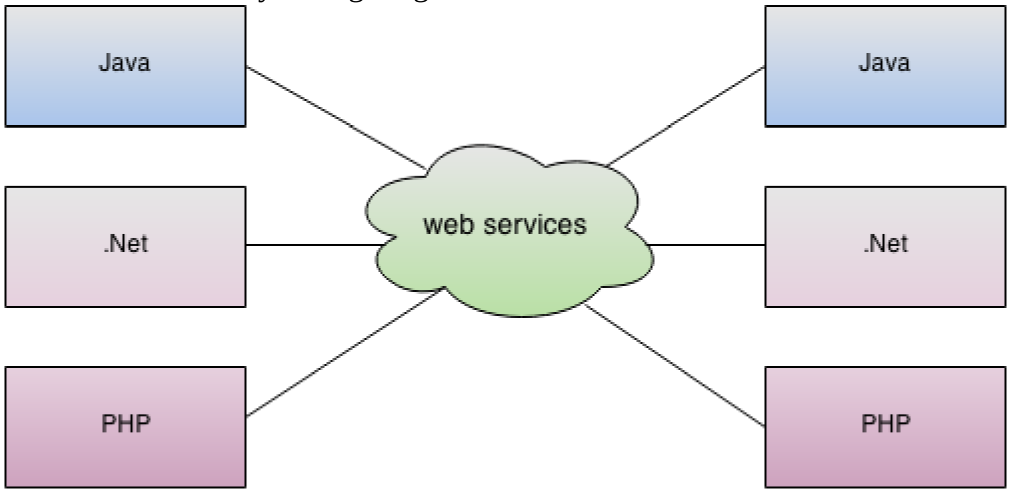
Web Services: Definition, Web services Architecture, Simple Object Access Protocol (SOAP) - goals, structure and contents of a SOAP Message, processing a SOAP message, Web Services Description language (WSDL) - Structure of WSDL interface, Implications of WSDL Model, Universal description discovery and integration (UDDI) - Goals, Information in a UDDI registry, UDDI data structures, UDDI Registry API.

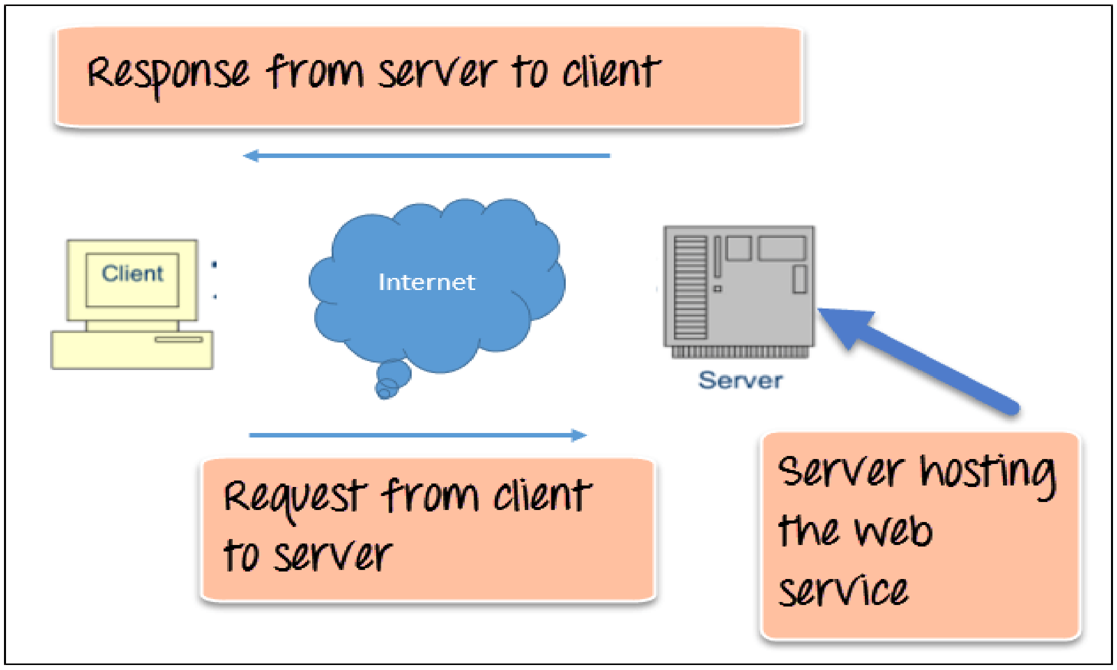
A **Web Service** is can be defined by following ways:

- is a client server application or application component for communication.
- method of communication between two devices over network.
- is a software system for interoperable machine to machine communication.
- is a collection of standards or protocols for exchanging information between two devices or application.

Let's understand it by the figure given below:



As you can see in the figure, java, .net or PHP applications can communicate with other applications through web service over the network. For example, java application can interact with Java, .Net and PHP applications. So web service is a language independent way of communication.


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

The above diagram shows a very simplistic view of how a web service would actually work. The client would invoke a series of web service calls via requests to a server which would host the actual web service.

These requests are made through what is known as remote procedure calls. Remote Procedure Calls(RPC) are calls made to methods which are hosted by the relevant web service. As an example, Aamzon provides a web service that provides prices for products sold online via amazon.com. The front end or presentation layer can be in .Net or Java but either programming language would have the ability to communicate with the web service.

The main component of a web service is the data which is transferred between the client and the server, and that is XML. XML (Extensible markup language) is a counterpart to HTML and easy to understand the intermediate language that is understood by many programming languages.

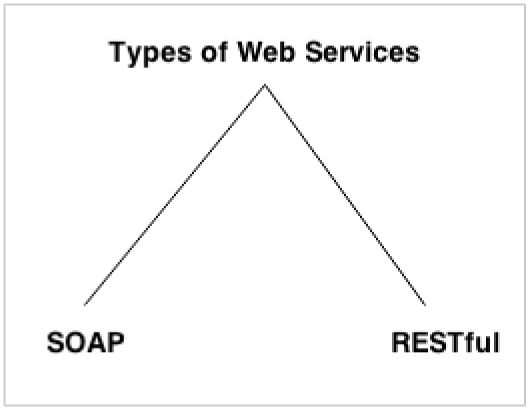
So when applications talk to each other, they actually talk in XML. This provides a common platform for application developed on various programming languages to talk to each other.

Web services use something known as SOAP (Simple Object Access Protocol) for sending the XML data between applications. The data is sent over normal HTTP. The data which is sent from the web service to the application is called a SOAP message. The SOAP message is nothing but an XML document. Since the document is written in XML, the client application calling the web service can be written in any programming language.

