, messages from producing clients are routed to consuming clients via a queue. As discussed earlier, the most common queue used is a FIFO queue, in which messages are sorted in the order

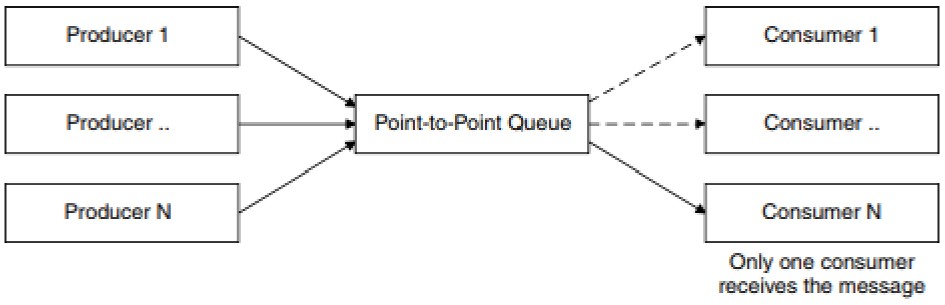


Figure 1.6 Point-to-point messaging model

in which they were received by the message system and as they are consumed, they are removed from the head of the queue. While there is no restriction on the number of clients who can publish to a queue, there is usually only a single consuming client, although this is not a strict requirement. Each message is delivered only once to only one receiver. The model allows multiple receivers to connect to the queue, but only one of the receivers will consume the message. The techniques of using multiple consuming clients to read from a queue can be used to easily introduce smooth, efficient load balancing into a system. In the point-to-point model, messages are always delivered and will be stored in the queue until a consumer is ready to retrieve them. Request-Reply Messaging Model This model is designed around the concept of a request with a related response. This model is used for the World Wide Web (WWW), a client requests a page from a server, and the server replies with the requested web page. The model requires that any producer who sends a message must be ready to receive a reply from consumers at some stage in the future. The model is easily implemented with the use of the point-to-point and publish/subscribe model and may be use

Ques. Transaction Oriented Middleware (TOM) (or Distributed Tuples)

A distributed relational database offers the abstraction of distributed tuples (i.e. particular instances of an entity), and is the most widely deployed kind of middleware today. It uses

Structured Query Language (SQL) which allows programmers to manipulate sets of these tuples in an English-like language yet with intuitive semantics and rigorous mathematical foundations based on set theory and predicate calculus. Distributed relational databases also offer the abstraction of a transaction (which can also be performed using Transactional SQL or TSQL). Distributed relational database products typically offer heterogeneity across programming languages, but most do not offer much, if any, heterogeneity across vendor implementations. Transaction Processing Monitors (TPMs) are commonly used for end-to-end resource management of client queries, especially server-side process management and managing multi-database transactions. As an example consider the JINI framework (built on top of JavaSpaces) which is tailored for intelligent networked devices, especially in homes.

Advantages

Users can access virtually any database for which they have proper access rights from anywhere in the world (as opposed to their deployment in closed environments where users access the system only via a restricted network or intranet)

They address the problem of varying levels of interoperability among different database structures.

They facilitate transparent access to legacy database management systems (DBMSs) or applications via a web server without regard to database-specific characteristics.

Disadvantages

This is the oldest form of middleware – hence it lacks many features of much recent forms of middleware.

Does not perform failure transparency

Tight coupling between client and server

Q) Explain various hosting services.
A web hosting service is a type of Internet hosting service that allows individuals and organizations to make their website accessible via the World Wide Web. Web hosts are companies that provide space on a server owned or leased for use by clients, as well as providing Internet connectivity, typically in a data center. Web hosts can also provide data center space and connectivity to the Internet for other servers located in their data center

Various types of hostimg services

1.Shared web hosting services
One's website is placed on the same server as many other sites, ranging from a few sites to hundreds of websites. Typically, all domains may share a common pool of server resources, such as RAM and the CPU. The features available with this type of service can be quite basic and not flexible in terms of software and updates. Resellers often sell shared web hosting and web companies often have reseller accounts to provide hosting for clients.

2.Reseller web hosting

Allows clients to become web hosts themselves. Resellers could function, for individual domains, under any combination of these listed types of hosting, depending on who they are affiliated with as a reseller. Resellers' accounts may vary tremendously in size: they may have their own virtual dedicated server to a colocated server. Many resellers provide a nearly identical service to their provider's shared hosting plan and provide the technical support themselves.

3.Dedicated hosting services
The user gets his or her own Web server and gains full control over it (user has root access for Linux/administrator access for Windows); however, the user typically does not own the server. One type of dedicated hosting is self-managed or unmanaged. This is usually the least expensive for dedicated plans. The user has full administrative access to the server, which means the client is responsible for the security and maintenance of his own dedicated server.