Types of Web Services

There are mainly two types of web services.

1. SOAP web services.
2. RESTful web services.



Web Service Components

There are three major web service components.

1. SOAP
2. WSDL
3. UDDI

SOAP

SOAP is an acronym for Simple Object Access Protocol.
SOAP is a XML-based protocol for accessing web services.
SOAP is a W3C recommendation for communication between applications.
SOAP is XML based, so it is platform independent and language independent. In other words, it can be used with Java, .Net or PHP language on any platform.

WSDL

WSDL is an acronym for Web Services Description Language.
WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.

WSDL is a part of UDDI. It acts as a interface between web service applications.
WSDL is pronounced as wiz-dull.

UDDI

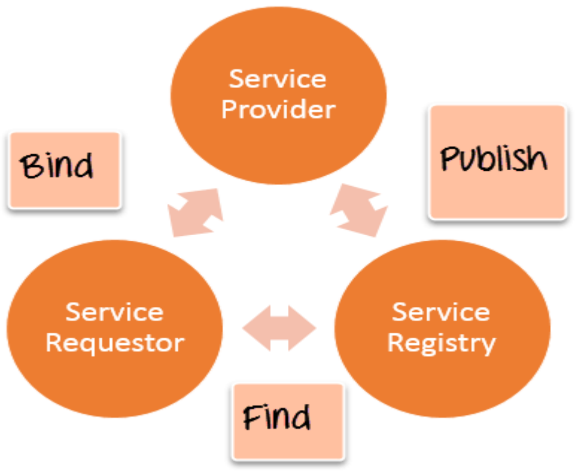
UDDI is an acronym for Universal Description, Discovery and Integration.
UDDI is a XML based framework for describing, discovering and integrating web services.
UDDI is a directory of web service interfaces described by WSDL, containing information about web services.

Web service Architecture

Every framework needs some sort of architecture to make sure the entire framework works as desired. Similarly, in web services, there is an architecture which consists of three distinct roles as given below

1. **Provider** - The provider creates the web service and makes it available to client application who wants to use it. The service provider implements the service and makes it available on the Internet.
2. **Requestor** - A requestor is nothing but the client application that needs to contact a web service. The client application can be a .Net, Java, or any other language based application which looks for some sort of functionality via a web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.
3. **Registry (Broker)** - The broker is nothing but the application which provides access to the UDDI. The UDDI, enables the client application to locate the web service. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

The diagram below showcases how the Service provider, the Service requestor and Service registry interact with each other.



1. **Publish(WSDL)** - A provider informs the broker (service registry) about the existence of the web service by using the broker's publish interface to make the service accessible to clients
2. **Find(UDDI)** - The requestor consults the broker to locate a published web service
3. **Bind(SOAP)** - With the information it gained from the broker(service registry) about the web service, the requestor is able to bind, or invoke, the web service.

Web service Characteristics

Web services have the following special behavioral characteristics:

1. **They are XML-Based** - Web Services uses XML to represent the data at the representation and data transportation layers. Using XML eliminates any networking, operating system, or platform sort of dependency since XML is the common language understood by all.
2. **Loosely Coupled** – Loosely coupled means that the client and the web service are not bound to each other, which means that even if the web service changes over time, it should not change the way the client calls the web service. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.
3. **Synchronous or Asynchronous functionality**- Synchronicity refers to the binding of the client to the execution of the service. In synchronous operations, the client will actually wait for the web service to complete an operation. An example of this is probably a scenario wherein a database read and write operation are being performed. If data is read from one database and subsequently written to another, then the operations have to be done in a sequential manner. Asynchronous operations allow a client to invoke a service and then execute other functions in parallel. This is one of the common and probably the most preferred techniques for ensuring that other services are not stopped when a particular operation is being carried out.
4. **Ability to support Remote Procedure Calls (RPCs)** - Web services enable clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote procedures expose input and output parameters that a web service must support.
5. **Supports Document Exchange** - One of the key benefits of XML is its generic way of representing not only data but also complex documents. These documents can be as simple as representing a current address, or they can be as complex as representing an entire book.
6. **Coarse-Grained-**Object-oriented technologies such as Java expose their services through individual methods. An individual method is too fine an operation to provide any useful capability at a corporate level. Building a Java program from scratch requires the creation of several fine-grained methods that are then composed into a coarse-grained service that is consumed by either a client or another service.

   Businesses and the interfaces that they expose should be coarse-grained. Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.

SOAP Web Services

SOAP is an open-standard, XML-based messaging protocol for exchanging information among computers. This is a brief tutorial that introduces the readers to the fundamentals of SOAP before moving on to explain its various elements, encoding, and how SOAP is transported.

Syntax Rules

Here are some important syntax rules:

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

The nature of SOAP –

Nature of SOAP:

- SOAP is a communication protocol designed to communicate via Internet.
- SOAP can extend HTTP for XML messaging.
- SOAP provides data transport for Web services.
- SOAP can exchange complete documents or call a remote procedure.
- SOAP can be used for broadcasting a message.
- SOAP is platform- and language-independent.
- SOAP is the XML way of defining what information is sent and how.
- SOAP enables client applications to easily connect to remote services and invoke remote methods.

Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP.

Other frameworks including CORBA, DCOM, and Java RMI provide similar functionality to SOAP, but SOAP messages are written entirely in XML and are therefore uniquely platform- and language-independent.

Advantages of Soap Web Services

1. WS Security: SOAP defines its own security known as WS Security.
2. Language and Platform independent: SOAP web services can be written in any programming language and executed in any platform.

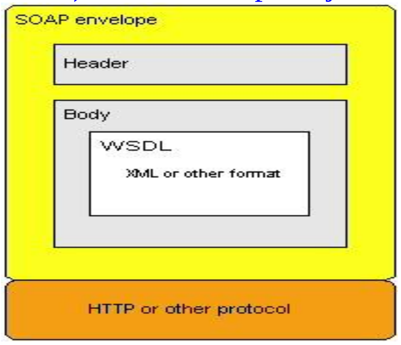
Disadvantages of Soap Web Services

1. Slow: SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.
2. WSDL dependent: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

SOAP Goals:

- Two major design goals: 1) Simplicity 2) Extensibility
- SOAP attempts to meet these goals by omitting features often found in messaging system and distributed object systems such as:
  1) Distributed garbage collection;
  2) Boxcarring or batching of messages;
  3) Activation(which requires objects-by-reference)
  4) Object-by-reference(which requires distributed garbage collection);

SOAP provides the envelope for sending Web Services messages over the Internet/Internet. It is part of the set of standards specified by the W3C. SOAP is an alternative to Representational State Transfer (REST) and JavaScript Object Notation (JSON).



The SOAP envelope contains two parts:

1. An optional header providing information on authentication, encoding of data, or how a recipient of a SOAP message should process the message.
2. The body that contains the message. These messages can be defined using the WSDL specification.

SOAP commonly uses HTTP, but other protocols such as Simple Mail Transfer Protocol (SMTP) may by used. SOAP can be used to exchange complete documents or to call a remote procedure.

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

- **Envelope −** Defines the start and the end of the message. It is a mandatory element.
- **Header −** Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point. It is an optional element.
- **Body −** Contains the XML data comprising the message being sent. It is a mandatory element.
- **Fault −** An optional Fault element that provides information about errors that occur while processing the message.

All these elements are declared in the default namespace for the SOAP envelope

SOAP Message Structure

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

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

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

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

</soap:Envelope>
```

Details of Structure:

1) The SOAP envelope indicates the start and the end of the message so that the receiver knows when an entire message has been received. The SOAP envelope solves the problem of knowing when you are done receiving a message and are ready to process it. The SOAP envelope is therefore basically a packaging mechanism.

The xmlns:soap Namespace

Notice the xmlns:soap namespace in the example above. It should always have the value of: "http://www.w3.org/2003/05/soap-envelope/".

The namespace defines the Envelope as a SOAP Envelope.

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

The encodingStyle Attribute

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements.

A SOAP message has no default encoding.

Syntax

soap:encodingStyle="*URI*"

Important points about SOAP envelope element −

- Every SOAP message has a root Envelope element.
- Envelope is a mandatory part of SOAP message.
- Every Envelope element must contain exactly one Body element.
- If an Envelope contains a Header element, it must contain no more than one, and it must appear as the first child of the Envelope, before the Body.
- The envelope changes when SOAP versions change.
- The SOAP envelope is specified using the *ENV* namespace prefix and the Envelope element.
- The optional SOAP encoding is also specified using a namespace name and the optional *encodingStyle* element, which could also point to an encoding style other than the SOAP one.
- A v1.1-compliant SOAP processor generates a fault upon receiving a message containing the v1.2 envelope namespace.
- A v1.2-compliant SOAP processor generates a *VersionMismatch* fault if it receives a message that does not include the v1.2 envelope namespace.

SOAP with HTTP POST

The following example illustrates the use of a SOAP message within an HTTP POST operation, which sends the message to the server. It shows the namespaces for the envelope schema definition and for the schema definition of the encoding rules. The *OrderEntry* reference in the HTTP header is the name of the program to be invoked at the tutorialspoint.com website.

```
POST /OrderEntry HTTP/1.1
Host: www.tutorialspoint.com
Content-Type: application/soap;  charset="utf-8"
Content-Length: nnnn
<?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">
   ...
   Message information goes here
   ...
</SOAP-ENV:Envelope>
```

2) The optional Header element offers a flexible framework for specifying additional application-level requirements. For example, the Header element can be used to specify a digital signature for password-protected services. Likewise, it can be used to specify an account number for pay-per-use SOAP services.

Important points about SOAP header element to take note of −

- It is an optional part of a SOAP message.
- Header elements can occur multiple times.
- Headers are intended to add new features and functionality.
- The SOAP header contains header entries defined in a namespace.
- The header is encoded as the first immediate child element of the SOAP envelope.
- When multiple headers are defined, all immediate child elements of the SOAP header are interpreted as SOAP header blocks.

SOAP Header Attributes

A SOAP Header can have the following two attributes −

Actor attribute

The SOAP protocol defines a message path as a list of SOAP service nodes. Each of these intermediate nodes can perform some processing and then forward the message to the next node in the chain. By setting the Actor attribute, the client can specify the recipient of the SOAP header.

The SOAP actor attribute is used to address the Header element to a specific endpoint.

Syntax

soap:actor="*URI*"

MustUnderstand attribute

It indicates whether a Header element is optional or mandatory. If set to true, the recipient must understand and process the Header attribute according to its defined semantics, or return a fault.

The following example shows how to use a Header in a SOAP message.

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

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

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

If you add mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it will fail when processing the Header.

Syntax

soap:mustUnderstand="0|1"

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

   The SOAP body is a mandatory element that contains the application-defined XML data being exchanged in the SOAP message. The body must be contained within the envelope and must follow any headers that might be defined for the message.

The body is defined as a child element of the envelope, and the semantics for the body are defined in the associated SOAP schema.

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

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

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

</soap:Envelope>
```

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP namespace.

4) SOAP Fault Element

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

The SOAP Fault element has the following sub elements:

Sub Element	**Description**
<faultcode>	A code for identifying the fault

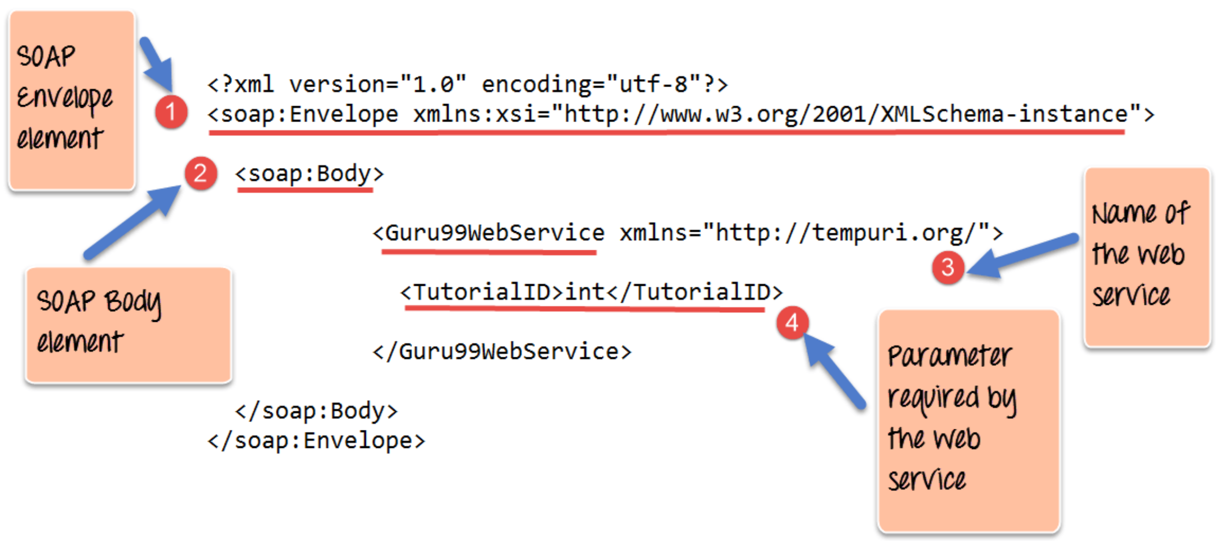
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