4.Managed hosting services
The user gets his or her own Web server but is not allowed full control over it (user is denied root access for Linux/administrator access for Windows); however, they are allowed to manage their data via FTP or other remote management tools. The user is disallowed full control so that the provider can guarantee quality of service by not allowing the user to modify the server or potentially create configuration problems. The user typically does not own the server. The server is leased to the client.

• Explain any One Message Oriented Middleware?

Ans:

Message-oriented middleware (MOM) is an infrastructure that involves the passing of data between applications using a common communication channel that carries self-contained messages.

In a MOM-based communication environment, messages are usually sent and received asynchronously.

MOM is interposed between the client and the server part of client–server architecture and handles asynchronous calls between clients and servers.

The messaging system is responsible for managing the connection points between messaging clients, and for managing multiple channels of communication between the connection points. The messaging system is usually implemented by a software module known as the message (or integration) broker.

Modern MOM technologies typically possess features that can be used as a springboard for developing Web services technologies, including:

- ◆Message multicast supporting event-driven processing, i.e., the publish/subscribe model.
- ◆Reliability and serialization of messages thus guaranteeing message delivery in the

appropriate order.

◆Subject-based (textual) names and attributes to abstract from the physical names and addresses that are interpreted and implemented by the network.

◆Support for multiple communications protocols such as store and forward, request/ reply, and publish/subscribe.

◆Support for transactional boundaries.

Unit 1

Question :- Define the term web service. Explain two types of web service

Web services are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP). As described by the World Wide Web Consortium (W3C), web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions, thanks to the use of XML. Web services can be combined in a loosely coupled way to achieve complex operations. Programs providing simple services can interact with each other to deliver sophisticated added-value services.

Types of Web Services

On a technical level, web services can be implemented in various ways. The two types of web services discussed in this section can be distinguished as "big" web services and "RESTful" web services.

Big Web Services

In Java EE 6, JAX-WS provides the functionality for "big" web services. Big web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats. Such systems often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

The SOAP message format and the WSDL interface definition language have gained widespread adoption. Many development tools, such as NetBeans IDE, can reduce the complexity of developing web service applications.

A SOAP-based design must include the following elements.

- A formal contract must be established to describe the interface that the web service offers. WSDL can be used to describe the details of the contract, which may include messages, operations, bindings, and the location of the web service. You may also process SOAP messages in a JAX-WS service without publishing a WSDL.
- The architecture must address complex nonfunctional requirements. Many web service specifications address such requirements and establish a common vocabulary for them. Examples include transactions, security, addressing, trust, coordination, and so on.
- The architecture needs to handle asynchronous processing and invocation. In such cases, the infrastructure provided by standards, such as Web Services Reliable Messaging (WSRM), and APIs, such as JAX-WS, with their client-side asynchronous invocation support, can be leveraged out of the box.

RESTful Web Services

In Java EE 6, JAX-RS provides the functionality for Representational State Transfer (RESTful) web services. REST is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service–API definitions.

Because RESTful web services use existing well-known W3C and Internet Engineering Task Force (IETF) standards (HTTP, XML, URI, MIME) and have a lightweight infrastructure that allows services to be built with minimal tooling, developing RESTful web services is inexpensive and thus has a very low barrier for adoption. You can use a development tool such as NetBeans IDE to further reduce the complexity of developing RESTful web services.

A RESTful design may be appropriate when the following conditions are met.

- The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.
- A caching infrastructure can be leveraged for performance. If the data that the web service returns is not dynamically generated and can be cached, the caching infrastructure that web servers and other intermediaries inherently provide can be leveraged to improve performance. However, the developer must take care because such caches are limited to the HTTP GET method for most servers.
- The service producer and service consumer have a mutual understanding of the context and content being passed along. Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it meaningfully. In the real world, most commercial applications that expose services as RESTful implementations also distribute so-called value-added toolkits that describe the interfaces to developers in popular programming languages.
- Bandwidth is particularly important and needs to be limited. REST is particularly useful for limited-profile devices, such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.
- Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style. Developers can use such technologies as JAX-RS and Asynchronous JavaScript with XML (AJAX) and such toolkits as Direct Web Remoting (DWR) to consume the services in their web applications. Rather than starting from scratch, services can be exposed with XML and consumed by HTML pages without significantly

refactoring the existing web site architecture. Existing developers will be more productive because they are adding to something they are already familiar with rather than having to start from scratch with new technology.