SOAP Fault Codes

The faultcode values defined below must be used in the faultcode element when describing faults:

Error	**Description**
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not proceed

Example below of a simple SOAP message and see what element actually does



```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
          <Guru99WebService xmlns="http://tempuri.org/">
            <TutorialID>int</TutorialID>
          </Guru99WebService>
     </soap:Body>
</soap:Envelope>
```

1. As seen from the above SOAP message, the first part of the SOAP message is the envelope element which is used to encapsulate the entire SOAP message.
2. The next element is the SOAP body which contains the details of the actual message.
3. Our message contains a web service which has the name of "Guru99WebService".
4. The "Guru99Webservice" accepts a parameter of the type 'int' and has the name of TutorialID.

Now, the above SOAP message will be passed between the web service and the client application.

You can see how useful the above information is to the client application. The SOAP message tells the client application what is the name of the Web service, and also what parameters it expects and also what is the type of each parameter which is taken by the web service.

<u>SOAP Binding</u>

The SOAP specification defines the structure of the SOAP messages, not how they are exchanged. This gap is filled by what is called "SOAP Bindings". SOAP bindings are mechanisms which allow SOAP messages to be effectively exchanged using a transport protocol.

Most SOAP implementations provide bindings for common transport protocols, such as HTTP or SMTP.

HTTP is synchronous and widely used. A SOAP HTTP request specifies at least two HTTP headers: Content-Type and Content-Length.

SMTP is asynchronous and is used in last resort or particular cases.

Java implementations of SOAP usually provide a specific binding for the JMS (Java Messaging System) protocol.

<u>Content-Type</u>

The Content-Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for the XML body of the request or response.

Syntax

Content-Type: MIMEType; charset=character-encoding

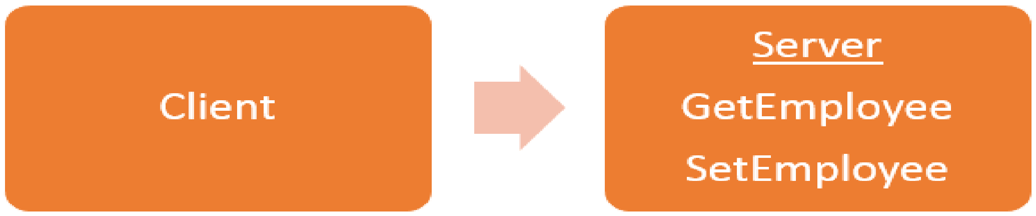
SOAP Communication Model

All communication by SOAP is done via the HTTP protocol. Prior to SOAP, a lot of web services used the standard RPC (Remote Procedure Call) style for communication. This was the simplest type of communication, but it had a lot of limitations.

Let's consider the below diagram to see how this communication works. In this example, let's assume the server hosts a web service which provided 2 methods as

- **GetEmployee** - This would get all Employee details
- **SetEmployee** – This would set the value of the details like employees dept, salary, etc. accordingly.

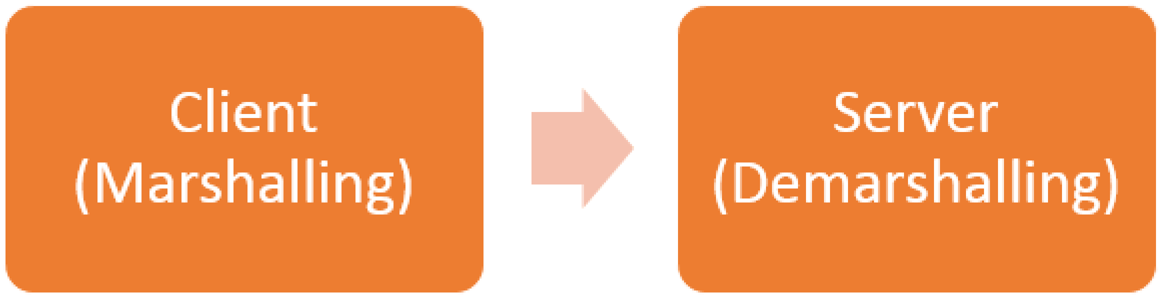
In the normal RPC style communication, the client would just call the methods in its request and send the required parameters to the server, and the server would then send the desired response.



The above communication model has the below serious limitations

1. **Not Language Independent** – The server hosting the methods would be in a particular programming language and normally the calls to the server would be in that programming language only.
2. **Not the standard protocol** – When a call is made to the remote procedure, the call is not carried out via the standard protocol. This was an issue since mostly all communication over the web had to be done via the HTTP protocol.
3. **Firewalls** – Since RPC calls do not go via the normal protocol, separate ports need to be open on the server to allow the client to communicate with the server. Normally all firewalls would block this sort of traffic, and a lot of configuration was generally required to ensure that this sort of communication between the client and the server would work.

To overcome all of the limitations cited above, SOAP would then use the below communication model



1. The client would format the information regarding the procedure call and any arguments into a SOAP message and sends it to the server as part of an HTTP request. This process of encapsulating the data into a SOAP message was known as **Marshalling.**
2. The server would then unwrap the message sent by the client, see what the client requested for and then send the appropriate response back to the client as a SOAP message. The practice of unwrapping a request sent by the client is known as **Demarshalling.**

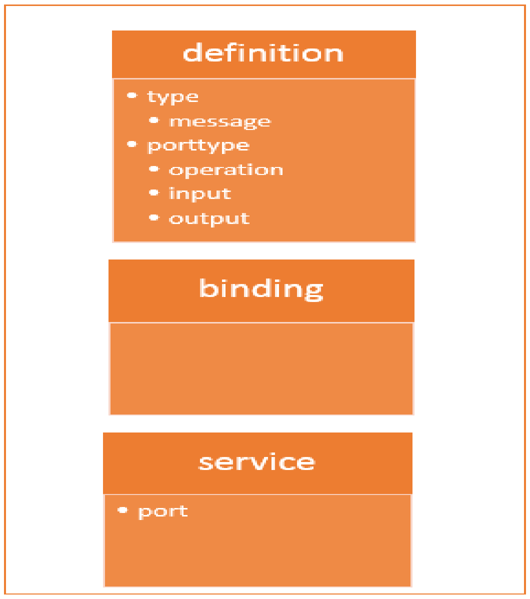
Summary

- SOAP is a protocol which is used to interchange data between applications which are built on different programming languages.
- SOAP is built upon the XML specification and works with the HTTP protocol. This makes it a perfect for usage within web applications.
- The SOAP building blocks consist of a SOAP Message. Each SOAP message consists of an envelope element, a header, and a body element.
- The envelope element is the mandatory element in the SOAP message and is used to encapsulate all of the data in the SOAP message.
- The header element can be used to contain information such as authentication information or the definition of complex data types.
- The body element is the main element which contains the definition of the web methods along with any parameter information if required.

WSDL – Web services description language

A web service is an important component in building modern day web applications. Their main purpose is to allow multiple applications built on various programming languages to talk to each other. For instance, we can have a .Net web application talks to a Java application via a Web service.

Below is a diagram on the structure of a WSDL file



WSDL Documents

An WSDL document describes a web service. It specifies the location of the service, and the methods of the service, using these major elements:

Element	**Description**
<types>	Defines the (XML Schema) data types used by the web service
<message>	Defines the data elements for each operation
<portType>	Describes the operations that can be performed and the messages involved.
<binding>	Defines the protocol and data format for each port type

The main structure of a WSDL document looks like this:

```
<definitions>

<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>
```

```
<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

WSDL Example

This is a simplified fraction of a WSDL document:

```
<definitions   name="MyService"   targetNamespace=http://example.org/math/
        xmlns=http://schemas.xmlsoap.org/wsdl/>
<!-- abstract definitions -->
<types> ...
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

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

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

<!—concrete definitions→

<binding>..

<service>..

</definition>
```

In this example the **<portType>** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.

The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".

The **<message>** elements define the **parts** of each message and the associated data types.

The WSDL file contains the following main parts:

1) The <types> tag is used to define all the complex datatypes, which will be used in the message exchanged between the client application and the web service. This is an important aspect for the client application, because if the web service works with a complex data type, then the client application should know how to process the complex data type. Data types such as float, numbers and strings are all simple data types, but there could be structured data types which may be provided by the web service.

For example, there could be a data type called EmployeeDataType which could have 2 elements called "EmployeeName" of type string and "EmployeeID" of type number or integer. Together they form a data structure which then becomes a complex data type.

2) The <messages> tag is used to define the message which is exchanged between the client application and the web server. These messages will explain the input and output operations which can be performed by the web service. An example of a message can be a message which accepts the EmployeeID of an employee, and the output message can be the name of the employee based on the EmpoyeeID provided.

3) The <portType> tag is used to encapsulate every input and output message into one logical operation. So there could be an operation called "GetEmployee" which combines the input message of accepting the EmployeeID from a client application and then sending the EmployeeName as the output message.

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

Type	**Definition**
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

4) The <binding> tag is used to bind the operation to the particular port type. This is so that when the client application calls the relevant port type, it will then be able to access the operations which are bound to this port type. Port types are just like interfaces. So if a client application needs to use a web service they need to use the binding information to ensure that they can connect to the interface provided by that web service.

5) The <service> tag is a name given to the web service itself. Initially, when a client application makes a call to the web service, it will do by calling the name of the web service. For example, a web service can be located at an address such as http://localhost/Guru99/Tutorial.asmx . The service tag will actually have the URL defined as http://localhost/Guru99/Tutorial.asmx, which will actually tell the client application that there is a web service available at this location.

A web service has the following key features

- It is built using the XML programming language. Almost all modern day technologies such as .Net and Java have corresponding commands that have the ability to work with XML. Hence, XML was taken as the most appropriate language for building web services.
- Web services communicate over HTTP. HTTP is a protocol used by all web based applications. Hence, it just made sense to ensure that Web services also had the ability to work over the HTTP protocol.
- Web services conform to a particular language specification. This specification is set by the W3C, which is the governing body for all web standards.
- Web services have a description language known as WSDL, which is used to describe the web service.

The WSDL file is used to describe in a nutshell what the web service does and gives the client all the information required to connect to the web service and use all the functionality provided by the web service.

A WSDL document is used to describe a web service. This description is required so that client applications are able to understand what the web service actually does.

- The WSDL file contains the location of the web service and
- The methods which are exposed by the web service.

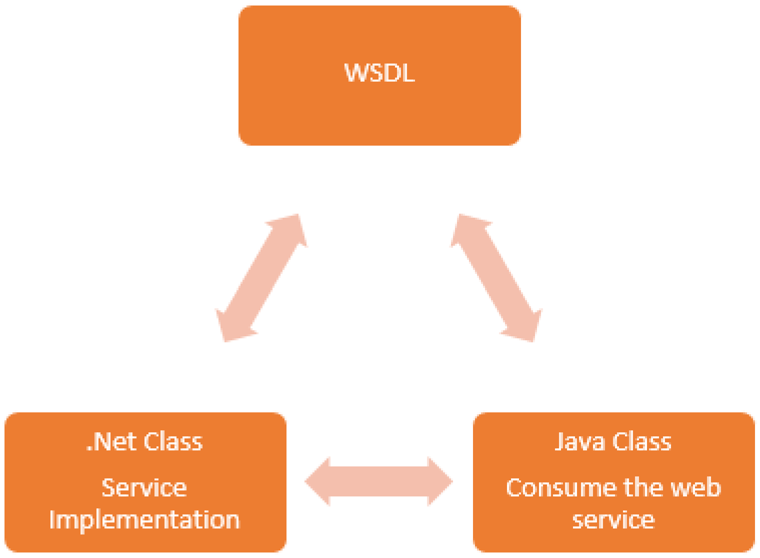
✓ The WSDL file itself can look very complex to any user, but it contains all the necessary information that any client application would require to use the relevant web service.
✓ One key thing to note here is that definition of messages, which is what is passed by the SOAP protocol is actually defined in the WSDL document.
✓ The WSDL document actually tells a client application what are the types of SOAP messages which are sent and accepted by the Web service.

- ✓ In other words, the WSDL is just like a postcard which has the address of a particular location. The address provides the details of the person who delivered the postcard. Hence, in the same way, the WSDL file is the postcard, which has the address of the web service which can deliver all the functionality that the client wants.

Why WSDL

The WSDL file is written in plain old XML. The reason that it is in XML is so that the file can be read by any programming language.

So if the client application was written in .Net, it would understand the XML file. Similarly, if the client application was written in the Java programming language then also it would be able to interpret the WSDL file.



The WSDL file is what binds everything together. From the above diagram, you can see that you can create a web service in the .Net language.

So this is where the service gets implemented. If you did not have the WSDL file and wanted a Java class to consume the web service, you would need a lot of coding effort to achieve this.

But now with the WSDL file which is in XML, which can be understood by any programming language, you can now easily have a Java class consume the .Net web service. Hence, the amount of coding effort is greatly reduced.

WSDL message part

The WSDL consists of a section called "messages" which is denoted by the **<message>** element.

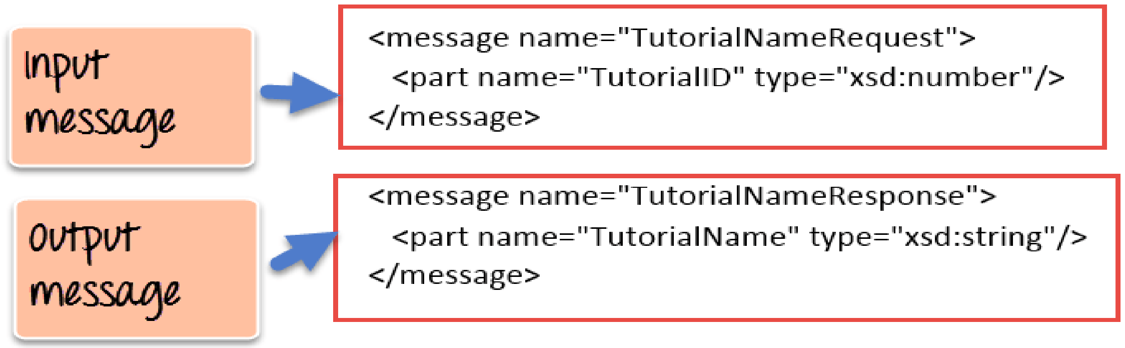
This element is basically used to describe the data that gets exchanged between the web service and the client application.

Each web service will always have 2 types of messages,

- One is for the input of the web service, and the other is for the output of the web service.
- The input is used to describe the parameters which are accepted by the web service. This is an important aspect of the client application so that it knows the values to be sent as parameters to the web service.
- The other type of message is the output message which tells what results are provided by the web service.

Each message, in turn, will have a **<part>** element which is used to describe the parameter used by the input and output message.

Below is a simple example, of what a message for a web service looks like. The functionality of the web service is to provide the name of a "Tutorial" once a "Tutorial ID" is submitted as a parameter to the web service.