Unit 1

Q. What are the two type of informational Services?Explain

Informational services are services of relatively simple nature. They either provide access to content interacting with an end user by means of simple request/response sequences, or alternatively may expose back-end business applications to other applications. Web services that typically expose the business functionality of the applications and components that underlie them are known as *programmatic services*. For instance, they may expose function calls, typically written in programming languages such as Java/EJB, Visual Basic, or C++. The exposed programmatic simple services perform a request/response type of business task and can be viewed as "atomic" (or singular) operations. Applications access these function calls by executing a Web service through a standard programmatic interface specified in the Web Services Description Language or WSDL (see Chapter 5).
Informational services can be subdivided into three subcategories according to the business problems they solve:
1. Pure *content services*, which give programmatic access to content such as weather report information, simple financial information, stock quote information, design information, news items, and so on.
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. Such service requests may have complicated realizations. Consider, for example, "pure" business services, such as logistic services, where automated services are the actual front-ends to fairly complex physical organizational information systems.
3. *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. Generally speaking, these services are offered by a third party and run the whole range from commerce-enabling services, such as logistics, payment, fulfillment, and tracking services, to other value-added commerce services, such as rating services. Typical examples of syndicated services might include reservation services on a travel site or rate quote services on an insurance site.
Informational services are singular in nature in that they perform a complete unit of work that leaves its underlying datastores in a consistent state. However, they are not transactional in nature (although their back-end realizations may be). An informational service does not keep any memory of what happens to it between requests. In that respect this type of service is known as a *stateless Web service*.

Unit 1

1)Explain in detail about Synchronicity?
Answer:-We may distinguish between two programming styles for services:

a)Synchronous or remote procedure call(RPC) style.
b)Asynchronous or message style.

a) Synchronous Services:

Clients of synchronous services express their request as a method call with a set of arguments, which returns a response containing a return value. This implies that when a client sends a request message, it expects a response message before continuing with its computation.This makes the whole invocation an all or nothing proposition. If one operation is unable to complete for any reason, all other dependent operation will fail. Because of this type of bilateral communication between the client and service provider. RPC style Web service are normally used when an application exhibits the following characteristics:-

a)The client invoking the service requires an immediate response.
b)The client and service work in a back and forth conversational way.

Examples of typical simple synchronous services with an RPC-style include reurning the current price for a given stock;providing the current weather conditions ina particular location; or checking the credit rating of a potential trading partner prior to the completion of a business transaction.

b) Asynchronous services:-

Asynchronous service are document style or message driven services. When a client invokes a message style services , the client typically sends it an entire document, such as a purchase order, rather than a discrete set of parameters. The service accepts the entire document, it processes it and may or may not return a result message. A client that involves an asynchronous service does not need to wait for a response before it continues with the remainder of its application. The response from the service, if any, can appear hours or even days later.

Asynchronous interactions(messaging) are akey design pattern in loosely coupled environments. Messaging enables a loosely coupled environments in which an application does not need to know the intimate details of how to reach and interface with other applications. This allows a communication operation between any two process to be a self contained, standalone unit of work. Document style web services are normally used when an application exhibits the following characteristics:

a)The client does not require an immediate response.
b)The service is document oriented (the client typically sends an entire document, e.g., a purchase order, rather discrete parameters).

Examples of document style Web services include processing a purchase order;
Responding to a request for quote order from a customer ; or responding to an order
Placement by a particular customer. In all these cases, the client sends an entire document, such as purchase order, to the web service and assumes that the Web service is processing it in some way, but the client does not require an immediate answer.

Unit 1

Question: What is SOAP ? Explain in detail.

Answer:

1. SOAP is known as the Simple Object Access Protocol

2. SOAP is an XML-based protocol for accessing web services over HTTP

3.Soap is a protocol which defines the rules of message passing between webservices or client applications

4.Soap work as a intermediate language between different webservices developing languages

5.SOAP gives standard specifications so as heteroginious programming languages can communicate with each other

6.SOAP was designed to work with XML over HTTP

7.Web services relay on SOAP for exchanging messages between computers regardless of their operating systems,programming environment,or object model framework

8.SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a means for transport

9.SOAP method is simply an HTTP request and response that complies with the SOAP encoding rules

10.SOAP can be defined as a lightweight wire protocol for exchanging structured and type information back and forth between disparate systems in a distributed environment such as internet or LAN

11.Lightweight protocol means that SOAP posses two fundamentals properties .It can send and receive HTTP transport protocol packets,and process XML messages.

ADVANTAGES:

a. Simplicity

b. Portability

c. Firewall

d. Open standards

e. Interoperability

f. Universal acceptance

g. Resilience to changes

DISADVANTAGES:

a.Statelessness

b.Serialization by value and not by reference

c. Slow

d.WSDL dependence