```
<message name="TutorialNameRequest">
  <part name="TutorialID" type="xsd:number"/>
</message>
```

```
<message name="TutorialNameResponse">
  <part name="TutorialName" type="xsd:string"/>
</message>
```

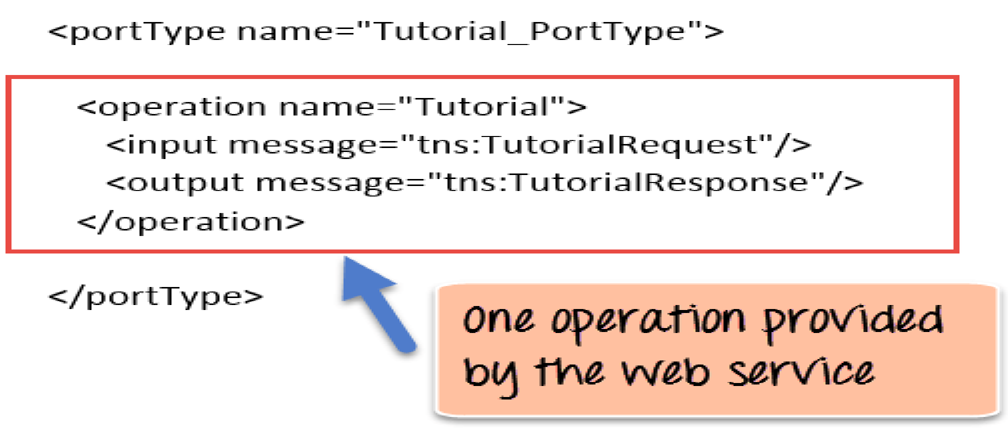
1. As we can see the web service has 2 messages, one for the input and the other for the output.
2. The input message is known as TutorialNameRequest which has one parameter called TutorialID. This parameter is of the type number which is specified by the xsd:number type
3. The output message is known as TutorialNameResponse which has one parameter called TutorialName. This parameter is of the type string which is specified by the xsd:string type

Port type binding

Ports are used in WSDL to define one complete operation which is offered by the web service.

In the previous topic, we saw that our web service provided 2 messages, one for the input called "TutorialNameRequest" and the other for the output called "TutorialNameResponse." Together the input and output message form is known as one complete operation.

WSDL provides an element called **<portType>** which is used to define the operations provided by the Web service.

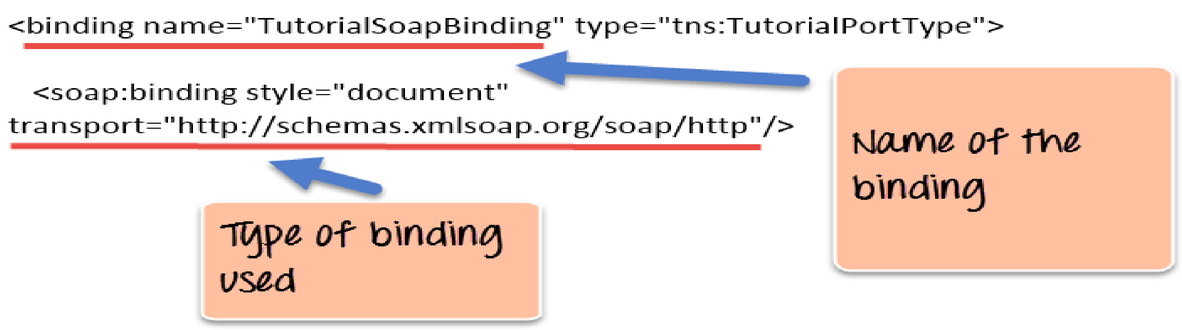


```
<portType name="Tutorial_PortType">
  <operation name="Tutorial">
    <input message="tns:TutorialRequest"/>
    <output message="tns:TutorialResponse"/>
  </operation>
</portType>
```

So in our above example we can note the following:

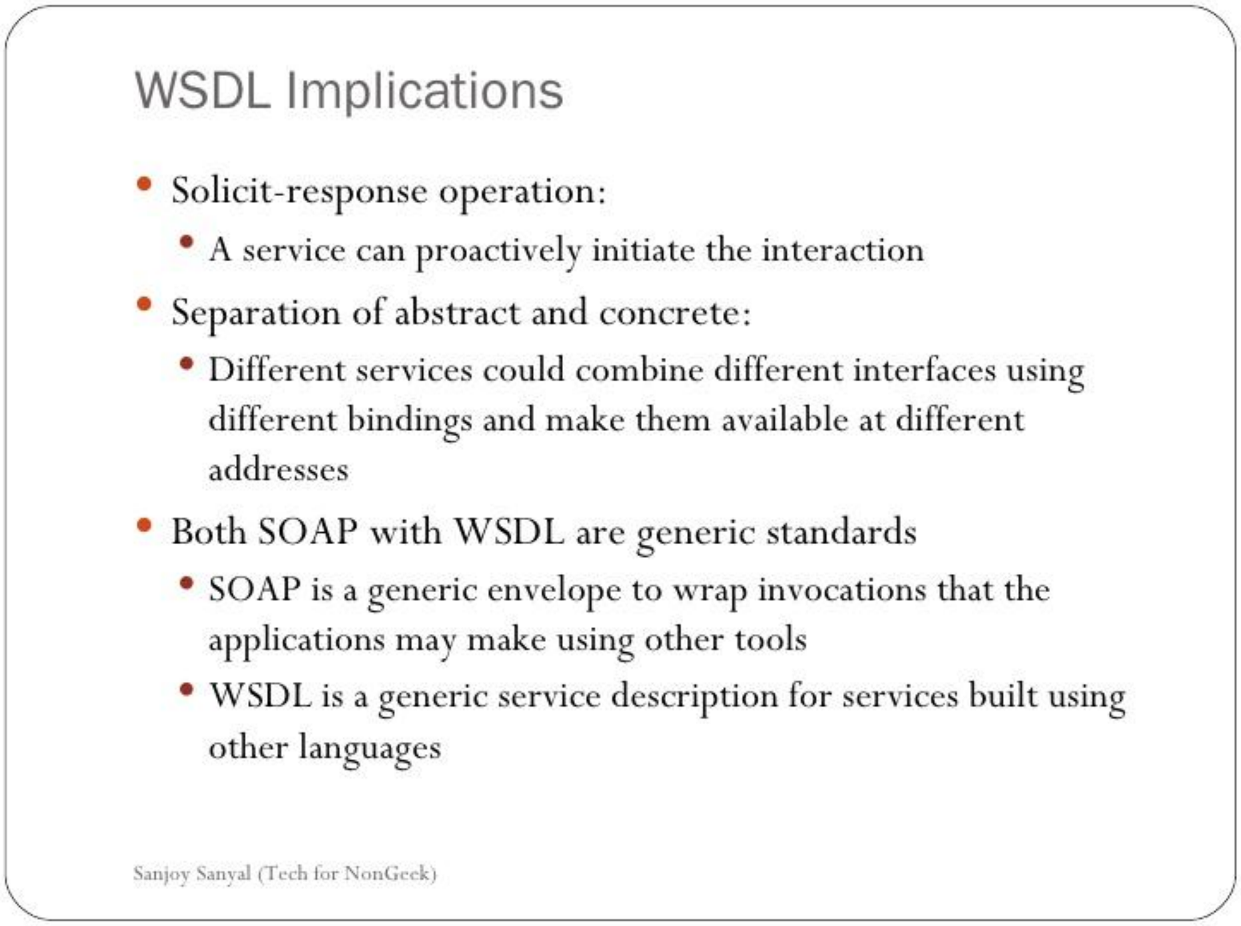
1. The name of the port Type which encapsulates the operation is given as "Tutorial_PortType."
2. The operation itself is given a name of "Tutorial". So our operation basically provides a TutorialName if the TutorialID is given as an input parameter.
3. Next is our 2 messages, one for the input and the other for the output which forms our operation

In addition to the <**portType**> element, there is also the <**binding**> element which is used to define how the messages will be transferred.



```
<binding name="TutorialSoapBinding" type="tns:TutorialPortType">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
```

1. The above example shows that the binding consists of a binding name which in our case is given as "TutorialSoapBinding". Binding in simple terms is the information which the client application uses to actually bind itself to the web service. Once it is actually bound to the web service, it then has the ability to call the various operations that are exposed by the web service.
2. The transport layer is given as http:// which means that the messages which will transfer over the HTTP protocol.



WSDL Implications

- Solicit-response operation:
  - A service can proactively initiate the interaction
- Separation of abstract and concrete:
  - Different services could combine different interfaces using different bindings and make them available at different addresses
- Both SOAP with WSDL are generic standards
  - SOAP is a generic envelope to wrap invocations that the applications may make using other tools
  - WSDL is a generic service description for services built using other languages

Sanjoy Sanyal (Tech for NonGeek)

Summary

- A WSDL document is a document that is used to describe a web service. This is key for any client application to know where the web service is located. It also allows the client application to understand the methods available in the web service.
- The WSDL file makes it very easy for the web service to be implemented in one programming language and called from a different programming language.
- The WSDL document normally consists of a message. For each web method, there are 2 messages, one is for the input, and the other is for the output. Together they form an operation.
- WSDL files normally get created in the editor which is used for the corresponding programming language.
- We have seen how we can consume a web service in Visual Studio. This can be done by creating another project which is a console application. Then by adding a service reference we are then able to access the web methods in our web service.

Restful web services

REST stands for REpresentational State Transfer. 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 is a way to access resources which lie on a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.

The key elements of a RESTful implementation are as follows:

Resources – The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is http://demo.guru99.com. Now in order to access an employee record resource via REST, one can issue the command http://demo.guru99.com/employee/1 - This command tells the web server to please provide the details of the employee whose employee number is 1.

Request Verbs - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example http://demo.guru99.com/employee/1 , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.

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

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

Response Body – This is the main body of the response. So in our example, if we were to query the web server via the request http://demo.guru99.com/employee/1 , the web server might return an XML document with all the details of the employee in the Response Body.

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

The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.

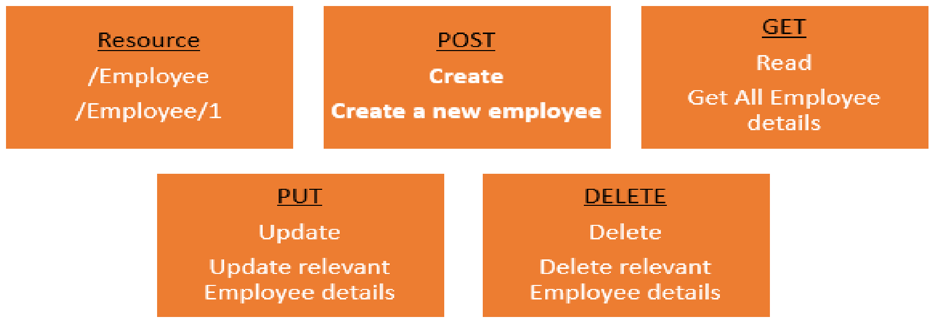
Below is what would happen If the respective verbs were sent by the client.

1. **POST** – This would be used to create a new employee using the RESTful web service
2. **GET** - This would be used to get a list of all employee using the RESTful web service
3. **PUT** - This would be used to update all employee using the RESTful web service
4. **DELETE** - This would be used to delete all employee using the RESTful web service

Let's take a look from a perspective of just a single record. Let's say there was an employee record with the employee number of 1.

The following actions would have their respective meanings.

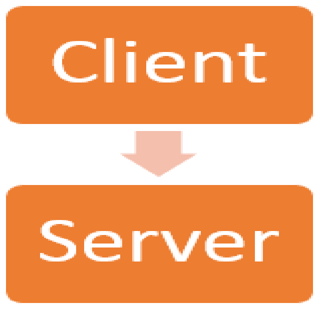
1. **POST** – This would not be applicable since we are fetching data of employee 1 which is already created.
2. **GET** - This would be used to get the details of the employee with Employee no as 1 using the RESTful web service
3. **PUT** - This would be used to update the details of the employee with Employee no as 1 using the RESTful web service
4. **DELETE** - This is used to delete the details of the employee with Employee no as 1



RESTFul Characteristics

The REST architecture is based on a few characteristics which are elaborated below. Any RESTful web service has to comply with the below characteristics in order for it to be called RESTful. These characteristics are also known as design principles which need to be followed when working with RESTful based services.

1. **Client-Server**



This is the most fundamental requirement of a REST based architecture. It means that the server will have a RESTful web service which would provide the required functionality to the client. The client send's a request to the web service on the server. The server would either reject the request or comply and provide an adequate response to the client.

2. **Stateless**

The concept of stateless means that its up to the client to ensure that all the required information is provided to the server. This is required so that server can process the response appropriately. The server should not maintain any sort of information between requests from the client. It's a very simple independent question-answer sequence. The client asks a question, the server answers it appropriately. The client will ask another question. The server will not remember the previous question-answer scenario and will need to answer the new question independently.

3. **Cache**

Client
Cache
Server

The Cache concept is to help with the problem of stateless which was described in the last point. Since each server client request is independent in nature, sometimes the client might ask the server for the same request again. This is even though it had already asked for it in the past. This request will go to the server, and the server will give a response. This increases the traffic across the network. The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information. This saves the amount of to and fro network traffic from the client to the server.

4. **Layered System**

The concept of a layered system is that any additional layer such as a middleware layer can be inserted between the client and the actual server hosting the RESTFul web service (The middleware layer is where all the business logic is created. This can be an extra service created with which the client could interact with before it makes a call to the web service.). But the introduction of this layer needs to be transparent so that it does not disturb the interaction between the client and the server.

5. **Interface/Uniform Contract**

This is the underlying technique of how RESTful web services should work. RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server

- POST - To create a resource on the server
- GET - To retrieve a resource from the server
- PUT - To change the state of a resource or to update it
- DELETE - To remove or delete a resource from the server

Summary

- REST stands for REpresentational State Transfer. REST is used to build web services that are lightweight, maintainable, and scalable in nature.
- More and more applications are moving to the Restful architecture. This is because there are a lot of people now using mobile devices and a wider variety of applications moving to the cloud.
- The main aspects of REST are the resources which reside on the server and the verbs of GET, POST, PUT and DELETE, which can be used to work with these resources.
- Visual Studio and.Net can be used to create Restful web services.
- When testing web services for POST and PUT, you need to use another tool called fiddler which can be used to send the POST and PUT request to the server.

Universal Description, Discovery, and Integration (UDDI)

Universal Description, Discovery, and Integration (UDDI) provides the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web Services providers, (2) the Web Services they make available, and (3) the technical interfaces which may be used to access those services. The idea is to "discover" organizations and the services that organizations offer, much like using a phone book or dialing information.

UDDI was first developed by UDDI.org and then transferred to OASIS. UDDI.org was comprised of more than 300 business and technology leaders working together to enable companies and applications to quickly, easily, and dynamically find, and use Web Services.

UDDI is based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP. It provides an infrastructure for a Web Services-based software environment for both publicly available services and services only exposed internally within an organization. The UDDI Business Registry system consists of three directories:

- UDDI white pages: basic information such as a company name, address, and phone numbers, as well as other standard business identifiers like Dun & Bradstreet and tax numbers.
- UDDI yellow pages: detailed business data, organized by relevant business classifications. The UDDI version of the yellow pages classifies businesses according to the newer NAICS (North American Industry Classification System) codes, as opposed to the SIC (Standard Industrial Classification) codes.
- UDDI green pages: information about a company's key business processes, such as operating platform, supported programs, purchasing methods, shipping and billing requirements, and other higher-level business protocols.

Basic goals of UDDI

- Framework for describing and discovering business services, and service providers
- Defines data structures and APIs for publishing services descriptions to the registry and querying the registry
- Support developers in finding information about services
- Determine the security and transport protocols supported by a given Web service
- Support looking for services based on a general keyword

Structure of UDDI

- XML documents
- APIs are specified in WSDL with SOAP binding
- UDDI registry itself can be accessed as a web service

Supports four core data structures:

□ businessEntity

□businessService

□bindingTemplate

tMode

SOAP vs REST Web Services

There are many differences between SOAP and REST web services. The important 10 differences between SOAP and REST are given below:

No.	SOAP	REST
1)	SOAP is a **protocol**.	REST is an **architectural style**.
2)	SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
3)	SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
5)	**JAX-WS** is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
6)	SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
8)	SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
9)	SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.